PRÁCTICA 1: HADOOP Y SPARK

Iñigo Gómez Carvajal y Jon Zorrilla Gamboa

Parte 1: Programación básica en Java con Hadoop

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

Para activar el sistema HDFS, se necesita modificar el fichero *core-site.xml*, donde se especifica la dirección y puerto donde HDFS se está ejecutando por defecto, en nuestro caso es http://localhost:9000.

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

Figura : Líneas a modificar en core-site.xml para definir la localización de HDFS.

Además de esto, se indica en el fichero *hdfs-site.xml* la propiedad referente al factor de replicación de los archivos dentro de HDFS, en nuestro caso 1.

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

Figura : Líneas a modificar en core-site.xml para cambiar el factor de replicación de los archivos dentro de HDFS.

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?

En este caso no sería suficiente, porque al establecer una dirección por defecto para HDFS, cualquier llamada a funciones de Hadoop se dedicará a buscar archivos dentro de la dirección que se haya especificado en el archivo *core-site.xml.* Para solucionar esto, debería ser suficiente con restaurar la configuración del archivo mencionado a la configuración por defecto.

Parte 3: Programación de aplicaciones de Hadoop con Java

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

La localización de HDFS se define en *dfs.datanode.data.dir* del fichero *hdfs-site.xml*. El valor por defecto es *file://${hadoop.tmp.dir}/dfs/data.* La variable *${hadoop.tmp.dir}* se puede modificar en el fichero *hdfs-site.xml* y su valor por defecto es *tmp/hadoop-${user.name}.*

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

Para borrar todo el contenido del HDFS y su estructura, se utiliza el siguiente comando:

$ hadoop namenode -format

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

Para volver a ejecutar WordCount como si fuese single node, se debe eliminar la propiedad *df.defaultFS* del fichero *core-site.xml* y la propiedad *dfs.replication* del fichero *hdfs-site.xml.*

3.4: ¿Cuáles son las 10 palabras más utilizadas?

Estas son las 10 palabras más utilizadas en El Quijote.

```
su      617
con     621
los     696
se      753
no      915
en      1155
el      1232
la      1423
a       1428
y       2585
de      2816
que     3055
[bigdata@localhost hadoop]$ bin/hdfs dfs -cat supernena/part* | sort -k2 -n
```

Figura : 10 palabras más utilizadas en ElQuijote.txt.

3.5: ¿Cuántas veces aparece el artículo "el" y la palabra "dijo"?

"Dijo" aparece 272 veces y "el" 1232 veces.

```
[bigdata@localhost hadoop]$ bin/hdfs dfs -cat supernena/part* | grep 'dijo'
dijo    272
[bigdata@localhost hadoop]$ bin/hdfs dfs -cat supernena/part* | grep '^el'
el      1232
```

Figura : Conteo de las palabras "dijo" y "el".

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

El resultado no coincide con el WordCount que se da en los ejemplos debido a que el de los ejemplos también tiene en cuenta los caracteres no alfabéticos y distingue entre mayúsculas y minúsculas; es por ello que nosotros hemos realizado un filtro para tener en cuenta este tipo de caracteres.

En la siguiente figura se puede observar el WordCount de los ejemplos:

```
árabes  1
árbol   1
árbol,  1
árboles 6
árboles,        3
áspero  1
échase  2
églogas,        2
él      205
él),    1
él,     32
él.     4
él:     2
él;     1
él?     2
émula   1
ése,    1
éso     1
ésta!   1
ésta?   1
éstas   1
éstas.  1
éste    1
ésto    7
ésto,   3
ídolo   1
ímpetu  1
ímpetus.        1
ínsula  6
ínsula, 4
ínsula. 2
ínsulas 4
ínsulas,        1
órden   2
órdenes 2
órdenes.        1
última  3
último  3
única   1
única,  1
único   3
[bigdata@localhost hadoop]$
```

Figura : Ejemplo del conteo de palabras mediante el WordCount de los ejemplos.

Por otro lado, en la siguiente figura se puede ver el WordCount realizado tras el filtro.

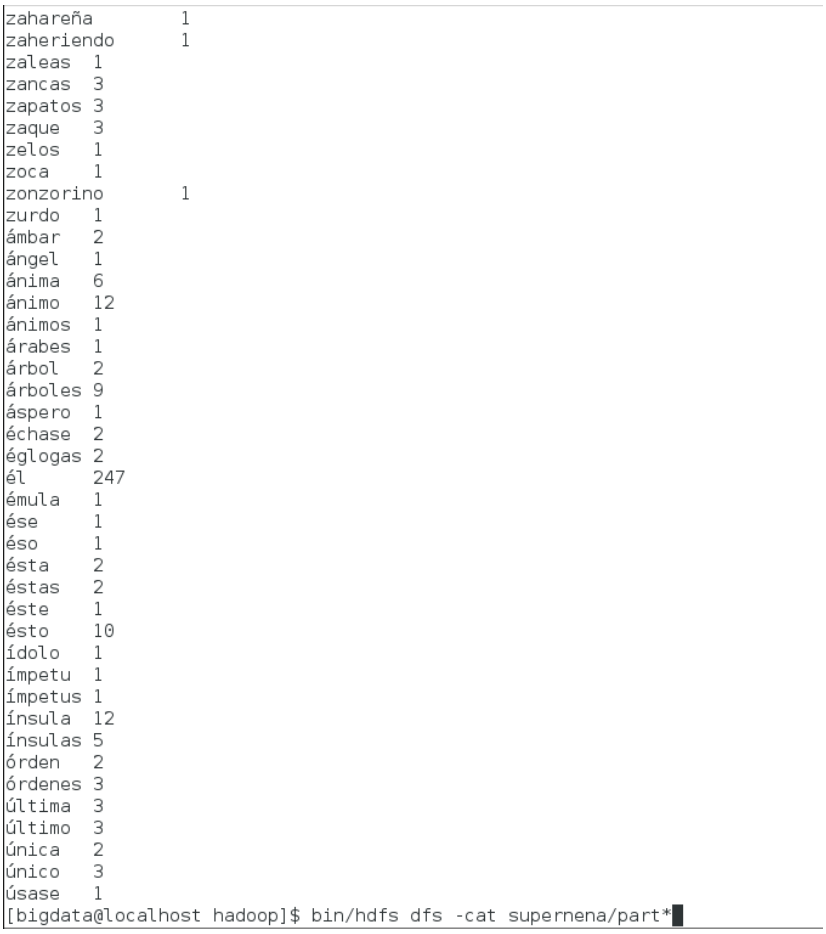

Figura :Ejemplo de conteo de palabras de WordCount tras el filtro.

Se puede observar que el conteo de las palabras es distinto. El primer WordCount, como se puede observar, tiene en cuenta palabras con signos de puntuación a sus lados. Por ejemplo, la palabra “él” se cuenta 205 veces, mientras que el conteo tras el filtro es de 247.

Parte 4: Modificación de parámetros MapReduce.

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

Para comprobar el tamaño de bloques de HDFS, cambiaremos el tamaño del archivo quijote.txt concatenándolo varias y así aumentando su tamaño a un tamaño mayor de 5MB (en nuestro caso, lo aumentamos hasta superar los 7MB). También, redujimos el tamaño de los bloques HDFS a los siguientes valores: 2.097.152 Bytes y 1.048.576 Bytes, dando lugar a los siguientes procesos Map, respectivamente:

```
Job Counters
        Launched map tasks=4
        Launched reduce tasks=1
        Data-local map tasks=4
```

Figura : 4 procesos Map para un tamaño de bloques de HDFS de 2.097.152 Bytes.

```
Job Counters
        Failed map tasks=1
        Killed map tasks=1
        Launched map tasks=9
        Launched reduce tasks=1
        Other local map tasks=1
        Data-local map tasks=9
```

Figura : 8 procesos Map para un tamaño de bloques de HDFS de 1.048.576 Bytes.

Mientras que el número de Maps para el tamaño de los bloques HDFS por defecto es el siguiente:

```
Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
```

Figura : 1 proceso Map para el tamaño de bloques de HDFS por defecto (128MB).

PySpark en Google Colab

Instalacion Octubre/2022

1. Instalacion Java
2. Instalacion de Spark
3. Instalar PySpark

De forma General para usar pyspark en Colab (2022) siga los siguientes pasos en una celda en Colab:

```
# instalar Java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

```
# Descargar la ultima versión de java ( comprobar que existen los path de descarga)
# Download latest release. Update if necessary

!wget -q https://downloads.apache.org/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz
```

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

```
total 292324
drwxr-xr-x 1 root root      4096 Oct  7 08:08 ./
drwxr-xr-x 1 root root      4096 Oct  7 08:06 ../
drwxr-xr-x 4 root root      4096 Oct  5 13:34 .config/
drwxr-xr-x 1 root root      4096 Oct  5 13:35 sample_data/
-rw-r--r-- 1 root root 299321244 Jun  9 20:39 spark-3.3.0-bin-hadoop3.tgz
```

```
!tar xf spark-3.3.0-bin-hadoop3.tgz
```

```
%ls -la
```

```
total 292328
drwxr-xr-x  1 root root      4096 Oct  7 08:08 ./
drwxr-xr-x  1 root root      4096 Oct  7 08:06 ../
drwxr-xr-x  4 root root      4096 Oct  5 13:34 .config/
drwxr-xr-x  1 root root      4096 Oct  5 13:35 sample_data/
drwxr-xr-x 13 1000 1000      4096 Jun  9 20:37 spark-3.3.0-bin-hadoop3/
-rw-r--r--  1 root root 299321244 Jun  9 20:39 spark-3.3.0-bin-hadoop3.tgz
```

```
# instalar pyspark
!pip install -q pyspark
```

```
     |████████████████████████████████| 281.3 MB 48 kB/s
     |████████████████████████████████| 199 kB 46.8 MB/s
  Building wheel for pyspark (setup.py) ... done
```

Variables de entorno

```
import os # libreria de manejo del sistema operativo
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.0-bin-hadoop3"
```

Iniciar la sesión de Spark con pyspark en el sistema

```
from pyspark.sql import SparkSession

APP_NAME = "PDGE-tutorialSpark1"
SPARK_URL = "local[*]"
spark = SparkSession.builder.appName(APP_NAME).master(SPARK_URL).getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
`v3.3.0`
Master
`local[*]`
AppName
`PDGE-tutorialSpark1`

```
sc = spark.sparkContext
#obtener el contexto de ejecución de Spark del Driver.
sc
```

SparkContext

Spark UI

Version
`v3.3.0`
Master
`local[*]`
AppName
`PDGE-tutorialSpark1`

```
array = sc.parallelize([1,2,3,4,5,6,7,8,9,10], 2)
array
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274
```

```
print(array.collect())
```

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

```
print(array.count())
```

```
10
```

```
num3 = array.map(lambda elemento: 3*elemento)
```

```
print(num3.collect())
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Ejemplos de operaciones con RDDs de Spark

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10],2)

print(numeros.reduce(lambda e1,e2: e1+e2))
```

```
55
```

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

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

print(rdd.collect())
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

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

```
#¿Cómo hacer para obtener una lista de los elementos al cuadrado?
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])

rdd = numeros.map(lambda e: e**2)
print(rdd.collect())
```

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

Pregunta TS1.2 ¿Cómo filtrar los impares?

```
#¿Cómo filtrar los impares?
rddi = numeros.filter(lambda e: e%2 != 0)
print(rddi.collect())
```

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

Ejemplos de operaciones con números

Sumar n numeros

```
rddnum=sc.range(1000000)
rddnum.reduce(lambda a,b: a+b)
```

```
499999500000
```

```
numeros = sc.parallelize([1,1,2,2,5])
unicos = numeros.distinct()
print (unicos.collect())
```

```
[2, 1, 5]
```

```
numeros = sc.parallelize([1,2,3,4,5])
rdd = numeros.flatMap(lambda elemento: [elemento, 10*elemento])
print (rdd.collect())
```

```
[1, 10, 2, 20, 3, 30, 4, 40, 5, 50]
```

Muestreo

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
rdd = numeros.sample(True, 1.0)
# True = sin reemplazar
# 1.0 = fracción de elementos = 100%
print (rdd.collect())
```

```
[2, 5, 6, 6, 7, 8, 10, 10, 10]
```

Union

```
pares = sc.parallelize([2,4,6,8,10])
impares = sc.parallelize([1,3,5,7,9])

numeros = pares.union(impares)

print (numeros.collect())
```

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

reducción

```
numeros = sc.parallelize([1,2,3,4,5])
# operación asociativa y conmutativa
# orden no está definido
print (numeros.reduce(lambda elem1,elem2: elem2+elem1))
```

```
15
```

Pregunta TS1.3 ¿Tiene sentido cambiar la suma por una diferencia? ¿Si se repite se obtiene siempre el mismo resultado?

```
#Tiene sentido la reducción(elem1-elem2)?
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])

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

```
15
```

```
#Tiene sentido la reducción(elem1-elem2) si la repetimos y los elementos están en otro orden?
numeros = sc.parallelize([2,1,4,3,5,6,7,8,9,10])

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

```
17
```

```
#Tiene sentido la reducción(elem1-elem2)? ¿qué pasa si cambiamos la paralelización a 5? ¿y si es 2?
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10],2)

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

```
15
```

Respuesta: No tiene sentido hacer la resta de elemento mediante reduce, debido a que Spark siempre asume que la operación indicada en la función es conmutativa y asociativa. Es por ello que es incosistente con los resultados, porque la manera que tiene de seleccionar particiones no siempre responde al orden del RDD que nosostros estamos viendo, lo que convierte a esta operación sensible tanto al orden de los elementos como la número de particiones.

Acciones

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

print (numeros.take(30))
#¿Qué sucede si ponemos 30 en vez de 3 elementos?
```

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

```
numeros = sc.parallelize([3,2,1,4,5])
#
print (numeros.takeOrdered(3, lambda elem: -elem))
# La función lambda se está utilizando para crear el índice de la lista de ordenación
```

```
[5, 4, 3]
```

Sí

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

```
#¿Cómo lo ordenarías para que primero aparezcan los impares y luego los pares?
numeros = sc.parallelize([3,2,1,4,5])
print(numeros.takeOrdered(5, lambda elem: elem%2==0))
```

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

Respuesta: Podemos realizar esa ordenación partiendo de la premisa de que takeOrdered ordena de forma ascendente. La condición de módulo2 = 0 dara 0 cuando los números sean impares y 1 cuando sean pares, cumpliendo la ordenación que se pide.

RDDs con string

```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])

pal_minus = palabras.map(lambda elemento: elemento.lower())

print (pal_minus.collect())
```

```
['hola', 'que', 'tal', 'bien']
```

```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])

pal_long = palabras.map(lambda elemento: len(elemento))

print (pal_long.collect())
```

```
[4, 3, 3, 4]
```

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

errors = log.filter(lambda elemento: elemento[0]=='E')

print (errors.collect())
```

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

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

palabras = lineas.flatMap(lambda elemento: elemento.split())

print (palabras.collect())
```

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

```
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']]
```

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

```
#¿Cuántos elementos tiene cada rdd? ¿Cuál tiene más?
print (len(palabras_flat.collect()))
print (len(palabras_map.collect()))
```

```
6
4
```

Respuesta: Palabras_flat y palabras_map tienen 6 y 4 elementos respectivamente. Esto tiene sentido, debido a que la operación flat_map por su método de aplanamiento puede dar el caso de que devuelva un mayor número de elementos que los que han entrado por input, mientras que en el caso de map, este tiene que devolver estrictamente tantos elementos como número de elementos que hayan sido pasados a la función.

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

Respuesta: El tipo de elementos que devuelve palabras_map es una lista de caracteres. Esto es lo que explica la segunda pregunta, puesto que, aunque la operación split convierte el primer elemento que contiene solamente una cadena vacía en nada, sigue siendo una lista de caracteres, solamente que ahora está vacía.

```
palabras_flat.take(1)
```

```
['a']
```

```
palabras_map.take(1)
```

```
[[]]
```

Pregunta TS1.7. Prueba la transformación distinct si lo aplicamos a cadenas.

```
# Prueba la transformación distinct si lo aplicamos a cadenas.¿Que realiza?
log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13', 'E: e45'])
unicos = log.distinct()
print(unicos.collect())
```

```
['I: i11', 'W: w12', 'W: w13', 'E: e45', 'E: e21']
```

Respuesta: Como se puede comprobar, esta operación está tomando los valores sin repetir del RDD sobre el que operamos.

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

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

infos = log.filter(lambda elemento: elemento[0]=='I')
errors = log.filter(lambda elemento: elemento[0]=='E')

inferr = infos.union(errors)

print (inferr.collect())
```

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

```
#¿Cómo se podría obtener la misma salida pero utilizando una sola transformación y sin realizar la unión?
log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13', 'E: e45'])
inferr = log.filter(lambda elem: elem[0]!='W')
print(inferr.sortBy(lambda x: x[0], ascending = False).collect())
```

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

Respuesta: Lo que estamos haciendo aquí es simplemente filtrar por que los elementos de la lista no empiecen por W, que es la única opción restante (aunque también se podría filtrar por que empezasen por I o E en la misma sentencia y haría lo mismo). También, para respetar el orden, aplicamos la función sortBy de forma descendente con las cadena para que también se respete el mismo orden que con el primer método.

Diferencias entre hacer map y recuce con números y cadenas

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

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

```
15
```

```
#Tiene sentido esta operación?¿Cómo sale el resultado? pruebe los resultados empezando con 2 elementos e ir incrementando

numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])

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

for i in range(2,6):
  numeros_slice = sc.parallelize([1,2,3,4,5,6,7,8,9,10], i)
  print (numeros_slice.reduce(lambda elem1,elem2: elem2-elem1))
```

```
5
5
```

```
-1
1
1
```

```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien', 'Chilling'])

pal_minus = palabras.map(lambda elemento: elemento.lower())

print (pal_minus.reduce(lambda elem1,elem2: elem2+"-"+elem1))
#y esta tiene sentido esta operación?
# Qué pasa si ponemos elem2+"-"+elem1
```

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

```
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', 1), ('B', 1)]
```

Pregunta TS1.9 ¿Cómo explica el funcionamiento de las celdas anteriores?

```
#palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])
#print (pal_minus.reduce(lambda elem1,elem2: elem1+"-"+elem2))
#y esta tiene sentido esta operación?
# Qué pasa si ponemos elem2+"-"+elem1
```

Respuesta: Esta operación que concatena strings mediante barras no tiene demasiado sentido, debido a que las operaciones entre strings no son conmutativas y por tanto no es replicable para cualquier tipo de permutación de elementos del RDD. Cabe destacar, sin embargo, que en caso de que el órden siempre fuese estricto sí puede tener algo de sentido, debido a que estas operaciones entre strings sí son asociativas. Por tanto, si se respetase el órden de los elementos de las lambdas, el resultado siempre se mostrará en el órden especificado.

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

Respuesta: Aquí se está realizando una reducción mediante las claves especificadas en la tupla. En este caso, como encuentra dos valores para las claves A y B respectivamente, aplica una reducción sumando sus valores.

```
#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 = rr.reduceByKey(lambda v1,v2:v1)
#print (rr2.collect())
```

Respuesta: Aquí entendemos que lo que se pretende hacer es una comparativa de distintos métodos de reduceByKey, el primero realizará lo mismo que en el anterior y el segundo tomará A y B y solo se quedará con el primer valor.

TS1.10 Responda a las preguntas planteadas al hacer los cambios sugeridos en las siguiente celdas

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

```
rr2 = rr1.reduceByKey(lambda v1,v2:'hola')
print (rr2.collect())
```

```
[('C', 4), ('A', 'hola'), ('B', 'hola')]
[('C', 4), ('A', 'hola'), ('B', 'hola')]
```

```
r = sc.parallelize([('A', 1),('C', 2),('A', 3),('B', 4),('B', 5)])
rr = r.groupByKey()
res= rr.collect()
for k,v in res:
    print (k, list(v))
# Que operación se puede realizar al RDD rr para que la operacion sea como un reduceByKey
rr = r.groupByKey().map(lambda elem: (elem[0], sum(elem[1])))
res = rr.collect()
for k,v in res:
    print (k, v)
#¿Y simular un group con un reduceByKey y un map?
rr_reducemap = r.map(lambda elem: (elem[0], [elem[1]])).reduceByKey(lambda elem1,elem2: elem1+elem2)
print(rr_reducemap.collect())
```

```
C [2]
A [1, 3]
B [4, 5]
C 2
A 4
B 9
[('C', [2]), ('A', [1, 3]), ('B', [4, 5])]
```

```
rdd1 = sc.parallelize([('A',1),('B',2),('C',3)])
rdd2 = sc.parallelize([('A',4),('B',5),('C',6)])

rddjoin = rdd1.join(rdd2)

print (rddjoin.collect())
# Prueba a cambiar las claves del rdd1 y rdd2 para ver cuántos elementos se crean
```

```
[('A', (1, 4)), ('B', (2, 5))]
```

Respuesta: Esta operación creará tantos elementos como claves en común haya entre los dos RDDs. Esto se debe a que el tipo de join que realiza esta función es un Inner Join, que simplemente hace cruces para elementos con la misma clave. En este caso añadir alguna clave diferente en un RDD no cambiará el resultado final, pero quitar alguna de las que podemos ver en uno de ellos hará que esa clave no aparezca en el resultado final.

```
rdd1 = sc.parallelize([('A',1),('B',2),('C',3)])
rdd2 = sc.parallelize([('A',4),('A',5),('B',6),('D',7)])

rddjoin = rdd1.join(rdd2)

print (rddjoin.collect())
#Modifica join por leftOuterJoin, rightOuterJoin y fullOuterJoin ¿Qué sucede?

rddjoin = rdd1.leftOuterJoin(rdd2)

print (rddjoin.collect())

rddjoin = rdd1.rightOuterJoin(rdd2)

print (rddjoin.collect())

rddjoin = rdd1.fullOuterJoin(rdd2)

print (rddjoin.collect())
```

```
[('A', (1, 4)), ('A', (1, 5)), ('B', (2, 6))]
[('A', (1, 4)), ('A', (1, 5)), ('B', (2, 6)), ('C', (3, None))]
[('A', (1, 4)), ('A', (1, 5)), ('B', (2, 6)), ('D', (None, 7))]
[('A', (1, 4)), ('A', (1, 5)), ('B', (2, 6)), ('C', (3, None)), ('D', (None, 7))]
```

Respuesta: En cada caso lo que se está aplicando son distintos tipos de cruces:

- LeftOuterJoin hace un cruce que conserva todas las claves de la izquierda e intenta cruzar con las claves comunes del RDD de la derecha.

- RightOuterJoin hace un cruce que conserva todas las claves de la derecha e intenta cruzar con las claves comunes del RDD de la izquierda.
- FullOuterJoin conserva las claves de ambos RDDs y hace cruces con todas las combinaciones de claves comunes posible.

```
rdd = sc.parallelize([('A',1),('B',2),('C',3),('A',4),('A',5),('B',6)])

res = rdd.sortByKey(False)

print (res.collect())
```

```
[('C', 3), ('B', 2), ('B', 6), ('A', 1), ('A', 4), ('A', 5)]
```

Utilización de ficheros

```
# Crea una lista de 1000 números y la guarda.
# Da error si 'salida' existe, descomenta la linea de borrar en ese caso.
%rm -rf salida
numeros = sc.parallelize(range(0,1000), 2)
numeros.saveAsTextFile('salida')
```

```
ls -la
```

```
total 292332
drwxr-xr-x  1 root root      4096 Oct  7 09:09 ./
drwxr-xr-x  1 root root      4096 Oct  7 08:06 ../
drwxr-xr-x  4 root root      4096 Oct  5 13:34 .config/
drwxr-xr-x  2 root root      4096 Oct  7 09:09 salida/
drwxr-xr-x  1 root root      4096 Oct  5 13:35 sample_data/
drwxr-xr-x 13 1000 1000      4096 Jun  9 20:37 spark-3.3.0-bin-hadoop3/
-rw-r--r--  1 root root 299321244 Jun  9 20:39 spark-3.3.0-bin-hadoop3.tgz
```

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

```
-rw-r--r-- 1 root root 1890 Oct  7 09:09 salida/part-00000
-rw-r--r-- 1 root root 2000 Oct  7 09:09 salida/part-00001
-rw-r--r-- 1 root root    0 Oct  7 09:09 salida/_SUCCESS
```

```
%cat salida/part-00000
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

```
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
```

```
# Recupera el fichero guardado y realiza la suma
n2 = sc.textFile('salida').map(lambda a:int(a))
print(n2.reduce(lambda v1,v2: v1 + v2))

# Prueba este código y mira qué genera?
# Borra la salida y cambia las particiones en parallelize ¿Qué sucede?
#    (pe c.parallelize(xrange(0,1000),8))
```

```
499500
```

Cambia el número de archivos generados al exportar el resultado, creando tantos archivos como particiones haya especificadas en el paralellize

Pregunta TS1.11 Borra la salida y cambia las particiones en parallelize ¿Qué sucede?

(pe c.parallelize(xrange(0,1000),8))

```
# Recupera el fichero guardado y realiza la suma
n2 = sc.textFile('salida').map(lambda a:int(a))
print(n2.reduce(lambda v1,v2: v1 + v2))

# Prueba este código y mira qué genera?
# Borra la salida y cambia las particiones en parallelize ¿Qué sucede?
```

```
499500
```

```
%rm -rf salida
numeros = sc.parallelize(range(0,1000), 5)
numeros.saveAsTextFile('salida')
%ls -la salida/*
```

```
-rw-r--r-- 1 root root 690 Oct  7 09:10 salida/part-00000
-rw-r--r-- 1 root root 800 Oct  7 09:10 salida/part-00001
-rw-r--r-- 1 root root 800 Oct  7 09:10 salida/part-00002
-rw-r--r-- 1 root root 800 Oct  7 09:10 salida/part-00003
-rw-r--r-- 1 root root 800 Oct  7 09:10 salida/part-00004
-rw-r--r-- 1 root root   0 Oct  7 09:10 salida/_SUCCESS
```

Respuesta: Alterar el número de particiones en parallelize lo que produce es, aparte de influir en el número de particiones que se usan en las operaciones, que pueden alterar los resultados en operaciones que no sean ni conmutativas y/o asociativas, también provoca que el guardado del archivo se haga en tantas particiones como le hayamos especificado.

El quijote

Montar el directorio de trabajo utilizando Google Drive Loading Your Data into Google Colaboratory.

1. First of all, Upload your Data to your Google Drive.
2. Run the following script in colab shell.
3. Copy the authorization code of your account.
4. Paste the authorization code into the output shell.
5. Congrats! Now your Google Drive is mounted to this location /content/gdrive/MyDrive/

```
from google.colab import drive
#drive.flush_and_unmount()
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!cp "/content/gdrive/My Drive/quijote.txt" .
```

```
%ls -la
```

```
total 292648
drwxr-xr-x  1 root root      4096 Oct  7 09:40 ./
drwxr-xr-x  1 root root      4096 Oct  7 08:06 ../
drwxr-xr-x  4 root root      4096 Oct  5 13:34 .config/
drwx------  5 root root      4096 Oct  7 09:40 gdrive/
-rw-------  1 root root    317618 Oct  7 09:40 quijote.txt
drwxr-xr-x  2 root root      4096 Oct  7 09:10 salida/
drwxr-xr-x  1 root root      4096 Oct  5 13:35 sample_data/
drwxr-xr-x 13 1000 1000      4096 Jun  9 20:37 spark-3.3.0-bin-hadoop3/
-rw-r--r--  1 root root 299321244 Jun  9 20:39 spark-3.3.0-bin-hadoop3.tgz
```

```
%pwd
```

```
'/content'
```

Procesando el QUIJOTE

```
quijote = sc.textFile("quijote.txt")
quijote.take(10)
```

```
['EL INGENIOSO HIDALGO DON QUIJOTE DE LA MANCHA',
 '',
 'Miguel de Cervantes Saavedra',
 '',
 '     Capítulo primero',
 '',
 '     Que trata de la condición y ejercicio del famoso hidalgo D.',
 'Quijote de la ',
 '     Mancha',
 '']
```

Transformaciones

```
charsPerLine = quijote.map(lambda s: len(s))
allWords = quijote.flatMap(lambda s: s.split())
allWordsNoArticles = allWords.filter(lambda a: a.lower() not in ["el", "la"])
allWordsUnique = allWords.map(lambda s: s.lower()).distinct()
sampleWords = allWords.sample(withReplacement=True, fraction=0.2, seed=666)
weirdSampling = sampleWords.union(allWordsNoArticles.sample(False, fraction=0.3))
# cómo funciona cada transformación
```

charsPerLine: Toma la longitud de cada línea allWords: Obtiene todas y cada una de las palabras del texto allWordsNoArticles: Filtra allWords para sacarse todas las palabras a excepcion de los artículos allWordsUnique: Saca las palabras únicas del texto sampleWords: Toma una muestra del 20% del tamaño del texto completo con reemplazamiento. weirdSampling: Genera una nueva muestra del 30% del texto completo sin artículos y sin reemplazamiento y lo une a la muestra de sampleWords

```
allWordsUnique.take(10)
```

```
['el',
 'hidalgo',
 'don',
```

```
'mancha',
'saavedra',
'que',
'condición',
'y',
'del',
'd.']
```

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.

Respuesta:

- map: Transforma N datos de tipo T de entrada en N datos de tipo R a la salida, por lo que en este caso N=M
- flatmap: Transforma N datos de tipo T de entrada en N colecciones de tipo R, que luego unifica en un solo RDD mediante aplanamiento. Esto influye en que el número de elementos que devuelve no es el mismo que N necesariamente, por tanto puede darse N<M, N>M o N=M
- filter: Obtiene las filas que cumplan la condicion *booleana* especificada, así que en este caso $N \geq M$
- distinct: Obtiene los valores del RDD, eliminando los duplicados. Por tanto, $N \geq M$
- sample: Crea una muestra, con o sin reemplazamiento, de los datos indicados. Debido a que no se puede generar más de un 100% del tamaño de la muestra $N \geq M$
- union: Concatena dos RDDs en uno, asi que el resultado sera $N \leq M$, pudiendo ser iguales en caso de que el RDD que se introduce por argumento esté vacío.

```
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
```

```
(5534,
 305678,
 ['procuremos.Levántate,',
  'estrechísimamente,',
  'Pintiquiniestra,',
  'entretenimiento,',
  'maravillosamente',
  'descansadamente;',
  'desenfadadamente',
  'quebrantamientos',
  'quebrantamiento,',
  'alternativamente',
  'encantamientos,',
  'Placerdemivida,',
  'encantamientos;',
  'malbaratándolas',
  'regocijadamente',
  'consentimiento,',
  'desaconsejaban,',
  'acontecimiento.',
  'agradeciéndoles',
  'encantamientos,'])
```

Pregunta TS2.2 Explica el funcionamiento de cada acción anterior

Implementa la opción count de otra manera:

- utilizando transformaciones map y reduce
- utilizando solo reduce en caso de que sea posible.

```
numLines = quijote.map(lambda e: 1).reduce(lambda a,b: a+b)
numLines
```

```
5534
```

Respuesta: En este caso, es tan sencillo como asignar a todas las lineas un valor 1 con map para luego con reduce sumar todos los elementos

```
numLines_reduce=quijote.reduce(lambda a,b: 2 if not isinstance(a,int) else (a+b if isinstance(b, int) else a+1))
numLines_reduce
```

```
5534
```

Respuesta: Este caso a lo mejor no es tan trivial, puesto que tenemos que tener en cuenta la casuistica de los tipos de datos con los que estamos tratando. Si el primer elemento sigue siendo una cadena de texto, simplemente computamos el resultado de ese reduce como 2, que sustituye a una suma de 1+1. En el segundo caso se busca cubrir el caso donde quede alguna particion con un elemento sin operar, en cuyo caso se suma 1 al valor del primer elemento. En el último se asume que está ocurriendo en reduce posteriores, donde todo son números y ya se opera con normalidad.

Operaciones K-V (Clave Valor)

```
import requests
import re
allWords = allWords.flatMap(lambda w: re.sub("""[;|:|\.|,|-|—|"|'|\s]""","  ", w.lower()).split(" ")).filter(lambda a: len(a)
allWords2 = sc.parallelize(requests.get("https://gist.githubusercontent.com/jsdario/9d871ed773c81bf217f57d1db2d2503f/raw/5
allWords2 = allWords2.flatMap(lambda w: re.sub("""[;|:|\.|,|-|—|"|'|\s]""","  ", w.decode("utf8").lower()).split(" ")).filter
```

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

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

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

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

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

```
words.take(10)
```

```
[('el', 1),
 ('ingenioso', 1),
 ('hidalgo', 1),
 ('don', 1),
 ('quijote', 1),
 ('de', 1),
 ('la', 1),
 ('mancha', 1),
 ('miguel', 1),
 ('de', 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])
```

```
[('que', 3032),
 ('de', 2809),
 ('y', 2573),
 ('a', 1426),
 ('la', 1423),
 ('el', 1232),
 ('en', 1155),
 ('no', 903),
 ('se', 753),
 ('los', 696)]
```

```
res = words.groupByKey().takeOrdered(10, key=lambda a: -len(a))
```

```
res # To see the content, res[i][1].data
```

```
[('el', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3990>),
 ('hidalgo', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3a50>),
 ('don', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3b90>),
 ('mancha', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3b10>),
 ('saavedra', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3c50>),
 ('que', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3c90>),
 ('condición', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3a10>),
 ('y', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3e90>),
 ('del', <pyspark.resultiterable.ResultIterable at 0x7f269d9a3f50>),
 ('d', <pyspark.resultiterable.ResultIterable at 0x7f269d9a35d0>)]
```

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

```
[('el', (1232, 4394)),
 ('hidalgo', (14, 42)),
 ('don', (370, 1606)),
 ('mancha', (26, 101)),
 ('saavedra', (1, 1)),
 ('que', (3032, 10040)),
 ('y', (2573, 9650)),
 ('del', (415, 1344)),
 ('en', (1155, 4223)),
 ('cuyo', (11, 35))]
```

```
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(lam
```

```
([('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)])
```

Pregunta TS2.3 Explica el proposito de cada una de las operaciones anteriores

Respuesta: Lo que se está buscando hacer es una forma de cuantificar la frecuencia de aparición de palabras comunes a ambas partes de El Quijote y comparar la frecuencia de aparicion de esas palabras entre partes, siendo un número positivo significado de que aparece más veces en la primera parte que en la segunda, y un número negativo lo contrario. Para ello, obtiene la segunda parte mediante una petición a la dirección especificada y comienza tratando ambas partes con una regularización y limpieza de las palabras. Acte seguido, hace un wordcount mediante una combinación de map y reduce y realiza un Inner Join de la cuenta de la primera parte con la segunda. Por último, aplica la fórmula de la comparativa, que consiste en dividir la diferencia de frecuencia de cada palabra entre libros entre la suma de la frecuencia y muestra los resultados más altos, y por tanto que aparecen mucho más en la primera parte que en la segunda, y los valores más bajos, que indican lo contrario.

Pregunta TS2.4 ¿Cómo puede implementarse la frecuencia con groupByKey y transformaciones?

```
frecuencies_groupbykey = allWords.map(lambda elem: (elem, 1)).groupByKey().map(lambda elem: (elem[0], sum(elem[1])))

print(frecuencies_groupbykey.takeOrdered(10, lambda elem: -elem[1]))
```

```
[('que', 3032), ('de', 2809), ('y', 2573), ('a', 1426), ('la', 1423), ('el', 1232), ('en', 1155), ('no', 903), ('se',
```

Optimizaciones

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

```
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(lam
```

```
    ([('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)])
```

```
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])
```

```
    ([('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)])
```

Respuesta: El segundo método que vemos lo que está realizando es almacenar los resultados del map en la memoria caché, lo que permite acceder a esos datos sin tener que volver a procesarlos y poder utilizarlo en repetidas ocasiones. Esto permite que este método sea más eficiente, porque mientras que en el primero está realizando dos veces la función map, en el segundo caso se almacena el resultado del map tras ejecutarlo una vez y lo utiliza tantas veces como necesite.

```
result.coalesce(numPartitions=2) # Avoids the data movement, so it tries to balance inside each machine
result.repartition(numPartitions=2) # We don't care about data movement, this balance the whole thing to ensure all machin
```

```
    MapPartitionsRDD[444] at coalesce at NativeMethodAccessorImpl.java:0
```

```
result.take(10)
allWords.cache() # allWords RDD must  stay in memory after computation, we made a checkpoint (well, it's a best effort, so
result.take(10)
```

```
    [('el', -0.5620334162815499),
     ('hidalgo', -0.5),
     ('don', -0.6255060728744939),
     ('mancha', -0.5905511811023622),
     ('saavedra', 0.0),
     ('que', -0.5361077111383109),
     ('y', -0.5789904278818621),
     ('del', -0.5281409891984082),
     ('en', -0.5704722945332837),
     ('cuyo', -0.5217391304347826)]
```

```
from pyspark import StorageLevel
```

```
# https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#rdd-persistence
allWords2.persist(StorageLevel.MEMORY_AND_DISK) # Now it will be preserved on disk also
```

```
    PythonRDD[448] at RDD at PythonRDD.scala:53
```

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

Pregunta TS2.6 Antes de guardar el fichero, utilice coalesce con diferente valores ¿Cuál es la diferencia?

```
!rm -rf palabras_parte2
allWords2.coalesce(numPartitions=1).saveAsTextFile("palabras_parte2")
```

Respuesta: La función coalesce permite reducir el número de particiones en las que se almacena el archivo. El fichero allWords2 por defecto se está almacenando en 2 particiones, pero en este caso si se ejecuta coalesce con valor numPartitions=1, se almacena como una sola partición.

```
# instalar Java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null



# Descargar la ultima versión de java ( comprobar que existen los path de descarga)
# Download latest release. Update if necessary

!wget -q https://downloads.apache.org/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz


!tar xf spark-3.3.0-bin-hadoop3.tgz


# instalar pyspark
!pip install -q pyspark
```

```
     |████████████████████████████████| 281.3 MB 48 kB/s
     |████████████████████████████████| 199 kB 60.3 MB/s
  Building wheel for pyspark (setup.py) ... done
```

```
import os # libreria de manejo del sistema operativo
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.0-bin-hadoop3"
```

```
from pyspark.sql import SparkSession

APP_NAME = "PDGE-tutorialSpark1"
SPARK_URL = "local[*]"
spark = SparkSession.builder.appName(APP_NAME).master(SPARK_URL).getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
`v3.3.0`
Master
`local[*]`
AppName
`PDGE-tutorialSpark1`

```
sc = spark.sparkContext
#obtener el contexto de ejecución de Spark del Driver.
sc
```

SparkContext

Spark UI

Version
`v3.3.0`
Master
`local[*]`
AppName
`PDGE-tutorialSpark1`

EJERCICIO OPCIONAL: Análisis de equipos de fútbol como locales en la Premier League

El dataset seleccionado es *results.csv*, obtenido de la página https://www.kaggle.com/zaeemnalla/premier-league/data. En él se recogen los resultados de los partidos jugados en la Premier League, desde la temporada 2006-2007 hasta la 2017-2018, que vamos a utilizar para hacer operaciones que nos permitan ver de forma más directa cómo de fuertes (o débiles) fueron los equipos a la hora de jugar como locales. Todos los experimentos se harán agrupando por equipo y temporada, puesto que evaluar solamente por equipos nos parece que no refleja el comportamiento real de los equipos, cuyas plantillas pueden sufrir cambios entre temporadas.

Hemos considerado que el uso de DataFrames era más apropiado para este ejercicio, además de ser una oportunidad de poder probar esta estructura que no hemos trabajado en la realización de los anteriores ejercicios y así familiarizarnos con la funcionalidad de la misma en Spark.

```
import pyspark.sql.functions as func
from pyspark.sql.types import *
resultados = spark.read.options(header=True).csv('results.csv')
resultados.show()
```

```
+-----------------+-----------------+----------+----------+------+---------+
|        home_team|        away_team|home_goals|away_goals|result|   season|
+-----------------+-----------------+----------+----------+------+---------+
| Sheffield United|        Liverpool|       1.0|       1.0|     D|2006-2007|
|          Arsenal|      Aston Villa|       1.0|       1.0|     D|2006-2007|
|          Everton|          Watford|       2.0|       1.0|     H|2006-2007|
| Newcastle United|   Wigan Athletic|       2.0|       1.0|     H|2006-2007|
|       Portsmouth| Blackburn Rovers|       3.0|       0.0|     H|2006-2007|
|          Reading|    Middlesbrough|       3.0|       2.0|     H|2006-2007|
|  West Ham United|Charlton Athletic|       3.0|       1.0|     H|2006-2007|
| Bolton Wanderers|Tottenham Hotspur|       2.0|       0.0|     H|2006-2007|
|Manchester United|           Fulham|       5.0|       1.0|     H|2006-2007|
|          Chelsea|  Manchester City|       3.0|       0.0|     H|2006-2007|
|          Watford|  West Ham United|       1.0|       1.0|     D|2006-2007|
|Tottenham Hotspur| Sheffield United|       2.0|       0.0|     H|2006-2007|
|      Aston Villa|          Reading|       2.0|       1.0|     H|2006-2007|
|  Manchester City|       Portsmouth|       0.0|       0.0|     D|2006-2007|
| Blackburn Rovers|          Everton|       1.0|       1.0|     D|2006-2007|
|Charlton Athletic|Manchester United|       0.0|       3.0|     A|2006-2007|
|           Fulham| Bolton Wanderers|       1.0|       1.0|     D|2006-2007|
|    Middlesbrough|          Chelsea|       2.0|       1.0|     H|2006-2007|
|        Liverpool|  West Ham United|       2.0|       1.0|     H|2006-2007|
|Charlton Athletic| Bolton Wanderers|       2.0|       0.0|     H|2006-2007|
+-----------------+-----------------+----------+----------+------+---------+
only showing top 20 rows
```

1. Ránking equipos más ganadores y más perdedores

```
victorias_por_equipo = resultados.filter(resultados["result"]=='H').groupBy(["home_team", "season"]).agg(func.count("resul
victorias_por_equipo.orderBy(func.desc("home_victories")).show()
```

```
+-----------------+---------+--------------+
|        home_team|   season|home_victories|
+-----------------+---------+--------------+
|  Manchester City|2011-2012|            18|
|Manchester United|2010-2011|            18|
|  Manchester City|2013-2014|            17|
|Tottenham Hotspur|2016-2017|            17|
|Manchester United|2007-2008|            17|
|          Chelsea|2009-2010|            17|
|          Chelsea|2016-2017|            17|
|  Manchester City|2017-2018|            16|
|        Liverpool|2013-2014|            16|
|Manchester United|2008-2009|            16|
|Manchester United|2009-2010|            16|
|Manchester United|2012-2013|            16|
|Manchester United|2006-2007|            15|
|          Chelsea|2014-2015|            15|
|          Arsenal|2017-2018|            15|
|Manchester United|2017-2018|            15|
|          Chelsea|2013-2014|            15|
|Manchester United|2011-2012|            15|
|          Arsenal|2009-2010|            15|
|  Manchester City|2014-2015|            14|
+-----------------+---------+--------------+
only showing top 20 rows
```

```
victorias_por_equipo.orderBy(func.asc("home_victories")).show()
```

```
+--------------------+---------+--------------+
|           home_team|   season|home_victories|
+--------------------+---------+--------------+
|        Derby County|2007-2008|             1|
| Queens Park Rangers|2012-2013|             2|
|         Aston Villa|2015-2016|             2|
|West Bromwich Albion|2017-2018|             3|
|             Watford|2006-2007|             3|
|Wolverhampton Wan...|2011-2012|             3|
|           Sunderland|2016-2017|             3|
|            Hull City|2008-2009|             3|
|West Bromwich Albion|2013-2014|             4|
|    Bolton Wanderers|2011-2012|             4|
|           Sunderland|2014-2015|             4|
|         Southampton|2017-2018|             4|
```

```
|       Middlesbrough|2016-2017|             4|
|         Aston Villa|2011-2012|             4|
|             Reading|2012-2013|             4|
|      Wigan Athletic|2012-2013|             4|
|             Burnley|2014-2015|             4|
|      Wigan Athletic|2010-2011|             5|
|      Wigan Athletic|2011-2012|             5|
|     West Ham United|2010-2011|             5|
+--------------------+---------+--------------+
only showing top 20 rows
```

Podemos comprobar que el equipo más ganador en casa durante estos años fue el Manchester City en la temporada 2011-2012 con 18 victorias en casa, empatado con el Manchester United en la temporada 2010-2011.

En contraste, la temporada 2007-2008 del Derby County fue un absoluto desastre en lo que se refiere a marcharse contentos del Pride Park Stadium, donde solo consiguieron una victoria.

¿Podríamos concluir aquí a la hora de hablar de equipos fuertes o débiles en casa?

2. Top equipos más y menos productivos en casa

El dato anterior, sin embargo, no se traduce necesariamente en un equipo eficaz (o poco eficaz) cuando juegan en sus respectivos estadios. Para ello, vamos a sacar una métrica que creemos que es más informativa, que son los puntos obtenidos en promedio cada vez que juegan en casa.

Para ello, vamos a crear una columna que traduzca los resultados a puntos para el local.

```
def compute_home_points(result):
  if result == "H":
    return 3
  elif result == "D":
    return 1
  else:
    return 0

def compute_home_victories(result):
  return 1 if result=="H" else 0

func_points = func.udf(compute_home_points, IntegerType())

func_victories = func.udf(compute_home_victories, IntegerType())

resultados_puntos = resultados.select("*")

resultados_puntos=resultados_puntos.withColumn("home_points", func_points("result")).withColumn("home_victory", func_victo

resultados_puntos.show()
```

```
+-----------------+-----------------+----------+----------+------+---------+-----------+------------+
|        home_team|        away_team|home_goals|away_goals|result|   season|home_points|home_victory|
+-----------------+-----------------+----------+----------+------+---------+-----------+------------+
| Sheffield United|        Liverpool|       1.0|       1.0|     D|2006-2007|          1|           0|
|          Arsenal|      Aston Villa|       1.0|       1.0|     D|2006-2007|          1|           0|
|          Everton|          Watford|       2.0|       1.0|     H|2006-2007|          3|           1|
| Newcastle United|   Wigan Athletic|       2.0|       1.0|     H|2006-2007|          3|           1|
|       Portsmouth| Blackburn Rovers|       3.0|       0.0|     H|2006-2007|          3|           1|
|          Reading|    Middlesbrough|       3.0|       2.0|     H|2006-2007|          3|           1|
|  West Ham United|Charlton Athletic|       3.0|       1.0|     H|2006-2007|          3|           1|
| Bolton Wanderers|Tottenham Hotspur|       2.0|       0.0|     H|2006-2007|          3|           1|
|Manchester United|           Fulham|       5.0|       1.0|     H|2006-2007|          3|           1|
|          Chelsea|  Manchester City|       3.0|       0.0|     H|2006-2007|          3|           1|
|          Watford|  West Ham United|       1.0|       1.0|     D|2006-2007|          1|           0|
|Tottenham Hotspur| Sheffield United|       2.0|       0.0|     H|2006-2007|          3|           1|
|      Aston Villa|          Reading|       2.0|       1.0|     H|2006-2007|          3|           1|
|  Manchester City|       Portsmouth|       0.0|       0.0|     D|2006-2007|          1|           0|
| Blackburn Rovers|          Everton|       1.0|       1.0|     D|2006-2007|          1|           0|
|Charlton Athletic|Manchester United|       0.0|       3.0|     A|2006-2007|          0|           0|
|           Fulham| Bolton Wanderers|       1.0|       1.0|     D|2006-2007|          1|           0|
|    Middlesbrough|          Chelsea|       2.0|       1.0|     H|2006-2007|          3|           1|
|        Liverpool|  West Ham United|       2.0|       1.0|     H|2006-2007|          3|           1|
|Charlton Athletic| Bolton Wanderers|       2.0|       0.0|     H|2006-2007|          3|           1|
+-----------------+-----------------+----------+----------+------+---------+-----------+------------+
only showing top 20 rows
```

```
resultados_puntos = resultados_puntos.groupBy(["home_team", "season"]).agg(func.mean("home_points").alias("avg_home_points
```

```
resultados_puntos.orderBy(func.desc("avg_home_points")).show()
```

```
+----------------+---------+------------------+-------------+
|        home_team|   season|   avg_home_points|home_victories|
+----------------+---------+------------------+-------------+
|  Manchester City|2011-2012|2.8947368421052633|           18|
|Manchester United|2010-2011|2.8947368421052633|           18|
|Tottenham Hotspur|2016-2017| 2.789473684210526|           17|
|Manchester United|2007-2008| 2.736842105263158|           17|
|          Chelsea|2009-2010| 2.736842105263158|           17|
|  Manchester City|2013-2014| 2.736842105263158|           17|
|          Chelsea|2016-2017|2.6842105263157894|           17|
|  Manchester City|2017-2018|2.6315789473684212|           16|
|Manchester United|2008-2009|2.6315789473684212|           16|
|          Chelsea|2014-2015|2.5789473684210527|           15|
|        Liverpool|2013-2014|2.5789473684210527|           16|
|Manchester United|2009-2010|2.5789473684210527|           16|
|Manchester United|2012-2013| 2.526315789473684|           16|
|          Chelsea|2013-2014| 2.526315789473684|           15|
|Manchester United|2006-2007| 2.473684210526316|           15|
|Manchester United|2017-2018| 2.473684210526316|           15|
|          Arsenal|2017-2018| 2.473684210526316|           15|
|Manchester United|2011-2012| 2.473684210526316|           15|
|          Arsenal|2007-2008| 2.473684210526316|           14|
|          Arsenal|2009-2010| 2.473684210526316|           15|
+----------------+---------+------------------+-------------+
only showing top 20 rows
```

```
resultados_puntos.orderBy(func.asc("avg_home_points")).show()
```

```
+--------------------+---------+--------------------+-------------+
|           home_team|   season|    avg_home_points|home_victories|
+--------------------+---------+--------------------+-------------+
|        Derby County|2007-2008|0.42105263157894735|            1|
|         Aston Villa|2015-2016| 0.5789473684210527|            2|
|Wolverhampton Wan...|2011-2012|  0.631578947368421|            3|
| Queens Park Rangers|2012-2013| 0.7368421052631579|            2|
|           Hull City|2008-2009| 0.7368421052631579|            3|
|          Sunderland|2016-2017| 0.7368421052631579|            3|
|    Bolton Wanderers|2011-2012| 0.8421052631578947|            4|
|          Portsmouth|2009-2010| 0.9473684210526315|            5|
|      Wigan Athletic|2012-2013| 0.9473684210526315|            4|
|          Sunderland|2013-2014| 0.9473684210526315|            5|
|West Bromwich Albion|2017-2018| 0.9473684210526315|            3|
|              Fulham|2013-2014| 0.9473684210526315|            5|
|       Middlesbrough|2016-2017| 0.9473684210526315|            4|
|            Watford|2006-2007| 0.9473684210526315|            3|
|      Wigan Athletic|2006-2007|                 1.0|            5|
|             Burnley|2014-2015|                 1.0|            4|
|         Southampton|2017-2018|                 1.0|            4|
|         Aston Villa|2011-2012|                 1.0|            4|
|    Blackburn Rovers|2011-2012|                 1.0|            6|
|              Fulham|2007-2008| 1.0526315789473684|            5|
+--------------------+---------+--------------------+-------------+
only showing top 20 rows
```

Esto es más revelador, porque nos permite sacar conclusiones que de la anterior manera no hemos podido ver antes. Por la parte positiva, podemos ver que el Tottenham Hotspur 2016-2017 fue ligeramente más efectivo que otros equipos que también tuvieron 17 victorias o que equipos como el Portsmouth 2009-2010 o el Blackburn Rovers 2011-2012, que no llegaban a aparecer entre los 20 peores equipos locales del dataset, fueron en realidad muy malos defendiendo su campo respecto a otros que, pese a que ganaban menos partidos, empataron en más ocasiones.

Por desgracia, no hay ninguna redención para el Derby County 2007-2008, que se encuentra a bastante distancia del segundo peor. Como curiosidad, esta temporada se considera a opinión de muchos expertos como la peor temporada que ha hecho un equipo en la historia de la Premier League.

3. Equipos más goleadores en casa

Ya por último, vamos a hacer una pequeña comprobación de qué equipos eran los más goleadores, siendo ello una posible muestra de bien cómo de entretenidos podía ser verlos jugar en casa, o también cómo de vapuleantes podían ser sus partidos. Para ello, vamos a computar el numero de goles marcados, así como la diferencia de goles, todo ello en promedio.

```
goles_locales = resultados.withColumn("goal_difference", func.col("home_goals") - func.col("away_goals"))

goles_locales = goles_locales.groupBy(["home_team", "season"]).agg(func.mean("home_goals").alias("avg_home_goals"), func
```

```
goles_locales.orderBy(func.desc("avg_home_goals")).show()
```

```
+----------------+---------+------------------+------------------+
|        home_team|   season|    avg_home_goals|     avg_goal_diff|
+----------------+---------+------------------+------------------+
|          Chelsea|2009-2010|3.5789473684210527|2.8421052631578947|
|  Manchester City|2013-2014|3.3157894736842106|2.6315789473684212|
|  Manchester City|2017-2018| 3.210526315789474| 2.473684210526316|
|          Chelsea|2016-2017|2.8947368421052633|               2.0|
|  Manchester City|2011-2012|2.8947368421052633| 2.263157894736842|
|          Arsenal|2017-2018|2.8421052631578947|1.7894736842105263|
|        Liverpool|2013-2014| 2.789473684210526|1.8421052631578947|
|Manchester United|2011-2012| 2.736842105263158| 1.736842105263158|
|Manchester United|2009-2010| 2.736842105263158|2.1052631578947367|
|Manchester United|2010-2011|2.5789473684210527|1.9473684210526316|
|          Arsenal|2009-2010| 2.526315789473684| 1.736842105263158|
|  Manchester City|2015-2016| 2.473684210526316| 1.368421052631579|
|Manchester United|2007-2008| 2.473684210526316|2.1052631578947367|
|Tottenham Hotspur|2016-2017| 2.473684210526316|               2.0|
|          Arsenal|2012-2013| 2.473684210526316| 1.263157894736842|
|Manchester United|2006-2007|2.4210526315789473|1.7894736842105263|
|Tottenham Hotspur|2007-2008|2.4210526315789473| 0.631578947368421|
|        Liverpool|2016-2017|2.3684210526315788|1.4210526315789473|
|        Liverpool|2017-2018|2.3684210526315788|1.8421052631578947|
|Manchester United|2012-2013|2.3684210526315788| 1.368421052631579|
+----------------+---------+------------------+------------------+
only showing top 20 rows
```

Seguramente, los aficionados del Chelsea en el año 2009-2010 debían disfrutar mucho de sus partidos en Stamford Bridge, teniendo en cuenta que promediaban cerca de 4 goles por partido y en promedio ganaban por una diferencia de 3.

De forma anecdótica, decir que el Tottenham Hotspur en la temporada 2007-2008 puntúa entre los equipos más goleadores, pero la diferencia de goles promedio es la más baja con bastante diferencia del resto, lo que nos indica que era un equipo de vertiente muy ofensiva, pero no tanto defensiva.