MUCD
20-21

Programación básica en Java con Hadoop

PROCESAMIENTO DE DATOS A GRAN ESCALA

ALEJANDRO CABANA SUÁREZ Y ÓSCAR GÓMEZ BORZDYNSKI

Preguntas a responder

Las rutas que no empiezan por "/" son relativas al directorio de hadoop "/opt/hadoop/".

1. ¿Dónde se crea hdfs? ¿Cómo se puede elegir su localización?

HDFS está constituido por *namenodes* y *datanodes*. El lugar donde un *datanode* almacena sus metadatos se define en la propiedad "dfs.name.dir" del fichero "etc/hadoop/hdfs-site.xml". Por otra parte, el *datanode* guarda sus datos en el directorio indicado en la propiedad "dfs.datanode.data.dir" del mismo fichero. Si no se indica alguna de estas propiedades (teniendo en cuenta que hemos ejecutado todos los comandos como *root*), por defecto se ubican en "/tmp/hadoop-root/dfs/data/".

2. Si estás utilizando hdfs ¿cómo puedes volver a ejecutar WordCount como si fuese *single node*?

Para ejecutar *WordCount* tomando los datos del sistema de ficheros local, podemos eliminar la propiedad "fs.defaultFS" del fichero "etc/hadoop/core-site.xml".

3. En el fragmento de El Quijote

a. ¿Cuáles son las 10 palabras más utilizadas?

Palabra	**Apariciones**
que	3055
de	2816
y	2585
a	1428
la	1423
el	1232
en	1155
no	916
se	753
los	696

b. ¿Cuántas veces aparecen las palabras "el" y "dijo"?

La palabra "el" aparece 1232 veces y "dijo", 272.

Memoria: Modificaciones a *WordCount*

En un primer momento, ejecutamos el programa *WordCount* original. Para hacerlo seguimos los siguientes pasos:

- Creamos un directorio en el que guardar los datos de entrada del programa.

```
> hdfs dfs -mkdir /user/bigdata/entrada_cabana_gomez
```

- Copiamos los datos del sistema de ficheros local al directorio recién creado en hdfs.

```
> hdfs dfs -copyFromLocal quijote.txt
   /user/bigdata/entrada_cabana_gomez/datos.txt
```

- Ejecutamos el *WordCount* de ejemplo.

```
> hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-
mapreduce-examples-2.8.1.jar wordcount
/user/bigdata/entrada_cabana_gomez/datos.txt
/user/bigdata/salida_cabana_gomez
```

- Recuperamos la salida.

```
> hdfs dfs -get /user/bigdata/salida_cabana_gomez/
```

- La salida de este programa se encuentra en la carpeta de entrega con el nombre "salida_cabana_gomez_repaso/".

El problema de esta implementación es que considera como dos palabras distintas a la misma palabra comenzando con mayúscula o con minúscula. Además, si hay un signo de puntuación pegado a la palabra, lo considera parte de ella y distingue entre las distintas combinaciones. Para solucionarlo realizamos las siguientes modificaciones:

- Añadimos nuevos separadores en el Tokenizer, de esta manera, todos los símbolos indicados no se cuentan como parte de las palabras.

```
StringTokenizer(value.toString(),"\n .,:;¡¿!?()[]{}\'\"><")
```

- Pasamos cada token a minúscula en el mapper.

```
itr.nextToken().toLowerCase()
```

Volvemos a realizar los comandos anteriores. Para comprobar la diferencia entre ambas implementaciones buscamos una palabra relevante, como "Quijote". Con la implementación original obtenemos las siguientes variaciones:

Variación	**Apariciones**
QUIJOTE,	1
Quijote!	1
Quijote,todo	1
QUIJOTE	2
Quijote:	12
Quijote;	21
Quijote.	35
Quijote	130
Quijote,	159

Sin embargo, con la modificación realizada obtenemos el siguiente conteo:

Variación	**Apariciones**
quijote	362

Es decir, hemos conseguido reducir los errores respecto a esta palabra. El fichero con el resultado obtenido se encuentra en el directorio “salida_cabana_gomez_ejercicio/” de la entrega.

Memoria: Parte opcional.

El objetivo de nuestra aplicación es describir un *dataset* numérico calculando el mínimo, el máximo y la media de todos ellos. Estos datos pueden ser útiles para hacer análisis estadístico del *dataset* introducido.

Dataset

Los datos que hemos utilizado para el cálculo han sido generados simulando los tiempos de ejecución de una aplicación distribuida. La mayoría de los datos se encuentran en torno a 45 segundos ya que corresponderían a nodos con una carga similar y unos pocos tienen tareas más complejas. Este *dataset* se adjunta a la entrega bajo el nombre "datos_tiempos.txt".

Este *dataset* es un ejemplo de prueba, pero nuestro programa funciona con cualquier conjunto de datos que consista en una serie numérica en formato columnar. Para cargar un nuevo conjunto de datos en hdfs basta con ejecutar el comando -copyFromLocal de hdfs.

Estructura del programa

1. ***Mappers***: Cada *mapper* recibe un subconjunto de los datos y genera tres pares clave-valor que se corresponden con el máximo, el mínimo y la media. Para los dos primeros realiza las operaciones correspondientes y para la media calcula la suma de los datos y el número de ellos.
2. ***Reducer:*** El *reducer* recibe las salidas de los *mappers* y, dependiendo de la clave, realiza distintas operaciones. Para "min" y "max", obtiene el mínimo y el máximo de los valores proporcionado. Para "mean", suma por un lado el número de datos y por otro la suma total, para finalmente obtener la media dividiendo.

Para que los valores con clave "mean" puedan contener dos datos (suma, cuenta), hemos necesitado una clase auxiliar de tipo *Writable* que lo permita: "TwovalueWritable.java".

Conclusiones

Pese a que nuestro programa es muy sencillo, hemos podido experimentar cómo es implementar una aplicación utilizando el modelo de programación *MapReduce* en *Hadoop* y también hemos aprendido a interactuar con un sistema de ficheros distribuido. Este modelo de programación abre las puertas al procesamiento paralelo de grandes volúmenes de información en un entorno distribuido.