UNIVERSITAT POLITÈCNICA DE VALÈNCIA

09/04/2024

Práctica

Análisis de Datos Usando MapReduce en Clusters Hadoop

Germán Moltó – gmolto@dsic.upv.es - Grupo de Grid y Computación de Altas Prestaciones

Práctica

Análisis de Datos Usando MapReduce en Clusters Hadoop

Introducción

MapReduce [1] es un modelo de programación de aplicaciones distribuidas débilmente acopladas que se basa, por un lado, en la definición del problema en dos etapas diferenciadas. La primera etapa (*map*) procesa de forma concurrente los datos de entrada, previamente particionados, obteniendo un conjunto de resultados intermedios. Estos resultados intermedios se agrupan y ordenan adecuadamente para que en una segunda etapa (*reduce*) se produzcan un conjunto de resultados finales que potencialmente pueden ser agrupados para su visualización, tal y como se muestra en la Figura 1.

La práctica pretende utilizar un cluster MapReduce levantado sobre una plataforma Cloud (como Amazon EC2 o un despliegue on-premise basado en OpenNebula) para la ejecución de ejemplos existentes y para el desarrollo de soluciones a problemas propuestos. De esta forma, el alumno puede comprobar por sus propios medios cómo funciona el modelo MapReduce y realizar ejecuciones sobre los conjuntos de datos proporcionados.

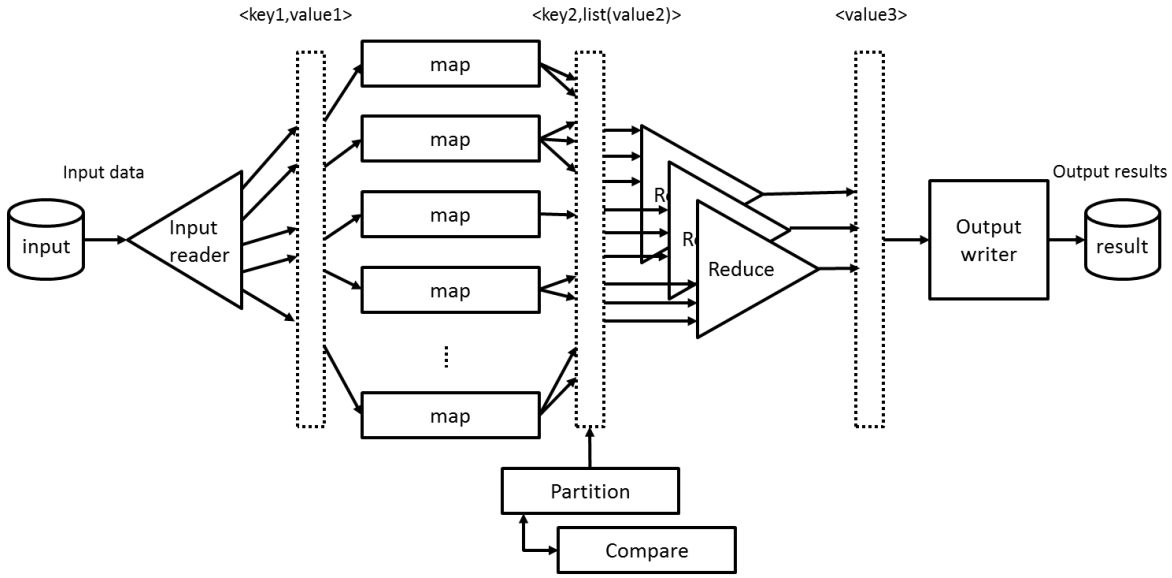

Figura 1. El modelo de programación MapReduce.

Como implementación del modelo de programación MapReduce se utilizará Apache Hadoop [2], una implementación de código abierto formada por varios proyectos (HDFS, HIVE, Pig, Cassandra, etc.). Por ejemplo, HDFS (*Hadoop Distributed File System*) permite construir un sistema de ficheros distribuido sobre el que ejecutar aplicaciones MapReduce.

Accederás a un cluster Hadoop que habrá sido previamente desplegado por el profesor, para que puedas modificar, compilar, ejecutar y desarrollar programas MapReduce que serán ejecutados en dicho cluster. Aunque en esta práctica trabajarás sobre un entorno de prácticas proporcionado por el instructor, al final de este documento, en el Anexo, se describen diferentes mecanismos para desplegar un entorno de trabajo basado en Hadoop, por si quisieras replicar las actividades realizadas en esta práctica en otro entorno.

En primer lugar, accederás al cluster Hadoop para compilar un programa MapReduce (en Java) que permite obtener la frecuencia de aparición de palabras de un texto y lo ejecutarás sobre conjuntos de datos de diferente tamaño. Esto te permitirá conocer de primera mano la gestión de trabajos e interacción con HDFS en un entorno Hadoop. Posteriormente, usarás Hadoop Streaming [3] para el desarrollo de programas MapReduce en Python utilizando un conjunto de datos relativamente grande

de transacciones de ventas de artículos, a partir del cual deberás extraer cierta información mediante el procesado de las transacciones.

En definitiva, la realización de esta práctica te permitirá iniciarte en las técnicas de Big Data usando Apache Hadoop para el procesado de grandes volúmenes de datos mediante el modelo de programación MapReduce.

Resultados de Aprendizaje

Se espera que, una vez finalizada la práctica, el alumno sea capaz de:

- Entender los conceptos básicos que rigen el modelo MapReduce.
- Compilar y ejecutar una aplicación paralela diseñada bajo MapReduce.
- Modificar aplicaciones de ejemplo para la resolución de problemas específicos de MapReduce.
- Entender las características de Hadoop Streaming para el diseño y testeo rápido de programas MapReduce.
- Desarrollar programas de mediana complejidad para el procesado de volúmenes de datos mediante Apache Hadoop sobre diferentes conjuntos de datos.

1. Tu profesor te habrá asignado un número. Los comandos que aparecen en el boletín hacen referencia al usuario alucloud00. Por favor**, substituye en los comandos que veas 00 por tu número** (por ejemplo, si tu número es 06, entonces el usuario que has de utilizar es alucloud06). Recuerda esta regla para facilitar tanto tu trabajo como el del resto de compañeros. Para facilitar tu trabajo, en los comandos se utiliza una variable de entorno llamada ID que se debe resolver a tu número.

Si quieres tener una copia fuera del cluster del código fuente que vas creando, una forma sencilla es seleccionar el código y copiarlo en el portapapeles y luego pegarlo en un documento de texto (o de Word) que tengas abierto en tu equipo local.

Desarrollo

Esta sección describe las actividades a realizar durante la práctica. En primer lugar, se explica la forma de acceso al cluster Hadoop y se introducen los programas de ejemplo a utilizar, sobre los que deberás trabajar para realizar las actividades propuestas en este boletín.

Acceso al Cluster Hadoop

Para poder acceder al cluster Hadoop (en adelante lo denominaremos tan solo cluster, por simplicidad) es necesario un cliente SSH, las credenciales de acceso y la IP del nodo principal del cluster. Esta información te la debe haber suministrado tu instructor.

Conéctate al cluster via SSH. En este boletín asumiremos que el nombre DNS es *hadoop.cursocloudaws.net* pero recuerda utilizar el que te indique el instructor:

```
$ ssh alucloud$ID@hadoop.cursocloudaws.net

aluclou00@hadoopmaster:~$
```

Es posible que el mensaje de bienvenida sea diferente al aquí mostrado.

Ejemplo: Frecuencia de Aparición de Palabras de un Texto

Un ejemplo canónico que a menudo se utiliza para ejemplificar el modelo de programación MapReduce es el de determinar la frecuencia de aparición de las palabras de un determinado texto. Dependiendo del tamaño del conjunto de datos (*dataset*) de entrada, esta tarea puede ser computacionalmente costosa, de ahí la necesidad de utilizar una herramienta como Hadoop para procesar conjuntos de datos masivos.

La Figura 2 resume la estructuración del problema de acuerdo al modelo MapReduce. La entrada (fase de *Input,* en la figura) es un fichero de texto de tamaño arbitrario. Cada Mapper (ejecución de la función *map*) recibirá un conjunto de líneas del fichero de entrada (sub-bloque de texto), se encargará de dividirlas en palabras y emitirá los pares (palabra, 1), indicando que dicha palabra ha sido encontrada una vez en el texto (fase de *Mapping*). Nótese que, en dichos pares, la palabra es la clave y el 1 es el valor. A continuación, se produce la fase de *Shuffle and Sort* encargada de distribuir los pares, ordenados por la clave, que serán recibidos por los Reducers (ejecución de la función *reduce*). Éstos agregarán el número de apariciones de cada palabra en cada sub-bloque de texto para determinar el número total de apariciones de palabras en todo el texto (fase de *Reducing*). El resultado final es, de nuevo, un conjunto de pares clave valor que determina el número de apariciones de cada palabra en el texto. La ejecución se realizará sobre un cluster Hadoop de manera distribuida donde los datos estarán almacenados en HDFS, el sistema de archivos distribuido de Hadoop.

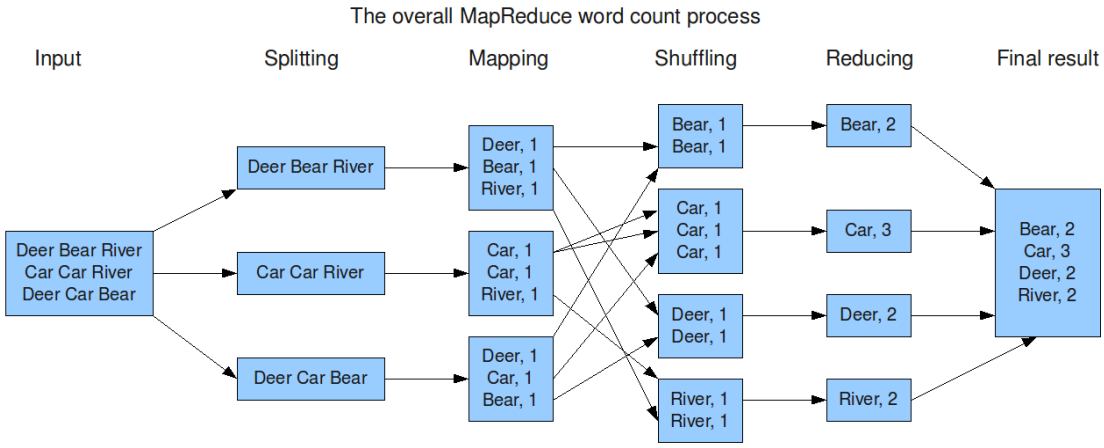

Figura 2. Aplicación de Contar Palabras Mediante el modelo de programación MapReduce.

La solución a este problema ya ha sido implementada y codificada en Java para ser ejecutada sobre Hadoop usando el modelo de programación MapReduce. El código está disponible como uno de los ejemplos de uso de Apache Hadoop. A continuación, procederemos a compilar la aplicación y a ejecutarla sobre dos conjuntos de datos para entender el funcionamiento Hadoop. Concretamente, para este se usarán los siguientes conjuntos de datos: i) un fichero de texto pequeño (12 KB) con la definición de Cloud Computing por parte del NIST, cuya redacción original la puedes encontrar en [5] y ii) un fichero de texto grande (128 MB) que contiene varios libros libres de habla inglesa descargados aleatoriamente de la web del proyecto Gutenberg [6]. Finalmente, se realizarán algunas modificaciones sobre el código existente para resolver problemas análogos.

Sobre la Implementación de WordCount.java

En el Anexo B encontrarás el código completo del fichero WordCount.java, disponible en `[23]`. Aunque no entraremos en todos los detalles sobre dicha clase (puedes encontrar mucha más información al respecto en [3]), sí que resulta interesante ver la estructura y el funcionamiento de las funciones (o métodos, al tratarse de una clase Java) *map* y *reduce*. El objetivo no es tanto que comprendas perfectamente la implementación de dichas funciones sino más bien que tengas una idea

del modelo que subyace a dichas funciones. Es posible observar la implementación de las funciones *map* y *reduce*, que han sido descritas en la sección anterior, editando el fichero WordCount.java. Para ello, puedes utilizar el editor *vim* (o tu editor preferido en GNU/Linux desde línea de comandos). Recuerda que puede abrir un fichero con *vim fichero.java;* pasas al modo inserción con la tecla *i* y puedes guardar los cambios pulsando *ESC* seguido de *:wq*

A continuación, se resumen los principales metodos:

Función MAP	

`public void map(Object key, Text value, Context context) throws IOException, InterruptedException {`
 	
`StringTokenizer itr = new StringTokenizer(value.toString());`
`  while (itr.hasMoreTokens()) {`
`  word.set(itr.nextToken());`
`  context.write(word, one);`
`  }`
`  }`
	Recibe líneas de texto y las divide en sus correspondientes palabras (usando StringTokenizer).
	
	Para cada palabra, emite el par (palabra,1) indicando que dicha palabra ha sido encontrada con frecuencia 1.
	
	Las variables `word` y `one` están definidas como atributos de la clase.

Función REDUCE	

`public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {`
`  int sum = 0;`
`  for (IntWritable val : values) {`
`  sum += val.get();`
`  }`
`  result.set(sum);`
`  context.write(key, result);`
`}`
	Recibe un conjunto de pares (palabra, frecuencia) agrupado por palabra (que es la clave), o lo que es lo mismo, una palabra y una lista de valores de frecuencia para dicha palabra.
	
	Se suma la lista de frecuencias para dicha palabra y se emite el par (palabra, frecuencia) como resultado de la operación.

A continuación, se procede a compilar el código analizado anteriormente para resolver el problema del conteo de la frecuencia de aparición de palabras en un texto.

Preparación y Compilación

Salvo que se indique lo contrario, todos los comandos que se muestran a continuación se ejecutarán desde el directorio $HOME del usuario. Esto permite que los comandos mostrados sean lo más compactos posible. Puedes acudir al directorio $HOME de tu usuario con el comando: `cd`

Para proceder a la compilación, es necesario preparar una estructura de directorios que permita almacenar y compilar el ejemplo. Se deben crear dentro del directorio de usuario dos carpetas llamadas *src* y *clases*, que almacenarán respectivamente el código fuente de la aplicación MapReduce que resuelve el problema y las clases Java ya compiladas. A continuación, hay que copiar en la carpeta *src*, el ejemplo de *WordCount* en Java mencionado anteriormente. Asegúrate de copiar el comando siguiente con todos los guiones, pues al copiar del boletín en PDF es posible que te desaparezca el guión entre mapreduce-project

```
:~$ mkdir -p wordcount/src wordcount/classes
:~$ wget https://raw.githubusercontent.com/apache/hadoop/master/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/WordCount.java -O wordcount/src/WordCount.java
```

. Verifica que se ha descargado correctamente el fichero y que dentro tienes código Java:

```
:~$ cat wordcount/src/WordCount.java
…
FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

De lo contrario, asegúrate de haber indicado la URL correctamente (y no te has olvidado de ningún guion ya que a veces al copiar y pegar al terminal ocurre).

A continuación, modificaremos el paquete al que pertenece la clase Java para asegurarnos de que ejecutamos nuestro código y no el código original del fichero WordCount.java, que puede estar disponible en alguno de los ficheros JAR de la distribución de Hadoop.

```
:~$ sed -i -- 's/org.apache.hadoop.examples/cursocloudaws/g' wordcount/src/WordCount.java
```

El comando anterior simplemente ha cambiado la línea del fichero WordCount.java:

```
package org.apache.hadoop.examples;
```

por la línea:

```
package cursocloudaws;
```

Procedemos a compilar el ejemplo y a empaquetarlo en un fichero .jar (que no es más que un contenedor de clases Java compiladas). Para ello, se deberán realizar las siguientes operaciones sobre el directorio que hemos creado con nuestro nombre. La compilación se realiza con el compilador javac, disponible en JDK 1.6. Se usa la versión 1.6, en lugar de la versión más reciente 1.7, puesto que la instalación de Hadoop en el entorno de prácticas ha sido instalada en base a librerías que fueron compiladas con JDK 1.6. Esto permite evitar problemas de incompatibilidad de versiones.

Para poder compilar el código, es necesario indicar al compilador la localización de las librerías necesarias que contienen las clases referenciadas en dicho código (típicamente el CLASSPATH, en Java). Dicho listado de librerías se puede obtener con el siguiente comando:

```
:~$ hadoop classpath

/opt/hadoop/etc/hadoop:/opt/hadoop/share/hadoop/common/lib/*:/opt/hadoop/sh
are/hadoop/common/*:/opt/hadoop/share/hadoop/hdfs:/opt/hadoop/share/hadoop/
hdfs/lib/*:/opt/hadoop/share/hadoop/hdfs/*:/opt/hadoop/share/hadoop/mapredu
ce/lib/*:/opt/hadoop/share/hadoop/mapreduce/*:/opt/hadoop/share/hadoop/yarn
:/opt/hadoop/share/hadoop/yarn/lib/*:/opt/hadoop/share/hadoop/yarn/*
```

Por ello, procedemos a compilar con el siguiente comando. Asegúrate de utilizar las comillas invertidas: ` en lugar de las comillas simples: ‘ en el comando anterior. De lo contrario, tendrás un problema de compilación:

```
:~$ javac -classpath `hadoop classpath` -d wordcount/classes
wordcount/src/WordCount.java

:~$ ls -lR wordcount/classes/
…
wordcount/classes/:
total 4
drwxrwxr-x 2 alucloud00 alucloud00 4096 Jan 29 18:22 cursocloudaws

wordcount/classes/cursocloudaws:
total 12
-rw-rw-r-- 1 alucloud00 alucloud00 1951 Jan 29 18:45 WordCount.class
-rw-rw-r-- 1 alucloud00 alucloud00 1774 Jan 29 18:45 WordCount$IntSumReducer.class
```

```
-rw-rw-r-- 1 alucloud00 alucloud00 1880 Jan 29 18:45 WordCount$TokenizerMapper.class
```

Si la compilación ha sido correcta no se obtendrá ningún mensaje por pantalla. De lo contrario, habrá que solucionar los posibles errores derivados de la compilación, aunque no debería haber ningún error al tratarse de un ejemplo de prueba. Fíjate que la compilación habrá generado los ficheros .class que contienen las clases Java compiladas (aparecen tres ficheros .class porque el fichero WordCount.java en realidad contenía la definición de tres clases).

A continuación, procedemos a empaquetar las clases compiladas en un fichero .jar, que es un contenedor de ficheros `.class`, es decir, de clases Java compiladas. **No te olvides de incluir el punto al final de la instrucción**. Esto permite empaquetar todos los ficheros del directorio *classes* en el fichero *wordcount.jar*.

```
:~$ jar -cvf wordcount/wordcount.jar -C wordcount/classes .
Added manifest
adding: cursocloudaws/(in = 0) (out= 0)(stored 0%)
adding: cursocloudaws/WordCount$IntSumReducer.class(in = 1774) (out=
759)(deflated 57%)
adding: cursocloudaws/WordCount.class(in = 1951) (out= 1054)(deflated 45%)
adding: cursocloudaws/WordCount$TokenizerMapper.class(in = 1880) (out=
822)(deflated 56%)
:~$ ls -l wordcount
total 12
drwxrwxr-x 3 alucloud00 alucloud00 4096 Dec 17 09:20 classes
drwxrwxr-x 2 alucloud00 alucloud00 4096 Dec 17 09:13 src
-rw-rw-r-- 1 alucloud00 alucloud00 3866 Dec 17 09:30 wordcount.jar
```

Tras este paso, tendremos en el directorio wordcount un fichero "wordcount.jar" que podremos ejecutar. Para su ejecución deberemos suministrarle un directorio (almacenado en HDFS) donde estarán (el o) los ficheros de texto de entrada y un directorio (también en HDFS) donde el programa guardará la salida. Por lo tanto, salvo que los datos de entrada ya hayan sido previamente almacenados en el sistema de archivos HDFS, deberán de ser copiados del sistema de archivos local del entorno de prácticas a HDFS.

Copia de Ficheros de Entrada a HDFS

La interacción con HDFS se puede utilizar el comando *hadoop*. Este comando permite realizar numerosas operaciones (no únicamente interactuar con HDFS), que pueden ser consultadas si ejecutamos dicho comando sin argumentos.

```
:~$ hadoop
Usage: hadoop [--config confdir] COMMAND
where COMMAND is one of:
  namenode -format     format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  mradmin              run a Map-Reduce admin client
  fsck                 run a DFS filesystem checking utility
  fs                   run a generic filesystem user client
  balancer             run a cluster balancing utility
  oiv                  apply the offline fsimage viewer to an fsimage
  fetchdt              fetch a delegation token from the NameNode
  jobtracker           run the MapReduce job Tracker node
  pipes                run a Pipes job
  tasktracker          run a MapReduce task Tracker node
```

```
  historyserver        run job history servers as a standalone daemon
  job                  manipulate MapReduce jobs
  queue                get information regarding JobQueues
  version              print the version
  jar <jar>            run a jar file
  distcp <srcurl> <desturl> copy file or directories recursively
  distcp2 <srcurl> <desturl> DistCp version 2
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  classpath            prints the class path needed to get the
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
```

La interacción con HDFS se realiza por medio del siguiente comando:

```
:~$ hadoop fs
Usage: hadoop fs [generic options]
           [-ls <path>]
           [-lsr <path>]
           [-du <path>]
           [-dus <path>]
           [-count[-q] <path>]
           [-mv <src> <dst>]
           [-cp <src> <dst>]
           [-rm [-skipTrash] <path>]
           [-rmr [-skipTrash] <path>]
           [-expunge]
           [-put <localsrc> ... <dst>]
           [-copyFromLocal <localsrc> ... <dst>]
           [-moveFromLocal <localsrc> ... <dst>]
           [-get [-ignoreCrc] [-crc] <src> <localdst>]
           [-getmerge <src> <localdst> [addnl]]
           [-cat <src>]
           [-text <src>]
           [-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>]
           [-moveToLocal [-crc] <src> <localdst>]
           [-mkdir <path>]
           [-setrep [-R] [-w] <rep> <path/file>]
           [-touchz <path>]
           [-test -[ezd] <path>]
           [-stat [format] <path>]
           [-tail [-f] <file>]
           [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
           [-chown [-R] [OWNER][:[GROUP]] PATH...]
           [-chgrp [-R] GROUP PATH...]
           [-help [cmd]]

Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|jobtracker:port>    specify a job tracker
-files <comma separated list of files>      specify comma separated files to be
copied to the map reduce cluster
-libjars <comma separated list of jars>     specify comma separated jar files to
include in the classpath.
-archives <comma separated list of archives>    specify comma separated archives to
be unarchived on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]
```

Si tienes cierta experiencia en manejo de sistemas GNU/Linux habrás podido comprobar el paralelismo existente entre los comandos de un sistema GNU/Linux y los de HDFS. Esto facilita la interacción con HDFS. A continuación, se resumen algunos de los comandos de interacción con HDFS más utilizados y su posible equivalencia en un sistema GNU/Linux.

Comando para HDFS	Descripción	Comando en GNU/Linux
hadoop fs -ls	Lista los ficheros del directorio de trabajo en HDFS del usuario (/user/alucloud$ID)	ls
hadoop fs -mkdir	Crea un directorio en HDFS. El usuario debe tener permisos de escritura en el directorio padre.	mkdir
hadoop fs -put	Copia un fichero en el sistema de archivos local a HDFS	cp
hadoop fs -get	Copia un fichero almacenado en HDFS al sistema de archivos local	cp
hadoop fs -text	Muestra el contenido de un fichero de texto situado en HDFS. El usuario debe tener permisos de lectura sobre el fichero y de acceso sobre la jerarquía de directorios que lo contiene.	cat
hadoop fs -rmr	Elimina de forma recursiva un directorio en HDFS (el directorio y todos los ficheros y directorios que éste contenga).	rm –rf

A continuación, se muestra un ejemplo de uso para que adquieras cierta soltura en la interacción con HDFS. Este ejemplo es idempotente, lo que significa que si sigues todos los pasos dejarás el sistema de archivos HDFS en exactamente el mismo estado que antes de realizar estos comandos. En el ejemplo se muestra el contenido de directorios, se crea una carpeta, se copia el fichero `/etc/hosts` al sistema de archivos HDFS, se muestra el contenido del fichero y, finalmente, se borra recursivamente el directorio creado.

```
:~$ hadoop fs -ls
(No obtiene ninguna salida porque tu directorio de trabajo en HDFS, que es
/user/alucloud$ID, está vacio. Se muestra la instrucción equivalente:)
:~$ hadoop fs -ls /user/alucloud$ID
:~$ hadoop fs -mkdir folder1
:~$ hadoop fs -ls
Found 1 items
drwxr-xr-x   - alucloud00 supergroup          0 2018-12-17 10:01
/user/alucloud00/folder1
:~$ hadoop fs -put /etc/hosts folder1
(Copia el fichero /etc/hosts en el sistema de archivos local a
/usr/alucloud$ID/folder1/hosts en HDFS)
:~$ hadoop fs -ls folder1
Found 1 items
-rw-r--r--   3 alucloud00 supergroup        135 2018-12-17 10:09
/user/alucloud00/folder1/hosts
:~$ hadoop fs -text folder1/hosts
127.0.0.1 localhost localhost.localdomain
10.210.138.231 hadoopmaster.localdomain hadoopmaster
10.236.161.157 wn-0.localdomain wn-0
:~$ hadoop fs -rm -r folder1
Deleted folder1
```

Habrás podido comprobar como los comandos casi siempre utilizan rutas relativas, pero nada impide especificar rutas absolutas. Recuerda que el directorio de trabajo por defecto del usuario en HDFS es `/user/alucloud$ID`.

Una vez conocido los aspectos básicos de HDFS, procederemos a copiar el fichero con la definición de Cloud Computing del NIST (disponible en `/opt/datasets/nistdef.txt`) a HDFS. Lo haremos sobre un directorio de nueva creación en el directorio de trabajo del usuario Posteriormente, verificaremos que dicho fichero ha sido copiado correctamente realizando una lectura del mismo desde HDFS.

```
:~$ hadoop fs -mkdir wordcount wordcount/input
(Se crean dos directorios con una sola instrucción)
:~$ hadoop fs -put /opt/datasets/nistdef.txt wordcount/input
:~$ hadoop fs -ls wordcount
Found 1 items
drwxr-xr-x   - alucloud00 supergroup          0 2018-12-17 10:20
/user/alucloud00/wordcount/input
:~$ hadoop fs -text wordcount/input/nistdef.txt
The NIST Definition of Cloud Computing
Special Publication 800-145
...
```

Ejecución y Monitorización del Trabajo MapReduce

Una vez hayamos copiado los ficheros apropiados al sistema de archivos distribuido de Hadoop (HDFS), podemos pasar a ejecutar el ejemplo mediante el comando `hadoop jar`. Si lo ejecutamos sin argumentos, comprobaremos la salida que necesita nuestro programa.

```
:~$ hadoop jar wordcount/wordcount.jar cursocloudaws.WordCount
Usage: wordcount <in> <out>
```

Fíjate cómo le indicamos el nombre completo de la clase WordCount, ya que pertenece al paquete `cursocloudaws`. Además, el programa espera dos parámetros, que son rutas en HDFS. El primer parámetro (in) es la ruta en HDFS al fichero de texto de entrada. Tanto el directorio en HDFS como el fichero ya han sido creados y copiados respectivamente anteriormente, de manera que el fichero de entrada reside en HDFS en /user/alucloud$ID/wordcount/input/nistdef.txt. El segundo parámetro (out) es una ruta a un directorio HDFS que el propio trabajo MapReduce creará y donde se almacenarán los resultados de la ejecución. Dicho directorio en HDFS debe existir previamente o Hadoop se negará a ejecutar el trabajo. Esto se hace para evitar sobreescrituras accidentales, pudiendo así perder datos de trabajos que pueden requerir gran tiempo de ejecución.

```
:~$ hadoop jar wordcount/wordcount.jar cursocloudaws.WordCount
wordcount/input/nistdef.txt wordcount/output
2019-03-08 13:51:01,307 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2019-03-08 13:51:01,422 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2019-03-08 13:51:01,422 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2019-03-08 13:51:01,979 INFO input.FileInputFormat: Total input files to process :
1
…
       File Input Format Counters
              Bytes Read=10557
       File Output Format Counters
              Bytes Written=6905
```

Si obtienes algún problema, asegúrate de que la línea de comandos que has introducido no tiene espacios adicionales (a veces ocurre al copiar del PDF).

Tras unos segundos (la ejecución debería tardar menos de un minuto) se obtendrá el resumen de la ejecución del trabajo. Cada trabajo (*job*) recibe un identificador único (en el caso anterior es

job_local1017277236_0001) y en el registro se observa tanto el porcentaje de progreso de los Mappers como de los Reducers.

Recuperación de la Salida del Trabajo

Si la ejecución del trabajo ha sido satisfactoria, los resultados de salida habrán sido escritos en el fichero HDFS en wordcount/output. Analicemos el contenido de dicho directorio:

```
:~$ hadoop fs -ls wordcount/output
Found 3 items
-rw-r--r--   3 alucloud00 supergroup          0 2018-12-16 15:11
/user/alucloud00/wordcount/output/_SUCCESS
-rw-r--r--   - alucloud00 supergroup          0 2018-12-16 15:10
/user/alucloud00/wordcount/output/part-r-00000
```

Se observa que existen dos ficheros. El fichero *_SUCCESS* se escribe en el directorio de salida para indicar que el trabajo se ha ejecutado satisfactoriamente. Finalmente, el fichero *part-r-00000* es el que contiene la salida del trabajo.

Para poder recuperar la salida del trabajo, podremos utilizar el comando `hadoop fs -text` mostrado previamente.

```
:~$ hadoop fs -text wordcount/output/part-r-00000
(FISMA) 1
(ITL)   1
(IaaS). 1
(NIST)  3
(OMB)   1
(PaaS). 1
(SaaS). 1
(September      1
(e.g.,  9
(private,       1
.1.     1
.2.     1
.Securing       1
1  2
1. 1
1.1     1
1.2     1
1.3     1
107-347. 1
2002,   1
2011    1
…
with    7
without 1
workstations).  1
would   1
Securing  1
```

Como la salida es un fichero de texto con muchas líneas, si quieres verlas una a una tienes varias opciones. A continuación, se indican tres formas:

1. Usar el comando *more* para ir mostrando un bloque de líneas cada vez:
   - hadoop fs -text wordcount/output/part-r-00000 | more
2. Redirigir la salida estándar a un fichero de texto:

- hadoop fs -text wordcount/output/part-r-00000 > wc.txt
- more wc.txt

3. Copiar el fichero de HDFS al sistema de archivos local
   - hadoop fs -get wordcount/output/part-r-00000 wc2.txt

Si quieres volver a ejecutar el ejemplo, por ejemplo utilizando otro fichero de entrada, es necesario eliminar previamente el directorio de salida "wordcount/output". Para ello, debes utilizar el comando `hadoop fs -rmr`, según se muestra en el ejemplo siguiente. Sin embargo, **te recomiendo que no elimines la salida del trabajo** puesto que será utilizada en la siguiente sección.

```
:~$ hadoop fs -rm -r wordcount/output
Deleted hdfs://ip-10-144-19-
37.ec2.internal:50001/user/root/alucloud00/output
```

Alternativamente, es posible volver a ejecutar el trabajo MapReduce especificando otro directorio en HDFS para guardar la salida:

```
:~$ hadoop jar wordcount/wordcount.jar cursocloudaws.WordCount
wordcount/input/nistdef.txt wordcount/output2
13/12/17 11:08:23 INFO input.FileInputFormat: Total input paths to process : 1
13/12/17 11:08:23 INFO util.NativeCodeLoader: Loaded the native-hadoop library
```

Ten en cuenta que esto provocará tener varias carpetas en HDFS con la salida de las diferentes ejecuciones del trabajo.

Modificación del ejemplo

Como habrás podido comprobar, el programa WordCount diferencia entre mayúsculas y minúsculas, con lo que el número de ocurrencias para las palabras "This" y "this", por ejemplo, se contabiliza de forma separada. Además, únicamente separa palabras por espacios o finales de línea, con lo que palabras como "cloud" y "cloud." se contabilizan como diferentes. Esto es muy sencillo de ver al analizar la salida del programa:

```
:~$ hadoop fs -text wordcount/output/part-r-00000 | grep loud
Cloud  6
cloud  24
cloud. 4
clouds).  1
:~$ hadoop fs -text wordcount/output/part-r-00000 | grep his
This  5
this  7
```

Si obtienes un mensaje indicando que el fichero no existe lo más probable es que hayas eliminado el directorio `wordcount/output`. Deberás volver a ejecutar el trabajo MapReduce para volver a recrear el directorio de salida y realizar las actividades propuestas en esta sección.

A continuación, se propone modificar el ejemplo para que no se incluya esta distinción, convirtiendo en la fase de "map" todas las palabras a minúsculas y además que separe las palabras teniendo en cuenta diversos separadores. Además, sería deseable que la salida únicamente incluyera las palabras más repetidas, por ejemplo sólo las que se repitan cinco o más veces.

Actividad 1 (No Evaluable)
Si tienes cierta experiencia en Java, dedica unos minutos a tratar de implementar la solución antes ver una propuesta de implementación. Como pista, deberás modificar el constructor de StringTokenizer para que tenga en cuenta los signos de puntuación (y comillas, signos de interrogación, corchetes, paréntesis, etc.) como elementos de separación de palabras (no únicamente el espacio, que es el comportamiento por defecto). Además, deberás usar un método de la clase String para pasar a minúsculas. Finalmente, el filtro de incluir únicamente las palabras que se repitan cinco o más veces debería ser implementado en la fase de reducción. Puedes encontrar la documentación de las clases Java en: http://download.oracle.com/javase/6/docs/api/

En la siguiente página encontrarás una propuesta de solución para la Actividad 1 pero te recomiendo que dediques unos minutos a tratar de modificar el ejemplo por tu cuenta.

Para resolver la Actividad 1 se propone alterar el código de las funciones *map* y *reduce* de WordCount.java con las siguientes modificaciones (mostradas en negrita y subrayadas). Se puede utilizar cualquier editor para la modificación del ejemplo. Por ejemplo, lo puedes abrir con vim:

```
:~$ vim wordcount/src/WordCount.java
```

Recuerda que puedes entrar en modo edición pulsando '*i*' y guardar pulsando en '*Esc*' y luego '*:w*', pudiendo salir con '*:q*'.

Modificación en la función *map* para considerar como separadores no únicamente el espacio sino los signos de puntuación (y comillas, signos de interrogación, corchetes, paréntesis, etc.) y para pasar las palabras a minúsculas:

```
  public void map(Object key, Text value, Context context)
                          throws IOException, InterruptedException {
   StringTokenizer itr = new StringTokenizer(value.toString(),
" \"\t\n\r\f,.:;?![]()'");
   while (itr.hasMoreTokens()) {
     word.set(itr.nextToken().toLowerCase());
     context.write(word, one);
    }
  }
```

Modificación en la función *reduce*:

```
public void reduce(Text key, Iterable<IntWritable> values,
   Context context) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);

     if (sum>4) context.write(key, result);
    }
  }
```

Tras las modificaciones, compilamos, empaquetamos (en un archivo .jar) y ejecutamos el comando de la misma manera que realizó con el ejemplo original. No es necesario volver a copiar el fichero de entrada a HDFS. Se resumen los comandos utilizados y se muestra un extracto de la salida del trabajo. Acuérdate que es necesario cambiar el directorio de salida de HDFS si este ya existe previamente (en este ejemplo se utilizará `wordcount/output3`). Los comandos que se muestran a continuación se ejecutan desde $HOME.

```
:~$ javac -classpath `hadoop classpath` -d wordcount/classes
wordcount/src/WordCount.java
:~$ jar -cvf wordcount/wordcount.jar -C wordcount/classes .
```

```
:~$ hadoop jar wordcount/wordcount.jar cursocloudaws.WordCount
wordcount/input/nistdef.txt wordcount/output3
:~$ hadoop fs -text wordcount/output3/part-r-00000 | more
1       6
a       27
abstraction 5
and     73
applications        7
are     6
as      11
be      14
but     5
by      15
can     7
capabilities        5
cloud 34
…
technology  13
that  10
the     83
this  12
to      30
use     10
with  7
```

Como habrás podido comprobar, los resultados obtenidos son los esperados, puesto que ahora la separación de palabras se realiza correctamente, no hay distinción entre mayúsculas y minúsculas y, además, se muestran únicamente las palabras que tienen más de 4 repeticiones en el texto.

Frecuencia de Aparición de Palabras en un Conjunto de Datos Mayor

Una vez comprendido el flujo de trabajo de MapReduce sobre un conjunto de datos pequeño (la definición de Cloud Computing del NIST), en esta sección se utilizará un conjunto de datos mayor. Se dispone de un conjunto de libros en formato texto escritos en inglés descargados de la web del Proyecto Gutenberg [6]. Los libros han sido concatenados hasta formar un gran fichero de texto de 129 MB. Dicho fichero ya ha sido pre-cargado en HDFS en la siguiente ruta: /datasets/gutenberg/gutenberg.txt

```
:~$ hadoop fs -ls /datasets/gutenberg/gutenberg.txt
Found 1 items
-rw-r--r--   3 root supergroup  134217728 2014-01-16 12:18
/datasets/gutenberg/gutenberg.txt
```

Actividad 2 (No Evaluable)

A estas alturas de la práctica seguro que ya sabes qué comando(s) utilizar para ejecutar un trabajo MapReduce de contar la frecuencia de aparición de palabras sobre dicho fichero de texto. Dedica unos minutos a practicar con dicho conjunto de datos para averiguar qué palabras son las más frecuentes en dichos libros. Ten en cuenta que la ejecución de este trabajo puede durar más de un minuto.

La solución está justo a continuación.

A continuación, se procede a ejecutar el trabajo MapReduce sobre el nuevo conjunto de datos. Fíjate que no es necesario introducir ninguna modificación en el código (que incorporará la modificación realizada en la actividad 1), sino tan solo indicar el nuevo fichero de entrada. Previamente, se creará un nuevo directorio en HDFS para almacenar el directorio con los resultados de salida, que creará el propio trabajo MapReduce. Fíjate como, a diferencia del caso anterior, evitamos copiar el fichero de

texto a nuestro directorio de usuario en HDFS ya que, cómo únicamente se utilizará para leer de él, no es necesario tenerlo replicado en las diferentes carpetas de usuarios en HDFS.

```
:~$ hadoop fs -mkdir gutenberg
:~$ hadoop jar wordcount/wordcount.jar cursocloudaws.WordCount
/datasets/gutenberg/gutenberg.txt gutenberg/output
2019-03-08 14:06:00,104 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2019-03-08 14:06:00,181 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2019-03-08 14:06:00,181 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2019-03-08 14:06:00,536 INFO input.FileInputFormat: Total input files to process :
1
2019-03-08 14:06:00,613 INFO mapreduce.JobSubmitter: number of splits:2
2019-03-08 14:06:00,764 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_local46433521_0001
2019-03-08 14:06:00,764 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-03-08 14:06:00,886 INFO mapreduce.Job: The url to track the job:
http://localhost:8080/
2019-03-08 14:06:00,887 INFO mapreduce.Job: Running job: job_local46433521_0001
2019-03-08 14:06:00,889 INFO mapred.LocalJobRunner: OutputCommitter set in config
null
2019-03-08 14:06:00,900 INFO output.FileOutputCommitter: File Output Committer
Algorithm version is 2
2019-03-08 14:06:00,907 INFO output.FileOutputCommitter: FileOutputCommitter skip
cleanup _temporary folders under output directory:false, ignore cleanup failures:
false
…
File Input Format Counters
Bytes Read=159154492
File Output Format Counters
Bytes Written=83359435

:~$ hadoop fs -text gutenberg/output/part-r-00000 | more
      7
#     5
#--------------------------------------------------------------------------
-     109
#1    6
$1    33
$10   11
$100  25
$15   8
$2    255
$20   6
$25   9
$3    9
$30   7
…
*when 10
*whether    7
*whit 5
*why  6
*wicked     6
*wickedness 7
*will 7
*with 26
*without    14
*work 8
```

```
*worthy     13 …
accident    240
accidental  18
accidentally      14
accidents   19
acclamations      6
…
□□□g  8
□□□□  54
□□□□□ 7
```

Fíjate que el resultado dependerá principalmente de la configuración del StringTokenizer en la operación de map, por lo que es posible que el resultado que obtengas no sea el mismo que el mostrado en el ejemplo. En concreto, fíjate como los primeros resultados ni siquiera son palabras habituales. De hecho, podrías filtrar en la fase *map* para evitar emitir cadenas que no sean palabras (por ejemplo, para evitar los números).

Por otra parte, quizá te sorprenda ver que la fase de Reduce comienza a ejecutarse antes de finalizar la fase Map. En realidad, antes de que la fase de Reduce comience se realiza la fase de Shuffle and Sort, y dicha fase está contabilizada en los logs como parte de la fase Reduce. El Shuffle and Sort se realiza en paralelo en cuanto hay datos disponibles tras algunos de los Mappers, para reducir la sobrecarga [9].

Actividad 3 (No Evaluable)
Introduce las modificaciones necesarias en WordCount.java para evitar que la salida del programa MapReduce incluya cadenas de texto que no sean consideradas como palabras (por ejemplo, las que empiezan por caracteres como asterisco (*). No es necesario que introduzcas aproximaciones sofisticadas. Céntrate por ejemplo filtrar algunas las cadenas que empiezan por *, usando el método startsWith de la clase String. Esto te ayudará a practicar con el proceso de edición, compilación y ejecución de programas MapReduce. Si no tienes experiencia con Java, puedes obviar esta actividad y céntrate en las siguientes, donde se utilizará Hadoop Streaming.

Hadoop Streaming – Conjunto de Datos de Compras

Hadoop Streaming [8] es una característica de Apache Hadoop que permite escribir programas bajo el modelo de programación MapReduce independientemente del lenguaje de programación utilizado. Básicamente consiste en escribir programas que implementan la función *map* y la función *reduce* de manera que lean texto de la entrada estándar y escriban el resultado por la salida estándar, de ahí que el lenguaje en el que estén implementadas no sea relevante.

En esta sección veremos el uso de Hadoop Streaming como un mecanismo simplificado para la creación de programas bajo el modelo de programación MapReduce en lenguajes diferentes a Java. En este caso concreto se usará Python, aunque otros lenguajes como Ruby podrían ser utilizados en su lugar. Para ello, se usará un conjunto de datos obtenido de [10] que incluye un registro de las compras realizadas en las diferentes tiendas de una cadena con el siguiente formato.

```
2012-01-01  09:00 San Jose    Men's Clothing    214.05      Amex
2012-01-01  09:00 Fort Worth  Women's Clothing  153.57      Visa
2012-01-01  09:00 San Diego   Music             66.08       Cash
2012-01-01  09:00 Pittsburgh  Pet Supplies      493.51      Discover
2012-01-01  09:00 Omaha       Kid's Clothing    235.63      MasterCard
...
```

Cada línea representa una compra (transacción) donde cada uno de los seis campos (columnas separadas por un tabulador) incluye información sobre la misma. En este orden, fecha de la compra (*date*) (1), hora de la compra (*time*) (2), ciudad de la tienda (*store name*) (3), categoría de producto (*item description*) (4), importe (*cost*) (5) y método de pago (*method of payment*) (6). Dicho fichero está disponible en el entorno de realización de prácticas en `/opt/datasets/purchases.txt` y contiene 4.138.476 transacciones (líneas de texto). Dicho fichero ya ha sido pre-cargado en el sistema de archivos HDFS del cluster Hadoop sobre el que se realizará la práctica, en la ruta `/datasets/purchases/purchases.txt`

Puedes asomarte a las cinco primeras líneas del fichero con el comando:

```
:~$ hadoop fs -text /datasets/purchases/purchases.txt | head -n 5
2012-01-01  09:00 San Jose    Men's Clothing    214.05      Amex
2012-01-01  09:00 Fort Worth  Women's Clothing  153.57      Visa
2012-01-01  09:00 San Diego   Music 66.08 Cash
2012-01-01  09:00 Pittsburgh  Pet Supplies      493.51      Discover
2012-01-01  09:00 Omaha Children's Clothing     235.63      MasterCard
```

Ventas Totales por Tienda

Supón que quieres obtener el importe total de las ventas realizadas en cada una de las tiendas, considerando todas las transacciones del conjunto de datos. Dado que el conjunto de datos es elevado, se propone realizar una aplicación MapReduce para que múltiples nodos puedan trabajar de forma colaborativa sobre el problema y así obtener el resultado de forma rápida.

Para este ejemplo, en lugar de construir un programa en Java, que es lo que se hizo en el ejemplo anterior, ahora se usará Hadoop Streaming para permitir desarrollar los programas usando Python. Para ello es necesario construir un programa que implemente la funcionalidad de los Mappers (lo que en código Java correspondía a la función *map)* y otro programa que implemente la funcionalidad de los Reducers (correspondiente a la función *reduce*). Ambos programas leerán de la entrada estándar y escribirán su resultado por la salida estándar.

A continuación, se muestra el código de ambos programas (`mapper.py` y `reducer.py`), obtenidos de [10] y disponibles en el entorno de prácticas en `/opt/hadoopcode`, para conseguir obtener la suma de todos los importes de las ventas realizadas, para cada una de las tiendas.

Programa mapper.py

```
#!/usr/bin/python

# Format of each line is:
# date\ttime\tstore name\titem description\tcost\tmethod of payment
#
# We want elements 2 (store name) and 4 (cost) to be written
# to standard output, separated by a tab

import sys

for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print("{0}\t{1}".format(store, cost))
```

El programa `mapper.py` lee por la entrada estándar líneas de texto con seis columnas (o campos) separadas por un tabulador en el siguiente orden (Fecha Tiempo Tienda Ítem Coste Pago). Para cada una de las líneas se comprueba si el número de campos es el correcto, en cuyo caso asigna cada uno de los campos a una variable identificada. Para cada línea se emite por la salida estándar el par tienda e importe separado por un tabulador. Este es el resultado de la fase de *map* en un programa MapReduce, donde la clave es el nombre de la tienda (que en realidad coincide con el nombre de la ciudad en la que se ubica la tienda).

A continuación, se muestra el programa `reducer.py`, encargado de agregar los resultados obtenidos a partir de la ejecución de `mapper.py`, que estarán ordenados por clave.

Programa reducer.py

```
#!/usr/bin/python

import sys

salesTotal = 0
oldKey = None

# Loop around the data
# It will be in the format key\tval
# Where key is the store name, val is the sale amount (cost)
#
# All the sales for a particular store will be presented,
# then the key will change and we'll be dealing with the next store

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisSale = data_mapped

    if oldKey and oldKey != thisKey:
        print(oldKey, "\t", salesTotal)
        oldKey = thisKey
```

```
        salesTotal = 0

    oldKey = thisKey
    salesTotal += float(thisSale)

if oldKey != None:
    print(oldKey, "\t", salesTotal)
```

El script `reducer.py` espera recibir pares clave, valor separados por un tabulador, donde la clave es el nombre (que coincide con la ubicación) de la tienda y el valor es el importe de la venta. Dichos pares se reciben de forma ordenada por clave dado que pasan por la fase *Shuffle & Sort* antes de llegar a la fase de Reduce, implementada en el script `reducer.py`. Eso significa que dicho script recibe una entrada como la siguiente:

```
:~$ cat /opt/datasets/purchases.txt | head -n 300 |
/opt/hadoopcode/mapper.py | sort -k 1,1 | head -n 20
Albuquerque  149.94
Albuquerque  440.7
Albuquerque  484.24
Anaheim      274.68
Anaheim      341.43
Anaheim      447.8
Anaheim      48.34
Anaheim      66.07
Anchorage    22.36
Anchorage    298.86
Anchorage    368.42
Anchorage    390.2
Anchorage    6.38
Arlington    25.46
…
```

Por ello, el script itera sobre las líneas de la entrada estándar, pero, a diferencia de la implementación en Java, ahora es necesario guardar algo de estado al procesar cada grupo de claves (de la misma tienda). Dado que las líneas están ordenadas por el nombre de la tienda (clave), hay que guardarse la última clave vista para detectar cuando se recibe una clave de un nuevo grupo de tiendas. De esta manera, si se detecta una nueva clave (nueva tienda) distinta a la guardada, entonces sabemos que comienza un nuevo bloque de datos para una nueva tienda. En ese caso es cuando hay que emitir la línea por la salida estándar correspondiente a la tienda y a la suma total de los importes de las ventas de dicha tienda. La última línea del script asegura la escritura de la línea correspondiente al procesado del bloque de datos de la última tienda.

Una de las ventajas de Hadoop Streaming es poder verificar la funcionalidad de los programas MapReduce sin necesidad de ejecutar trabajos sobre el propio cluster Hadoop, sobre un conjunto de datos de tamaño reducido. Esto facilita el proceso de creación y depuración de programas MapReduce.

A continuación, procedemos a copiar los ficheros `mapper.py` y `reducer.py` al directorio de trabajo de nuestro usuario. Posteriormente, se indica el comando en GNU/Linux para simular la ejecución de un trabajo MapReduce usando comandos propios del sistema. Como solo se trata de un ejemplo usaremos las 50 primeras líneas del fichero de entrada y únicamente mostraremos las primeras 20 líneas del resultado. Como siempre, los comandos se ejecutan desde el directorio HOME del usuario:

```
:~$ mkdir purchases
:~$ cp /opt/hadoopcode/*.py purchases
:~$ cat /opt/datasets/purchases.txt | head -n 50 | purchases/mapper.py |
sort -k 1,1 | purchases/reducer.py | head -n 20
Anchorage   327.6
Aurora      117.81
Austin      1176.98
Boston      418.94
Buffalo     483.82
Chandler    758.17
Chicago     31.08
Corpus Christi    25.38
Fort Wayne  370.55
Fort Worth  367.45
Fremont     222.61
Fresno      466.64
Greensboro  290.82
Honolulu    345.18
Houston     309.16
Indianapolis      135.96
Las Vegas   146.65
Lincoln     136.9
Madison     16.78
Minneapolis       182.05
```

El comando anterior permite obtener las 50 primeras líneas del fichero `/opt/datasets/purchases.txt` y pasárselas por la entrada estándar al programa `mapper.py` que emitirá los pares (tienda, importe) por la salida estándar. Dichos pares se ordenan mediante el comando `sort` (simulando así la fase *Shuffle & Sort* de MapReduce). Utilizamos `sort -k 1,1` para que la ordenación se haga exclusivamente por la primera columna (la clave). Los pares estarán ordenados por clave, que es el nombre de la tienda y éstos servirán de entrada al programa `reducer.py`, que se encargará de sumar los costes o importes asociados de cada tienda para emitir por la salida estándar los pares (tienda, importe total). Los resultados saldrán ordenados por la clave, que es el nombre de la tienda.

Fíjate que, al poder simular la ejecución de un trabajo MapReduce es muy sencillo detectar cualquier problema en el código, sin necesidad de esperar a lanzar los trabajos al cluster Hadoop. Además, estos mismos programas pueden lanzarse a ejecución en el cluster para poder trabajar con conjuntos de datos mayores. Para ello, en primer lugar, crearemos el directorio `purchases` en HDFS para almacenar los ficheros de salida de los trabajos. A continuación, usaremos el script `hs` (puedes averiguar qué hace dicho script viendo su código en `/usr/local/bin/hs`) para lanzar un trabajo MapReduce usando Hadoop Streaming al cluster Hadoop. Dicho script recibe como argumentos el fichero fuente del mappper y del reducer, el directorio en HDFS que contiene el fichero de texto de entrada para el proceso y un directorio en HDFS inexistente para poder escribir los resultados de salida de la ejecución del trabajo.

```
:~$ hadoop fs -mkdir purchases
:~$ hs purchases/mapper.py purchases/reducer.py /datasets/purchases purchases/output

packageJobJar: [mapper.py, reducer.py, /tmp/hadoop-alucloud00/hadoop-
unjar302293408738235597/] [] /tmp/streamjob3075393850699501467.jar
tmpDir=null
13/12/19 18:10:03 INFO util.NativeCodeLoader: Loaded the native-hadoop
library
13/12/19 18:10:03 WARN snappy.LoadSnappy: Snappy native library not loaded
13/12/19 18:10:03 INFO mapred.FileInputFormat: Total input paths to
```

```
process : 1
13/12/19 18:10:04 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-
alucloud00/mapred/local]
13/12/19 18:10:04 INFO streaming.StreamJob: Running job:
job_201812161039_0028
13/12/19 18:10:04 INFO streaming.StreamJob: To kill this job, run:
13/12/19 18:10:04 INFO streaming.StreamJob: /opt/hadoop-
1.2.1/libexec/../bin/hadoop job  -
Dmapred.job.tracker=hadoopmaster.localdomain:9001 -kill
job_201812161039_0028
…
13/12/19 18:17:12 INFO streaming.StreamJob:  map 100%  reduce 100%
13/12/19 18:18:03 INFO streaming.StreamJob: Job complete:
job_201812161039_0028
13/12/19 18:18:03 INFO streaming.StreamJob: Output: purchases/output
```

La ejecución de este trabajo puede llevar algunos minutos, dependiendo del tamaño del cluster. El comando *hs* no es específico de la instalación de Apache Hadoop sino que es un script de ayuda, inspirado en el usado en [10], que únicamente se encarga de invocar la ejecución de un trabajo con Hadoop Streaming.

```
:~$ more `which hs`
/opt/hadoop/bin/hadoop    jar    /opt/hadoop/share/hadoop/tools/lib/hadoop-
streaming-*.jar -mapper $1 -reducer $2 -file $1 -file $2 -input $3 -output
$4
```

Podemos ver las primeras 20 líneas de la salida del trabajo, como ya sabes, con el siguiente comando:

```
:~$ hadoop fs -text purchases/output/part-00000 | head -n 20
Albuquerque         10052311.420000063
Anaheim       10076416.359999852
Anchorage     9933500.400000015
Arlington     10072207.969999991
Atlanta       9997146.699999973
Aurora        9992970.920000017
Austin        10057158.900000056
…
Chula Vista         9974951.340000039
Cincinnati    10139505.740000065
```

Quizá te sorprenda que el fichero de salida se llame part-00000 en lugar de part-r-00000. Esto es así debido al uso del antiguo API Hadoop por parte de Hadoop Streaming, de ahí el cambio en la nomenclatura del fichero de salida [11].

Ventas Agregadas por Categoría de Producto a través de Todas las Tiendas

Una vez visto el ejemplo anterior y entendido el esquema de funcionamiento de MapReduce, deberías poder crear un nuevo programa para obtener el número total de ventas agrupadas por categorías de producto a través de todas las tiendas.

Actividad 4 (Evaluable). Dedica unos minutos a tratar de crear dos nuevos scripts `mapper.py` y `reducer.py`, para obtener la suma total de ventas para cada categoría de producto, independientemente de la tienda en la que se haya realizado la compra. Trata de aprovechar al máximo el código del ejemplo anterior por lo que te recomiendo que partas de los scripts anteriores para introducir las modificaciones necesarias para resolver el problema.

> Te sugiero que crees una carpeta para cada actividad que contenga el correspondiente fichero mappper.py y reducer.py (por ejemplo: actividad4/mapper.py, actividad4/reducer.py, etc.)

Para no sobrecargar el cluster Hadoop con numerosos trabajos MapReduce, puedes hacer las pruebas iniciales de los scripts con un subconjunto de líneas del fichero usando las herramientas propias del sistema. Por ejemplo, el siguiente comando permite verificar la funcionalidad de los scripts usando las primeras 1000 líneas del fichero de entrada y mostrando únicamente las 10 primeras líneas de resultados. Te aconsejo que no realices las pruebas con un número muy superior de líneas del fichero puesto que los procesos asociados se ejecutan en el nodo principal del cluster, al que pueden estar conectados el resto de alumnos y esto ralentizaría la ejecución de dicho nodo.

```
:~$ cat /opt/datasets/purchases.txt | head -n 1000 | ./mapper.py | sort -k 1,1 | ./reducer.py | head -n 10
Baby  12412.82
Books       13769.89
Cameras     13138.99
CDs   15183.28
Children's Clothing     13310.85
Computers   11632.52
Consumer Electronics    14587.07
Crafts      14329.82
DVDs  13050.68
Garden      13947.95
```

Si los resultados que obtienes no coinciden con los mostrados anteriormente, entonces deberás modificar el código de los ficheros mapper.py y reducer.py. En caso contrario, procede a continuación a verificar el funcionamiento de los scripts contra el dataset completo mediante la ejecución del trabajo MapReduce. Fíjate que esta es una forma cómoda de verificar el funcionamiento de los scripts sin tener que esperar a la ejecución del trabajo MapReduce, facilitando el proceso de desarrollo y prueba inicial de los scripts.

A continuación, se muestra el resultado que debes obtener a partir del dataset `purchases.txt` para que puedas comparar tus resultados con la solución correcta. Tendrás que usar el comando hs para ejecutar el trabajo en el clúster contra el dataset completo.

```
Baby   57491808.44
Books  57450757.91
CDs    57410753.04
Cameras      57299046.64
Children's Clothing       57624820.94
Computers    57315406.32
Consumer Electronics      57452374.13
Crafts       57418154.5
DVDs   57649212.14
Garden       57539833.11
Health and Beauty   57481589.56
Men's Clothing      57621279.04
Music  57495489.7
Pet Supplies 57197250.24
Sporting Goods      57599085.89
Toys   57463477.11
Video Games  57513165.58
Women's Clothing    57434448.97
```

Valor de la Venta de Mayor Importe para Cada Tienda

Para esta actividad interesa calcular cual es el valor de la venta de mayor importe para cada una de las tiendas.

Actividad 5 (Evaluable). Dedica unos minutos a tratar de crear dos nuevos scripts `mapper.py` y `reducer.py`, para obtener el valor de la venta de mayor importe para cada tienda. Trata de aprovechar al máximo el código de algún ejemplo anterior por lo que te recomiendo que partas de scripts anteriores para introducir las modificaciones necesarias para resolver el problema.

NOTA: Como consejo, ten en cuenta que, en Python, si deseas usar el operador < ó > para comparar valores (por ejemplo para comparar importes), deberás transformar el tipo del dato usando la operación float(valor).

Al igual que en el ejemplo anterior, conviene que realices las pruebas iniciales de los scripts (mapper.py y reducer.py) con un subconjunto del dataset original, sin necesidad de ejecutar un trabajo MapReduce completo sobre el cluster Hadoop.

Por ejemplo, el siguiente comando permite verificar la funcionalidad de los scripts usando las primeras 1000 líneas del fichero de entrada y mostrando únicamente las 10 primeras líneas de resultados.

```
:~$ cat /opt/datasets/purchases.txt | head -n 1000 | ./mapper.py | sort -k
1,1 | ./reducer.py | head -n 10
Albuquerque       484.24
Anaheim     456.53
Anchorage   450.6
Arlington   400.08
Atlanta     484.31
Aurora      495.97
Austin      483.1
Bakersfield       498.93
Baltimore   467.63
Baton Rouge       411.54
```

Verifica si tus resultados coinciden con los aquí mostrados. En ese caso, procede a la ejecución del programa MapReduce contra el dataset completo (usando el script *hs*). A continuación se muestra parte del resultado que debes obtener a partir del dataset `purchases.txt` para que puedas comparar tus resultados con la solución correcta.

```
Albuquerque     499.98
Anaheim         499.98
Anchorage       499.99
Arlington       499.95
Atlanta         499.96
Aurora          499.97
…
Virginia Beach  499.98
Washington      499.98
Wichita         499.97
Winston-Salem   499.98
```

Hadoop Streaming – Logs de un Servidor Web

En los ejemplos anteriores se ha trabajado con diferentes conjuntos de datos (*datasets*) para experimentar con la programación en MapReduce sobre Apache Hadoop. En esta sección se trabaja con otro conjunto de datos, obtenido de [10], para resolver varios problemas interesantes relacionados con el análisis de los archivos de registro (*logs*) de un servidor web (Apache [21], en este caso). En las páginas de comercio electrónico, como es el caso de Amazon [18], es habitual que el propio sitio nos sugiera artículos para comprar en función de los artículos que hemos comprado previamente, los artículos que hemos visto en las últimas sesiones y en función de nuestro perfil de usuario. Toda esta

información se puede obtener (y de hecho se obtiene) mediante el análisis de nuestras interacciones (páginas visitadas) con el sitio de comercio electrónico. Determinar patrones de conducta y predecir, para anticiparse, a los futuros intereses del consumidor requiere el uso de técnicas de Big Data para procesar la ingente información que se obtiene de los logs de los servidores web y convertir dichos datos en información valiosa para la empresa.

En esta sección se trabaja con un conjunto de datos que incluye el log de acceso al servidor web Apache[1] de una empresa. En primer lugar se describe el formato en el que los datos se estructuran en dicho fichero (*Common Log Format*). Posteriormente, se plantean actividades sobre dicho conjunto de datos. Las actividades son variantes de las que se pueden encontrar en [10].

El Formato Common Log Format

Common Log Format [20] es un formato estandarizado de ficheros de texto usado por los servidores web para generar archivos de registro. La estandarización facilita el posterior procesado de la información por medio de herramientas de análisis. A continuación se muestra un ejemplo de línea en un fichero estructurado con dicho formato:

```
     0         1 2              3         4     5           6              7       8     9
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/lightbox.js HTTP/1.1" 200 25960
```

Existen 7 campos bien diferenciados que incluyen la siguiente información:

- La dirección IP del cliente (en el ejemplo se corresponde con 10.223.157.186).
- La identidad del cliente, o "-" si es desconocida (en el ejemplo se corresponde con el primer guión).
- El identificador del usuario que realiza la petición http, o "-" si es desconocido (en el ejemplo se corresponde con el segundo guión).
- El instante de tiempo (incluyendo la zona horaria) cuando el servidor terminó de procesar la petición del usuario (en el ejemplo se corresponde con [15/Jul/2009:15:50:35 -0700]).
- La petición del cliente, incluyendo el método, la ruta al recurso y el protocolo empleado (en el ejemplo se corresponde con "GET /assets/js/lightbox.js HTTP/1.1").
- El código de estado de la petición (en el ejemplo se corresponde con el 200)
- El tamaño en bytes del objeto entregado al cliente (en el ejemplo se corresponde con el 25960).

Ten en cuenta, por otra parte, que la línea anterior **está compuesta por 10 campos separados por un espacio** de manera que, por ejemplo, la ruta del recurso está situada en el campo número 6 (si se comienza a contar desde 0), tal y como se muestra en los números de la figura anterior.

Número Total de Accesos a un Recurso

Se desea calcular el número total de accesos que un determinado recurso (o página web) ha recibido a lo largo de todo el periodo de tiempo que abarque el fichero de registros de acceso del servidor web

[1] Típicamente el fichero el fichero /var/log/httpd/access_log o /var/log/apache2/access_log dependiendo de si se trata de una distribución de GNU/Linux basada en RedHat (CentOS, RHEL, Fedora) o basada en Debian (Ubuntu).

Apache donde estaba alojada. Esto permite conocer, por ejemplo, aquellos recursos que son más populares para destacarlos en la portada de una web o incluso para aumentar el coste de la publicidad que en ellas se muestra. Dispones de un conjunto de datos de acceso a un servidor web en `/opt/datasets/access_log` en el entorno de prácticas que ya ha sido copiado a HDFS y está disponible en `/datasets/accesslog/access_log`. El dataset ocupa 482 MBytes y consta de 4.477.843 líneas de texto en formato Common Log Format.

Puedes obtener las primeras 20 líneas del fichero con el siguiente comando:

```
:~$ hadoop fs -text /datasets/accesslog/access_log | head -n 10
10.223.157.186 - - [15/Jul/2009:14:58:59 -0700] "GET / HTTP/1.1" 403 202
10.223.157.186 - - [15/Jul/2009:14:58:59 -0700] "GET /favicon.ico HTTP/1.1" 404 209
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET / HTTP/1.1" 200 9157
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/lowpro.js HTTP/1.1" 200 10469
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/css/reset.css HTTP/1.1" 200 1014
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/css/960.css HTTP/1.1" 200 6206
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/css/the-associates.css HTTP/1.1" 200 15779
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/the-associates.js HTTP/1.1" 200 4492
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/lightbox.js HTTP/1.1" 200 25960
10.223.157.186 - - [15/Jul/2009:15:50:36 -0700] "GET /assets/img/search-button.gif HTTP/1.1" 200 168
```

Actividad 6 (Evaluable). Se te pide que crees un programa MapReduce mediante dos scripts (`mapper.py` y `reducer.py`) que utilicen las capacidades de Hadoop Streaming para obtener el número total de accesos a cada uno de los recursos de la web de la que se disponen los logs. Un ejemplo de ruta existente en dicho conjunto de datos es `/assets/img/home-logo.png`. Ten en cuenta que, si utilizamos el espacio como separador entre campos, el nombre del recurso está en el campo 6 (comenzando a contar desde el 0).

De cara a procesar el fichero de logs, no es necesario que realices comprobaciones de unicidad de un recurso web de manera que, por ejemplo, las siguientes rutas se consideran que pertenecen a recursos diferentes: `/assets/img/home-logo.png`, `//assets/img/home-logo.png` ó `/assets/img/home-logo.png?id=606`

Para poder agilizar el desarrollo de los scripts, recuerda que al estar implementados con Hadoop Streaming podemos hacer las pruebas sobre un dataset pequeño desde la línea de comandos. Esto evitará sobrecargar al cluster lanzando trabajos MapReduce para cada prueba que queramos hacer si introducimos una pequeña modificación en nuestros scripts. Una vez verificados los scripts, ya desplegaremos el trabajo MapReduce sobre el dataset completo. Por ejemplo, con el siguiente comando podemos verificar el resultado de los scripts que hemos creado, considerando únicamente las primeras 1000 líneas del dataset y mostrando únicamente las primeras 15 líneas de resultados:

```
:~$ cat /opt/datasets/access_log | head -n 1000 | ./mapper.py | sort -k 1,1
| ./reducer.py  | head -n 15
/     37
/about-us/  6
/about-us   6
/about-us/campaigns/    1
/about-us/campaigns     1
/about-us/people/       2
/about-us/people  2
/assets/css/960.css     54
/assets/css/reset.css   54
/assets/css/the-associates.css      54
/assets/flv/dummy/home-media-block/district-13.flv    2
/assets/img/about-us-logo.png       5
```

```
/assets/img/closelabel.gif    53
/assets/img/contact-us-logo.png     5
/assets/img/download-icon.gif       6
```

Verifica si tus resultados coinciden con los mostrados anteriormente. De lo contrario, puedes modificar tus scripts y repetir las pruebas de forma sencilla. Una vez obtengas los mismos resultados ya puede ejecutar el programa MapReduce sobre el conjunto de datos completo. Luego puedes filtrar fácilmente los resultados de salida (con un grep, por ejemplo) para obtener el número total de accesos de un recurso concreto. A continuación se te proporcionan algunos resultados para que puedas comparar con el resultado que obtienes mediante tu programa MapReduce contra el dataset completo. Estos resultados no tienen porqué aparecer consecutivos en el listado de salida que obtengas.

Recurso Web	Número total de accesos registrados
/assets/img/home-logo.png	98744
/assets/css/reset.css	2382
/assets/js/lowpro.js	293

Si te coinciden estos tres datos entonces puedes dar por buena la solución implementada. Fíjate que si únicamente se quisiera conocer el número total de accesos a un determinado recurso web, podría ser implementado de forma más eficiente filtrando por dicho recurso en la fase de Map (en el script `mapper.py`).

Número Total de Accesos desde una Misma Dirección IP

Otro análisis interesante es determinar el número total de accesos a cualquier recurso web desde una misma dirección IP, para conocer el grado de vinculación o retorno de dicho usuario al sitio web.

Actividad 7 (Evaluable). Crea un programa MapReduce mediante dos scripts (`mapper.py` y `reducer.py`) que utilicen las capacidades de Hadoop Streaming para obtener el número total de accesos desde la IP 10.223.157.186 a cualquiera de los recursos de la web de la que se disponen los logs.

Fíjate que sería sencillo obtener dicha información mediante herramientas de línea de comandos de Linux, por ejemplo con el siguiente comando:

```
:~$ cat /opt/datasets/access_log | grep 10.223.157.186 | wc -l
115
```

Sin embargo, de esta manera la ejecución se realiza sobre un único nodo mientras que, al utilizar MapReduce, la ejecución se realiza sobre un cluster de nodos, de manera que en función del número de nodos del cluster y el tamaño del conjunto de datos, optar por una solución basada en Hadoop puede resultar más eficiente (aunque no es el caso de este ejemplo).

Recurso Web Más Popular

A continuación, se desea conocer cual es el recurso web más popular de todos los que aparecen en el conjunto de datos del log de acceso al servidor web. Recuerda que no es necesario que realices ninguna comprobación de unicidad del recurso web. Además, por simplicidad no es necesario que verifiques el estado de las peticiones HTTP. Tan solo hay que contar el número de veces que aparece dicho recurso web en una línea del fichero de registro.

Actividad 8 (Evaluable). Crea un programa MapReduce mediante dos scripts (`mapper.py` y `reducer.py`) que utilicen las capacidades de Hadoop Streaming para obtener el recurso web más popular, es decir, aquel que recibió el mayor número de accesos. El script reducer.py deberá mostrar únicamente la ruta al recurso más popular y el número total de accesos.

Puedes probar los scripts que has creado de forma sencilla con un subconjunto del dataset completo. Por ejemplo, aquí se te muestra el resultado que deberías obtener usando únicamente las 1000 primeras líneas del dataset:

```
:~$ cat /opt/datasets/access_log | head -n 1000 | ./mapper.py | sort -k 1,1
| ./reducer.py
/assets/css/960.css     54
```

A continuación, se muestra la solución para el dataset completo para que puedas compararla con la que obtienes. El recurso web más popular es `/assets/css/combined.css` con 117348 hits [22].

Conclusiones

En esta práctica has aprendido los conceptos básicos de la programación mediante el modelo MapReduce y su implementación práctica usando Apache Hadoop, todo ello realizando ejecuciones reales sobre un cluster Hadoop desplegado sobre Amazon EC2. Habrás podido comprobar como el modelo MapReduce no sirve para cualquier tipo de aplicación pero supone una potente herramienta para la creación de programas que requieren el procesado de grandes volúmenes de datos de forma distribuida.

Referencias

[1] MapReduce. http://research.google.com/archive/mapreduce.html
[2] Apache Hadoop. http://hadoop.apache.org
[3] Hadoop Streaming. http://hadoop.apache.org/docs/stable1/streaming.html
[4] T. White, Hadoop: The Definitive Guide. O'Reilly Media, 2012. Disponible en: http://shop.oreilly.com/product/0636920021773.do
[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Final)," 2011, disponible en http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
[6] Project Gutenberg. http://www.gutenberg.org
[7] Hadoop Log Location and Retention. http://blog.cloudera.com/blog/2010/11/hadoop-log-location-and-retention/
[8] Hadoop Streaming. http://hadoop.apache.org/docs/r1.1.2/streaming.html
[9] Why map and reduce at the same time? http://stackoverflow.com/questions/18793830/why-map-and-reduce-run-at-the-same-time
[10] Introduction to Hadoop and MapReduce. https://www.udacity.com/course/ud617
[11] Upgrading to the New MapReduce API. http://www.slideshare.net/sh1mmer/upgrading-to-the-new-map-reduce-api
[12] Cloudera. http://www.cloudera.com/
[13] Cloudera QuickStart VM. http://www.cloudera.com/content/cloudera-content/cloudera-docs/DemoVMs/Cloudera-QuickStart-VM/cloudera_quickstart_vm.html
[14] CDH. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html
[15] Hortonworks. http://hortonworks.com
[16] Hortonworks Sandbox. http://hortonworks.com/products/hortonworks-sandbox/
[17] MapR. http://www.mapr.com
[18] MapR Academy. http://www.mapr.com/academy/
[19] Amazon. http://www.amazon.es
[20] Common Log Format. http://en.wikipedia.org/wiki/Common_Log_Format
[21] Apache. http://httpd.apache.org

[22] Hit (Internet). http://en.wikipedia.org/wiki/Hit_(internet)
[23] WordCount.java. https://raw.githubusercontent.com/apache/hadoop/master/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/WordCount.java

Anexo A. Despliegue de Entorno Hadoop

Este anexo describe algunas de las formas de despliegue de un entorno de trabajo con Apache Hadoop:

- Despliegue de un Cluster Hadoop en Amazon EC2.
- Descarga de entornos virtualizados pre-configurados.

Despliegue de un Cluster Hadoop en Amazon EC2

Para la puesta en marcha del cluster Hadoop, se pueden utilizar las extensiones de EC2 para Hadoop presentes en la versión 1.2.1. Estas extensiones permiten crear un cluster Hadoop en Amazon EC2, incluyendo los grupos de seguridad y las claves. Internamente utiliza una AMI pre-configurada con una versión relativamente antigua de Hadoop (0.19), pero que sirve para demostrar las capacidades de este framework.

Las características de dicha AMI son las siguientes:

- Región: US – Virginia (us-east-1)
- Nombre imagen: ami-fa6a8e93
- Fedora 8, 32 bits
- Cuenta de usuario: root
- Instancias compatibles: m1.small o superiores

Alternativamente, Hadoop se puede instalar en cualquier máquina GNU/Linux (y OS X) a través de los paquetes existentes en las direcciones siguientes:

- http://www.apache.org/dyn/closer.cgi/hadoop/core/
- http://apache.rediris.es/hadoop/core/hadoop-1.2.1/hadoop-1.2.1-bin.tar.gz

En nuestro caso, ya se encuentran instaladas en /opt/hadoop-1.2.1.

En el directorio /opt/cloud/hadoop-1.2.1/src/contrib/ec2/bin se encuentran las extensiones que permiten definir, levantar y apagar un cluster Hadoop.

Para configurar el cluster Hadoop en AWS, es necesario modificar el fichero de configuración (`hadoop-ec2-env.sh`), con lo que utilizamos una copia local de dicho directorio. Para su funcionamiento hay que asegurarse de que las entradas siguientes tienen el valor correcto:

```
# Your Amazon Account Number.
AWS_ACCOUNT_ID=XXX
# Your Amazon AWS access key.
AWS_ACCESS_KEY_ID=XXX
# Your Amazon AWS secret access key.
AWS_SECRET_ACCESS_KEY=XXX
# The EC2 key name used to launch instances.
# The default is the value used in the Amazon Getting Started guide.
KEY_NAME=alucloud00-keypair
```

La interacción de Hadoop con Amazon EC2 se basa en el script *hadoop-ec2*. Los propios scripts usan una AMI adecuada. Estos scripts además crean, en la primera ejecución los grupos de seguridad necesarios para la puesta en marcha.

Para la puesta en marcha del cluster Hadoop utilizamos el comando `hadoop-ec2 launch-cluster`, indicando el número de nodos del cluster.

```
$ ./hadoop-ec2 launch-cluster AWS-hadoop-cluster 4
Testing for existing master in group: AWS-hadoop-cluster
Starting master with AMI ami-fa6a8e93
Waiting for instance i-3eb61743 to start
.....Started as ip-10-144-19-37.ec2.internal
ssh: connect to host ec2-54-242-143-246.compute-1.amazonaws.com port 22: Connection refused
Warning: Permanently added 'ec2-54-242-143-246.compute-1.amazonaws.com,54.242.143.246' (RSA) to the list of known hosts.
Copying private key to master
alucloud00-priv.pem                                   100% 1672      1.6KB/s 00:00
Master is ec2-54-242-143-246.compute-1.amazonaws.com, ip is 54.242.143.246, zone is us-east-1d.
Adding AWS-hadoop-cluster node(s) to cluster group AWS-hadoop-cluster with AMI ami-fa6a8e93
i-cb91b9b1
i-c991b9b3
i-cf91b9b5
i-cd91b9b7
```

A partir de este momento será posible conectarse como usuario root al cluster Hadoop, que contendrá una despliegue de Apache Hadoop ya pre-configurado y funcional junto con el sistema de archivos HDFS configurado y listo para el almacenamiento de datos para ser procesados en el marco de aplicaciones MapReduce.

Una vez finalizadas las actividades, el apagado del clúster Hadoop se realiza mediante el comando `hadoop-ec2 terminate-cluster` desde la máquina donde se inició el mismo.

```
$ bin/hadoop-ec2 terminate-cluster AWS-hadoop-cluster

Running Hadoop instances:

INSTANCE i-8de0dde4 ami-ee53b687 ec2-174-129-150-48.compute-1.amazonaws.com domU-12-31-39-00-56-56.compute-1.internal running test-keypair 0 m1.small 2011-07-14T11:20:01+0000 us-east-1d aki-a71cf9ce ari-a51cf9cc monitoring-disabled

…

INSTANCE i-e3e3de8a ami-ee53b687 ec2-75-101-190-104.compute-1.amazonaws.com domU-12-31-39-00-59-47.compute-1.internal running test-keypair 0 m1.small 2011-07-14T11:22:34+0000 us-east-1d aki-a71cf9ce ari-a51cf9cc monitoring-disabled

INSTANCE i-e5e3de8c ami-ee53b687 ec2-174-129-141-147.compute-1.amazonaws.com domU-12-31-39-00-A9-25.compute-1.internal running test-keypair 1 m1.small 2011-07-14T11:22:34+0000 us-east-1d aki-a71cf9ce ari-a51cf9cc monitoring-disabled

Terminate all instances? [yes or no]: yes

INSTANCE i-8de0dde4 running shutting-down

...

INSTANCE i-e3e3de8a running shutting-down
```

```
INSTANCE        i-e5e3de8c      running shutting-down
```

Descarga de entornos virtualizados pre-configurados y tutoriales.

Algunas distribuciones de Hadoop ofrecen entornos virtualizados ya pre-configurados con una instalación de Hadoop para poder iniciarse en el manejo de esta herramienta. Por ejemplo, Cloudera [12], la empresa detrás de CDH (Cloudera Distribution for Hadoop) [14] ofrece una máquina virtual con una instalación de CDH, disponible en [13]. Esta viene pre-configurada con Hadoop y otros servicios para iniciarse en el campo del Big Data.

Otro ejemplo es Hortonworks [15], una spin-off de Yahoo, que ofrece una distribución de Hadoop y el Hortonworks Sandbox [16] que incluye un entorno Hadoop acompañado de numerosos tutoriales para poder aprender Hadoop de forma cómoda.

Existen otras distribuciones de Hadoop, como es el caso de MapR [17], que ofrece el servicio MapR Academy [18] con numerosos vídeos relacionados con esta tecnología.

Anexo B. El Fichero WordCount.java

```
/**
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 */


package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {
```

```
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```