**[11.6.2. MapReduce – Datos Entrada - Completo](https://campusformacionnebrija.imf.com/general/prt/prg/leccgen/newud.php?codcurso=C7274C13&unidad=2&trozo=24&param=contenido)**

En este epígrafe se presenta un ejemplo completo de MapReduce. Para ello se parte del ejemplo wordcount proporcionado por la documentación de Hadoop y, dada una serie de datos de entrada, se verá cómo el sistema realiza los cálculos del número de palabras que aparecen en los mismos y el número de veces que se repite cada una de ellas.

El punto de partida del proceso de MapReduce es la **fase de mapeo**. Durante esta fase, se capturan los datos de entrada y se transforman en un par clave valor que se empleará en el proceso intermedio. Los registros transformados no tienen por qué ser el mismo número de registros que los de entrada. Un par de entrada puede devolver cero elementos mapeados o múltiples pares, e incluso pueden ser de distinto tipo.

La implementación del mapeo se envía mediante la clase indicada en la tarea con la función *setMapperClass*. El *framework* de Hadoop llamará a la función *map* (WritableComparable, Writable, Context) para cada uno de los elementos clave valor que recibe en la entrada.

**Fase de Mezcla:** todos los pares clave - valor intermedios resultantes de la fase de mapeo se emplearán como entrada a la función de mezcla o agrupación. Esta función se controla mediante el método *setCombinerClass*:

*job.setCombinerClass(IntSumReducer.class);*

En este caso, el proceso de mezcla o combinación emplea la misma clase que el proceso de reducción.

Por último, se aplica la **fase de reducción** para, partiendo de los elementos de la mezcla, obtener el resultado. En este caso la función se especifica mediante el método *setReducerClass*.

**Processo**

En primer lugar, se parte del siguiente fichero Java que contiene las funciones *map* y *reduce*.

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;

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();

// Function 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);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

// Function 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);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class); // implementación del mapeo

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class); // implementación de reducción

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

Debemos generar un jar encargado de ejecutar las funciones anteriores. Para ello se siguen los siguientes pasos:

**1.** **Crear un fichero wordcount.java con el código anterior**

*$ sudo mkdir /home/bigdata/mapreduce*

*$ cd /home/bigdata/mapreduce*

*$ sudo nano WordCount.java*

**2. Compilar el fichero**

*$ sudo javac -classpath $HADOOP\_HOME/share/hadoop/common/hadoop-common-2.8.0.jar:$HADOOP\_HOME/share/hadoop/common/lib/hadoop-annotations-2.8.0.jar:$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.8.0.jar /home/bigdata/mapreduce/WordCount.java*

**3.** **Crear un jar**

*$ sudo jar cf wc.jar WordCount\*.class*

<https://docs.oracle.com/javase/tutorial/deployment/jar/build.html>

**4.** **Crear los ficheros de entrada de datos**

*$ sudo mkdir wc-in*

*$ sudo chmod -R 777 wc-in*

*$ sudo echo "Hello World. Bye World." > wc-in/a.txt*

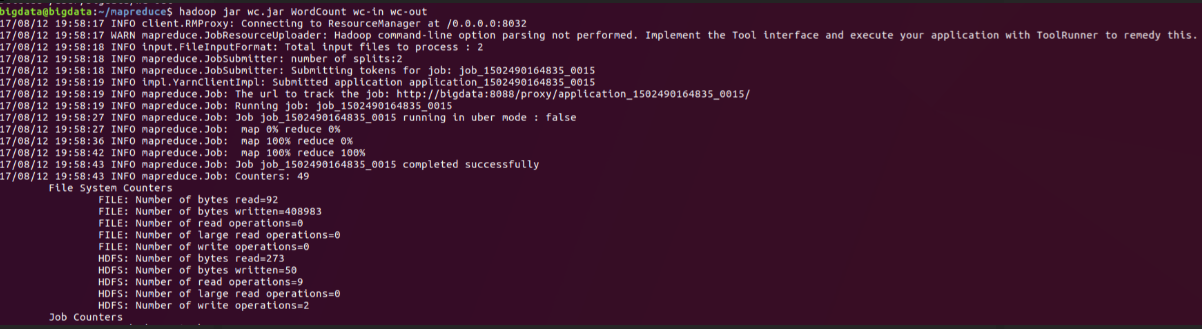
*$ sudo echo "Hello Hadoop Goodbye Hadoop" > wc-in/b.txt*

*$ hdfs dfs -mkdir /user/bigdata/wc-in*

*$ hdfs dfs -put wc-in/\* /user/bigdata/wc-in*

**5. Ejecución del jar:**

*$ hadoop jar wc.jar WordCount wc-in wc-out*



*$ hdfs dfs -cat /user/bigdata/wc-out/part-r-00000*

