Prática sesión 5 - mapreduce ejemplo

CURSO INTRODUCCIÓN A BIG DATA

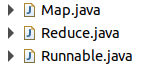
Resumen

En este documento se trata de explicar mediante un ejemplo práctico el funcionamiento de MapReduce y cuáles son los métodos que intervienen en este tipo de procesamiento, utilizando para ello una serie de plantillas con las que ir elaborando el esquema funcional de MapReduce.

## Estudio de ejemplo HADOOP\_MAPREDUCE

Dentro del workspace de eclipse tenemos un ejemplo de proyecto java para MapReduce que implementa un algoritmo básico para contar el número de ocurrencias por palabra que aparece dentro de un documento.

Si miramos el proyecto en Eclipse, tenemos 3 clases:



A continuación, vamos a describir, paso a paso cómo interviene cada una de las clases en el proceso Mapreduce.

### 1.1. Runnable.class

En este ejemplo, la clase Runnable va a ser la encargada de gestionar el trabajo a través de implementar la interfaz Tool, que es el estándar para cualquier aplicación MapReduce. Para el manejo de los argumentos a través de la línea de comando de Hadoop delega en ToolRunner.

Dentro del método run, podemos reseñar las siguientes instrucciones:

#### 1.1.1. Obtenemos la configuración del sistema

Modificamos el Configuration de Tool añadiendo las rutas donde están los ficheros de configuración de nuestra instalación de hadoop.

|  |
| --- |
| conf.addResource(**new** Path("file:///opt/hadoop/etc/hadoop/core-site.xml"));  conf.addResource(**new** Path("file:///opt/hadoop/etc/hadoop/hdfs-site.xml"));  conf.addResource(**new** Path("file:///opt/hadoop/etc/hadoop/yarn-site.xml")); |

#### 1.1.1. Declaramos a través de Jobconf la configuración de nuestra tarea MapReduce

Utilizamos JobConf para describir un trabajo mapreduce indicando el Mapper, Reducer, Combinear, InputFormat y OutputFormat, etc.

|  |
| --- |
| JobConf job = **new** JobConf(conf, Runnable.**class**); |

#### 1.1.2. Indicamos el directorio de entrada y salida del HDFS

|  |
| --- |
| Path inDir = **new** Path("inWordCount");  Path outDir = **new** Path("outWordCount"); |

#### 1.1.3. Si el directorio de salida existe lo borramos

|  |
| --- |
| outDir.getFileSystem(conf).delete(outDir, **true**); |

#### 1.1.4. Establecemos el nombre del job

|  |
| --- |
| job.setJobName("WordCount"); |

#### 1.1.5. Indicamos al job dónde tiene que tomar y dejar los ficheros.

FileInputFormat: es la clase base para todos los InputFormat basados en archivos. Cuando iniciamos un trabajo Hadoop, F.I.F. recibe una ruta que contiene archivos para leer. FIF leerá todos los archivos y los dividirá en uno o más InputSplits

|  |
| --- |
| FileInputFormat.*setInputPaths*(job,inDir);  FileOutputFormat.*setOutputPath*(job,outDir); |

#### 1.1.6. Establecemos la naturaleza de la clave/valor

|  |
| --- |
| job.setOutputKeyClass(Text.**class**);  job.setOutputValueClass(IntWritable.**class**); |

#### 1.1.7. Declaramos los trabajos para cada una de las clases

|  |
| --- |
| job.setMapperClass(Map.**class**);  job.setCombinerClass(Reduce.**class**);  job.setReducerClass(Reduce.**class**); |

#### 1.1.8. Especificamos el qué tipo de entrada va a tener el job, en este caso será un texto plano sin formato.

|  |
| --- |
| job.setInputFormat(TextInputFormat.**class**);  job.setOutputFormat(TextOutputFormat.**class**); |

### 1.2. Map.class

La siguiente clase que vamos a analizar es la que implementa Mapper.

Con Mapper asignamos pares clave/valor a un conjunto de clave/valor intermedios:

<KEY\_IN, VALUE\_IN, KEY\_OUT,VALUE\_OUT>.

MapReduce generará una tarea MAP para cada InputSplit generado por el InputFormat que a su vez, divide los archivos en línea. Por ello, en los parámetro del método map econtramos:

* **LongWritable key**, es la posición de cada línea en el archivos.
* **Text value** son las líneas del texto.
* **OutPutCollector** recoge los datos para las salidas intermedias.
* **Reporter** proporciona información del progreso o simplemente para indicar que su estado es correcto. En marcos de trabajo con tareas pesadas para procesar claves/valor es crucial ya que el framework puede asumir que la tarea ha agotado el tiempo de espera y la elimine.

|  |
| --- |
| **public** **class** Map **extends** MapReduceBase **implements** Mapper<LongWritable,Text,Text,IntWritable>{  **private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);  **private** Text word = **new** Text();  **public** **void** map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,  Reporter reporter) **throws** IOException{    //Resto  }  } |

#### Mapeamos la información y lo dejamos en las salidas intermedias.

Para ello tokenizamos cada línea y le asignamos el valor “1” a cada palabra.

|  |
| --- |
| String line = value.toString();  StringTokenizer tokenizer = **new** StringTokenizer(line);  **while** (tokenizer.hasMoreTokens()){  word.set(tokenizer.nextToken());  output.collect(word, ***one***);  } |

OutPutCollector recoge los pares clave/valor intermedios.

### 1.3. Reduce.class

En la fase Reduce vamos agregamos los valores por las diferentes claves que nos llegan desde el OutputColector.

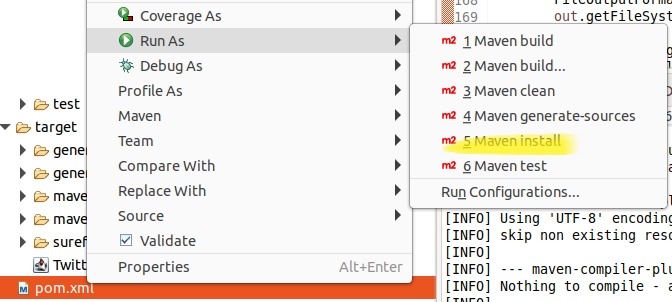
|  |
| --- |
| **public** **class** Reduce **extends** MapReduceBase **implements** Reducer<Text,IntWritable,Text,IntWritable> {    **public** **void** reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable> output,  Reporter reporter) **throws** IOException {    //Resto  }  } |

#### 1.3.1. Por último sumamos todos los valores que vienen en el iterator y lo añadimos a la clave

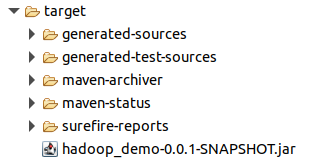
|  |
| --- |
| **int** sum=0;  **while** (values.hasNext()) {  sum+=values.next().get();  }  output.collect(key,**new** IntWritable(sum)); |

### 1.4. Ejecución

Nos situamos sobre el pom.xml y seleccionamos Run As –> Maven Install para generar el .jar



Esto nos habrá quedado un .jar en el directorio Target.



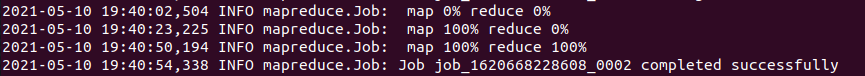
Por último, nos descargamos un fichero con el que poder probar el proceso mapreduce, para ello vamos a utilizar El Quijote en texto plano: <https://gist.github.com/jsdario/6d6c69398cb0c73111e49f1218960f79>

Una vez lo hemos descargado creamos el directorio en HDFS y lo copiamos mediante:

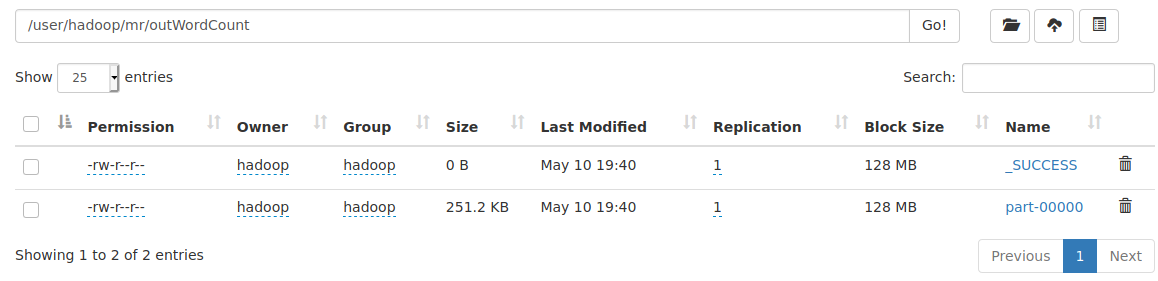
|  |
| --- |
| hdfs dfs –mkdir –p /user/hadoop/mr/inWordCount  hdfs dfs –put el\_quijote.txt //user/hadoop/mr/inWordCount |

Comprobamos con jps que Yarn esté levantado y lanzamos el proceso MapReduce:

# 



Si todo ha ido bien, podremos ir al directorio que hemos especificado en Runnable.class y comprobar la salida a través del UI de HDFS



Si hacemos una visualización de la cabecera del fichero part-00000 podemos ver que es correcto:

