



MBIT School
Madrid Business Intelligence Technology

MapReduce 2ª parte



Diego J. Bodas Sagi

- Maven
- Empaquetando la solución
- Eclipse y los servicios de Cloudera
- Los contadores en MapReduce
- Pruebas unitarias y MRUnit



Maven

- Maven es un complemento al compilador Java Ant (uno de los más usados en Java) que proporciona una estructura consistente de proyectos (todos los proyectos Maven tienen por defecto los mismos directorios)
- También proporciona herramientas para gestionar la complejidad de los proyectos de software: gestión avanzada de dependencias, informes sobre pruebas...





Configuración del proyecto Maven

- Archivo pom.xml: en este archivo se describen las librerías necesarias, el orden de compilación ...
- Podemos encontrar un ejemplo de este archivo de configuración en: <https://gist.github.com/jnatkins/3517129>
- La ejecución de un archivo POM siempre genera un "**artefacto**"
- Maven trabaja modularizando los proyectos. De esta forma tendremos varios módulos que conforman un sólo proyecto
- Para denotar esta relación en Maven, se crea un proyecto **padre** de tipo POM (esto es que no genera un binario en sí) y los módulos se definen como otros archivos pom que heredan del primero
- Lo anterior permite centralizar en el pom padre las variables (como el nombre del proyecto o el número de versión), las dependencias, los repositorios, etc. que son comunes a los módulos, eliminando duplicidad de código





Crear un proyecto Maven

● Maven Archetypes

- Los **arquetipos** son artefactos especiales de Maven que sirven como plantillas para crear proyectos
- Maven cuenta con algunos predefinidos y terceros han hecho los suyos para crear proyectos con tecnologías específicas, como es el caso de la plantilla anterior
- Desde el directorio en el que queramos crear el proyecto:
 - *mvn archetype:generate*
 - *Pero ¡salen más de 300 opciones! Muy incómodo*





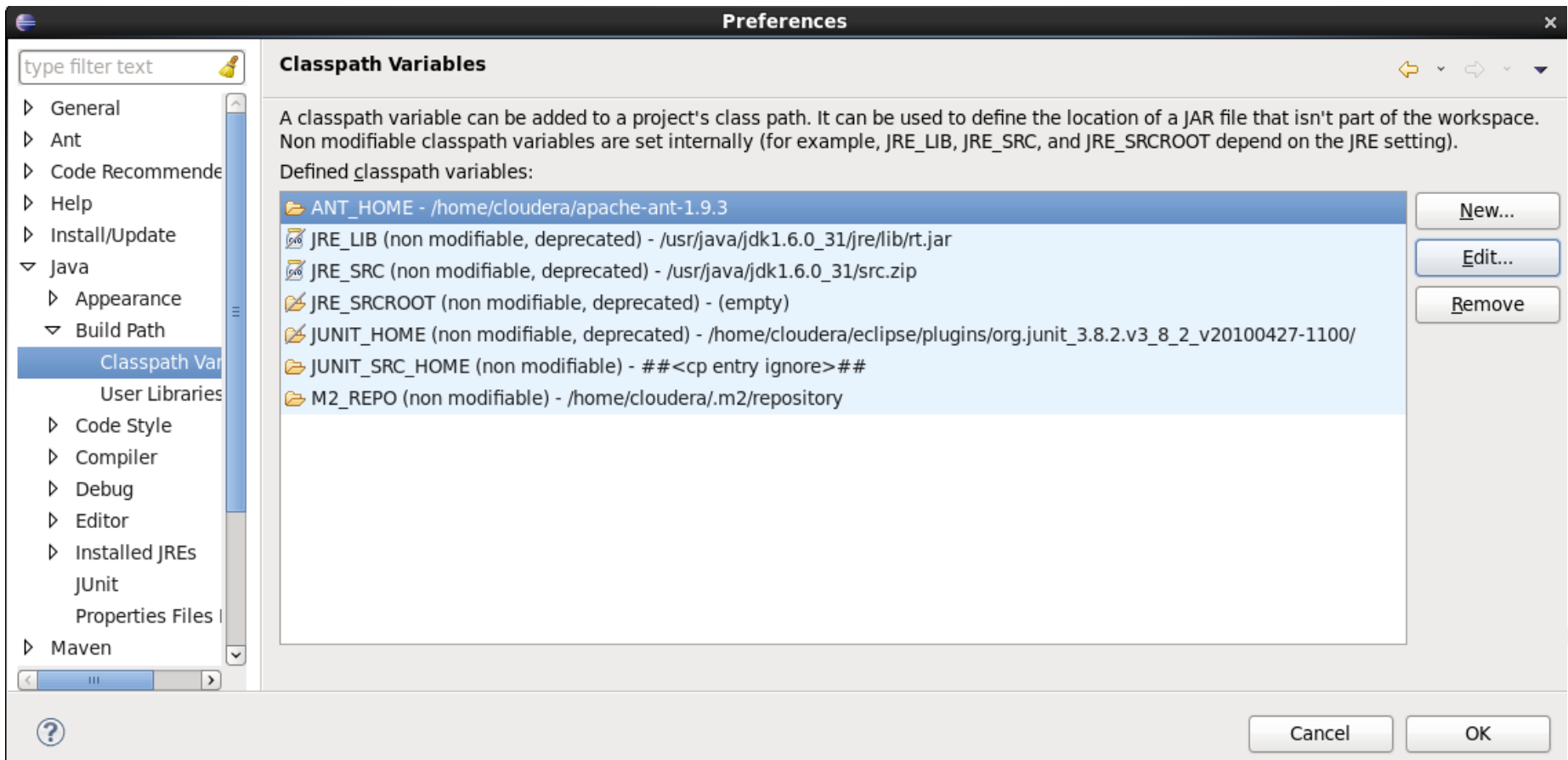
Configuración de Eclipse

- Actualizar software nada más empezar
- Añadir m2eclipse Plugin
 - <http://download.eclipse.org/technology/m2e/releases>
- Descargar ANT y configurar la variable ANT_HOME





Class Paths





Hadoop necesita tools.jar

type filter text

General

Ant

Code Recommender

Help

Install/Update

Java

Appearance

Build Path

Classpath Variables

User Libraries

Code Style

Compiler

Debug

Editor

Installed JREs

JUnit

Properties Files

Maven

Installed JREs

Add, remove or edit JRE definitions. By default, the build path of newly created Java projects uses the default JRE.

Installed JREs:

Name	Location
<input checked="" type="checkbox"/> jdk1.6.0_31	/usr/java/jdk1.6.0_31

JRE Definition

Specify attributes for a JRE

JRE home:

/usr/java/jdk1.6.0_31

Directory...

JRE name:

jdk1.6.0_31

Default VM arguments:

Variables...

JRE system libraries:

/usr/java/jdk1.6.0_31/jre/lib/resources.jar

/usr/java/jdk1.6.0_31/jre/lib/rt.jar

/usr/java/jdk1.6.0_31/jre/lib/jsse.jar

/usr/java/jdk1.6.0_31/jre/lib/jce.jar

/usr/java/jdk1.6.0_31/jre/lib/charsets.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/localedata.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/sunpkcs11.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/sunjce_provider.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/dnsns.jar

/usr/java/jdk1.6.0_31/lib/tools.jar

Add External JARs...

Javadoc Location...

Source Attachment...

Remove

Up

Down

Restore Default





Formato de código en Hadoop

- Window -> Preferences.
- Java->Code Style -> Formatter.
- Importar archivo de referencia
- Buenas prácticas
 - Window->Preferences->Java->Editor->Save Actions
 - Seleccionar “Perform the selected actions on save”, “Format source code”, “Format edited lines”.
 - NO seleccionar “Organize imports”
 - Riesgo de colisión con la API vieja, mejor a mano





Problemas con el pom.xml

- Debe examinarse cada problema de forma particular
- El manejo no es obvio
- Aunque compensa debido al potencial de reutilización
- ¡Mucho cuidado con las versiones reflejadas en el archivo pom.xml!



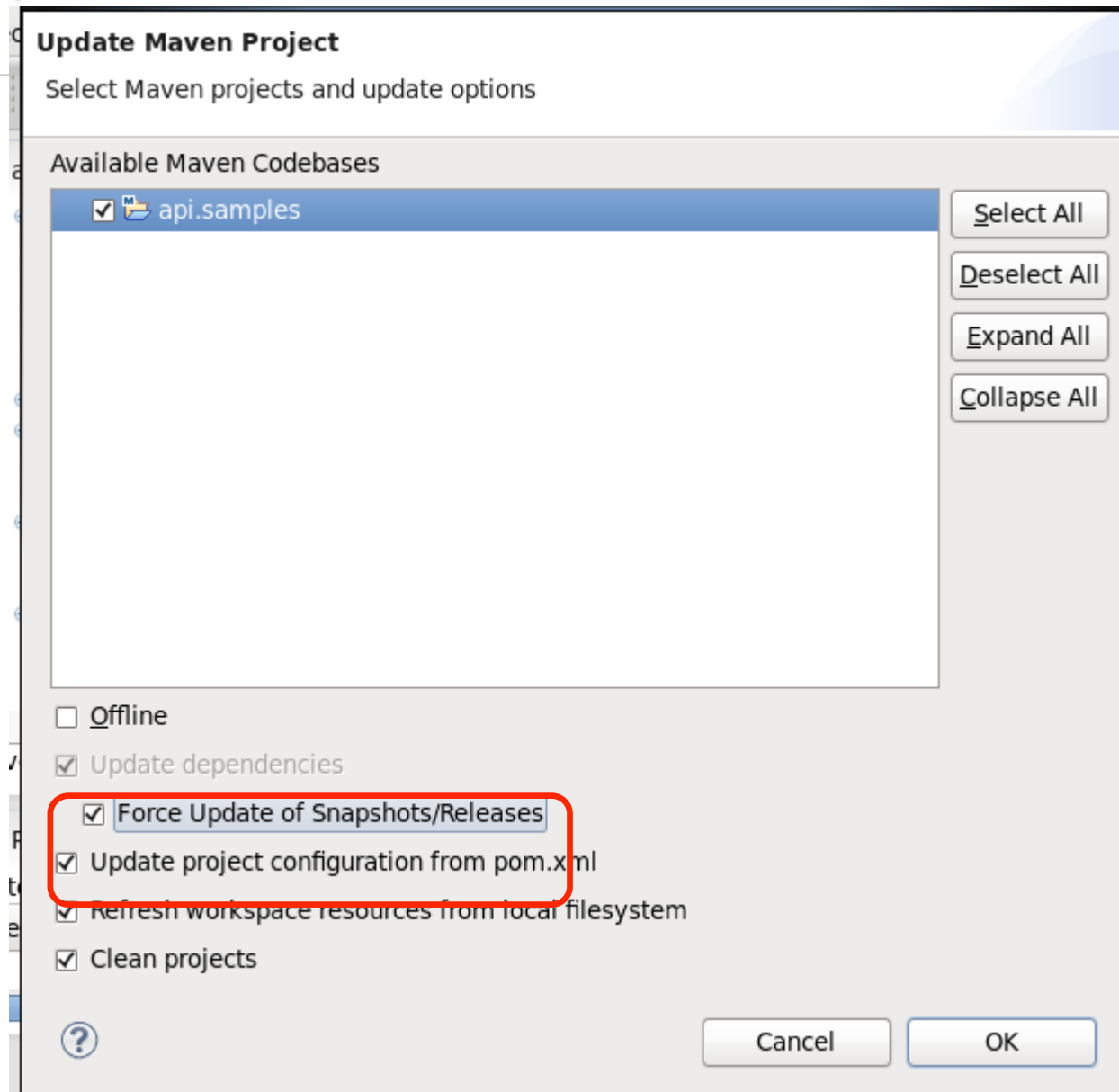


Versiones

- A veces, arreglar un error es tan sencillo como consultar (en Cloudera por ejemplo) y poner la versión adecuada
 - http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/latest/CDH-Version-and-Packaging-Information/cdhvd_topic_8.html
- Desde la consola
 - “\$ hadoop version”

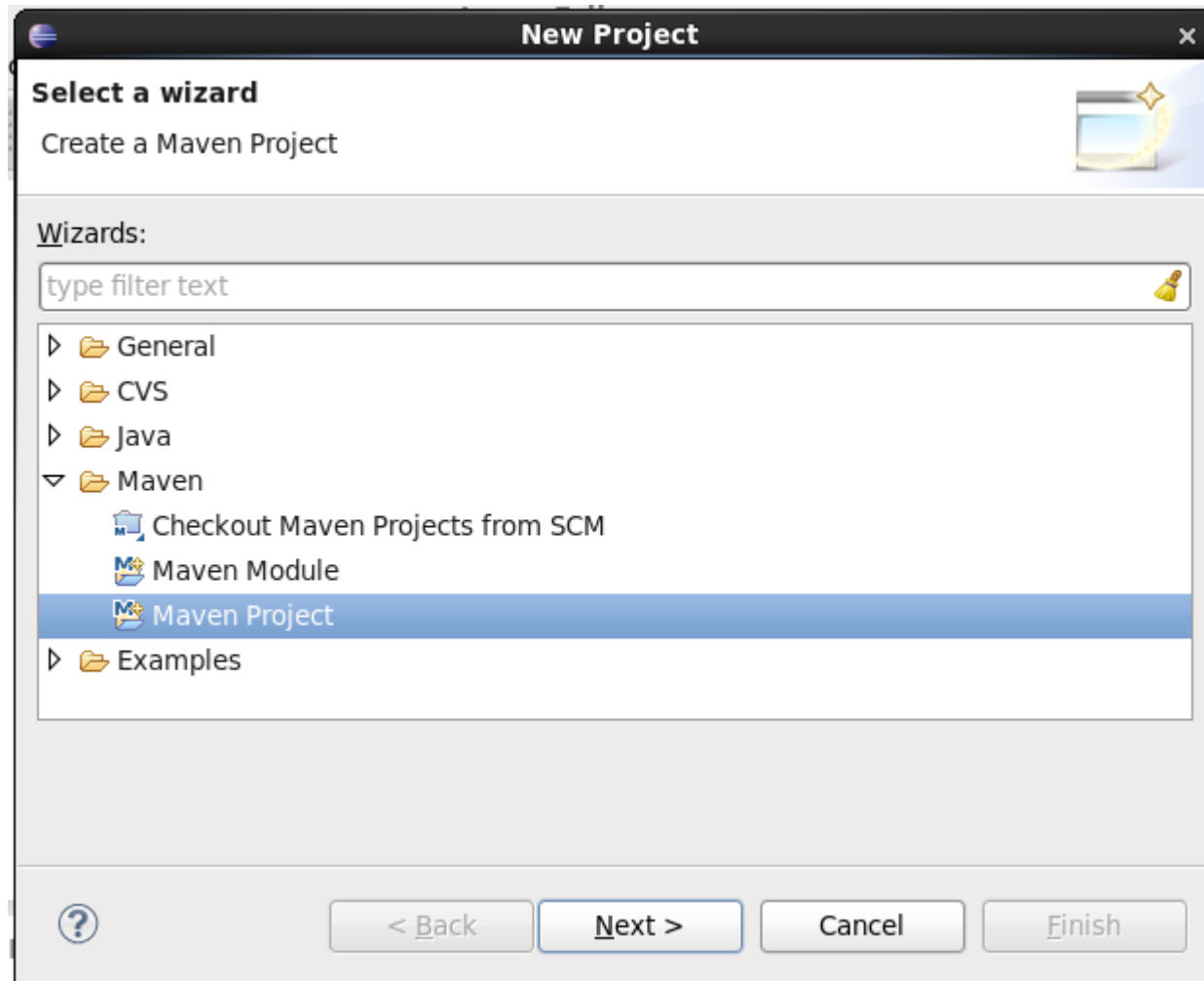


En el caso de problemas ...





Generando el proyecto Maven





Se necesita...

- Desde eclipse se debe modificar el archivo pom.xml añadiendo la información de versiones de java (1.6 por ejemplo) y repositorio
- Importante también las dependencias con hadoop core y hadoop client (MapReduce y APIs)
- Cada vez que se modifique el archivo de configuración pom.xml hay que actualizar el proyecto (opciones del menú del botón derecho del ratón)






Paso a paso

- El proyecto anterior que hemos creado lo vamos a dejar para cuestiones más complicadas
- Para realizar los primeros ejemplos con la API Hadoop para Java vamos a empezar paso a paso con proyectos muy sencillos



New Maven Project

Configure project



Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:


Parent Project

Group Id:

Artifact Id:

Version:

▶ **Advanced**





¡Cuidado!

- Está intentando usar la versión 1.5 de Java
- Debemos completar añadiendo el código respectivo en el archivo de configuración pom.xml
- Se añade desde eclipse
 - Seleccionando la pestaña de pom.xml porque es más fácil trabajar con el fichero fuente





Añadido

- Se añade la parte de maven-compiler del pom anterior (se especifica 1.6 como la versión correcta de java)
- Se añade la información del repositorio de cloudera (donde están todas las librerías)
- Se añaden las dependencias que tenemos
 - Para este ejemplo, sólo hadoop-common y hadoop-client
- Fijar las propiedades que sean necesarias



Actualizar proyecto



Java - api.samples/pom.xml - Eclipse

File Edit Source

Package Explorer

api.samples

- src/main
- src/main
- src/test
- src/test
- JRE System
- Maven
- src
- target
- pom.xml
- maven-hadoop
- training

Context Menu:

- New
- Go Into
- Open in New Window
- Open Type Hierarchy F4
- Show In Shift+Alt+W
- Copy Ctrl+C
- Copy Qualified Name
- Paste Ctrl+V
- Delete Delete
- Remove from Context Shift+Ctrl+Alt+Down
- Build Path
- Source Shift+Alt+S
- Refactor Shift+Alt+T
- Import...
- Export...
- Refresh F5
- Close Project
- Assign Working Sets...
- Run As
- Debug As
- Validate
- Team
- Compare With
- Restore from Local History...
- Maven

Right-click context menu options:

- Add Dependency
- Add Plugin
- New Maven Module Project
- Download JavaDoc
- Download Sources
- Update Project... Alt+F5
- Disable Workspace Resolution

XML Content:

```
<cdh-version></version>
<group>org.apache.hadoop</group>
<artifactId>hadoop-client</artifactId>
<cdh-version></version>
<id>cloudera-releases</id>
<url>https://repository.cloudera.com/artifactory/cloudera-releases</url>
<enabled>true</enabled>
<enabled>false</enabled>
<scope>test</scope>
<scope>runtime</scope>
<scope>test</scope>
```





Listos para trabajar

- Si observamos que la librería Java que aparece es la 1.6 y
- No hay errores o avisos
- ¡Ya estamos listos para trabajar y hemos descubierto otra forma sencilla de empezar!





Llevando el proyecto al cluster

- Empaquetado
 - Run As: generamos un “Maven build”
 - Name: api.samplesMavenPackage
 - Goals: clean package
 - El resultado aparecerá en el directorio “target”
- En la consola, nos ubicamos dentro del directorio del proyecto

```
cloudera@localhost:~/workspace/api.samples
File Edit View Search Terminal Help
[cloudera@localhost api.samples]$ ls
inputFiles  outputFiles  pom.xml  src  target
[cloudera@localhost api.samples]$
```





Llevando el proyecto al cluster (II)

- El fichero de entrada hay que llevarlo al cluster
 - Creamos en el cluster el directorio apiSamples
 - Dentro del directorio anterior, creamos inputFiles y outputFiles
 - Copiamos fileinput.txt a apiSamples/inputFiles





Copiando archivos fuentes

```
cloudera@localhost:~/workspace/api.samples/inputFiles
File Edit View Search Terminal Help

[cloudera@localhost api.samples]$ hdfs dfs -ls
Found 2 items
drwx----- - cloudera cloudera      0 2014-03-28 05:55 .staging
drwxr-xr-x - cloudera cloudera      0 2014-03-27 05:21 examples
[cloudera@localhost api.samples]$ hdfs dfs -mkdir apiSamples
[cloudera@localhost api.samples]$ hdfs dfs -ls
Found 3 items
drwx----- - cloudera cloudera      0 2014-03-28 05:55 .staging
drwxr-xr-x - cloudera cloudera      0 2014-04-07 03:25 apiSamples
drwxr-xr-x - cloudera cloudera      0 2014-03-27 05:21 examples
[cloudera@localhost api.samples]$ hdfs dfs -mkdir apiSamples/inputFiles
[cloudera@localhost api.samples]$ hdfs dfs -mkdir apiSamples/outputFiles
[cloudera@localhost api.samples]$ hdfs dfs -ls apiSamples
Found 2 items
drwxr-xr-x - cloudera cloudera      0 2014-04-07 03:26 apiSamples/inputFiles
drwxr-xr-x - cloudera cloudera      0 2014-04-07 03:26 apiSamples/outputFiles
[cloudera@localhost api.samples]$ cd inputFiles/
[cloudera@localhost inputFiles]$ hdfs dfs -put fileinput.txt apiSamples/inputFiles
[cloudera@localhost inputFiles]$ hdfs dfs -ls apiSamples/inputFiles
Found 1 items
-rw-r--r--  3 cloudera cloudera    3106 2014-04-07 03:29 apiSamples/inputFiles
/fileinput.txt
[cloudera@localhost inputFiles]$
```





Ejecutando el Job

- Usaremos el comando “hadoop jar”
 - Como parámetros, debemos pasar el nombre completo del archivo JAR
 - El nombre completo de la clase que implementa el Job: `com.utad.api.wordcount.DriverWordCount`
 - El parámetro de entrada: `apiSamples/inputFiles/fileinput.txt`
 - El parámetro de salida: `apiSamples/outputFiles/allWords.txt`





Resultado parcial

```
cloudera@localhost:~/workspace/  
File Edit View Search Terminal Help  
  
[cloudera@localhost api.samples]$ hadoop jar target/api.samples-0.0.1-SNAPSHOT.jar com.utad.:  
/outputFiles/allWords.txt  
14/04/07 03:40:37 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.  
14/04/07 03:40:37 INFO input.FileInputFormat: Total input paths to process : 1  
14/04/07 03:40:40 INFO mapred.JobClient: Running job: job_201404070129_0003  
14/04/07 03:40:41 INFO mapred.JobClient: map 0% reduce 0%  
14/04/07 03:40:58 INFO mapred.JobClient: map 100% reduce 0%  
14/04/07 03:41:07 INFO mapred.JobClient: map 100% reduce 100%  
14/04/07 03:41:10 INFO mapred.JobClient: Job complete: job_201404070129_0003  
14/04/07 03:41:10 INFO mapred.JobClient: Counters: 32  
14/04/07 03:41:10 INFO mapred.JobClient: File System Counters  
14/04/07 03:41:10 INFO mapred.JobClient: FILE: Number of bytes read=2691  
14/04/07 03:41:10 INFO mapred.JobClient: FILE: Number of bytes written=329849  
14/04/07 03:41:10 INFO mapred.JobClient: FILE: Number of read operations=0  
14/04/07 03:41:10 INFO mapred.JobClient: FILE: Number of large read operations=0  
14/04/07 03:41:10 INFO mapred.JobClient: FILE: Number of write operations=0  
14/04/07 03:41:10 INFO mapred.JobClient: HDFS: Number of bytes read=3254  
14/04/07 03:41:10 INFO mapred.JobClient: HDFS: Number of bytes written=2661  
14/04/07 03:41:10 INFO mapred.JobClient: HDFS: Number of read operations=2  
14/04/07 03:41:10 INFO mapred.JobClient: HDFS: Number of large read operations=0  
14/04/07 03:41:10 INFO mapred.JobClient: HDFS: Number of write operations=1  
14/04/07 03:41:10 INFO mapred.JobClient: Job Counters  
14/04/07 03:41:10 INFO mapred.JobClient: Launched map tasks=1  
14/04/07 03:41:10 INFO mapred.JobClient: Launched reduce tasks=1
```



Usando el Cloudera Manager

En la interfaz jobTracker

localhost Hadoop Map/Reduce Administration - Mozilla Firefox

File Edit View History Bookmarks Tools Help

localhost Hadoop Map/Reduce ...

localhost:50030/jobtracker.jsp

Most Visited Cloudera Cloudera Manager Hue HDFS NameNode Hadoop JobTracker HBase Master Solr How-to: Configure E...

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201404070129_0003	NORMAL	cloudera	api.samples-0.0.1-SNAPSHOT.jar	100.00% <div></div>	1	1	100.00% <div></div>	1	1





Generación de estadísticas

MBIT School
Technology

Hadoop job_201404070129_0003 on localhost - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Hadoop job_201404070129_00... +

localhost:50030/jobdetails.jsp?jobid=job_201404070129_0003&refresh=0

Most Visited Cludera Cludera Manager Hue HDFS NameNode Hadoop JobTracker HBase Master S

Job-ACLs: All users are allowed

Job Setup: [Successful](#)

Status: Succeeded

Started at: Mon Apr 07 03:40:38 PDT 2014

Finished at: Mon Apr 07 03:41:10 PDT 2014

Finished in: 32sec

Job Cleanup: [Successful](#)

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	2,691	2,691
	FILE: Number of bytes written	165,083	164,766	329,849
	FILE: Number of read operations	0	0	0
	FILE: Number of large read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	3,254	0	3,254
	HDFS: Number of bytes written	0	2,661	2,661
	cloudera@localhost:~/workspace/api.samples	2	0	2



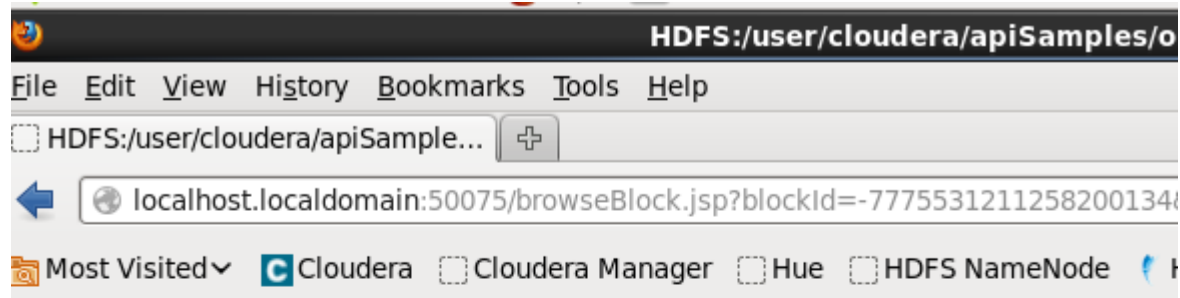
Observando los resultados

- Aunque lo hayamos llamado allWords.txt, la salida siempre es una carpeta
 - `$hdfs dfs -ls apiSamples/outputFiles/allWords.txt`
 - `$hdfs dfs -cat apiSamples/outputFiles/allWords.txt/part-r-00000`
- Suele ser habitual traer una copia al sistema de ficheros local para analizarla
 - `$hdfs dfs -get apiSamples/outputFiles/allWords.txt/part-r-00000`





Utilizando la interfaz web para ver el resultado



File: [/user/cloudera/apiSamples/outputFiles/allWords.txt/part-r-00000](#)

Goto :

[Go back to dir listing](#)

[Advanced view/download options](#)

```
a      7
acabar 1
acordarme 1
adarga 1
administración 1
afición 1
al      1
alababa 1
algo    1
```





Comprobando la versión MapReduce

The screenshot shows the Cloudera Manager web interface. The browser address bar displays 'localhost:7180/cmf/home'. The main content area lists various services: hdfs1, hive1, hue1, impala1, mapreduce1, oozie1, solr1, sqoop1, yarn1, and zookeeper1. The 'mapreduce1' service is highlighted, and a context menu is open over it. The menu options are: Start, Stop (circled in red), Restart, Rolling Restart, Rename, and Delete. To the right of the service list, there are two graphs: 'Cluster Network' (bytes / second) and 'Running MapRec' (jobs). The 'Cluster Network' graph shows a peak around 2.0K/s. The 'Running MapRec' graph shows a peak around 1 job.

Service	Status	Actions
hdfs1	Running	Stop, Restart, Rolling Restart
hive1	Running	
hue1	Running	
impala1	Running	
mapreduce1	Running	Start, Stop, Restart, Rolling Restart, Rename, Delete
oozie1	Running	
solr1	Running	
sqoop1	Running	
yarn1	Running	
zookeeper1	Running	





Gestión de jobs

- Obtener la lista de procesos Hadoop ejecutándose
 - \$ `hadoop job -list`
- Detener un job
 - \$ `hadoop job -kill "jobid"`
 - *El Jobid lo proporcionará el comando anterior*





Pero, ¡cuidado!

Debido a las dependencias existentes, los servicios deben iniciarse en un orden concreto

✓ Para iniciar los servicios

- ☐ HDFS
- ☐ MapReduce
- ☐ YARN
- ☐ ZooKeeper
- ☐ Hbase
- ☐ Hue
- ☐ Oozie
- ☐ Impala
- ☐ Flume
- ☐ Cloudera Management Services

✓ Para parar los servicios

- ☐ Cloudera Management Services (CMS)
- ☐ Flume
- ☐ Impala
- ☐ Oozie
- ☐ Hue
- ☐ HBase
- ☐ ZooKeeper
- ☐ YARN
- ☐ MapReduce
- ☐ HDFS





Interacción con los servicios

- Vamos a practicar la gestión de servicios, a través del Cloudera Manager, pararemos todos los servicios en el orden indicado anteriormente
 - Debemos tener en cuenta los que NO están iniciados
 - Apagamos CMS, Flume, zooKeeper, MapReduce, HDFS





Ejecución de comandos

- \$ hdfs dfs -ls
 - ls: Call From localhost.localdomain/127.0.0.1 to localhost.localdomain:8020 failed on connection exception: java.net.ConnectException: Connection refused...
- Desde Eclipse, intentamos ejecutar DriverIdentity
 - ¿FUNCIONA?





Eclipse y Cloudera

¿Cómo puede funcionar si tenemos desactivados todos los servicios?

- Cuando ejecutamos código MapReduce desde Eclipse, Hadoop se ejecuta en un modo especial llamado ***LocalJobRunner***.
- En este modo, los procesos de Hadoop se ejecutan bajo un mismo hilo de la JVM (Java Virtual Machine)
- También funciona porque todas las referencias al sistema de ficheros son referencias locales, no HDFS. Por ello funciona a pesar de tener desactivado el servicio HDFS
- Activemos los servicios HDFS, MapReduce, zooKeeper y CMS para continuar con el curso (en este orden)





Hadoop MapReduce Counters

- Los “contadores” forman parte de un sistema proporcionado por Hadoop para el progreso o el número de operaciones que lleva a cabo un programa MapReduce
- Son útiles para evaluar el rendimiento y mejorar el diseño del programa
- Trabajan teniendo en cuenta el proceso distribuido
- Empleados también para el resumen que aparece al final de la ejecución
- Es posible que el usuario personalice sus propios contadores
 - Por ejemplo, podemos poner un contador para saber cuántos registros “falsos” o atípicos encontramos
- No abusar de ellos, consumen recursos



Contadores agrupados por tipo al final de la ejecución



MBIT School
Madrid Business Intelligence Technology

File System Counters

FILE: Number of bytes read=12858

FILE: Number of bytes written=340709

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

Map-Reduce Framework

Map input records=29

Map output records=542

Map output bytes=5180

Map output materialized bytes=6270

Input split bytes=132

Combine input records=0

.....





Built-in MapReduce task counters

Counter	Description
Map input records (MAP_INPUT_RECORDS)	The number of input records consumed by all the maps in the job. Incremented every time a record is read from a <code>RecordReader</code> and passed to the map's <code>map()</code> method by the framework.
Map skipped records (MAP_SKIPPED_RECORDS)	The number of input records skipped by all the maps in the job. See “Skipping Bad Records” on page 217 .
Map input bytes (MAP_INPUT_BYTES)	The number of bytes of uncompressed input consumed by all the maps in the job. Incremented every time a record is read from a <code>RecordReader</code> and passed to the map's <code>map()</code> method by the framework.
Split raw bytes (SPLIT_RAW_BYTES)	The number of bytes of input split objects read by maps. These objects represent the split metadata (that is, the offset and length within a file) rather than the split data itself, so the total size should be small.
Map output records (MAP_OUTPUT_RECORDS)	The number of map output records produced by all the maps in the job. Incremented every time the <code>collect()</code> method is called on a map's <code>OutputCollector</code> .
Map output bytes (MAP_OUTPUT_BYTES)	The number of bytes of uncompressed output produced by all the maps in the job. Incremented every time the <code>collect()</code> method is called on a map's <code>OutputCollector</code> .





Built-in MapReduce task counters

Map output materialized bytes (MAP_OUTPUT_MATERIALIZED_BYTES)	The number of bytes of map output actually written to disk. If map output compression is enabled this is reflected in the counter value.
Combine input records (COMBINE_INPUT_RECORDS)	The number of input records consumed by all the combiners (if any) in the job. Incremented every time a value is read from the combiner's iterator over values. Note that this count is the number of values consumed by the combiner, not the number of distinct key groups (which would not be a useful metric, since there is not necessarily one group per key for a combiner; see "Combiner Functions" on page 34 , and also "Shuffle and Sort" on page 205).
Combine output records (COMBINE_OUTPUT_RECORDS)	The number of output records produced by all the combiners (if any) in the job. Incremented every time the <code>collect()</code> method is called on a combiner's <code>OutputCollector</code> .
Reduce input groups (REDUCE_INPUT_GROUPS)	The number of distinct key groups consumed by all the reducers in the job. Incremented every time the reducer's <code>reduce()</code> method is called by the framework.
Reduce input records (REDUCE_INPUT_RECORDS)	The number of input records consumed by all the reducers in the job. Incremented every time a value is read from the reducer's iterator over values. If reducers consume all of their inputs, this count should be the same as the count for Map output records.
Reduce output records (REDUCE_OUTPUT_RECORDS)	The number of reduce output records produced by all the maps in the job. Incremented every time the <code>collect()</code> method is called on a reducer's <code>OutputCollector</code> .
Reduce skipped groups (REDUCE_SKIPPED_GROUPS)	The number of distinct key groups skipped by all the reducers in the job. See "Skipping Bad Records" on page 217 .
Reduce skipped records (REDUCE_SKIPPED_RECORDS)	The number of input records skipped by all the reducers in the job.
Reduce shuffle bytes	The number of bytes of map output copied by the shuffle to reducers.





Built-in MapReduce task counters

Counter	Description
(REDUCE_SHUFFLE_BYTES)	
Spilled records (SPILLED_RECORDS)	The number of records spilled to disk in all map and reduce tasks in the job.
CPU milliseconds (CPU_MILLISECONDS)	The cumulative CPU time for a task in milliseconds, as reported by <i>/proc/cpuinfo</i> .
Physical memory bytes (PHYSICAL_MEMORY_BYTES)	The physical memory being used by a task in bytes, as reported by <i>/proc/meminfo</i> .
Virtual memory bytes (VIRTUAL_MEMORY_BYTES)	The virtual memory being used by a task in bytes, as reported by <i>/proc/meminfo</i> .
Committed heap bytes (COMMITTED_HEAP_BYTES)	The total amount of memory available in the JVM in bytes, as reported by <code>Runtime.getRuntime().totalMemory()</code> .
GC time milliseconds (GC_TIME_MILLIS)	The elapsed time for garbage collection in tasks in milliseconds, as reported by <code>GarbageCollectorMXBean.getCollectionTime()</code> . From 0.21.
Shuffled maps (SHUFFLED_MAPS)	The number of map output files transferred to reducers by the shuffle (see “Shuffle and Sort” on page 205). From 0.21.
Failed shuffle (FAILED_SHUFFLE)	The number of map output copy failures during the shuffle. From 0.21.
Merged map outputs (MERGED_MAP_OUTPUTS)	The number of map outputs that have been merged on the reduce side of the shuffle. From 0.21.





Otros

Table 8-3. Built-in filesystem task counters

Counter	Description
<i>Filesystem</i> bytes read (BYTES_READ)	The number of bytes read by each filesystem by map and reduce tasks. There is a counter for each filesystem: <i>Filesystem</i> may be Local, HDFS, S3, KFS, etc.
<i>Filesystem</i> bytes written (BYTES_WRITTEN)	The number of bytes written by each filesystem by map and reduce tasks.

Table 8-4. Built-in FileInputFormat task counters

Counter	Description
Bytes read (BYTES_READ)	The number of bytes read by map tasks via the FileInputFormat.

Table 8-5. Built-in FileOutputFormat task counters

Counter	Description
Bytes written (BYTES_WRITTEN)	The number of bytes written by map tasks (for map-only jobs) or reduce tasks via the FileOutputFormat.





Built-in Job Counters

Table 8-6. Built-in job counters

Counter	Description
Launched map tasks (TOTAL_LAUNCHED_MAPS)	The number of map tasks that were launched. Includes tasks that were started speculatively.
Launched reduce tasks (TOTAL_LAUNCHED_REDUCES)	The number of reduce tasks that were launched. Includes tasks that were started speculatively.
Launched uber tasks (TOTAL_LAUNCHED_UBERTASKS)	The number of uber tasks (see “YARN (MapReduce 2)” on page 194) that were launched. From 0.23.
Maps in uber tasks (NUM_UBER_SUBMAPS)	The number of maps in uber tasks. From 0.23.
Reduces in uber tasks (NUM_UBER_SUBREDUCES)	The number of reduces in uber tasks. From 0.23.
Failed map tasks (NUM_FAILED_MAPS)	The number of map tasks that failed. See “Task Failure” on page 200 for potential causes.
Failed reduce tasks (NUM_FAILED_REDUCES)	The number of reduce tasks that failed.
Failed uber tasks (NUM_FAILED_UBERTASKS)	The number of uber tasks that failed. From 0.23.
Data-local map tasks (DATA_LOCAL_MAPS)	The number of map tasks that ran on the same node as their input data.





Built-in Job Counters

Rack-local map tasks
(`RACK_LOCAL_MAPS`)

The number of map tasks that ran on a node in the same rack as their input data, but that are not data-local.

Other local map tasks
(`OTHER_LOCAL_MAPS`)

The number of map tasks that ran on a node in a different rack to their input data. Inter-rack bandwidth is scarce, and Hadoop tries to place map tasks close to their input data, so this count should be low. See [Figure 2-2](#).

Total time in map tasks
(`SLOTS_MILLIS_MAPS`)

The total time taken running map tasks in milliseconds. Includes tasks that were started speculatively.

Total time in reduce tasks
(`SLOTS_MILLIS_REDUCE`)

The total time taken running reduce tasks in milliseconds. Includes tasks that were started speculatively.

Total time in map tasks waiting after reserving slots
(`FOLLOW_SLOTS_MILLIS_MAPS`)

The total time spent waiting after reserving slots for map tasks in milliseconds. Slot reservation is Capacity Scheduler feature for high-memory jobs, see [“Task memory limits” on page 316](#). Not used by YARN-based MapReduce.

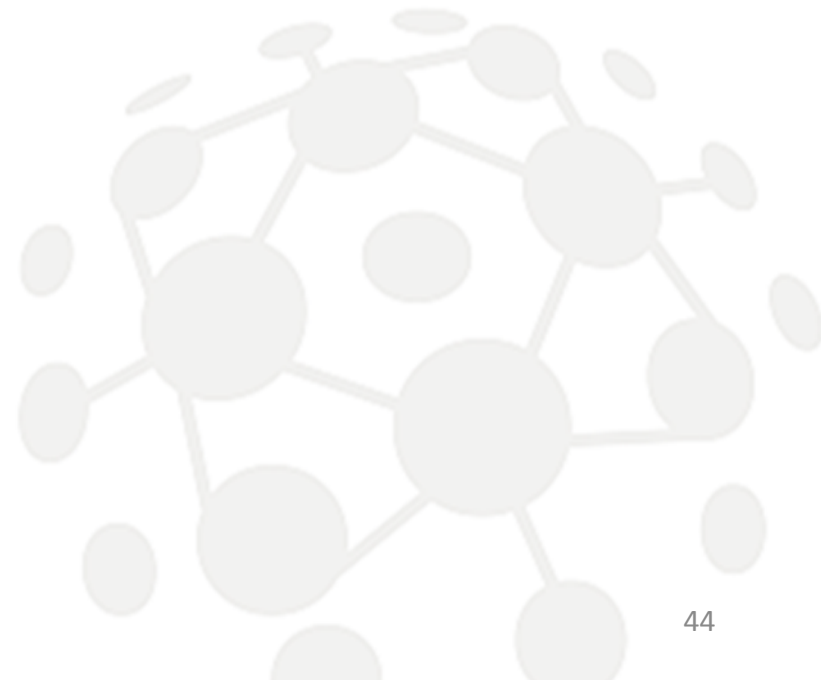




Un contador propio

```
public static enum MY_COUNTER {  
    OK_INPUT, INVALID_INPUT  
};
```

- Muy frecuente emplear estructuras para agruparlos





Ejemplo

- Cogemos el ejemplo WordCount
- En el Mapper programamos un filtro para quitar del proceso aquellas expresiones que no son palabras y tienen, por ejemplo, números
- Contaremos el número de expresiones que descartamos mapear por no ajustarse al patrón
- Debe proporcionarse un nombre para el grupo del contador y para el contador mismo
- Los contadores también se pueden emitir al contexto





Implementación

```
if (word.matches("[a-záéíóúñç]+")) {  
    WordAsText.set(word);  
    context.write(WordAsText, IntWritable_1);  
    // Declare Counter and increment by 1 using increment method  
    context.getCounter("MyCounters", "OK_Input").increment(1);  
}  
  
else {  
    // Declare Counter and increment by 1 using increment method  
    context.getCounter("MyCounters", "Invalid_Input").increment(1);  
}
```





Ejecución

```
Reduce input records=0
Reduce output records=291
Spilled Records=1084
Shuffled Maps =0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=37
CPU time spent (ms)=0
Physical memory (bytes) snapshot=0
Virtual memory (bytes) snapshot=0
Total committed heap usage (bytes)=331227136
MyCounters
  Invalid_Input=569
  OK_Input=542
File Input Format Counters
  Bytes Read=3106
File Output Format Counters
  Bytes Written=2693
```





Empleando estructuras

En este caso emplearemos el ejemplo WordCountCombiner

```
public static enum MY_COUNTER {  
    OK_INPUT, INVALID_INPUT  
};  
  
@Override  
protected void map(LongWritable key, Text textLine, Context context)  
    throws IOException, InterruptedException {  
    // TODO Auto-generated method stub  
    for (String word : textLine.toString().split("\\b+")) {  
        word = word.toLowerCase();  
        if (word.matches("[a-zAÉÍÓÚÑÇ]+")) {  
            WordAsText.set(word);  
            context.write(WordAsText, IntWritable_1);  
            context.getCounter(MY_COUNTER.OK_INPUT).increment(1);  
        } else {  
            context.getCounter(MY_COUNTER.INVALID_INPUT).increment(1);  
        }  
    }  
}
```





Resultado 2

```
Spilled Records=0
Shuffled Maps =0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=55
CPU time spent (ms)=0
Physical memory (bytes) snapshot=0
Virtual memory (bytes) snapshot=0
Total committed heap usage (bytes)=331227136
com.utad.api.wordcountcombiner.MapperWordCountCombiner$MY_COUNTER
INVALID_INPUT=569
OK INPUT=542
File Input Format Counters
  Bytes Read=3106
File Output Format Counters
  Bytes Written=2693
```





Manipulando los contadores en tiempo de ejecución

- El objeto “job” empleado en el Driver dispone del método `.getCounters()` para acceder a los distintos contadores disponibles para el proceso concreto
- Esta lista de contadores se puede almacenar y procesar a través de una instancia de la clase *Counters*
- El método `.findCounter(grupo, nombre)` nos permitirá acceder a un contador concreto





Ejemplo 3

```
boolean result = job.waitForCompletion(true);

Counters counters = job.getCounters();
long ok_inputs =
counters.findCounter("MyCounters", "OK_Input").getValue();

long invalid_inputs =
    counters.findCounter("MyCounters", "Invalid_Input").getValue();
System.out.println("-----");
System.out.println("Ok_Inputs = " + ok_inputs);
System.out.println("Invalid_Inputs = " + invalid_inputs);
System.out.println("RATIO OK/INVALID "
    + (((double) ok_inputs) / invalid_inputs));
System.out.println("-----");

return result ? 0 : 1;
}
```





Resultado 3

```
Failed Spillies=0
Merged Map outputs=0
GC time elapsed (ms)=44
CPU time spent (ms)=0
Physical memory (bytes) snapshot=0
Virtual memory (bytes) snapshot=0
Total committed heap usage (bytes)=331227136
MyCounters
  Invalid_Input=569
  OK_Input=542
File Input Format Counters
  Bytes Read=3106
File Output Format Counters
  Bytes Written=2693
```

```
-----
Ok_Inputs = 542
Invalid_Inputs = 569
RATIO OK/INVALID 0.9525483304042179
-----
```





Test Driven Development (TDD)

Ciclo de Pruebas de Calidad de Software

- Estimar esfuerzo de prueba
- Diseño de la estrategia y plan de pruebas
- Elaboración de pruebas por componentes
- Ejecutar las pruebas
- Informe de resultados e incidencias
- Registros e informes finales

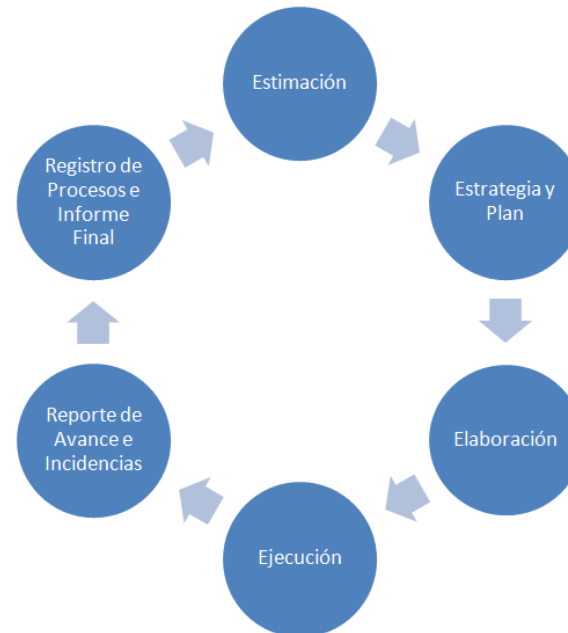


Imagen elaborada por www.pmoinformatica.com





Pruebas unitarias

- Las pruebas unitarias pretenden verificar que cada uno de los componentes funcionen individualmente como se supone que lo tiene que hacer
- Se complementan con las pruebas de integración donde se verifica que el conjunto interactúa según lo esperado
- Se debe centrar el enfoque en **localizar errores**, no en corregir errores
- Deben ser fácil de realizar y centrarse sobre la menor porción de código posible



- Es un framework que permite gestionar de forma y sencilla la fase de pruebas unitarias del código
- Evalúa si el funcionamiento de cada uno de los métodos de la clase es el esperado
- También es útil para controlar las pruebas de regresión. Éstas se realizan cuando un código se ha modificado y queremos comprobar que la modificación no ha alterado la funcionalidad
- En el caso de MapReduce, disponemos de la utilidad MRUnit





MRUnit

- Se integra con JUnit
- Ejecuta las pruebas de procesos MapReduce dentro del framework JUnit

Apache
MRUnitTM





Pruebas MRUnit en WordCount

- Comenzamos añadiendo la librería correspondiente en el archivo de configuración pom.xml

```
<dependency>  
  <groupId>org.apache.mrunit</groupId>  
  <artifactId>mrunit</artifactId>  
  <version>1.1.0</version>  
  <classifier>hadoop2</classifier>  
  <scope>test</scope>  
</dependency>
```
















- ☐ “Hadoop2” porque es para la librería nueva
- ☐ “Test” para no usarlo en producción, sólo en modo debug





Comprobando la configuración

En Maven Dependencies

- ▷  management-api-3.0.0-b012.jar - /home/...
- ▷  grizzly-http-server-2.1.1.jar - /home/...
- ▷  grizzly-rcm-2.1.1.jar - /home/cloud...
- ▷  grizzly-http-servlet-2.1.1.jar - /home/...
- ▷  grizzly-framework-2.1.1-tests.jar -
- ▷  jersey-guice-1.8.jar - /home/cloud...
- ▷  hadoop-yarn-server-common-2.0.0
- ▷  hadoop-mapreduce-client-shuffle-2
- ▷  netty-3.2.4.Final.jar - /home/cloud...
- ▷  hadoop-yarn-api-2.0.0-cdh4.2.0.jar
- ▷  hadoop-mapreduce-client-core-2.0.
- ▷  hadoop-yarn-common-2.0.0-cdh4.2
- ▷  hadoop-mapreduce-client-jobclient
- ▷  mrunit-1.0.0-hadoop2.jar - /home/c
- ▷  JRE System Library [JavaSE-1.6]





Creando una prueba

- Dentro de src/test/java
- Vamos a probar el wordcountcombiner
- El nombre del paquete puede ser `com.utad.api.test.wordcountcombiner`
- Clase: `TestWordCount`





TestWordCount

- Declaramos una clase privada *MapReduceDriver* que nos permitirá lanzar un proceso MapReduce
 - Como es similar a un driver, necesito especificar la **entrada** y **la salida** de la fase mapper y la **salida** de la fase reducer
 - `private MapReduceDriver <LongWritable, Text, Text, IntWritable, Text, LongWritable> MyMapReduceDriver;`
 - Importar la librería correcta `...mrunit.mapreduce...`





Configurando la prueba

- Necesitamos crear un método que es el que vamos a ejecutar
 - `public void configTest ()`
 - Antes del método colocaremos la anotación `@Before`, empleada para decir a JUnit que queremos ejecutar este método antes de cualquiera otra prueba
- Instanciación
 - Se emplea el método `MapReduceDriver.newMapReduceDriver(...)`
 - El método recibe un mapper y un reducer que deben ser también instanciados
 - Atendiendo al código anterior que hemos hecho, podemos pasar *`new MapperWordCountCombiner ()`* y *`new ReducerWordCountCombiner()`*
 - Importamos las clases Mapper y Reducer que necesitamos





Añadiendo un Combiner

- Una vez instanciado nuestro objeto `MyMapReduceDriver`, podemos emplear el método `setCombiner()` para añadir un Combiner al proceso
 - *`MyMapReduceDriver.setCombiner(new CombinerWordCountCombiner())`*





El método de ejecución

- Crearemos un método llamado `runMapReduceTest()`
- Etiquetamos el método con la anotación `@Test` para indicar a JUnit dónde está la prueba que queremos realizar
- Para realizar la prueba, no leeremos un fichero de texto, sino que pasaremos nosotros una línea y estudiaremos cómo reacciona





Añadiendo la entrada para la prueba

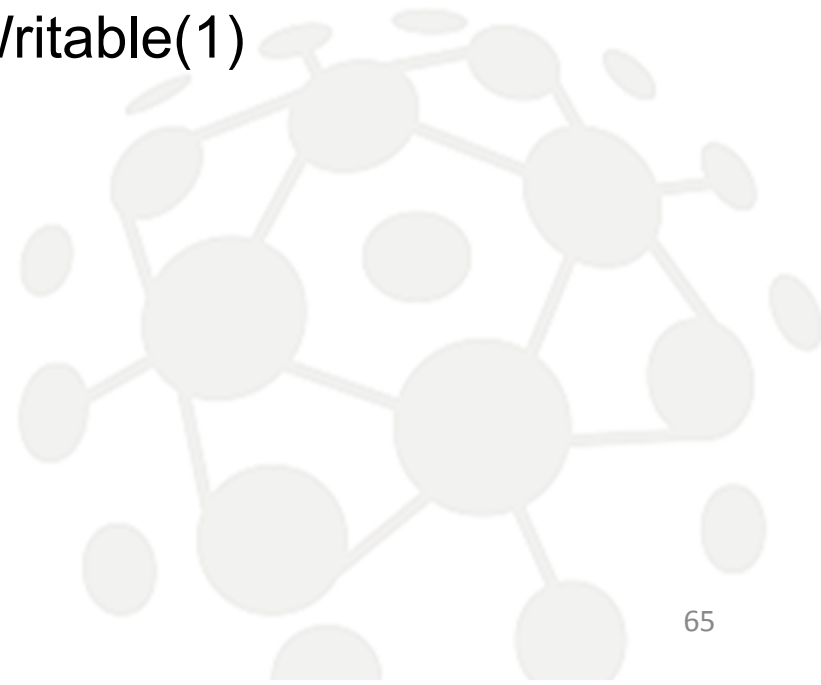
- `.withInput()` es el método que emplearemos para añadir una entrada a nuestra prueba `MyMapReduceDriver()`
- Este método debe recibir los elementos necesarios para ejecutar una prueba, en nuestro caso:
 - `new LongWritable(0)`
 - `new Text("Hola que tal todo 44 5 hola TODO es")`





Especificando la salida esperada

- Tenemos varios métodos como opciones
- Emplearemos por ejemplo el método `.addOutput()`
- Este método recibe como parámetros pares <clave, valor> que se esperan en la salida
- Por ejemplo y, en base al texto introducido en la prueba
 - `new Text("que"), new LongWritable(1)`





runTest

- El método `.runTest` es el método encargado de lanzar la prueba
- Es un método sobrecargado que presenta dos opciones
 - Un opción sin parámetros
 - Una opción con Booleanos ***empleado en el caso en que el orden de la salida deba ser el mismo que el especificado***
 - Usaremos `.runTest(false)` indicando que no nos importa el orden





```
new 1 x
9  import com.utad.api.wordcountcombiner.ReducerWordCountCombiner;
10 import com.utad.api.wordcountcombiner.CombinerWordCountCombiner;
11
12 public class TestWordCount {
13
14     private MapReduceDriver<LongWritable, Text, Text, IntWritable, Text, LongWrita
15
16     @Before
17     public void configTest() {
18         MyMapReduceDriver =
19             MapReduceDriver.newMapReduceDriver(new MapperWordCountCombiner(),
20                 new ReducerWordCountCombiner());
21         MyMapReduceDriver.setCombiner(new CombinerWordCountCombiner());
22     }
23
24     @Test
25     public void runMapReduceTest() throws IOException {
26         // Input
27         MyMapReduceDriver.withInput(new LongWritable(0), new Text(
28             "Hola que tal todo 44 5 hola TODO es"));
29         // Puedo añadir más input línea a línea de la misma forma
30
31         // Output
32         MyMapReduceDriver.addOutput(new Text("hola"), new LongWritable(2));
33         MyMapReduceDriver.addOutput(new Text("que"), new LongWritable(1));
34         MyMapReduceDriver.addOutput(new Text("tal"), new LongWritable(1));
35         MyMapReduceDriver.addOutput(new Text("todo"), new LongWritable(2));
36         MyMapReduceDriver.addOutput(new Text("44"), new LongWritable(1));
37         MyMapReduceDriver.addOutput(new Text("5"), new LongWritable(1));
38         MyMapReduceDriver.addOutput(new Text("es"), new LongWritable(1));
39
40         MyMapReduceDriver.runTest(false);
41
42     }
```





The screenshot shows an IDE window with a Java file named `m/utad/api/test/wordcount/TestWord`. The code defines a `test()` method that throws `IOException` and contains several `input()` calls with `LongWritable` and `Text` objects. A red circle highlights the `JUnit Test` option in the context menu. The context menu is open, showing various actions like `Open Declaration`, `Cut`, `Copy`, `Paste`, `Quick Fix`, `Source`, `Refactor`, `Local History`, `References`, `Declarations`, `Add to Snippets...`, `Run As` (highlighted), `Debug As`, `Validate`, `Team`, `Compare With`, `Replace With`, and `Preferences`. The `Run As` option is highlighted in blue.

```
test() throws IOException {  
    input(new LongWritable(0), new Text(  
        "44 5 hola TODO es"));  
  
    input(new Text("hola"), new LongWritable(  
        0));  
    input(new Text("que"), new LongWritable(  
        0));  
    input(new Text("tal"), new LongWritable(  
        0));  
    input(new Text("todo"), new LongWritable(  
        0));  
    input(new Text("44"), new LongWritable(2));  
    input(new Text("5"), new LongWritable(2));  
    input(new Text("es"), new LongWritable(2));  
  
    test(false);  
}
```

JUnit Test Shift+Alt+X T

Run Configurations...

Declaration Console

Application] /usr/java/jdk1.6.0_31/bin/java

Writable Sm





No hemos
contado bien.

Corregimos y
volvemos a
probar

Finished after 1.82 seconds

Runs: 1/1 Errors: 0 Failures: 1

com.utad.api.test.wordcount.TestWordCount [f
runMapReduceTest (1.798 s)

Failure Trace

ected output (que, 2), Missing expected output (e
do, 1). Received unexpected output (que, 1). Rec
assertNone(Errors.java:61)
TestDriver.java:711)
TestDriver.java:575)

```
26 }  
27  
28 @Test  
29 public void runMapReduceTest() throws IOException {  
30     // Input  
31     MyMapReduceDriver.withInput(new LongWritable(0), new Text(  
32         "Hola que tal todo 44 5 hola TODO es"));  
33     //Puedo añadir más input de la misma forma  
34  
35     // Output  
36     MyMapReduceDriver.addOutput(new Text("hola"), new LongWritable(2));  
37     MyMapReduceDriver.addOutput(new Text("que"), new LongWritable(2));  
38     MyMapReduceDriver.addOutput(new Text("tal"), new LongWritable(2));  
39     MyMapReduceDriver.addOutput(new Text("todo"), new LongWritable(2));  
40     MyMapReduceDriver.addOutput(new Text("44"), new LongWritable(2));  
41     MyMapReduceDriver.addOutput(new Text("5"), new LongWritable(2));  
42     MyMapReduceDriver.addOutput(new Text("es"), new LongWritable(2));  
43  
44     MyMapReduceDriver.runTest(false);  
45  
46 }  
47  
48 }
```

Problems @ Javadoc Declaration Console

<terminated> TestWordCount (1) [JUnit] /usr/java/jdk1.6.0_31/bin/java (Apr 12, 2014)





Seguimos teniendo un error

- Estamos contando los números como palabras
- Debemos recordar que pusimos un filtro (con expresiones regulares) para descartar ciertas cadenas

```
44 MyMapReduceDriver.runTest(false);
```

Problems @ Javadoc Declaration Console

<terminated> TestWordCount (1) [JUnit] /usr/java/jdk1.6.0_31/bin/java (Apr 12, 2014 3:05:02 PM)

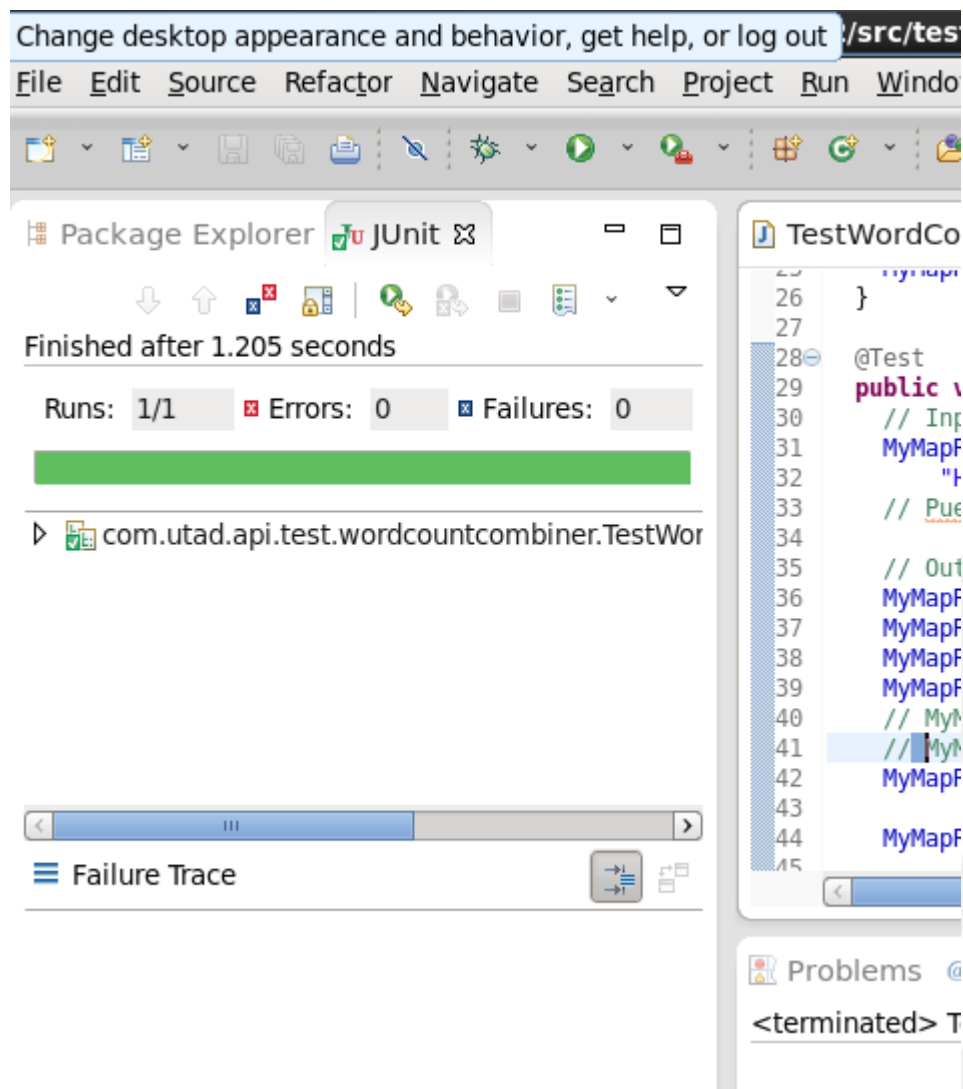
2014-04-12 15:05:04 ERROR TestDriver:49 - Missing expected output (5, 1)

2014-04-12 15:05:04 ERROR TestDriver:49 - Missing expected output (44, 1)





Si quitamos las líneas erróneas...





MRUnit – Varias pruebas

- Se pueden añadir varias líneas en un texto
- Se pueden añadir varias pruebas dentro de una misma clase prueba
- Para ello, introduciremos dos métodos distintos (ambos con la anotación **@Test**)
 - Esta anotación es la que hace que se lance “una prueba” JUnit





Contadores en las pruebas

- También puede resultar útil introducir algún contador en la prueba con el objetivo de determinar si el valor medido es el esperado
- JUnit emplea “assertEquals” para comparar dos valores





Ejemplo con contador

```
@Test
public void runMapReduceTest() throws IOException {
    // Input
    MyMapReduceDriver.withInput(new LongWritable(0), new Text(
        "Hola que tal todo 44 5 hola TODO es"));
    // Puedo añadir más input línea a línea de la misma forma

    // Output
    MyMapReduceDriver.addOutput(new Text("hola"), new LongWritable(2));
    MyMapReduceDriver.addOutput(new Text("que"), new LongWritable(1));
    MyMapReduceDriver.addOutput(new Text("tal"), new LongWritable(1));
    MyMapReduceDriver.addOutput(new Text("todo"), new LongWritable(2));
    // MyMapReduceDriver.addOutput(new Text("44"), new LongWritable(1));
    // MyMapReduceDriver.addOutput(new Text("5"), new LongWritable(1));
    MyMapReduceDriver.addOutput(new Text("es"), new LongWritable(1));

    MyMapReduceDriver.runTest(false);

    assertEquals(7,
        MyMapReduceDriver.getCounters().findCounter(MY_COUNTER.OK_INPUT)
            .getValue());
}
```





Resultado



Java - api.samples2/src/test/java/com/

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 2.23 seconds

Runs: 1/1 Errors: 0 Failures: 0

com.utad.api.test.wordcountcombiner.TestWor

runMapReduceTest (2.165 s)

Failure Trace

```
30
31 @Test
32 public void runMapRe
33 // Input
34 MyMapReduceDriver.
35 "Hola que tal
36 // Puedo añadir má
37
38 // Output
39 MyMapReduceDriver.
40 MyMapReduceDriver.
41 MyMapReduceDriver.
42 MyMapReduceDriver.
43 // MyMapReduceDriv
44 // MyMapReduceDriv
45 MyMapReduceDriver.
46
47 MyMapReduceDriver.
48
49
50 assertEquals(7,
```





Empaquetado y distribución

- Al empaquetar la solución con maven, se ejecutan las pruebas
- Sólo se produce el empaquetado si las pruebas NO fallan
- Run As → Maven Build
 - Goals: package

T E S T S

Running [com.utad.api.test.wordcountcombiner.TestWordCount](#)

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.78 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0





Resumen

- Se ha insertado un Combiner al ejemplo WordCount
- Sabemos como distribuir un proyecto MapReduce codificado a través del IDE Eclipse
- Hemos comprendido la importancia de los contadores para analizar las características de un proceso MapReduce
- Conocemos los fundamentos de las pruebas unitarias en MapReduce

