



Conceptos y Aplicaciones de Big Data

Ecosistema Hadoop

Prof. Waldo Hasperué
whasperue@lidi.info.unlp.edu.ar

Temario

- Ecosistema Hadoop
 - HDFS
 - Ejecución de una aplicación en el entorno Hadoop
- Introducción al paradigma MapReduce
- Etapas de un trabajo en MapReduce
 - Map
 - Shuffle
 - Sort
 - Reduce

Historia

- Para procesar grandes conjuntos de datos, en 2003 Google creó el framework Hadoop capaz de poder procesar grandes volúmenes de datos.
- En 2006, Yahoo continúa con el desarrollo del proyecto Hadoop. Aparece Hadoop MapReduce.
- Actualmente pertenece a Apache
 - Apache Hadoop (hadoop.apache.org)

Hadoop

- Es un framework que soporta procesamiento de grandes bases de datos en un ambiente distribuido
- Ejecuta aplicaciones para el tratamiento de grandes volúmenes de datos
- Incluye un sistema de archivos distribuidos (HDFS)
- Tolerante a fallas

Hadoop

- ✓ Diseñado para el procesamiento off-line de los datos (procesamiento en batch)
- ✓ Funciona con la idea de "escriba una sola vez y lea muchas"
- ✗ No permite lectura aleatoria
- ✗ No permite el procesamiento on-line
- Se ejecuta en el "lugar" donde se encuentran los datos

Componentes Hadoop

- **Common** (I/O, serialización, RPC)
- **HDFS** (file system distribuido)
- **Zookeeper** (servicio de coordinación de procesos)
- **MapReduce** (modelo de procesamiento de datos)
- **Pig** (lenguaje de scripting sobre MapReduce)
- **Cascading** (framework que simplifica el uso de MapReduce)
- **Hive** (lenguaje basado en SQL)

Componentes Hadoop

- Almacenamiento: Hadoop Distributed File System (HDFS)
 - Los archivos están distribuidos
 - Un mismo archivo podría estar almacenado en varias computadoras.
 - Ofrece transparencia al usuario permitiendo operar con todos los archivos del file system.

Componentes Hadoop

- En Hadoop la administración de los procesos que se ejecutan en el cluster la lleva a cabo un framework llamado Yarn MapReduce.
- Básicamente Yarn realiza los trabajos usando dos procesos diferentes:
 - Job tracker: maneja todos los trabajos a ser procesados. Tiene en cuenta el mapa del cluster al momento de crear los procesos Task
 - Task tracker: son los encargados de realizar el procesamiento de los datos



Apache Hadoop Ecosystem



Ambari

Provisioning, Managing and Monitoring Hadoop Clusters



Scoop

Data Exchange



Flume

Log Collector



Zookeeper

Coordination



Oozie

Workflow



Pig

Scripting



Mahout

Machine Learning

R Connectors

Statistics



Hive

SQL Query



YARN Map Reduce v2

Distributed Processing Framework

HDFS

Hadoop Distributed File System



**APACHE
HBASE**

Hbase

Columnar Store

DFS (Distributed File System)

- Cuando el volumen de datos a almacenar supera la capacidad de una única computadora es necesario guardarlo en varias (cluster)
- Resulta interesante ver TODO el contenido del cluster como si fuera un único filesystem

DFS (Distributed File System)

- El filesystem distribuído nos permite realizar esa tarea, vemos un **único** filesystem (archivos y carpetas) pero estos pueden estar físicamente almacenados en diferentes máquinas dentro de un cluster.
- Hadoop tiene su propio filesystem distribuido: el HDFS (Hadoop Distributed FileSystem)

DFS

- Hay varios sistemas de archivos distribuidos
 - HDFS
 - HFTP
 - HSFTP
 - HAR
 - FTP
 - S3

HDFS

- Los archivos están distribuidos
- Ofrece transparencia al usuario permitiendo operar con todos los archivos del file system.
- Un mismo archivo podría estar almacenado en varias computadoras.
- Tolerante a fallas: si un nodo se cae los datos almacenados en ese nodo deberían estar accesibles.

HDFS

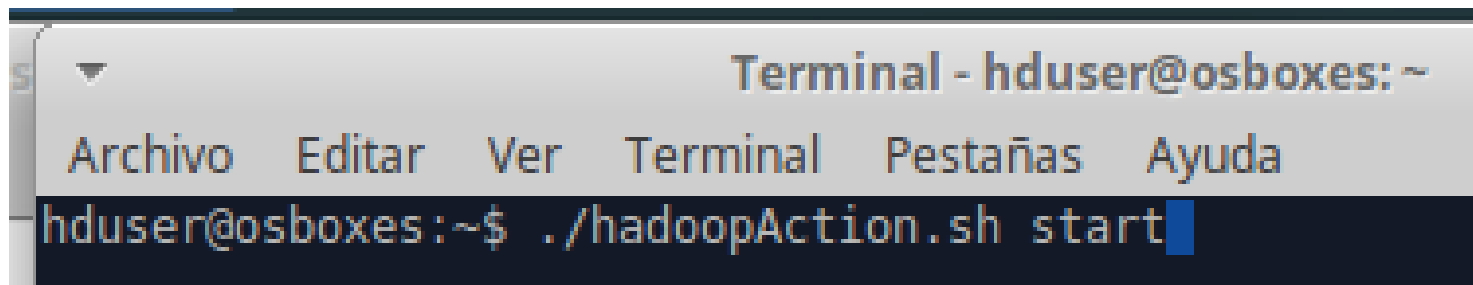
- Todos los archivos se dividen en bloques del mismo tamaño (64MB por defecto, aunque es configurable)
- Los bloques pueden estar físicamente en cualquier computadora
- Permite la réplica de bloques para optimización y recupero de fallas

Procesos del HDFS

- Namenode
 - Maneja el árbol del filesystem y los metadatos de cada archivo y carpeta.
 - Conoce para cada bloque del FS que datanode lo maneja.
 - Vínculo con el filesystem del SO
- Datanode
 - Son lo que llevan a cabo la lectura y escritura de los bloques en el filesystem del SO.
 - Llave a cabo la creación, borrado y replicado de los bloques.
- Secondary namenode: realiza tareas auxiliares al name node.

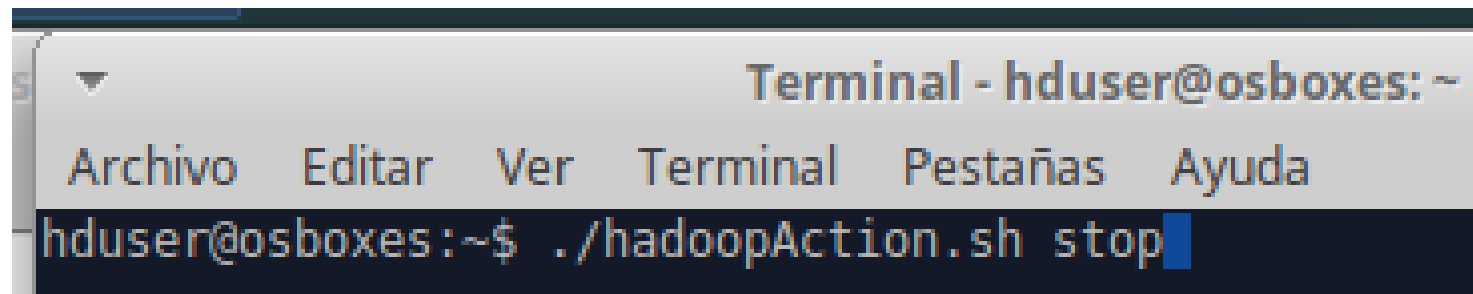
Hadoop - HDFS

- Para iniciar o detener el servicio de Hadoop y el HDFS se puede usar el script `hadoopAction.sh`



A screenshot of a terminal window titled "Terminal - hduser@osboxes: ~". The window has a menu bar with "Archivo", "Editar", "Ver", "Terminal", "Pestañas", and "Ayuda". The command prompt shows "hduser@osboxes:~\$./hadoopAction.sh start" with a blue cursor at the end of the line.

```
Terminal - hduser@osboxes: ~
Archivo  Editar  Ver    Terminal  Pestañas  Ayuda
hduser@osboxes:~$ ./hadoopAction.sh start
```



A screenshot of a terminal window titled "Terminal - hduser@osboxes: ~". The window has a menu bar with "Archivo", "Editar", "Ver", "Terminal", "Pestañas", and "Ayuda". The command prompt shows "hduser@osboxes:~\$./hadoopAction.sh stop" with a blue cursor at the end of the line.

```
Terminal - hduser@osboxes: ~
Archivo  Editar  Ver    Terminal  Pestañas  Ayuda
hduser@osboxes:~$ ./hadoopAction.sh stop
```


Hadoop

- Al iniciar el servicio corroborar que se inicien los cinco servicios esenciales de Hadoop.

```
master: starting nodemanager, logging to /usr/
manager-master.out
2480 NameNode
2948 ResourceManager
2805 SecondaryNameNode
3065 NodeManager
2601 DataNode
3098 Jps
starting historyserver, logging to /usr/local/
erver-master.out
hduser@master:~$
```

El comando HDFS

- HDFS permite crear, borrar, renombrar archivos y carpetas dentro del FS distribuido.
- Ofrece dos operaciones adicionales
 - Copiar un archivo del FS local al HDFS
 - Copiar un archivo del HDFS al FS local

El comando HDFS

- Listar archivos y directorios
 - `hdfs dfs -ls`
 - `hdfs dfs -ls <nombre_directorio>`
- Ver contenido de un archivo
 - `hdfs dfs -cat <nombre_archivo>`
- Crear directorio
 - `hdfs dfs -mkdir <nombre_directorio>`

El comando HDFS

- Borrar directorio
 - `hdfs dfs -rm -r <nombre_directorio>`
- Copiar archivos del FS local al DFS
 - `hdfs dfs -copyFromLocal <nomarch_FS> <nomarch_DFS>`
 - `hdfs dfs -copyFromLocal -f <nomarch_FS> <nomarch_DFS>`
(para sobrescribir)
- Copiar archivos del DFS al FS local
 - `hdfs dfs -copyToLocal <nomarch_DFS> <nomarch_FS>`

MapReduce

- Es un framework para distribuir tareas en múltiples nodos
- El espíritu de MapReduce es "escriba una vez y lea muchas"
- Ventajas
 - Paralelización y distribución de datos automática
 - Escalable
 - Tolerante a fallos
 - Monitoreo y capacidad de seguridad
 - Flexibilidad de programación (Java, Python, C#, Ruby, C++)
 - Abstracción al programador

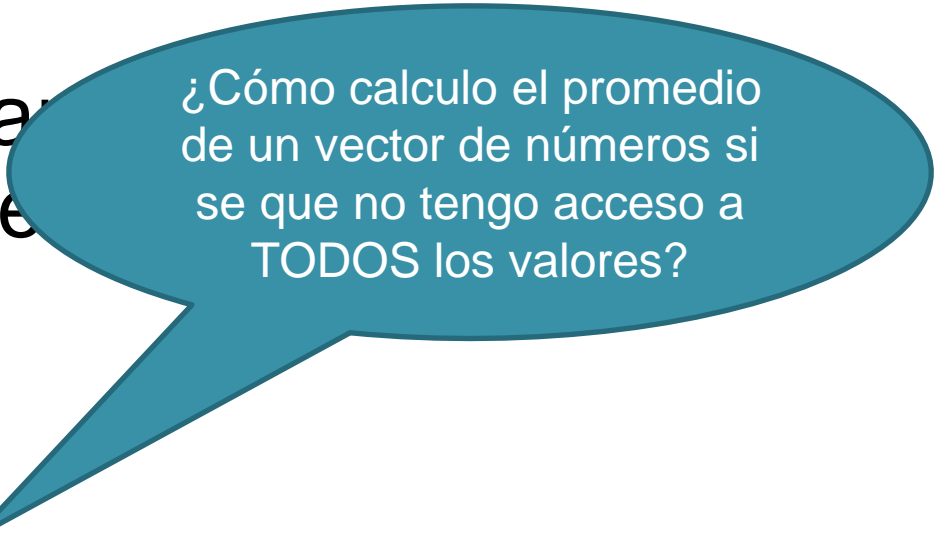
MapReduce

- Es, a su vez, un paradigma de programación.
- Hay que pensar como resolver un problema sin tener acceso a todos los datos
- Ejemplo:

```
int suma = 0, i;  
for(i=0; i<datos.length; i++)  
    suma+= datos[i];  
double promedio = suma / datos.length;
```

MapReduce

- Es, a su vez, un paradigma de programación.
- Hay que pensar el problema sin tener todos los datos



¿Cómo calculo el promedio de un vector de números si se que no tengo acceso a TODOS los valores?

- Ejemplo:

```
int suma = 0, i;  
for(i=0; i<datos.length; i++)  
    suma+= datos[i];  
double promedio = suma / datos.length;
```

MapReduce

- El problema del cálculo del promedio se debe "repensar".

```
int suma = 0, cantidad = 0, i;  
for(i = cantidad de nodos del cluster){  
    cantidad += datos_contados_nodo_i;  
    suma += datos_sumados_nodo_i;  
}  
double promedio = suma / cantidad;
```


MapReduce

- El problema del cálculo del promedio se debe "repensar".

```
int suma = 0, cantidad = 0, i;  
for(i = cantidad de nodos del cluster){  
    cantidad += datos_contados_nodo_i;  
    suma += datos_sumados_nodo_i;  
}  
double promedio = suma / cantidad;
```

Esto se
ejecuta en
paralelo

MapReduce

- Necesitamos un proceso genérico que permita resolver cualquier problema
 - Paradigma MapReduce
- Toda tarea MapReduce se divide en dos fases:
 - FaseMap, en la que los datos de entrada son procesados, uno a uno, y transformados en un conjunto intermedio de datos.
 - FaseReduce, se reúnen los resultados intermedios y se reducen a un conjunto de datos resumidos, que es el resultado final de la tarea.

MapReduce

En el ejemplo del promedio:

- Fase map: cada nodo i suma ($acumulado_i$) y cuenta (n_i) los valores que le tocó procesar.
- Fase reduce: suma todos los $acumulado_i$ y los n_i y calcula el promedio.

MapReduce

En el ejemplo del promedio:

```
int suma = 0, cantidad = 0, i;
```

Map

```
for(i = cantidad de nodos del cluster){  
    cantidad += datos_contados_nodo_i;  
    suma += datos_sumados_nodo_i;  
}
```

Reduce

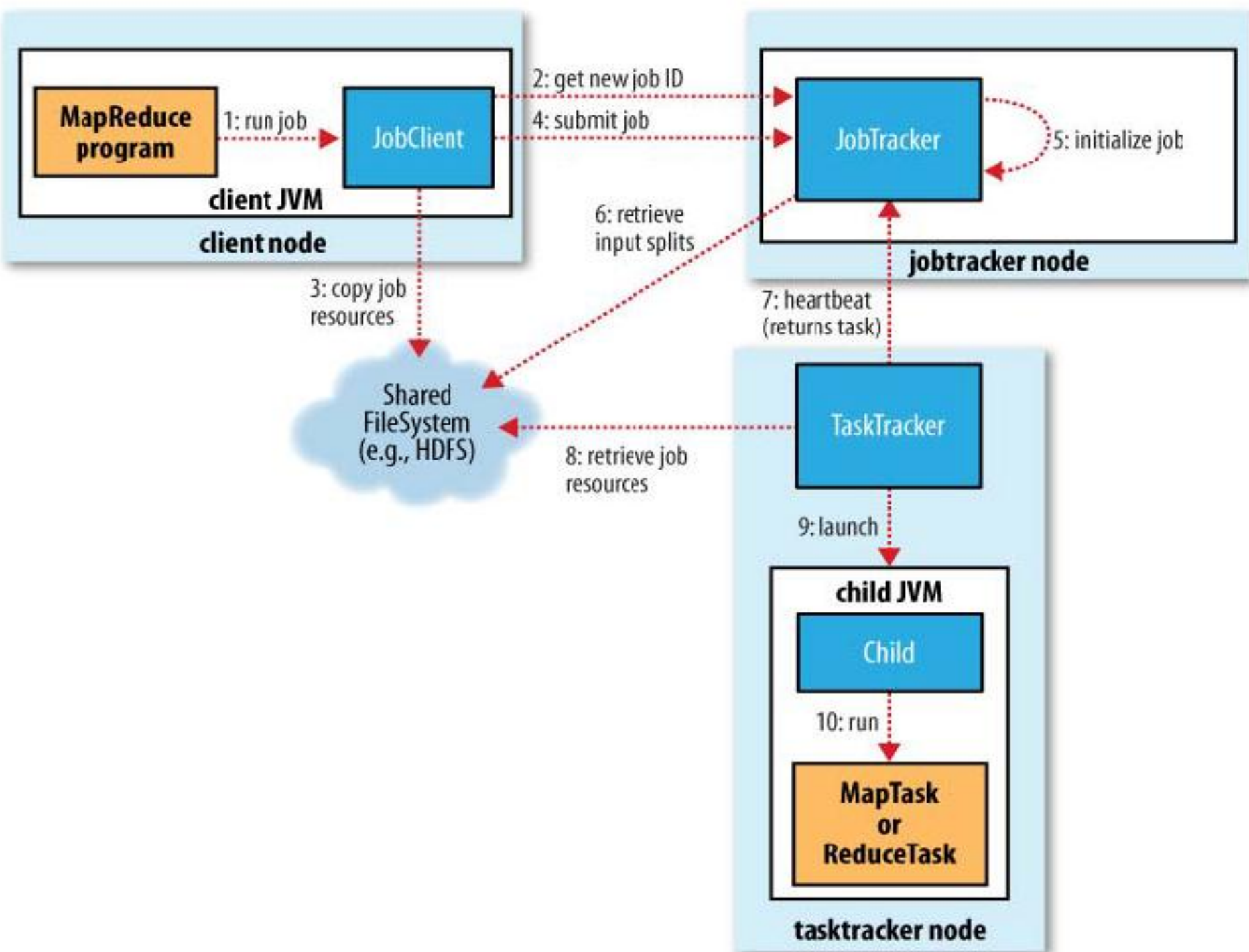
```
double promedio = suma / cantidad;
```

MapReduce

- Entrada de datos
 - MapReduce se "alimenta" de uno o mas archivos:
 - En el caso de archivos de texto, plano, cada línea del archivo es un dato a procesar.
- El o los archivos de entrada son divididos en "splits" y cada TaskTracker trabaja sobre un "split".

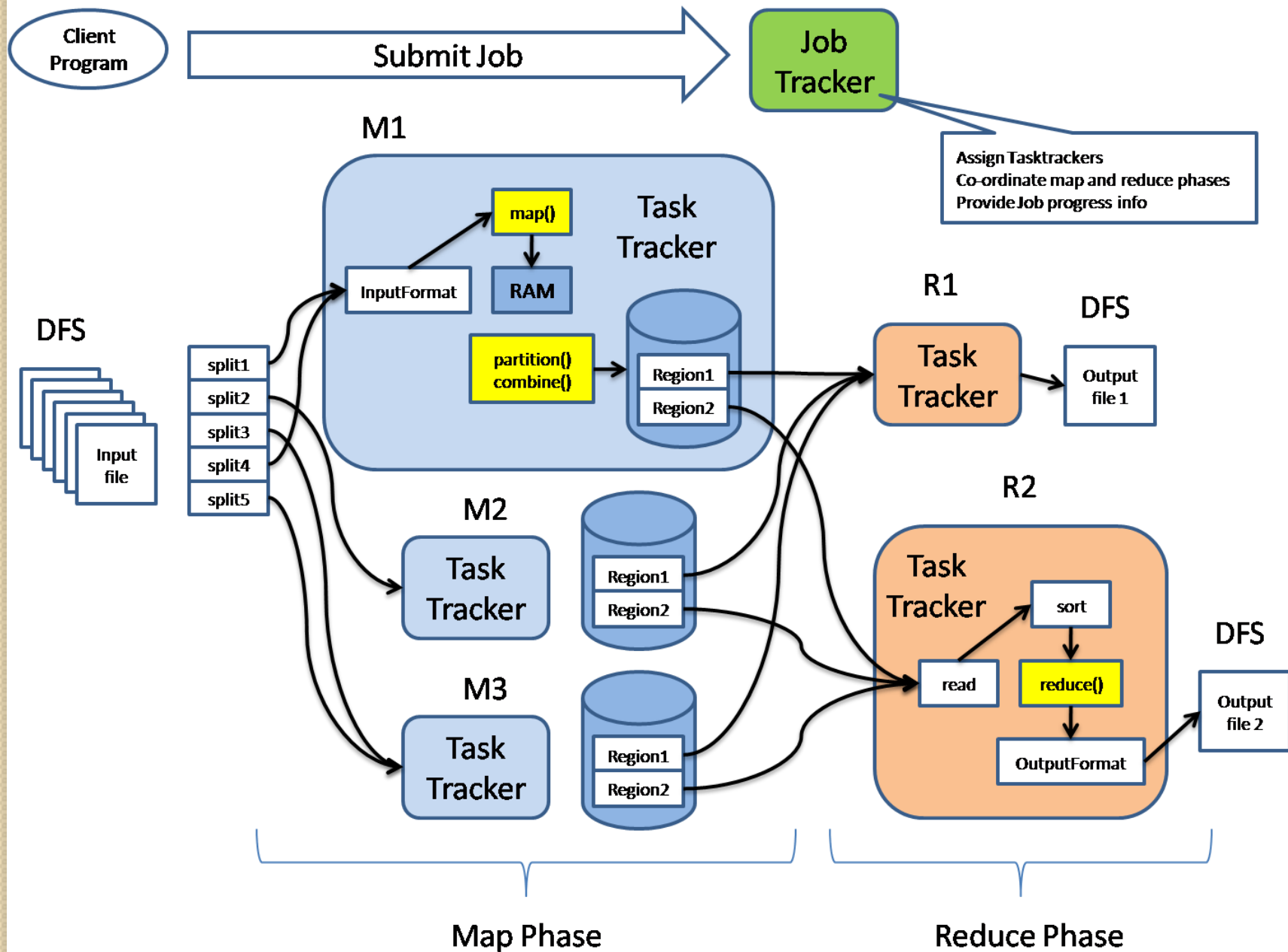
MapReduce

- La unidad de trabajo de MapReduce es un Job
- Un Job se divide en una tarea map y una tarea reduce.
- Los Jobs de MapReduce son controlados por un daemon conocido como JobTracker, el cual reside en el "nodo master"
- Los clientes envían Jobs MapReduce al JobTracker y este distribuye la tarea usando otros nodos del cluster
- Esos nodos se conocen como TaskTracker y son responsables de la ejecución de la tarea asignada y reportar el progreso al JobTracker

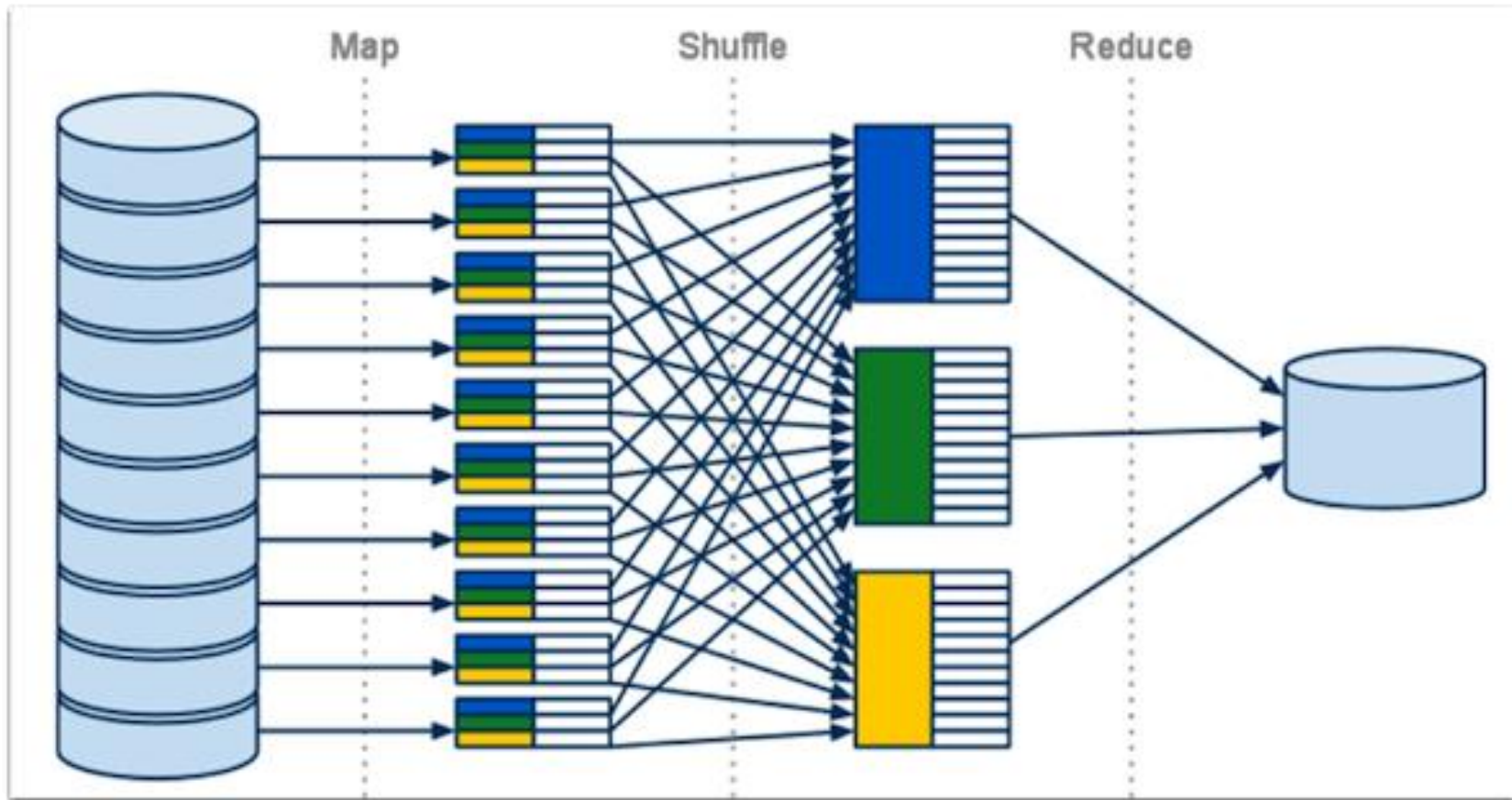


MapReduce

- Un job MapReduce es un proceso que se divide en cuatro fases:
 - Map -> Sort -> Shuffle -> Reduce
- Map y Reduce son las tareas que se deben programar para la aplicación.
- Cada TaskTracker ejecuta la tarea encomendada (map o reduce)
- Sort y Shuffle son internas en la ejecución del job.

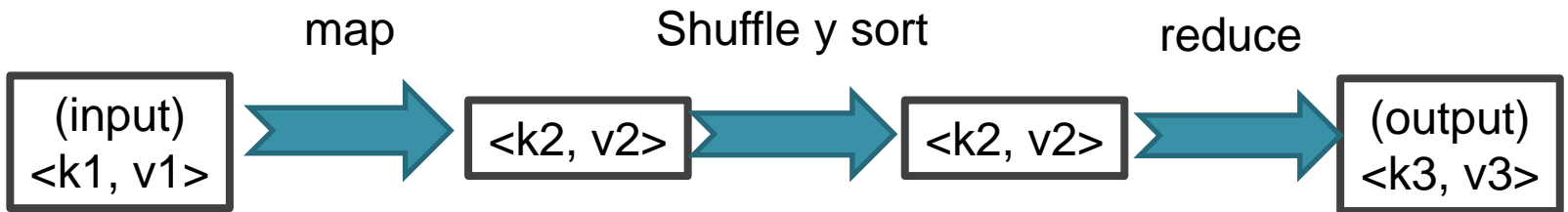


MapReduce



MapReduce

- Las tareas de map y reduce trabajan con el concepto de <clave, valor>



Tarea map

- Se ejecutan múltiples instancias de la tarea map sobre diferentes porciones del dataset
- Se intenta que cada map se ejecute sobre una copia local del dataset para minimizar el tráfico de datos
- El resultado de la tarea map debería ser una lista de pares $\langle k2, v2 \rangle$ donde $k2$ es un identificador utilizado para agrupar datos

Tarea map

- Una tarea map solo ve una porción del dataset de entrada.
- La tarea map lee los datos en forma de pares $\langle k1, v1 \rangle$ y produce una lista de cero o más pares $\langle k2, v2 \rangle$.
 - $\langle k1, v1 \rangle \rightarrow \text{list}(\langle k2, v2 \rangle)$
- Una tarea map debería analizar una tupla en forma independiente del resto de las tuplas.

Tarea map - Ejemplo

- Se posee un dataset con resultados de eventos. De cada evento se conoce su resultado (positivo, neutral, negativo).
- Se desea saber cuantos eventos positivos y negativos hay en todo el dataset.

Tarea map - Ejemplo

- El dataset es uno o más archivos de texto donde en cada línea está el resultado del evento

```
...  
POSITIVO  
POSITIVO  
NEGATIVO  
NEUTRO  
POSITIVO  
NEUTRO  
NEGATIVO  
NEGATIVO  
NEUTRO  
POSITIVO  
NEGATIVO  
...
```

Tarea map - Ejemplo

- Nuestra entrada será del tipo: `<int, string>` donde el valor será el string "positivo", "negativo" y "neutro".
 - La "clave" en los casos de datasets de texto siempre es el offset del archivo donde se encuentra la línea "valor".
- Como nuestra intención es sumar todos los positivos y negativos la salida será del tipo: `<string, int>` donde la clave será "positivo" o "negativo" y el valor el número 1.

Tarea map - Ejemplo

Input	Output
<1, POSITIVO>	<POSITIVO, 1>
<2, POSITIVO>	<POSITIVO, 1>
<3, NEGATIVO>	<NEGATIVO, 1>
<4, NEUTRO>	<POSITIVO, 1>
<5, POSITIVO>	<NEGATIVO, 1>
<6, NEUTRO>	<NEGATIVO, 1>
<7, NEGATIVO>	<POSITIVO, 1>
<8, NEGATIVO>	<NEGATIVO, 1>
<9, NEUTRO>	
<10, POSITIVO>	
<11, NEGATIVO>	

Tarea map - Ejemplo

Input

<1, POSITIVO>

<2, POSITIVO>

<3, NEGATIVO>

<4, NEUTRO>

<5, POSITIVO>

<6, NEUTRO>

<7, NEGATIVO>

<8, NEGATIVO>

<9, NEUTRO>

<10, POSITIVO>

<11, NEGATIVO>

Output

<POSITIVO, 1>

<POSITIVO, 1>

<NEGATIVO, 1>

<POSITIVO, 1>

<NEGATIVO, 1>

<NEGATIVO, 1>

<POSITIVO, 1>

<NEGATIVO, 1>

Nodo1

Nodo2

Nodo3

Tarea reduce

- Finalizada las tareas intermedias *shuffle* y *sort* se ejecuta la tarea reduce
- La tarea reduce lee los datos en forma de pares $\langle k_2, \text{list}(v_2) \rangle$ y produce una lista de cero o más pares $\langle k_3, v_3 \rangle$.
 - $\langle k_2, \text{list}(v_2) \rangle \rightarrow \text{list}(\langle k_3, v_3 \rangle)$
- La tarea reduce tiene todos los elementos para poder realizar cualquier operación de "resumen".

Tarea reduce - Ejemplo

- Siguiendo con el ejemplo de los eventos cada *reducer* recibe todas las tuplas del tipo "positivo" o del tipo "negativo" en tuplas del tipo <string, int>
- Solo hay que contar cuantas ocurrencias existen.
- La salida es la contabilidad realizada: <string, int>

Tarea reduce – Ejemplo "positivo"

Input

Output

<POSITIVO, 1>

<POSITIVO, 11>

<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

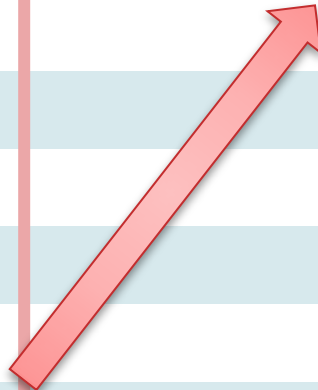
<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

<POSITIVO, 1>

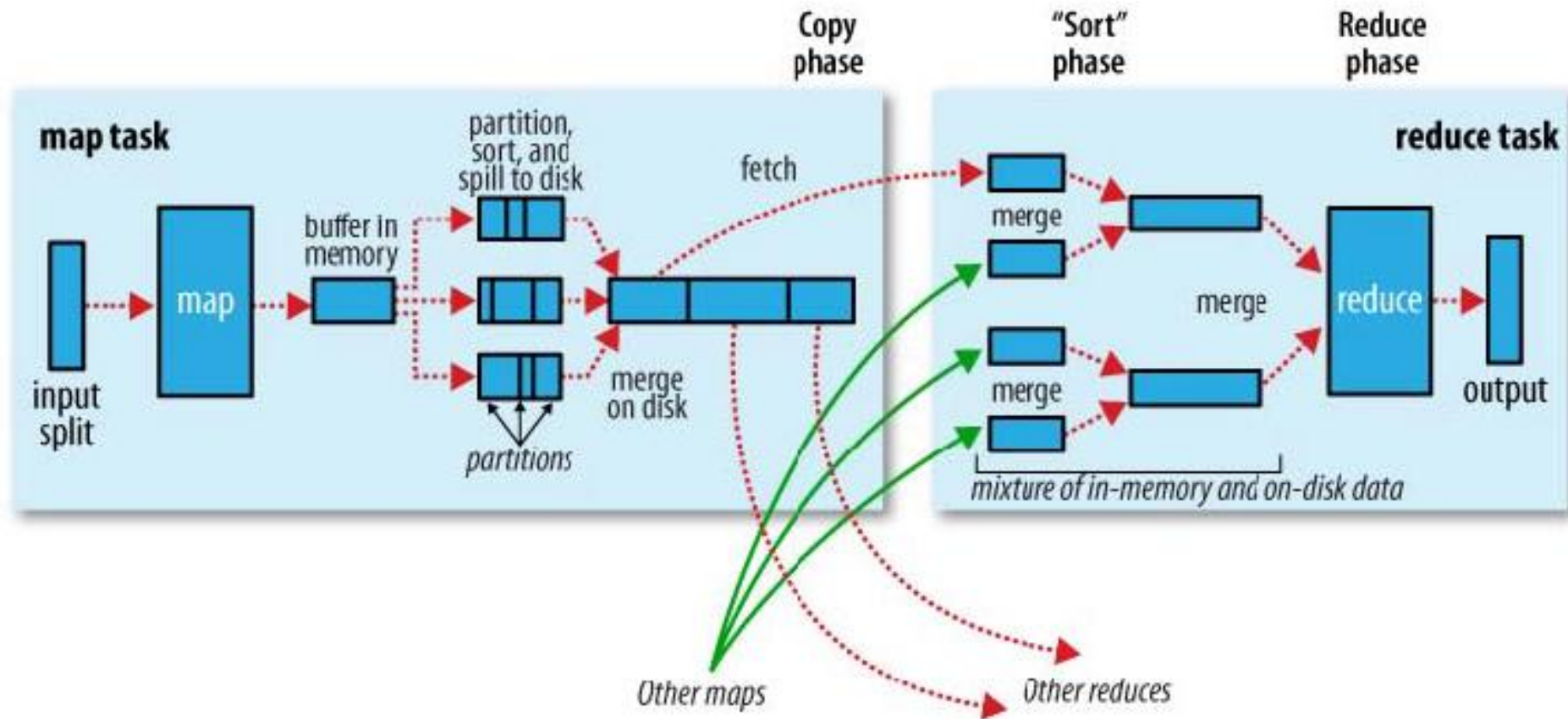
Nodo4



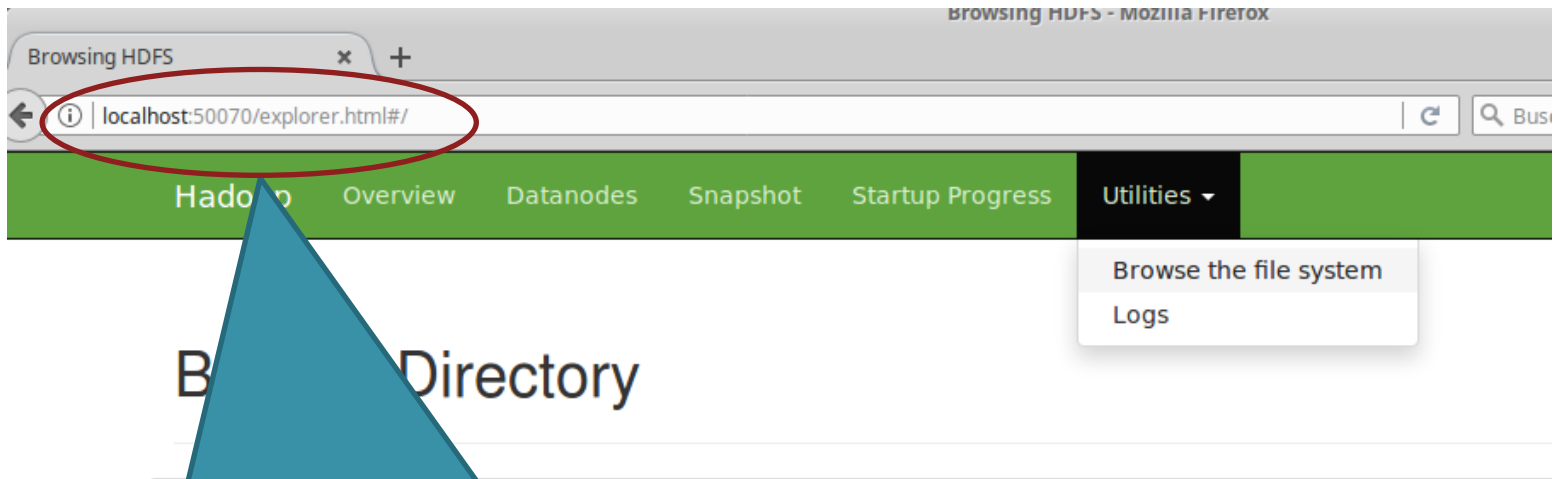
Tareas shuffle y sort

- *Shuffle* es la tarea que garantiza que toda la salida producida por los *mappers* sea reunida por clave.
- Asegura que todos los pares $\langle k2, v2 \rangle$ correspondientes a una misma clave sean recibidos por el mismo *reducer*, de esa manera un *reducer* recibe todas las tuplas correspondientes a una misma clave
- Un *reducer* puede recibir más de una de clave, en este caso la tarea *sort* se asegura que le sea enviada de manera ordenada por clave.

Tareas shuffle y sort



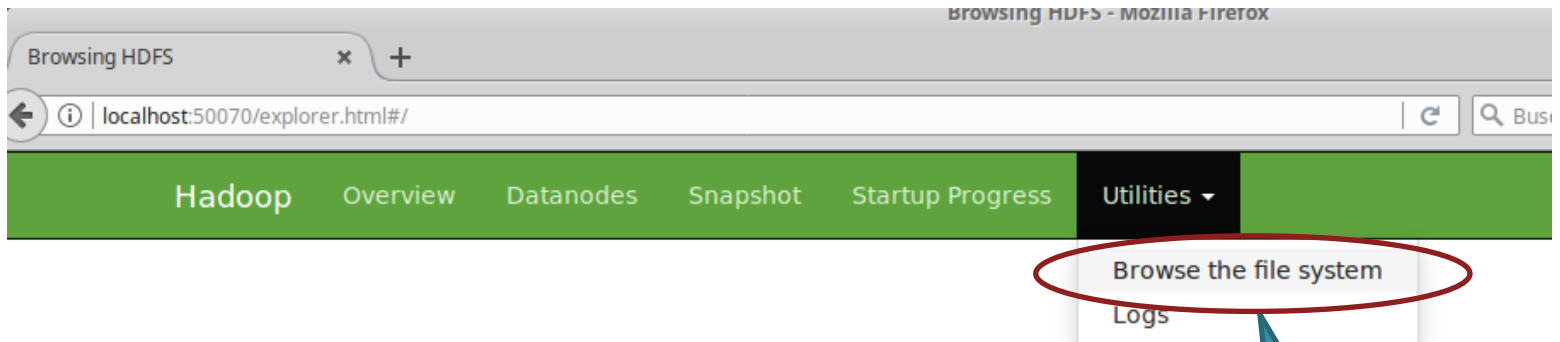
Browser Hadoop



En VirtualBox => <http://localhost:50070/>

En Docker => <http://localhost:9870/>

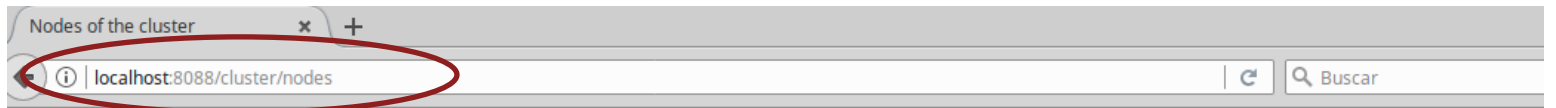
Browser Hadoop



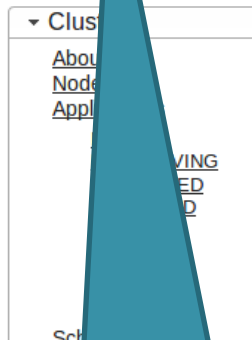
Browse Directory

Permite navegar por el HDFS

Browser Cluster Hadoop



Nodes of the cluster



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	8 GB	0 B	0	8	0

Show 20 entries

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report
	/default-rack	RUNNING	osboxes:43580	osboxes:8042	20-mar-2017 14:45:23	

Showing 1 to 1 of 1 entries

<http://localhost:8088>

EJERCICIO 1

- Compile y ejecute el proyecto WordCount dado por la cátedra. Utilice para ello el dataset Libros
- ¿Cuál es el top 20 de las palabras más usadas?

EJERCICIO 2

- El dataset Encuesta.rar tiene los resultados de diferentes encuestas realizadas a los usuarios de una empresa de telefonía móvil.
- Implemente una aplicación MapReduce para contabilizar cuantos clientes están "Muy satisfecho", "Algo satisfecho", "Poco satisfecho" y "Muy disconforme".