

Tema 7: Introducción a Hadoop

Máster Universitario en Ingeniería Informática
Extracción y Explotación de la Información

Índice

- ¿Qué es Hadoop?
 - Arquitectura Hadoop
 - NameNodes y DataNodes
 - JobTracker y TaskTracker
 - Ejecución de tareas
- Utilización de Hadoop

¿Qué es Hadoop?

- Hadoop es un framework de Apache <http://hadoop.apache.org/> para el procesamiento distribuido de grandes datasets en clusters de computación.
 - Grandes datasets → Terabytes o petabytes de datos
 - Grandes clusters → Cientos o miles de nodos
- Hadoop es una implementación open-source para Google MapReduce, que es un modelo de programación sencilla.
- Hadoop se basa en un modelo de datos simple, y permite trabajar cualquier tipo de dato.

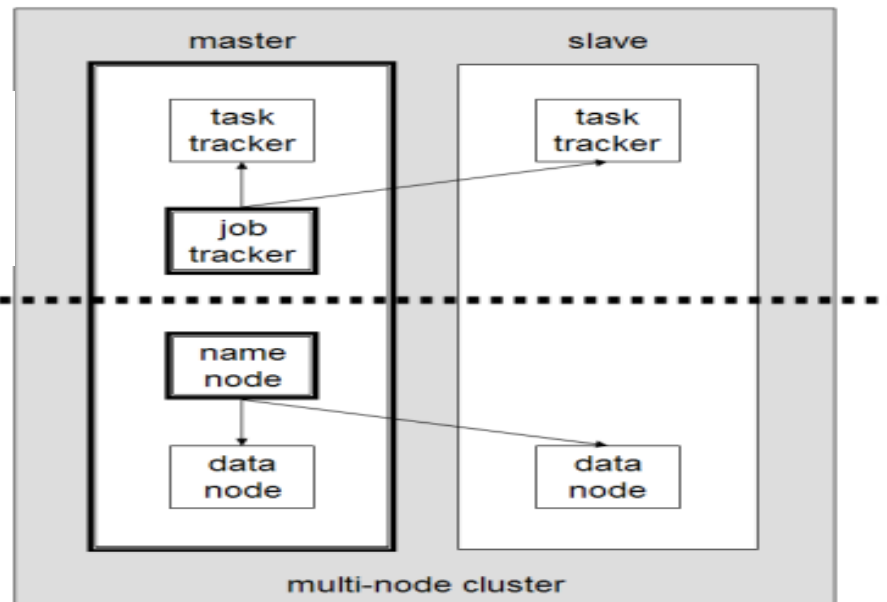


¿Qué es Hadoop?

- El framework Hadoop consiste en 2 capas principales
 - Sistema distribuido de ficheros (HDFS)
 - Motor de ejecución (MapReduce)

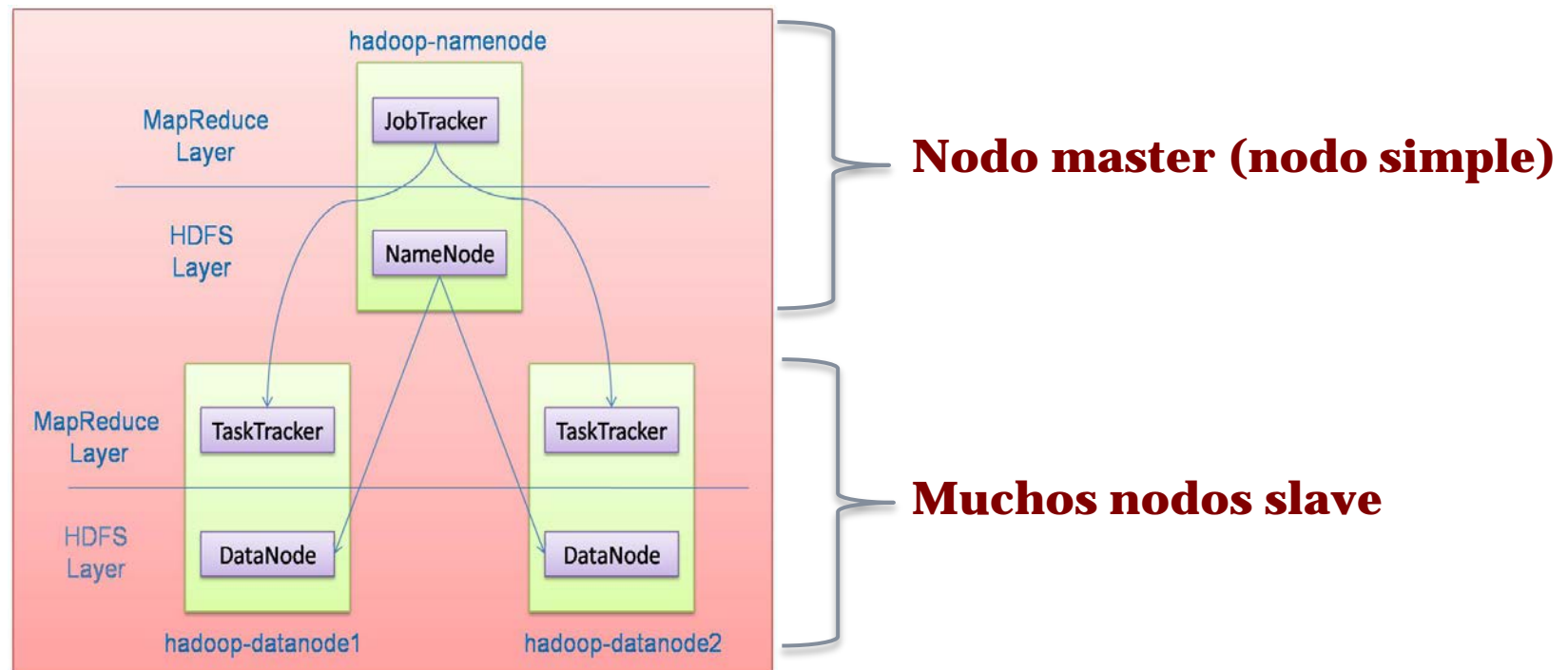
Distributed Data Processing (capa MapReduce)

Distributed Data Storage (capa HDFS)



Arquitectura Hadoop maestro/esclavo

- Hadoop esta diseñado como una arquitectura master-slave.
- Es “shared-nothing” pues los nodos master/slave no comparten recursos de hardware (RAM, disco) entre ellos (podemos variar la escalabilidad añadiendo o quitando nodos al cluster).

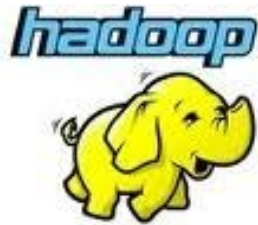


¿Por qué Hadoop?

- Necesidad de procesar big-data.
- Necesidad de paralelizar computación a lo largo de miles de nodos.
- Componentes de hardware (commodities):
 - Gran número de máquinas de bajo coste trabajando en paralelo para resolver un problema de computación.
- Esto contrasta con Bases de datos paralelas:
 - Pequeño número de servidores de alto coste.
- Principios de diseño:
 - **Paralelización y distribución automática:** Transparente para el usuario.
 - **Tolerancia a fallos y recuperación automática:** Los nodos/tareas que fallan recuperarán los datos de forma automática (replicación de datos).
 - **Abstracción de la programación limpia y simple:** Los usuarios sólo utilizan 2 funciones: “map” and “reduce”

Análisis de datos a gran escala

- El paradigma de computación map-reduce (Hadoop) es la alternativa a los sistemas de bases de datos tradicionales

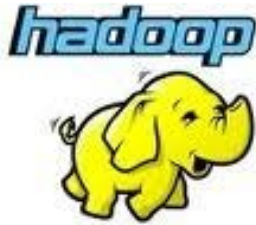


vs.



- Muchas empresas están optando por Hadoop
 - Especialmente por el uso de aplicaciones que generan big data (aplicaciones Web, redes sociales, ...)

Diferencias con bases de datos



Escalabilidad (petabytes de datos, miles de máquinas)



Flexibilidad en aceptar todos los formatos de datos (no esquema)



Mecanismo de tolerancia de fallos eficiente y simple



Hardware de bajo coste



Rendimiento (uso intensivo de indexación, ajuste, estructuración de datos)

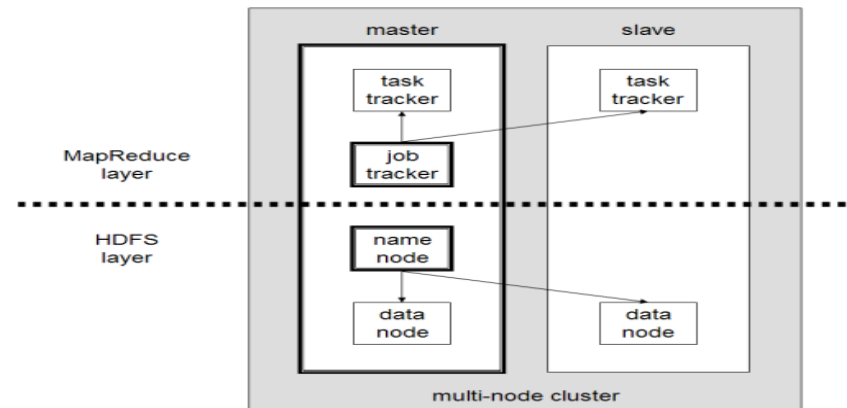


Aspectos:

- Tipos de datos, interrelaciones entre los datos, triggers, restricciones, etc.
- Gestión de metadatos
- Tecnología probada, consistente, y altamente soportada por compañías.

Arquitectura

- Un cluster hadoop se compone
 - NameNode
 - Secondary NameNode
 - DataNode
 - JobTracker
 - TaskTracker



Namenodes y Datanodes

- El NameNode es el master de HDFS
 - Se encarga de gestionar el espacio de nombres del sistema de ficheros.
 - Regula el acceso a los ficheros por parte de los clientes.
- Los DataNodes serían los slaves del cluster. Un sistema HDFS funcional tendría más de uno, con los datos replicados en todos ellos.
 - Un fichero de datos se divide en uno o varios bloques (de tamaño típico 64Mb), y los conjuntos de bloques son almacenados en los DataNodes.
 - Los DataNodes responden a las solicitudes que le hacen los NameNodes para realizar operaciones en el sistema de ficheros (lectura/escritura/creación de bloques/borrado/replicación).

Descripción del NameNode

- El NameNode, por tanto, se encarga de mantener el sistema de ficheros.
 - Controla todas las localizaciones de bloques y el mapping bloque-fichero.
 - En un cluster controla donde está cada fichero pero NO guarda ficheros en sí mismo.
 - Cualquier cambio en el sistema de ficheros es controlado por el NameNode.
- Las aplicaciones hacen consultas al NameNode para localizar un fichero o hacer alguna operación I/O (añadir/copiar/mover/borrar).
- Como el NameNode no almacena datos ni realiza tareas o trabajos de las aplicaciones cliente, se reduce la carga de trabajo.
- Sin el NameNode, el sistema no puede ser usado.
- Es un punto vulnerable en caso de fallo de la máquina (todos los ficheros de HDFS se perderían!!)

Descripción del NameNode

- Recomendaciones para uso en producción (<http://wiki.apache.org/hadoop/NameNode>)
 - Usar un buen servidor con bastante RAM (a más RAM, mayor es el sistema de ficheros).
 - Disponer de un SecondaryNameNode permite tener otra copia del directorio de ficheros y un simple fallo no corromperá los metadatos.
 - No tener otros elementos del cluster en el mismo sistema (servicios DataNode, JobTracker o TaskTracker).

Solicitudes de lectura y escritura

- Cuando un cliente envía una petición para CREAR un fichero, lo hace primero al NameNode, que determina cuantos bloques de datos son necesarios.
- Entonces, el NameNode determina el número de bloques y en qué Data Nodes deben ser escritos (los bloques y sus réplicas); se optimiza el tráfico de red para evitar hacerlo en DataNodes alejados al mismo tiempo que se asegura que no hay bloques de datos con su réplica en el mismo punto.
- Dicha información se le pasa al cliente, que se encarga de escribir los bloques de ficheros al DataNode más cercano.
- A continuación, cada DataNode se encarga de la copia de las réplicas de bloques al siguiente DataNode que corresponda.

Solicitudes de lectura y escritura

- El esquema sería similar al siguiente

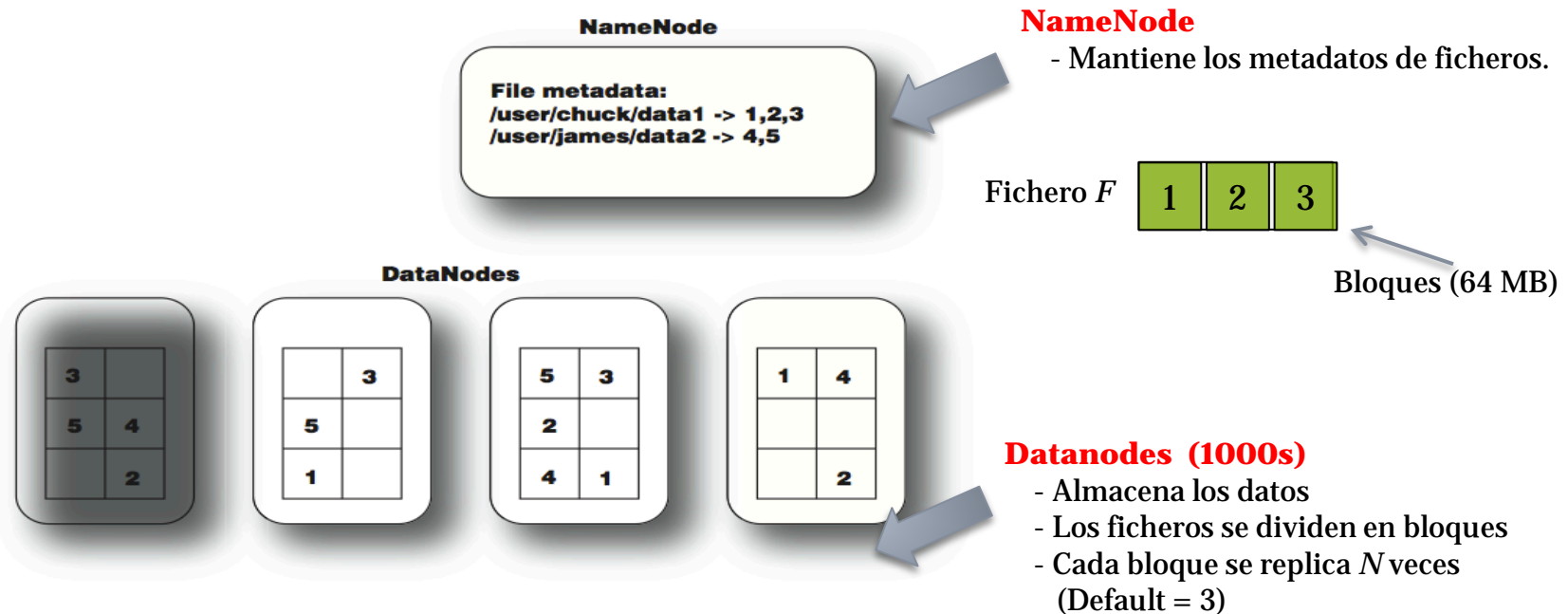


Figure 2.1 NameNode/DataNode interaction in HDFS. The NameNode keeps track of the file metadata—which files are in the system and how each file is broken down into blocks. The DataNodes provide backup store of the blocks and constantly report to the NameNode to keep the metadata current.

Solicitudes de lectura y escritura

- Cuando un cliente envía una petición para LEER un fichero, lo hace primero al NameNode, que determina si los datos están presentes y cuáles son los bloques que están involucrados.
- En este caso, el NameNode retorna al cliente la lista de DataNodes relevantes donde residen los datos solicitados.
- El cliente, con dicha información, accede a los bloques de datos en los DataNodes correspondientes.

JobTracker y TaskTracker

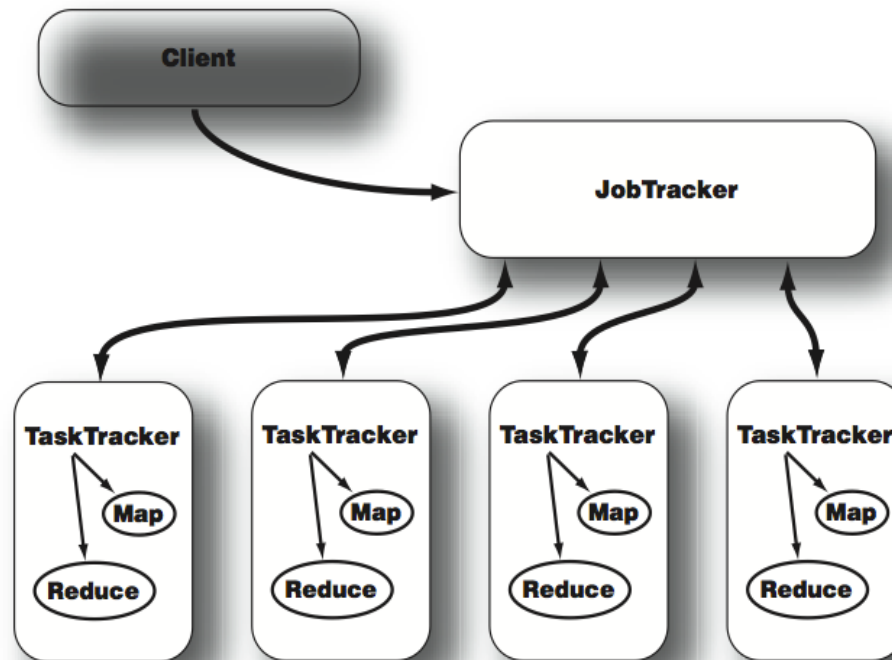
- El JobTracker es el servicio que se encarga de distribuir las tareas a los nodos específicos del cluster
 - Gestiona y coordina el procesamiento paralelo de datos usando la metodología MapReduce.
 - Si una tarea falla, se vuelve a lanzar automáticamente.
 - Las aplicaciones cliente submitten los trabajos al JobTracker.
- Los TaskTracker son los nodos del cluster que aceptan las tareas (operaciones map, reduce y shuffle) de un JobTracker.
 - Se encarga de que la tarea asignada al nodo se complete.
 - Los TaskTracker son slaves a nivel de tareas de computación, del mismo modo que los DataNodes son slaves a nivel de almacenamiento de datos.

Ejecución de tareas

- Las aplicaciones cliente submiten las tareas (jobs) al JobTracker.
- El JobTracker consulta al NameNode primero para saber la localización de los datos y luego localiza los nodos TaskTracker cercanos a estos datos (mismo host del DataNode de los datos, o el mismo rack, etc.).
- Cada TaskTracker es configurado con un conjunto de slots, que indican el número el número de tareas que pueden aceptar.
- Sólo puede haber un TaskTracker por nodo, pero puede lanzar múltiples procesos JVM para manejar distintas tareas en paralelo.
- El TaskTracker monitoriza estos procesos, y notifican al JobTracker el estado de las tareas de forma continua (generalmente cada pocos minutos) para asegurar que siguen en ejecución (heart beat).
- Mediante las notificaciones, el JobTracker sabe los slots que están disponibles a efectos de poder ir distribuyendo nuevas tareas.
- Si el JobTracker no recibe la notificación de una tarea en un TaskTracker, asume que ésta ha dado error y decide qué hacer (resubmite el trabajo en otro nodo, marcar el registro como algo a evitar o poner al TaskTracker en una blacklist como no fiable). En cualquier caso, se asegura que la tarea no fallará aunque lo haga un nodo del cluster.

Ejecución de tareas

- El esquema sería similar al siguiente



JobTracker and TaskTracker interaction. After a client calls the **JobTracker** to begin a data processing job, the **JobTracker** partitions the work and assigns different map and reduce tasks to each **TaskTracker** in the cluster.

Los clientes

- No forman parte del cluster.
- Se utilizan para cargar los datos en el cluster.
- Mediante ellos se submiten las tareas al cluster.
- HDFS provee una JAVA API para usar en las aplicaciones.
- Esta API se puede utilizar en Python, C, etc...
- Un navegador HTTP se puede utilizar para navegar por los ficheros de un sistema HDFS.

Más información...

- Una explicación más detallada se puede encontrar en:
 - <http://www.rohitmenon.com/index.php/introducing-hadoop-part-i/>
 - <http://www.rohitmenon.com/index.php/introducing-hadoop-part-ii/>

Índice

- ✓ ¿Qué es Hadoop?
- Utilización de Hadoop
 - Instalación de Hadoop
 - El shell de HDFS
 - La metodología MapReduce
 - Ejemplo

Instalación de Hadoop

- La página del proyecto Apache Hadoop contiene información sobre la instalación de Hadoop en un sistema Linux.

<http://hadoop.apache.org/index.html>

- Sin embargo, hay múltiples webs que describen el proceso de instalación bajo diferentes SO

<https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-on-ubuntu-13-10>
<http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>

- La instalación (2.x) incluye los siguientes módulos:
 - Hadoop Common: Las utilidades comunes que soportan el resto de módulos.
 - Hadoop Distributed File System (HDFS): El sistema de ficheros distribuido.
 - Hadoop YARN: Un framework para planificación de tareas y gestión de recursos del cluster.
 - Hadoop MapReduce: El sistema basado en YARN para procesamiento distribuido de datos masivos.

Archivos de configuración


- Hay varios archivos de configuración que comentamos a continuación.
- `/conf/*-site.xml`

`conf/core-site.xml`

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>
```

Establece el host y puerto del NameNode

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system. A URI whose
  scheme and authority determine the FileSystem implementation. The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class. The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>
```



Archivos de configuración

- Hay varios archivos de configuración que comentamos a continuación.
- `/conf/*-site.xml`

conf/mapred-site.xml

Establece el host y puerto del JobTracker

```
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
    at.  If "local", then jobs are run in-process as a single map
    and reduce task.
  </description>
</property>
```


Archivos de configuración

- Hay varios archivos de configuración que comentamos a continuación.
- `/conf/*-site.xml`

`conf/hdfs-site.xml`

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
    The actual number of replications can be specified when the file is
    created.
    The default is used if replication is not specified in create time.
  </description>
</property>
```

Cluster nodo simple

- El arranque de los procesos asociados a HDFS, NameNode y DataNodes se hace mediante el comando

```
bigdata@ubuntu:~$ start-dfs.sh
```

- El arranque del servicio MapReduce (2.x) YARN es

```
bigdata@ubuntu:~$ start-yarn.sh
```

- De forma opcional, el servicio de histórico de jobs se arranca

```
bigdata@ubuntu:~$ mr-jobhistory-daemon.sh start historyserver
```

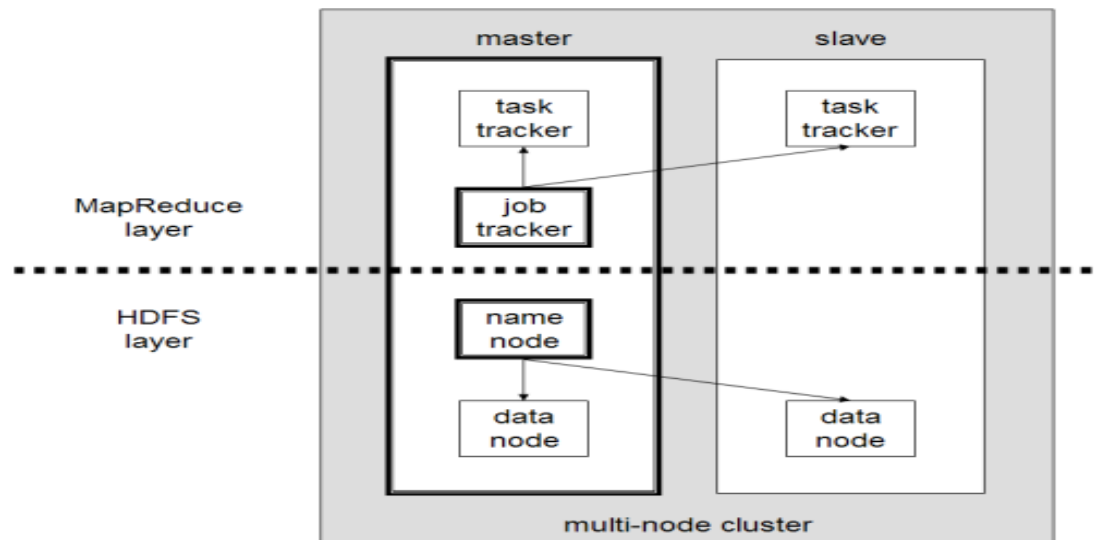
- La ejecución del comando “jps” permite ver los servicios arrancados.
- La sustitución de “start” por “stop” en los comandos anteriores realiza las paradas de los servicios.

Web interfaces de user (UI) de Hadoop

- El arranque de los procesos asociados a HDFS, NameNode y DataNodes se hace mediante el comando
 - <http://localhost:50070/> - web UI de NameNode
 - <http://localhost:50090/> - web UI de SecondaryNameNode
 - <http://localhost:8088/> - web UI de YARN Resource Manager
 - <http://localhost:19888/jobhistory> - web UI of job history

Cluster multi-nodo

- El nodo maestro hace correr los “master” daemons NameNode de la capa HDFS y JobTracker de la capa MapReduce.
- En los nodos esclavos (el master puede serlo también) corren los “slave” daemons DataNode (HDFS) y TaskTracker (MapReduce).



- En el nodo “master” hay que configurar los archivos `conf/masters` y `conf/slaves` especificando los hostnames.

El shell de HDFS

- El intérprete de comandos de HDFS se utiliza para las operaciones de ficheros en el cluster.
- Se invoca como: `bin/hadoop dfs [args]`, donde `args` puede ser:
 - `[-ls <path>]`
 - `[-du <path>]`
 - `[-cp <src> <dst>]`
 - `[-mkdir <path>]`
 - `[-rm <path>]`
 - `[-put <localsrc> <dst>]`
 - `[-copyFromLocal <localsrc> <dst>]`
 - `[-moveFromLocal <localsrc> <dst>]`
 - `[-get [-crc] <src> <localdst>]`
 - `[-copyToLocal [-crc] <src> <localdst>]`
 - `[-moveToLocal [-crc] <src> <localdst>]`
 - `[-cat <src>]`
 - `[-help [cmd]]`

Comandos básicos HDFS

- Veamos algunos ejemplos básicos

```
bigdata@ubuntu:~$ hadoop fs -ls
```

```
bigdata@ubuntu:~$ hadoop fs -mkdir /user/bigdata
```

```
bigdata@ubuntu:~$ hadoop fs -mkdir /user/bigdata/input
```

```
bigdata@ubuntu:~$ hadoop fs -put etc/hadoop /user/bigdata/input
```

```
bigdata@ubuntu:~$ hadoop fs -put /workspace/hadoop/readme.txt  
/user/bigdata/input
```

```
bigdata@ubuntu:~$ hadoop fs -ls /input
```

```
bigdata@ubuntu:~$ hadoop fs -rm -R /user/bigdata/output
```

Comandos básicos HDFS

- Veamos algunos ejemplos básicos

```
bigdata@ubuntu:~$ hadoop fs -cat /user/bigdata/input/readme.txt
```

```
bigdata@ubuntu:~$ hadoop fs -du README
```

```
bigdata@ubuntu:~$ hadoop fs -dus /user/bigdata
```

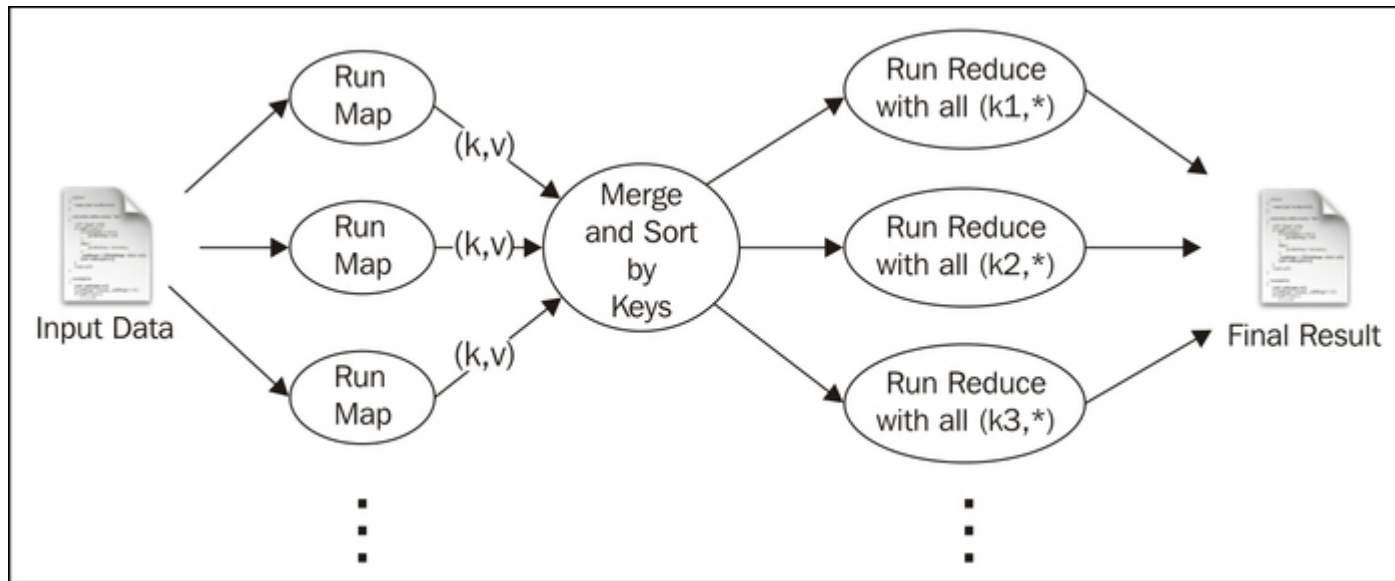
```
bigdata@ubuntu:~$ hadoop fs --help
```

La metodología MapReduce

- MapReduce es un modelo simple de programación que tiene su origen en Google.
- Para ejecutar una tarea MapReduce, se ha de proporcionar una función “map”, una función “reduce”, datos de entrada (“input”) y una localización para datos de salida (“output”).
- Cuando se ejecuta una tarea MapReduce en Hadoop se llevan a cabo los siguientes pasos:
 - Hadoop divide los datos “input” en múltiples items y ejecuta la función “map” sobre cada item. Esta ejecución retorna una o más parejas “clave-valor”.
 - Hadoop recoge todas las parejas clave-valor generadas por la función “map”, las ordena por “clave”, y agrupa todos los valores con la misma “clave”.
 - Para cada clave distinta, Hadoop ejecuta la función “reduce” con la lista de valores de dicha clave.
 - La función “reduce” retorna una o más parejas “clave-valor” y Hadoop las escribe en un archivo de salida.

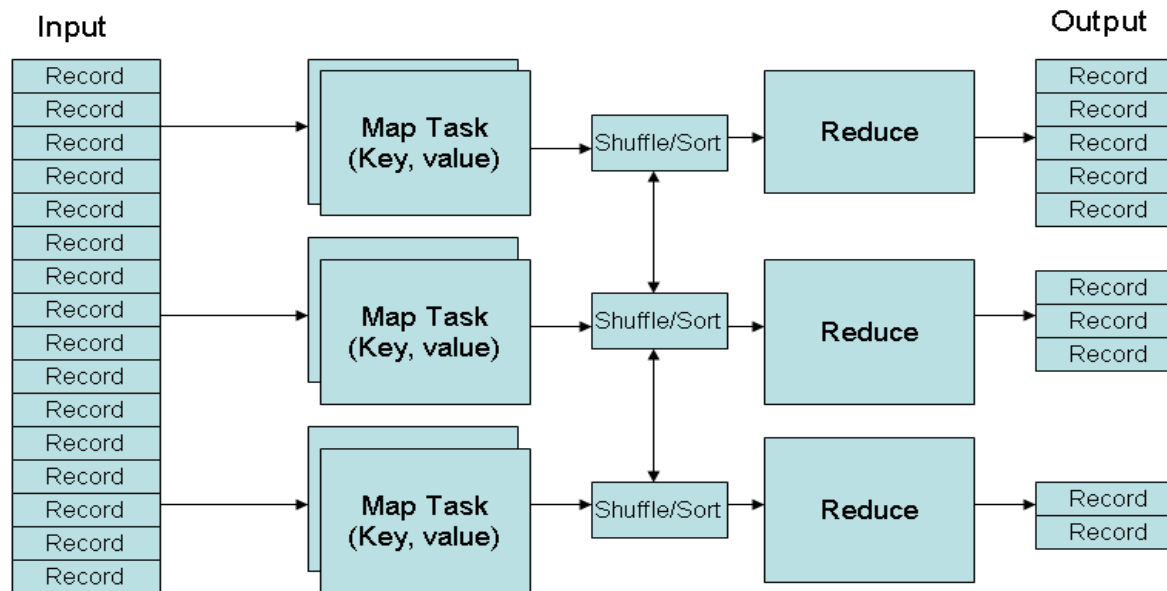
La metodología MapReduce

- El esquema sería similar a



La metodología MapReduce

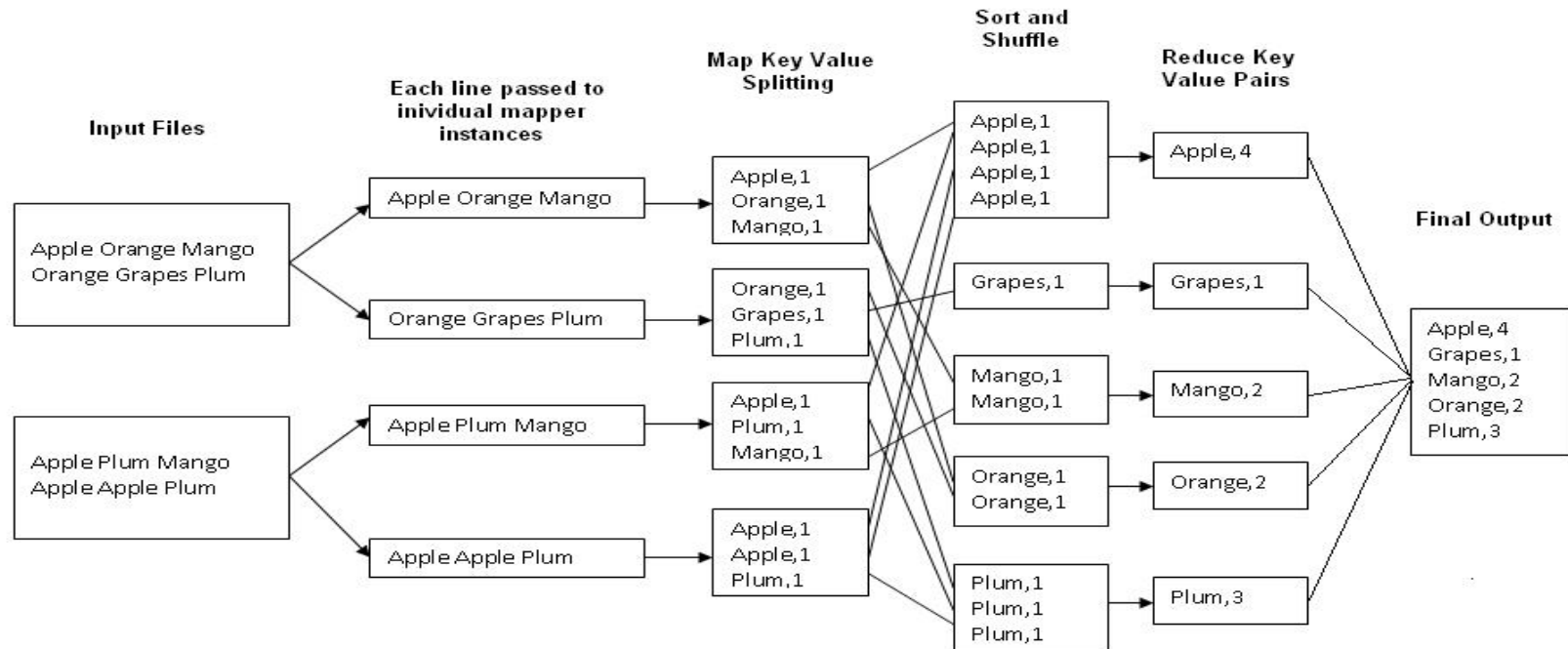
- De forma más detallada



- Map:
 - Acepta *input* clave-valor (registro por registro)
 - Retorna una pareja *intermedia* clave-valor
- Reduce :
 - Acepta una pareja *intermedia* clave-valor (grupo de registros)
 - Retorna *output* clave-valor

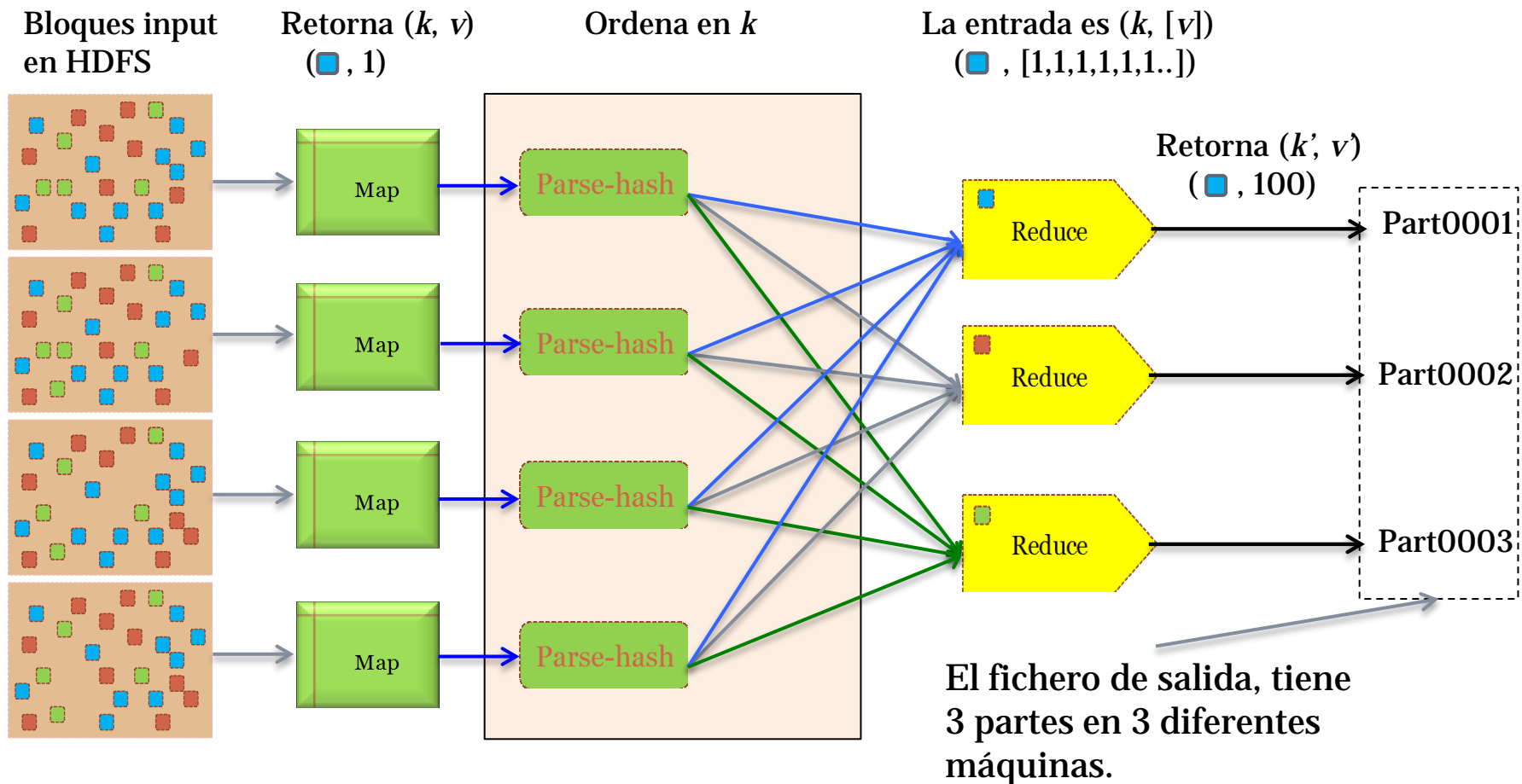
La metodología MapReduce

- Ejemplo clásico: Contar palabras



La metodología MapReduce

- Ejemplo: Frecuencia de colores en un data set



Ejemplo: Wordcount (java)

- El código Wordcount.java tiene 3 partes: mapper, reducer, y main.

mapper

```
import org.apache.hadoop.mapreduce.Mapper;

public static class Mapper {
    ... ..
    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, 1);
        }
    }
}
```

Ejemplo: Wordcount (java)

- El código Wordcount.java tiene 3 partes: mapper, reducer, y main.

reducer

```
public static class Reducer {  
    ...  
    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);  
    }  
}
```

Ejemplo: Wordcount (java)

- El código Wordcount.java tiene 3 partes: mapper, reducer, y main.

main

```
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf();
    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(Mapper.class);
    job.setReducerClass(Reducer.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);
}
```

Ejemplo: Wordcount (java)

- Para compilar y ejecutar:

```
bigdata@ubuntu:~$ cd workspace/Java_wordcount
```

```
bigdata@ubuntu:~$ rm -R bin/chapter1
```

```
bigdata@ubuntu:~$ javac -classpath
```

```
$HADOOP_HOME/share/hadoop/common/lib/hadoop-annotations-  
2.4.1.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-  
2.4.1.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-  
client-core-2.4.1.jar:$HADOOP_HOME/share/hadoop/common/lib/commons-  
cli-1.2.jar -d bin src/chapter1/WordCount.java
```

```
bigdata@ubuntu:~$ cd bin
```

```
bigdata@ubuntu:~$ jar cf chapter1.jar chapter1/*.class
```

```
bigdata@ubuntu:~$ hadoop jar chapter1.jar chapter1.WordCount input  
output
```


Ejemplo: Wordcount (python)

- En python tenemos 2 partes: mapper.py y reducer.py.

mapper.py

```
#!/usr/bin/env python
import sys

#--- get all lines from stdin ---
for line in sys.stdin:
    #--- remove leading and trailing whitespace---
    line = line.strip()

    #--- split the line into words ---
    words = line.split()

    #--- output tuples [word, 1] in tab-delimited format---
    for word in words:
        print '%s\t%s' % (word, "1")
```

Ejemplo: Wordcount (python)

reducer.py

```
#!/usr/bin/env python
import sys

# maps words to their counts
word2count = {}

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue

    try:
        word2count[word] = word2count[word]+count
    except:
        word2count[word] = count

# write the tuples to stdout (Note: they are unsorted)
for word in word2count.keys():
    print '%s\t%s'% ( word, word2count[word] )
```

Ejemplo: Wordcount (python)

- Para ejecutar:

```
bigdata@ubuntu:~$ cd workspace/Python_wordcount
bigdata@ubuntu:~$ hadoop jar /usr/local/hadoop-2.4.1/share/hadoop/tools/lib/hadoop-streaming-2.4.1.jar \
-file ./mapper.py -mapper ./mapper.py \
-file ./reducer.py -reducer ./reducer.py \
-input input4 -output output
```