Introducción Big Data

Presentaciones y logística

Instructores

Alumnos

- Experiencia con Hadoop
- Experiencia como programadores (lenguajes)
- Propio o por empresa
- Qué esperas de este curso

Logística del curso

Motivación Big Data

1. Problemas actuales de los sistemas de corporativos tradicionales
2. Requerimientos de una nueva aproximación

Problemas actuales de los sistemas de corporativos tradicionales

Tradicionalmente el hecho de computar información ha sido un importante Stopper, por lo que:

- Se trabajaba con pocos datos
- Los procesos eran complejos

Solución en aquella época:

- Comprar computadores con más potencia y capacidad
  - Core
  - Ram
  - HD
  - …
- Pero esto no solucionaba el problema
- Paradoja del buey y la rana

Problemas actuales de los sistemas de corporativos tradicionales

Una mejor solución: conectar varios computadores, pero:

- Conseguir que varios computadores trabajasen de manera sincronizada en el mismo job era extremadamente complicado
- Se perdía más tiempo en la gestión de errores que en solucionar el Caso de Uso
- El ancho de banda entre ordenadores es finito
- Fallos parciales hacen que haya que volver a empezar

Problemas actuales de los sistemas de corporativos tradicionales

Cuello de Botella

- Tradicionalmente los datos se almacenaban en sistemas centralizados potentes
- Los usuarios interactuaban con estos datos a través de terminales:
  - Problemas de rendimiento del mainframe (ordenador para procesar gran cantidad de datos)
  - Limitaciones de la terminal si se importaban los datos a ella
- Cada vez se generan más datos
- Introducción de las 3V
  - **Volumen** -> Los datos se reciben en grandes volúmenes.
  - **Velocidad** -> La información se genera y se debe almacenar a gran velocidad
  - **Variedad** -> Muchos los datos a almacenar y distintas fuentes y tipos.
  - *Veracidad y valor (añadimos)*

Requerimientos de una nueva aproximación

Es necesario otra forma de trabajar

→ Hadoop

Pero antes de esto, ¿qué es para vosotros Big Data?

Requerimientos de una nueva aproximación

Es necesario otra forma de trabajar

Hadoop (10 años) es un framework opensource para almacenar datos y ejecutar aplicaciones en clusters de hardware básicos. (En disco, Spark en memoria)

Pero antes de esto, qué es para vosotros Big Data?

- Turismo, salud, publicidad.
- (http://www.elmundo.es/papel/futuro/2017/11/27/5a1ab61322601dd03b8b4603.html)

Analisis twitter

El 1-0 con el referéndum catalán:

Nº de tweets → 11,8 millones +

Nº usuarios → 1,4 millones +

Hashtags:

#1Oct

#CatalanReferendum

#Catalonia

Genero:
39% mujeres
61% hombres

Idioma:
Castellano 56,6%
Catalán 24,9%
Resto 18,5%

Analisis datos

En 2013 su dueño tenía la intención de venderlo y comprar otros caballos.

Contrató a una pequeña empresa para analizar los caballos y quedarse con el mejor de ellos.

El mejor era el suyo, pero aún no lo sabía.

Dieciocho meses después, en 2015, **se convirtió en el primer caballo en más de tres décadas en ganar la Triple Corona.**

Midiendo el tamaño de los órganos, un gran corazón y bazo. (Comparó datos de otros caballos ganadores)

Hadoop

1. Introducción

2. HDFS

3. Map Reduce
    1. V1
    2. Yarn

4. Ecosistema

Hadoop

Historia

- Hadoop está inspirado en los documentos de Google para MapReduce y GFS (Google File System)
- Actualmente es un proyecto de alto nivel de Apache soportado por la comunidad global de contribuyentes.
- Fue creado por Dough Cutting, que lo nombró así por el elefante de juguete de su hijo
- Fue inicialmente desarrollado para apoyar la distribución del proyecto de un motor de búsquedas llamado Nutch
- Principales distribuciones:
  - Cloudera
  - Hortonworks
  - MapR

Hadoop

Justificación

- Cada vez se producen más datos en el mundo: más dispositivos conectados
  - Sensores, Logs, IoT, Transacciones, RRSS, Clickstream, etc…
- Y cada vez más rápidamente
  - Conectividad constante
  - Incremento de la automatización de las industrias
  - Seguimiento más granularizado del usuario

Hadoop

Valor de los datos

- Predicción
- Mejora de procesos (+completos, +potentes, +sencillos..)
- Segmentación → conocimiento 360º
- Ahorro de costes de almacenamiento
- Disminución del tiempo de procesado
- Ejecución de procesos antes imposibles

Hadoop

Lo más importante en Hadoop es:

- Llevar el procesamiento a los datos

También muy importante

- Distribuir los datos según llegan al Cluster
- Aplicaciones escritas en alto nivel
- Los nodos se comunican entre sí lo menos posible
- Los datos se replican para obtener
  - Disponibilidad
  - Fiabilidad
  - Tolerancia a fallos

Hadoop

Hadoop es

- Escalable horizontalmente y con coste reducido
  - + Datos
  - + Potencia
- Tolerante a fallos
  - Que un servidor falle es inevitable
  - Pero los sistemas deben seguir funcionando
  - De manera transparente al usuario
  - Sin perder datos
  - Debe recuperarse sin intervención del usuario
- Los jobs han de ejecutarse de manera independiente
- La salida de un job no debe condicionar la salida de otro job, salvo que sea necesario
- El rendimiento de un job se puede ver alterado por la ejecución de otro job dado que la potencia del Cluster es la que es, pero es algo que ya se tiene en cuenta.

Hadoop

Paradigma de Hadoop.

Ha de ser:

- Sencillo de aprender
- Fácil de manejar desde un punto de vista conceptual
- Sencillo de sacar partido: APIs para programadores novatos o gente de negocio

Hadoop

- Hadoop está programado en Java
- Originalmente se interactúa con él, programando en Java
- Aunque existe la posibilidad de hacerlo con otros lenguajes de programación o herramientas
- El programador solo se preocupa de codificar el código que resuelve el caso de uso, no se encarga de gestionar la coordinación, sincronización o los posibles fallos del sistema

Hadoop

Tipos de datos en Hadoop:

- Estructurados
- Semi-estructurados
- No estructurados

Tiempo de procesado:

- Originalmente procesado por lotes
- Near Real Time
- Real Time (Hbase-> db noSQL, Storm-> no tiene una arquitectura de origen y final, se procesa y se analiza continuamente)

Tipo de procesado:

- Ejecución en paralelo
- Sobre datos distribuidos (HDFS)

Nociones Básicas y HDFS

Hadoop se compone de

- HDFS
  - Almacena datos en el cluster
- MapReduce
  - Procesa datos en el cluster
- Ecosistema de Herramientas
  - Conjunto de herramientas que hacen más fácil trabajar con Hadoop

Nociones Básicas y HDFS

Cluster

- Un cluster se compone de un conjunto de servidores (nodos) que trabajan juntos para conseguir un objetivo común
- Hay dos grandes tipos de nodos que componen un cluster:
  - Maestros (gobernar el cluster)
  - Esclavos (procesar/almacenar la inf)
- Cada nodo tiene sus Demonios corriendo, dependiendo del tipo que sea
  - Detalle más adelante

HDFS

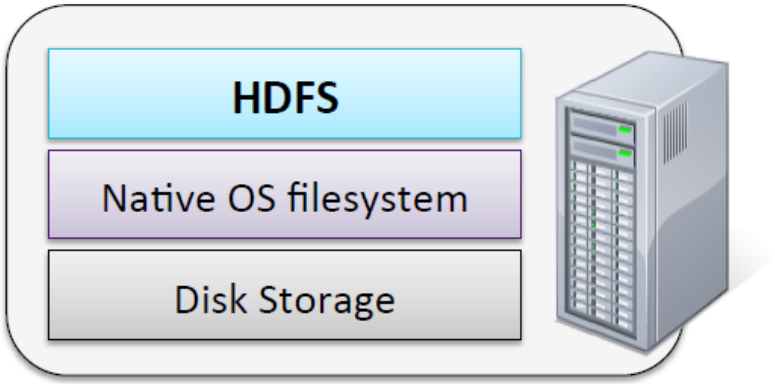
HDFS es un sistema de archivos nativo de Google escrito en Java

Está desplegado sobre el sistema de archivos nativo de cada servidor

- Típicamente Línux: EXT3, EXT4 o XFS

Su característica más importante es que provee de almacenamiento redundante de grandes volúmenes de datos transparente al usuario

- El usuario no tiene que indicar que quiere replicar la información almacenada



HDFS

HDFS es óptimo cuando trabaja con un cantidad moderada de archivos de gran tamaño

- Millones mejor que Billones
- Cada archivo de 100 MB o más
- Disco óptico

HDFS está pensado para escribir una vez y leer muchas

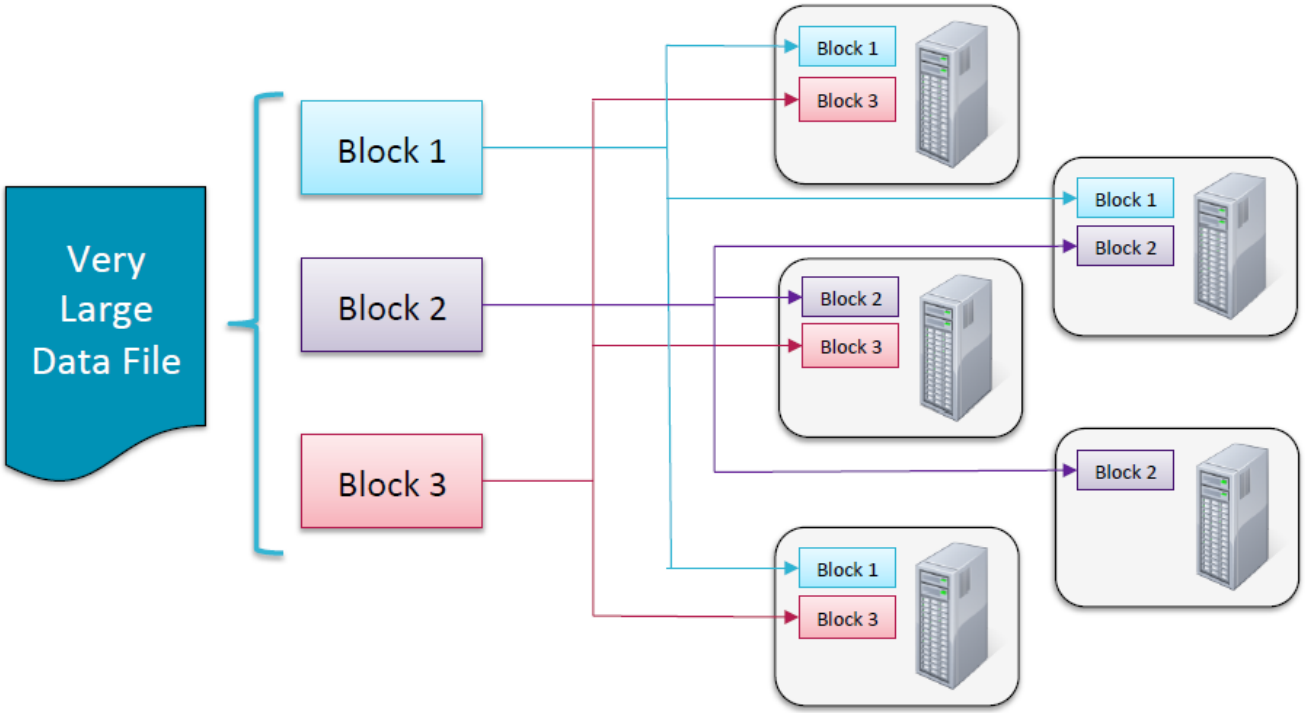
- Se pueden modificar archivos escritos, pero es extremadamente costoso, por lo que se prefiere eliminar el archivo y copiarlo nuevamente actualizado.

HDFS está pensado para leer óptimamente archivos de gran tamaño, no para lecturas aleatorias

HDFS

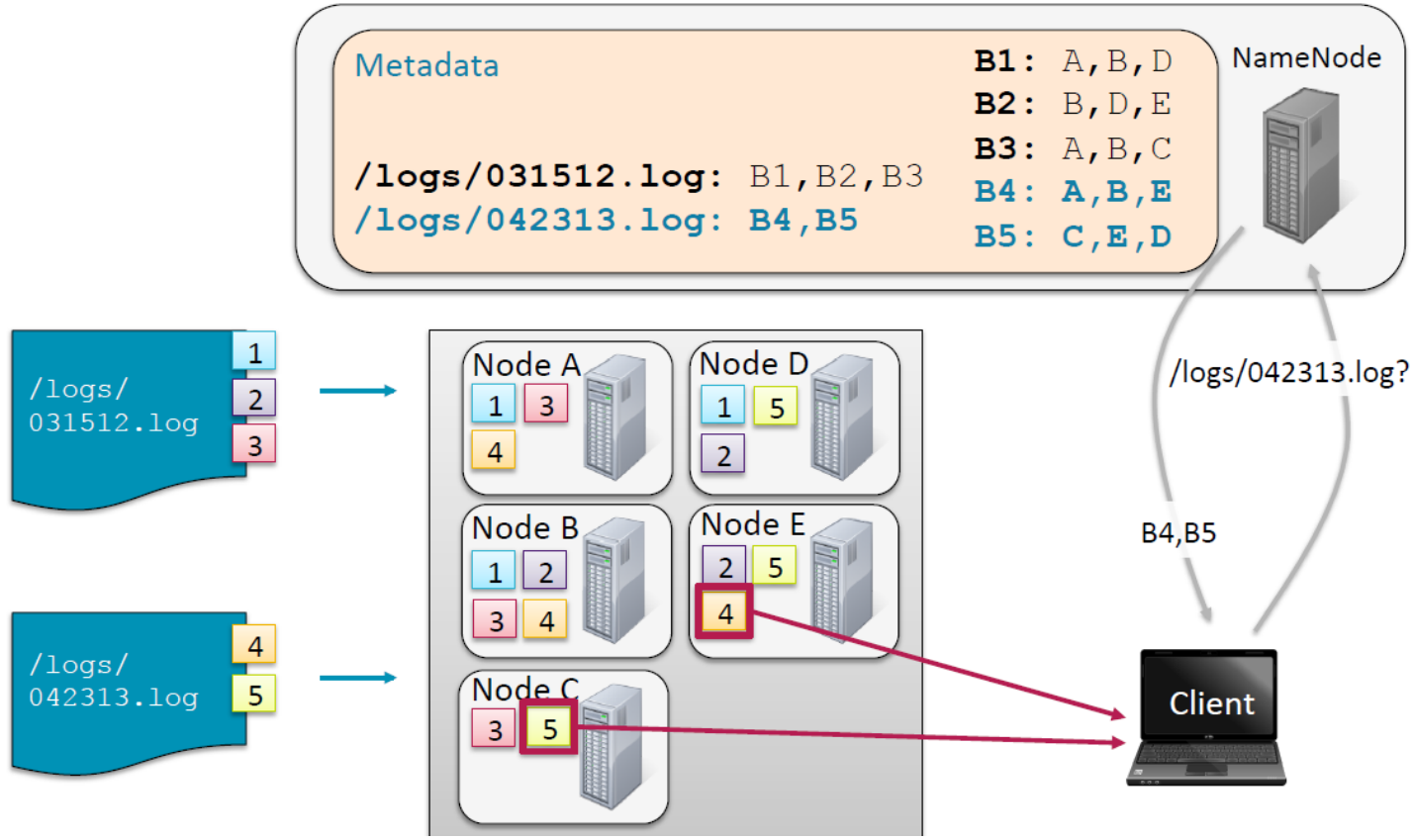
Funcionamiento 1

- Cada archivo es dividido en bloques y distribuido entre los nodos del cluster en tiempo de escritura
- Cada archivo se replica 3 veces, por defecto, en el cluster, aunque este número es modificable



HDFS

Funcionamiento 2



HDFS

Demonios de HDFS

- Nodo Maestro. Sus demonios son
  - NameNode
    - Se encarga del almacenamiento y gestión de los metadatos
    - Información de permisos y usuarios
    - Qué bloques componen un archivo
    - Dónde está cada bloque que compone un archivo
    - Los metadatos son guardados en disco y cargados en memoria cuando el cluster arranca en un fichero llamado *fsimage*
    - Los cambios realizados sobre los metadatos son almacenados en un fichero llamado *edits* en memoria
  - Secondary Namenode
    - Se encarga de realizar las labores de mantenimiento (*edits - fsimage*)
  - Standby Namenode
    - Para arquitecturas con Alta Disponibilidad
    - Se encarga de realizar las labores de mantenimiento (*edits - fsimage*)
    - Automáticamente *toma el relevo*

HDFS

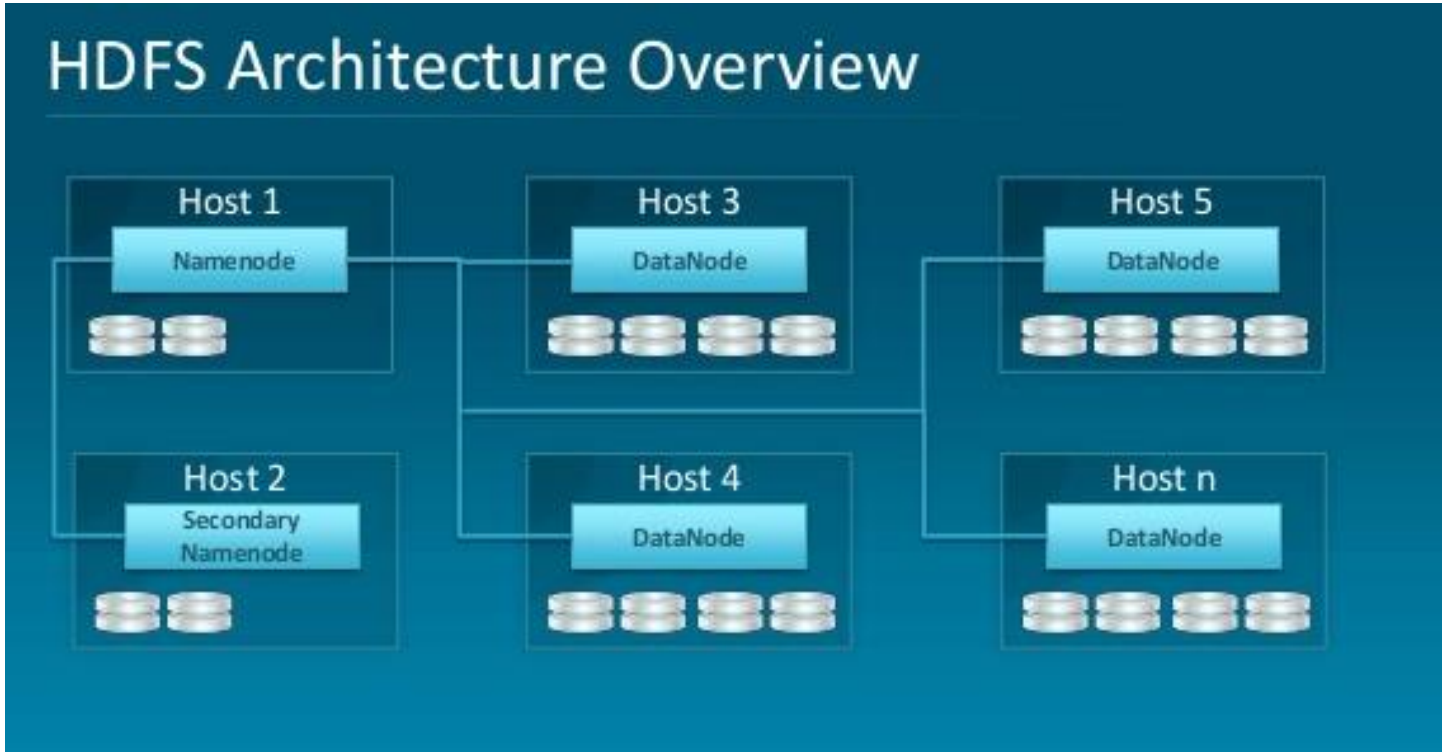
Demonios de HDFS

- Nodos esclavo. Su demonio es
  - DataNode
    - Los nodos que gobierna almacenan los bloques de datos, no información referente al bloque en sí.
    - Es el encargado de acceder a dichos bloques (blk_xxxxxxx)
    - Cada bloque se almacena varias veces, dependiendo del factor de replicación
    - Cuando un fichero entra en el cluster y se divide en bloques, cada nodo esclavo se encarga de hacer una copia al siguiente nodo esclavo hasta llegar al factor de replicación
    - Se encargan de gestionar las tasks que componen un Job

Checkpoint del Secondary Namenode

- Cada hora o cada 1M de transacciones, el SNN realiza el checkpoint de los metadatos:
  - 1. Llama al NameNode para obtener el fichero edits
  - 2. Obtiene los ficheros fsimage y edits del NameNode
  - 3. Carga el fichero fsimage en memory y agrega los cambios del fichero edits
  - 4. Crea el nuevo fichero consolidado de fsimage
  - 5. Envía el nuevo fichero consolidado al NameNode
  - 6. El NameNode reemplaza el antiguo fsimage por el nuevo

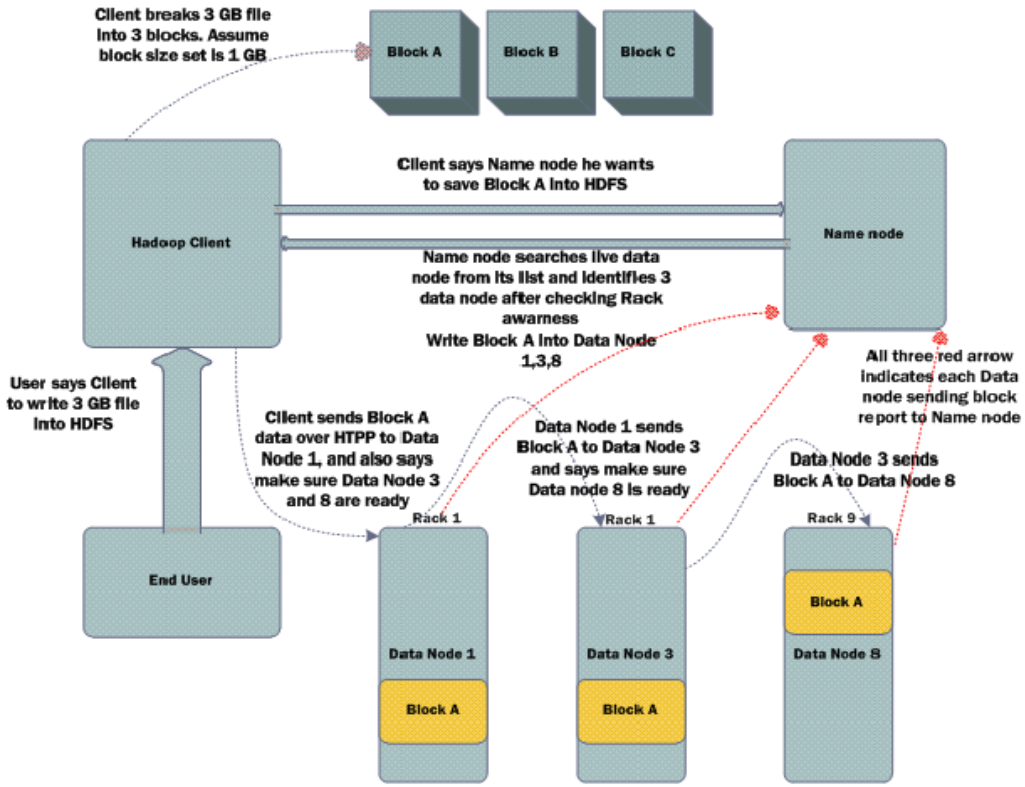
HDFS


HDFS Architecture Overview
Host 1
Namenode
Host 3
DataNode
Host 5
DataNode
Host 2
Secondary Namenode
Host 4
DataNode
Host n
DataNode

HDFS

Procedimiento de escritura en HDFS

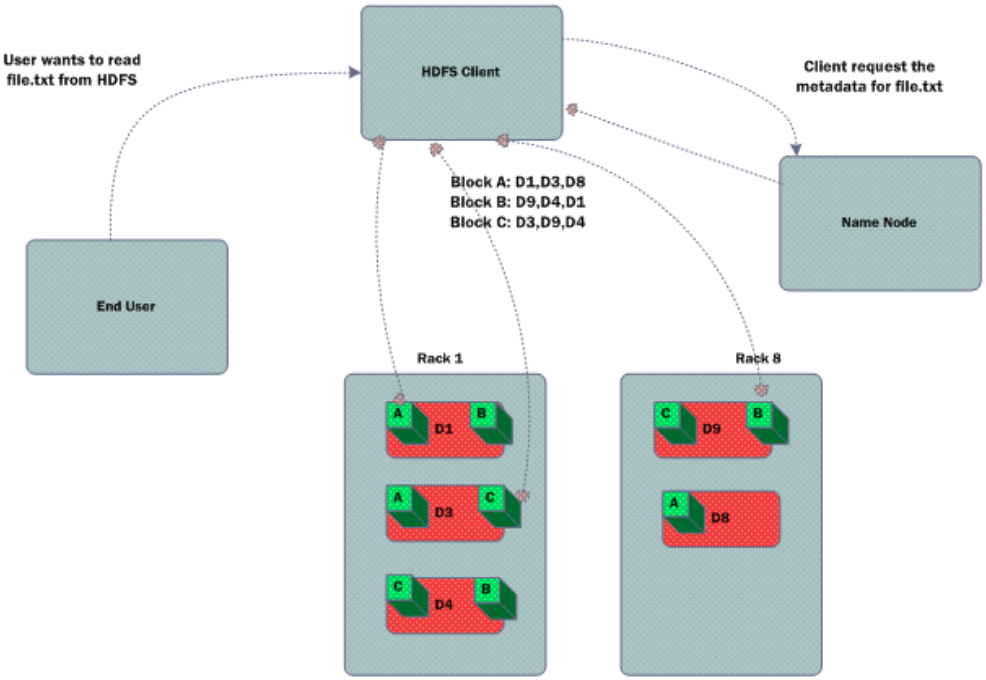
1. Cliente **conecta** con el NameNode
2. El NameNode busca sus metadatos y **devuelve el nombre del bloque y la lista de los DataNode**.
3. El Cliente conecta con el primer DataNode de la lista y **empieza el envío de los datos**
4. Se conecta con el segundo DataNode para realizar envío y lo mismo con el tercero.
5. Finaliza el envío
6. El cliente indica al NameNode donde ha realizado la escritura



HDFS

Procedimiento de lectura en HDFS

1. Cliente **conecta** con el NameNode
2. El NameNode devuelve una **lista con los DataNode** que contienen ese bloque
3. El cliente conecta con el primer ode los DataNode y comienza la lectura del bloque



HDFS

Fiabilidad y recuperación de datos en HDFS

- Los DataNode envían heartbeats al NameNode cada 3 segundos para indicar su estado
- Si pasado un tiempo (por defecto 5 minutos) el NameNode no recibe el heartbeat de alguno de los nodos, da por perdido ese nodo
  - El NameNode averigua que bloques había en el nodo perdido
  - El NameNode busca otros DataNode donde realizar la copia de los bloques perdidos y así mantener el número de replicación. Los DataNode serán los encargados de realizarse la copia de los bloques "perdidos"
- En caso de recuperar el nodo perdido, el sistema automáticamente decidirá que bloques eliminar para mantener el número de replicación de bloques.
- Speculative Execution:
  - Si el master detecta que un nodo esclavo está ejecutando las tasks más lento de lo normal, ordena lanzar los trabajos de ese nodo esclavo en otro nodo. El primer nodo que termina entrega el resultado. Los procesos que quedan en el otro nodo, e más lento, se matan.

HDFS

WEB UI

- A través de la web UI del NameNode seremos capaces de ver el estado de nuestro cluster:
  - Espacio total del HDFS
  - Espacio ocupado del HDFS
  - Espacio disponible del HDFS
  - Estado de los Nodos
  - Navegación por el sistema de ficheros
  - ……
- Por defecto se encuentra en el puerto 50070

HDFS

Acceso a datos en HDFS a través de la línea de comandos

- El sistema de archivos es muy similar al de Linux y sus comandos también:
  - Copiar un fichero de disco local a HDFS
    - $ hadoop fs -put /…/foo.txt /…/foo.txt
  - Listar el directorio Home del usuario
    - $ hadoop fs -ls
  - Listar el directorio root del usuario
    - $ hadoop fs –ls /
  - Mostrar el contenido de un fichero, por ej: /user/fred/bar.txt
    - $ hadoop fs -cat /user/fred/bar.txt
  - Copiar un fichero de HDFS al local, por ejemplo baz.txt
    - $ hadoop fs -get /user/fred/bar.txt baz.txt
  - Crear un directorio en el home de HDFS del usuario llamado input
    - $ hadoop fs -mkdir input
  - Borrar un directorio y todo su contenido
    - $ hadoop fs -rm -r input_old

HDFS

Ejercicios

1. Descubriendo HDFS
   - Ver documento de ejercicios adjunto.

MapReduce

Es un paradigma de programación que se ejecuta en tres etapas

- Fase Map
- Fase Shuffle&Sort
- Fase Reduce

También es un método para distribuir tareas a lo largo del cluster

Automáticamente las paraleliza y distribuye

Es tolerante a fallos

Está pensado para que sea una abstracción para los programadores

- Los programas se escriben típicamente en Java, como Hadoop
- A través de *Hadoop Streaming* se puede programar en otros lenguajes de programación
- El programador solo programa, no se encarga de NADA más

Un MapReduce típico se asemeja a la concatenación del siguiente conjunto de comandos:

- cat /ficheros/ | grep '\.html' | sort | uniq –c > /salida.txt

Uniq: mostrar solo 1 ocasión a cada palabra

grep: seleccionar lo que necesitas

MapReduce

Conceptos fundamentales

- El programador solo tiene que definir
  - El Mapper
  - El Reducer
  - El Driver

Mapper

- Actúa sobre cada registro de entrada
- Cada tarea, Task, opera sobre un único bloque en HDFS, siempre que sea posible
- Cada Task opera en el nodo donde el bloque está almacenado
- La salida del Map es un par (Clave/Valor)

Suffle&Sort

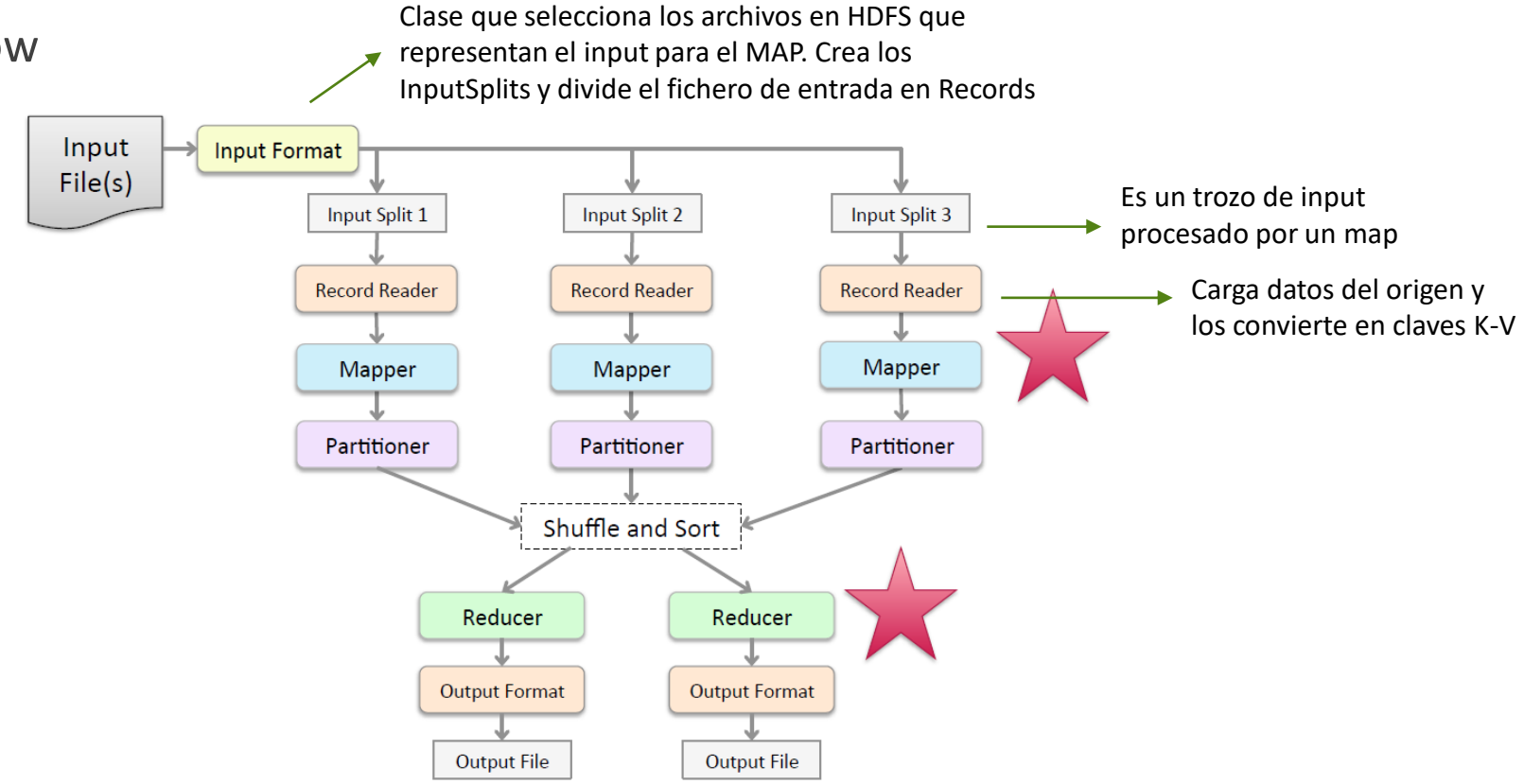
- Ordena por clave y agrupa todos los datos intermedios de todos los Mappers de una misma clave en el formato (K, [V1…Vn])
- Ocurre justo después de que todos los Mappers hayan terminado y justo antes de que la fase Reduce comience

Reducer

- Recibe la salida del S&S y realiza operaciones sobre ella
- Produce la salida final

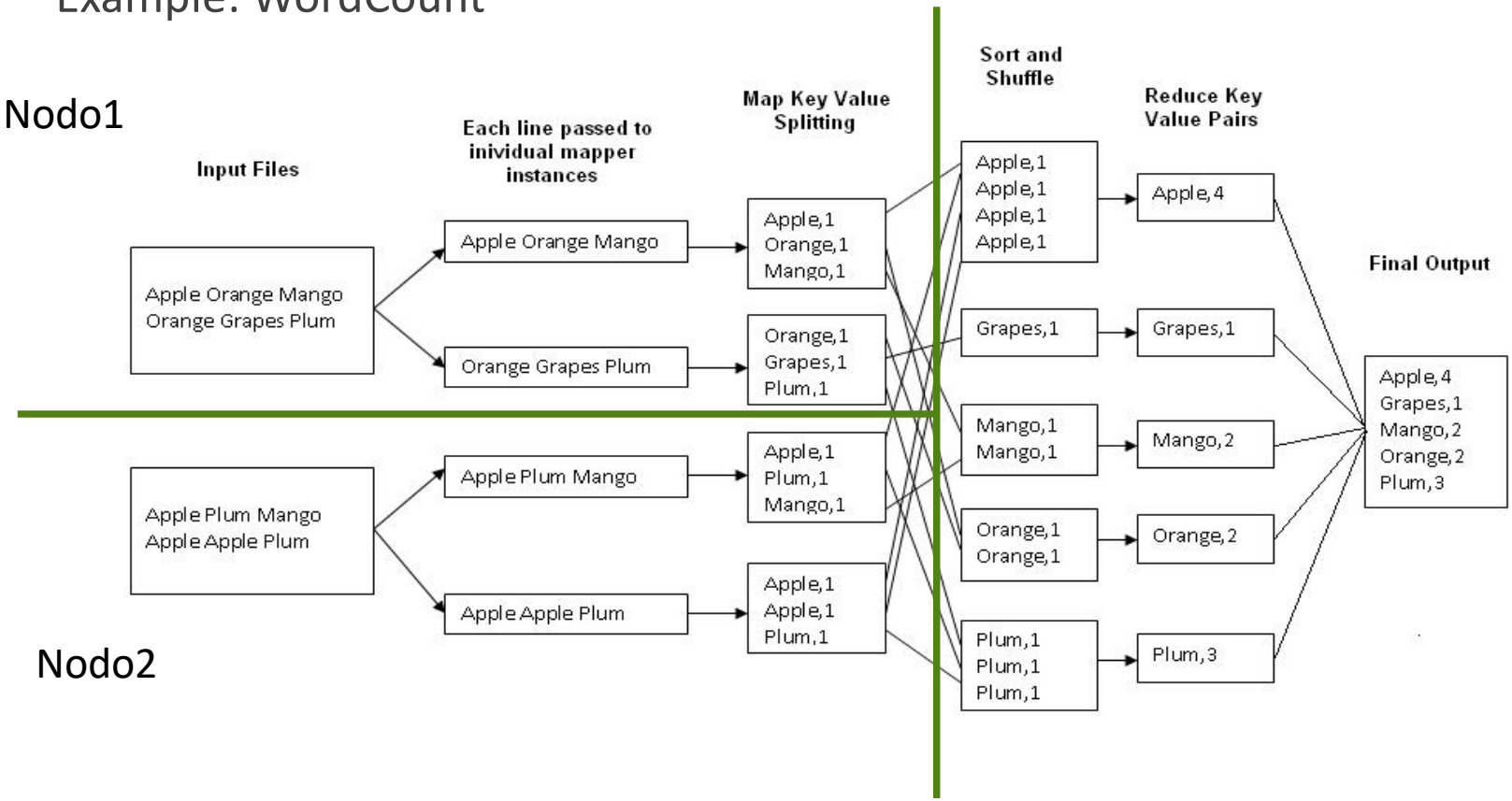
MapReduce

Workflow



MapReduce

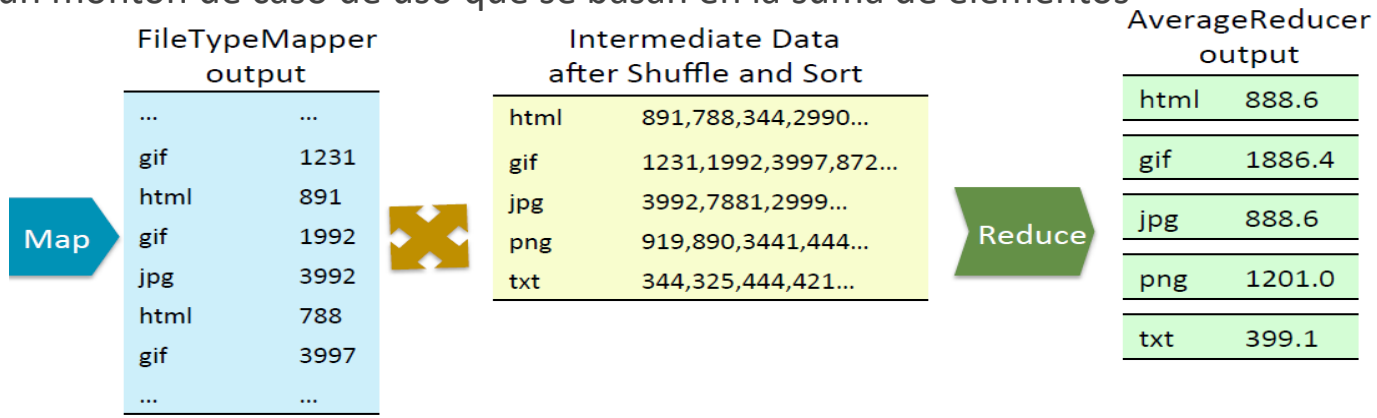
Example: WordCount



MapReduce

¿Por qué WordCount?

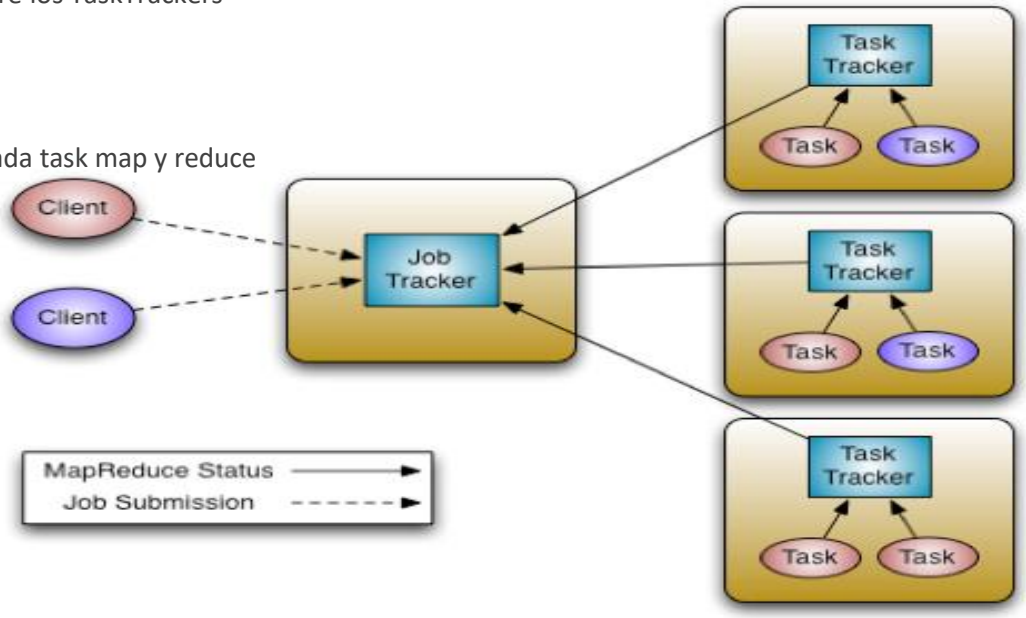
- Contar palabras es uno de los desafíos más típicos cuando se trabaja sobre enormes cantidades de datos
  - Si lo hiciéramos con un solo servidor tardaría horas
  - Almacenar cada palabras en memoria sería imposible
- La agregación de valores es una operación muy usada en Análisis de Datos
  - Ejemplos: máximo, mínimo, suma, contador, etc.
- Una de las características más importantes es que MR permite subdividir tareas complejas en un conjunto de otras más simples y ejecutarlas en paralelo
- Hay un montón de caso de uso que se basan en la suma de elementos



MapReduce

Demonios de MapReduce V1

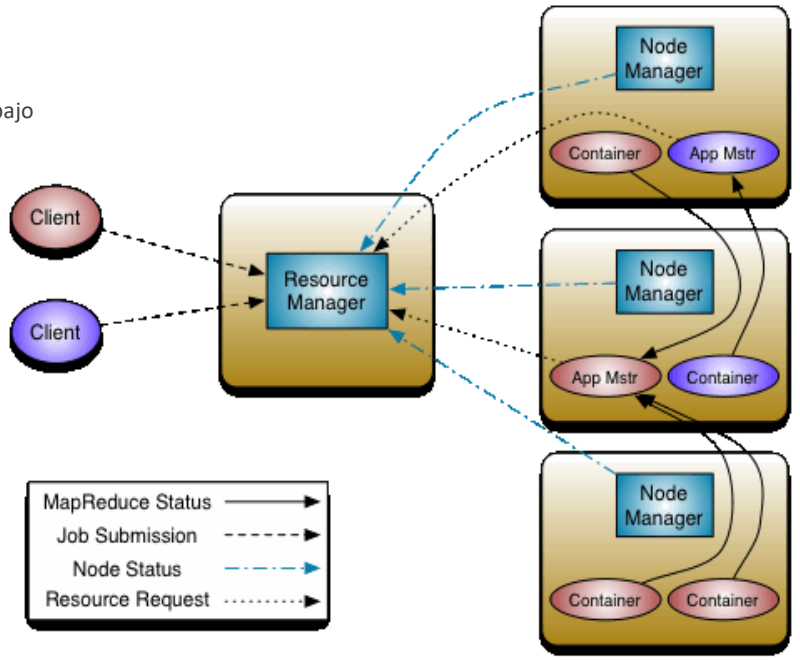
- JobTracker
  - Uno por Cluster
  - Corre en el nodo o nodos maestro activo
  - Gestiona los jobs MapReduce
  - Distribuye las Task entre los TaskTrackers
- TaskTracker
  - Uno por nodo esclavo
  - Ejecuta y monitorea cada task map y reduce



MapReduce

Demonios de MapReduce V2-Yarn

- Su objetivo es la de descargar de trabajo al JobTracker, que se encarga de todo en la versión V1. Es algo así como contratar encargados que se encarguen de trabajos específicos para que el jefe se pueda encargar de otras cosas de otra índole o mayor nivel. Sus demonios son:
- Resource Manager
  - Uno por cluster
  - Arranca el ApplicationMasters
  - Dota de recursos (CPU, RAM) a los nodos esclavos para que puedan hacer su trabajo
- ApplicationMasters
  - Uno por Job
  - Pide recursos
  - Gestiona cada task map y reduce en el conjunto de nodos en los que se están ejecutando las tareas asociadas al job
- Node Manager
  - Uno por nodo esclavo
  - Gestiona los recursos de cada nodo esclavo
- Containers
  - Conjunto de recursos cedidos por el RM tras una petición
- JobHistory
  - Uno por cluster
  - Almacena las métricas de los jobs y los metadatos



MapReduce

Terminología

- Un ***Job*** es un programa entero, una ejecución completa de Maps y Reduces sobre un dataset
- Una ***Task*** es la ejecución de un solo Map o Reduce sobre una porción de datos, habitualmente un Bloque.
- Un intento de Task se refiere a una ejecución concreta de una Task sobre un bloque de datos
  - Hay veces que una Tarea falla o va lenta
  - En ese caso existe algo llamado *Speculative Execution* que lo que hace es lanzar la misma Task sobre otro bloque en otro nodo y tomar como buena la primera que termina
  - Esto es iniciado por el JobTracker o el ApplicationMaster

MapReduce

Localidad del dato

- Cuando es posible, un Map Task se ejecuta en el nodo donde el bloque está localizado
- De otro modo, la Map Task se traerá el bloque de dato hasta el nodo donde se ejecuta a través de la red
  - Esto puede ocurrir por ejemplo cuando un nodo está sobrecargado de trabajo y merece la pena perder un poco de tiempo en el tráfico de red que esperar a que el nodo se libere

Datos Intermedios

- Son los datos generados por la fase de Shuffle&Sort
- Los datos intermedios se almacenan en disco local, no en HDFS
- Se borran una vez que el job ha terminado completamente

Reducers

- En ellos no hay concepto de Data Locality (es decir, sus datos no se procesan en el nodo donde se encuentran)
- La fase de S&S genera gran cantidad de movimiento a través de la red
- El cluster decide en qué nodos se van a ejecutar los Reducers
- Se envía la información a ellos a través de la red y se procesa
- La salida de los Reducers se escribe en HDFS

Suffle&Sort

- Se puede pensar que la fase de S&S es un cuello de botella
- Los Reducers no pueden empezar hasta que toda la fase de S&S haya terminado
- Pregunta: ¿La fase S&S puede empezar antes de que toda la fase Map haya terminado? Slide 36

MapReduce

S&S

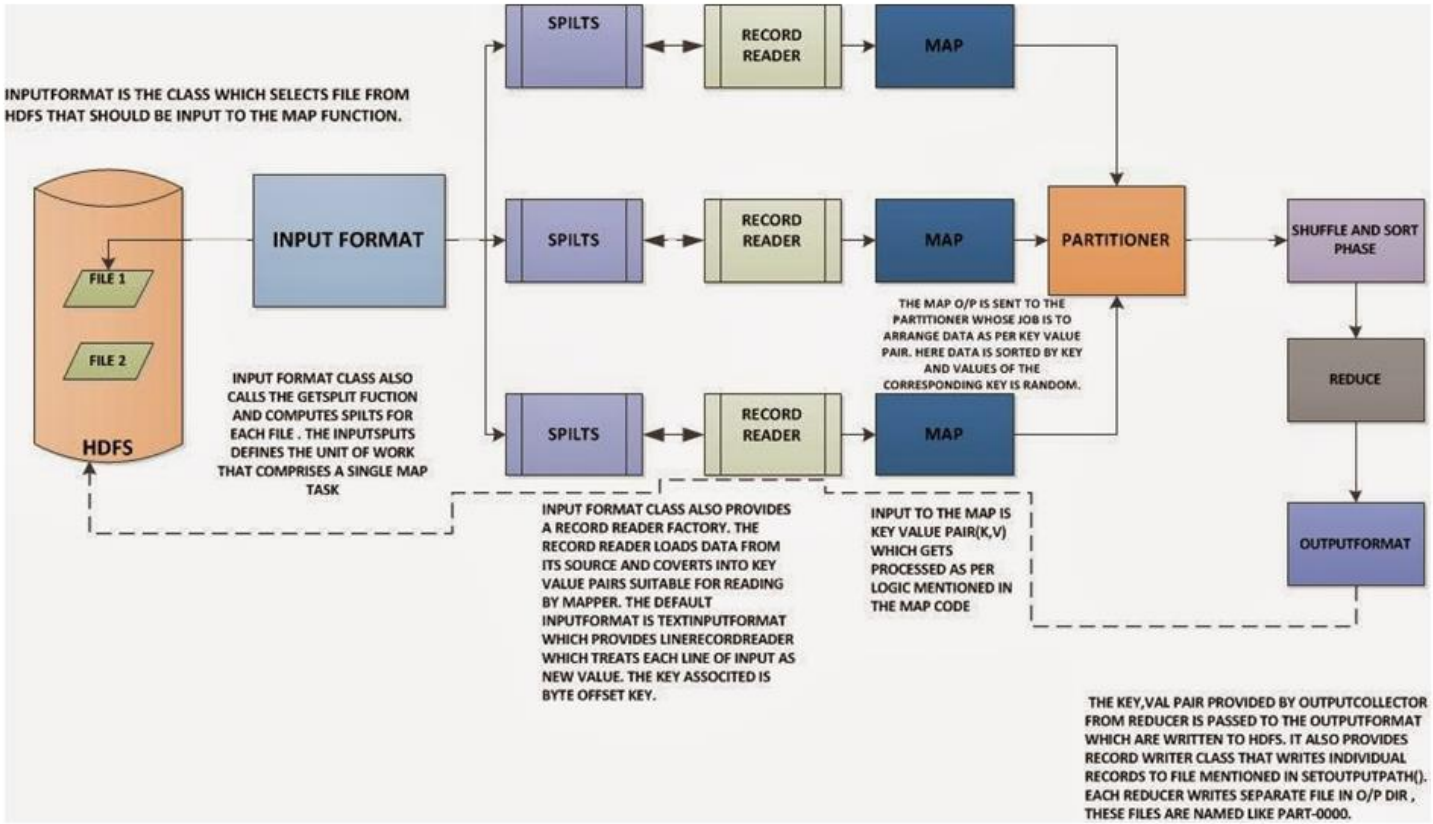
- Pregunta: ¿La fase S&S puede empezar antes de que toda la fase Map haya terminado?
- La realidad es que los Mappers empiezan a enviar datos a los Reducers tan pronto como van acabando sus Task a través de la fase de S&S, de modo que todo el tráfico de red se genera de forma progresiva, no al mismo tiempo
- Sigue siendo cierto que los Reducers no empiezan hasta que S&S haya acabado completamente

MapReduce

Detalle del proceso

- En la primera fase, los datos son divididos en Splits, tantos como sea necesario
- Cada Split es procesado por un Map
- Habitualmente un Split tiene el tamaño de un bloque
- Una vez procesados los datos por los Mappers, son almacenados en el sistema de archivos local a la espera de ser enviados al correspondiente Reducer
- Cuando se inicia la transferencia de datos intermedios al Reducer, se realiza la tarea de S&S, donde los datos son ordenados por su correspondiente clave, para que lleguen al mismo Reducer todos los datos asociados a una clave
- Se procesan los datos en el Reducer
- Por cada Reducer se genera un fichero con datos almacenado en HDFS
- Los datos producidos por cada Reducer están ordenados por Clave, pero el conjunto de datos de todos los Reducers no siguen un orden determinado
- Para conseguir un orden total por clave de todas las claves, bien usamos 1 solo Reducer o usamos Partitioners.

MapReduce


INPUTFORMAT IS THE CLASS WHICH SELECTS FILE FROM HDFS THAT SHOULD BE INPUT TO THE MAP FUNCTION.
HDFS
FILE 1
FILE 2
INPUT FORMAT
INPUT FORMAT CLASS ALSO CALLS THE GETSPLIT FUCTION AND COMPUTES SPILTS FOR EACH FILE . THE INPUTSPLITS DEFINES THE UNIT OF WORK THAT COMPRISES A SINGLE MAP TASK
SPILTS
RECORD READER
MAP
SPILTS
RECORD READER
MAP
SPILTS
RECORD READER
MAP
PARTITIONER
THE MAP O/P IS SENT TO THE PARTITIONER WHOSE JOB IS TO ARRANGE DATA AS PER KEY VALUE PAIR. HERE DATA IS SORTED BY KEY AND VALUES OF THE CORRESPONDING KEY IS RANDOM.
SHUFFLE AND SORT PHASE
REDUCE
OUTPUTFORMAT
INPUT FORMAT CLASS ALSO PROVIDES A RECORD READER FACTORY. THE RECORD READER LOADS DATA FROM ITS SOURCE AND COVERTS INTO KEY VALUE PAIRS SUITABLE FOR READING BY MAPPER. THE DEFAULT INPUTFORMAT IS TEXTINPUTFORMAT WHICH PROVIDES LINERECORDREADER WHICH TREATS EACH LINE OF INPUT AS NEW VALUE. THE KEY ASSOCITED IS BYTE OFFSET KEY.
INPUT TO THE MAP IS KEY VALUE PAIR(K,V) WHICH GETS PROCESSED AS PER LOGIC MENTIONED IN THE MAP CODE
THE KEY,VAL PAIR PROVIDED BY OUTPUTCOLLECTOR FROM REDUCER IS PASSED TO THE OUTPUTFORMAT WHICH ARE WRITTEN TO HDFS. IT ALSO PROVIDES RECORD WRITER CLASS THAT WRITES INDIVIDUAL RECORDS TO FILE MENTIONED IN SETOUTPUTPATH(). EACH REDUCER WRITES SEPARATE FILE IN O/P DIR , THESE FILES ARE NAMED LIKE PART-0000.

MapReduce

Código WordCount: Driver

```
public class WordCount {
  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.out.printf("Usage: WordCount <input dir> <output dir>\n");
      System.exit(-1);
    }
    Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

MapReduce

Código WordCount: Mapper

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();

    for (String word : line.split("\\W+")) {
      if (word.length() > 0) {

        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}
```

MapReduce

Código WordCount: Reducer

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int wordCount = 0;

    for (IntWritable value : values) {
      wordCount += value.get();
    }

    context.write(key, new IntWritable(wordCount));
  }
}
```

MapReduce: wordcount por partes

Código WordCount: Driver

Imports necesarios

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
```

MapReduce: wordcount por partes

Código WordCount: Driver

Método Main

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

Acepta dos argumentos

- Directorio de input
- Directorio de output

```
if (args.length != 2) {
  System.out.printf("Usage: WordCount <input dir> <output dir>\n");
  System.exit(-1);
}
```

- Lo primero que hace el método main es asegurarse de que se han pasado estos dos argumentos. Sino, da error y para la ejecución.

MapReduce: wordcount por partes

Código WordCount: Driver

Configuración del job

```
Job job = new Job();
job.setJarByClass(WordCount.class);
```

- Se crea el objeto Job
- Se identifica el jar que contiene el Mapper y el Reducer especificando una clase en él

El Job permite establecer la configuración para tu job MapReduce.

- Clases Map y Reduce
- Directorios de entrada y salida
- Otras opciones

Las configuraciones no establecidas en el driver las cogerá de /etc/hadoop/conf

Otras opciones no establecidas se establecerán según los valores por defecto de Hadoop

MapReduce: wordcount por partes

Código WordCount: Driver

Le damos al Job un nombre

```
job.setJobName("Word Count");
```

Especificamos los directorios de

- **Entrada:** desde donde se leerán los datos a procesar
- **Salida:** donde se dejarán los datos procesados

```
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

El InputFormat por defecto es (TextInputoFormat)

Para determinar otro InputFormat se usa

```
job.setInputFormatClass(KeyValueTextInputFormat.class)
```

MapReduce: wordcount por partes

Código WordCount: Driver

Le damos al Job el nombre de las clases que representa el Mapper y el Reducer.

```
job.setMapperClass(WordMapper.class);
job.setReducerClass(SumReducer.class);
```

Si no se indican, Hadoop ejecutará la función identidad por defecto

-IdentityMapper

mahout	an elephant driver

map()

mahout	an elephant driver

-IdentityReducer

bow	a knot with two loops and two loose ends
	a weapon for shooting arrows
	a bending of the head or body in respect

reduce()

bow	a knot with two loops and two loose ends
bow	a weapon for shooting arrows
bow	a bending of the head or body in respect

MapReduce: wordcount por partes

Código WordCount: Driver

Especificamos los tipos de datos intermedios para las K y las V producidos por el Mapper

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
```

Especificamos los tipos de datos de salida para las K y las V producidos por el Reducer.

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

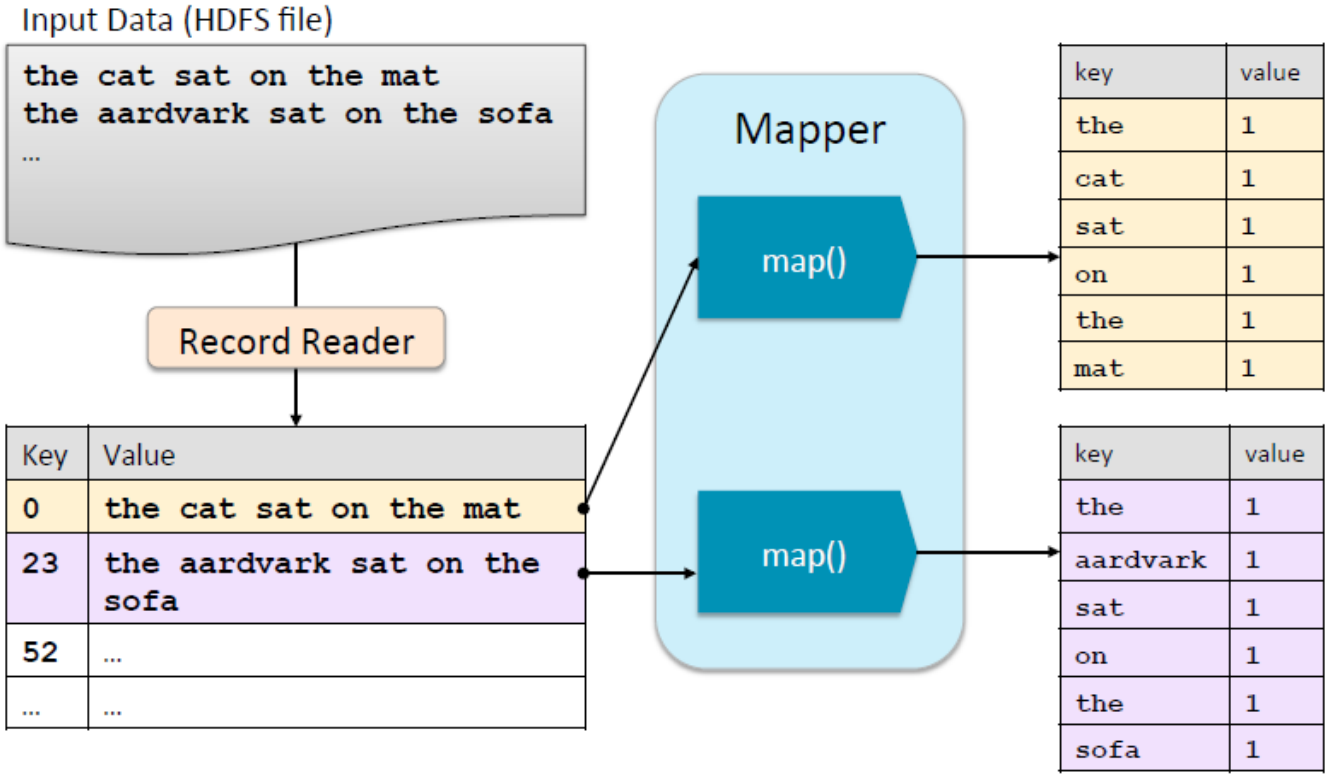
Arrancamos el job y esperamos a que se complete. (true) para mostrar el proceso

```
boolean success = job.waitForCompletion(true);
System.exit(success ? 0 : 1);
```

MapReduce: wordcount por partes

Código WordCount: Mapper

Recordemos



MapReduce: wordcount por partes

Código WordCount: Mapper

Imports

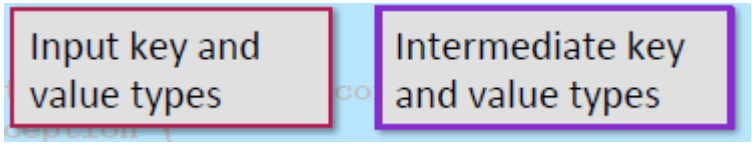
```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

El Mapper extiende de la clase base Mapper

```
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable>
```

Declaran 4 parámetros

- Input Key
- Input Value
- Output Key (datos intermedios)
- Output Value (datos intermedios)



Las **K** deben ser *WritableComparable*, las **V** *Writable*

MapReduce: wordcount por partes

Código WordCount: Mapper

Método map

```
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
```

Se le pasa la K, V y el objeto context. El Context es usado para escribir datos intermedios. Contiene info de la configuración del job.

Como el Value es un objeto Text, devolvemos el String que lo contiene

```
String line = value.toString();
```

MapReduce: wordcount por partes

Código WordCount: Mapper

Dividimos el string en palabras usando una expresión regular (el delimitador es un carácter no-alfanumérico) y hacemos un bucle que recorra cada palabra

```
for (String word : line.split("\\W+")) {
  if (word.length() > 0) {
```

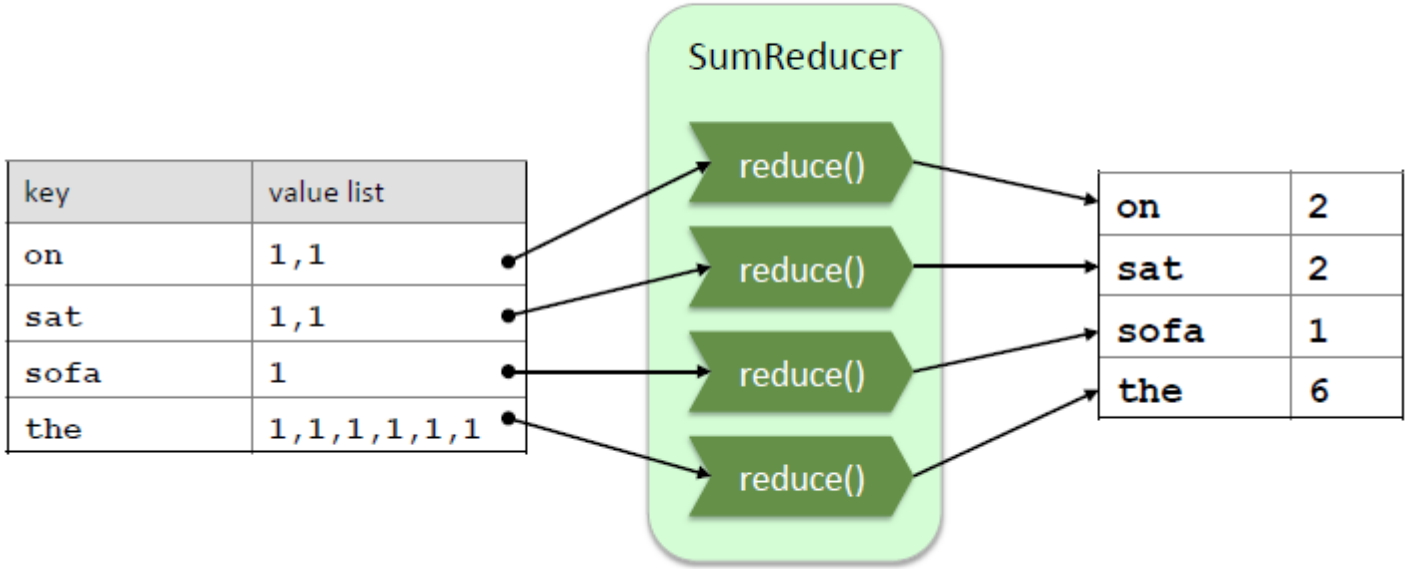
Se emite un par (K,V) llamando al método write del objeto Context. La K es cada palabra del bucle, y su valor es un 1.

```
context.write(new Text(word), new IntWritable(1));
```

MapReduce: wordcount por partes

Código WordCount: Reducer

Recordemos



key	value list
on	1,1
sat	1,1
sofa	1
the	1,1,1,1,1,1

on	2
sat	2
sofa	1
the	6

MapReduce: wordcount por partes

Código WordCount: Reducer

Imports

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

La clase Reducer extiende de la clase base Reducer

```
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
```

Intermediate key and value types

Output key and value types

Tiene 4 parámetros

- Input Key (datos intermedios)
- Input Value (datos intermedios)
- Output Key
- Output Value

MapReduce: wordcount por partes

Código WordCount: Reducer

El método reduce recibe como parámetros una K y una colección de objetos Iterables (que son los valores emitidos por el mapper para cada K). También recibe un objeto context

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
```

Para cada elemento de la colección iterable se van sumando los valores

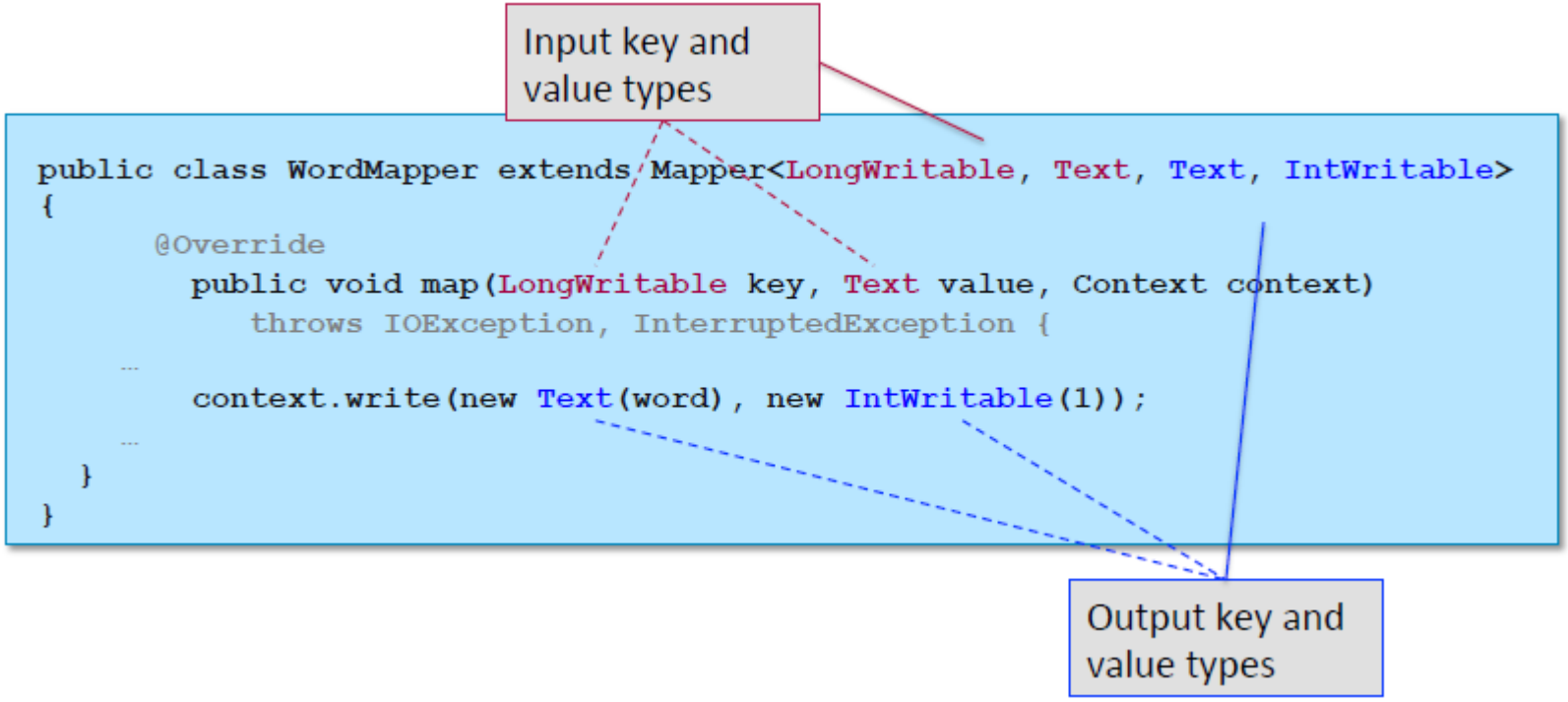
```
int wordCount = 0;

for (IntWritable value : values) {
  wordCount += value.get();
}
```

Finalmente, escribimos los K-V en HDFS usando el método write del objeto Context

```
context.write(key, new IntWritable(wordCount));
```

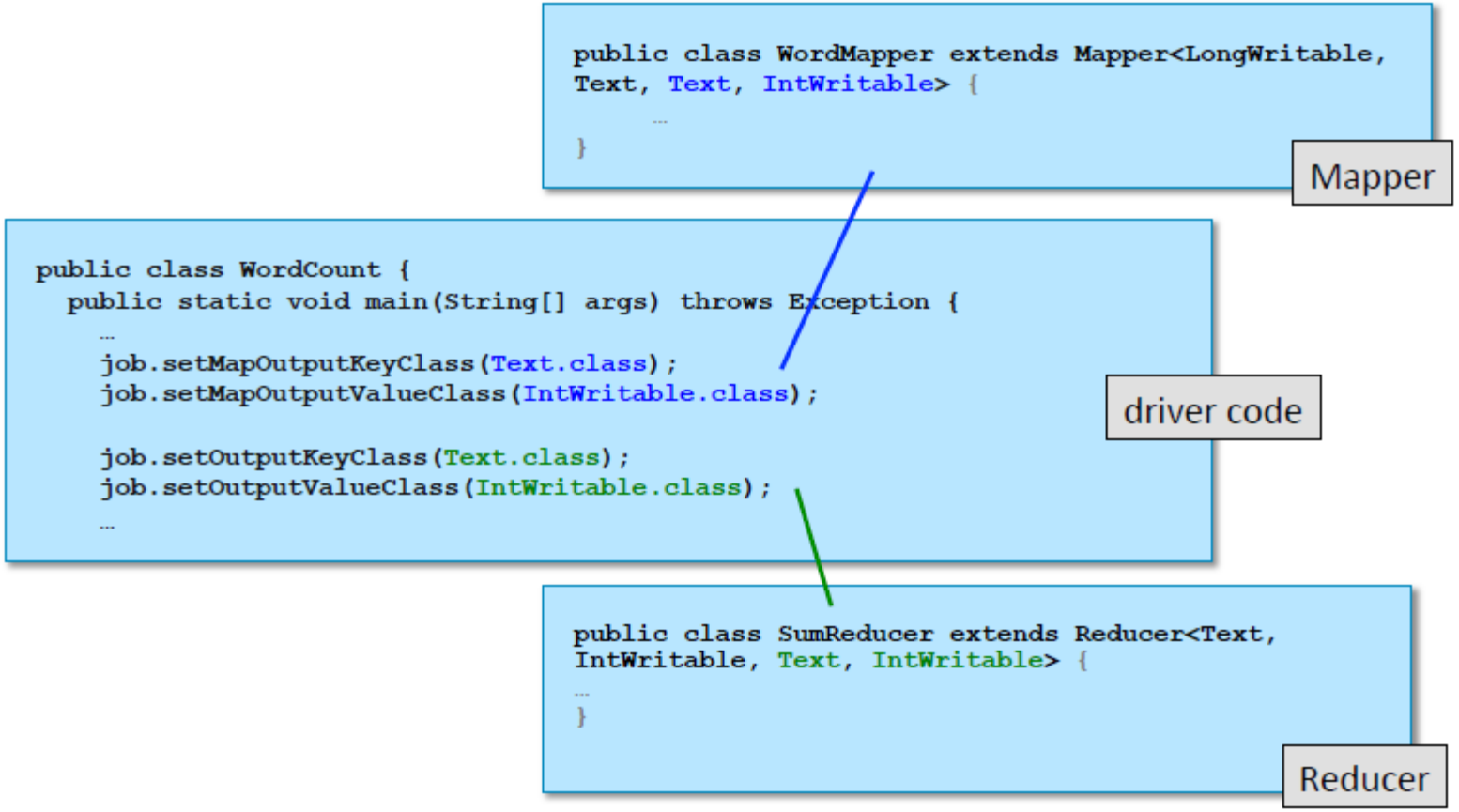
MapReduce: wordcount por partes

Código WordCount: Mapper



MapReduce: wordcount por partes

Código WordCount: Reducer



MapReduce

Ejercicios

2_Ejecutando un MapReduce: wordcount

- Ver documento de ejercicios adjunto.

Introducción al Ecosistema Hadoop

- Hadoop por sí solo no sería suficiente como entorno de trabajo en producción
- Las empresas necesitan agilidad a la hora de integrar nuevos sistemas
- Sus profesionales no siempre están acostumbrados a trabajar en entornos tecnológicos puros.
- Hay analistas de datos tradicionales que no saben que es /root
- Hay un montón de orígenes de datos válidos para su estudio
- Hay muchas herramientas de exploración y descubrimiento de datos
- Hoy en día casi todos los proveedores de SW del mercado proveen conectores de sus herramientas con Hadoop
- A continuación presentamos alguna de las herramientas principales empleadas en la mayoría de soluciones Big Data

Introducción al Ecosistema Hadoop

- Hive
  - Es un SW para interrogar/consultar y manipular datos escritos en HDFS
  - Fue creado por Facebook para suplir la carencia de conocimiento de java de sus programadores
  - A través de metadatos es posible trabajar con datasets con un lenguaje similar al SQL, llamado HQL
  - Su característica fundamental es que traduce Queries HQL a MapReduce, aprovechando toda la potencia del cluster
  - Conforme las versiones aumentan, se va añadiendo más funcionalidad
  - Un ejemplo:

```
SELECT sample_07.description, sample_07.salary
FROM
  sample_07
WHERE
( sample_07.salary > 100000)
SORT BY sample_07.salary DESC
```

Introducción al Ecosistema Hadoop

- Pig
  - Es una plataforma para analizar grandes datasets almacenados en HDFS
  - Desarrollado por Yahoo! Por la misma razón que FB creó Hive
  - Consta de un lenguaje propio llamado Pig Latin semejante a HQL pero menos que Hive
  - Su principal característica es que transforma consultas en PigLatin en MapReduce
  - Ejemplo

```
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
    group, COUNT(D);
STORE F INTO 'output';
```

Introducción al Ecosistema Hadoop

- Sqoop
  - Su función principal es la de traer datos desde BBDD relacionales a HDFS
  - Su sintaxis consta de una parte de configuración más otra parte de SQL
  - Originalmente fue creado por Cloudera
  - Actualmente existen conectores de casi todas las bases de datos relacionales con esta herramienta
  - Ejemplo

```
--target-dir /biginsights/hive/warehouse/obsidiana.db/big_CLDOMICILIO
$SQOOP_HOME/bin/sqoop import --options-file
/gneis/datos/comunes/conexion/sqoop.cfg/sqoop-options-SRVepifis.txt --fields-terminated-
by "\t" --m 1 --null-non-string "\\\\N" --hive-import  --hive-overwrite --hive-table
obsidiana.HST_EXT_CONTACTOS_BKTEL    --query  "select FECHA,TRANSACION,PERSONA
from epistage.HST_EXT_CONTACTOS_BKTEL where \$CONDITIONS"
```

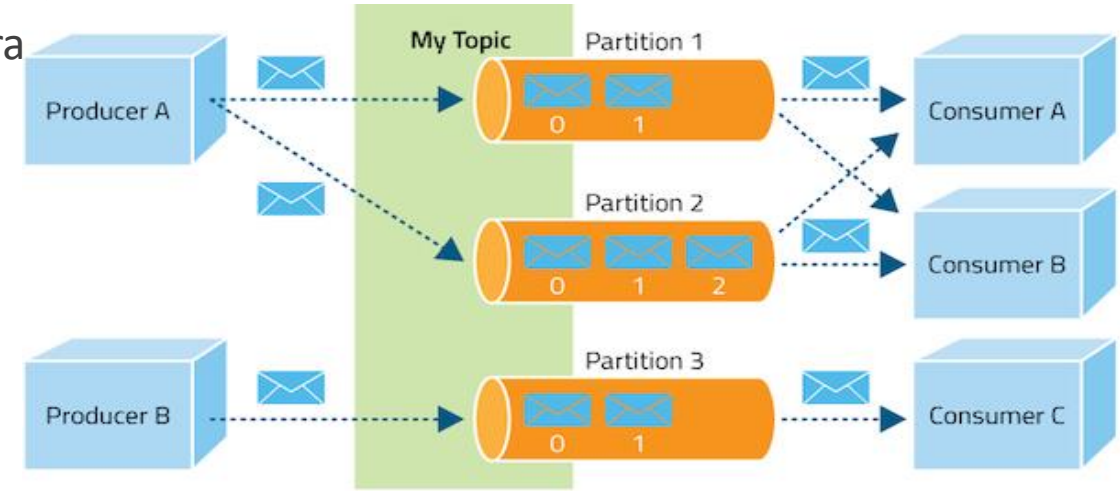
Introducción al Ecosistema Hadoop

- Flume
  - Herramienta diseñada para importar datos en un cluster en tiempo real
  - Pensada para orígenes de datos diferentes a BBDD relacionales (servidores web, servidores de correo, logs de dispositivos, etc.)
  - Fue inicialmente creado por Cloudera
  - Consta de tres partes que hay que configurar dependiendo de lo que queramos obtener o necesitemos
    - Source
    - Channel
    - Sink
  - Ejemplo

```
a1.channels = c1
a1.sources = r1
a1.sinks = k1

a1.channels.c1.type = memory

a1.sources.r1.channels = c1
a1.sources.r1.type = avro
# For using a thrift source set the following instead of the above line.
# a1.source.r1.type = thrift
a1.sources.r1.bind = 0.0.0.0
a1.sources.r1.port = 41414

a1.sinks.k1.channel = c1
a1.sinks.k1.type = logger
```

Introducción al Ecosistema Hadoop

- Kafka
  - Proyecto cuyo objetivo es proporcionar una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos
  - Puede verse como una cola de mensajes bajo el **patrón publicación-subscripción**, **masivamente escalable** y concebida como un registro de transacciones distribuidas
  - Desarrollado por Linkedin
  - Arquitectura

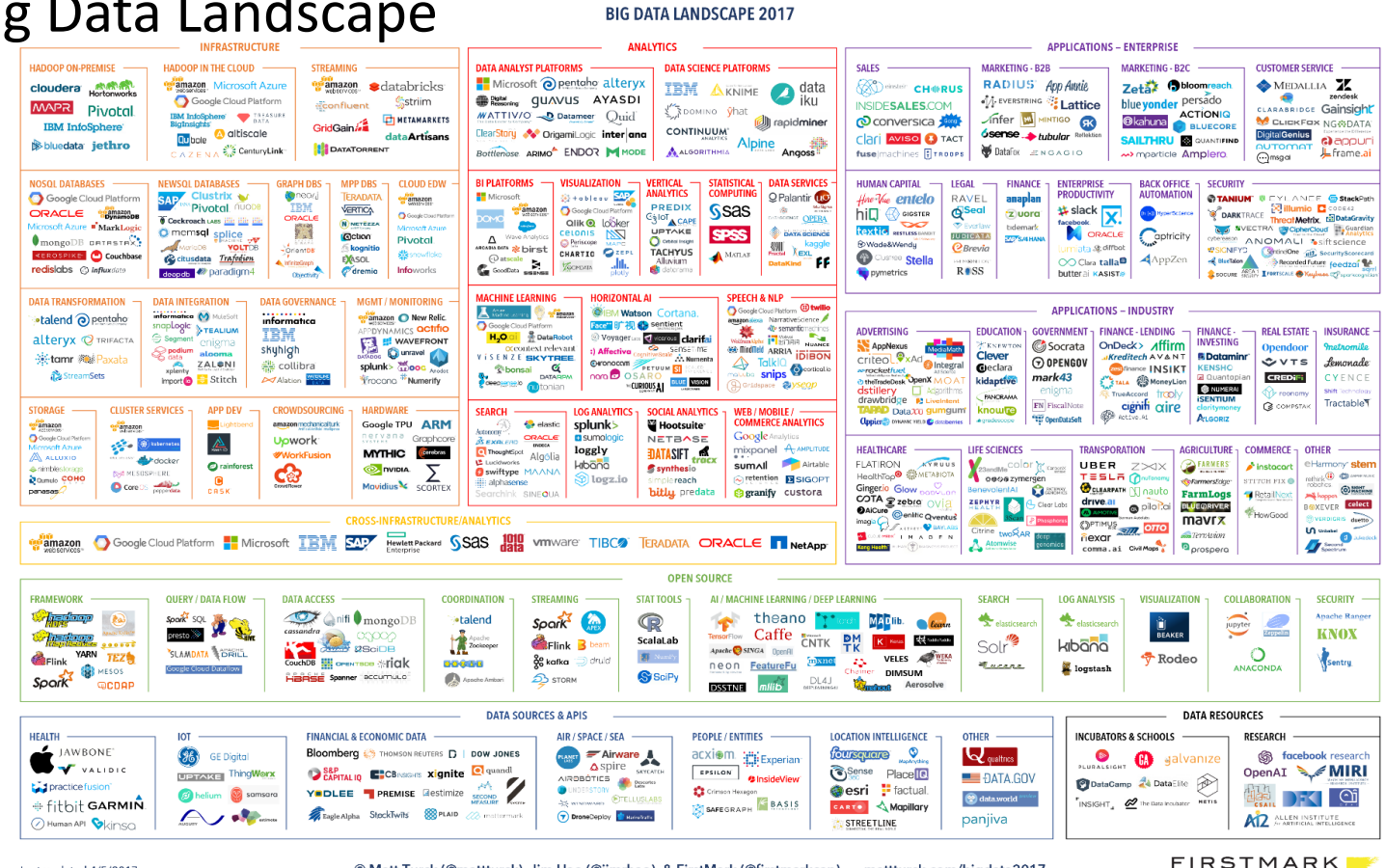


Introducción al Ecosistema Hadoop

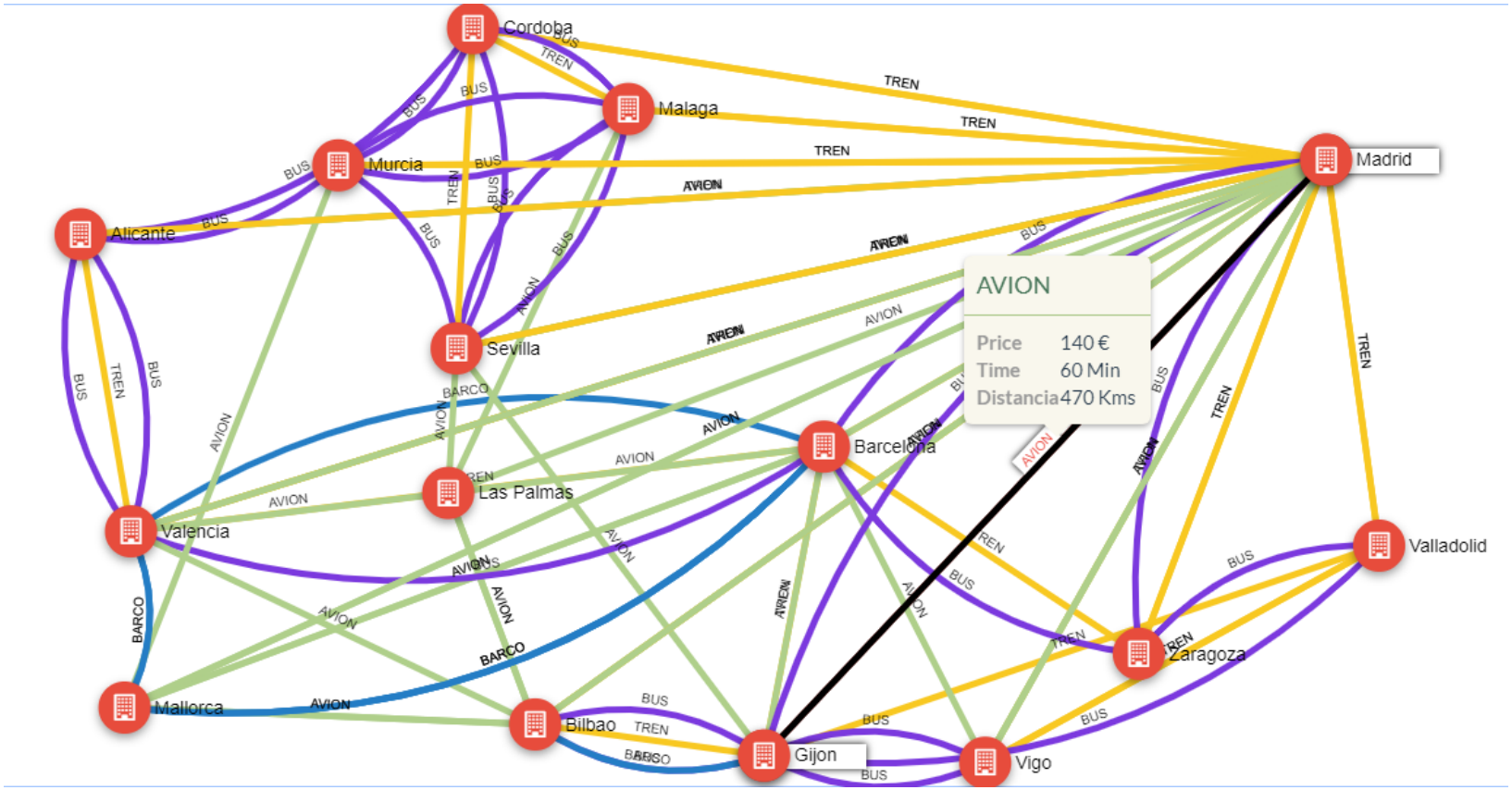
- Bases de datos No-SQL
  - No-SQL significa “Not Only SQL”, es decir, que no usan por defecto el lenguaje SQL para hacer consultas
  - Pensadas para trabajar con datos multiestructurados
  - Permiten distribución de datos y ejecución en paralelo
  - No garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad)
  - Escalan muy bien horizontalmente
  - Pensadas para millones de columnas por billones de filas
  - Hay varios tipos de ellas
    - De clave valor como Cassandra
    - Orientadas a documentos como MongoDB
    - Orientadas a grafos como Neo4j
    - Orientada a columnas del tipo clave valor como Hbase
    - Otras

Introducción al Ecosistema Hadoop

- Big Data Landscape



Neo4j


Cordoba
Malaga
Murcia
Alicante
Sevilla
Madrid
Barcelona
Las Palmas
Valencia
Mallorca
Bilbao
Gijon
Vigo
Zaragoza
Valladolid
AVION
Price 140 €
Time 60 Min
Distancia 470 Kms
TREN
BUS
AVION
BARCO

Planificación cluster Hadoop

- Planificar las capacidades de un cluster Hadoop no es complicado siempre y cuando sepamos lo que vamos a hacer con él
- Hay muchos factores que importan
  - Volumen de datos actual
  - Previsión de crecimiento anual
  - Casos de uso que vamos a llevar a cabo en él
  - Tipo de datos que vamos a incorporar
  - Herramientas que vamos a instalar
- Ejemplo
  - Supongamos que queremos crear un cluster que permita incorporar 6 TB de datos al mes
  - Tenemos un factor de replicación de 3
  - Por lo que necesitamos 3x6 TB = 18TB de almacenamiento nuevo cada mes
  - A lo que hay que añadir un 25% de espacio para uso del propio cluster
  - Necesitaremos alrededor de 24TB de almacenamiento nuevo al mes
  - Actualmente se están montando nodos con estas características
    - dual Xeon octocore, 256 GB RAM, 8 x 3TB SATA III, dual 10 Gbit de red
  - Por lo que necesitaremos un nodo nuevo cada mes

Planificación cluster Hadoop

Como sabemos, hay dos tipos de nodos

- Maestros
- Esclavos

- Los maestros están pensados para almacenar metadatos y gobernar el cluster (SPOF)
- Los esclavos para procesar/almacenar la información
- Originalmente, cuando Hadoop fue creado, se pensó para procesamiento por lotes, por lo que era justificable, en base al uso que se le daba a cada nodo, que los maestros tuvieran más RAM y menos disco y los esclavos más disco y menos RAM
- Con la llegada de nuevas herramientas y el enorme éxito que está teniendo Big Data, cada vez se pide más que todo el procesamiento sea en memoria para que se realice más rápidamente, Near Real Time o Real Time (Spark, Storm, Flink …)
- Por esta razón los nodos de los cluster actuales tiene características similares entre ellos
- No es necesario un gasto extra en CPU dado que habitualmente los cuellos de botella de los clusters se encuentran en el Disco/Memoria y en el tráfico de red
- Si nuestro cluster va a ser utilizado para trabajar con algoritmos de ML, sí es recomendable gastar dinero en un procesador potente, dado que nos ahorraremos bastantes horas de trabajo esperando a que terminen los procesos
- Algunos clusters incorporan discos SSD (discos de estado sólido) para agilizar algunos procesos (Solr) aunque resultan caros todavía en comparación a los discos ópticos

Big Data: Negocio

1. Cuándo implantar una solución Big Data
2. Como impacta Big Data en las empresas
3. Beneficios del Big Data
   - Democratización del Dato: transversalidad
   - Unificación de conocimiento departamental
   - Decisiones basadas en el Dato
   - Mejora de procesos
   - Generación de nuevos procesos
   - Cambio de mentalidad directiva

Preguntas/Debate