

Informática como Servicio

Map Reduce

Ana M. Ferreiro

ana.fferreiro@udc.es

Departamento de Matemáticas

Máster Universitario en Ingeniería Informática

1 Big Data

2 Map–Reduce

Big Data

Muchos datos se están recogiendo y almacenando

- Datos de la Web, comercio electrónico
- Compras en almacenes/tiendas
- Bancos
- Redes Sociales

Explosión de Datos

- **Air Bus A380:**
 - 1 billón de código
 - cada motor genera 10 TB c/30 min
 - 640 TB por vuelo
- **Twiter** genera aproximadamente 12 TB de datos al día
- **New York Stock** intercambia 1 TB de datos al día

**Capacidad de almacenamiento se ha duplicado
aproximadamente cada tres años desde la década de 1980**

Explosión de Datos

- **Ciencia:**

Bases de datos de astronomía, genoma, datos medioambientales, datos de transporte, ...

- **Humanidades y Ciencias Sociales:**

Libros escaneados, documentos históricos, datos de las interacciones sociales, las nuevas tecnologías como el GPS ...

- **Comercio y Negocios:**

Las ventas corporativas, transacciones del mercado de valores, el tráfico aéreo, ...

- **Entretenimiento:**

Imágenes de Internet, películas de Hollywood, archivos MP3, ...

- **Medicina:**

Resonancia magnética y tomografías computarizadas, registros de pacientes, ...

Big Data

Big Data

”Volumen masivo de datos, tanto estructurados como no-estructurados, los cuales son **demasiado grandes y difíciles de procesar** con las bases de datos y el software tradicionales” (ONU, 2012)

Big Data

Los grandes datos permiten una mayor inteligencia de negocios mediante el almacenamiento, el procesamiento y el análisis de datos que se ha ignorado con anterioridad debido a las limitaciones de las tecnologías tradicionales de gestión de datos.

Source: Harness the Power of Big Data: The IBM Big Data Platform

Características de Big Data

- Eficiente procesamiento cada vez mayor de grandes **Volumenes** de datos
- En respuesta a la creciente **Velocidad**
- Analizar la amplia **Variedad**
- Establecer la **Veracidad** de las fuentes de datos grandes

Características de Big Data

VOLUMEN

- Hoy en día, se estima que estamos produciendo 2,5 trillones de bytes de datos por día.
- La velocidad a la que se producen los datos está en constante aumento.
- Aproximadamente el 90 % de los datos producidos son no estructurados, i.e., procedentes de fuentes que no son las bases de datos convencionales.
- Se producen a un ritmo muy elevado.

Por ejemplo, en 60 segundos hay más de 80 mil publicaciones en Facebook, 370 miles de llamadas de Skype y más de 170 millones de mensajes de correo electrónico enviados.

Características de Big Data

VELOCIDAD

- Los datos que recogemos hoy en día, en muchos casos llegan a los sistemas de procesamientos a tasas muy altas.
- Esto no es un problema, si la única tarea es almacenar los datos. **Pero sí lo es si se quiere analizar y detectar una situación que se modela en los datos en tiempo de ejecución y realizar algún tipo de acción inmediata.**
- En el pasado, no se estaba al tanto de un problema antes de que alguien informara del problema. **En la era moderna, un problema se puede detectar en tiempo real.**

Características de Big Data

VARIEDAD

Existe una necesidad no sólo de manejar grandes volúmenes de datos, sino también de administrar una gran variedad de ellos

- Los datos de la redes sociales : publicaciones en Facebook, llamadas de Skype, tweets y mensajes de correo electrónico enviados.
- Dispositivos en una ciudad:
 - las cámaras en diferentes puntos de la ciudad permiten a los empleados de vigilancia ver un incidente en el momento en que sucede,
 - la temperatura en diversos puntos de la ciudad,
 - sensores inteligentes que son capaces de conocer el tráfico en una calle, la ubicación de cada autobús o tren.
- Datos transaccionales: un número de transacciones de tarjetas de crédito debe desencadenar inmediatamente una alarma de actividades sospechosas y bloquear la tarjeta a espera de verificación.

Características de Big Data

VERACIDAD

La confianza que uno pone en los datos

- Dado que los datos provienen de fuentes que pueden no estar bajo nuestro control, pueden tener muchos problemas de calidad de datos, no pueden ser certificados, etc.
- Hay una necesidad de medir la confianza que ponemos en los datos, que a su vez significa la confianza que ponemos en las conclusiones y puntos de vista que hemos obtenido mediante el análisis de datos.

Características de Big Data

Los aspectos en que los usuarios quieren interactuar con sus datos...

- **Procesamiento:** los usuarios tienen un mayor deseo de procesar y analizar todos los datos disponibles.
- **Exploración:** los usuarios quieren aplicar enfoques analíticos en el que el esquema se define en respuesta a la naturaleza de la consulta.
- **Frecuencia:** los usuarios tienen un deseo de aumentar la velocidad de análisis, con el fin de generar inteligencia de negocios más precisa y oportuna.
- **Eficiencia:** necesidad de los usuarios para equilibrar la inversión en tecnologías y capacidades existentes, con la adopción de nuevas tecnologías.

El valor de Big Data

- Predecir el comportamiento del cliente en todos los ámbitos.
- Comprender el comportamiento del cliente
- Mejorar la eficacia operativa
 - Máquinas/sensores: predecir fracasos, ataques a la red
 - Gestión de riesgos financieros: reducir el fraude, aumentar la seguridad
- Reducir el costo del almacenamiento de datos (*data warehouse*)
 - Integrar las nuevas fuentes de datos sin aumentar el costo base de datos

Importancia del Big Data

- **Gobierno**

- En 2012, el gobierno de Obama anunció financiamiento para la Investigación en Big Data

- **Sector privado**

- Walmart (multinacional americana) maneja más de 1 millón de transacciones de los clientes cada hora, que se almacenan en sus bases de datos, que se estima que contienen más de 2,5 petabytes de datos
- Falcon Credit Card Fraud Detection System protege 2100000000 cuentas en todo el mundo

- **Ciencia**

- Gran Telescopio para Rastreo galáctico genera 140 terabytes de datos cada 5 días
- Cálculos de Medicina como la decodificación del genoma

Ejemplo uso Big Data

- Predicción de análisis de datos para la elección EE.UU. en 2012.
- IBM y el concurso de IBM Watson Deep Blue
 - Un ordenador debería responder a las preguntas de conocimiento como las que los seres humanos responden normalmente
 - El secreto fue añadir un nivel para procesar enormes cantidades de información escrita en texto en lenguaje natural, principalmente la información de Wikipedia, y en poco tiempo a hacer las combinaciones necesarias para formular las respuestas
- Wal-Mart registra miles de transacciones de sus clientes
 - Las analiza para entender la popularidad de los productos, y asociar con sus preferencias y hábitos
- Empresas financieras como American Express
 - Están ejecutando extensos análisis y técnicas de inteligencia de negocio en las transacciones realizadas por sus clientes para tratar de entender sus intenciones. Por ejemplo, para identificar potenciales clientes insatisfechos en los diferentes servicios financieros y predecir los clientes que desaparecerán en el futuro

Sistema de ficheros distribuido

- Un **Sistema de ficheros distribuido** es sistema de ficheros en red que permite el acceso a ficheros distribuidos por varios servidores.
- Uno o más servidores almacenan ficheros que se pueden acceder, con los derechos adecuados de autorización, por cualquier cliente remoto en la red
- Cuando el cliente solicita un archivo, el servidor envía una copia del archivo al usuario, que es capaz de trabajar con el archivo como si se tratara de su propio ordenador.
- Cuando un archivo es alterado por el cliente, las modificaciones se devuelven a través de la red al servidor.

Sistema de ficheros distribuido

- Se construye a partir de un clúster de nodos.
- Los nodos se dividen en dos tipos: un nodo maestro y varios nodos de tipo Worker (conocidos como Chunkservers).
- Cada archivo se divide en trozos de tamaño fijo (por defecto, 64 MB) y cada trozo se replica en varios sitios.
- Cliente accede a los trozos mediante la consulta primero al servidor maestro para ubicar los trozos deseados. Luego, contacta los chunkserver y recibe los datos directamente.

1 Big Data

2 Map–Reduce

- Concepto
- Ejemplos
- Implementaciones

MapReduce

Es un modelo de programación diseñado para dar soporte al cálculo paralelo sobre colecciones de datos de gran tamaño y tolerancia a fallos en grandes sistemas de commodity hardware

- Desarrollado por GOOGLE en 2004
- Pensado para procesar grandes conjuntos de datos, distribuidos a lo largo de un clúster de servidores
- Por regla general se abordan problemas con conjunto de datos de gran tamaño, alcanzando los petabytes de tamaño.
- Puede tener lugar sobre datos almacenados en sistemas de ficheros como en bases de datos
- Los datos son tan grandes, que deber ser distribuidos a través de miles de máquinas con el fin de ser procesados en un período razonable de tiempo

MapReduce

- Esta distribución implica computación paralela ya que los mismos cálculos se realizan en cada CPU, pero con un conjunto de datos diferentes.
- **Map–Reduce es una abstracción que permite realizar cálculos sencillos, al tiempo que oculta los detalles de paralelización, distribución de datos, equilibrio de carga y tolerancia a fallos.**
- Popularizado por la implementación open source Apache Hadoop
 - Usado por múltiples organizaciones como Facebook, Twitter, Tuenti, Last.fm, eBay, LinkedIn, Rackspace, AWS, etc.

¿Para qué se usa?

- En Google:
 - Construcción de índices para el buscador (pagerank)
 - Clustering de artículos en Google News
 - Búsqueda de rutas en Google Maps
 - Traducción estadística
- En Facebook:
 - Minería de datos
 - Optimización de adds
 - Detección de spam
 - Gestión de logs
- En investigación:
 - Análisis astronómico, bioinformática, física de partículas, simulación climática, procesamiento del lenguaje natural, ...

MapReduce

Funciones de base de Map–Reduce

- Es un modelo de programación inspirado en los lenguajes funcionales.
- Permite al desarrollador expresar sus algoritmos utilizando básicamente dos funciones, **map** y **reduce**

Por ejemplo, en PYTHON:

- ```
>>> f = lambda x: x**2
>>> map (f, [1, 2, 3, 4])
[1, 4, 9, 16]
```
- ```
>>> g = lambda x,y: x+y
>>> reduce (g, [1, 2, 3, 4])
10
```
- # Suma de los cuadrados de las diferencias

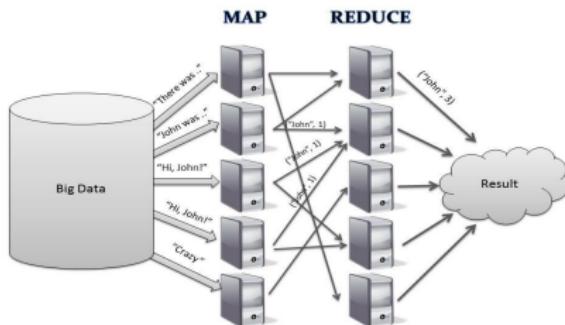
```
>>> a1 = numpy.array ([1, 2, 3, 4])
>>> a2 = numpy.array ([5, 6, 7, 8])
>>> reduce (g, map (f, a1-a2))
64
```

MapReduce

- Entrada:
conjunto de pares clave/valor (k / v)
- El usuario proporciona dos funciones:
 - $\text{map } (k, v) \longrightarrow \text{list } (k_1, v_1)$
 - $\text{reduce } (k_1, \text{list}(v_1)) \longrightarrow v_2$
- Salida:
conjunto de pares (k_1 / v_2)

MapReduce. Fundamentos

- Es un framework que proporciona un sistema de procesamiento de datos paralelo y distribuido
- Utiliza un sistema de archivos distribuido
- Tiene una arquitectura maestro/esclavo
- Cuenta con un servidor maestro o **JobTracker** y varios servidores esclavos o **TaskTrackers**, uno por cada nodo del clúster.
- El **JobTracker** es el punto de interacción entre los usuarios y el framework Map–Reduce.



MapReduce. Fundamentos

- ① Los usuarios envían trabajos MapReduce al **JobTracker**, que los pone en una cola de trabajos pendientes y los ejecuta en el orden de llegada
- ② El **JobTracker** gestiona la asignación de tareas y delega las tareas de los **TaskTrackers**
- ③ Los **TaskTrackers** ejecutan tareas bajo la orden del JobTracker y también manejan el movimiento de datos entre la fase Map y Reduce.

MapReduce

Paralelización automática:

- Dependiendo del tamaño de los datos de entrada sin procesar \Rightarrow instancia de múltiples tareas MAP
- Del mismo modo, dependiendo de la cantidad de producto intermedio `<clave, valor>` particiones \Rightarrow intancia múltiples tareas REDUCE

En tiempo de ejecución:

- Partición de datos
- Planificación de tareas
- Manejo de fallos
- Gestión de comunicación entre máquinas

Completamente transparente para el programador/usuario

MapReduce

- Inspirado en programación funcional.
- Dos componentes: **mapper** y **reducer**
- Los datos se trocean para su procesamiento
- Cada dato asociado a una clave
- Transforma $[(\text{clave1}, \text{valor1})]$ en $[(\text{clave2}, \text{valor2})]$



MapReduce

Mapper

- El Mapper (función Map) recibe como parámetros un par (`clave, valor`) y devuelve una lista de pares
- Se encarga del mapeo y se aplica a cada elemento de la entrada de datos, por lo que se obtendrá una lista de pares por cada llamada a la función Map
- Después se agrupan todos los pares con la misma clave de todas las listas, crea un grupo por cada una de las diferentes claves generadas

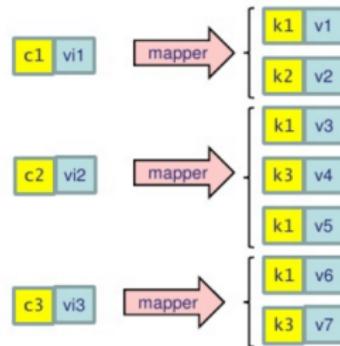
`Map(clave_1, valor_1) -> lista (clave_2, valor_2)`

- La operación Map se paralleliza.
El conjunto de archivos de entrada se divide en varias tareas, dicho proceso se llama FileSplit
- Las tareas se distribuyen a los nodos TaskTrackers, y estos a su vez pueden realizar la misma tarea si hiciera falta

MapReduce

Mapper

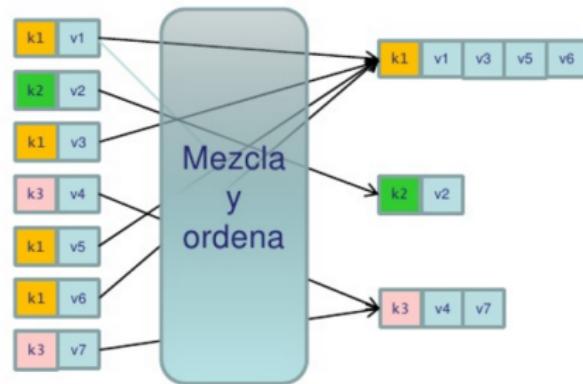
- **Para cada (clave1,valor1) devuelve una lista de (clave2,valor2)**
- **Tipo:** (clave1,valor1) \rightarrow [(clave2,valor2)]



MapReduce

Mapper

- **Mezcla y ordenación de claves:** el sistema se encarga de mezclar y ordenar los resultados intermedios en función de las claves



MapReduce

Reducer

- El Reducer (función Reduce) se aplica en paralelo para cada grupo creado por la función Map
- La función Reduce se llama cada vez para cada clave única de la salida de la función Map
- Junto con esta clave, se pasa una lista de todos los valores asociados con la clave para que pueda realizar alguna función para producir un conjunto más pequeño de los valores

Reduce(clave_2, lista(valor_2)) -> lista (valor_2)

- Cuando se inicia la tarea Reduce, la entrada se encuentra dispersa en varios archivos a través de los nodos en las tareas de Map
- Los datos obtenidos de la fase Map se ordenan para que los pares clave-valor sean contiguos (fase de ordenación, *sort fase*)
- Una vez que todos los datos están disponibles a nivel local, el archivo se fusiona (*merge*) de forma ordenada
- Al final, la salida consistirá en un archivo de salida por tarea `reduce` ejecutada

MapReduce

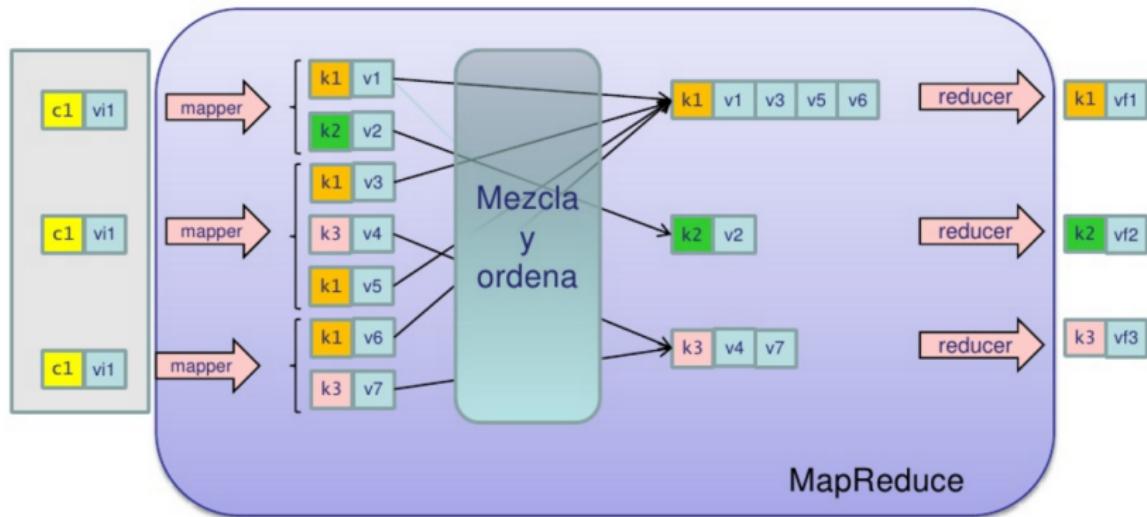
Reducer

- Para cada clave2, toma la lista de valores asociada y los combina en uno solo
- Tipo: $(\text{clave2}, [\text{valor2}]) \rightarrow (\text{clave2}, \text{valor3})$

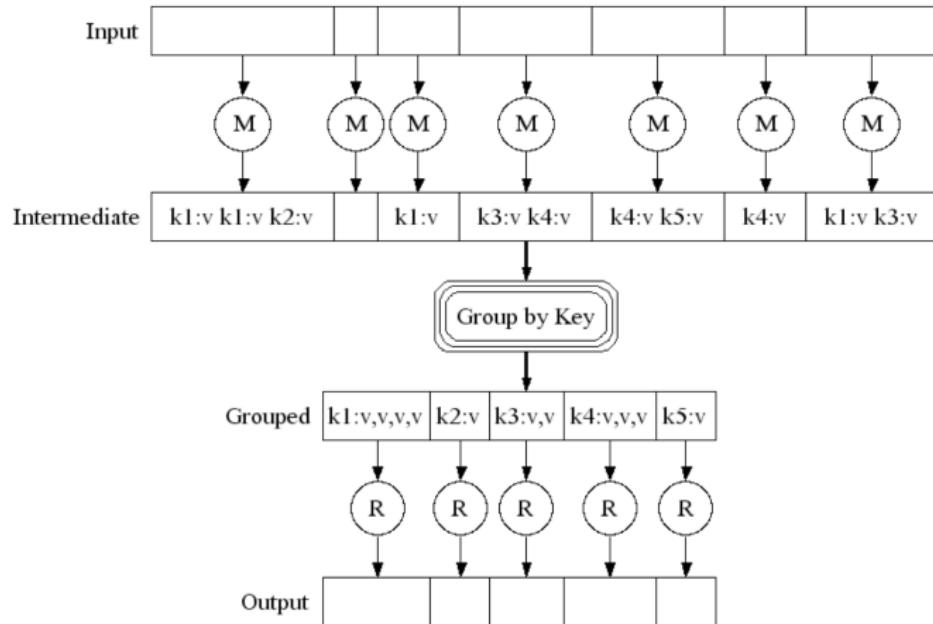


MapReduce

Esquema general



MapReduce

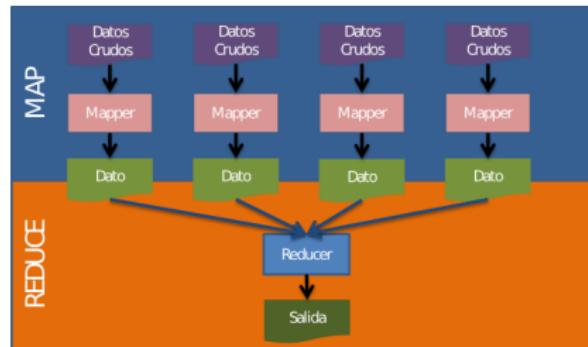


MapReduce

Típos de Entrada y salida de un trabajo MapReduce

(Entrada) $(k_1, v_1) \rightarrow \text{Map} \rightarrow (k_2, v_2) \rightarrow \text{Reduce} \rightarrow (k_3, v_3)$ (salida)

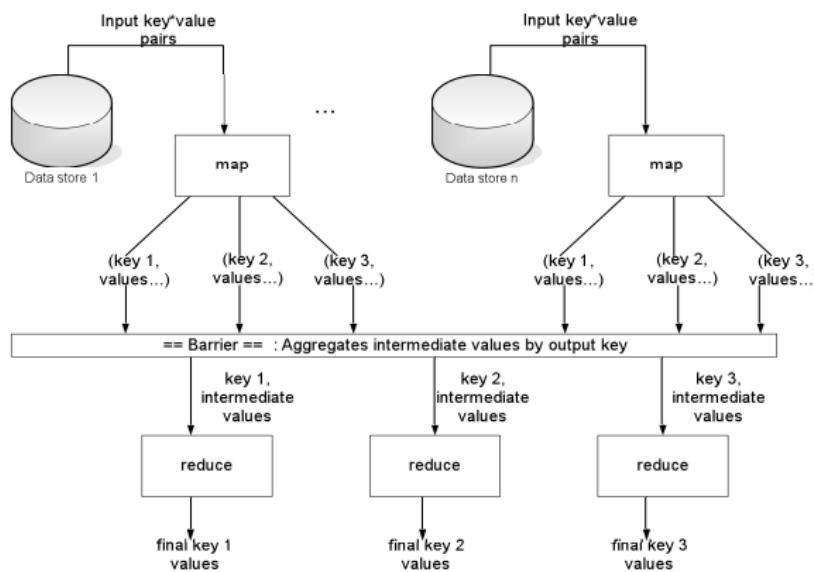
	Entrada	Salida
Map	(k_1, v_1)	$\text{list}((k_2, v_2))$
Reduce	$(k_2, \text{list}(v_2))$	$\text{list}((k_2, v_2))$



MapReduce

- Las funciones **map** trabajan en paralelo, creando diferentes valores intermedios para diferentes conjuntos de datos
- Las funciones **reduce** trabajan también en paralelo, cada una sobre una clave

Cuello de botella: la fase **reduce** no puede comenzar hasta que hayan terminado todos los procesos **map**



Tolerancia a fallos

- Si una tarea no progresiona (*straggler* o *rezagada*)
 - Se lanza una segunda copia de la tarea en otro nodo (ejecución especulativa)
 - Se toma la salida de la tarea que acaba antes, y se mata a la otra
 - Situación bastante común en clusters grandes
 - Debidos a errores de hardware, bugs software, fallos de configuración, etc.
 - Una tarea rezagada puede relantizar de forma importante un trabajo
- Si falla el Master:
 - Se intenta relanzar de nuevo
 - Las tareas en proceso o acabadas durante el reinicio, se relanzan
 - Si continua fallando, el trabajo se aborta y se notifica al usuario

Ejemplo: Contar palabras

Contar las apariciones de cada palabra en un conjunto de ficheros

- Entrada: ficheros de texto
- Mapper: devuelve cada palabra con un 1

```
mapper(d,ps){  
    for each p in ps:  
        emit(p,1)  
}
```

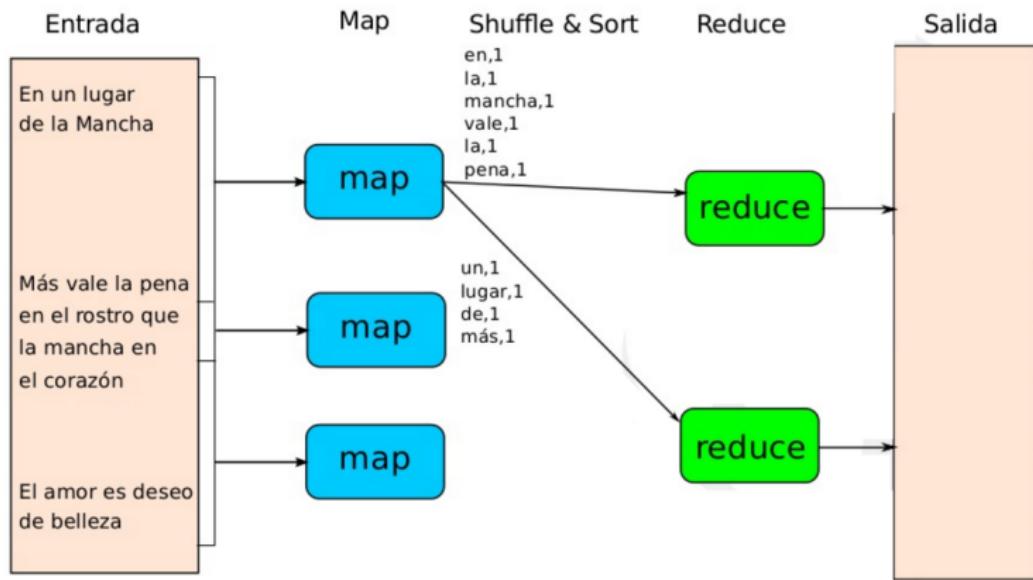
El mapper divide un documento en palabras (es decir lo tokeniza) mediante el empleo de un analizador léxico, y emite una serie de tuplas de la forma (clave, valor) donde la clave es la palabra y el valor es 1.

- Reducer: suma la lista de números de cada palabra

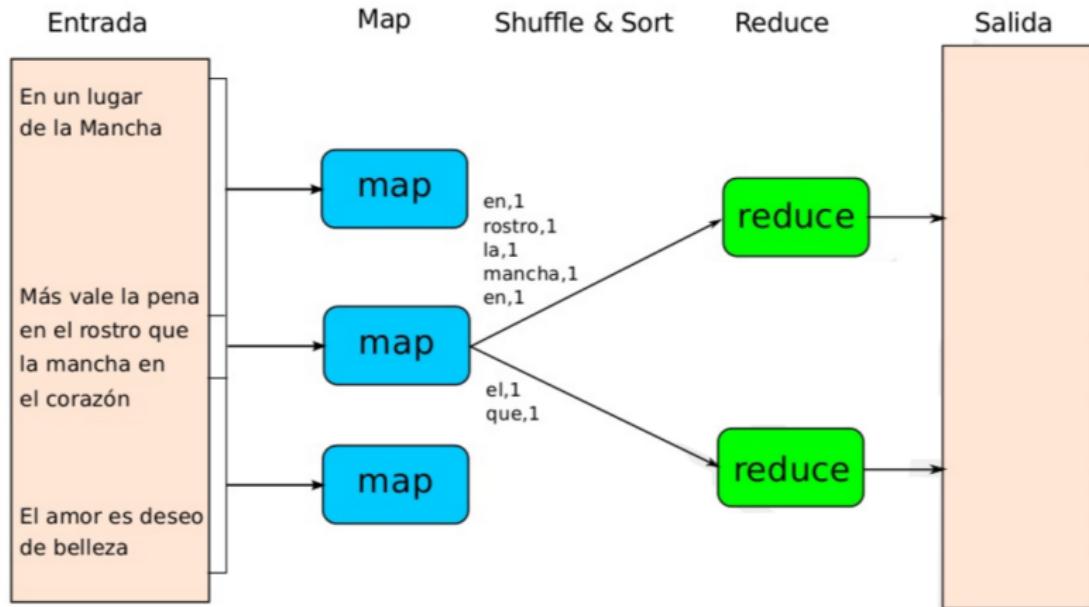
```
reducer (p,ns){  
    sum=0  
    for each n in ns:  
        sum+=n  
    emit(p,sum)  
}
```

El framework reúne todos los pares con la misma clave y alimenta a la misma llamada Reduce, para sumar todos los valores de su entrada para encontrar el total de las apariciones de esa palabra

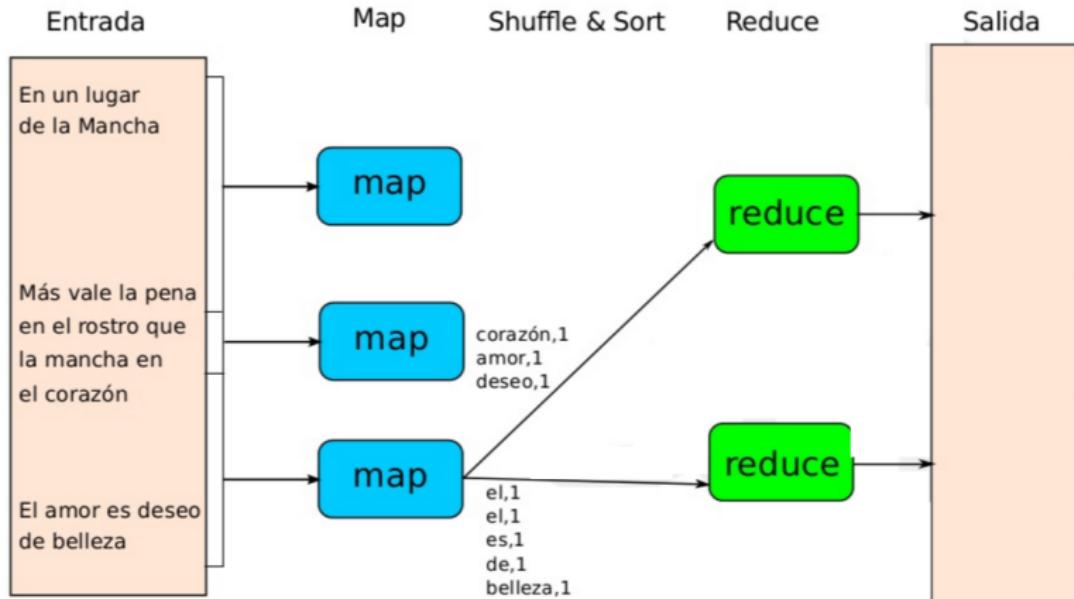
Ejemplo: Contar palabras



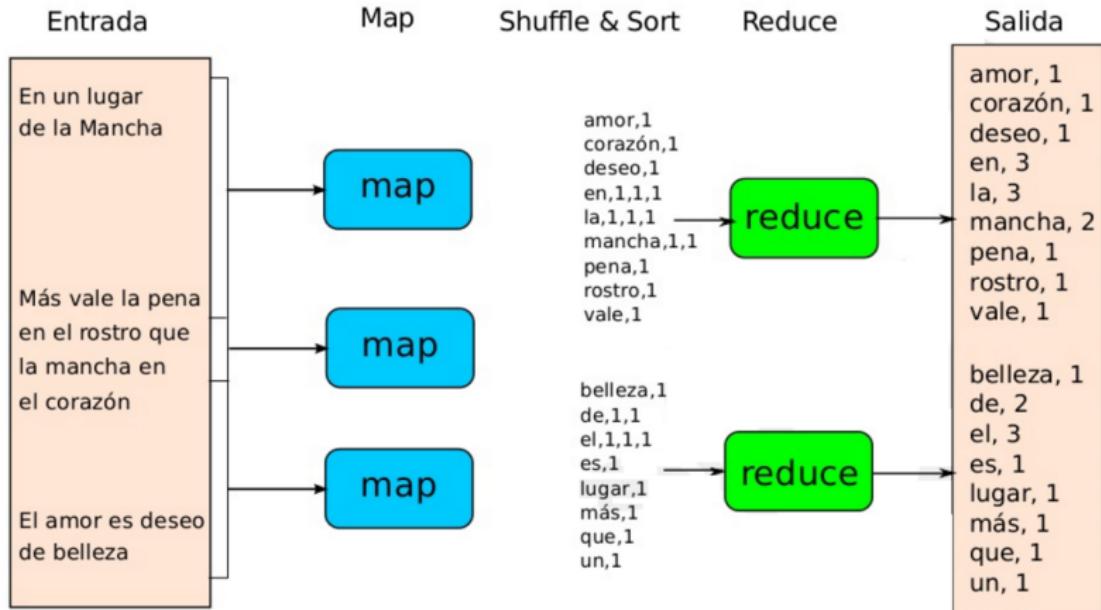
Ejemplo: Contar palabras



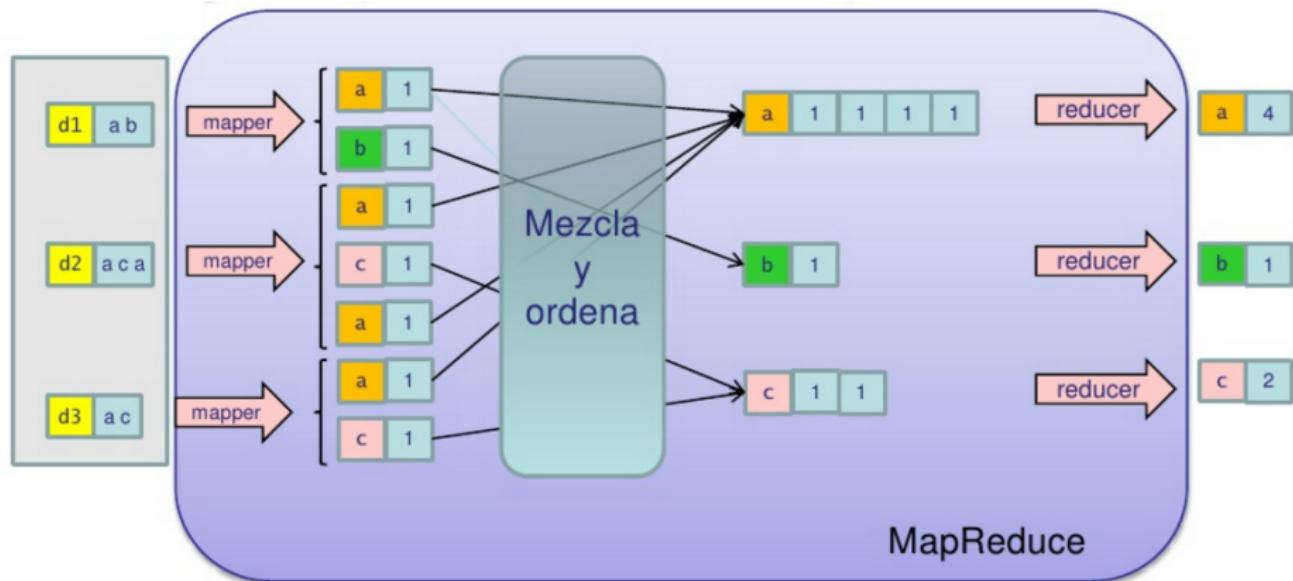
Ejemplo: Contar palabras



Ejemplo: Contar palabras



Ejemplo: Contar palabras



Ejemplo: grep

GREP distribuido

- Es la tarea de encontrar un patrón en un número muy grande de documentos
- Se puede utilizar, por ejemplo, para buscar en un conjunto de documentos, las líneas que coincidan con una expresión regular.

- Entradas: ficheros de texto

- Map (key, value):

```
# key: line number; value: text of line  
if match (value, pattern) emit (value, 1)
```

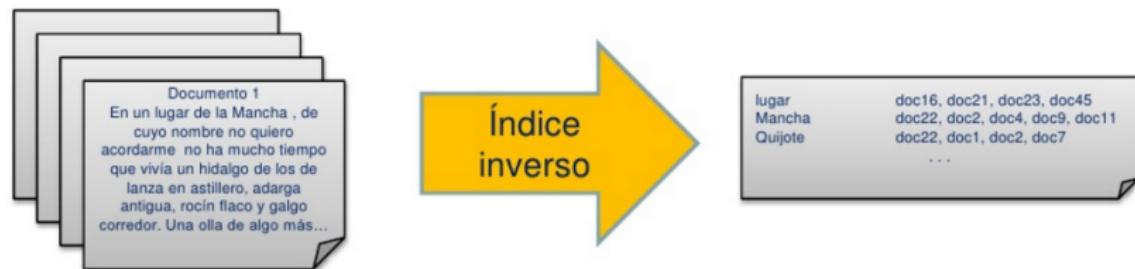
- Reduce:

```
# función identidad
```

- Salida: líneas que incluyen un patrón de búsqueda

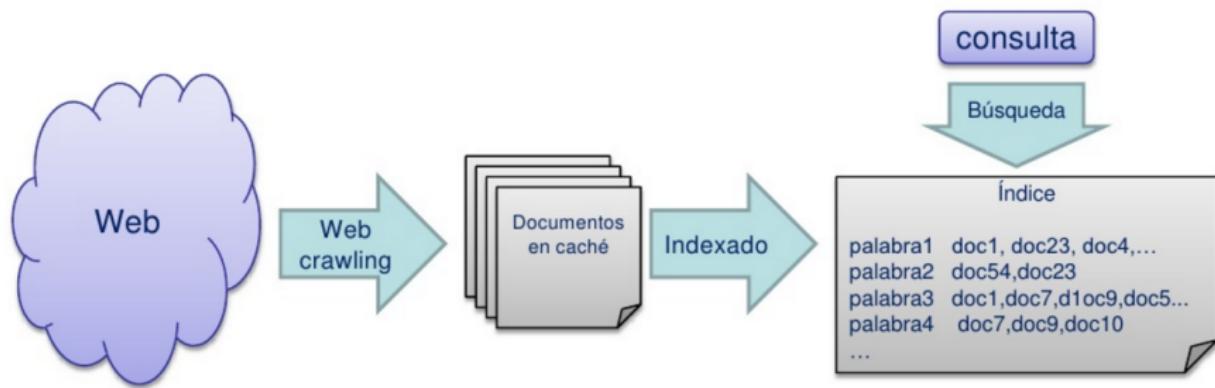
Ejemplo: Índices inversos

- Dada una serie de documentos, obtener la lista de palabras asociando a cada palabra el documento en el que aparece
- ordenar los documentos según el mayor número de apariciones



Ejemplo: Índices inversos

- Esquema básico de un buscador



Ejemplo: Índices inversos

- Entradas: pares (nombres de ficheros, texto)
- Mapper: devuelve cada palabra con su documento

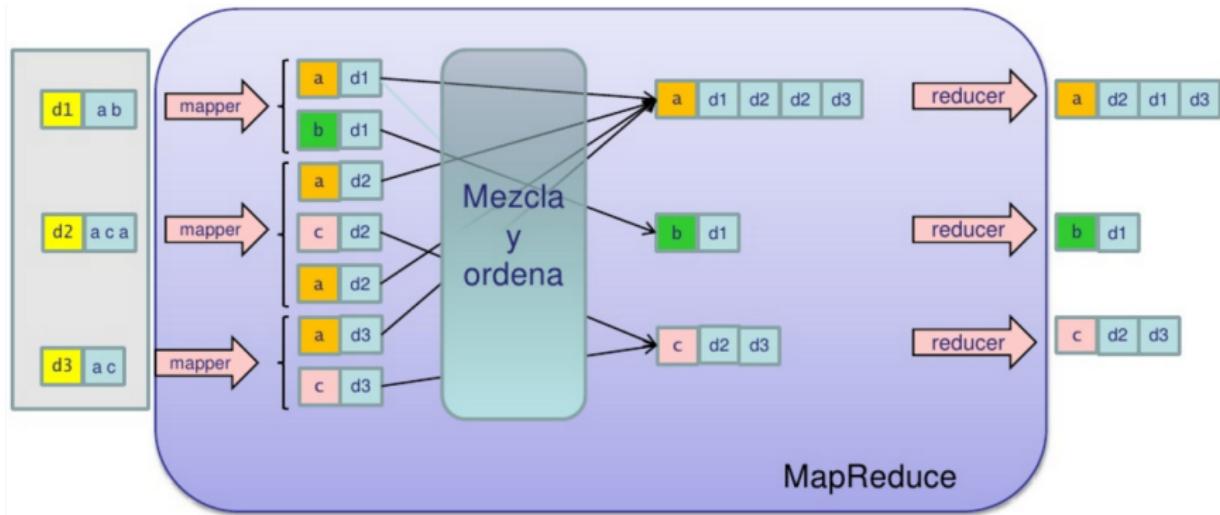
```
mapper(d,ps){  
    for each p in ps:  
        emit(p,d)  
}
```

- Reducer: ordena la lista de documentos por importancia

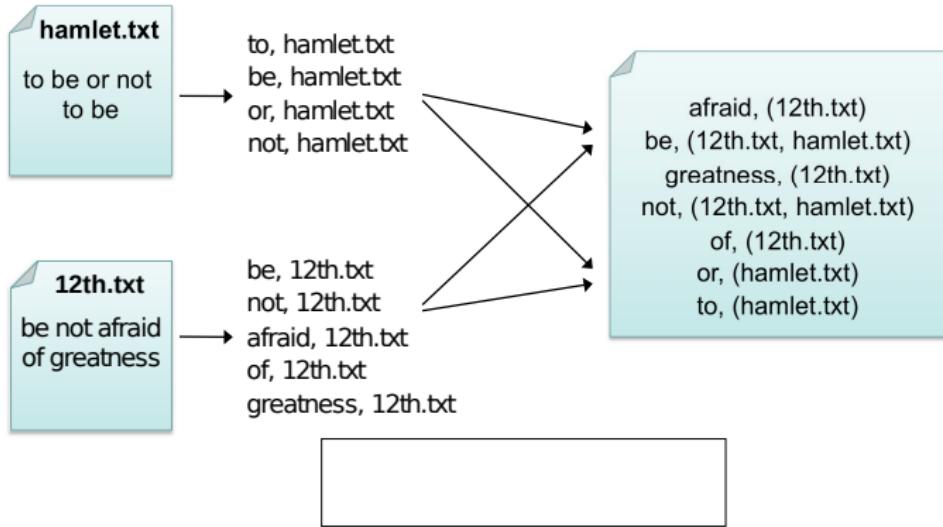
```
reducir(p,ds){  
    ds1=ordena(ds)  
    emit(p,ds1)  
}
```

- Salida: lista ordenada de ficheros que contienen cada palabra

Ejemplo: Índices inversos



Ejemplo: Índices inversos



Ejemplo: Canciones populares

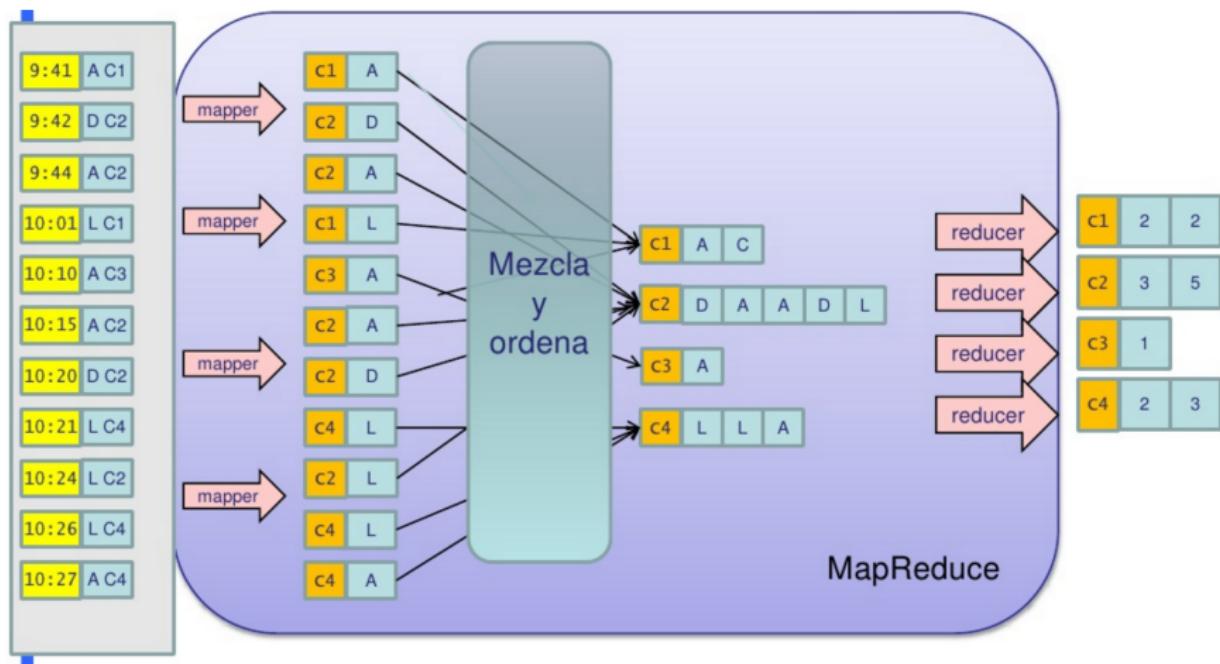
- A partir de los *logs* de los usuarios de un servidor de música, obtener el número de veces que se escucha cada canción (inspirado en *last.fm*)

```
2/3/2010 9:41 Ana C1
2/3/2010 9:42 Dani C2
2/3/2010 9:44 Ana C2
2/3/2010 10:01 Luis C1
2/3/2010 10:10 Ana C3
2/3/2010 10:15 Ana C2
2/3/2010 10:20 Dani C2
2/3/2010 10:21 Luis C4
2/3/2010 10:24 Luis C2
2/3/2010 10:26 Luis C4
2/3/2010 10:27 Ana C4
```



```
C1 2 oyentes, 2 escuchas
C2 3 oyentes, 5 escuchas
C3 2 oyentes, 2 escuchas
C4 2 oyentes, 3 escuchas
```

Ejemplo: Canciones populares



Map–Reduce

Cuando un usuario invoca una implementación de Map–Reduce, se ejecutan —por orden— las siguientes acciones:

1. Lectura de la entrada: la biblioteca de funciones Map–Reduce divide los ficheros de entrada en M bloques (de entre 16 y 64 Mb c.u.), e inicia muchas copias del programa en el clúster de máquinas.
 - una de dichas copias (la copia maestra) es diferente al resto; las otras son trabajadores (*workers*) que ejecutan las tareas asignadas por el maestro
 - existen M tareas *map* y R tareas *reduce*; el maestro elige nodos “ociosos” y les asigna tareas
 - los datos son leídos de un almacenamiento estable y se generan unos pares *clave / valor*
 - un ejemplo puede ser la lectura de un directorio completo, y la devolución de un registro por cada línea de cada fichero.

Map–Reduce

2. Cada función *map* recibe una serie de pares *clave / valor*, los procesa individualmente y devuelve otros pares *clave / valor*

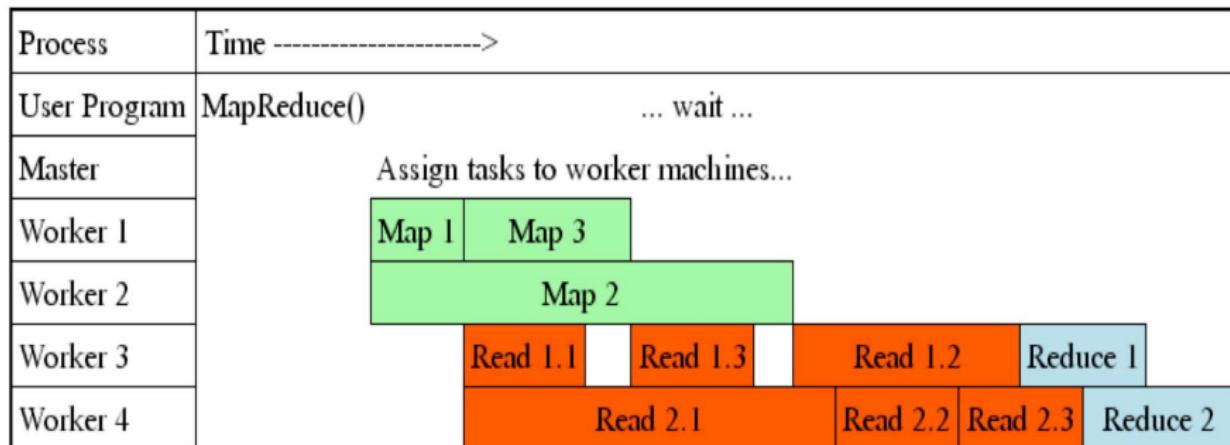
- cada trabajador al que se le asigna una tarea *map* lee los contenidos del subconjunto correspondiente, obtiene los pares *clave/valor* y los pasa a la función *map* definida por el usuario. Esta genera otros pares *clave/valor* intermedios
- los tipos de datos de la entrada y de la salida de la función *map* suelen ser distintos
- los pares intermedios se escriben periódicamente en disco, particionados en R regiones. Su localización se transmite de vuelta al maestro. Este ordena toda la información según las claves intermedias, para que las tareas de la misma clave vayan al mismo grupo

Map–Reduce

3. Las salidas de cada nodo *map* son asignadas a un nodo *reduce* concreto, en función del resultado obtenido por la aplicación de partición
4. La función *reduce* se llama una vez por cada clave distinta
 - el nodo *reduce* itera sobre la información intermedia ordenada, y pasa cada clave y el conjunto de valores intermedios a la función *reduce* (definida por el usuario), quien devuelve (o no) otro(s) valores
5. Cuando todas las tareas *reduce* se han completado, el nodo maestro devuelve el control al código del usuario.
La función de escritura escribe la salida del *reduce* en un almacenamiento estable

Ejecución: simultaneidad de tareas

- La fase reduce no puede comenzar hasta que se haya completado la fase map



Implementaciones

Existen multitud de implementaciones del modelo Map–Reduce; por ejemplo,

- el framework Map–Reduce de **GOOGLE**, implementado en C++, con interfaces en PYTHON y JAVA
- **HADOOP**, framework open–source desarrollado en JAVA; forma parte del proyecto APACHE. Acepta lenguajes de programación como PYTHON o C++.
- **GREENPLUM**, implementación comercial con soporte para PYTHON, PERL, SQL y otros lenguajes
- **PHOENIX**, implementación en memoria compartida, escrita en C
- MARS, para GPUs de NVIDIA empleando CUDA.

Hadoop

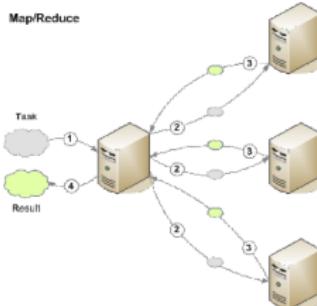
- Hadoop es un proyecto *open-source* de Apache de código abierto que proporciona soluciones para la computación distribuida fiable y escalable
- Está inspirado en los proyectos *Map–Reduce* y *Google File System* de GOOGLE
- Hadoop es un sistema de archivos distribuido y motor de procesamiento de datos que está diseñado para manejar altos volúmenes de datos en cualquier estructura
- Hadoop tiene dos componentes:
 - El sistema de archivos distribuido Hadoop (HDFS)
 - sistema de ficheros distribuido inspirado en GFS
 - estructura de directorios única para todo el cluster
 - implementa tolerancia a fallos replicando datos
 - El paradigma de programación MapReduce para la gestión de aplicaciones en varios servidores distribuidos
- Se caracteriza por: la redundancia, la arquitectura distribuida y el procesamiento paralelo
- Tres formas de programar los trabajos
 - Java API
 - Hadoop streaming (para Python, Perl, ...)
 - Pipes API (para C++)

Hadoop

Componentes Hadoop:

- **Hadoop Distributed File System (HDFS)**
 - Almacenamiento redundante masivo a través de un cluster básico
- **MapReduce**
 - **Map**: distribuye los problemas de cálculo en el cluster
 - **Reduce**: el nodo maestro recoge las respuestas a todos subproblemas y las combina.

Framework de programación para el desarrollo de aplicaciones y algoritmos



• Varias distribuciones:

- Elastic MapReduce (EMR) de Amazon Web Services (AWS).
- Cloudera
- Hortonworks Data Platform (HDP)
- MapR Technologies
- Teradata

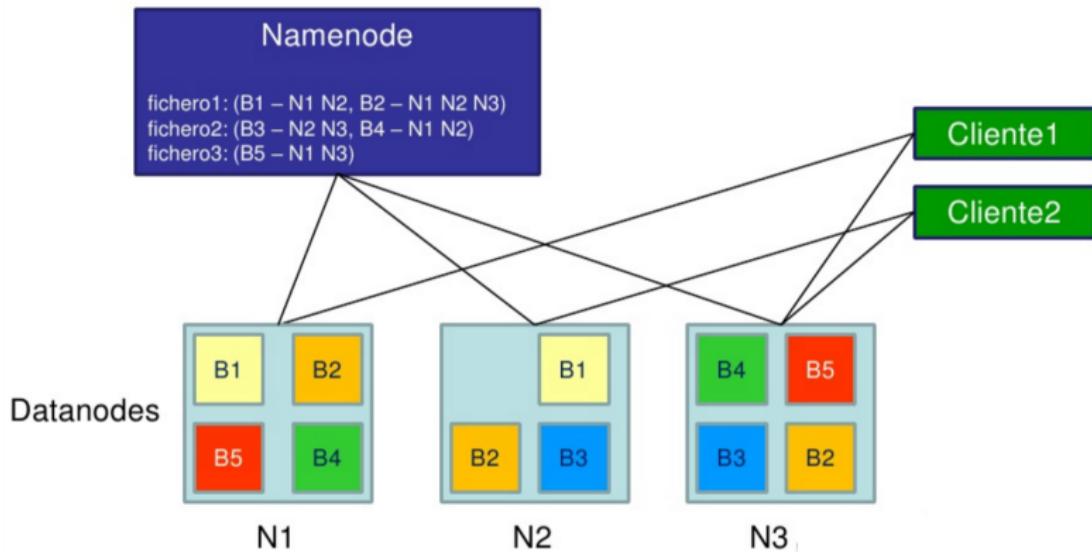
HDFS: sistema de archivos distribuido, escalable y portátil

- HDFS se compone de un NameNode y DataNode
- Un NameNode es un servidor maestro
- NameNode gestiona el espacio de nombres del sistema de ficheros y está vinculado a DataNode para gestionar el almacenamiento de datos
- Los ficheros se dividen en bloques de tamaño (por defecto, 64 MB) fijo, que distribuyen en uno o en varios DataNodes
 - El valor de replicación por defecto es $x3$.
 - Si los datos se almacenan en 3 nodos: dos en el mismo DataNode, y otro en otro DataNode distinto
- Los DataNodes realizan la creación, eliminación, y la replicación de los bloques, según las instrucciones procedentes del NameNode

Hadoop

HDFS: sistema de archivos distribuido, escalable y portátil

- Una desventaja importante es la pérdida de archivos debido a la falta de disponibilidad de una máquina
- **HDFS resuelve este problema mediante la replicación de cada bloque en diferentes DataNodes**



NameNode y DataNodes

- Cualquier ordenador que soporte Java puede ser NameNode o DataNode
- Una estructura típica es un único NameNode (esto simplifica la arquitectura del sistema)
 - Regula el acceso a los archivos de los usuarios
 - Ejecuta operaciones en el sistema de archivos como abrir, cerrar y renombrar ficheros y directorios
 - Determina la asignación de bloques en los DataNodes
 - Internamente, un archivo de dividido en uno o más bloques, los cuales se guardan en un conjunto de DataNodes
- Los otros ordenadores del clúster son DataNode (esto es lo general, aunque puede haber múltiples DataNodes en una misma máquina)
 - Son los responsables de leer y escribir las peticiones de los usuarios
 - Llevan a cabo la creación, eliminación y replicación de bloques, siguiendo las instrucciones de NameNode

Modos de ejecución:

- local (*standalone*): puede ejecutarse en modo no distribuido, como un proceso JAVA aislado. Configuración por defecto
- pseudo-distribuido: cada proceso de Hadoop se ejecuta en diferentes procesos Java, pero sobre la misma máquina
- distribuido: cada proceso de Hadoop se ejecuta en diferentes nodos de un clúster.

Código ejemplo de MapReduce de streaming de Python

Ejemplo: Contar palabras

- Usaremos la *Hadoop Streaming API* para ayudarnos a pasar datos entre Map y Reduce
- Se realiza vía `STDIN` (entrada estándar) y `STDOUT` (salida estándar)
- Se usa `sys.stdin` de Python para leer datos de entrada e imprimir la salida vía `sys.stdout`
- Los datos deben ser un par `clave-valor`, separados por un carácter de tabulación

Ejemplo disponible en: <http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/#python-mapreduce-code>

Código ejemplo de MapReduce de streaming de Python

Map: [mapper.py](#)

- Creamos el fichero `mapper.py`
- Lee los datos vía `STDIN`
- La salida es una lista de palabras sacada vía `STDOUT`
- En este ejemplo la salida mostrará: `palabra 1`. De este modo una palabra puede aparecer varias veces (este comportamiento puede modificarse)
- En este ejemplo el reductor se encarga de la suma final

Código ejemplo de MapReduce de streaming de Python

Map: `mapper.py`

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

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output).
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

- La siguiente línea es obligatoria, y sirve para indicar que tiene que usar hadoop para compilar el código

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

Código ejemplo de MapReduce de streaming de Python

Reduce: `reducer.py`

- El reducer leerá los resultados del `mapper.py` vía STDIN (la salida de `mapper.py` tiene que ser la entrada de `reducer.py`)
- El framework de hadoop garantiza que todos los valores asociados con la misma clave (palabra) van al mismo reducer
- Los resultados se muestran vía `STDOUT`

Código ejemplo de MapReduce de streaming de Python

Reduce: reducer.py

```
#!/usr/bin/env python
import sys
current_word = None
current_count = 0
word = None
# 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:
        # count was not a number, so silently ignore/discard this line continue
        # this IF-switch only works because Hadoop sorts map output
        # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Código ejemplo de MapReduce de streaming de Python

Testear el código

- Usaremos ebooks del Proyecto Gutenberg.
 - *The Outline of Science, Vol.1 (of 4) by J. Arthur Thomson*
<http://www.gutenberg.org/etext/20417>
 - *The Notebooks of Leonardo Da Vinci*
<http://www.gutenberg.org/etext/5000>
 - *Ulysses by James Joyce*
<http://www.gutenberg.org/etext/4300>
- Descargar cada ebook como ficheros de texto: Plain Text UTF-8
- Guardarlos en la carpeta libros

Código ejemplo de MapReduce de streaming de Python

Testear el código

- Es recomendable probar los códigos `mapper.py` y `reducer.py` localmente antes de usarlas en un trabajo MapReduce

```
$ echo "foo foo quux labs foo bar quux" | python mapper.py
foo      1
foo      1
quux    1
labs     1
foo      1
bar     1
quux    1

$ echo "foo foo quux labs foo bar quux" | python mapper.py
foo      1
foo      1
quux    1
labs     1
foo      1
bar     1
quux    1

## usando un ebook como entrada
$ cat libros/4300-0.txt | python mapper.py
Project 1
Gutenberg      1
Literary       1
Archive        1
Foundation,     1
...
...
```

Código ejemplo de MapReduce de streaming de Python

- Situamos la carpeta `libros` en el sistema de archivos distribuidos
`# hdfs dfs -put libros/`
- Comprobamos que la carpeta `libros` está en el sistema de archivos distribuidos
`# hdfs dfs -ls`
- Lanzamos el trabajo MapReduce:

```
/usr/bin/hadoop jar
/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.7.0.jar
-input libros
-output salida_libros
-mapper mapper.py
-reducer reducer.py
-file mapper.py
-file reducer.py
```

Código ejemplo de MapReduce de streaming de Python

- Vemos el contenido de la carpeta de salida

```
# hdfs dfs -ls salida_libros
```

- Ver los resultados

```
hdfs dfs -cat salida_libros/part-00000
```

- Ver los resultados ordenados:

```
# hdfs dfs -cat salida_libros/part-00000 | sort -k 2 -n
```

- Subir la carpeta de resultados:

```
# hdfs dfs -get salida_libros
```

Ejercicio 1: Buscar temperatura máxima y mínima

- Buscar el lugar en que hizo más calor y en el que hizo más frío en el año 2017.
Indicar el nombre de la ciudad junto con la temperatura.
- Usaremos datos de mediciones diarias proporcionados por la *NCDC (National Climate Center - NOAA)*
- Usando los del siguiente directorio:
<ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2017/>
- Se considera que hace calor si la temperatura es superior a $27^0 C$ y que hace frío si la temperatura es inferior a $-1^0 C$
- En el siguiente fichero se explica la organización de los ficheros:
<ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/README.txt>
- Realizar las ejercuciones forzando a que haya 2 o más `reducer`
- Se debe implementar un código, sencillo, para obtener el resultado final a partir de los ficheros generados vía los `reducers`.

Ejercicio 2: logs de un Web Client

- Usar los datos disponibles en moodle.
 - Los ficheros contienen los registros de las solicitudes HTTP
 - La organización de los datos se detalla en el documento:
estructura_datos_servidor_log.pdf
 - Usando los ficheros de las carpetas: condensed/272/Feb95
- ① Extraer el usuario que accedió a más ficheros en formato .ps. Mostrar usuario y número de ficheros a los que accedió (en formato .ps).
- ② Determinar la URL más visitada, indicando el número total de visitas recibidas.

Ejercicio 3: calidad del vino

- Usar el siguiente conjunto de datos:

<http://archive.ics.uci.edu/ml/datasets/Wine+Quality>

- Descargar los datos correspondientes a los dos tipos de vino: [winequality-white.csv](#) y [winequality-red.csv](#)

<http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/>

- Los atributos de las columnas son:

- ① acidez fija
- ② acidez volátil
- ③ ácido cítrico
- ④ azúcar residual
- ⑤ cloruros
- ⑥ dióxido de azufre libre
- ⑦ dióxido de azufre total
- ⑧ densidad
- ⑨ pH
- ⑩ sulfatos
- ⑪ alcohol
- ⑫ calidad

- Por cada tipo de vino, extraer la media de todos los atributos recogidos en los ficheros.

Bibliografía

- TOM WHITE, *Hadoop: The definitive guide*. 4th edition. O'Reilly, 2015
- MICHAEL G. NOLL, *Writing a Hadoop MapReduce Program in Python*.
[http://www.michael-noll.com/tutorials/
writing-an-hadoop-mapreduce-program-in-python/](http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/)
- A Guide to Python Frameworks for Hadoop: [https://blog.cloudera.com/
blog/2013/01/a-guide-to-python-frameworks-for-hadoop/](https://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/)
- A. RAJARAMAN, J. D. ULLMAN, *Mining of massive datasets*. Cambridge U. P., 2011
- A. LUENGO, *Desarrollo de un entorno basado en MapReduce para la ejecución distribuida de algoritmos genéricos paralelos*. Facultade de Informática, UDC, 2010

Docker Cloudera Quickstart

<https://medium.com/@SnazzyHam/>

how-to-get-up-and-running-with-clouderas-quickstart-docker-container-732c04ed0280

- Teclear:

```
docker pull cloudera/quickstart:latest  
desde la shell de ms2 (Windows) o desde bash (linux/mac)
```

- Lanzar Cloudera:

```
# docker run --hostname=quickstart.cloudera  
--privileged=true -t -i --cpus=4  
-v /home/user/practicas_ICS_Mapreduce:/home/cloudera/practicas  
--publish-all=true -p 7180 cloudera/quickstart  
/usr/bin/docker-quickstart
```