Computación distribuida

Sergio Nesmachnow
(sergion@fing.edu.uy)

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Contenido

1. Computación distribuida y computación cloud
2. Procesamiento de grandes volúmenes de datos
3. El modelo de computación Map-Reduce
4. El framework Hadoop y su ecosistema
5. Almacenamiento: HDFS y HBase
6. Aplicaciones de Map Reduce sobre Hadoop: conteo, índice invertido, filtros
7. Procesamiento de datos en tiempo real: Apache Spark
8. Ejemplos de aplicaciones en Spark y el lenguaje Scala
9. Análisis de datos utilizando Spark y el lenguaje R.
10. Aplicaciones iterativas: Google Pregel y Apache Giraph

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Apache Hadoop

Apache Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Apache Hadoop es un framework de software para el desarrollo y ejecución de aplicaciones distribuidas bajo una licencia libre.
- Permite a las aplicaciones trabajar con grandes infraestructuras de cómputo (miles de nodos) y grandes volúmenes (petabytes) de datos.
- Hadoop se inspiró en los documentos Google para MapReduce y Google File System (GFS).
- Hadoop fue desarrollado por el centro de investigación de Yahoo!, y es utilizado por una comunidad global de programadores y usuarios.
- Hadoop es la opción más conocida para implementar y ejecutar tareas MapReduce. Sin embargo, el framework también provee otras funcionalidades.

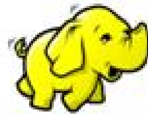
Apache Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Common: componentes e interfaces para sistemas distribuidos y entrada/salida (serialización, persistencia, RPC). Contiene los jars y scripts necesarios para la ejecución.
- Avro: sistema de serialización eficiente, RPC y persistencia de información.
- MapReduce: motor de ejecución de tareas MapReduce.
- Hadoop Distributed File System (HDFS): sistema de archivos distribuido de Hadoop.

Apache Hadoop: ecosistema

- Existe un gran conjunto de proyectos/frameworks basados en Hadoop



Ecosistema de Apache Hadoop

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

Hive
SQL Query

Hbase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

Apache Hadoop: frameworks

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Pig: lenguaje que permite ejecutar rutinas MapReduce de forma similar a sentencias en lenguaje SQL.
- Hive: datawarehouse distribuido inicialmente desarrollado por Facebook. Provee un lenguaje (HQL) muy similar a SQL que traduce en tiempo de ejecución a rutinas MapReduce (en general, utiliza varias rutinas encadenadas).
- Hadoop Database (HBase): base de datos distribuida orientada a columnas. Hbase utiliza HDFS y está basado en Google BigTable.
- Zookeeper: sistema de coordinación distribuido de alta disponibilidad.
- Sqoop: herramienta de migración de datos entre RDBMS y HDFS.

Hadoop: arquitectura

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El paquete de software Hadoop Common proporciona acceso a los sistemas de archivos soportados por Hadoop.
  - Contiene los archivos (.jar) y los scripts necesarios para ejecutar Hadoop, el código fuente, la documentación, y proyectos de la Comunidad Hadoop.
  - Requiere tener disponible SSH y JRE 1.6 o superior en los nodos.
- Las aplicaciones Hadoop utilizan la información sobre la ubicación de cada sistema de archivos para ejecutar trabajos en el mismo nodo o en el mismo switch donde están los datos, reduciendo así el tráfico de red.
- El sistema de archivos usa la información para replicación, conservando copias diferentes de los datos (réplicas) y mejorar la confiabilidad.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

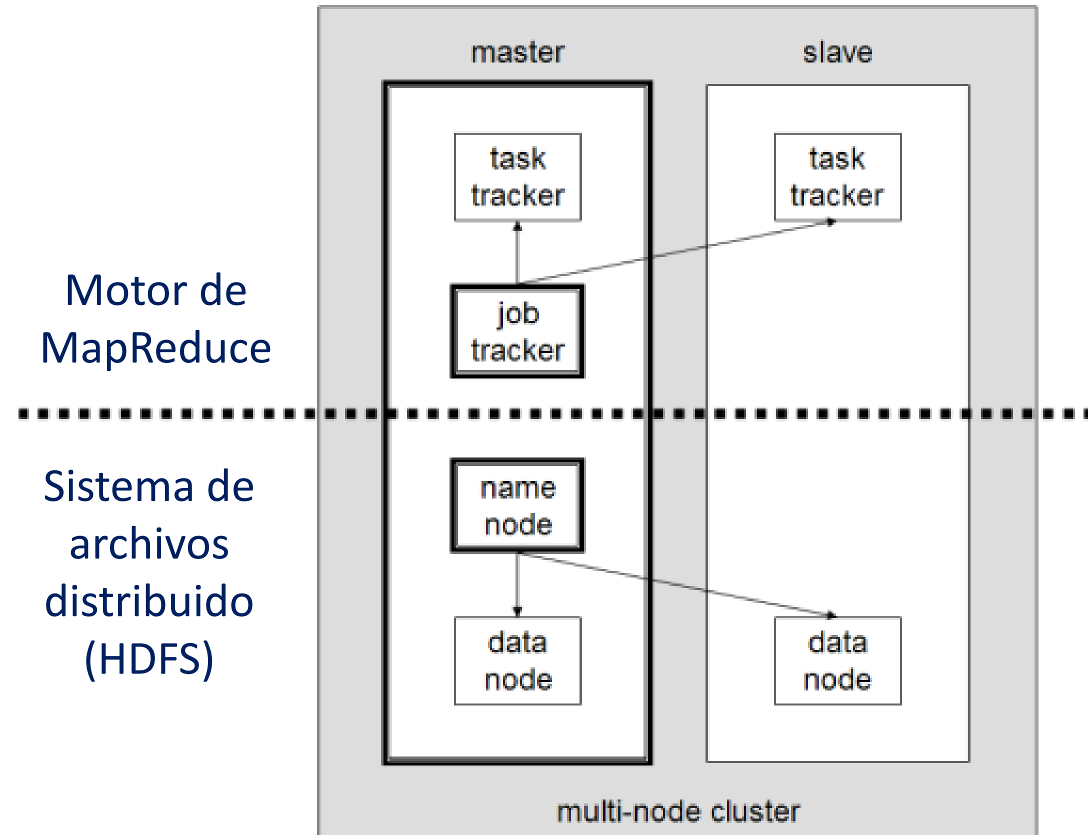
Hadoop: arquitectura

Hadoop: arquitectura

- Componentes (clusters virtuales para almacenamiento de datos y procesamiento distribuido):
  - Sistema de archivos distribuido
  - Motor de MapReduce: cluster de nodos, con master y slaves.
- Hadoop v1: manejo de recursos en los propios nodos de procesamiento.
- Hadoop v2: desacopla manejo de recursos de procesamiento de datos. Permite ejecutar otros tipos de aplicaciones (no solamente MapReduce).

Hadoop v1: arquitectura

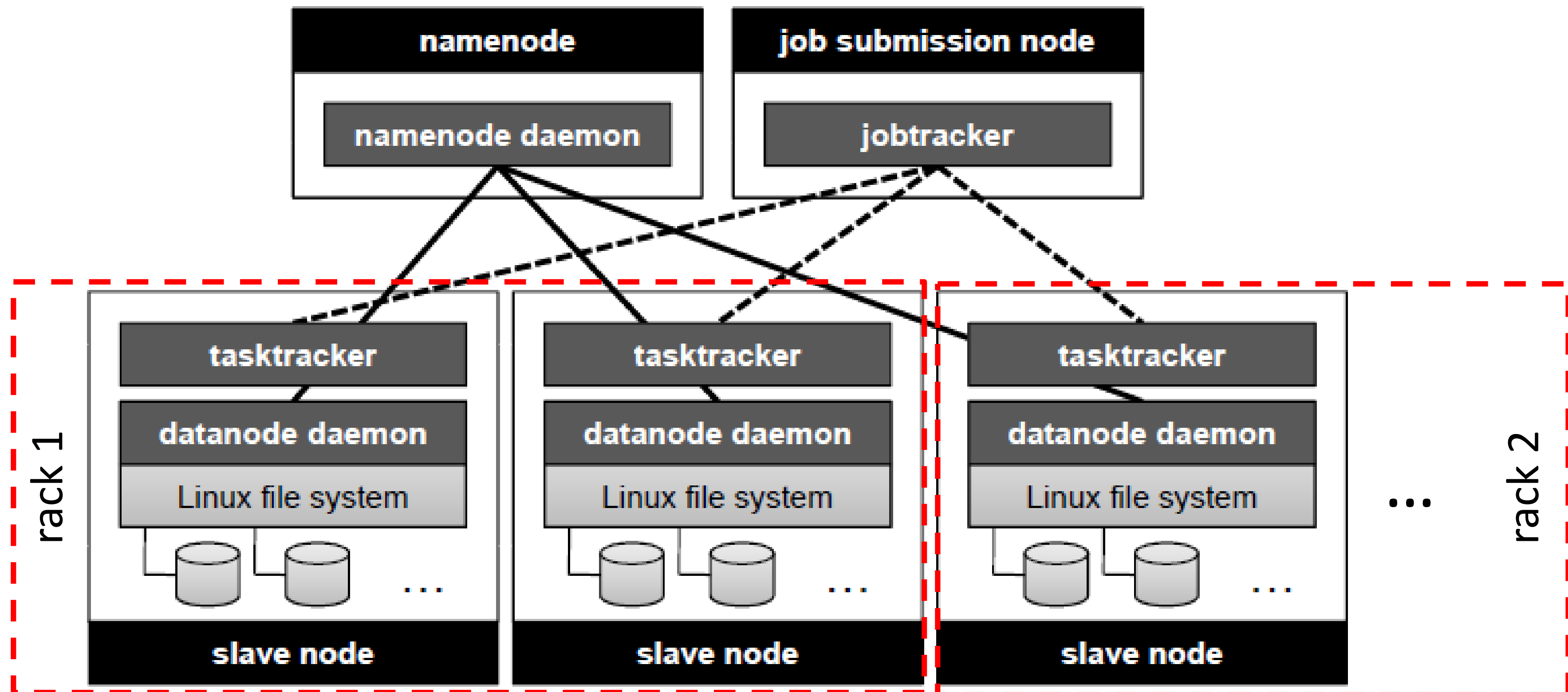
- En Hadoop v1 un cluster Hadoop típico incluye un nodo maestro y múltiples nodos esclavos.
- El nodo maestro se compone de un JobTracker (rastreador de trabajos), un TaskTracker (rastreador de tareas), un NameNode (nodo de nombres) y un DataNode (nodo de datos). Un esclavo (nodo de cómputo) está formado por un DataNode y un TaskTracker.



Hadoop v1: arquitectura

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Componentes (clusters virtuales para almacenamiento y procesamiento):
  1. Sistema de archivos distribuido (hasta un DataNode por nodo).
  2. Motor de MapReduce (un TaskTracker por nodo, todos subordinados a un único JobTracker).

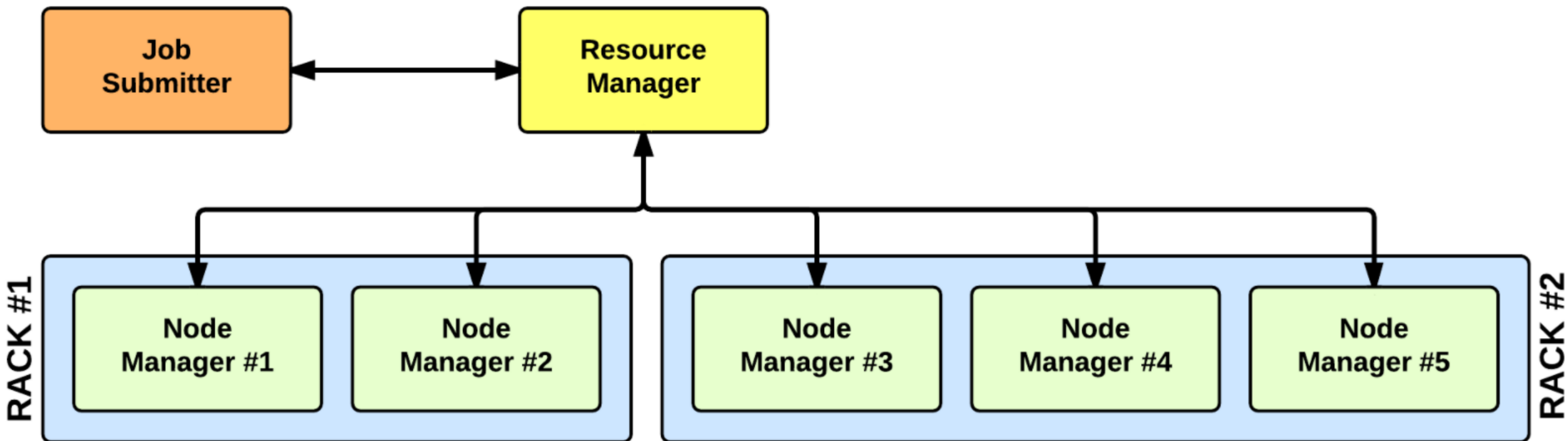


Hadoop v2: arquitectura

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En Hadoop v2 se desacopla el manejo de recursos del cómputo propiamente dicho. Se dividen las funciones del JobTracker en componentes separados.

- Entidades:
  - Job Submitter (el cliente)
  - Resource Manager (el master)
  - Node Manager (el slave)



Hadoop: protocolo de comunicaciones

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En Hadoop, todas las comunicaciones entre procesos se realizan utilizando una implementación de RPC personalizada para el framework:
  - Simple de modificar y extender.
  - Está definida a través de interfaces Java.
  - Las interfaces están implementadas a través de server object.
  - Client proxy objects son automáticamente creados.
- Todos los mensajes se originan en el cliente
  - De este modo se previenen ciclos y posibles deadlocks en las comunicaciones.

Hadoop: sistema de archivos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop Distributed File System (HDFS) es un sistema de archivos distribuido, escalable y portátil, escrito en Java, para Hadoop.
- Cada nodo en una instancia Hadoop típicamente tiene un único (o ningún) nodo de datos; un cluster de datos forma el cluster HDFS.
- Cada nodo sirve bloques de datos sobre la red, usando un protocolo de bloqueo específico para HDFS.
- El sistema de archivos usa TCP/IP para la comunicación, mientras que los clientes usan RPC para comunicarse entre ellos.
- HDFS almacena archivos grandes a través de múltiples nodos (hosts): el tamaño de particionamiento (*split*) en Hadoop HDFS es de 128 MB.
- La confiabilidad se implementa mediante replicación de datos en múltiples hosts (no se requiere almacenamiento RAID en ellos). Con el valor de replicación por defecto (3), los datos se almacenan en tres nodos: dos en el mismo rack, y otro en un rack distinto.

Hadoop: sistema de archivos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Existe un protocolo propietario entre nodos para realizar tareas específicas de HDFS como reequilibrar datos, mover copias, y conservar alta la replicación de datos.
- HDFS es portable entre varias plataformas, pero no cumple estrictamente con el estándar POSIX para sistemas de archivos.
- HDFS fue diseñado para proveer máxima eficacia y rendimiento en aplicaciones MapReduce desarrolladas sobre Hadoop y para gestionar archivos muy grandes.
- HDFS tradicional no proporciona alta disponibilidad (HDFS-HA está disponible en Hadoop 2).
- Aunque HDFS es el sistema de archivos nativo de Hadoop, el framework también soporta otros sistemas de archivos para cloud computing, como Amazon S3, CloudStore, FTP, HTTP(S).

HDFS: funcionalidades

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. Tolerancia a fallos. El mecanismo de replicación proporciona un alto nivel de tolerancia a fallos utilizando almacenamiento redundante configurable.
2. Acceso eficiente a los datos. HDFS prove un gran ancho de banda para que las aplicaciones MapReduce desarrolladas sobre Hadoop puedan procesar grandes volúmenes de datos. El ancho de banda es más importante que la latencia de acceso a los datos, ya que en general el modelo de computación se aplica en procesos batch (fuera de línea) y no en tiempo real.
3. Modelo de coherencia simple. HDFS almacena los datos según el modelo write-once-read-many, soportando aplicaciones Hadoop donde los datos de entrada se escriben una vez y se leen tantas veces como sea necesario.

HDFS: funcionalidades

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

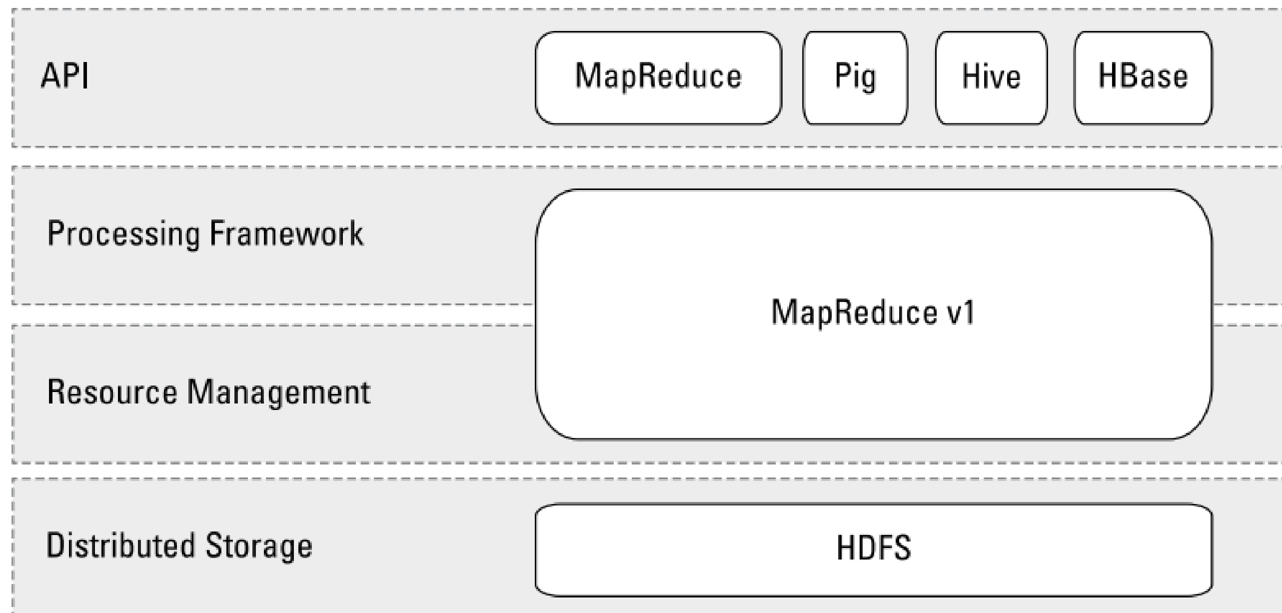
4. Conviene mover la computación y no los datos. Hadoop sigue la idea de que es menos costoso mover las aplicaciones/algoritmos que mover los datos. Esta idea es válida en general al considerar juegos de datos muy grandes. HDFS provee interfaces especializadas en este sentido.
5. Portabilidad entre hardware y software heterogeneo. HDFS es portable entre distintas plataformas (Java compatibles).

SE PRESENTARÁN LAS CARACTERÍSTICAS DE HDFS EN EL TEMA 5

Hadoop v1: gestión y administración

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un único servicio master (JobTracker) y varios TaskTrackers (uno por nodo del cluster).
- Los TaskTrackers manejan slots de recursos en cada nodo slave para ejecutar tareas Map y Reduce.



Hadoop v1: el motor MapReduce

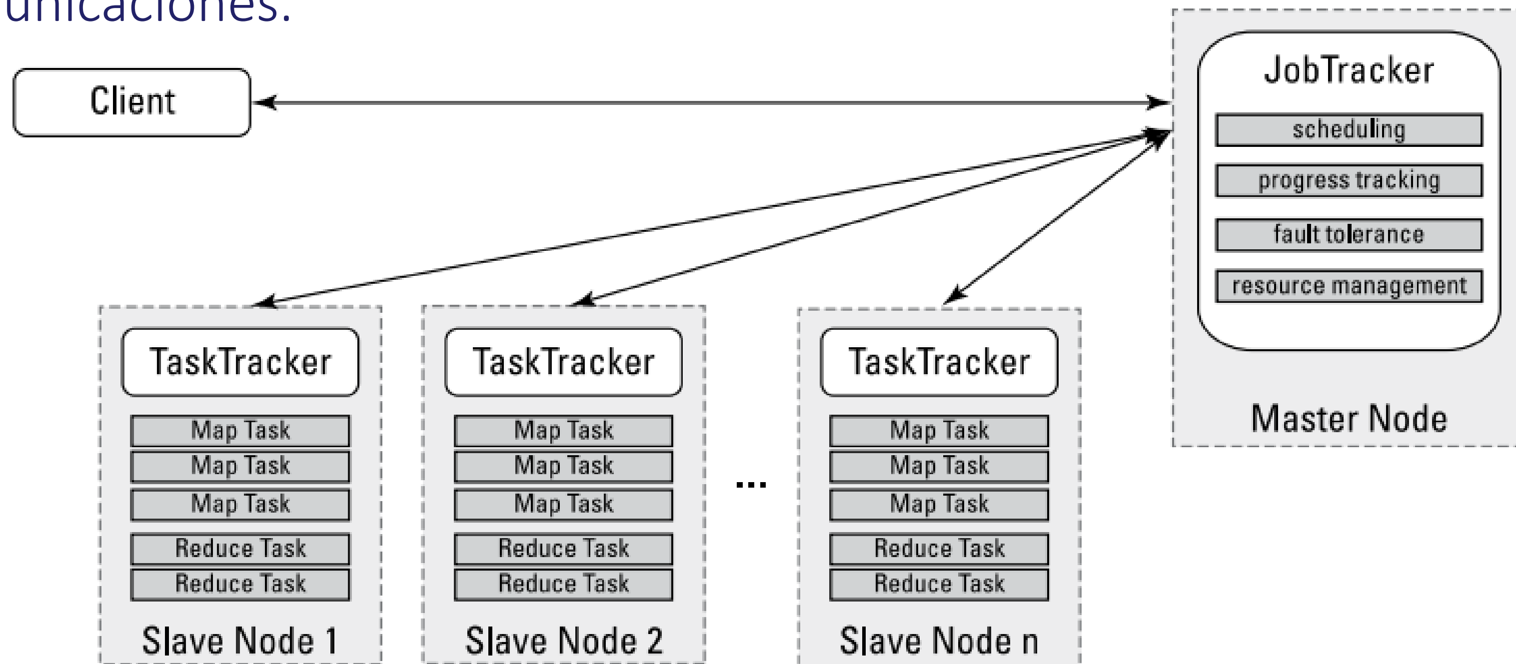
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El motor MapReduce consiste en un JobTracker (rastreador de trabajos), al cual las aplicaciones cliente envían trabajos MapReduce.
- El JobTracker envía tareas a nodos TaskTracker disponibles en el cluster, intentando mantener cercanía con los datos a procesar (nodo/rack/etc.).
- El TaskTracker genera en cada nodo un proceso JVM separado, para evitar que el propio TaskTracker falle si el trabajo en cuestión tiene problemas.
- El TaskTracker envía información periódicamente al JobTracker para comprobar su estado (heartbeats).
- Si un Task Tracker falla o su respuesta no llega, la tarea se reprograma y se envía a otro nodo.
- Hadoop versión 0.21 añadió checkpoints en el TaskTracker para no perder un trabajo en curso. Al iniciar, el JobTracker busca datos para recomenzar un trabajo desde un checkpoint previo.

Hadoop v1: gestión y administración

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Los slots son definidos en base a los recursos.
- Los slots para tareas Map y Reduce son distintos:
  - Los mappers son asignados basándose en localidad de datos, dependen de I/O de disco y CPU.
  - Los reducers son asignados según disponibilidad y enfocándose en las comunicaciones.



Hadoop v1: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Las principales limitaciones de la arquitectura utilizada en el motor MapReduce de Hadoop v1 son:
  - La asignación de trabajos por parte de de los JobTrackers es muy sencilla. Cada TaskTracker tiene un número de plazas disponibles (ranuras o slots). El JobTracker asigna las tareas Map o Reduce al TaskTracker más cercano a los datos con un slot disponible. No hay ninguna consideración de la carga activa actual de la máquina asignada, y por tanto de su disponibilidad real.
  - Si un TaskTracker es lento (no responde, o demora en responder) se puede retrasar toda la aplicación MapReduce, especialmente hacia el final de un trabajo, donde varias tareas pueden estar a la espera de una sola tarea lenta. Hadoop proporciona una opción de *ejecución especulativa* (paralelismo optimista) que al activarse permite a una tarea simple ejecutar en múltiples nodos de cómputo.

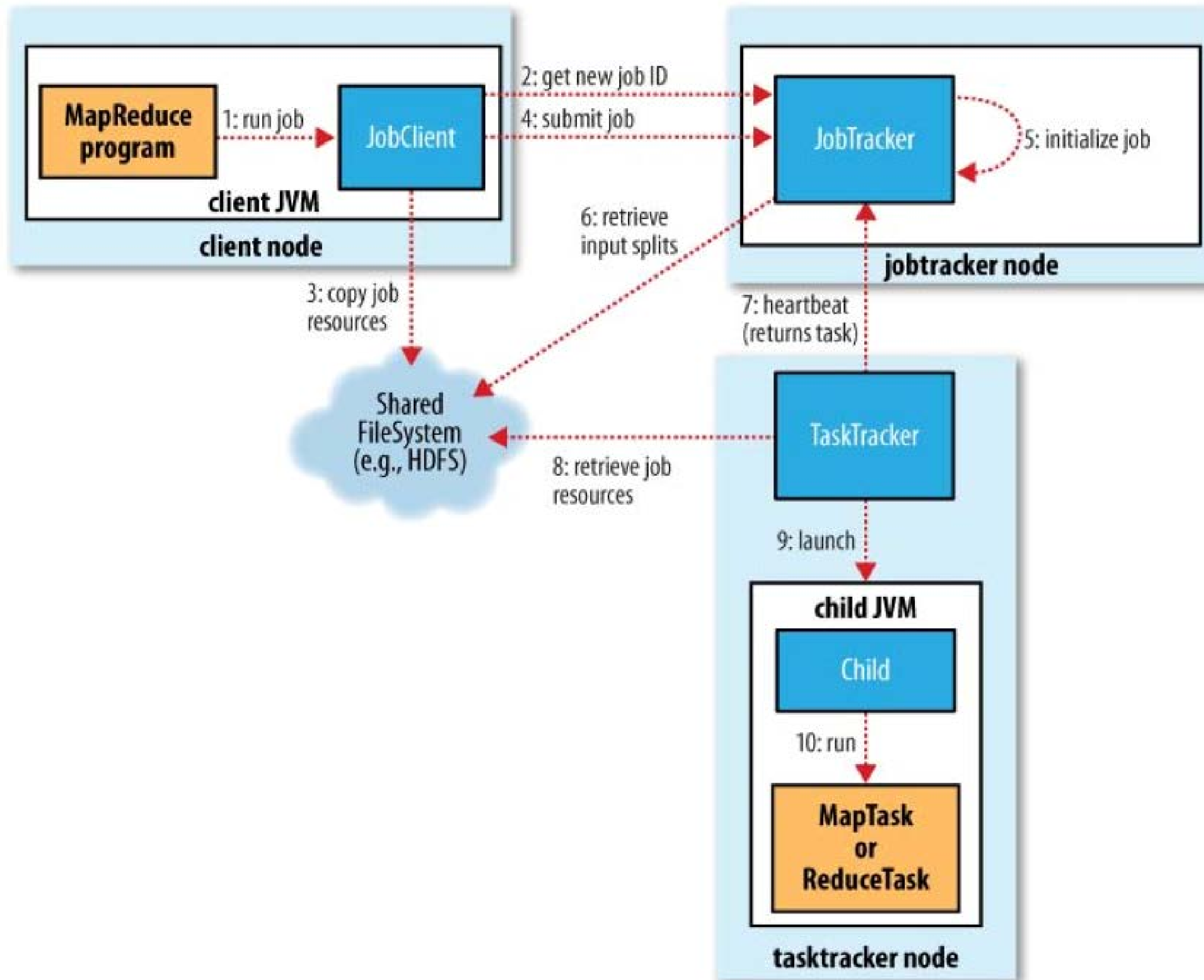
HADOOP: ejecución de una aplicación MapReduce (versión 1)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. La aplicación cliente envía una solicitud al JobTracker.
2. El JobTracker determina los recursos necesarios examinando los archivos que la aplicación necesitará del NameNode y calculando el número de tareas Map y Reduce.
3. El JobTracker encola las tareas Map y Reduce, tomando en cuenta el estado de los nodos.
4. Las tareas Map se inician cuando hay recursos disponibles. Mappers asignados a bloques específicos se asignan a nodos donde esos datos están almacenados.
5. El JobTracker monitorea el progreso de las tareas y reejecuta en caso de fallos. Luego de alcanzado el número máximo de intentos, el trabajo falla.
6. Luego de finalizar los Mappers, los Reducers procesan las salidas intermedias.
7. Los conjuntos de resultado son retornados a la aplicación cliente.

Hadoop v1: el motor MapReduce

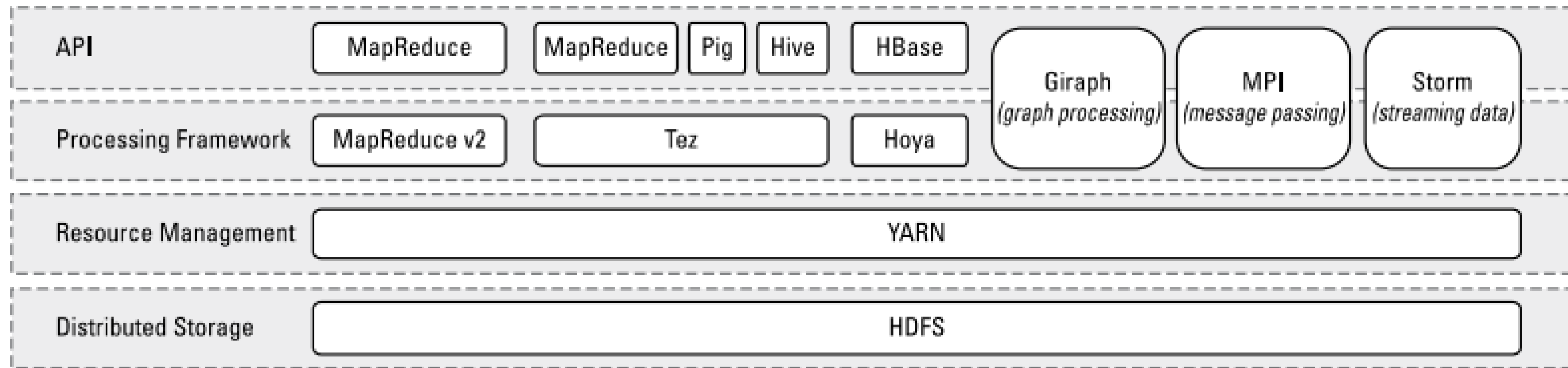
UNIVERSIDAD DE LA REPÚBLICA URUGUAY



Hadoop v2: gestión y administración

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

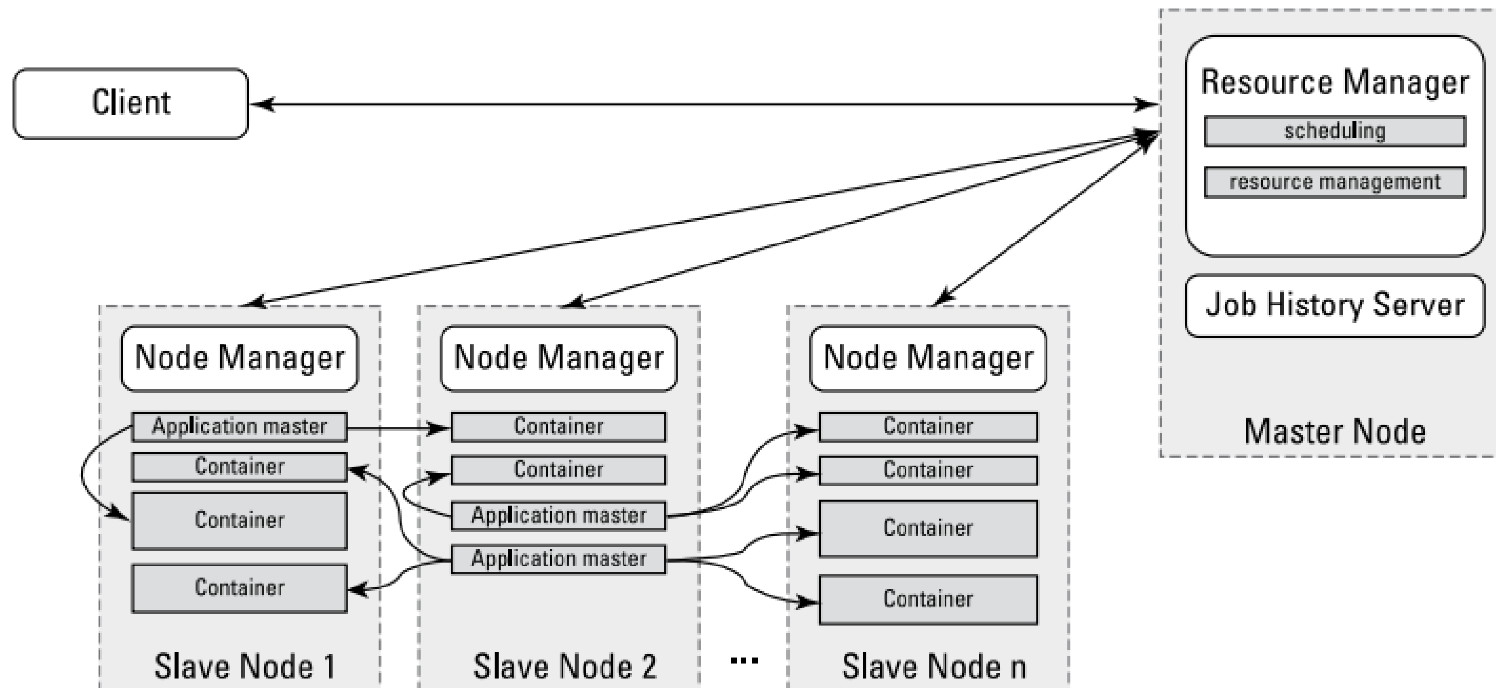
- YARN: sistema de administración de recursos, prove servicios de gestión y planificación para ejecutar otros tipos de tareas en un cluster Hadoop.



- La principal diferencia con la version 1 es que trabaja con contenedores (containers), unidades genéricas de recursos para ejecutar aplicaciones.
- A diferencia de los slots de JobTracker/TaskTracker los containers pueden ejecutar cualquier tipo de aplicaciones y no solo tareas Map y Reduce.
- Los containers pueden requerirse y dimensionarse con cualquier número de recursos (no como los slots, que son uniformes).

Hadoop v2: arquitectura

- Componentes: Resource Manager (scheduler, orquestador, similar a JobTracker), Node Manager (similar a TaskTracker), Application Master y Job History Server (ambos sin análogos en version 1).
- El Application Master maneja recursos y heartbeats para cada aplicación.



Hadoop v2: el motor MapReduce

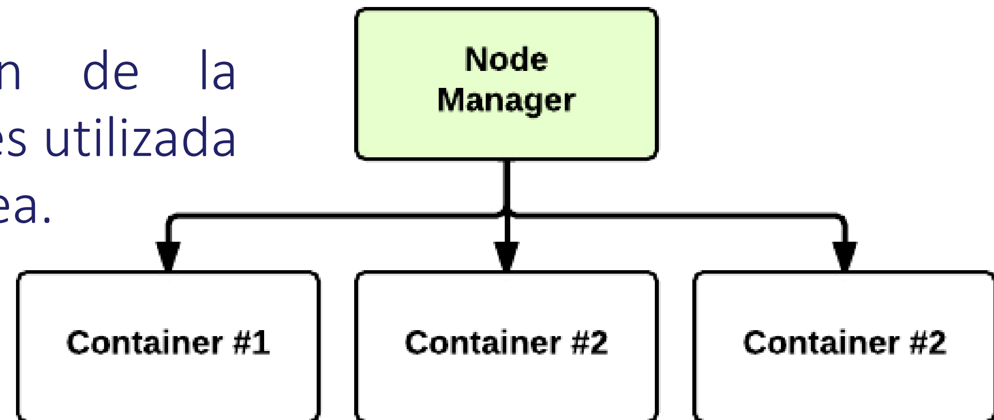
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El Resource Manager es el master del motor.
  - Existe un Resource Manager por cluster Hadoop.
  - Conoce la ubicación de los esclavos (rack awareness) y cuántos recursos dispone cada uno.
  - Ejecuta varios servicios, incluyendo el scheduler, los monitores de nodos y manejadores de eventos.
- El Resource Manager es el púnto único de falla.
  - Debe ejecutar en un nodo dedicado.
  - Utilizando Application Masters, YARN distribuye en el cluster la información y las tareas relacionadas con la ejecución de aplicaciones. De este modo se reduce la carga del Resource Manager y hace más simple su recuperación en caso de fallos.

Hadoop v2: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El Node Manager es el esclavo de la infraestructura.
  - Existen muchos Node Managers por cluster.
  - Cuando inicia se anuncia al Resource Manager, al cual le envía heartbeats periódicamente.
  - Cada Node Manager ofrece recursos al cluster. La capacidad del Node Manager queda dada por la cantidad de memoria y el número de cores virtuales que ofrece.
  - En tiempo de ejecución, el scheduler utiliza la información sobre la capacidad para asignar tareas.
  - Utiliza containers: una fracción de la capacidad del Node Manager que es utilizada por el cliente para ejecutar una tarea.



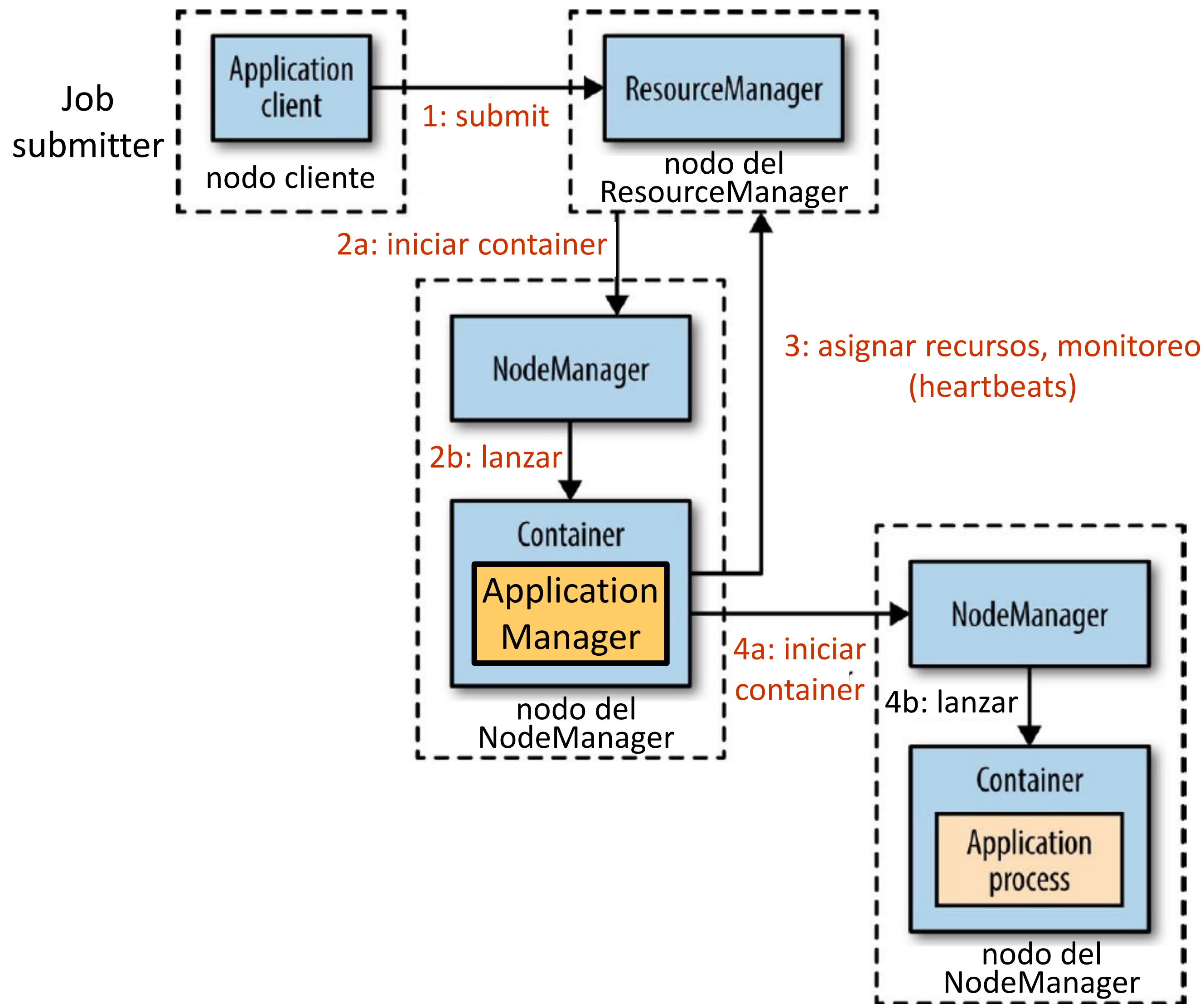
Hadoop v2: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El Application Master es responsable de la ejecución de cada aplicación.
  - Solicita containers al Resource Scheduler (en Resource Manager) y ejecuta programas específicos (main de clases Java) en los containers obtenidos.
  - Conoce la lógica de la aplicación, por lo cual es específico del framework que se ejecuta. El framework MapReduce prove su propia implementación de un Application Master.
- El Job Submitter envía trabajos al cluster.
  - El método submit crea una instancia de JobSubmitter y envía un job.
  - Solicita un application ID al Resource Manager, chequea directorios de entrada y de salida (que no debe existir), computa las particiones de entrada (input splits), copia los recursos necesarios para ejecutar el job al filesystem distribuido: jar (con muy alto factor de replicación), configuración del job, input splits.
  - Envía el job invocando submitApplication() en el Resource Manager.
  - waitForCompletion() chequea el progreso del trabajo (polling cada 1s).

Hadoop v2: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY



Hadoop v2: ejecución de una aplicación MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. Una aplicación cliente (Job Submitter) envía una solicitud al Resource Manager.
2. El Resource Manager solicita a un Node Manager crear una instancia de un Application Master. El Node Manager obtiene un container para la aplicación y la inicia.
3. El nuevo Application Master se inicializa y se registra con el Resource Manager.
4. El Application Master determina los recursos necesarios, examinando los archivos que la aplicación necesitará del NameNode y calculando el número de tareas Map y Reduce, y los solicita al Resource Manager. El Application Master envía mensajes (heartbeat) al Resource Manager con una lista de recursos requeridos y sus modificaciones (por ejemplo, una solicitud de fin).
5. El Resource Manager acepta el pedido y lo encola junto con otros pedidos.
6. Cuando se identifican recursos disponibles en los nodos slave, el Resource Manager otorga al Application Master (containers) en los nodos slave. El AM solicita el container (con todos sus detalles) al NM, que lo crea y lo inicia.

Hadoop v2: ejecución de una aplicación MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

7. La aplicación ejecuta en el container y el Application Master monitorea su progreso. Si se detecta un error del container se reinicia la ejecución en otro container (hasta un número máximo de reintentos). El Application Master también puede enviar una señal de fin al Node Manager para terminar un container (por un cambio de prioridades o porque la aplicación finalizó).
8. Para una aplicación MapReduce, luego que finalizan las tareas map, el Application Master solicita recursos para ejecutar las tareas reduce para procesar los resultados intermedios.
9. Cuando todas las tareas finalizan, el Application Master envía los resultados a la aplicación cliente, informa al Resource Manager que la aplicación finalizó con éxito, se desregistra del Resource Manager y finaliza su ejecución.

Hadoop: tolerancia a fallos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El mecanismo de tolerancia a fallos en Hadoop se basa en reasignar tareas cuando una ejecución particular falla.
  - Cuando el Application Master recibe una notificación de fallo de una tarea la reasigna. Intenta evitar el uso de un Node Manager que falló previamente.
  - El máximo número de intentos de ejecución de una tarea se indica con mapreduce.map.maxattempts y mapreduce.reduce.maxattempts.
  - Algunas aplicaciones pueden no abortar si unas pocas tareas fallan. El porcentaje de tareas que pueden fallar sin abortar se indica por mapreduce.map.failures.maxpercent y mapreduce.reduce.failures.maxpercent.
  - Las tareas pueden ser matadas por ejecución especulativa o porque un Node Manager fue marcado por fallos. Tareas matadas no cuentan en los intentos.
- El esquema funciona correctamente en cluster locales, pero no es adecuado para grandes sistemas distribuidos
  - El mecanismo de reasignar trabajos puede ser muy ineficiente.
  - No es aplicable para entornos pervasivos, dinámicos y/o voluntarios.

Hadoop: tolerancia a fallos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Fallos del Application Master
  - Se reintentan (propiedad mapreduce.am.max-attempts), por defecto es 2.
  - Debe modificarse junto a yarn.resourcemanager.am.max-attempts.
- Fallos del Node Manager
  - Por caída o por ejecución muy lenta (no envía heartbeat en 10 minutos): se elimina del pool de recursos para asignar. Tareas y Application Master ejecutando en ese nodo se recuperan como se explicó.
  - Maps que corresponden a jobs que no finalizaron pueden ser reejecutados (la salida intermedia puede no ser accessible a los nuevos reducers).
  - Node Managers pueden ser colocados en una lista negra (por parte del Application Master) si tienen fallos muy frecuentes.
- Fallos del Resource Manager
  - Serio, no se pueden ejecutar jobs containers.
  - Replicación es necesaria para lograr alta disponibilidad: un RM secundario y datos en un sistema confiable (ZooKeeper, HDFS), para recuperación.

Hadoop: HDFS + MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El framework Hadoop trabaja sobre el sistema de archivos HDFS y el motor de MapReduce, a los que considera como 'clusters virtuales' para la ejecución de aplicaciones.

- HDFS y Map-Reduce están acoplados, y fueron diseñados para trabajar conjuntamente.
  - HDFS para manejar datos (se almacenan y acceden como archivos y directorios).
  - Map-Reduce para la ejecución de trabajos y el scheduling.

- De todas formas, nada impide que una tarea MapReduce en particular consuma información de otras fuentes (ej: Hbase, REST API, etc).

Hadoop: modos, instalación y configuración

Modos de Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop propone tres modos de ejecución:
  1. Local (standalone)
     - Útil para paqueños testeos y debug de aplicaciones.
  2. Local (seudo-distribuida)
     - Las tareas se ejecutan en diferentes máquinas virtuales.
     - Se configura (casi) del mismo modo que una instalación completa en modo cluster.
  3. Totalmente distribuida (cluster)
     - Las máquinas del cluster se configuran individualmente.
     - Se utiliza una topología master-slave para los nodos que proveen recursos y almacenamiento.

Hadoop: modo local (standalone)

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En el modo standalone, una configuración mínima es requerida.

- Instalación:
  - Descargar Hadoop de http://hadoop.apache.org
  - Descomprimir y editar el archivo conf/hadoop-env.sh
  - Definir la variable *JAVA_HOME* al path del instalador de Java

- ¿ Cómo verificar si funciona ?
  - Intentar ejecutar $HADOOP_PATH/bin/hadoop
  - Por defecto mostrará los parámetros a utilizar por parte de Hadoop.

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En el modo seudodistribuido todos los servicios de Hadoop, incluidos los servicios que manejan el nodo maestro y los nodos esclavos, se ejecutan en un solo nodo de cálculo.
- Útil como entorno de desarrollo, para verificar y analizar rápidamente aplicaciones y para enseñanza.

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Instalación

- Crear un usuario que no sea root para hadoop

```
adduser hadoop
passwd hadoop
```

- Crear un par de claves para ese usuario

```
su - hadoop
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Instalación

- Descargar la versión de Hadoop de interés (para nuestro curso se descarga Hadoop 2.6.5).
- Descomprimir en el directorio de instalación (en el ejemplo se descomprime en el directorio */home/hadoop/hadoop*)

```
cd ~
wget http://www-eu.apache.org/dist/hadoop/common/hadoop-2.6.5/hadoop-2.6.5.tar.gz
tar xzf hadoop-2.6.5.tar.gz
mv hadoop-2.6.5 hadoop
```

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración

- Configurar las variables de ambiente para Hadoop en el archivo **~/.bashrc**

```
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Directorio donde se descomprimió hadoop

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración

- Configurar la variable JAVA_HOME en el archivo $HADOOP_HOME/etc/hadoop/hadoop-env.sh.
- Por ejemplo, incluir en el archivo la siguiente línea:

```
export JAVA_HOME=/opt/jdk1.8.0_131/
```

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración

- Configurar datos en el archivo **$HADOOP_HOME/etc/hadoop/core-site.xml.**
- Se configura la propiedad que indica el esquema (HDFS), la autoridad (localhost) y el puerto (9000):

```
<configuration>
<property>
  <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración

- Configurar el archivo **$HADOOP_HOME/etc/hadoop/hdfs-site.xml**

```
<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value>
</property>

<property>
  <name>dfs.name.dir</name>
   <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>

<property>
  <name>dfs.data.dir</name>
   <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
</property>
</configuration>
```

Factor de replicación en 1. No es necesario que HDFS replique bloques de archivos

Directorios utilizados por los diversos componentes de HDFS para almacenar datos.

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración

- Configurar el archivo **$HADOOP_HOME/etc/hadoop/mapred-site.xml**
- Especificar el nombre del framework para MapReduce.
- MapReduce que se ejecutará como una aplicación YARN.

```
<configuration>
 <property>
  <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
</configuration>
```

Hadoop: modo seudo-distribuido

Configuración

- Configurar el archivo **$HADOOP_HOME/etc/hadoop/yarn-site.xml**
- La propiedad *yarn.nodemanager.aux-services* indica al *NodeManager* que será necesario implementar el servicio auxiliar *mapreduce.shuffle.* A continuación debe indicarse un nombre de clase para implementar ese servicio.
- Esta configuración particular indica a YARN cómo realizar la etapa de shuffle (para aplicaciones MapReduce)

```
<configuration>
 <property>
  <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
 </property>
</configuration>
```

Hadoop: modo seudo-distribuido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Configuración y ejecución

- Formatear el NameNode
  - Para que NameNode se inicie, necesita inicializar el directorio donde contendrá sus datos.
  - El proceso de formato utilizará el valor asignado a dfs.name.dir en hdfs-site.xml (en el ejemplo es /home/hadoop/hadoopdata/hdfs/namenode). El proceso de formateo destruye toda la información contenida en el directorio y configura un nuevo sistema de archivos.

```
hdfs namenode -format
```

- Iniciar el cluster hadoop con el script **$HADOOP_HOME/sbin/start-dfs.sh**

- Iniciar yarn con el script **$HADOOP_HOME/sbin/start-yarn.sh**

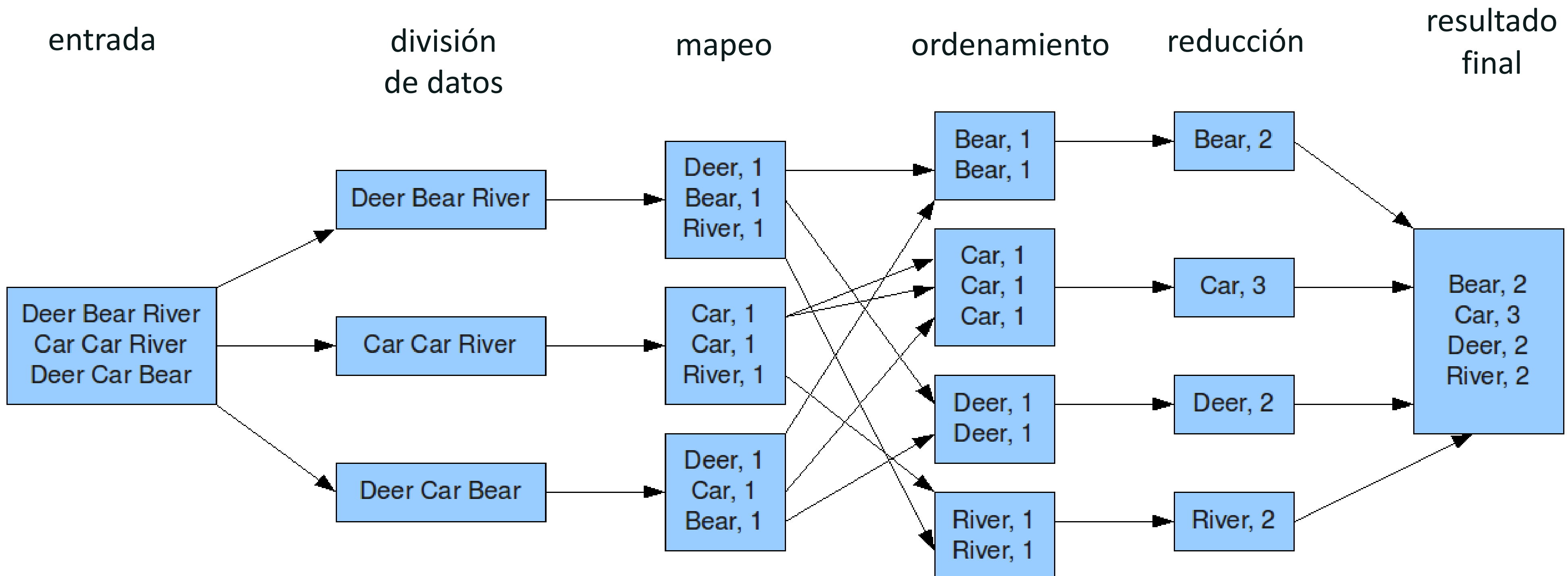
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Hadoop: ejemplo de aplicación MapReduce

Hadoop: word count

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación de aplicación MapReduce para conteo de palabras en un archivo de texto en Hadoop.



Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Creación del trabajo (job) con la clase main

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Clave y valor de la salida a utilizar: palabra (texto) y contador (entero).

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Caminos para archivos de entrada y salida

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Ejecutar el job MapReduce y esperar que finalice

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    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, one);
      }
    }
}
```

Extiende la clase org.apache.hadoop.mapreduce.Mapper, que tiene 4 tipos genéricos: KEY_IN,VAL_IN,KEY_OUT,VAL_OUT

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    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, one);
      }
    }
}
```

Implementa la función map(KEY_IN key, VAL_IN value, Context context) que recibe un par clave-valor para el Mapper y retornará pares clave-valor

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  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, one);
    }
  }
}
```

Cada mapper recibe datos (líneas) de entrada, de acuerdo al formato (InputFormat) que se utilice. Por defecto se envían 128MB del archivo de entrada a cada mapper, partiéndolo por líneas. La clave es el offset (en bytes) desde el comienzo del archivo.

Word count: mapper

```
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    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, one);
      }
    }
}
```

Separa la línea recibida por espacio. StringTokenizer es un iterador que permite recorrer los valores separados.

Word count: mapper

```
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    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, one);
      }
    }
}
```

Por cada token (palabra) se genera un tipo de dato Text y se retorna <palabra, 1>.

Word count: reducer

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Extiende la clase org.apache. hadoop.mapreduce.Reducer que recibe como clave una palabra y un iterador con todos los valores emitidos por el Mapper

```java
public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    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);
    }
}
```

Word count: reducer

```java
public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    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);
    }
}
```

Implementa el método reduce(KEY_IN key, Iterable<VAL_IN> values, Context); KEY_IN es el tipo de dato de la clave emitida por el Mapper, VAL_IN es el tipo de dato del valor emitido por el Mapper.

Word count: reducer

```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  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);
  }
}
```

Los valores recibidos son una lista de unos. El Reducer suma los valores y escribe como salida <palabra, suma de ocurrencias>

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El ejemplo, de implementación concisa y simple, está disponible en https://wiki.apache.org/hadoop/WordCount

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en C++ usando Hadoop Pipes

```
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"

class WordCountMap: public HadoopPipes::Mapper {
public:
  WordCountMap(HadoopPipes::TaskContext& context){}
  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue(), " ");
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], "1");
    }
  }
};
```

https://wiki.apache.org/hadoop/C++WordCount

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en C++ usando Hadoop Pipes

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  WordCountReduce(HadoopPipes::TaskContext& context){}
  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(),HadoopUtils::toString(sum));
  }
};

int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(HadoopPipes::TemplateFactory<WordCountMap,
                              WordCountReduce>());
}
```

https://wiki.apache.org/hadoop/C++WordCount

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en Python
- Mapper.py

```
#!/usr/bin/env python
import sys
# la entrada llega desde STDIN (entrada estándar)
for line in sys.stdin:
    # remover espacios en blanco al inicio y al final (si los hubiera)
    line = line.strip()
    # separar la línea en palabras
    words = line.split()
    # incrementar contadores
    for word in words:
        # escribir los resultados en STDOUT (salida estándar);
        # la salida será la entrada de la etapa de Reduce (reducer.py)
        print '%s\t%s' % (word, 1)
```

http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en Python: Reducer.py

```
#!/usr/bin/env python
import sys
from operator import itemgetter
current_word = None
current_count = 0
word = None
for line in sys.stdin: # la entrada llega desde STDIN (entrada estándar)
    line = line.strip() # remover espacios en blanco al inicio y al final (si los hubiera)
    word, count = line.split('\t', 1) # analizar a entrada que llega desde mapper.py
    # convertir count a entero (llega como  string)
    try: count = int(count) except ValueError:
         continue # si hay error se ignora la línea (count no era un número)
    # este if funciona porque Hadoop ordena la salida de los mappers por clave (cada palabra)
    if current_word == word:
         current_count += count
    else: if current_word:
              print '%s\t%s' % (current_word, current_count) # escribir a salida estándar
    current_count = count
    current_word = word
# imprimir las estadísticas de la última palabra
if current_word == word: print '%s\t%s' % (current_word, current_count)
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MapReduce en Hadoop

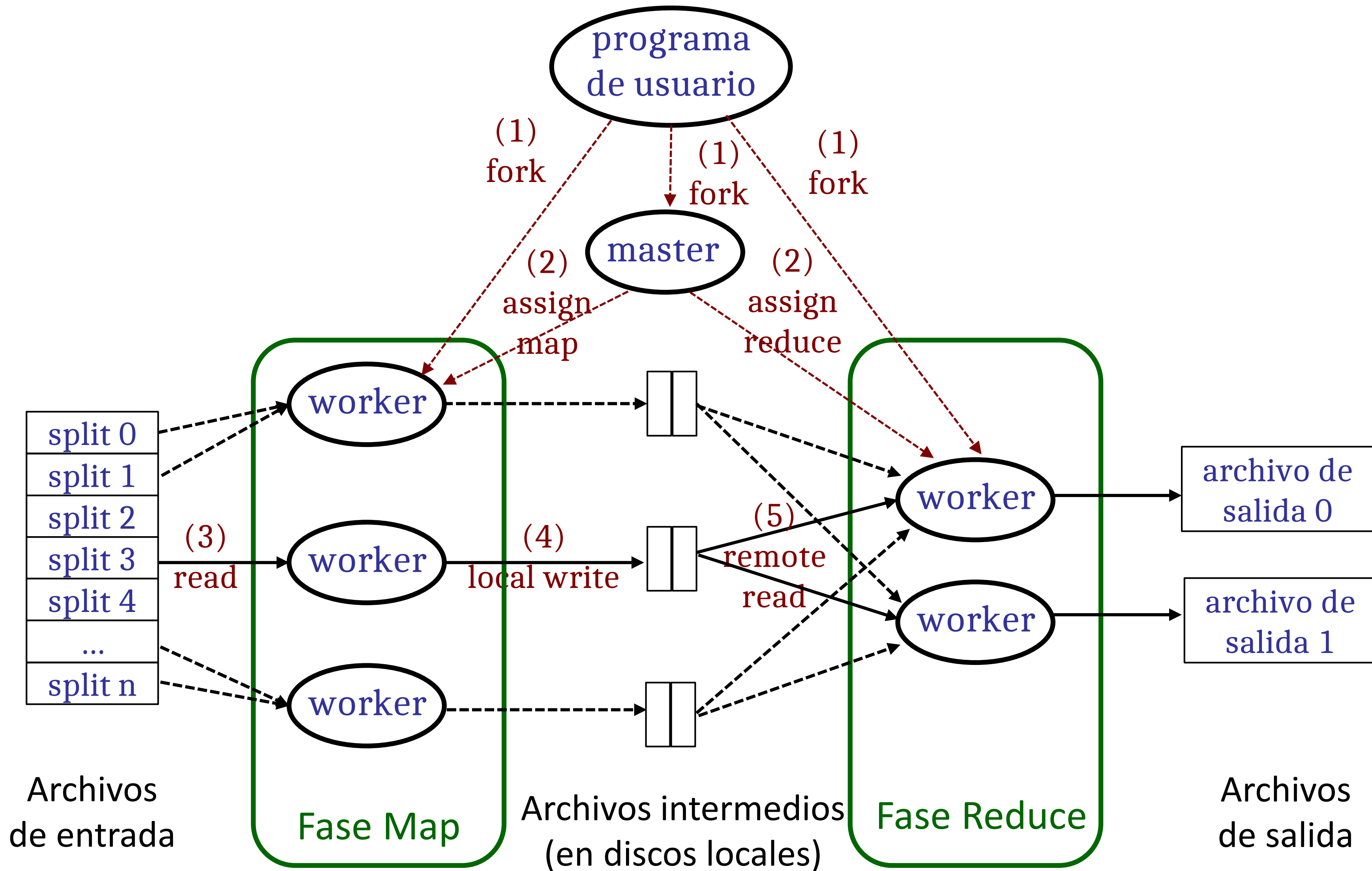
MapReduce en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

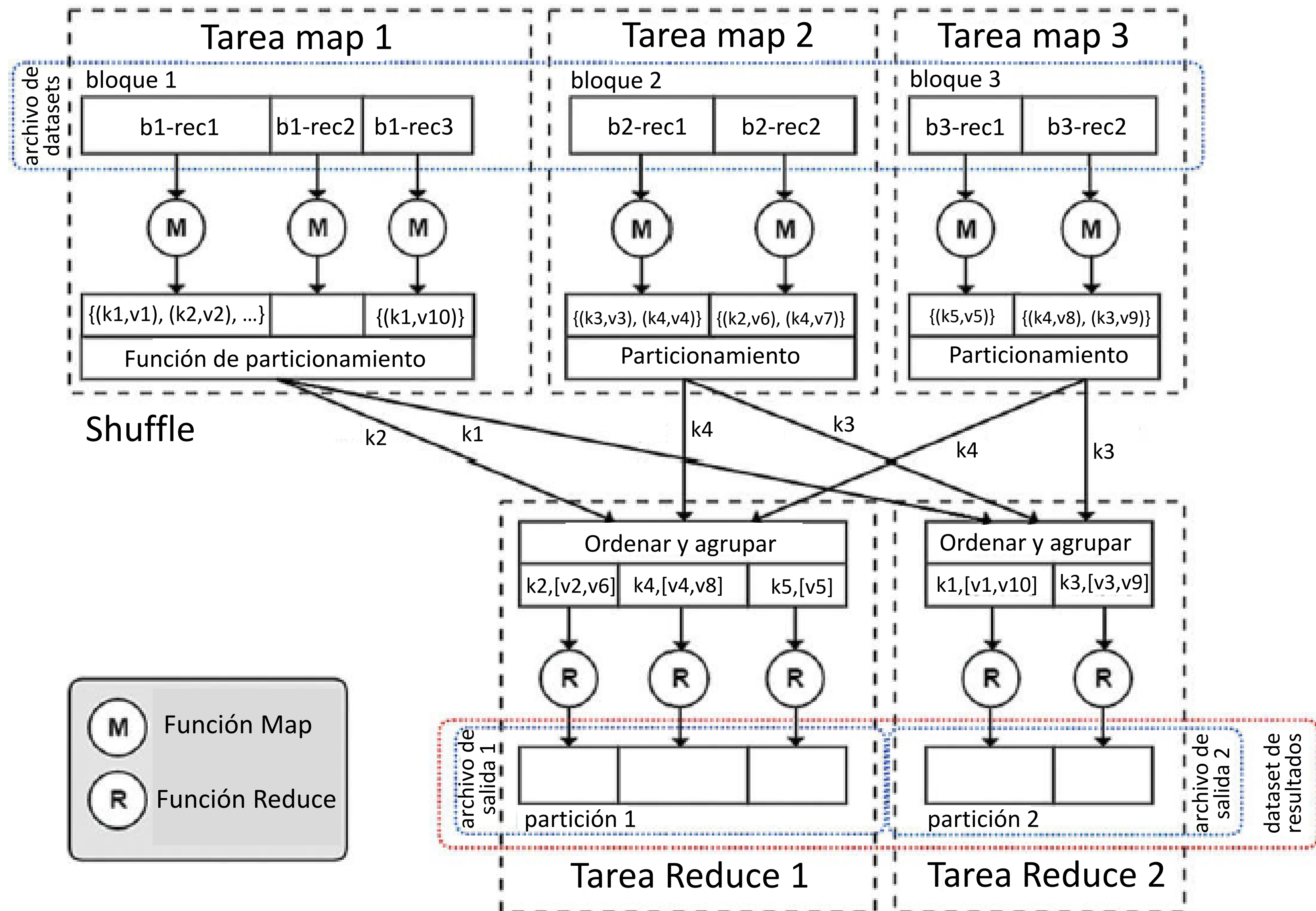
1. Esquema genérico
2. Etapas
3. Map (entrada, particionamiento, shuffle)
4. Reduce
5. Combiner
6. Implementación

MapReduce en Hadoop: esquema genérico

UNIVERSIDAD DE LA REPÚBLICA URUGUAY



MapReduce en Hadoop: etapas



MapReduce en Hadoop

- La tarea Map-Reduce queda definida por las dos funciones:
  - map: (f,{(k1;v1)}) →{(k2;v2)}
  - reduce: (k2;{v2}) →{(k3;v3)}
- Los tipos corresponden a

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }
  protected void map(KEYIN key, VALUEIN value,
      Context context) throws IOException, InterruptedException {
    // ...
  }
}

public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends ReducerContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }
  protected void reduce(KEYIN key, Iterable<VALUEIN> values,
      Context context) throws IOException, InterruptedException {
    // ...
  }
}
```

MapReduce en Hadoop: map

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En Hadoop, una tarea Map tiene cuatro fases: input, mapper, combiner y partitioner.
- Input utiliza las funciones input format y record reader para leer los registros en formato clave-valor para el procesamiento.
- Map aplica función(es) a cada par clave-valor en una porción del dataset.
  - Si el dataset está almacenado en un sistema de archivos como HDFS or S3, la porción es un bloque en el filesystem.
- Si hay n bloques de datos en el dataset de entrada existirán n tareas Map (Mappers) → Los datos de entrada determinan el número de Mappers !
  - El programador puede especificar el número de tareas Reduce (reducers).
- La fase Map está diseñada para no compartir estado ni datos entre procesos: cada Mapper procesamiento sus datos independientemente.
- Se asegura que cada registro es procesado una única vez (salvo en caso de fallos o de ejecución especulativa).

MapReduce en Hadoop: map

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En el ejemplo, tres Mappers operan sobre tres bloques en el filesystem (block1, block2 y block3). Cada Mapper ejecuta la función map() para cada registro (clave-valor), b1-rec1, b1-rec2, etc.
- La función map() recibe un par clave-valor y emite como salida cero o más pares clave-valor, que son datos intermedios del procesamiento.

```
map (in_key,in_value) → list(intermediate_key,intermediate_value)
```

- Ejemplos de función map:
  - Filtrar mensajes de error en un log:

    ```
    let map(k,v) = if (ERROR in v) then emit(k,v)
    ```

  - Convertir valores a minúsculas: `let map(k, v) = emit(k,v.toLower())`
  - Las funciones map pueden concatenarse y combinarse con operadores lógicos.
- El Mapper recolecta las listas de pares clave-valor emitidas por cada función map en una única lista agrupada por la clave del registro intermedio. Esta lista se pasa como entrada a la función de particionamiento.

MapReduce en Hadoop: entrada

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La entrada a una tarea MapReduce es un conjunto de archivos almacenados en HDFS. Los archivos son formateados por un *input format* que define cómo se divide un archivo en *input splits*.
  - Un input split es una partición del archivo de entrada a la tarea map.
- Se pueden implementar input format personalizados. El input format de texto (TextInputFormat) de Hadoop es útil para la mayoría de tareas de procesamiento de datos

- El *record reader* transforma un input split generado por el input format en registros. El record reader parsea los datos en registros pero no parsea los registros. Pasa los datos al mapper en formato clave-valor
- Generalmente la clave es información de posición (offset) en el archivo de entrada (tipo long) y el valor es una partición (chunk) de datos (texto).
- Se pueden implementar record readers personalizados, el que se incluye por defecto en Hadoop es útil para la mayoría de tareas de procesamiento de datos.

MapReduce : particionamiento

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La función de particionamiento (*partitioner*) se encarga de que cada clave intermedia y su lista de valores se envíe a una y solo una tarea de Reduce.
- La implementación más común es mediante un particionamiento por correspondencia (‘hash’), que crea un hash (identificador único) para la clave y divide el espacio de claves ‘hasheadas’ en n particiones.
- El número de particiones está relacionado con el número de tareas reduce (Reducers) que especifica el programador.
- Particionamientos específicos pueden ser implementados dependiendo de la lógica del procesamiento a realizar.
  - Por ejemplo, particionar datos cronológicos por año, mes o semana.
- La función de particionamiento se invoca para cada clave y su salida corresponde al Reducer encargado de procesar el resultado.
  - Típicamente, retorna un entero entre 0 y n-1, sindo n el número de Reducers especificado por el programador.

MapReduce: shuffle

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El procesamiento realizado por los Mappers es independiente y no require comunicar datos ni sincronizar procesos.
- La etapa de recombinar claves intermedias y sus valores asociados provenientes de multiples Mappers (en general, ejecutando en diferentes recursos de cómputo) require sincronizar y comunicar datos.
- La etapa shuffle toma la salida de cada Mapper y la envía a cada Reducer especificado por la función de particionamiento.
- En general, esta es la tarea más costosa en una aplicación MapReduce, ya que require la transferencia física de datos entre nodos a través de la red.
- Shuffle incluye el ordenamiento (por clave), mediante ‘merge’ de los datos recibidos de diferentes Mappers.
- Shuffle es asincrónico, permite solapar cómputo y comunicaciones.
- Shuffle soporta encriptado (HTTPS/SSL), pero reduce el desempeño.

MapReduce: reduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cada Reducer ejecuta la función reduce() para cada clave intermedia y su lista de valores asociados.
- Su salida es un conjunto (puede ser vacío) de pares clave-valor, que pueden ser parte de la salida del procesamiento o pueden ser entrada de una nueva fase Map en un workflow o pipeline.

```
reduce(key,list(intermediate_value)) → key, out_value
```

- Las funciones reduce() usualmente incluyen operaciones de agregación como sumas, conteos, promedios, etc.
- Ejemplo de función reduce() de suma:

```
let reduce(k,list<v>) = sum=0 for int i in list<v>:sum+=i emit(k, sum)
```

- El conteo se implementa sumando el valor 1 por cada elemento a contar.

MapReduce en Hadoop: reduce

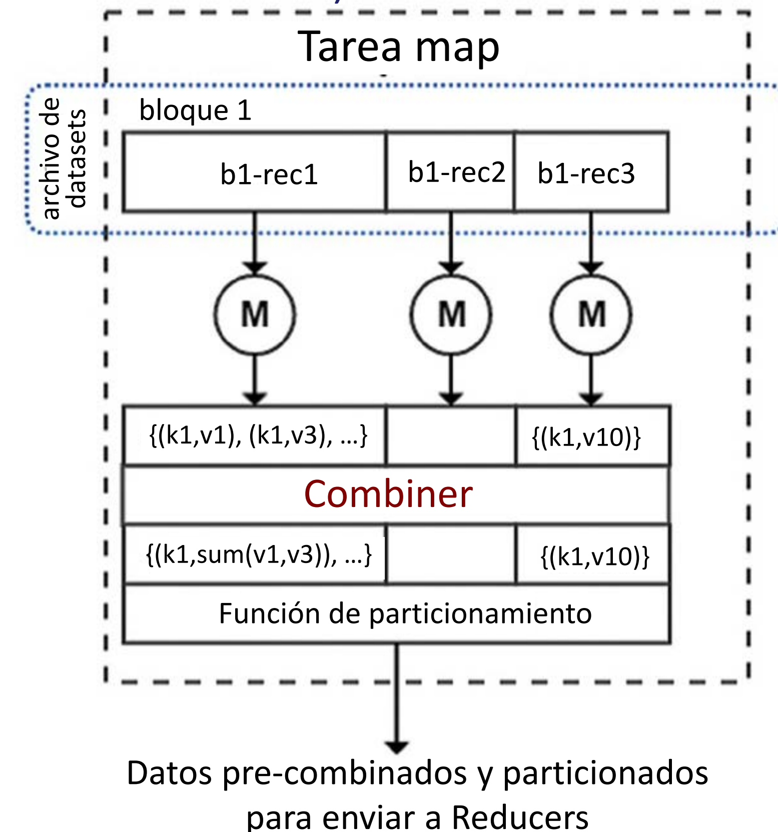
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Normalmente el reducer retorna un único par clave/valor para cada clave que procesa. Sin embargo, los pares clave valor pueden ser expandidos tanto cómo se necesite.
- Cuando una tarea reduce finaliza, retorna un archivo de resultados y lo almacena en HDFS.
- Como se presentó en el ejemplo, HDFS replica automáticamente los resultados.

MapReduce: combiner

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Es un reducer localizado (opera sobre datos generados en un solo recurso de cómputo), que permite combinar información al final de la etapa Map, para reducir la cantidad de información enviada a la etapa Reduce (disminuye los tiempos de transferencia de datos).
- Si la función de reducción en conmutativa y asociativa, es posible utilizar el mismo Reduce como Combiner.
- En general aporta importantes mejoras al desempeño del algoritmo implementado, al reducir el ancho de banda de las comunicaciones.



MapReduce: etapas en Hadoop

Shuffle
Sort
Reduce

Tarea Map
input split
map
buffer in memory
partition, sort, and spill to disk
particiones
merge on disk
fetch

Tarea Reduce
merge
merge
merge
reduce
output
datos en memoria y en disco

otras tareas Map
otras tareas Reduce

MapReduce en Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Driver (mandatorio): aplicación (shell) invocada desde el cliente. Configura la clase MapReduce Job (no personalizable) y realiza el submit al Resource Manager (o JobTracker en Hadoop 1).
- Clase Mapper (mandatoria): implementa la definición de los formatos clave/valor para entrada y salida utilizados al procesar registros. La clase tiene un único método map, donde se codifica el procesamiento y la clave/valor de salida.
- Clase Reducer (opcional para aplicaciones map-only): implementa la operación de reducción.
- Clase Combiner (opcional): generalmente definida como el reducer, pero puede ser diferente.
- Clase Partitioner (opcional): para personalizar el particionamiento (ejemplo, sort secundario, datos dispersos, etc.)
- Clases RecordReader y RecordWriter (opcionales): para lectura y escritura de formatos personalizados.

MapReduce en Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Desde el driver se puede utilizar la API de MapReduce, que incluye métodos de fábrica para crear instancias de todos los componentes de la aplicación.
- La API genérica Hadoop Streaming permite utilizar código implementado en otros lenguajes (comúnmente C, Python y Perl).
- Ventaja: permite a usuarios con poco conocimiento de Java desarrollar código MapReduce.
- Desventaja: requiere capas adicionales de abstracción para implementar el streaming, que impacta en el desempeño (tiempo de ejecución y uso de memoria).
- Las funciones map y reduce pueden implementarse con Hadoop Streaming, pero Record readers/writers y partitioners deben estar escritos en Java. Por este motivo las aplicaciones Hadoop Streaming son utilizables para procesar datos de texto solamente.

MapReduce en Hadoop: detalles

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Map

- Las tareas map no escriben sus salidas a disco. Utilizan buffers circulares en memoria (de tamaño io.sort.mb, por defecto 100 MB). Cuando se supera una cota (io.sort.spill.percent, por defecto 80%) un nuevo thread escribe a disco local (en mapred.local.dir) de modo Round-Robin. Luego, la escritura a buffer y a disco continuan en paralelo.
- El thread que escribe divide los datos en particiones para los reducers, en cada partición los datos se ordenan.
- Si hay combiner, ejecuta sobre los datos ordenados y produc datos más compactos y ordenados, de modo de transmitir menos al reducer.
- La salida de los mappers pueden comprimirse para más eficiencia (debe habilitarse en settingmapred.compress.map.output).
- Las salidas de los mappers quedan disponibles para los reducers sobre HTTP.

MapReduce en Hadoop: detalles

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Reduce

- El reducer verifica qué particiones necesita. La copia se inicia apenas están disponibles, con threads que copian en paralelo (mapred.reduce.parellel.copies, por defecto 5).
- La copia se realiza directamente a la memoria de la JVM del reducer (a disco para gran volumen de datos). Un thread que ejecuta en background los combina en archivos grandes ordenados. Si las salidas de los mappers están comprimidas, deben descomprimirse primero.
- Cuando se copian todos los datos se inicia la etapa de sort (en realidad es un merge, porque los datos ya fueron ordenados en el mapper).
- El merge se realiza en rondas: salidas de mappers/archivos a combinar (io.sort.factor, por defecto 10). El merge envía directamente los archivos a memoria del reducer para no escribir a disco.
- La reducción trabaja sobre su entrada y escribe la salida al filesystem, típicamente HDFS.

MapReduce en Hadoop: eficiencia

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La etapa crítica es Shuffle and Sort. Claves para mejorar la eficiencia:
  - Usar Combiner para reducer los datos transferidos a los reducers.
  - Elegir el número óptimo de reducers. Muchos datos: no usar un solo reducer. Muchos reducers implican muchas particiones en los mappers.
  - Iniciar los reducers luego que un porcentaje de mappers ha finalizado. Se selecciona con mapreduce.job.reduce.slowstart.completedmaps (defecto 0.05). Reducers que se inician más temprano quedarán ociosos.
  - Comprimir la salida de los mappers.
  - Configurar parámetros: [mapreduce.task.io].sort.mb/io.sort.factor, mapreduce.map.sort.spill.percent, mapreduce.shuffle.max.threads, [mapreduce.reduce.shuffle].input.buffer.percent/merge.percent,, [mapreduce.reduce].input.buffer.percent/shuffle.parallelcopies, etc.
  - Lista completa en https://hadoop.apache.org/docs/r2.4.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Hadoop MapReduce para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Análisis de datos climáticos: National Climatic Data Center (NCDC)

- Formato texto, cada línea es un registro con datos variables u opcionales.
- Ejemplo: `0057332130999991950010103004+51317+028783FM-12+017199999V02032…`
- Ejemplo (línea separada por claridad)

```
0057
332130     # identificador de estación (USAF)
99999      # identificador de estación (WBAN)
19500101   # fecha de observación
0300       # hora de observación
4
+51317     # latitud (grados x 1000)
+028783    # longitud (grados x 1000)
FM-12
+0171      # elevación (metros)
99999
V020
32         # dirección del viento (grados)
…
```

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Los archivos se organizan por fecha y estación meteorológica: un directorio por año desde 1901 a 2001, con un archivo gzip por estación.
- Ejemplo, para 1990

```
ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
…
```

- Los reportes incluyen a decenas de miles de estaciones.
- Cada estación tiene sus datos en archivos pequeños (en forma genérica es más eficiente procesar muchos archivos pequeños que pocos archivos grandes, pero debe recordarse que en Hadoop es a la inversa).

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo: hallar la temperatura máxima para cada año
- Los datos pueden analizarse en Linux

```
#!/usr/bin/env bash
for year in all/*; do
   echo -ne `basename $year .gz`"\t"
   gunzip -c $year | \
   awk '{ temp = substr($0, 88, 5) + 0;
   q = substr($0, 93, 1);
   if (temp !=9999 && q ~ /[01459]/ && temp > max)
            max = temp }
   END { print max }'
done
```

Imprime el año

Extrae los campos relevantes: temperatura del aire (cast a entero) y código de calidad

9999 = valor erróneo
1459 = lectura sospechosa

- El bloque END se ejecuta luego de procesar todas las líneas del archivo, para imprimir el valor máximo de temperatura.

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejecución

```
./max_temperatura.sh
1901 317
1902 244
1903 289
1904 256
1905 283
…
```

- Procesar 100 años (16 GB de datos comprimidos):
  - 42 minutos en una instancia EC2 High-CPU Extra Large (8 vCPU, 8GB RAM, 0.52/0.84 USD hora).
  - 78 minutos en ozzy (cloudera en VM sobre i3 dualcore, 8GB RAM).
  - 44 minutos en nebula (AMD Opteron 6172 2.10GHz) de Cluster-FING.
- Cómo mejorar la eficiencia del análisis? Aplicar paralelismo sobre memoria distribuida: procesar diferentes años en diferentes procesos sobre diferentes unidades de cómputo.

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Paralelismo sobre memoria distribuida, aspectos a analizar:
- División de datos. Dividir por año no garantiza balance de carga y el tiempo queda determinado por el proceso más lento. Dividir los datos equitativamente demanda trabajo adicional de preprocesamiento.
- Combinar los resultados. Si se divide por año es trivial, porque los archivos son independientes. Si se utiliza un particionamiento equitativo se debe aplicar una lógica específica para la combinación.
- Plataforma a utilizar. El procesamiento puede exceder la capacidad de una único recurso de cómputo. Al utilizar multiples recursos surgen otros factores relevantes: organización de las tareas, coordinación, confiabilidad, tolerancia a fallos, etc.
- Utilizar un framework como Hadoop ayuda a resolver estos aspectos de modo simple.

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MapReduce sobre Hadoop

- Entrada: datos en formato NCDC.
- Input format de texto (cada línea es un valor de texto).
- Clave: offset del inicio de la línea, contado desde el inicio del archivo (no es necesaria para el procesamiento).
- Map: filtra registros erróneos/sospechosos y extrae año y temperatura
- Ejemplo: considerando los datos de entrada

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
```

- La función map recibe los pares clave-valor

```
(  0, 0067011990999991950051507004…9999999N9+00001+99999999999…)
(106, 0043011990999991950051512004…9999999N9+00221+99999999999…)
(212, 0043011990999991950051518004…9999999N9-00111+99999999999…)
(318, 0043012650999991949032412004…0500001N9+01111+99999999999…)
```

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Map filtra los datos e imprime año y temperatura

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

- La salida del Map se ordena y agrupa por clave (año).
- La función reduce recibe registros con año y lista de temperaturas

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

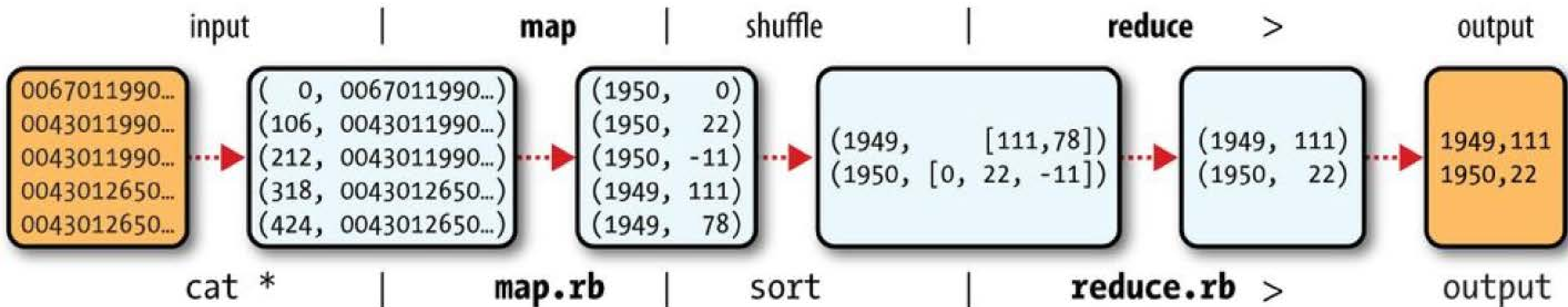
- Reduce itera sobre la lista y calcula el máximo

```
(1949, 111)
(1950, 22)
```

Hadoop para análisis de datos

- Flujo de datos y ejecución

Fases MapReduce en Hadoop



Fases análogas en un pipeline de Linux

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
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;

public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

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

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

provee tipos de datos optimizados para serialización

clave-valor de entrada

clave-valor de salida

extrae substrings de interés

escribe en el contexto

Hadoop para análisis de datos

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

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

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

clave-valor de entrada

clave-valor de salida

Iteración y comparación

escribe en el contexto

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

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

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

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

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

Objeto job, se empaqueta en un jar para distribución en el cluster

Archivos de entrada y salida

Map y Reduce a utilizar

Tipos de salida, deben concordar con los definidos en la clase Reduce

Ejecución del trabajo

Hadoop para análisis de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejecución

```
export HADOOP_CLASSPATH=hadoop-examples.jar
hadoop MaxTemperature sample.txt output
```

- Hadoop se invoca con el nombre de la clase Mapreduce y lanza una JVM para su ejecución (incluyendo bibliotecas y configuracion).
- La variable `HADOOP_CLASSPATH` agrega las clases al classpath.
- En modo standalone, los comandos deben ejecutarse desde el directorio que tiene los ejemplos, o utilizar rutas absolutas.
- La salida muestra información: ID del job, número de tareas Map y Reduce, estadísticas de datos procesados (‘Counters’).
- La salida se escribe en el directorio de salida, que contendrá un archivo por Reducer ejecutado (part-r-XXXXX). En el ejemplo es solo un archivo:

```
Hadoop fs -cat output/part-r-00000
1949 111
1950 22
```

Patrones de MapReduce

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones: modelos para soluciones reutilizables a problemas similares. *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al. 1995).
- Patrones de MapReduce: enfocados en el procesamiento de grandes volúmenes de información.
- Independientes del dominio de aplicación.
- No son ‘recetas’, no proporcionan una solución ‘mágica’ a un problema.
- Son modelos genéricos que deben instanciarse para resolver cada problema en particular.
- Proporcionan la base para herramientas de alto nivel (incluyendo Pig, Hive, etc.)

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de conteo/suma/índices
- Patrones de filtro
- Patrones de organización de datos
- Patrones de combinación (join)
- Metapatrones
- Patrones de entrada y salida

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de conteo/suma/índices:
  - Algoritmos que cuentan o retornan un conjunto resumido de datos.
  - Idea: agrupar registros por un campo y calcular una función numérica de agregación por grupo. Los datos deben ser numéricos y admitir agregación.
  - Ejemplos: cálculo de índices, conteo de determinado tipo de datos, cálculo de datos estadísticos.
  - Ejemplos concretos: cálculo de visitas a una determinada página web, cálculo de ganancias en una determinada transacción, etc.
  - Similar a SELECT MIN(col1), COUNT(*) FROM Table GROUP BY col2; (de SQL) o a b = GROUP a BY col2; c = FOREACH b GENERATE group, MIN(a.col1); de Pig
  - Patrones incluidos: *numerical summarizations*, *inverted index*, *counting with counters*.
- Permiten obtener una vision de alto nivel de valores numéricos resultantes del análisis de los datos.

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de filtro:
  - Se basan en encontrar un subconjunto de datos del conjunto inicial (sin modificarlo).
  - Ejemplos: descartar datos no válidos o no útiles, obtener listas de mejores rankeados, obtener información sobre un determinado tema o generada en un determinado período.
  - Se basan en resumir información para poder analizar un conjunto reducido de los datos. Se pueden utilizar funciones y bibliotecas de Java para expresiones regulares.
  - Similar a SELECT * FROM table WHERE value < 3; (de SQL) o a b = FILTER a BY value < 3; (de Pig)
  - Patrones incluidos: *sampling, filtering, bloom filtering, top ten, distinct*.
- Permiten filtrar datos de interés

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de organización de datos:
  - Se basan en transformar o reorganizar un conjunto de datos (creando nuevos datos con diferente estructura).
  - En ocasiones, las tareas MapReduce son utilizadas para realizar un procesamiento primario de la información (‘masticar’) y organizarla de forma que luego otros procesos puedan procesarla y manipularla eficientemente.
  - Ejemplos: transformar datos no estructurados a estructurados, particionar, ordenar, barajar, etc.
  - Patrones incluidos: *structured to hirarchical pattern, partitioning and binning patterns, total order sorting, shuffling patterns.*

- Permiten modificar la organización de los datos

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de combinación (join)
  - Se basan en realizar cruzamientos de información entre distintos conjuntos de datos.
  - En una base de datos tradicional el join es una tarea sencilla, sin embargo, al trabajar con datos no estructurados es una tarea compleja y muchas veces no muy eficiente.
  - Inner join, outer join, antijoin, product cartesiano.
  - Patrones: *reduce side join*, *reduce side join with bloom filter*, *replicated join*, *composite join*, *cartesian product*.

- Permiten combinar diferentes conjuntos de datos, procedentes de diversos repositorios o de diversas fuentes.

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de entrada/salida:
  - Se basan en personalizar la entrada y la salida de las tareas.
  - I/O estándar en Hadoop: usa InputFormat/RecordReader y OutputFormat/RecordWriter
  - Para resolver problemas realistas puede ser necesario modificar la forma en que una implementación de MapReduce (por ejemplo, Hadoop) lee o almacena la información.
  - Ejemplos: *generating data* (genera datos en línea y en paralelo) *,* *external source input y external source output* (entrada desde/salida a repositorios externos, diferentes de Hadoop/HDFS) y *partition prunning* (eliminar particiones, muy útil en data warehouse), etc.
- Permiten acceder a datos o escribir datos en ubicaciones no estándar

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Metapatrones:
  - Definen patrones combinando los patrones básicos comentados previamente.
  - Proponen diferentes estrategias para el manejo de los patrones y cómo relacionarlos entre si.
  - En general, una única tarea MapReduce no realiza todo el trabajo. Es necesario encadenar varias tareas, utilizando diferentes patrones.
  - Patrones: *job chaining* (encadenamiento), uno de los más utilizados en MapReduce y *job merging*.
  - El encadenamiento se implementa con un *master driver*, que lance varios drivers (mains) para los diferentes jobs.
- Permiten resolver problemas complejos combinando multiples patrones simples.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Job chaining: el master driver invoca en secuencia cada job MapReduce.
- Debe asegurarse que la salida (output path) del primer trabajo corresponde a la entrada (input path) del segundo.
  - Consejo: almacenar el path temporal como una variable string compartida.
- Los directorios temporales deben limpiarse cuando se completa un job.
  - Consejo: tener cuidado con el volume de datos temporales creados, que deben almacenarse en el filesystem (HDFS/Hbase, etc.)
- Se pueden ejecutar múltiples jobs en paralelo usando Job.submit() en lugar de Job.waitForCompletion(). Job.submit() ejecuta en background.
  - Job.isComplete(): chequeo de finalización de un job (no bloqueante), puede usarse en modo poll.
  - Debe chequearse que el job finalize correctamente, para no ocasionar errores en la cadena de trabajos.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de job chaining con master driver: desplegar usuarios e información, separando en categorías (por encima o debajo del promedio).
  - Un trabajo realiza los conteos.
  - Otro trabajo (map only) crea las categorías y el particionamiento.
  - El ejemplo incluye cuatro patrones diferentes: sumarización, conteo con contadores, particionamiento (binning) y un join replicado.
  - Salida: user ID, número de posts y su reputación.
  - El promedio de posts por usuario se calcula con los contadores del framework.
  - Los datos de los usuarios se almacenan en DistributedCache, una utilidad de Hadoop que garantiza que un archivo en HDFS esté presente en el filesystem local de cada tarea que requiera el archivo, para que queden disponibles para el segundo trabajo.
  - Los datos se enriquecen incluyendo la reputación del usuario, obtenida de un dataset separado.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 1

```
public static class UserIdCountMapper extends
        Mapper<Object, Text, Text, LongWritable> {
    public static final String RECORDS_COUNTER_NAME = "Records";
    private static final LongWritable ONE = new LongWritable(1);
    private Text outkey = new Text();
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
    Map<String, String> parsed = MRDPUtils.transformXmlToMap(value
        .toString());
    String userId = parsed.get("OwnerUserId"); // atributo user ID
    if (userId != null) {
        outkey.set(userId);
        context.write(outkey, ONE);
        context.getCounter(AVERAGE_CALC_GROUP,RECORDS_COUNTER_NAME).
            increment(1);             // incrementa el contador de registros
        }
    }
}
```

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Reducer del job 1

```
public static class UserIdSumReducer extends
        Reducer<Text, LongWritable, Text, LongWritable> {
    public static final String USERS_COUNTER_NAME = "Users";
    private LongWritable outvalue = new LongWritable();
    public void reduce(Text key, Iterable<LongWritable> values,
        Context context) throws IOException, InterruptedException {
    // Incrementar el contador de usuarios, cada reduce agrupa un usuario
    context.getCounter(AVERAGE_CALC_GROUP,USERS_COUNTER_NAME).increment(1);
    int sum = 0;
    for (LongWritable value : values) {
        sum += value.get();
    }
    outvalue.set(sum);
    context.write(key, outvalue);
    }
}
```

Itera sobre los valores de entrada y suma. Se utiliza otro contador para el número de grupos (para calcular el promedio).

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 2

- Setup:
  1. Toma el promedio de posts por usuario, del objeto Context (inicializado en la configuración del job);
  2. MultipleOutputs se utiliza para escribir la salida a diferentes categorías;
  3. Los datos de usuario se recuperan del DistributedCache y se parsean para construir un mapa (user ID:reputación) que se utilizará para enriquecer los datos de salida.
- Método map:
  1. Parsea la entrada para obtener user ID y número de posts y construye el registro de salida (userID, número de posts, reputación).
  2. El número de posts del usuario se compara con el promedio y se clasifica al usuario en la categoría correcta. El cuarto parámetro (opcional) de MultipleOutputs.write se usa como nombre de cada archive de partición.
  3. Se crean directorios separados para cada categoría.
- Cleanup: cierra MultipleOutputs.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 2

```
public static class UserIdBinningMapper extends
        Mapper<Object, Text, Text, Text> {

    public static final String AVERAGE_POSTS_PER_USER = "avg.posts.per.user";

    public static void setAveragePostsPerUser(Job job, double avg) {
        job.getConfiguration().set(AVERAGE_POSTS_PER_USER,
                Double.toString(avg));
    }

    public static double getAveragePostsPerUser(Configuration conf) {
        return Double.parseDouble(conf.get(AVERAGE_POSTS_PER_USER));
    }

    private double average = 0.0;
    private MultipleOutputs<Text, Text> mos = null;
```

MultipleOutputs se utiliza para escribir la salida a diferentes categorías.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 2

```
private Text outkey = new Text(), outvalue = new Text();
private HashMap<String, String> userIdToReputation =
        new HashMap<String, String>();

protected void setup(Context context) throws IOException,
        InterruptedException {
    average = getAveragePostsPerUser(context.getConfiguration());

    mos = new MultipleOutputs<Text, Text>(context);

    Path[] files = DistributedCache.getLocalCacheFiles(context
            .getConfiguration());

    // Read all files in the DistributedCache
    for (Path p : files) {
        BufferedReader rdr = new BufferedReader(
                new InputStreamReader(
                        new GZIPInputStream(new FileInputStream(
                                new File(p.toString())))));
```

Toma el promedio de posts por usuario del context

Recupera datos de usuario del DistributedCache

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 2

```
            String line;
            // For each record in the user file
            while ((line = rdr.readLine()) != null) {
                // Get the user ID and reputation
                Map<String, String> parsed = MRDPUtils
                        .transformXmlToMap(line);
                // Map the user ID to the reputation
                userIdToReputation.put(parsed.get("Id"),
                        parsed.get("Reputation"));
            }
        }
    }
```

Parsear datos de usuario para construir el mapa (user ID:reputación)

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper del job 2

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

    String[] tokens = value.toString().split("\t");

    String userId = tokens[0];
    int posts = Integer.parseInt(tokens[1]);

    outkey.set(userId);
    outvalue.set((long) posts + "\t" + userIdToReputation.get(userId));

    if ((double) posts < average) {
        mos.write(MULTIPLE_OUTPUTS_BELOW_NAME, outkey, outvalue,
                MULTIPLE_OUTPUTS_BELOW_NAME + "/part");
    } else {
        mos.write(MULTIPLE_OUTPUTS_ABOVE_NAME, outkey, outvalue,
                MULTIPLE_OUTPUTS_ABOVE_NAME + "/part");
    }
}
```

Parsea la entrada y construye el registro de salida (userID, número de posts, reputación).

Clasifica al usuario en la categoría correcta.

Metapatrones de MapReduce: job chaining

Mapper del job 2

```
    protected void cleanup(Context context) throws IOException,
            InterruptedException {
        mos.close();
    }
}
```

Cleanup: cierra MultipleOutputs.

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Job chaining: master driver

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

    Configuration conf = new Configuration();
    Path postInput = new Path(args[0]);
    Path userInput = new Path(args[1]);
    Path outputDirIntermediate = new Path(args[2] + "_int");
    Path outputDir = new Path(args[2]);
    // Setup first job to counter user posts
    Job countingJob = new Job(, "JobChaining-Counting");
    countingJob.setJarByClass(JobChainingDriver.class);
    // Set mapper and reducer, use the API's long sum reducer for a combiner!
    countingJob.setMapperClass(UserIdCountMapper.class);
    countingJob.setCombinerClass(LongSumReducer.class);
    countingJob.setReducerClass(UserIdSumReducer.class);

    countingJob.setOutputKeyClass(Text.class);
    countingJob.setOutputValueClass(LongWritable.class);
    countingJob.setInputFormatClass(TextInputFormat.class);

    TextInputFormat.addInputPath(countingJob, postInput);
    countingJob.setOutputFormatClass(TextOutputFormat.class);
    TextOutputFormat.setOutputPath(countingJob, outputDirIntermediate);
    // Execute job and grab exit code
    int code = countingJob.waitForCompletion(true) ? 0 : 1;
```

Primer job: contar posts

Ejecutar y esperar finalización

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Job chaining: master driver

Obtener contadores

Segundo job: separador

```
if (code == 0) {
    // Calculate the average posts per user by getting counter values
    double numRecords = (double) countingJob.getCounters()
            .findCounter(AVERAGE_CALC_GROUP,UserIdCountMapper.RECORDS_COUNTER_NAME).getValue();

    double numUsers = (double) countingJob.getCounters()
            .findCounter(AVERAGE_CALC_GROUP, UserIdSumReducer.USERS_COUNTER_NAME).getValue();
    double averagePostsPerUser = numRecords / numUsers;

    // Setup binning job
    Job binningJob = new Job(new Configuration(), "JobChaining-Binning");
    binningJob.setJarByClass(JobChainingDriver.class);
    // Set mapper and the average posts per user
    binningJob.setMapperClass(UserIdBinningMapper.class);
    UserIdBinningMapper.setAveragePostsPerUser(binningJob, averagePostsPerUser);

    binningJob.setNumReduceTasks(0);

    binningJob.setInputFormatClass(TextInputFormat.class);
    TextInputFormat.addInputPath(binningJob, outputDirIntermediate);

    // Add two named outputs for below/above average
    MultipleOutputs.addNamedOutput(binningJob,
            MULTIPLE_OUTPUTS_BELOW_NAME, TextOutputFormat.class,Text.class, Text.class);
```

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Job chaining: master driver

Configurar y cargar MultipleOutputs y DistributedCache

```
        MultipleOutputs.addNamedOutput(binningJob,
                MULTIPLE_OUTPUTS_ABOVE_NAME, TextOutputFormat.class,
                Text.class, Text.class);

        MultipleOutputs.setCountersEnabled(binningJob, true);
        TextOutputFormat.setOutputPath(binningJob, outputDir);

        // Add the user files to the DistributedCache
        FileStatus[] userFiles = FileSystem.get(conf).listStatus(userInput);
        for (FileStatus status : userFiles) {
            DistributedCache.addCacheFile(status.getPath().toUri(),
                    binningJob.getConfiguration());
        }
```

```
        // Execute job and grab exit code
        code = binningJob.waitForCompletion(true) ? 0 : 1;
    }
```

Limpiar datos intermedios

```
    // Clean up the intermediate output
    FileSystem.get(conf).delete(outputDirIntermediate, true);
```

```
    System.exit(code);
}
```

Metapatrones de MapReduce: job chaining

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Parallel job chaining: dos (o más) trabajos independientes se crean y ejecutan en paralelo y se monitorean hasta su finalización.
- Ejemplo: tomar la salida del trabajo previo para calcular la reputación promedio de los usuarios en cada categoría.

Metapatrones: parallel jobs

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Parallel jobs: master driver para ejecución en paralelo

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

    Configuration conf = new Configuration();

    Path belowAvgInputDir = new Path(args[0]);
    Path aboveAvgInputDir = new Path(args[1]);
    Path belowAvgOutputDir = new Path(args[2]);
    Path aboveAvgOutputDir = new Path(args[3]);

    Job belowAvgJob = submitJob(conf, belowAvgInputDir, belowAvgOutputDir);
    Job aboveAvgJob = submitJob(conf, aboveAvgInputDir, aboveAvgOutputDir);

    // While both jobs are not finished, sleep

    while (!belowAvgJob.isComplete() || !aboveAvgJob.isComplete()) {
        Thread.sleep(5000);
    }

    if (belowAvgJob.isSuccessful()) {
        System.out.println("Below average job completed successfully!");
    } else {
        System.out.println("Below average job failed!");
    }

    if (aboveAvgJob.isSuccessful()) {
        System.out.println("Above average job completed successfully!");
    } else {
        System.out.println("Above average job failed!");
    }

    System.exit(belowAvgJob.isSuccessful() &&
            aboveAvgJob.isSuccessful() ? 0 : 1);
}
```

Metapatrones: parallel jobs

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Parallel job chaining: master driver

```
private static Job submitJob(Configuration conf, Path inputDir,
        Path outputDir) throws Exception {

    Job job = new Job(conf, "ParallelJobs");
    job.setJarByClass(ParallelJobs.class);

    job.setMapperClass(AverageReputationMapper.class);
    job.setReducerClass(AverageReputationReducer.class);

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

    job.setInputFormatClass(TextInputFormat.class);
    TextInputFormat.addInputPath(job, inputDir);

    job.setOutputFormatClass(TextOutputFormat.class);
    TextOutputFormat.setOutputPath(job, outputDir);

    // Submit job and immediately return, rather than waiting for completion
    job.submit();
    return job;
}
```

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Job chaining es el patrón más complicado de implementar y manejar.
- No es una funcionalidad provista 'out of the box' el el framework MapReduce de Hadoop.
- Hadoop permite especificar y manejar un job MapReduce de modo simple, pero el manejo de Jobs con múltiples etapas implica codificación manual de varias tareas de coordinación y operación (por ejemplo, manejo de fallos en las diferentes etapas del job, limpiar salidas intermedias, etc.)
- Proyectos como Apache Oozie proven funcionalidades para construir workflows y coordinar la ejecución de multiples jobs.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MapReduce en Hadoop: aspectos avanzados

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Tolerancia a fallos
- Si un Mapper falla, el proceso master automáticamente reasigna la tarea a otro nodo (preferentemente a uno que tenga una copia del mismo bloque de datos, para mantener la localidad).
- Por defecto, el número de reintentos antes de declarar un trabajo como fallido es cuatro, pero este valor puede modificarse.
- Si un Reducer falla, el proceso master automáticamente la reejecuta en otro nodo, utilizando los mismos datos intermedios, que se mantienen almacenados mientras el trabajo está en ejecución.

MAP-REDUCE: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

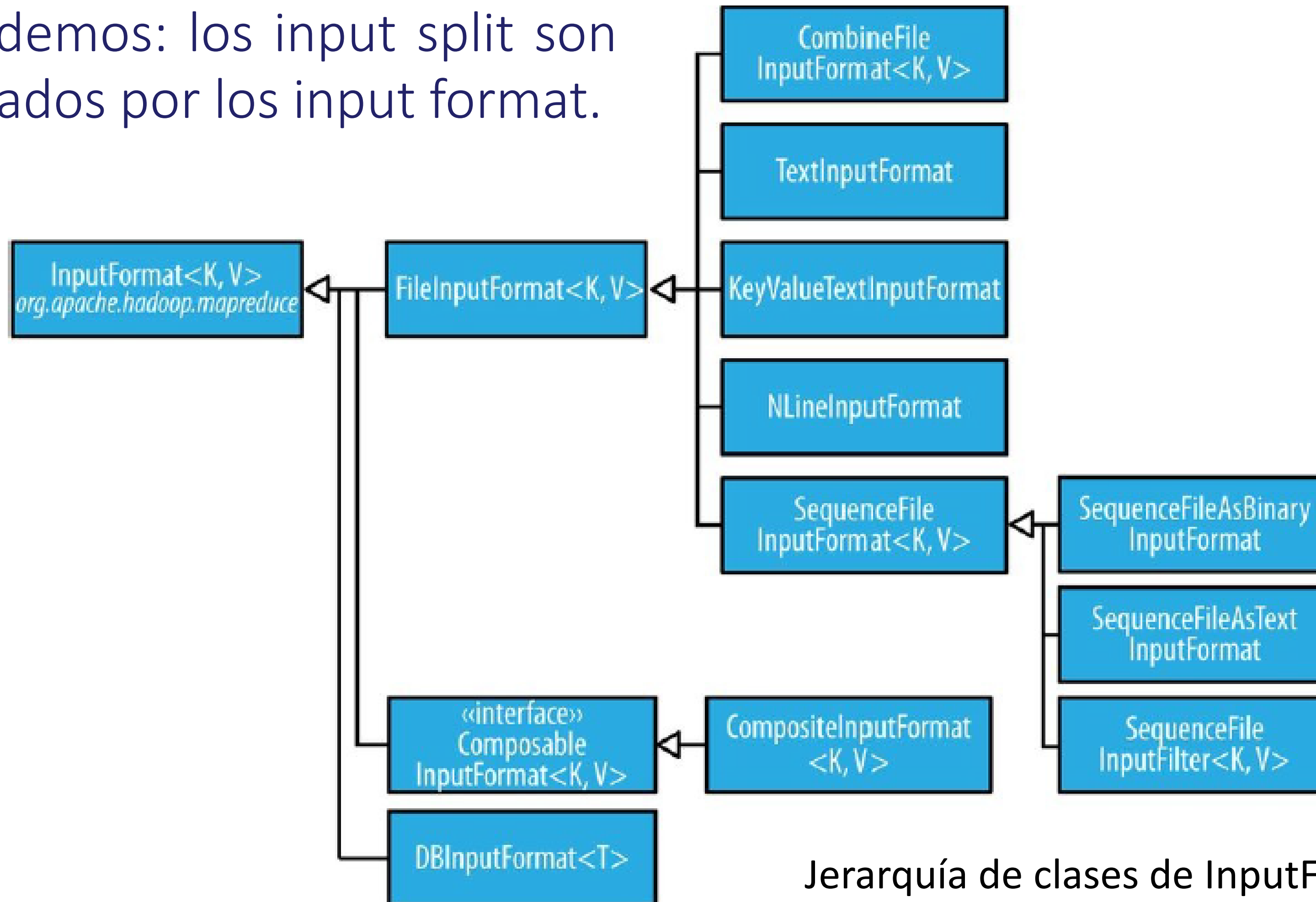
- Balance de carga y ejecución especulativa
- Las fases Map y Reduce son asimétricas por naturaleza, y los diferentes Mappers son también asimétricos entre sí, al trabajar sobre diferentes datos de entrada (por ejemplo, filtrar y procesar logs por IP).
- La asimetría puede causar que algunas tareas map o reduce sean más lentas que otras y puede generarse un cuellos de botella (todos los mapper tienen que teminar antes de iniciar la fase de Reduce).
- En el driver es posible especificar la ejecución especulativa, que analiza el progreso de las tareas y en caso de que no cumpla una cierta tolerancia crea una copia (duplicado exacto) de la tarea para trabajar con los mismos datos.
- Se utilizan los resultados de la tarea que finalice primero.
- Evita que un nodo lento, sobrecargado o inestable se transforme en cuello de botella de la ejecución.

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada

- Recordemos: los input split son generados por los input format.



Jerarquía de clases de InputFormat

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada

- FileInputFormat es la clase base para la implementación de formatos que usan archivos como fuentes de datos.
- Prove métodos para definir qué archivos se incluyen como entrada a un trabajo (input paths) y una implementación para generar splits de los archivos de entrada.
- FileInputFormat prove cuatro métodos para indicar archivos de entrada:

```
public static void addInputPath(Job job, Path path)
public static void addInputPaths(Job job, String commaSeparatedPaths)
public static void setInputPaths(Job job, Path... inputPaths)
public static void setInputPaths(Job job, String commaSeparatedPaths)
```

- Un path puede representar un archivo, un directorio o una colección de archivos y directorios utilizando globs (expresiones regulares).
  - Un path representando un directorio incluye todos los archivos en ese directorio como entrada a un trabajo. No es recursivo !

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada

- FileInputFormat solo crea splits de archivos 'largos' (de mayor tamaño que un bloque de HDFS, 128MB).
- El tamaño por defecto es el de un bloque de HDFS (128MB), pero puede modificarse configurando `mapreduce.input.fileinputformat.split.minsize`, `mapreduce.input.fileinputformat.split.maxsize` (solamente disponible en la versión 2) y `dfs.blocksize`.

Minimum split size	Maximum split size	Block size	Split size	Comentario
1 (default)	`Long.MAX_VALUE` (default)	128 MB (default)	128 MB	Por defecto, el split size es igual al block size de HDFS.
1 (default)	`Long.MAX_VALUE` (default)	256 MB	256 MB	Incrementar el block size de HDFS permite tener splits de mayor tamaño
256 MB	`Long.MAX_VALUE` (default)	128 MB (default)	256 MB	Fijar el mínimo split size por encima del block size permite trabajar con más datos, pero se puede perder la localidad.
1 (default)	64 MB	128 MB (default)	64 MB	Permite tener splits de menor tamaño que el block size de HDFS

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada:

- Archivos pequeños y CombineFileInputFormat
- Hadoop trabaja mejor con pocos archivos de tamaño razonable que con muchos archivos pequeños (FileInputFormat genera splits de tamaño menor o igual a un archivo).
- Si el archivo de entrada es muy pequeño (en relación al bloque de HDFS) habrán muchas tareas map y cada una procesará muy pocos datos, implicando un overhead importante de almacenamiento y transferencia.
  - Si un archivo de 1 GB se divide en 10.000 bloques de 100KB (en lugar de 8 de 128 MB), existirán 10.000 maps que ejecutarán mucho más lentamente.
- CombineFileInputFormat permite empaquetar muchos archivos en cada split, para que cada mapper pueda trabajar con más datos.
- Considera localidad de nodo y de rack para decidir los bloques a incluir en cada split, para no comprometer el desempeño.

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada:

- Archivos completos, WholeFileInputFormat.
- Es posible procesar un archivo completo como un registro, accediendo al contenido completo sin generar splits.
- WholeFileInputFormat define un formato donde las claves no se usan (tipo NullWritable) y los valores son los contenidos de los archivos (tipo BytesWritable):
    1. El formato debe especificar que los archivos de entrada no deben dividirse, sobreescribiendo la función isSplitable() para que retorne false.
    2. Debe implementarse createRecordReader() para retornar una implementación personalizada de RecordReader.
- El mapper puede acceder al nombre del archivo con getPath().

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada:

- TextInputFormat, el InputFormat por defecto de Hadoop.
- Cada registro es una línea del archivo de entrada
  - Clave (LongWritable): offset en bytes desde el inicio del archivo.
  - Valor (Text): contenido de la línea (sin salto de línea).

archivo

```
Cuando mi zarco maceta
se empezó a poner vichoco,
pa que descansara un poco
me compré una bicicleta.
```

⇒

registros (clave,valor)

```
(0,  Cuando mi zarco maceta)
(23, se empezó a poner vichoco,)
(49, pa que descansara un poco
(74, me compré una bicicleta.)
```

- Para detectar líneas corruptas (extremadamente largas) que pueden ocasionar errores de memoria y fallo de las tareas, se puede fijar un máximo largo esperado (en bytes) para las líneas con `mapreduce.input.linerecordreader.line.maxlength`.
- El record reader ignora las líneas que superan el máximo.

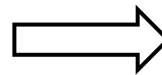
MapReduce: aspectos avanzados

Formatos de entrada:

- Clave-valor, KeyValueTextInputFormat.
- Las líneas del archivo de entrada tienen un formato (con separador)
  - El separador por defecto es el tabulador, puede especificarse otro con `mapreduce.input.keyvaluelinerecordreader.key.value.separator`.
- Los registros identifican los campos considerando el separador.
  - Clave: primer campo.
  - Valor: segundo campo.

archivo

```
L1→Cuando mi zarco maceta
L2→se empezó a poner vichoco,
L3→pa que descansara un poco
L4→me compré una bicicleta.
```



registros (clave,valor)

```
(L1,  Cuando mi zarco maceta)
(L2, se empezó a poner vichoco,)
(L3, pa que descansara un poco
(L4, me compré una bicicleta.)
```

MapReduce: aspectos avanzados

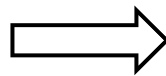
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de entrada:

- Múltiples líneas, NLineInputFormat.
- Con TextInputFormat y KeyValueTextInputFormat cada mapper recibe un número (variable) de líneas.
  - Depende del split size y del largo de las líneas.
- NLineInputFormat permite especificar un número exacto de líneas (N).
  - N se fija con `mapreduce.input.lineinputformat.linespermap`.

archivo

```
Cuando mi zarco maceta
se empezó a poner vichoco,
pa que descansara un poco
me compré una bicicleta.
```



registros (clave,valor)

```
(0,  Cuando mi zarco maceta)        mapper
(23, se empezó a poner vichoco,)      1
(49, pa que descansara un poco      mapper
(74, me compré una bicicleta.)        2
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Otros formatos de entrada:

- XML
  - StreamInputFormat y StreamXmlRecordReader.
  - Deben especificarse los patrones para inicio y fin de tags.
  - Una version avanzada está disponible en Mahout (XmlInputFormat).
- Binarios
  - SequenceFileInputFormat, SequenceFileAsTextInputFormat (convierte a objetos de texto), SequenceFileAsBinaryInputFormat (objetos binarios opacos, encapsulados como BytesWritable), FixedLengthInputFormat (registros binarios de largo fijo sin delimitadores, el largo se indica en fixedlengthinputformat.record.length)
- Base de datos (I/O)
  - DBInputFormat para bases relacionales, usando JDBC.
  - TableInputFormat para interacción con HBase.

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Otros formatos de entrada
- MultipleInputs: permite manejar entradas múltiples (en diferentes formatos) a un trabajo MapReduce
  - Útil para el para procesamiento de diferentes conjuntos de datos, joins, etc.
  - Diferentes formatos (e.g., texto y binario) o el mismo formato con diferente estructura/representación.
- La clase MultipleInputs permite especificar qué InputFormat y qué Mapper utilizer para procesar cada entrada.
- La función addInputPath(JobConf conf, Path path, Class<? extends InputFormat> inputFormatClass, Class<? extends Mapper> mapperClass) agrega un path con un InputFormat y un Mapper a la lista de entradas del job MapReduce.
- Una invocación a addInputPath reemplaza a las invocaciones a FileInputFormat.addInputPath() y job.setMapperClass().

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

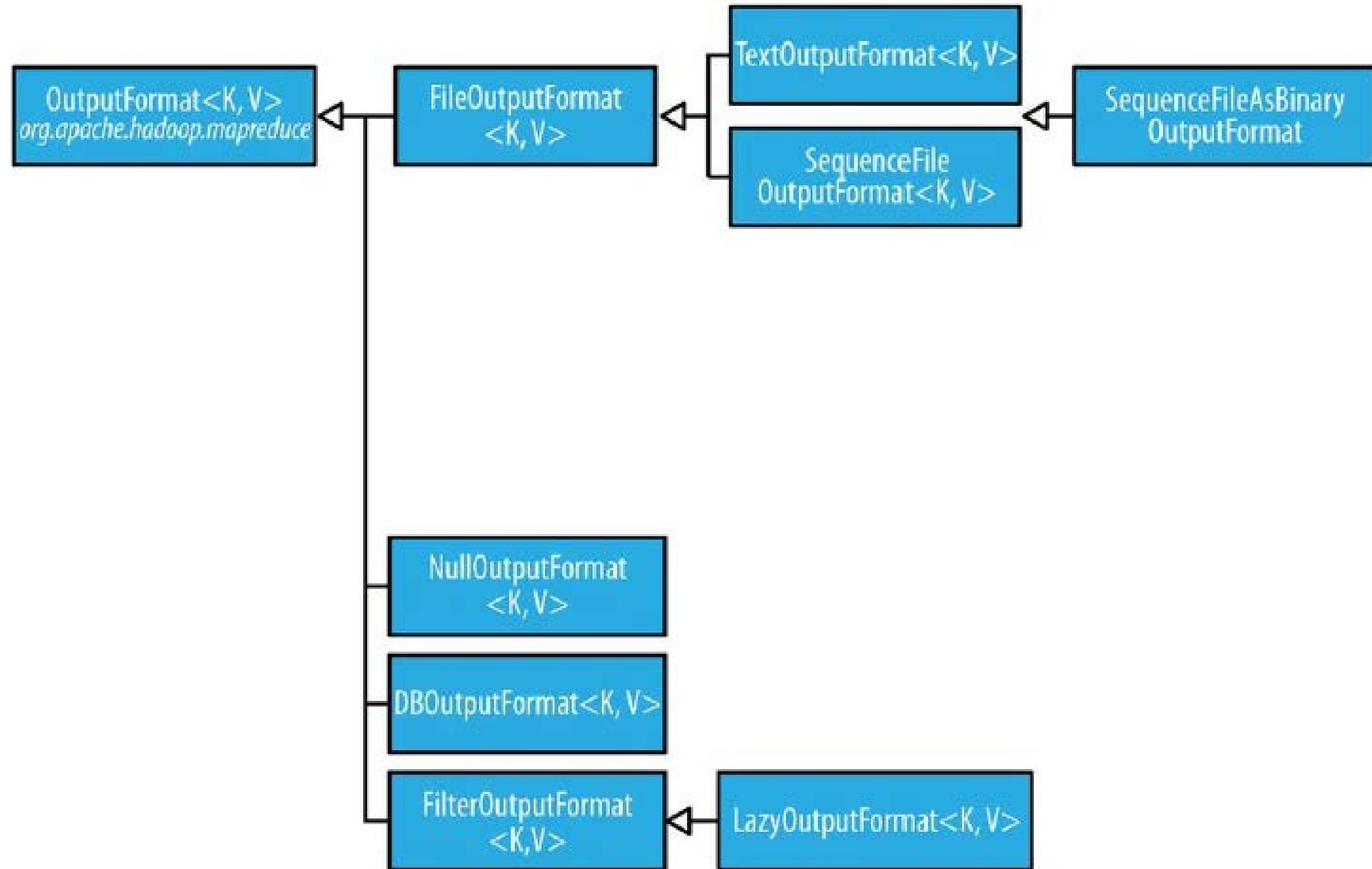
- Ejemplo: procesar datos climáticos de UK Met Office y NCDC

```
MultipleInputs.addInputPath(job,ncdcInputPath,TextInputFormat.class,
MaxTempMapper.class);
MultipleInputs.addInputPath(job,metOffInputPath,TextInputFormat.class,
MetOffMaxTempMapper.class);
```

  - Reemplaza a FileInputFormat.addInputPath() y job.setMapperClass().
  - Ambos son texto (TextInputFormat) pero con diferentes campos. MaxTempMapper lee datos NCDC y extrae la temperatura máxima. MetOffMaxTempMapper lee datos Met Office y extrae año y temperatura.
  - Ambos mappers tienen el mismo tipo de salida, por lo cual los reducers leen los datos agregados sin importer quién los produjo.
- MultipleInputs tiene una versón sobrecargada de addInputPath() que no recibe un mapper: public static void addInputPath(Job job, Path path, Class<? extends InputFormat> inputFormatClass) para usar cuando hay un único mapper (fijado con job.setMapperClass()) que trabaja con multiples formatos.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de salida



MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de salida

- Text Output: el formato por defecto
- Escribe registros como líneas de texto
  - Clave y valor pueden ser de cualquier tipo, TextOutputFormat los convierte en strings utilizando toString().
  - El separador de clave-valor es el tabulador, pero puede cambiarse con la propiedad `mapreduce.output.textoutputformat.separator`.
  - La contraparte de TextOutputFormat para leer es KeyValueTextInputFormat, que separa líneas en pares clave-valor.
  - Se puede suprimir la clave o el valor de la salida usando el tipo NullWritable.
  - En este caso, el separador tampoco se escribe, y la salida puede ser leída utilizando TextInputFormat.
  - Se pueden omitir clave y valor (equivalente al formato NullOutputFormat), que no emite nada.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de salida: formatos binarios

- SequenceFileOutputFormat: escribe archivos (sequence files).
  - Es útil para concatenar con otro trabajo MapReduce (es compacto y se comprime a través de métodos en SequenceFileOutputFormat).
- SequenceFileAsBinaryOutputFormat: escribe pares clave-valor en formato binario (raw) en un sequence file container.
  - Contraparte de SequenceFileAsBinaryInputFormat.
- MapFileOutputFormat: escribe map files (sequence file ordenados con un índice que permite la búsqueda y el acceso por clave).

  Las claves deben agregarse en orden, por lo cual los reducers deben emitir las claves de manera ordenada.
- MultipleOutputs: permite escribir datos en archivos cuyos nombres se derivan de las claves y valores de salida (o de un string arbitrario *name*).
  - Permite a cada reducer crear más de un archivo (name-r-nnnnn).

MapReduce: aspectos avanzados

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Formatos de salida:

- LazyOutputFormat: wrapper de FileOutputFormat que asegura que no se generen archivos de salida vacíos (solo se genera cuando se emite el primer registro para una partición).
  - Se activa con el método setOutputFormatClass(job, output format).

```
if (createLazily) {
   LazyOutputFormat.setOutputFormatClass(job,TextOutputFormat.class);
} else {
   job.setOutputFormat(TextOutputFormat.class);
}
```

- Database Output
  - DBOutputFormat: para crear dumps de salida de trabajos (de tamaño moderado) en una base de datos.
  - TableOutputFormat: para escribir salidas de trabajos en tablas de Hbase.

OTROS EJEMPLOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplos de aplicaciones que pueden ser desarrolladas con MapReduce:
  - BloomFilter (estructura probabilística para chequear pertenencia de un elemento a un conjunto).
  - Grep distribuido (búsqueda de elemento).
  - Sort distribuido (ordenación).
  - Análisis de frecuencia de acceso a una URL.
  - Indices invertidos.
  - PageRank de Google.

MapReduce y bases de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Diferencias entre problemas que se resuelven con MapReduce y con una base de datos relacional

Característica	RDBMS tradicionales	MapReduce
Tamaño de datos	gigabytes	petabytes
Acceso	interactivo y fuera de línea	fuera de línea
Actualizaciones	escribe y lee muchas veces	escribe una vez, lee muchas veces
Estructura	estática	dinámico
Integridad	alta	bajo
Escala	no lineal	lineal

MapReduce y bases de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Una base de datos tradicional en general es una solución complementaria y no sustituta a MapReduce.
- Una base de datos relacional es eficiente para actualizaciones puntuales y para el manejo de datos transaccionales.
- La base de datos relacional también garantiza cierto tiempo de respuesta (al hacer un buen uso de índices y estructura de la DB en disco). Por otro lado, MapReduce está pensado para ser ejecutado en modalidad fuera de línea y no en tiempo real.
- MapReduce es bueno para problemas de grándes volumenes de datos que son escritos una vez y leidos muchas veces.

MapReduce: beneficios

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reduce la complejidad de la sincronización entre procesos.
- Realiza un particionamiento automático de datos.
- Maneja la tolerancia a fallos de forma transparente.
- Maneja el balance de carga.
- Implementación sencilla (el desarrollador solo se debe encargar de implementar la función Map y la función Reduce).
- Permite el análisis de información que anteriormente era inviable (por los tiempos de ejecución o implementación).
- Provee códigos y patrones fácilmente reutilizables.

MapReduce: algunas desventajas

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Enfocado en aplicaciones que ejecutan en modo batch (fuera de línea).
- Puede forzar a 'adaptar' aplicaciones que no siguen estrictamente el modelo MapReduce.
- Puede ser necesario crear etapas adicionales para adaptarse al modelo.
- Puede ser necesario emitir valores intermedios 'extraños' (no intuitivos).
- Puede ser necesario crear funciones superfluas (por ejemplo, funciones map y/o reduce de tipo identidad).

MapReduce de Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El modelo fue extendido para su aplicación a la resolución eficiente de aplicaciones distribuidas para procesar grandes volúmenes de datos.
- Google App Engine es un servicio cloud de plataforma (PaaS) para el desarrollo y hosting de implementaciones distribuidas (web) en datacenters manejados por Google.
- Las aplicaciones ejecutan aisladas (en un sandbox) en múltiples servidores. Google App Engine escala automáticamente las aplicaciones a demanda (dependiendo del número de peticiones o requests).
- App Engine MapReduce es una implementación de Map-Reduce que incluye límites de performance, permitiendo a los desarrolladores controlar los costos de ejecución de sus aplicaciones.
- Soporta Python, Java (y otros lenguages JVM como Groovy, JRuby, Scala, Clojure, etc.), y (en versión experimental) Go y PHP.

MapReduce de Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Procesamiento por etapas:
  1. Lectura de datos: desde storage (u otra fuente). Existen lectores predefinidos (por ej. leer registros de un tipo específico).
  2. Map: ejecuta una vez para cada valor de entrada, retornando una lista de pares (clave, valor). El framework divide la entrada en porciones que son manejadas en paralelo en múltiples instancias de la aplicación.
  3. Shuffle: agrupa los pares (clave, valor) que tienen la misma clave, para pasarlos a la etapa de reducción.
  4. Reduce: ejecuta una vez para cada clave en la lista agrupada de (clave, valor). Retornal una lista de valores que se pasan a la siguiente etapa.
  5. Escritura de salida: un proceso escritor concatena las salidas de la función de reducción en un orden arbitrario y las escribe en un medio de almacenamiento persistente. Es posible elegir entre un conjunto de escritores predefinidos con formatos específicos.

MapReduce: Hadoop vs Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Diferencias entre ambos frameworks

- Hadoop: El reducer recibe una clave y un iterador sobre todos los valores asociados con esa clave no ordenados.
  - En Hadoop, puede 'resolverse' esta limitación formando claves compuestas que incorporen parte del valor, pero se requiere procesamiento adicional.
- Google Apps: Tiene una fucionalidad built-in que permite especificar una clave secundaria para ordenar los valores (si así se desea).

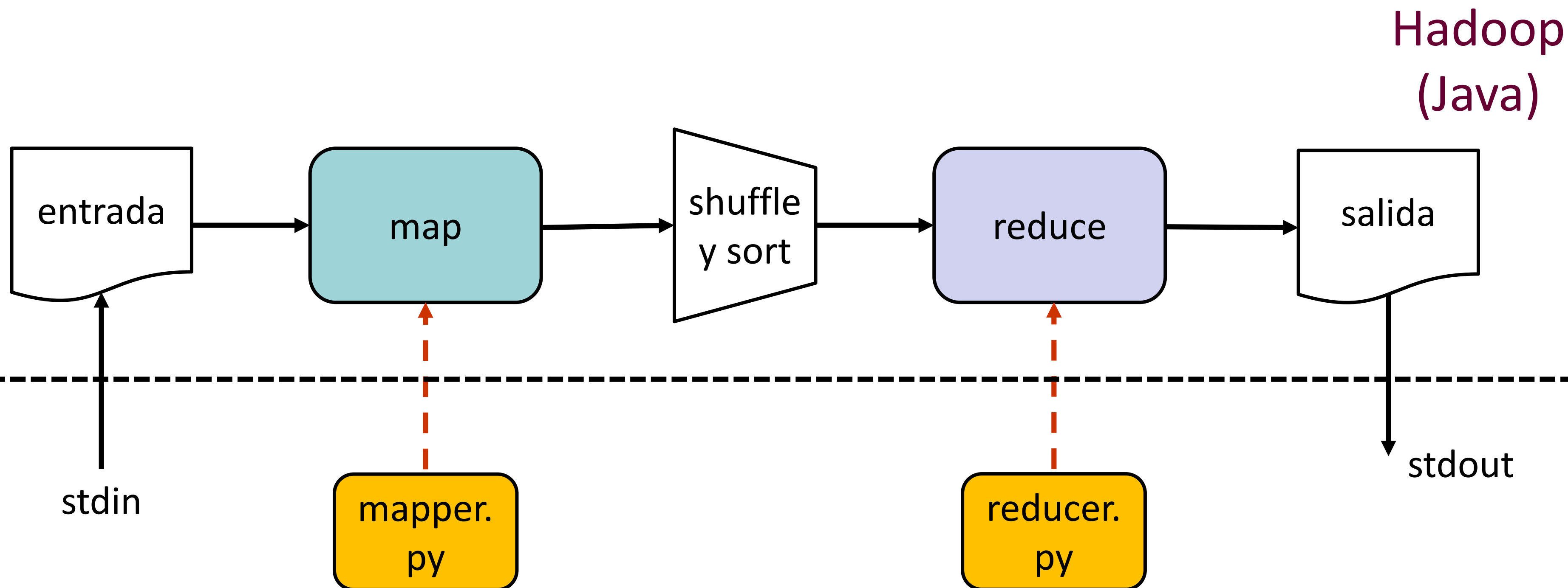
- Google Apps: La clave de salida del Reducer debe ser exactamente la misma que la clave de entrada.
- Hadoop: En Hadoop no existe tal restricción, y el Reducer puede emitir un número arbitrario de pares (clave, valor) de salida, con diferentes claves.

Hadoop Streaming

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop prove una API para MapReduce que permite escribir las funciones map y reduce en lenguajes diferentes a Java.
- Utiliza las herramientas de entrada/salida de Unix y Linux, permitiendo utilizar cualquier lenguaje que pueda leer de entrada estándar y escribir a salida estándar.
- Streaming está muy adaptado para el procesamiento de texto.
  - Los datos llegan a la tarea Map a través de la entrada estándar, se procesan línea por línea y el resultado se escribe en salida estándar, en formato clave-valor (delimitado por tabulador).
  - La entrada para la tarea Reduce corresponde a la salida de Map (formato clave-valor delimitado por tabulador), los pares están ordenados por clave, y se leen de entrada estándar.

Hadoop Streaming



Otros
lenguajes

Hadoop Streaming: Python

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Calcular temperaturas máximas
- Map

```
#!/usr/bin/env python
import re
import sys
for line in sys.stdin:
   val = line.strip() # trim: eliminar espacios al inicio y al final
   (year, temp, q) = (val[15:19], val[87:92], val[92:93])
   if (temp != "+9999" and re.match("[01459]", q)):
      print "%s\t%s" % (year, temp)
```

- El programa itera sobre líneas recibidas por la entrada estándar (stdin, una función del modulo del sistema sys).
- Se toman los valores relevantes de cada línea, se chequea que el valor de temperatura sea válido y se escribe el año y la temperatura, separada por un tabulador.

Hadoop Streaming: Python

- El script solo opera con entrada y salida estándar, se puede testear sin utilizar Hadoop, con pipes de Linux:

```
cat input/ncdc/sample.txt | \
ch02-mr-intro/src/main/python/max_temperature_map.py
```

- La salida es

```
1950 +0000
1950 +0022
1950 -0011
1949 +0111
1949 +0078
```

Hadoop Streaming: Python

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reduce:

```
#!/usr/bin/env python
import sys
(last_key, max_val) = (None, -sys.maxint)
for line in sys.stdin:
   (key, val) = line.strip().split("\t")
   if last_key and last_key != key:
      print "%s\t%s" % (last_key, max_val)
      (last_key, max_val) = (key, int(val))
   else:
      (last_key, max_val) = (key, max(max_val, int(val)))
if last_key:
print "%s\t%s" % (last_key, max_val)
```

- Itera sobre líneas recibidas por la entrada estándar. Almacena el estado (año y temperatura máxima encontrada para ese año) mientras procesa un grupo de registros con la misma clave. La API de Java prove un iterador para cada grupo de claves, en Streaming se deben encontrar los límites entre grupos.
- MapReduce garantiza que los registros están ordenados por clave.

Hadoop Streaming: Python

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se puede testear el pipeline MapReduce en Linux

```
cat input/ncdc/sample.txt | \
ch02-mr-intro/src/main/python/max_temperature_map.py | \
sort | ch02-mr-intro/src/main/python/max_temperature_reduce.py
```

- La salida es la misma que con el programa Java

```
1949 111
1950 22
```

- Ejecución en Hadoop, usando jar y el JAR de Streaming

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-*.jar \
    -input input/ncdc/sample.txt -output output/ \
    -mapper /home/cloudera/hadoop-book-master\
           ch02-mr-intro/src/main/python/max_temperature_map.py \
    -reducer /home/cloudera/hadoop-book-master/ \
           ch02-mr-intro/src/main/python/max_temperature_reduce.py
```

Hadoop Streaming: Python

- Si se ejecuta sobre un gran volumen de datos en un cluster, se puede utilizar la opción -combiner para especificar un combiner:

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-*.jar \
    -input input/ncdc/sample.txt -output output/ \
    -mapper /home/cloudera/hadoop-book-master\
           ch02-mr-intro/src/main/python/max_temperature_map.py \
    -combiner  /home/cloudera/hadoop-book-master/\
           ch02-mr-intro/src/main/python/max_temperature_reduce.py \
    -reducer /home/cloudera/hadoop-book-master/ \
           ch02-mr-intro/src/main/python/max_temperature_reduce.py
```

- La implementación utiliza el mismo reducer como combiner.

Hadoop Streaming: Ruby

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Calcular temperaturas máximas
- Map

```
#!/usr/bin/env ruby
STDIN.each_line do |line|
   val = line
   year, temp, q = val[15,4], val[87,5], val[92,1]
   puts "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
end
```

- El programa itera sobre líneas recibidas por la entrada estándar (STDIN, una variable global de tipo I/O).
- Se toman los valores relevantes de cada línea, se chequea que el valor de temperatura sea válida y se escribe el añó y la temperature, separada por un tabulador.

Hadoop Streaming: Ruby

- El script solo opera con entrada y salida estándar, se puede testear sin utilizar Hadoop, con pipes de Linux:

```
cat input/ncdc/sample.txt | \
ch02-mr-intro/src/main/ruby/max_temperature_map.rb
```

- La salida es

```
1950 +0000
1950 +0022
1950 -0011
1949 +0111
1949 +0078
```

Hadoop Streaming: Ruby

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reduce:

```
#!/usr/bin/env ruby
last_key, max_val = nil, -1000000
STDIN.each_line do |line|
   key, val = line.split("\t")
   if last_key && last_key != key
      puts "#{last_key}\t#{max_val}"
      last_key, max_val = key, val.to_i
   else
      last_key, max_val = key, [max_val, val.to_i].max
   end
end
puts "#{last_key}\t#{max_val}" if last_key
```

- Itera sobre líneas recibidas por la entrada estándar. Almacena el estado (año y temperatura máxima encontrada para ese año) mientras procesa un grupo de registros con la misma clave. La API de Java prove un iterador para cada grupo de claves, en Streaming se deben encontrar los límites entre grupos.
- MapReduce garantiza que los registros están ordenados por clave.

Hadoop Streaming: Ruby

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se puede testear el pipeline MapReduce en Linux

```
cat input/ncdc/sample.txt | \
ch02-mr-intro/src/main/ruby/max_temperature_map.rb | \
sort | ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
```

- La salida es la misma que con el programa Java

```
1949 111
1950 22
```

- Ejecución en Hadoop, usando jar y el JAR de Streaming

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-input input/ncdc/sample.txt -output output \
-mapper ch02-mr-intro/src/main/ruby/max_temperature_map.rb \
-reducer ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
```

Hadoop Streaming: Ruby

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Si se ejecuta sobre un gran volumen de datos en un cluster, se puede utilizar la opción -combiner para especificar un combiner:

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-files ch02-mr-intro/src/main/ruby/max_temperature_map.rb,\
ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb \
-input input/ncdc/all -output output \
-mapper ch02-mr-intro/src/main/ruby/max_temperature_map.rb \
-combiner ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb \
-reducer ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
```

- La opción –files permite especificar los archivos a enviar para ejecución al cluster Hadoop.
- La implementación utiliza el mismo reducer como combiner.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Map-Reduce: Índice invertido (Inverted Index)

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se utiliza para generar un índice de un conjunto muy grande de datos, para poder aplicarlo para realizar búsquedas eficientes en el conjunto.

- Es generalmente utilizado cuando se requiere que consultas sobre determinados textos sean lo más rápidas posible. También es utilizado para la búsqueda de datos y documentos, y es muy aplicado por los motores de búsqueda actuales.

- Un índice invertido almacena las relaciones de datos/palabras/ información a sus ubicaciones en los documentos de destino, bases de datos y otros repositorios de información.

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un ejemplo básico....
  - Dados los textos:
    - T1 = “BigData es un tema … ”
    - T2 = “Si es intersante … ”
    - T3 = “… si es un tema ☺”
  - Se tendrá el siguiente índice invertido:
    - “BigData”: {1}
    - “es”: {1, 2, 3}
    - “un”: {1,3}
    - “tema”: {1,3}
    - “si” : {2,3}
    - “☺” : {3}

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un ejemplo básico....
  - Dados los textos:
    - T1 = “BigData es un tema … ”
    - T2 = “Si es intersante … ”
    - T3 = “… si es un tema ☺”
  - Se tendrá el siguiente índice invertido:
    - “BigData”: {1}
    - “es”: {1, 2, 3} ← ciertas palabras pueden repetirse muchas veces
    - “un”: {1,3}
    - “tema”: {1,3}
    - “si” : {2,3} ← es la comparación sensible a mayúsculas/minúsculas?
    - “☺” : {3} ← como se manejan los símbolos y otros caracteres?

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- MapReduce es muy utilizado para la construcción de índices invertidos.
- Los índices invertidos se guardan en bases de datos relacionales para ser utilizados en búsquedas.
- Permiten el análisis de información no estructurada fuera de línea.
- Realizan un pre-procesamiento de la información para hacerla disponible de forma más ágil.

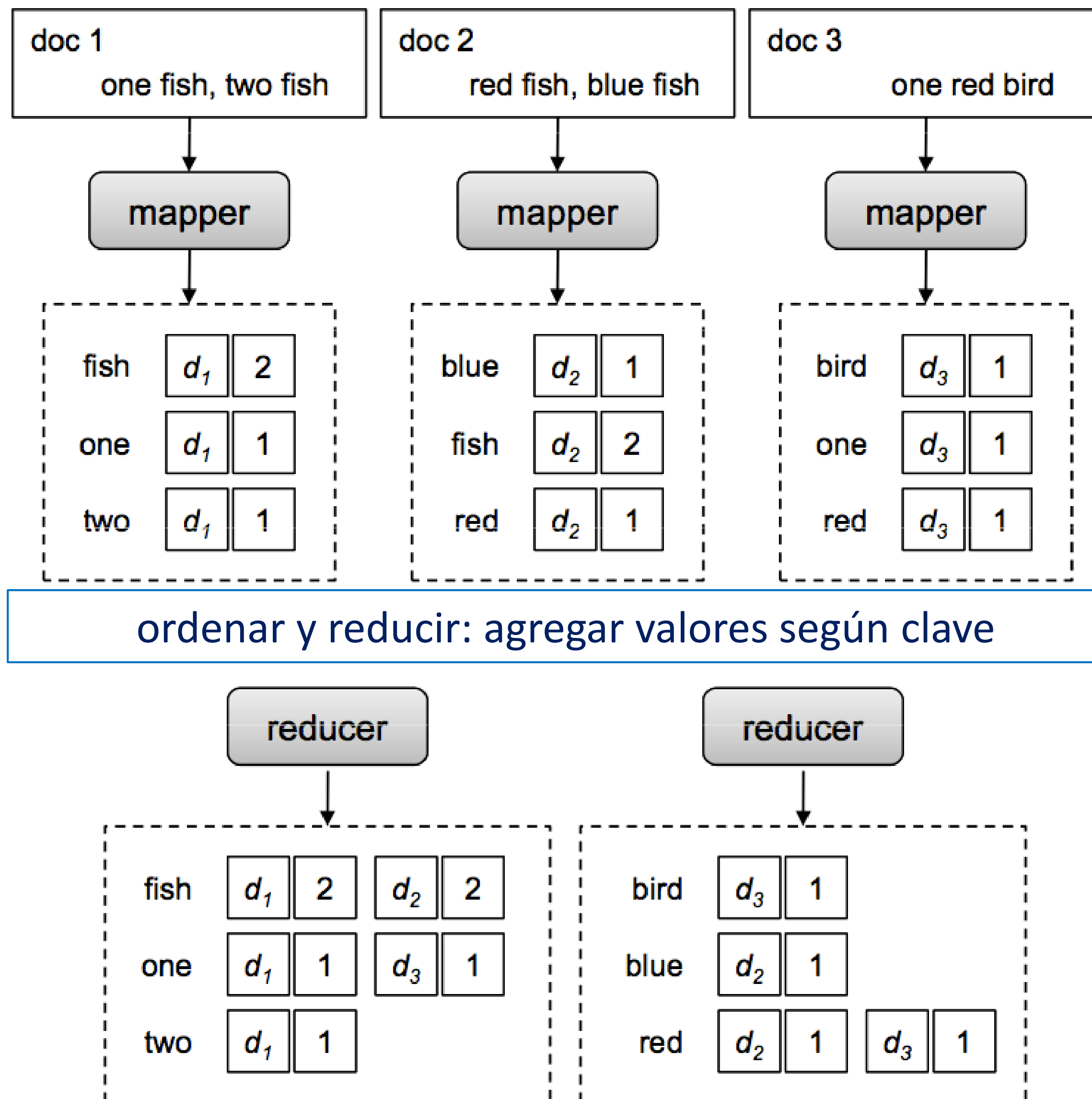
Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un índice invertido, además de la información de ubicación (en que documento está presente), puede contener información adicional:
  - Frecuencia de aparición en el documento.
  - Posición de la palabra en el documento.
  - Información adicional relacionada a la palabra que pueda servir para categorizar su relevancia en el texto (ej: si es un título o un subtítulo o una palabra simplemente, si aparece en negrita, etc).
- La información del índice invertido se puede guardar de diversas maneras: en una base de datos, en archivos de forma semi-estructurada, etc.

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY



- La estructura general para la creación de un índice invertido involucra determinar el mapeo (documento, frecuencia) por palabra.
- El mapper emite clave=palabra, valor=(doc_id, frecuencia).
- El reducer almacena la información de forma conveniente (en una base de datos, concatenando los identificadores de los documentos, etc).

Índice Invertido

- El costo computacional asociado a la creación de un índice invertido está asociado a los mappers.
- Depende también del costo de parsear la información y a la cantidad de claves utilizadas y el balanceo de las mismas.
- Para datos semi-estructurados (json, xml), se tiene un costo adicional al tener que interpretar la información que se está consumiendo.
- Como se vio anteriormente, hay muchas palabras que dependiendo del idioma se repiten demasiado y producen un desbalance de carga ("the" en inglés, "el"/"la" en español). En estos casos, muchas veces se opta por no mapear estas palabras. Lo mismo sucede para las diferencias entre mayúsculas y minúsculas.

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Índice invertido de referencias a Wikipedia en StackOverflow.
- Para cada comentario en StackOverflow, buscando referencias a Wikipedia. Agrupa todos los ID de comentarios que referencian a la misma página de Wikipedia.
  - Puede ser usado para actualizar cada página de Wikipedia con los comentarios que la referencian.

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Índice invertido de referencias a Wikipedia en StackOverflow.
- Para cada comentario en StackOverflow, buscando referencias a Wikipedia. Agrupa todos los ID de comentarios que referencian a la misma página de Wikipedia.
  - Puede ser usado para actualizar cada página de Wikipedia con los comentarios que la referencian.
- Mapper: lee posts en StackOverflow y obtiene los ID de los que referencian a una página de Wikipedia.
- Extrae atributos XML para texto, tipo de post e ID.
- Para tipos que no son respuesta (tipo 2) se busca una URL de Wikipedia, si se encuentra emite (clave = URL, valor = ID).

Índice Invertido: mapper

```
public static class WikiExtractor extends Mapper<Object, Text, Text, Text> {
private Text link = new Text();
private Text outkey = new Text();
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {

    Map<String, String> parsed = MRDPUtils.transformXmlToMap(value.toString());
    // obtener atributos XML
    String txt = parsed.get("Body");
    String posttype = parsed.get("PostTypeId");
    String row_id = parsed.get("Id");
    // saltear posts con cuerpo vacío o posts que son de tipo pregunta (1)
    if (txt == null || (posttype != null && posttype.equals("1"))) {
        return;
    }

    txt = StringEscapeUtils.unescapeHtml(txt.toLowerCase()); // Unescape HTML
    link.set(getWikipediaURL(txt));
    outkey.set(row_id);
    context.write(link, outkey);
    }
}
```

Índice Invertido: reducer

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reducer: itera sobre los valores de entrada y agrega cada ID a un string. Se emite la clave recibida y la concatenación de IDs.

```
public static class Concatenator extends Reducer<Text,Text,Text,Text> {
private Text result = new Text();
public void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {
    StringBuilder sb = new StringBuilder();
    boolean first = true;
    for (Text id : values) {
        if (first) {
            first = false;
        } else {
            sb.append(" ");
        }
        sb.append(id.toString());
    }
    result.set(sb.toString());
    context.write(key, result);
    }
}
```

Puede ser usado como Combiner para concatenar antes de la fase reduce.

MapReduce: Top N

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Mapper

```
public static class TopNMapper extends Mapper {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    @Override public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
        String cleanLine = value.toString().toLowerCase(). \
            replaceAll("[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-:()?!\"']", " ");
        StringTokenizer itr = new StringTokenizer(cleanLine);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken().trim());
            context.write(word, one);
        }
    }
}
```

- Separa palabras y emite (clave = palabra, valor = 1).

MapReduce: Top N

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Reducer

```
public static class TopNReducer extends Reducer {
    private Map countMap = new HashMap<>();
    @Override public void reduce(Text key, Iterable values, Context context)
        throws IOException, InterruptedException {
        // acumular ocurrencias de cada palabra
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        // salvar ocurrencias en el HashMap
        countMap.put(key, new IntWritable(sum));
    }
```

- Calcula la suma de los valores recibidos de los mappers para cada clave (ocurrencia de cada palabra). Clave y suma se agregan a un HashMap.

MapReduce: Top N

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reducer (continuación)

```
    @Override protected void cleanup(Context context)
        throws IOException, InterruptedException {
        Map sortedMap = sortByValues(countMap); // función Java
        int counter = 0;
        for (Text key: sortedMap.keySet()) {
            if (counter ++ == 20) {
                break;
            }
            context.write(key, sortedMap.get(key));
        }
    }
}
```

- El ordenamiento solo debe realizarse luego de recibir todos los valores !
- El método cleanup() es invocado por Hadoop luego que el reducer ha recibido todos los datos. Se ordena el HashMap por valores y se retornan los primeros 20.