

Big data y Hadoop

noviembre 2022

Contenido

1. Consideraciones sobre BigData
2. Hadoop
3. Almacenamiento de datos
4. Tecnología de gestión de datos
5. Estructuras
6. Otras soluciones comerciales

Consideraciones sobre Big Data

Sistemas Distribuidos

Computación en paralelo

Orígenes de datos tradicionales (ERP, CRM...)

Orígenes de datos no tradicionales (IoT, Social...)

El futuro que ya está aquí

- ▶ Debido al crecimiento de las nuevas tecnologías, dispositivos y modos de comunicación como las redes sociales la información esta creciendo rápidamente cada año
- ▶ Estimaciones (sin confirmar) indicaban que la información producida hasta el 2003 era de 5 billones de GB (un campo de futbol lleno de discos)
- ▶ En 2011 se creaba esa información cada 2 días
- ▶ En 2013 se creaba esa información cada 10 minutos
- ▶ En la actualidad es cuestión de segundos

Big Data - Definición

- Concepto hace referencia a los sistemas que manipulan grandes conjuntos de datos. No es una única técnica o tecnología, incluye múltiples áreas de trabajo y tecnologías

Ejemplos

- ▶ Cajas Negras: captura voces de la tripulación de cabina recogida por los micrófonos y la información de rendimiento del avión
- ▶ Redes Sociales: Instagram, Facebook, Twitter... Contienen información publicada por millones de personas
- ▶ Consumos eléctricos: Información de consumos a partir de los contadores inteligentes instalados en viviendas
- ▶ Transportes: Información sobre desplazamientos, modelos, capacidad y distribución de vehículos
- ▶ Buscadores: Obtener información útil a partir de múltiples orígenes distintos

Principales características

- ▶ Heterogeneidad
- ▶ Volatilidad de los datos
- ▶ Volumen
- ▶ Velocidad de generación

Tipos de información

- ▶ Información estructurada: bases de datos relacionales
- ▶ Información semiestructurada: archivos de datos en formatos como XML y JSON
- ▶ Información No estructurada: documentos de texto, archivos PDF, archivos de texto, logs de redes sociales.

Retos

- ▶ Captura de datos (ingesta)
- ▶ Depuración de contenidos (Curation)
- ▶ Almacenamiento de tanta información
- ▶ Capacidad para realizar búsquedas eficientes
- ▶ Compartición
- ▶ Posibilidad de llevar a cabo análisis efectivos
- ▶ Visualización de los datos

Oportunidades

- ▶ Hallar correlaciones aplicando algoritmos más complejos y proporcionando unos niveles de visibilidad que trasforman por completo el contacto de la empresa con su entorno, otorgándole la capacidad de descubrir sus oportunidades.

Oportunidades

- ▶ Agencias de marketing están analizando la información de redes sociales como respuesta a sus campañas en las mismas o en otros medios
- ▶ Empresas de producción y comercializadores analizan las preferencias y la percepción de los consumidores, organizando así su producción
- ▶ Analizando la información de los historiales médicos los hospitales buscan dar un mejor servicio a sus pacientes

Tecnologías

► NoSQL

- Ofrece capacidades operativas para trabajar en tiempo real con altos volúmenes de datos
- Estas tecnologías están diseñadas para obtener ventaja de las nuevas arquitecturas de computación en la nube
- Permite niveles altos de procesamiento con unos costes menores
- Un ejemplo podría ser MongoDB

Tecnologías

- ▶ MPP (Massively Parallel Processing)
 - ▶ Permiten añadir capacidades analíticas para realizar complejos procesos de análisis con la información
 - ▶ Permiten el escalado en el proceso en múltiples nodos, desde un solo punto hasta miles de máquinas
 - ▶ Ejemplo: MapReduce

Soluciones Tradicionales

- ▶ Las empresas almacenan y procesan la información en un servidor o un cluster de servidores.
- ▶ El usuario trabaja con un sistema centralizado de gestión que opera contra una base de datos relacional.
- ▶ Funciona bien con volúmenes de datos que pueden ser procesados por un servidor
- ▶ Desventaja es que no está pensado para crecer exponencialmente ni en almacenamiento ni en procesamiento de la información

Soluciones Big Data

- ▶ Utilización de algoritmos que dividan la tarea en partes pequeñas y que cada una pueda ejecutarse en diferentes ordenadores, agrupando el resultado en un conjunto de resultados únicos
- ▶ El usuario interactúa con el sistema que procesa la información en paralelo en un número variable de ordenadores y sobre diferentes fuentes de datos
- ▶ Google diseño la aproximación que se convirtió en la base del proyecto Hadoop

Hadoop

- ▶ Introducción al universo Hadoop
- ▶ Hadoop MapReduce

Ecosistema Hadoop

Principales elementos

Ecosistema

- ▶ Colección de herramientas para solucionar los diferentes aspectos del Big Data
- ▶ Existen una gran variedad de herramientas que hace difícil cual utilizar
- ▶ La variedad permite localizar la herramienta más acertada para cada entidad

MapReduce

- ▶ Framework de trabajo básico
- ▶ Establece la base para procesar grandes cantidades de datos
- ▶ Características
 - ▶ Tareas Mapeo, seleccionar la información
 - ▶ Tareas de Reducción, obtienen los datos necesarios

Yarn

- ▶ Es un elemento central que ofrece los servicios de cluster distribuido
 - ▶ Global Resource Manager
 - ▶ Gestor de recursos globales de la aplicación.
 - ▶ Existe uno por cluster
 - ▶ Node Manager
 - ▶ Gestiona los recursos de cada nodo
 - ▶ Se comunica con el administrador de recursos

HBASE

- ▶ Es un sistema de bases de datos no relacional
- ▶ Utiliza HDFS para almacenar los datos de forma persistente
- ▶ Es un sistema de columnas, con filas similar a los sistemas relacionales
- ▶ Cada celda tiene versiones que identifica la celda
- ▶ Perfecto para ordenar información

HIVE

- ▶ Minería de datos
- ▶ Es una herramienta pensada para el proceso por lotes de información
- ▶ Utiliza HDFS y MapReduce
- ▶ Organiza la información de tres maneras
 - ▶ Tablas, similares a las relacionales
 - ▶ Particiones, una tabla puede tener particiones
 - ▶ Cubos, permite almacenar la información en archivos

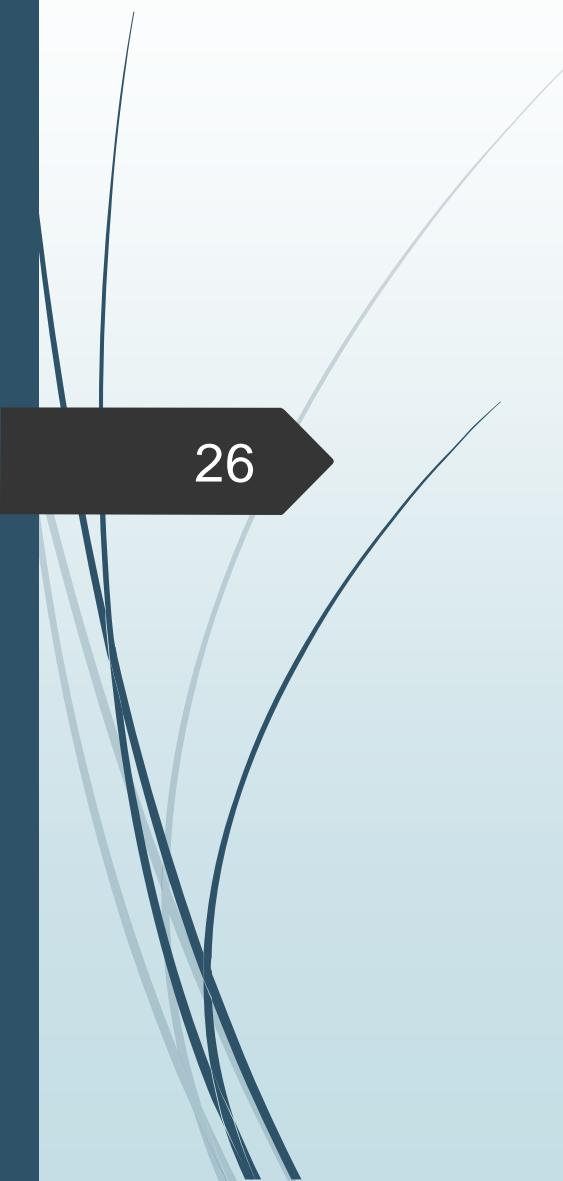
- ▶ Lenguaje de programación que permite leer datos procesarlos y transformarlos en la salida deseada
- ▶ Permite la ejecución en modo local en una sola máquina o en modo Hadoop utilizando MapReduce
- ▶ También se pueden lanzar los programas de tres maneras
 - ▶ Script
 - ▶ Grunt, interprete de comandos
 - ▶ Embebidos en programas java

Sqoop

- ▶ Permite mover datos entre SQL y Hadoop
- ▶ Funciona con un interprete en la línea de comandos
- ▶ Algunas características son
 - ▶ Importación por bloques
 - ▶ Entrada directa
 - ▶ Interacción con datos
 - ▶ Exportación de datos

Zookeeper

- ▶ Permite coordinar todos los elementos de las aplicaciones distribuidas
- ▶ Coordinación de procesos
- ▶ Administración de la configuración
- ▶ Comunicación entre nodos



26

MapReduce

Conceptos

Conceptos

- ▶ Modelo de programación para procesamiento de datos
- ▶ Hadoop admite código en Java, Ruby, Python
- ▶ Programación distribuida por diseño

Un problema

- ▶ Obtener la temperatura máxima por año a partir de las lecturas de las estaciones
- ▶ Cada año de cada estación en un .gz
- ▶ Cada fichero texto plano según el National Climatic Data Center Records

Soluciones no hadoop

- ▶ Utilizando AWK en Unix podemos pensar:
 - ▶ Procesar diferentes años en diferentes procesos, nos podemos encontrar con grandes diferencias de tamaños
 - ▶ Combinar los resultados, el problema es dividir la información con cargas similares
 - ▶ Limitación de ejecución de la máquina

Aproximación mapreduce

- ▶ Establecer dos fases
- ▶ Fase I, Establecer los elementos a extraer
- ▶ Fase II, Reducir los resultados al objetivo
- ▶ Cada fase utiliza la pareja key/value para el input y output
- ▶ Programar funciones, Map, Reduce y ejecución del trabajo

Ejemplo MAP

- Seleccionar sólo la información relevante, utilizando como key el offset del archivo

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```



Ejemplo reduce

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

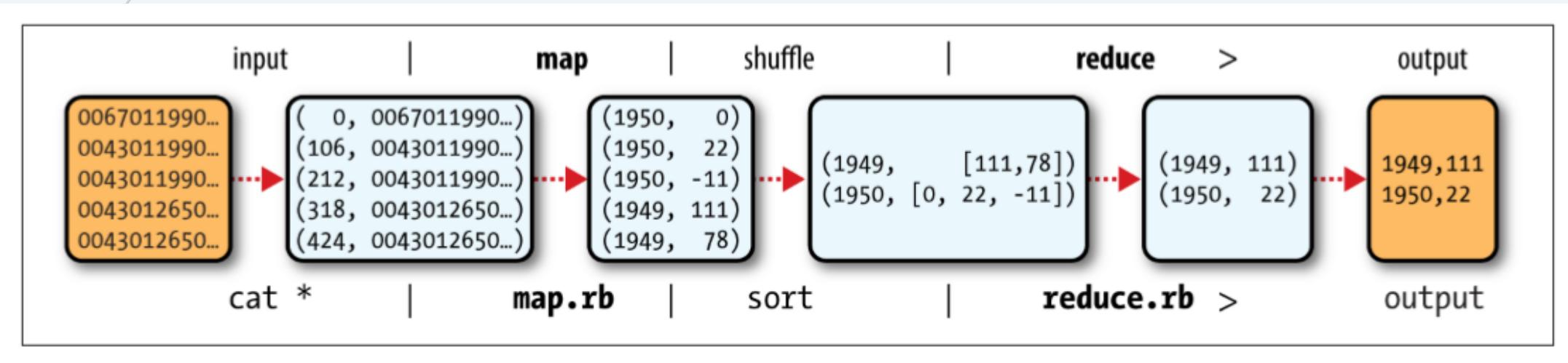


(1949, [111, 78])
(1950, [0, 22, -11])



(1949, 111)
(1950, 22)

FLujo



Maper Ejemplos de código

```
import java.io.IOException;

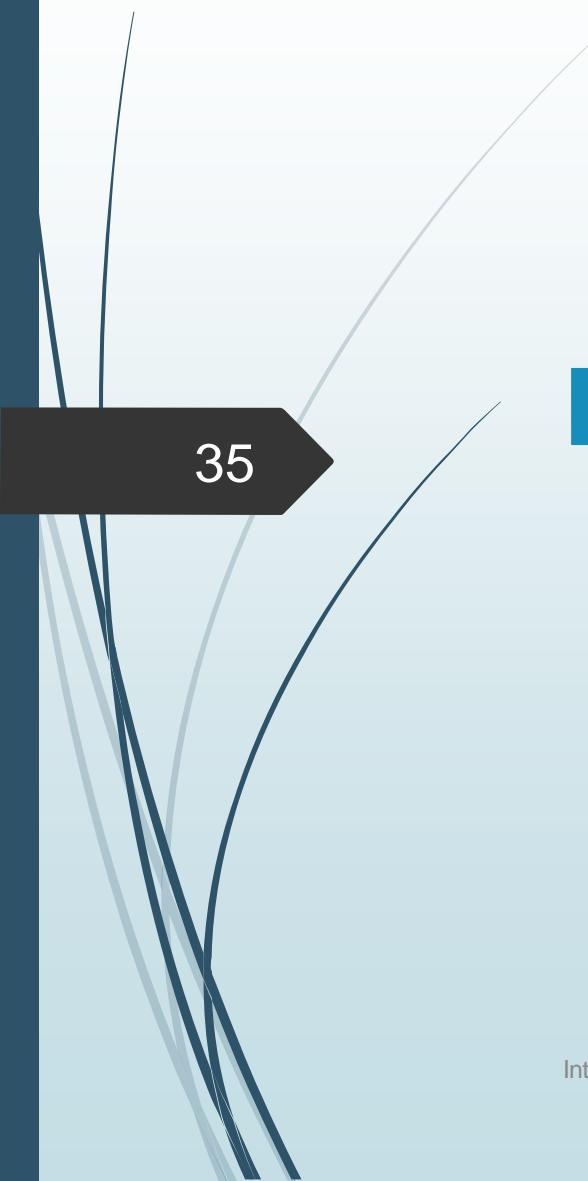
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

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));
        }
    }
}
```



35

Escalado

Escalado

- ▶ Trabajo MapReduce: unidad de trabajo que se tiene que ejecutar
- ▶ Contiene
 - ▶ Datos de entrada
 - ▶ Programa Mapreduce
 - ▶ Configuración
- ▶ Hadoop divide el trabajo en dos tipos de tareas
 - ▶ Tareas Map
 - ▶ Tareas Reduce

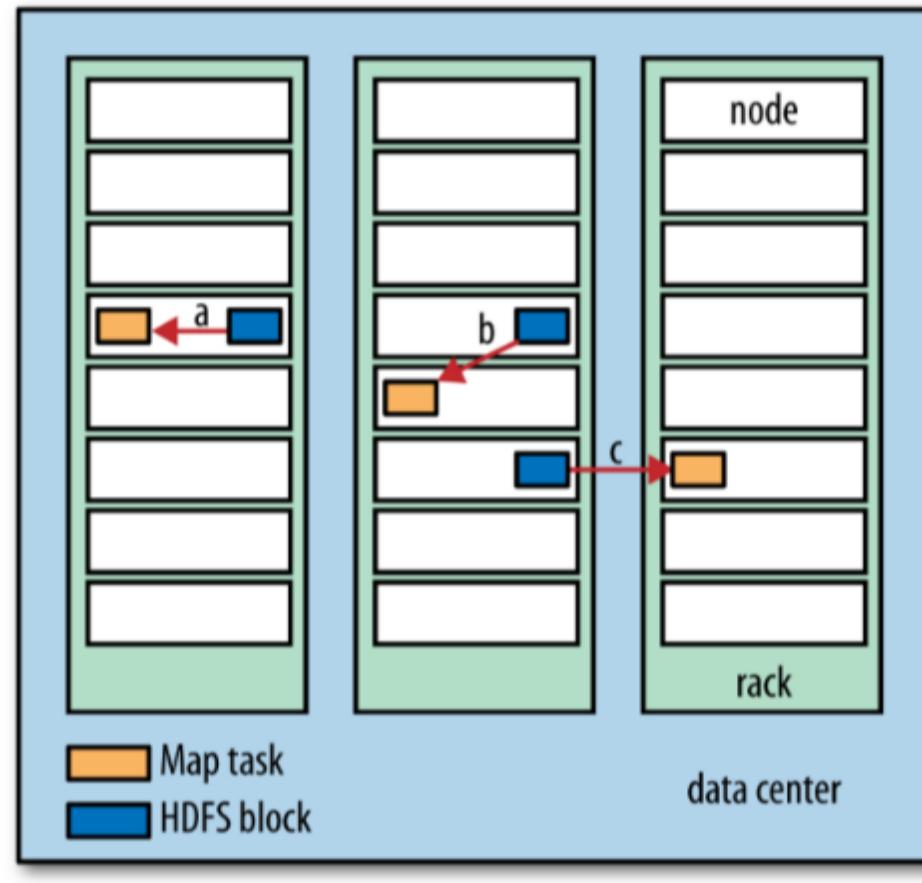
Mapreduce y yarn

- ▶ La ejecución de las tareas se programa con YARN
- ▶ Si la tarea falla se ejecuta en otro nodo diferente
- ▶ Hadoop divide el input en splits y crea una tarea Map para cada Split de información
- ▶ El tiempo de proceso mejora alternando varios Split
- ▶ Si son demasiado pequeños el tiempo de gestión domina el tiempo de ejecución
- ▶ Un buen tamaño es 128 MB

Optimización

- ▶ Para mejorar la ejecución es mejor optimizar la ubicación
- ▶ Recomendable ejecutar la tarea Map con la información en HDFS
- ▶ También puede estar la información en otro nodo y en otro rack
- ▶ El tamaño de 128 MB coincide con el bloque de HDFS

Optimización



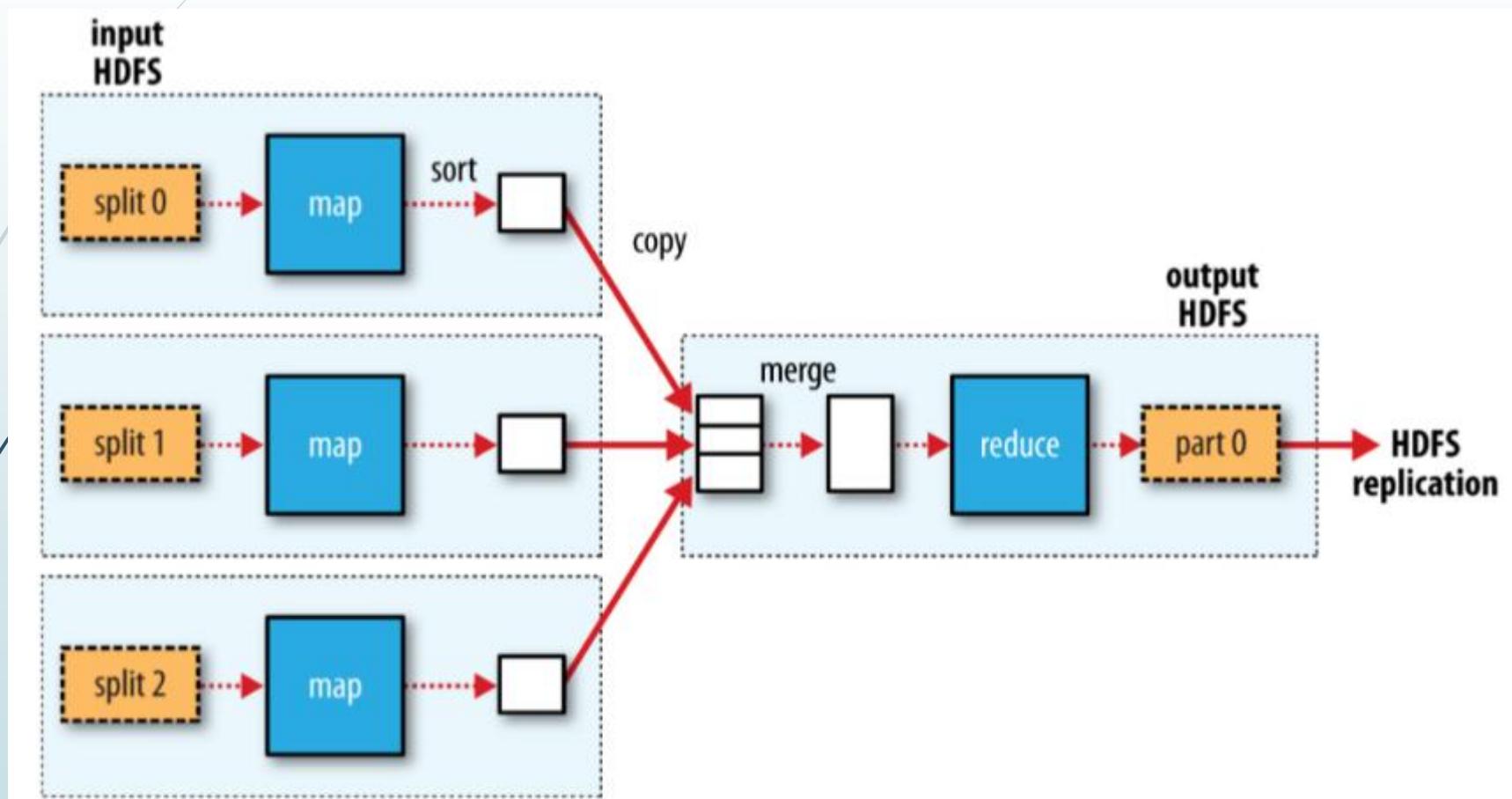
Ejecución tareas map

- ▶ Las tareas Map escriben la salida en disco local no en HDFS
- ▶ La salida de la tarea Map es intermedia y se reduce para el resultado final
- ▶ Si el nodo falla, se lanza otra tarea map en otro nodo

Ejecución tareas Reduce

- ▶ El input de una tarea local no es con datos con ubicación
- ▶ El input es el output de las tareas Map
- ▶ Los output de la tarea Map se transfieren por la red al nodo de la tarea Reduce
- ▶ El Output de la tarea Reduce se almacena en HDFS por fiabilidad.

Ejecución de tareas



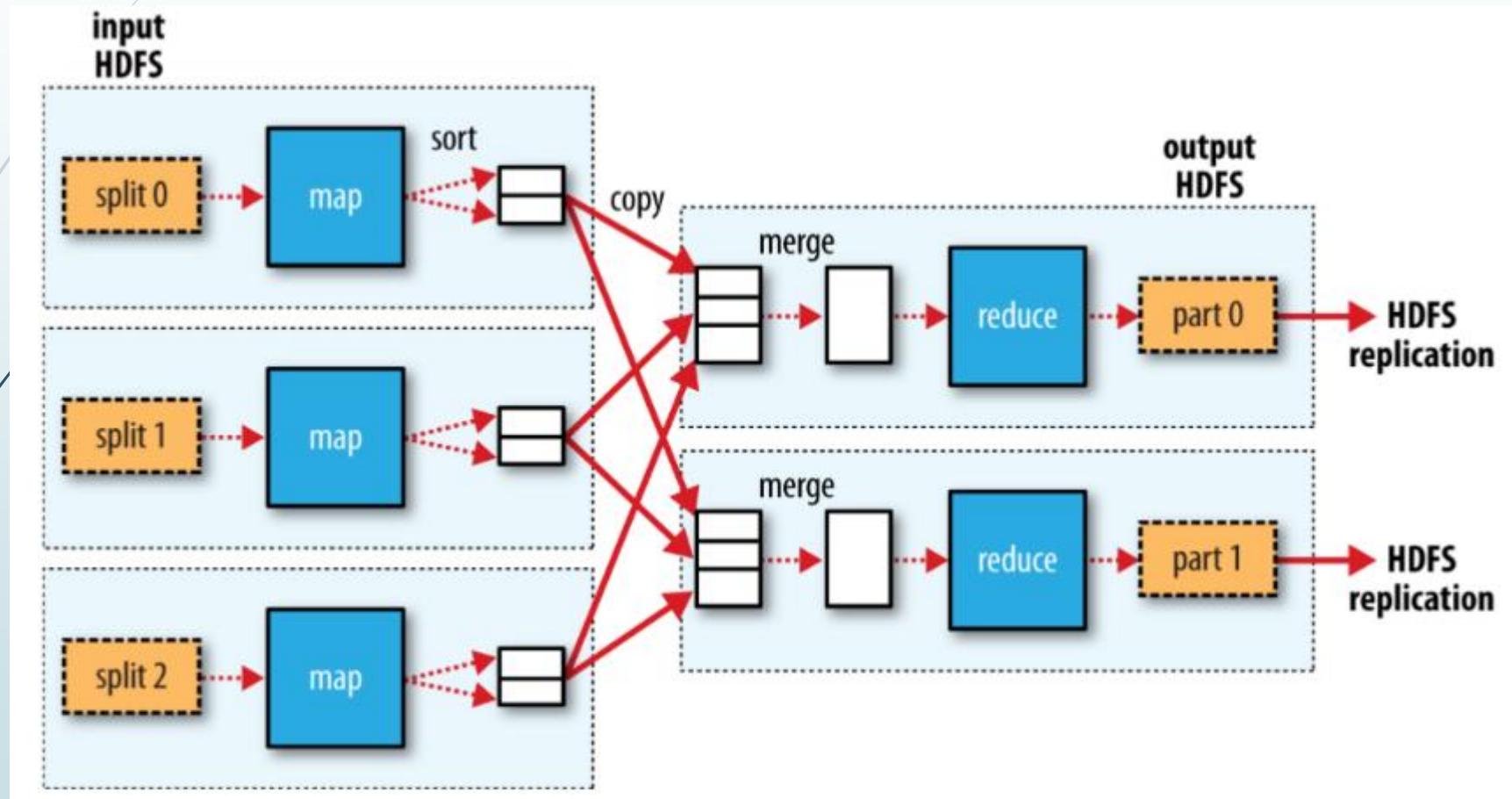
Ejecución Reduce

- ▶ El número de tareas Reduce influye en el rendimiento
- ▶ El objetivo es lograr que generen bloques de HDFS de información útil
- ▶ El número de tareas reduce no depende el tamaño de la información de entrada

Tarea Reduce Por defecto

- ▶ Cuando hay múltiples tareas Reduce las taras Map reducen su output
- ▶ Puede haber varios pares Key/value en cada partición
- ▶ Toda los value están en la misma partición que su Key

Varias Tareas Reduce



Funciones de combinación

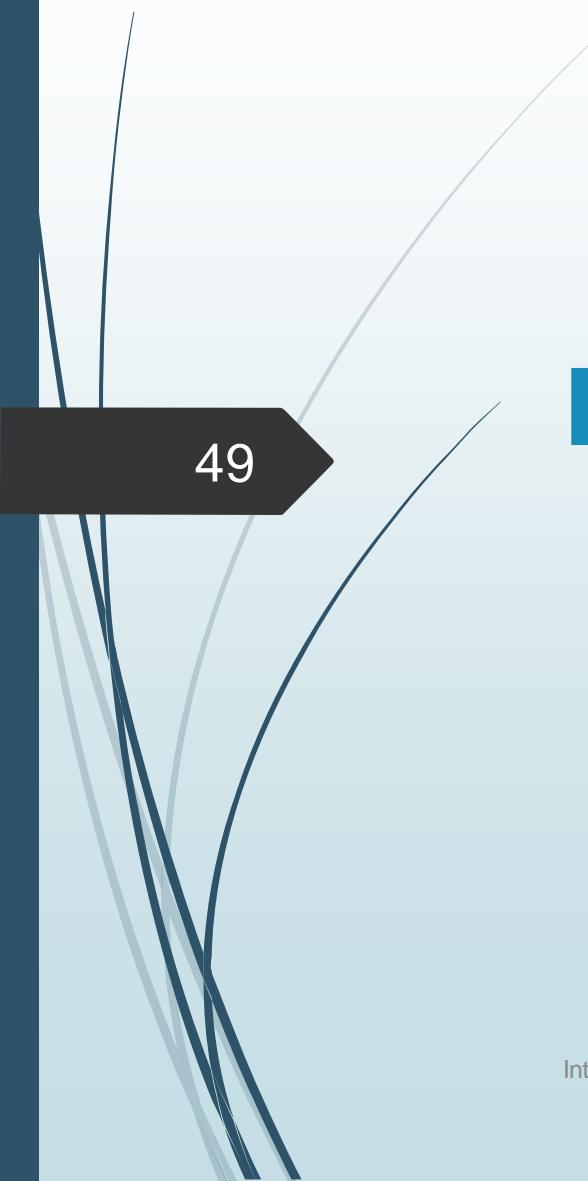
- ▶ Para evitar problemas de ancho de banda creamos funciones de combinación
- ▶ Reduce el tráfico de información entre las tareas Map y Reduce
- ▶ Hadoop permite establecer funciones que se ejecuten en el output de las tareas Map, pero no garantiza el número de veces que se ejecutarán
- ▶ El resultado de la tarea Reduce es el mismo

Concepto función de combinación

- ▶ Tarea Map A (1950, 0) (1950, 20) (1950, 10)
- ▶ Tarea MAP B (1950, 25) (1950, 15)
- ▶ La función de reducción se le pasaría: (1950, [0, 20, 10, 25, 15]) with output: (1950, 25)
- ▶ Podemos crear una función de reducción que le pase (1950, [20, 25])
- ▶ Y generaría el mismo resultado

Ejemplo de código

```
public class MaxTemperatureWithCombiner {  
  
    public static void main(String[] args) throws Exception {  
        if (args.length != 2) {  
            System.err.println("Usage: MaxTemperatureWithCombiner <input path> " +  
                "<output path>");  
            System.exit(-1);  
        }  
  
        Job job = new Job();  
        job.setJarByClass(MaxTemperatureWithCombiner.class);  
        job.setJobName("Max temperature");  
  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
        job.setMapperClass(MaxTemperatureMapper.class);  
        job.setCombinerClass(MaxTemperatureReducer.class);  
        job.setReducerClass(MaxTemperatureReducer.class);  
  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
  
        System.exit(job.waitForCompletion(true) ? 0 : 1);  
    }  
}
```



49

Hadoop Streaming

Hadoop Streaming

- ▶ Utiliza los streams estándar de Unix
- ▶ Pensado para proceso de textos
- ▶ Datos de input de las tareas Map se pasan sobre la entrada estándar que procesa línea a línea
- ▶ El input para la función de reducción tiene el mismo formato. Lee las líneas ordenadas por Key y escribe el resultado en la salida estándar

Ejemplo Python tarea MAP

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

import re
import sys

for line in sys.stdin:
    val = line.strip()
    (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)
```

Ejemplo Python tarea 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)))

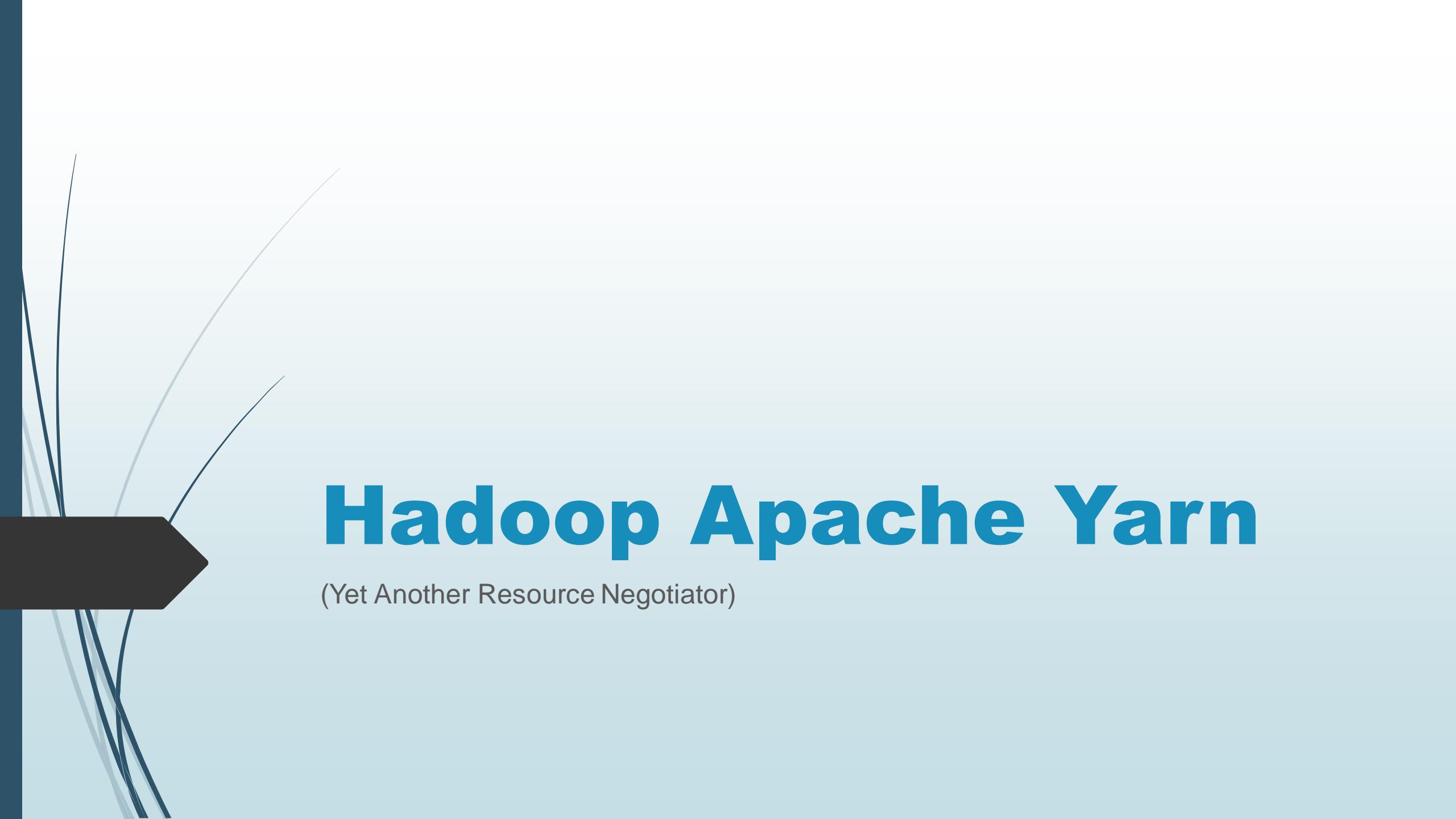
if last_key:
    print "%s\t%s" % (last_key, max_val)
```

Configuración

- ▶ Utiliza la clase Configuration org.apache.hadoop.conf
- ▶ Se representa por una colección de propiedades y sus valores
- ▶ Principales archivos
 - ▶ core.-site.xml → información básica de la configuración
 - ▶ Hdfs-site.xml → configuración, replicación, infraestructura

Hadoop Pipes

- ▶ Es la interface de C++ a MapReduce
- ▶ En vez de utilizar input/output estándar utiliza sockets
- ▶ Se definen
 - ▶ función Map (HadoopPipes::MapContext...)
 - ▶ Función Reduce(HadoopPipes::ReduceContext...)
- ▶ Objeto Context, obtiene los medios para leer y escribir
- ▶ Key/Values son buffers de bytes → hay que convertirlos al tipo que queramos
- ▶ El proceso Main llama a HadoopPipes RunTask



Hadoop Apache Yarn

(Yet Another Resource Negotiator)



56

Introducción

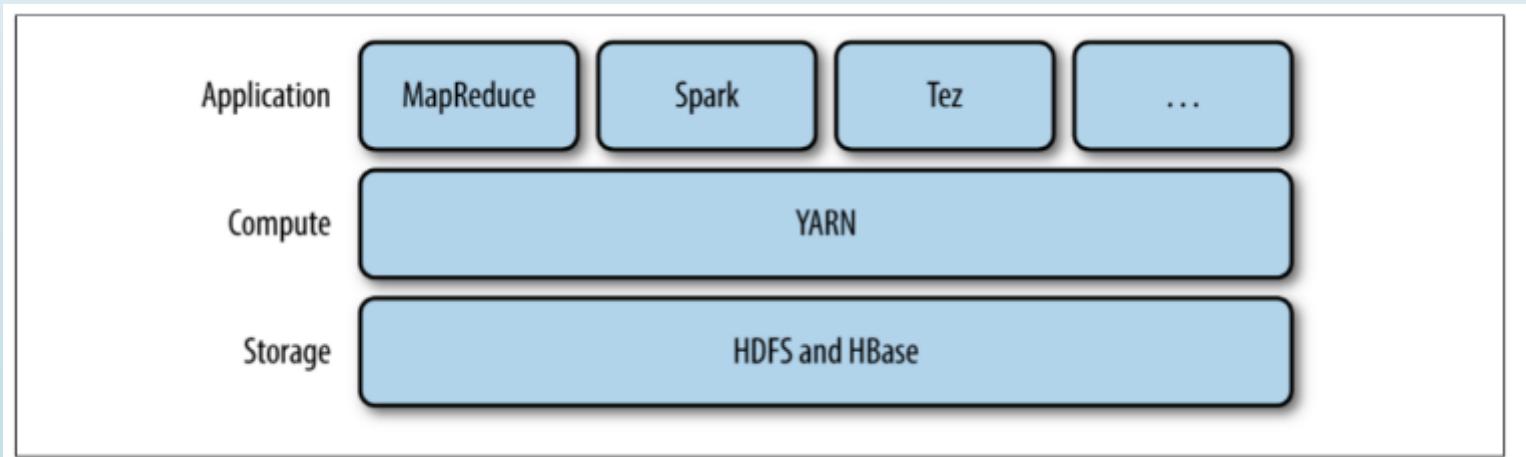
Fundamentos

Introducción

- ▶ Gestor de recursos del cluster de Hadoop
- ▶ Aparece en la versión 2.0 para mejorar el rendimiento de MapReduce
- ▶ Ofrece API's para trabajar con los recursos del cluster
- ▶ No se utiliza directamente por el código del usuario

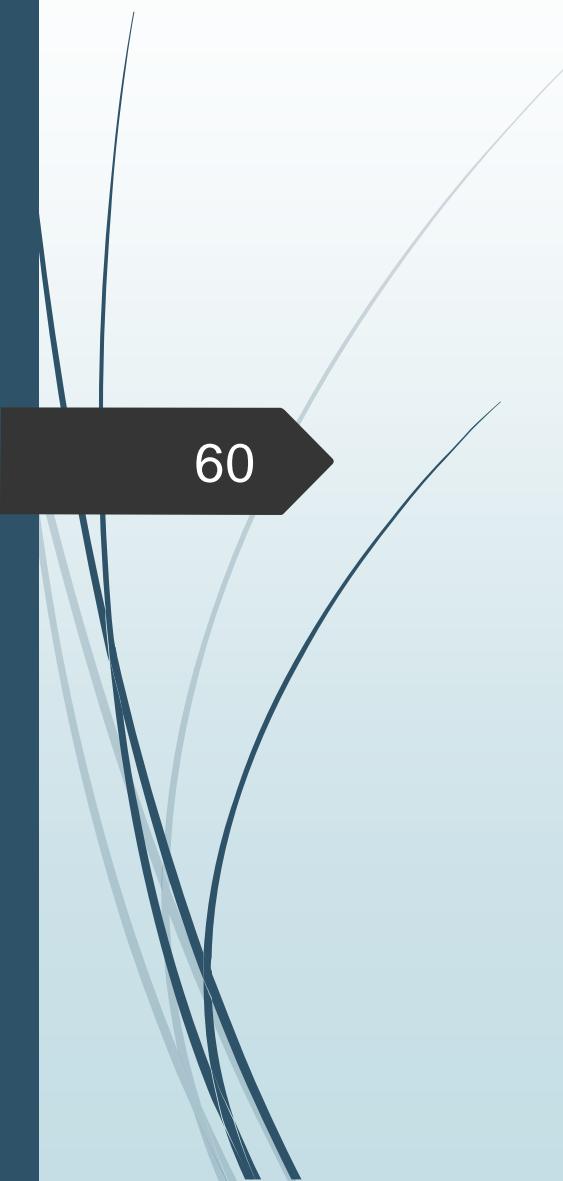
Introducción

- ▶ El código de usuario se escribe sobre frameworks de desarrollo (MapReduce y Spark)
- ▶ Ofrecen capa de aplicación para otras herramientas (Pig, Hive...)



Introducción

- ▶ Utiliza dos daemon para ofrecer sus servicios
 - ▶ Administrador de recursos
 - ▶ Uno por cluster
 - ▶ Gestionar el uso de recursos
 - ▶ Administrador de nodos
 - ▶ Ejecuta en todos los nodos
 - ▶ Monitorizar los contenedores
- ▶ Contenedor → ejecuta un trabajo con recursos concretos

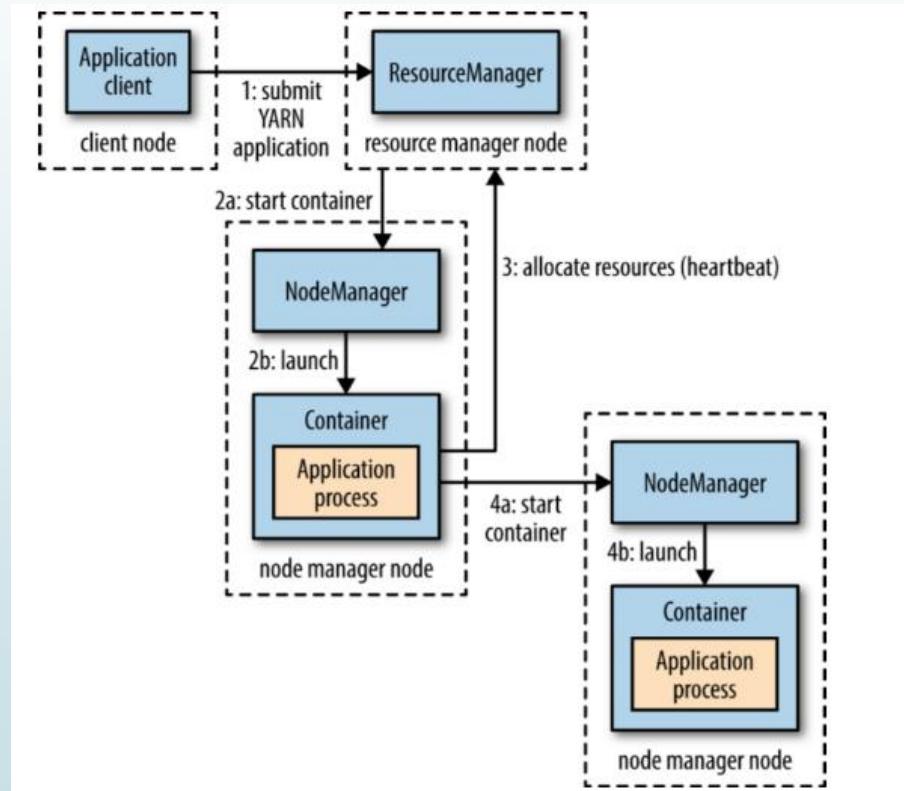


60

YARN - EJECución

Proceso de ejecución y petición de recursos

Ejecución



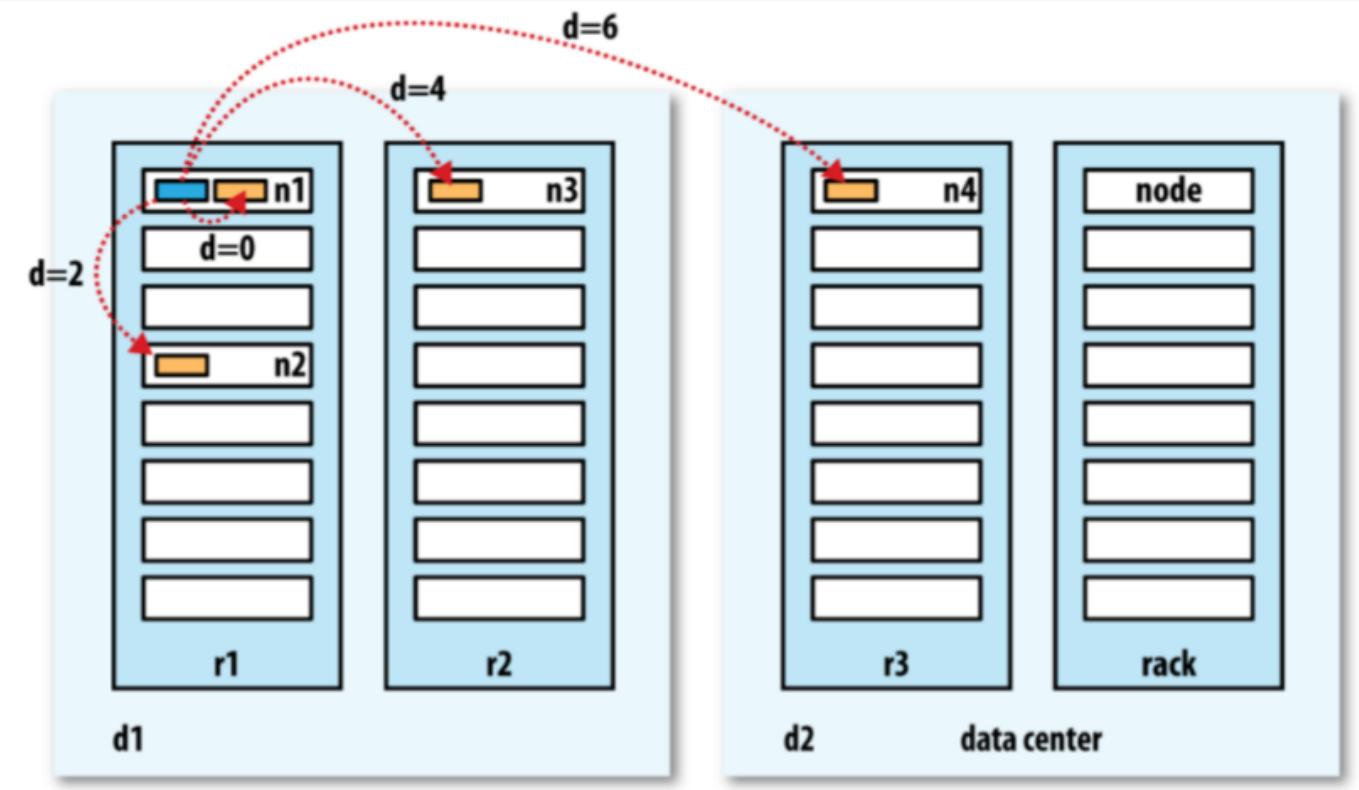
YARN Petición de recursos

- ▶ Es un modelo flexible
- ▶ Se puede indicar los recursos requeridos para cada contenedor (CPU, memoria)
- ▶ También puede establecer limitaciones a la ubicación
- ▶ La ubicación se considera crítica para el rendimiento al utilizar el ancho de banda efectivamente

YARN Recursos

- ▶ Con alto volumen de datos, la transferencia de datos puede ser limitador
- ▶ Hadoop calcula el ancho de banda por aproximación
- ▶ Calcula la distancia entre sus nodos representando la red en un árbol
- ▶ La distancia es el número de nodos a su antecesor común más cercano
 - ▶ Procesos en el mismo nodo
 - ▶ Diferentes nodos en el mismo rack
 - ▶ Nodos en diferentes rack en el mismo nodo

Yarn cálculo distancia



Yarn – Limitaciones de ubicación

- ▶ Es necesario para mejorar el rendimiento
- ▶ Especificar ubicación del contenedor
- ▶ Ubicación relajada → cuando no se puede poner en el nodo
 - ▶ Por ejemplo, si no puede en el nodo, lo intenta en el mismo rack o si no en el mismo cluster

Yarn ubicación

- ▶ Las aplicaciones pueden hacer la petición en cualquier momento
- ▶ Spark lo hace al principio, reservando todos los contenedores
- ▶ MapReduce lo hace sobre la marcha cuando hace la tarea de mapa y luego cuando tiene que hacer la tarea de Reducción

Yarn Programación

Establecer los diferentes sistemas para ejecutar los trabajos

Yarn Tiempo de vida

- ▶ Las aplicaciones pueden ser de segundos o meses
- ▶ Clasificación por como se organizar los trabajos
 - ▶ Una aplicación por trabajo de usuario (MapReduce)
 - ▶ Una aplicación por flujo de trabajo (Spark)
 - ▶ Una aplicación de larga duración que se comparte entre usuarios (Impala)

Yarn programación

- ▶ En situación ideal las peticiones siempre se atienden
- ▶ En la realidad, recursos limitados → tiempos de espera
- ▶ YARN tienen varias maneras de establecer recursos
 - ▶ FIFO
 - ▶ Capacidad
 - ▶ Programación Justa (Fair)

FIFO (first in first out)

- ▶ Trabajos en cola en orden de llegada
- ▶ Es la más simple y no necesita configuración
- ▶ No encaja en cluster compartidos con trabajos largos y cortos

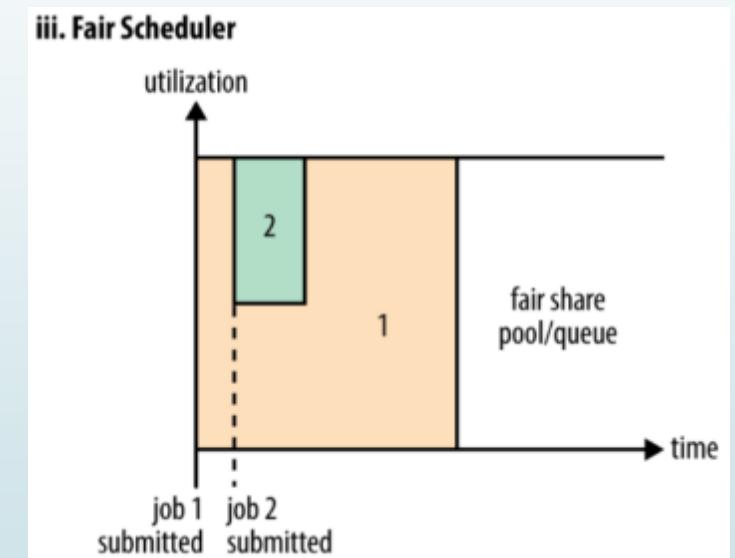
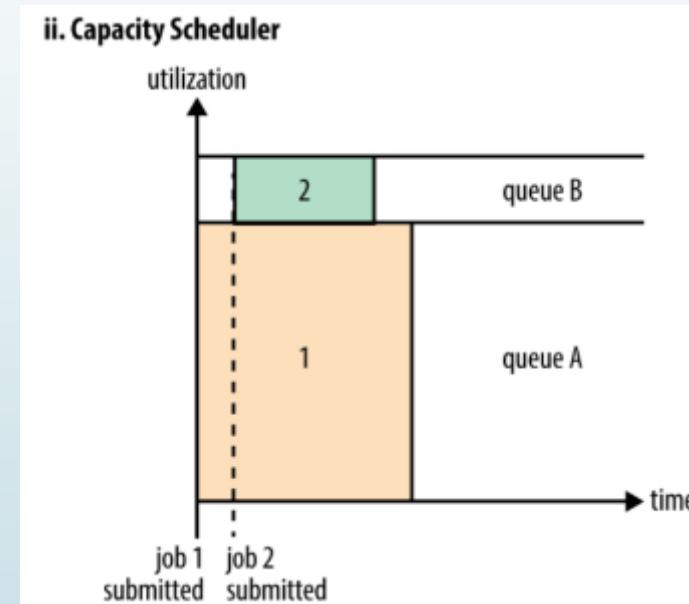
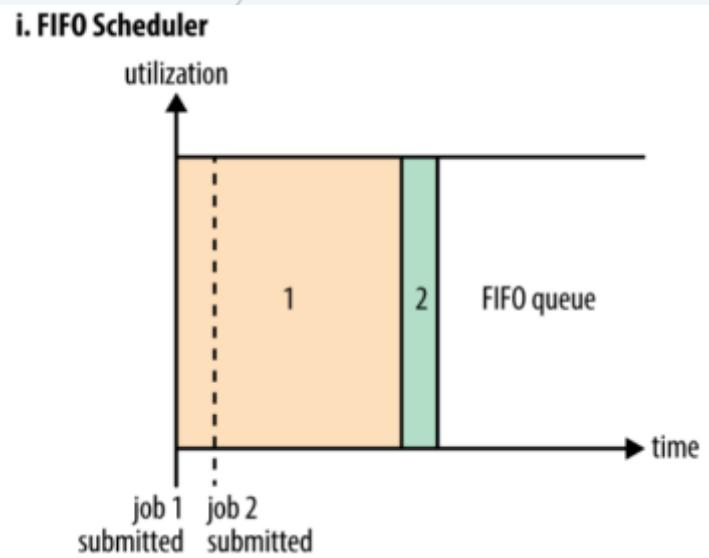
YARN Capacidad

- ▶ Dispone de una cola específica para trabajos cortos
- ▶ Globalmente el rendimiento es menor
- ▶ Trabajos largos acaban más tarde que con FIFO

YARN Programación FAir

- ▶ No se reserva capacidad
- ▶ Dinámicamente reparte los recursos
- ▶ Hay tiempo de demora en la asignación
- ▶ Ejecución de trabajos grandes y pequeños menor que los anteriores

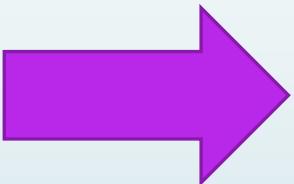
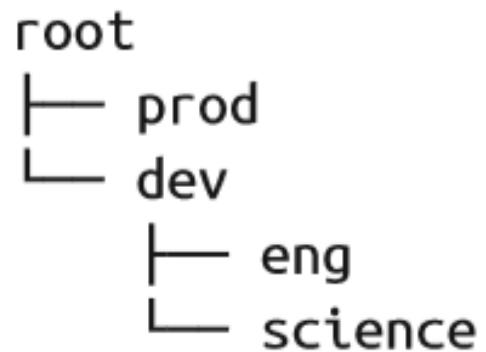
YARN Programación



YARN Configuración

- ▶ La programación por capacidad permite establecer líneas con unos recursos preasignados
- ▶ Las colas pueden dividirse por jerarquías
- ▶ Dentro de cada cola → FIFO
- ▶ Elasticidad de las colas → Superar su capacidad si hay otra que no está en uso.

Yarn Configuración



```
<configuration>
  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>prod,dev</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.queues</name>
    <value>eng,science</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>40</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>60</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.maximum-capacity</name>
    <value>75</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.eng.capacity</name>
    <value>50</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.science.capacity</name>
    <value>50</value>
  </property>
</configuration>
```

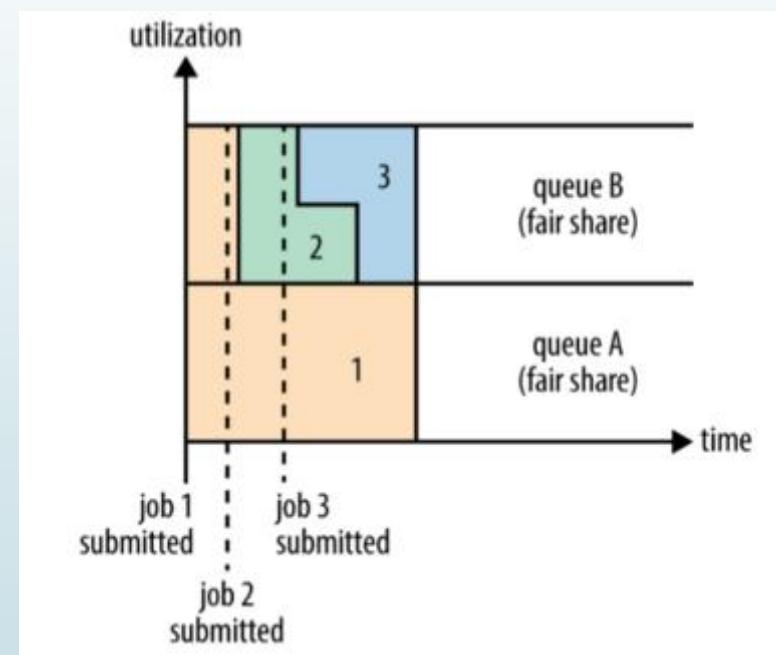
YARN Configuración

- Definir la cola depende del FrameWork
 - MapReduce → mapreduce.job.queue.name
 - Si no existe la cola genera un error
 - Si no se especifica la cola se pone en la cola “default”

YARN Configuración FAIR

- ▶ Intenta asignar recursos para todas las aplicaciones
- ▶ Puede hacer reparto en la misma cola o entre colas
- ▶ Si una cola no está en uso toma los recursos
- ▶ Si entra un trabajo se ejecuta en la cola correspondiente
- ▶ Si hay más trabajos en la misma cola se reparte

Yarn Configuración



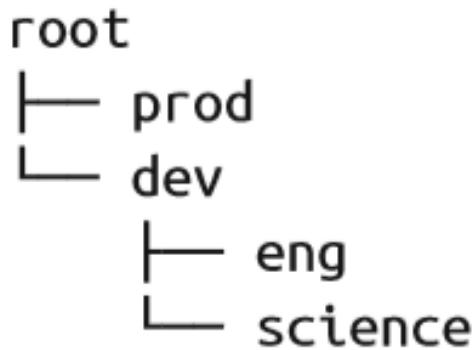
YARN configuración FAIR

- ▶ Normalmente se utiliza la programación por capacidad
- ▶ Modificar yarn-site.xml, yarn.resourcemanager.scheduler.class
- ▶ La configuración de la cola utiliza fair-Schedule.xml
- ▶ Si no hay archivo de configuración se crea una cola por usuario la primera vez que se ejecuta una aplicación
- ▶ Se puede establecer jerarquías

Yarn Configuración fair

- ▶ Reglas para establecer las colas
 - ▶ Específica: nombre de la cola
 - ▶ Usuario: el nombre de usuario de ejecución
 - ▶ Default: cuando no se ejecuta ninguna

YARN Configuración FAir



```
<allocations>
  <defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>

  <queue name="prod">
    <weight>40</weight>
    <schedulingPolicy>fifo</schedulingPolicy>
  </queue>

  <queue name="dev">
    <weight>60</weight>
    <queue name="eng" />
    <queue name="science" />
  </queue>

  <queuePlacementPolicy>
    <rule name="specified" create="false" />
    <rule name="primaryGroup" create="false" />
    <rule name="default" queue="dev.eng" />
  </queuePlacementPolicy>
</allocations>
```

Yarn configuración fair

- ▶ Cuando un trabajo se envía a una cola vacía en un cluster ocupado tiene que esperar.
- ▶ Preemption permite hacer el cálculo de cuando va a quedar libre
- ▶ Permite acabar con el procesamiento de un contenedor para liberar recursos
- ▶ Hay que configurar `yarn.scheduler.fair.preemption` a True y asignarles tiempos
 - ▶ `DefaultMinSharePreemptionTimeOut`
 - ▶ `FairSharePreemptionTimeOut`



83

YARN retrasos

Gestión de los retraso

Yarn Retrasos

- ▶ Cuando se asigna un trabajo es posible que esté ocupado → asigna a otro nodo
- ▶ Este proceso de asignación se puede retrasar para esperar que se libere
- ▶ Soportado por programación por capacidad y Fair

YARN Retrasos

- ▶ Cada nodo envía un pulso al administrador de recursos cada segundo
- ▶ Envía información sobre contenedores y recursos disponibles
- ▶ Propiedad DelayScheduling indica el número máximo de pulsos
 - ▶ `yarn.scheduler.capacity.node-locality-delay`
 - ▶ `yarn.scheduler.fair.locality.threshold.nod`

YARN Recurso Dominante

- ▶ Si hay un solo recurso la capacidad es fácil de calcular
- ▶ Cuando hay más de un recurso (memoria, cpu)
- ▶ YARN mira el recurso dominante para cada usuario

YarN DRF

- ▶ Un cluster con 100 CPU, 10 TB de memoria
- ▶ Aplicación A pide contenedores 2CPU, 300 GB
- ▶ Aplicación B pide contenedores 6 CPU y 100 GB
- ▶ A quiere 2%,3% del cluster, memoria es dominante
- ▶ B requiere 6%,1%, cpu es dominante
- ▶ Como B es el doble en el recurso dominante, le asignaran la mitad que al A en una programación FAIR

Hadoop

HDFS Hadoop Distributed File System

Introducción

Conceptos generales

introducción

- ▶ Problema: Conjunto de datos que crece por encima de la capacidad de almacenamiento de una máquina física
- ▶ Solución: Sistema de archivos distribuido
- ▶ Aumenta complejidad por ser un sistema en red
- ▶ Reto: evitar perdida de información por fallo de nodos

Intro

- ▶ HDFS está diseñado para almacenar archivos muy grandes con patrones de acceso a datos en streaming

Intro

- ▶ Archivos muy grandes → Cientos de MB o GB
- ▶ Datos en streaming → patrón 1 escritura , varias lecturas
- ▶ Hardware Comun → económico, genérico, con posibilidad de errores

Intro

- ▶ Entorno desaconsejados para HDFS
 - ▶ Tiempos de latencia bajos <10 ms
 - ▶ Muchos archivos pequeños → los metadatos se almacenan en memoria
 - ▶ Cada entrada 150 bytes, almacena archivos, directorios, bloques
 - ▶ 1 Millón archivos de 1 bloque → 300 MB
 - ▶ 1 Billón archivos →
 - ▶ Multiples escrituras y lecturas → HDFS sólo se escribe por un proceso al final del archivo



94

HDFS Bloques

Conceptos básicos

HDFS Bloques

- ▶ Un disco tiene un tamaño mínimo que puede ser escrito o leído
- ▶ Los archivos se dividen en bloques múltiplos de los del disco
- ▶ Tamaño de los bloques del disco 512 byte
- ▶ Trasparente al usuario

HDFS Bloques

- ▶ El mismo concepto distinto tamaño
- ▶ Normalmente 128 MB
- ▶ Archivos con bloques que pueden estar en diferentes unidades
- ▶ Un archivo de tamaño menor que el bloque no ocupa todo el bloque
 - ▶ Si el archivo es de 1 MB ocupa 1MB no 128 MB

HDFS Bloques

- ▶ Tamaño 128 MB
- ▶ Busca minimizar el tiempo de búsqueda
- ▶ Si el bloque es grande el tiempo de copiar es muy superior al de buscar
- ▶ Permite copiar a la velocidad de transferencia

HDFS Bloques

- Para un tiempo de búsqueda de 10 ms y una velocidad de trasferencia de 100 MB/s si queremos que el tiempo de búsqueda sea el 1% de la trasferencia el bloque tiene que ser de 100 MB

HDFS Bloques

- ▶ HDFS trabaja con nivel de abstracción a la hora de manejar bloques
- ▶ Beneficios
 - ▶ Un archivo puede ser mayor que un disco, los bloques se pueden almacenar en cualquier disco del cluster
 - ▶ Tratar como unidad de abstracción el bloque en vez del archivo simplifica la gestión
 - ▶ Tienen un tamaño fijo para los cálculos
 - ▶ Los metadatos no se almacenan con los datos

HDFS Bloques - tolerancia

- ▶ Trabajar con bloques permite realizar replicación para mejorar tolerancia a fallos
- ▶ Se replican en máquinas físicas distintas (3 normalmente)
- ▶ Si uno falla, otro puede ser leído de forma transparente
- ▶ Un bloque corrompido puede sustituirse para mantener el factor de replicación
- ▶ Herramienta para consultar bloques “fsck”

101

HDFS nameNode y datanode

La clave de hdfs

HDFS Bloques

- ▶ HDFS tiene dos tipos de nodos
- ▶ Trabajan con el patrón maestro-obrero
- ▶ Maestro → NameNode
- ▶ Obrero → DataNode

HDFS NameNode

- ▶ El cálculo de memoria para un NameNode es aproximado
- ▶ Tiene una referencia de cada bloque de archivos
- ▶ Depende del número de bloques por archivo, longitud, directorios, versión
- ▶ Una estimación 1000MB de memoria 1 Millón de bloques
- ▶ Hadoop-env.sh. Se puede configurar el tamaño en JVM (Java Virtual Machine)
 - ▶ Hadoop-Namenode_OPTS y Hadoop-Secondary_OPTS

HDFS Bloques – NameNode

- ▶ Mantiene el árbol del sistema de archivos
- ▶ Almacena de forma persistente en dos archivos
 - ▶ Namespace image
 - ▶ Edit Log
- ▶ Conoce la ubicación de los DataNode, pero no los guarda de forma persistente
- ▶ El código de usuario no necesita saber de estos elementos para funcionar

HDFS tolerancia

- ▶ Sin el NameNode no se puede recuperar el sistema de archivos
- ▶ Hadoop tiene dos métodos para tolerancia a fallos
 - ▶ Backup → los archivos que hacen persistente el NameNode.
 - ▶ Hace varias copias de los archivos
 - ▶ Asincrónos y atómicos
 - ▶ Normalmente escribir en el disco local y en sistema de archivos remoto

HDFS Bloques - DATAnode

- ▶ Son el caballo de trabajo
- ▶ Almacena y recupera bloques cuando se lo indican los clientes o NameNode
- ▶ Reporta al NameNode periódicamente la lista de nodos que almacena

HDFS Tolerancia

- ▶ NameNode Secundario
 - ▶ No actua como NameNode
 - ▶ Establece puntos de control cruzando el Edit Log con el Namespace Image para que el log no crezca demasiado
 - ▶ Se suele ejecutar en otra máquina por las necesidades de hardware

hDFS caching

- ▶ Normalmente la información se lee del disco
- ▶ Por frecuencia de la lectura se pueden cachear
- ▶ Se cachea en un DataNode en memoria
- ▶ Se mejora el rendimiento (MapReduce y Spark)
- ▶ Las aplicaciones de usuario especifican los archivos y el tiempo

HDFS FEderation

- ▶ NameNode tiene información de todo el sistema de archivos
- ▶ Al ser muy grande puede ser una limitación para el escalado del sistema
- ▶ HDFS Federation permite usar varios NameNode
- ▶ Cada uno gestiona una parte del sistema
- ▶ Por ejemplo uno para /user y otro para /share

HDFS Federation

- ▶ Cada NameNode gestiona un espacio de nombres que contiene todos los bloques de un espacio de nombres
- ▶ Estos espacios de nombres son independientes para que no se comuniquen
- ▶ La caída de un NameNode no afecta al otro



111

Alta Disponibilidad

Conceptos

HDFS alta Disponibilidad

- ▶ Replicación de metadatos del namenode (local, NFS) + name node secundario para checkpoints= protección perdida de datos
- ▶ NO OFRECE ALTA DISPONIBILIDAD
- ▶ NameNode sigue siendo clave (SPOF Single point of Failure)
- ▶ Si falla cae escritura, lectura, listado de directorios
- ▶ Si falla mapeo Archivo-Bloque todos los clientes (incluido MapReduce) falla
- ▶ Hadoop sistema KO

HDFS Alta Disponibilidad

- ▶ Hasta que no se reconstruya el namenode Hadoop está fuera de juego
- ▶ Es un caso poco habitual
- ▶ Tiene un coste de tiempo muy alto
- ▶ Importante tenerlo en cuenta para preparar un plan de recuperación para estos casos.

HDFS Alta Disponibilidad

- ▶ A partir de Hadoop 2.x
- ▶ 2 Namenodes en configuración activo/reserva
- ▶ Fallo del activo → El que está en reserva sigue con las tareas
- ▶ El paso es automático
- ▶ Son necesarios cambios architectura

HDFS Alta disponibilidad

Cambios en la arquitectura

- ▶ Namenode Edit Log almacenar en almacenamientos compartidos.
 - ▶ El nodo de reserva lee el final del log para sincronizarse con el nodo activo y continua leyendo las nuevas entradas
- ▶ DataNodes → comunicar a los dos namenodes.
 - ▶ Tienen que enviar los informes de bloques ya que estos se almacenan en memoria y no en disco

HDFS Alta Disponibilidad

Cambios (continuación)

- ▶ Las aplicaciones cliente implementen mecanismos para el fallo de namenode transparentes para el usuario
- ▶ El rol del namenode secundario es asumido por el namenode de reserva
 - ▶ Realiza puntos del control del espacio de nombres gestionado por el de principal

HDFS Alta Disponibilidad

- ▶ Hay dos opciones para el almacenamiento compartido en alta disponibilidad
- ▶ NFS (Network File Sistem)
- ▶ QJM (Quorum Journal Manager) → Implementación recomendada de HDFS para alta disponibilidad

HDFS Alta disponibilidad

- ▶ QJP es una implementación de HDFS diseñada para ofrecer alta disponibilidad para el Edit Log.
- ▶ Se ejecuta como un conjunto de archivos de diario y cada edición se tienen que escribir en la mayoría de los nodos
- ▶ Normalmente hay tres nodos, por lo que puede fallar uno y todavía tener alta disponibilidad
- ▶ Es similar a la forma de trabajar de ZooKeeper

HDFS Alta Disponibilidad

- ▶ Si el namenode falla el de reserva puede activarse muy rápido
 - ▶ Dispone del último estado de la memoria (acceso a las entradas del Edit Log y el mapa de bloques actualizado)
- ▶ Aunque está disponible en decimas de segundo en la realidad puede tardar unos minutos → El sistema tiene que decidir si el namenode principal ha caído o no
- ▶ Si el namenode de reserva no se inicia el administrador puede iniciararlo desde un arranque en frío ya que Hadoop está configurado para ello

HDFS Alta Disponibilidad - balanceo

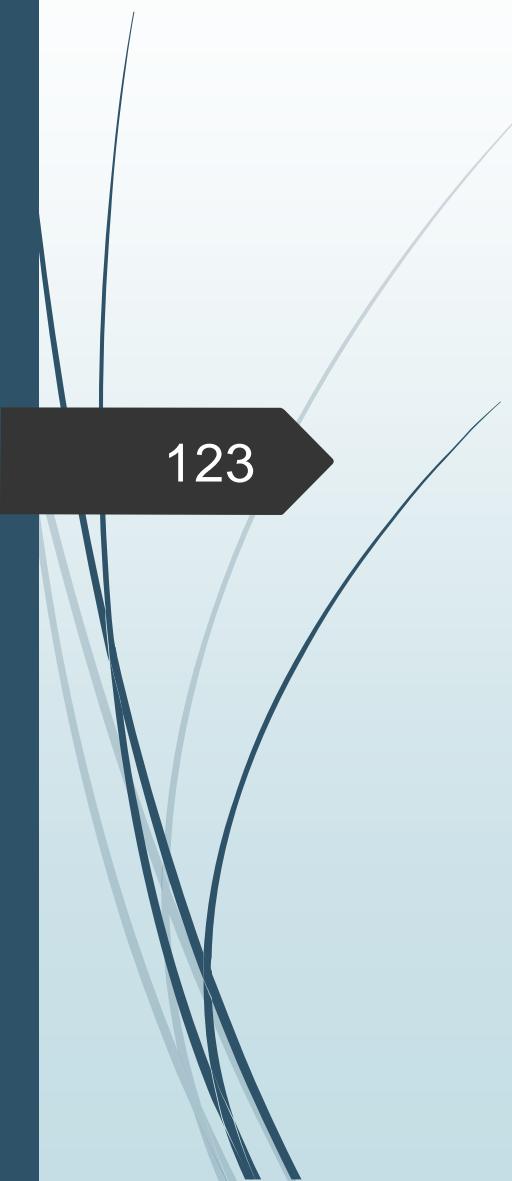
- ▶ La transición entre un namenode a otro lo gestiona el Failover Controller
- ▶ La implementación por defecto utiliza ZooKeeper
- ▶ Se asegura que sólo un namenode está activo
- ▶ Cada namenode ejecuta un proceso mínimo del control de comutación
 - ▶ Un pulso que envía genera un failover si no responde

Alta Disponibilidad - Balanceo

- ▶ La commutación se puede iniciar manualmente → mantenimiento
- ▶ En estos casos el namenode activo puede pensar que sigue estando activo ya que puede haber un retraso en las comunicaciones
- ▶ La implementación en Alta Disponibilidad de Hadoop se asegura que esto no pueda suceder con el método llamado Fencing
 - ▶ QJM sólo permite a un namenode escribir en el Edit Log, pero todavía puede leer el nodo activo anterior

Alta Disponibilidad - Fencing

- ▶ QJM recomendable escribir comando SSH para matar el proceso del namenode
- ▶ NFS, es necesario establecer un método más estricto ya que no se puede limitar a que escriba sólo un namenode
- ▶ Estos métodos pueden ser
 - ▶ Limitar el acceso del namenode al sistema compartido
 - ▶ Desactivar el puerto de red vía comando remoto
 - ▶ STONISH (Shoot the other node in the head) → apagar la máquina



123

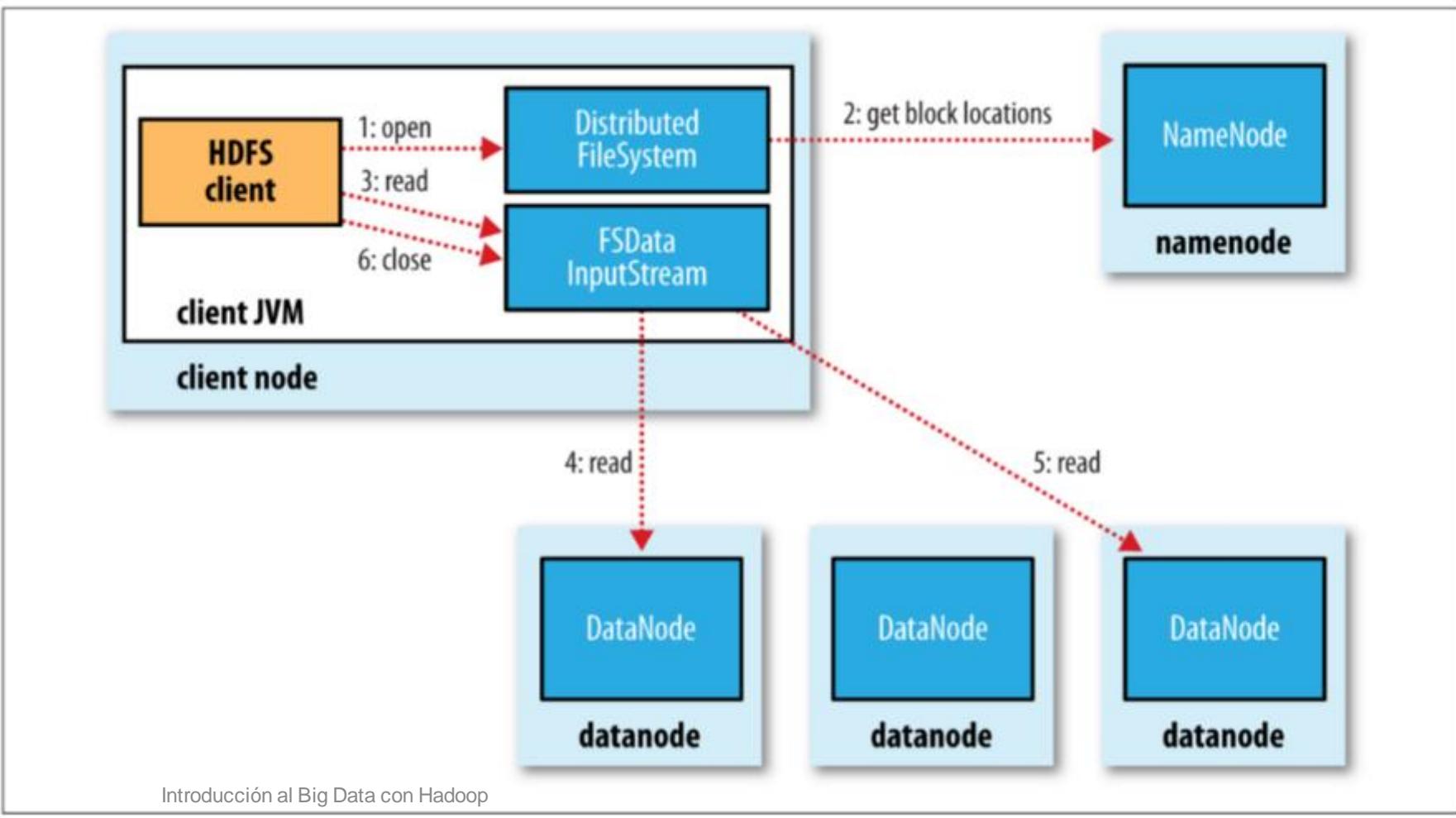
HDFS – JAVA Interface

Flujo de datos y ejemplos de lectura- escritura en java Interface

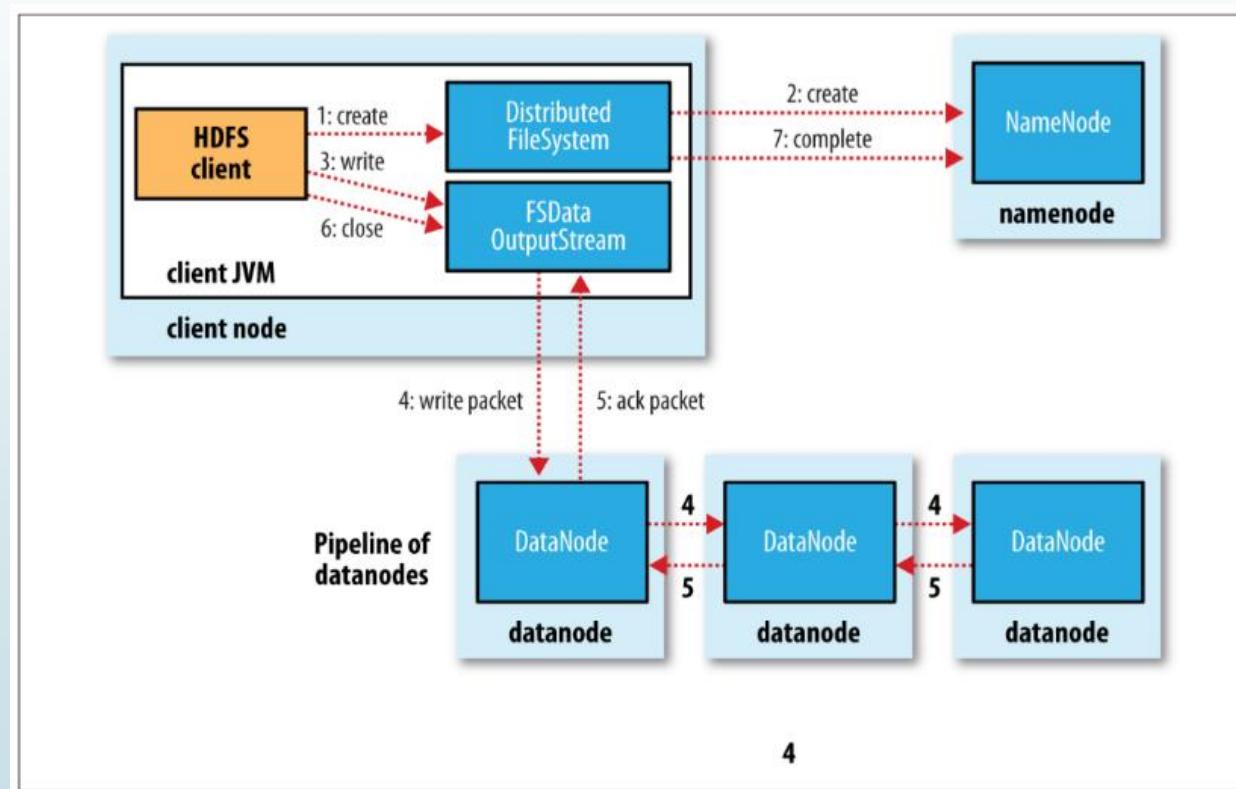
Java interface

- ▶ Hadoop tiene una visión abstracta del sistema de archivos
- ▶ HDFS es una implementación
- ▶ Interface está definida en `org.apache.hadoop.fs.FileSystem`
- ▶ Al estar escrito en JAVA el formato más común es utilizar la clase abstracta `FileSystem`
- ▶ `DistributedFileSystem` es la implementación de HDFS

HDFS Flujo de datos - Lectura



Flujo de datos - Escritura



Java Interface – leer datos

- Leer datos desde Hadoop URL con java.net.URL

```
InputStream in = null;
try {
    in = new URL("hdfs://host/path").openStream();
    // process in
} finally {
    IOUtils.closeStream(in);
}
```

Java Interface – Leer datos

- Mostrar información de un archivo utilizando URLStreamHandler

```
public class URLCat {  
  
    static {  
        URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());  
    }  
  
    public static void main(String[] args) throws Exception {  
        InputStream in = null;  
        try {  
            in = new URL(args[0]).openStream();  
            IOUtils.copyBytes(in, System.out, 4096, false);  
        } finally {  
            IOUtils.closeStream(in);  
        }  
    }  
}
```

Introducción al Big Data con Hadoop

JAVA interface Leer datos

- ▶ Si no es posible utilizar URLStreamHandlerFactory se puede utilizar API FileSystem
- ▶ El objeto configuración contiene la información del cliente o servidor almacenada en archivos como *etc/hadoop/core-site.xml*

```
public static FileSystem get(Configuration conf) throws IOException  
public static FileSystem get(URI uri, Configuration conf) throws IOException  
public static FileSystem get(URI uri, Configuration conf, String user)  
throws IOException
```

Java InterFace – Leer datos

- ▶ Una vez tenemos la instancia de FileSystem, invocamos el método Open
- ▶ Si no se especifica el tamaño del buffer se asigna por defecto 4 KB

```
public FSDataInputStream open(Path f) throws IOException  
public abstract FSDataInputStream open(Path f, int bufferSize) throws IOException
```

Java InterFACE Leer datos

```
% hadoop FileSystemCat hdfs://localhost/user/tom/text.txt
```

```
public class FileSystemCat {  
  
    public static void main(String[] args) throws Exception {  
        String uri = args[0];  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(URI.create(uri), conf);  
        InputStream in = null;  
        try {  
            in = fs.open(new Path(uri));  
            IOUtils.copyBytes(in, System.out, 4096, false);  
        } finally {  
            IOUtils.closeStream(in);  
        }  
    }  
}
```

JAVA InterFACE Leer datos

- El método Open de FileSystem Devuelve un FsDataInputStream - permite acceso aleatorio

```
public class FileSystemDoubleCat {  
  
    public static void main(String[] args) throws Exception {  
        String uri = args[0];  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(URI.create(uri), conf);  
        FSDatagramInputStream in = null;  
        try {  
            in = fs.open(new Path(uri));  
            IOUtils.copyBytes(in, System.out, 4096, false);  
            in.seek(0); // go back to the start of the file  
            IOUtils.copyBytes(in, System.out, 4096, false);  
        } finally {  
            IOUtils.closeStream(in);  
        }  
    }  
}
```

JavA Interface - Escribir

- ▶ FileSystem tiene varios métodos
- ▶ Path – crea el archivo y devuelve un stream para escribir
- ▶ Dispone de sobrecarga – Progressable para indicar la evolución de la escritura
- ▶ Se puede añadir información con el método Append

Java Interface - Escritura

- ▶ Copiar archivo a Hadoop con indicador de progresión

```
public class FileCopyWithProgress {  
    public static void main(String[] args) throws Exception {  
        String localSrc = args[0];  
        String dst = args[1];  
  
        InputStream in = new BufferedInputStream(new FileInputStream(localSrc));  
  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(URI.create(dst), conf);  
        OutputStream out = fs.create(new Path(dst), new Progressable() {  
            public void progress() {  
                System.out.print(".");  
            }  
        });  
  
        IOUtils.copyBytes(in, out, 4096, true);  
    }  
}
```

Java interface - directorios

- ▶ Podemos crear un directorio con el método mkdirs() de java.io.File
- ▶ Normalmente no se utiliza ya que al utilizar create() se construye la estructura de directorios

```
public boolean mkdirs(Path f) throws IOException
```

JAvA Interface - Metadata

- ▶ La clase `FileStatus` encapsula todo los metadatos de archivos y directorios
 - ▶ Longitud, tamaño del bloque, replicación, hora de modificación, propietario e información de permisos
- ▶ El método `GetFileStatus()` de `FileSystem` obtiene esta información

Java Interface - Metadatos

- Ejemplo UnitTesting
FileStatus()

```
public void fileStatusForFile() throws IOException {
    Path file = new Path("/dir/file");
    FileStatus stat = fs.getFileStatus(file);
    assertThat(stat.getPath().toUri().getPath(), is("/dir/file"));
    assertThat(stat.isDirectory(), is(false));
    assertThat(stat.getLen(), is(7L));
    assertThat(stat.getModificationTime(),
        is(lessThanOrEqualTo(System.currentTimeMillis())));
    assertThat(stat.getReplication(), is((short) 1));
    assertThat(stat.getBlockSize(), is(128 * 1024 * 1024L));
    assertThat(stat.getOwner(), is(System.getProperty("user.name")));
    assertThat(stat.getGroup(), is("supergroup"));
    assertThat(stat.getPermission().toString(), is("rw-r--r--"));
}
```

Java Interface - Directorios

- ▶ Método ListStatus de FileSystem devuelve lista de objetos FileStatus
 - ▶ Si es un archivo devuelve un objeto
 - ▶ Si es un directorio devuelve en un vector con todos los archivos y subdirectorios
- ▶ Dispone de sobrecarga incluyendo PathFilter para seleccionar los archivos

JAVA Interface - Directorios

- Ejemplo de obtener una colección de rutas

```
public class ListStatus {  
  
    public static void main(String[] args) throws Exception {  
        String uri = args[0];  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(URI.create(uri), conf);  
        Path[] paths = new Path[args.length];  
        for (int i = 0; i < paths.length; i++) {  
            paths[i] = new Path(args[i]);  
        }  
  
        FileStatus[] status = fs.listStatus(paths);  
        Path[] listedPaths = FileUtil.stat2Paths(status);  
        for (Path p : listedPaths) {  
            System.out.println(p);  
        }  
    }  
}
```

JAVA Interface - Globbing

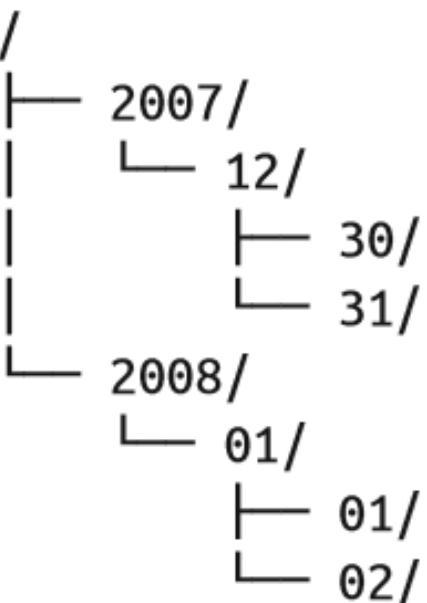
- ▶ Globbing es la utilización de caracteres especiales para identificar archivos o directorios en grupo en vez de uno en uno

```
public FileStatus[] globStatus(Path pathPattern) throws IOException  
public FileStatus[] globStatus(Path pathPattern, PathFilter filter)  
throws IOException
```

JAVA INTERFACE

Glob	Name	Matches
*	<i>asterisk</i>	Matches zero or more characters
?	<i>question mark</i>	Matches a single character
[ab]	<i>character class</i>	Matches a single character in the set {a, b}
[^ab]	<i>negated character class</i>	Matches a single character that is not in the set {a, b}
[a-b]	<i>character range</i>	Matches a single character in the (closed) range [a, b], where a is lexicographically less than or equal to b
[^a-b]	<i>negated character range</i>	Matches a single character that is not in the (closed) range [a, b], where a is lexicographically less than or equal to b
{a,b}	<i>alternation</i>	Matches either expression a or b
\c	<i>escaped character</i>	Matches character c when it is a metacharacter

JAVA INTERFACE



/*	/2007/2008
/*/*	/2007/12/2008/01
/*/12/*	/2007/12/30/2007/12/31
/200?	/2007/2008
/200[78]	/2007/2008
/200[7-8]	/2007/2008
/200[^01234569]	/2007/2008
/*/*/{31,01}	/2007/12/31/2008/01/01
/*/*/{3{0,1}}	/2007/12/30/2007/12/31
/*/{12/31,01/01}	/2007/12/31/2008/01/01



HIVE

Introducción

- ▶ Nace como una necesidad para aprender del gran volumen de datos de Facebook
- ▶ Utiliza características de SQL junto con Java para ejecutar consultas en grandes volúmenes de datos almacenados en HDFS
- ▶ Se considera una aplicación para procesar información genérica
- ▶ La habilidad para ejecutar consultas de sql contra los ficheros de datos lo que da a Hive su poder

Conceptos

- ▶ En un uso normal, Hive se ejecuta en la maquina de trabajo
- ▶ Convierte la consulta de SQL en trabajos para ejecutarlos en el cluster de Hadoop
- ▶ Hive almacena la información en tablas como medio de dotar de estructura a los datos almacenados en HDFS
- ▶ Los metadatos se almacenan en una base de datos llamada MetaStore

Introducción

- ▶ La configuración por defecto crea la base de datos en la máquina local por lo que no se puede compartir con otros usuarios.
- ▶ En entornos de producción, lo normal ese configura un metaestore remoto
- ▶ La instalación es sencilla, es necesario tener la misma versión de Hadoop instalada en la máquina de trabajo que en el cluster.
- ▶ La instalación de hadoop tiene que estar en la ruta indicada en la variable de entorno HADOOP_HOME

Introducción El Shell de Hive

- ▶ Hive dispone de un entorno de comandos o Shell.
- ▶ Utiliza el lenguaje HiveSQL que es un dialecto de SQL basado en MySQL
- ▶ Las sentencias tienen que acabar con ";" para que se ejecute
- ▶ Por ejemplo, el comando para mostrar las tablas al iniciar por primera vez Hive devuelve que no hay ninguna

```
hive> SHOW TABLES;  
OK  
Time taken: 0.473 seconds
```

Introducción El Shell de Hive

- ▶ Como los comandos de SQL no importa escribir en mayúsculas o minúsculas, salvo para la comparación de cadenas
- ▶ El tabulador completa comandos
- ▶ La primera vez que se ejecuta le cuesta un poco más para crear el MetaStore
- ▶ Almacena los archivos de la base de datos MetaStore en un directorio con el nombre metastore_db

Introducción El Shell de Hive

- ▶ Algunas opciones para los comandos
 - ▶ -f ejecuta los comando almacenados en el archivo indicado
 - ▶ -s no muestra los mensajes de información solo los resultados
 - ▶ -e permite ejecutar consultas cortas sin especificar el ; final

Introducción Instrucciones

► Create Table

- Crear tabla indicando nombre, columnas, tipos de datos
- Clausula Row Format, propia de HiveSql, permite identificar los delimitadores para los campos
- Las filas las interpreta por el carácter fin de línea
- Las tablas se crean como directorios dentro de la ruta de ejecución del Hive

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

Introducción Instrucciones

► Load Data

- Cargar información en hive
- Copia el archivo especificado en la carpeta de la tabla
- No intenta trasformar los datos para añadirlos a la tabla
- No realiza ninguna modificación en los archivos
- Parámetro Overwrite indica que se borren los archivos existentes, si no se especifica añade los archivos a la tabla

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'  
OVERWRITE INTO TABLE records;
```

Introducción Configuración

- ▶ Por defecto está configurado para trabajar en el archivo local
 - ▶ Propiedad fs.DefaultFS contienen [file:///](#)
- ▶ La ruta de trabajo se define en la propiedad hive.metastore.warehouse.dir
 - ▶ Por defecto es /user/hive/warehouse
- ▶ Siguiendo el ejemplo
 - ▶ Los archivos de la tabla record están en
 - ▶ % ls /user/hive/warehouse/records/ sample.txt

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'  
OVERWRITE INTO TABLE records;
```

Introducción Instrucciones

- ▶ Select, ejecuta una consulta de SQL
- ▶ Internamente transforma la consulta en un trabajo y muestra los resultados en la consola de salida

```
hive> SELECT year, MAX(temperature)
> FROM records
> WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
> GROUP BY year;
1949      111
1950      22
```

Configuración

- ▶ El archivo de configuración es Hive-site.xml que está en el directorio conf
- ▶ El archivo de documentación está en el mismo sitio con el nombre hive-default.xml
- ▶ Para ejecutar Hive con una configuración determinada utilizar – config indicando el directorio donde se encuentra el archivo hive.default.xml que queremos ejecutar

```
% hive --config /Users/tom/dev/hive-conf
```

configuración

- ▶ Para tener más de un usuario trabajando con Hive sobre el mismo cluster es necesario dar permisos de escritura al directorio de Hive a todos los usuarios
- ▶ Se | `% hadoop fs -mkdir /tmp` | configuración sólo para la ejecución de una consulta con SET, también sirve para consultar el valor

```
hive> SET hive.enforce.bucketing;  
hive.enforce.bucketing=true
```

[Configuration+Properties](#)

Proceso de Ejecución

- ▶ Originalmente Hive estaba creado para trabajar con MapReduce
- ▶ Actualmente puede trabajar también con Spark y TED que son más rápidos
 - ▶ Por ejemplo Spark permite a Hive especificar en el plan de ejecución que las salidas intermedias se almacenen en memoria
- ▶ La propiedad `hive.execution.engine` por defecto está definida a `mr` (MapReduce)

```
hive> SET hive.execution.engine=tez;
```

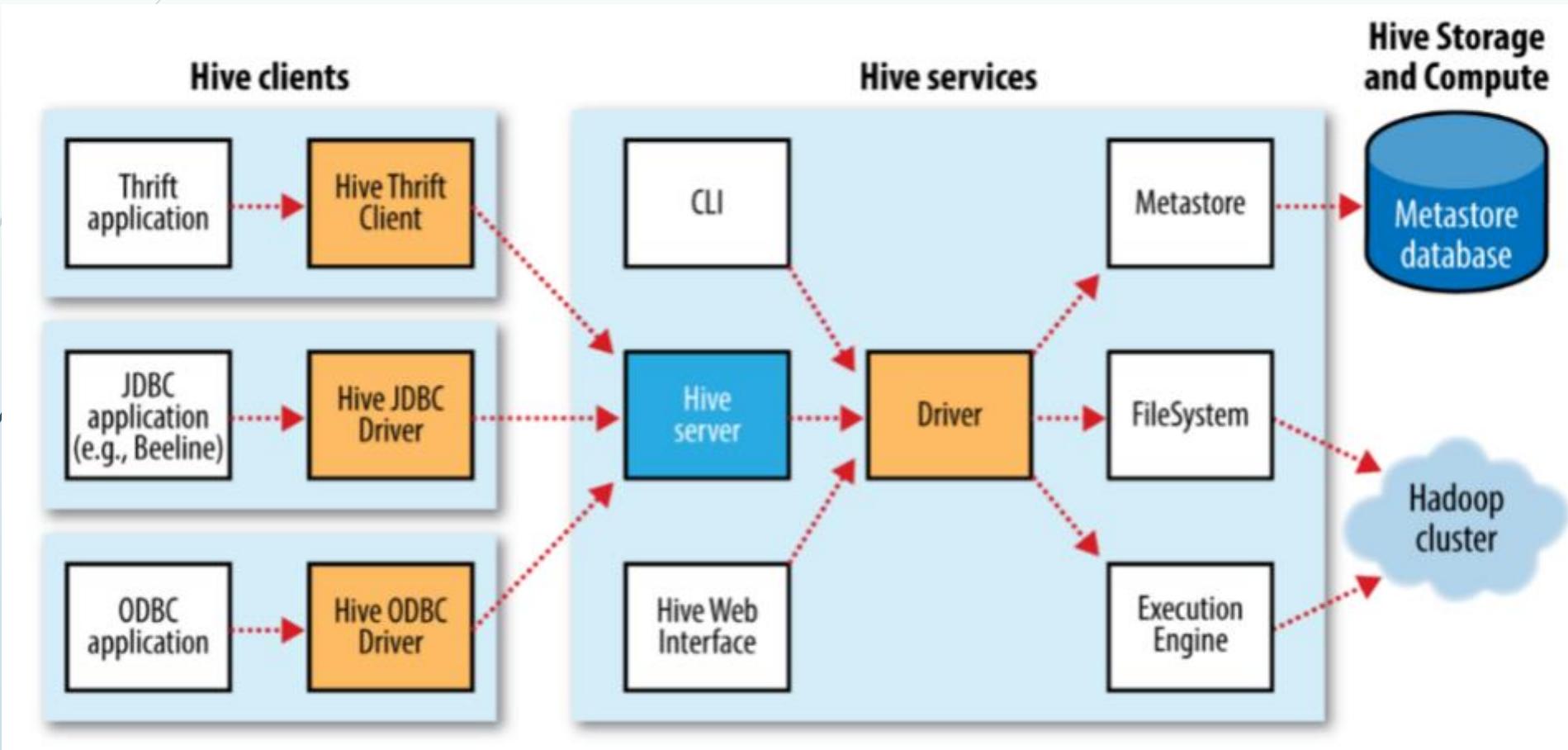
Arquitectura hive

- ▶ Dispone de varios servicios que puede ejecutar.
 - ▶ CLI: Shell de línea de comandos
 - ▶ HIVESERVER2, expone un servicio Thrift para permitir el acceso desde clientes en diferentes lenguajes. Propiedad `hive.server2.thrift.port` indica el puerto donde escucha
 - ▶ Hwi, es un interface web para versiones de Hive anteriores a 2.2 Una alternativa es WebHCat (rest API) para intergrar con otras aplicaciones
 - ▶ Jar, permite ejecutar aplicaciones java que incluyan clases de hadoop y hive

Arquitectura Hive

- ▶ Si se ejecuta Hive como servidor (hive –service hiveserver2) hay múltiples clientes que pueden consumir sus servicios
- ▶ Thrift: cualquier lenguaje de programación que soporte este cliente puede interactuar con el servidor (<https://thrift.apache.org/>)
- ▶ JDBC Driver: ejecuta un driver java en un proceso separado
- ▶ ODBC Driver: permite a aplicaciones de BI conectarse. No viene incluido

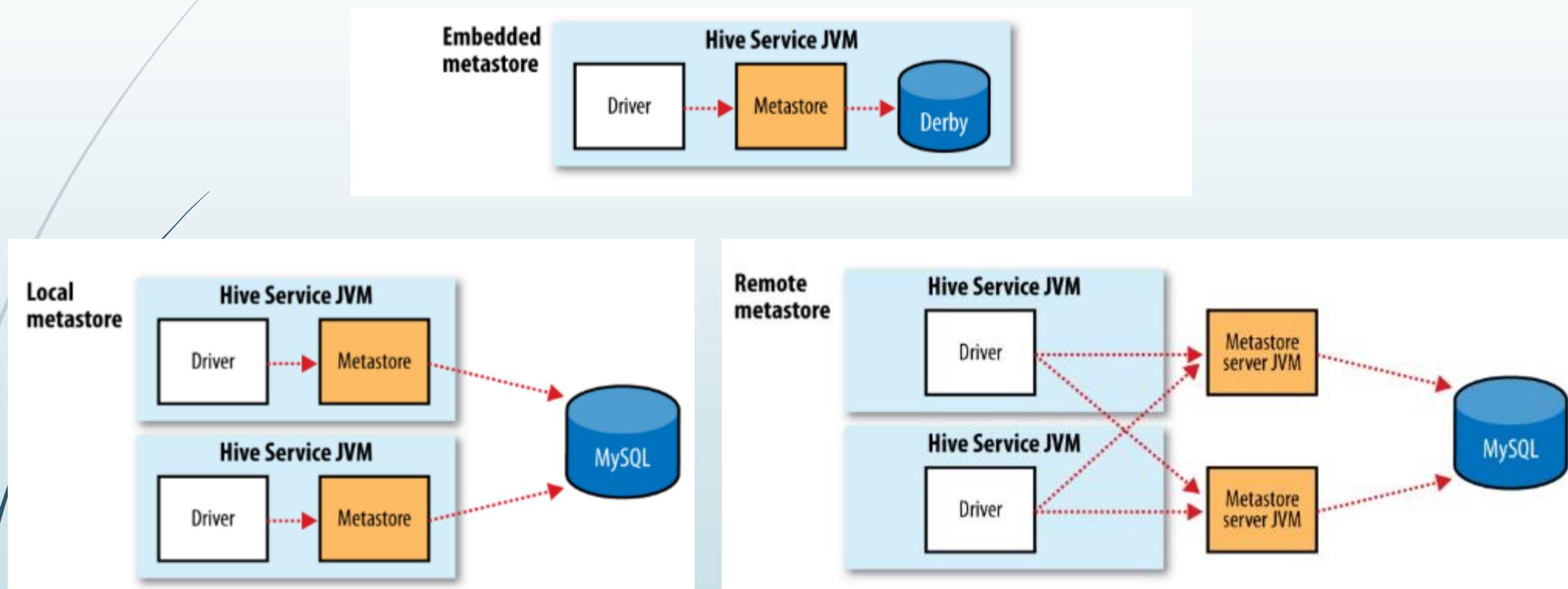
Arquitectura hive



metastore

- ▶ Es el almacén central de metadatos
- ▶ Ejecuta un servicio normalmente en la misma JVM que Hive que contiene una base de datos embebida (<https://db.apache.org/derby/>)
- ▶ También ejecuta un proceso de backup en disco para la base de datos
- ▶ Por defecto sólo una base de datos Derby puede acceder a los archivos de disco almacenados en la base de datos, por lo que no vale para entornos multiusuario
- ▶ La solución es utilizar una base de datos independiente en vez de la embebida. Sirve cualquier base de datos JDBC

metastore



metastore

- ▶ MySQL se suele utilizar como base de datos independiente para MetaStore
- ▶ Configurar propiedades
 - ▶ javax.jdo.option.ConnectionURL=jdbc:mysql://host/dbname?createDatabaseIfNotExist=true
 - ▶ javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
 - ▶ Usuario y contraseña
- ▶ La ruta del archivo para el driver JAR tiene que estar en la variable classpath

metastore

<code>hive.metastore.warehouse.dir</code>	URI	/user/hive/warehouse	The directory relative to <code>fs.defaultFS</code> where managed tables are stored.
<code>hive.metastore.uris</code>	Comma-separated URIs	Not set	If not set (the default), use an in-process metastore; otherwise, connect to one or more remote metastores, specified by a list of URLs. Clients connect in a round-robin fashion when there are multiple remote servers.
<code>javax.jdo.option.ConnectionURL</code>	URI	<code>jdbc:derby:;databaseName=metastore_db;create=true</code>	The JDBC URL of the metastore database.
<code>javax.jdo.option.ConnectionDriverName</code>	String	<code>org.apache.derby.jdbc.EmbeddedDriver</code>	The JDBC driver classname.
<code>javax.jdo.option.ConnectionUserName</code>	String	APP	The JDBC username.
<code>javax.jdo.option.ConnectionPassword</code>	String	mine	The JDBC password.

Comparación con bases de datos

- ▶ Aunque tiene muchas diferencias ya que está basado en HDFS y procesamiento con MapReduce, cada vez está evolucionando para tener un comportamiento más parecido a las bases de datos tradicionales
- ▶ Aspectos a comparar
 - ▶ Esquema en la lectura vs Esquema en la escritura
 - ▶ Actualizaciones, transacciones e índices
 - ▶ SQL on Hadoop alternativas

Comparación con bases de datos

- ▶ Esquema en la lectura vs Esquema en la escritura
 - ▶ En una base tradicional se carga el esquema en la carga de los datos, si los datos no encajan en el esquema entonces son rechazados. Este esquema se llama esquema en la escritura ya que se aplica antes de escribirlos en la base de datos. → Hace la carga más lenta
 - ▶ Hive comprueba el esquema cuando va a aplicar la consulta → Cargas de datos más rápidas

Comparación con bases de datos

- ▶ Updates, transacciones e Índices
 - ▶ Aunque son elementos clave en sistemas tradicionales en Hive al basarse en HDFS tiene una forma totalmente distinta de ejecutarse por lo que la comparación es difícil.
 - ▶ En el caso de las actualizaciones, se guardan en un archivo temporal y se ejecutan trabajos en background para mezclarlos con los bloques que correspondan
 - ▶ Transacciones no están soportadas en Hive

Comparación con bases de datos

- ▶ Comparando con Impala, la ejecución de consultas utiliza un servicio que está continuamente en ejecución, lo que agiliza el procesamiento
- ▶ Hive utiliza MapReduce, lo que hace el proceso más lento ya que tiene que ejecutar la aplicación maestra cada vez que se inicia, aunque el procesamiento con TEZ ha sido una mejora.

hiveSQL

- ▶ Es una mezcla de SQL-92, MySQL y Oracle SQL

- ▶ Comparando las características de HiveSQL con SQL

Feature	SQL	HiveQL
Updates	UPDATE, INSERT, DELETE	UPDATE, INSERT, DELETE
Transactions	Supported	Limited support
Indexes	Supported	Supported
Data types	Integral, floating-point, fixed-point, text and binary strings, temporal	Boolean, integral, floating-point, fixed-point, text and binary strings, temporal, array, map, struct
Functions	Hundreds of built-in functions	Hundreds of built-in functions

Hive SQL

- Continuando con la comparación de funcionalidad:

	Multitable inserts	Not supported	Supported
CREATE TABLE...AS SELECT		Not valid SQL-92, but found in some databases	Supported
SELECT	SQL-92		SQL-92. SORT BY for partial ordering, LIMIT to limit number of rows returned
Joins	SQL-92, or variants (join tables in the FROM clause, join condition in the WHERE clause)		Inner joins, outer joins, semi joins, map joins, cross joins
Subqueries	In any clause (correlated or noncorrelated)		In the FROM, WHERE, or HAVING clauses (uncorrelated subqueries not supported)
Views	Updatable (materialized or nonmaterialized)		Read-only (materialized views not supported)

Particiones y cubos

- ▶ Hive organiza las tablas en particiones para mejorar el rendimiento en la búsqueda de información
- ▶ Las particiones se pueden dividir en cubos, permitiendo búsquedas más complejas.
 - ▶ Impone estructura extra a las tablas
 - ▶ Permite especificar que columnas queremos empaquetar y en cuantos elementos

Formatos de almacenamiento

- ▶ Hive tiene en cuenta dos dimensiones, el formato de fila y el de fichero
- ▶ El formato de fila indica como se almacenan las filas y los campos dentro de las filas
- ▶ Es importante para el proceso de SerDe (serialización deserialización) a la hora de realizar las consultas
- ▶ El formato de fichero define el formato del contenedor

Formatos de almacenamiento

- ▶ Texto Delimitado, es el formato por defecto
 - ▶ El delimitador de fila es el carácter CTRL A
 - ▶ El delimitador de campo es el carácter CTRL B
- ▶ Formatos de almacenamientos binarios: Sequence files, Avro datafiles, Parquet files, RCFiles, and ORCFiles
- ▶ Formatos personalizados SerDe

Joins

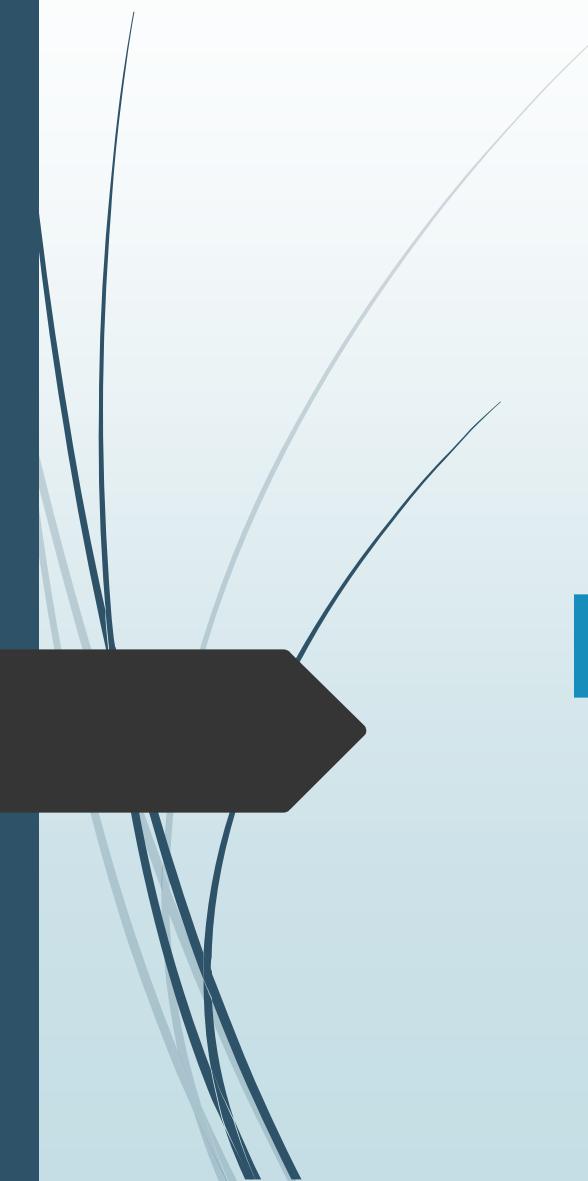
- ▶ Hive permite realizar joins con sintaxis muy parecida a la de SQL tradicional
- ▶ Los Join se transforman en trabajos de MapReduce que procesan la unión
- ▶ Podemos utilizar la clave Explain, para que MapReduce devuelva el plan de ejecución

EXPLAIN

```
SELECT sales.* , things.*  
FROM sales JOIN things ON (sales.id = things.id);
```

Hive

- ▶ La funcionalidad de HIVE permite trabajar con sentencias similares a SQL facilitando el análisis de la información
- ▶ Quedaría pendiente
 - ▶ Queries
 - ▶ Funciones definidas por el usuario
 - ▶ Importación de datos
 - ▶ Optimización de scripts de MapReduce



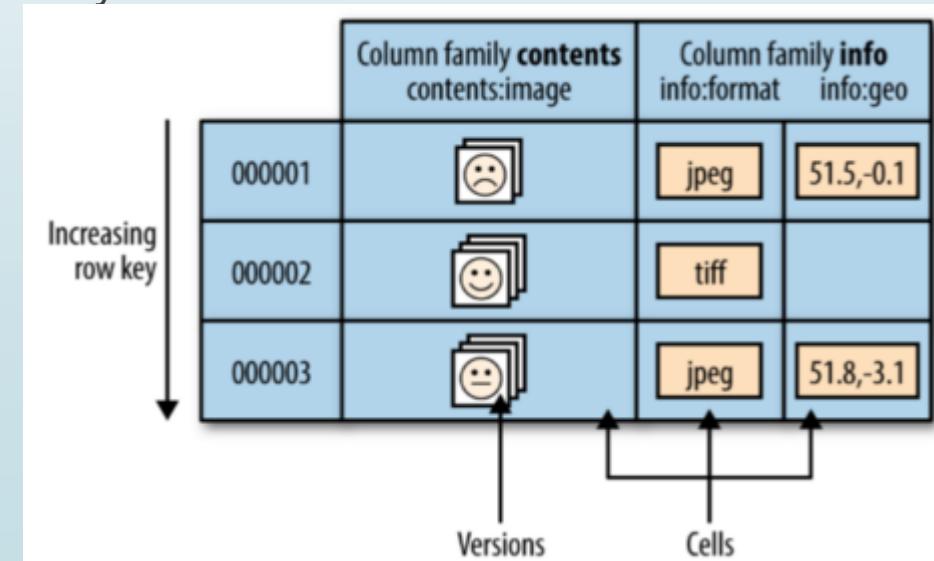
Hbase

Introducción

- ▶ Hbase es una base de datos distribuida organizada en columnas y basada en HDFS
- ▶ Permite acceso aleatorio a datos en grandes conjuntos de datos
- ▶ La aproximación de HBASE a los problemas de escalado se basa en ir añadiendo nodos cuando se necesite
- ▶ No es una base de datos relacional y no soporta SQL
- ▶ Parte del proyecto BigTable de Google de 2007. Empezó como un elemento de Hadoop hasta convertirse en un proyecto independiente

Conceptos

- ▶ Las aplicaciones almacenan información en tablas etiquetadas
- ▶ Las tablas están hechas por filas y columnas
- ▶ Las celdas tienen versiones. Una marca de tiempo automática cuando se inserta información
- ▶ Las celdas contienen un array de bytes



Conceptos

- ▶ La key de la fila es también un array de Bytes
- ▶ Como las claves son un array de bytes, todo tipo de datos puede servir como campo clave
- ▶ Las filas se ordenan por el campo clave
- ▶ Todas las tablas se acceden a través de su clave primaria

Conceptos

- ▶ Las columnas se agrupan en familias de columnas
- ▶ Todas las columnas de una familia tienen un prefijo, separado por ":"
 - ▶ Info:geo, info:format son de una familia y contents:image de otra
- ▶ Físicamente, las familias de columnas se almacenan juntas.
- ▶ Es más correcto definir HBASE como un sistema de bases de datos orientado a familias de columnas que ha columnas
 - ▶ Las modificaciones de la configuración se hacen a nivel de familia no ha nivel de columna
- ▶ Si la familia de columna existe, se pueden añadir nuevas columnas en cualquier momento

Conceptos

- ▶ Las tablas se dividen automáticamente en regiones, cada región tiene un subconjunto de filas.
- ▶ La región se identifica por la primera y la última fila
- ▶ Inicialmente la tabla tiene una región, pero cuando crece por encima del límite que esté configurada se divide en dos regiones de tamaño similar.
- ▶ Las regiones son la unidad que se distribuye en un cluster HBASE. El conjunto ordenado de todas las regiones constituyen la tabla
- ▶ El bloqueo es a nivel de fila, no a nivel de tabla

Estructura

- ▶ Tiene una filosofía similar a HDFS y YARN (NameNode y DataNode, Administrador de Recursos y Administrador de Nodos)
- ▶ Hbase está formado por un nodo Hbase Master gestionando el cluster de uno o más Servidores de Regiones

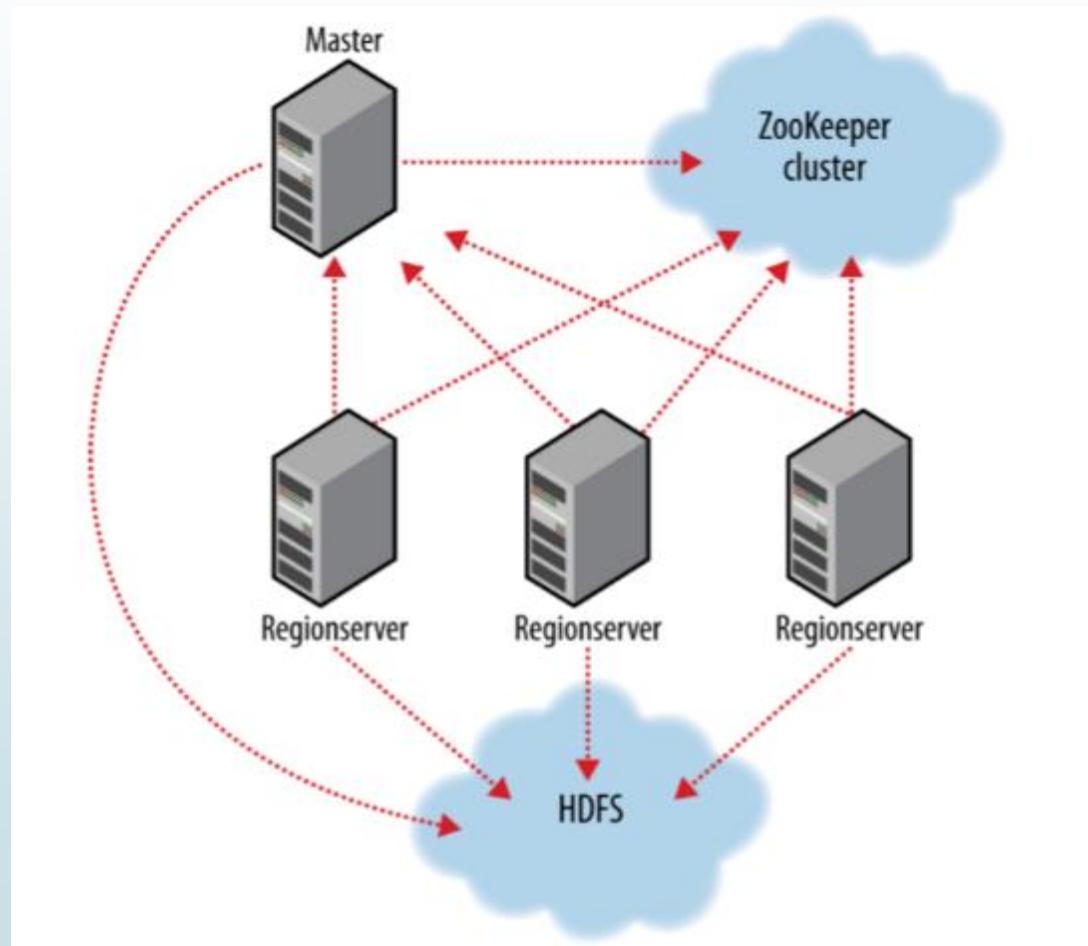
Estructura

- ▶ HBase Master es responsable de
 - ▶ la división de la base de datos inicial
 - ▶ Asignar regiones a servidores de regiones
 - ▶ Recuperar servidores de regiones que puedan fallar
- ▶ Los Servidores de regiones
 - ▶ Pueden tener 0 o más regiones
 - ▶ Contienen las peticiones de los clientes de lectura y escritura
 - ▶ Informa al HBase Master de las regiones hija

Estructura

- ▶ HBase depende de Zookeeper para gestionar la ubicación de los elementos
- ▶ Zookeeper mantiene la información de la ubicación del catálogo de tablas así como del HBAse Master.
- ▶ La asignación de las regiones se realiza mediante Zookeeper, de tal manera que si hay algún problema permite al sistema seguir los pasos y continuar dónde se quedó
- ▶ A las conexiones clientes de HBASE se les pasa una instancia de Zookeeper para que pueda seguir por la jerarquía y encontrar los nodos con los atributos

Estructura



Estructura

- ▶ HBase utiliza el sistema de archivos de Hadoop para almacenar información.
- ▶ Por defecto almacena la información en el sistema de archivos local
- ▶ En entornos de producción hay que modificar la configuración para que trabaje con HDFS
- ▶ Internamente, HBase tiene una tabla especial de catálogo `hbase:meta`
 - ▶ Contiene la lista, estado y ubicación de todas las regiones
 - ▶ Contiene una clave con el nombre de la tabla, región, filas, horas y hash MD5

Hbase shell

- ▶ Create: crea una tabla
 - ▶ Create ‘test’ ,’data’ → Crea la tabla test con una columnas data con la configuración por defecto
- ▶ List: Lista las tablas existentes
- ▶ Put: introducir datos en la tabla
 - ▶ Put ‘test’,’row1’,’data:1’, ‘value’ → En la tabla Test, crea una columna nueva Data:1 y le introduce en el la primera fila el valor “value”
- ▶ Get: obtiene la información de la columna y la celda

HBase Shell

- ▶ Para eliminar una tabla primero tenemos que desactivarla
- ▶ Disable para desactivarla
- ▶ Drop para borrarla
- ▶ Los comandos de Shell se pueden ejecutar desde Java

Hbase Java

- ▶ Las clases de HBase están en los namespaces `org.apache.hadoop.hbase` y `org.apache.hadoop.hbase.client`
- ▶ Clases básicas
 - ▶ `HBaseConfiguration` → permite crear un objeto configuración que lee la parametrización de `hbasesite.xml` y `hbase-default.xml`
 - ▶ `HBaseAdmin` → Permite gestionar el cluster de Hbase, añadir y quitar tablas
 - ▶ `HTable` → Permite acceder a una tabla específica
- ▶ `ConnectionFactory` → permite crear objetos `Connection` con los métodos `getAdmin` `GetTable`

Hbase Java

- ▶ Los objetos SCANNER son como los cursores en bases de datos tradicionales o Java iterators
 - ▶ Tienen que cerrarse después de su uso
 - ▶ Devuelven las filas en orden
 - ▶ Se generan desde un objeto Table.getScanner
 - ▶ Se pasa la fila a partir de la cual trabajar
 - ▶ Qué columnas tiene que devolver
 - ▶ Y filtro a ejecutar en el lado del servidor
 - ▶ Por defecto devuelve de 100 en 100 filas (configurable hbase.client.scanner.caching)

Hbase Java administracion

- ▶ Ejemplo de código

HBASE java Mapreduce

- ▶ Dispone de clases para utilizar HBase como origen o destino de trabajos de MapReduce en el namespace org.apache.hadoop.hbase.mapreduce
- ▶ TableInputFormat → Permite crear Splits de regiones para los trabajos Maps
- ▶ TableOutputFormat → escribe el output de reduce en HBase

HBASE java mapreduce

- ▶ Ejemplo de código

Hbase vs rdbms

- ▶ HBASE está pensado para grandes volúmenes de datos, millones de filas y millones de columnas, se divide en regiones que se distribuyen por miles de nodos automáticamente. El esquema de la tabla es un espejo del archivo físico.
- ▶ RDBMS, siguen un esquema fijo, orientado en tablas, pensando en la abstracción e integridad de los datos. Utiliza consultas estructuradas en SQL, permite fácilmente crear índices secundarios, joins, agrupaciones
- ▶ Para la mayoría de las empresas por la madurez de la tecnología RDBMS es más que suficiente.
- ▶ Si el problema es la escalabilidad del sistema, entonces HBase puede ser una opción

Hbase vs rdbms

- ▶ No tiene índices
 - ▶ Las filas se almacenan secuencialmente, no hay problemas con el mantenimiento de índices
 - ▶ Los tiempos de inserción son independientes del tamaño de la tabla
- ▶ Particionamiento automático
 - ▶ Conforme crecen las tablas, son automáticamente divididas y repartidas por los servidores de regiones
- ▶ Escalado lineal con nuevos nodos
 - ▶ Sólo hay que añadir un nuevo nodo, apuntarlo al cluster y el servidor de regiones se reajusta

Hbase vs RDBMS

- ▶ Hardware común
 - ▶ Equipos con coste menor que servidores
- ▶ Tolerancia a fallos
 - ▶ Al tener múltiples nodos, la caída de uno no afecta al rendimiento
- ▶ Proceso por lotes
 - ▶ La integración con MapReduce permite procesamiento paralelo de información



Mongodb

conceptos

- ▶ Documento es la unidad básica de datos, sería similar a una fila
- ▶ Colección, se podría asimilar a una tabla con un esquema dinámico
- ▶ Una sola instancia de MongoDB puede tener multiples bases de datos independientes, y cada una puede tener sus propias colecciones
- ▶ Cada documento tiene una clave especial “`_id`” que es única en una colección
- ▶ Dispone de un Shell de comandos con javascript para administración

conceptos

- ▶ No soporta transacciones, se puede simular pero no existe una estructura que obligue a obedecer las cláusulas semánticas
- ▶ Unir diferentes tipos de datos mediante varias dimensiones
- ▶ El número de aplicaciones diseñadas para trabajar con MongoDB está creciendo pero no son tantas como otras plataformas

Documentos

- ▶ Es un conjunto ordenado de conjuntos de parejas key/value.
- ▶ Cada lenguaje de programación lo representa de forma diferente, pero en la mayoría de los casos coincide con un diccionario.

```
{"greeting" : "Hello, world!"}
```

```
{"greeting" : "Hello, world!", "foo" : 3}
```

Documentos

- ▶ La clave puede contener cualquier UTF-8 carácter
- ▶ No se puede utilizar Null ya que lo utiliza para marcar el final del campo clave
- ▶ Los caracteres “\$” y “.” tienen propiedades especiales en algunas circustancias, por lo que se consideran también reservados
- ▶ MongoDB es case-sensitive {"foo":3} y {"Foo":3} son diferentes
- ▶ Las key en un documento no pueden estar repetidas

Documentos

- ▶ Las parejas Key/value en un documento están ordenadas
- ▶ Normalmente los lenguajes de programación no necesitan el orden de los documentos
- ▶ $\{x:1;y:2\}$ es distinto a $\{y:2; x:1\}$

Colecciones – Esquemas dinámicos

- ▶ Los documentos de una misma colección pueden tener diferentes formas
 - ▶ Pueden contener diferentes tipos para los valores y diferentes keys

```
{"greeting" : "Hello, world!"}  
{"foo" : 5}
```

Colecciones – esquemas dinámicos

- ▶ Si podemos tener diferentes tipos de documentos, ¿porqué necesitamos diferentes colecciones?
 - ▶ Mantener documentos de diferentes tipos puede ser una locura para garantizarnos que las consultas devuelven documentos de determinado tipo
 - ▶ Es más rápido obtener una lista de colecciones de diferentes tipos que una lista de tipos dentro de una colección para hacer la consulta al tipo correcto
 - ▶ Agrupar documentos del mismo tipo en una colección, permite una mejor ubicación de la información
 - ▶ Los índices son por colección, si tenemos documentos de mismo tipo, serán más eficientes

Colecciones - Nombres

- Al igual que los documentos, las colecciones pueden contener los caracteres UTF-8 con excepciones
 - La cadena vacía “” no se puede utilizar como nombre
 - Al igual que los documentos no puede contener el null (\0)
 - No se deben crear colecciones que empiecen con system. Es un prefijo reservado para colecciones internas
 - No se debe usar el “\$”, algunas colecciones de sistema lo utilizan pero no colecciones de usuario

Colecciones - Subcolecciones

- ▶ Una forma de organizar colecciones es utilizar un espacio de nombres y el grupo separadas por “.”
 - ▶ Blog.posts y Blog.autores pueden ser dos subcolecciones de documentos
- ▶ Es únicamente para organizar la información, no hay ninguna relación entre las colecciones ni con la colección “padre” (Blog en el ejemplo, que no tiene por qué existir)
- ▶ Herramientas que trabajan con Mongo facilitan el uso de las subcolecciones, por ejemplo GridFS

Bases de datos

- ▶ Las bases de datos son agrupaciones de colecciones.
- ▶ Una instancia de MongoDB puede tener varias bases de datos y cada una puede tener 0 o más colecciones
- ▶ Cada base de datos tiene diferentes permisos y se almacena en diferentes sistemas de archivos en el disco
- ▶ Es recomendable almacenar los datos de la misma aplicación en una base de datos

Bases de datos

- ▶ Las bases de datos se identifican por nombre
- ▶ Los nombres siguen los mismos criterios que en los casos anteriores UTF-8 con excepciones
 - ▶ “” no se puede utilizar
 - ▶ No puede contener /, \, ., ", *, <, >, :, |, ?, \$, un espacio en blando single space), o \0 (character null).
 - ▶ Los nombres son case-sensitive, incluso en sistemas de archivos que no lo sean
 - ▶ El máximo son 64 caracteres
- ▶ Cada base de datos acaba siendo un archivo, por eso alguna de las restricciones

Bases de datos

- ▶ Hay bases de datos especiales propias del sistema
- ▶ admin: en términos de autenticación es la base de datos “root”. Al agregar un usuario a esta base de datos, automáticamente hereda permisos para todas las bases de datos
- ▶ local: esta base de datos no se replica, puede contener colecciones locales a un solo servidor
- ▶ config: cuando se utiliza la base de datos en una configuración dividida, esta base de datos almacena la configuración

Bases de datos

- ▶ Los namespaces se consideran como el nombre totalmente cualificado de una colección
- ▶ Se obtienen concatenando el nombre de la base de datos con el nombre de la colección
- ▶ Tiene limitación de longitud, en teoría a 128 bytes
- ▶ Por ejemplo la colección “blog.posts” dentro de la base de datos “cms”, su namespace sería “cms.blog. posts”

ejecución

- ▶ Normalmente se ejecuta como servidor, de tal manera que los clientes pueden conectarse y realizar operaciones.
- ▶ Por defecto utiliza el puerto 27017, si no está disponible no se puede ejecutar
- ▶ Ejecuta también un servidor HTTP en un puerto 1000 direcciones superior al que esté utilizando, normalmente 28017 → <http://localhost:28017>
- ▶ Se puede acceder a información administrativa

Shell

- ▶ Desde el Shell de comandos se pueden realizar labores administrativas, inspeccionar una instancia o hacer pruebas sencillas
- ▶ Es un interprete de javascript, por lo que podemos ejecutar scripts

```
> Math.sin(Math.PI / 2);
1
> new Date("2010/1/1");
"Fri Jan 01 2010 00:00:00 GMT-0500 (EST)"
> "Hello, World!".replace("World", "MongoDB");
Hello, MongoDB!
```

Shell - cliente

- ▶ Realmente el Shell es un cliente de MongoDB
- ▶ Al iniciarse se conecta a la base de datos “test” y la asigna a la variable global “db”
- ▶ Si escribimos db en el Shell devuelve la base de datos activa en el Shell
- ▶ Para cambiar de base de datos, utilizar el comando “use” <nombre bd>
- ▶ Si escribimos “db”.<colección> devuelve la lista de colecciones de la base de datos.

Shell – comandos básicos

- ▶ Podemos ejecutar los comandos CRUD (Create, Read, Update y Delete)
- ▶ Método Insert() añade un documento a una colección de la base de datos
- ▶ Método find() muestra el contenido de una colección

```
> post = {"title" : "My Blog Post",
... "content" : "Here's my blog post.",
... "date" : new Date()}
{
    "title" : "My Blog Post",
    "content" : "Here's my blog post.",
    "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

```
> db.blog.insert(post)
```

Shell - comandos básicos

- ▶ Leer información con find() o findOne()
- ▶ Permite hacer filtros para seleccionar los documentos
- ▶ Por defecto devuelve hasta 20 documentos que cumplan el criterio

```
> db.blog.find()
{
  "_id" : ObjectId("5037ee4a1084eb3ffef7228"),
  "title" : "My Blog Post",
  "content" : "Here's my blog post.",
  "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

Shell – comandos básicos

- ▶ Update permite actualizar un documentos
- ▶ Necesita al menos dos parámetros
 - ▶ El criterio para encontrar el documento
 - ▶ El nuevo documento

Shell – comandos básicos

- ▶ Ejemplo añadir una nueva key al post con un vector de comentarios

```
> post.comments = []  
  
> db.blog.update({title : "My Blog Post"}, post)  
  
> db.blog.find()  
{  
    "_id" : ObjectId("5037ee4a1084eb3ffef7228"),  
    "title" : "My Blog Post",  
    "content" : "Here's my blog post.",  
    "date" : ISODate("2012-08-24T21:12:09.982Z"),  
    "comments" : [ ]  
}
```

Shell – comandos básicos

- ▶ Para borrar permanentemente un documento de la base de datos, utilizar el comando Delete
- ▶ Si se ejecuta sin parámetros, borra todos los documentos de la colección
- ▶ Si pasamos como parámetro una key, borra el documento

```
> db.blog.remove({title : "My Blog Post"})
```

Tipos de datos

- ▶ Los documentos se pueden considerar conceptualmente similares a los archivos JSON
- ▶ Los tipos de datos básicos que utiliza son los mismos null, boolean, numeric, string, array, y object, ampliados con algunos como Date, Regularexpression, documentos embebidos o NumberInt

Mongodb – Por ver

- ▶ Trabajar con funciones CRUD
- ▶ Definir queries y cursores
- ▶ Crear índices para las aplicaciones, simples, avanzados, geoespaciales
- ▶ Analizar aplicaciones, comandos de agregación, soporte para MapReduce
- ▶ Replicación, sincronización, copias de seguridad
- ▶ Administración y monitorización
- ▶ Sharding – división de la base de datos

Spark

<http://spark.apache.org/>

conceptos

- ▶ Es un framework de trabajo para procesamiento de largos volúmenes de datos
- ▶ Spark tiene su propio motor de ejecución por lo que no utiliza MapReduce para ejecutar los trabajos aunque tiene una filosofía parecida
- ▶ Tiene un alto nivel de integración con Hadoop, puede utilizar Yarn y almacenar los datos en HDFS

conceptos

- ▶ La principal característica es almacenar grandes cantidades de datos en memoria entre trabajos
- ▶ Esta capacidad permite mejorar el rendimiento de MapReduce, donde los conjuntos de datos son siempre almacenados en disco
- ▶ Spark dispone de algoritmos iterativos, de tal manera que una función es aplicada a un dataset hasta que se cumpla una condición
- ▶ Tiene también análisis interactivo, lo que permite a un usuario hacer consultas sobre un dataset

conceptos

- ▶ Dispone de Apis en tres idiomas, SCALA, Java y Python
- ▶ Dispone de múltiples módulos para completar las herramientas
 - ▶ SQL DataFrames
 - ▶ Spark Streaming
 - ▶ Mlib (Machine Learning)
 - ▶ GraphX

conceptos

- ▶ Spark utiliza como MapReduce trabajos, pero con una visión más general
- ▶ Utiliza un motor DAG (directed acyclic graph) de etapas, que permite ejecutar listas de operaciones y traducirlas en trabajos. podría asimilarse al concepto de las tareas Map y Reduce
- ▶ RDD (Resilient Distributed DataSet) es una colección de objetos que está dividido por múltiples máquinas en un cluster
- ▶ Normalmente uno o varios RDD se crean como input y mediante transformaciones se convierten en el destino

Conceptos

- ▶ Las tareas se dividen en tareas por el runtime de Spark y se pueden ejecutar en paralelo en porciones de RDD a lo largo del cluster
- ▶ Los trabajos siempre se ejecutan en el contexto de una aplicación (una instancia de SparkContext)
- ▶ Permite agrupar variables y RDDs
- ▶ Una aplicación puede ejecutar más de un trabajo en serie o en paralelo y ofrece los mecanismos para que un trabajo pueda acceder a un RDD que haya sido cacheado por otro trabajo previo en la misma aplicación.
- ▶ Una sesión del Shell es una aplicación

Spark-Shell - Ejemplo

- ▶ Es un editor de comandos en Scala
- ▶ Al iniciar el editor genera una variable de contexto “sc” que es el punto de entrada a Spark

```
scala> val lines = sc.textFile("input/ncdc/micro-tab/sample.txt")
lines: org.apache.spark.rdd.RDD[String] = MappedRDD[1] at textFile at
<console>:12
```

Spark-Shell - Ejemplo

- ▶ Una vez cargado un RDD podemos realizar transformaciones. La primera es dividir las líneas en campos
- ▶ Utiliza el método map para separar por el campo que queremos
- ▶ Convierte de un string a un array de string de Scala

```
scala> val records = lines.map(_.split("\t"))
records: org.apache.spark.rdd.RDD[Array[String]] = MappedRDD[2] at map at
<console>:14
```

Spark-Shell Ejemplo

- ▶ La siguiente transformación es quitar los elementos que no nos interesen aplicando un filtro
- ▶ El método filter() toma un predicado que devuelve un booleano, en este caso que la temperatura no sea válida

```
scala> val filtered = records.filter(rec => (rec(1) != "9999"  
    && rec(2).matches("[01459]")))  
filtered: org.apache.spark.rdd.RDD[Array[String]] = FilteredRDD[3] at filter at  
<console>:16
```

Spark-Shell Ejemplo

- ▶ Para poder utilizar funciones de agrupación, necesitamos que el RDD esté en formato key/value, por lo que podemos hacer otra función map() con la transformación
- ▶ Map() puede utilizar expresiones Lambda

```
scala> val tuples = filtered.map(rec => (rec(0).toInt, rec(1).toInt))
tuples: org.apache.spark.rdd.RDD[(Int, Int)] = MappedRDD[4] at map at
<console>:18
```

Spark-Shell Ejemplo

- ▶ Para obtener la temperatura máxima, tenemos que agrupar la información, para ello transformamos el RDD utilizando el método ReduceByKey(). Ahora lo podemos utilizar ya que tenemos el RDD en formato key/value
- ▶ La temperatura máxima se obtiene con la función de java Math.max

```
scala> val maxTemps = tuples.reduceByKey((a, b) => Math.max(a, b))
maxTemps: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[7] at
reduceByKey at <console>:21
```

Spark-Shell Ejemplo

- ▶ La salida del resultado se puede hacer por pantalla, con `println()` utilizando la función `foreach()` para recorrer la variable con el resultado.
- ▶ Esta operación hace que Spark genere otro trabajo para que procese los valores y se puedan presentar a través de la función `println()`

```
scala> maxTemps.foreach(println(_))  
(1950,22)  
(1949,111)
```

Spark-Shell Ejemplo

- ▶ Otra alternativa es guardar el resultado en un archivo utilizando el método `SaveAsTextFile()`
- ▶ Este método crea un directorio con el archivo resultante
- ▶ El método `SaveAsTextFile()` genera un trabajo de Spark, que lo escribe.

```
scala> maxTemps.saveAsTextFile("output")
```

```
% cat output/part-*  
(1950,22)  
(1949,111)
```

Scala ejemplo código

```
import org.apache.spark.SparkContext._  
import org.apache.spark.{SparkConf, SparkContext}  
  
object MaxTemperature {  
    def main(args: Array[String]) {  
        val conf = new SparkConf().setAppName("Max Temperature")  
        val sc = new SparkContext(conf)  
  
        sc.textFile(args(0))  
            .map(_.split("\t"))  
            .filter(rec => (rec(1) != "9999" && rec(2).matches("[01459]")))  
            .map(rec => (rec(0).toInt, rec(1).toInt))  
            .reduceByKey((a, b) => Math.max(a, b))  
            .saveAsTextFile(args(1))  
    }  
}
```

Scala Ejemplo ejecución

- ▶ Spark-Submit ejecuta el programa, pasando la aplicación JAR, y los comandos de input con el archivo y la salida en el directorio output
- ▶ --class indica el nombre de la aplicación
- ▶ --master indica dónde se tiene que ejecutar, en este caso en una JVM en local

```
% spark-submit --class MaxTemperature --master local \
  spark-examples.jar input/ncdc/micro-tab/sample.txt output
% cat output/part-*
(1950,22)
(1949,111)
```

RDD Resilient distributed dataset

- ▶ RDD son los elementos claves en los trabajos de Spark
- ▶ Procesos a tener en cuenta con RDDs son
 - ▶ Creación
 - ▶ Transformación
 - ▶ Persistencia
 - ▶ Serialización

RDD - Creación

- ▶ Se pueden crear de tres maneras distintas
 - ▶ A partir de un objeto en memoria
 - ▶ Obtener un conjunto de datos por ejemplo de HDFS
 - ▶ Transformando un RDD existente

RDD Creación

- ▶ A partir de un objeto en memoria es interesante para realizar procesos de computación intensiva con pequeñas cantidades de información en paralelo
- ▶ El nivel de paralelismo se indica en la propiedad spark.default.parallelism que tiene como valor por defecto en local el número de cores de la máquina y en cluster el de todas las máquinas
- ▶ El método Parallelize indica el número de procesos paralelos

```
val params = sc.parallelize(1 to 10)
val result = params.map(performExpensiveComputation)
```

RDD - Creación

- ▶ Crearlo como una referencia de un dataset externo en un directorio local o en HDFS
- ▶ Utiliza internamente TextInputFormat de la API de MapReduce para leer la información, por lo que la división del archivo es la misma que en Hadoop, por lo que en el caso de HDFS se hace una partición por cada bloque de HDFS

```
val text: RDD[String] = sc.textFile(inputPath)
```

RDD - CREACIÓN

- ▶ También se puede leer todo el archivo en memoria como pares dónde se especifica la ruta del archivo y el contenido, esto solo es posible para archivos pequeños, ya que traslada todo el contenido a la memoria

```
val files: RDD[(String, String)] = sc.wholeTextFiles(inputPath)
```

RDD - transformaciones

- ▶ Spark tiene dos categorías
 - ▶ Transformaciones que generan un nuevo RDD a partir de uno existente
 - ▶ Acciones que realiza un proceso dentro del RDD y realiza una tarea con el resultado como mostrarlo al usuario o almacenarlo en disco
- ▶ Las acciones tienen un resultado inmediato, mientras que las transformaciones esperan a que se realice una acción en el RDD transformado
- ▶ Una manera de distinguirlas es mirar el tipo devuelto, si es RDD es una transformación, en cualquier otro caso es una acción.

RDD – Transformaciones Ejemplo

- Convertimos en minúsculas un RDD

```
val text = sc.textFile(inputPath)
val lower: RDD[String] = text.map(_.toLowerCase())
lower.foreach(println(_))
```

- Esta función no se ejecuta hasta que el método foreach es llamado, generando Spark un trabajo que lee el archivo de entrada y ejecuta la función en cada línea antes de imprimirla

RDD Transformaciones

- ▶ Hay tres funciones de agrupación que trabajan con pares key/value, , a partir de un conjunto de datos obtienen un valor único para cada clave
- ▶ ReducedByKey(), FoldByKey(), AggregateByKey()

```
val pairs: RDD[(String, Int)] =  
    sc.parallelize(Array(("a", 3), ("a", 1), ("b", 7), ("a", 5)))  
val sums: RDD[(String, Int)] = pairs.reduceByKey(_+_)  
assert(sums.collect().toSet === Set(("a", 9), ("b", 7)))
```

Rdd persistencia

- ▶ Cache() permite almacenar en memoria un RDD intermedio
- ▶ Al llamar al método cache() no cachea la información hasta que realizamos una acción, simplemente lo marca como cacheable.
- ▶ BlockManagerInfo muestra información de las particiones y del número de RDD en memoria
- ▶ No tiene que volver a cargarlo desde archivo como en MapReduce
- ▶ Existen diferentes tipos de persistencia MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, MEMORY_AND_DISK_SER

Rdd persistencia ejemplo

```
scala> val tuples = filtered.map(rec => (rec(0).toInt, rec(1).toInt))
tuples: org.apache.spark.rdd.RDD[(Int, Int)] = MappedRDD[4] at map at
<console>:18
```

```
scala> tuples.cache()
res1: tuples.type = MappedRDD[4] at map at <console>:18
```

```
scala> tuples.reduceByKey((a, b) => Math.max(a, b)).foreach(println(_))
INFO BlockManagerInfo: Added rdd_4_0 in memory on 192.168.1.90:64640
INFO BlockManagerInfo: Added rdd_4_1 in memory on 192.168.1.90:64640
(1950,22)
(1949,111)
```

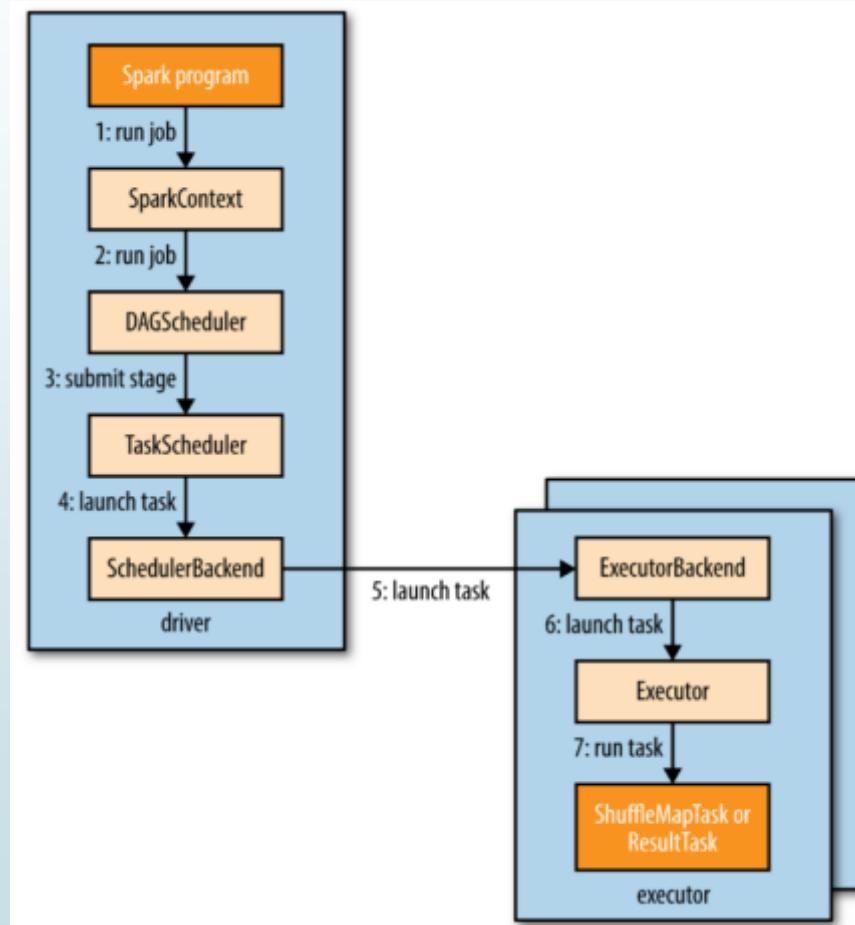
RDD Serialización

- ▶ Tenemos que tener en cuenta serialización de datos y serialización de funciones
- ▶ Serialización de datos utiliza por defecto la implementación `java.io.Serializable` o `java.io.Externalizable`
 - ▶ Se pueden utilizar otras serializaciones como Kryo, hay que modificar la configuración
 - ▶ `conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")`
- ▶ La serialización de funciones la realiza de forma automática Spark

Funcionamiento de los trabajos en spark

- ▶ Existen dos entidades independientes
 - ▶ Driver: que contiene la aplicación en ejecución. (sparkContext) y organizar la ejecución de las tareas
 - ▶ Executor: que es exclusivo de la aplicación y ejecuta las tareas
- ▶ Cuando se ejecuta un trabajo, se programa la ejecución y se divide en los procesos en etapas DAG y el programador que gestiona la ejecución de las etapas

Funcionamiento de los trabajos en spark

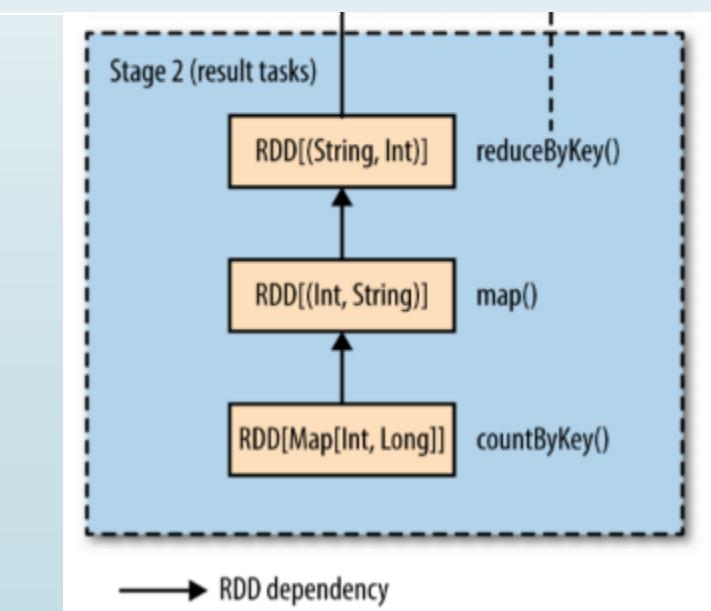
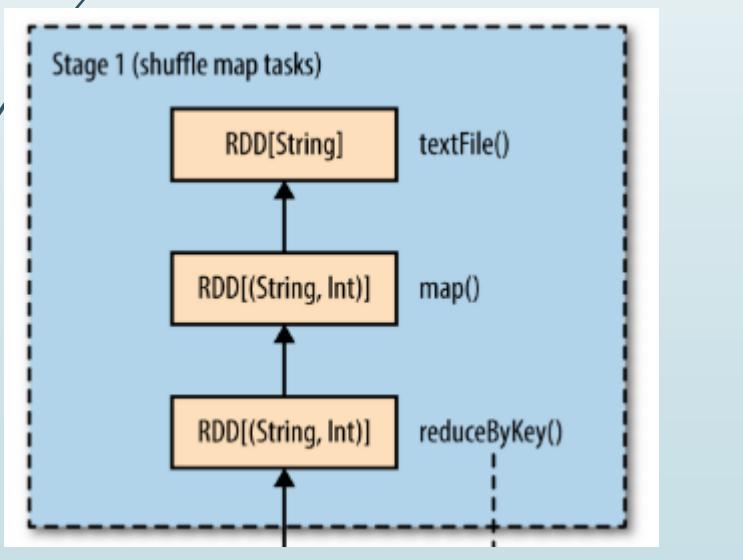


Funcionamiento de los trabajos en spark

- ▶ Programador DAG tiene la función de dividir los trabajos en etapas.
- ▶ Hay dos tipos de tareas
 - ▶ Tareas mapa mezclado (shuffle), similares a las tareas Map, ejecutan procesos en un RDD y el resultado lo escriben en una nueva partición en RDD. Estas tareas se ejecutan en todas las etapas menos en la final
 - ▶ Tareas de resultado, se ejecutan en la etapa final, que devuelve el resultado al programa

Funcionamiento de los trabajos en spark

```
val hist: Map[Int, Long] = sc.textFile(inputPath)
    .map(word => (word.toLowerCase(), 1))
    .reduceByKey((a, b) => a + b)
    .map(_.swap)
    .countByKey()
```



Funcionamiento de los trabajos en spark

- ▶ Programación de tareas, cuando al programador de tareas le envía un conjunto de tareas, usa su lista de ejecutores y construye una mapa de ejecución para las tareas
- ▶ Luego asigna las tareas a cada executor que tiene cores libres hasta que las tareas están completas
- ▶ El orden de asignación de las tareas a cada executor es primero las tareas locales, luego tareas del nodo y por último tareas locales

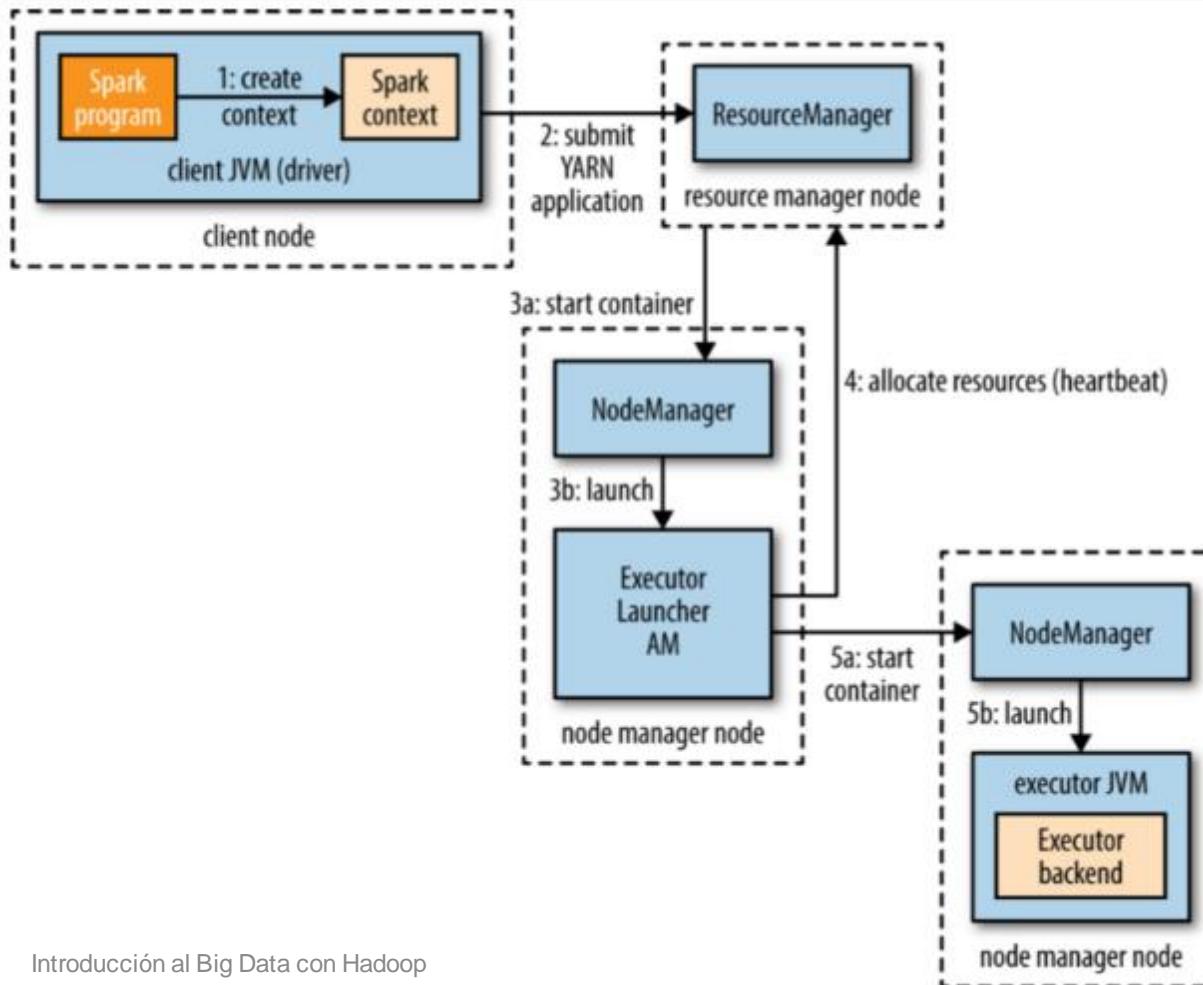
Funcionamiento de los trabajos en spark

- ▶ Ejecución de tareas
- ▶ El ejecutor ejecuta las tareas siguiendo un orden
 - ▶ Primero se asegura que el JAR y los archivos de dependencias están actualizados
 - ▶ Deserializa el código de las tareas, que se le envío junto con la tarea
 - ▶ Se ejecuta el código de la tarea en la misma JVM que el ejecutor

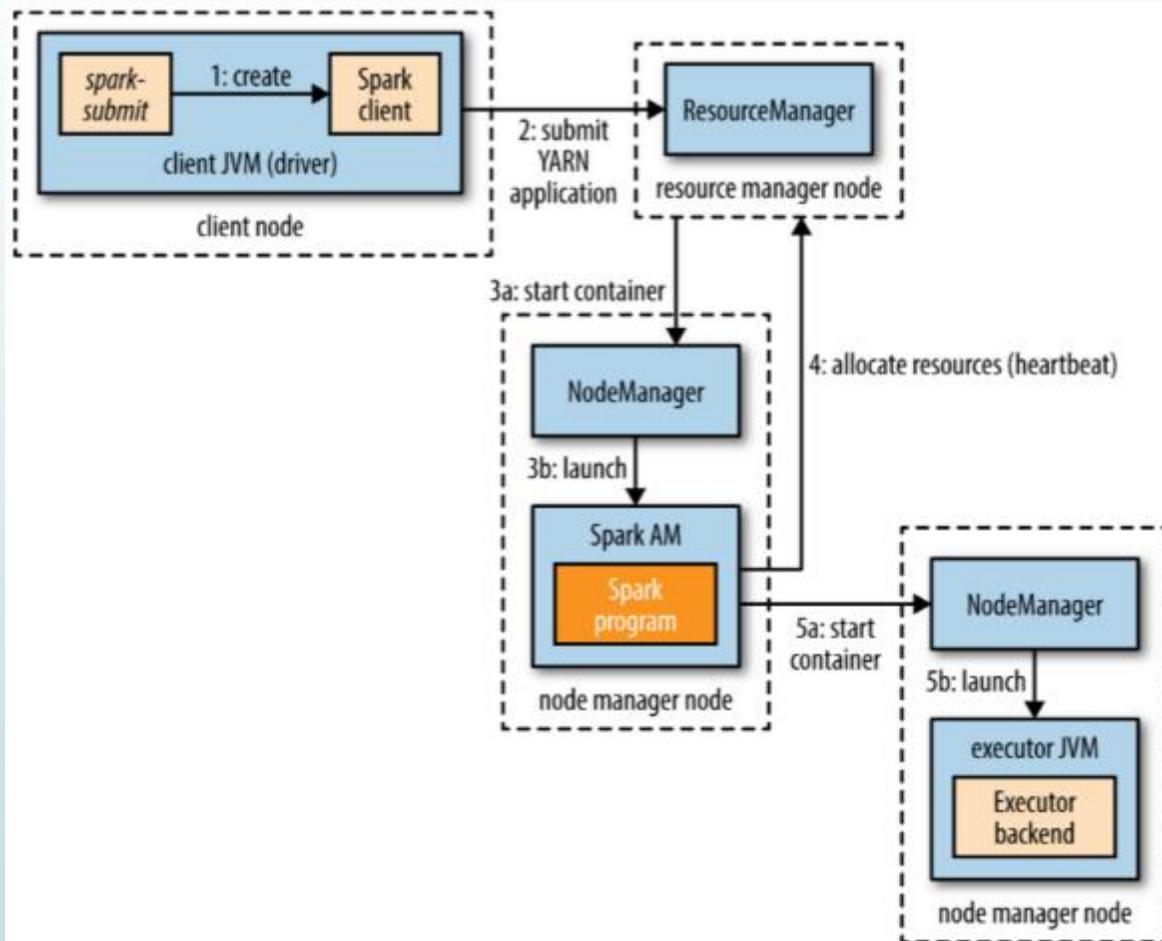
Spark con YARN

- ▶ Permite la mayor integración con otras herramientas Hadoop
- ▶ YARN Client Mode, cuando el driver se ejecuta en el cliente
 - ▶ Se utiliza en clientes que utilizan modos interactivos como Spark-Shell
- ▶ YARN cluster Mode, cuando el driver se ejecuta en el master application del cluster
 - ▶ Adecuado para entornos de producción, ya que la aplicación se ejecuta completamente en el servidor

Spark con YARN Client



Spark con YARN cluster



SPARK - Conclusiones

- ▶ Framework de desarrollo con kernel propio para trabajar con HDFS y YARN directamente sin depender de MapReduce
- ▶ Complementarlo con conjunto de herramientas como Spark Streaming

SPARK SQL, DataFrames y DataSet

- ▶ RDD es la abstracción más básica de Spark
 - ▶ Dependencias, indica como construir el RDD
 - ▶ Particiones, divide el trabajo para paralelizar el procesamiento
 - ▶ Funciones integradas
- ▶ Spark Structured APIs
 - ▶ Patrones para el proceso de datos. Filtros, selecciones, contar, agregaciones...
 - ▶ Java, Python, Spark R, SQL
 - ▶ Aplicación de esquema a los datos



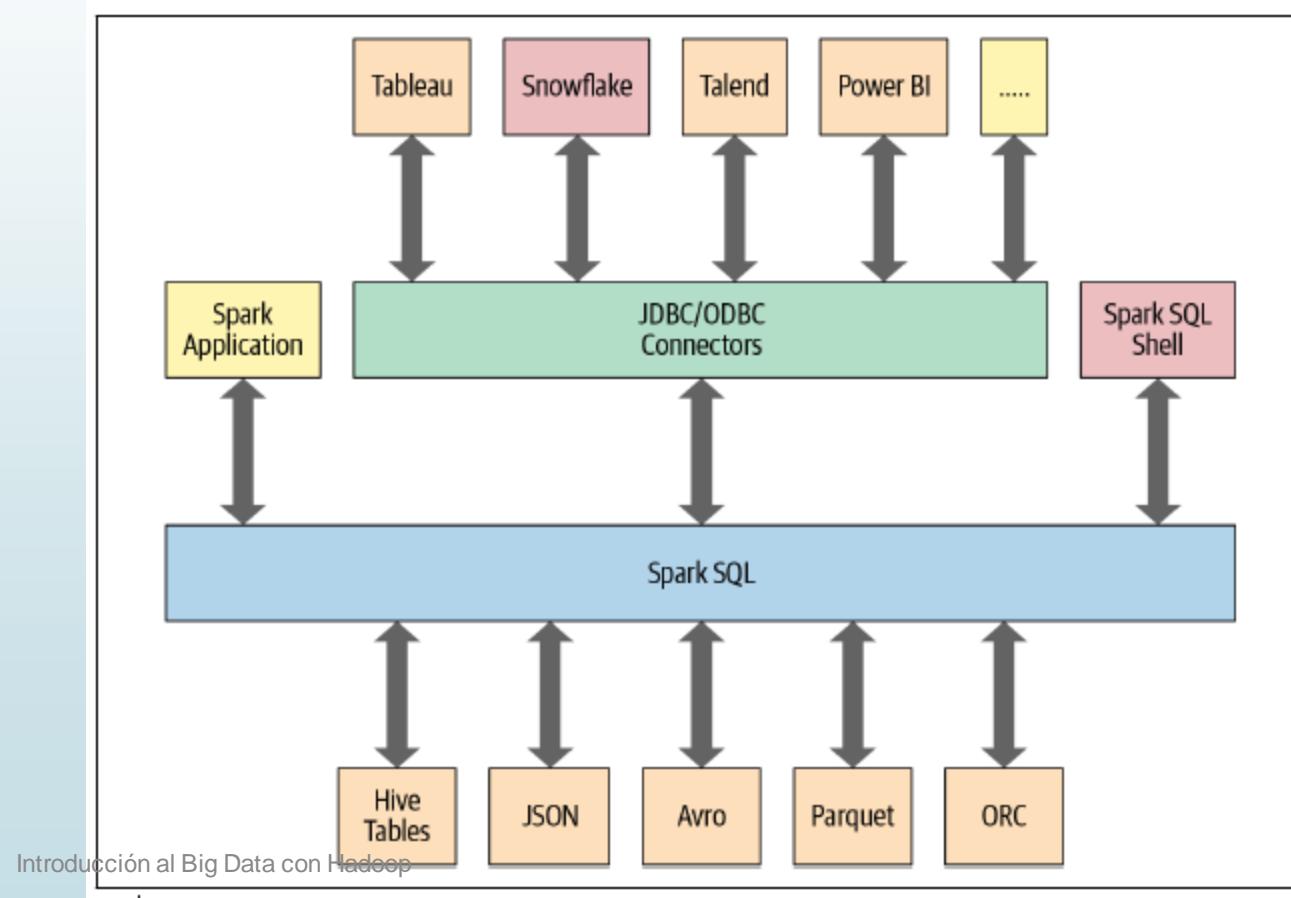
```
# In Python
# Create an RDD of tuples (name, age)
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30),
    ("TD", 35), ("Brooke", 25)])
# Use map and reduceByKey transformations with their lambda
# expressions to aggregate and then compute average

agesRDD = (dataRDD
    .map(lambda x: (x[0], (x[1], 1)))
    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
    .map(lambda x: (x[0], x[1][0]/x[1][1])))
```



name	avg(age)
Brooke	22.5
Jules	30.0
TD	35.0
Denny	31.0

Spark SQL, DAtaFrames, DATASET



Data type	Value assigned in Python	API to instantiate
ByteType	int	DataTypes.ByteType
ShortType	int	DataTypes.ShortType
IntegerType	int	DataTypes.IntegerType
LongType	int	DataTypes.LongType
FloatType	float	DataTypes.FloatType
DoubleType	float	DataTypes.DoubleType
StringType	str	DataTypes.StringType
BooleanType	bool	DataTypes.BooleanType
DecimalType	decimal.Decimal	DecimalType

Data type	Value assigned in Python	API to instantiate
BinaryType	bytearray	BinaryType()
TimestampType	datetime.datetime	TimestampType()
DateType	datetime.date	DateType()
ArrayType	List, tuple, or array	ArrayType(dataType, [nullable])
MapType	dict	MapType(keyType, valueType, [nullable])
StructType	List or tuple	StructType([fields])
StructField	A value type corresponding to the type of this field	StructField(name, dataType, [nullable])

```
from pyspark.sql.types import *
schema = StructType([StructField("author", StringType(), False),
                     StructField("title", StringType(), False),
                     StructField("pages", IntegerType(), False)])
```

```
schema = "author STRING, title STRING, pages INT"
```

```
// Create a DataFrame by reading from the JSON file
// with a predefined schema
val blogsDF = spark.read.schema(schema).json(jsonFile)
// Show the DataFrame schema as output
blogsDF.show(false)
```

Dataframe col - row

- ▶ Col, devuelve un tipo Column
 - ▶ .WithColumns() permite trabajar con columnas
- ▶ Row, devuelve un objeto con una o varias columnas
 - ▶ Permite recorrer la colección de columnas con un índice
 - ▶ Puede usarse como método para crear DataFrames

```
# In Python
rows = [Row("Matei Zaharia", "CA"), Row("Reynold Xin", "CA")]
authors_df = spark.createDataFrame(rows, ["Authors", "State"])
authors_df.show()
```

DAtaFrame Operaciones comunes

- ▶ DataFrameReader. Interfaz para leer información
- ▶ DataFrameWriter. Interfaz para escribir información en múltiples formatos

```
# Use the DataFrameReader interface to read a CSV file
sf_fire_file = "/databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv"
fire_df = spark.read.csv(sf_fire_file, header=True, schema=fire_schema)
```

SPARK DataSet

- ▶ Siempre necesitamos tener el esquema de los datos. DataFrames permite trabajar con datos sin estructura.
- ▶ Utilizado por Java y Scala
- ▶ Dispone de funciones a alto nivel para realizar operaciones

Ejemplos

- ▶ 9 Ejecución de aplicación con SQL DataFrames y Databricks
- ▶ 10 Integración con herramientas de BI

SPARK SQL – Instrucciones SQL

- ▶ Crea las tablas en la base de datos que se esté utilizando

```
spark.sql("CREATE DATABASE learn_spark_db")
spark.sql("USE learn_spark_db")
```

- ▶ Utilizar spark.sql (“ sentencia SQL “)

```
spark.sql("CREATE TABLE managed_us_delay_flights_tbl (date STRING, delay INT,
distance INT, origin STRING, destination STRING)")
```

- ▶ Tiene el equivalente en DataFrame API

```
# In Python
# Path to our US flight delays CSV file
csv_file = "/databricks-datasets/learning-spark-v2/flights/departuredelays.csv"
# Schema as defined in the preceding example
schema="date STRING, delay INT, distance INT, origin STRING, destination STRING"
flights_df = spark.read.csv(csv_file, schema=schema)
flights_df.write.saveAsTable("managed_us_delay_flights_tbl")
```

Spark SQL UDF (User Defined functions)

- ▶ Funciones de usuario.
- ▶ Se generan para la sesión activa y no perduran en el tiempo
- ▶ Se crean con “def” <nombre>= <codigo spark>
- ▶ @udf, permite definir una función y convertirla a udf en un solo paso

```
states = ["CA", "TX", "NY", "WA"]

@udf(returnType=StringType())
def random_state():
    return str(random.choice(states))
```

Spark Streaming

- ▶ Extensión del core de Spark
- ▶ Ofrece procesamiento de datos de live stream con tolerancia a fallos y alta escalabilidad
- ▶ Admite una gran cantidad de orígenes de datos Kafka, Flume, Kinesis o TCP sockets
- ▶ Permite procesar la información utilizando algoritmos de alto nivel como map, reduce, join y window
- ▶ La salida se puede redirigir al sistema de archivos, bases de datos

Conceptos



Conceptos

- ▶ Spark streaming recibe los datos en live streaming
- ▶ Divide la tabla en bloques
- ▶ Estos bloques son procesados por el motor de Spark Stream
- ▶ Generan bloques de información procesada
- ▶ Utiliza un nivel alto de abstracción para la serialización llamado Discretized Stream (Dstream)

Conceptos



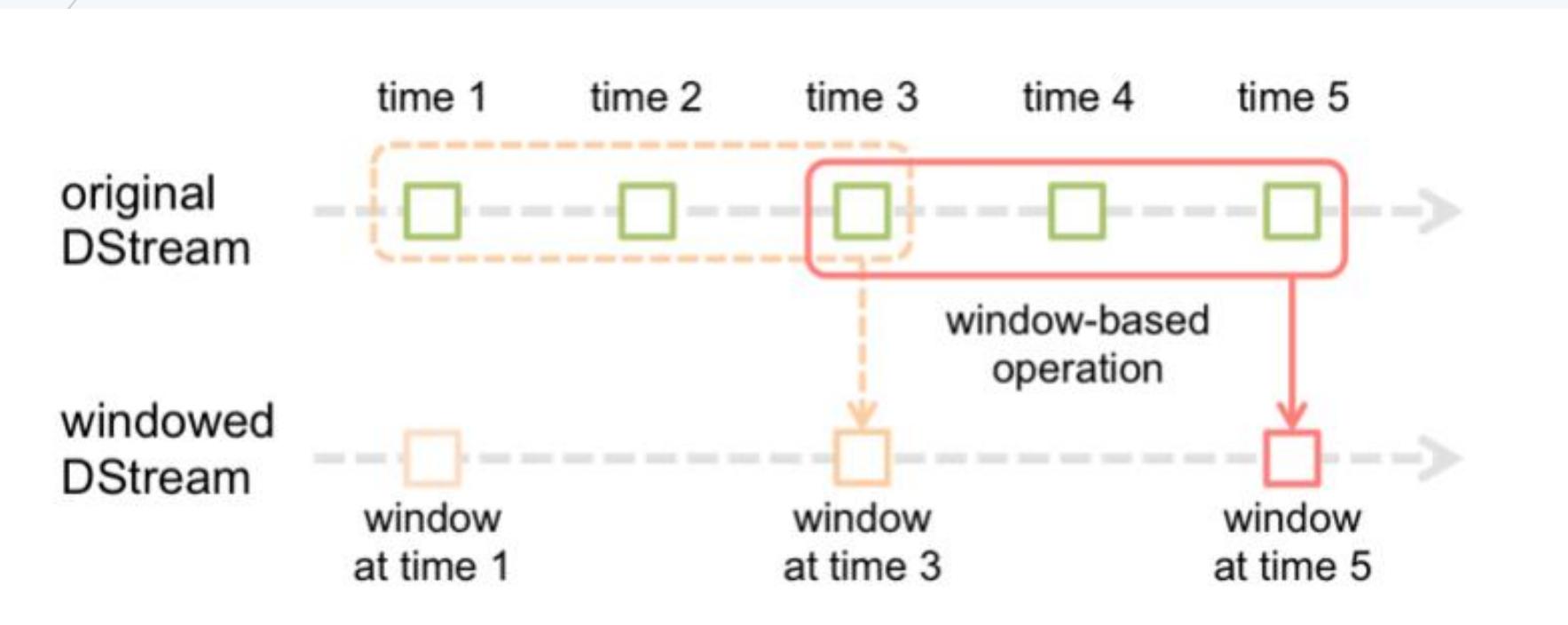
Elementos de Programación

- ▶ De forma similar a otros entornos, lo primero es crear un objeto contexto de la clase SparkConf. El espacio de nombres es org.apache.spark.streaming
- ▶ Definir el origen de datos para Dstream
- ▶ Definir las tareas de computación y transformación para Dstream
- ▶ Iniciar la recepción de los datos con streamingContext.start()
- ▶ Esperar que el proceso acabe (manualmente o por error) streamingContext.stop() o streamingContext.awaitTermination()

Procesamiento Window

- ▶ Permite establecer procesos y transformaciones sobre una ventana de datos en movimiento
- ▶ Cada vez que la ventana se mueve por el stream, los RDDs se combinan y se operan para producir el RDD de la ventana
- ▶ Hay que definir la longitud de la ventana y cada cuanto se agrupa la información para procesarla.
- ▶ Estos dos parámetros tienen que ser múltiplos del que define la división en bloques del stream

Procesamiento Window



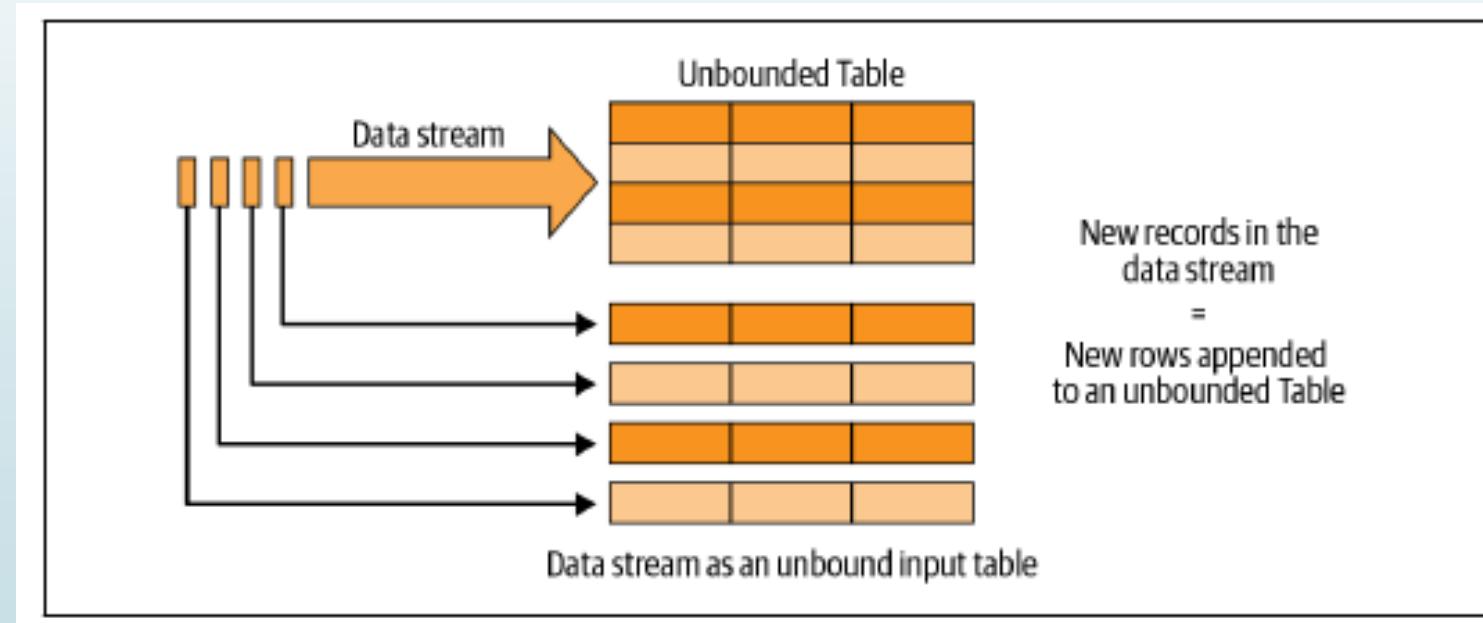
Procesamiento Window

- ▶ Spark Streaming dispone de varias funciones para procesar RDDs
 - ▶ `window(windowLength, slideInterval)` → Dstream para trabajar
 - ▶ `countByWindow(windowLength, slideInterval)` → cuenta los elementos en la ventana
 - ▶ `reduceByWindow(func, windowLength, slideInterval)` → devuelve un stream con los elementos en ese intervalo aplicando una función
 - ▶

```
// Reduce last 30 seconds of data, every 10 seconds
val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10))
```

Spark Structured Streaming

- ▶ Cada registro se agrega a la tabla de datos



Fundamentos structured streaming query

- ▶ Crear DataFrames con spark.ReadStream para crear un DataStreamReader
- ▶ Procesos de transformación son los mismos
- ▶ Escribir resultados con spark.WriteStream
- ▶ Definir procesamiento de los datos. Trigger (default, Processing Time, Once, Continuous, CheckPoint location)
- ▶ Iniciar query

Código de ejemplo

```
# In Python
from pyspark.sql.functions import *
spark = SparkSession...
lines = (spark
    .readStream.format("socket")
    .option("host", "localhost")
    .option("port", 9999)
    .load())

words = lines.select(split(col("value"), "\\s").alias("word"))
counts = words.groupBy("word").count()
checkpointDir = "..."
streamingQuery = (counts
    .writeStream
    .format("console")
```

```
.outputMode("complete")
.trigger(processingTime="1 second")
.option("checkpointLocation", checkpointDir)
.start())
streamingQuery.awaitTermination()

// In Scala
import org.apache.spark.sql.functions._
import org.apache.spark.sql.streaming._
val spark = SparkSession...
val lines = spark
    .readStream.format("socket")
    .option("host", "localhost")
    .option("port", 9999)
    .load()

val words = lines.select(split(col("value"), "\\s").as("word"))
val counts = words.groupBy("word").count()

val checkpointDir = "..."
val streamingQuery = counts.writeStream
    .format("console")
    .outputMode("complete")
    .trigger(Trigger.ProcessingTime("1 second"))
    .option("checkpointLocation", checkpointDir)
    .start()
streamingQuery.awaitTermination()
```

Data Lakes

- ▶ Solución de almacenamiento distribuida.
- ▶ Permite grandes cantidades de datos
- ▶ Apache Spark soporta diferentes Data Lakes
- ▶ Delta Lake, proyecto open source de los creadores de Apache Spark
 - ▶ Soporta funciones ACID

Delta lake

- ▶ Cargar datos en una tabla Delta Lake, indicar en spark.write el formato “delta”

```
spark
  .read
  .format("parquet")
  .load(sourcePath)
  .write
  .format("delta")
  .save(deltaPath)
```

- ▶ Agregar una vista a la tabla con .createOrReplaceTempView

```
# Create a view on the data called loans_delta
spark.read.format("delta").load(deltaPath).createOrReplaceTempView("loans_delta")
```

Delta Lake con Streams

- ▶ Permite agregar información desde archivos y streams a la misma tabla
- ▶ Se puede añadir información con múltiples streams
- ▶ Garantiza los procesos ACID (actualizar, insertar y borrar)
- ▶ Permite forzar esquemas a la hora de escribir para prevenir errores

Delta Lake Transformaciones

- ▶ Actualizaciones, método update

```
# In Python
from delta.tables import *

deltaTable = DeltaTable.forPath(spark, deltaPath)
deltaTable.update("addr_state = 'OR'", {"addr_state": "'WA'")
```

- ▶ Borrado selectivo de información

```
# In Python
deltaTable = DeltaTable.forPath(spark, deltaPath)
deltaTable.delete("funded_amnt >= paid_amnt")
```

- ▶ Combinación datos, merge

```
# In Python
(deltaTable
 .alias("t")
 .merge(loanUpdates.alias("s"), "t.loan_id = s.loan_id")
 .whenMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute())
```

Delta Lake - histórico

- Guarda un registro de transacciones donde guarda los commit realizados

```
// In Scala/Python  
deltaTable.history().show()
```

version	timestamp	operation	operationParameters
5	2020-04-07	MERGE	[{"predicate": "t.`loan_id` = s.`loan_id`"}]
4	2020-04-07	MERGE	[{"predicate": "t.`loan_id` = s.`loan_id`"}]
3	2020-04-07	DELETE	[{"predicate": "(CAST(`funded_amnt` ..."}]

DeLTa LaKE Versiones previas

- Dispone de instantáneas de versiones anteriores con timestampAsOf y versionAsOf

```
# In Python
(spark.read
  .format("delta")
  .option("timestampAsOf", "2020-01-01") # timestamp after table creation
  .load(deltaPath))

(spark.read.format("delta")
  .option("versionAsOf", "4")
  .load(deltaPath))
```

Ejemplos

- ▶ 11 Carga de datos en streaming
- ▶ 12 Vinculación con Power BI
- ▶ 13 Transformaciones Delta Lake

Apache Mahout

Conceptos

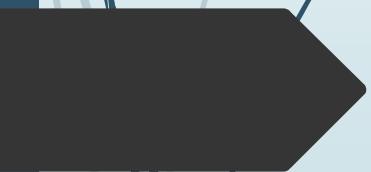
- ▶ La idea es tener una implementación de los algoritmos utilizados normalmente dentro del Machine Learning
- ▶ Esta pensado para bases de datos grandes, con un enfoque Big Data
- ▶ Está escrito en Java y se basa en Hadoop y MapReduce

Conceptos

- ▶ Ya existen soluciones de machine learning, como Weka en Java, lenguaje R y otras, pero no están pensadas para grandes volúmenes de datos
- ▶ Crear modelos que permitan analizar la información cada vez es más difícil por el volumen y la variedad de los datos
- ▶ Hay muchas empresas que están adoptando Hadoop para analizar bigdata, por lo que es más fácil incorporar Mahout

Requisitos

- Dentro del ecosistema Hadoop necesitamos una serie de herramientas para trabajar con Mahout. Las versiones mínimas son
 - Java 1.6
 - Maven 2.0
 - Hadoop 1.2.1
 - Eclipse with Maven plugin
 - Mahout 0.9



PIG

<http://pig.apache.org/>

conceptos

- ▶ Se basa en MapReduce
- ▶ Le dota de más complejidad a la hora de procesar la información
- ▶ Está formado por
 - ▶ El lenguaje utilizado para representar los flujo es Pig Latin
 - ▶ El entorno de ejecución, en local con una JVM y en entorno distribuido en Hadoop cluster
- ▶ Cada programa contiene una serie de operaciones o transformaciones que se aplican a unos datos de entrada para lograr una salida

conceptos

- ▶ Pig es un lenguaje de Scripts, pensado para explorar grandes conjuntos de datos
- ▶ Una de las quejas de MapReduce es el tiempo de desarrollo y compilación de los diferentes elementos.
- ▶ Pig permite analizar datos con unas pocas líneas de flujo de datos de Pig Latin, aligerando los tiempos de desarrollo.

MODOS DE EJECUCIÓN

- ▶ Modo local, se ejecuta en una sola JVM y accede al sistema de archivos local
 - ▶ Dispone de un Shell interactivo llamado GRUNT
- ▶ Modo MapReduce (en la última versión también está para Spark)
 - ▶ Traduce las queries a trabajos MapReduce y los ejecuta en el cluster de Hadoop
 - ▶ Pigs tiene que ser compatible con la versión de Hadoop que se almacena en la variable HADOOP_HOME

Modos de ejecución

- ▶ Los programas Pig se pueden ejecutar de tres maneras
 - ▶ Script: ejecuta comandos almacenados en un archivo
 - ▶ Grunt: permite ejecutar comandos de forma interactiva y también contenidos en archivos. Shell interactivo que tiene ayudas en la creación de script como el editor ReadLine (CTRL E, tabulador completa comandos...)
 - ▶ Embedded: utilizando la clase PigServer permite integrar comandos Pig en java

ejemplo

- Para crear el mismo ejemplo que con MapReduce tenemos un código menor en PigLatin

```
-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
    AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
    quality IN (0, 1, 4, 5, 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
    MAX(filtered_records.temperature);
DUMP max_temp;
```

Ejemplo

- ▶ Load, es un operador de relaciones que produce un conjunto de tuplas, en el ejemplo de años, temperatura y calidad
- ▶ La notación de los parámetros en nombre del campo:tipo de datos

```
grunt> records = LOAD 'input/ncdc/micro-tab/sample.txt'  
>> AS (year:chararray, temperature:int, quality:int);
```

Ejemplo

- ▶ Podemos ver el contenido de la relación “records” con DUMP
- ▶ Podemos saber la estructura de cada tupla con DESCRIBE

```
grunt> DUMP records;  
(1950,0,1)  
(1950,22,1)  
(1950,-11,1)  
(1949,111,1)  
(1949,78,1)
```

```
grunt> DESCRIBE records;  
records: {year: chararray,temperature: int,quality: int}
```

Ejemplo

- Guardamos las tuplas que no cumplen las condiciones de temperatura

```
grunt> filtered_records = FILTER records BY temperature != 9999 AND
>>   quality IN (0, 1, 4, 5, 9);
grunt> DUMP filtered_records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
```

ejemplo

- ▶ Group, para agrupar información, en este caso por años, genera una tupla con cada registro agrupado y sus valores
- ▶ Los valores los representa como una lista desordenada de valores entre { }. Pig los identifica como Bag

```
grunt> grouped_records = GROUP filtered_records BY year;
grunt> DUMP grouped_records;
(1949,{(1949,78,1),(1949,111,1)})
(1950,{(1950,-11,1),(1950,22,1),(1950,0,1)})
```

Ejemplo

- ▶ Una vez agrupado , tenemos que obtener la temperatura.
- ▶ Es importante entender el formato de la información agrupada para obtener el valor del campo deseado.
- ▶ FOREACH, procesa cada fila para obtener el resultado

```
grunt> DESCRIBE grouped_records;
grouped_records: {group: chararray,filtered_records: {year: chararray,
temperature: int,quality: int}}
```

```
grunt> max_temp = FOREACH grouped_records GENERATE group,
>>   MAX(filtered_records.temperature);
```

Ejemplo

- Podemos visualizar de forma más clara los pasos con ILLUSTRATE, que muestra la información en formato tabular con encabezados

```
grunt> ILLUSTRATE max_temp;  
-----  
| records | year:chararray | temperature:int | quality:int |  
|         | 1949             | 78              | 1           |  
|         | 1949             | 111             | 1           |  
|         | 1949             | 9999            | 1           |  
-----  
| filtered_records | year:chararray | temperature:int | quality:int |  
|                 | 1949             | 78              | 1           |  
|                 | 1949             | 111             | 1           |  
-----  
| grouped_records | group:chararray | filtered_records:bag{:tuple( |  
|                   |                   | year:chararray,temperature:int, |  
|                   |                   | quality:int)} |  
|                   | 1949             | {(1949, 78, 1), (1949, 111, 1)} |
```

Pig y bases de datos

- ▶ Aunque hay comando parecidos como Group o Describe, hay diferencias
- ▶ La principal es que Pig Latin es una programación de flujo de datos mientras que SQL es declarativo,
 - ▶ Uno va paso a paso ejecutando instrucciones y el otro es un conjunto de instrucciones que se ejecutan a la vez
- ▶ RDBMS almacena datos en tablas con esquemas fijos, mientras que PIG no necesita un esquema o se puede definir en tiempo de ejecución
- ▶ Pig soporta estructuras complejas, mientras que SQL trabaja con estructuras de datos planas

Pig latin estamentos

- ▶ La estructura de un programa de Pig Latin es una colección de estamentos separados por ";" por lo que un estamento puede ocupar varias líneas
- ▶ El interprete de Pig Latin, analiza el código para verificar que está bien construido añadiendo al plan lógico del programa cada línea que va interpretando.
- ▶ Hasta que no llega a un estamento que obliga a compilar el plan lógico en plan físico.
- ▶ En el ejemplo hasta que no llega a la última instrucción DUMP
- ▶ EL plan físico es un conjunto de MapReduce que se ejecutan en la JVM o en Hadoop
- ▶ El plan físico se puede ver con el comando Explain

Pig Latin Tipos

- Se basa en los tipos de Java

Pig – Latin Schemas

- ▶ Una relación en Pig tiene asociado un esquema, que da a los elementos en la relación, nombres y tipos
- ▶ Se puede definir el esquema de multiples maneras, sólo los nombres, los nombres y algunos tipos o sin esquema
- ▶ Si no se define el esquema, sólo se puede acceder a los campos por su posición
- ▶ Para no tener que repetir el esquema se puede utilizar Hcatalog que es un componente de HIVE que permite almacenar la definición de los esquemas

```
% pig -useHCatalog  
grunt> records = LOAD 'records' USING org.apache.hcatalog.pig.HCatLoader();
```

Pig Esquemas

- ▶ En el caso de que un dato no pueda encajar con el esquema dado, lo sustituye por NULL
- ▶ No para los procesos, muestra un warning
- ▶ Este tipo Null se puede usar para filtrar los datos y quitar los que no nos valen

```
grunt> corrupt_records = FILTER records BY temperature is null;
```

```
grunt> grouped = GROUP corrupt_records ALL;
grunt> all_grouped = FOREACH grouped GENERATE group, COUNT(corrupt_records);
```

Pig funciones

- ▶ Tenemos cuatro tipos de funciones
 - ▶ Funciones de Evaluación, toman una o varias expresiones y devuelven otra expresión, por ejemplo MAX
 - ▶ Funciones de Filtro, es un tipo de funciones de evaluación, que devuelven un booleano. Se utilizan con FILTER para realizar acciones, por ejemplo IsEmpty
 - ▶ Función LOAD, especifica como cargar los datos
 - ▶ Función Store, funciones que almacenan el contenido extenamente

Funciones UDF (user defined functions)

- ▶ Pig facilita la creación y reutilización de funciones de usuario como elemento clave en la programación.
- ▶ Podemos crear cualquier tipo de función, en general las funciones implementan la clase abstracta EvalFunc.

```
public abstract class EvalFunc<T>  
{ public abstract T exec(Tuple input) throws IOException; }
```

Funciones udf ejemplo filtro

- Generar una función de filtro para quitar las tuplas que no son validas (Ejemplo en imagen a parte)

```
filtered_records = FILTER records BY temperature != 9999 AND  
quality IN (0, 1, 4, 5, 9);
```



```
grunt> filtered_records = FILTER records BY temperature != 9999 AND  
>> com.hadoopbook.pig.IsGoodQuality(quality);
```

Paralelismo

- ▶ Cuando se ejecuta en modo MapReduce, calcula que por cada Giga de input en las tareas map necesita una tarea reducer adicional con un máximo de 999 reducers
- ▶ Se puede personalizar desde los parámetros `pig.exec.reducers .bytes.per.reducer` y `pig.exec.reducers .max`
- ▶ También se puede establecer “`grunt> set default_parallel 30`” y tendrá efecto para las consultas posteriores

- ▶ Aunque tiene parecido a comandos SQL la forma de trabajo es totalmente distinta
- ▶ Facilita la creación de trabajos de MapReduce reduciendo los tiempos de desarrollo de las funciones.



Otras Soluciones de Big Data

Cloudera

- ▶ [Cloudera Manager: Hadoop Administration tool](#)
- ▶ [Hortonworks Sandbox \(cloudera.com\)](#)

Amazon

- ▶ [Análisis big data | Ejecución marcos Hadoop | Amazon EMR](#)
 - ▶ [Apache Hadoop en Amazon EMR - Plataforma para big data - Amazon Web Services](#)
 - ▶ [Databricks en la nube de AWS \(amazon.com\)](#)
 - ▶ [AWS Cloud9 – Amazon Web Services](#)

Google cloud

- ▶ [Servicios de cloud computing | Google Cloud](#)
 - ▶ [Dataproc | Google Cloud](#)
 - ▶ [Kubernetes: Google Kubernetes Engine \(GKE\) | Kubernetes Engine](#)

Microsoft Azure

- ▶ [Directorio de Azure Cloud Services | Microsoft Azure](#)
 - ▶ [Azure HDInsight: Hadoop, Spark y Kafka | Microsoft Azure](#)
 - ▶ [Azure Databricks | Microsoft Azure](#)
 - ▶ [Visual Studio Code - Code Editing. Redefined](#)

Análisis de datos con Tableau

- [Software de análisis e inteligencia de negocios \(tableau.com\)](http://tableau.com)

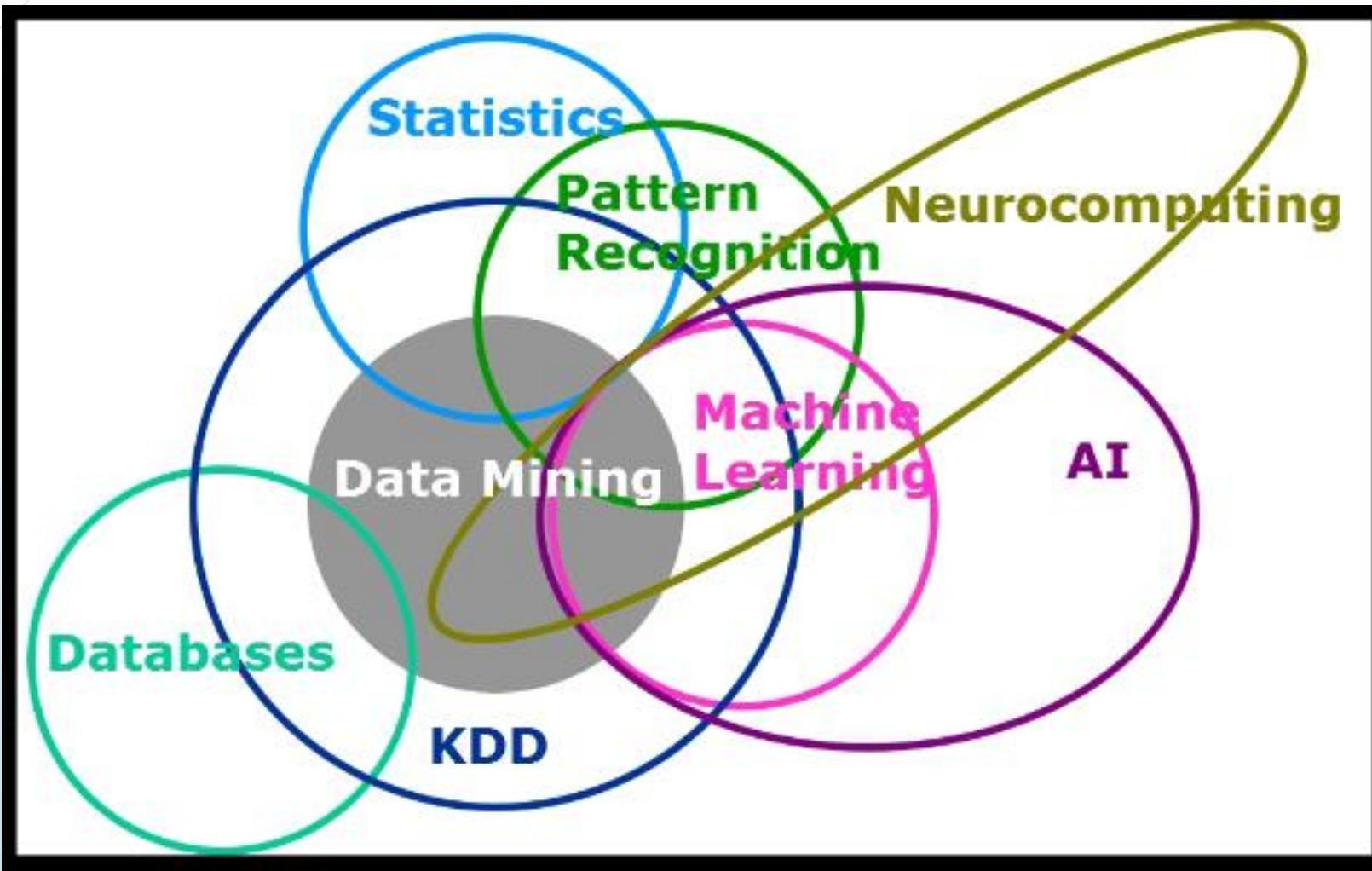


BlockChain

- ▶ ¿Qué es BlockChain ([IEBS](#))?
- ▶ <https://www.ibm.com/es-es/topics/what-is-blockchain>
 - ▶ Casos de uso, visión general
- ▶ <https://es.hyperledger.org/>
 - ▶ Framework de desarrollo Blockchain
- ▶ <https://consensys.net/quorum/qbs/>
 - ▶ Adoptado por Microsoft



Machine Learning, IA, DataSets



Machine Learning, IA, DataSets

- ▶ [Tensor Flow](#)
 - ▶ Plataforma para crear modelos de aprendizaje automático
- ▶ [MNIST](#) (Modified National Institute of Standards and Technology)
 - ▶ Base de datos de imágenes
- ▶ <https://www.kaggle.com/>
 - ▶ Código de programación
 - ▶ DataSets
- ▶ [Azure Open DataSets](#)
 - ▶ Conjuntos de datos para pruebas
 - ▶ Ejemplos de programación

Machine Learning, IA, DataSets

- ▶ [DALL-E](#)
 - ▶ Creación de imágenes por IA
 - ▶ Ejemplo de utilización

- ▶ [DALL-E 2](#)
 - ▶ Api de desarrollo
 - ▶ Fusión de conceptos

