Introducción a Hadoop

Patrocinado por Bahía Software

Tomás Fernández Pena

Centro Singular de Investigación en Tecnoloxías da Información

Universidade de Santiago de Compostela

Curso de verano BDDS

citius.usc.es

¿Qué es MapReduce?

Modelo de programación data-parallel diseñado para escalabilidad y tolerancia a fallos en grandes sistemas de commodity hardware

- Basado en la combinación de operaciones Map y Reduce

Diseñado originalmente por Google (2004)

- Usado en múltiples operaciones
- Manejo de varios petabytes diarios

Popularizado por la implementación open source Apache Hadoop

- Usado por múltiples organizaciones como Facebook, Twitter, Tuenti, Last.fm, eBay, LinkedIn, Rackspace, Yahoo!, AWS, etc.

Hadoop

Implementación open-source de MapReduce

- Procesamiento de enormes cantidades de datos en grandes clusters de hardware barato (commodity clusters)
  - Escala: petabytes de datos en miles de nodos

Características de Hadoop

Incluye

- Almacenamiento: HDFS
- Procesamiento: MapReduce

Ventajas

- Bajo coste: clusters baratos o cloud
- Facilidad de uso
- Tolerancia a fallos

Instalación

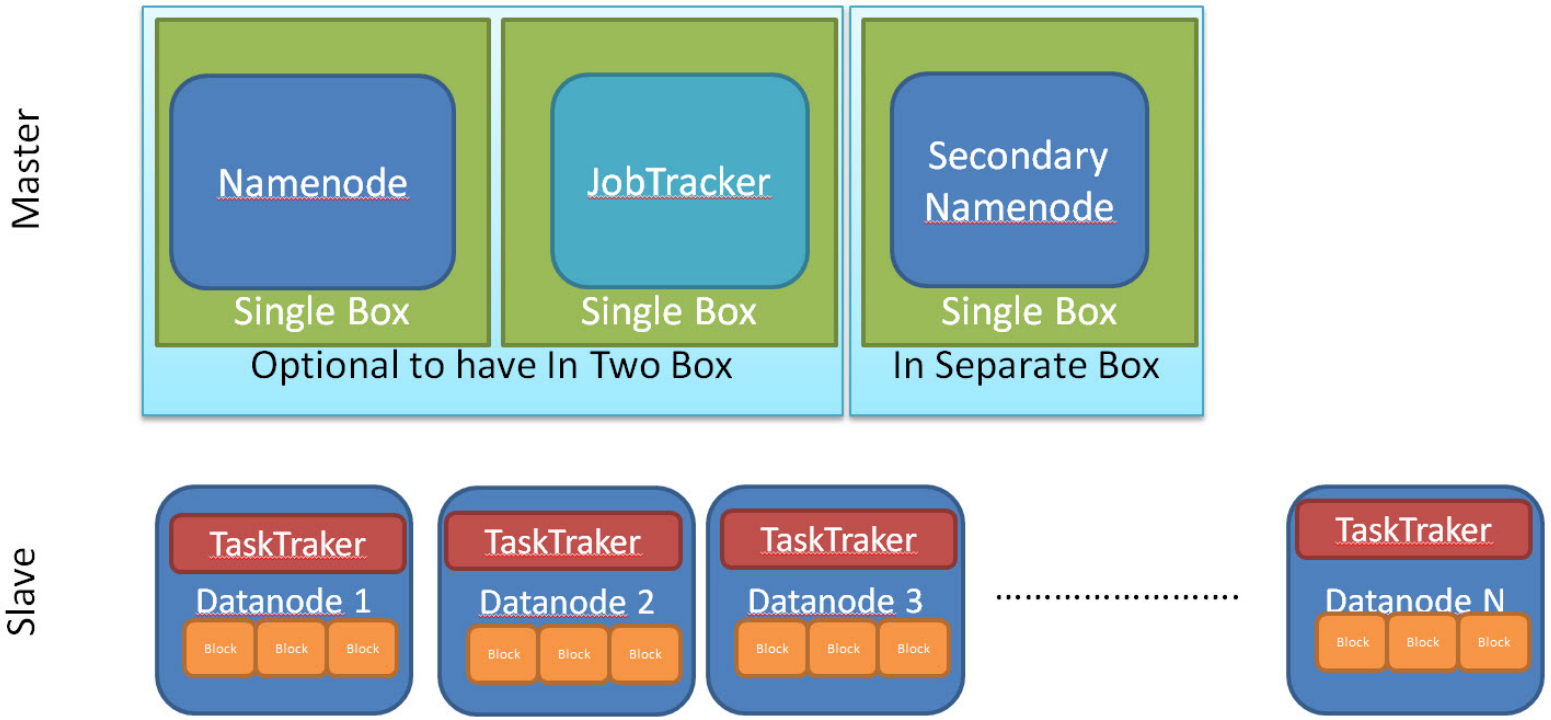
Instalación relativamente simple: aplicación Java (múltiples opciones de optimización)

- Paquete fuente: http://hadoop.apache.org/releases.html
- Sistemas preconfigurados proporcionados por empresas como Cloudera (www.cloudera.com), MapR (www.mapr.com) o Hortonworks (hortonworks.com)

Modos de funcionamiento:

- Standalone: todo en un nodo, para pruebas
- Pseudodistribuido: funciona como una instalación completa, pero en un solo nodo
- Totalmente distribuido, en un cluster

Arquitectura



Cluster de prueba (instalado en AWS):

- 6 máquinas en AWS: 1 máster, 4 workers y 1 máster secundario

YARN

Este esquema se ha modificado en Hadoop 2.0 con YARN (**Yet Another Resource Negociator**)

- separa las dos funcionalidades del Jobtracker (gestión de recursos y job-scheduling/monitorización) en demonios separados
- permite que diferentes tipos de aplicaciones (no solo MapReduce) se ejecuten en el cluster

Ecosistema Hadoop (I)

Diversas tecnologías relacionadas:

- Pig: lenguaje data-flow de alto nivel para facilitar la programación MapReduce
- Hive: infraestructura de data-warehouse construida sobre Hadoop
- Avro: sistema de serialización de datos (alternativa a los Writables)
- Oozie, Cascading, Azkaban, Hamake: planificadores de workflows para gestionar trabajos Hadoop
- Whirr: herramientas para iniciar clusters Hadoop y otros servicios en diferentes proveedores cloud
- Ambari: herramienta basada en web para provisionar, gestionar y monitorizar clusters Hadoop
- Hue: interfaz web para simplificar el uso de Hadoop

Ecosistema Hadoop (II)

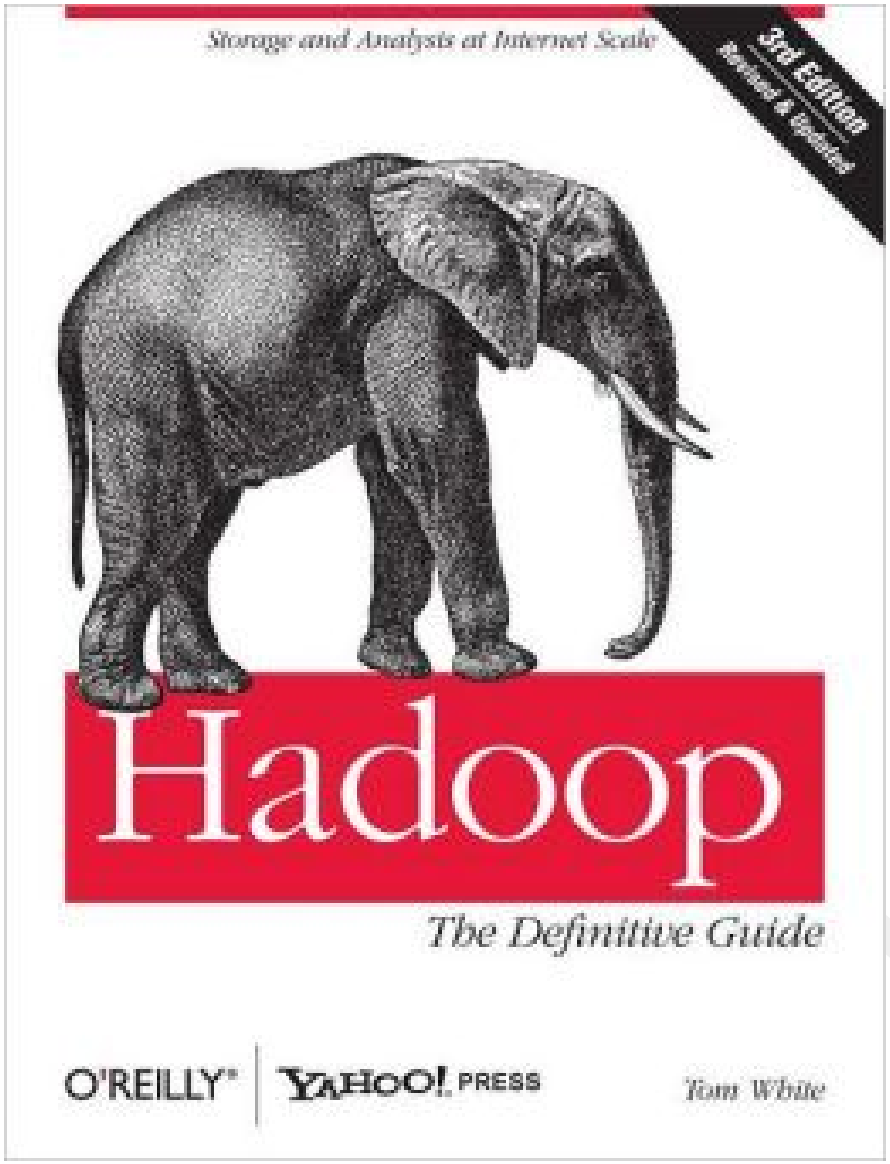
- HBase: base de datos distribuida no-relacional (NoSQL) que corre sobre HDFS (inspirado en Google BigTable)
- Sqoop: transferencia eficiente de datos eficiente entre Hadoop y bases de datos relacionales
- ZooKeeper: servicio centralizado de configuración, nombrado, sincronización distribuida y servicios de grupos para grandes sistemas distribuidos
- HCatalog: capa de abstracción de diferentes formatos de datos en Hadoop (texto, CSV, RCFiles o Sequence Files)
- WebHCat: API REST-like para HCatalog y componentes Hadoop relacionados (antes Templeton)

Ecosistema Hadoop (III)

- S4, Storm: procesamiento de flujos de datos (stream processing)
- Hama: framework de computación Bulk Synchronous Parallel sobre HDFS
- Flume: obtención, agregación y movimiento de grandes ficheros de log a HDFS
- Twister: aplicaciones MapReduce iterativas
- Mahout: algoritmos escalables de machine learning y minería de datos sobre Hadoop
- Chukwa: sistema de recogida de datos para monitorizar grandes sistemas distribuidos

Más ejemplos en bigdata-hadoop.pragsis.com/pages/2/glosario_big_data

El libro


Storage and Analysis at Internet Scale
3rd Edition
Revised & Updated
Hadoop
The Definitive Guide
O'REILLY
YAHOO! PRESS
Tom White

HDFS: *Hadoop Distributed File System*

Hadoop puede acceder a diferentes tipos de filesystems (local, HDFS, KFS, S3,. . .)

- Se recomienda HDFS: *Hadoop Distributed File System*

HDFS: Ventajas

- Diseñado para almacenar ficheros muy grandes en *commodity hardware*
- Elevado ancho de banda
- Fiabilidad mediante replicacion

HDFS: Inconvenientes

- Elevada latencia
- Poco eficiente con muchos ficheros pequeños

Conceptos de HDFS

Bloques

Por defecto 64 MB

Namenode

Mantiene la información (metadatos) de los ficheros que residen en el HDFS

Datanodes

Mantienen los datos

Secondary namenodes

Mantienen checkpoints del Namenode

Interfaz con HDFS

Tres interfaces:

1. Interfaz en línea de comandos: comando `hadoop fs`
2. Interfaz Web
3. API de programación

Interfaz en línea de comandos:

- Permite cargar, descargar y acceder a los ficheros HDFS desde línea de comandos
- Ayuda: `hadoop fs -help`

Interfaz Web:

- Puerto 50070 del namenode

Interfaz en línea de comandos HDFS

Algunos comandos

Comando	**Significado**
`hadoop fs -ls <path>`	Lista ficheros
`hadoop fs -cp <src> <dst>`	Copia ficheros HDFS a HDFS
`hadoop fs -mv <src> <dst>`	Mueve ficheros HDFS a HDFS
`hadoop fs -rm <path>`	Borra ficheros en HDFS
`hadoop fs -rmr <path>`	Borra recursivamente
`hadoop fs -cat <path>`	Muestra fichero en HDFS
`hadoop fs -mkdir <path>`	Crea directorio en HDFS
`hadoop fs -chmod ...`	Cambia permisos de fichero
`hadoop fs -chown ...`	Cambia propietario/grupo de fichero
`hadoop fs -put <local> <dst>`	Copia de local a HDFS
`hadoop fs -get <src> <local>`	Copia de HDFS a local

Hands-on

Hans-on HDFS

1. Abrir sesión (usando NX) en la máquina de trabajo (54.229.71.134)
2. Abrir un terminal y copiar los datos que usaremos a vuestro home en HDFS
   - `hadoop fs -put /opt/bdds/datos-hdp/datos .`
3. Comprobar que se han copiado con `hadoop fs -ls`
   - `hadoop fs -ls`
   - `hadoop fs -ls datos`
   - `hadoop fs -ls datos/all`
4. Abrir un navegador y conectarse al namenode, puerto 50070
   - `http://10.0.0.10:50070`
   - En *Browse filesystem* acceder a nuestro home en HDFS (`/user/usern_usuario`)
   - Comprobar que los ficheros grandes se encuentran divididos en varios bloques y cada bloque replicado 3 veces

Modelo de programación

Entrada y salida: listas de pares clave/valor

- El programador especifica las funciones map y reduce
- Función Map: genera claves/valores intermedios

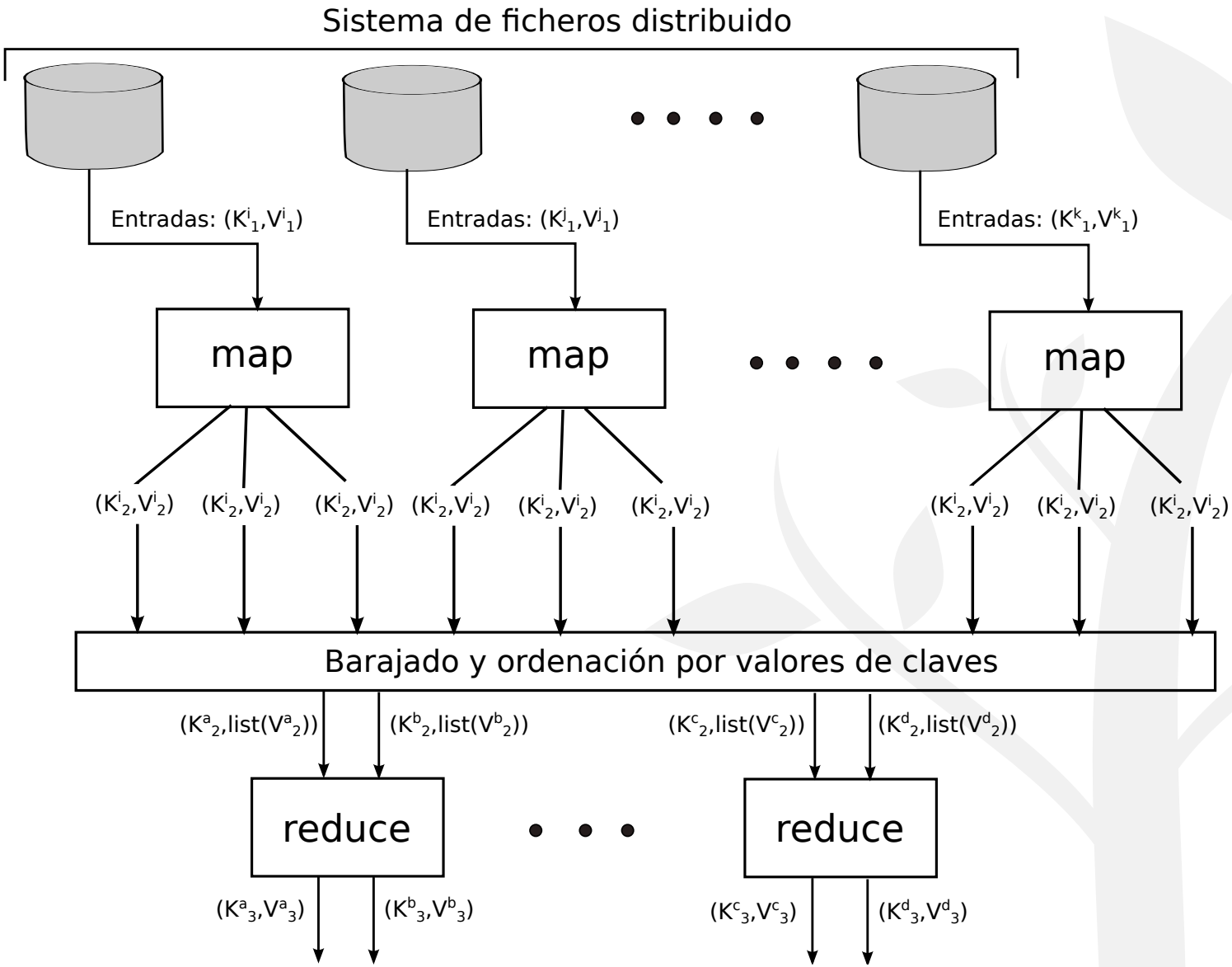
$$map(K_1, V_1) \rightarrow list(K_2, V_2)$$

  - Para cada K_1 y V_1 se obtiene una lista intermedia
  - (K_2, V_2) es un par clave/valor intermedio
- Funcion Reduce: combina los valores intermedios para cada clave particular

$$reduce(K_2, list(V_2)) \rightarrow (K_3, list(V_3))$$

  - Para cada clave de salida K_3 se genera una lista de valores
    - $list(V_3)$ suele tener un único valor
    - A menudo, $K_3 = K_2$

Automatización del proceso



Hands-on

Hands-on MapReduce

- Copiar los ficheros de ejemplo a nuestro $HOME[1]
  - `cp -r /opt/bdds/HadoopTaller/ $HOME`
- Probaremos cuatro formas de usar MapReduce
  - Programando con Java
  - Usando Hadoop Streaming (Python)
  - Languaje de alto nivel: Pig
  - Lenguage de consultas tipo SQL: Hive
- Datos de entrada
  - Fichero de citas de patentes (`cite75_99.txt`)
    - Formato: `patente,patente_a_la_que_cita`
  - Fichero de descripción de patentes (`apat63_99.txt`)

[1] Ejemplos basados en los del libro “Hadoop in Action”, C. Lam, Manning, 2011

Hands-on MapReduce Java

- Directorio `taller-hadoop_java`
- Cinco ejemplos sencillos
  1. CitingPatents: ejemplo de mapreduce simple
  2. CitationNumberByPatent_chained: encadenamiento Map | Reduce | Map
  3. CitationNumberByPatent_workflow: encadenamiento con Oozie
  4. CitationNumberByPatent_1step: ejemplo anterior en un sólo paso
  5. SimpleReduceSideJoin: ejemplo de un *inner join*
- Acceder al directorio y crear los paquetes usando Maven
  - cd $HOME/HadoopTaller/taller-hadoop_java
  - mvn package

Ejemplo 1: CitingPatents

- Objetivo: para cada patente, obtener la lista de las que la citan
  - Formato salida: `patente  patente1,patente2...`
- Mapper:
  - 3858245,3755824 $\rightarrow$ 3755824 3858245
- Reducer:
  - 3755824 {3858245 3858247...} $\rightarrow$ 3755824 3858245,3858247...
- Combiner: función de agregación local para las claves repetidas de cada map
  - Se ejecuta en el mismo nodo que el map
  - Habitualmente, misma función que el reducer
  - Es opcional (en programa debe funcionar igual con o sin Combiner)

Ejemplo 1: CitingPatents ejecución

- Acceder al directorio

```
cd $HOME/HadoopTaller/taller-hadoop_java/CitingPatents
```

- Ejecución en local (para depurado)

```
hadoop jar target/CitingPatents-0.0.1.jar -fs file:///
-jt local /opt/bdds/datos-hdp/datos/mini/cite75_99.txt
salida/CP
```

- Salida en el fichero `salida/CP/part-r-00000`
  - Un fichero `part` por cada proceso reducer
- Ejecución en el cluster

```
hadoop jar target/CitingPatents-0.0.1.jar
datos/all/cite75_99.txt salida/CP
```

  - Salida en HDFS
  - Información sobre ejecución: `http://10.0.0.10:50070`

Ejemplo 2: CitationNumberByPatent_chained

- Objetivo: para cada patente, obtener el número de las que la citan
- Encadena CitingPatents con un mapper que cuenta el número de patentes que citan
- Mapper2:
  - 3755824 3858245,3858247... $\rightarrow$ 3755824 9
- Utiliza *ChainMapper* y *ChainReducer*
  - Ejecuta una cadena de mappers en un único mapper y un reducer seguido de una cadena de mappers en un único reducer (M+RM*)
  - Sólo disponibles en la API antigua

Ejemplo 2: CitationNumberByPatent_chained ejecución

Ejecución en el cluster

- `hadoop jar target/CitationNumberByPatent_chained-0.0.1.jar datos/all/cite75_99.txt salida/CNBP_c`

Ejemplo 3: CitationNumberByPatent_workflow

- Lo mismo que el ejemplo 2, pero usando Oozie

Apache Oozie:

- Ejecución de workflows de trabajos dependientes
- Workflows compuestos de diferentes tipos de trabajos Hadoop (MapReduce, Pig, Hive, etc.)
- Permite la ejecución en un cluster de miles de workflows compuestos por decenas de trabajos y implifica la re-ejecución de workflows fallidos
- Definición del workflow escrita en XML usando HPDL (*Hadoop Process Definition Language*)

Ejemplo 3: CitationNumberByPatent_workflow ejecución

1. Copiar el .jar del CitationNumberByPatent_chained al directorio lib de CitationNumberByPatent_workflow
2. Copiar todo el directorio CitationNumberByPatent_workflow al HDFS
   - `hadoop fs -put CitationNumberByPatent_workflow .`
3. Lanzar el workflow con oozie
   - `oozie job -config CitationNumberByPatent_workflow/job.properties -run`
4. Controlar la ejecución
   - `oozie job -info` *`código_del_job`*
   - Interfaz web: `http://10.0.0.10:11000`

Ejemplo 4: CitationNumberByPatent_1step

- Objetivo: lo mismo que en el ejemplo 2, pero con un único MapReduce
- Mapper:
  - $3858245,3755824 \rightarrow 3755824 \quad 1$
- Reducer:
  - $3755824 \quad \{1\ 1 \ldots\} \rightarrow 3755824 \quad 9$
- Combiner:
  - Misma función que el reducer

Ejemplo 4: CitationNumberByPatent_1step ejecución

Ejecución en el cluster

- `hadoop jar target/CitationNumberByPatent_1step-0.0.1.jar datos/all/cite75_99.txt salida/CNBP_1s`

Ejemplo 5: SimpleReduceSideJoin

- Objetivo: unir datos de dos entradas
  - (a) Salida de CitingPatents (ejemplo 1)
  - (b) Fichero con información de las patentes (apat63_99.txt)
- Salida:
  - ▷ patente, país, n_citas
- Un mapper diferente para cada entrada
  - (a) Mapper-a (CNBPTaggedMapper)

    3755824 3858245,3858247... → 3755824 ''cite'', 9
  - (b) Mapper-b (PBCMapper)

    3755824,1973,4995,1971,''US'',''NY''... → 3755824 ''country'', US
- Reducer: hace un join de los dos mappers por patente
  - ▷ 3755824 ''cite'', 9
    3755824 ''country'', US → 3755824 US, 9

Ejemplo 5: SimpleReduceSideJoin ejecución

Ejecución en el cluster

- `hadoop jar target/SimpleReduceSideJoin-0.0.1.jar salida/CP datos/all/apat63_99.txt salida/SRSJ`

Alternativas a Java

- Ventajas de Java
  - Eficiencia
  - Flexibilidad
- Inconvenientes
  - Dificultad de programación
- Alternativas a Java
  - Hadoop Pipes: interfaz C++ a Hadoop MapReduce
  - Hadoop Streaming
    - Permite usar cualquier lenguaje que pueda leer/escribir en la entrada/salida estándard
    - Ejemplos: Python, Ruby, Perl, bash,...
    - Desarrollo rápido de códigos
    - Menor eficiencia y flexibilidad
  - Alternativas de alto nivel: Pig y Hive

Hands-on MapReduce Streaming

- Directorio
  `$HOME/HadoopTaller/taller-hadoop_python`
- Dos ejemplos sencillos escritos en Python (fichero apat63_99.txt)
  6. CountryClaims: Obtiene la media de reivindicaciones (“claims”) de las patentes por países
  7. CuentaAtributosUsandoAggregate: Suma las ocurrencias de un campo determinado

Ejemplo 6: CountryClaims

- Mapper
  - lee las líneas del fichero y produce el país (campo 4) y el nº de claims (campo 8)
- Reducer
  - lee la salida del mapper (ordenadas por países)
  - para cada país acumula los claims para obtener la media
  - debe detectar el cambio de país en los datos de entrada

Ejemplo 6: CountryClaims ejecución

Ejecución en el cluster

- ```
  hadoop jar
  /opt/hadoop/contrib/streaming/hadoop-streaming*.jar
  -input datos/all/apat63_99.txt -output salida/CC
  -mapper CountryClaimsMapper.py
  -reducer CountryClaimsReducer.py
  -file CountryClaimsMapper.py
  -file CountryClaimsReducer.py
  -numReduceTasks 1
  ```
  ```

Ejemplo 7: CuentaAtributosUsandoAggregate

- Suma las ocurrencias de un campo dado en un fichero CSV
- Mapper
  - Para el campo seleccionado emite el valor del campo (clave) y un 1 (valor)
- Reducer
  - Utiliza la opción `LongValueSum` del paquete Aggregator
- Otras opciones del paquete Aggregator son:

DoubleValueSum	suma una secuencia de doubles
(String/Long)Value(Max/Min)	obtiene el máximo/minimo
UniqValueCount	para cada clave, obtiene el número de valores únicos
ValueHistogram	para cada clave, obtiene el número de valores únicos, y el mínimo, mediana, máximo, media y desviación estándar de las ocurrencias de cada valor

Ejemplo 7: CuentaAtributosUsandoAggregate ejecución

Ejecución en el cluster

- ```
  hadoop jar
  /opt/hadoop/contrib/streaming/hadoop-streaming*.jar
  -input datos/all/apat63_99.txt -output salida/CAUA
  -mapper 'CuentaAtributosUsandoAggregate.py 1'
  -reducer aggregate
  -file CuentaAtributosUsandoAggregate.py
  -numReduceTasks 1
  ```
  ```

Pig

- Iniciado en *Yahoo! Research*
- Eleva el nivel de abstracción para procesar grandes conjuntos de datos
  - Facilita centrarse en el “qué” en vez de en el “cómo”
- Expresa secuencias de trabajos MapReduce
- Modelo de datos: “bags” anidadas de items
  - “Pig bag”: colección de tuplas
- Proporciona operadores relacionales (JOIN, GROUP BY, etc.)
- Fácil definir y usar funciones definidas por el usuario (UDFs) escritas en Java

Pig

- Dos elementos principales:
  - Un lenguaje propio: *Pig Latin*
  - Un entorno de ejecución: local o Hadoop
- Programas ejecutados como:
  - Scripts en Pig Latin
  - *Grunt*: shell interactivo para ejecutar comandos Pig
  - Comandos Pig ejecutados desde Java
- PigPen: entorno de desarrollo para Eclipse

Hands-on Pig

- Directorio `$HOME/HadoopTaller/taller-hadoop_pig`
- Dos ejemplos sencillos
  8. CitationNumberByPatent.pig: hace lo mismo que los ejemplos 2, 3 o 4
  9. SimpleJoin.pig: hace lo mismo que el ejemplo 5

Ejemplo 8: CitationNumberByPatent.pig ejecución

Ejecutarlo en el cluster en modo interactivo

- Ejecutar el comando `pig`
  - Con `pig -x local` lo ejecutaríamos en local
- En el prompt de `grunt` ir ejecutando una a una las instrucciones del script
- Cambiar $input por datos/all/cite75_99.txt y $output por salida/CNBP_pig
- Después de una asignación se puede usar el comando ILLUSTRATE para ver una muestra del resultado, por ejemplo
  - grunt> ILLUSTRATE citas
- El comando STORE lanza la ejecución del MapReduce

Ejemplo 9: SimpleJoin.pig ejecución

Ejecutarlo en el cluster en modo script

- `pig -param citas=datos/all/cite75_99.txt -param info=datos/all/apat63_99.txt -param output=salida/SJ_pig SimpleJoin.pig`

Hive

- Desarrollado en *Facebook*
- Infraestructura de *data warehouse* y lenguaje declarativo tipo SQL para el análisis de grandes datasets en filesystems compatibles con Hadoop (HDFS, S3, ...)
- *HiveQL*: dialecto de SQL
  - Peticiones HiveQL se convierten en un grafo dirigido acíclico de trabajos MapReduce
- Organiza los datos del filesystem en tablas
  - Listas de metadatos (p.e. esquemas de tablas) almacenados en una base de datos (por defecto, Apache Derby)
- Soporta particionado de tablas, clustering, tipos de datos complejos, etc.
- Puede llamar a scripts de Hadoop Streaming desde HiveQL

Hands-on hive

- Directorio `$HOME/HadoopTaller/taller-hadoop_hive`
- Un ejemplos sencillos
  10. SimpleJoin.hive: hace el mismo join que los ejemplos 5 y 10

Ejemplo 10: SimpleJoin.hive ejecución

Ejecutarlo en el cluster en modo interactivo

- Ejecutar el comando `hive`
  - Con `hive -f` *`script.hive`* ejecutaríamos el script completo
  - Con `hive -e '`*`comando_hive`*`'` se puede ejecutar una orden simple
- En el prompt de `hive` ir ejecutando una a una las instrucciones del script

Ejemplo 10: SimpleJoin.hive ejecución

Algunas características

- Usando Derby (por defecto), la metainformación (metastore) se guarda en el directorio local
  - Alternativa: usar otra base de datos como MySQL para la metastore
- La operación LOAD mueve el fichero al directorio de Hive
  - Es una simple operación de ficheros
  - Los ficheros no se parsean ni se modifican
  - Si no se quiere mover los ficheros (p.e. para seguir usando Hadoop o Pig) hay que crear la tabla EXTERNAL
  - Alternativa: usar HCatalog
    - Permite que Hadoop o Pig usen la metastore de Hive

¿Preguntas?

tf.pena@usc.es