Ingeniería para el Procesado Masivo de Datos

Dr. Pablo J. Villacorta

Tema 2. HDFS y MapReduce

Noviembre de 2025

unir LA UNIVERSIDAD EN INTERNET

Universidad Internacional de La Rioja

Objetivos del tema

- Conocer las características propias de HDFS
- Entender la arquitectura de HDFS
- Familiarizarse con los comandos más habituales de HDFS

Características y arquitectura de HDFS

Hadoop Distributed File System (HDFS)

Sistema de archivos distribuido para almacenar archivos muy grandes con patrones de acceso en streaming, pensado para clusters de ordenadores convencionales

- Sistema de archivos distribuido: almacenamiento en una red de máquinas.
  - Máquinas *commodity* (hardware convencional, que puede fallar).
  - Escalable (más capacidad: añadir más nodos, en lugar de nodos más grandes)
  - Incorpora mecanismos software de recuperación frente a fallo de un nodo
- Permite archivos mayores que la capacidad del disco de una máquina individual
  - Cientos de GB, varios TB, incluso PB
- Archivos con patrón de acceso write-once, read-many.
  - No es importante el tiempo de acceso a partes individuales sino que se suele utilizar el archivo completo en las aplicaciones (modo batch, no interactivo)
  - No soporta modificación de archivos existentes. Sólo lectura, escritura y borrado
- No funciona bien para:
  - Aplicaciones que requieran baja latencia para acceder a registros individuales
  - Muchos archivos pequeños (generan demasiados metadatos)
  - Archivos que se modifiquen con frecuencia

Recurso con la arquitectura detallada de HDFS: http://www.aosabook.org/en/hdfs.html

Hadoop Distributed File System (HDFS)

- Está escrito en lenguaje Java
- Se instala *encima* de los sistemas de ficheros nativos de máquinas Linux (ej: ext4)
  - Utiliza la API del Sistema Operativo para leer y escribir datos en los nodos.
  - No interactúa con los dispositivos físicos (discos duros)

- Su antecesor fue Google File System (GFS) en 2003
  - Ghemawat, S.; Gobioff, H.; Leung, S. T. (2003). "The Google File System". *Proceedings of the 19th ACM Symposium on Operating Systems Principles - SOSP '03, 29 – 43.*

- Un recurso **excelente** sobre la arquitectura de HDFS: http://www.aosabook.org/en/hdfs.html

Bloques en HDFS

- Bloque físico (sector) de disco: cantidad de información que se puede leer o escribir en una sola operación de disco. Habitualmente 512 bytes.
- Bloque de sistema de archivos: conjunto de sectores que se pueden reservar para leer o escribir un archivo. Suele ser configurable (ej: Linux ext4 permite 1KB, 2KB …). Habitualmente 4 KB. Debe contener un número de sectores físicos potencia de 2.
- En sistemas de archivos convencionales (ext4, NTFS, FAT32…) los archivos menores que el bloque de sistema de archivos siguen ocupando un bloque completo.
- HDFS tiene su propio tamaño de bloque, configurable (por defecto 128 MB), pero los archivos de menos de un bloque no desperdician espacio (aunque siempre usan su propio bloque de datos; nunca se comparte un bloque de HDFS entre varios archivos)
- Implicaciones del tamaño de bloque de HDFS:
  - Un archivo se parte en bloques que pueden almacenarse en máquinas diferentes. Así se puede almacenar un archivo mayor que el disco de una sola máquina
  - Cada bloque requiere metadatos (que se almacenan en el *namenode*) para mantenerlo localizado. Bloques pequeños: demasiados metadatos. Bloques muy grandes: limitan el paralelismo de frameworks que operan a nivel de bloque
  - Bloques de datos replicados para alta disponibilidad y máximo paralelismo: cada bloque está en *k* máquinas (*k* : factor de replicación, por defecto 3, configurable individualmente para cada fichero: hadoop dfs -setrep -w 3 /user/hdfs/file.txt)

Namenode y datanodes

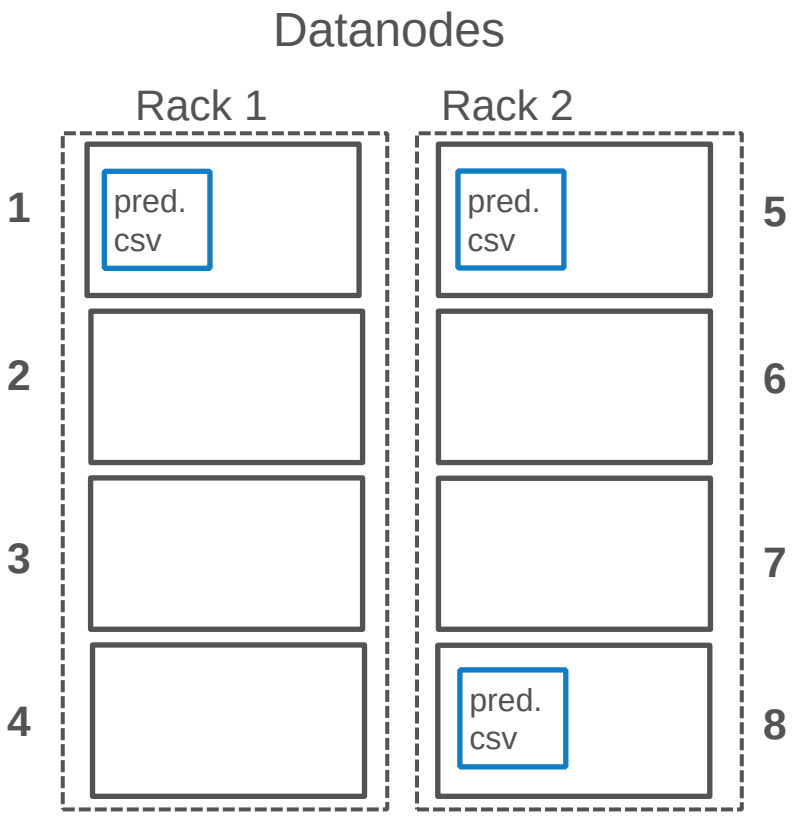
- Dos tipos de nodos diferentes: namenode (al menos uno) y datanodes
  - Namenode: almacena metadatos (estructura de directorios, ubicación bloques).
  - Datanodes: almacenan y sirven los bloques de datos que componen los ficheros
- Ejemplo 1: archivo flights1994.csv (500 MB) que se ha subido mediante copyFromLocal

```
[pvillacorta@meetupds1 ~]$ hdfs fsck /SparkMeetup/flights1994.csv -blocks -files
Connecting to namenode via http://meetupds1:50070/fsck?ugi=pvillacorta&blocks=1&files=1&path=%2FSparkMeetup%2Fflights1994.csv
FSCK started by pvillacorta (auth:SIMPLE) from                for path /SparkMeetup/flights1994.csv at Thu Apr 18 21:43:27 CEST 2019
/SparkMeetup/flights1994.csv 501558665 bytes, 4 block(s):  OK
0. BP-2126204769-           -1526901840376:blk_1073977383_236559 len=134217728 repl=3
1. BP-2126204769-           -1526901840376:blk_1073977384_236560 len=134217728 repl=3
2. BP-2126204769-           -1526901840376:blk_1073977385_236561 len=134217728 repl=3
3. BP-2126204769-           -1526901840376:blk_1073977386_236562 len=98905481 repl=3

Status: HEALTHY
 Total size:    501558665 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      4 (avg. block size 125389666 B)
 Minimally replicated blocks:   4 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
FSCK ended at Thu Apr 18 21:43:27 CEST 2019 in 1 milliseconds


The filesystem under path '/SparkMeetup/flights1994.csv' is HEALTHY
```

HDFS lo ha dividido automáticamente en 4 bloques, el último más pequeño. Cada bloque está replicado 3 veces

Namenode y datanodes

- Ejemplo 2: archivo *pred.csv* que ocupa menos de un bloque (ej: 40 MB)



Namenode

/misdatos/pred.csv (40 MB) nodos 1,5,8
…
…
…

- Rack - awareness: distribución de bloques optimizada para la topología concreta del cluster.
- Minimiza pérdidas si cae (falla) un rack completo de nodos y minimiza tráfico de red.
- *No más de una réplica en un mismo nodo, y no más de dos réplicas en el mismo rack*

Namenode y datanodes

- El namenode mantiene la estructura de directorios y los metadatos. Esta información se guarda de manera persistente como (i) imagen del namespace, y (ii) log de edición
- Los datanodes almacenan bloques de datos. Los devuelven a petición del namenode o del programa cliente que está accediendo a HDFS.
- El namenode es punto único de fallo (SPOF). Sin él, no es posible utilizar HDFS.
- Alta disponibilidad de HDFS: par de namenodes (activo / stand-by). Log de edición compartido. Los datanodes reportan a ambos. Requiere re-implementar los clientes.
- Escalando el namenode (memoria debido a metadatos): **federación** de namenodes
  - Varios namenodes se encargan de directorios distintos del sistema de archivos (sin solapamiento, ej.: /user vs /share). El fallo de uno no afecta al resto.
  - Los datanodes sí pueden almacenar bloques de archivos de varios namespaces

Configuración de HDFS

- Los ficheros de configuración se encuentran en etc/hadoop.

Filename	Format	Description
hadoop-env.sh	Bash script	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Hadoop configuration XML	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and MapReduce.
hdfs-site.xml	Hadoop configuration XML	Configuration settings for HDFS daemons: the namenode, the secondary namenode, and the datanodes.
mapred-site.xml	Hadoop configuration XML	Configuration settings for MapReduce daemons: the jobtracker, and the tasktrackers.
masters	Plain text	A list of machines (one per line) that each run a secondary namenode.
slaves	Plain text	A list of machines (one per line) that each run a datanode and a tasktracker.
hadoop-metrics.properties	Java Properties	Properties for controlling how metrics are published in Hadoop
log4j.properties	Java Properties	Properties for system logfiles, the namenode audit log, and the task log for the tasktracker child process

Configuración de HDFS

File Name	Parameter Name	Parameter value	Description
core-site.xml	fs.defaultFS/fs.default.name	hdfs://<namenode_ip>:8020	Namenode ip address or nameservice (HA config)
hdfs-site.xml	dfs.block.size, dfs.blocksize	128 MB	Block size at which files will be stored physically.
hdfs-site.xml	dfs.replication	3	Number of copies per block of a file for fault tolerance
hdfs-site.xml	dfs.namenode.http-address	0.0.0.0:50070	Namenode Web UI. By default it might use ip address of namenode.
hdfs-site.xml	dfs.datanode.http.address	0.0.0.0:50075	Datanode Web UI
hdfs-site.xml	dfs.name.dir, dfs.namenode.name.dir	<directory_location>	Directory location for FS Image and edit logs on name node
hdfs-site.xml	dfs.data.dir, dfs.datanode.data.dir	<directory_location>	Directory location for storing blocks on data nodes
hdfs-site.xml	fs.checkpoint.dir, dfs.namenode.checkpoint.dir	<directory_location>	Directory location which will be used by secondary namenode for checkpoint.
hdfs-site.xml	fs.checkpoint.period, dfs.namenode.checkpoint.period	1 hour	Checkpoint (merging edit logs with current fs image to create new fs image) interval.
hdfs-site.xml	dfs.namenode.checkpoint.txns	1000000	Checkpoint (merging edit logs with current fs image to create new fs image) transactions.

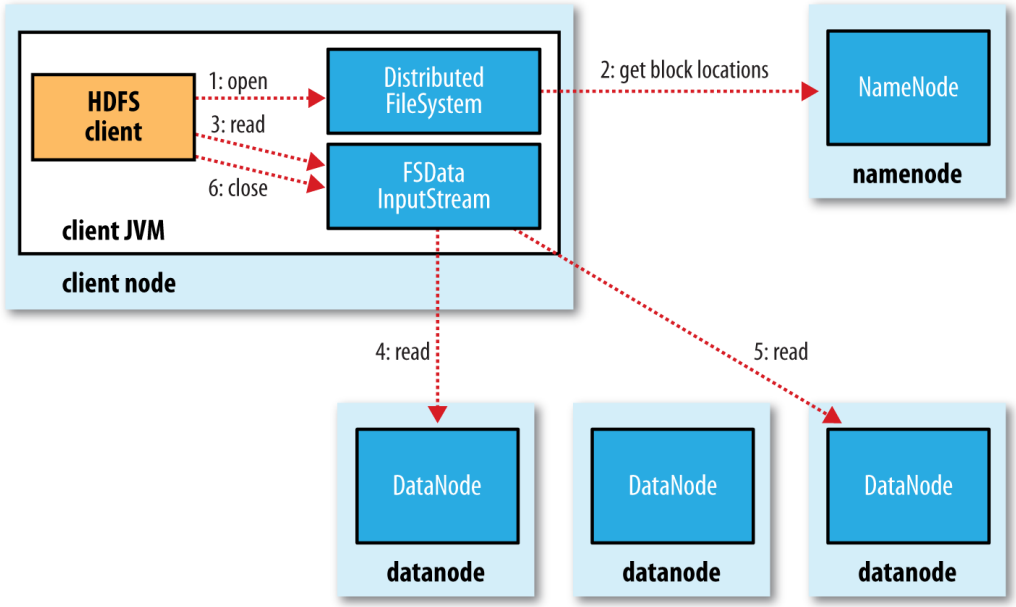
Fuente: https://slideplayer.com/slide/12131503/

Configuración de HDFS

```
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>dfs.replication</name>
   <value>1</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>
```

Proceso de lectura de HDFS

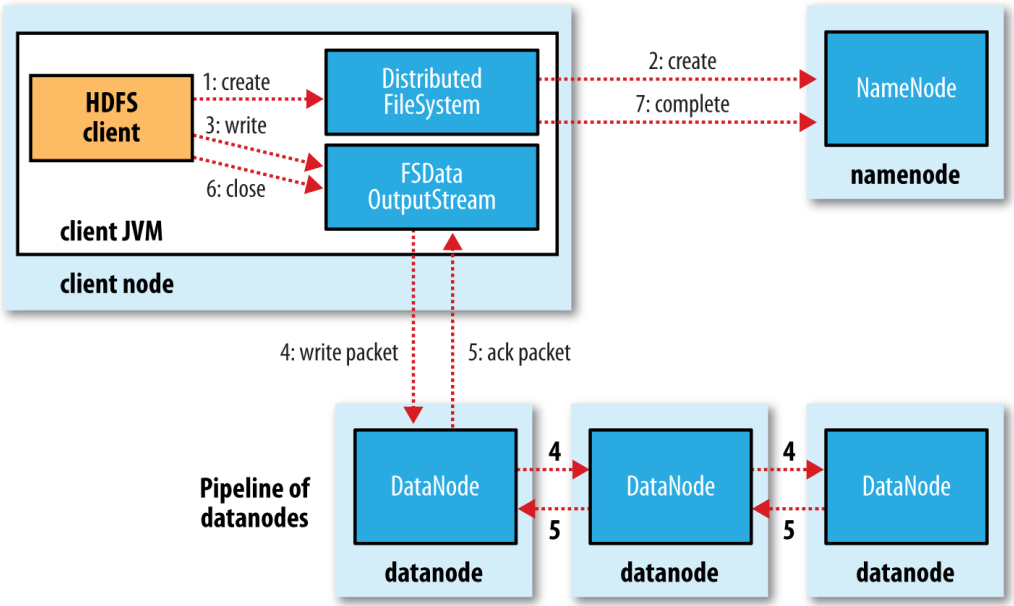
- El comando hadoop fs (FileSystem Shell) es un cliente de HDFS escrito en Java, que parsea los comandos y usa la API de Java para interactuar con HDFS



El namenode **no** devuelve los datos (sería cuello de botella para múltiples clientes concurrentes)

- El cliente invoca open() sobre el objeto DistributedFileSystem, que contacta con el namenode vía RPC (remote procedure call). Devuelve un objeto FSDataInputStream que gestiona las lecturas (conoce la ubicación de los bloques de los que ir leyendo para el archivo solicitado)
- El cliente va haciendo lecturas sobre el objeto FSDataInputStream y lo percibe como un flujo continuado. Dicho objeto contacta con los datanodes necesarios para cada bloque, que devuelven al cliente los datos, y **ocasionalmente** con el namenode para obtener la (mejor) ubicación de los siguientes bloques según se van necesitando. Todo de manera transparente al cliente.

Proceso de escritura en HDFS



Los propios datanodes se envían los datos para ir creando las réplicas

- El cliente llama a create() que pide al namenode la creación mediante RPC, sin bloques asociados
- Se comprueba que no exista ya y el cliente posea permisos, y devuelve un FSDataOutputStream que gestiona la escritura. Este objeto divide los datos en paquetes, los encola y los va escribiendo, y contacta de nuevo con el namenode para obtener los datanodes donde escribir más bloques.
- Para cada bloque, los datanodes (varios, debido al factor de replicación) que lo guardarán forman un pipeline. El primer datanode escribe el bloque y lo reenvía al segundo, y así sucesivamente. Según se van escribiendo, se devuelve una señal de confirmación *ack* que también se almacenan en una cola. En caso de fallo de algún datanode durante la escritura, se establecen mecanismos para que el resto de datanodes del pipeline no pierdan ese paquete y asegurar la replicación.

Comandos HDFS más habituales

Listar comandos existentes

```
[root@sandbox ~]# hadoop fs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
```

Mostrar archivos

```
[root@sandbox ~]# hadoop fs -ls /
Found 12 items
drwxrwxrwx   - yarn   hadoop          0 2016-10-25 08:10 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:54 /apps
drwxr-xr-x   - yarn   hadoop          0 2016-10-25 07:48 /ats
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 08:01 /demo
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:48 /hdp
drwxr-xr-x   - mapred hdfs            0 2016-10-25 07:48 /mapred
drwxrwxrwx   - mapred hadoop          0 2016-10-25 07:48 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:47 /ranger
drwxrwxrwx   - spark  hadoop          0 2017-02-02 11:46 /spark-history
drwxrwxrwx   - spark  hadoop          0 2016-10-25 08:14 /spark2-history
drwxrwxrwx   - hdfs   hdfs            0 2016-10-25 08:11 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 08:11 /user
[root@sandbox ~]#
```

Crear directorios

```
[root@sandbox ~]# hadoop fs -mkdir /raul
[root@sandbox ~]# hadoop fs -ls /
Found 13 items
drwxrwxrwx   - yarn   hadoop          0 2016-10-25 08:10 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:54 /apps
drwxr-xr-x   - yarn   hadoop          0 2016-10-25 07:48 /ats
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 08:01 /demo
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:48 /hdp
drwxr-xr-x   - mapred hdfs            0 2016-10-25 07:48 /mapred
drwxrwxrwx   - mapred hadoop          0 2016-10-25 07:48 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 07:47 /ranger
drwxr-xr-x   - root   hdfs            0 2017-02-02 11:48 /raul
drwxrwxrwx   - spark  hadoop          0 2017-02-02 11:48 /spark-history
drwxrwxrwx   - spark  hadoop          0 2016-10-25 08:14 /spark2-history
drwxrwxrwx   - hdfs   hdfs            0 2016-10-25 08:11 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2016-10-25 08:11 /user
```

Copiar desde el disco local a HDFS y viceversa

```
[root@sandbox raul]# ls -la
total 12
drwxr-xr-x 2 root root 4096 Feb  2 11:50 .
dr-xr-x--- 1 root root 4096 Feb  2 11:49 ..
-rw-r--r-- 1 root root   16 Feb  2 11:50 MyBigFile.txt
[root@sandbox raul]# hadoop fs -copyFromLocal MyBigFile.txt /raul/MyBigFileinHDFS.txt
[root@sandbox raul]# hadoop fs -ls /raul/
Found 1 items
-rw-r--r--   1 root hdfs         16 2017-02-02 11:51 /raul/MyBigFileinHDFS.txt
[root@sandbox raul]#
```

hadoop fs –copyFromLocal *<localsrc> <dst>*

hadoop fs –copyToLocal *<src> <localdst>*

```
[root@sandbox raul]# hadoop fs -ls /demo/data/CDR/
Found 2 items
-rwx------   1 hdfs hdfs     710436 2016-10-25 08:01 /demo/data/CDR/cdrs.txt
-rwx------   1 hdfs hdfs      68095 2016-10-25 08:01 /demo/data/CDR/recharges.txt
[root@sandbox raul]# hadoop fs -copyToLocal /demo/data/CDR/cdrs.txt myCdrs.txt
[root@sandbox raul]# ls -la
total 708
drwxr-xr-x 2 root root   4096 Feb  2 11:55 .
dr-xr-x--- 1 root root   4096 Feb  2 11:49 ..
-rw-r--r-- 1 root root     16 Feb  2 11:50 MyBigFile.txt
-rw-r--r-- 1 root root 710436 Feb  2 11:55 myCdrs.txt
[root@sandbox raul]#
```

Inspeccionando ficheros

```
[root@sandbox raul]# hadoop fs -tail /demo/data/CDR/recharges.txt
00
6641609561|20130209|094637|3|100
6650359180|20130209|125420|3|100
6638378345|20130209|121231|3|300
6659538250|20130209|191504|3|100
6662032971|20130209|211136|3|500
8333654388|20130209|100458|3|100
6623568405|20130209|121240|3|100
```

```
[root@sandbox raul]# hadoop fs -cat /demo/data/CDR/recharges.txt | more
PHONE|DATE|CHANNEL|PLAN|AMOUNT
7852121521|20130209|090721|3|100
7642140929|20130209|181648|3|100
7552204414|20130209|224815|3|100
7785846460|20130209|173731|3|100
7972930496|20130209|003527|3|100
7782957598|20130209|200016|3|100
7352795440|20130209|000429|3|100
```

Copiar, mover y borrar

```
[root@sandbox raul]# hadoop fs -ls /demo/data/CDR/
Found 2 items
-rwx------   1 hdfs hdfs     710436 2016-10-25 08:01 /demo/data/CDR/cdrs.txt
-rwx------   1 hdfs hdfs      68095 2016-10-25 08:01 /demo/data/CDR/recharges.txt
[root@sandbox raul]# hadoop fs -cp /demo/data/CDR/cdrs.txt /demo/data/CDR/cdrs2.txt
[root@sandbox raul]# hadoop fs -ls /demo/data/CDR/
Found 3 items
-rwx------   1 hdfs hdfs     710436 2016-10-25 08:01 /demo/data/CDR/cdrs.txt
-rw-r--r--   1 root hdfs     710436 2017-02-02 12:01 /demo/data/CDR/cdrs2.txt
-rwx------   1 hdfs hdfs      68095 2016-10-25 08:01 /demo/data/CDR/recharges.txt
[root@sandbox raul]# hadoop fs -rm /demo/data/CDR/cdrs2.txt
17/02/02 12:01:52 INFO fs.TrashPolicyDefault: Moved: 'hdfs://sandbox.hortonworks.com:80
/user/root/.Trash/Current/demo/data/CDR/cdrs2.txt
[root@sandbox raul]# hadoop fs -ls /demo/data/CDR/
Found 2 items
-rwx------   1 hdfs hdfs     710436 2016-10-25 08:01 /demo/data/CDR/cdrs.txt
-rwx------   1 hdfs hdfs      68095 2016-10-25 08:01 /demo/data/CDR/recharges.txt
[root@sandbox raul]#
```

```
hadoop fs -cp <src> <dst>
hadoop fs -mv <src> <dst>
hadoop fs -rm [-r] <path> ...
```

Otros comandos

```
[root@sandbox raul]# hadoop fs -help | more
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
```

Otros comandos

```
-chown [-R] [OWNER][:[GROUP]] PATH... :
  Changes owner and group of a file. This is similar to the shell's chown command
  with a few exceptions.

  -R  modifies the files recursively. This is the only option currently
      supported.

  If only the owner or group is specified, then only the owner or group is
  modified. The owner and group names may only consist of digits, alphabet, and
  any of [-_./@a-zA-Z0-9]. The names are case sensitive.

  WARNING: Avoid using '.' to separate user name and group though Linux allows it.
  If user names have dots in them and you are using local file system, you might
  see surprising results since the shell command 'chown' is used for local files.

-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst> :
  Identical to the -put command.

-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst> :
  Identical to the -get command.

-count [-q] [-h] [-v] [-t [<storage type>]] <path> ... :
  Count the number of directories, files and bytes under the paths
  that match the specified file pattern.  The output columns are:
  DIR COUNT FILE COUNT CONTENT SIZE PATHNAME
```

Otros comandos

```
[root@sandbox raul]# hadoop fs -count /demo/data/CDR/
           1            2             778531 /demo/data/CDR
[root@sandbox raul]# hadoop fs -count /demo/data/
           8           10             881591 /demo/data
[root@sandbox raul]# hadoop fs -count /demo/
           9           10             881591 /demo
[root@sandbox raul]#
```

```
-test -[defsz] <path> :
  Answer various questions about <path>, with result via exit status.
    -d  return 0 if <path> is a directory.
    -e  return 0 if <path> exists.
    -f  return 0 if <path> is a file.
    -s  return 0 if file <path> is greater          than zero bytes in size.
    -w  return 0 if file <path> exists          and write permission is granted.
    -r  return 0 if file <path> exists          and read permission is granted.
    -z  return 0 if file <path> is          zero bytes in size, else return 1.
```

Formatos de archivo frecuentes

Archivos Parquet

- Apache Parquet (2013): formato columnar, comprimido (binario). Utiliza record-shredding para almacenar tipos de datos complejos y anidados (un CSV no es capaz)
- Almacena el esquema junto a los datos. También máx/mín por columna (resúmenes)
- Los valores de cada columna se guardan físicamente juntos (almacenamiento columnar). Beneficios:
  - Compresión de datos por cada columna para ahorrar espacio
  - Se pueden aplicar técnicas de compresión diferentes, específicas según el tipo de dato que almacena cada columna
  - Las consultas con filtrados sobre columnas concretas no necesitan leer toda la fila, mejorando el rendimiento
- Preparado para añadir más esquemas de codificación que se desarrollen en el futuro
- No requiere especificar separador de columnas, ni si se incluyen o no nombres de columna (siempre se incluyen por defecto)
- Es con diferencia el formato más ampliamente utilizado para procesamiento en batch, tanto para datos de entrada como para guardar resultados, con datos estructurados.

Archivos ORC

- ORC: Optimized Row Columnar
- Similar a Parquet: binario, columnar, con compresión y con índices, además de almacenar estadísticas simples sobre los valores por grupos de filas
- Almacena el esquema junto a los datos.
- Como HDFS está pensado para accesos donde se escribe una sola vez (inmutables) y se lee muchas veces, los propios formatos Parquet y ORC no soportan modificaciones.
- Parquet y ORC optimizan las lecturas a costa de escrituras más costosas en tiempo

Archivos Avro

- Formato binario orientado a filas, con compresión. Típico para serialización
- El esquema se almacena en un fichero separado: AVRO schema file (.avsc)
- Finalidad principal: datos cuyo esquema evoluciona

El paradigma MapReduce

MapReduce

Modelo abstracto de programación, e implementación del modelo (bibliotecas) de procesamiento paralelo y distribuido de grandes datasets con la técnica divide y vencerás

- Desarrollado inicialmente por Google en el año 2003. Publicado en:

  MapReduce: Simplified Data Processing on Large Clusters
  OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137 – 150

- Gran ventaja: nos abstrae de todos los detalles de hardware, redes, comunicaciones para centrarnos solamente en el desarrollo de software. Además, código abierto

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

MapReduce

- Usaremos las mismas diapositivas que los autores en el congreso de 2004 cuando dieron a conocer MapReduce al mundo
https://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0002.html

Motivation: Large Scale Data Processing

Many tasks: Process lots of data to produce other data

Want to use hundreds or thousands of CPUs

- ... but this needs to be easy

MapReduce provides:

- Automatic parallelization and distribution
- Fault-tolerance
- I/O scheduling
- Status and monitoring

MapReduce

- Punto de partida: archivos muy grandes almacenados (por bloques) en HDFS
- ¿Podríamos aprovechar las CPUs de los datanodes del cluster para procesar en paralelo (simultáneamente) los bloques de un archivo?
  - No queremos mover datos (el tráfico de red es muy lento): **llevamos el cómputo a los datos** (en lugar de hacerlo al revés, como se hacía antes).
  - Analogía con la Transformación Digital: el enfoque *data-centric*. El dato debe ser el core de la empresa, y todas las aplicaciones tienen que trabajar para los datos: beber de una misma fuente de datos (única), usar un único almacén, de manera que toda la empresa tenga una única visión (lógica) del dato. Prohibido moverlos y replicarlos; evitar silos departamentales
  - Las decisiones de la empresa deben guiarse por los datos.
- El usuario **sólo necesita escribir dos funciones** (enfoque *divide y vencerás*):
  - Mapper: función del usuario llamada por el framework en paralelo sobre cada bloque de datos de entrada (configurable, generalmente coinciden con un bloque de HDFS) y que genera resultados intermedios en forma de (clave, valor)
  - Reducer: función del usuario llamada por el framework en paralelo para cada clave distinta, pasándole todos los valores que comparten esa clave

MapReduce

https://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0003.html

Programming model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

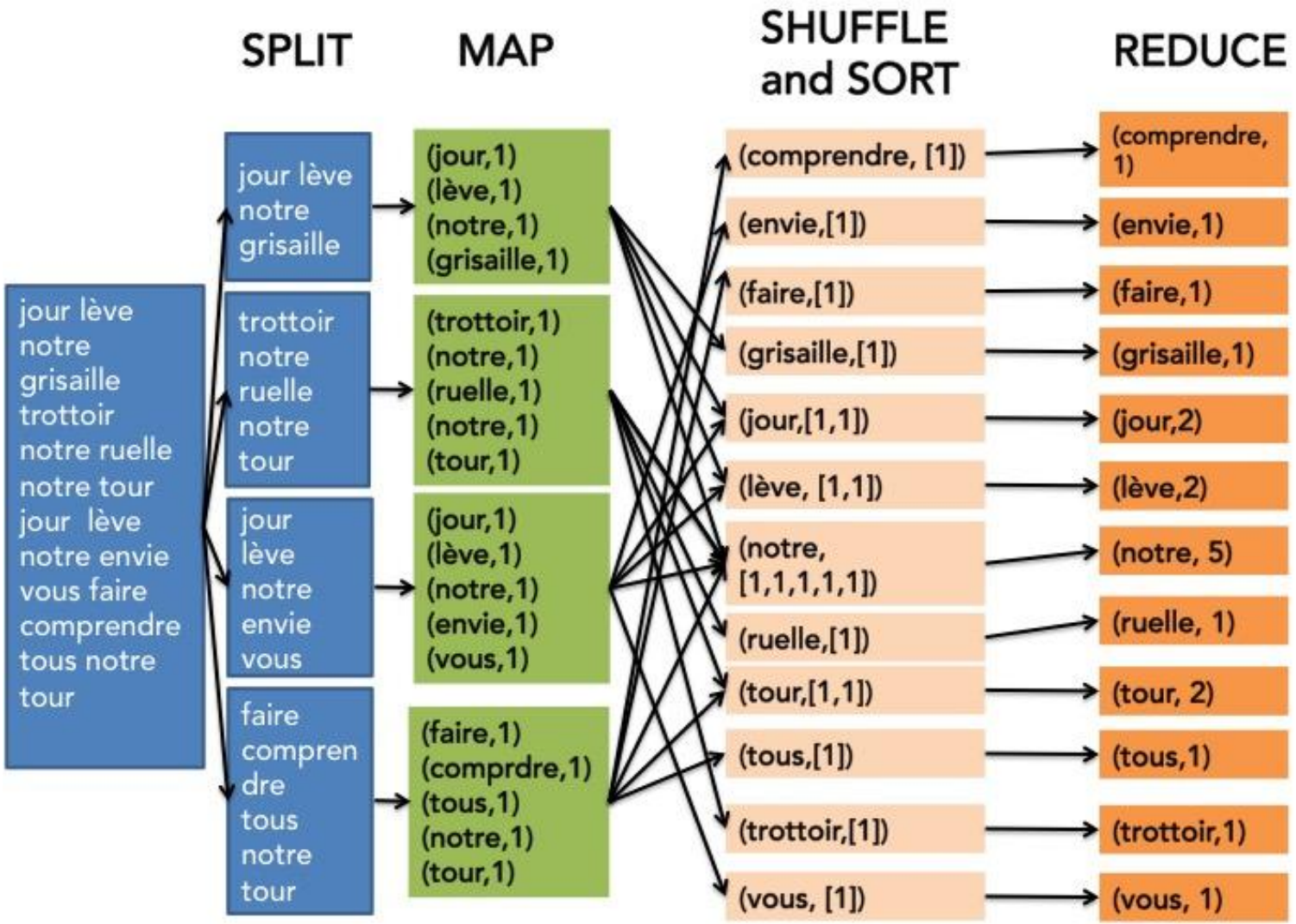
- Processes input key/value pair
- Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

Inspired by similar primitives in LISP and other languages

MapReduce: contar ocurrencias de cada palabra



MapReduce para contar palabras

- Queremos contar cuántas veces aparece cada palabra de un documento
- La función *map* recibe una tupla (*clave_entrada*, *valor*), siendo *clave_entrada* el número de línea (ignorado) y *valor* una línea completa de texto
- La función *map* del usuario puede:
  - Opción (a): dividir la línea en palabras y para cada palabra procesada, devolver la tupla (*palabra*, 1) indicando que *palabra* ha aparecido una vez. Ejemplo: («hola», 1), («el», 1), («el», 1). La palabra es la *out_key*.
  - Opción (b): dividir la línea en palabras y devolver una tupla con la palabra y el número de veces que aparece en la línea. Ej: («hola», 1), («el», 2)
  - La opción (a) genera mucho tráfico de red al generar muchas tuplas.
  - La opción (b) requiere más memoria
- Función *reduce*: es invocada con una clave y la lista de valores asociados en las tuplas. Agrega resultados (suma ocurrencias de una misma palabra).
  - Ejemplo desde mappers enfoque (a):

    (hola, [1, 1, 1, 1, 1, 1, 1]) → 7

  - Ejemplo desde mappers enfoque (b):

    (hola, [1, 3, 2, 1]) → 7

…eso es todo por hoy…

:-)

Cualquier duda, consulta o comentario,

mensaje a través de la plataforma!

www.unir.net