Tema 4 - Sistemas operativos distribuidos

Sistemas de ficheros

Felipe Ortega, Enrique Soriano

GSyC, ETSIT. URJC.

Sistemas Distribuidos (SD)

26 de octubre, 2020

Contenidos

4.1 RAID

4.2 Sistemas de ficheros distribuidos

4.3 Otros sistemas de ficheros en red

4.4 Arquitectura Apache Hadoop

Referencias

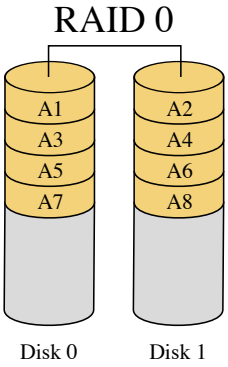
4.1 RAID

Discos: RAID

- RAID: Redundant Array of Inexpensive Disks.
- Es un caso de *virtualización de almacenamiento*.
- Pueden estar implementados por HW o por el SO.

Discos: RAID 0

- *Striping (interleaving)* por bloques.
- Sin redundancia.
- Bloques contiguos se escriben en paralelo.



Discos: RAID 1

- *Mirror*.
- Distintas lecturas en paralelo.

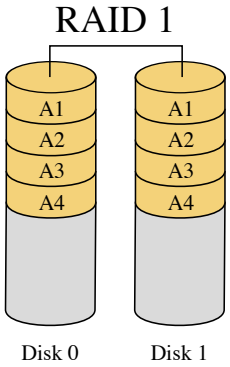
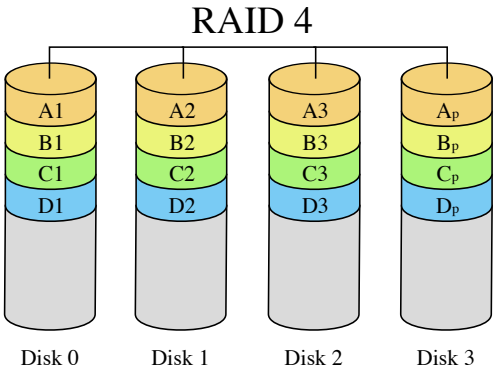


Imagen: (cc) from Wikipedia

Discos: RAID 4

- *Striping (interleaving)* por bloques.
- Dedica un disco entero para la paridad.
- Se soporta el fallo de un disco.



Discos: RAID 5

- *Striping (interleaving)* por bloques.
- La paridad está distribuida por los discos. Se soporta el fallo de un disco.
- Soluciona el cuello de botella de RAID 4.
- Problema: después de un fallo de un disco, hay probabilidad de un fallo en **cualquiera de los otros discos** mientras se reconstruye el RAID $\rightarrow$ pierdes todos los datos.

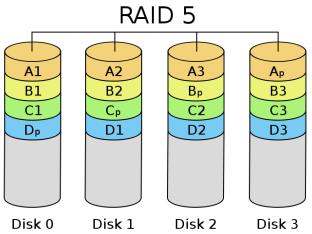


Imagen: (cc) Wikipedia

Discos: RAID 1+0

- Ejemplo de mezcla de tipos de RAID (hay muchas).
- Idea: mezclar RAID 0 (striping) con RAID 1 (mirroring).
- Compromiso redundancia y rapidez.

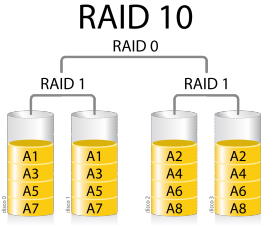


Imagen: (cc) Wikipedia

SAN

- SAN (Storage Area Network): proporciona almacenamiento de bloques a través de la red.
- Es un *disco en red*, no es un sistema de ficheros en red.
- Ejemplos: iSCSI, ATA over Ethernet (AoE), HyperSCSI.

4.2 Sistemas de ficheros distribuidos

Sistemas de ficheros en red

- Permite compartir ficheros a través de la red siguiendo un esquema cliente/servidor.
- El cliente y el servidor deben hablar un protocolo de sistema de ficheros.
- Protocolos: NFS, CIFS/SMB, WebDAV, 9P, AFP, ...
- NAS (Network-Attached Storage): HW especializado en servir un FS en red mediante estos protocolos.

Sistemas de ficheros en red

- *Stateless*
  - Sencillo.
  - Operaciones autocontenidas.
  - Los *handles* que da el servidor tienen que ser únicos (eg. fs id + inodo + versión).
  - No requiere recuperación tras un fallo.
  - El servidor se puede reemplazar.
- *Stateful*
  - Sesiones / estado permanente.
  - El servidor puede olvidar los *handles* después de una sesión.
  - Fallo del servidor $\rightarrow$ recuperar del estado o abortar.
- ¿Cache coherente? ¿*Locking/ Leases*? ¿*Callbacks*? ¿*Read-ahead*? ¿*DMEXCL*?

Semánticas de compartición de ficheros

Las más comunes:

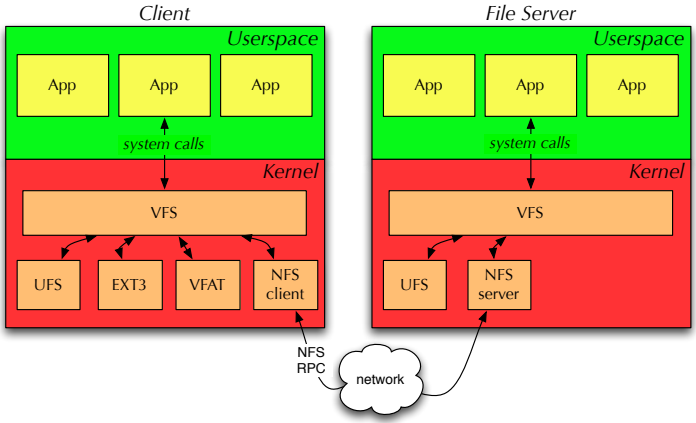
- Semántica UNIX (Strong Consistency): cualquier cambio es automáticamente visible al resto.
- Coherencia débil (Weak Consistency): existe una ventana en la que el fichero es incoherente, a largo plazo todos tienen la misma visión del fichero.
- Semántica de sesión: los cambios se hacen visibles al resto cuando se cierra la sesión.
- Ficheros inmutables: un cambio en un fichero implica una nueva versión del mismo (copy-on-write).
- Transacciones: las operaciones se empaquetan en transacciones (ACID).

NFS

- Varias versiones diferentes.
- Hasta NFSv3 era *stateless*:
  - *File handles* opacos, elegidos por el servidor.
  - No hay `open` ni `close`.
  - Problemas con locking, coherencia de cache, etc.
- NFSv4: *stateful*, RPCs compuestas, etc.
- Muy complejo:

```
ACCESS, BACKCHANNEL_CTL, BIND_CONN_TO_SESSION, CLOSE, COMMIT, CREATE, CREATE_SESSION, DELEGPURGE, DELEGRETURN, DESTROY_CLIENTID,
DESTROY_SESSION, EXCHANGE_ID, FREE_STATEID, GETATTR, GETDEVICEINFO, GETDEVICELIST, GETFH, GET_DIR_DELEGATION, LAYOUTCOMMIT, LAYOUTGET,
LAYOUTRETURN, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN_CONFIRM, OPEN_DOWNGRADE, PUTFH, PUTPUBFH, PUTROOTFH,
READ, READDIR, READLINK, RECLAIM_COMPLETE, RELEASE_LOCKOWNER, REMOVE, RENAME, RENEW, RESTOREFH, SAVEFH, SECINFO, SECINFO_NO_NAME, SEQUENCE,
SETATTR, SET_CLIENTID, SET_CLIENTID_CONFIRM, SET_SSV, TEST_STATEID, VERIFY, WANT_DELEGATION, WRITE...
```

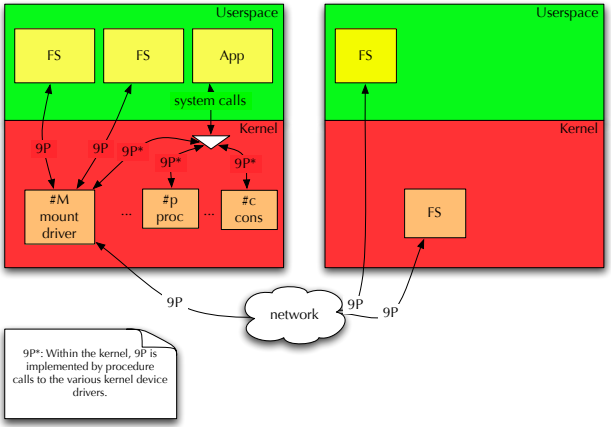
UNIX + NFS


Client
Userspace
App
App
App
system calls
Kernel
VFS
UFS
EXT3
VFAT
NFS client
File Server
Userspace
App
App
App
system calls
Kernel
VFS
UFS
NFS server
NFS RPC
network

9P

- Stateful: el servidor mantiene el estado de la conexión.
- Stateful: posibilita la exportación de ficheros sintéticos.
- Sencillo: sólo 13 RPCs.
- Totalmente integrado con el SO: el kernel de Plan 9 es un multiplexor de 9P.

9P


Userspace
FS
FS
App
system calls
Kernel
9P
9P
9P*
9P*
9P*
#M
mount
driver
#p
proc
#c
cons
Userspace
FS
Kernel
FS
9P
network
9P
9P
9P*: Within the kernel, 9P is implemented by procedure calls to the various kernel device drivers.

9P: RPCs

- `size[4] Tversion tag[2] msize[4] version[s]`
  `size[4] Rversion tag[2] msize[4] version[s]`
  Acuerda la versión del protocolo y comienza una nueva sesión.

- `size[4] Tauth tag[2] afid[4] uname[s] aname[s]`
  `size[4] Rauth tag[2] aqid[13]`
  Autenticación (externa a través de P9SK1).

- `size[4] Rerror tag[2] ename[s]`
  Error para todos los tipos de peticiones.

- `size[4] Tflush tag[2] oldtag[2]`
  `size[4] Rflush tag[2]`
  Aborta la petición identificada con `oldtag`.

9P: RPCs

- `size[4] Tattach tag[2] fid[4] afid[4] uname[s] aname[s]`
  `size[4] Rattach tag[2] qid[13]`

  El cliente se ata al raíz del sistema de ficheros. A partir de este momento se referirá a él con el FID. El servidor responde con un identificador único para el raíz QID (versión, tipo y path).

- `size[4] Twalk tag[2] fid[4] newfid[4] nwname[2] nwname*(wname[s])`
  `size[4] Rwalk tag[2] nwqid[2] nwqid*(wqid[13])`

  Resuelve un path, que será referido con el FID `newfid`. Si no se pueden atravesar todos los nombres, `newfid` será `fid`. El servidor devuelve la lista de QIDs de los nombres que se han podido atravesar.

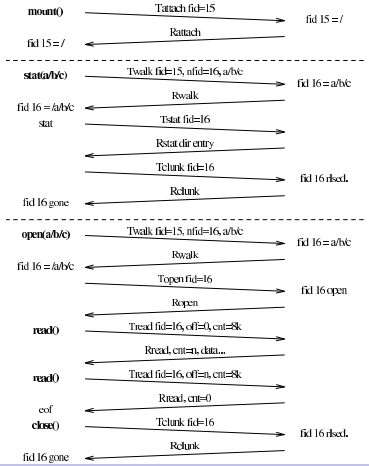
9P: RPCs

- `size[4] Topen tag[2] fid[4] mode[1]`
  `size[4] Ropen tag[2] qid[13] iounit[4]`
- `size[4] Tcreate tag[2] fid[4] name[s] perm[4] mode[1]`
  `size[4] Rcreate tag[2] qid[13] iounit[4]`
- `size[4] Tread tag[2] fid[4] offset[8] count[4]`
  `size[4] Rread tag[2] count[4] data[count]`
- `size[4] Twrite tag[2] fid[4] offset[8] count[4] data[count]`
  `size[4] Rwrite tag[2] count[4]`

9P: RPCs

- `size[4] Tremove tag[2] fid[4]`
  `size[4] Rremove tag[2]`
- `size[4] Tstat tag[2] fid[4]`
  `size[4] Rstat tag[2] stat[n]`
- `size[4] Twstat tag[2] fid[4] stat[n]`
  `size[4] Rwstat tag[2]`
- `size[4] Tclunk tag[2] fid[4]`
  `size[4] Rclunk tag[2]`
  Cierra el fichero representado por ese FID, ya no se hará referencia a ese FID.

9P: RPCs


mount()
Tattach fid=15
fid 15 = /
Rattach
fid 15 = /
stat(a/b/c)
Twalk fid=15, nfid=16, a/b/c
fid 16 = a/b/c
Rwalk
fid 16 = /a/b/c
stat
Tstat fid=16
Rstat dir entry
Tclunk fid=16
fid 16 released.
Rclunk
fid 16 gone
open(a/b/c)
Twalk fid=15, nfid=16, a/b/c
fid 16 = a/b/c
Rwalk
fid 16 = /a/b/c
Topen fid=16
fid 16 open
Ropen
read()
Tread fid=16, off=0, cnt=8k
Rread, cnt=n, data...
read()
Tread fid=16, off=n, cnt=8k
Rread, cnt=0
eof
close()
Tclunk fid=16
fid 16 released.
Rclunk
fid 16 gone

4.3 Otros sistemas de ficheros en red

Otros tipos de sistemas de ficheros en red

- Distributed FS (réplicas distribuidas).
- Distributed version control systems.
- Cluster FS.

Distributed FS (réplicas)

- Los nodos tienen una réplica (on-disk cache) del árbol de ficheros.
- Finalidad: trabajar desconectado y sincronizar cuando hay conexión. Semántica desconectada.
- Hay que resolver conflictos a la hora de sincronizar las réplicas.
- Ejemplos: CODA, AFS, etc.

Distributed version control systems

- Sistemas que sirven para mantener copias de un árbol con control de versiones.
- Guarda las distintas versiones de los ficheros, gestiona *commits*, puede crear distintas ramas, etc.
- Finalidad: principalmente, coordinar cambios en el código fuente de un proyecto de forma distribuida.
- También hay que resolver conflictos a la hora de sincronizar las réplicas.
- Ejemplos: Git, Mercurial, etc.

Cluster FS

- Los bloques de los ficheros están distribuidos en un cluster.
- Finalidad: *file striping*.
- Ejemplos: Google FS, Hadoop Distributed File System (HDFS), Red Hat Ceph, Microsoft Cluster Shared Volumes (CSV), etc.

Caso Cluster FS: Google FS

- Tolera fallos de los servidores.
- Optimizado para *append* y lecturas.
- Escalabilidad: miles de nodos por cluster, petabytes de datos.
- No muchos ficheros, pero ficheros muy grandes.
- Sacrifican latencia por *throughput*.
- Coherencia *relajada*: las réplicas pueden diferir en ciertas partes. Las aplicaciones deben lidiar con esto.
- Ficheros partidos en *chunks* de 64Mb (con checksum).

Caso Cluster FS: Google FS

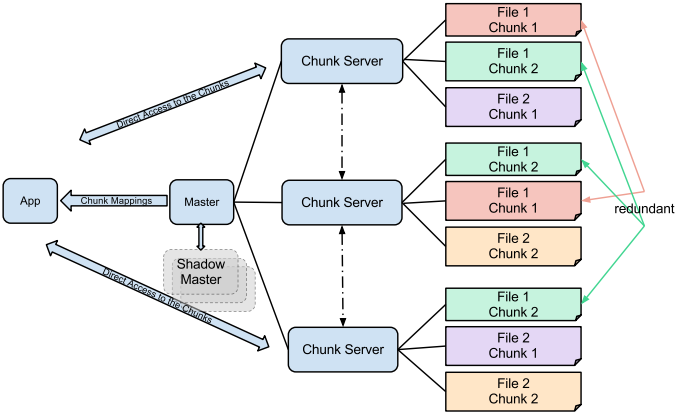


Imagen: (CC) Wikipedia

Caso Cluster FS: Google FS

Master:

- ¿Único punto de fallo?
  - Shadow Masters.
- ¿Cuello de botella? No, **sólo mantiene el control**, no hace I/O, ni coordina modificaciones en los chunks (mutación).

Caso Cluster FS: Google FS

Máster:

- Almacena sólo los metadatos: todo en memoria, log en disco para asegurar coherencia.
- Las actualizaciones de metadatos son atómicas.
- Controla el espacio de nombres.
- Hace *polling* para localizar los *chunks*, rebalancear y controlar caídas de *chunk-servers*.
- Crea nuevos *chunks*.
- Ordena la replicación de *chunks* en *chunk-servers* (3+).
- Controla los *leases* de las réplicas primarias.
- Controla los *locks* sobre ficheros/directorios.
- Realiza la recolección de basura.
- **No** coordina las *mutaciones* (un write en un chunk).

Caso Cluster FS: Google FS

Cliente:

- No tienen cache de datos.
- Tiene cache (con tiempo de caducidad) de metadatos (*chunk-handles*, etc.).
- Suele resolver distintos *chunks* a la vez.

Caso Cluster FS: Google FS

Mutación: `Write` (poco usada)

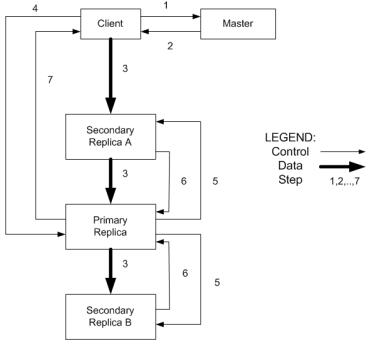


Figure extracted from the original GFS paper (SOSP 2003).

1. El cliente pide al Master la lista de réplicas y el primario.
2. El Master elige un primario si no hay. El primario adquiere un *lease*. El cliente *cachea* la respuesta del Master para futuras peticiones. Esos datos pueden caducar.
3. El cliente envía los datos a todas las réplicas. Las réplicas guardan los datos en sus buffers.
4. El cliente envía una petición `write` al primario.
5. El primario serializa las distintas mutaciones que recibe y las aplica al *chunk*. Va enviando la secuencia al resto de réplicas.
6. Cada réplica aplica las modificaciones en el orden que dicta el primario. Al aplicar un cambio, notifica al primario.
7. Tras recibir las notificaciones de las réplicas para el `write`, se responde al cliente. Si hay error en las réplicas, el cliente puede reintentar (desde el paso 3 o desde el paso 1). Después del fallo habrá zonas de la réplica fallida incoherentes $\rightarrow$ semántica relajada.

Caso Cluster FS: Google FS

Mutación: `Atomic Record Append` (muy usada).

1. El cliente envía los datos del append a todas las réplicas.
2. El cliente ordena el `atomic-append` al primario.
3. El primario comprueba si se desborda el último *chunk*.
   - Si se desborda, se rellena con un hueco el *chunk* y se le indica al cliente que la operación se debe realizar en el siguiente *chunk* (el Master lo creará) → Fragmentación interna. El tamaño de un append está limitado para evitar mucha fragmentación (1/4 de tamaño de *chunk*).
   - Si no se desborda, se serializa la escritura en el *chunk* como en un `write`.
4. Si falla alguna réplica secundaria, se repite la operación append
   - El offset se avanza de todas formas en las réplicas que fallaron: todos los chunks tienen el mismo tamaño).
   - Los datos del append aparecen repetidos en las réplicas que no fallaron → semántica **at-least-once**.

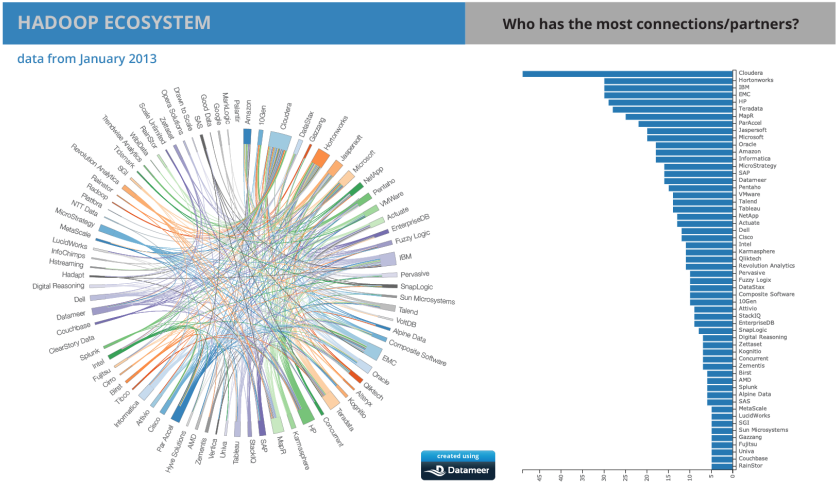
4.4 Arquitectura Apache Hadoop

Ecosistema Apache Hadoop

Proveedores Apache Hadoop

- La infraestructura de análisis de datos de Hadoop no consta de un único componente, sino que requiere instalar y configurar múltiples componentes en diferentes capas:
  - Sistema de ficheros distribuido: HDFS. Además, formatos de serialización de datos tales como Avro, Parquet, etc.
  - Bases de datos columnares: HBase, Cassandra.
  - Núcleo o motor de procesamiento de datos: Hadoop, Spark.
  - Sistema de gestión de recursos: YARN, Mesos.
  - Sistema de diseño y monitorización de tareas: Zookeper, Oozie.
  - APIs de análisis de datos: Hive, Pig, Mahout, APIs de Spark.
- Existen empresas que proporcionan **distribuciones Hadoop** con estos componentes (y otros propios) ya preinstalados y configurados, además de ofrecer servicio de soporte, mantenimiento, configuración y personalización.

Mercado proveedores Apache Hadoop



Proveedores Apache Hadoop

- Actualmente, **Cloudera** sigue siendo el **principal proveedor** de Hadoop a nivel mundial, con una cuota de mercado estimada en más del 50 %. Su política de negocio a día de hoy pasa por ofrecer un servicio de soporte integral, pero sobre ciertos componentes propietarios exclusivos de su plataforma (e.g. Cloudera Manager).
  - Esto crea importantes focos de incompatibilidad con otras distribuciones.
- Otras distribuciones como MapR siguen una política similar.
- No obstante, tanto estos dos proveedores como la mayoría de las restantes empresas de este sector contribuyen muy activamente **al desarrollo** de los pricipales **componentes open source** del ecosistema.
  - Se trata de un esquema típico de **colaboración-competición**. Las empresas colaboran en los componentes comunes para afianzar y mejorar la plataforma global, concentrándose luego en ofertar productos y servicios de valor añadido sobre la plataforma.

Open Data Platform Initiative

- Recientemente, un grupo de empresas del sector han formado una alianza llamada Open Data Platform Initiative (ODPI).
- Surge para luchar contra la **fragmentación tecnológica** que vive el ecosistema Hadoop actualmente, con proyectos similares que ofrecen funcionalidad similar pero incompatibles entre sí.
- El objetivo es proporcionar una **plataforma open source uniforme** para Hadoop, que garantice la interoperabilidad de los productos y acelere la implementación de **casos de uso**.

Hadoop: HDFS

- Modelado a partir del Google File System [2].
- Optimizado para maximizar el throughput, mejor cuanto más grandes sean los archivos.
- Tamaño de bloques grande, intenta optimizar distribución de datos en nodos siguiendo principios de localidad espacial y temporal (similar a jerarquía de memoria).
- Las primeras versiones utilizaban un nodo para metadatos (*NameNode*) y varios nodos para almacenar y trabajar con los datos (*DataNodes*). En las versiones más modernas, datos y metadatos están distribuidos.

Hadoop: Almacenamiento en la nube

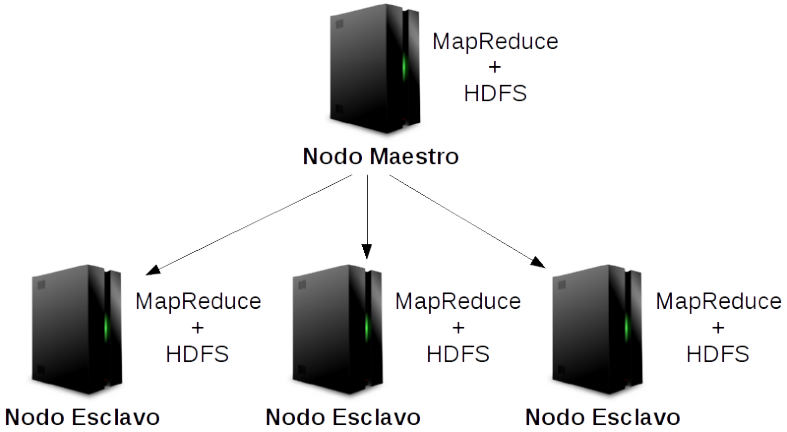
- Es posible poder ejecutar instancias Hadoop sobre servicios de computación en la nube.
- Ejemplo: servicios web de Amazon.
  - Podemos utilizar instancias de cómputo y almacenamiento de datos en Amazon EC2 para ejecutar nuestras tareas desde Hadoop (importación de recursos).
  - Adicionalmente, Amazon también proporciona Amazon Elastic MapReduce (EMR), un entorno de gestión alternativo similar a Hadoop, que permite gestionar las tareas y recursos de cluster.
  - EMR también puede albergar otros entornos para procesado de flujos de datos, como Apache Spark o Presto.
- Otros proveedores: Microsoft Azure, etc.

Hadoop

- El núcleo de todo el ecosistema de aplicaciones.
- Hadoop Common Package (abstracciones) + HDFS (sistema de ficheros distribuido) + YARN (MapReduce engine).
- Se aplica el paradigma MapReduce a datos almacenados en múltiples nodos.
- Arquitectura Maestro-Esclavo (en su primera versión).

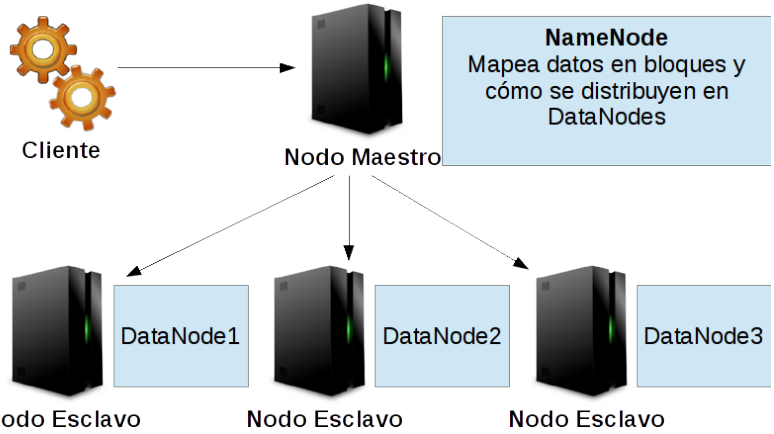
Hadoop: MR1

- Infraestructura para Hadoop (MR1).



Hadoop: MR1

- Infraestructura para Hadoop (MR1).



Hadoop: MR1

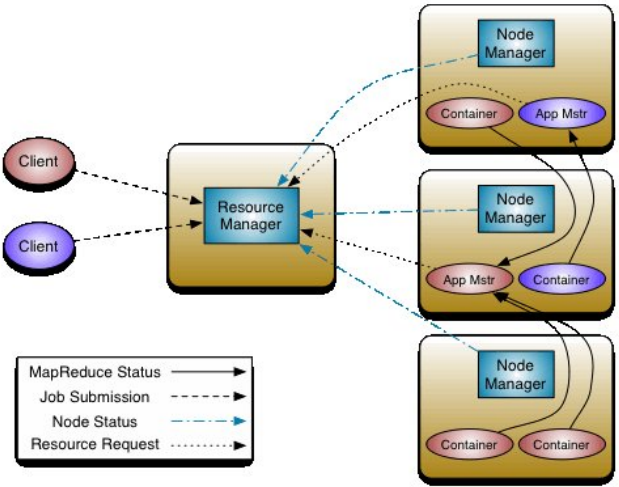
- Gestión de tareas MapReduce (MR1).
- El JobTracker mantiene en memoria del nodo maestro información sobre todas las tareas planificadas y en ejecución.
- Se gestionan tanto las tareas de tipo `map` como las de tipo `reduce`, asociadas a trabajos de alto nivel enviados por el cliente.
- Límites versión 1 (MapReduce/MR1).
  - ~5.000 NODOS, 40.000 tareas concurrentes.
  - Distribución fija de recursos entre procesos *map* y `reduce`.
  - Fallos acaban con trabajos en ejecución y encolados (catastrófico).

Hadoop: MR vs YARN

- Principales diferencias de YARN (MR2) respecto a MapReduce (MR1).
  - Soporte para mútiples estrategias (batch, flujo de datos, interactivo).
  - Data Operating System (un solo conjunto de datos, múltiples instancias).
  - Gestión de metadatos y trabajos distribuido.
  - Mejor aprovechamiento de los recursos de computación y almacenamiento.
  - Introducción de aspectos de seguridad y autenticación en el diseño.
  - Compatibilidad hacia atrás (evitar grandes cambios para Hive, Pig, etc.).

Hadoop: YARN

- YARN/MR2.



Hadoop: YARN

- Principales elementos de YARN (MR2).
- **Aplicación**.
  - Representación a alto nivel de un trabajo de procesamiento de datos.
  - Puede ser un trabajo MapReduce o un script de shell.
- **Contenedor**.
  - Unidades de particionado de los recursos hardware subyacentes.
  - Permiten la distribución de los recursos computacionales entre diferentes aplicaciones en ejecución, intentando maximizar el aprovechamiento de los mismos.

Hadoop: Planificación de tareas

- Principales componentes de YARN (MR2).
- **Resource Manager**.
  - Gestión de tareas de alto nivel.
  - Gestión de colas jerárquicas de trabajos.
- **Application Master**.
  - Uno por cada instancia a nivel de aplicación.
  - Gestiona recursos, progreso de tareas, planificación, etc.
- **Node Manager**.
  - Agentes encargados de la gestión y monitorización de contenedores en cada nodo del cluster.

Hadoop: Planificación de tareas

- Avances en YARN (MR2).
- El Resource Manager continúa siendo un punto de fallo singular. Genera graves trastornos en caso de pérdida de información.
- Solución: intentar reconstruir la información que contiene en caso de fallo.
- Propuesta: Zookeper (u otro meta-gestor de recursos) debe monitorizar las instancias de Resource Managers y actuar en caso de fallo.

Bibliografía I

- *A. S. Tanenbaum.* Operating Systems, design and implementaiton. Pearson Prentice Hill.
- *A. S. Tanenbaum.* Distributed Systems. Pearson Prentice Hill.
- *A. S. Tanenbaum.* Modern Operating Systems. Pearson Prentice Hill.
- *A. Silberschatz.* Operating Systems. Wiley.
- *M. L. Powell , S. R. Kleiman , S. Barton , D. Shah , D. Stein , M. Weeks*. SunOS Multi-thread Architecture. Winter 1991 USENIX Conference.
- *D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, S. Zhou*. 2000. Process migration. ACM Comput. Surv. 32, 3, 241-299.
- *John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson and Brent B. Welch*, The Sprite Network Operating System, IEEE Computer, 21. 1988.
- *N. Deshpande et al.. Analysis of the Go runtime scheduler*

Bibliografía II


[van Steen & Tanenbaum, 2017] van Steen, M., Tanenbaum, A. S.
Distributed Systems.
Third Edition, version 01. 2017.


[Colouris et al., 2011] Colouris, G., Dollimore, J., Kindberg, T., Blair, G.
Distributed Systems. Concepts and Design.
Pearson, May, 2011.


[Ortega et al., 2005] Ortega, J., Anguita, M., Prieto, A.
Arquitectura de Computadores.
Ediciones Paraninfo, S.A. 2005.