Universidad Internacional de Valencia

Procesamiento de Datos Masivos

03MBID

Tema 3: Hadoop MapReduce

Basado en el mayerial del Prof. Tomás y Prof. Jesús Morán

Yudith Cardinale

Diciembre 2022

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

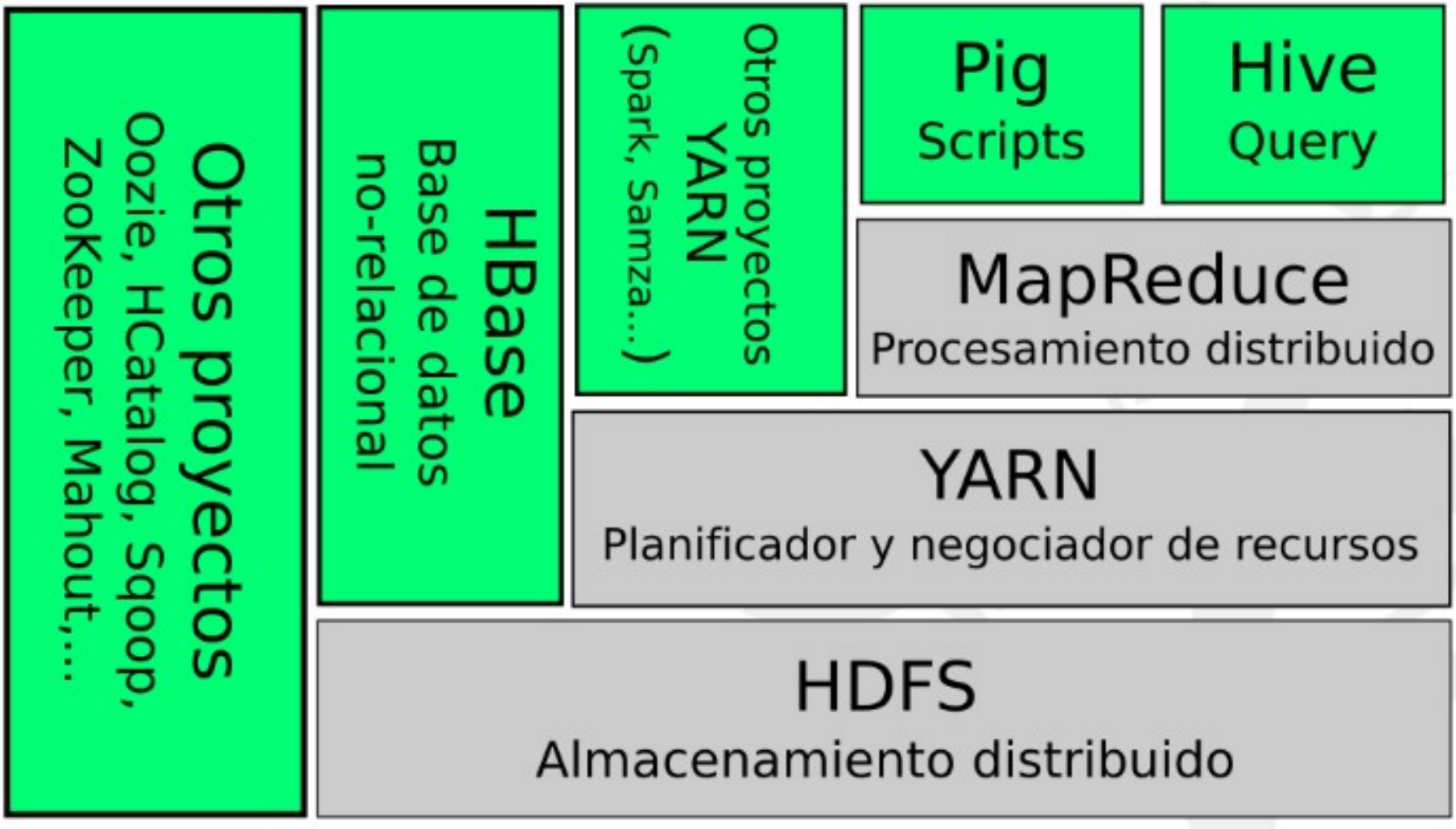
Características de Hadoop

- Tres componentes principales
  - Almacenamiento distribuido: **HDFS**
  - Planificación de tareas y negociación de recursos: **YARN**
  - Procesamiento distribuido: **MapReduce**

- Ventajas
  - Bajo costo: *clusters* baratos o cloud
  - Facilidad de uso
  - Tolerancia a fallos
  - Popular, escalable, *open source*

Características de Hadoop

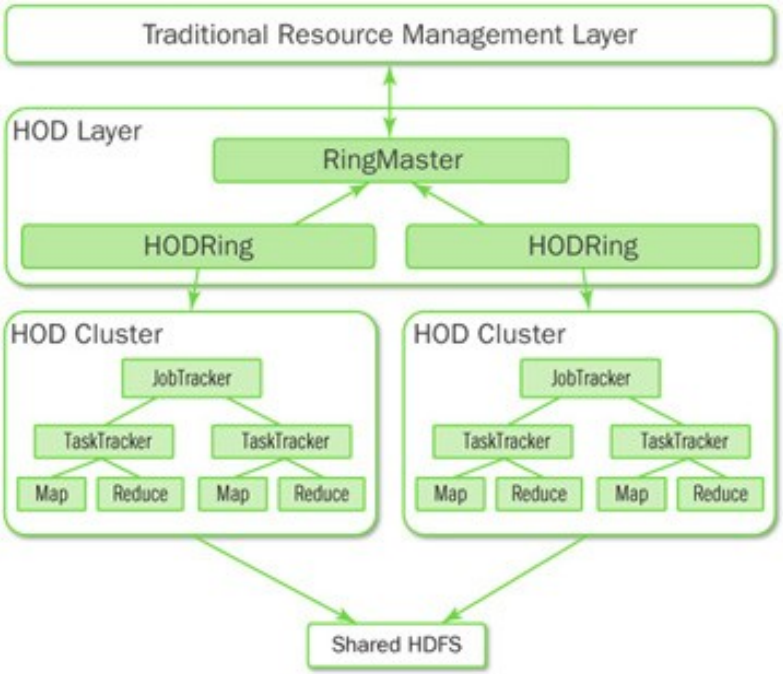
Hadoop

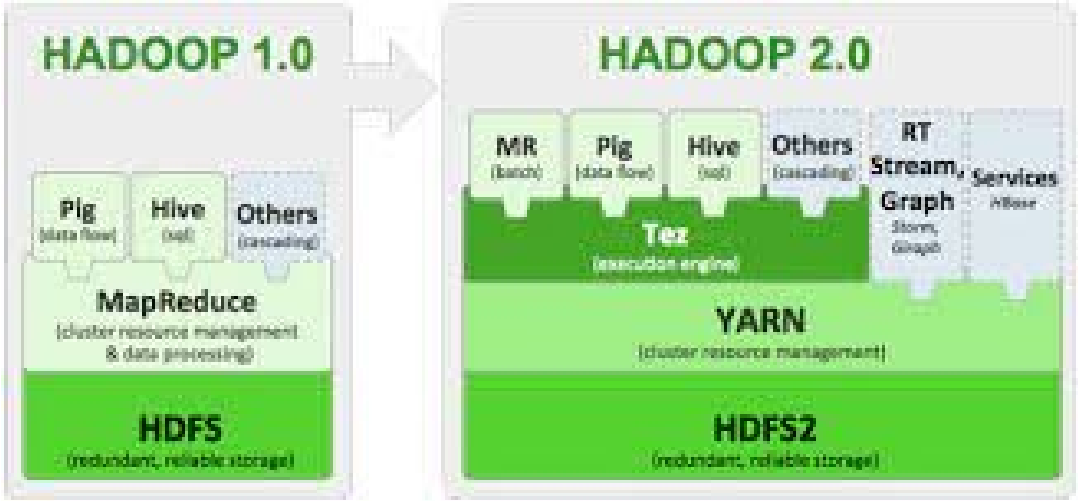


Evolución de Hadoop

viu Universidad Internacional de Valencia

- Según Murphy et al., la evolución lleva cuatro fases
  - Fase 0: ad-hoc cluster
  - Fase 1: Hadoop *on demand*
  - Fase 2: **Hadoop 1**
  - Fase 3: **Hadoop 2**



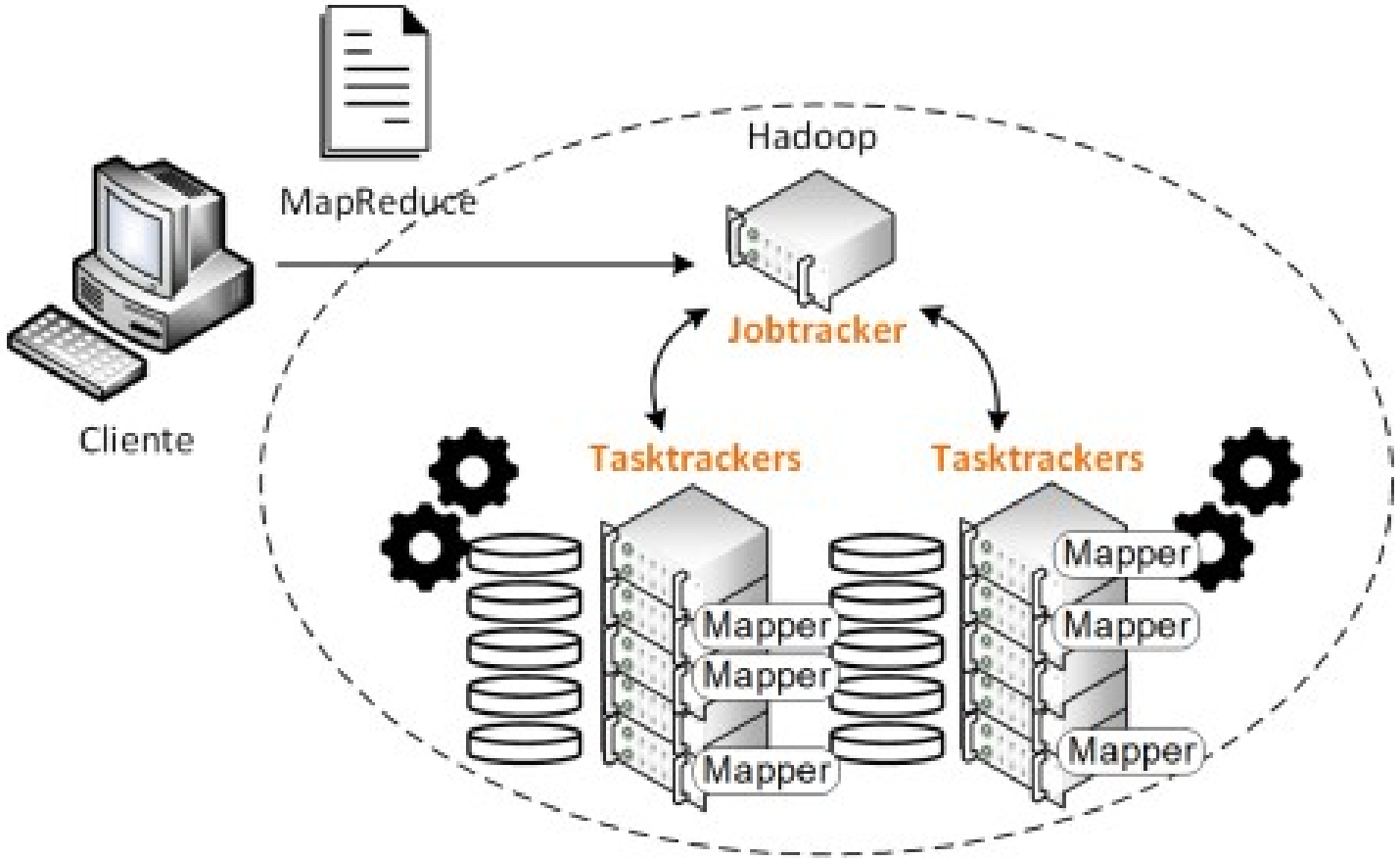


Murthy, A. C., Vavilapalli, V. K., Eadline, D., & Markham, J. (2014). Apache Hadoop YARN: moving beyond MapReduce and batch processing with Apache Hadoop 2. Pearson Education: London, United Kingdom.

Evolución de Hadoop

Hadoop 1: Componentes

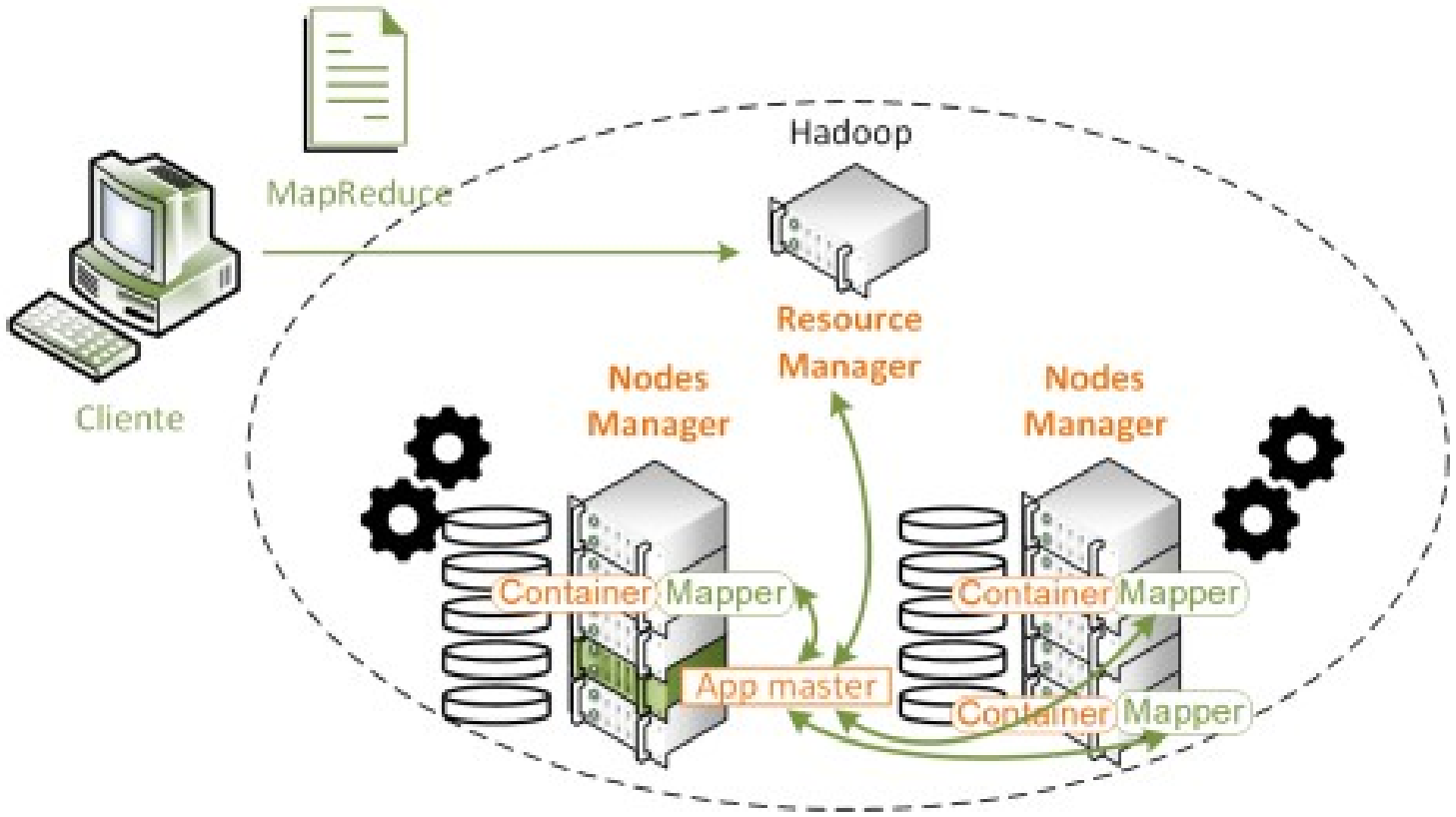
- Sistema de archivos: HDFS, S3, …
- Cómputo distribuido: Hadoop MapReduce



Evolución de Hadoop

Hadoop 2: Componentes

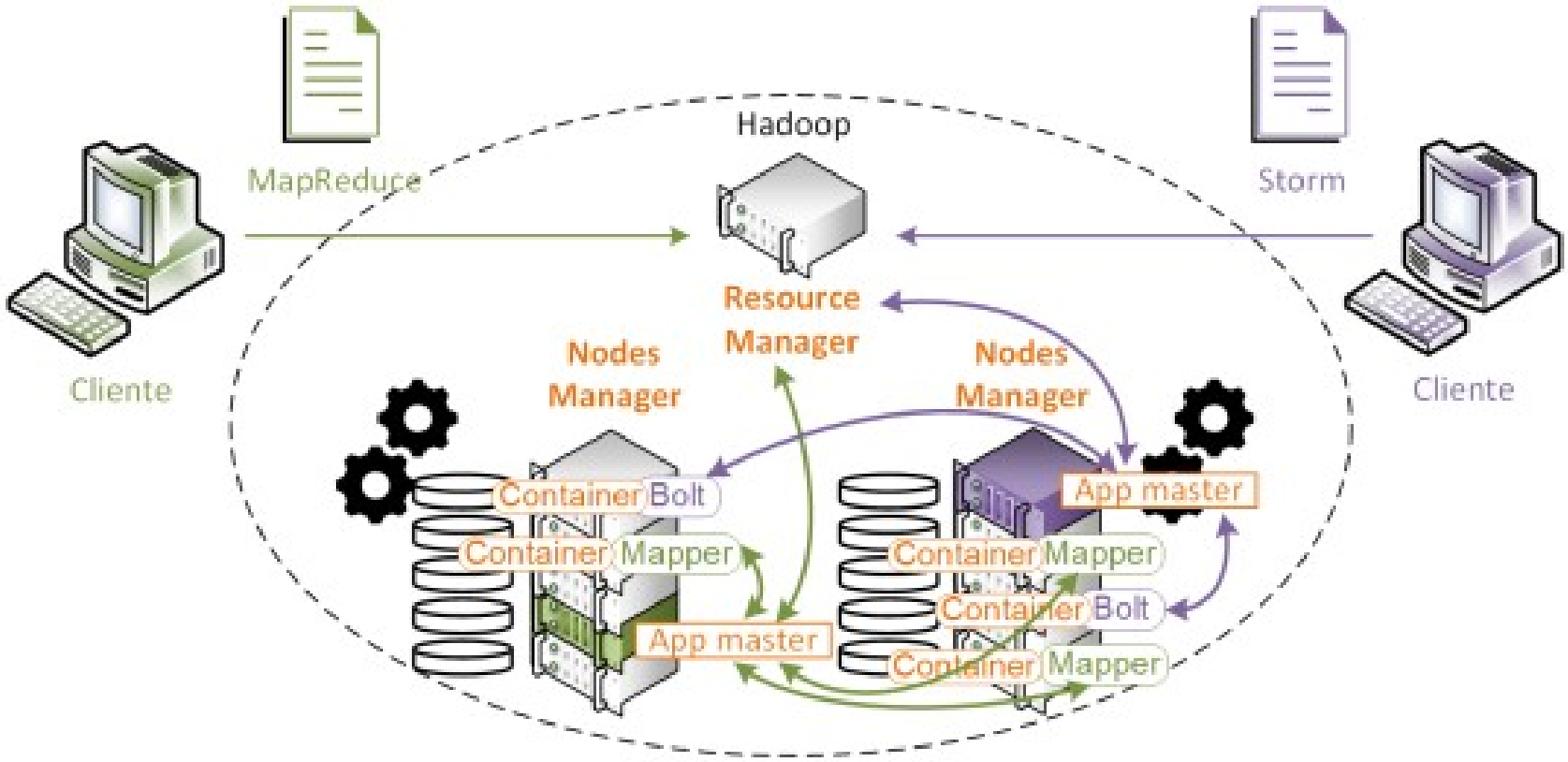
- Sistema de archivos: HDFS, S3, …
- Cómputo distribuido: Hadoop MapReduce, Spark, Storm



Evolución de Hadoop

Hadoop 2: Componentes

- Sistema de archivos: HDFS, S3, …
- Cómputo distribuido: Hadoop MapReduce, Spark, Storm



Instalación de Hadoop

Instalación

Instalación relativamente simple: aplicación Java

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

Modos de funcionamiento:

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

Instalación de Hadoop

viu Universidad Internacional de Valencia

- AWS workspace:
  - Instalar docker-compose
  - Cluster de 3 máquinas
  - Recomendación: Se deben hacer copias de seguridad de todos los programas
  - Problemas de acceso: contactar con soporte
- Instalación virtualizada
  - Sistema operativo en VM: Almalinux, Centos, Ubuntu …
  - Instalación de Hadoop:
    - Local
    - Pseudo-distribuido
    - Contenedores: docker-compose

Instalación de Hadoop

- Instalación en varias máquinas
  - Hardware, operativos, redes, usuarios, puertos,…
  - Edge node
  - Cluster
    - Sistema de archivos distribuido
    - Framework Big Data
    - Configuración

- Instalación en la nube: AWS
  - IAM: Creación de cuentas y permisos
  - S3: almacenamiento masivo
  - EMR: Cluster Big Data
  - Se deben hacer copias de seguridad de los programas

Instalación de Hadoop

- **Google Colab**
  - No se requiere instalar nada en el ordenador
  - Notebook web
  - Instalar Hadoop
  - Importar programas
  - Se deben hacer copias de seguridad de los programas

Filesystems en Hadoop

Filesystems en Hadoop

Hadoop tiene una noción abstracta de los filesystems

- HDFS es un caso particular de filesystem

Algunos filesystems soportados:

FS	URI	Descripción
`Local`	*file*	Disco local
`HDFS`	*hdfs*	Sistema HDFS
`HFTP`	*hftp*	RO acceso a HDFS sobre HTTP
`HSFTP`	*hsftp*	RO acceso a HDFS sobre HTTPS
`WebHDFS`	*webhdfs*	RW acceso a HDFS sobre HTTP
`S3` (nativo)	*s3n*	Acceso a S3 nativo
`S3` (block)	*s3*	Acceso a S3 en bloques

Ejemplo:

- `hadoop fs -ls file:///home/pepe`

Para usar con HDFS se recomienda el comando `hdfs dfs`:

- `hdfs dfs -help`

Filesystems en Hadoop

- **HDFS**
  - Sistema de archivos distribuido
  - Persistencia masiva
  - Bloques de 128MB (por defecto)
  - Replicación en tres servidores (por defecto)
  - API: programática, CLI (*command line interface*)

HDFS

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
- Modificaciones siempre al final de los ficheros
- No permite múltiples escritores (modelo *single-writer, multiple-readers*

HDFS

Conceptos de HDFS

Namenode

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

Datanodes

Mantienen los bloques de datos

- No tienen idea sobre los ficheros

Conceptos de HDFS (cont.)

Bloques

Por defecto 128 MB, tamaño configurable por fichero

- bloques pequeños aumentan el paralelismo (un bloque por Map)
- bloques más grandes reducen la carga del Namenode

Replicados a través del cluster

- Por defecto, 3 réplicas (configurable por fichero)

Checkpoint node

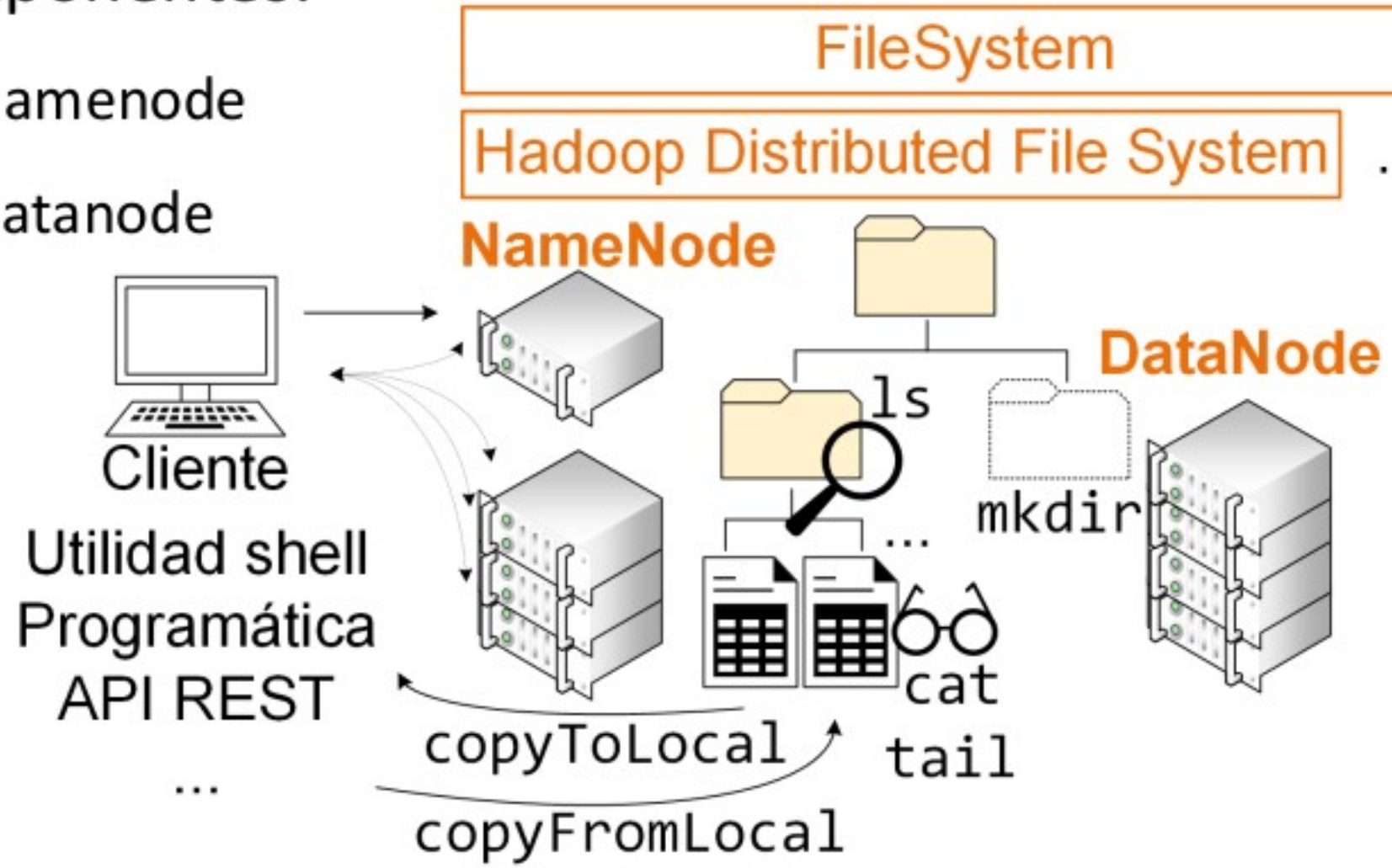
Mantienen checkpoints del Namenode

- debería ejecutarse en un sistema con características similares al Namenode

HDFS

Conceptos de HDFS (cont.)

- Componentes:
  - Namenode
  - Datanode



HDFS: propiedades configurables (I)

Múltiples propiedades configurables (fichero `hdfs-site.xml`)

- `dfs.namenode.name.dir`: lista (separada por comas) de directorios donde el namenode guarda sus metadatos (una copia en cada directorio), por defecto `file://${hadoop.tmp.dir}/dfs/name`
- `dfs.datanode.data.dir`: lista (separada por comas) de directorios donde los datanodes guarda los bloques de datos (cada bloque en sólo uno de los directorios), por defecto `file://${hadoop.tmp.dir}/dfs/data`
- `dfs.namenode.checkpoint.dir`: lista (separada por comas) de directorios donde el CheckPoint node guarda los checkpoints (una copia en cada directorio), por defecto `file://${hadoop.tmp.dir}/dfs/namesecondary`

HDFS: propiedades configurables (II)

- `dfs.blocksize`: tamaño de bloque para nuevos ficheros, por defecto 128MB
- `dfs.replication`: nº de réplicas por bloque, por defecto 3
- `dfs.replication.max`: máximo nº de réplicas permitido por bloque, por defecto 512
- `dfs.namenode.replication.min`: mínimo nº de réplicas permitido por bloque, por defecto 1

Interfaz con HDFS

Varias interfaces:

1. Interfaz en línea de comandos: comando `hdfs dfs`
2. Interfaz web
3. Interfaz Java

Interfaz en línea de comandos:

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

Más información: hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html, hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-common/FileSystemShell.html

HDFS

Interfaz en línea de comandos (I)

Algunos comandos de manejo de ficheros

Comando	Significado
`hdfs dfs -ls <path>`	Lista ficheros
`hdfs dfs -ls -R <path>`	Lista recursivamente
`hdfs dfs -cp <src> <dst>`	Copia ficheros HDFS a HDFS
`hdfs dfs -mv <src> <dst>`	Mueve ficheros HDFS a HDFS
`hdfs dfs -rm <path>`	Borra ficheros en HDFS
`hdfs dfs -rm -r <path>`	Borra recursivamente
`hdfs dfs -cat <path>`	Muestra fichero en HDFS
`hdfs dfs -tail <path>`	Muestra el final del fichero
`hdfs dfs -stat <path>`	Muestra estadísticas del fichero
`hdfs dfs -mkdir <path>`	Crea directorio en HDFS
`hdfs dfs -chmod ...`	Cambia permisos de fichero
`hdfs dfs -chown ...`	Cambia propietario/grupo de fichero
`hdfs dfs -du <path>`	Espacio en bytes ocupado por ficheros
`hdfs dfs -du -s <path>`	Espacio ocupado acumulado
`hdfs dfs -count <paths>`	Cuenta nº dirs/ficheros/bytes

Interfaz en línea de comandos (II)

Movimiento de ficheros del sistema local al HDFS:

Comando	Significado
`hdfs dfs -put <local> <dst>`	Copia de local a HDFS
`hdfs dfs -copyFromLocal ...`	Igual que -put
`hdfs dfs -moveFromLocal ...`	Mueve de local a HDFS
`hdfs dfs -get <src> <loc>`	Copia de HDFS a local
`hdfs dfs -copyToLocal ...`	Copia de HDFS a local
`hdfs dfs -getmerge ...`	Copia y concatena de HDFS a local
`hdfs dfs -text <path>`	Muestra el fichero en texto

Interfaz en línea de comandos (III)

Otros comandos:

Comando	Significado
`hdfs dfs -setrep <path>`	Cambia el nivel de replicación
`hdfs dfs -test -[defsz] <path>`	Tests sobre el fichero
`hdfs dfs -touchz <path>`	Crea fichero vacío
`hdfs dfs -expunge`	Vacía la papelera
`hdfs dfs -usage [cmd]`	Ayuda uso de comandos

Más información: http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

Otras herramientas para mover datos

Es posible mover datos a/desde HDFS usando otras herramientas

- `distcp` Transferir datos en paralelo entre dos filesystems Hadoop
  - Ejemplo

```
hadoop distcp hdfs://nnode1/foo hdfs://nnode2/bar
```

  - Aplicación MapReduce map-only
  - Puede usar otros filesystems (HFTP, WebHDFS, etc.)
  - Interesante para mover cantidades masivas de datos
  - Más opciones: `hadoop distcp`
- Apache Flume servicio para recoger, agregar y mover grandes cantidades de datos de log a HDFS
- Apache Sqoop transferencia masivas de datos entre bases de datos estructuradas y HDFS

HDFS

Herramientas para la gestión del HDFS

Hadoop proporciona un conjunto de herramientas para chequear y optimizar el HDFS

- `hdfs dfsadmin`: optiene información del estado del HDFS
- `hdfs fsck`: chequeo del filesystem
- `hdfs balancer`: herramienta de rebalanceo de bloques entre datanodes

hdfs dfsadmin

Algunas opciones (usar `hdfs dfsadmin` **comando**)

Comando	Significado
`-help`	Ayuda
`-report`	Muestra estadísticas del filesystem
`-setQuota`	Fija una cuota en el número de nombres en un directorio (nº de ficheros/directorios)
`-clrQuota`	Borra la cuota de nombres
`-setSpaceQuota`	Fija una cuota en el espacio ocupado en un directorio
`-clrSpaceQuota`	Borra la cuota de espacio
`-refreshNodes`	Actualiza los nodos que se pueden conectar
`-safemode`	fija o chequea el *safe mode*
`-saveNameSpace`	en *safe mode*, salva el filesystem en memoria a un nuevo fichero `fsimage` y resetea el fichero `edits`

HDFS

hdfs fsck

Chequea la salud de los ficheros en HDFS

- Chequea los bloques:
  - *Over-replicated*: con replicas de más
  - *Under-replicated*: con replicas de menos
  - *Misreplicated*: replicas mal colocadas
  - Corruptos
  - *Missing replicas*: sin réplicas
- Ejemplos:
  - Chequea recursivamente todo el HDFS

    ```
    hdfs fsck /
    ```
  - Informa del número de bloques de un fichero y su localización

```
hdfs fsck /user/pepe/foo -files -blocks -racks
```

HDFS

Namenode principal

Estructura de directorios

```
${dfs.namenode.name.dir}/
├── current
│   ├── VERSION
│   ├── edits_0000000000000000001-0000000000000000019
│   ├── edits_inprogress_0000000000000000020
│   ├── fsimage_0000000000000000000
│   ├── fsimage_0000000000000000000.md5
│   ├── fsimage_0000000000000000019
│   ├── fsimage_0000000000000000019.md5
│   └── seen_txid
└── in_use.lock
```

Ficheros en el namenode

Ficheros en `dfs.namenode.name.dir`

- `VERSION` información sobre la versión de HDFS
- Ficheros `edits_startID-endID`: logs de transacciones ya finalizadas
- Fichero `edits_inprogress_startID`: logs de transacciones actuales
- Ficheros `fsimage`: información de los metadatos del filesystem
  - Contiene información de directorios y ficheros, incluyendo los bloques (inodos) que los forman
  - La localización de los bloques en los Datanodes se guarda en memoria

Inicio del Namenode

Cuando se inicia el Namenode:

1. Carga el último `fsimage` en memoria y aplica las modificaciones indicadas en `edits`
2. Con esta imagen reconstruida, crea un nuevo `fsimage` y un `edits` vacío
3. Espera a que los Datanodes le envíen información de los bloques que tienen
   - Esta información se guarda en memoria

Modo seguro

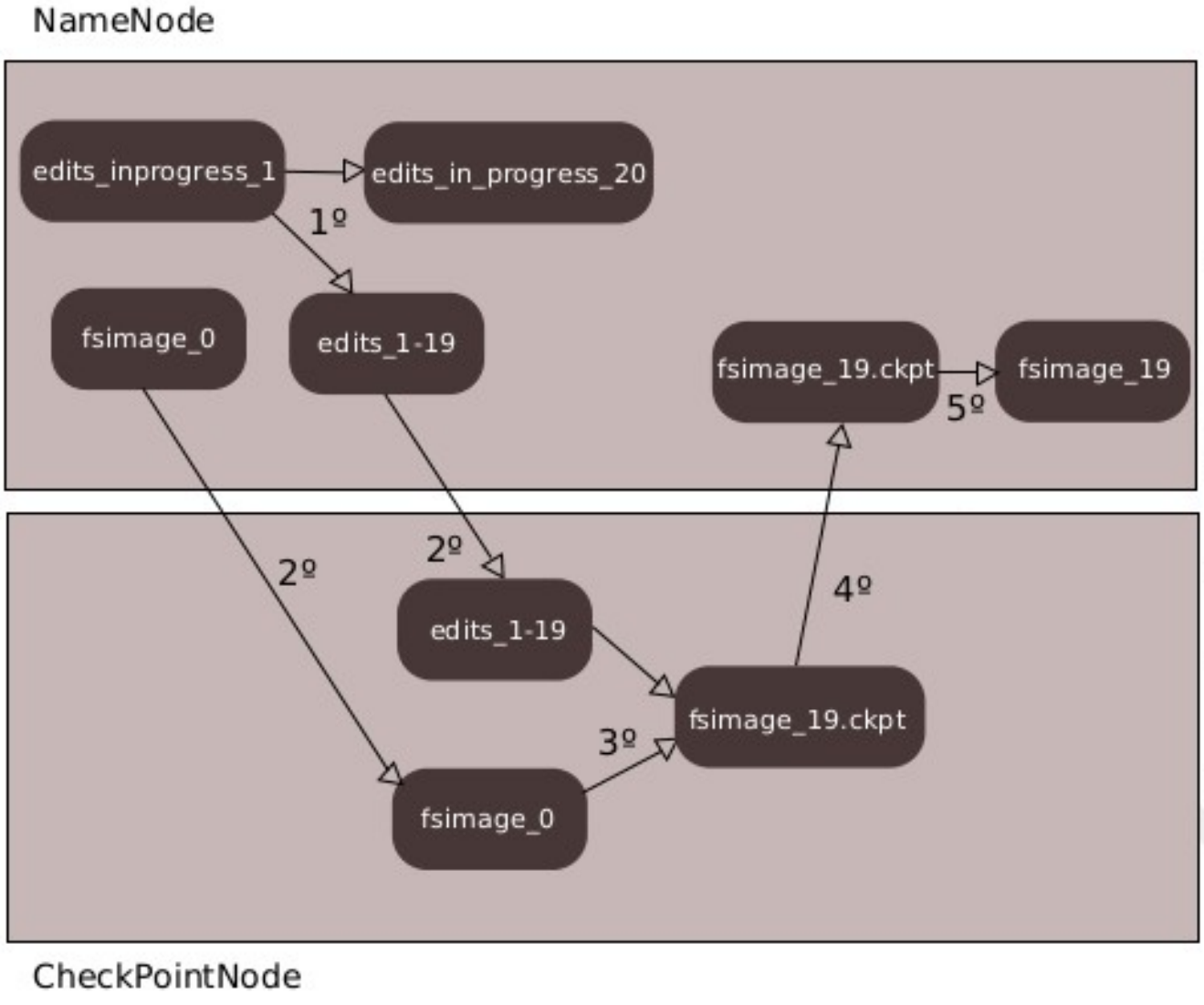
Durante la inicialización, el sistema está en modo seguro (*safe mode*)

- Solo permite acceso de lectura
- El modo seguro termina 30 segundos después de que el 99.9 % de los bloques alcancen un nivel mínimo de replicación
- Propiedades ajustables:

Propiedad	**Por defecto**
`dfs.namenode.replication.min`	1
`dfs.namenode.safemode.threshold-pct`	0.999
`dfs.namenode.safemode.extension`	30 s

HDFS

Checkpoint node (aka Namenode secundario)



1. NN rota el fichero de edits actual. En seen_txid guarda el ID de la última transacción
2. CPN obtiene el último fsimage y edits del NN
3. CPN mezcla los ficheros
4. CPN transfiere la mezcla al NN
5. NN renombra fsimage

Checkpoint node

En un sistema ocupado, el fichero del `edits` puede crecer demasiado

- El Checkpoint Node se ocupa de mezclar `edits` y `fsimage` para inicializarlo
  - Proceso costoso en recursos
  - El CPN tiene requisitos de memoria similares a los del NN
- Checkpoint realizado cada hora (`dfs.namenode.checkpoint.period`) o cada 1 M transacciones (`dfs.namenode.checkpoint.txns`)
- Se puede cambiar por un Backup Node
  - Replica completa de la memoria del NN (necesita la misma cantidad de memoria)
  - Realiza los checkpoints

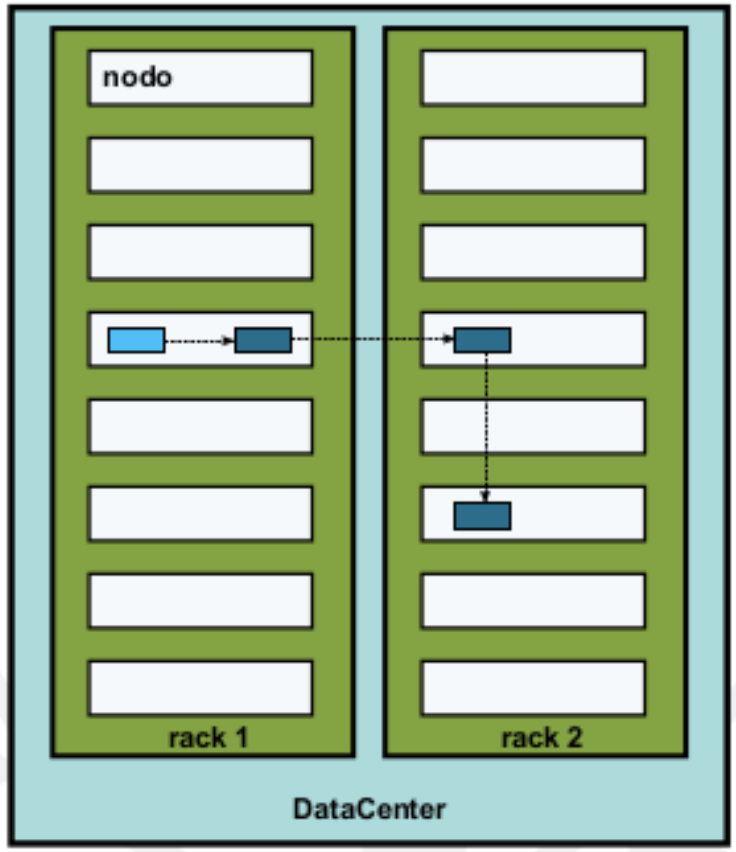
En caso de fallo total del Namenode, se puede recuperar el último checkpoint

- Iniciar el demonio del Namenode usando `hdfs namenode -importCheckpoint`

Localización de las replicas

Política por defecto:

- 1ª réplica: en el nodo del cliente o en un nodo al azar
- 2ª réplica: en un **rack** diferente de la primera (elegido al azar)
- 3ª réplica: en el mismo rack que la 2ª, pero en otro nodo
- Otras réplicas: al azar (se intenta evitar colocar demasiadas réplicas en el mismo rack)



- Más información hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication

Problemas con el Namenode

El Namenode es un *single point of failure* (SPOF)

- Si falla es imposible acceder a los datos
- Posibilidad de recuperación a partir de los checkpoints
- Conveniente guardar varias réplicas de los datos del namenode (RAID, indicar en `dfs.namenode.name.dir` directorios en diferentes máquinas, etc)

Mejoras en la versión 2.0

- HDFS High-Availability
- HDFS Federation

HDFS High-Availability

Un par de Namenodes en configuración activo-standby

- si falla el Namenode activo, el otro ocupa su lugar

Consideraciones

- Los Namenodes deben usar un almacenamiento compartido de alta disponibilidad
- Los Datanodes deben enviar informes de bloques a los dos Namenodes (el block mapping va en memoria, no en disco)
- Los Clientes deben manejar el fallo del Namenode de forma transparente

Más información: hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html, hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

HDFS Federation

El Namenode mantiene, en memoria, referencias a cada fichero y bloque en el filesystem

- problemas de escalabilidad

HDF Federation, introducida en la versión 2.0

- Permite usar varios Namenodes
- Cada uno gestiona una porción del espacio de nombres del filesystem
- Los Namenodes no se coordinan entre sí
- Cada Datanodes se registra con todos los Namenodes

Más información: hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/Federation.html

Hadoop v3

Novedades en HDFS v3

- Uso de **códigos de borrado** (*erasure coding*) para reducir el overhead de la replicación
  - Reduce el overhead a no más del 50 %
  - Ejemplo: ficheros de 6 bloques:
    - replicación x3: 18 bloques
    - EC: 9 bloques (6 datos + 3 paridad)
  - Implica un mayor coste de procesamiento
- Soporte de múltiples NameNodes en stand-by
- Soporte de balanceo de datos intra-nodo

YARN

YARN: *Yet Another Resource Negociator*

Se encarga de la gestión de recursos y job-scheduling/monitorización usando tres demonios:

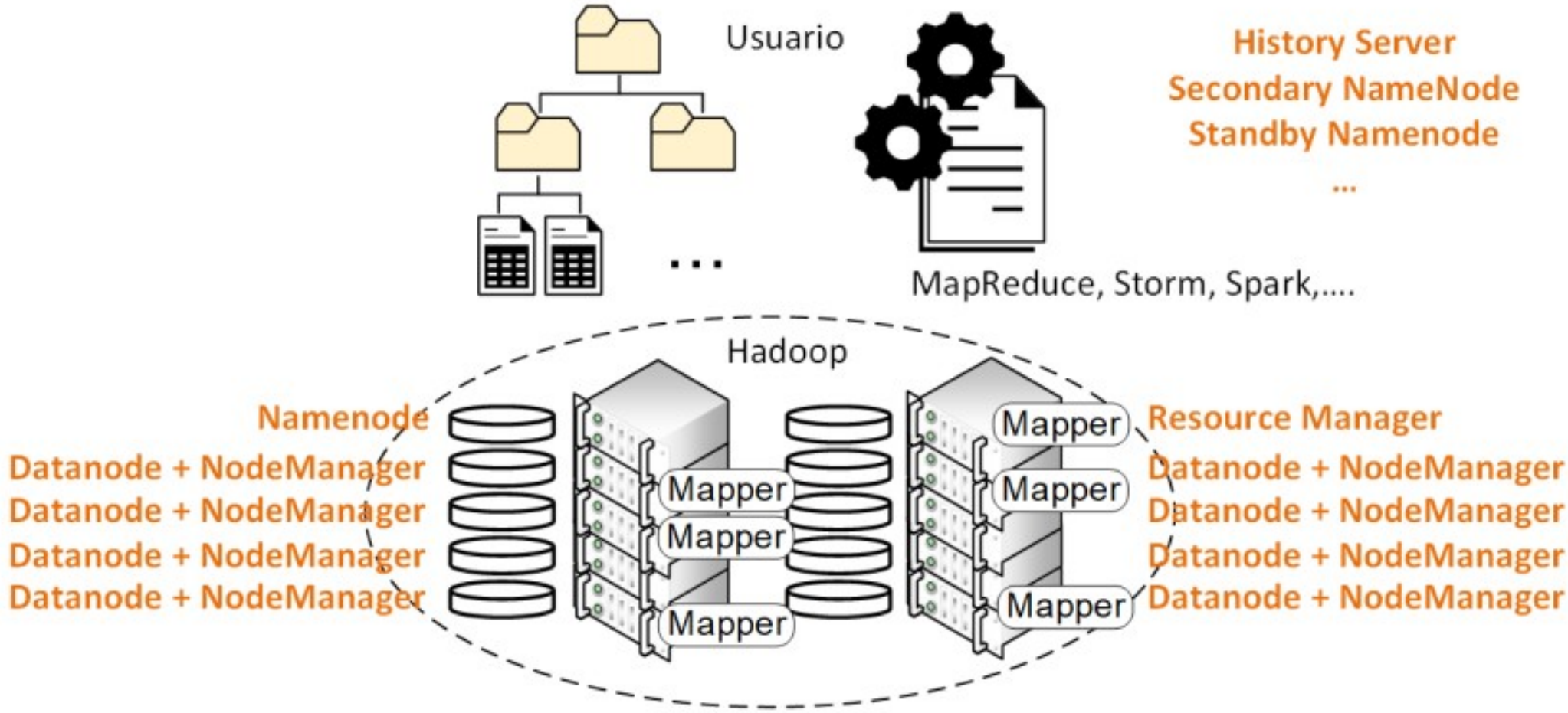
- *Resource manager* (RM): planificador general
- *Node managers* (NM): monitorización, uno por nodo
- *Application masters* (AM): gestión de aplicaciones, uno por aplicación

Permite que diferentes tipos de aplicaciones (no solo MapReduce) se ejecuten en el cluster

- Las aplicaciones se despliegan en contenedores (YARN JVMs)
- En Hadoop v3 se pueden usar Dockers

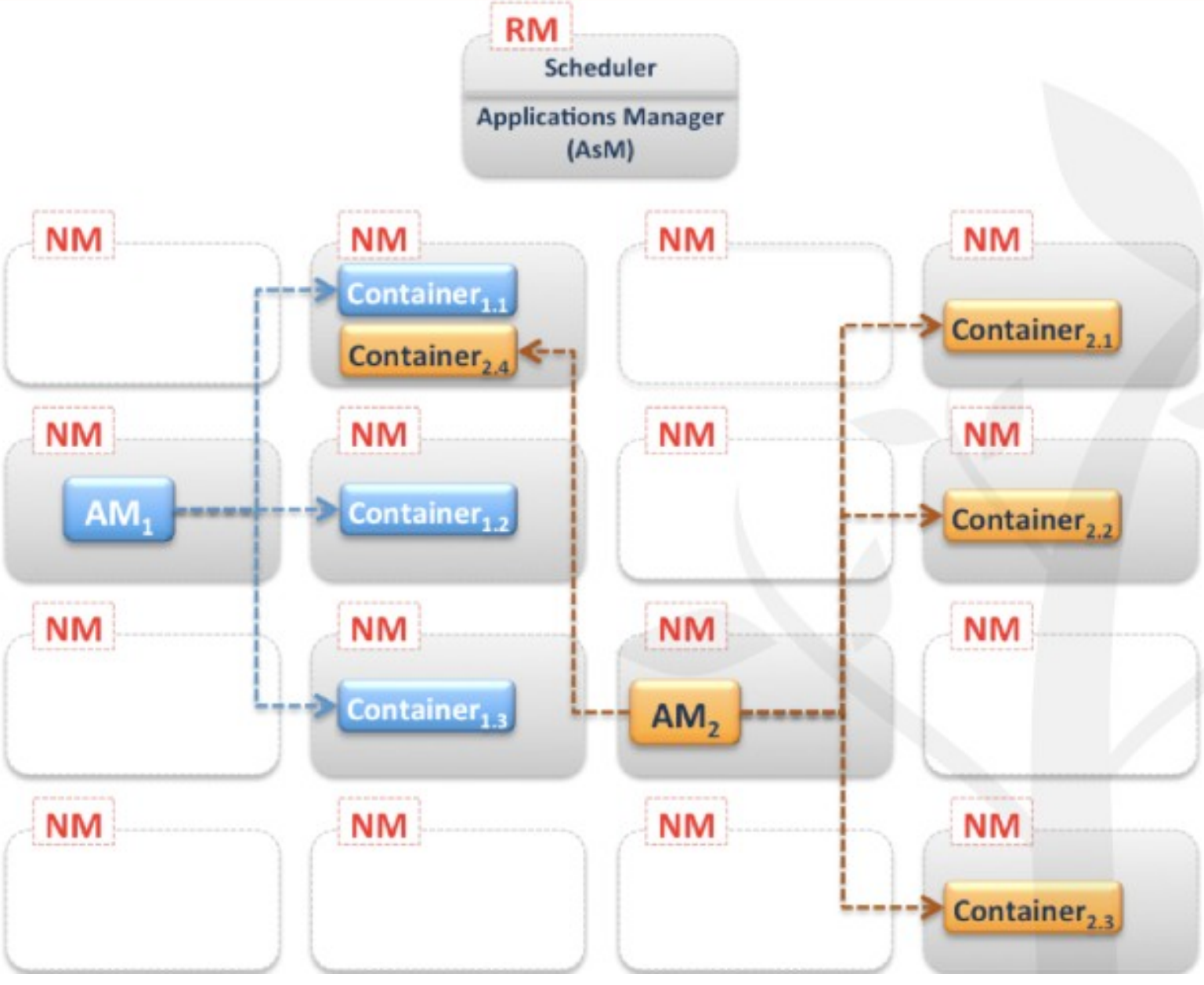
YARN

Hadoop v2: YARN



YARN

Arquitectura YARN



Demonios YARN (I)

Resource manager

- arbitra los recursos entre las aplicaciones en el sistema
- demonio global, obtiene datos del estado del cluster de los node managers
- dos componentes:
  - Scheduler: planifica aplicaciones en base a sus requerimientos de recusos
  - Applications Manager: acepta trabajos, negocia contenedores y gestiona fallos de los Application Masters

Node managers

- uno por nodo
- monitorizan los recursos del cluster

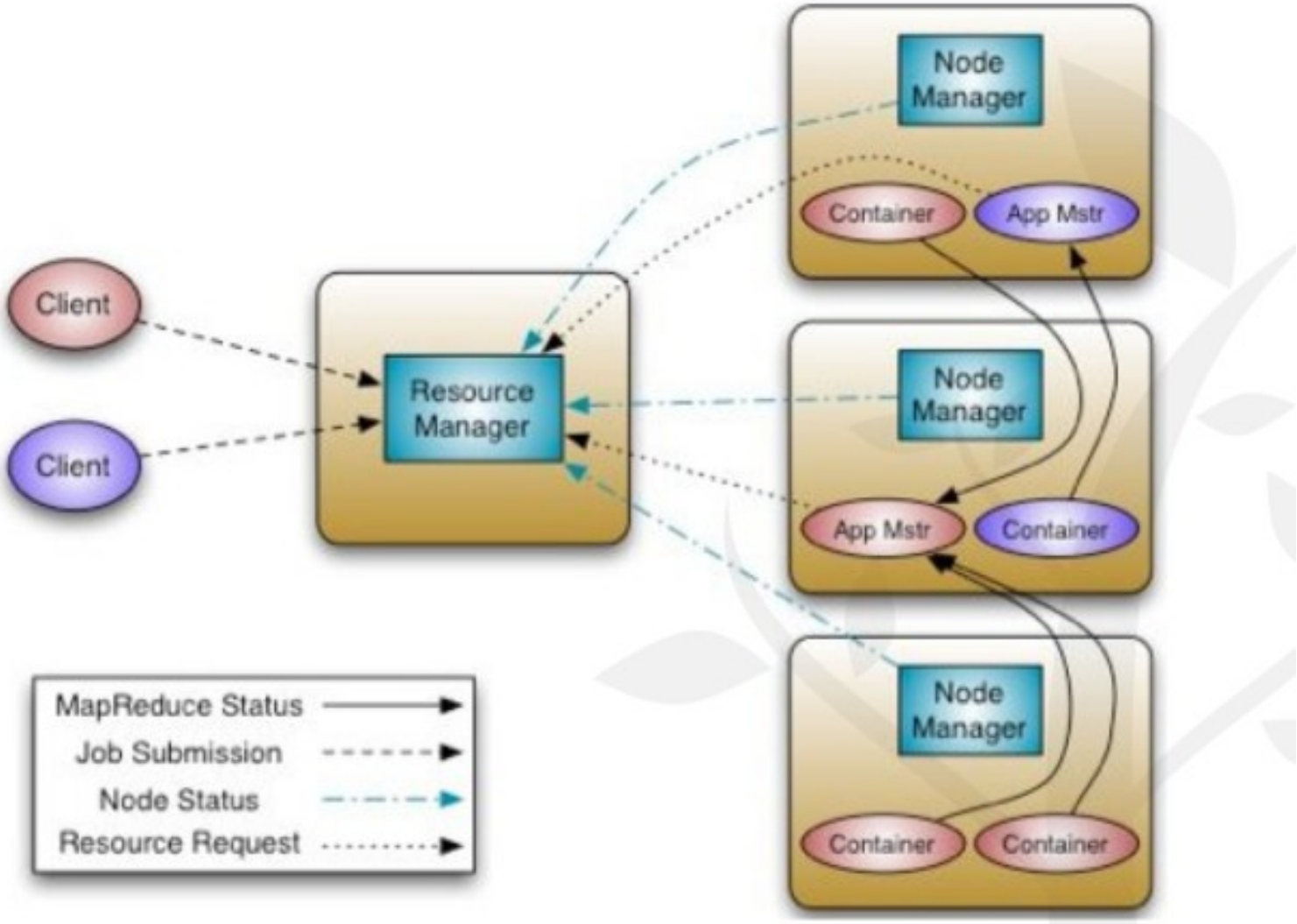
Demonios YARN (II)

Application masters

- uno por aplicación, se encarga de gestionar el ciclo de vida de la aplicación
- solicita recursos (**contenedores**) al Resource manager y ejecuta la aplicación en esos contenedores
  - en una aplicación Mapeduce en un contenedor se ejecutan tareas Map o Reduce
  - el AM se ejecuta en su propio contenedor
- trabaja con los Node managers para ejecutar y monitorizar las tareas

YARN

Elementos de control YARN



Fuente: A. Murthy, V. Vavilapalli, "Apache Hadoop YARN", Addison-Wesley, marzo 2014

YARN: propiedades configurables (I)

Múltiples propiedades configurables (fichero `yarn-site.xml`)

- `yarn.resourcemanager.hostname`: el host ejecutando el ResourceManager
- `yarn.nodemanager.aux-services`: lista de servicios auxiliares que deben implementar los NodeManagers (uno de ellos, el barajado MapReduce)
- `yarn.nodemanager.resource.memory-mb`: cantidad de memoria que puede reservarse para contenedores YARN en un nodo

YARN: propiedades configurables (II)

- `yarn.scheduler.maximum-allocation-vcores`, `yarn.scheduler.minimum-allocation-vcores`: nº máximo y mínimo de cores virtuales (threads) que pueden ser concedidos a un contenedor
- `yarn.scheduler.maximum-allocation-mb`, `yarn.scheduler.minimum-allocation-mb`: memoria máxima y mínima que puede ser concedida a un contenedor (la memoria solicitada se redondea a un múltiplo del mínimo)

Comando yarn

Permite lanzar y gestionar trabajos en YARN:

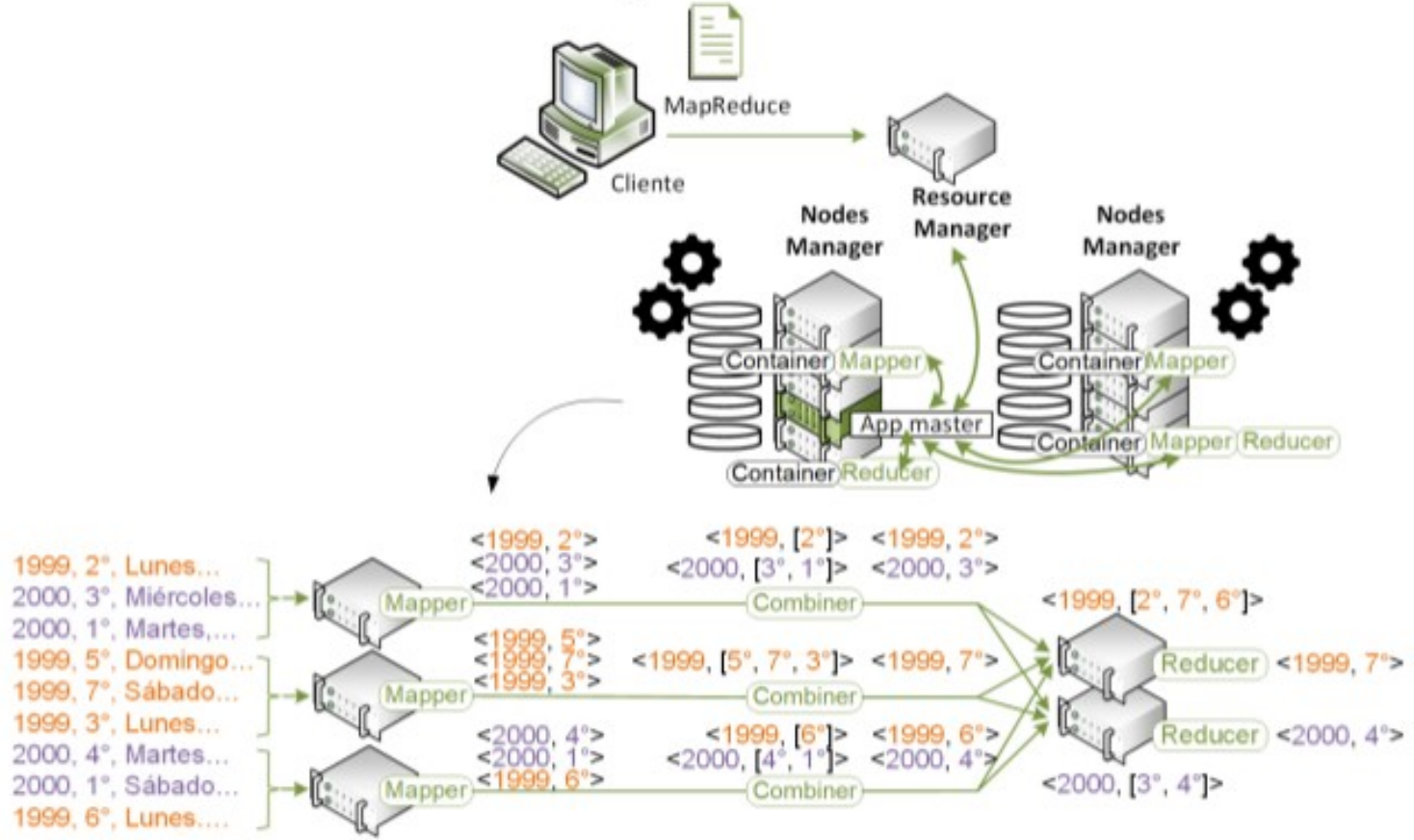
- `yarn jar`: ejecuta un fichero jar
- `yarn application`: información sobre las aplicaciones ejecutándose en YARN
- `yarn container`: información sobre los contenedores
- `yarn node`: información sobre los nodos
- `yarn top`: información sobre el uso del cluster
- `yarn rmadmin`: comandos para la administración del cluster

Más información: hadoop.apache.org/docs/stable3/hadoop-yarn/hadoop-yarn-site/YarnCommands.html

YARN

Hadoop v2: YARN

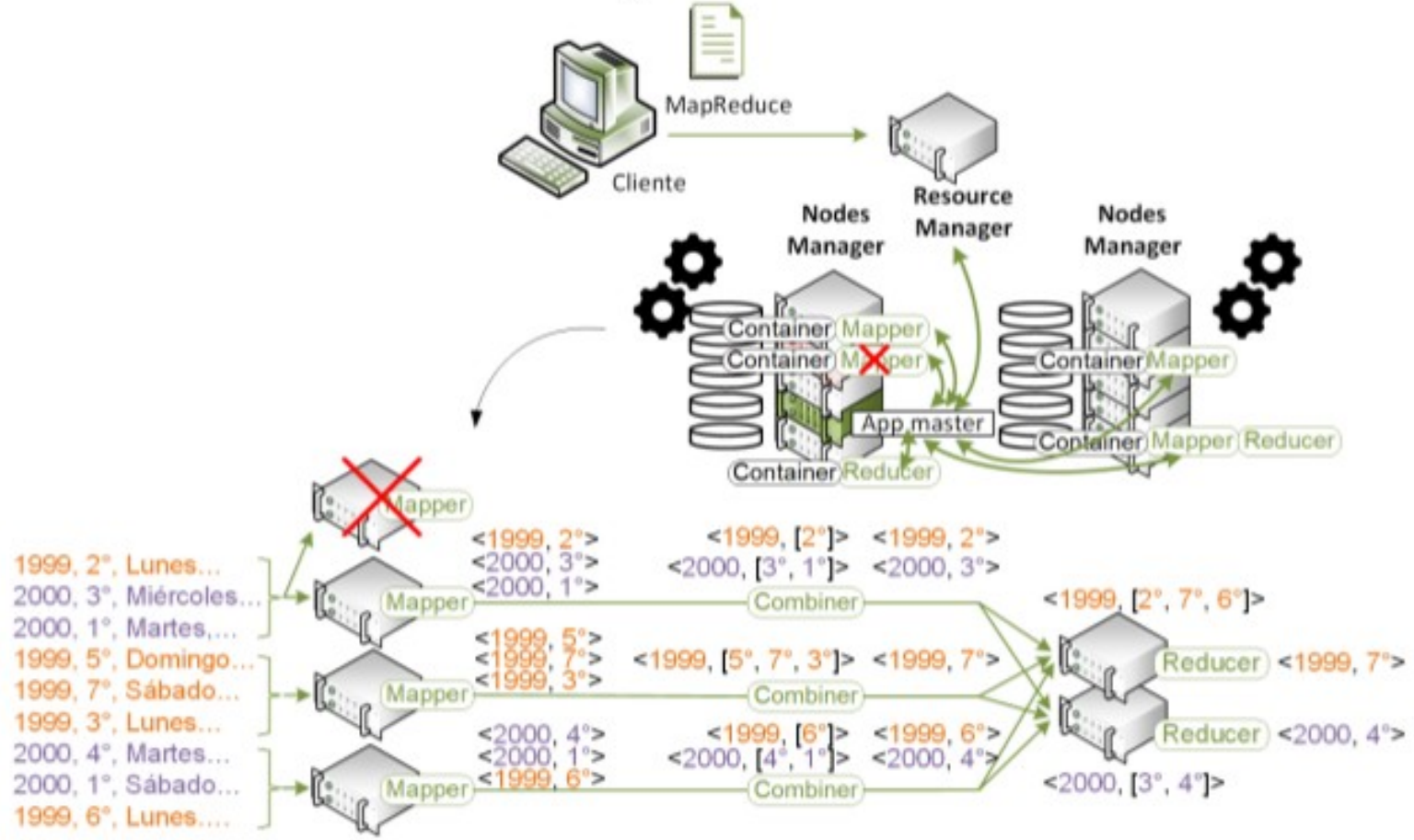
- Ejecución: Calcular la temperatura máxima de cada año



YARN

Hadoop v2: YARN

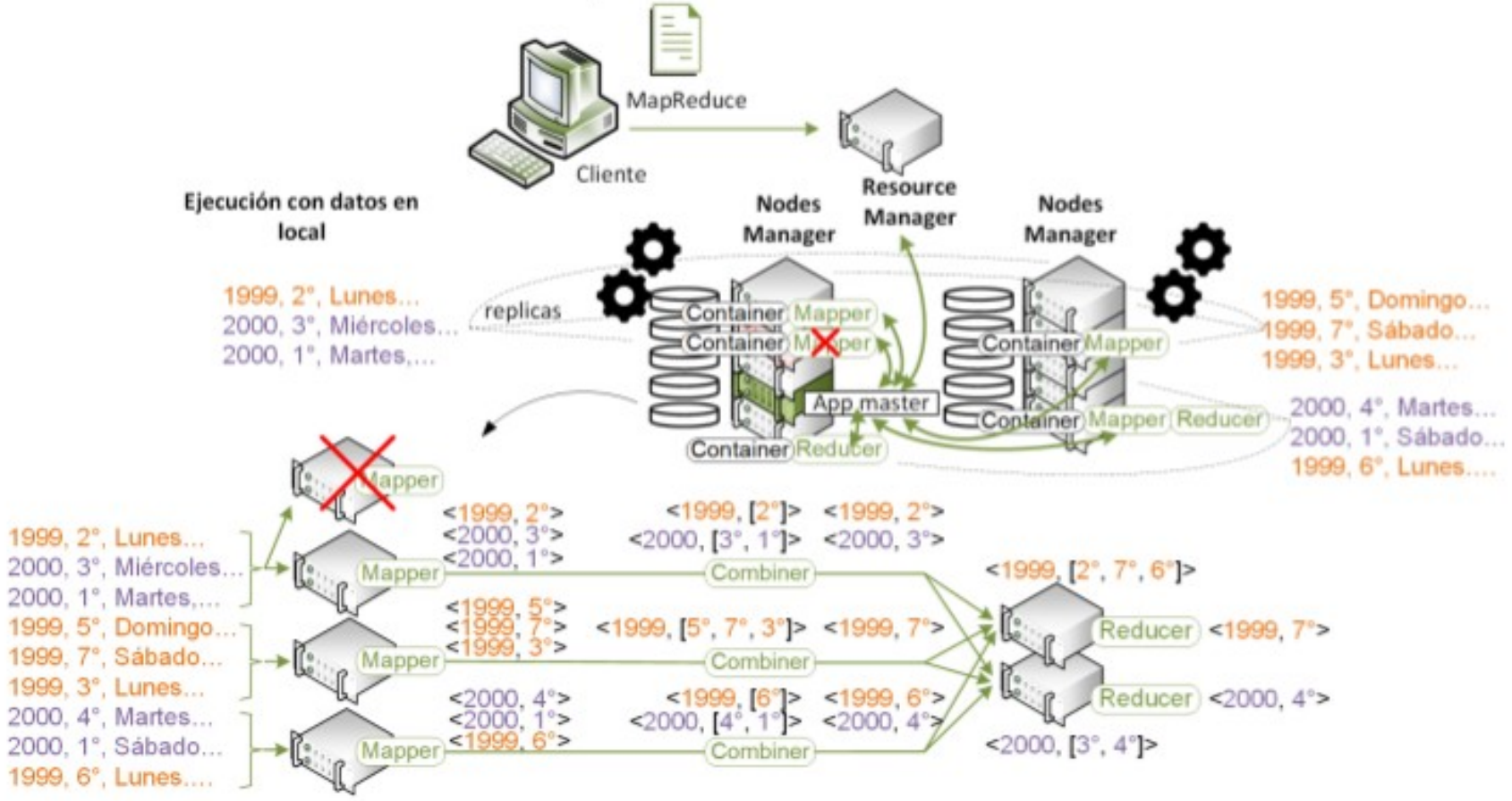
- Ejecución: Calcular la temperatura máxima de cada año



YARN

Hadoop v2: YARN

- Ejecución: Calcular la temperatura máxima de cada año



MapReduce

Mapreduce en Hadoop

Hadoop incorpora una implementación de MapReduce

- Programable en Java
- Uso de otros lenguajes mediante sockets (C++) o Streaming (Python, Ruby, etc.)

MapReduce

Mapreduce en Hadoop

Múltiples propiedades configurables (fichero `mapred-site.xml`)

- `yarn.app.mapreduce.am.resource.cpu-vcores`: cores virtuales usados por el AM
- `yarn.app.mapreduce.am.resource.mb`: cantidad de memoria requerida para el AM
- `yarn.app.mapreduce.am.command-opts`: opciones Java para el AM (p.e. el heap máximo, especificado como -Xmx1024m)
- `mapreduce.{map,reduce}.cpu.vcores`: cores virtuales para cada tarea map o reduce
- `mapreduce.{map,reduce}.memory.mb`: memoria física máxima para cada tarea map o reduce
- `mapreduce.{map,reduce}.java.opts`: opciones Java para los contenedores

Parámetros de configuración de YARN y MapReduce

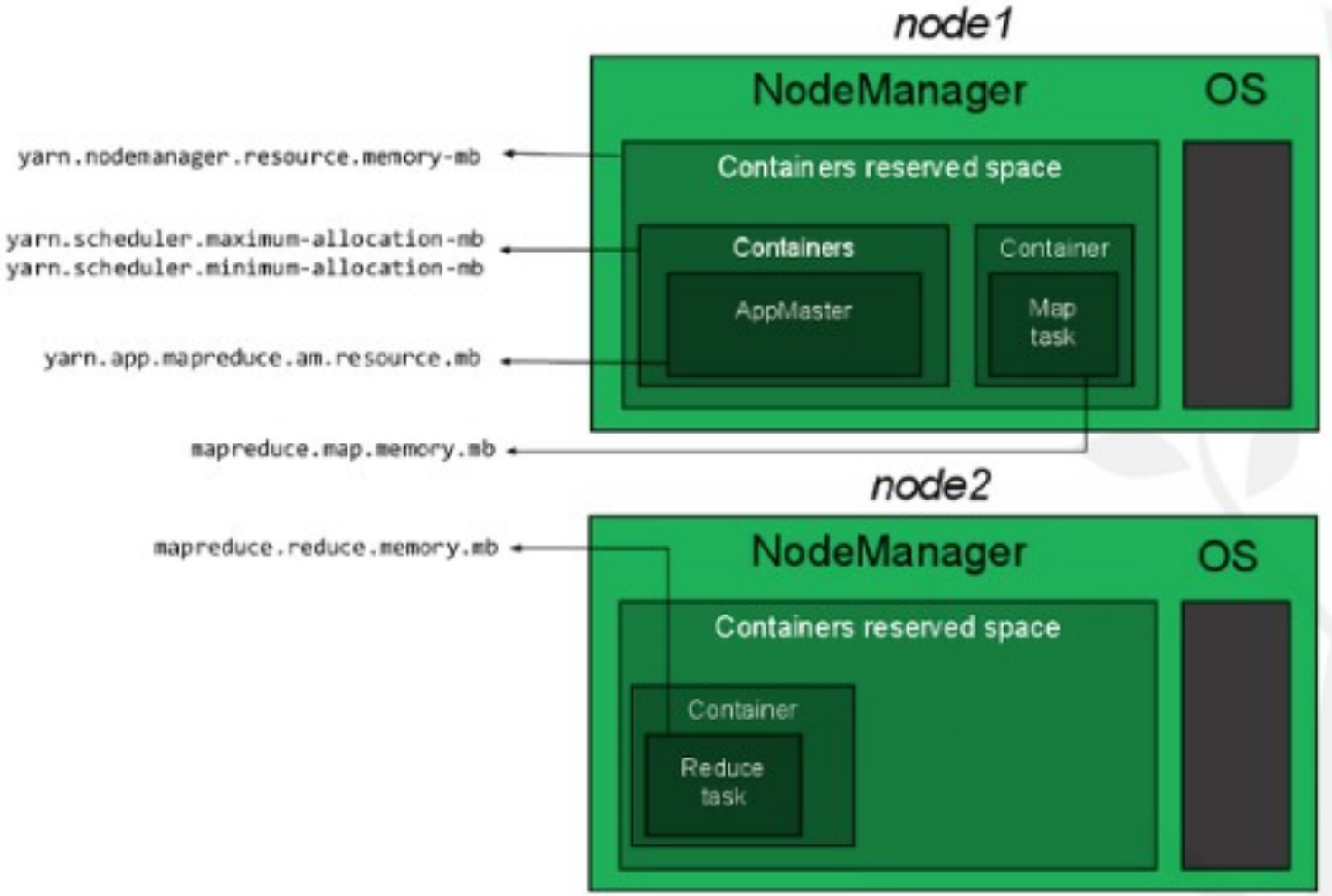
Necesitamos balancear el uso de RAM, cores y discos

- Ajustar los parámetros de Hadoop al hardware disponible

Los parámetros más sensibles son los referidos a la memoria

- yarn.scheduler.maximum-allocation-mb, yarn.scheduler.minimum-allocation-mb, yarn.nodemanager.resource.memory-mb
- yarn.app.mapreduce.am.resource.mb, yarn.app.mapreduce.am.command-opts, mapreduce.map.memory.mb, mapreduce.reduce.memory.mb, mapreduce.map.java.opts, mapreduce.reduce.java.opts

Memoria en YARN y MapReduce



Hadoop puede seleccionar el valor yarn.nodemanager.resource.memory-mb de forma automática.

Fuente: https://docs.deistercloud.com/Technology.50/Hadoop/Hadoop cluster.20.xml

Estimación de los valores

No existe una fórmula mágica para determinar los mejores valores

- Diferentes ajustes para diferentes cargas de trabajo
- Aproximaciones heurísticas como la presentada por Hortonworks
- Parte de:
  - Memoria disponible por nodo
  - Número de cores por nodo
  - Número de discos por nodo

Memoria disponible por nodo

Memoria total menos la reservada para el sistema

- La reservada será un porcentaje de la total
- Una aproximación es la de la tabla

Memoria total por nodo	Memoria para el sistema
< 8GB	1 GB
8GB - 16 GB	2 GB
24 GB	4 GB
48 GB	6 GB
64 GB - 72 GB	8 GB
96 GB	12 GB
128 GB	24 GB
> 128 GB	MemTotal/8

Número de contenedores por nodo

Función de la memoria disponible, nº de cores y nº de discos:

$$\begin{aligned} \text{Ncontenedores} = \min(&2 \times \text{Ncores}, \\ &1.8 \times \text{Ndiscos}, \\ &\text{RAMdisponible}/\text{TamañoMínimoContenedor}) \end{aligned}$$

Memoria por contenedor

La memoria mínima por contenedor va a depender:

- Del la memoria total del nodo y la memoria disponible
- Del número de contenedores por nodo

Memoria total por nodo	Tamaño Mínimo por Contenedor
< 4GB	256 MB
4 GB - 8 GB	512 MB
8 GB - 24 GB	1024 MB
> 24 GB	2048 MB

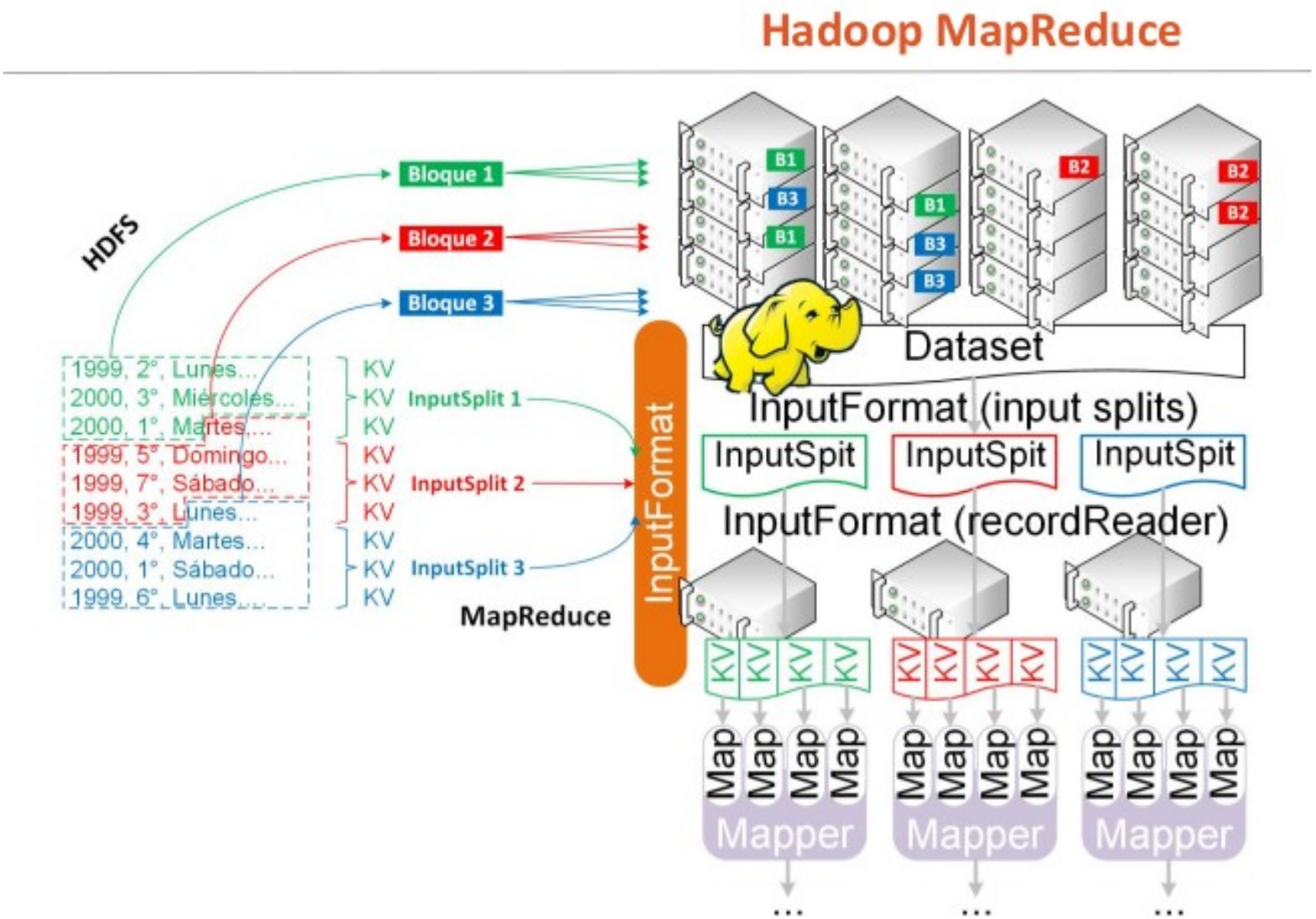
RAMporcontenedor = max(TamañoMínimoContenedor,
RAMdisponible/Ncontenedores)

MapReduce

Valores de los parámetros

Parámetro	Valor
yarn.nodemanager.resource.memory-mb	Ncontenedores × RAMporcontenedor
yarn.scheduler.minimum-allocation-mb	RAMporcontenedor
yarn.scheduler.maximum-allocation-mb	Ncontenedores × RAMporcontenedor
mapreduce.map.memory.mb	RAMporcontenedor
mapreduce.reduce.memory.mb	2 × RAMporcontenedor
mapreduce.map.java.opts	0.8 × RAMporcontenedor
mapreduce.reduce.java.opts	0.8 × 2 × RAMporcontenedor
yarn.app.mapreduce.am.resource.mb	2 × RAMporcontenedor
yarn.app.mapreduce.am.command-opts	0.8 × 2 × RAMporcontenedor

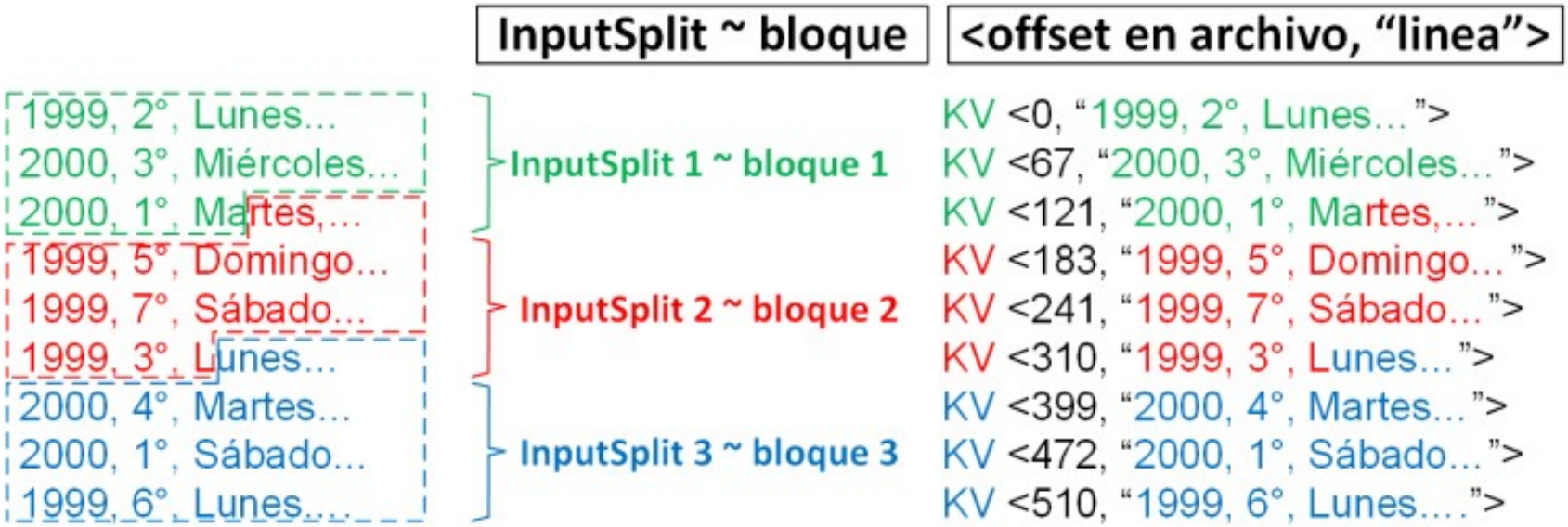
MapReduce


Hadoop MapReduce
HDFS
Bloque 1
Bloque 2
Bloque 3
1999, 2°, Lunes...
2000, 3°, Miércoles...
2000, 1°, Martes...
1999, 5°, Domingo...
1999, 7°, Sábado...
1999, 3°, Lunes...
2000, 4°, Martes...
2000, 1°, Sábado...
1999, 6°, Lunes...
KV
InputSplit 1
InputSplit 2
InputSplit 3
MapReduce
InputFormat
B1
B2
B3
Dataset
InputFormat (input splits)
InputSpit
InputSpit
InputSpit
InputFormat (recordReader)
Map
Mapper
...

MapReduce

Hadoop MapReduce

- InputFormat:
  - FileInputFormat: la entrada son archivos
    - TextInputFormat

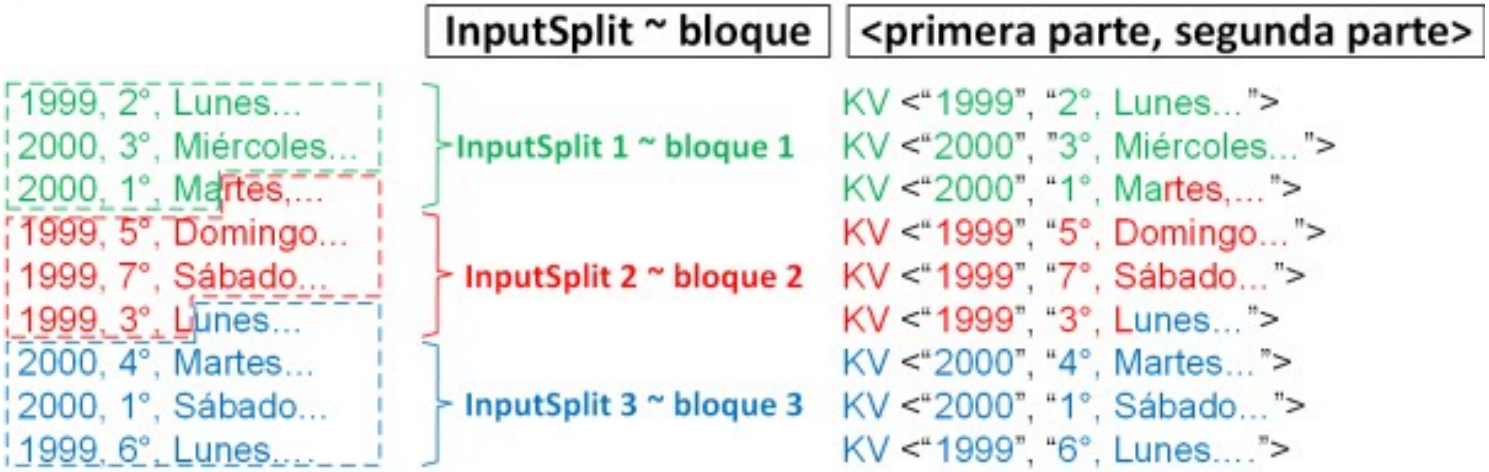


MapReduce

Hadoop MapReduce

- InputFormat:
  - FileInputFormat: la entrada son archivos
    - TextInputFormat
    - KeyValueTextInputFormat

CONFIGURACIÓN	
mapreduce.input.keyvaluelinerecordreader.key.value.separator	,



MapReduce

- InputFormat:
  - FileInputFormat: la entrada son archivos
    - TextInputFormat
    - KeyValueTextInputFormat
    - NLineInputFormat
    - SequenceFileInputFormat
    - ...
  - DBInputFormat
- Crear InputFormat:
  - https://hadoop.apache.org/docs/r3.0.1/api/org/apache/hadoop/mapreduce/InputFormat.html

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
abstract **RecordReader<K,V>**	**createRecordReader**(**InputSplit** split, **TaskAttemptContext** context) Create a record reader for a given split.
abstract **List<InputSplit>**	**getSplits**(**JobContext** context) Logically split the set of input files for the job.

MapReduce

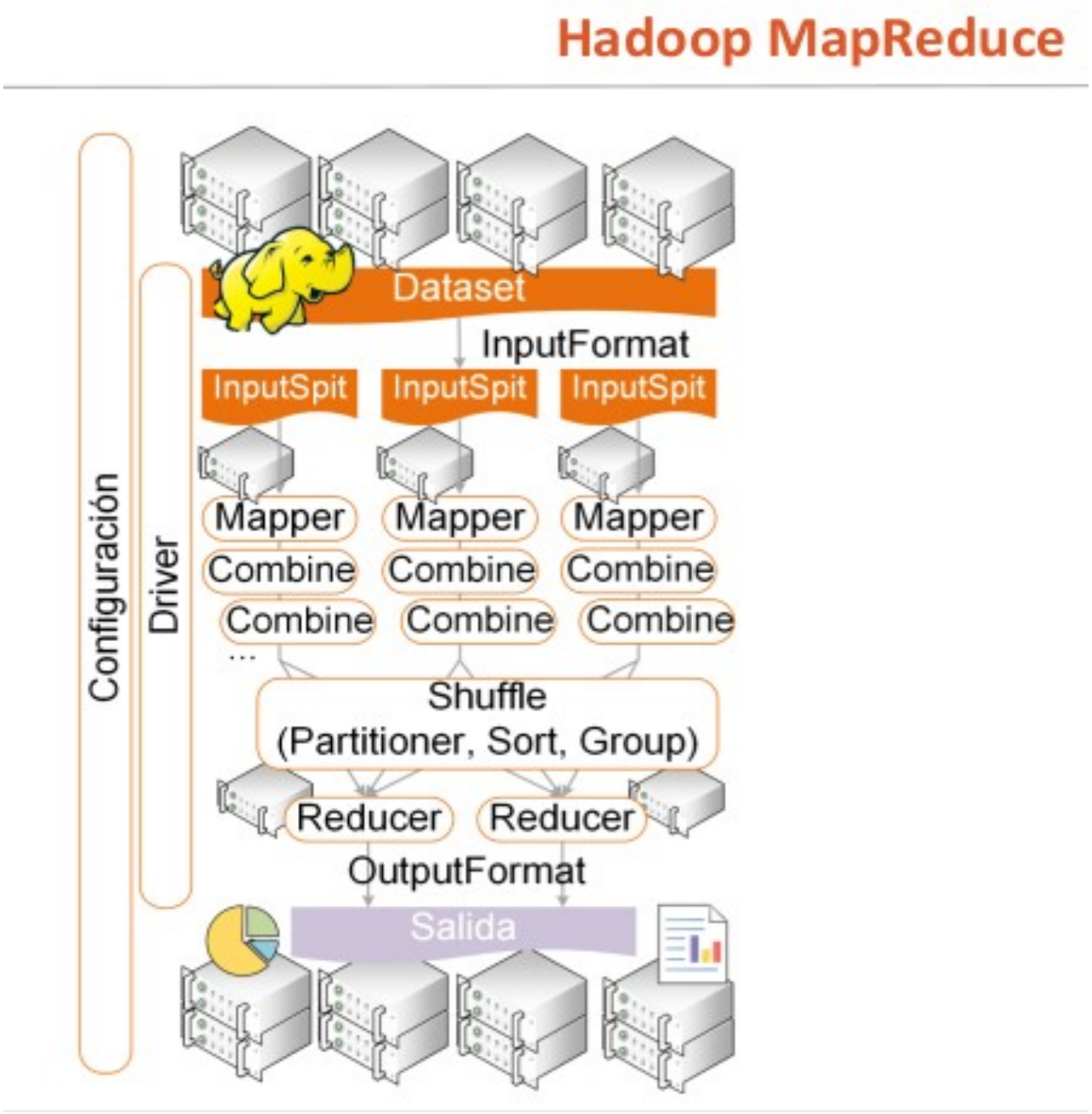
Hadoop MapReduce

- Crear InputFormat:
  - https://hadoop.apache.org/docs/r3.0.1/api/org/apache/hadoop/mapreduce/InputFormat.html
  - RecordReader

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
abstract void	**close**() Close the record reader.
abstract **KEYIN**	**getCurrentKey**() Get the current key
abstract **VALUEIN**	**getCurrentValue**() Get the current value.
abstract float	**getProgress**() The current progress of the record reader through its data.
abstract void	**initialize**(**InputSplit** split, **TaskAttemptContext** context) Called once at initialization.
abstract boolean	**nextKeyValue**() Read the next key, value pair.

MapReduce


Hadoop MapReduce
Configuración
Driver
Dataset
InputFormat
InputSpit
InputSpit
InputSpit
Mapper
Mapper
Mapper
Combine
Combine
Combine
Combine
Combine
Combine
...
Shuffle
(Partitioner, Sort, Group)
Reducer
Reducer
OutputFormat
Salida

MapReduce

Hadoop MapReduce

- Shuffle:
  - Partitioner: decide a qué Reducer se envía cada <clave, valor>
    - HashPartitioner

```
public int getPartition(K2 key, V2 value, int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
}
```

https://hadoop.apache.org/docs/r2.8.3/api/org/apache/hadoop/mapreduce/Partitioner.htm

MapReduce

- Shuffle:
  - Partitioner: decide a qué Reducer se envía cada <clave, valor>
    - KeyFieldBasedPartitioner: clave compuesta particionada por subclave(s)
      - Indicamos que se utiliza KeyFieldPartitioner:
        - Java: Job -> setPartitionerClass(KeyFieldBasedPartitioner.class) ó
        - Python: -partitioner org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedPartitioner
      - Indicamos cómo se separan las sub-claves:
        - Configuración -> mapreduce.map.output.key.field.separator
      - Indicamos cómo se particiona: utilizamos la sintaxis de unix -k pos1[, pos2,]
        - Configuración -> mapreduce.partition.keypartitioner.options

MapReduce

- Shuffle:
  - Partitioner: decide a qué Reducer se envía cada <clave, valor>
  - Sort: ordena los pares <clave, valor> en base a la clave
    - RawComparator: ordena en binario (des-serializa + compara)
    - Si la clave es WritableComparable -> ordena según la implementación (implementa RawComparator)

RawComparator

Job -> setSortComparatorClass
Configuración -> mapreduce.job.output.key.comparator.class

Modifier and Type	Method and Description
int	`compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)` Compare two objects in binary.

WritableComparable

Method Summary

Methods inherited from interface org.apache.hadoop.io.Writable

`readFields, write`

Methods inherited from interface java.lang.Comparable

`compareTo`

MapReduce

Hadoop MapReduce

- Shuffle:
  - Partitioner: decide a qué Reducer se envía cada <clave, valor>
  - Sort: ordena los pares <clave, valor> en base a la clave
  - Group: agrupa los valores para tener <clave, [lista valores]>

RawComparator

Job -> setGroupingComparatorClass
Configuración -> mapreduce.job.output.key.comparator.class

Modifier and Type	Method and Description
int	`compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)` Compare two objects in binary.

Method Summary

WritableComparable

Methods inherited from interface org.apache.hadoop.io.Writable

`readFields, write`

Methods inherited from interface java.lang.Comparable

`compareTo`

MapReduce

- Serialización de datos
  - Writable
    - IntWritable
    - FloatWritable
    - LongWritable
    - Text
    - NullWritable
    - ...

All Methods | Instance Methods | Abstract Methods | **Writable**

Modifier and Type	Method and Description
void	**readFields**(DataInput in) Deserialize the fields of this object from in.
void	**write**(DataOutput out) Serialize the fields of this object to out.

Method Summary **WritableComparable**

Methods inherited from interface org.apache.hadoop.io.Writable

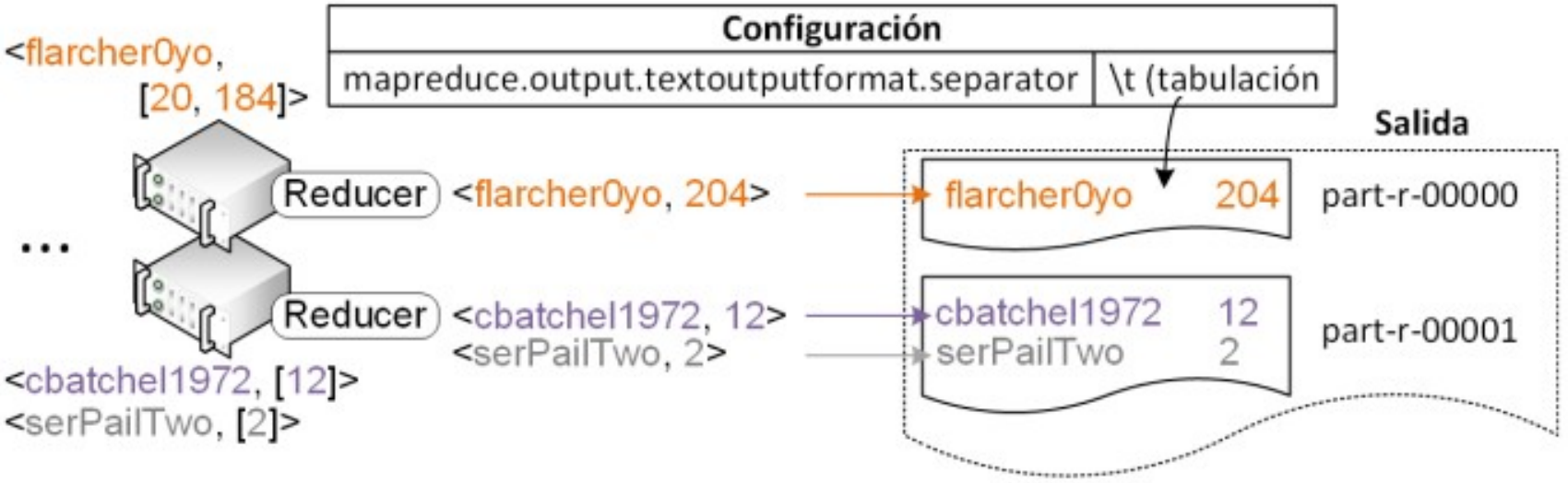
readFields, write

Methods inherited from interface java.lang.Comparable

compareTo

MapReduce

- OutputFormat
  - FileOutputFormat: la salida son archivos
    - TextOutputFormat



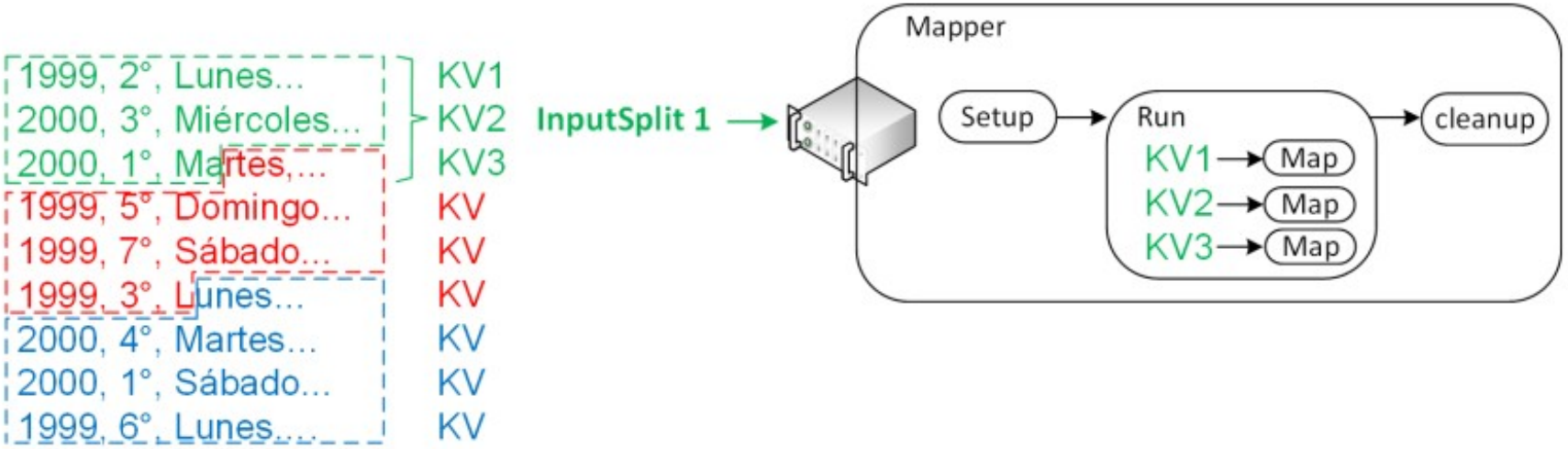
MapReduce

- OutputFormat
  - FileOutputFormat: la salida son archivos
    - TextOutputFormat
    - SequenceFileOutputFormat
  - DBOutputFormat
  - ...

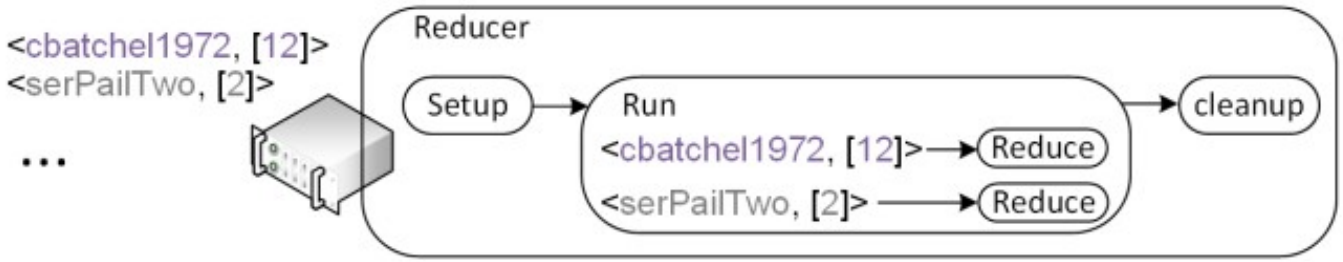
MapReduce

Hadoop MapReduce

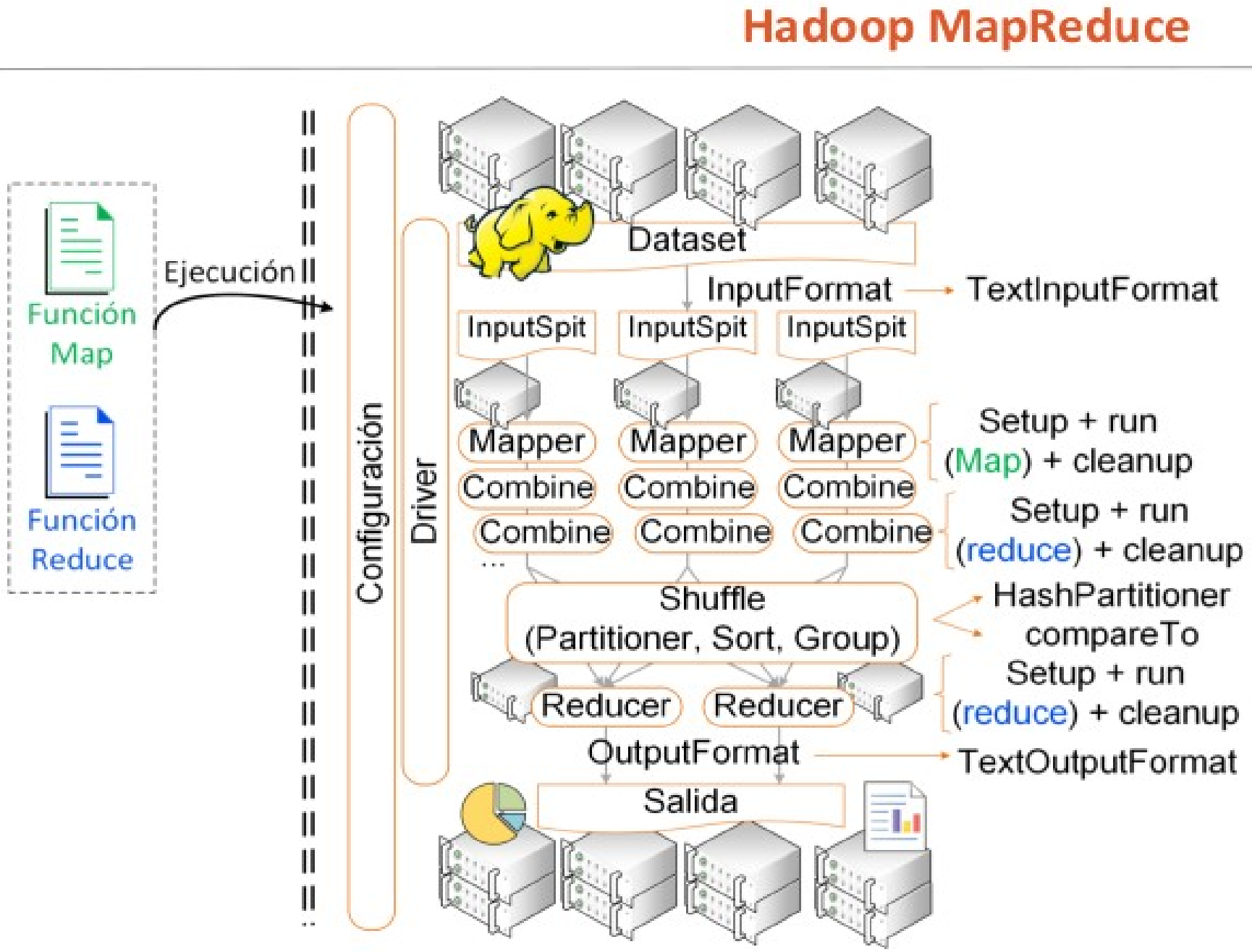
- Mapper



- Reducer



MapReduce


Hadoop MapReduce
Función Map
Función Reduce
Ejecución
Configuración
Driver
Dataset
InputFormat
TextInputFormat
InputSpit
InputSpit
InputSpit
Mapper
Mapper
Mapper
Setup + run (Map) + cleanup
Combine
Combine
Combine
Combine
Combine
Combine
Setup + run (reduce) + cleanup
Shuffle (Partitioner, Sort, Group)
HashPartitioner
compareTo
Reducer
Reducer
Setup + run (reduce) + cleanup
OutputFormat
TextOutputFormat
Salida