HBase

Repaso de lo visto hasta ahora

¿Por qué HBase?

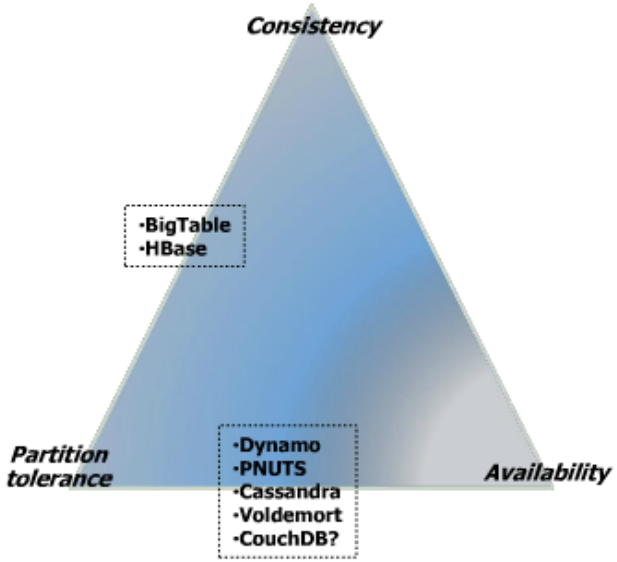
- Forma parte del ecosistema Hadoop (usa HDFS, **no MapReduce**)
- Puede llegar a sustituir ciertos despliegues en RDBMS
- Escala horizontalmente apoyándose en Hadoop (TBs – PBs)
- Soporta modelos de datos flexibles
- Soporta acceso aleatorio de registros y lectura/escritura de registros
- El reparto de datos (Sharding) es automático sin tener que realizar tareas específicas (como en el caso de RDBMS)

Teorema CAP

Un sistema **distribuido** solo puede cumplir **dos** de las siguientes tres propiedades:

- **Consistencia**: todos los clientes ven siempre los mismos datos. La base de datos es veraz. Puede ser Fuerte (atomic and immediate), Secuencial, Casual, Eventual o Débil.
- **Disponibilidad**: el sistema está siempre disponible y, por lo tanto, también los datos
- **Tolerancia** (con respecto a las particiones): el sistema funciona aun cuando un fallo de red cree dos grupos desconectados (la red puede llegar a perder algunos mensajes enviados de un nodo a otro). Esta propiedad todavía es discutible

CAP Positioning



HBase is Eventually Consistent and Implements Consistency and Partition Tolerance

(e.g. Si un Region Server cae el sistema se recupera automáticamente pero los datos pueden no estar disponible por un período de tiempo)

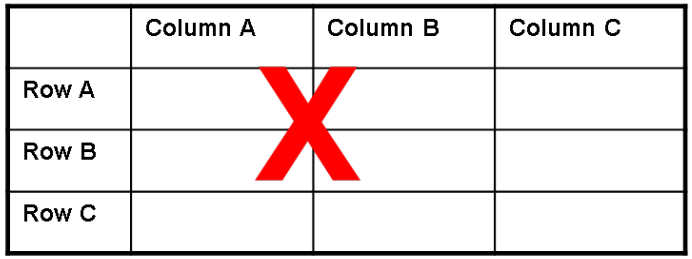
¿Cómo es HBASE?

- En algunos casos se define como una base de datos orientada a columnas… pero no es del todo exacto

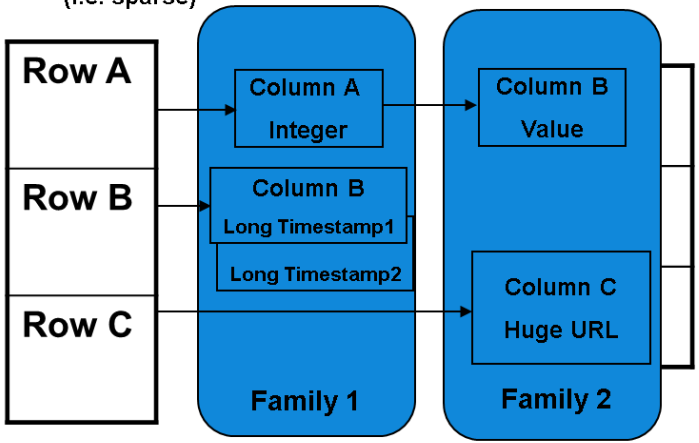
- Es más correcto definirla como una colección (mapa) de parejas clave-valor multidimensional

http://en.wikipedia.org/wiki/Associative_array

http://research.google.com/archive/bigtable-osdi06.pdf



Note: Column Families contain Columns with time stamped versions. Columns only exist when inserted (i.e. sparse)



*See "HBase The Definitive Guide" by Lars George

Modelo de datos de HBase

- Un modelo de datos se apoya en una o varias tablas
- Las Tablas contienen Column-Families, agrupaciones lógicas de información.
- Las columnas pueden tener múltiples versiones (con su timestamp correspondiente) cada una de ellas en su propia celda
  - El hecho de contar con distintas versiones en el tiempo hace pensar en contar con tablas en tres dimensiones
- Los registros asocian distintos valores de las columnas en una unidad *lógica*
- Las Column-Families pueden ser modificadas en vuelo, añadiendo más columnas. En realidad, una columna como tal es una asociación de una Column Family + un Cualificador: Column => Column Family + Qualifier
  - e.g. SocialMediaData:{'LN':'Smith'}
  - e.g. SocialMediaData: {'FN':'John', 'LN':'Smith'} … SocialMediaData: {'FN':'John', 'LN':'Smith', 'Email':john.smith@ibm.com'}
- Las Columnas pueden ser NULL, en cuyo caso no son almacenadas

Por ejemplo

Partiendo de este ejemplo de registros en una base de datos relacional...

Customer Number	Last Name	First Name	Account Number	Type of Access	Timestamp
01234	Smith	John	abcd1234	Checking	20120118
01235	Johnson	Michael	wxyz1234	Checking	20120118
01235	Johnson	Michael	aabb1234	Checking	20111123

Visión lógica de los “registros” en HBase

Número de clave	Valores: familyname:{columna:valor}
01234	info: {‘lastname’: ‘Smith’, ‘firstname’: ‘John’} acct: {‘checking’: ‘abcd1234’}
01235	info: {‘lastname’: ‘Johnson’, ‘firstname’: ‘Michael’} acct: {‘checking’: ‘wxyz1234’ @ts=20120118, ‘checking’: ‘aabb1234’ @ts=20121123}

HBase versus RDBMS

	HBase	RDBMS
Data Layout	Un mapa multidimensional disperso, distribuido y persistente	Orientado a fila o a columna
Transacciones	Soporte ACID a nivel de una única fila	Si
Lenguaje de consulta	get/put/scan only unless combinada con Hive o con otras tecnologías	SQL
Seguridad	Autenticación y Autorización	Autenticación y Autorización
Indexación	Solo en el valor de la clave de fila o a través de una table especial	Si
Rendimiento	Millones de consultas por Segundo (¿?)	Millones de consultas por Segundo (¿?)
Tamaño máximo de base de datos	PBs (¿?)	Miles de TBs (¿?)

ACID Properties

- **Atomicidad:**

  Si una operación consiste en una serie de pasos, o bien todos ellos se ejecutan o bien ninguno. Es decir, las transacciones son completas.

- **Consistencia: (Integridad)**

  Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.

- **Aislamiento:**

  Esta propiedad asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error. Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes. El aislamiento puede alcanzarse en distintos niveles, y es una propiedad esencial a la hora de seleccionar la base de datos.

- **Durabilidad: (Persistencia)**

  Esta propiedad asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema y que de esta forma los datos sobreviven de alguna manera.

HBASE y las propiedades ACID

- **Atomicidad**
  - Las lecturas y escrituras de datos se realizan en un único servidor (el Region Server)
  - Todos los clientes dialogan con el Region Server para obtener los datos
  - Aporta atomicidad a nivel de fila/registro
- **Consistencia y aislamiento**
  - Todas las filas devueltas por una llamada devuelven una fila completa que existió en algún momento del tiempo en la tabla
  - Un barrido (scan) no es una visión consistente de una tabla. Cualquier fila devuelta por un barrido mostrará una visión consistente (la correspondiente a que la fila existió en algún momento en el tiempo)
- **Durabilidad**
  - Todos los datos visibles son también durables. Una lectura no puede devolver datos que no han persistido en disco
- **Concurrencia**
  - HBase realiza el bloqueo antes de proceder a la escritura y lo libera después de la escritura
  - El usuario puede controlar el bloqueo manualmente

http://hbase.apache.org/acid-semantics.html

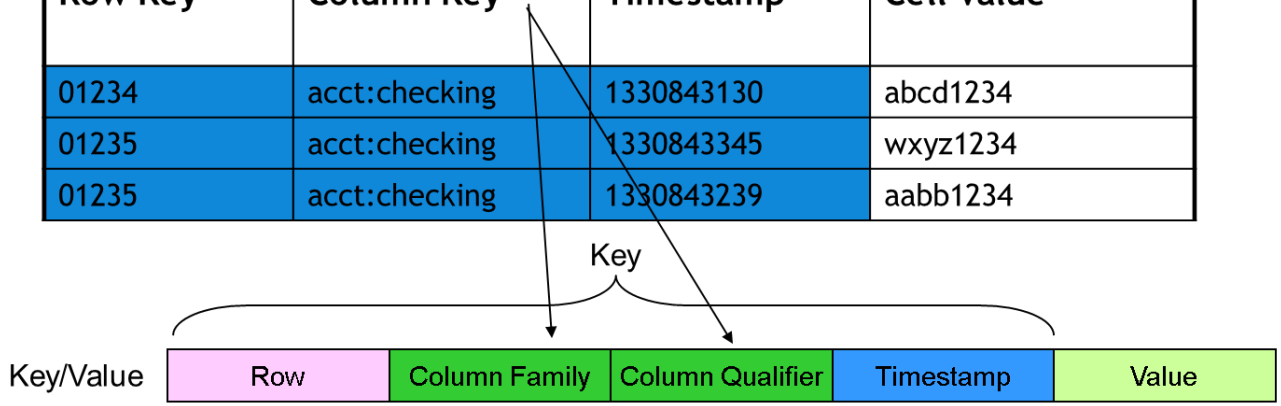
Pero los datos se almacenan físicamente de una forma especial…

info **Column Family**

Row Key	Column Key	Timestamp	Cell Value
01234	info:fname	1330843130	John
01234	info:lname	1330843130	Smith
01235	info:fname	1330843345	Michael
01235	info:lname	1330843345	Johnson

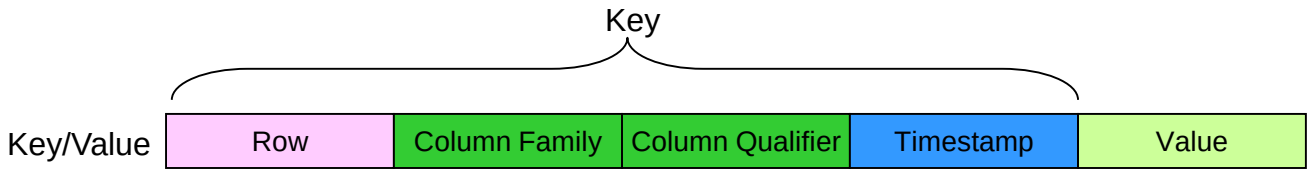
acct **Column Family**

Row Key	Column Key	Timestamp	Cell Value
01234	acct:checking	1330843130	abcd1234
01235	acct:checking	1330843345	wxyz1234
01235	acct:checking	1330843239	aabb1234

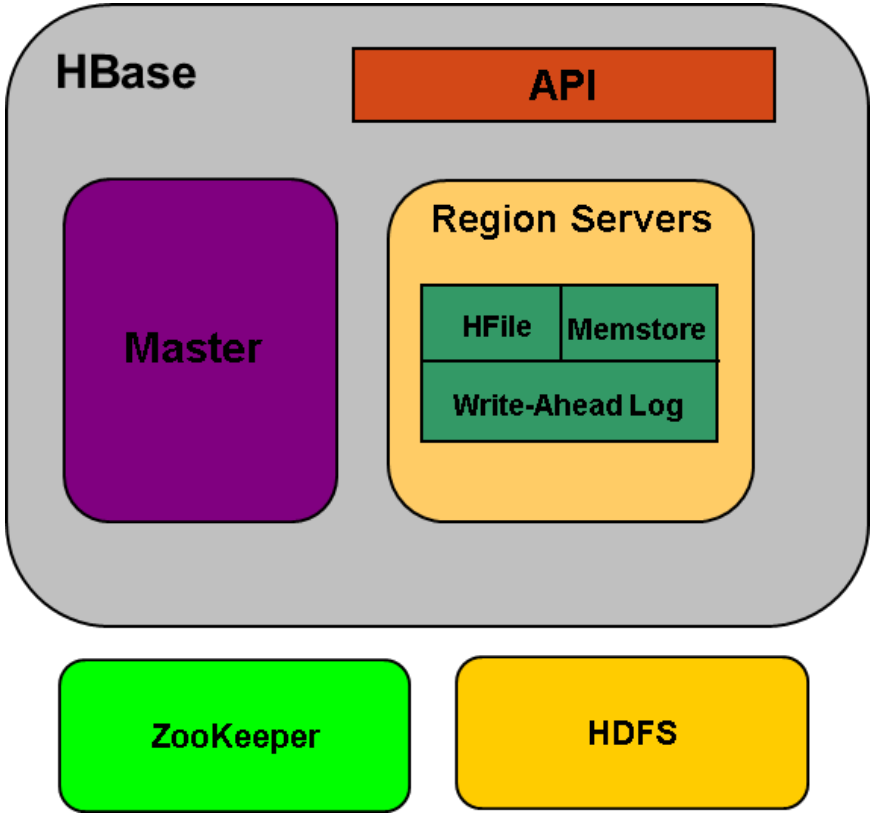


Más sobre el modelo de datos

- No hay esquema para una tabla Hbase (tal y como interpretamos en RDBMS)
  - Excepto que hay que declarar las Familias de Columnas
    - Ya que determina la organización física en disco
  - Cada fila puede tener entonces diferentes juegos de Columnas
- Recordemos que describimos Hbase como una base de datos clave-valor
  - La clave se genera con la clave de la columna, la Column Family, su Nombre y un timestamp
- Cada pareja de clave-valor es versionada
  - Puede ser un timestamp o un entero
- Todos los datos son arrays de bytes, incluyendo el nombre de la tabla, los nombres de la Familia de Columnas y el nombre de las columnas



Arquitectura de alto nivel



Componentes: Definiciones y Roles

- **Region**
  - Un subset de las filas de las Tablas
  - Repartidas automáticamente cuando crecen mucho
- **Region Server**
  - Almacena las Tablas, ejecuta las lecturas y el "buffer" de las escrituras
  - Los Clientes se conectan directamente a ellos para las lecturas y las escrituras
- **Master**
  - Responsable de la coordinación de los Region Servers(s)
  - Detecta el estado de los Region Servers y se encarga de balancear la carga entre ellos
  - Asigna las Regiones a los Region Servers
  - Puede haber más de uno
    - Pero no cooperan entre ellos: hay un primario y uno o más servidores de backup
  - El Master se encarga de gestionar las operaciones del cluster:
    - Asigna regiones, maneja el balanceo de carga y las divisiones de las regiones
  - No forma parte del proceso de lectura/escritura
  - ZooKeeper se encarga de mantener su alta disponibilidad y la de los servidores de backup

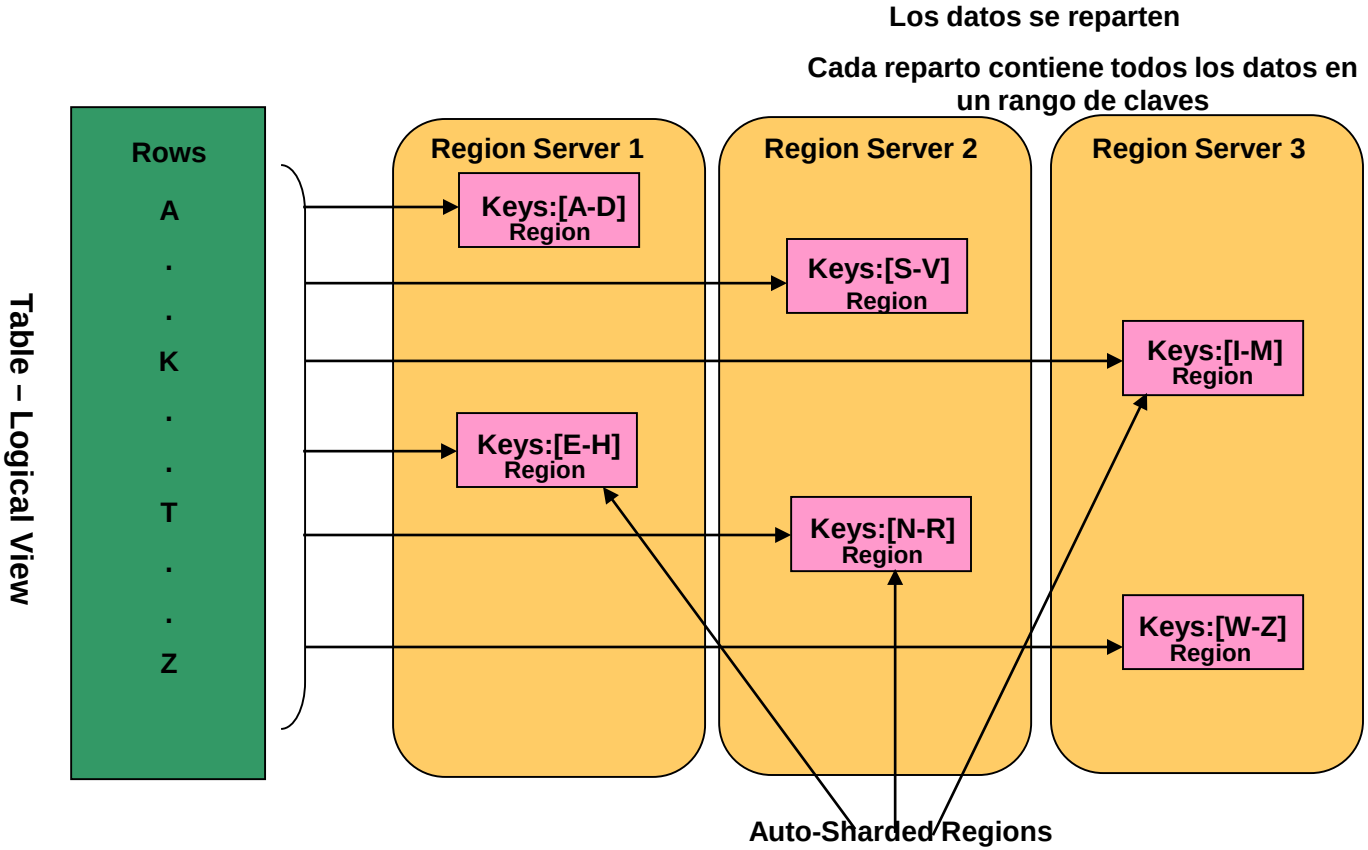
Componentes: Definiciones y Roles

- Zookeeper
  - Componente crítico ya habitual en el entorno de Hadoop
  - Asegura que hay un Master activo
  - Suministra información sobre la localización de las Regiones
  - Registra los Region Servers
  - Maneja las caídas de los Region y del Master server

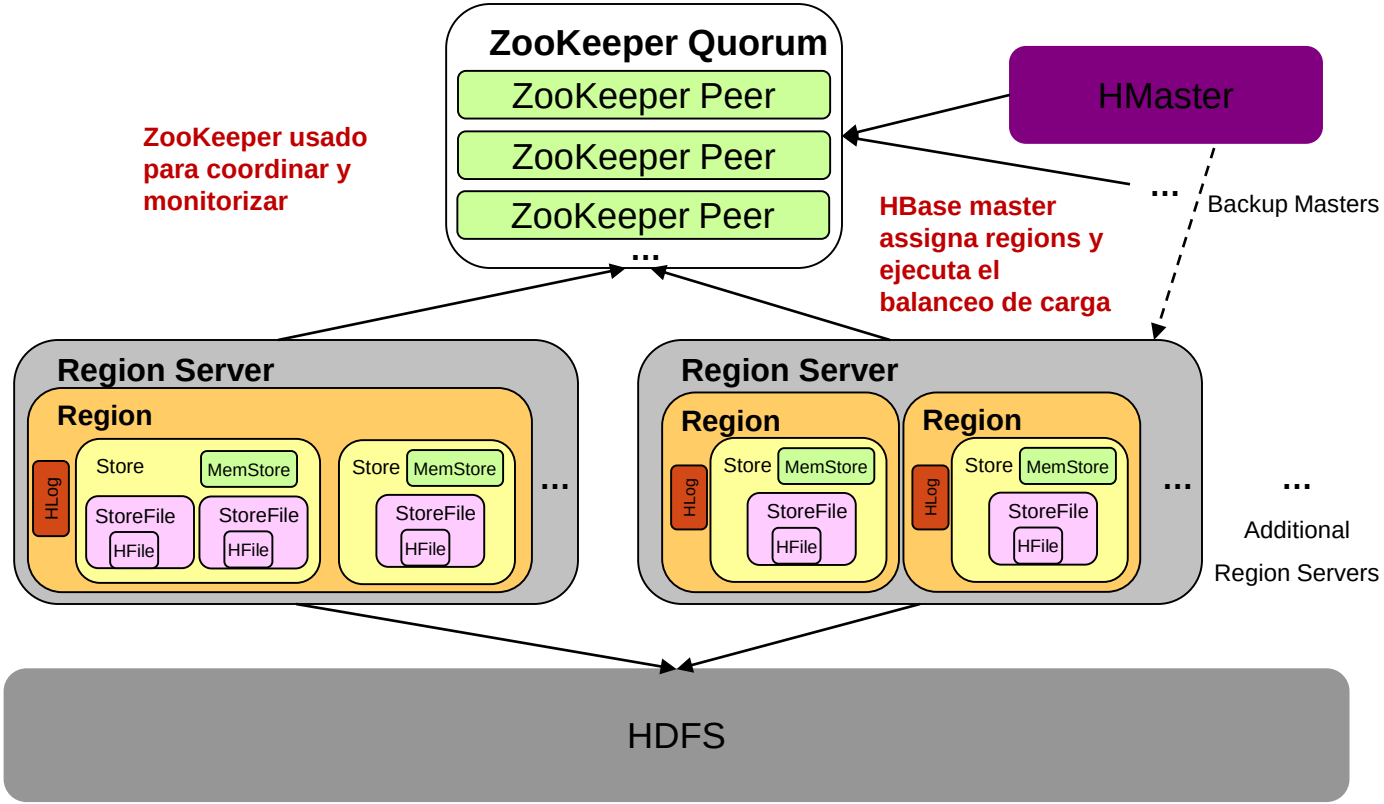
Auto-Sharding y escalado in HBase

- Sharding es automático en HBase
  - La unidad básica de escalabilidad y balanceo en Hbase es la Region
  - Inicialmente hay una región por Tabla
  - Sin embargo, cuando la tabla supera un valor límite es automáticamente dividida por la mitad (es lo que denominamos auto-sharding)
  - A medida que los datos almacenados crecen, las tablas se componen de múltiples regions y estas estarán servidas por los Region Servers
  - Los Region Servers pueden servir múltiples Regions
- A medida que los datos crezcan, el administrador puede necesitar añadir más Region Servers al cluster para mejorar o mantener el rendimiento
- El Master se encarga de asignar las regiones a los nuevos Region Servers

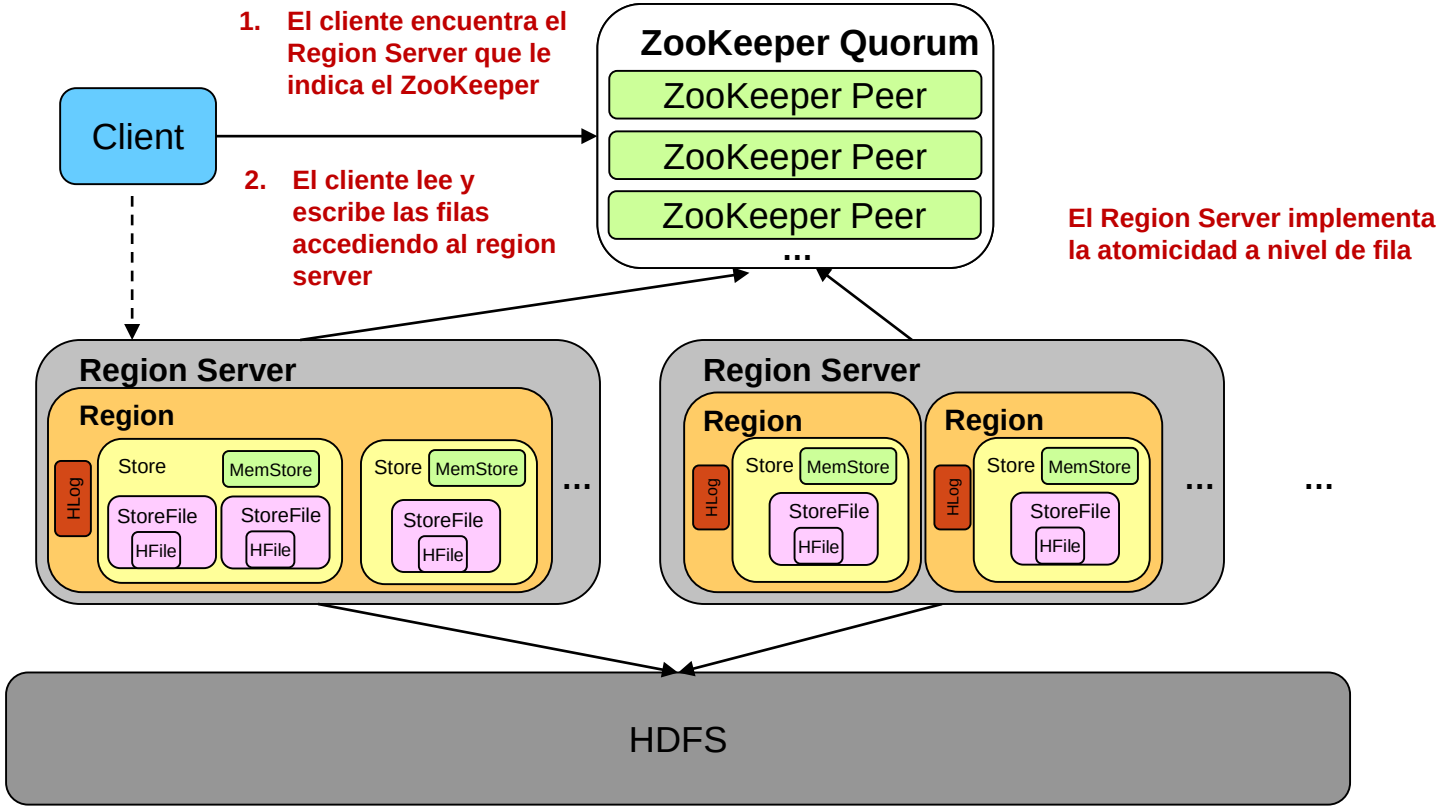
¿Cómo se almacenan?



Visión más completa



Acceso de clientes

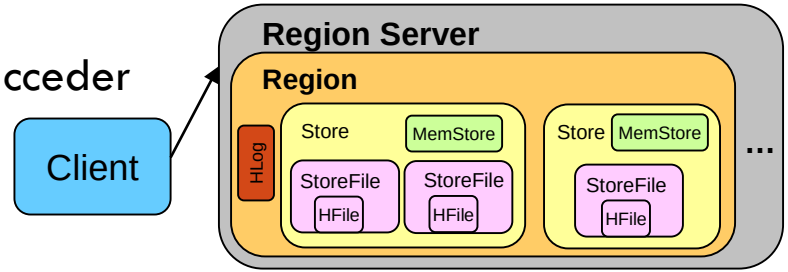


¿Cómo trabaja una aplicación cliente?

- El primer paso es conocer la situación del cluster. Hay una tabla especial de HBase que se llama **META** y que contiene toda la información de las regiones que hay en el cluster. Zookeeper almacena la localización de esa tabla.
- Lo primero que hace un cliente es preguntar a Zookeeper qué nodo tiene la tabla META
- La aplicación cliente acude a ese nodo y pregunta a qué region server corresponde el registro al que quiere acceder. El cliente guarda (cachea) esa información (el servidor que aloja la META y la información que contiene). Esa información la irá consultando a medida que necesite acceder a una fila (registro).
- Gracias a esa información la aplicación cliente va directa siempre al o a los Region Server correspondientes.
- En general la aplicación cliente no tendrá que recurrir a refrescar la tabla META a menos que obtenga algún error (por ejemplo un Region Server ha caído y lo ha reemplazado otro nodo)

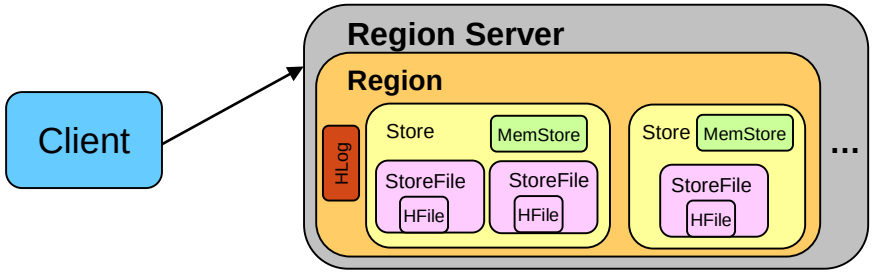
¿Cómo se hacen las escrituras o actualizaciones?

- Para escribir una fila
  - El cliente contacta el Region Server que le corresponde
  - La fila es escrita primero en el Write Ahead Log (WAL). Un log en disco que ayuda a mantener la persistencia y que se utiliza para casos de recuperación
  - Entonces la fila se escribe en una cache de escritura en memoria MemStore (write cache). Los datos en el MemStore son ordenados y se devuelve ya un acuse de recibo a la aplicación. Hay un MemStore por Column Family
  - Cuando el MemStore se llena es persistido a un HFile en disco. Una Column Family puede necesitar múltiples HFiles
  - Los HFiles están indexados para poder acceder al dato sin tener que leer todo el fichero



¿Y cómo se lee?

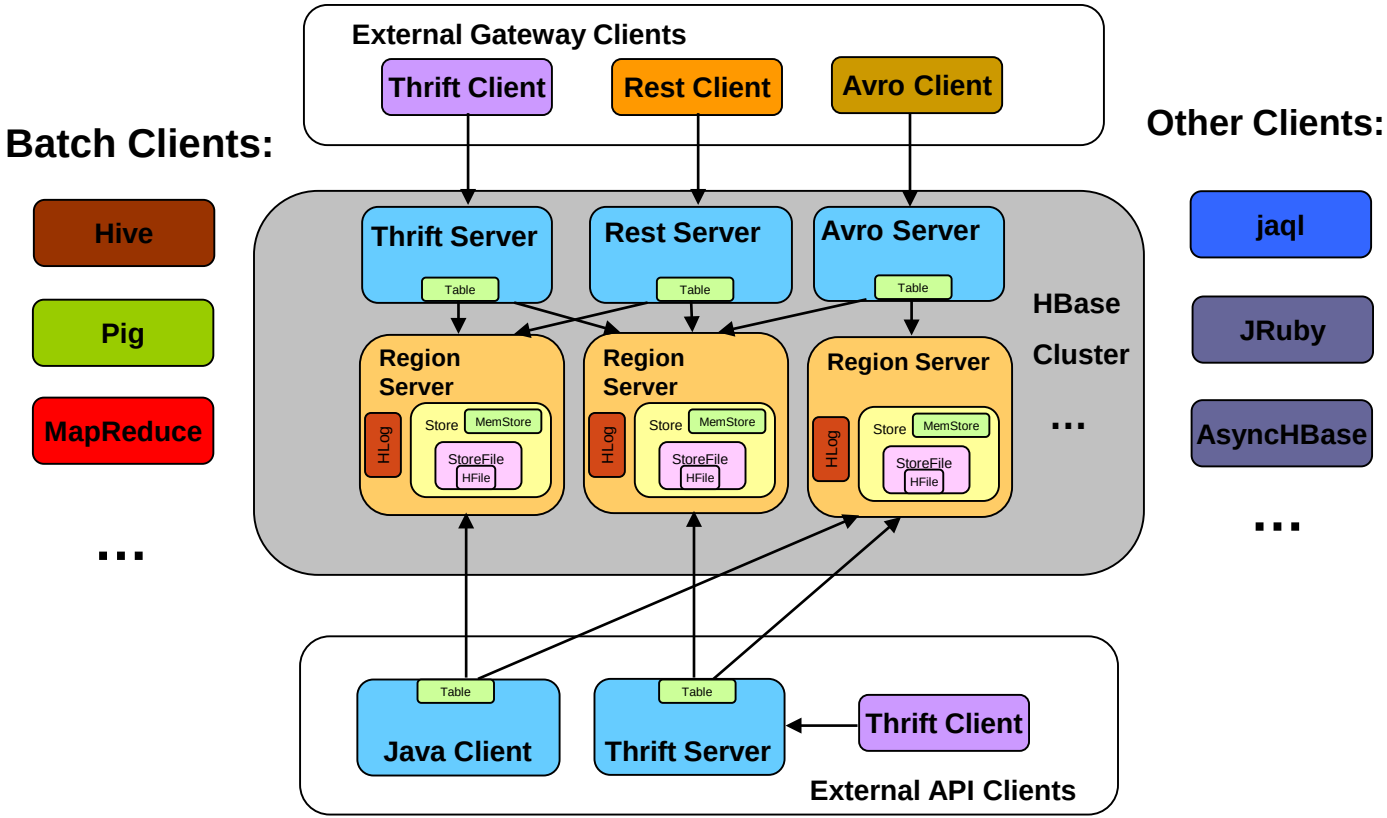
- Los bloques de dato leídos de Hfiles son cacheados en el BlockCache
- El BlockCache es configurable (el tamaño por defecto es de 64KB)
- Un tamaño más pequeño es más conveniente para lecturas aleatorias
  - Tamaños más pequeños exigen mayores índices y consume más RAM
- Un tamaño más grande es más conveniente para lecturas secuenciales



HBase Major/Minor Compactaciones

- Una Region puede llegar a contener muchos HFiles
  - Para obtener el resultado de una lectura se necesitan consultar y "combinar" todos los Hfiles con lo que se hace necesario optimizar ese proceso
- Los HFiles se combinan en majors o minors compactaciones
- Una minor compactación combina registros de varios HFiles en un HFile
  - Pueden ser configurados con hbase.hstore.compactionThreshold
  - Los clientes se bloquean si se supera el límite del MemStore
- Una major compaction combina todos los HFiles de una Region en un único HFile
  - Pueden ser configurados con hbase.hregion.majorcompaction
  - Generalmente se hace una vez al día
  - Las versiones expiradas o borradas son quitadas durante una major compaction

Arquitecturas de clientes de HBase



Opciones de conectividad HBase

- Native Java Client / API
  - Get, Scan, Put, Delete classes, and HTable for read/write
- Cliente no Java
  - Apahe Thrift (Ruby, PHP, Java, C++, Python, …) – http://thrift.apache.org/
  - REST server – Web access to HBase
  - Apache Avro (Ruby, PHP, Java, C++, Python, …) – http://avro.apache.org/
- HBase Shell
  - JRuby's IRB - supports put, delete, get, scan
  - También permite opciones administrativas
- Pig – Apache Pig Latin
- Hive
- MapReduce – Si se desea proceso tipo batch de datos masivos

HBase es un almacen de datos tolerante a fallos

- Si un Region Server cae
  - El Master asigna un nuevo Region Server a cada Region que atendía el Region Server que ha caido
- Si cae el Master Server
  - Zookeeper se encarga de que un servidor backup tome el role
- Los clientes encuentran el Master y el Region Server a través de ZooKeeper, que es fault-tolerant por si mismo
- Replicación de datos
  - HBase por si misma no replica datos (aunque se soporta la replicación a otro cluster)
  - HDFS (por debajo) puede aportar replicación
  - Esto también significa que puede necesitar un DFS (+ HDFS or GPFS) para ser fault-tolerant
- Detección de caídas
  - Zookeeper se utiliza para identificar las caídas de los Region Servers y de los Master Servers

Replicación de datos entre Centros

- La replicación se hace enviando el WAL
  - El log se replica en el Centro remoto
- Consistencia de la replicación:
  - Aunque hay múltiples logs (uno por Region Server), hay solo uno que almacene la clave con lo que las "transacciones" son consistentes después de la replicación
  - La secuencia de actualizaciones a una clave se serializan siempre en un Region Server y el log
  - Así la replicación preserva la consistencia
- El estado de la replicación se mantiene en Zookeeper

Otras opciones adicionales en HBase: Coprocessors

- Basados en el mismo concepto de la BigTable de Google
- Disponible desde HBase 0.92
- Permite ejecutar código dentro del Region Server y el Master Server
- "Triggers"
  - Las funciones se ejecutan cuando ocurran ciertos eventos
- "Procedimientos almacenados"
  - Pueden ser invocados en cualquier momento por el cliente
- Algunos otras funciones que se pueden ejecutar como co-processors
  - Security access control
  - Funciones de Agregación
    - sum, avg, …
  - Indices secundarios
  - …alguno más

Integración con MapReduce

- TableInputFormat
  - Convierte los datos de Hbase en un format consumible por Map/Reduce
  - Scan

- TableOutputFormat
  - Convierte la salida de Map/Reduce y la escribe en una tabla HBase
  - Put, delete

- HFileOutputFormat
  - Carga masiva
  - Utiliza un job MapReduce para que la salida esté en format interno de las tablas de Hbase
  - Con lo que la carga de datos es directa en el cluster

HBase Backup - Background

- Las bases de datos relacionales habitualmente soportan la recuperación de los datos en un punto determinado en el tiempo a partir de un backup y realizando backups periódicos y archivado de logs. Así la recuperación se realiza a partir del backup más cercano en el tiempo y luego aplicando los logs hasta llegar al momento en el tiempo que se desea.

- Hbase tiene un API para hacer un Import/Export que se puede usar para hacer backup y restore de tablas individuales.

Visión general

- **Full** Backup en base a HBase Snapshot
- Backup **Incremental** basado en los transaction logs
- **Backup incremental** a nivel de Tabla
- Point-In-Time **Restore**
- **Conversión** On-the-fly and Off-line de los HLogs en HFiles
- Off-line **Merge** Backup Images
- Self-contained Backup Image con **Manifest** File
- **Usability** features:
  - progress, status, and history reports
  - purge old Backup Images

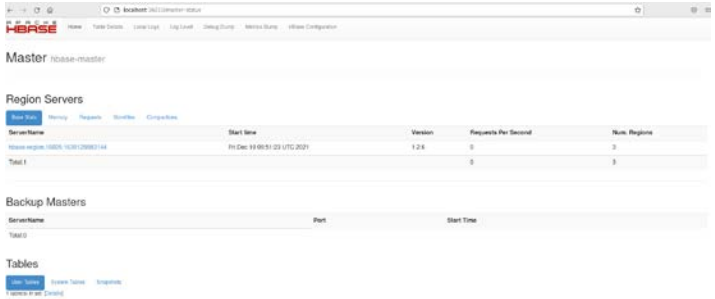
Practicando con HBase

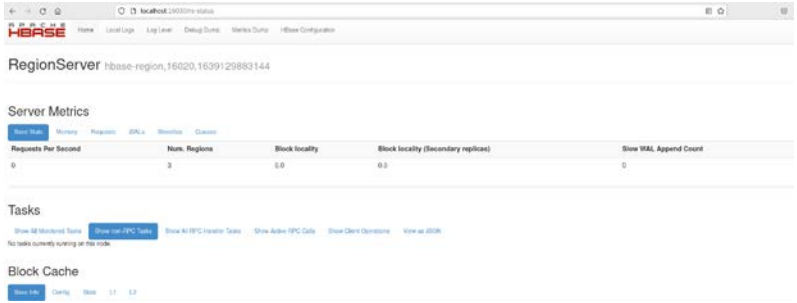
- Arrancamos la imagen de VirtualBox del Master
- Accedemos como umaster
- En una terminal:
  - `cd Master_Platform`
  - `systemctl start docker`
  - `systemctl status docker`
  - `docker-compose start`
  - `docker stats #hasta que identifiquemos mínima actividad`

Confirmamos estado de HBase

```
[umaster@ibmuamdocker Master_Platform]$ docker-compose ps
          Name                        Command                  State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------
datanode                    /entrypoint.sh /run.sh           Up (healthy)   0.0.0.0:50075->50075/tcp,:::50075->50075/tcp
hbase-master                /entrypoint.sh /run.sh           Up             16000/tcp, 0.0.0.0:16010->16010/tcp,:::16010->16010/tcp
hbase-regionserver          /entrypoint.sh /run.sh           Up             16020/tcp, 0.0.0.0:16030->16030/tcp,:::16030->16030/tcp
historyserver               /entrypoint.sh /run.sh           Up (healthy)   0.0.0.0:8188->8188/tcp,:::8188->8188/tcp
hive-metastore              entrypoint.sh /opt/hive/bi ...   Up             10000/tcp, 10002/tcp, 0.0.0.0:9083->9083/tcp,:::9083->9083/tcp
hive-metastore-postgresql   /docker-entrypoint.sh postgres   Up             5432/tcp
hive-postgresql             ./bin/launcher run               Up             0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
hive-server                 entrypoint.sh /bin/sh -c s ...   Up             0.0.0.0:10000->10000/tcp,:::10000->10000/tcp, 10002/tcp
namenode                    /entrypoint.sh /run.sh           Up (healthy)   0.0.0.0:50070->50070/tcp,:::50070->50070/tcp
nodemanager                 /entrypoint.sh /run.sh           Up (healthy)   0.0.0.0:8042->8042/tcp,:::8042->8042/tcp
resourcemanager             /entrypoint.sh /run.sh           Up (healthy)   0.0.0.0:8088->8088/tcp,:::8088->8088/tcp
zoo                         /docker-entrypoint.sh zkSe ...   Up             0.0.0.0:2181->2181/tcp,:::2181->2181/tcp, 2888/tcp, 3888/tcp
```

Arrancamos Firefox dentro de la imagen virtual y accedemos a localhost:16010 y localhost:16030





User Interface - CLI

- Conéctate al Docker de HBase
  - `docker-compose exec hbase-master bash`
- Ejecuta la shell de hbase

```
root@hbase-master:/# hbase shell
2021-12-10 10:31:25,942 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```

- Otras opciones:
  - hbase pe –h #performance evaluation
  - hbase ltt –h #load test tool
  - hbase wal –h
  - hbase hbck #hbase check
  - hbase clean

Comandos hbase - datos

- Gestión de tablas
  - list, create, alter, disable, enable, drop
- Gestión de datos
  - get, count,
  - scan (timerange, filter, limit, startrow, stoprow, timestamp, maxlength, columns)
  - put, incr
  - delete, deleteall

Comandos hbase - gestión

- Generales:
  - status, versión, whoami
- Region:
  - move, split, unassign, close
- Data:
  - compact, flush
- Replication
  - List_peer, remove_peer, enable/disable_peer, start_replication, stop_replication
- Security:
  - grant, revoke, user_permision

Practicando con Hbase

- Descarga los ficheros zip de Moodle con los ficheros de la práctica en la imagen virtual

- Copialos en el Docker de HBase
  - `docker cp ./hbase hbase-master:/root`

- Conéctate al Docker de HBase
  - `docker-compose exec hbase-master bash`

- El fichero zip contiene los datos de ejemplo y los comandos para poder utilizarlos

Practicando con Hbase (II)

Para los ejemplos de comandos:

- Utilizamos la Shell de hbase y le pasamos los comandos a través de un fichero
  - `hbase shell ejemplo.comandos.hbase`

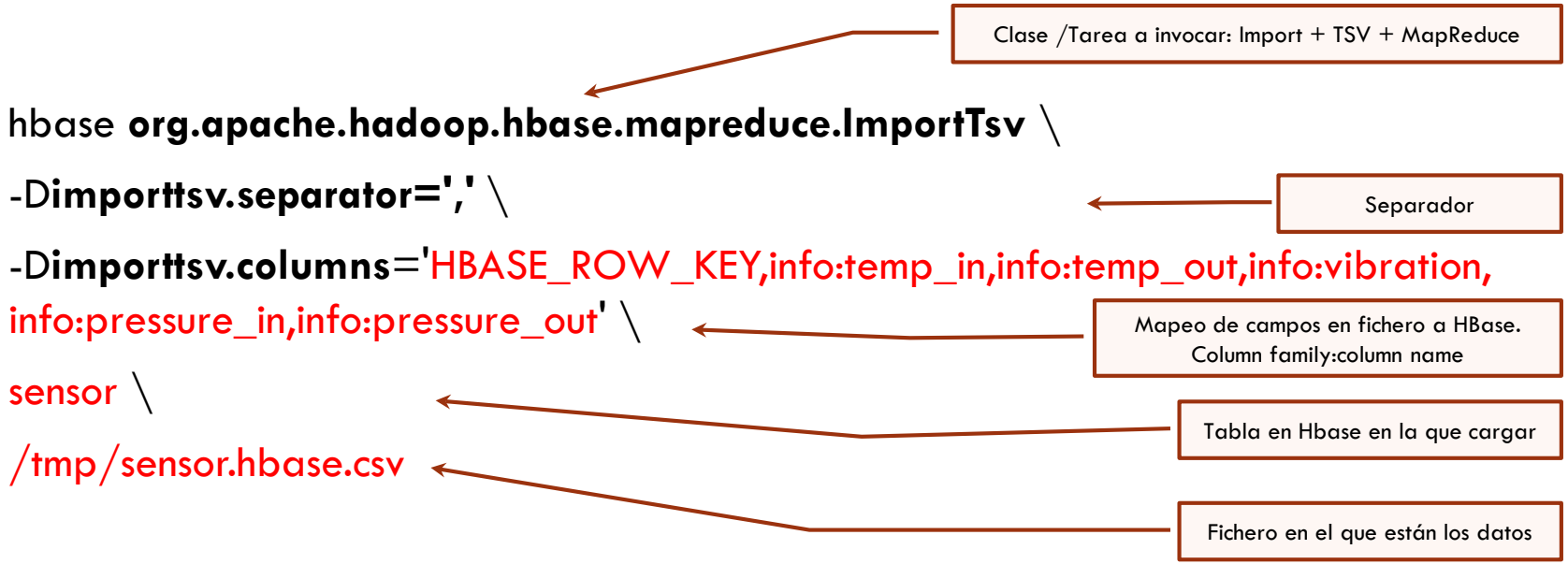
Para los ejemplos de sensores

- Crea la tabla de sensores:
  - `hbase shell sensor.hbase.createtable`
- Copia el fichero con datos en /tmp:
  - `cp sensor.hbase.csv /tmp`
- Carga los datos en hbase:
  - `sh sensor.carga.fichero.hbase.sh`
- Ejecuta algunos comandos para comprobar contenido
  - `hbase shell sensor.comandos.hbase`
- Para borrar la tabla
  - `hbase shell sensor.hbase.droptable`

Presta atención a los comandos utilizados y a su salida

Carga realizada en HBase

sensor.carga.fichero.hbase.sh



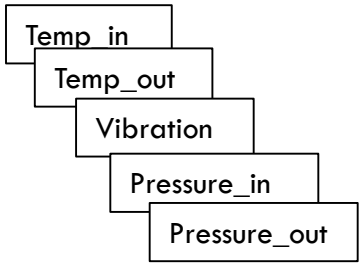
Prestad atención al nombre de tablas y column families, formato de los datos y a los caracteres de fin de línea (formato DOS vs formato Linux)

Mapa de Column Family

Tabla: sensor

Column Families:

- Device
- Info



Info

Row Key

Row Key	Temperatura	Vibración	Presión
5842	52	32,4	242,342
5843	53	33,5	243,343
5844	54	34,6	244,344
5845	55	35,7	245,345
5846	56	36,8	246,346
5847	57	37,9	247,347
5848	58	38,1	248,348
5849	59	39,2	249,349
5851	51	31,3	241,341

- No hay ningún dato de la column family Device
- Hay un valor de cada campo por número de dispositivo
- Si volvemos a cargar otro registro del mismo dispositivo contaríamos con más valores de cada campo (si el número de las versiones configuradas nos lo permiten)

Consideraciones finales

- HBase no es la solución a todos los problemas
- ¿Tenemos datos suficientes para garantizar/justificar el uso de Hbase?
  - ¿Cientos de millones o billones de filas?
  - ¿Pocos miles o millones de filas?
  - Hay que considerar presente y futuro
- ¿Vamos a necesitar funciones extra habituales de otras bases de datos?
  - Por ejemplo, tipos de columnas, índices secundarios, transacciones, lenguajes de consulta avanzados, etc.
- ¿Tenemos la infraestructura adecuada?
  - Generalmente escala bien horizontalmente incorporando más nodos
  - HDFS tiene un punto de partida en infraestructura que no podemos desdeñar

Consideraciones finales…

- ¿El modelo de datos es fijo o muy estable, o necesitamos alta flexibilidad?

- ¿Tenemos necesidad de un crecimiento automático?, ¿tenemos crecimientos en volumen de datos inesperados o estivales?

- ¿Algún requerimiento relacionado con las propiedades ACID?, ¿Transaccionabilidad impactando a más de una tabla?