HBase. Tecnologías “on-line”, “real-time” y sistemas híbridos

Programa experto en Big Data

Profesor: Ignacio Marrero Hervás

Sesión arquitectura, 1º Trimestre
Curso 2014-2015

Mi experiencia

Actualmente

Arquitecto de soluciones Big Data en Zed Worldwide

Experiencia

- Arquitecturas Big Data desde 2011
- Arquitectura de sistemas, especialmente con tecnologías Java
- Infraestructuras Cloud o virtualizadas
- Ingeniería de software en Java
- Data Science y análisis de datos

Bibliografía

HBase book (O'Reilly 2011)

HBase: The Definitive Guide

The Apache HBase Reference Guide

https://hbase.apache.org/book.html

HBase shell Reference Guide

http://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

HBase API

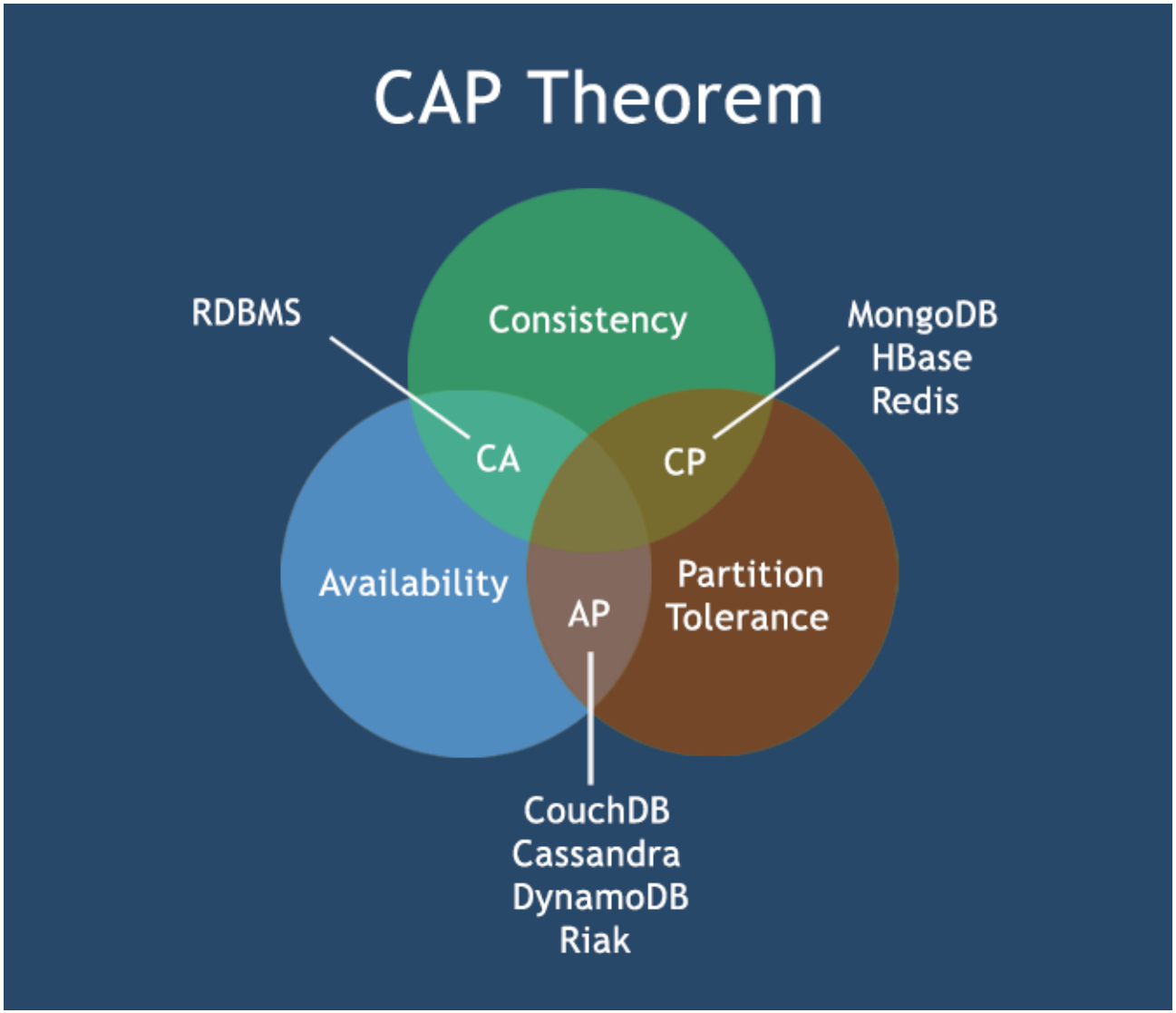
http://archive.cloudera.com/cdh5/cdh/5/hbase/apidocs/

ÍNDICE

¿Qué es NoSQL?

- Not-Only SQL
- CAP (Consistency, Availability, Partition tolerance)
- Consistencia: Estricta, Secuencial, Causal, Eventual, Débil
- Baremo para la clasificación de sistemas NoSQL:
  - Modelo de datos
  - Modelo de almacenamiento
  - Modelo de consistencia
  - Modelo físico
  - Rendimiento RW
  - Índices secundarios
  - Gestión de fallos
  - Compresión
  - Balanceo
  - Atomicidad RUW
  - Bloqueos

¿Qué es NoSQL?


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

¿Por qué NoSQL?

Ventajas

- Escalabilidad ➔ Sistemas distribuidos
- Hardware genérico ➔ Menor coste por GB

Desventajas

- No totalmente ACID (Atomicity, Consistency, Isolation, Durability)

Los lenguajes SQL ya no son un problema: están presentes en las tecnologías SQL y NoSQL

¿Por qué HBase?

Características de HBase

- Una base de datos NoSQL de tipo columnar
- Consistente y tolerante a las Particiones
- Tiene una arquitectura escalable
- Su almacenamiento está basado en HDFS
- Tiene punto único de fallo

Garantías ACID (http://hbase.apache.org/acid-semantics.html):

- Atomicidad. La operación "`put`" es consistente para cada fila
- Consistencia. Todos los accesos son consistentes para cada fila

  Consistencia de un "`scanner`": las filas pueden contener mutaciones posteriores a la creación del "`scan`"
- Aislamiento. Las mutaciones de cada fila son independientes
- Persistencia. Todos los "`commit`" exitosos son persistentes

¿Por qué HBase?

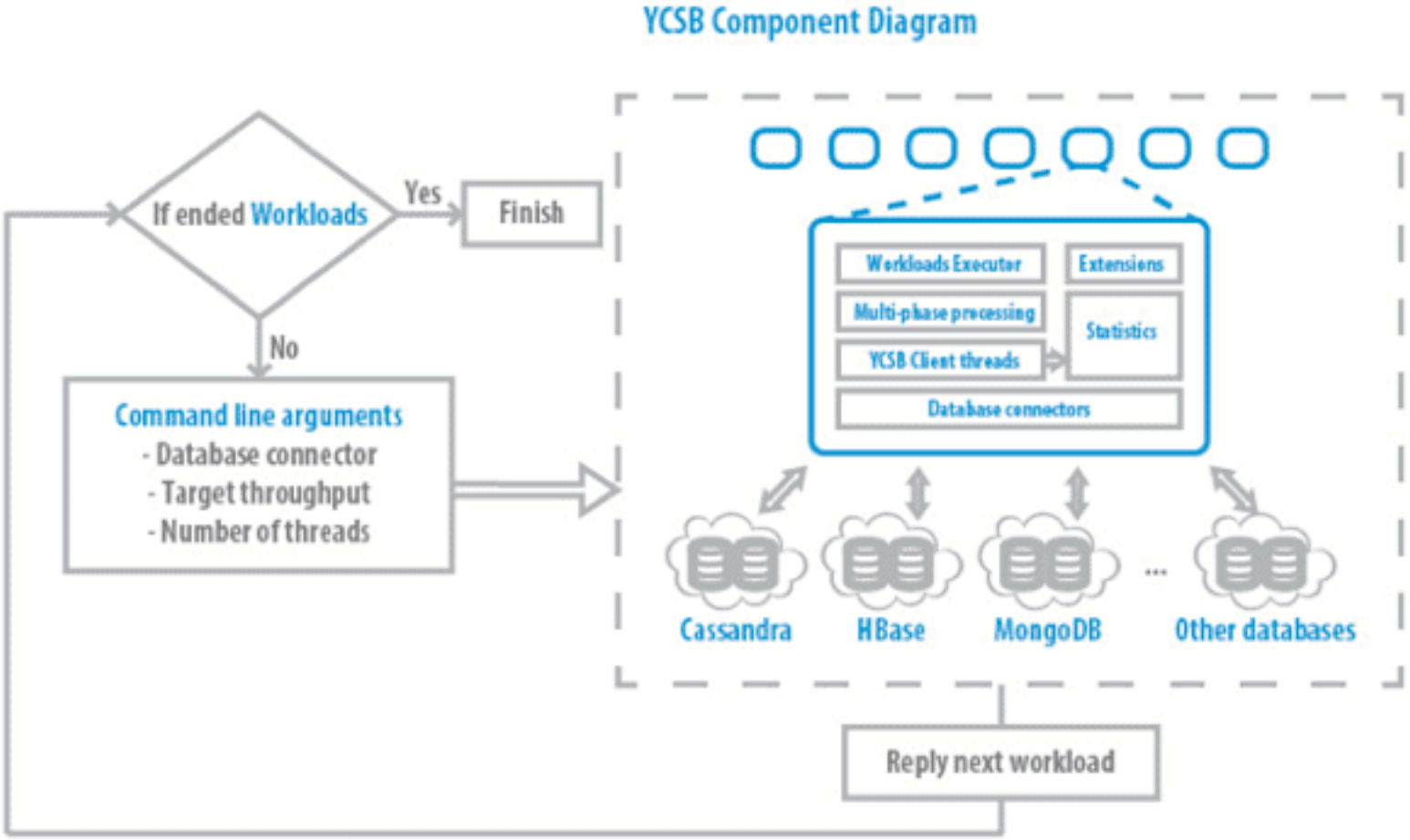
Reflexiones necesarias para tomar la decisión

- Escalado vs. coste en tiempo y hardware
- Complejidad del mantenimiento ➔ costes
- Confianza en el diseño software de la solución NoSQL
- Completitud del software: desastres, seguridad, extensible
- Performance adecuada para el problema a tratar

Dos casos reales (MongoDB, Riak, HBase, Cassandra)

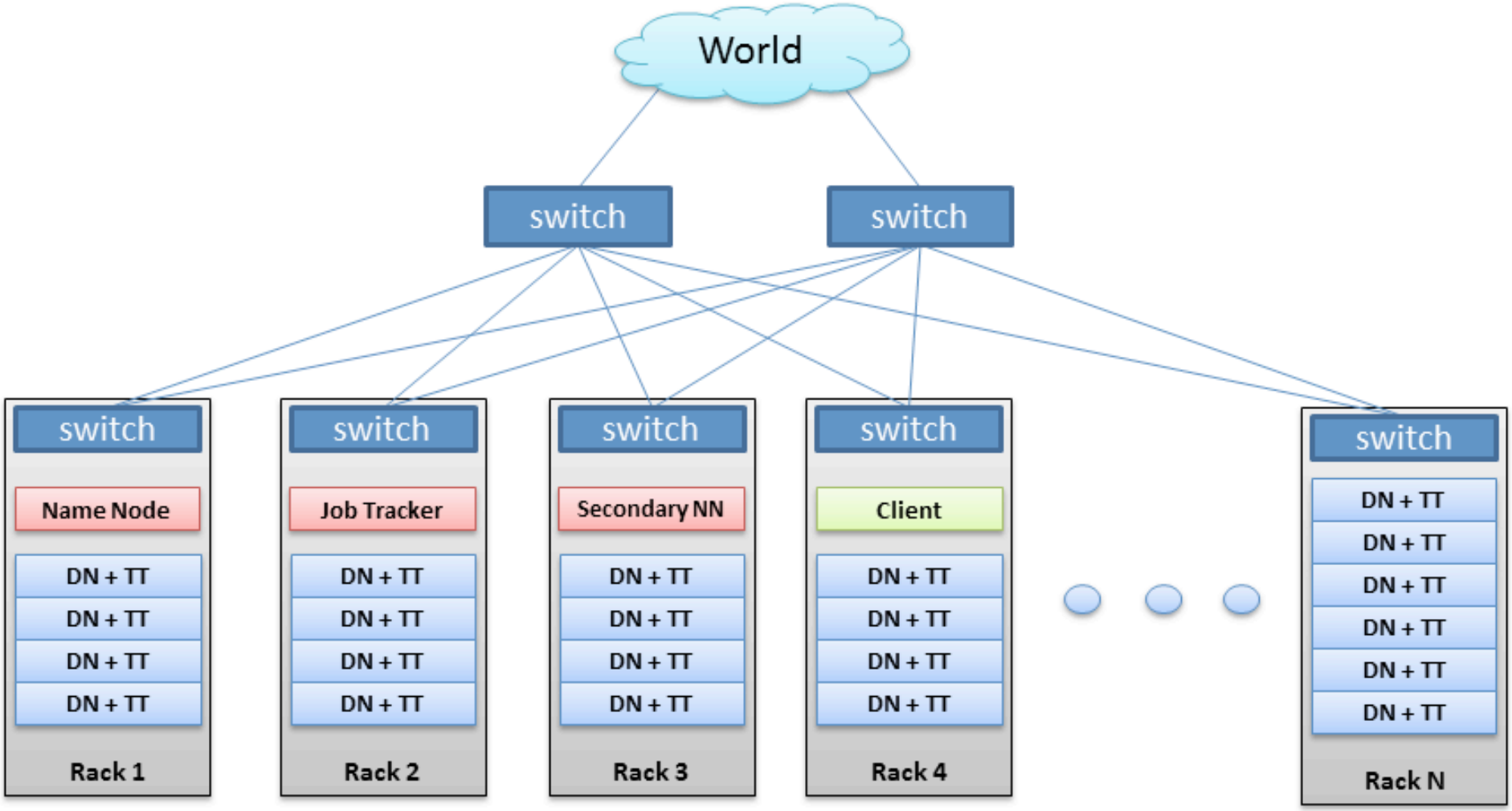
1. MarkedUp (2013)
   http://blog.markedup.com/2013/02/cassandra-hive-and-hadoop-how-we-picked-our-analytics-stack/
2. Estudio de Altoros Systems Inc. (2012)
   http://www.networkworld.com/news/tech/2012/102212-nosql-263595.html?page=1

¿Por qué HBase?

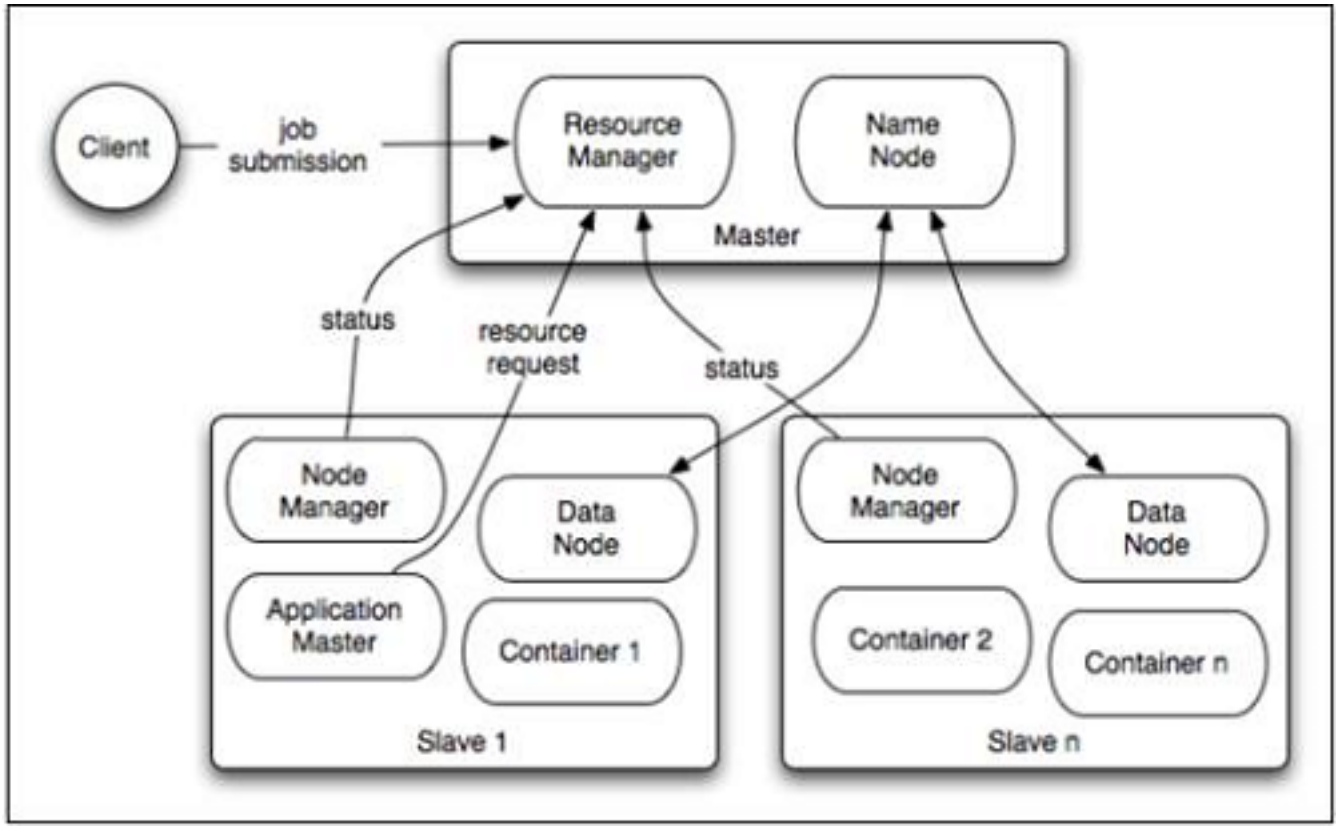


Yahoo! Cloud Serving Benchmark

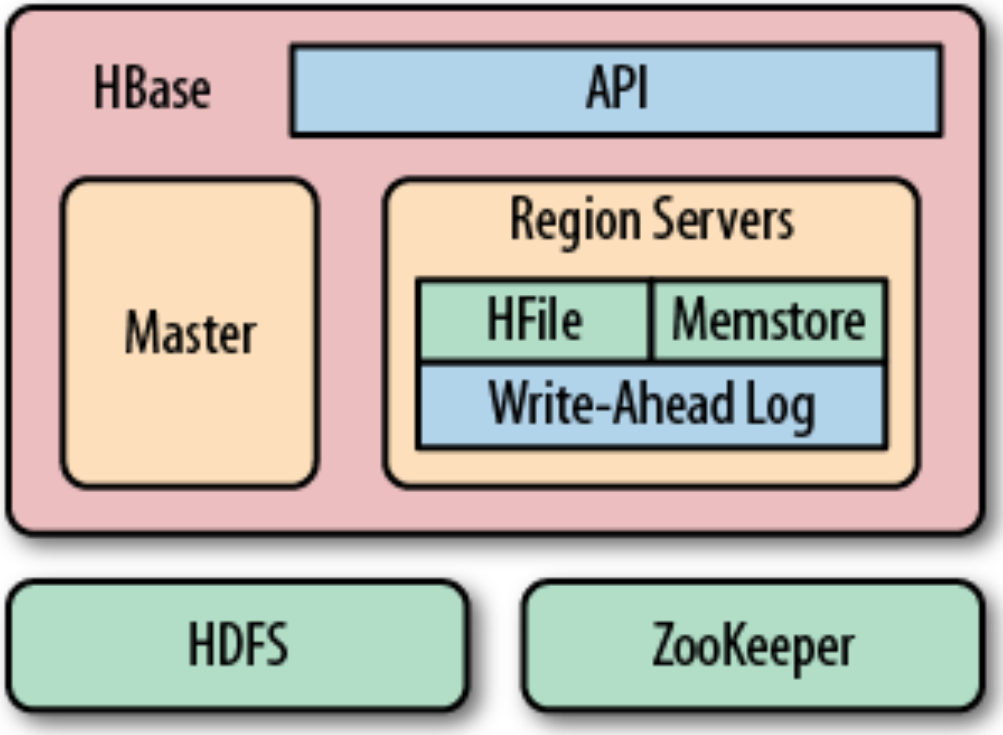
Arquitectura de Hadoop



Arquitectura de Hadoop



Arquitectura de HBase



Conceptos de Arquitectura

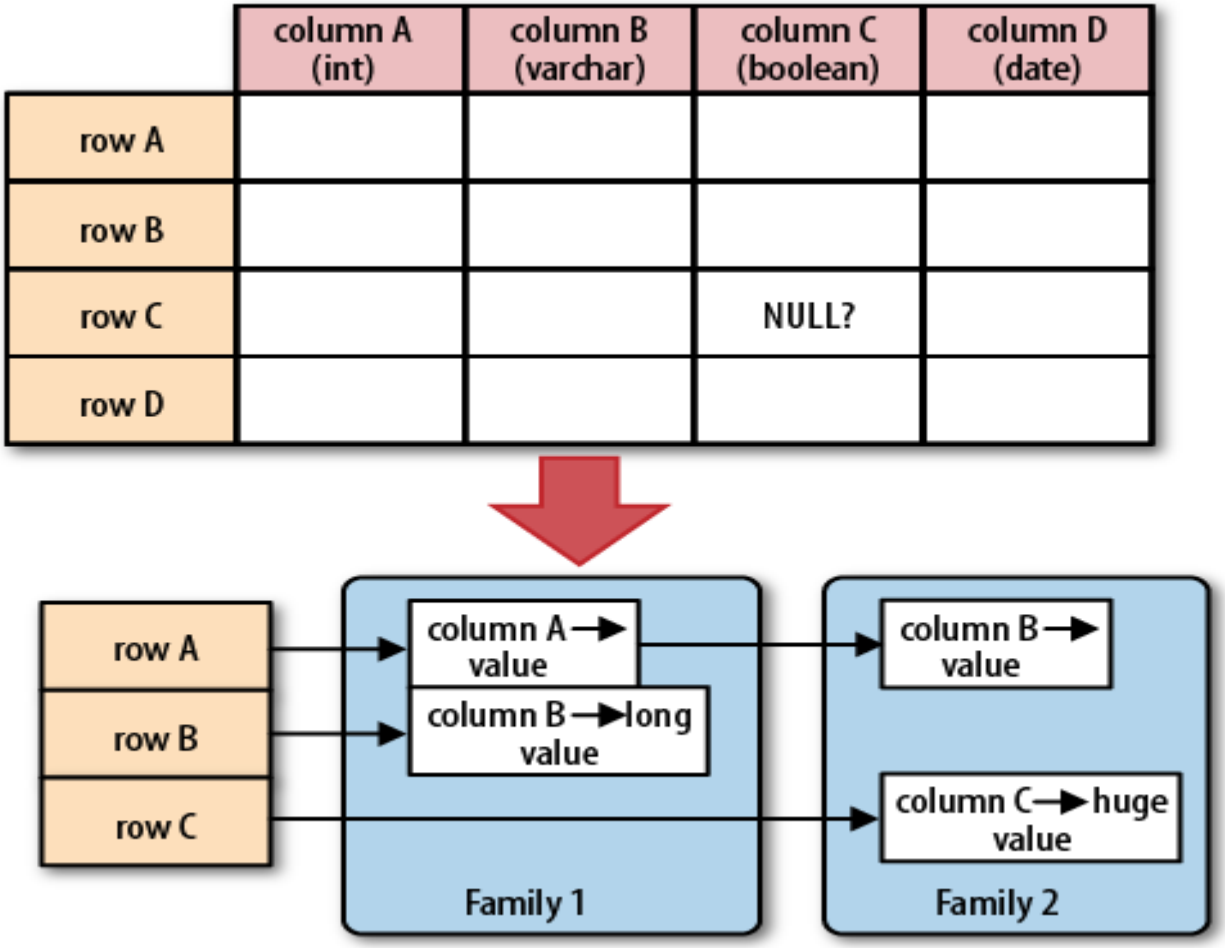
¿Qué es una celda en HBase?

`(Table, RowKey, Column Family, Qualifier, Timestamp) → Value`

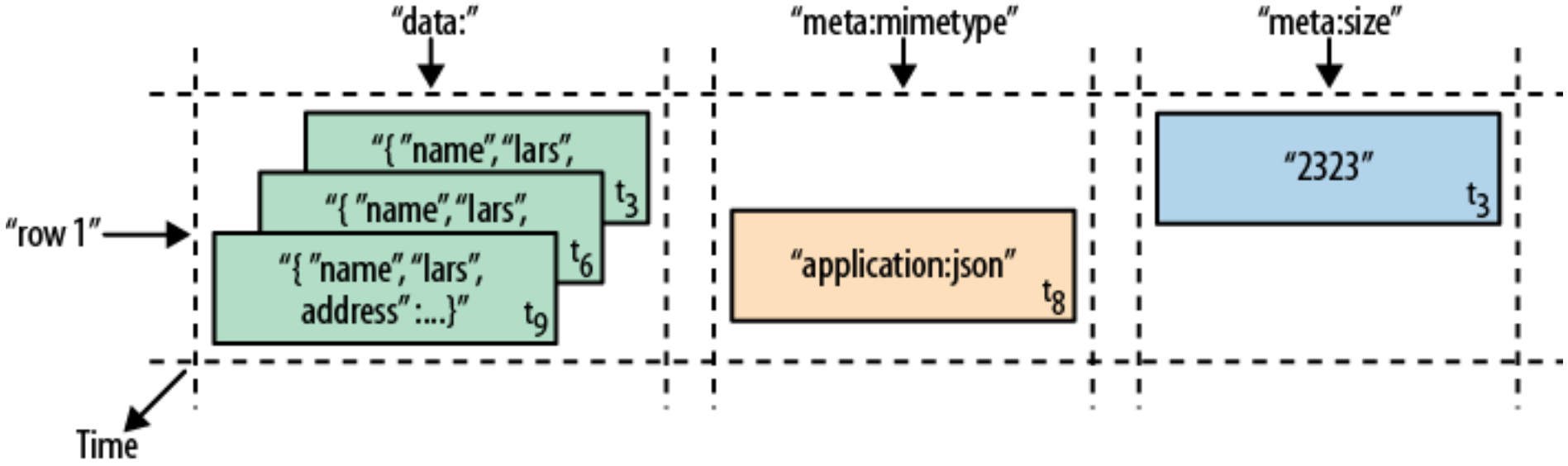
```
SortedMap<
       RowKey, List<
              SortedMap<
                     Column, List<
                            Value, Timestamp
                     >
              >
       >
>
```

Clase	Tipo
RowKey	Byte array
Column Family	string
Qualifier	Byte array
Timestamp	Long
Value	Byte array

Conceptos de Arquitectura



Conceptos de Arquitectura



Row Key	Time Stamp	Column "data:"	Column "meta:" "mimetype"	Column "meta:" "size"	Column "counters:" "updates"
"row1"	t_3	"{ "name": "lars", "address": ...}"		"2323"	"1"
	t_6	"{ "name": "lars", "address": ...}"			"2"
	t_8		"application/json"		
	t_9	"{ "name": "lars", "address": ...}"			"3"

Conceptos de Arquitectura

Namespaces

Provee de la capacidad de crear bases de datos independientes con asignación de recursos propios. En la práctica es un contenedor de tablas

Table

Estructura lógica similar a la tabla relacional: conjunto de filas que son a su vez series de columnas identificadas por una Rowkey

Column Family

Contenedor físico de columnas

Rowkey

Identificador único de fila. Es el índice primario y de particionamiento

Conceptos de Arquitectura

Qualifier

Identificador de columna

Cell

Cada uno de los pares clave-valor que componen una row

RegionScanner

Controla la búsqueda de una Rowkey en una Región

StoreScanner

Controla la búsqueda de una Rowkey en un Store File

Conceptos de Arquitectura

Namespace “hbase”

Contiene las tablas del sistema

Namespace “default”

Contiene las tablas que no tienen namespace

Tabla “hbase:namespaces”

Tabla con meta-información sobre los namespaces

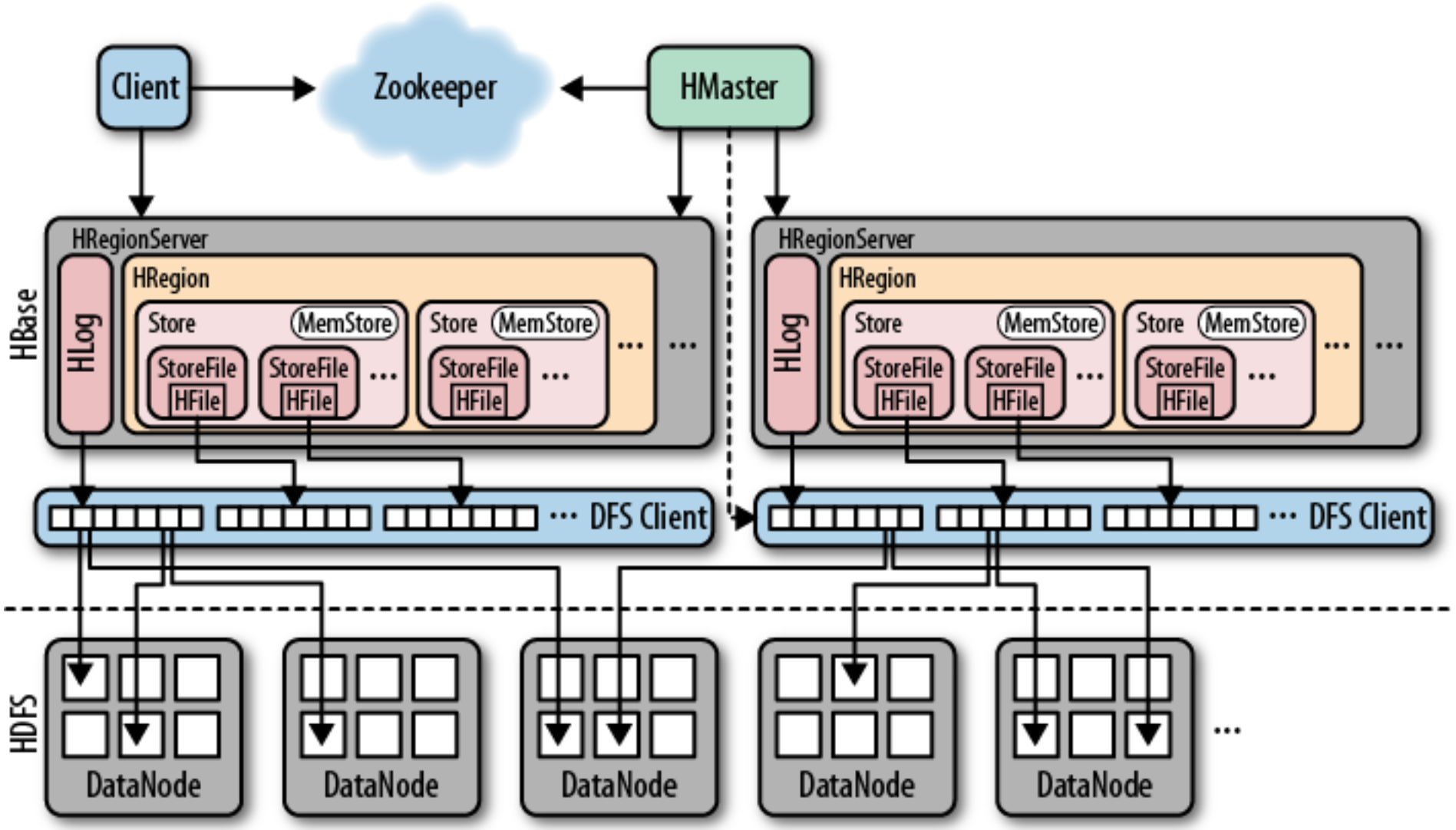
Tabla “hbase:meta”

Almacena en que Region Server están las Regiones correspondientes a cada tabla

Cliente de HBase

Realiza los procesos de CRUD. Cachea la relación entre Regiones y Region Servers para optimizar el proceso de búsqueda

Arquitectura física



Arquitectura física. Conceptos

Zookeeper

- Servicio que permite la coordinación de los distintos procesos del cluster
- Cada Region Server crea un nodo efímero en Zookeeper (znode). Los nodos efímeros se vinculan a las sesiones de los clientes
- Almacena los cambios de estado de las Regiones
- Almacena la tabla META: asociación Regiones - Region Server

HMaster

- Gestiona el balanceo de Regiones en Region Servers
- Gestiona el ciclo de vida de las Regiones
- Gestiona las creaciones de tablas y los cambios de esquema de las tablas
- Mantiene el estado del cluster

Arquitectura física. Conceptos

HTable y HConnection

Instancia que permite a los clientes acceder mediante RPC a los servicios de HBase. Controla el acceso a las tablas

Region Server (HRegionServer)

Proceso que controla un cierto número de Regiones. El Region Server tiene un cache de bloques, dentro del que también está el cache de índices de Hfiles

Region (HRegion)

Instancias que controlan cada Región asociada a una tabla determinada

Store

Instancias que controlan cada Column Family de una tabla para una Región determinada

Arquitectura física. Conceptos

MemStore

Instancia que controla el almacenamiento en memoria de los datos de entrada

HFile

Instancia que controla cada fichero de almacenamiento en disco

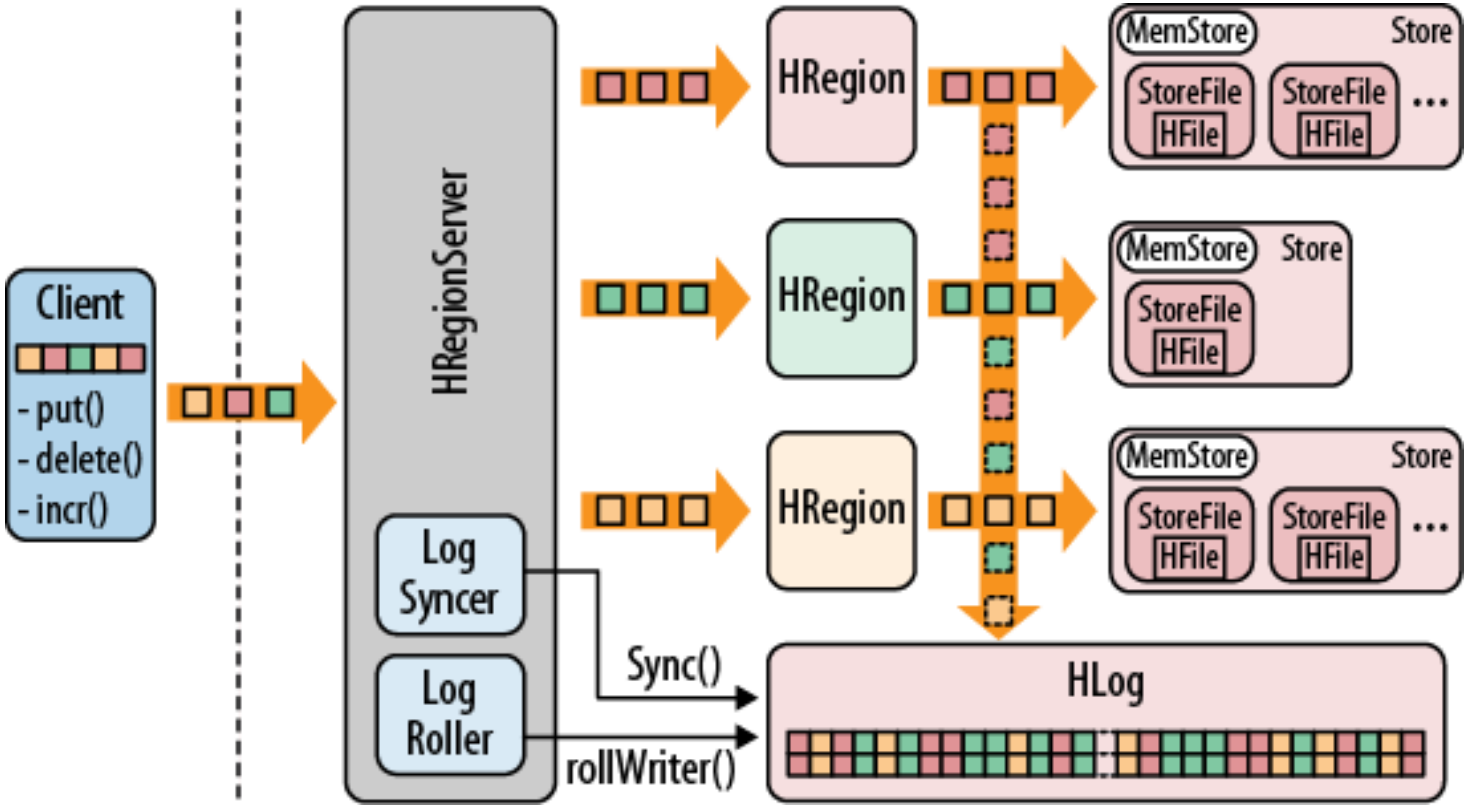
HLog

Instancia que controla el almacenamiento del sistema de Write Ahead Log

Cliente de HBase

Permite acceder a los datos almacenados mediante una instancia de HTable o HConnection. Permite cachear los datos

Arquitectura física. Write Ahead Log



Regiones. Operación

Objetivo

El balanceo de carga es esencial en el funcionamiento óptimo de HBase, por ello tiene mecanismos automáticos asíncronos de control del balanceo de carga. No obstante también existe la posibilidad de balancearlo manualmente

Balanceador

El balanceador es un proceso asíncrono que se ejecuta en periodos de "`hbase.balancer.period`" buscando igualar la carga de Regiones entre los distintos Region Servers

Split

Cuando el tamaño de cualquier Store File de una Región supera "`hbase.hregion.max.filesize`" se inicia el "split" que divide la Región en dos Regiones, cada una con la mitad de las filas

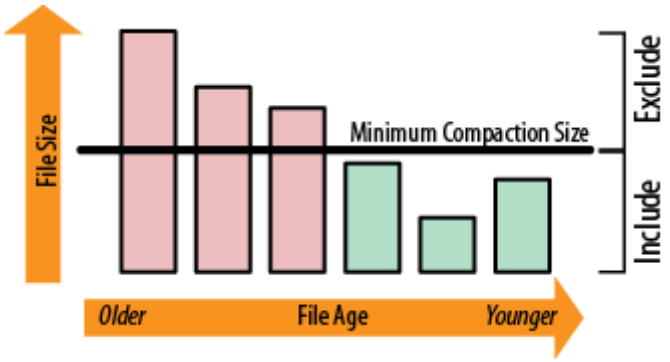
Regiones. Operación

Major compaction

En cada Region Server un hilo verifica que tipo de compaction debe ser ejecutada en base a la configuración. Si es major se intentan compactar todos los Store Files en uno solo

Minor compaction

Si es minor compaction, su comportamiento está regulado por configuración según el número y tamaño de los Store File que existan en ese momento



Regiones. Ciclo de vida

El ciclo de vida de una Región es controlado por el HMaster

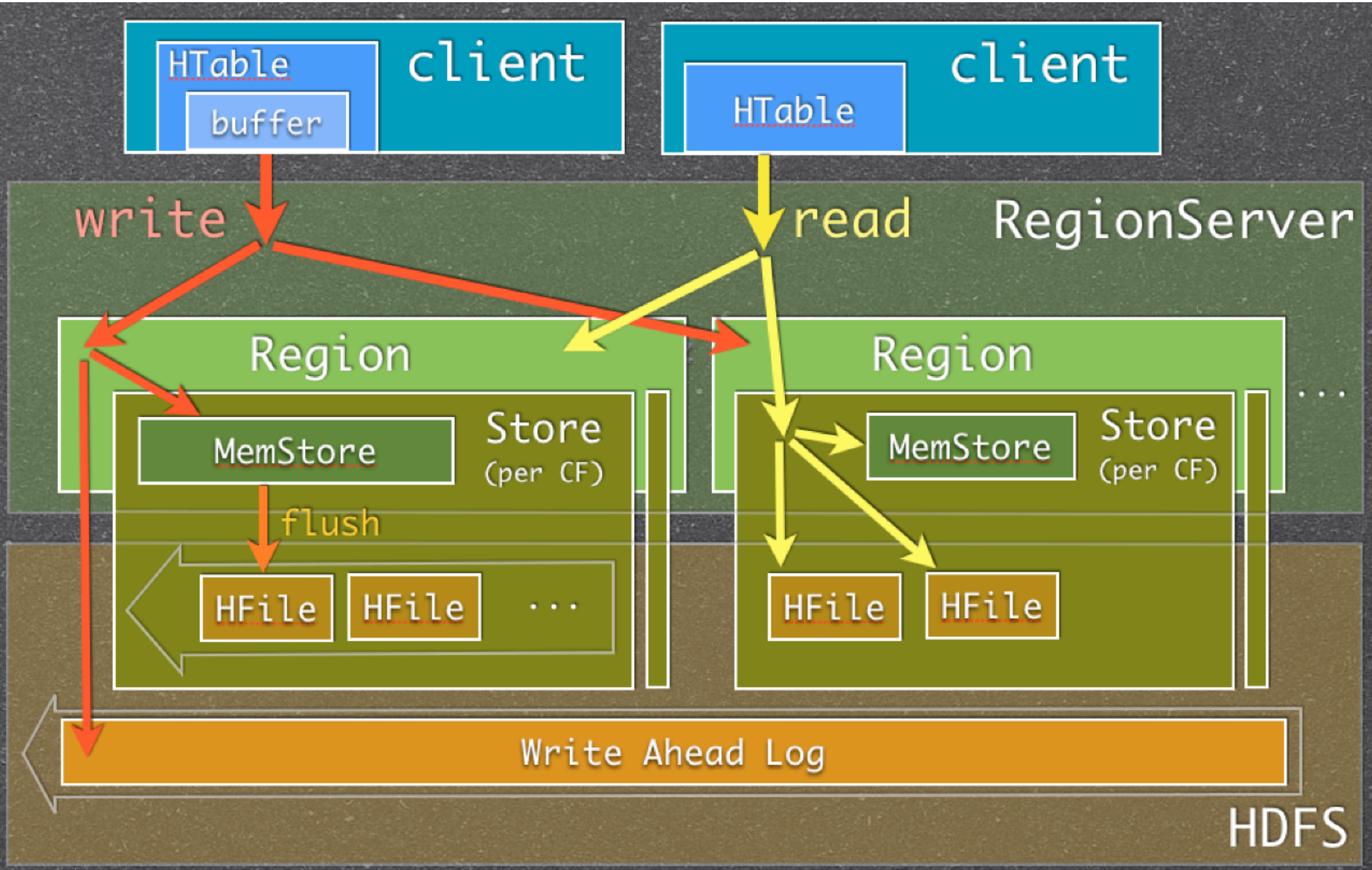
Estado	Descripción
Offline	La Región está offline
Pending Open	Enviada la petición de Open al servidor
Opening	El servidor está abriendo la Región
Open	La Región está abierta y disponible
Pending Close	Enviada la petición de Close al servidor
Closing	El servidor está cerrando la Región
Closed	La Región está cerrada
Splitting	El servidor está dividiendo la Región
Split	La Región ha sido dividida

Almacenamiento. Write Path

Secuencia de escritura de un dato en HBase

1. El cliente contacta Zookeeper para obtener el Region Server que tiene la tabla META
2. El cliente contacta ese Region Server y obtiene la región de la tabla META
3. El cliente busca en la tabla META y obtiene el Region Server que aloja la Región que contiene la Rowkey
4. El cliente le envía los datos al Region Server
5. El Region Server escribe en el WAL
6. El Region Server almacena los datos en MemStore de la Región correspondiente al Column Family
7. Si el MemStore esta lleno se hace flush a disco y se crea un nuevo Store File

Almacenamiento. Lógica

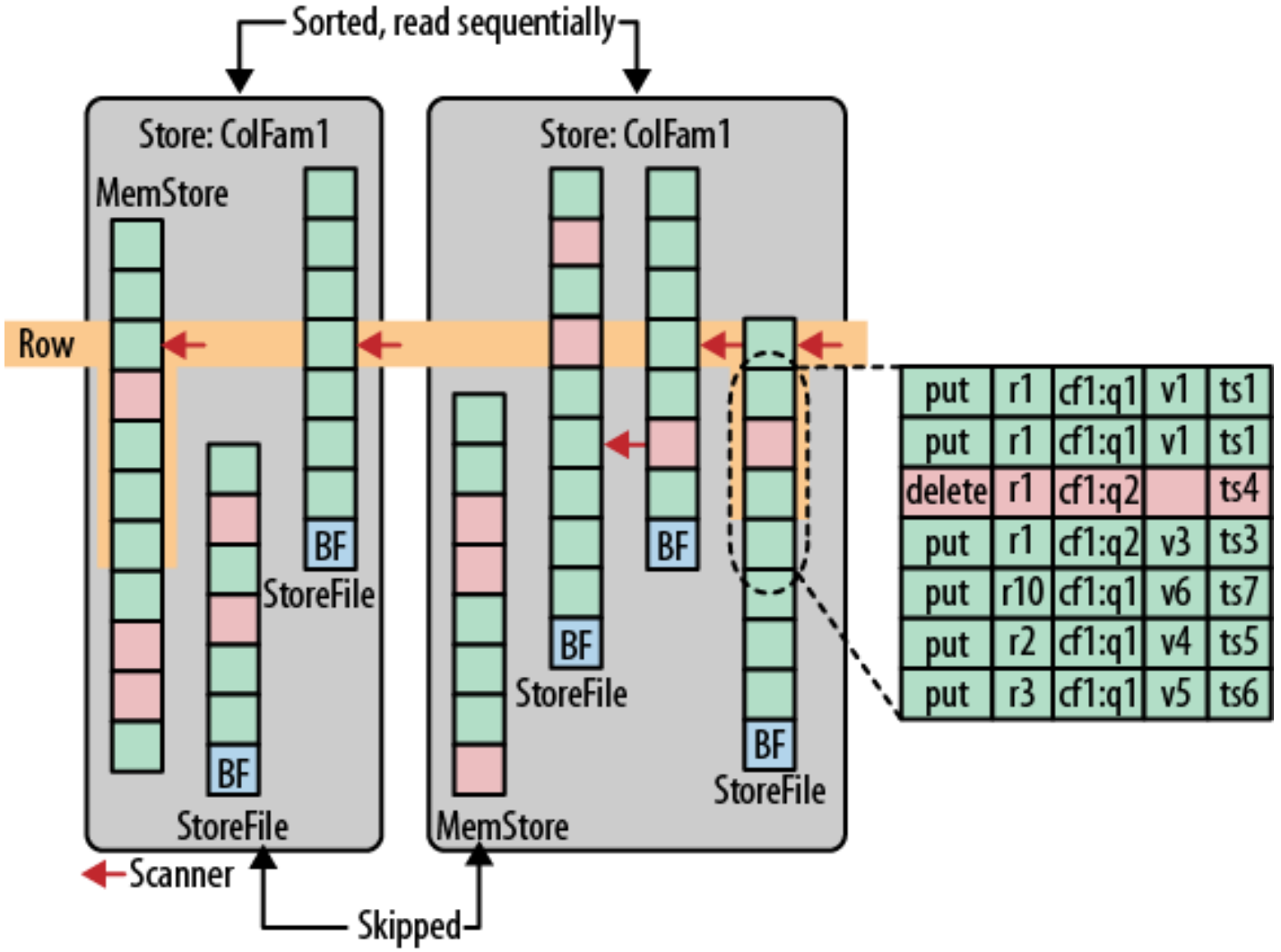


Almacenamiento. Read Path

Secuencia de lectura de una fila

1. El cliente contacta Zookeeper para obtener el Region Server que tiene la tabla META
2. El cliente contacta ese Region Server y obtiene la región de la tabla META
3. El cliente busca en la tabla META y obtiene el Region Server que aloja la Región que contiene la Rowkey
4. El cliente pide la Rowkey al Region Server y se inicia el proceso de búsqueda del dato
5. El RegionScanner y StorageScanner buscan el dato

Almacenamiento. Read Path



Almacenamiento. Estructura de ficheros

En HDFS se almacena toda la estructura de ficheros

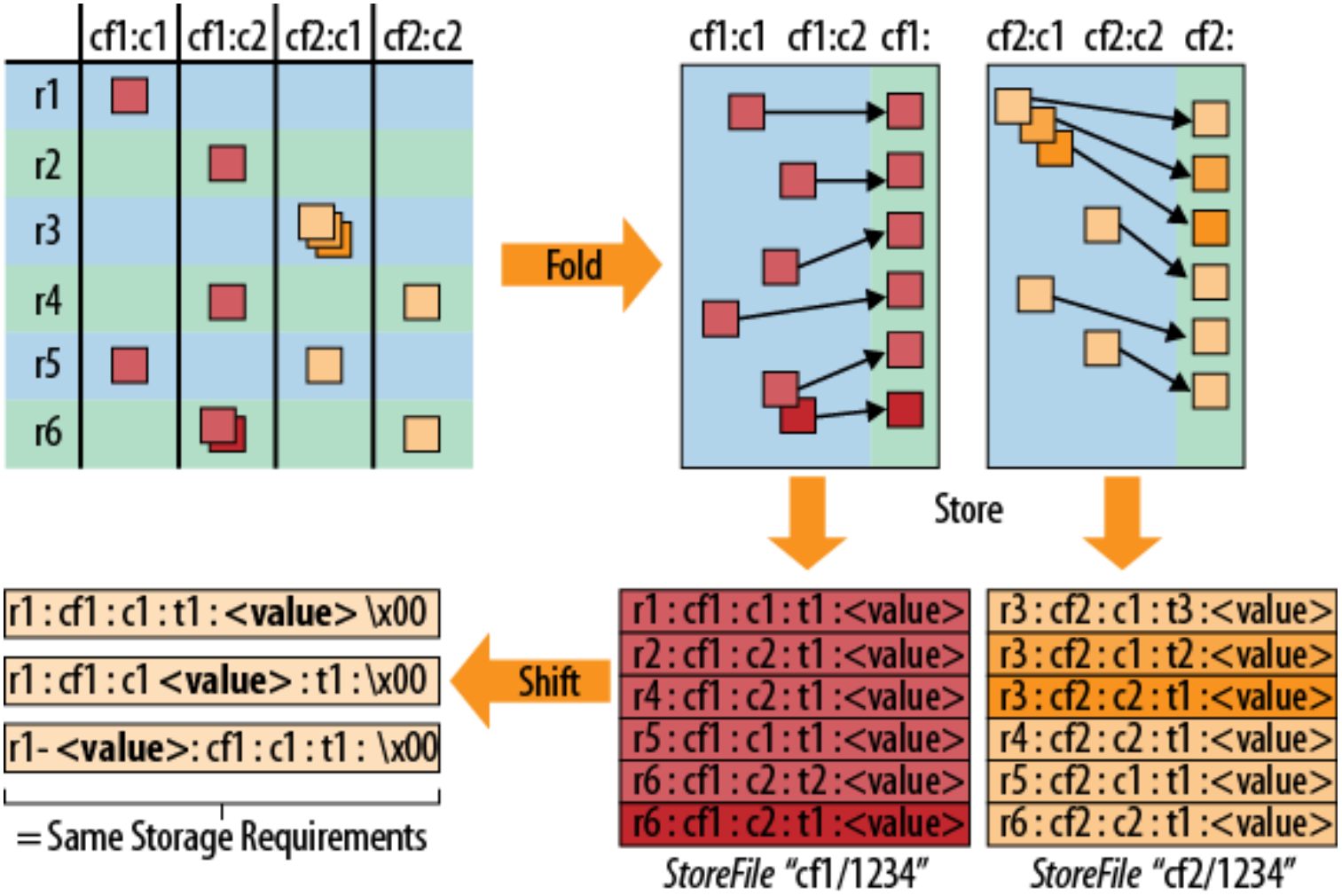
```
/<hbase-root-dir>/data/<namespace>/<tablename>/<encoded-regionname>/<column-family>/<filename>
```

El "`encoded-regioname`" está compuesto por

1. Nombre de la tabla
2. Rowkey inicial para esa Región
3. ID = timestamp de creación de la Región

Almacenamiento. Estructura de datos

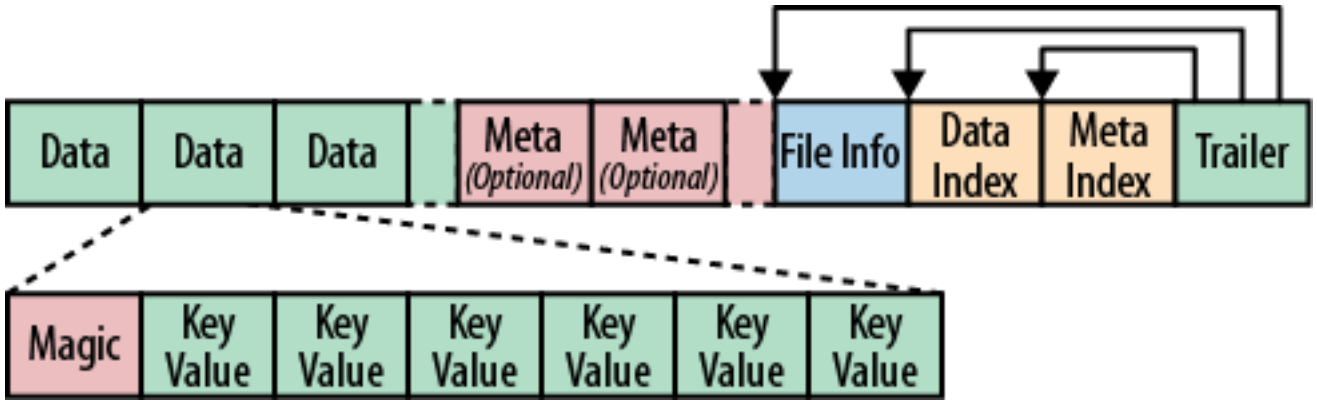
Organización física del modelo de datos



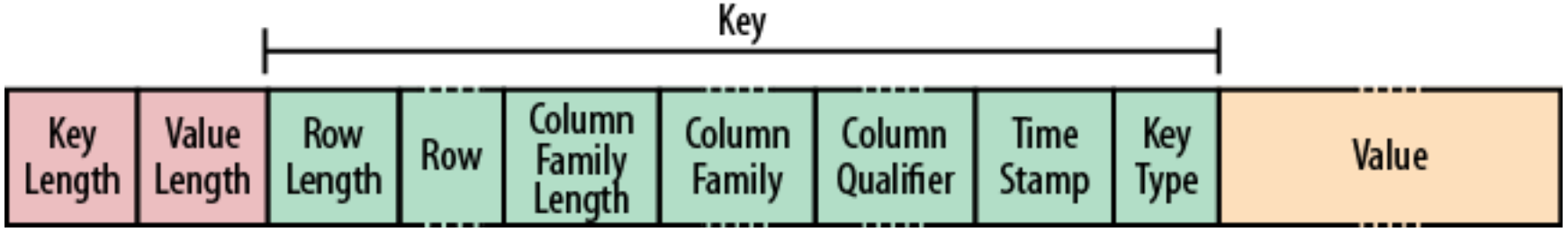
Almacenamiento. Estructura de datos

Las Rowkeys se almacenan ordenadas en sentido ascendente y sus versiones por Timestamp descendente

Estructura de un HFile



Formato de KeyValue



Instalación

Configuración mínima en producción

HW / SW	Característica	Valor
Red	Arquitectura	ToR + CaS
Servidores	CPU	2x quad-core
	RAM	Master = 24Gb Slave = 48Gb
	Disco	Master = RAID 1+0 Slave = JBOD IOPS = 100 / disco

Instalación

Configuración mínima en producción

HW / SW	Característica	Valor
Software	Sistema Operativo	Linux (CentOS,…) Swap = off
	File System	Ext4,…
	JRE Heap	Master = 4Gb Slave = 12Gb
	Java version	Oracle 1.6
	Hadoop version	0.20.x
	DNS	host `hostname -f` host `hostname –i`
	File handles	10000
	Tiempo	NTP

Configuración

hbase-env.sh

variables de entorno y parámetros JVM

hdfs-site.xml

properties relacionadas con HDFS

hbase-site.xml

properties relacionadas con HBase

regionservers

almacena todos los DNS de Region Server

log4j.properties

configuración de los niveles de log

zoo.cfg

configuración de zookeeper

Despliegue

Apache Whirr: basado en scripts

https://github.com/steveloughran/whirr

http://www.cloudera.com/

Puppet: basado en módulos

https://forge.puppetlabs.com/tags/hadoop

Chef: basado en cookbooks

http://community.opscode.com/cookbooks/hadoop

Cloudera Manager (Cloudera): interfaz web

Despliegue, gestión y monitorización

Ambari (Hortonworks): interfaz web

Despliegue, gestión y monitorización

Operación básica

Inicio

```
/etc/init.d/hbase-master start
/etc/init.d/hbase-regionserver start
```

Parada

```
/etc/init.d/hbase-master stop
/etc/init.d/hbase-regionserver stop
```

Información sobre el sistema en las interfaces de usuario

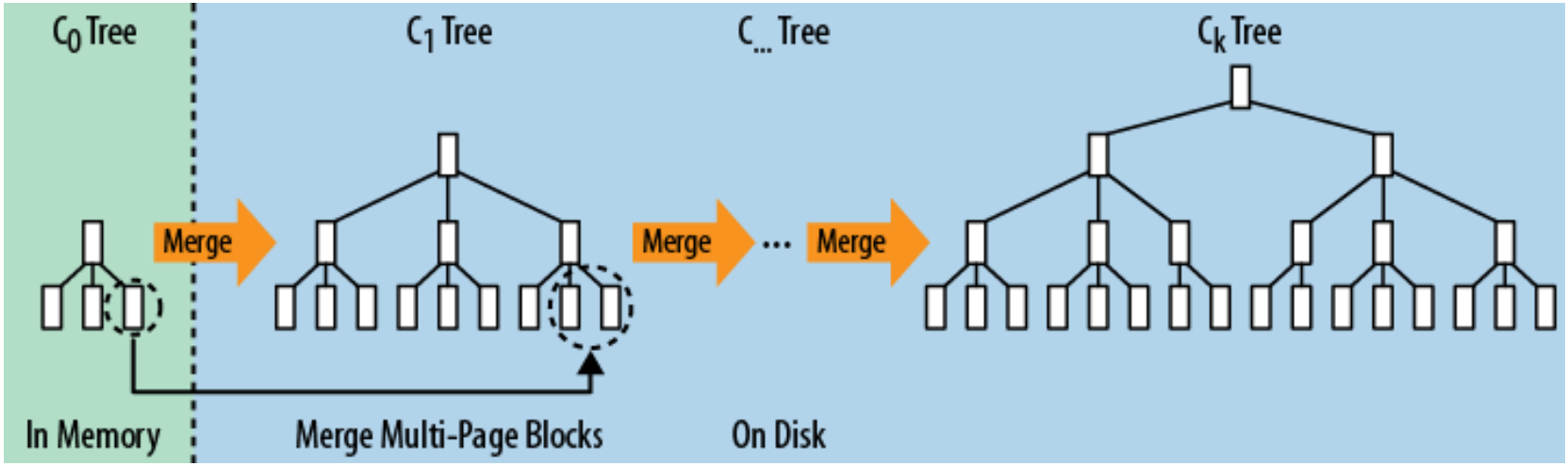
Master: http://localhost:60010

RegionServer: http://localhost:60030

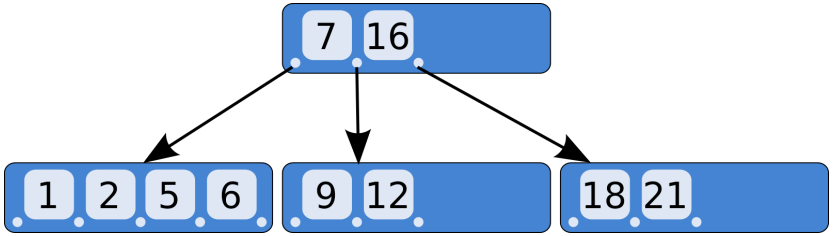
Seek vs. Transfer

La tecnología actual posibilita la mejora del rendimiento en los procesos de “transfer” mas que en los de “seek” ➔

HBase utiliza Log-Structured Merge-Trees



B-Tress ➔



Seek vs. Transfer

Caracterísiticas de LSM-trees

1. Almacenamiento en memoria hasta llenar el buffer
2. Flush a disco para consolidar Store Files
3. Store Files organizados en B-trees optimizados para acceso secuencial

Por tanto HBase estará optimizado en un escenario en el que las operaciones de “transfer” prevalezcan sobre las de “seek”: grandes volúmenes de inserciones

Seek vs. Transfer

En este escenario las ventajas de la arquitectura de HBase son:

- Ritmo de Insert constante
- Write and Read independientes
- El número de Seeks por cada acceso está acotado

Optimización. Compresión

La compresión siempre es una opción deseable porque minimiza recursos salvo por la CPU requerida para comprimir y descomprimir

No obstante, incluso este consumo de CPU es inferior al requerido para leer los mismos datos descomprimidos

snappy

- Ha sido desarrollado por Google pensando para optimizar la velocidad. Su performance es un poco mejor que la de LZO al codificar e igual al decodificar

lzo

- Lempel-Ziv-Oberhumer es un algoritmo pensado para optimizar la velocidad. Hasta que apareció Snappy era el mejor con diferencia

Optimización. Compresión

gz

Es el que mejor comprime pero el consumo de CPU es demasiado alto y puede afectar al rendimiento del cluster

```
$ hbase org.apache.hadoop.hbase.util.CompressionTest
14/02/12 00:13:33 WARN conf.Configuration: hadoop.native.lib is deprecated.
Instead, use io.native.lib.available
Usage: CompressionTest <path> none|gz|lzo|snappy

For example:
  hbase class org.apache.hadoop.hbase.util.CompressionTest file:///tmp/testfile
gz
```

Replicación

La replicación permite sincronizar datos de forma asíncrona desde un clúster maestro a dos o más clústeres esclavos

La replicación sigue una arquitectura maestro-esclavo con un patrón “master-push” de funcionamiento

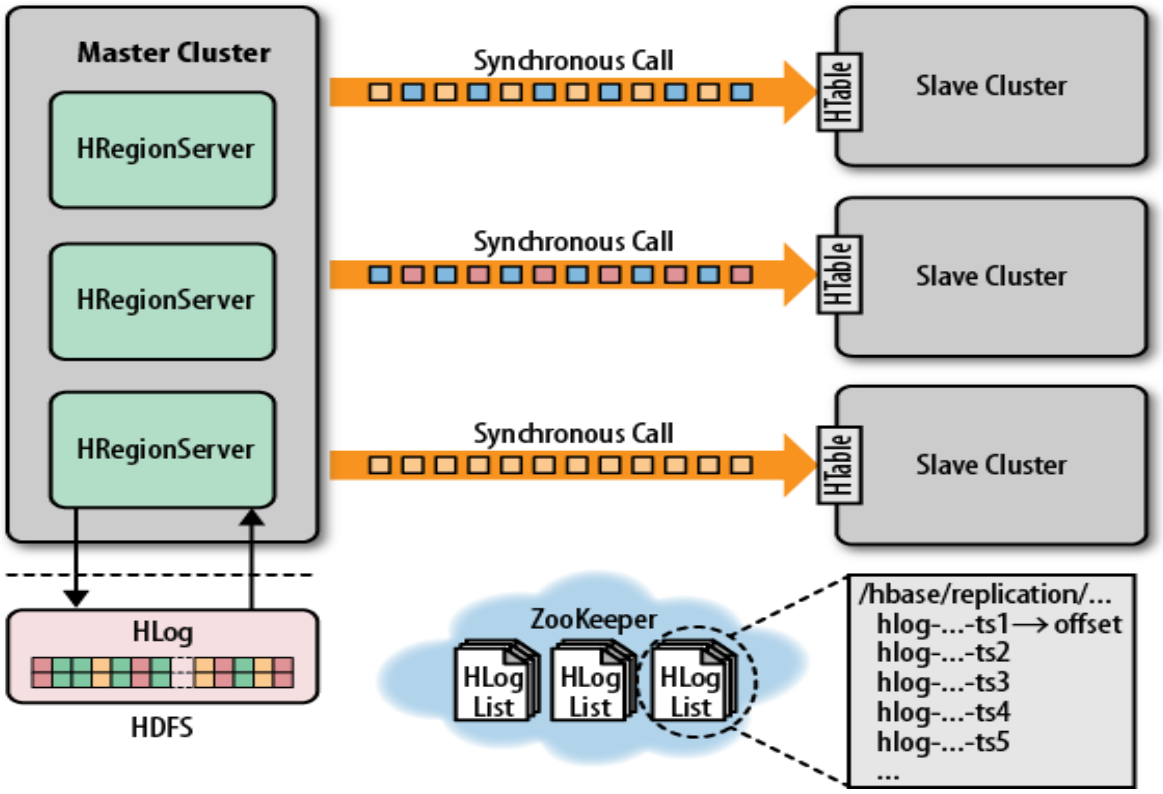
El código de selección de los cambios a replicar intercepta el flujo normal de edits hacia el WAL

Los Region Servers encolan los ficheros de log que se van rotando y hacen streaming de los cambios registrados en el WAL

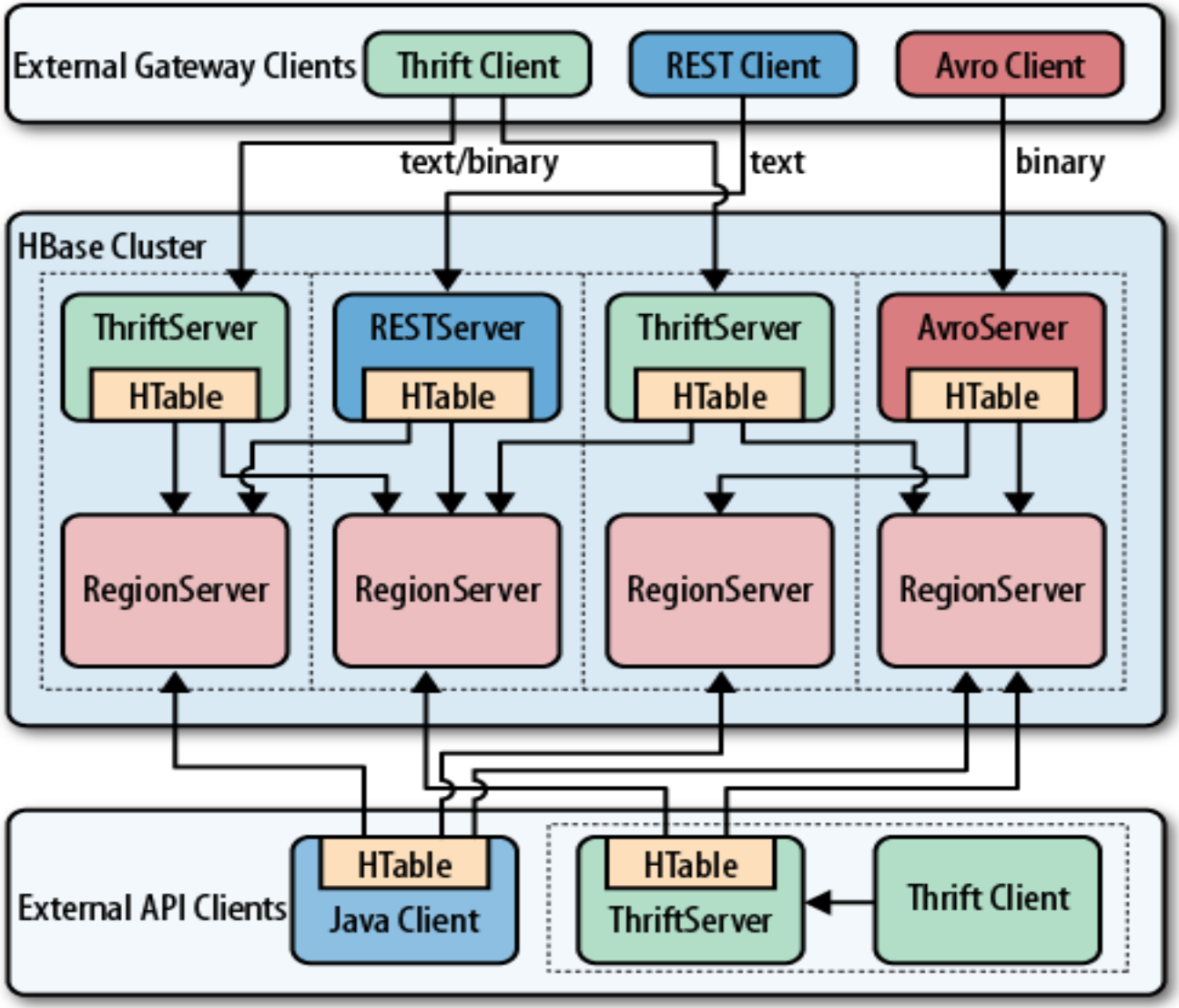
Tiene una granularidad de Column Family, es decir, se pueden sincronizar Column Families en concreto y dejar otras o bien se pueden sincronizar tablas completas

Replicación

El sistema es tolerante a fallos puesto que en caso de indisponibilidad las colas de logs se actualizan adecuadamente



Clientes de HBase



Clientes de HBase

Configuración

Está localizada en el fichero “hbase-site.xml”

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>hbase.zookeeper.quorum</name>
<value>zk1.foo.com,zk2.foo.com,zk3.foo.com</value>
</property>
</configuration>
```

Clientes interactivos

Java

- Protocolo RPC
- Disponible en: Java

Clientes de HBase

REST

```
hbase rest [start | stop]
hbase-daemon.sh [start | stop] rest
```

Protocolo poco óptimo ➔ baja performance
HTTP -> Interoperable ➔ Sistemas heterogéneos

Thrift (Facebook)

```
hbase thrift [start | stop]
hbase-daemon.sh [start | stop] thrift
```

Protocolos binarios compactos ➔ máxima performance
Disponible en: Java, C++, Perl, PHP, Python, Ruby,…

Clientes de HBase

Avro (Hadoop)

```
hbase avro [start | stop]
hbase-daemon.sh [start | stop] avro
```

Protocolos binarios compactos ➔ máxima performance
Disponible en: Java, C++, Perl, PHP, Python, Ruby,…

Otros

JRuby (HBase shell)
HBql (http://www.hbql.com)
HBql-DSL (https://github.com/nearinfinity/hbase-dsl/wiki)
JPA/JPO (http://www.datanucleus.org)
PyHBase (https://github.com/hammer/pyhbase)
AsyncHBase (https://github.com/stumbleupon/asynchbase)

Clientes de HBase

Clientes batch

Hive

Es un componente del ecosistema Hadoop que permite usar el lenguaje SQL para acceder a los datos almacenados en HDFS

Pig

Es un componente del ecosistema Hadoop que permite usar un lenguaje de scripting para acceder a los datos de las tablas y crear scripts que finalmente generan jobs mapreduce

Clientes de HBase

Clientes batch: Mapreduce

Java API

El API mapreduce para HBase permite lanzar jobs mapreduce usando HBase como `sink` y como `source`

Clojure (https://github.com/mudphone/hbase-runner/)

Permite acceder a las tablas de HBase desde Clojure

Cascading (http://www.cascading.org/)

Es un framework basado en Java para desarrollar procesos mapreduce sobre el ecosistema Hadoop

Clientes de HBase. Shell

Inicio de la shell

```
$ hbase shell
```

Inicio de la shell apuntando a un cluster diferente

```
$ HBASE_CONF_DIR="/<your-other-config-dir>/"
$ hbase shell
```