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

Programa experto en Big Data

Profesor: Ignacio Marrero Hervás

Sesión modelos, 1º Trimestre
Curso 2014-2015

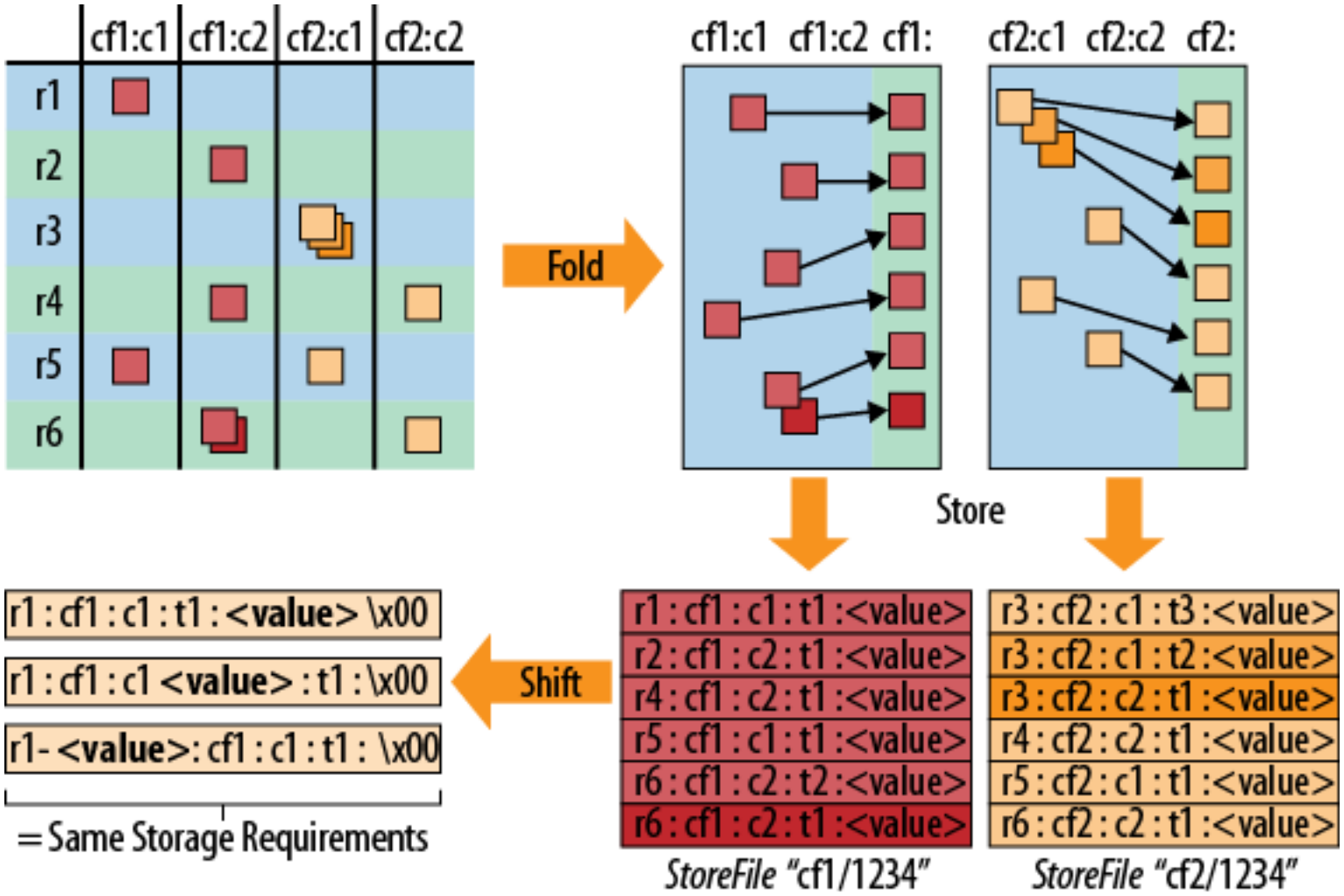
ÍNDICE

Modelo de datos. Conceptos

- La celda es la unidad lógica mas básica que integra una tabla
- La Column Family es la estructura física que más impacta en la organización y rendimiento del acceso a datos en una tabla
- El Timestamp de una celda permite el versionado
- Dos versiones de una celda físicamente son como dos celdas diferentes que se guardan secuencialmente en los Store Files
- Las diferentes celdas se guardan secuencialmente como objetos KeyValue en Store Files para cada Column Family

Modelo de datos. Almacenamiento

Organización física del modelo de datos



Modelo de datos. Diseño

- Objetivo del diseño de los modelos
  - Buena performance
  - Extensible
- Por performance se debe diseñar el modelo teniendo en cuenta la operación mayoritaria: lectura o escritura
- Los modelos deben priorizar el balanceo de carga entre Region Servers
- El diseño del modelo para el caso de lecturas mayoritarias debe ser de tal forma que:
  - Las búsquedas accedan al mínimo número de filas
  - Las búsquedas accedan al mínimo número de Store Files
  - Las búsquedas accedan al mínimo número de datos de una celda priorizados por performance: objeto Key, objeto Value

Modelo de datos. Diseño

- El rendimiento prima sobre la normalización del modelo
- Los joins no son una buena opción
- Acotar la versión de una celda afecta a la performance dado que se buscará en menor número de Store Files
  - Las celdas se almacenan por orden inverso de Timestamp
  - Se puede controlar la cantidad de versiones por Column Family con las funcionalidades TTL (Time To Live) o VERSIONS en las sentencias DDL y en el API
- Las operaciones de “Delete” sólo marcan la fila de tal forma que no se tenga en cuenta. Esta celda será borrada físicamente durante las compactaciones
  - HBase no está optimizada para muchos Deletes

Modelo de datos. Índices

Rowkey

Es el índice primario de las tablas. Puede ser tan complicada como sea necesario, pero debe ser mantenible para que sea operativa

Índices secundarios

Permiten acceder a los datos ordenados por un segundo índice adicional a la Rowkey

Modelo de datos. Índices

Formas de implementar índices secundarios

- Gestionada por el cliente. Generan problemas de transaccionalidad pero se pueden implementar usando
  - Column Family y Qualifiers
  - Tablas de mapeo
  - Coprocesors
- Implementaciones Open Source
  - Proyecto HIndex (https://github.com/Huawei-Hadoop/hindex)
  - Los proyectos “Indexed-Transactional HBase” e “Indexed Hbase” están obsoletos

Modelo de datos. Versionado

Existen dos modos de versionado

- Versionado por defecto. El sistema pone el Timestamp actual del servidor y permite sobrescribir este dato con un Timestamp propio
- Versionado personalizado. El cliente puede sobrescribir el Timestamp del dato con otro tipo de dato mientras sea “long”

Se puede controlar el número de versiones en los descriptores de las Column Families

- VERSIONS
- TTL

Modelo de datos. Rowkey

El diseño de la Rowkey tiene que favorecer la distribución de la carga entre los Region Servers

Técnicas de diseño de la Rowkey

1. Clave parcial

   Consiste en diseñar una Rowkey que permita un scan sin filtros simplemente indicando los parámetros “startrow” y “stoprow”, donde

```
stoprow = [byte menos significativo de startrow] + 1
```

Problemas de este tipo de aproximación

- Evolucionar el modelo puede ser muy complicado

Modelo de datos. Rowkey

2. Series temporales

- Colocar el timestamp a la derecha

  Se coloca un prefijo no secuencial y la parte menos significativa es el timestamp

```
Rowkey = <id>-<timestamp>
```

Problemas de este tipo de aproximación

  - Si el prefijo no permite balancear se puede descompensar el cluster

- Rowkeys salteadas

  Delante de la rowkey se coloca un prefijo que permite balancear

```
byte prefix = (byte) (Long.hashCode(timestamp) % <number of region servers>);
Rowkey = <prefix>-<id>-<timestamp>
```

Modelo de datos. Rowkey

- Rowkeys aleatorias

  Se genera un número aleatorio basado en el timestamp

```
Rowkey = MD5(timstamp)
```

Problemas de este tipo de aproximación

- No se pueden hacer querys por clave parcial

3. Modelos tridimensionales

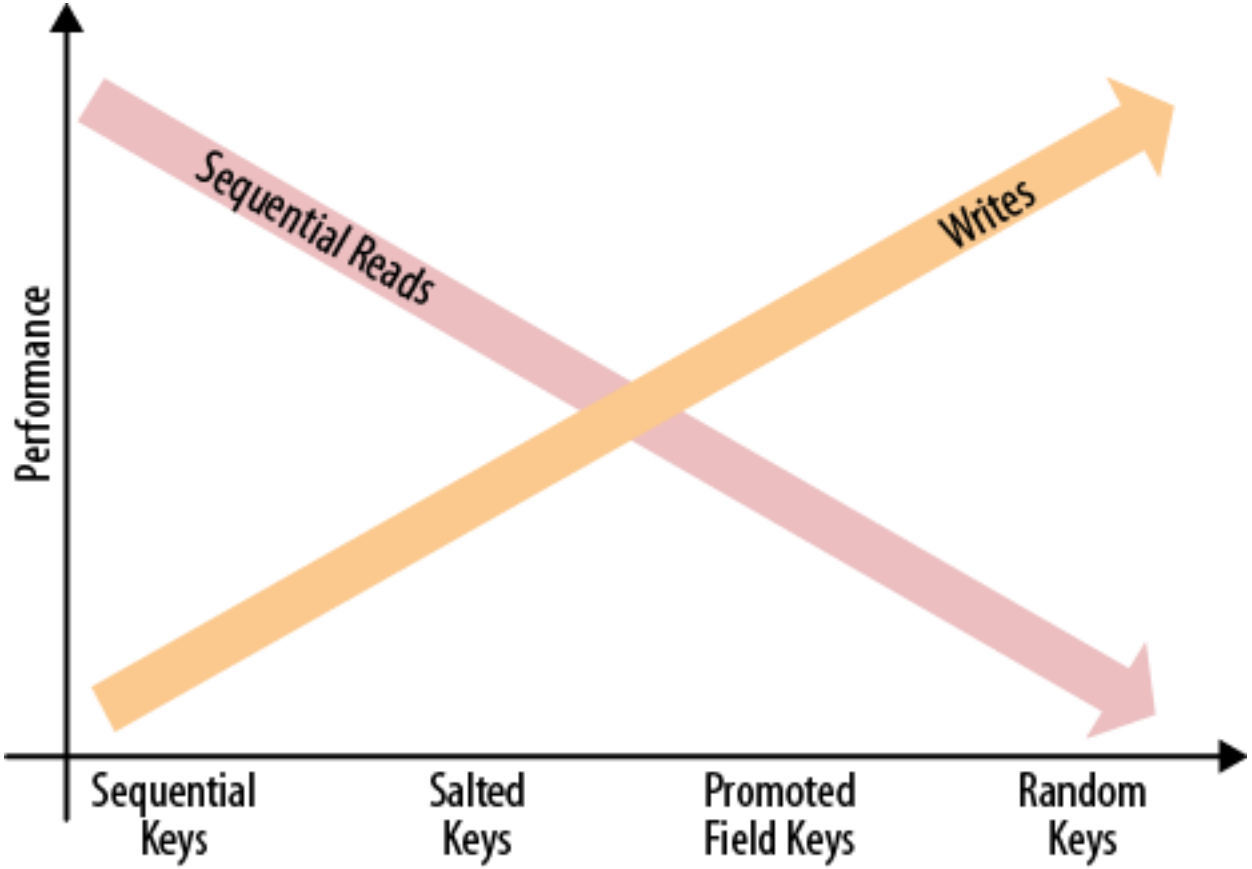
   Se trata de usar el timestamp para almacenar las versiones de cada celda

Problemas de este tipo de aproximación

- Si hay muchas versiones el modelo puede escalar mal si se generan Storage Files grandes

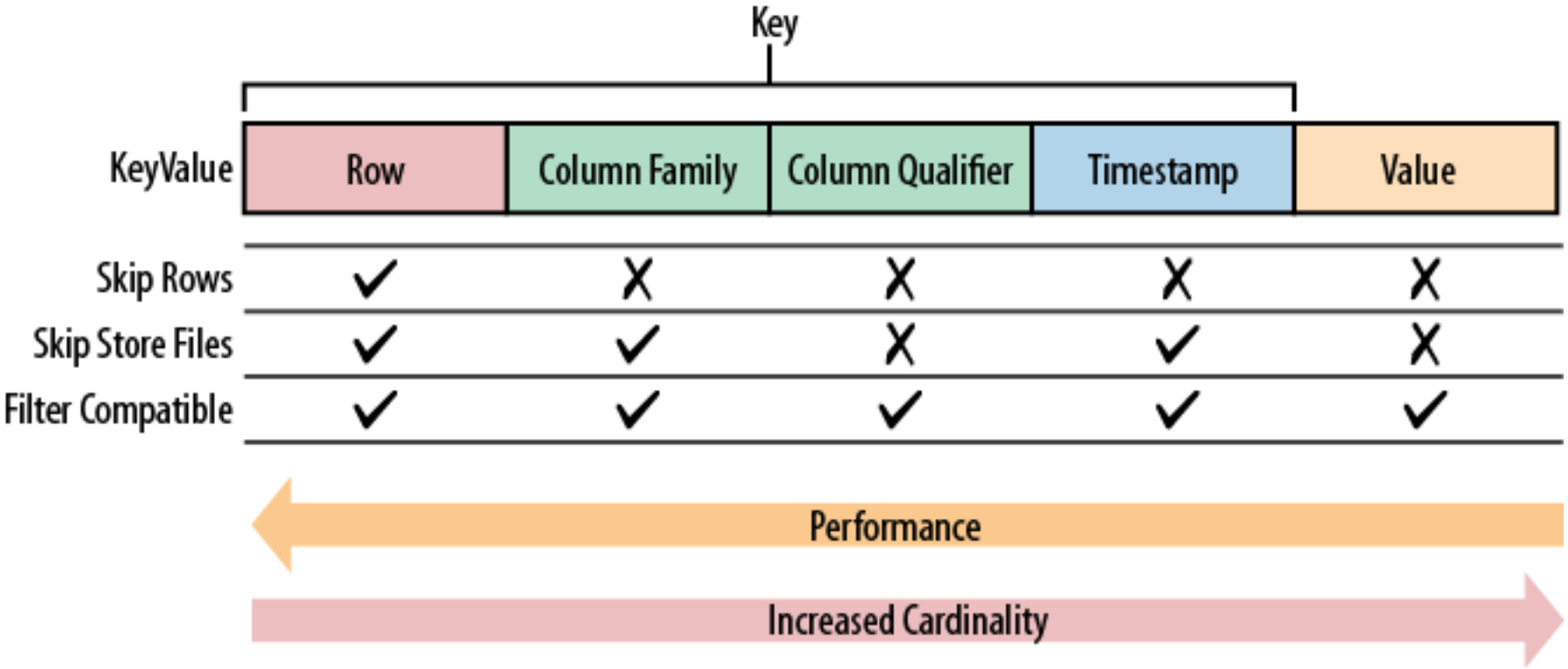
Modelo de datos. Performance

Gráfica que ilustra como plasmar en el diseño de la Rowkey la estrategia de performance



Modelo de datos. Performance

Esta gráfica ilustra la estrategia de performance a seguir en un modelo principalmente de lectura



Modelo de datos. Ejemplos

Caso de uso: servicio de e-mail

Nomenclatura = <rowkey> : <column family> : <qualifier> : <timestamp> : <content>
Modelo Flat-wide
`<userId> : data : <messageId> : <timestamp> : <email-message>`
Modelo Tall-Narrow
`<userId>-<messageId> : data : content : <timestamp> : <email-message>`

Varios modelos adecuados para querys por clave parcial

Rowkey	**Permite acceder a**
<userId>	Mensajes de un usuario
<userId>-<date>	Mensajes de un usuario para una fecha
<userId>-<date>-<messageId>	Partes de un mensaje dado para un usuario y fecha
<userId>-<date>-<messageId>-<attachmentId>	Adjuntos de un mensaje dado para un usuario y fecha

Bloom Filters. Conceptos

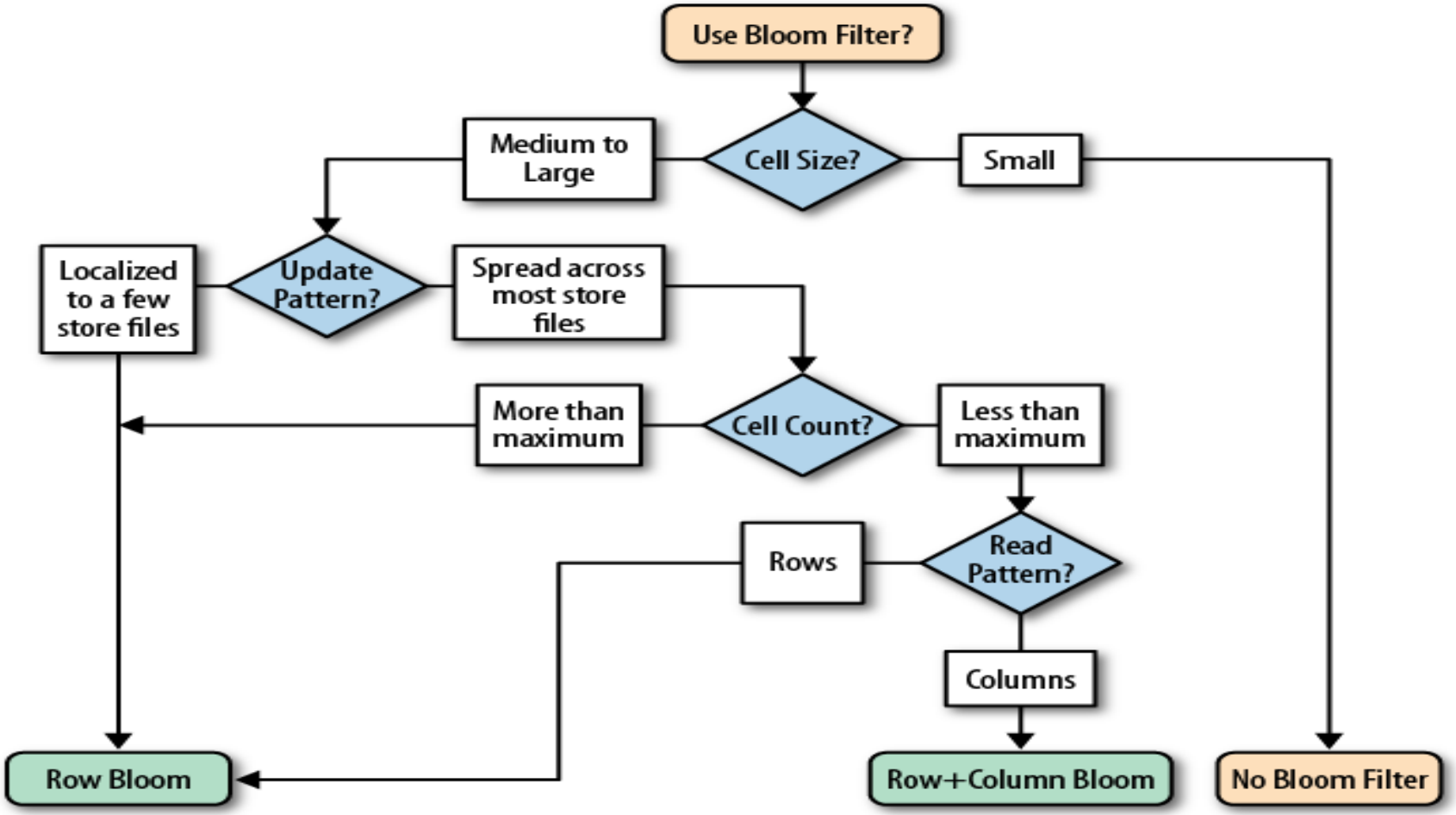
- Permiten saber si un determinado Store File contiene una determinada clave, por tanto optimiza el número de archivos que tienen que ser leídos y escaneados
- El filtro nunca proporcionará falsos negativos pero tiene un 1% de falsos positivos
- En ausencia del Bloom Filter el Region Server sólo tiene el HFile index para saber si una determinada clave está y por tanto es muy probable que tenga que cargar por completo todos los bloques de cada Store File de la Región y escanearlos
- La frecuencia de Update empeora el problema puesto que genera nuevos archivos que tendrán que ser escaneados

Bloom Filters. Activación

- Se activan en los descriptores de las Column Family en las sentencias DDL con BLOOMFILTER = {none, row, rowcol}
- Cada entrada del filtro ocupa 1 byte
- Hay dos tipos: fila, fila+columna. Para elegir el tipo adecuado se debe calcular si compensa, en función del caso de uso, la memoria que ocupa el filtro frente a la optimización conseguida

Bloom Filters

Diagrama que muestra la lógica de selección del tipo de Filtro



Transacciones

HBase es por su naturaleza atómico y transaccional cuando implica a una sola fila y no transaccional cuando implica varias filas

Soluciones (cliente) para implementar transaccionalidad

- Proyecto HAcid. Implementa transacciones multi fila (https://bitbucket.org/staltz/hacid/overview)
- Proyecto Datanucleus. Implementa JPA (http://www.datanucleus.org/products/accessplatform/datastores/hbase.html)

Otras soluciones podrían estar basadas en Zookeeper por su naturaleza de coordinador de procesos. Por ejemplo el proyecto Cages (http://code.google.com/p/cages/)

SQL

- Trafodion (https://wiki.trafodion.org/wiki/index.php/Main_Page)
- Phoenix (http://phoenix.apache.org)

Enlaces relevantes

Ampliación de conocimientos

- https://github.com/tdunning/YCSB
- http://hortonworks.com/blog/apache-hbase-region-splitting-and-merging/
- http://hortonworks.com/blog/hbase-blockcache-101/

HBase en producción. EBay

- http://files.meetup.com/1350427/EBAY-HBase-Ops.pptx

HBase en producción. Flurry

- http://www.slideshare.net/ddlatham/hbase-at-flurry

Cache

HBase usa un cache en memoria de tipo LRU que le permite acelerar el proceso de búsqueda y recuperación de los datos. Su tamaño total es

```
number of region servers * heap size * hfile.block.cache.size * 0.85
```

Por defecto "`hfile.block.cache.size=0.25`", es decir el 25% del heap total

Dentro del heap están los siguientes datos:

- Tabla .META.
- Índices de bloques de HFile: Rowkey y bloque
- Cache de bloques
- Filtros de Bloom