HIVE

Introducción

- Nace como una necesidad para aprender del gran volumen de datos de Facebook
- Utiliza características de SQL junto con Java para ejecutar consultas en grandes volúmenes de datos almacenados en HDFS
- Se considera una aplicación para procesar información genérica
- La habilidad para ejecutar consultas de sql contra los ficheros de datos lo que da a Hive su poder

Conceptos

- En un uso normal, Hive se ejecuta en la maquina de trabajo
- Convierte la consulta de SQL en trabajos para ejecutarlos en el cluster de Hadoop
- Hive almacena la información en tablas como medio de dotar de estructura a los datos almacenados en HDFS
- Los metadatos se almacenan en una base de datos llamada MetaStore

Introducción

- La configuración por defecto crea la base de datos en la máquina local por lo que no se puede compartir con otros usuarios.
- En entornos de producción, lo normal ese configura un metaestore remoto
- La instalación es sencilla, es necesario tener la misma versión de Hadoop instalada en la máquina de trabajo que en el cluster.
- La instalación de hadoop tiene que estar en la ruta indicada en la variable de entorno HADOOP_HOME

Introducción El Shell de Hive

- Hive dispone de un entorno de comandos o Shell.
- Utiliza el lenguaje HiveSql que es un dialecto de SQL basado en MySQL
- La sentencias tienen que acabar con “;” para que se ejecute
- Por ejemplo, el comando para mostrar las tablas al iniciarlo por primera vez Hive devuelve que no hay ninguna

```
hive> SHOW TABLES;
OK
Time taken: 0.473 seconds
```

Introducción El Shell de Hive

- Como los comandos de SQL no importa escribir en mayúsculas o minúsculas, salvo para la comparación de cadenas
- El tabulador completa comandos
- La primera vez que se ejecuta le cuesta un poco más para crear el MetaStore
- Almacena los archivos de la base de datos MetaStore en un directorio con el nombre metastore_db

Introducción El Shell de Hive

- Algunas opciones para los comandos
  - -f ejecuta los comando almacenados en el archivo indicado
  - -s no muestra los mensajes de información solo los resultados
  - -e permite ejecutar consultas cortas sin especificar el ; final

Introducción Instrucciones

- Create Table
  - Crear tabla indicando nombre, columnas, tipos de datos
  - Clausula Row Format, propia de HiveSql, permite identificar los delimitadores para los campos
  - Las filas las interpreta por el carácter fin de línea
  - Las tablas se crean como directorios dentro de la ruta de ejecución del Hive

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';
```

Introducción Instrucciones

- Load Data
  - Cargar información en hive
  - Copia el archivo especificado en la carpeta de la tabla
  - No intenta trasformar los datos para añadirlos a la tabla
  - No realiza ninguna modificiación en los archivos
  - Parámetro Overwrite indica que se borren los archivos existentes, si no se especifica añade los archivos a la tabla

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

Introducción Configuración

- Por defecto está configurado para trabajar en el archivo local
  - Propiedad fs.DefaultFS contienen file:///
- La ruta de trabajo se define en la propiedad hive.metastore.warehouse.dir
  - Por defecto es /user/hive/warehouse
- Siguiendo el ejemplo
  - Los archivos de la tabla record están en
    - % ls /user/hive/warehouse/records/ sample.txt

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

Introducción Instrucciones

- Select, ejecuta una consulta de SQL
- Internamente transforma la consulta en un trabajo y muestra los resultados en la consola de salida

```
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
    > GROUP BY year;
1949    111
1950    22
```

Configuración

- El archivo de configuración es Hive-site.xml que está en el directorio conf
- El archivo de documentación está en el mismo sitio con el nombre hive-default.xml
- Para ejecutar Hive con una configuración determinada utilizar – config indicando el directorio donde se encuentra el archivo hive.default.xml que queremos ejecutar

```
% hive --config /Users/tom/dev/hive-conf
```

configuración

- Para tener más de un usuario trabajando con Hive sobre el mismo cluster es necesario dar permisos de escritura al directorio de Hive a todos los usuarios

```
% hadoop fs -mkdir /tmp
% hadoop fs -chmod a+w /tmp
```

- Se p a configuración sólo para la ejecución de una consulta con SET, también sirve para consultar el valor

```
hive> SET hive.enforce.bucketing;
hive.enforce.bucketing=true
```

- Configuration+Properties

Proceso de Ejecución

- Originalmente Hive estaba creado para trabajar con MapReduce
- Actualmente puede trabajar también con Spark y TED que son más rápidos
  - Por ejemplo Spark permite a Hive especificar en el plan de ejecución que las salidas intermedias se almacenen en memoria
- La propiedad hive.execution.engine por defecto está definida a mr (MapReduce)

```
hive> SET hive.execution.engine=tez;
```

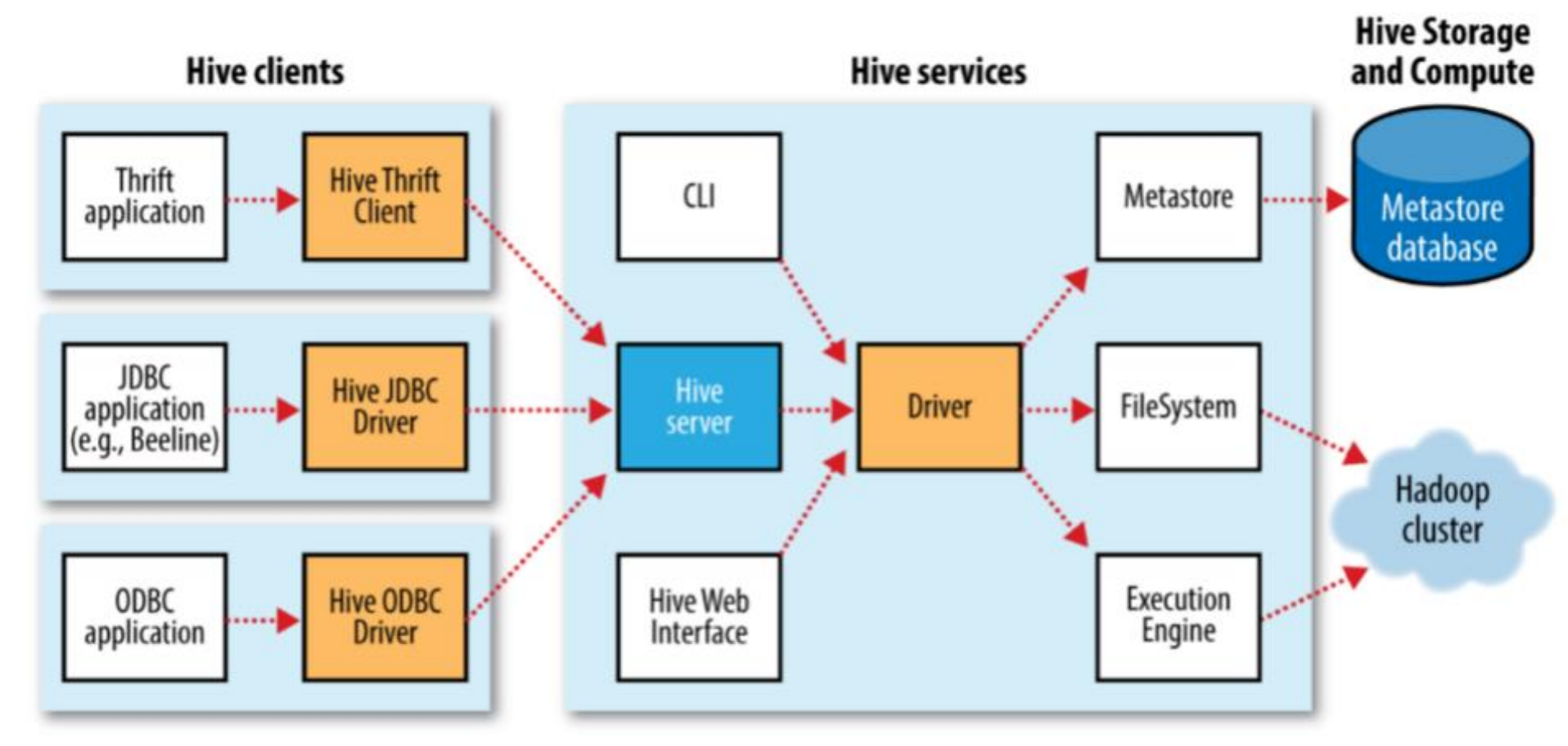
Arquitectura hive

- Dispone de varios servicios que puede ejecutar.
  - CLI: Shell de línea de comandos
  - HIVESERVER2, expone un servicio Thrift para permitir el acceso desde clientes en diferentes lenguajes. Propiedad hive.server2.thrift.port indica el puerto donde escucha
  - Hwi, es un interface web para versiones de Hive anteriores a 2.2 Una alternativa es WebHCat (rest API) para intergrar con otras aplicaciones
  - Jar, permite ejecutar aplicaciones java que incluyan clases de hadoop y hive

Arquitectura Hive

- SI se ejecuta Hive como servidor (hive –service hiveserver2) hay múltiples clientes que pueden consumir sus servicios
- Thrift: cualquier lenguaje de programación que soporte este cliente pued interactuar con el servidor (https://thrift.apache.org/)
- JDBC Driver: ejecuta un driver java en un proceso separado
- ODBC Driver: permite a aplicaciones de BI conectarse. No viene incluido

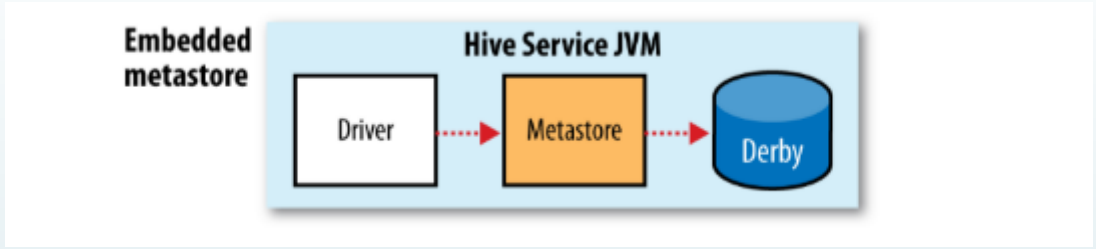
Arquitectura hive

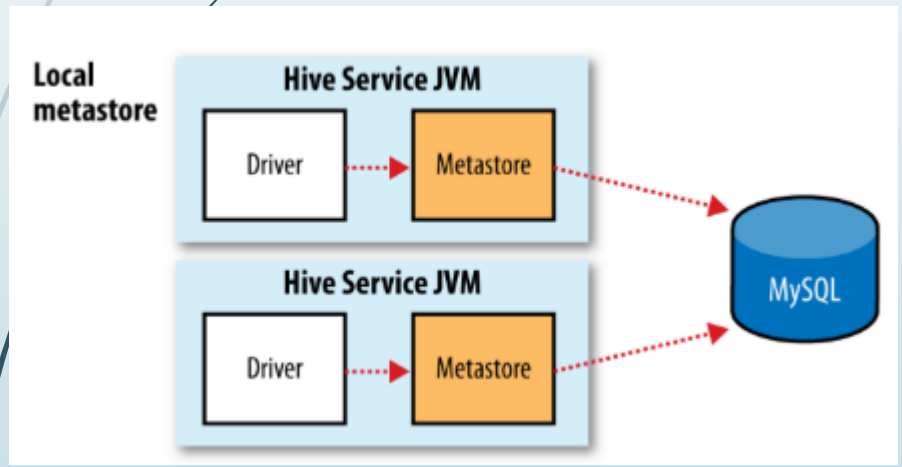


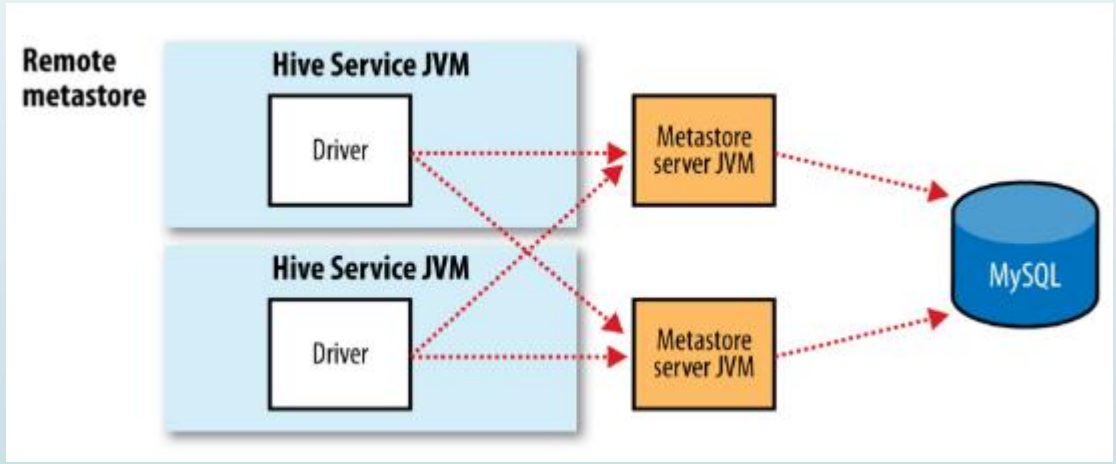
metastore

- Es el almacén central de metadatos
- Ejecuta un servicio normalmente en la misma JVM que Hive que contiene una base de datos embebida (https://db.apache.org/derby/)
- También ejecuta un proceso de backup en disco para la base de datos
- Por defecto sólo una base de datos Derby puede acceder a los archivos de disco almacenados en la base de datos, por lo que no vale para entornos multiusuario
- La solución es utilizar una base de datos independiente en vez de la embebida. Sirve cualquier base de datos JDBC

metastore


Embedded
metastore
Hive Service JVM
Driver
Metastore
Derby


Local
metastore
Hive Service JVM
Driver
Metastore
Hive Service JVM
Driver
Metastore
MySQL


Remote
metastore
Hive Service JVM
Driver
Hive Service JVM
Driver
Metastore
server JVM
Metastore
server JVM
MySQL

metastore

- MySql se suele utilizar como base de datos independiente para MetaStore
- Configurar propiedades
  - javax.jdo.option.ConnectionU=jdbc:mysql://host/dbnam?createDatabaseIfNotExist=true
  - javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
  - Usuario y contraseña
- La ruta del archivo para el driver JAR tiene que estar en la variable classpath

metastore

hive.metastore.warehouse.dir	URI	/user/hive/warehouse	The directory relative to fs.defaultFS where managed tables are stored.
hive.metastore.uris	Comma-separated URIs	Not set	If not set (the default), use an in-process metastore; otherwise, connect to one or more remote metastores, specified by a list of URIs. Clients connect in a round-robin fashion when there are multiple remote servers.
javax.jdo.option.ConnectionURL	URI	jdbc:derby:;databaseName=metastore_db;create=true	The JDBC URL of the metastore database.
javax.jdo.option.ConnectionDriverName	String	org.apache.derby.jdbc.EmbeddedDriver	The JDBC driver classname.
javax.jdo.option.ConnectionUserName	String	APP	The JDBC username.
javax.jdo.option.ConnectionPassword	String	mine	The JDBC password.

Comparación con bases de datos

- Aunque tiene muchas diferencias ya que está basado en HDFS y procesamiento con MapReduce, cada vez está evolucionando para tener un comportaminto más parecido a las bases de datos tradicionales
- Aspectos a comparar
  - Esquema en la lectura vs Esquema en la escritura
  - Actualizaciones, transacciones e índices
  - SQL on Hadoop alternativas

Comparación con bases de datos

- Esquema en la lectura vs Esquema en la escritura
  - En una base tradicional se carga el esquema en la carga de los datos, si los datos no encajan en el esquema entonces son rechazados. Este esquema se llama esquema en la escritura ya que se aplica antes de escribirlos en la base de datos. → Hace la carga más lenta
  - Hive comprueba el esquema cuando va a aplicar la consulta → Cargas de datos más rápidas

Comparación con bases de datos

- Updates, transacciones e Índices
  - Aunque son elementos clave en sistemas tradicionales en Hive al basarse en HDFS tiene una forma totalmente distinta de ejecutarse por lo que la comparación es difícil.
  - En el caso de las actualizaciones, se guardan en un archivo temporal y se ejcutan trabajos en backgroud para mezclarlos con los bloques que correspondan
  - Transacciones no están soportadas en Hive

Comparación con bases de datos

- Comparando con Impala, la ejecución de consultas utiliza un servicio que está continuamente en ejecución, lo que agiliza el procesamiento
- Hive utiliza MapReduce, lo que hace el proceso más lento ya que tiene que ejecutar la aplicación maestra cada vez que se inicia, aunque el procesamiento con TEZ ha sido una mejora.

hiveSQL

- Es una mezcla de SQl-92, MySQL y Oracle SQL
- Comparando las característi
HiveSQL c

Feature	SQL	HiveQL
Updates	UPDATE, INSERT, DELETE	UPDATE, INSERT, DELETE
Transactions	Supported	Limited support
Indexes	Supported	Supported
Data types	Integral, floating-point, fixed-point, text and binary strings, temporal	Boolean, integral, floating-point, fixed-point, text and binary strings, temporal, array, map, struct
Functions	Hundreds of built-in functions	Hundreds of built-in functions

Hive SQL

- Continuando con la comparación de funcionalida

Multitable inserts	Not supported	Supported
CREATE TABLE...AS SELECT	Not valid SQL-92, but found in some databases	Supported
SELECT	SQL-92	SQL-92. SORT BY for partial ordering, LIMIT to limit number of rows returned
Joins	SQL-92, or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins, cross joins
Subqueries	In any clause (correlated or noncorrelated)	In the FROM, WHERE, or HAVING clauses (uncorrelated subqueries not supported)
Views	Updatable (materialized or nonmaterialized)	Read-only (materialized views not supported)

Particiones y cubos

- Hive organiza las tablas en particiones para mejorar el rendimiento en la búsqueda de información
- Las particiones se pueden dividir en cubos, permitiendo búsquedas más complejas.
  - Impone estructura extra a las tablas
  - Permite especificar que columnas queremos empaquetar y en cuantos elementos

Formatos de almacenamiento

- Hive tiene en cuenta dos dimensiones, el formato de fila y el de fichero
- El formato de fila indica como se almacenan las filas y los campos dentro delas filas
- Es importante para el proceso de SerDe (serialización deserialización) a la hora de realizar las consultas
- El formato de fichero define el formato del contenedor

Formatos de almacenamiento

- Texto Delimitado, es el formato por defecto
  - El delimitador de fila es el carácter CTRL A
  - El delimitador de campo es el carácter CTRL B
- Formatos de almacenamientos binarios: Sequence files, Avro datafiles, Parquet files, RCFiles, and ORCFiles
- Formatos personalizados SerDe

Joins

- Hive permite realizar joins con sintaxis muy parecida a la de SQL tradicional
- Los Join se transforman en trabajos de MapReduce que procesan la unión
- Podemos utilizar la clave Explain, para que MapReduce devuelva el plan de ejecución

```
EXPLAIN
SELECT sales.*, things.*
FROM sales JOIN things ON (sales.id = things.id);
```

Hive

- La funcionalidad de HIVE permite trabajar con sentencias similares a SQL facilitando el análisis de la información
- Quedaría pendiente
  - Queries
  - Funciones definidas por el usuario
  - Importación de datos
  - Optimización de escripts de MapReduce

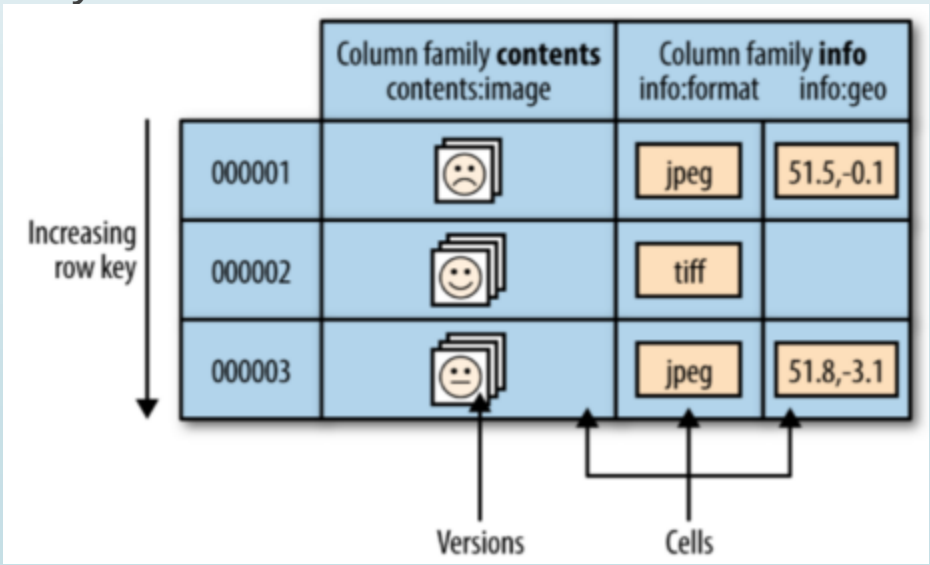
Hbase

Introducción

- Hbase es una base de datos distribuida organizada en columnas y basada en HDFS
- Permite acceso aleatorio a datos en grandes conjuntos de datos
- La aproximación de HBASE a los problemas de escalado se basa en ir añadiendo nodos cuando se necesite
- No es una base de datos relacional y no soporta SQL
- Parte del proyecto BigTable de Google de 2007. Empezó como un elemento de Hadoop hasta convertise en un proyecto independiente

Conceptos

- Las aplicaciones almacenan información en tablas etiquetadas
- Las tablas están hechas por filas y columnas
- Las celdas tienen versiones. Una marca de tiempo automática cuando se inserta información
- Las celdas contienen un array de bytes



Conceptos

- La key de la fila es también un arrray de Bytes
- Como las claves son un array de bytes, todo tipo de datos puede servir como campo clave
- Las filas se ordenan por el campo clave
- Todas las tablas se acceden a través de su clave primaria

Conceptos

- Las columnas se agrupan en familias de columnas
- Todas las columnas de una familia tienen un prefijo, separado por “:”
  - Info:geo, info:format son de una familia y contents:image de otra
- Físicamente, las familias de columnas se almacenan juntas.
- Es más correcto definir HBASE como un sistema de bases de datos orientado a familias de columnas que ha columnas
  - Las modificaciones de la configuración se hacen a nivel de familia no ha nivel de columna
- Si la familia de columna existe, se pueden añadir nuevas columnas en cualquier momento

Conceptos

- Las tablas se dividen automáticamente en regiones, cada región tiene un subconjunto de filas.
- La región se identifica por la primera y la última fila
- Inicialmente la tabla tiene una región, pero cuando crece por encima del limite que esté configurada se divide en dos regiones de tamaño similar.
- Las regiones son la unidad que se distribuye en un cluster HBASE. El conjunto ordenado de todas las regiones constituyen la tabla
- El bloqueo es a nivel de fila, no a nivel de tabla

Estructura

- Tiene una filosofía similar a HDFS y YARN (NameNode y DataNode, Administrador de Recursos y Administrador de Nodos)
- Hbase está formado por un nodo Hbase Master gestionando el cluster de uno o más Servidores de Regiones

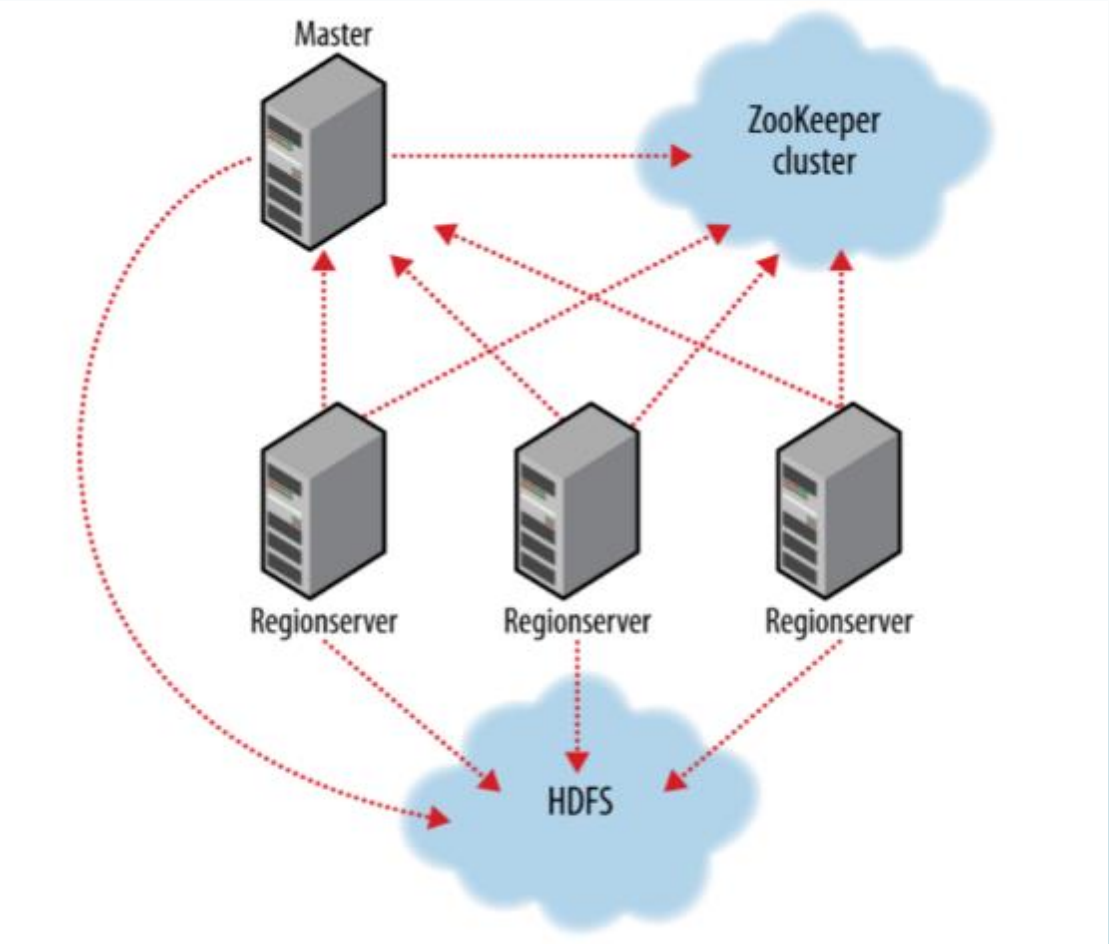
Estructura

- HBase Master es responsable de
  - la división de la base de datos inicial
  - Asignar regiones a servidores de regiones
  - Recuperar servidores de regiones que puedan fallar
- Los Servidores de regiones
  - Pueden tener 0 o más regiones
  - Contienen las peticiones de los clientes de lectura y escritura
  - Informa al HBase Master de las regiones hija

Estructura

- HBase depende de Zookeeper para gestionar la ubicación de los elementos
- Zookeeper mantiene la información de la ubicación del catálogo de tablas así como del HBAse Master.
- La asignación de las regiones se realiza mediante Zookeeper, de tal manera que si hay algún problema permite al sistema seguir los pasos y continuar dónde se quedó
- A las conexiones clientes de HBASE se les pasa una instancia de Zookeeper para que pueda seguir por la jerarquía y encontrar los nodos con los atributos

Estructura


Master
ZooKeeper cluster
Regionserver
Regionserver
Regionserver
HDFS

Estructura

- HBase utiliza el sistema de archivos de Hadoop para almacenar información.
- Por defecto almacena la información en el sistema de archivos local
- En entornos de producción hay que modificar la configuración para que trabaje con HDFS
- Intenamente, HBase tiene una tabla especial de catálogo hbase:meta
  - Contiene la lista, estado y ubicación de todas las regiones
  - Contiene una clave con el nombre de la tabla, región, filas, horas y hash MD5

Hbase shell

- Create: crea una tabla
  - Create ‘test’ ,’data’ → Crea la tabla test con una columnas data con la configuración por defecto
- List: Lista las tablas existentes
- Put: introducir datos en la tabla
  - Put ‘test’,’row1’,’data:1’, ‘value’ → En la tabla Test, crea una columna nueva Data:1 y le introduce en el la primera fila el valor “value”
- Get: obtiene la información de la columna y la celda

HBase Shell

- Para eliminar una tabla primero tenemos que desactivarla
- Disable para desactivarla
- Drop para borrarla
- Los comandos de Shell se pueden ejecutar desde Java

Hbase Java

- Las clases de HBase están en los namespaces org.apache.ha doop.hbase y org.apache.hadoop.hbase.client
- Clases básicas
  - HBaseConfiguration → permite crear un objeto configuración que lee la parametrización de hbasesite.xml y hbase-default.xml
  - HBaseAdmin → Permite gestionar el cluster de Hbase, añadir y quitar tablas
  - HTable → Permite acceder a una tabla específica
- ConnectionFactory → permite crear objetos Connection con los métodos getAdmin GetTable

Hbase Java

- Los objetos SCANNER son como los cursores en bases de datos tradicionales o Java iterators
  - Tienen que cerrarse después de su uso
  - Devuelven las filas en orden
  - Se generan desde un objeto Table.getScanner
  - Se pasa la fila a partir de la cual trabajar
  - Qué columnas tiene que devolver
  - Y filtro a ejecutar en el lado del servidor
  - Por defecto devuelve de 100 en 100 filas (configurable hbase.cli ent.scanner.caching)

Hbase Java administracion

- Ejemplo de código

HBASE java Mapreduce

- Dispone de clases para utilizar HBase como origen o destino de trabajos de MapReduce en el namespace org.apache.hadoop.hbase.mapreduce
- TableInputFormat → Permite crear Splits de regiones para los trabajos Maps
- TableOutputFormat → escribe el output de reduce en HBase

HBASE java mapreduce

- Ejemplo de código

Hbase vs rdbms

- HBASE está pensado para grandes volumenes de datos, millones de filas y millones de columnas, se divide en regiones que se distribuyen por miles de nodos automáticamente. El esquema de la tabla es un espejo del archivo físico.
- RDBMS, siguen un esquema fijo, orientado en tablas, pensando en la abstracción e integridad de los datos. Utiliza consultas estructuradas en SQL, permite fácilmente crear índices secundarios, joins, agrupaciones
- Para la mayoría de las empresas por la madurez de la tecnología RDBMS es más que suficiente.
- Si el problema es la escalabilidad del sistema, entonces HBase puede ser una opción

Hbase vs rdbms

- No tiene índices
  - Las filas se almacenan secuencialmente, no hay problemas con el mantenimiento de índices
  - Los tiempos de inserción son independientes del tamaño de la tabla
- Particionamiento automático
  - Conforme crecen las tablas, son automáticamente divididas y repartidas por los servidores de regiones
- Escalado lineal con nuevos nodos
  - Sólo hay que añadir un nuevo nodo, apuntarlo al cluster y el servidor de regiones se reajusta

Hbase vs RDBMS

- Hardware común
  - Equipos con coste menor que servidores
- Tolerancia a fallos
  - Al tener múltiples nodos, la caída de uno no afecta al rendimiento
- Proceso por lotes
  - La integración con MapReduce permite procesamiento paralelo de información

Mongodb

conceptos

- Documento es la unidad básica de datos, sería similar a una fila
- Colección, se podría asimilar a una tabla con un esquema dinámico
- Una sola instancia de MongoDB puede tener multiples bases de datos independientes, y cada una puede tener sus propias colecciones
- Cada documento tiene una clave especial “_id” que es única en una colección
- Dispone de un Shell de comandos con javascript para administración

conceptos

- No soporta transacciones, se puede simular pero no existe una estructura que obligue a obedecer las clausulas semánticas
- Unir diferentes tipos de datos mediante varias dimensiones
- El número de aplicaciones diseñadas para trabajar con MogoDB está creciendo pero no son tantas como otras plataformas

Documentos

- Es un conjunto ordenado de conjuntos de parejas key/value.
- Cada lenguaje de programación lo representa de forma diferente, pero en la mayoría de los casos coincide con un diccionario.

```
{"greeting" : "Hello, world!"}
```

```
{"greeting" : "Hello, world!", "foo" : 3}
```

Documentos

- La clave puede contener cualquier UTF-8 carácter
- No se puede utilizar Null ya que lo utiliza para marcar el final del campo clave
- Los caracteres “$” y “.” tienen propiedades especiales en algunas circustancias, por lo que se consideran también reservados
- MongoDB es case-sensitive {“foo”:3} y {“Foo”:3} son diferentes
- Las key en un documento no pueden estar repetidas

Documentos

- Las parejas Key/value en un documento están ordenadas
- Normalmente los lenguajes de programación no necesitan el orden de los documentos
- {x:1;y:2} es distinto a {y:2; x:1}

Colecciones – Esquemas dinámicos

- Los documentos de una misma colección pueden tener diferentes formas
  - Pueden contener diferentes tipos para los valores y diferentes keys

```
{"greeting" : "Hello, world!"}
{"foo" : 5}
```

Colecciones – esquemas dinámicos

- Si podemos tener diferentes tipos de documentos, ¿porqué necesitamos diferentes colecciones?
  - Mantener documentos de diferentes tipos puede ser una locura para garantizarnos que las consultas devuelven documentos de determinado tipo
  - Es más rápido obtener una lista de colecciones de diferentes tipos que una lista de tipos dentro de una colección para hacer la consulta al tipo correcto
  - Agrupar documentos del mismo tipo en una colección, permite una mejor ubicación de la información
  - Los índices son por colección, si tenemos documentos de mismo tipo, serán más eficientes

Colecciones - Nombres

- Al igual que los documentos, las colecciones pueden contener los caracteres UTF-8 con excepciones
  - La cadena vacía “” no se puede utilizar como nombre
  - Al igual que los documentos no puede contener el null (\0)
  - No se deben crear colecciones que empiecen con system. Es un prefijo reservado para colecciones internas
  - No se debe usar el “$”, algunas colecciones de sistema lo utilizan pero no colecciones de usuario

Colecciones - Subcolecciones

- Una forma de organizar colecciones es utilizar un espacio de nombres y el grupo separadas por “.”
  - Blog.posts y Blog.autores pueden ser dos subcolecciones de documentos
- Es únicamente para organizar la información, no hay ninguna relación entre las colecciones ni con la colección “padre” (Blog en el ejemplo, que no tiene por qué existir)
- Herramientas que trabajan con Mongo facilitan el uso de las subcolecciones, por ejemplo GridFS

Bases de datos

- Las bases de datos son agrupaciones de colecciones.
- Una instancia de MongoDB puede tener varias bases de datos y cada una puede tener 0 o más colecciones
- Cada base de datos tiene diferentes permisos y se almacena en diferentes sistemas de archivos en el disco
- Es recomendable almacenar los datos de la misma aplicación en una base de datos

Bases de datos

- Las bases de datos se identifican por nombre
- Los nombres siguen los mismos criterios que en los casos anteriores UTF-8 con excepciones
  - “” no se puede utilizar
  - No puede contener /, \, ., '', *, <, >, :, |, ?, $, un espacio en blando single space), o \0 (character null).
  - Los nombres son case-sensitive, incluso en sistemas de archivos que no lo sean
  - El máximo son 64 caracteres
- Cada base de datos acaba siendo un archivo, por eso alguna de las restricciones

Bases de datos

- Hay bases de datos especiales propias del sistema
- admin: en términos de autenticación es la base de datos “root”. Al agregar un usuario a esta base de datos, automáticamente hereda permisos para todas las bases de datos
- local: esta base de datos no se replica, puede contener colecciones locales a un solo servidor
- config: cuando se utiliza la base de datos en una configuración divida, esta base de datos almacena la configuración

Bases de datos

- Los namespaces se consideran como el nombre totalmente cualificado de una colección
- Se obtienen concatenando el nombre de la base de datos con el nombre de la colección
- Tiene limitación de longitud, en teoría a 128 bytes
- Por ejemplo la colección “blog.posts” dentro de la base de datos “cms”, su namespace sería “cms.blog. posts

ejecución

- Normalmente se ejecuta como servidor, de tal manera que los clientes pueden conectarse y realizar operaciones.
- Por defecto utiliza el puerto 27017, si no está disponible no se puede ejecutar
- Ejecuta también un servidor HTTP en un puerto 1000 direcciones superior al que esté utilizando, normalmente 28017 → http://localhost:28017
- Se puede acceder a información administrativa

Shell

- Desde el Shell de comandos se pueden realizar labores administrativas, inspeccionar una instancia o hacer pruebas sencillas
- Es un interprete de javascript, por lo que podemos ejecutar scripts

```
> Math.sin(Math.PI / 2);
1
> new Date("2010/1/1");
"Fri Jan 01 2010 00:00:00 GMT-0500 (EST)"
> "Hello, World!".replace("World", "MongoDB");
Hello, MongoDB!
```

Shell - cliente

- Realmente el Shell es un cliente de MongoDB
- Al iniciarse se conecta a la base de datos “test” y la asigna a la variable global “db”
- Si escribimos db en el Shell devuelve la base de datos activa en el Shell
- Para cambiar de base de datos, utilizar el comando “use” <nombre bd>
- Si escribimos “db”.<colección> devuelve la lista de colecciones de la base de datos.

Shell – comandos básicos

- Podemos ejecutar los comandos CRUD (Create, Read, Update y Delete)
- Método Insert() añade un documento a una colección de la base de datos
- Método find() muestra el contenido de una colección

```
> post = {"title" : "My Blog Post",
... "content" : "Here's my blog post.",
... "date" : new Date()}
{
    "title" : "My Blog Post",
    "content" : "Here's my blog post.",
    "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

```
> db.blog.insert(post)
```

Shell - comandos básicos

- Leer información con find() o findOne()
- Permite hacer filtros para seleccionar los documentos
- Por defecto devuelve hasta 20 documentos que cumplan el criterio

```
> db.blog.find()
{
    "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
    "title" : "My Blog Post",
    "content" : "Here's my blog post.",
    "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

Shell – comandos básicos

- Update permite actualizar un documentos
- Necesita al menos dos parámetros
  - El criterio para encontrar el documento
  - El nuevo documento

Shell – comandos básicos

- Ejemplo añadir una nueva key al post con un vector de comentarios

```
> post.comments = []
```

```
> db.blog.update({title : "My Blog Post"}, post)
```

```
> db.blog.find()
{
    "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
    "title" : "My Blog Post",
    "content" : "Here's my blog post.",
    "date" : ISODate("2012-08-24T21:12:09.982Z"),
    "comments" : [ ]
}
```

Shell – comandos básicos

- Para borrar permanentemente un documento de la base de datos, utilizar el comando Delete
- Si se ejecuta sin parámetros, borra todos los documentos de la colección
- Si pasamos como parámetro una key, borra el documento

```
> db.blog.remove({title : "My Blog Post"})
```

Tipos de datos

- Los documentos se pueden considerar conceptualmente similares a los archivos JSON
- Los tipos de datos básicos que utiliza son los mismos null, boolean, numeric, string, array, y object, ampliados con algunos como Date, Regularexpression, documentos embebidos o NumberInt

Mongodb – Por ver

- Trabajar con funciones CRUD
- Definir queries y cursores
- Crear índices para las aplicaciones, simples, avanzados, geoespaciales
- Analizar aplicaciones, comandos de agregación, soporte para MapReduce
- Replicación, sincronización, copias de seguridad
- Administración y monitorización
- Sharding – división de la base de datos