HBase

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías da Información (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022

Guion

- Introducción
- Modelado de Datos
- Arquitectura de componentes

Introducción

- Almacén de datos (data store) y no base de datos
  - No proporciona funcionalidad requerida a un Sistema Gestor de Bases de Datos
    - Tipos de datos para las columnas
    - Índices secundarios
    - Disparadores
    - Lenguaje de consulta avanzado
- Escalabilidad horizontal es la motivación principal
- Características
  - Lecturas y escrituras consistentes.
  - Sharding automático.
  - Recuperación ante fallos en los nodos
  - Integración con Hadoop/HDFS
  - Paralelización del procesamiento con MapReduce
  - APIs de cliente: Java, Thrift/REST (para entornos no-Java)
  - Caché de bloques y Bloom Filters
  - Gestión operacional

Introducción

- Cuando usar HBase
  - Gran volumen de datos (cientos o miles de millones de filas)
  - No se necesitan todas las características de un Sistema Gestor de Bases de Datos
  - Arquitectura hardware en forma de cluster.
    - HDFS necesita al menos 5 nodos para funcionar bien.
    - La configuración standalone (una sola máquina) es solo para desarrollo
- Diferencia con Hadoop/HDFS
  - HDFS: sistema de archivos distribuidos.
    - No da soporte a procesos de búsqueda rápida de registros.
  - HBase diseñado para búsquedas y modificaciones rápidas en archivos muy grandes

Modelo de datos

- Modelo de tablas con filas y columnas, pero no comparable con el modelo relacional
  - Realmente lo que proporciona HBase es un modelo anidado de mapeos (maps)

Claves y valores son arrays de bytes. Sin tipo.

cell

row	column family	column	timestamp	value
r1	cf1	c11	t1	v1
		c12	t1	v2
		c13	t1	v3
	cf2	c21	t2	v4
		c22	t3	v5
r2	cf1	c11	t1	v6
		c21	t2	v7
	cf2	c25	t1	v8

Modelo de datos

- Tabla (table): varias filas
- Fila (row)
  - Clave de fila + una o varias columnas con valores asociados
  - Almacenadas en orden por la clave de fila
    - Diseño de la clave muy importante en el rendimiento
    - Intentar almacenar filas relacionadas (recuperadas juntas) cerca
- Columna (columna): identificador de una familia de columnas seguido de un calificador de columna
- Familia de columnas (columna family - CF)
  - Permiten almacenar varias columnas juntas
  - Cada fila debe de tener las mismas familias de columnas
  - El número de calificadores de las columnas de cada familia puede variar de una fila a otra.
- Celda: Combinación de fila, CF y columna.
  - Almacena un valor y una marca de tiempo (timestamp)
- TimeStamp: Identifica a una versión de una celda. Se guarda el tiempo de escritura en el servidor. Puede cambiarlo el usuario.

Modelo de datos

- Físicamente, las tablas se almacenan por familias de columnas
- Si no se especifica una marca de tiempo en la consulta, se obtiene la más reciente
- Espacio de nombres (namespace)
  - Similar al concepto de base de datos en un SGBDs
  - Permite
    - Gestionar la quota
    - Administrar seguridad
    - Enlazarlo a un grupo servidores concreto
  - Se pueden crear, borrar y alterar.
  - Predefinidos
    - hbase: espacio de nombres para tablas del sistema
    - default: espacio de nombres para almacenar tablas sin espacio de nombres

Modelo de datos

- Operaciones
  - GET
    - Obtiene los datos de una fila concreta
  - PUT
    - Añade o modifica filas
  - SCAN
    - Itera sobre las filas de una tabla
  - DELETE
    - Borra una fila de una tabla
      - Realmente se marcan las filas como borradas para ser eliminadas en la próxima compactación de datos

Modelo de datos

- Versiones
  - Puede haber varias versiones del mismo dato en la misma celda
  - La versión es el único dato que se almacena como un entero largo y no como una cadena de bytes
    - Milisegundos desde 1970
  - Almacenamiento en orden descendiente
    - La más reciente primero
  - Se puede escribir cualquier versión, no solo la más reciente
  - ¿Número de versiones a mantener?
    - Se especifica a nivel de familia de columnas: Máximo y mínimo
  - Operaciones
    - Get/Scan
      - Se obtiene la más reciente a no ser que se especifique lo contrario
        - Get.readVersions() especifica el número de versiones a devolver. Las x últimas.
      - Se puede especificar el intervalo de versiones a devolver
        - Get.setTimeRange() especifica el tiempo inicial y final
    - Put: Crea una nueva versión, especificando el timestamp o usando el actual
    - Delete: Se pueden borrar todas las versiones o una concreta

Modelo de datos

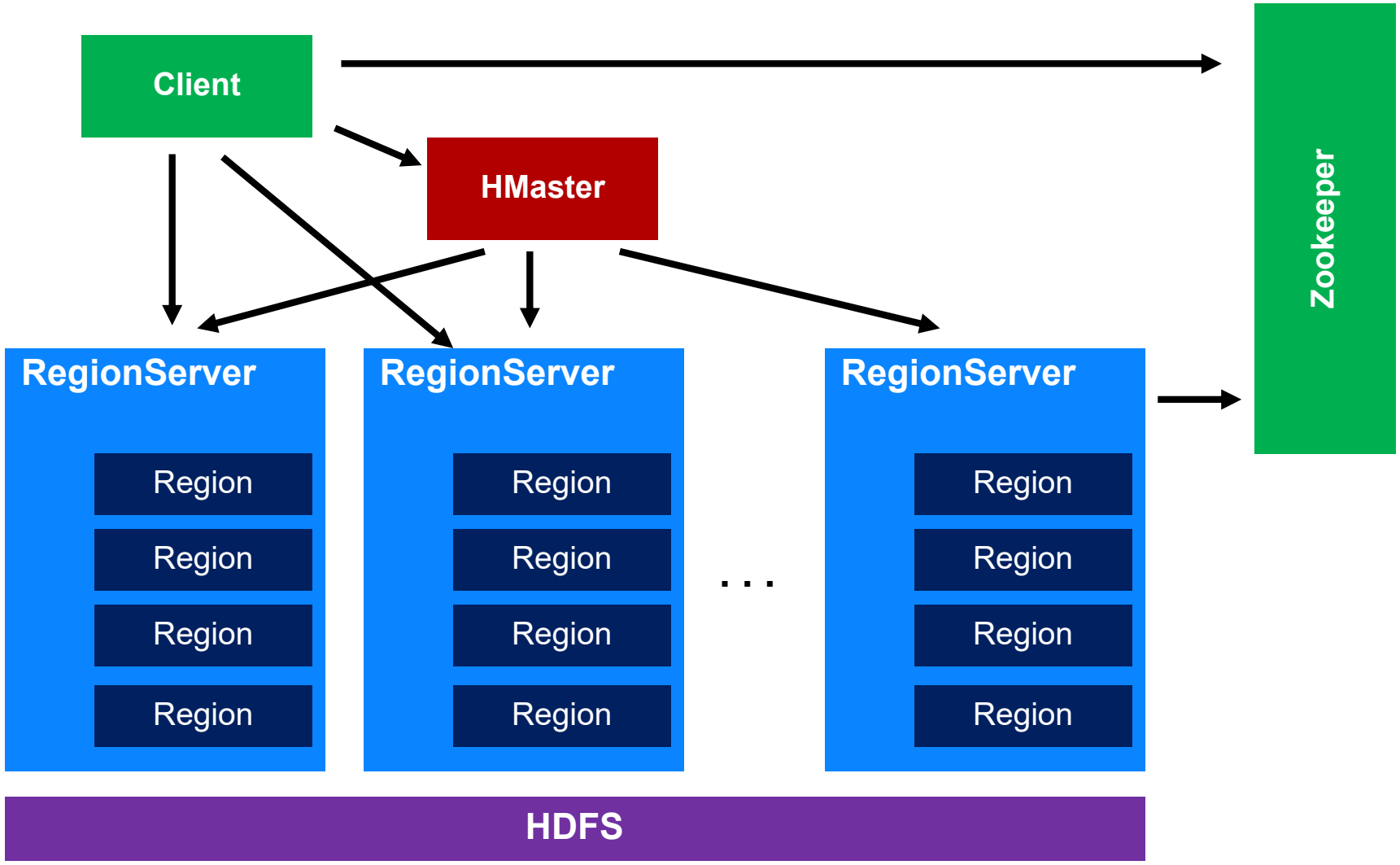
- Ordenación
  - Los datos se devuelven ordenados por fila, CF, columna y versión (descendente)
- Metadatos de columna
  - Los nombres de las columnas de un CF solo se pueden conocer procesando las filas (no hay esquema)
- JOINS
  - No se pueden hacer Joins en el servidor
- ACID
  - Atomicidad: A nivel de fila.
    - CheckAndPut: se ejecuta de forma atómica
  - Consistencia y aislamiento: Filas devueltas son consistentes
  - Consistencia en los Scan
    - No proporciona visión consistente de una tabla (sin aislamiento de instantáneas)
    - Read Committed: Lee datos comprometidos pero no garantiza lectura repetible
  - Visibilidad y durabilidad: Cuando se confirma un cambio es visible para todos.
    - Todos los datos visibles son durables
  - Configurable: Se pueden relajar las condiciones de arriba

Modelo de datos

- Diseño del esquema de las tablas
  - El esquema se crea y modifica con la Shell o con el objeto Admin del API de Java
    - Necesario deshabilitar la tabla antes para modificar el esquema
  - Recomendaciones
    - Regiones (particiones) de entre 10 y 50 GB
    - Celdas no mayores de 10 MB
      - En caso contrario almacenar en un archivo HDFS
    - Entre 1 y 3 CF por tabla
    - Entre 50 y 100 regiones (particiones) por tabla
    - Los nombres de CF y los nombre de columna deben de ser lo más cortos posible
      - Se almacenan con cada valor
    - Cuando tenemos datos históricos no modificables, se pueden tener más regiones que las 100 indicadas arriba

Arquitectura de componentes

- Componentes



Arquitectura de componentes

- Catalogo: Tabla hbase:meta
  - Almacena metadatos de las regiones (particiones) de cada tabla
  - clave: identifica a la región (tabla, clave inicio región, id de región)
  - valores
    - Información de la región
    - RegionServer en el que se almacena la región
    - Instante de inicio del proceso RegionServer que da acceso a la región
  - Al dividir una región padre en dos hijas se crean dos nuevas columnas en esta tabla
    - splitA y splitB, que apuntan a las dos regiones hijas generadas en la división
  - Similar a un árbol binario de búsqueda por la clave de fila.

Arquitectura de componentes

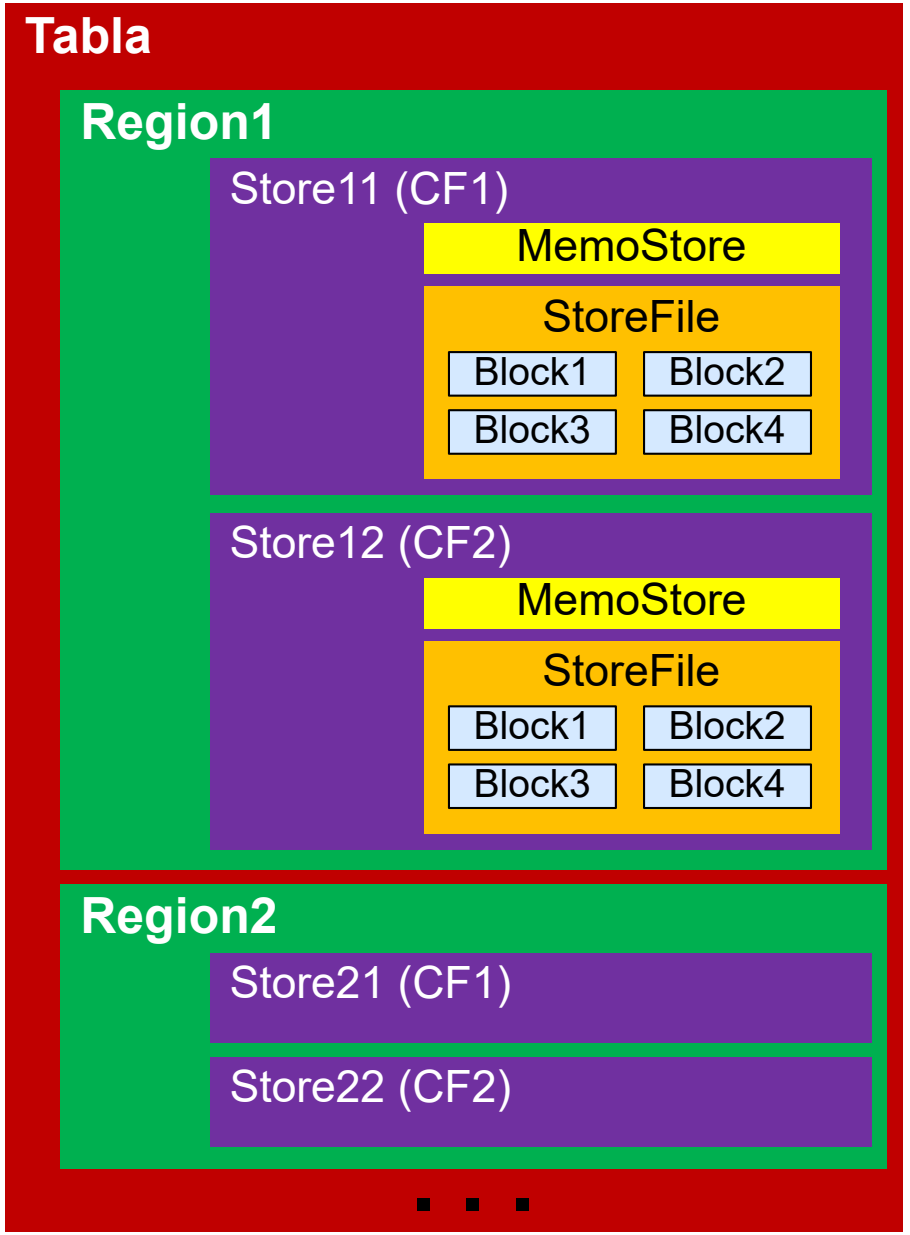
- Client
  - Encuentra los RegionServer que dan acceso al rango de filas que se buscan
  - Debe consultar la tabla hbase:meta
  - Utiliza una caché para no tener que repetir varias veces la misma consulta de metadatos
  - Las operaciones Get y Scan pueden tener filtros
    - Los filtros se aplican en los RegionServer
  - Los filtros incluyen filtros sobre los valores de las columnas y filtros sobre las claves (fila, CF y calificador de columna)
    - Filtros de columna
      - RegexStringComparator, SubstringComparator, BinaryPrefixComparator, BinaryComparator, BinaryComponentComparator
    - Filtros de clave
      - FamilyFilter, QualifierFilter, ColumnPrefixFilter, MultipleColumnPrefixFilter, ColumnRangeFilter
    - Filtros de fila
      - RowFilter
        - También se puede especificar clave inicial y final en el Scan.

Arquitectura de componentes

- HMaster
  - Monitoriza a todos los RegionServer
  - Interfaz para todos los cambios en los metadatos
  - Procesos principales
    - Balanceador de carga: mueve las regiones entre RegionServers
    - CatalogJanitor: gestiona los metadatos
  - En un cluster HDFS se ejecuta en un NameNode
- RegionServer
  - Da acceso a los datos de las regiones
  - En un cluster HDFS se ejecuta en un DataNode
  - Su interfaz proporciona métodos relacionados con el acceso a los datos y con la gestión de las regiones
    - get, put, delete, next, etc. sobre los datos
    - división y compactación de regiones
  - Tiene una implementación de una caché de bloques en memoria
  - Gestiona un Write Ahead Log (WAL)
    - Registro histórico.

Arquitectura de componentes

- Regiones
  - Elemento básico de disponibilidad y distribución de las tablas
  - Se forman para cada CF
  - Jerarquía de almacenamiento
  - Número de regiones (recomendación)
    - Entre 20 y 200 regiones de tamaño grande (5-20 GB) por servidor
    - ¿Pocas regiones?
      - 100 es un buen número
      - Consumimos memoria RAM por cada CF de cada región
  - Balanceador de carga asigna regiones a los RegionServer
  - Localidad
    - En HDFS la primera copia de cada bloque siempre en local



Arquitectura de componentes

- HDFS
  - Cada StoreFile se guarda en un archivo HDFS
  - Importante conocer como funciona HDFS
    - NameNode (Metadatos)
    - DataNode (Bloques de los archivos)
  - A parte de la replicación HDFS, se pueden replicar las regiones para tener alta disponibilidad
    - Datos de la región no accesibles si falla el RegionServer
    - Solución (Se indica el número de replicas en la creación de la tabla)
      - Replicar la misma región en varios RegionServer
      - Esquema de replica primaria y replicas secundarias

HBase

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

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022