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

Programa experto en Big Data

Profesor: Ignacio Marrero Hervás

Sesión API, 1º Trimestre
Curso 2014-2015

ÍNDICE

1. API de HBase. Estructura
2. Filtros
3. Paginado
4. Catching vs Batching

API de HBase. Estructura

La raíz del código de HBase es **org.apache.hadoop.hbase**

master

clases relacionadas con la gestión del Master de HBase

regionServer

clases relacionadas con la gestión del RegionServer de HBase

metrics

clases relacionadas con las métricas de auditoria de la plataforma

monitoring

clases relacionadas con las monitorización en tiempo real

filter

clases que implementan los filtros

client

clases que implementan la base para los clientes de HBase

coprocessor

clases para coprocesadores

rest

clases que implementan el cliente rest

API de HBase. Estructura

thrift

clases que implementan el cliente thrift

security

clases que implementan la seguridad en HBase

snapshot

clases que implementan los snapshots

replication

clases que implementan la replicación

util

clases que implementan algunas utilidades ofrecidas en hbase shell y en línea de comandos

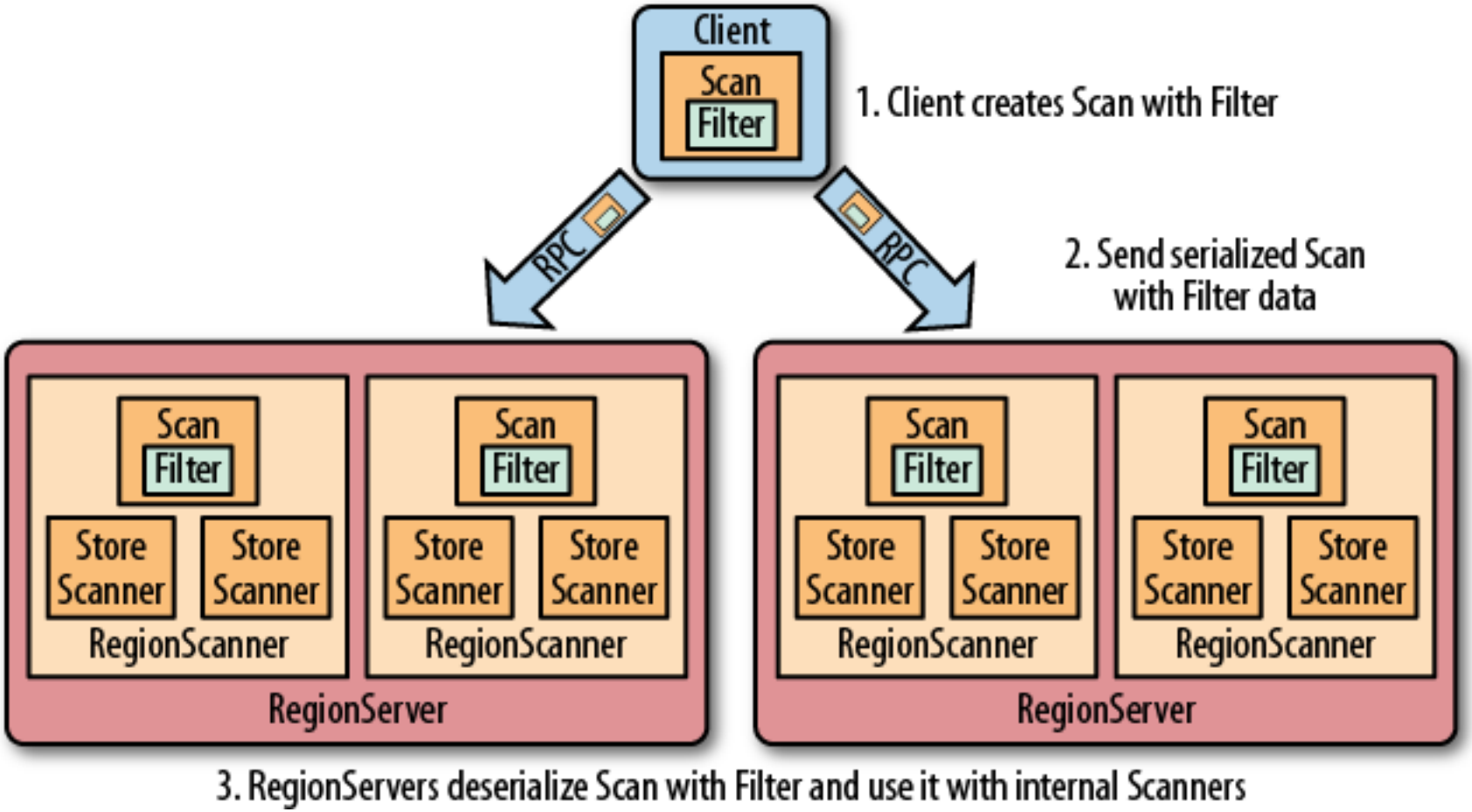
zookeeper

clases para el control de zookeeper

mapreduce

clases que implementan HBase como source y sink

Filtros. Funcionamiento



Filtros. Tipos

Los filtros se dividen en 3 tipos

Filtros de comparación

Permiten comparar las partes del objeto KeyValue con unas proporcionadas a la instancia del filtro. Son: RowFilter, FamilyFilter, QualifierFilter, ValueFilter

```
CompareFilter(CompareOp valueCompareOp,
        WritableByteArrayComparable valueComparator)
```

Filtros dedicados

Implementan determinada funcionalidad: paginado, selección por timestamp, selección por prefijos

Filtros decoradores

Permiten recubrir otros filtros y tomar decisiones a nivel de fila sobre los resultados proporcionados. Son: SkipFilter, WhileMatchFilter

Filtros. Tipos

Un ejemplo

```
FamilyFilter(CompareFilter.CompareOp.EQUAL,
     new BinaryComparator(Bytes.toBytes(columnFamilty1)))
```

Filtros compuestos por una lista de otros filtros

Pueden contener cadenas de filtros enlazados por AND (MUST_PASS_ALL) y OR (MUST_PASS_ONE)

```
new FilterList(FilterList.Operator, List<Filter> filters)
```

Filtros. Operadores

Operador	Descripción
LESS	Devuelve true cuando los valores consultados son menores que el proporcionado
LESS_OR_EQUAL	Devuelve true cuando los valores consultados son menores o iguales que el proporcionado
EQUAL	Devuelve true cuando los valores consultados son iguales al proporcionado
NOT_EQUAL	Devuelve true cuando los valores consultados son distintos al proporcionado
GREATER_OR_EQUAL	Devuelve true cuando los valores consultados son mayores o iguales que el proporcionado
GREATER	Devuelve true cuando los valores consultados son mayores que el proporcionado
NO_OP	Excluye todo

Filtros. Comparadores

Operador	Descripción
BinaryComparator	Uses Bytes.compareTo() to compare the current with the provided value.
BinaryPrefixComparator	Similar to the above, but does a lefthand, prefix-based match using Bytes.compareTo().
NullComparator	Does not compare against an actual value but whether a given one is null, or not null.
BitComparator	Performs a bitwise comparison, providing a BitwiseOp class with AND, OR, and XOR operators.
RegexStringComparator	Given a regular expression at instantiation this comparator does a pattern match on the table data.
SubstringComparator	Treats the value and table data as String instances and performs a contains() check.

Paginado

El paginado es importante dado el volumen de datos. Se pueden usar distintas técnicas

Una posibilidad es usar conjuntamente

1. Filtros por fila: PageFilter
2. Filtros por columna: ColumnPaginationFilter
3. Usando la clave parcial con STARTROW, STOPROW

Paginado

Caso práctico con clave parcial

Tenemos una Rowkey compuesta por “empresa/timestamp”

```
DELL/2001-01-01
DELL/2001-01-02
DELL/2001-02-01
ZED /2001-01-02
ZED /2001-02-03
```

Queremos paginar todos los datos de una determinada empresa

```
STARTROW => ‘DELL’, STOPROW => ‘DELM’, PageFilter(2)
```

Queremos paginar para fecha > 2001-01-02

```
STARTROW => ‘DELL/2001-01-02\x00’, STOPROW => ‘DELM’, PageFilter(2)
```

Caching vs Batching

El cliente de HBase permite establecer el número de peticiones RPC que serán necesarias para recuperar los resultados.

```
void setBatch(int batch)
void setCaching(int caching)
```

caching es el número de Result por RPC
batch es el número de celdas en cada Result

Caching vs Batching

Dada una tabla con 10 filas y 20 columnas cada una

Caching	Batch	Resultados	RPCs	Notas
1	1	200	201	Cada columna es un Result y cada Result se transfiere en un RPC
200	1	200	2	Cada columna es un Result pero todos se transfieren en un sólo RPC
2	10	20	11	Cada fila son 2 Result que se transfieren en un solo RPC
5	100	10	3	Cada fila es un sólo Result y cada RPC tendrá 5 filas

caching es el número de Result por RPC

batch es el número de celdas en cada Result sin transpasar los límites de la row