Tecnologías "on-line", "real-time" y sistemas híbridos: **Cassandra**

Programa Experto en Big Data

Alejandro Cuevas

1º Sesión
Curso 2014-2015

Objetivos

- Aprender a realizar operaciones sobre cassandra
- Entender la estructura interna de datos
- Diseñar soluciones software para la capa de acceso a datos
- Modelado de datos en Cassandra

El porqué de las cosas

- Experimento: 10 monos en una jaula de dos pisos electrificada
- En el piso de arriba hay plátanos
- Cuando algún mono intenta subir, todos reciben una descarga eléctrica
- Los monos aprenden a no subir al piso de arriba

El porqué de las cosas

- Reemplazan uno de los monos por uno nuevo
- El nuevo intenta subir
- El resto de monos no le dejan, y le dan una paliza para que aprenda la lección
- Reemplazan un segundo mono (de los originales) y se repite el comportamiento

El porqué de las cosas

- Continúan reemplazando monos hasta que ninguno de los monos de la jaula ha sufrido nunca una descarga.
- Cuando entra un nuevo mono, se le sigue “aleccionando” a no subir a por los plátanos

Antes de empezar

- Requisitos:
  - Java
  - Cassandra
  - Eclipse
  - Dev Center (DataStax)
- Entorno
  - 1 nodo local
  - Ejecución de pruebas desde eclipse, CLI y Dev Center

Instalación

- Copiar la carpeta cassandra
- Editar el fichero cassandra/1.2.5/bin/cassandra.in.sh
  - CASSANDRA_HOME=/usr/local/Cellar/cassandra/1.2.5
- Abrir el proyecto clase-cassandra en eclipse
- Ejecutar com.utad.cassandra.basic.FirstContact

Astyanax

- Creado, usado y mantenido por Netflix
- Sobre thrift, evolución de Hector
- Se utiliza el modelo de datos interno de Cassandra
- CQL3 en beta
- Pool de conexiones

First Contact

- Conexión a la base de datos
- Crear un keyspace
- Crear un column family
- Escritura (clave – valor)
- Lectura (valor)
- ¿Por qué se crean el keyspace y el column family dentro de un try-catch?

First Contact 2

- Implementar el método getKeyspace en la clase Utils. Debe conectar con cassandra y recuperar un keyspace existente.
- Qué pasa si se intenta recuperar un keyspace no existente?

Escrituras y lecturas por bloques

- Ejecutar WriteWithMutationBatch1
- Comprobar tiempos
- Adaptar el código en WriteWithMutationBatch2 para escribir y leer uno a uno
- Comparar resultados

Tipos de Datos

- Uso de Identificadores numéricos como clave. Ejecutar UseDataTypes
  - Validación de claves
  - Ordenación!
  - Ejecutar UseDataTypes
  - ¿Cómo se han ordenado los resultados?

Tipos de Datos 2

- Adaptar el ejemplo anterior para crear el column family users3 usando
  - String como row key
  - Integer como column key
  - IntegerType como key_validation_class
  - IntegerType como comparator_type
  - Usar ids de usuario del 80 al 99

Tipos de Datos 3

- Leer del column family users3 (ejercicio anterior) usando
  - String como row key
  - String como column key

Lecturas

- Leer del column family users
- Uso de la abstracción de Rangos de Astyanax
  - .setLimit(int n)
  - .setStart(String|long|int… start)
  - .setEnd(String|long|int… n)
- Uso de slices
  - .withColumnSlice(Collection c)

```
…getKey(key).withColumnRange(new RangeBuilder()[*aquí van los
modificadores*].build())
…getKey(key).withColumnSlice([*objeto que extienda Collection*])
```

Lecturas

- Reading1 : leer los 20 primeros usuarios
- Reading2 : leer todos los usuarios a partir del usuario con id 50
- Reading3 : leer todos los usuarios desde el principio hasta el 50
- Reading4 : leer los usuarios entre el 50 y el 60
- Reading5 : leer los usuarios entre el 50 y el 60, pero más de 8
- Reading6 : leer los usuarios 1, 3, 5, 7 y 11
- Reading7: leer los usuarios 1, 3, 5, 7 y 11, pero sólo a partir del usuario 5

Paginación

```
for (int i = 1; i <= 100000; i++) {
```

- En la clase Pagination, aumentar el valor para que provoque una excepción por falta de memoria
- Copiar en la clase Pagination1 y modificar para que escriban páginas de tamaño pageSize

Paginación II

- En la clase Pagination2, leer los datos del ejercicio anterior usando lecturas paginadas:
  - Usamos un rango con limit = pagesize
  - Marcamos .autopaginate(true)
  - Hacemos tantas lecturas como sea necesario (mientras haya datos en el resultado)

```
.getKey("usersById").autoPaginate(true)
.withColumnRange(new RangeBuilder().setLimit(pageSize).build());

while (!(columns = query.execute().getResult()).isEmpty()) {
          for (Column<Integer> c : columns) {
          }
}
```

Paginación 3

```
        .getKey("usersById").autoPaginate(true)
        .withColumnRange(new RangeBuilder().
setLimit(pageSize).build());

      while (!(columns =
query.execute().getResult()).isEmpty()) {
            for (Column<Integer> c : columns) {
            }
      }
```

Contadores

- El tipo de datos para los valores es long
- Es un contador distribuído!
- Usaremos rowKey = id del usuario
- Column Key = id del producto
- Valor = número de veces que se ha visitado
- Para incrementar el contador (en 1):

```
clm.incrementCounterColumn(key, 1);
```

Contadores 2

- Para crear el column family:

```
.createColumnFamily(
    cfUsers,
    ImmutableMap
            .<String, Object> builder()
            .put("default_validation_class",
                    "CounterColumnType")
            .put("replicate_on_write", true).build());
```

- Para incrementar el contador (en 1):

```
clm.incrementCounterColumn(key, 1);
```

Claves Compuestas

- Usaremos un separador
  - Ninguna de las partes puede contener el separador
  - clave1:clave2
- Combinar las claves del ejemplo anterior usando un único rowKey, en un nuevo column family user_visists_product2

Claves Compuestas 2

- Usar rangos para leer los productos visitados por el usuario 3

Claves Compuestas 3

- Diseñar e implementar un column family para buscar usuarios a partir de su código postal y nombre de la calle de residencia.
- Los datos a obtener son, para el CP 28001 y la calle Gran Via:
  - Id del usuario
  - email
  - Nombre

CLI

- Command Line Inteface
- Está siendo reemplazado por cqlsh
- Nos permitirá ver los datos de los ejercicios usando Astyanax (cqlsh no)

CLI 2

- cassandra-cli -host localhost -port 9160
- use keyspace;
- list column_family;

- ASSUME column_family KEYS AS utf8;
- ASSUME column_family COMPARATOR AS utf8;
- ASSUME column_family VALIDATOR AS utf8;

Java Driver

- Creado y mantenido por Datastax
- Permite ejecutar consultas CQL
- Usar el ejemplo de la clase UsingJDriver para recrear la solución de las ciudades y códigos postales