BigData - Proyecto parte 1

Parte 1 Proyecto asignatura BigData.
Máster Universitario en Ingeniería Informática - Universidad Pablo de Olavide

Ejecución del laboratorio de la parte 1

Escribimos las siguientes líneas en un fichero `compose.yaml`, cambiando el mapeo del volumen `data` por el de su equipo:

```
services:
hbase-pseudo:
    image: jsgifbec/custom-hbase-pseudo:latest
    container_name: hbase-pseudo
    hostname: hbase-pseudo
    volumes:
    - /media/SHARED/repositories/BigDataProject/docker/services/hba
    ports:
    - 2181:2181
    - 8080:8080
    - 8085:8085
    - 9090:9090
    - 9095:9095
    - 16000:16000 # master
    - 16010:16010 # master-ui
    - 16201-16210:16201-16210 # region servers
    - 16301-16310:16301-16310 # region servers - ui
    restart: unless-stopped
```

y, ejecutamos en el mismo directorio del fichero:

```
docker compose up -d
```

Abrimos una consola bash en el contenedor mediante:

```
docker exec -it hbase-pseudo bash
```

Los comandos son explicados en el punto 3 de este documento.

Enlace del vídeo

Vídeo BigData parte 1

Requisitos del trabajo final - parte1

Requisitos obligatorios

1. Justificación de la estructura de datos

1.1 RowId

La rowId es formada de la siguiente manera:

```
private static String GetRowKey(SynteticData mr) {
    int bucket = computeBucket(mr.getSensorAsString() + mr.getDayAs
    String rowKey = bucket + "#" + mr.getSensorAsString() + "#" + m
    return rowKey;
}

// Es decir, hash + "#" + sensorId + "#" + day
```

De esta forma la rowId será igual para todos las lecturas de un sensor en un mismo día.

El hash, se calcula de la siguiente manera:

```
private static int computeBucket(String key, int buckets) {
    int rawHash = key.hashCode();
    int positiveHash = rawHash & Integer.MAX_VALUE;
    return positiveHash % (buckets);
}
```

Siendo `buckets` el número de servidores de región, 3. Ese hash devolverá valores [0, N_LOCAL_REGION_SERVERS - 1]. Asegurando de esta manera, una distribución equitativa entre las regiones porque el número de divisiones (splits), en el `createTable` también está condicionado por el número de regiones:

```
byte[][] splits = new byte[Commons.N_LOCAL_REGION_SERVERS - 1][];
for (int i = 1; i < Commons.N_LOCAL_REGION_SERVERS; i++) {
    splits[i - 1] = Bytes.toBytes(Integer.toString(i));
}
```

Para 3 servidores de región:

- Región 1: desde el inicio hasta 1
- Región 2: 1 → 2
- Región 3: 2 → desde 2 en adelante

1.2 Familia de columnas y columnas

La estructura de las familias de columna escogida es la siguiente:

general	measure1	measure2		measureC
sensorId = 1DGXXXX1, day = 2013-12-01	00:00 => valor_s1m1_00:00	00:00 => valor_s1m2_00:00	...	00:00 => valor_s1mC_00:00
	00:10 => valor_s1m1_00:10	00:10 => valor_s1m2_00:10	...	00:10 => valor_s1mC_00:10
	00:20 => valor_s1m1_00:20	00:20 => valor_s1m2_00:00	...	00:20 => valor_s1mC_00:20
	...	...	...	...
sensorId = 1DGXXXX2, day = 2013-12-01	00:00 => valor_s2m1_00:00	00:00 => valor_s2m2_00:00	...	00:00 => valor_s2mC_00:00
	00:10 => valor_s2m1_00:10	00:10 => valor_s2m2_00:10	...	00:10 => valor_s2mC_00:10
	00:20 => valor_s2m1_00:20	00:20 => valor_s2m2_00:00	...	00:20 => valor_s2mC_00:20

De esta manera, todas las medidas tomadas por el lector1 del sensor 1DGXXXXX, el día 2013-12-01, es almacenada en una sola familia de columna para cada rowId. Siendo necesario leer tan solo dos familias de columnas para extraer toda la información que necesitamos, la general (contiene el sensorId y el día) y la familia de columna correspondiente al C introducido.

1.3 Problemas encontrados.

Esta estrucuta y, generación de rowId, hace necesario ordenar las filas extraidas. Primero por la columna general:sensorId y luego por general:day. Pero, no hace lecturas innecesarias.

2. Detalle de las herramientas desarrolladas

Las herramientas están definidas en dos subcomandos de la aplicación:

1. load-table

@Command(name = "load-table", description = "Herramienta de carga") class LoadTable implements Runnable { @Option(names = {"-f", "--file"}, required = true, description = "Archivo CSV a cargar") String file;

```
    @Option(names = {"--factor-c"}, required = true, description =
    int factorC;

    @Option(names = {"--factor-f"}, required = true, description =
    int factorF;

    public void run() {

        try {
            // Borramos todas las tablas
            dropTables();

            // Creamos la estructura de la tabla
            HTableDescriptor tableDescriptor = defineTable(factor(
            createTable(tableDescriptor);

            // Leemos el fichero y aplicamos el bootstrapping
            List<SynteticData> synteticData = generateSyntheticRe

            // Insertamos en Hbase
            insertDataIntoHbase(tableDescriptor, synteticData);

        } catch (Exception e) {
            System.out.println(e.toString());
        }

    }
    ...


}
```

2. retrieve-table

@Command(name = "retrieve-data", description = "Herramienta de extracción") class RetrieveData implements Runnable { @Option(names = {"-c", "--column"}, required = true, description = "Columna de extración") int retriveColumn;

```
        @Option(names = {"-r", "--row"}, required = true, description
        int retriveRow;

        @Option(names = {"-o", "--output"}, required = true, descript
        String output;

        public void run() {
            try {
                String cF = String.format("%s%d", Commons.CF_MEASUREX
                String fId = String.format("%dDG", retriveRow);
                writeCsv(output, generateHeader(), getRowsBySensorPre

            } catch (Exception e) {
                System.out.println(e.toString());
            }

        }
        ...


    }
```

3. Invocación de las herramientas de carga y extracción

3.1 Invocación de la herramienta de carga

El uso es el siguiente:

```
Usage: BigData load-table -f=<file> --factor-c=<factorC> --factor-f
Herramienta de carga
    -f, --file=<file>           Archivo CSV a cargar
    --factor-c=<factorC>        Factor de multiplicación de columna
    --factor-f=<factorF>        Factor de multiplicación de fila
```

Ejemplo:

```
bigdata load-table --file SET-dec-2013.csv --factor-f 5 --factor-c
```

3.2 Invocación de la herramienta de extracción

El uso es el siguiente:

```
Usage: BigData retrieve-data -c=<retriveColumn> -o=<output> -r=<ret
Herramienta de extracción
-c, --column=<retriveColumn>    Columna de extración
-o, --output=<output>           Archivo de salida
-r, --row=<retriveRow>          Fila de extración
```

Ejemplo:

```
bigdata retrieve-data -o salida.csv -r 3 -c 3
```

4. Scripts de creación de tablas y limpieza

Los scripts de creación y eliminación de la tabla están integrados en la herramienta de carga. El orden de ejecución dentro de la misma es el siguiente:

```
public void run() {

        // Borramos todas las tablas
        dropTables();

        // Creamos la estructura de la tabla
        HTableDescriptor tableDescriptor = defineTable(factorC);
        createTable(tableDescriptor);

        ...
}
```

4.1 Script de creación de la tabla

Para crear la tabla, primero hay que definirla. Esto se realiza mediante el método `defineTable` que recibe cómo parámetro el número de medidas de los sensores (ligado al factor de multiplicación de columnas):

```
private static HTableDescriptor defineTable(int nMeasuresBySensor)
    HTableDescriptor tableDescriptor = new HTableDescriptor(TableNa

    tableDescriptor.addFamily(new HColumnDescriptor(Commons.CF_GENE

    for (int m = 1; m <= nMeasuresBySensor; m++) {
        tableDescriptor.addFamily(new HColumnDescriptor(String.form
    }
    System.out.println("Definida la nueva estructura de la tabla");
    return tableDescriptor;
}
```

Una vez creado el objeto que define la tabla, se lo pasamos al método `createTable` quien define las regiones y crea la tabla:

```
private static void createTable(HTableDescriptor tableDescriptor) t
    try (Connection connection = HBaseConnector.getConnection()) {
        Admin admin = connection.getAdmin();

        byte[][] splits = new byte[Commons.N_LOCAL_REGION_SERVERS -
        for (int i = 1; i < Commons.N_LOCAL_REGION_SERVERS; i++) {
            splits[i - 1] = Bytes.toBytes(Integer.toString(i));
        }

        admin.createTable(tableDescriptor, splits);
    }
    System.out.println("Creada la nueva tabla");
}
```

4.2 Script de limpieza

El script de limpieza está integrado en la herramienta de carga. Su funcionamiento es el siguiente:

1. Deshabilita la tabla.
2. Elimina la tabla.

El código es el siguiente:

```
private static void dropTables() throws IOException {
    try (Connection connection = HBaseConnector.getConnection()) {
        Admin admin = connection.getAdmin();

        for (TableName table : admin.listTableNames()) {
            if(!admin.isTableDisabled(table))
                admin.disableTable(table);
            admin.deleteTable(table);
        }
    }
    System.out.println("Eliminadas las tablas existentes");
}
```

5. Código fuente de las herramientas desarrolladas

El código fuente se encuentra adjunto a este documento y publicado en github:

Repositorio en Github - tag:parte1