UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Big Data Management and Analytics
Session: Distributed Storage: Key-Value (HBase) III

Lecturers: Petar Jovanovic and Josep Berbegal

1 Tasks To Do Before The Session

It is important that you: (1) carefully read the **instruction sheet** for this lab session, (2) introduce yourself to the **lab's main objectives**, (3) understand the **theoretical background**, and (4) get familiar with the **tools being used**.

2 Part A: Examples & Questions (15min)

In the first 15 minutes, we will first clarify the main objectives of this lab. We will them learn some basics on using HBase Java library and how it can be used to programmatically customize the way we write and read data in HBase.

3 Part B: In-class Practice (2h 45min)

Before going ahead in this practice, you should first fix how much data you are going to insert in HBase. You should chose a considerable amount of data to demonstrate your work, but also remember that you also have a limited space available (check previous sessions for a reference).

Amount of data used: 150 MB

3.1 Exercise 1: Implement vertical partitioning

Importantly, before starting with this part of the assignment, **rename** the existing `HBase-0.1.jar` JAR file from the previous sessions in your master node to `HBase-0.1-base.jar`.

In HBase shell, create a table with four families for this exercise:

```
create 'wines_c_1', 'col_1', 'col_2', 'col_3', 'col_4'
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

1. **Inside your fork of the HBase Java project**, implement the function **toFamily()** in *MyHBaseWriter_C_1* so that different attributes go into different families as shown in Table 1. You will also need to update the *Main* class to load *MyHBaseWriter_C_1* as writer (uncomment the right one). Then compile your code and deploy a new `HBase-0.1.jar` JAR in your master node and insert as much data as you have decided:

```
hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar write -hbase -size <SIZE> wines_c_1
```

Attribute	**Family**
m_acid	*col_1*
ash	*col_2*
alc	*col_1*
alc_ash	*col_1*
mgn	*col_1*
t_phenols	*col_2*
flav	*col_1*
nonflav_phenols	*col_1*
proant	*col_2*
col	*col_1*
hue	*col_1*
od280/od315	*col_2*
proline	*col_3*
type	*col_4*
region	*col_4*

Table 1: List of attributes and families

2. Use HDFS to check how much disk space is consumed by each family. How much is it? Does it make sense with respect to the number of attributes we inserted in each family?

```
hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines_c_1/*/col_1
hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines_c_1/*/col_2
hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines_c_1/*/col_3
hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines_c_1/*/col_4
```

Answer: Per a col_1, 506,7 MB; per a col_2, 261 MB; per a col_3, 69,6 MB; per a col_4, 115,1 MB. Té sentit, ja que cada registre per a la column family col_1 té 8

3. Implement the function **scanFamilies()** in *MyHBaseReader_C_1* so that HBase scan is configured to only retrieve families *col_3* and *col_4*. You will also need to update the *Main* class to load *MyHBaseReader_C_1* as reader (uncomment the right one). Then recompile and redeploy

the `HBase-0.1.jar` JAR to your master node. You can check the output by reading the table:

```
hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar read -hbase wines_c_1
```

4. Compare the total time needed to scan the whole table by using the old *MyHBaseReader* and your new *MyHBaseReader_C_1*. Discuss the impact of this vertical partitioning on queries that only need *proline* and *region* attributes.

```
time hadoop-2.7.4/bin/hadoop jar HBase-0.1-base.jar read -hbase wines_c_1 > /dev/null
time hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar read -hbase wines_c_1 > /dev/null
```

Answer: Per al mètode antic, 2m 55.336s; per al mètode nou, 1m 15.236s. El particionat vertical redueix el temps de consulta, ja que com les dades es guarden per column

3.2 Exercise 2: Implementing the key design

Recreate the wines table we have been using for this exercise:

```
create 'wines_c_2', 'all'
```

Recall the key design discussion in the previous lab session (Exercise 3) and implement it. Imagine queries that retrieve only data for wines of a specific type and region.

1. Implement the function **nextKey()** in *MyHBaseWriter_C_2* to generate row keys based on the key design you have found useful to reduce the amount of data retrieved for this case. You will also need to update the *Main* class to load *MyHBaseWriter_C_2* as writer (uncomment the right one). Then compile your code and deploy the `HBase-0.1.jar` JAR in your master node. Then, insert as much data as you have decided:

```
hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar write -hbase -size <SIZE> wines_c_2
```

2. Implement the functions **scanStart()** and **stopScan()** in *MyHBaseReader_C_2* to query for wines of type *type_3* and region *0* without scanning all the table. You will also need to update the *Main* class to load *MyHBaseReader_C_2* as reader (uncomment the right one). Then compile your code and deploy the `HBase-0.1.jar` JAR in your master node. You can check the output by reading the table:

```
hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar read -hbase wines_c_2
```

3. Compare the total time needed to scan the table by using the old *MyHBaseReader* and your new *MyHBaseReader_C_2*.

```
time hadoop-2.7.4/bin/hadoop jar HBase-0.1-base.jar read -hbase wines_c_2 > /dev/null
time hadoop-2.7.4/bin/hadoop jar HBase-0.1.jar read -hbase wines_c_2 > /dev/null
```

Answer: Per al mètode antic, 2m 41.722s; per al mètode nou, 0m 19.456s. Podem veure que el nou mètode, que només accedeix als registres amb rowKey

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

You might need to check the Scan API, which is available at:

`https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html`

Additional comments: