UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

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

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 then see how HBase tables are physically stored. Next, we will see how data are distributed in HBase and what affects the load balancing. We will then discuss the ways to improve the load balancing and good practices in key design.

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

3.1 Exercise 1 (1h): On the HBase balancing

In the following exercises, we are going to populate the HBase with little more data and then we will experiment on more advanced features. To do so, we will need the data generator you should have compiled at the beginning of the session. Then, create a new table called *wines* with only one family called *all*. Thus, in an HBase shell, run the following:

```
create 'wines', 'all'
```

The next step is to actually populate the wines table. Bulk 150 MB into it (it takes around 4 min). You can do that by running (now in the Linux shell, not in the HBase one) the next command.

```
hadoop-2.7.4/bin/hadoop jar labo2.jar write -hbase -size 0.15 wines
```

Answer the following questions (*labo2.jar* is the JAR built for this session):

1. Explore a bit what is inside the */hbase/data/default/wines* folder in HDFS and its subfolders. Forget about metadata folders in there (e.g., *.tabledesc*, *.tmp*, *.regioninfo*, etc.). Can you relate each subfolder with one of the components from the HBase logical structure? List these relationships.

   ```
   hadoop-2.7.4/bin/hdfs dfs -ls /hbase/data/default/wines
   ```

   Answer: /hbase/data/default/wines/74735a0700e3f530f4bdce2842de1866 -> This is one Region
   /hbase/data/default/wines/dd48fb82c68434194379b642

2. To check the size of a given table in HBase, we need to do that through the HDFS, which is the file system storing all the HBase data. Then, run the following command. What is the size of this table? Does this make sense to you considering that we inserted 150 MB of data? Discuss this result.

   ```
   hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines
   ```

   Answer: The physical table size is 875.5 M. It does make sense since for each key, value HBase generated metadata.

3. On the web UI you might find other information in a user-friendlier manner. For instance, how many regions were created for the table wines? In which RegionServers are they stored?

   Answer: There were created 2 Regions in the Slave1 (RegionServer).

4. Then check the size of each region by means of the next command. How much is it? Do you think they are well balanced? Compare your results with the classmate next to you.

   ```
   hadoop-2.7.4/bin/hdfs dfs -du -s -h /hbase/data/default/wines/*
   ```

   Answer: Region 1: 373.2 M
   Region 2: 502.2 M
   No, we don't think they are well balanced. The default

3.2 Exercise 2 (1h): On HBase balancing improvement

Let us try to make a better balancing. In order to do that, HBase provides DBAs with presplitting, which is a technique to split the table before insertions occur. This way, in the short term, HBase performance should be boosted since all the workload is distributed across all the RegionServers. In the long term, HBase split policy is supposed to uniformly distribute across servers so we shall not worry about it. Yes, HBase needs a lot of data to give its best.

In order to perform presplitting, we are going to take advantage of the fact that we know how many rows are in 150 MB of data we inserted. To figure this out, in a HBase shell, run the following:

```
count 'wines', INTERVAL => 100000, CACHE => 10000
```

They should be around 1300000, which we are going to round up to 1500000. Afterwards, we more or less uniformly create a new *wines.1500000* presplit table by setting a key prefix for each region. This should be run in HBase shell.

```
create 'wines.1500000', 'all', SPLITS => ['12', '14', '2', '4', '6', '8']
```

And finally populate it with 1500000 rows. Next command should be run in Linux console.

```
hadoop-2.7.4/bin/hadoop jar labo2.jar write -hbase -instances 1500000 wines.1500000
```

1. With such results at hand, say whether the balancing now is better or not. What about the regions? In which RegionServers are they stored? Compare this with the previous results.

   Answer: Now the regions are much more balanced:
   131.0
   M /hbase/data/default/wines.1500000/0154029b963be

2. Justify how many rows should be in each region due to the key prefixes we chose *'12', '14', '2', '4', '6'* and *'8'*

   Answer: According to the HBase UI:
   From key 0 to key 12 - 222224 rows
   From key 12 to key 14 - 222222 rows

3.3 Exercise 3 (45h): On the key design

One important thing in HBase is to decide what or how the row keys are going to be defined. Scans, for instance, can benefit from the B+ tree and the clustered index to only read data of interest and to avoid wasting resources on non-relevant data for the current query. As a simple example, imagine a query that uses a specific attribute to filter out the rows quite often and thus we decide row keys have the value of such attribute at the beginning. Queries could then benefit from the lexicographical order from querying through row key prefix.

Now discuss the importance of the key design when writing in HBase. To do so, try to think about using the timestamp at which the insertion happens as row key. Do you think this is a good design?

Answer: For the writing operation, using the this is not a good design solution, because in fact it makes the HBase not balanced. Using keys with very close values, which is

Justify your answer and, if you think it is not, propose a solution.

Answer: One likely solution would be to modify the order of the timestamp parameters, in a way that it wouldn´t represent close values. For instance, instead of HHMMSS, to consider S[2]H[2]M[2]S[1]H[1]M[1]. Additionally, there is another technique we found, which could be to use reverse timestamp, which ensure values that wouldn't be close to each other, and would allow a more balance for the regions.

Additional comments: