UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Concepts for Specialised Databases
The Hadoop 2.x Ecosystem: YARN, HDFS, HBase, and MapReduce

Alberto Abelló, Sergi Nadal and Oscar Romero

Objectives

This is a hands-on exercise with HBase and MapReduce. Obviously, HDFS will be in the middle, but we will focus on the first two. The list of objectives for this session is as follows:

- Manage an HBase store (i.e., create / modify / drop tables, insert / drop data),
- Write simple Map / Reduce tasks on top of files.

Lab organization

On session 1, the lecturer will be present at the lab and will help you out. Thus, it is a session to solve your doubts and help you to prepare the final delivery.

On session 2, you must hand in (upload) the deliverables specified in the corresponding section of this document.

Important: You are highly advised to attend the first session. Furthermore, if you do not work on this lab during the first week expect a heavy workload for the last one (we estimate 6h of work per person per week). Thus, you are the ultimate responsible for a reasonable scheduling of this session.

Required knowledge

This practice requires understanding the "Key-Value stores (I)" and "Key-Value stores (II)" and "MapReduce" lectures. Also, a training is provided. **Your lab mate for this practice will be that of the corresponding team creation event.**

Instructions

This session consists of two independent parts (i.e., HBase and MapReduce). The first part must be performed in the provided virtual machine (see the Virtech manual for further details), while the second can be performed in your local computer. From now on, we will assume you are working in a Linux environment (for Windows, you will find some extra configuration instructions of IntelliJ).
Before going to the next section, first carefully read and understand the following documents (all of them available in the course's website):

1. Intellij setup
2. Virtech manual
3. HBase training
4. MapReduce training

Delivery

This lab will be assessed as follows: HBase (4 pt) + MapReduce (6 pt). Each line (or blocks of code) of your implemented code has to be commented to let the lecturer understand what you are doing. Upload all Java files into the corresponding Moodle activity BEFORE the end of the second lab. Failing to send these files before the second session will invalidate the whole session.

The second lab session, is merely a deadline for the submission of the solutions/deliverables to the first lab session. Thus, there will be no second lab session with the teacher.

Session assignment

Dataset

Both parts of this session consist on processing and analyzing a dataset containing information about wines and their properties (you can check the details at `https://archive.ics.uci.edu/ml/datasets/wine`). Its schema consists of the following 15 attributes:

- `type` (*string*)
- `region` (*int*)
- `alc` (*double*)
- `m_acid` (*double*)
- `ash` (*double*)
- `alc_ash` (*double*)
- `mgn` (*double*)
- `t_phenols` (*double*)
- `flav` (*double*)
- `nonflav_phenols` (*double*)
- `proant` (*double*)
- `col` (*double*)
- `hue` (*double*)
- `od280od315` (*double*)
- `proline` (*double*)

Part 1: HBase

Exercise 1: HBase vertical partitioning

In this exercise, we are going to implement vertical partitioning in HBase. We will programatically define the partitions and access them. To this end, we provide you the classes `MyHBaseWriter` and `MyHBaseReader` that interact with HBase's Java API to read/write data. Note you are not required to interact with HBase's API, you will be provided specific classes that inherit their functionalities. Now, assume a query Q_1 that needs to scan the whole table but only fetching attributes *type*, *region* and *flav*.

- **[1.1] Create a new table with the best layout (i.e., define the appropriate families) to accommodate such query. Write as a comment the `create` statement in `MyHBaseWriter_VerticalPartitioning.java`.**
- **[1.2] Implement the method `toFamily` in class `MyHBaseWriter_VerticalPartitioning` so that it correctly populates the new layout for `wines2` for the attribute passed as parameter.**
- **[1.3] Implement the method `scanFamilies` in class `MyHBaseReader_VerticalPartitioning` so that it correctly fetches attributes *type*, *region* and *flav* from `wines2`.**

Note. You might use `-write -hbase-vertical-partitioning X wines2` as run configuration (where X is the number of instances you want to generate) and `-read -hbase-vertical-partitioning wines2` (see the *IntelliJ setup* manual for more details on run configurations).

Exercise 2: HBase key design

This exercise asks you to design the key of the table to benefit from the distributed B-tree of HBase and avoid table scans when doing specific searches. Thus, assume a query Q_2 that retrieves all attributes where $type = type\_3$ and $region = 0$. First, recreate again a table with only one column family like the training.

```
create 'cursbdXYtable', 'all'
```

- **[2.1] Implement the method `nextKey` in class `MyHBaseWriter_KeyDesign` so that it generates the most appropriate key design to efficiently answer Q_2.**
- **[2.2] Implement the method `scanStart` and `scanStop` in class `MyHBaseReader_KeyDesign` so that it generates the appropriate range of keys to answer Q_2.**

Hint. Remember you have available the following objects from `MyHBaseWriter` (**see more details in HBase's training!**):

- `protected HashMap<String,String> data`: for the instance being currently generated contains its list of attributes as keys and the data for this attributes as values.
- `protected int key`: for the instance being currently generated contains an autoincremental value.

Note. You might use the run configuration `-write -hbase-key-design X cursbdXYtable` (where X is the number of instances you want to generate) and `-read -hbase-key-design cursbdXYtable` (see the *eclipse setup* manual for more details on run configurations).

Part 2: MapReduce

Exercise 3: Selection

Implement the selection (i.e., filtering) of the dataset. Precisely, we want only to output those tuples where the attribute `type` has a value `type_1`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT * FROM wines WHERE type = "type_1"
```

- **[3.1] Write the required classes (i.e., map, combiner and/or reduce) in the file `Selection.java`.**

Use the following run configuration to execute the job.

```
-selection path/to/wines.1000.sf path/to/selection.out
```

Exercise 4: Aggregation average

Implement the aggregation with an average function of the dataset. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT type, AVG(col) FROM wines GROUP BY type
```

- **[4.1] Write the required classes (i.e., map, combiner and/or reduce) in the file `AggregationAvg.java`.**

Use the following run configuration to execute the job.

```
-aggregationAvg path/to/wines.1000.sf path/to/aggregationAvg.out
```

Exercise 5: Join

Implement the join of elements of `type_1` with elements of `type_2` that have the same `region`. If such dataset was stored in a relational table, we would be performing the following query:

```
SELECT external.*, internal.*
FROM wines as internal, wines as external
WHERE external.type="type_1" AND internal.type="type_2" AND external.region=internal.region
```

- **[5.1] Write the required classes (i.e., map, combiner and/or reduce) in the file `Join.java`.**

Use the following run configuration to execute the job.

```
-join path/to/wines.1000.sf path/to/join.out
```