Implementation of a (Big) Data Management Backbone: Part 1

Authors:
Àlex Martorell and Enric Reverter

Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona
March 2022

How to Run the Code

In order to run the pipeline, execute the `p1.py` file. It will access the required functions to automatically load both the temporal and persistent zones using the data collectors. It starts by running the auxiliary functions contained in `populate_temporal.py`, which are a series of methods that populate the *HDFS* backbone with the data from all the sources we have, including the collector for the external source. After that, a server object is started in order to activate the *MongoDB* instance from the virtual machine. Finally, the methods from `populate_persistent.py` are initiated and the data located into *HDFS* is properly transformed to be loaded into the *MongoDB* database.

Contents

1 Landing Zone

1.1 Summary

Three datasets (if we exclude the Lookup table) are treated in the Landing Zone. The first and second datasets are provided in JSON and CSV file format respectively. The choice for the third dataset is a list of all cultural facilities in the city of Barcelona *(Llista d'espais i equipaments de cultura i lleure de la ciutat de Barcelona)* [2]. This is retrieved via an API that collects the file in JSON format.

For the choice of tools in the Landing Zone, we proceed as follows:

- **General.** *Python* is the programming language chosen to build the architecture.
- **Temporal Zone.** *HDFS* is used to store the files given to us as well as the collected one. The `hdfs3` library allows to connect into it using python.
- **Persistent Zone.** *MongoDB* is used to store the data into different collections. The `pymongo` library allows to connect into it using python.

The drawing below (1.1) shows the data flows between both zones and machines. First, the collectors retrieve the data from the sources and load it into *HDFS* in a similar file structure as the one provided. Then, this data is loaded into the respective collections of MongoDB through the main loader.

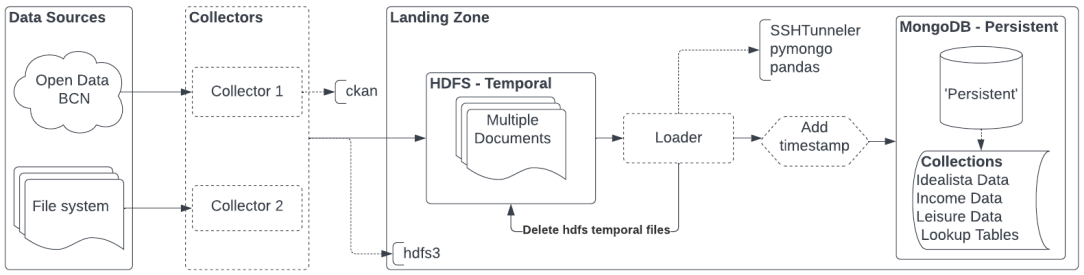

Figure 1.1: Overall process since data is retrieved until it is loaded into the DDBMS of choice, *MongoDB*. Note that elements after brackets indicate the dependencies needed at the given step.

1.2 Assumptions

Below are some of the assumptions we made for this project, regarding tool choice and design of both temporal and persistent zone.

- The size of the data, mainly the retrieved from Idealista, ranges from GB to TB.
- Even though the data provided in a zip file, in a real case scenario (Big Data), we can assume that data would be scrapped through different collectors in real time. (See section 2.1 for the consequences of this).
- In reference to the past assumption, we assume regular (i.e. daily) retrievals for files belonging to idealista dataset but yearly updates regarding the other ones. Recall

that the given Open Data BCN dataset is a list of the income per neighborhood (posted and updated once per year), and the collected one (cultural facilities) is of similar kind. This means that the bulk of the data comes from idealista.

- Data retrieved from idealista might mutate its schema through time.
- Multiple queries over different fields will be needed. For example, some queries might tackle only a range of neighborhoods, etc.

2 Tool Choices

As mentioned above, *HDFS* is used to store data in the temporal zone while in the persistent zone *MongoDB* is preferred.

2.1 Temporal zone

Our first assumption states that we think that in a real case scenario we would have a lot more data. As such, using a distributed file system to store temporal data is better.

Also, *HDFS* is easily scalable, so if more storage is needed, just adding more nodes solves the issue. Then, it also provides high availability of data thanks of its replication. Finally, it should be said that *HDFS* deals well with the variety of files and changes in sources.

2.2 Persistent zone

In this section, we try to detail why we opt for a Document Store architecture.

1. **General concept**: Key-value databases give way to very high simplicity with a slight loss in flexibility. Values cannot be queried, so one must modify the key to contain the necessary information beforehand.

   On the other hand, Document Stores are a Semi-structured Database model. Schema-wise, there exists some structure. However, this amount of metadata creates some redundancy when the data is loaded. For example, in the case of idealista, only the keys of the values account for 35% of the document.

2. **Fragmentation**: Recall that in [1], distinctions are made between when to use either Horizontal or Vertical fragmentation. Horizontal boosts data locality both for querying and inserts / updates. Vertical is better for read-only workloads.

   Observe that *HBase* allows horizontal and vertical fragmentation. However, for vertical fragmentation, families must be explicitly declared. *MongoDB* only supports horizontal fragmentation.

   Since we assume that some data might mutate its contents through time, defining the set of families might not be suitable.

3. **Structure Complexity**: This an important question to address in No-SQL databases. It is observed that nowadays semi-structured data is more and more present (in the form of XML or JSON documents) because of the advantages it brings in terms of representing data. This is specially seen when we compare it to RDBMS tables.

In (1) we portrayed the absence of any kind of schema in key-value stores. In Document Stores, not only do documents behave as semi-structured data, but there is also the possibility to define a JSONschema, which allows integrity constraints.

In our case, we see that the idealista dataset shows some nested structure. The `suggestedTexts` key contains a nested dictionary by having two attributes inside it. Therefore, it seems that this structure is of value and should be kept.

4. **Queries**: *MongoDB* allows for querying in the sense that it keeps similar SQL query properties as well as indexing capabilities. On the contrary, key-value store provides no SQL style query langauge. If we chose Key-value over Document Store, we would have to choose the key wisely or to implement a naming convention. This is because the only way to retrieve data is by key. *HBase* offers minimal possibilities through the *Put* and *Get* operations.

   Also, intra-query parallelism is limited in *HBase* but inter-query parallelism is supported. MapReduce can be used on it. *MongoDB* also supports inter-query parallelism, but operator parallelism is very limited.

   According to the assumptions made, *MongoDB* seems more suitable for this project thansk to its querying capabilities. However, not being able to parallelise the joins is a drawback.

5. **Indexing**: *MongoDB* allows indexing in its values. If an index is not available, *MongoDB* scans the whole collection, looking for documents that satisfy the query, which impacts performance. Indexes are defined for a collection, and work similarly to RDBMSs. Note also that it creates a default index called `_id` that cannot be altered. Apache *HBase* does not have support for such indexes.

   In our case, we consider that having the capability is positive for queries. However, it must be stated that using many indexes comes at a cost as they take up space. Reading may be faster, but updates are slower.

3 Conclusions

It is important to acknowledge that there is not just one correct answer to the problem. The number of valid options exceeds the incorrect ones: arguments can be made in favor or against. Our choice was two-fold: Firstly, we considered assumptions listed in 1.2. Afterwords, a list of advantages and disadvantages was made for each tool. In the end, we chose based on what seemed more adaptable to the project statement. The need for some kind of schema structure, query language and indexing capabilities made *MongoDB* a very strong choice. Additionally, the fact that a Big Data Management problem deals with copious amounts of changing data made *HDFS* a valid option for the Temporal Zone.

References

[1] Alberto Abelló and Sergi Nadal. *Big Data Management*. Universitat Politècnica de Catalunya, 2022.

[2] Open Data BCN. List of culture and leisure spaces and facilities in the city of Barcelona. `https://opendata-ajuntament.barcelona.cat/data/en/dataset/equipament-cultura-i-lleure`, 2022.