BDM: Lab Assignment 1

Arnau Abella
Antoni Casas

Universitat Politècnica de Catalunya

April 18, 2021

A Data Model

A.1 Design

The design is optimized upon the following descriptive KPIs: "Average number of new listings per day", "Correlation of rent price and family income per neighborhood", "Listings per hospital in 1km radius", "Average listings divided by residents per neighborhood". We also further assumed an even distribution over the querying of these KPIs.

The data from OpenData BCN was kept mostly equal, with the exception of the removal of the OpenData codes, which hold no meaning outside of the OpenData domain.

The data from Idealista regarding housing was severely changed. It was organized into a list of values, in a tabular fashion maintaining the attributes of propertyCode, price, district, neighborhood, date, propertyType, status, size, rooms, bathroom, latitude, longitude, distance, newDevelopment and priceByArea.

The third dataset considered was data from OpenData, the dataset "Hospitals and primary health services of the city of Barcelona", it hasn't been added in this part of the project, but its future addition has been taken into account, and due to the KPIs we propose, only storing the longitude, latitude, name, street, street number, type (CAP/Hospital), district.

The data from the lookup tables was transformed to make conciliation between OpenData and Idealista datasets more efficient. The approach chosen was to take the Idealista lookup table and the OpenData lookup table, and bridging them together, so that instead of translating to an intermediate ID, they translate to each other. This way we can choose any of the datasets as being reconciliated by default, reducing the amount of reconciliation needed.

A.2 Storage

For its storage, we mainly took into account the expected need for high scalability of the design. Taking into account this property we chose HBase.

We discarded MongoDB, due to its lesser horizontal scalability, due to how its replica sets operate, requiring many more machines for every shard. While MongoDB would offer better vertical scalability due to its cache, this isn't enough to offset the lesser horizontal scalability.

However, MongoDB would offer the ability to define indexes, which could speed up reads via maximizing the documents selected to read, and, while this is a major loss, we consider this to not be such a major loss so as to choose MongoDB. We also consider the possibility of defining materialized views on HBase and HDFS, where we store a file with the keys of the documents following the desired property, so if indexes become important in future queries, their design can be extended to accommodate them, even if at a heavy application layer cost.

HDFS was discarded due to again, its horizontal scalability, while the use of special file formats like Parquet would aid in selections for queries, we would pay an upfront cost in metadata, and since we expect to compute many predictive KPIs, which will require as much of the data as possible to learn, any metadata will lead to a performance degradation when taking into account these workloads. However, the biggest issue with HDFS was its lack of a distributed catalog, relying instead on a single NameNode, which would limit the horizontal scalability of the design.

In the end, HBase offers the best horizontal scalability, due to its distributed catalog and nodes. Which is what we chose to optimize for our storage choice, we also took into account the existence of columnar families to maximize the useful read when reading the dataset for certain queries.

Due to how values are stored in HBase, we originally considered compressing the value data for efficiency, though we chose against it due to the low benefit in such a distributed architecture, that is because, either the decoding cost is paid by each node reading data, in which case the network cost, which is the bottleneck, stays constant, only improving on disk read times at the cost of decoding times. Or the data is sent compressed, in which case the receiving client must decompress all the data himself, which may worsen query times.

The column families of each store were chosen in respect of the proposed descriptive KPIs.

A.2.1 Lookup Tables

key: domainA value: domainB

We generate multiple stores, so our key varies, but the concept stays equal, that is, the store is a translation from A to B, using the key as A and the value as B. This was chosen as it is the optimal way to represent a translation from A to B.

HBase offers no join capacities, so data reconciliation should be inefficient, however, due to lookup table design chosen, the amount of reconciliation needed has been reduced as much as possible due to this inefficiency.

The lookup table has been designed as a table that translates all district neighborhood pairs into their Idealista version, and viceversa. Its future expansion has been considered, designing a column family for each translation from idealista into the new dataset and viceversa. Idealista was chosen as the reconciliated language due to simply having more elements, reducing this way the amount of reconciliation substantially.

A.2.2 Idealista

Key: propertyCode

Value: price, district, neighborhood, date, propertyType, status, size, rooms, bathroom, latitude, longitude, distance, newDevelopment, priceByArea

Our key is chosen as the only unique identifier that exists, as Idealista generates its property codes, and we can't make no guarantees over this ID other than it should be unique.

Needed data for each KPI:

- Average number of new listings per day: (date)
- Correlation of rent price and family income per neighborhood: (district, neighborhood,price)
- Listings per hospital in 1km radius: (latitude,longitude)
- Average listings divided by residents per neighborhood: (district, neighborhood)

Following these, we obtained that our column families, to optimize these KPIs, should be as follows:

- Family 1: price, district, neighborhood
- Family 2: date
- Family 3: latitude, longitude
- Family 4: propertyType, status, size, rooms, bathroom, latitude, longitude, distance, newDevelopment, priceByArea

Of importance is how to solve the problem of the reconciliation between OpenData and Idealista, since Idealista data is not segregated by years, as

OpenData's is. For this reason, we propose generating a new store for every year of listing, this comes at an application layer cost, as it behaves as a materialized view, but allows to select only the needed Idealista data for the KPIs, as the OpenData data is dependant on year.

A.2.3 OpenData

Key: district+neighborhood+year

Value: avg_rfd, population

Our key is chosen to be able to define basic selections, first, over all neighborhoods in a district, secondly, over all district and neighborhood pairs. This allows using the main catalog for some basic querying.

Needed data for each KPI:

- Average number of new listings per day: -
- Correlation of rent price and family income per neighborhood: (district, neighborhood, year,avg_rfd)
- Listings per hospital in 1km radius: -
- Average listings divided by residents per neighborhood: (district, neighborhood, year, population)

Since the triplet of district neighborhood year is our designed key, we only need to worry about how to distribute avg_rfd and population in our column families, however, since they are our only values, and the useful read is always at least 50%, we consider no need to split into column families.

B Data Transformation

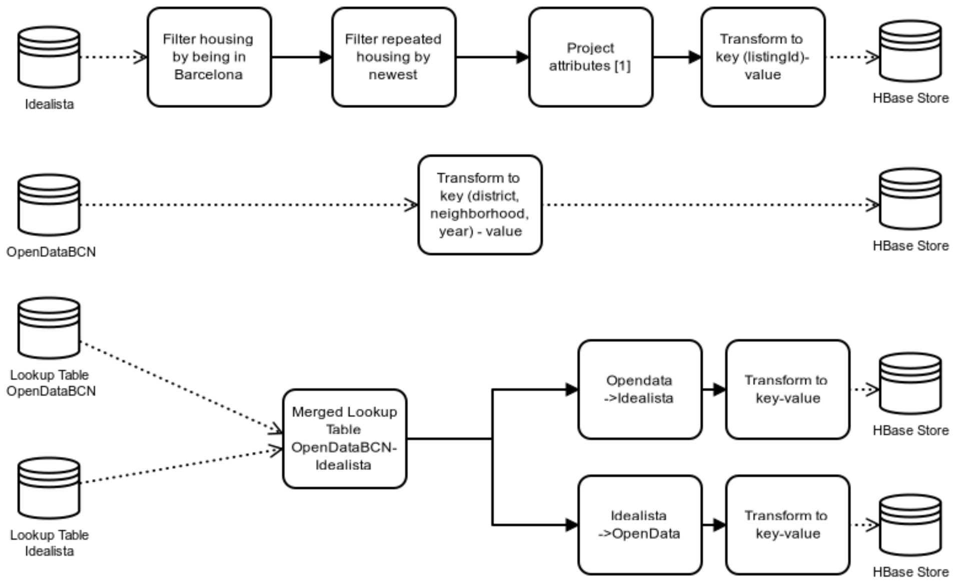

Figure 1: ETL Process

[1]: The projection obtains the following atributes: propertyCode, price, district, neighborhood, date, propertyType, status, size, rooms, bathroom, latitude, longitude, distance, newDevelopment, priceByArea

Our ETL process is extremely simple, only offering some complexity when dealing with the lookup tables, where they are fused into a single translation. Otherwise all it does is select and project before preparing it in a key-value format to store.