UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

OPEN DATA

Semantic COMPSs

Distributing data lakes and queries

Authors:
Ramon Amela Milian
Pol Alvarez Vecino

Professor:
Oscar Romero

May 8th 2017, Barcelona

Contents

1 Introduction

This project implements a distributed RDF data lake. Its purpose is to be able to have many data nodes containing RDF graphs, which can be different from sources or smaller partitions of the global (which may be too big to fit in a single node), that can be queried in a regular fashion.

The integration will be the main focus of the project describing the mappings in RDF and allowing distributed SPARQL queries without needing to merge all datasets into a single source. The only requirement for the distributed queries is that the user must explicitly define the class of subjects and objects of the query.

2 Goal

Considering the practice scope, the project has two main objectives:

Data integration automation

Create a tool to integrate several RDF databases in an automated fashion through a mappings configuration file.

Develop an update mechanism which handles the parsing and enrichment - for non-rdf sources - and the data update of each data node.

Distributed query system

In order to test the good behavior of the framework provided, data coming from different sources will be shown. The access will be done in a transparent way, hiding the fact that the information is coming from different sources. The graphs will be as complex as the integrated system allows.

3 Data lake and data flows

In our use case, we have two initial sources:

UNdata

United Nations information is presented as tabular data. More precisely, the information is in XML format.

United Nations

- Gross domestic product by expenditures at current prices
- http://data.un.org/Data.aspx?d=SNA&f=group_code%3a101
- Format: XML
- Fields used:
-
  - Country or Area: String with the name
  - Item: describing which expenditures the row references to
  - SNA 93 item Code: System of National Accounts Item code
  - Year: accounting year
  - Series: of the data
  - Currency: of the country

  - SNA System: system of national accounts
  - Fiscal year type: accounting period style
  - Value: gross product.

DBpedia

This source is already in RDF format so it is possible to import it directly. However we decided not to download it. Our system allow user to specify a list of SPARQL endpoints where the data resides. So UN data will reside in a data node (see Figure 2 while DBPedia will be queried directly.

DBpedia

- DBpedia Ontology RDF type statements (Instance Data)
- http://wiki.dbpedia.org/services-resources/ontology
- Format: RDF
- Fields used depend on the query.

3.1 Integration

The integration between the UN data and the DBPedia will be done using the country name. In the case of the UN data the attribute used will be the "country or area" column of the XML. For the DBPedia, the attribute used will be foaf:name.

Figure 1 details the UML of the UN Data. It does not provide the schema of the DBPedia because the data will already be in RDF and the user will the one responsible of knowing the schema before-hand in order to be able to query the data.

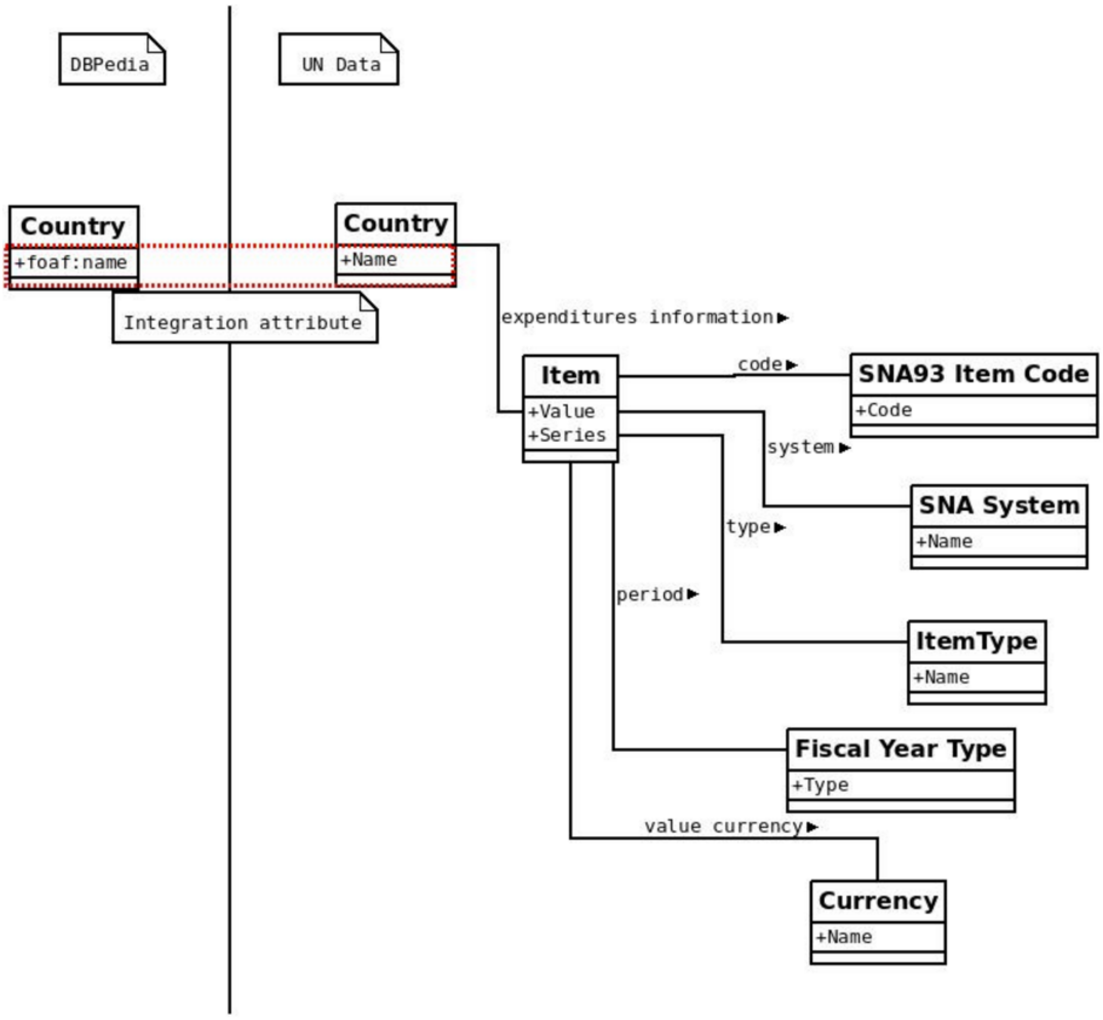

Figure 1: Schema showing the integration hypothesis for the two datasets.

.

Despite using the whole DBpedia to do the preliminary tests we will select:

- Capital (http://dbpedia.org/property/capital)
- Area (http://dbpedia.org/ontology/PopulatedPlace/areaTotal)
- Value (Item value) of the “Gross Capital Formation” (ItemType name) of each country.

3.2 COMP Superscalar

COMP Superscalar (COMPSs) is a framework which aims to ease the development and execution of applications for distributed infrastructures, such as Clusters, Grids and Clouds. It features a task-based programming model that will deal with all the infrastructure handling (i.e. communication between nodes, the queries to the storage nodes and query computation). The COMPSs runtime will be the actual endpoint (Query node) to which the user will send the queries. For this project, the query node will be a regular java application, but COMPSs allows to set the master node (the one orchestrating all the execution) as a web service which would be the next step when using this system in production.

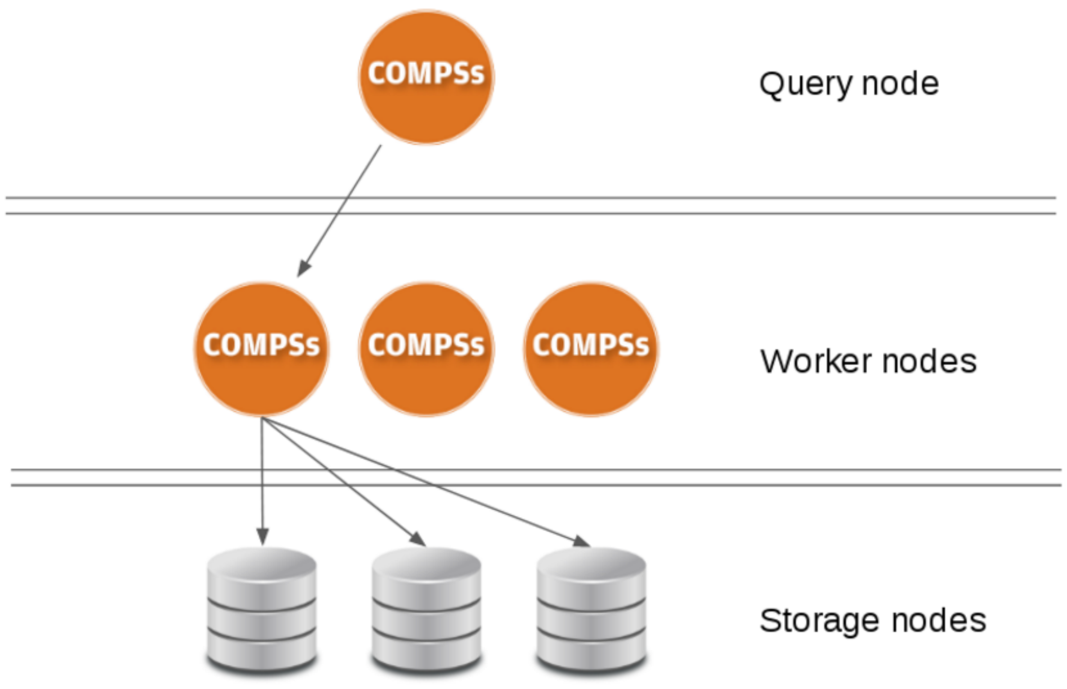

Figure 2: Distributed architecture schema. All nodes are orchestrated by a COMPSs master which is the one parsing and splitting the queries, keeping track of the cached values, and deciding the compute node to do the query (using data locality). Storage nodes are responsible of gathering a subgroup of data requested by the master. Worker nodes receive a list of subgraphs (produced by storage nodes), add them to their own current graph, and perform the query on their updated graph.

.

3.3 Workflow description

The workflow has been designed to allow complicated queries on a graph that doesn't fit a single machine. In addition, the layered schema permits a good scalability considering that the load can be distributed easily, both in the storage and the computational side. Query node

The query node will act as front-end, receiving all the queries from the users. The main constraint to the queries that are supported by the system is that the type of every subject and object in the query has to be specified (It will be possible to specify that the object is a literal). This way, we are able to detect which nodes of the canonical model are involved. For each case, the graph containing this nodes and all the edges linking them is built. This work is done in the worker nodes.

The queries done to the storage nodes are like the ones presented next (they are construct because we want graphs to be returned):

```
CONSTRUCT {?a ?b ?c}
WHERE {
        ?a rdf:type TYPE1 .
    ?a ?b ?c .
    FILTER(isLiteral(?c)).
}

CONSTRUCT ?a ?b ?c
WHERE {
  ?a rdf:type TYPE1 .
  ?a ?b ?c .
  ?c ref:type TYPE2.
}
```

Each query is executed in all the data lake sources. Once all the responses has been obtained, an entity resolution at instance level is performed in order to create a Jena model that contains all the information regarding this instances. This models created can be reused in the following queries. It has to be noted that it has been considered that a node or an edge are the minimum subset of the canonical model. Combining nodes and edges, a subset of the canonical model can be build in order to compute the query done by the user.

On the other hand, as orchestrator, the master will know which part is present in each worker node, and will schedule each query to be executed in a given node under a criteria of data locality to avoid the transferences between the different nodes of the system. If a part of the canonical model has never been loaded from the data nodes, the master orders to do this loading in a node.

3.3.1 Worker Nodes

The workers are responsible of computing each one of the queries that arrives to the system. There will be two kind of tasks executed by each worker:

Select tasks

1. Instance select

   The input for this kind of task will be a type. The output will be a model having all the instances of this type. In addition, this select will load all the relations between a given class and literals. Doing the analogy with the UML representation of the model, this select will load a class and its attributes. For example, to retrieve an Instance select of the class Country, the following queries will be performed:

   ```
   PREFIX xsd: <http://www.w3.org/2001/XMLSchema\#>
   PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
   PREFIX type: <http://dbpedia.org/class/yago/>
   PREFIX prop: <http://dbpedia.org/property/>
   SELECT distinct ?country ?relation ?literal
   WHERE {
   {
     ?country a type:Country108544813 .
     ?country ?relation ?literal .
     filter isLiteral(?literal) .
   }
   UNION
   {
     ?country rdfs:label ?lbl .
     ?country a type:Country108544813 .
   }}
   ```

2. Triplets select

   The input for this kind of task will be the two types involved. The output will be a model containing all the triplets between the instances of the two given types. In addition, for each triplet the corresponding instance model will be checked to verify the entity resolutions that has been done in order to update the subjects and objects in the relations.

Instance selects bring to the compute nodes all instances in separated subsets allowing to cache, combine and reuse them in future triplet selects (queries). Triplets select bring to the compute nodes all the edges linking the involved classes.

3. Query tasks A task will be have as input all the models containing the classes and the relations involved in the query. If the model is present in the node, it will be directly charged from memory. Otherwise, COMPSs will orchestrate all the transferences. The model will be transferred from an available node. Afterwards, all the models are unified and the query is computed.

The workflow can be resumed as follows:

1. New query is received
2. All the necessary models needed to compute the query that are not present in the node are transferred from the nodes that have done the computation before. This way we balance the network load. A part of the transference load is between the computation nodes and an other part is between the computation and the storage nodes.
3. Once all the models has been acquired, the triplets are all inserted into the node's local graph instance with an SPARQL update query.
4. The query is executed on the local graph instance.
5. The response is sent to the query node that will either output it to the terminal or to a file (depending on the selected execution mode)

3.3.2 Storage Nodes

In a first step, all the databases will be transformed to RDF format. Each node will be responsible for a number of external databases. Each night (or at another convenient time) the node will attempt to get the data from the sources again (e.g. download an XML/JSON) and update the node RDF information with it (previously converting it to RDF if needed). If the source is no longer available the node will just keep the data that already has guaranteeing the availability even if the source is down. The update process and possible mappings to RDF will be manually configured previously.

The conversion from XML (or any other source) to RDF will be done using R2RML-parser, included in Apache Jena. All the relations will be mapped to the canonical model. This way we ensure that there is no need to do entity resolution at predicate level.

All the rest of the workflow and query management is done with Python. Rdflib [1] and SparqlWrapper [2] provide an easy interface to query endpoints, build graphs, and manipulate them from Python. In additions, pyCOMPSs (python version of COMPSs) handles natively arbitrary-length functions allowing us to pass a variable number of inputs per query without needing to define a function for each size.

Finally, the storage nodes (for the scope of this project only one containing the UN data) and the local graphs (for each worker node) are a Fuseki instance with persistence through TDB and different endpoint.

Storage nodes are supposed to be up and running prior to queries execution and they are not handled by the system (just the list of endpoints needs to be passed to the query node). On the other hand, each worker node has a Fuseki server assigned to it. This one is started by COMPSs. All worker fuseki run on the same address but different port (to allow multiple workers in the same machine if needed) which is exposed to Python through environment variables.

[1] http://rdflib.readthedocs.io/en/stable/
[2] https://rdflib.github.io/sparqlwrapper/

We consider that, at this stage, all the databases have been already stored in RDF format and we are able to map each single database to the canonical model. In the second case, this fact is assured knowing that we will define manually the mappings from the SQL database to the RDF, which will be coherent with the canonical model.

4 Entity Resolution

The foaf:name and "country or area" information are the attributes that will be used to join the two datasets. We will use Levenshtein distance to detect matches between the attributes. For each pair of countries (resolution is only performed in countries subjects/objects, and our requirements ensure they will be alway selectable by class), if one of the two contains the other or the Levenshtein distance is smaller than 4 their are considered equal. The resolution process then checks if there is a tuple specifying that this pair resolution is incorrect (using owl:differentFrom relation stored in a configuration file) and, if there is not, create the mapping. Similarly, once all pairs have been checked, the same configuration file will be checked to add relations that have not been detected (using owl:sameAs).

This task is first performed during triples select where it creates the mappings between instances. This mappings will be passed along the query inputs to the query tasks. It is there were all the URI present in the mappings will be replaced by their corresponding replacement. This replaced triples will then be added to the local graph together with the ones that did not required replacement.