Ephedra: SPARQL federation over RDF data and services

Andriy Nikolov, Peter Haase, Johannes Trame, Artem Kozlov

metaphacts GmbH, Walldorf, Germany
{an, ph, jt, ak}@metaphacts.com

Abstract. Knowledge graph management use cases often require addressing hybrid information needs that involve a multitude of data sources, a multitude of data modalities (e.g., structured, keyword, geospatial search), and availability of computation services (e.g., machine learning and graph analytics algorithms). Although SPARQL queries provide a convenient way of expressing data requests over RDF knowledge graphs, the level of support for hybrid information needs is limited: existing query engines usually focus on retrieving RDF data and only support a set of hard-coded built-in services accessible via SPARQL 1.1 queries. To deal with this problem, we present Ephedra: a SPARQL federation engine aimed at processing hybrid queries, which provides a flexible declarative mechanism for including hybrid services into a SPARQL federation.

1 Introduction

In many practical knowledge graph management use cases there is a need to address *hybrid information needs*. Such needs can be characterized by the following dimensions:

- *Variety of data sources*, including RDF repositories as well as other datasets presented as RDF (e.g., a relational database exposed using R2RML mappings).
- *Variety of data modalities*, in addition to RDF graph data: e.g., textual, temporal, or geospatial data.
- *Variety of data processing techniques*, e.g., graph analytics, statistical analysis, machine learning, etc.

The main motivation for this work comes from our experience with the *metaphactory* knowledge graph management platform[1], which is used in a variety of application domains (e.g., cultural heritage, life sciences, pharmaceutics, and IoT infrastructure). Typical application scenarios often require dealing with a multitude of the above-listed aspects simultaneously: e.g., an example request like "give me the artists who collaborated with Rembrandt and others similar to them" involves (a) keyword search for an RDF resource based on the keyword "rembrandt", (b) structured search over the RDF graph for collaborators, and (c) applying an external model (vector space similarity, such as word2vec [1]) to find other similar entities.

To handle such use case scenarios involving hybrid information needs, we developed Ephedra: a federated SPARQL query processing engine targeted at processing hybrid

[1] http://www.metaphactory.com/

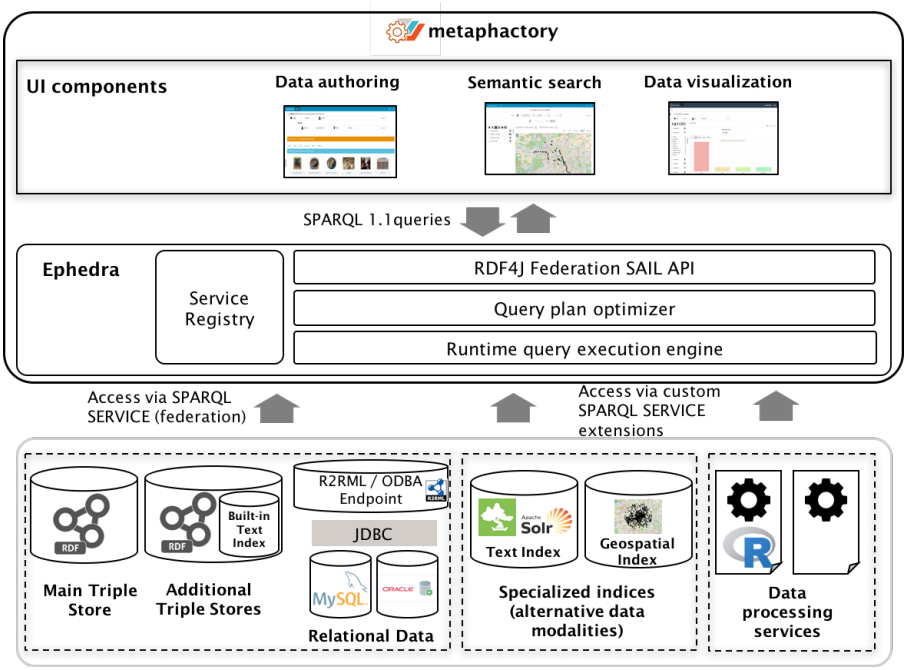

Fig. 1. Ephedra in the *metaphactory* platform architecture.

queries. SPARQL 1.1 with its SERVICE clauses provides a convenient data retrieval formalism: a complex information request over several data sources can be expressed using a single query. With Ephedra we adopt the SPARQL 1.1 federation mechanism, but we broaden its usage to include custom services as data sources and optimize such hybrid queries to be executed efficiently.

Expressing a complex hybrid information request using a SPARQL query can be non-trivial due to the variety of potential types of services and the limitations of the SPARQL syntax: e.g., a service can take as input a single set of parameters or a list of arbitrary length; it can return as output one value, several values or a table of multiple records, etc. With Ephedra we take these factors into account to support practical hybrid federation use cases of the *metaphactory* platform and address hybrid information needs in an efficient way. In this paper, we propose a reusable architecture in which hybrid services can be easily plugged in, described in a declarative way, and invoked using federated SPARQL queries.

2 Hybrid SPARQL federation

To support the hybrid querying functionality, we developed the Ephedra query processing engine as a part of the *metaphactory* platform. A crucial requirement is the ability to plug in additional services with minimal effort and reference them from SPARQL.

Figure 1 shows the generic architecture of the *metaphactory* platform. Ephedra is used as a hybrid query federation layer to access the data repositories and services. In our use case scenarios, the user's information need is captured interactively: the user can define search clauses, explore partial results, incrementally add new clauses, while the system provides relevant suggestions. These interactions generate information requests that are expressed as SPARQL 1.1 queries and given to Ephedra to process them.

As the basis for Ephedra implementation, we used the RDF4J Federation SAIL API[2] reusing the common functions such as query parsing and accessing remote SPARQL endpoints. However, Ephedra extends the RDF4J object model and overrides the static optimization and query execution strategies to deal with hybrid queries. The Ephedra query evaluation strategy sends the sub-clauses of the query to the corresponding data sources and invokes the relevant processing services, then gathers the partial results, combines them using the union and join operations, and produces the final result set. In this way, processing becomes transparent: hybrid information needs are processed in the same way as ordinary SPARQL queries to an RDF triple store without the need to integrate related processing services at the UI level.

In order to configure the services as federation members, the system requires relevant information about the *service type* as well as *service instances*. Ephedra includes two types of hybrid services: *extension services* and *aggregate services*. Extension services take as input a partial query solution (binding set) and extend it with additional variable bindings. Extension services are called in the query via a SPARQL SERVICE clause. On the contrary, aggregate services operate over a set of multiple query solutions as the SPARQL aggregate functions (e.g., AVG, MIN, MAX) do: they take as input a list of records and produce one or more resulting binding sets. As with the SPARQL aggregates, aggregate services are referenced as function calls in the SELECT clause.

Relevant meta-level information about the hybrid service types is summarized using the service descriptors structured according to the *service description ontology*. The ontology expands the well-known SPIN[3] ontology for SPARQL query engines to capture the relevant parameters of services.

A service descriptor contains the following information:

- Input parameters and their expected datatypes. An input parameter is described using the SPIN ontology vocabulary as a *spl:Argument* resource.
- Output parameters and their expected datatypes. An output parameter is described as a *spin:Column* resource in the SPIN ontology.
- Expected graph pattern. The special triple patterns expected by the service are expressed using the SPIN SPARQL syntax[4]. The placeholders for input/output parameters are expressed as resources which are referenced from the input/output parameter descriptors.
- Input and output cardinalities of a service call (optional).

```
:WikidataTextSearch a eph:Service ;
  rdfs:label "A wrapper for the Wikidata test search." ;
  eph:hasSPARQLPattern ([
```

[2] http://docs.rdf4j.org/sail/

[3] http://spinrdf.org/

[4] http://spinrdf.org/sp.html#sp-TriplePattern

```
        sp:subject :_uri ;
        sp:predicate wikidata:search ;
        sp:object :_token ]) ;
  spin:constraint [
    a spl:Argument ;
    rdfs:comment "Input token" ;
    spl:predicate :_token ;
    spl:valueType xsd:string ] ;
  spin:column [
    a spin:Column ;
    rdfs:comment "URI of the Wikidata resource" ;
    spl:predicate :_uri ;
    spl:valueType rdf:Resource ] .
```

A descriptor for an aggregation service declares the input and output parameters in a similar way, but instead of the list of triple patterns it defines a custom aggregate function which will be referenced by its URI.

2.1 Implementing service extensions

To simplify the integration of new hybrid services into the framework, the architecture provides a generic API to wrap arbitrary services and include them as SPARQL federation members. To this end, Ephedra reuses and extends the RDF4J SAIL API. A service is represented as a SAIL module which is responsible for extracting the values of input parameters from a given SPARQL tuple expression, executing the actual service call, and returning the results by binding resulting values to the output variables. Ephedra provides abstract implementations for a generic service SAIL as well as a specific wrapper for HTTP services. The common routines, such as extracting the input values and output variables and wrapping the results as binding sets do not depend on the actual service and are performed in a generic way using the declarative service descriptor.

3 Outlook

The main directions for the future work concern further minimizing the adaptation effort needed to deploy *metaphactory* in a new use case. This involves, for example, building a library of reusable data analytics services (e.g., for common machine learning algorithms). There are also several promising directions for improving the query performance, in particular, exploiting more detailed meta-data about other types of federation members: e.g., summary of the content for RDF triple stores and sets of R2RML mappings for relational databases.

Acknowledgements

This work has been supported by the Eurostars project DIESEL (E!9367) and by the German BMWI Project GEISER (project no. 01MD16014).

References

1. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. (2013) 3111–3119