A Comparison of Federation over SPARQL Endpoints Frameworks

Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas

Digital Enterprise Research Institute
National University of Ireland, Galway
{firstname.lastname}@deri.org

Abstract. The increasing amount of Linked Data and its inherent distributed nature have attracted significant attention throughout the research community and amongst practitioners to search data, in the past years. Inspired by research results from traditional distributed databases, different approaches for managing federation over SPARQL Endpoints have been introduced. SPARQL is the standardised query language for RDF, the default data model used in Linked Data deployments and SPARQL Endpoints are a popular access mechanism provided by many Linked Open Data (LOD) repositories. In this paper, we initially give an overview of the federation framework infrastructure and then proceed with a comparison of existing SPARQL federation frameworks. Finally, we highlight shortcomings in existing frameworks, which we hope helps spawning new research directions.

1 Introduction

The Resource Description Framework (RDF) was introduced since 1998 and now has become a standard for exchanging data in the Web. At present, huge amount of data has been converted to RDF. The SPARQL Protocol and RDF Query Language (SPARQL)[1] was officially introduced in 2008 to retrieve RDF data as easily as SQL does for relational databases. As Web of data grows and more applications rely on it, the number of SPARQL Endpoints constructing SPARQL queries over Web of Data using HTTP also grows fast. SPARQL Endpoint becomes main preferences to access data because it is a flexible way to interact with Web of Data by formulating query like SQL in traditional database. Additionally, it returns query answer in several formats, such as XML and JSON which are widely used as a data exchange standards in various applications. This situation has attracted people to aggregate data from multiple SPARQL Endpoints akin to conventional distributed databases. For instance, NeuroWiki[2] collects data from multiple life science RDF stores by utilizing LDIF framework [32] and RKBexplorer(`http://www.rkbexplorer.com/explorer/`) gathers research

[1] `http://www.w3.org/TR/rdf-sparql-query/`
[2] `http://neurowiki.alleninstitute.org/`

publication information from more than 20 datasets under `rkbexplorer.com` domain [20].

Contributions. In this study, we will focus primarily on the federation over SPARQL Endpoint infrastructure, as the LOD cloud statistics[3] reports that 68.14% of the RDF repositories are equipped with SPARQL Endpoints. Aside from giving an overview of querying over SPARQL Endpoints, we will compare the existing federation frameworks based on their platform, infrastructure properties, query processing strategies, etc.—to chose any framework for small and large-scale systems. Further, we highlight shortcomings in the current federation frameworks that could open an avenue in the research of federated queries.

Related Works. More general investigations w.r.t. querying Linked Data have been performed elsewhere [6, 9, 18]. [6] mentioned nine myths and five challenges arising in the Federation over Linked Data. Based on their observation, they suggested to consider Linked Data as a service not as distributed data. [9] explained the Federated SPARQL query infrastructure, whereas [18] focused on the basics of federated query processing strategy.

A number of studies [23, 34] compares federation frameworks by evaluating their performance. [23] tests federation frameworks by using FedBench [29] in various networking environment and data distribution. Similar to [23], [34] conducts an experiment in the FedBench to evaluate federation frameworks on large scale Life Science datasets. In this survey, we investigate and compare more existing Federation over SPARQL Endpoint frameworks based on their strategy such as source selection and execution plan.

Structure of the Paper. An overview of federation architectures of querying over SPARQL Endpoints are presented in Section 2. Section 3 introduces the existing federation frameworks, supporting either SPARQL 1.0 or 1.1[4]. We also categorize them based on their architecture and querying process and investigate features that should be added in the existing frameworks. Finally, we discover challenges that should be considered in the future development of federation query in Section 5. We conclude our findings in Section 6. For full version of this paper we refer to [27].

2 Architecture of Federation over SPARQL Endpoints

SPARQL 1.1 is designed to tackle limitations of the SPARQL 1.0, including updates operations, aggregates, or federation query support. As of this writing, not all query engines support SPARQL 1.1. Therefore, we discuss the federation frameworks that support either SPARQL 1.0 or SPARQL 1.1. There are three kinds of architecture of federation over SPARQL Endpoints (Figure 1) namely a) the framework has capability to execute SPARQL 1.1 query, b) the framework accepts SPARQL query without specifying SPARQL Endpoint address, then it rewrites query to SPARQL 1.1 syntax before passing it to the SPARQL 1.1 engine

[3] `http://lod-cloud.net/state/`

[4] `http://www.w3.org/TR/sparql11-query/`

Query 1.1. Example of Federated SPARQL Query in the SPARQL 1.0

```
SELECT ?drugname ?indication {
FROM <http://localhost/dbpedia.rdf> {   ?drug a dbpedia-owl:
    Drug .
        ?drug rdfs:label ?drugname .
        ?drug owl:sameAs ?drugbank . }
FROM <http://localhost/drugbank.rdf> {  ?drugbank drugbank:
    indication ?indication . }
}
```

Query 1.2. Example of Federated SPARQL Query in the SPARQL 1.1

```
SELECT ?drugname ?indication {
SERVICE <http://dbpedia.org/sparql> {   ?drug a dbpedia-owl:
    Drug .
        ?drug rdfs:label ?drugname .
        ?drug owl:sameAs ?drugbank . }
SERVICE <http://www4.wiwiss.fu-berlin.de/drugbank/sparql> { ?
    drugbank drugbank:indication ?indication . } }
```

and c) the framework handles SPARQL query without SERVICE keyword and processes the query in several phases by interacting with the SPARQL 1.0/1.1 engine of each SPARQL Endpoints. Those systems that have already supported SPARQL 1.1 allow user to execute query federation over SPARQL Endpoints by using *SERVICE* keyword (Figure 1.a). The query processor distributes each sub query to defined SPARQL Endpoints and join the result from SPARQL Endpoint. Basically, SPARQL 1.0 allows us to query data from remote data sources, however it does not retrieve specified remote SPARQL Endpoints. As described in the Query 1.1., it only fetches remote graphs or graphs with the name in a local store.

At present, the SPARQL 1.1 is the simplest solution to yield data from multiple sources. The W3C recommendation of the SPARQL 1.1 formalizes rules to query in multiple SPARQL Endpoints by using *SERVICE* operator. However, users must have prior knowledge regarding the data location before writing a query because the data location must be mentioned explicitly. As seen in the Query 1.2, the Drugbank and DBPedia SPARQL Endpoints are mentioned after SERVICE operator to obtain the list of drugs and their associated diseases. In order to assist users in term of data source address, it allows us to define a list of SPARQL Endpoints as data beforehand and attach the list of SPARQL Endpoint address as variable in the SPARQL query. Besides *SERVICE*, SPARQL 1.1 also introduces *VALUES* as one of the SPARQL Federation extension which can reduce the intermediate results during query execution by giving constrains from the previous query to the next query.

Query 1.3. Example of Federation SPARQL Query in the SPARQL 1.0 without SPARQL Endpoint specified

```
SELECT ?drugname ?indication {
?drug a dbpedia-owl:Drug . ?drug rdfs:label ?drugname .
?drug owl:sameAs ?drugbank . ?drugbank drugbank:indication ?
    indication . }
```

The lack of knowledge of data information is a main problem to execute federated query throughout multiple single RDF stores. Thus, several efforts have been introduced to address that issue (Figure 1.b). The user can write a query blindly without knowing the data location. These federation models can execute Query 1.2 or Query 1.1 without a SPARQL Endpoint declared. By removing SERVICE or FROM keywords, those two queries can be replaced by Query 1.3. These framework architectures provide an interface to translate query from SPARQL 1.0 to SPARQL 1.1 format. The core part of this interface is query rewriting component. After parsing and decomposing the query, this component adds destination address of this query by inserting *SERVICE* operators in each sub query. Further on, the result of query rewriter will be executed by internal SPARQL 1.1 processor system.

Since not all of the existing SPARQL Endpoints can handle SPARQL 1.1 query, several systems (Figure 1.c) developed query execution processor to execute the federated SPARQL query without SPARQL Endpoint address declared. The processor has responsibility to manage query processing such as maintain data catalogue, determine relevant sources, plan the query execution and join all results after retrieving data from SPARQL Endpoints. The details of phase

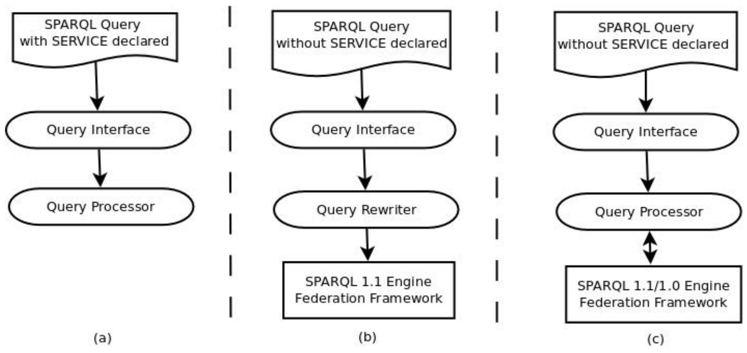
SPARQL Query with SERVICE declared
Query Interface
Query Processor
(a)
SPARQL Query without SERVICE declared
Query Interface
Query Rewriter
SPARQL 1.1 Engine Federation Framework
(b)
SPARQL Query without SERVICE declared
Query Interface
Query Processor
SPARQL 1.1/1.0 Engine Federation Framework
(c)

Fig. 1. Architecture of Federation over SPARQL Endpoint

of querying process in the Federated Over SPARQL Endpoint for architecture **b** and **c** can be found at [27].

3 The Existing Federation over SPARQL Endpoints Frameworks

This section presents the insight on existing federation over SPARQL Endpoint based on their architectures. To give better explanation, they will be classified based on their features. Ultimately, we propose several features that should be considered for next development.

3.1 The Existing Federation over SPARQL Endpoints Frameworks Based on Their Architecture

As described in the section 2, three federation architectures categories have been developed recently. We explain the existing federation based on those categories in this section.

Frameworks Support SPARQL 1.1 Federation Extension
As of this writing, several RDF store systems have been able to process federation query, but not all of them support *VALUES* keyword. Instead of handling *VALUES*, a number of frameworks has supported *BINDING* which is also addressed to reduce the size of intermediate results. The existing frameworks supporting SPARQL 1.1 are as follows:

ARQ[5], a query engine processor for Jena, has supported federated query by providing *SERVICE* and *VALUES* operator. ARQ implements nested loop join to gather retrieved result from multiple SPARQL Endpoints. In term of security, the credential value to connect ARQ service must be initialized in the pre-configuration[6].

Sesame. Previously, Sesame already supported federation SPARQL query by using SAIL AliBaba extension [7] at 2009, but Sesame can not execute SPARQL 1.1. Instead, SAIL Alibaba integrates multiple datasets into a virtual single repository to execute federated query in SPARQL 1.0. It can execute federation SPARQL query either RDF dump or SPARQL Endpoint by using its API. The data source must be registered in advance during setup phase. The simple configuration file only containing the list of SPARQL Endpoint address can cause poor performance since it sends query to all data sources without source selection. In order to optimize the query execution, Sesame offers additional features in the configuration file namely predicate and subject prefixes owned by one dataset. According to its configuration, Sesame can do prefix matching to predict the relevant source for a sub query. The join ordering is based on calculation of the size of basic graph pattern. The new version of Sesame (2.7)[8] is able to handle

[5] `http://jena.apache.org/documentation/query/index.html`
[6] `http://jena.hpl.hp.com/Service#`
[7] `http://www.openrdf.org/doc/alibaba/2.0-alpha2/alibaba-sail-federation/index.html`
[8] `http://www.openrdf.org/index.jsp`

SPARQL 1.1 which provides Federation extension features including *SERVICE* and *VALUES* operator.

SPARQL-FED Virtuoso. 6.1.6 allows to execute SPARQL queries to remote SPARQL Endpoint through SPARQL-FED[9]. The remote SPARQL Endpoint must be declared after SERVICE operator.

SPARQL-DQP. SPARQL-DQP is built on top of the OGSA-DAI [3] and OGSA-DQP [19] infrastructures. SPARQL-DQP transforms the incoming query from SPARQL to SQL, as it implements SQL optimization techniques to generate and optimize query plans. The optimization strategy is based on OGSA-DQP algorithm which does not need any statistic information from data sources. No SPARQL Endpoint registration is required because the SPARQL Endpoint must be written in the query. The OGSA-DAI manages a parallel hash join algorithm to reorder query execution plan.

Frameworks Supporting Federation Without Re-writing Queries to SPARQL 1.1

In order to overcome the lack of knowledge of data location, several federation frameworks have been developed recently without specifying SPARQL Endpoint address in the query. The federation framework acts as mediator [16] transferring SPARQL query from user to multiple data sources either RDF repository or SPARQL Endpoints. Before delivering a query to related source, it breaks down a query into sub queries and selects the destination of each sub query. In the end, the mediator must join the retrieved results from the SPARQL Endpoints. Following are overview of the current of federation frameworks and summarized in the Table 1.

DARQ(Distributed ARQ) [25] is an extension of ARQ which provides transparent query to access multiple distributed endpoints by adopting query mediator approach. The service description which consists of data description and statistical information has to declare in advance before query processing since it assists to predict where a sub query should go. According to the list of predicates in the service description, DARQ re-writes the query, creates sub query and designs the query planning execution. The query planning is based on estimated cardinality cost. DARQ implements two join strategies : Nested Loop Join and Bind Join.

Splendid [10] extends Sesame which employs VoID as data catalogue. The VoID of the dataset is loaded when the system started then the ASK SPARQL query is submitted to each dataset for verification. Once the query is arrived, the system builds sub queries and join order for optimization. Based on the statistical information, the bushy tree execution plan is generated by using dynamic programming [35]. Similar to DARQ, Splendid computes join cost based on cardinality estimation. It provides two join types: hash join and bind join to merge the results locally.

FedX [33] is also developed on top of the Sesame framework. It is able to run queries over either Sesame repositories or SPARQL Endpoints. During

[9] `http://www.openlinksw.com/dataspace/dav/wiki/Main/VirtSparqlCxml`

initial phase, it loads the list of data sources without its statistical information. The source selection is done by sending SPARQL ASK queries. The result of a SPARQL ASK query is stored in a cache to reduce communication for the successive query. The intermediate result size can be minimized by a rule based join optimizer. FedX implements Exclusive Groups to cluster related patterns for one relevant data source. Besides grouping patterns, it also groups related mapping by using single SPARQL UNION query. Those strategies can decrease the number of query transmission and eventually, it reduces the size of intermediate results. As complementary, it came with Information Workbench for demonstrating the federated approach to query processing with FedX.

ADERIS (Adaptive Distributed Endpoint RDF Integration System) [17] fetches the list of predicates provided by data source during setup stage. The predicate list can be used to decide destination source for each sub query pattern. During query execution, ADERIS constructs predicate tables to be added in query plan. One predicate table belongs to one sub query pattern. The predicate table consists of two columns: subject and object which is filled from the intermediate results. Once two predicate tables have been completed, the local joining will be started by using nested loop join algorithm. The predicate table will be deleted after query is processed. ADERIS is suitable for data source who does not expose data catalogue, but it only handles limited query patterns such as UNION and OPTIONAL. ADERIS provides a simple GUI for configuration and query execution.

Avalanche [4] does not maintain the data source registrations as its data source participants depends on third-parties such as search engines and web directories. Apart from that, it also stores sets of prefixes and schemas to special endpoints. The statistics of a data source is always up to date since Avalanche always requests the related data sources statistic to the search engine after query parsing. To detect the data sources that contributes to answer a sub query, Avalanche calculates the cardinality of each unbound variables. The combinations of sub queries are constructed by utilizing the best first search approach. All sub queries are executed in parallel processes. To reduce the query response time, Avalanche only retrieves the first K results from SPARQL Endpoints.

Graph Distributed SPARQL (GDS) [37] overcomes the limitation of their previous work [38] which can not handle multiple graphs. GDS is developed on top of Jena platform by implementing Minimum Spanning Tree (MST) algorithm and enhancing BGP representation. Based on Service Description, MST graph is generated by exploiting Kruskal algorithm. The MST graph estimates the minimum set of triple patterns evaluation and the lowest cost of execution order. The query planning execution can be done by either semi join or bind join which is assisted by a cache system.

Distributed SPARQL. In contrast to aforementioned frameworks, users must declare the SPARQL Endpoint explicitly in the SPARQL query at Distributed SPARQL [39]. Since it is developed for SPARQL 1.0 user, the SPARQL Endpoint address is mentioned after *FROM NAMED* clause. Consequently, this framework does not require any data catalogue to execute a query. As part of

Table 1. The existing frameworks support federation over SPARQL Endpoint without reformulating queries to SPARQL 1.1.

Framework	Catalogue	Platform	Source Selection	Cache	Query Execution	Source Tracking	GUI
DARQ	Service Description	Jena	Statistic of Predicate	✓	Bind Join or Nested Loop Join	Static	✗
ADERIS	Predicate List during setup phase	✗	Predicate List	✗	Nested Loop Join	Static	✓
FedX	✗	Sesame	ASK	✓	Bind Join parallelization	Dynamic	✓
Splendid	VoID	Sesame	Statistic + ASK	✗	Bind Join or Hash Join	Static	✗
GDS	Service Description	Jena	Statistic of Predicate	✓	Bind Join or Semi Join	Dynamic	✗
Avalanche	Search Engine	Avalanche	Statistic of predicates and ontologies	✓	Bind join	Dynamic	✗
Distributed SPARQL	✗	Sesame	✗	✗	Bind join	✗	✗

Networked Graphs [28], it is also built on the top of Sesame. To minimize the number of transmission query during execution, Distributed SPARQL applies distributed semi join in the query planning.

Frameworks Re-writing Queries to SPARQL 1.1

A number of frameworks were developed to accepts SPARQL query federation in SPARQL 1.0 format, but they are built on top of SPARQL query engine that support SPARQL 1.1 (Table 2.)

ANAPSID. ANAPSID [1] is a framework to manage query execution with respect to data availability and runtime condition for SPARQL 1.1 federation. ANAPSID enhances XJoin [36] operator and combines it with Symmetric Hash Join [8]. Both of them are non blocking operator that save the retrieved results to the hash table. Similar to others frameworks, ANAPSID also has data catalogue containing the list of predicates. Additionally, the execution time-out information of SPARQL Endpoint is added in the data catalogue. Therefore, the data catalogue is updated on the fly. Apart from updating data catalogue, ANAPSID also updates the execution plan at runtime. The Defender [21, 22] in ANAPSID has purpose to split up the query from SPARQL 1.0 format to SPARQL 1.1 format. Not only splitting up the query, Defender also composes related sub query in the same group by exploiting the bushy tree strategy.

SemWIQ. SemWIQ is another system building on top of ARQ and part of the Grid-enabled Semantic Data Access Middleware (G-SDAM). It provides

Table 2. The Existing Frameworks Supports Federation over SPARQL Endpoints, Reformulate query to SPARQL 1.1.

Framework	Catalogue	Platform	Source Selection	Cache	Query Execution	Source Tracking	GUI
SemWIQ	RdfStats+VoID	Jena	Statistic + Service	✓	Bind Join	Dynamic	✓
Anapsid	Predicate List and Endpoint status	Anapsid	Predicate List	✗	Symmetric Hash Join and XJoin	Dynamic	✓
WoDQA	VoID Stores	Jena	List of predicates and ontologies	✗	✗	Dynamic	✓

Query 1.4. Example of Link Predicate Problem

```
SELECT ?drug ?compoundname {
?drug drugbank:keggCompoundID ?compound .
?compound rdfs:label ?compoundname . }
```

a specific wrapper to allows data source without equipped SPARQL Endpoint connected. The query federation relies on data summaries in RDFStats and SDV[10]. RDFStats is always up-to-date statistic information since the monitoring component periodically collects information at runtime and stores it into a cache. As the RDFstats also covers histogram String, Blank Node etc, it is more beneficial for SemWIQ to be able to execute any kind of query pattern. SDV is based on VoID which is useful for data source registration. The query is parsed by Jena SPARQL processor ARQ before optimization process. SemWIQ applies several query optimisation methods based on a number of statistic cost estimations such as push-down of filter expressions, push down of optional group patterns, push-down of joins and join and union reordering. During optimization, the federator component inserts SERVICE keyword and SPARQL Endpoint for each sub query.

WoDQA. WoDQA (Web of Data Query Analyzer) [2] also uses ARQ as a query processor. The source selection is done by analysing metadata in the VoID stores such as CKAN[11] and VoIDStore[12]. The source observation is based on Internationalized Resource Identifier (IRI), linking predicate and shared variables. WoDQA does not exploit any statistic information in the VoID of each dataset, but it only compares IRI or linking predicate to subject, predicate and object. The same variables in the same position are grouped in one sub query. After detecting relevant sources for each sub query, the SERVICE keyword is appended following with SPARQL Endpoint address.

[10] `http://purl.org/semwiq/mediator/sdv#`
[11] `http://ckan.net/`
[12] `http://void.rkbexplorer.com/`

4 Desired Features

We have seen existing federation SPARQL frameworks along with their behaviours and properties. Based on our summary and experience, we suggest several features that could be added into their framework.

Hybrid Data Catalogue. As described in the section 3, the data source registration could be done by the mediator as well as third party such as search engine. In term of querying in the Linked Open Data, the data source registered should be not limited. The framework could combine static and dynamic data sources registration where the data source in the static registration is given higher priority than data source in the dynamic registration before delivering a query.

Link Predicate Awareness. Link predicate has ability to connect one entity to another entity in the different sources. The details of Link predicate definition can be found at [27]. For instance, the `drugbank:keggCompoundID` links the entity `drugbank:drugs` in the Drugbank dataset with the class `kegg_resource:Compound` in the Kegg dataset and `owl:sameAS` connects the entity `dbpedia-owl:Drug` in the DBpedia dataset with the entity `drugbank:drugs` in Drugbank. Assuming the link predicate connects two datasets, we should avoid to deliver the same destination of query patten containing the link predicate. In the case of Query 1.4, pattern *?compound rdfs:label ?compoundname* should not be sent to the Drugbank dataset as `drugbank:keggCompoundID` is a link predicate, even predicate *rdfs:label* occurs in all datasets.

5 Challenges

According to our investigation in Section 3, we note several challenges to be addressed in the future of federation framework development. Federation over SPARQL Endpoint has become actively developed over the last four years, especially in the source selection part. This field area is still infancy which faces several challenges that need to be tackled:

Data Access and Security. The data source and mediator are usually located in different locations, therefore the secure communication process among mediator and data sources should be concerned. Several SPARQL Endpoints provide authentication feature to restrict query access for limited user. However, the unauthorized interception between mediator and data sources have not been undertaken by any federation frameworks yet. The public key cryptography could be implemented in the federation frameworks where mediator and data source share public and private key for data encryption in the interaction process.

Data Allocation. Since several RDF stores crawl data from other data source, the data redundancy could not be avoided in the Linked Open Data Cloud. Consequently, the federation over SPARQL Endpoints framework detects relevant sources from multiple locations. This such condition could increase communication cost during selection source and query execution stage particularly for federation system employing data statistic from third-party. Furthermore, the redundancy data could increase the intermediate results as more data duplication from multiple source. On the one hand, using popular vocabulary allows user to query easily, but on the other hand, the source prediction for a query will be a hard task. As pointed out by [26] when popular entities and vocabularies are distributed over multiple data sources, the performance of federated query is getting worse.

Data Freshness. The freshness is one of important measurement in the querying data because each data source might have different freshness value. Having up-to-date data catalogue is a must in the federation framework to achieve high freshness value. The inaccurate results could arise from inaccurate data catalogue. Nevertheless, updating data catalogue is a costly operation in term of query execution and traffic between data source and mediator. Apart from data catalogue being static, the freshness could not be obtained when the high network latency occurs during communication process.

Benchmark. To date, benchmarks are generally proposed for single RDF stores such as LUBM [13], BSBM [7], and SP2Bench [31]. Hence, they are not suitable for distributed infrastructure. FedBench [30] is the only benchmark proposed for federated query which evaluated the federated query infrastructure performance including loading time and querying time. Those performance metrics are lack to evaluate federation framework. The federation framework benchmark should take into account several performance measurements from traditional distributed database such as query throughput. In addition, several metrics particularly occurring in the federation framework should be considered. For instance, the size of intermediate result, number of request, amount of data sent, etc. Apart from performance metric, due to heterogeneous data in the federated query, the evaluation of data quality become important measurement namely freshness, consistency, completeness and accuracy. [21] added two more FedBench measurements, namely Endpoint Selection time and completeness. Furthermore, it evaluated performance federation framework in various environment. Since the FedBench has static dataset and query set, it is difficult task to evaluate framework for other dataset. To address this problem, SPARQL Linked Open Data Query Generator (SPLODGE) [11] generates random query set for specified dataset. The query set generation is based on predicates statistic. Beside predicate statistic, it also considers the query structure and complexity such as number of join, the query shape, etc to produce the query set.

Overlapping Terminologies. The data is generated, presented and published using numerous expressions, namespaces, vocabularies and terminologies, that significantly contain duplicate or complementary data [24, 5]. As an example, there are multiple datasets in the LSLOD describing the concept *Molecule*- in Bio2RDFs kegg dataset, it is represented using *kegg:Compound* whereas in chebi, these are identified as *chebi:Compound* and in BioPax they are denoted as *biopax-level3.owl#SmallMolecule*, i.e, using different vocabularies and terminologies to explain the similar or related concepts [15]. Moreover, different datasets contains different fractions of data or predicates about the same entities e.g: Chebi dataset contains data regarding the mass or the charge of a Molecule whereas Kegg dataset explains Molecules interaction with biological entities such as Proteins. Conceptual overlap and different datasets share data about the same entities in LD4LS can be seen in the Figure 2 . Therefore, the mapping rules among heterogeneous schemas is highly required in the federated query. This task could be done by having global schema catalogue that maps related concepts or properties and generating more links among related entities.

Provenance. Apart from the number of sources, data redundancy often occurs in the Federation SPARQL query, particularly in Federation over Linked Open Data. It is because several publisher expose the same dataset. For example, Sindice contains DBpedia data: while a user is requesting DBpedia data, the DBpedia and Sindice SPARQL Endpoints are able to answer that query. The redundant result can not be avoided by Federation Framework using third party catalogue. Hence, the data provenance is the important factor in the Federation over SPARQL Endpointa. [14] explains a notable provenance implementation in the Federation System called OPENPHACTS (`http://www.openphacts.org/`). In order to tackle provenance issue, OPENPHACTS utilizes a Nanopublication [12] which supports provenance, annotation, attribution and citation.

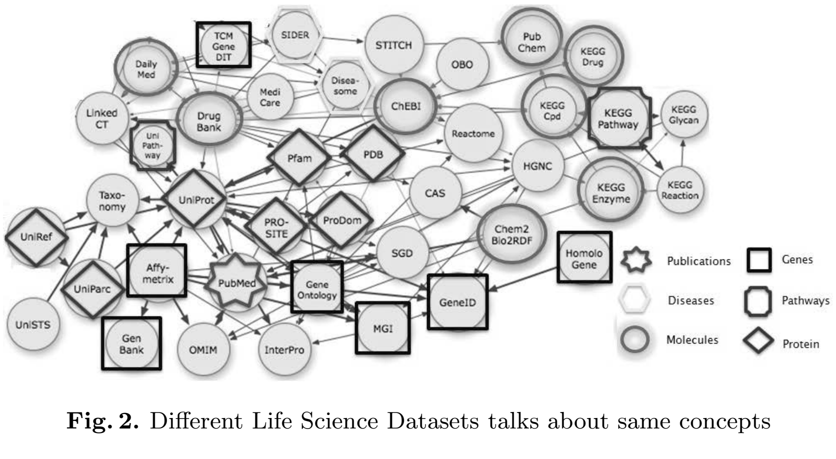

Fig. 2. Different Life Science Datasets talks about same concepts

6 Conclusion

Federation query over SPARQL Endpoints made a significant progress in the recent years. Although a number of federation frameworks have already been developed, the field is still relatively far from maturity. Based on our experience with the existing federation frameworks, the frameworks mostly focus on source selection and join optimization during query execution.

In this work, we have presented a list of federation frameworks over SPARQL Endpoints along with their features. According to this list, the user can have considerations to choose the suitable federation framework for their case. We have classified those frameworks into three categories: i) framework interprets SPARQL 1.1 query to execute federation SPARQL query covering *VALUES* and *SERVICE* operator; ii) framework handles SPARQL query without specifying SPARQL Endpoint address and has responsibility to find relevant source for a query and join the incoming results from SPARQL Endpoints; and iii) framework accepts SPARQL query without specifying SPARQL Endpoint address and translate the incoming query to SPARQL 1.1 format. Based on the current generation of federation frameworks surveyed in this paper, it still requires further improvements to make frameworks more effective in a broader range of applications. Finally, we point out challenges for future research directions.

Acknowledgement. This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

[1] Acosta, M., Vidal, M.E.: Evaluating adaptive query processing techniques for federations of sparql endpoints. In: 10th International Semantic Web Conference (ISWC) Demo Session (November 2011)

[2] Akar, Z., Hala, T.G., Ekinci, E.E., Dikenelli, O.: Querying the web of interlinked datasets using void descriptions. In: Linked Data on the Web, LDOW 2012 (2012)

[3] Antonioletti, M., Hong, N.P.C., Hume, A.C., Jackson, M., Karasavvas, K., Krause, A., Schopf, J.M., Atkinson, M.P., Dobrzelecki, B., Illingworth, M., McDonnell, N., Parsons, M., Theocharopoulous, E.: Ogsa-dai 3.0 - the whats and whys. In: UK e-Science All Hands Meeting (2007)

[4] Basca, C., Bernstein, A.: Avalanche: Putting the spirit of the web back into semantic web querying. In: The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems, SSWS 2010 (2010)

[5] Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., Couch, P., Cruickshank, D., Delderfield, M., Dunlop, I., et al.: Why linked data is not enough for scientists. Future Generation Computer Systems (2011)

[6] Betz, H., Gropengieŝer, F., Hose, K., Sattler, K.U.: Learning from the history of distributed query processing - a heretic view on linked data management. In: Proceedings of the 3rd Consuming Linked Data Workshop, COLD 2012 (2012)

[7] Bizer, C., Schultz, A.: The berlin sparql benchmark. International Journal On Semantic Web and Information Systems (2009)

[8] Deshpande, A., Ives, Z., Raman, V.: Adaptive query processing. Found. Trends Databases 1(1), 1–140 (2007)

[9] Görlitz, O., Staab, S.: Federated Data Management and Query Optimization for Linked Open Data. In: Vakali, A., Jain, L.C. (eds.) New Directions in Web Data Management 1. SCI, vol. 331, pp. 109–137. Springer, Heidelberg (2011)

[10] Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Proceedings of the 2nd International Workshop on Consuming Linked Data, Bonn, Germany (2011)

[11] Görlitz, O., Thimm, M., Staab, S.: SPLODGE: Systematic generation of SPARQL benchmark queries for linked open data. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 116–132. Springer, Heidelberg (2012)
[12] Groth, P., Gibson, A., Velterop, J.: The anatomy of a nanopublication. Inf. Serv. Use 30(1-2), 51–56 (2010)
[13] Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web 3(2-3), 158–182 (2005); selcted Papers from the International Semantic Web Conference, ISWC 2004
[14] Harland, L.: Open phacts: A semantic knowledge infrastructure for public and commercial drug discovery research. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW 2012. LNCS, vol. 7603, pp. 1–7. Springer, Heidelberg (2012)
[15] Hasnain, A., Fox, R., Decker, S., Deus, H.F.: Cataloguing and Linking Life Sciences LOD Cloud. In: 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2012), OEDW 2012 (2012)
[16] Hose, K., Schenkel, R., Theobald, M., Weikum, G.: Database foundations for scalable rdf processing. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 202–249. Springer, Heidelberg (2011)
[17] Lynden, S., Kojima, I., Matono, A., Tanimura, Y.: Adaptive integration of distributed semantic web data. In: Kikuchi, S., Sachdeva, S., Bhalla, S. (eds.) DNIS 2010. LNCS, vol. 5999, pp. 174–193. Springer, Heidelberg (2010)
[18] Ladwig, G., Tran, T.: Linked data query processing strategies. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 453–469. Springer, Heidelberg (2010)
[19] Lynden, S., Mukherjee, A., Hume, A.C., Fernandes, A.A.A., Paton, N.W., Sakellariou, R., Watson, P.: The design and implementation of ogsa-dqp: A service-based distributed query processor. Future Gener. Comput. Syst. 25(3), 224–236 (2009)
[20] Millard, I., Glaser, H., Salvadores, M., Shadbolt, N.: Consuming multiple linked data sources: Challenges and experiences. In: COLD (2010)
[21] Montoya, G., Vidal, M.E., Acosta, M.: Defender: a decomposer for queries against federations of endpoints. In: 9th Extended Semantic Web Conference (ESWC) Demo Session (Mai 2012)
[22] Montoya, G., Vidal, M.E., Acosta, M.: A heuristic-based approach for planning federated sparql queries. In: Proceedings of the 3rd Consuming Linked Data Workshop, COLD 2012 (2012)
[23] Montoya, G., Vidal, M.-E., Corcho, O., Ruckhaus, E., Buil-Aranda, C.: Benchmarking federated SPARQL query engines: Are existing testbeds enough? In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part II. LNCS, vol. 7650, pp. 313–324. Springer, Heidelberg (2012)
[24] Quackenbush, J.: Standardizing the standards. Molecular Systems Biology 2(1) (2006)
[25] Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)

[26] Rakhmawati, N.A., Hausenblas, M.: On the impact of data distribution in federated sparql queries. In: 2012 IEEE Sixth International Conference on Semantic Computing (ICSC), pp. 255–260 (September 2012)
[27] Rakhmawati, N.A., Umbrich, J., Karnstedt, M., Hasnain, A., Hausenblas, M.: Querying over federated sparql endpoints - a state of the art survey. CoRR abs/1306.1723 (2013)
[28] Schenk, S., Staab, S.: Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 585–594. ACM, New York (2008)
[29] Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011)
[30] Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A benchmark suite for federated semantic data query processing. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011)
[31] Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: Sp2bench: A sparql performance benchmark. CoRR abs/0806.4627 (2008)
[32] Schultz, A., Matteini, A., Isele, R., Mendes, P.N., Bizer, C., Becker, C.: LDIF - A Framework for Large-Scale Linked Data Integration. In: 21st International World Wide Web Conference (WWW2012), Developers Track (April 2012)
[33] Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: A federation layer for distributed query processing on linked open data. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part II. LNCS, vol. 6644, pp. 481–486. Springer, Heidelberg (2011)
[34] Schwarte, A., Haase, P., Schmidt, M., Hose, K., Schenkel, R.: An experience report of large scale federations. CoRR abs/1210.5403 (2012)
[35] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, SIGMOD 1979, pp. 23–34. ACM, New York (1979)
[36] Urhan, T., Franklin, M.J.: XJoin: A reactively-scheduled pipelined join operator. IEEE Data Engineering Bulletin 23(2), 27–33 (2000)
[37] Wang, X., Tiropanis, T., Davis, H.: Querying the web of data with graph theory-based techniques. In: Web and Internet Science (2011)
[38] Wang, X., Tiropanis, T., Davis, H.C.: Evaluating graph traversal algorithms for distributed SPARQL query optimization. In: Pan, J.Z., Chen, H., Kim, H.-G., Li, J., Wu, Z., Horrocks, I., Mizoguchi, R., Wu, Z. (eds.) JIST 2011. LNCS, vol. 7185, pp. 210–225. Springer, Heidelberg (2012)
[39] Zemánek, J., Schenk, S.: Optimizing sparql queries over disparate rdf data sources through distributed semi-joins. In: International Semantic Web Conference (Posters & Demos) (2008)