How good is your SPARQL endpoint? [⋆]

A QoS-Aware SPARQL Endpoint Monitoring and Data Source Selection Mechanism for Federated SPARQL Queries

Muhammad Intizar Ali and Alessandra Mileo

Insight Centre for Data Analytics,
National University of Ireland, Galway, Ireland
`ali.intizar@deri.org`
`alessandra.mileo@deri.org`

Abstract. Due to the decentralised and autonomous architecture of the Web of Data, data replication and local deployment of SPARQL endpoints is inevitable. Nowadays, it is common to have multiple copies of the same dataset accessible by various SPARQL endpoints, thus leading to the problem of selecting optimal data source for a user query based on data properties and requirements of the user or the application. Quality of Service (QoS) parameters can play a pivotal role for the selection of optimal data sources according to the user's requirements. QoS parameters have been widely studied in the context of web service selection. However, to the best of our knowledge, the potential of associating QoS parameters to SPARQL endpoints for optimal data source selection has not been investigated.
In this paper, we define various QoS parameters associated with the SPARQL endpoints and represent a semantic model for QoS parameters and their evaluation. We present a monitoring service for the SPARQL endpoint which automatically evaluates the QoS metrics of any given SPARQL endpoint. We demonstrate the utility of our monitoring service by implementing an extension of the SPARQL query language, which caters for user requirements based on QoS parameters and selects the optimal data source for a particular user query over federated sources.

1 Introduction

The popularity of semantic technologies is leading to a continuously growing amount of data available on the Web, referred to as Linked Data or the Web of Data. Nowadays, there exist many public SPARQL endpoints providing free access to various linked datasets, which makes it more likely that multiple data sources can qualify to answer a certain query. For example, there are 491

[⋆] This research has been partially supported by Science Foundation Ireland (SFI) under grant No. SFI/12/RC/2289 and EU FP7 CityPulse Project under grant No.603095. http://www.ict-citypulse.eu

SPARQL endpoints listed at DataHub[1], while LODstats[2] also verifies the presence of over 450 SPARQL endpoints on the Web. A simple keyword based search for *dbpedia*[3] on DataHub provides a list of 80 different datasets. The availability of such a large number of data sources and related endpoints is expected to increase the amount of information we can have access to, but it is also increasing the complexity of optimisation and execution of SPARQL queries in a federated infrastructure.

In such a scenario, selecting the optimal data source to answer a certain query is the core ingredient for efficient query processing over federated data sources. Ideally, any increase in number of data sources which can answer a certain triple pattern should not only contribute towards the completeness of the result but can also contribute to efficient federated query processing. However, most of the existing federated SPARQL engines locate relevant datasources either by using precomputed indexes [13, 7, 1] or by using brute force mechanism (SPARQL ASK) on the fly [17]. As a result, the availability of each new candidate dataset adds extra processing for the query engine, which can have serious impact on the overall performance. This gets more and more crucial in dynamic environments where timely response is key. Monitoring quality metrics of SPARQL endpoints and calculating a ranking of the SPARQL endpoints based on the evaluated quality metrics can drastically reduce the computation cost of federated queries over multiple SPARQL endpoints. Moreover any such quality metrics can facilitate users to choose the best data source based on their preferences.

Several characteristics of the SPARQL endpoints and datasets have been considered for optimal data source selection. Freshness of the data is considered in [19], where authors investigated the tradeoff between fresh and fast processing of the query answers. Recently, data replication has been considered as another important factor for the selection of data sources in federated query processing [15, 9]. However, the characteristics of the SPARQL endpoints are not limited to data freshness or replication. Several other factors like quality and licensing of the data as well as performance and availability of SPARQL endpoints etc., are also major concerns for data consumers. Hence, another important factor for selecting the optimal data source when multiple datasets qualify to answer a certain query is which source is best with respect to user's or application's requirements. For example, decision support applications are more concerned about the highest quality of the data while real-time adaptive applications desire to achieve higher performance. However, all of the above mentioned solutions leave the decision of selecting the optimal data source at the discretion of the query engine without taking application-specific requirements into account.

In this paper, we propose a QoS-aware data source selection mechanism for federated SPARQL queries, which takes user's QoS related requirements into account while selecting the optimal data source for query execution. We present a semantic data model for QoS parameters description and calculation by using

[1] A catalogue of Linked data collections. `http://datahub.io` (l.a. May, 2014.)
[2] `http://stats.lod.eu` (l.a. May, 2014.)
[3] A linked data version of the wikipedia: `www.dbpedia.org`

two popular SPARQL endpoint description vocabularies, namely (i) Vocabulary of Interlinked Datasets (VoID) [2] and, (ii) SPARQL Service Description (SD)[4]. Our main contributions in this paper can be summarised as:

- ⋆ We identify relevant QoS parameters associated with SPARQL endpoints and present a semantic model for their representation.
- ⋆ We devise a monitoring system to calculate the values of the QoS parameters.
- ⋆ We extend the SPARQL query language to enable its users to provide QoS based requirements.
- ⋆ We evaluate our system using well known benchmark and show case the reduction in the number of the selected data sources as compared to the state of the art federated query engines while achieving QoS compliance at the same time.

Structure of the paper: We lay the foundations of our study by identifying various QoS parameters associated with SPARQL endpoints and define a semantic model to represent them in Section 2. We present QoS parameters calculation model in Section 3. We demonstrate the utility of our QoS monitoring service for the problem of data source selection and present the conceptual architecture of our proposed solution in Section 4. Section 5 presents our extension of SPARQL query language to cater for QoS requirements. We evaluated our QoS-aware datasource selection algorithm by comparing its performance with the state of the art federated SPARQL query engines in Section 6. We discuss the related work in Section 7 before concluding in Section 8.

2 QoS Parameters for SPARQL Endpoints

In this section we identify various QoS parameters associated with the SPARQL endpoints and categorise them. Later in this section, we design a QoS ontology to represent QoS parameters for SPARQL endpoints.

2.1 QoS Parameters: Definition and Categorisation

QoS is defined in ISO 8042 as: "The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs"[14]. QoS parameters can play a pivotal role for optimal data source selection in the federated query processing, especially when a set of relevant SPARQL endpoints capable of answering a user's query or triple pattern have been discovered. Table 1 gives an overview of the categorisation and brief description of the QoS parameters associated with the SPARQL endpoints. We intend to give the reader an overview of the QoS parameters which can possibly contribute towards optimal data source selection, hence, this is not intended to be an exhaustive list. We briefly describe each of the QoS categories and their relevant parameters below:

[4] http://www.w3.org/TR/sparql11-service-description/

Table 1. Quality of Service Parameters for SPARQL Endpoints

QoS Category	**QoS Parameter**	**QoS Category**	**QoS Parameter**
Performance	Response Time	Data Quality	Accuracy
	Execution Time		Data Consistency
	Throughput		Completeness
	Error Rate		Freshness
Interoperability	SPARQL Version		UpTime
	Additional Features	Availability	DownTime
	Restricted Features		MeanUpTime
Dataset Description	VoID		MTTR
	Service Description	Licensing	PDDL
ResultSet	SizeLimit		ODC-By
	ResultFormat		ODC-ODbL
			CC0 1.0 Universal

Performance: Performance is the ability to generate the desired outcomes by utilising the appropriate amount of resources and time. Performance of the SPARQL endpoints is determined by accumulative values of the various performance parameters including (i) *response time:* the amount of time required for a SPARQL endpoint to respond a query once received, (ii) *execution time:* amount of time required to execute a certain query over a dataset, execution time is calculated as the amount of time required between the query has been posted by the user and the results are being produced, (iii) *throughput:* number of queries per second a SPARQL endpoint can handle, and (iv) *error rate:* rate at which a certain SPARQL endpoint returns some exceptions or error messages in response to the user queries.

Data Quality: Quality of the data provided by SPARQL endpoints can be judged by evaluating certain data oriented features, like (i) *accuracy:* the degree to which delivered information is correct, (ii) *data consistency:* the degree to which datasets are free from contradictions, (iii) *completeness:* represents that the result set provided by SPARQL endpoint is of sufficient size or number of triples returned for the particular query, and (iv) *data freshness:* refers to the age of the data. It is the degree to which extent data provided by the endpoint is up-to-date.

Interoperability Interoperability among SPARQL endpoints is their ability to answer certain query language specific features. Most of the public SPARQL endpoints has capacity to process SPARQL 1.0[5], while there are some early adaptors of the SPARQL 1.1[6] as well [5]. *SPARQL version:* reflects that which version of the SPARQL query language is supported by the SPARQL endpoint. *Restricted features:* provide a list of restricted features of a query language not supported by the SPARQL endpoint. It is common practice to restrict some features of the underlying query language to avoid work load or network traffic e.g. SPARQL ASK queries. *Additional features:*provides

[5] `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`
[6] `http://www.w3.org/TR/sparql11-query/`

a list of the additional features supported by any SPARQL endpoints as a result of the extension of the query language.

Availability: Ability of the SPARQL endpoint to answer an authorised user's query at any given time [14]. Availability is very critical in the distributed and autonomous architecture where users or applications lack the control over the underlying query engines. Availability related QoS parameters of SPARQL endpoint are, (i)*uptime:* denotes the date and time when the SPARQL endpoint was made available for access, (ii)*downtime:* refers to the time and date since when a SPARQL endpoint is offline, (iii)*mean up time:* is the average amount of time a SPARQL endpoint remains available for authorised users within a certain time frame, and (iv)*mean time to recover (MTTR):* is the average amount of time a SPARQL endpoint takes to recover if it goes offline due to some technical or operational reasons.

Dataset Description: Ability of the SPARQL endpoints to provide meta information about the datasets hosted using data description vocabularies. Data description vocabularies are also used by application and query engines to automatically discover relevant datasets. Two popular methods for SAPRQL endpoint dataset description are VoID and SD.

ResultSet: Data oriented nature of the SPARQL endpoints brings some additional QoS parameters relevant to the result set generated by the endpoint while giving answer to a certain query. *Size limit* is mostly imposed by SPARQL endpoints to limit the maximum number of triples which can be returned by a SPARQL endpoint while answering a certain query. Limit is imposed to reduce the network traffic caused by queries having reasonably larger result size. Another important factor of user's concern about the result set generated by SPARQL endpoint is the format of the results.

Licensing: Licensing defines the liabilities, legal requirements and usage permission for the data sets. Several licensing agreements are in practice, VoID describes four popular data licensing agreements used by SPARQL endpoints. We also limit ourselves to these four types of licenses which are, (i) Public Domain Dedication and License (PDDL), (ii) Open Data Commons Attribution (ODC-By), (iii) Open Database License (ODC-ODbL), and (iv) Creative Commons Public Domain Waiver (CC0 1.0 Universal).

2.2 QoS Parameters: Semantic Description Model

Inspired by the QoS ontology for Web services described in [20], we defined a QoS ontology for quality related parameters of SPARQL endpoints. We partially reuse existing vocabularies such as VoID, SD and QoS Ontology [20, 8]. We define a *QoS Profile* class which can be attached to the *void:dataset*[7] or *sd:service*[8] class using *qs:hasQoSProfile*[9]. Each QoS parameter belongs to a specific category, which is a subclass of QoS. We use some existing properties of VoID and

[7] prefix void: `<http://rdfs.org/ns/void#>`
[8] prefix sd: `<http://www.w3.org/ns/sparql-service-description#>`
[9] prefix qs: `<http://www.deri.org/QOS#>`

SD for the QoS parameters. For example, QoS parameter *ResultFormat* as defined previously, can be easily described by using *sd:resultFormat* property of SD. Similarly, we use *dcterms:license* from VoID to describe QoS parameter *Licesning.* We use *Measurement Units Ontology* [10] to measure the values of the QoS parameters. We limited ourselves to the domain independent generic QoS parameters, however any additional domain specific or domain independent QoS parameters can be easily augmented. It is also worth mentioning that we only provided the definition and categorisation of the subset of the QoS parameters while defining and categorising a complete set of all possible QoS parameters is out of the scope of this paper.

Definition 1. *Let* $P_{qos} = \{qos_i \ |i = 1..s\}$ *be the QoS profile of a SPARQL endpoint, representing the set of all QoS parameters associated to that endpoint. We define* P^U_{qos} *as the set of user's defined QoS attributes and* P^D_{qos} *as the set of default QoS parameters indicated by the system, such that* $P^U_{qos} \cup P^D_{qos} \subseteq P_{qos}$.

Furthermore, we define $PQ_{qos} = \{qos_k \mid k = 1..t\}$ *as a QoS profile for a query* Q *such that* $PQ_{qos} = P^U_{qos} \cup P^D_{qos}$ *and* $PQ_{qos} \subseteq P_{qos}$.

Most of time, a user doesn't want to specify (or is not aware of) all of the QoS parameters associated to a SPARQL endpoint, therefore we introduce the concept of the P^D_{qos} which describes some critical QoS parameters and their minimum threshold expected from any SPARQL endpoint listed by the federated engine. We assume this is set by the federated query engine in order to ensure that a reasonable QoS standard is maintained even without any explicit requirements, i.e. when $P^U_{qos} = \phi$. In this case, $PQ_{qos} = P^D_{qos}$. Listing 1 shows a sample QoS description of a SPARQL endpoint using our QoS semantic model.

Listing 1. A Sample QoS Profile of a SPARQL Endpoint

```
<http://www.deri-testbed/QoS/endpoint1> a  sd:Service;
sd:endpointUrl <http://deri.org/QoS/sparql>;
sd:supportedLanguage sd:SPARQL11Query;
sd:resultsFormat <http://www.w3.org/ns/formats/RDF_XML> ;
dcterms:license <http://www.opendatacommons.org/licenses/pddl/> ;

qs:hasQoSProfile qs:QoSProfileEndpoint120140514T105940 .

qs:QoSProfileEndpoint120140514T105940
qs:isGeneratedAt "2014-05-14T10:59:40Z" ;
qs:contains qs:QoSProfileEndpointParameter20140514T105940Completeness.

qs:QoSProfileEndpointParameter20140514T105940Completeness
qs:hasName "Completeness"  ;
qs:hasTendency "increase" ;
qs:isMeasuredIn muo:UnitOfMeasurement ;
qs:metricType qs:NumericMetric ;
qs:hasValue "70" .
```

[10] prefix muo: `http://purl.oclc.org/NET/muo/muo#UnitOfMeasurement`

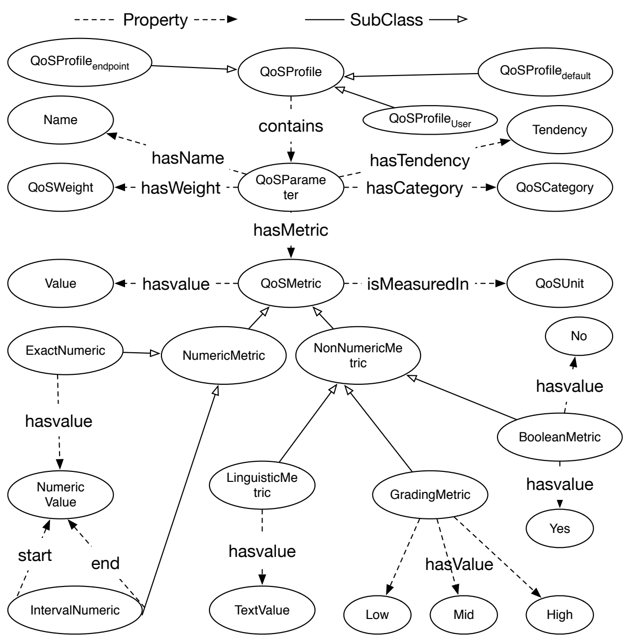

Fig. 1. A QoS Parameters Evaluation Metrics Ontology [8]

3 Evaluation of QoS Parameters

In this section, we present a QoS parameters calculation ontology by defining QoS metrics and their calculation method.

3.1 QoS Prameters: Evaluation Metrics Semantic Model

We present an extensible semantic model for the QoS parameters evaluation of any corresponding SPARQL endpoint. Our presented semantic data model can be easily extended to cater for new QoS parameters or categories and their evaluation mechanism. QoS ontology for SPARQL endpoints evaluation metrics is designed by closely examining the VoID and SD as well as existing QoS specifications for web services [20]. Figure 1 depicts fundamental concepts of QoS parameters and their evaluation metrics. Top level concept is *QoSProfile* as defined in the previous section which contains three subclasses namely, (i) $QoSProfile_{User}$ reflecting user's defined QoS requirements, (ii) $QoSProfile_{default}$ represents default values of the QoS requirements configured within the system, and (iii) $QoSProfile_{endpoint}$ is the QoS summary of a SPARQL endpoint.

QoS Parameter indicates the quality attribute of the *QoS Profile* while *QoS Metric* defines the measurement of the *QoS Parameter. QoS Metric* can be either, (i) *Numeric:* where value of a QoS parameter can be measured as an exact numeric value or an interval between two numeric values , or (ii) *NonNumeric:* whose value can be boolean (true, false), grades (high, middle, low), or in linguistics as any text literal e.g. SPARQL1.1 is a linguistic value for the *SPARQL Version* parameter. *QoS Tendency* indicates the effects of the change in the value of QoS parameter on the aggregated QoS. Tendency can be mentioned as either increase or decrease, where increase denotes that any increase in the QoS parameter value is directly proportional to the QoS value (e.g. increase in the accuracy level results into higher data quality), while decrease denotes that any increase in the QoS parameter value is inversely proportional to the QoS value (e.g. increase in execution time results into lower performance). Tendency is mostly relevant for the QoS parameters containing numeric values.

3.2 QoS Parameters: Monitoring & Value Calculation

Measurement of the value of the QoS is a challenging task because of multidimensional nature of the QoS parameters and their tendency. We used simple evaluation method by executing monitoring queries with the mix of both active approach (continuous monitoring of SPARQL endpoints by periodic sample and testing) and passive approach (runtime QoS evaluation at the query arrival). Below we discuss methods we used to calculate the value of the each of the QoS parameter discussed in Section 2.

Performance: Response time of a SPARQL endpoint can be measured by executing simple ASK queries. However, to avoid network congestion and malicious attacks many SPARQL endpoints restrict ASK queries. We use a below mentioned simple SPARQL SELCT query where $< s >$ and $< o >$ are randomly generated URI to make sure that query result set is empty.

```
Q1.    SELECT ?p where { <s>  ?p  <o> }
```

However to avoid the influence of any indexes maintained by SPARQL endpoint over the performance evaluation, we used some additional *Subject-Subject*, *Object-Object* and *Subject-Object* join queries as described in [6]. Q2 is a sample *Object-Object* join query.

```
Q2.    SELECT ?o where { s1  p1  ?o
                         s2  p2 ?o }
```

We estimated query execution time by calculating the time required to get the answers of the Q1, Q2 and Q3 respectively and subtracted their response time. Q2 can also be further extended to formulate complex queries while covering more Basic Graph Patterns (BGPs). We executed the query with varying result size by imposing LIMIT of 10, 100 and 1000 respectively.

```
Q3.    SELECT * where { ?s ?p  ?o } LIMIT 1000
```

Throughput was observed by repeatedly sending Q1 and error rate was measured by counting the number of exceptions returned by the SPARQL endpoint while answering all executed queries.

Data Quality: Measurement of data quality is not an easy task because multiple factors can contribute to the data quality. We use only simplified basic methods to calculate completeness and freshness of the data. Data completeness of a SPARQL endpoint is checked by comparing the result size (number of triples) generated by a SPARQL endpoint over multiple executions. We used VoID property *dcterms:modified* to check the freshness of data. We have to rely on *dcterms:created* in cases where *dcterms:modified* was non-existent. We left the application of any sophisticated methods for the evaluation of accuracy and consistency of the result set and to find out any incorrectness, inconsistency and/or contradictions in the data as part of our future work.

Interoperability: *SPARQL Version* parameter indicates which version of the language is supported by the endpoint. We used SPARQL1.1 test data set[11], which provides complete set of tests to check the compliance of the SAPRQL endpoint with SPARQL 1.1. Similarly, restricted features can be listed by evaluating the same test queries to evaluate supported features of the language. We rely on the endpoint service provider for the provision of the initial list of features if any additional features are supported, however additional features can be validated using the same approach as used for restricted features evaluation.

Availability: Initial value for the *UpTime* of a SPARQL endpoint is obtained through *dcterms:date* property of the VoID and can be periodically monitored by executing Q1. The value of *DownTime* is achieved by continuously monitoring the SPARQL endpoint with Q1 till any non-response or exception in the execution of Q1 is reported. *Mean UpTime* and *MTTR* is calculated after analysing the historical data of observations of *UpTime* and *DownTime*. We monitored our testbed endpoints only for a period of 90 days, however Linked Open Data Analytics initiatives like SPARQLES can play an important role for the evaluation of these metrics [5].

Dataset Description: SPARQL endpoints are mostly described by VoID or SD. Service description is retrieved by dereferencing the endpoint URI, while VoID description is obtained by executing Q4.

```
Q4.     PREFIX void: <http://rdfs.org/ns/void#>
        SELECT ?ds WHERE {
                ?ds a void:Dataset .
                ?ds void:SPARQLEndpoint <SPRQLEnpointURI> }
```

ResultSet: Most of the SPARQL endpoints limit the size of result set to reduce the network traffic. Variation of Q3 with varying limit size is executed to monitor if there is any limit imposed on the size of the result set. A query over SD using Q5 can provide a list of supported formats by any SPARQL endpoint where <endpointURI> corresponds to the URI of the relevant SPARQL endpoint.

[11] http://www.w3.org/2009/sparql/docs/tests/data-sparql11/

```
Q5.
PREFIX sd: <http://www.w3.org/ns/sparql-service-description#>
SELECT ?format WHERE {
                ?s a sd:service .
                ?s sd:endpoint <endpointURI> .
                ?s sd:resultFormat ?format . }
```

Licensing: VoID provides four basic licensing modes for the licensing of the datasets. Q6 returns the particular license type supported by the SPARQL endpoint.

```
Q6.     PREFIX dcterms: <http://purl.org/dc/terms/>
        SELECT ?license WHERE {
                ?ds a void:Dataset .
                ?ds dcterms:license ?license .
                }
```

4 QoS-Aware Data Source Selection

In this section, we define the problem of QoS-aware data source selection and showcase the utility of our QoS monitoring service by presenting the conceptual architecture of our proposed solution.

4.1 Problem Statement

Data source selection is the process of determining which candidate data source will be selected if there are multiple sources available to answer a certain query. The general problem of data source selection is as in Definition 2.

Definition 2. *Given a user's query Q and set of n data sources $DS = \{ds_i \mid i = 1..n\}$, we define $DS_c = \{dsc_j \mid j = 1..m\}$ as a set of candidate datasources that can potentially contribute to answer query Q, where $DS_c \subseteq DS$ and $1 \leq m \leq n$.*

QoS-Aware data source selection as described in Definition 3 is the process of determining which candidate data sources will contribute to answer a certain query Q in such a way to be compliant with the QoS requirements specified by the user or the application.

Definition 3. *Given a user's query Q and set of candidate data sources DS_c, we define $DS_{qos} = \{\ dsqos_k \mid k = 1..l\}$ as the set of optimal data sources that can potentially contribute to the answer query Q and are compliant with the QoS requirements mentioned in the query, where $DS_{qos} \subseteq DS_c$ and $1 \leq l \leq m \leq n$.*

4.2 Conceptual Architecture

Figure 2 depicts the conceptual architecture of our proposed QoS-aware federated SPARQL query processing system. A user's query with QoS requirements is processed by our extended *QoS-aware Query parser* which generates QoS profile of the query and sends it (without QoS requirements) to the federated query engine. We left at the discretion of the federated query engine to find out multiple candidate sources containing similar data for each triple pattern. Mostly

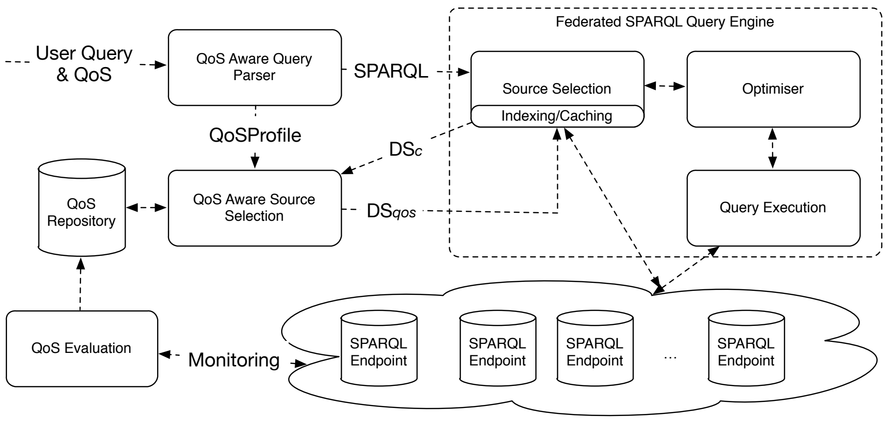

Fig. 2. QoS-aware Federated SPARQL Query Processing

existing federated query engines calculate triple pattern wise candidate sources either using precomputed indexes or evaluating it at run time by sending ASK queries. The *QoS-Aware Source Selection* component receives triple pattern wise candidate data sources (DS_c) from the source selection component of the federated query engine and returns a reduced number of data sources (DS_{qos}) after evaluating their QoS compliance with QoS requirements specified in the query. QoS compliance is verified using the QoS repository generated by the *QoS Evaluation* component which continuously monitors QoS parameters of all SPARQL endpoints accessible by the federated query engine.

5 QoS Aware SPARQL Query Extension

In this section we introduce our QoS-aware extension of the SPARQL query language and describe our algorithm for optimal QoS-aware data source selection.

5.1 Syntax and Semantics

Our aim for the extension of the SPARQL query language is to enhance its ability to cater for user requirements in terms of QoS parameters. We introduced a new keyword *QOSREQ* by extending the grammar of SPARQL 1.1 extension for federated queries. The set of query related QoS requirements qos_k as per Definition 1, are attached to each *triple pattern.* User can apply *QOSREQ* operator either at BGP level and/or triple pattern level. In the former case, if a user defines QoS constraints at Basic Graph Pattern (*BGP*) level, all the QoS constraints are equally assigned to each of the triple patterns within the *BGP* accordingly. However, in the later case, if user describes QoS requirements for few of the individual triple patterns, remaining triple patterns will be assigned default values of the QoS requirements using default QoS profile. In cases where QoS requirements are defined for both BGP and triple pattern, QoS requirements defined at triple pattern level will supersede the QoS requirements defined at BGP level. *QOSREQ* operator attaches the QoS requirements to immediately preceding triple pattern or BGP .

Definition 4. *Let us consider the set of query-related QoS parameters PQ_{qos} as per Definition 1. We define set of Qos requirements*

$$R_{qos} = \{r_k = \langle qos_k \ op \ val_k \rangle \mid qos_k \in PQ_{qos}, \ op \in \{<, >, =\}\}$$

where val_k is the value constrained by operator op to qos_k.

Definition 5. *Let us consider the set of QoS parameters P_{qos} as per Definition 1, related to a datasource $dsc_j \in DS_c$. We define the set of QoS configuration for a candidate datasource as*

$$C_{qos}(dsc_j) = \{c_i(dsc_j) = \langle qos_i \ setval_i \rangle \mid qos_i \in P_{qos}, \ dsc_j \in DS_c\}$$

where $setval_i$ is the value of qos_i for the candidate datasource dsc_j.

Listing 2. A Sample QoS-Aware SPARQL Query

```
  SELECT ?drug ?keggUrl ?chebiImage
  WHERE {
  ?drug rdf:type drugbank:drugs .
  QOSREQ[qs:Completeness = 80, qs:SizeLimit > 10000]

  ?drug drugbank:keggCompoundId ?keggDrug .
  ?keggDrug bio2rdf:url ?keggUrl .

{ ?drug drugbank:genericName ?drugBankName .
  ?chebiDrug purl:title ?drugBankName . }
  QOSREQ[qs:DatasetDescription = 'VoID',
  qs:ResponseTime < 30 ]

  ?chebiDrug chebi:image ?chebiImage . }
```

Note that in the current implementation we consider all set of operators (e.g. $op \in \{<, >, =\}$) and their combinations (e.g. $\leq$, $\geq$) for qos_k with numeric and grading values, while for qos_k ranging over string values, only equality is considered as a valid operator ($op \in \{=\}$), e.g. $\langle license = \text{“}PDDL\text{”}\rangle$.

If a user query explicitly defines the endpoint using the *SERVICE* keyword, this is given priority over the QoS constraints (which are all discarded) and the SPARQL 1.1 query is directly fed into the federated query engine supporting the language. We do not show extension of the SPARQL 1.1 grammar rules for the introduction of *QOSREQ* operator due to space limitation. Listing 2 shows a sample QoS-aware SPARQL query over federated sources. Prefixes are ignored due to space limitations.

5.2 Source Selection

Algorithm 1 shows the QoS-aware optimal data source selection process. Input for the algorithm are (i) the user query with explicit definition of the requirements $r_k \in R_{qos}$ attached to the query or to a triple pattern in the query and (ii) the set of candidates data sources $DS_c(tp_i)$ for each triple pattern tp_i. The output of the algorithm is a reduced subset DS_{qos} of the candidate services DS_c which comply with the QoS requirements in R_{qos}. In case multiple sources can fulfil the set of QoS requirements for the query or for a BGP, we let the underlying federated query engine to make the final decision, however it can also be prioritise on the overall score of the QoS parameters or reputation of the SPARQL endpoint. Similarly, if none of the data sources fulfil the QoS criteria specified in the query, we let federated query engine to opt for either returning empty result set or compromising over QoS criteria. We limit ourselves to the individual values of the QoS parameters, we refer our readers to the extensive literature on methods of calculating the overall QoS score of a service by aggregation of all the QoS parameters while considering their weight-age, tendency and preferences [8, 11]. Let's identify the triple patterns in the query as $TP = \{tp_1,\ tp_2, \ldots,\ tp_t\}$ and parametrise the set of requirements R_{qos} with the pattern they refer to as $R_{qos}(tp_t) = \{r_k(tp_t) \mid r_k \in R_{qos},\ tp_t \in TP\}$. We do the same for the candidate sources DS_c and for the optimal sources DS_{qos} obtaining $DS_c(tp_t) = \{dsc_i(tp_t) \mid dsc_i \in DS_c,\ tp_t \in TP\}$ and $DS_{qos}(tp_t) = \{dsqos_j(tp_t) \mid dsqos_j \in DS_{qos},\ tp_t \in TP\}$. The process is iterated for each tp_t in the query as shown in Algorithm 1.

```
input  : $R_{qos}(tp_t)$ , $DS_c(tp_t)$
output: $DS_{qos}(tp_t)$

foreach $tp_t \in TP$ do
    $DS_{qos}(tp_t)=\emptyset$
    foreach $dsc_i(tp_t) \in DS_c(tp_t)$ do
        flag = true;
        foreach $r_k(tp_t) = \langle qos_k\ op\ val_k \rangle \in R_{qos}(tp_t)$ do
            if $setval_k\ op\ val_k\ when\ c_k(dsc_i) = \langle qos_k\ setval_k \rangle$ then
                do nothing;
                else
                    flag = false;
                end
            end
        end
        if $flag == true$ then
            $DS_{qos}(tp_t) \leftarrow DS_{qos}(tp_t) \cup dsc_i(tp_t)$
        end
    end
    return $DS_{qos}(tp_t)$
end
```

Algorithm 1: QoS-Aware Data Source Selection Algorithm

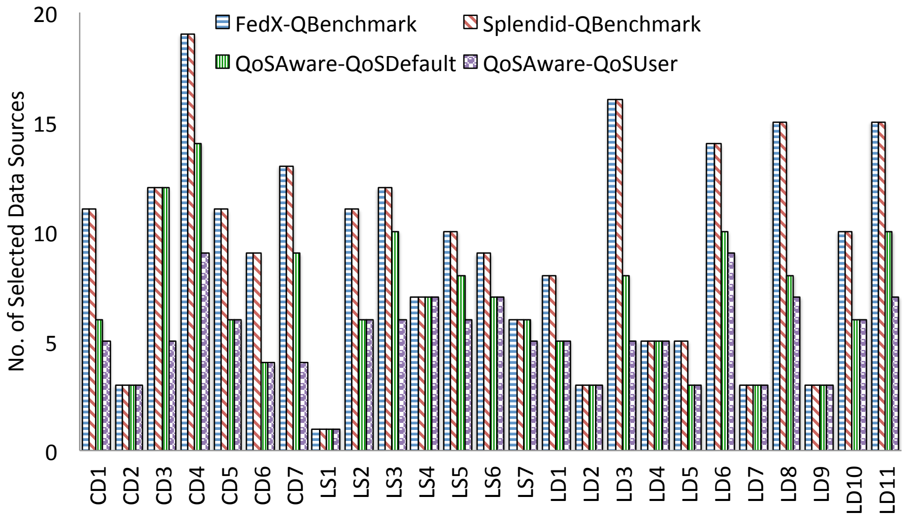

Fig. 3. Comparison of the Number of Data Sources Selected (Federated Engines)

6 Experimental Evaluation

Implementation & Testbed: We extended SPARQL 1.1. grammar to introduce *QOSREQ* operator. Implementation was carried out using Java 1.7 and antlr[12] was used for java code generation from SPARQL grammar rules. We used Fedbench [16] benchmark for the simulation of the federated queries and deployed the benchmark datasets over 9 SPARQL endpoints[13]. All the endpoints were deployed over Virtuoso[14]. We created one replica of each dataset and hosted it on different endpoint with varying configuration. This variation helped us for the generation of different QoS parameter's values of two SPARQL endpoints hosting same datasets. We opted for a controlled environment for the initial evaluation because of the availability of the evaluation results of the existing federated query engines using FedBench dataset and queries. However, our monitoring service can be easily deployed over public SPARQL endpoints.

Monitoring QoS Parameters: QoS parameters summaries were maintained and indexed for all deployed SPARQL endpoints. We used QoS parameters defined in Table 1 as an initial set of QoS parameters. We monitored each of the

[12] `http://www.antlr.org`

[13] Authors are thankful to Muhammad Saleem, AKSW, University of Leipzig for the provision of FedBench datasets. Detailed description of the SPARQL endpoints can be found at `http://deri-srvgal29.nuig.ie:8080/QoS`

[14] `http://virtuoso.openlinksw.com`

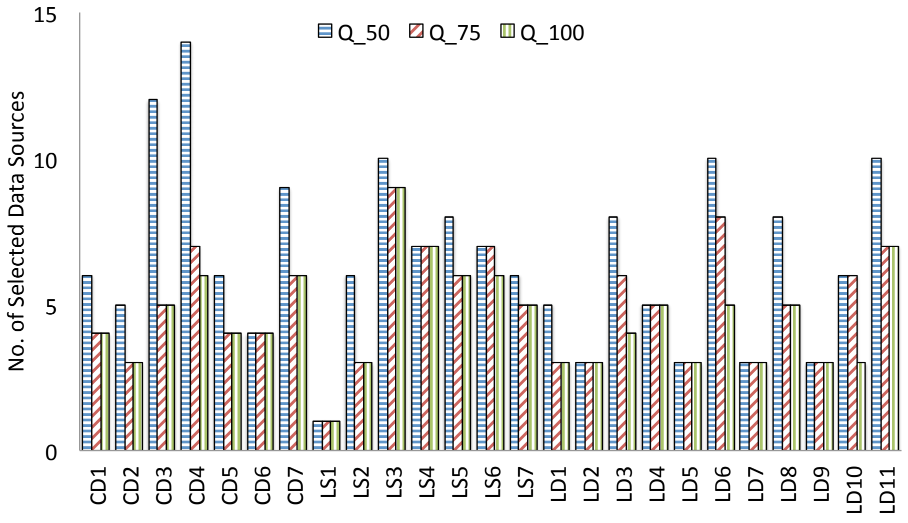

Fig. 4. Comparison of the Number of Data Sources Selected (QoS Variations)

SPARQL endpoint over a period of 90 days to observe any changes in QoS parameters values and updated them accordingly. We maintained a QoS repository of all the SPARQL endpoints of the testbed[15]. Regular monitoring was set at every 24 hours with equal distribution between 12:00 a.m and p.m. to reduce the effect of network traffic during busy hours. Predictably, there were not so much frequency of changes in the values of the QoS parameters because experimental evaluation was conducted in a controlled local network. In order to increase the frequency of the fluctuation in the QoS parameters values some technical hazards and endpoint data policies were changed with human interaction at random intervals e.g. stopping virtuoso server to reflect unavailability or updates in one dataset while keeping the replica outdated. We opted for the in-memory storage of QoS parameters repository due to its smaller size.

Evaluation Results: We ran 25 queries provided with Fedbench over the testbed. We counted total number of triple pattern wise selected sources for each query. We used SPLENDID (*index based*)[7] and FedX(*index free*)[17] query engines for the execution of the federated queries and demonstrate compatibility of our extension with both types of SPARQL engines. We ran three versions of the each of the benchmark query with (i)$Q_{Benchmark}$: containing Fedbench queries without any modification, (ii) $QoS_{default}$:values of the QoS parameters defined within QoS default profile were attached to the benchmark queries, and (iii) QoS_{User}: we assigned QoS requirements randomly to each of the benchmark

[15] A sample QoS repository is available at: `http://deri-srvgal29.nuig.ie:8080/QoS`

query [16]. We ran 10 execution of each query and counted the triple pattern wise average total number of data sources selected for each query. Figure 3 shows the decrease in the number of the *tp-wise* selected sources while executing *QoS-Aware* queries over Fedbench as compared to the number of data sources selected by the existing federated query engines. $Q_{Benchmark}$ queries were executed using FedX and SPLENDID, while $QoS_{default}$ and QoS_{User} were executed using our QoS-Aware source selection algorithm. QoS Aware source selection algorithm considers candidate data sources returned by SPLENDID. If QoS constraints are defined, our system compares QoS profile of the query with the QoS profile of the relevant data sources, and returns the reduced number of the data sources. We verified 100% compliance of the QoS requirements for each of the selected data source by our system while executing $QoS_{default}$ and QoS_{User}. In order to observe the effect of increase and decrease in the threshold of the QoS parameters, we devised three versions of the each of benchmark query with different threshold levels, (i) Q_{50} , (ii) Q_{75}, and (iii) Q_{100}, where values of the QoS parameters were set at <50%, between 50 -75% and over 75% of the best possible value for a parameter respectively, while taking tendency of the QoS parameters into account. Three possible values for the QoS *GradingMetrics* , LOW, MED, HIGH were assigned within Q_{50}, Q_{75} and Q_{100} respectively. It was interesting to observe that most of the QoS parameters fall into the category of either Q_{50} or Q_{100} mostly because of the boolean nature of the majority of the selected QoS parameters. Figure 4 shows the decrease in the number of selected data sources for Q_{50} and Q_{75} as compared to Q_{100}.

7 Related Work

Federated query processing over distributed SPARQL endpoints have been widely studied and different approaches to process federated queries and their optimisation have been proposed [17, 7, 1, 13]. Federated SPARQL engines can be classified into two categories (i) *Index based*, and (ii) *Index free*. Independently of the use of an index, in both approaches an initial list of all SPARQL endpoints URI has to be provided to initialise the query engine [17]. This might result in limited flexibility to cater for new sources, but also produce an increased number of potential sources providing the required data. Recently, data replication has been considered to reduce the number of potential data sources which can answer a user's query [9, 15]. Data source selection based on the tradeoff between data freshness and fast query processing is presented in [19]. However, broader categories of the QoS parameters associated with SPARQL endpoints for more effective data and application driven source selection has not been explored yet.

QoS parameters have been extensively studied in the context of Web services [20]. Initial work is focused on identification and definition of QoS parameters, while later work was built on critical aspects of relevant web service selection especially when a set of relevant sources is discovered [10]. A QoS ontology

[16] Detailed description of the QoS Repository, $Q_{Benchmark}$, $QoS_{default}$, QoS_{User}, Q_{50}, Q_{75}, and Q_{100} are accessible at: `http://deri-srvgal29.nuig.ie:8080/QoS/`

based service selection model is presented in [20]. However, nature of SPARQL endpoints is more inclined towards data services rather than the traditional Web services paradigm. In [18, 3, 4], authors advocated that data services are more concerned about quality, ownership and usage permission of the data which makes them different from the traditional Web services. QoS parameters for SPARQL endpoints while considering the data oriented nature of the SPARQL endpoints have not been realised so far.

User QoS related requirements have been traditionally studied in many fields of science including databases [12]. Two prevalent approaches to cater for user's QoS requirements in the databases are (i) *query extenstion:* existing query languages are extended to express user's QoS requirements within query language, or (ii) *QoS-aware query language:* a new query language is proposed to express user QoS requirements. Compatibility and non-adaptability are two major concerns of the later approach, therefore we opted for the former approach and propose an extension of the SPARQL query language which enables expressivity of the QoS requirements of a particular user within the SPARQL query language.

8 Conclusion and Future Directions

In this paper, we presented a QoS-aware SPARQL endpoint monitoring and data source selection mechanism for SPARQL queries over federated endpoints. We identified various QoS parameters of SPARQL endpoints and presented semantic model to represent and evaluate these parameters. We calculated values of the QoS parameters by continuous monitoring and generating QoS profiles of the SPARQL endpoints. We extended SPARQL query language to enable users to define their QoS requirements. We showcase the effectiveness of our QoS monitoring service by using the data source selection problem of the federated SPARQL queries. Our evaluation results show the reduction in the number of selected data sources as compared to state of the art federated query engines, while achieving QoS compliance according to the user's specified requirements.

In future work, we plan to further explore broader range of QoS parameters and generate a single aggregated QoS value for each SPARQL endpoint particularly for the public SPARQL endpoints. We also plan to integrate user feedback for the fair evaluation of the QoS parameters. Application of more sophisticated methods for the evaluation of Quality of Information (QoI) parameters of the datasets accessible via SPARQL endpoints is also part of our future agenda.

In the context of smart cities applications, we plan to extend our QoS-aware federated query for linked stream data sources, enabling users and application to adaptively select optimal data source based on the changing requirements, preferences and context.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In *Proc. of ISWC 2011*, pages 18–34. 2011.

2. K. Alexander and M. Hausenblas. Describing linked datasets, the vocabulary of interlinked datasets. In *Proc. of LDOW, WWW , 2009*, 2009.
3. M. I. Ali, R. Pichler, H. L. Truong, and S. Dustdar. Data concern aware querying for the integration of data services. In *Proc. of ICEIS (1)*, pages 111–119, 2011.
4. M. I. Ali, R. Pichler, H.-L. Truong, and S. Dustdar. Incorporating data concerns into query languages for data services. In *Proc. of Enterprise Information Systems*, pages 132–145. 2012.
5. C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *Proc. of ISWC (2)*, pages 277–293, 2013.
6. M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world sparql queries. In *In Proc. of USEWOD 2011, at WWW*, 2011.
7. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. *In Proc. of COLD*, 782, 2011.
8. G. Guo, F. Yu, Z. Chen, and D. Xie. A method for semantic web service selection based on qos ontology. *Journal of Computers*, 6(2), 2011.
9. K. Hose and R. Schenkel. Towards benefit-based rdf source selection for sparql queries. In *Proc. of SWIM*, page 2, 2012.
10. A. F. Huang, C.-W. Lan, and S. J. Yang. An optimal qos-based web service selection scheme. *Journal of Information Sciences*, 179(19):3309–3322, 2009.
11. K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benrernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro. A survey on service quality description. *ACM Computing Surveys (CSUR)*, 46(1):1, 2013.
12. D. Mobedpour, C. Ding, and C.-H. Chi. A qos query language for user-centric web service selection. In *Proc. of SCC, 2010*, pages 273–280. IEEE, 2010.
13. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *Proc. of The Semantic Web: Research and Applications*, pages 524–538. Springer, 2008.
14. I. Rafique, P. Lew, M. Q. Abbasi, and Z. Li. Information quality evaluation framework: Extending iso 25012 data quality model. In *Proc. of World Academy of Science, Engineering and Technology*, number 65, 2012.
15. M. Saleem, A.-C. N. Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *Proc. of ISWC*, pages 574–590. Springer, 2013.
16. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *Proc. of ISWC*, pages 585–600. Springer, 2011.
17. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *Proc. of ISWC*, pages 601–616. Springer, 2011.
18. H. L. Truong and S. Dustdar. On evaluating and publishing data concerns for data as a service. In *Proc. of APSCC*, pages 363–370, 2010.
19. J. Umbrich, M. Karnstedt, A. Hogan, and J. X. Parreira. Hybrid sparql queries: fresh vs. fast results. In *Proc. of ISWC*, pages 608–624. Springer, 2012.
20. X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A qos-aware selection model for semantic web services. In *Proc. of ICSOC*, pages 390–401. Springer, 2006.