SPARQL with XQuery-based Filtering[⋆]

Takahiro Komamizu

Nagoya University, Japan
taka-coma@acm.org

Abstract. Linked Open Data (LOD) has been proliferated over various domains, however, there are still lots of open data in various format other than RDF. Document-centric XML data are such open data that are connected with entities in LOD as supplemental documents for these entities. To utilize document-centric XML data linked from entities in LOD, in this paper, a SPARQL-based seamless access method on RDF and XML data is proposed. In particular, an extension to SPARQL, `XQueryFILTER`, which enables XQuery as a filter in SPARQL is proposed. For efficient query processing of the combination of SPARQL and XQuery, a query optimization is proposed. Experimental scenarios using real-world data showcase the effectiveness of `XQueryFILTER` and optimization efficiency.

Keywords: SPARQL Extension · XQuery Filtering · Query Processing

1 Introduction

Linked Open Data (LOD) [4] started by Sir Tim Berners-Lee is a bunch of factual records represented by RDF (Resource Description Framework). Although LOD has been proliferated its population over various domains, there are still lots of data which suffer from converting into LOD. In particular, documents such as manuals and minute books are in this line. There are sophisticated techniques converting document structure into RDF (e.g., information extraction [5]), however they still require human efforts on error-pruning.

In this paper, the way to utilize document open data which are linked from entities in LOD is explored. To this end, in this paper, a SPARQL-based seamless access method, namely a dedicated filtering expression, `XQueryFILTER`, is proposed. This expression enables XQuery-based filtering by accessing underlying XML data while processing a SPARQL query. To realize efficient `XQueryFILTER` implementation, database theory-based query optimization is applied.

2 Related Work

There are few works on combining queries for XML (like XPath, XSLT and XQuery) into SPARQL query except [6]. Droop et al. [6] have proposed a translation-based XPath embedding for SPARQL in order to enable XPath processing in

[⋆] Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

SPARQL query processor. To this end, they have proposed translation model from XML to RDF and from XPath to SPARQL. There are three major differences from their work to the proposed filtering in this paper. One is query type for XML, namely, XQuery and XPath. In general, XQuery is more expressive than XPath. Another difference is that, in the proposed approach, no preprocessing is applied to XML data, while [6] requires translation into RDF. The other difference is that, the proposed approach fully utilizes the dedicated query processing technique in XML DB, but [6] translates an XPath instance into a complicated SPARQL query and processes it on a SPARQL query processor.

The related context is to query XML and RDF which can be translated each other. So-called data-centric XML data [8] are designed for representing objects with hierarchical attributes, while document-centric (a.k.a. content-oriented) XML data preserve document structures where they are still understandable without XML tags. Therefore, data-centric XML data are easier to convert into RDF, while document-centric XML data require large efforts on designing ontologies and mappings. The existing approaches assume data-centric XML data and RDF data as their translation, which can be roughly classified into the following: (1) to use XQuery to query on RDF data [7, 2], (2) to use SPARQL to query on XML data [1], and (3) query translation between XQuery and SPARQL [3].

3 SPARQL with XQuery-based Filtering

In this section, the proposed extension of SPARQL, the basic architecture of mixed databases, and the query optimization technique are introduced.

Query Definition: XQuery-based filtering aims to filter bindings of graph pattern in SPARQL that XML documents in the bindings satisfy XQuery. To this end, this XQuery is required to return boolean value. In the extended query, `XQueryFILTER` expression is added to the SPARQL definition based on the `FILTER` expression, which argument is an XQuery expression including variables in SPARQL. The following is an example of SPARQL with `XQueryFILTER`.

```
SELECT ?s
WHERE{ ?s a :Country; :safetyInfo ?doc; :populationTotal ?pop.
       FILTER ( ?pop > 10,000,000 ) . }
       XQueryFILTER (
          LET $x := doc(?doc)//mail[leaveDate > xs:date('2020-03-01')]
          RETURN contains($x, 'coronavirus')
       ). }
```

This query is to search for countries with more that 10 million people and with safety information about the coronavirus after March, 2020. Its `XQueryFILTER` contains SPARQL variable `?doc` which referred by `ex:safetyInfo` for a safety information XML file. During query evaluation, the XQuery in the `XQueryFILTER` can be performed by replacing `?doc` variable with one of its bindings. If the XQuery returns `true`, the binding remains in results, eliminated otherwise.

System Architecture: Fig. 1 represents a basic architecture for realizing SPARQL with `XQueryFILTER`. A basic assumption is that RDF and XML data

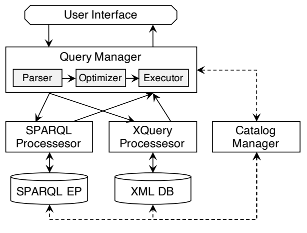


Fig. 1: Basic Architecture

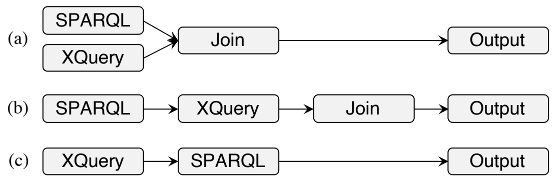


Fig. 2: Query Plans; (a) Parallel, (b) SPARQL First, and (c) XQuery First

are separately stored in SPARQL EP (endpoint) and XML DB, respectively. Dedicated query processors are associated to communicate with the databases, namely, SPARQL processor and XQuery processor. Query manager handles user query, decomposes it into SPARQL and XQuery, explores optimal query plans, executes them and merges the results. User interface communicates with users by receiving queries and returns their results.

Optimization: To realize efficient SPARQL with `XQueryFILTER`, the optimizer in the query manager chooses one query plan with the least cost among three possible plans to execute SPARQL with `XQueryFILTER` as shown in Fig. 2.

***Parallel*:** Execute SPARQL and XQuery in parallel and join results afterward.
***SPARQL First*:** Execute SPARQL first, push its bindings down into XQuery, and evaluate the bindings with XQuery results.
***XQuery First*:** Execute XQuery first and push its results down into SPARQL.

Execution costs of these query plans modeled on the basis of the following idea. Let C_{SPARQL}, C_{XQuery} respectively denote the processing costs of SPARQL and XQuery, and C_{Join} denotes the join cost. The costs of the parallel plan, $C^{(p)}$, the SPARQL first plan, $C^{(s)}$, and the XQuery first plan is as follows.

$$C^{(p)} = \max(C_{SPARQL}, C_{XQuery}) + C_{Join}, \tag{1}$$

$$C^{(s)} = C_{SPARQL} + \rho_{SPARQL} \cdot C_{XQuery} + C_{Join}, \tag{2}$$

$$C^{(x)} = C_{XQuery} + \rho_{XQuery} \cdot C_{SPARQL}, \tag{3}$$

where ρ_{SPARQL} and ρ_{XQuery} denote the selectivities of the preceding SPARQL and XQuery, respectively.

4 Experimental Evaluation

Experimental evaluation for showing efficiency of the proposed SPARQL with `XQueryFILTER` and query optimization was conducted. In this experiment, three scenarios are prepared by collecting XML documents related with LOD datasets, which have different settings of databases in terms of network latency and data size. XML data are stored in a local XML database using eXist-db (v. 5.2.0)[1], and RDF data in this experiment are stored locally using Apache Jena Fuseki

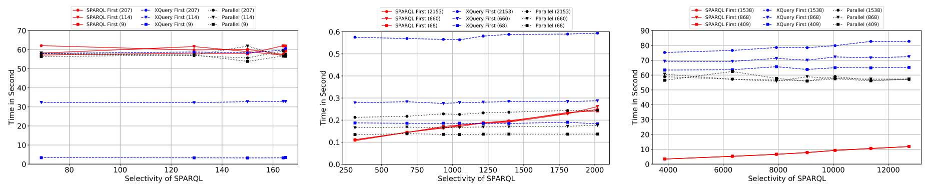
[1] `http://exist-db.org/`

Table 1: Expected DB Performance

Scenario	SPARQL EP	XML DB
C.S.	low	high
L.S.	comparable	comparable
D.S.	high	low

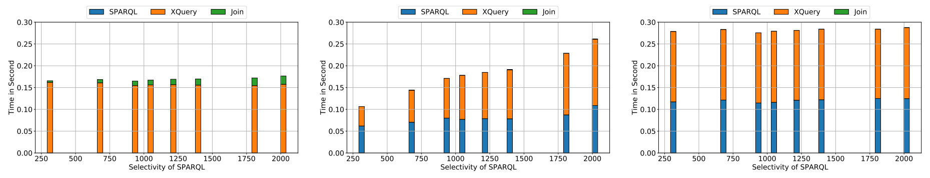
Table 2: Data Size for Scenarios

Scenario	XML (#doc)	LOD (#ent)
C.S.	207	207 + DBpedia
L.S.	8,435	106,341
D.S.	107,193	317,552



(a) C.S. (Country Search) (b) L.S. (Law Search) (c) D.S. (Discussion Search)

Fig. 3: Execution Time in Scenarios (Number in Legend: XQuery Selectivity)



(a) Parallel (b) SPARQL First (c) XQuery First

Fig. 4: Detail of Execution Time in L.S. Scenario with XQuery Selectivity of 660

(v. 3.14.0)[2]) or stored in external SPARQL endpoints (i.e., DBpedia SPARQL endpoint). Here, query processing efficiency is measured by query execution time.

Scenarios: In this experiment, three scenarios are prepared in order to observe query performances over different performances of underlying databases as summarized in Table 1. The first scenario (C.S. scenario) is to search countries with governmental safety information messages. This scenario is that cost of querying to SPARQL endpoint is high due a federated query using the `SERVICE` clause and that of querying to XML DB is low due to the small number of XML documents. The second scenario (L.S. scenario) is to search acts with their body texts. This scenario is that costs of querying SPARQL and XQuery are comparable due to using the local SPARQL endpoint and the moderate number of XML documents. The third scenario (D.S. scenario) is to search discussions in minute books about law enactment. This scenario is that cost of querying SPARQL is low and that of querying XQuery is high due to using the local SPARQL endpoint and the large number of XML documents. Queries are generated by using templates to control the selectivities of SPARQL and XQuery.

Results: Fig. 3 shows selected results for each scenario in terms of selectivities of SPARQL and XQuery. The common observation is as follows. First, the

[2] `https://jena.apache.org/documentation/fuseki2/index.html`

SPARQL first plan (resp. XQuery first plan) is linearly performed w.r.t. selectivity of SPARQL (resp. XQuery) query and it is scarcely affected by selectivity of XQuery (resp. SPARQL) query. Second, the parallel plan is nearly constant w.r.t.both selectivities of SPARQL and XQuery when their execution performances have a large gap like C.S. and D.S. scenarios. When these performances are comparable (as the L.S. scenario), this plan depends on both selectivities.

In Fig. 3(b), the optimal plan for XQuery selectivity of 660 is switched from the SPARQL first plan to the parallel plan around SPARQL selectivity of 1,000. Fig. 4 indicate a reason for this switching. This figure shows a breakdown of execution time in query plans in the form of a stacked bar graph of execution times of SPARQL, XQuery and Join (blue, orange and green colors, respectively). Basically, in this SPARQL with `XQueryFILTER`, SPARQL is executable relatively faster than XQuery (Fig. 4(b) and Fig. 4(c)). Therefore, SPARQL first plan is better plan as far as the number of XML documents being queried afterward is small. In this scenario, the number of queried XML documents more than 1,000 is a turning point that the parallel plan overcomes the SPARQL first plan.

This experiment indicates that the three query plans reflect the pros and cons of underlying databases. The proposed optimization technique captures these characteristics of plans by the cost equations in Equation 1, 2 and 3 and can successfully discover the best plan if statistics of database performances and selectivity estimations of SPARQL and XQuery are accurate.

Acknowledgements
This work was partly supported by JSPS KAKENHI Grant Number JP18K18056 and the Artificial Intelligence Research Promotion Foundation.

References

1. Akhtar, W., Kopecký, J., Krennwallner, T., Polleres, A.: XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In: ESWC 2008. pp. 432–447 (2008)
2. Almendros-Jiménez, J.M., Becerra-Terón, A., Torres, M.: Integrating and Querying OpenStreetMap and Linked Geo Open Data. Comput. J. **62**(3), 321–345 (2019)
3. Bikakis, N., Tsinaraki, C., Stavrakantonakis, I., Gioldasis, N., Christodoulakis, S.: The SPARQL2XQuery interoperability framework - Utilizing Schema Mapping, Schema Transformation and Query Translation to Integrate XML and the Semantic Web. World Wide Web **18**(2), 403–490 (2015)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst. **5**(3), 1–22 (2009)
5. Cowie, J.R., Lehnert, W.G.: Information Extraction. Commun. ACM **39**(1), 80–91 (1996)
6. Droop, M., Flarer, M., Groppe, J., Groppe, S., Linnemann, V., Pinggera, J., Santner, F., Schier, M., Schöpf, F., Staffler, H., Zugal, S.: Embedding Xpath Queries into SPARQL Queries. In: ICEIS 2008. pp. 5–14 (2008)
7. Groppe, S., Groppe, J., Linnemann, V., Kukulenz, D., Hoeller, N., Reinke, C.: Embedding SPARQL into XQuery/XSLT. In: SAC 2008. pp. 2271–2278 (2008)
8. Sherkhonov, E.: Data Exchange for Document-Centric XML. In: Proc. PhD Symposium@SIGMOD 2014. pp. 26–30 (2014)