Fediscount: Shopping Online at a Federated Store Using FedUP as SPARQL Federation Engine

Julien Aimonier-Davat, Minh-Hoang Dang, Pascal Molli, Brice Nédelec, Hala Skaf-Molli

Fediscount: Shopping Online at a Federated Store Using FedUP as SPARQL Federation Engine

Julien Aimonier-Davat
julien.aimonier-davat@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Minh-Hoang Dang
minh-hoang.dang@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Pascal Molli
pascal.molli@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Brice Nédelec
brice.nedelec@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Hala Skaf-Molli
hala.skaf@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Abstract

Processing SPARQL queries over large federations of SPARQL endpoints is essential for maintaining the Semantic Web decentralized. However, existing federation engines struggle to query more than a dozen of endpoints. We recently proposed FedUP, a new type of federation engine based on unions-over-joins query plans that outperforms state-of-the-art federation engines by orders of magnitude on large federations. This demonstration paper introduces Fediscount, a federated online shopping application based on the FedShop benchmark, illustrating the capabilities of FedUP. The application is based on standard Semantic Web technologies, enabling end-users to shop online in a virtual federated store comprising 20, 100, or even 200 SPARQL endpoints. This breakthrough opens up promising new avenues for developing and deploying federated applications.

1 INTRODUCTION

Processing SPARQL queries over large federations of SPARQL endpoints is crucial for maintaining the Semantic Web decentralized. Unfortunately, despite the existence of hundreds of SPARQL endpoints [5, 6, 11], existing state-of-the-art federation engines [8, 9, 10] only scale up to dozens [4].

FedUP [1] is a new type of federation engine that outperforms existing engines by orders of magnitude on the FedShop benchmark [4]. Unlike traditional federation engines, which build joins-over-unions query plans [3], FedUP builds unions-over-joins query plans where each subquery returns results. When processing queries, unions-over-joins query plans enable a drastic reduction in the size of intermediate results through more intensive uses of exclusive groups [10]. While SPARQL federation engines were traditionally confined to querying a dozen of knowledge graphs as in LargeRDFBench [7], the performance improvements offered by FedUP open new perspectives for federated query processing.

In this demonstration, we present Fediscount: a federated online shopping application powered by FedUP. Fediscount is based on the FedShop federated benchmark [4]. Like FedShop, Fediscount considers autonomous e-commerce vendors who sell products using SPARQL endpoints as backends. A federated store is a virtual store defined as a federation of vendor SPARQL endpoints. The federation engine gives the illusion that all shops belong to the same virtual store, enabling end-users to search and browse all products supplied by all shops. During the demonstration, users can explore a federated store of 20, 100 and 200 shops and observe the effect of changing the federation on the application. We have also integrated a feature for visualizing unions-over-joins query plans, allowing users to see the plans and execution times of all queries used in the application. To the best of our knowledge, this is the first presentation of such a federated application. Fediscount represents just a single example within a larger class of applications that can be realized using federation engines.

This paper is organized as follows: Section 2 presents basic notions of federation engines and the key elements of FedUP. Section 3 provides the necessary steps to create the tiny quotient summaries on which is relying FedUP. Section 4 details the characteristics of Fediscount, i.e., its data, queries, and different deployments. Section 5 concludes and outlines future work and perspectives.

2 FEDUP: A SPARQL FEDERATION ENGINE

We illustrate the key idea of FedUP [1] on the federation F_1 with 4 SPARQL endpoints $D1$, $D2$, $D3$, and $D4$ as follows:

http://D1/Scorpions	foaf:name	"Scorpions"	@http://D1
http://D1/Scorpions	foaf:based_near	http://D2/Hanover	@http://D1
http://D2/Hanover	geo:parentFeature	http://D2/Germany	@http://D2
http://D2/Germany	geo:names	"Federal...Germany"	@http://D2
http://D3/Kraftwerk	foaf:based_near	http://D4/Berlin	@http://D3
http://D3/Kraftwerk	foaf:name	"Kraftwerk"	@http://D3
http://D4/Berlin	geo:parentFeature	http://D4/Germany	@http://D4
http://D4/Germany	geo:names	"Federal...Germany"	@http://D4

The query $S6$ from LargeRDFBench [7] requests the bands that come from the Federal Republic of Germany. It has 4 triple patterns tp_1, tp_2, tp_3, and tp_4. The notation $tp_i@D_j$ means that the triple pattern tp_i must be evaluated on the endpoint D_j:

```
SELECT * WHERE {
    ?artist foaf:name ?name .                          #tp1@D1,D3
    ?artist foaf:based_near ?location .                #tp2@D1,D3
    ?location geo:parentFeature ?germany .             #tp3@D2,D4
    ?germany geo:name "Federal...Germany" . }          #tp4@D2,D4
```

The execution of such a query $S6$ over the federation F_1 returns 2 results comprising "Scorpions" and "Kraftwerk":

	?artist	?name	?location	?germany
μ_1	http://D1/Scorpions	Scorpions	http://D2/Hanover	http://D2/Germany
μ_2	http://D3/Kraftwerk	Kraftwerk	http://D4/Berlin	http://D4/Germany

To execute $S6$ over F_1, a traditional federation engine builds a logical joins-over-unions query plan composed of a root multi-join operator ($\bowtie$) with multi-union ($\cup$) operators as children. It produces the minimal query plan $S6_j$ where removing any element of multi-unions suppresses a result of $S6$:

$$\begin{aligned} S6_j = \bowtie \{ & \cup\{tp1@D1, tp1@D3\}, \\ & \cup\{tp2@D1, tp2@D3\}, \\ & \cup\{tp3@D2, tp3@D4\}, \\ & \cup\{tp4@D2, tp4@D4\}\} \end{aligned}$$

By applying the distributive property of joins over unions on $S6_j$, we obtain an equivalent Normalized unions-over-joins query plan, named $N(S6_j)$:

$$\begin{aligned} N\{S6_j\} = \cup\{ & \bowtie\{tp1@D1, tp2@D1, tp3@D2, tp4@D2\}, \\ & \bowtie\{tp1@D1, tp2@D1, tp3@D2, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D1, tp3@D4, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D1, tp3@D4, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D3, tp3@D2, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D3, tp3@D2, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D3, tp3@D4, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D1, tp2@D3, tp3@D4, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D1, tp3@D2, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D1, tp3@D2, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D1, tp3@D4, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D1, tp3@D4, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D3, tp3@D2, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D3, tp3@D2, tp4@D4\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D3, tp3@D4, tp4@D2\}, \quad (\text{is } \varnothing) \\ & \bowtie\{tp1@D3, tp2@D3, tp3@D4, tp4@D4\}\} \end{aligned}$$

However, most $N(S6_j)$ multi-joins return empty results. After Pruning empty multi-joins, we obtain the simplified unions-over-joins query plan $P(N(S6_j))$:

$$\begin{aligned} P(N(S6_j) = \cup\{ & \bowtie\{tp1@D1, tp2@D1, tp3@D2, tp4@D2\}, \\ & \bowtie\{tp1@D3, tp2@D3, tp3@D4, tp4@D4\}\} \end{aligned}$$

As several triple patterns share the same endpoints in multi-joins, we apply the Exclusive Group optimization [10] and obtain the final $S6_u = EG(P(N(S6_j)))$ logical query plan:

$$\begin{aligned} S6_u = \cup\{ & \bowtie\{\bowtie\{tp1, tp2\}@D1, \bowtie\{tp3, tp4\}@D2\} \\ & \bowtie\{\bowtie\{tp1, tp2\}@D3, \bowtie\{tp3, tp4\}@D4\}\} \end{aligned}$$

When we compare the number of calls to endpoints, $S6_j$ makes 8 calls to endpoints, whereas $S6_u$ makes only 4 calls to endpoints, i.e., $S6_u$ is a better logical query plan for $S6$ on F_1. Unfortunately, transforming $S6_j$ into $S6_u$ may generate an exponential number of multi-joins that must be pruned. The converse is false: transforming $S6_u$ into $S6_j$ is straightforward. This highlights the gap in information between a joins-over-unions and the corresponding unions-over-joins query plan. Unions-over-joins query plans encode more information than joins-over-unions ones.

To avoid the costly normalise-and-prune step, FedUP (i) evaluates $S6$ over the federation F_1, (ii) tracks the provenance of every result, (iii) and extracts the unions-of-joins logical plan from the provenance of these results [1].

To illustrate, consider the 2 mappings resulting from the execution of Query $S6$ on the federation F_1, and replace the values by their provenance and the variables by their corresponding triple patterns, e.g., `http://D1/Scorpions` comes from Endpoint $D1$ and `?artist` is produced by tp_1. If we apply this simple rule to all results, we obtain a table that indicates the endpoint where each triple pattern has to be evaluated:

	tp1	tp2	tp3	tp4
μ_1	http://D1	http://D1	http://D2	http://D2
μ_2	http://D3	http://D3	http://D4	http://D4

We observe that a row corresponds to the multi-joins of a unions-over-joins query plan ($P(N(S6_j))$), while a column corresponds to the multi-unions of a joins-over-unions query plan ($S6_j$).

Although extracting query plans from results looks appealing, this introduces a vicious cycle: building the logical plan requires query results, and getting query results requires the execution of a logical plan. To tackle this issue, FedUP extracts query plan from results obtained over a tiny quotient summary of the federation rather than the federation itself.

3 FEDUP'S SUMMARIES

FedUP's summaries are inspired by HiBISCuS' [8]: for each triple (*subject*, *predicate*, *object*) of the federation, FedUP only keeps the authority of subject or object IRIs (e.g., `http://vendor1/Product2` becomes `http://vendor1`), and replaces any literal by "any" (e.g., "Federal Republic of Germany" becomes "any"). FedUP builds its quotient summary by executing the query Q_s over the federation members:

```
CONSTRUCT { ?ps ?p ?po } WHERE {
  ?s ?p ?o .  FILTER ISIRI( ?s )
  BIND(URI(REPLACE(STR( ?s ), "^(https?://?.*?)/.*", "$1")) AS ?ps )
  BIND(IF(ISIRI( ?o ),
          URI(REPLACE(STR( ?o ),
                      "^(https?://?.*?)/.*", "$1")),"any") AS ?po )}
```

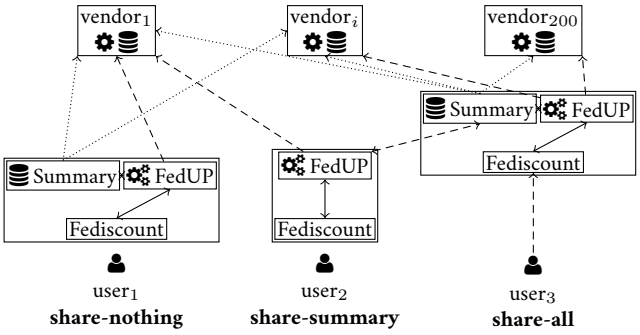


Figure 1: Some application deployments of Fediscount. Dotted arrows represent construct queries of summaries. Dashed arrows represent remote queries or subqueries.

By executing Q_s on each SPARQL endpoint of F_1, FedUP builds the summary $S(F_1)$:

http://D1	foaf:name	any	http://D1
http://D1	foaf:based_near	http://D2	http://D1
http://D2	geo:parentFeature	http://D2	http://D2
http://D2	geo:names	any	http://D2
http://D3	foaf:based_near	http://D4	http://D3
http://D3	foaf:name	any	http://D3
http://D4	geo:parentFeature	http://D4	http://D4
http://D4	geo:names	any	http://D4

To evaluate $S6$ on $S(F_1)$, FedUP first applies the same summary function to triple patterns of $S6$, e.g., $S(S6)$ turns tp_4 of $S6$ into `?germany geo:name "any"`.

Using quotient summaries introduces inaccuracy, i.e., some multi-joins that return results on the summary may not return results on the federation. However, the converse is false; if there is no result in the summary, there is no result in the federation.

4 FEDISCOUNT: FEDERATED SHOPPING

For this demonstration, we propose Fediscount[1]: a federated Web application based on the use case of the FedShop benchmark [4]. It involves a customer navigating through a virtual shop defined as a federation of SPARQL endpoints representing vendors and rating sites. The navigation is powered by federated SPARQL queries evaluated over the federation. This federation gives the illusion of one single endpoint hosting all the different vendors. The federated SPARQL queries retrieve products based on certain criteria, obtain more information about products, compare products, find similar products, and locate product reviews. The overall schema of the application involves vendors offering products for a price, a time span, and delivery conditions. Rating sites provide reviews with user comments and ratings for products. Products themselves are described using features, product types, and producers.

[1] Fediscount's name comes from a popular French e-commerce Website.

Fediscount data We filled the vendors and rating sites endpoints using the synthetic data of FedShop200[2], i.e., a configuration of FedShop made of 200 endpoints: 100 vendors and 100 rating sites. The dataset comprises 43,165,055 quads. Vendors and rating sites choose products at random from a Gaussian distribution in a shared catalog of 200,000 products. The full description of products with features, product type, and producer description are replicated to all vendors to ensure that vendors are fully autonomous.

Fediscount queries We integrated in Fediscount the 12 template queries of FedShop, i.e., we build all forms required to instantiate template queries. For example, the "explore" use case starts by searching for products given a product type, 2 product features and a filter parameter, as shown in the query $Q1$:

```
SELECT DISTINCT ?product ?label  WHERE {
  ?product rdfs:label ?label                                    .
  ?product rdf:type ?localProductType                           .
  ?localProductType owl:sameAs %ProductType%                    .
  ?product bsbm:productFeature ?localProductFeature1            .
  ?localProductFeature1 owl:sameAs %ProductFeature1%            .
  ?product bsbm:productFeature ?localProductFeature2            .
  ?localProductFeature2 owl:sameAs %ProductFeature2%            .
  ?product bsbm:productPropertyNumeric1 ?value1                 .
  FILTER (?value1 > %constValue1%)
} ORDER BY ?label LIMIT 10
```

Once $Q1$ retrieved the matching products from the federation, the reference of each of these products may be used to call the next query of the FedShop use case (e.g., provide detailed information about the targeted product, or recommend similar products).

Fediscount deployment There are several ways to deploy Fediscount with different trade-off as depicted in Figure 1.

share-nothing: $user_1$ runs Fediscount on the user side; she composes her own federation, manages her summary, and independently runs federated queries.

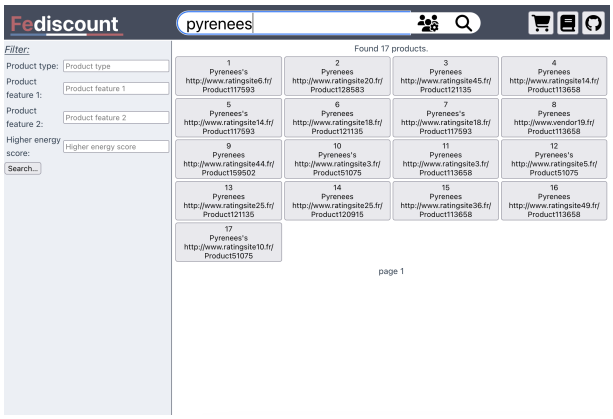
share-summary: $user_2$ shares summaries with a Fediscount provider but runs federated queries independently. She is not limited by Fediscount provider's processing resources.

share-all: $user_3$ trusts a Fediscount provider to compose the federation, manage summaries, and run federated queries. Query processing is limited by Fediscount provider's resources.

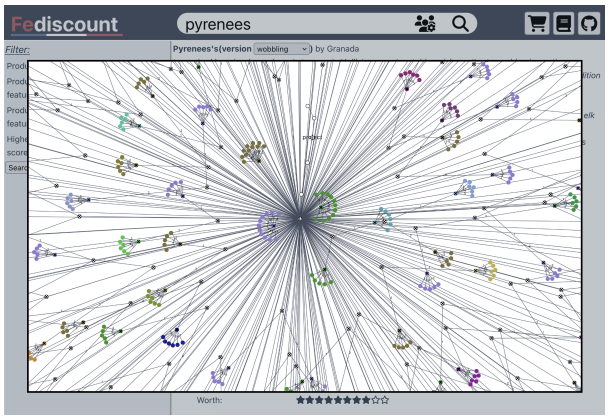
Fediscount summaries Summaries are computed as described in the previous section. We provide 3 federations: one with the first 20 endpoints, a second one with 100 endpoints, and the last with all 200 endpoints. These summaries are extremely compact:

	#quads	size
FedShop200	43,165,055	12GB
10 vendors + 10 ratings	1420	261KB
50 vendors + 50 ratings	2916	520KB
100 vendors + 100 ratings	5800	1MB

[2] https://zenodo.org/records/8059913



(a) Fediscount's keyword search $Q06$ returns 17 products.



(b) FedUP's plans to retrieve similar products ($Q05$).

Figure 2: Fediscount screenshots to search products related to "pyrenees" in a federation comprising 100 endpoints.

4.1 Live Demonstration

The demonstration code is available at https://github.com/GDD-Nantes/fediscount-website. The code of FedUP is available at https://github.com/GDD-Nantes/fedup. The datasets and queries of FedShop are available at https://zenodo.org/records/8339384.

A single Virtuoso server emulates the federation by hosting the 200 virtual endpoints representing the shops and rating sites. FedUP's summaries are managed by Apache Jena. The whole demonstration can run on a single computer.

From the end-user point of view, Fediscount behaves as conventional e-commerce websites, i.e., searching for products, compare prices, see reviews, etc. However, when users select larger federations of vendors and rating sites, they increase the number of available offers and reviews coming from more Fediscount shops.

Once launched, the main screen of the demonstration looks like Figure 2. The first step is to choose which federation we explore the federated shop, i.e., 20, 100 or 200 endpoints. Next, the users have several ways to search for products, with keywords, product type and features, etc. For instance, Figure 2a shows the result of a keyword search for "pyrenees". These queries return products with offers and/or reviews from different shops and rating sites. Once a product is selected, more detailed information is printed.

As shown in Figure 2b, we integrated into this demonstration the possibility of displaying the query plan of all queries executed during the live performance. This allows observing the real unions-over-joins query plans computed by FedUP over the federation. Users can observe the different kinds of federated queries used by Fediscount and their evolution when the federation increases.

5 CONCLUSION

In this demonstration, we introduced Fediscount, a pioneering federated e-commerce application capable of managing a federation of 200 shops while maintaining reasonable response times. Fediscount is the first federated application to exploit a federation of SPARQL endpoints, a breakthrough facilitated by FedUP, allowing scalability across federation sizes. This advancement demonstrates the practical feasibility of such applications and paves the way for a whole range of new applications for federation engines.

Keeping the federation up requires obtaining and maintaining summaries of the federation. For this demonstration, summaries were generated through SPARQL queries targeting the SPARQL endpoints of vendors and rating sites. However, the practical application of this method faces challenges due to quotas forced on public SPARQL endpoints for fair usage of resources. These restrictions often result in summary queries timing out, leading to incomplete data retrieval from public endpoints such as DBPedia or Wikidata. Nevertheless, the support for sampling emerges as a promising direction for computing and maintaining summaries [2].

Future work also includes better optimizations of unions-over-joins plans. Fediscount's visualisation tool depicts incredibly complex plans with many subqueries to factorize, cache, and rank.

Acknowledgments This work is supported by the French ANR project DeKaloG (Decentralized Knowledge Graphs) ANR-19-CE23-0014, and the French Labex CominLabs project MiKroloG (Microdata Knowledge Graph).

REFERENCES

[1] Julien Aimonier-Davat, Minh-Hoang Dang, Pascal Molli, Brice Nédelec, and Hala Skaf-Molli. Fedup: Querying large-scale federations of sparql endpoints. In *The Web Conference 2024 (WWW'2024)*, Singapore, May 2024.

[2] Julien Aimonier-Davat, Brice Nédelec, Minh-Hoang Dang, Pascal Molli, and Hala Skaf-Molli. RAW-JENA: Approximate query processing for SPARQL endpoints. In *The Semantic Web - ISWC 2023 - 22th International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings*, 2023.

[3] Sijin Cheng and Olaf Hartig. FedQPL: A language for logical query plans over heterogeneous federations of RDF data sources. In *the 22nd International Conference on Information Integration and Web-Based Applications & Services*, page 436–445. Association for Computing Machinery, 2021.

[4] Minh-Hoang Dang, Julien Aimonier-Davat, Pascal Molli, Olaf Hartig, Hala Skaf-Molli, and Yotlan Le Crom. FedShop: A benchmark for testing the scalability of SPARQL federation engines. In *The Semantic Web - ISWC 2023 - 22th International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings*, 2023.

[5] Ali Hasnain, Qaiser Mehmood, and Syeda Sana e Zainab ang Aidan Hogan. SPORTAL: profiling the content of public SPARQL endpoints. *Int. J. Semantic Web Inf. Syst.*, 12(3):134–163, 2016.

[6] Pierre Maillot, Olivier Corby, Catherine Faron, Fabien Gandon, and Franck Michel. IndeGx: A model and a framework for indexing RDF knowledge graphs with SPARQL-based test suits. *J. Web Semant.*, 76:100775, 2023.

[7] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation. *J. Web Semant.*, 48:85–125, 2018.

[8] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In *European Semantic Web Conference (ESWC)*, pages 176–191. Springer, 2014.

[9] Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. CostFed: Cost-based query optimization for SPARQL endpoint federation. In *14th International Conference on Semantic Systems (SEMANTICS)*, pages 163–174. Elsevier, 2018.

[10] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference (ISWC)*. Springer, 2011.

[11] Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil-Aranda. SPARQLES: Monitoring public SPARQL endpoints. *Semantic web*, 8(6):1049–1065, 2017.