Parallelizing Federated SPARQL Queries in Presence of Replicated Data

Thomas Minier, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli

Parallelizing Federated SPARQL Queries in Presence of Replicated Data *

Thomas Minier[1], Gabriela Montoya[2], Hala Skaf-Molli[1], and Pascal Molli[1]

[1] LS2N – Nantes University, France
{thomas.minier1@etu.,hala.skaf@,pascal.molli@}univ-nantes.fr
[2] Department of Computer Science – Aalborg University, Denmark
gmontoya@cs.aau.dk

Abstract. Federated query engines have been enhanced to exploit new data localities created by replicated data, e.g., FEDRA. However, existing replication aware federated query engines mainly focus on pruning sources during the source selection and query decomposition in order to reduce intermediate results thanks to data locality. In this paper, we implement a replication-aware parallel join operator: PEN. This operator can be used to exploit replicated data during query execution. For existing replication-aware federated query engines, this operator exploits replicated data to parallelize the execution of joins and reduce execution time. For Triple Pattern Fragment (TPF) clients, this operator exploits the availability of several TPF servers exposing the same dataset to share the load among the servers. We implemented PEN in the federated query engine FEDX with the replicated-aware source selection FEDRAand in the reference TPF client. We empirically evaluated the performance of engines extended with the PEN operator and the experimental results suggest that our extensions outperform the existing approaches in terms of execution time and balance of load among the servers, respectively.
Keywords: Linked Data · Parallel Query Processing · Fragment Replication · Federated SPARQL Queries Processing · Triple Pattern Fragment · Load Balancing.

1 Introduction

Following the Linked Data principles, billions of RDF triples are made available through SPARQL endpoints. Even if federated SPARQL query engines [9,17,1] allow to execute SPARQL queries over multiple SPARQL endpoints, data availability and reliability of SPARQL endpoints is still an issue [5].

Data replication is a common practice to overcome availability issues in distributed databases [15]. However, data replication in Linked Data is more challenging: the autonomy of data providers hosting SPARQL endpoints, and data consumers running federated query engines, prevent data replication to be designed. The fragmentation schema and the replication schema remain unknown

* Pre-print of a paper accepted in The Semantic Web: ESWC 2017 Satellite Events. The final publication is available at Springer via https://doi.org/10.1007/978-3-319-70407-4_33

until a data consumer defines a federation of SPARQL endpoints in a federated query engine.

Existing replication-aware [13,14] and duplicate-aware [16] federated query engines focus on source selection and query decomposition in order to prune redundant sources and use data-locality to reduce intermediate results. We point out that replicated data can also be used to parallelize query processing, and consequently reduce execution time.

In the previous work [12], we proposed PeNeLoop, abbreviated as Pen in this paper, a replication-aware parallel join operator. More precisely, Pen solves the parallel join problem with fragment replication (PJP-FR). Given a SPARQL query and a set of data sources with replicated fragments, the problem is to use all data sources to reduce query execution time while preserving answer completeness and reducing data redundancy.

In contrast to *inter-operator parallelism* proposed in the state-of-the-art federated query engines [1,17], Pen introduces parallelization at the operator level in order to preserve properties ensured by replicated-aware source selection strategies [13] and replication-aware query decompositions [14]. Pen is based on Bound Join operator implemented in FedX [17]. Bound joins were originally designed to reduce the number of requests sent in a nested loop join [15]. Pen extends bound joins processing to use all relevant endpoints with replicated fragments and distribute join processing among them.

In this work, we extend TPF client [18] with Pen. Pen will exploit the availability of several TPF servers exposing the same dataset to share the load among the servers. We implemented Pen in the reference TPF client. This paper presents our contribution to SPARQL federation and TPF federation : (i) We present Pen, a novel replication-aware parallel join operator that uses replicated fragments to reduce query execution time.

(ii) We extend federated query engine FedX [17] and the source selection strategy Fedra [13] and the TPF client with Pen.

(iii) We experiment FedX, FedX + Fedra FedX + Fedra + Pen and TPF + Pen in diffexrent setups. We show that FedX + Fedra + Pen outperforms FedX and FedX + Fedra in terms of execution time while preserving properties of Fedra in terms of reduced number of transferred tuples and answer completeness. The improvements are significative for queries with a large number of intermediate results. (iv) We show that TPF + Pen does not improve execution time, however, it equally distributes the load among servers.

The paper is organized as follows: Section 2 provides background and motivations. Section 3 presents the Pen approach and algorithm. Section 4 presents our experimental setup and describes our results. Section 5 summarizes related works. Finally, conclusions and future works are outlined in Section 6.

2 Background and Motivations

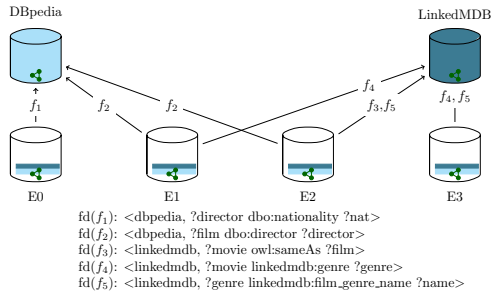
For replicating data, we follow the approach of replicated fragments introduced in [13,14]. Data consumers replicate fragments composed of RDF triples that sat-

(a) Fragment description

triples(f): { dbr:A_Knight's_Tale
dbo:director dbr:Brian_Helgeland,
dbr:A_Thousand_Clowns
dbo:director dbr:Fred_Coe,
dbr:Alfie_(1966_film)
dbo:director dbr:Lewis_Gilbert,
dbr:A_Moody_Christmas
dbo:director dbr:Trent_O'Donnell,
dbr:A_Movie dbo:director
dbr:Bruce_Conner, ··· }

fd(f): <dbpedia, ?film dbo:director ?director>

(b) Replicated fragments



(c) Federated SPARQL query $Q1$ and its relevant fragments and endpoints

```
select distinct *
where {
  ?director dbo:nationality ?nat.          (tp1)
  ?film db:director ?director.             (tp2)
  ?movie owl:sameAs ?film.                 (tp3)
  ?movie linkedmdb:genre ?genre.           (tp4)
  ?genre linkedmdb:film_genre_name ?gname. (tp5)
}
```

Triple pattern	Relevant fragment	Relevant endpoint
tp_1	f_1	E_0
tp_2	f_2	E_1, E_2
tp_3	f_3	E_2
tp_4	f_4	E_1, E_3
tp_5	f_5	E_2, E_3

Fig. 1: A federation with replicated fragments

isfy a given triple pattern. Figure 1a shows a fragment from DBpedia which contains RDF triples that match the triple pattern `?film dbo:director ?director`. Fragments are described using a 2-tuple `fd` that indicates the authoritative source of the fragment, e.g. DBpedia, and the triple pattern met by the fragment's triples.

Figure 1b shows a federation with four SPARQL endpoints: E_0, E_1, E_2 and E_3. These endpoints expose replicated fragments from DBpedia and LinkedMDB. Figure 1c describes a federated SPARQL query $Q1$ executed against this federation and its relevant fragments. For instance, the triple pattern tp_4 has relevant fragment f_4 that has been replicated at E_1 and E_3.

The logical plan of Q1 produced by FedX [17] is presented in Figure 2a. As FedX is not replication-aware, *i.e.*, it does not know that the evaluation of tp_2 at E_1 or E_2 will produce the same results, query execution following this plan will retrieve redundant data from endpoints and increase significantly the query execution time.

The Fedra [13] replication-aware source selection prunes redundant sources in order to minimize intermediate results. Fedra selects E_2 for tp_2, tp_3 and tp_5, E_1 for tp_4 and E_0 for tp_1. Next, Fedra lets FedX builds the logical plan of Figure 2b that minimizes intermediate results. Notice that this plan sends a large subquery to E_2, which can be heavy to compute, reducing the endpoint's

(a) FedX Left-Linear plan for $Q1$ **(b)** FedX + Fedra Left-Linear plan for $Q1$

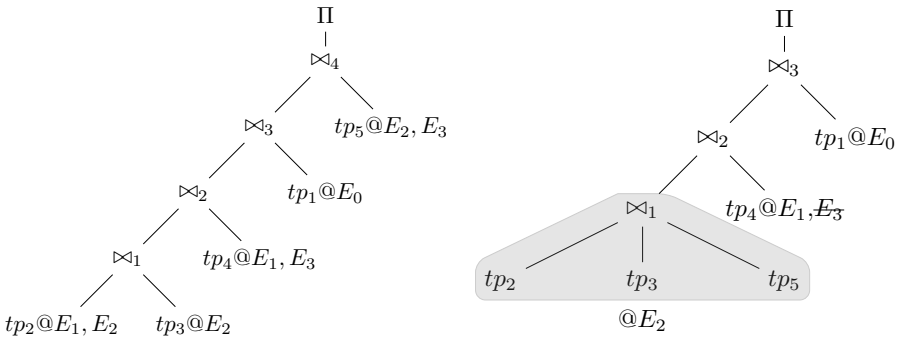


Fig. 2: Logical plans generated by FedX and FedX + Fedra for $Q1$

availability. Therefore, data locality have a negative impact on load balancing by creating hotspots.

As pointed in Figure 2b, Fedra has removed E_3 from selected sources of tp_4. However, it also removes an opportunity of parallelization. Indeed, it is possible to use both endpoints to perform in parallel half of the join of $\bowtie_2$ with E_1 and the other half with E_3, as they mirror each other [3].

The same parallelization principle can be applied in the context of Triple Pattern Fragments (TPF) interface [18]. Traditionally, a TPF client decompose a SPARQL query Q into simple triple patterns queries (TPQs) evaluated by a TPF server, and results are joined locally. TPQs processing can be distributed among TPF servers that replicate the same relevant data for Q.

Moreover, as TPF interfaces only accept TPQs, a TPF client cannot use data locality during query processing. This generate more remote HTTP calls, but also prevent the creation of hotspots as those seen before. Parallelization of these remote calls will then balance the load among servers and increase their availability.

Such parallelization can be obtained with a replication-aware query decomposer or with intra-operator [15] parallelism. In this paper, we focus on intra-operator parallelism because it can be easily embedded in current (federated) query engines. Consequently, the challenge is to build replication-aware parallel operators to speed-up query execution.

Parallel Join Problem with Fragment Replication (PJP-FR)

Given S_1 and S_2 two disjoint sets of replicated data sources. A set of replicated data sources is a set of endpoints that replicate the same fragments. Given a join $\bowtie_i$ between O_1 and O_2 with relevant sources respectively, S_1 and S_2. The parallel join problem with fragment replication is to distribute the execution of

[3] Note that joins $\bowtie_1$ and $\bowtie_3$ cannot be parallelized in this way, because $\bowtie_1$ is a local join performed at E_2, and tp_1 has only one relevant source.

join $\bowtie_i$ among endpoints of S_1 and S_2 in order to minimize the execution time while guaranteeing complete query answers.

3 Pen : A Replication-Aware Nested Loop Join Operator

Pen is a solution for parallel join problem with fragment replication with the following assumptions: (i) we focus on nested loop join (NLJ), (ii) we do not consider the load of different endpoints, (iii) we consider that replicated fragments are synchronized, (iv) replicated sources are determined by a replication-aware source selection algorithm, such as Fedra before pruning.

Pen can be used for any federation of SPARQL processing services, *endpoint*, that provides access to replicated data. These services can process unrestricted SPARQL queries, e.g., SPARQL endpoints, or restricted SPARQL queries, e.g., TPF servers.

3.1 NLJ Processing

During a NLJ processing, the query engine iteratively evaluates each triple pattern, starting with a single pattern and substituting the set of mappings produced by the pattern's execution in the next evaluation step. Even if a NLJ is more efficient when the first evaluated triple pattern is more selective than the others, it still produces many remote requests in a distributed setting. For federated SPARQL queries, Schwarte et al. [17] proposed the Bound Join (BJ) operator to minimize the number of join steps and the number of requests sent in nested loop joins. A BJ consists of a nested loop join where sets of mappings are grouped in *blocks*, *i.e.*, as a single subquery using SPARQL UNION constructs. The subquery is then sent to the relevant endpoint in a single remote request. This technique acts as a distributed semijoin and allows to reduce the number of requests by a factor equivalent to the size of the *block*.

SPARQL query processing with Triple Pattern Fragments [18] (TPF) also resolves joins in a NLJ fashion and rely on dynamic iterators that optimize locally each join step. The TPF reference server provides only support for BJ with block size $b = 1$, i.e., simple nested loop (SNJ) join. In the following sections we detail a new strategy to evaluate BJs, but naturally this applies to the particular case of SNJ.

Pen proposes to parallelize the BJ operator itself. Instead of sending all blocks to the same endpoint, Pen uses the knowledge about replicated sources to further parallelize the bound join operator. When processing a join in a basic graph pattern (BGP), if the current triple pattern has N relevant sources that replicate the same fragment, Pen sends each block to a different endpoint in a Round Robin fashion, *i.e.*, the block b_i is sent to the endpoint E_k, $k = i \mod N$. Therefore, Pen does not increase the number of remote calls while increasing the parallelization during join processing.

Algorithm 1: PeNeLoop join algorithm

Input: $tp = <s, p, o>$: a triple pattern, $E = \{E_0, \ldots, E_{m-1}\}$: relevant endpoints of tp, $NextOp$: next operator in the pipeline, b: maximum number of mappings per block

Data: M_i: a set of mappings produced by the previous operator in the pipeline, $B = \{M_1, \ldots, M_n\}$: block of sets of mappings waiting to be sent

Init: $B = \{\}$, $k = 0$

```
1  SendBlock(block, tp):
2  |  Q = GroupedSubquery(block, tp)
3  |  SendQuery(Q) to E_k
4  |  B = {}
5  |  k = (k + 1) mod Size(E)
6  ↯ onMappings(M_i):
7  |  B = B ∪ {M_i}
8  |  if Size(B) ⩾ b then
9  |  |  SendBlock(B, tp)
10 |  end
11 ↯ onResults(R):
12 |  Send(R) to NextOp
13 ↯ onEnd():
14 |  if Size(B) ⩾ 0 then
15 |  |  SendBlock(B, tp)
16 |  end
17 |  Close()
```

3.2 Pen Algorithm

Pen is defined as part of a pipelining approach allowing for intermediate results to be processed by the next operator as soon as they are ready, providing higher throughput than a blocking model.

Algorithm 1 describes the Pen algorithm using an event driven paradigm. Sets of mappings M_i are produced by the previous operator in the pipeline and sent in continuous to Pen operator. When a set M_i arrives (Line 6), it is stored in the next block B. When B reaches its maximum size b (Line 8), Pen generates a subquery in a Bound Join fashion using B and tp (Line 2). Then, the subquery is sent to the endpoint E_k (Line 3), B is cleared and the next endpoint is selected using our Round Robin approach (Line 5).

When results, *i.e.*, new sets of mappings, arrive from the requested endpoints (Line 11), they are sent to the next operator in the pipeline. Finally, when the previous operator has completed its work and will not produce any more data (Line 13), Pen sends the last non-empty block and then close the operator.

In the following, we illustrate Pen processing for the query $Q1$ (Figure 1c) using the query plan generated by FedX + Fedra (Figure 2b). For simplicity, we fix $b = 2$.

Figure 3 illustrates a snapshot of the pipeline during the evaluation of the triple pattern tp_4 of the query $Q1$. We focus on processing of join $\bowtie_2$, performed using Pen. Two blocks $\{M_1, M_2\}$ and $\{M_3, M_4\}$ have been already sent to E_1 and E_3, respectively. A set of mappings M_5 arrived from the join $\bowtie_1$ and was placed in the next block. When another set of mappings M_6 arrives, the block will be full and sent to the next endpoint E_1. Join $\bowtie_2$ ends when no more mappings are produced by join $\bowtie_1$.

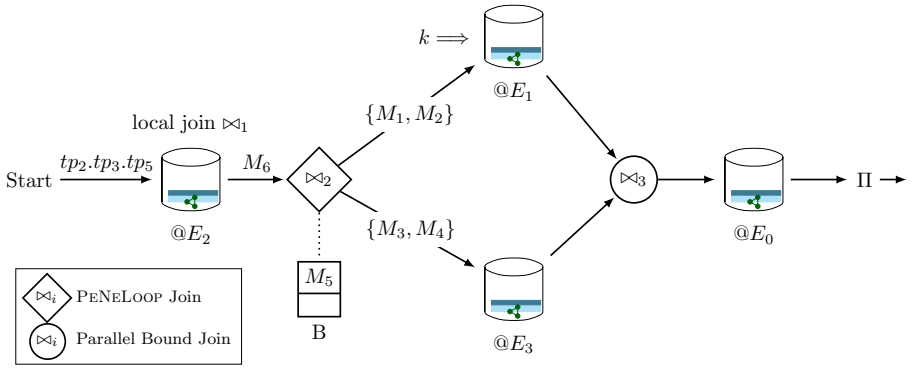


Fig. 3: Join processing of federated query $Q1$ with Pen

4 Experimental Study

The goal of the experimental study is to evaluate the impact of Pen parallelization on execution time. For federated SPARQL queries, such reduction is obtained without degrading the reduced number of transferred tuples and the answer completeness granted by Fedra. For Triple Pattern Fragments, Pen parallelization has a positive effect on load balancing between servers.

For federated SPARQL queries, we compare the performance of the federated query engine FedX alone, FedX with the addition of Fedra (FedX + Fedra) and FedX with both Fedra and Pen (FedX + Fedra + Pen). For TPF, we compare the performance of the reference TPF client alone and an extension with Pen (TPF + Pen). Note that combination of TPF with Fedrais not possible because the TPF server cannot take advantage of data locality.

We expect to see that FedX + Fedra + Pen exhibits lower query execution time than FedX and FedX + Fedra, while maintaining the same number of transferred tuples and answer completeness. We also expect to see that TPF + Penexhibits similar query execution time than the reference, but reduce the number of HTTP calls addressed per server.

Dataset and Queries: We use one instance of the Waterloo SPARQL Diversity Test Suite (WatDiv) synthetic dataset [2,3] with 10^5 triples. We generate 50,000 queries from 500 templates. Next, we unbound subjects and objects of each query. 100 queries with at least one join are then randomly picked to be executed against our federations. Generated queries are STAR, PATH and SNOWFLAKE shaped queries, we use the DISTINCT modifier.

Queries that failed to deliver an answer due to a query engine internal error are excluded from the final results.

Federations: For the SPARQL endpoints, we consider replication of dataset fragments, i.e., partial replication. We setup three federations with respectively 10, 20 and 30 SPARQL endpoints, and generate three versions of each of these federations by randomizing the fragmentation schema. Every schema is distinct

from the others. Fragments are created from the 100 random queries and are replicated exactly three times to provide opportunities of parallelization. For the TPF federations we consider TPF server that provide access to the same dataset, i.e., total replication. For experiments with TPF, we setup up to five TPF servers, each one using four workers and a HDT backend [7].

To measure the number of transferred tuples and the repartition of HTTP calls, query engines accesses SPARQL endpoints and TPF servers through a proxy. All the federation endpoints and TPF servers are deployed on the same machine, and to simulate the network latency, the proxies were configured to add a delay of 30ms to each request.

Hardware configuration: One machine with Intel Xeon E5-2680 v2 2.80GHz and 128GB of RAM hosts the SPARQL endpoints and performs the queries. Each SPARQL endpoint is deployed using Jena Fuseki 1.1.1[4]. Fuseki is configured to handle incoming queries on only one executing thread to increase the stress load and study the effect of the parallelization done by the engine. Endpoints have no limitations in term of memory used.

Implementations: FedX + Fedra implementation[5] (in Java) has been modified to preserve the multiple sources that provide the same relevant fragments. For TPF, Pen is implemented on top of the reference TPF client[6].

Additionally, FedX join processing has been modified to remove some redundant synchronization barriers imposed by FedX on the first join of a plan, *i.e.*, the right operand can start execution before the left one has finished its evaluation, and to use Pen operator when possible[7]. Every configuration of this experimental study has received the same modifications. Proxies used to measure results are implemented in Java 1.7, using the Apache HttpComponents Client library 4.3.5[8].

4.1 Pen with Federated SPARQL queries

Evaluation Metrics: *i*) *Execution Time (ET):* is the elapsed time since the query is posed until the complete answer is produced. We used a timeout of 1800 seconds. *ii*) *Number of parallelized queries (NPQ):* is the number of queries where at least one join has been parallelized by Pen. Queries marked as *improved* have a lower execution time (ET) with FedX+Fedra+Pen than with FedX+Fedra. *iii*) *Number of Transferred Tuples (NTT):* is the number of transferred tuples from all the endpoints to the query engine during a query evaluation. This metric is only used for federated SPARQL queries. *iv*) *Completeness (C):* is the ratio between the answers produced by the query execution engine and the answers produced by the evaluation of the query over the set of all triples available in the federation; values range between 0.0 and 1.0.

[4] `http://jena.apache.org/`, January 2015.
[5] `https://github.com/gmontoya/fedra`, June 2016.
[6] `https://github.com/LinkedDataFragments/Client.js`, June 2017
[7] Implementation available at: `https://github.com/Callidon/peneloop-fedx`
[8] `https://hc.apache.org/`, October 2014.

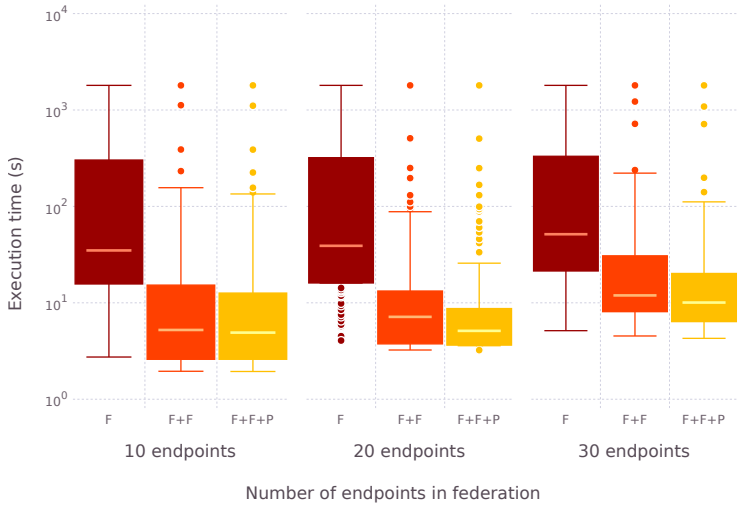


Fig. 4: Average execution time with FedX (F), FedX + Fedra (F+F) and FedX + Fedra + Pen (F+F+P).

Results presented for *ET*, *NTT* and *C* correspond to the average over the three versions generated for each size of federation. In all cases, FedX+Fedra+Pen is able to produce the same answers as FedX + Fedra for all queries (detailed completeness (*C*) results are presented in [12].)

Statistical Analysis: The Wilcoxon signed rank test [19] for paired non-uniform data is used to study the significance of the improvements on performance obtained when the join execution benefits from replicated fragments.[9]

Execution time Figure 4 summarizes the execution time (*ET*) for the three federations. Execution time (*ET*) with FedX + Fedra + Pen is better for all federations than with FedX and FedX + Fedra. As queries have unbounded subjects and unbounded objects, they generated more intermediate results during joins, which allow Pen to distribute more bindings between relevant sources. Figure 5 presents the execution time for queries with a large number of intermediate results (at least 1000 tuples). This represents 562 queries out of 865 for all federations. Pen is even more efficient for queries with a large number of intermediate results. This is an important result because generally the number of the intermediate results impacts negatively the query execution time.

Both FedX + Fedra and FedX + Fedra + Pen benefit from the reduction of transferred tuples granted by Fedra, which reduce the number of mappings that Pen can distribute.

[9] The Wilcoxon signed rank test was computed using the R project (`http://www.r-project.org/`)

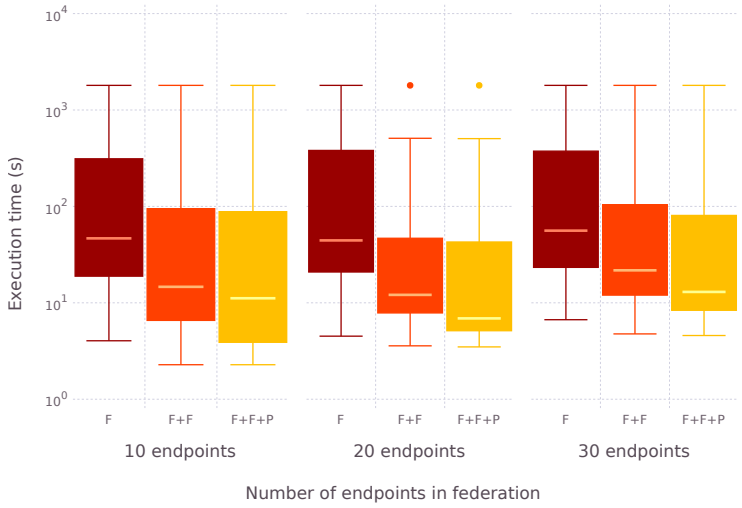


Fig. 5: Average execution time with FedX (F), FedX + Fedra (F+F) and FedX + Fedra + Pen (F+F+P) for queries with at least 1000 intermediate results.

To confirm that Pen reduces the execution time of FedX+Fedra, a Wilcoxon signed rank test was run for results of Figure 4 with the hypotheses:
H0: Pen does not change the engine query execution time.
H1: Pen reduces FedX + Fedra's query execution time.

We obtain p-values no greater than 1.639×10^{-4} for each federation. These low p-values allow for rejecting the null hypothesis that the execution time of FedX+Fedra and FedX+Fedra+Pen are the same. Additionally, it supports the acceptance of the alternative hypothesis that FedX + Fedra + Pen has a lower execution time.

Number of Parallelized Queries Figure 6 presents the number of parallelized queries (NPQ) in FedX+Fedra+Pen for the three versions of each federation. Pen increases query parallelization during join processing, especially in larger federations where fragments are more scattered across endpoints. In most cases, queries parallelized by Pen are improved, *i.e.*, they exhibit a lower execution time compared to FedX + Fedra. Parallelized queries with unimproved execution time are those that do not have a large number of intermediate results. Parallelization of such queries does not improve query performance, as their joins were not originally costly to evaluate.

As pointed in Figure 6, the number of parallelized queries is not constant within different versions the same federation, because the replication schema directly influences query parallelization. When this schema is not designed, as

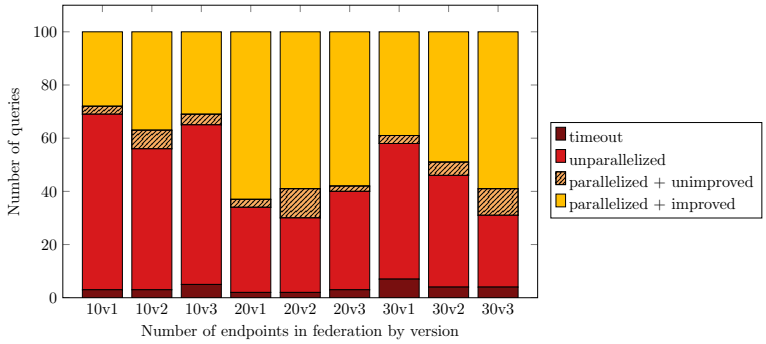


Fig. 6: Number of parallelized queries with FedX + Fedra + Pen.

in Linked Open Data, Pen creates parallelization where locality cannot be used by Fedra to optimize the query execution plan.

Number of transferred tuples Figure 7 summarizes the number of transferred tuples (*NTT*) in different federations. FedX + Fedra + Pen transfers the same amount of tuples as FedX + Fedra. This demonstrates that Pen does not deteriorate the reduction of transferred tuples provided by Fedra. Moreover, modifications performed on FedX to remove some synchronisation barriers do not introduce any difference between FedX + Fedra and FedX + Fedra + Pen in terms of number of transferred tuples and do not impact FedX + Fedra performance.

4.2 Pen with Triple Pattern Fragments

Evaluation Metrics: *i*) *Execution Time (ET):* is the elapsed time since the query is posed until the complete answer is produced. We used a timeout of 1800 seconds. *ii*) *Percentage of HTTP calls per server (PHC):* is, for a given server, the ratio between the number of HTTP calls received by the server and the total number of HTTP calls produced by the query.

Results presented for *ET* correspond to the average over three consecutive executions of our random queries.

Execution time Figure 8a summarizes the average execution time (*ET*) with the reference TPF client (1 server) and TPF + Pen(using 2 to 5 servers). Pen does not reduce the query execution time of SPARQL queries. As we are in a context were servers are not under a heavy load, they respond quickly to TPQs issued by the client. Therefore, parallelizing these remote calls does not have an impact on the query execution as they are already very cheap to execute.

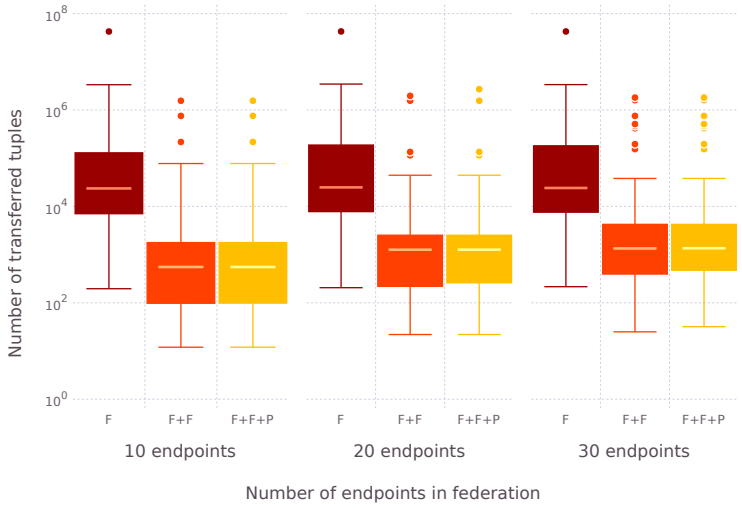


Fig. 7: Average number of transferred tuples with FedX (F), FedX + Fedra (F+F) and FedX + Fedra + Pen (F+F+P).

Repartition of HTTP calls Figure 8b summarizes the percentage of HTTP calls per server (*PHC*) with TPF + Penusing up to 5 servers. Pen is able to evenly distribute the load between servers, increasing the availability of each server. However, Pen only affects join processing, so the first triple pattern of a query will still be evaluated against the first server (E_1), which has a slightly heavier load than the others.

4.3 Synthesis

Experimental study results confirm that Pen can further increase the performance of join processing in presence of replicated fragments.

For federated SPARQL queries, execution time in average is lower with FedX + Fedra + Pen than with FedX or FedX + Fedra, and the reduced number of transferred tuples granted by Fedra is maintained. Answer completeness is not degraded. Pen is able to parallelize a significant number of queries in presence of replicated fragments and shows to be more efficient on larger federations. Query performance are significantly improved for queries with a large number of intermediate results, and the time to evaluate joins is reduced by taking advantage of parallel processing.

For Triple Pattern Fragments, execution time is not reduced by Pen, but the load is evenly balanced between all servers used, increasing the overall availability.

(a) Average execution time

(b) Average repartition of HTTP calls per server

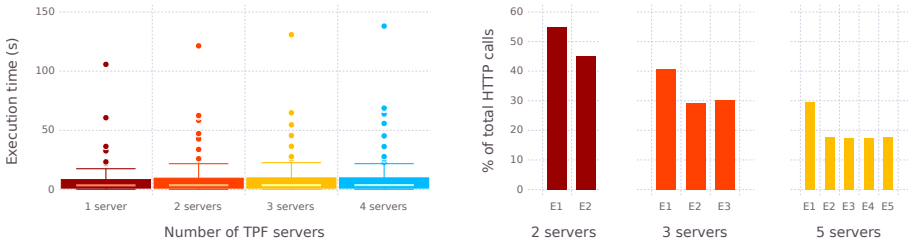


Fig. 8: Experimental results for TPF + Pen

5 Related Work

Fedra [13] is a replication-aware source selection that uses data locality produced by replicated fragments to enhance federated query engines performances. Fedra uses Union and BGP reductions to prune data sources and finds as many sub-queries that can be executed against the same endpoint as possible, leading to evaluation of local joins and a reduced number of transferred tuples. Pen uses replicated fragments differently. As seen in Section 2, Fedra prunes redundant endpoints that cannot be used to creates localities, whereas Pen uses these endpoints to create more opportunities of parallelization.

LILAC [14] is a replication-aware decomposer. Compared to Fedra, LILAC is able to reduce intermediate results by allocating a triple pattern to several endpoints. As for Fedra, Pen can reuse source selection performed by LILAC to introduce intra-operator parallelism.

Other existing sources selection techniques reduce the number of selected sources by a federated SPARQL query engine. BBQ [10] and DAW [16] use sketches to estimate the overlapping among sources, but they only operate on duplicated sources and not on replication itself. They do not provide information about replicated fragments that allow Pen to efficiently parallelize join processing.

Parallel join processing in distributed database systems has been the subject of significant investigation. Parallel nested loop algorithms have been investigated in [4,6], but they do not use replication for parallelization. Instead, replication is mostly used for fault tolerance and to locate data closer to their access points [11,15], improving query performance by reducing communication time. Pen does not use localities created by data redundancy, but opportunities of parallelization created by this redundancy.

Parallel join processing has been also studied in federated query engines. For instance, [1,17,8] propose parallel architectures for executing queries concurrently at different data sources. Anapsid [1] takes advantage of bushy query

execution plans to create *inter-operator parallelism*. FedX [17] implements *bound joins* in a distributed and highly parallelized environment where different subqueries can be executed at the endpoints concurrently. Pen creates intra-operator parallelism and proposes a more advanced parallel join processing using replication. Similar to FedX, subqueries are executed concurrently, but they are distributed between endpoints, increasing parallelization.

To our knowledge, none of existing federated query engines propose to take advantage of replicated data for join processing or propose a replication-aware parallel join operator.

The Triple Pattern Fragments (TPF) [18] propose to shift complex query processing from servers to clients to improve availability and reliability of servers, at the cost of performance. A low-cost triple pattern-based interface is deployed server-side, and a client-side algorithm decomposes a SPARQL query into triple patterns that are evaluated against this interface. Compared to TPF, Pen allows for the execution of a single query over different TPF servers that replicate the same dataset. This improves the load balancing by distributing the query processing over several servers instead of one.

6 Conclusions and Future Works

In this paper, we extended a replication-aware federated query engine and the reference TPF client with a new replication-aware parallel join operator Pen. Pen provides intra-operator parallelism relying on replicated data. In this way, Pen preserves properties of source-selection and query decomposition replication-aware federated query engines. We implemented Pen in both FedX and TPF. Evaluation results demonstrates that Pen improves significantly query performance, in terms of execution time for FedX and in terms of load balancing for TPF.

Pen is the first attempt to use replicated data to parallelize query processing in Linked Open Data and opens several perspectives.

First, we made the assumption that the load of the endpoints is uniform during query execution. We can leverage this hypothesis by making Pen adaptive to the performances of endpoints.

Second, we focused on a Nested Loop Join operator, we can also parallelize others operators such as Symmetric Hash-Join [20] used in Anapsid.

Acknowledgments. This work is partially supported through the FaBuLA project, part of the AtlanSTIC 2020 program.

References

1. Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: Anapsid: an adaptive query processing engine for sparql endpoints. In: International Semantic Web Conference. pp. 18–34. Springer (2011)

2. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of rdf data management systems. In: International Semantic Web Conference. pp. 197–212. Springer (2014)
3. Aluç, G., Ozsu, M., Daudjee, K., Hartig, O.: chameleon-db: a workload-aware robust rdf data management system. university of waterloo. Tech. rep., Tech. Rep. CS-2013-10 (2013)
4. Bitton, D., Boral, H., DeWitt, D.J., Wilkinson, W.K.: Parallel algorithms for the execution of relational database operations. ACM Transactions on Database Systems (TODS) 8(3), 324–353 (1983)
5. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: Sparql web-querying infrastructure: Ready for action? In: International Semantic Web Conference. pp. 277–293. Springer (2013)
6. DeWitt, D.J., Naughton, J.F., Burger, J.: Nested loops revisited. In: Parallel and Distributed Information Systems, 1993., Proceedings of the Second International Conference on. pp. 230–242. IEEE (1993)
7. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary rdf representation for publication and exchange (hdt). Web Semantics: Science, Services and Agents on the World Wide Web 19, 22–41 (2013)
8. Görlitz, O., Staab, S.: Splendid: Sparql endpoint federation exploiting void descriptions. In: Proceedings of the Second International Conference on Consuming Linked Data - Volume 782. pp. 13–24. COLD'11, CEUR-WS.org, Aachen, Germany, Germany (2010), http://dl.acm.org/citation.cfm?id=2887352.2887354
9. Görlitz, O., Staab, S.: Federated Data Management and Query Optimization for Linked Open Data, vol. 331, pp. 109–137. Springer, Heidelberg (2011)
10. Hose, K., Schenkel, R.: Towards benefit-based rdf source selection for sparql queries. In: Proceedings of the 4th International Workshop on Semantic Web Information Management. p. 2. ACM (2012)
11. Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys (CSUR) 32(4), 422–469 (2000)
12. Minier, T., Montoya, G., Skaf-Molli, H., , Molli, P.: Peneloop: Parallelizing federated sparql queries in presence of replicated fragments. In: Joint Proceedings of the 2nd RDF Stream Processing (RSP 2017) and the Querying the Web of Data (QuWeDa 2017) workshops. pp. 37–50. CEUR Workshop Proceedings (2017)
13. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Federated sparql queries processing with replicated fragments. In: International Semantic Web Conference. pp. 36–51. Springer International Publishing (2015)
14. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Decomposing federated queries in presence of replicated fragments. Web Semantics: Science, Services and Agents on the World Wide Web 42, 1 – 18 (2017), //www.sciencedirect.com/science/article/pii/S1570826816300580
15. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
16. Saleem, M., Ngomo, A.C.N., Parreira, J.X., Deus, H.F., Hauswirth, M.: Daw: Duplicate-aware federated query processing over the web of data. In: International Semantic Web Conference. pp. 574–590. Springer (2013)
17. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: International Semantic Web Conference. pp. 601–616. Springer (2011)
18. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple pattern fragments: A low-

cost knowledge graph interface for the web. Web Semantics: Science, Services and Agents on the World Wide Web 37, 184–206 (2016)
19. Wilcoxon, F.: Individual comparisons by ranking methods. In: Breakthroughs in Statistics, pp. 196–202. Springer (1992)
20. Wilschut, A.N., Apers, P.M.: Dataflow query execution in a parallel main-memory environment. Distributed and Parallel Databases 1(1), 103–128 (1993)