Eurostars Project

3DFed – Dynamic Data Distribution and Query Federation

Project Number: E!114681 **Start Date of Project:** 2021/04/01 **Duration:** 36 months

Deliverable 3.4
Final Report on the Dynamic Data Exchange

Dissemination Level	Public
Due Date of Deliverable	September 30, 2023
Actual Submission Date	September 30, 2023
Work Package	WP3, Automatic Data Distribution & Dynamic Exchange
Deliverable	D3.4
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	8

Abstract: In deliverable D3.3, we propose a method to do the dynamic data exchange between nodes and did initial evaluation based on the Semantic Web Dog Food dataset. The evaluation results certainly hinted at the usefulness of the proposed method. In this deliverable, we present more detailed results about the proposed dynamic data exchange solution.

History

Version	Date	Reason	Revised by
0.1	01/09/2023	Initial Template & Deliverable Structure	Mohammad Sajjadi
0.2	10/09/2023	Initial Draft	Asal Alikhani
0.3	23/09/2023	Issued for review	Muhammad Saleem
0.4	28/09/2023	Review	Milos Jovanovik
1.0	30/09/2023	Final Submission	Mohammad Sajjadi

Author List

Organization	Name	Contact Information
elevait GmbH & Co. KG	Asal Alikhani	asal.alikhani@elevait.de
University of Paderborn	Muhammad Saleem	saleem@informatik.uni-leipzig.de
elevait GmbH & Co. KG	Mohammad Sajjadi	mohammad.sajjadi@elevait.de
OpenLink Software	Milos Jovanovik	mjovanovik@openlinksw.com

Contents

1 Introduction

In previous deliverables D3.1 and D3.2 it was shown that partitioning techniques that take data locality into account, greatly minimize the inter-communication between partitions, thus potentially leading to better query runtimes. In T3.1, two novel workload-based RDF graph partitioning techniques – namely PCM and PCG – were developed by using different clustering techniques. The proposed approach makes use of the predicates' co-occurrences in the querying workload. The evaluation results show that the proposed techniques have better query runtime performances in comparison to other partitioning techniques.

The proposed algorithms are based on querying history, which changes with time. As such, we need to adapt the proposed distribution according to the change in querying history with time. To this end, we need a means for a dynamic data exchange between data nodes, according to the new querying workload. In deliverable D3.3, we proposed a method to do the dynamic data exchange between nodes and did initial evaluation based on the Semantic Web Dog Food dataset. The evaluation results certainly hinted at the usefulness of the proposed method.

In this deliverable, we present more detailed results about the proposed dynamic data exchange mechanism. While the previous evaluation results were based on the Koral engine and the Semantic Web Dog Food dataset, in this deliverable, we chose the DBpedia dataset due to it being more diverse in terms of the information it contains, as well as the opportunity if provides for a more diverse set of querying. The previous evaluation was based on the distributed triplestore Koral, but this evaluation is based on CostFed [6], a pure federated SPARQL query processing on top of multiple endpoints. The deliverable is generally shorter, as we mainly focus on the evaluation results.

In the next section, we first present an extended summary of the proposed approach, followed by the evaluation results and discussion.

2 Summarized Approach

We propose to collect a one week query log and make a new data distribution, which better reflects the workload from the logs. To perform this task, we follow the steps below.

- We get a querying workload W1 and perform predicate-clustering, using PCM or PCG. The corresponding cluster of predicates C1 is then assigned to physical partitions as discussed before.
- The triplestore is then used in practice for one week, and the new workload W2 is collected. We then perform the predicate-clustering C2 again, using PCM or PCG, using the new workload.
- We compare C1 and C2 for any changes, i.e., we check if the predicate clusters have changed in C2 with regards to C1. If there are no changes, we do not make any dynamic data distribution among the physical partitions. If there are changes, we carry on the required changes (insertion or deletion of triples) in the current physical partitions, to exactly reflect C2.
- We repeat these steps every week (or every two weeks), depending on the workload querying frequency.

3 Evaluation Setup

Here, we reused the exact evaluation setup discussed in D3.2, in D3.3, and in [1]. The reasons for choosing this evaluation setup are two-fold: (1) since our proposed techniques require query workloads, we wanted to use

real-world query workloads (i.e., collected from public SPARQL endpoints of real-world RDF datasets), and real-world RDF benchmarks, (2) we wanted our results to be comparable with the results presented in [1], in D3.2, and in D3.3.

3.1 Dataset

We use the DBpedia 3.5.1 dataset for evaluation. We chose this dataset because it was previously used in evaluating different partitioning techniques [1]. Furthermore, it is also used in many evaluations [4, 5, 7, 2] pertaining to SPARQL query processing. The dataset contains 42,849,609 RDF triples, 9,495,865 distinct subjects, 1,063 distinct predicates, 13,620,028 distinct objects, and a structuredness [4] value of 0.196. The real-world query logs for the DBpedia 3.5.1 dataset are freely available from LSQ dataset [8].

3.2 Workloads (train) and Benchmark (test) Queries

Our proposed approach needs a sample set of SPARQL queries to be used for training. We make use of the real-world queries from DBpedia 3.5.1 public SPARQL endpoints. The queries are available from LSQ v2.0 [8]. The required DBpedia 3.5.1 queries were fetched from the public LSQ endpoints `http://lsq.aksw.org/sparql` using the SPARQL query given in Listing 1.

```
# Count all SELECT queries from dbpedia divided weekly by execution
    timestamp in 2010.

PREFIX lsqv: <http://lsq.aksw.org/vocab#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX sd: <http://www.w3.org/ns/sparql-service-description#>

SELECT Distinct ?text #?timeStamp
From <http://lsq.aksw.org/dbpedia>
WHERE
{
    ?query lsqv:text ?text .
    ?query lsqv:hasRemoteExec ?re .
    ?re prov:atTime ?timeStamp .
    ?query lsqv:hasSpin ?spin .
    ?spin a <http://spinrdf.org/sp#Select> .
     FILTER(
       (YEAR(?timeStamp)= 2010 && MONTH(?timeStamp)= 5 && DAY(?
          timeStamp)>= 1 && DAY(?timeStamp)<(1+7))
       )
}
ORDER BY ?timeStamp
```

Listing 1: SPARQL query

Once we retrieved all queries from the LSQ endpoint, we divided them into a time slot of one week. Then we used the first week queries for training and the next week queries for testing, and so on. This is exactly the same process adopted in deliverable D3.3.

3.3 Partitioning Environment

In D3.3, we only used a clustered or distributed RDF storage environment, where the given dataset is distributed among n data nodes of a clustered triplestore. In this deliverable, we used the second partitioning environment, i.e., a purely federated environment, in which the dataset is distributed among multiple SPARQL endpoints

that are physically separated from each other and a federation engine is used to perform the query processing task. We used *CostFed* [6], a federated SPARQL query processing engine for federation over multiple SPARQL endpoints. We chose *CostFed* because it is developed in the context of the 3DFed project.

Number of Partitions. Inspired by D3.2, D3.3, [1] and [3], we generated 10 partitions of the selected datasets. Each partition was loaded into a separate Virtuoso triplestore and has a unique public SPARQL endpoint URL.

Selected RDF Graph Partitioning Technique. We used the PCM clustering algorithm because it has proven to be performing better than PCG in terms of clustering generation. In addition, it was also used in D3.3.

Performance Measures. We used Queries per Second (QpS), and the average query execution time to compare the performance of the proposed dynamic data exchange. We used a six minutes timeout for query execution of each query. We also measured the amount of dynamic data exchange between partitions using the Gini coefficient, defined as follows.

Definition 1 (Partitioning Shuffling) *Let n be the total number of partitions generated by a partitioning technique and $P_1, P_2, \ldots P_n$ be the set of these partitions, ordered according to the increasing size of the number of triples. The shuffling in partitions is defined as a Gini coefficient:*

$$b := \frac{2\sum_{i=1}^{n}(i \times |P_i|))}{(n-1) \times \sum_{j=1}^{n}|P_j|} - \frac{n+1}{n-1}, 0 \leq b \leq 1$$

Hardware and Software Specifications. The hardware and software configuration for our techniques is the same as in [1], i.e., all of our experiments are executed on a Ubuntu-based machine with Intel i7-11370H 3.30 GHz, 4 cores and 32GB of RAM. We conducted our experiments on local copies of Tenforce / Virtuoso (version 7.2.5) SPARQL endpoints. We used the default configurations of the CostFed engine.

4 Results

The goal of our evaluation is to show how the query runtime performance is improved with dynamic data partitioning. We also measured by what extent the data shuffling was performed by the proposed approach.

Train query	Partition 0	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5	Partition 6	Partition 7	Partition 8	Partition 9
W1	1452687	69699	840721	232624	41958	264940	0	10925705	0	29721158
W3	106	7899267	6766022	0	0	0	0	0	0	28884097
W5	27997	2107781	1724004	10925811	8173242	6998646	1629215	264940	1115104	10582752
W7	0	0	0	0	0	0	0	0	0	43549492
W9	8554361	840721	934182	10959549	218811	61578	738786	126458	9337706	11777340
W11	6766022	882679	11818035	252655	748228	1452687	8052658	570859	2024314	10981355

Table 1: Distributed triples in 10 partitions by PCM based on train queries.

Data Shuffling. Table 1 shows the number of triples assigned to each of the selected 10 partitions by the proposed algorithms after every two weeks. We can clearly see a high variations in the number of triples. This

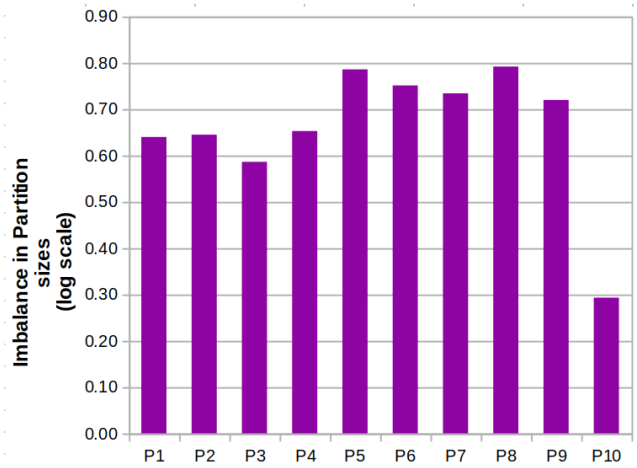


Figure 1: Partitioning shuffling within each partition.

means that a high data shuffling was required to perform more optimized query execution. The amount of data shuffling within each partition is given in Figure 1. A Gini coefficient value of 0 means highly balanced or no variation, and 1 means highly unbalanced. We can clearly see the value is higher than 0.5 for the majority of partitions, suggesting a high data shuffling (i.e., dynamic data exchange) was performed by the proposed algorithm.

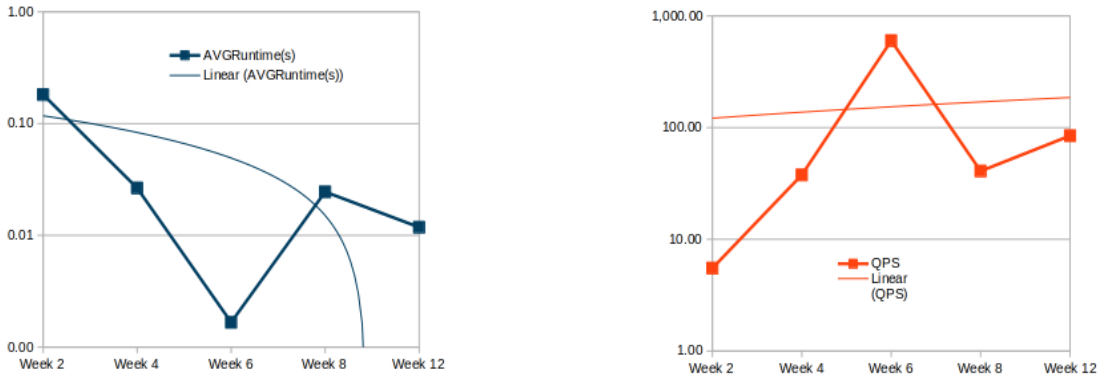
Average Query Runtime. Figure 2a shows a comparison of the average query runtime for 6 (W1 to W11) different partitions, as a result from dynamic data shuffling after two-week querying workloads. The smaller the runtime, the better the query execution performance. The decrease in the trend line suggests that query runtime is decreased with dynamic data distribution, while going from week 2 to week 12. This shows the effectiveness of the proposed dynamic data distribution. However, it can also be seen that the average runtime is even increased from week 6 to week 8. This means it is also possible that the proposed shuffling results into performance decrease. However, in majority of the cases, it is not the case.

Query per Second (QpS). Figure 2b shows a comparison of the QpS values for the 6 dynamic data shuffling and final partitions based on biweekly querying loads. The higher the QpS, the better the query runtime performance. Each query load is before its benchmark queries. Again, we can clearly see the QpS (in general) is improved with dynamic data distribution while going from week 2 to week 12. This shows the effectiveness of the proposed dynamic data distribution. However, we also see that the increase in QpS is not linear. For example the QpS of week 6 is better than week 8. The possible reason is that the test queries selected for this experiments might be more complex compared to the test queries in week 6. However, the increasing trend line suggests that the proposed approach was able to improve the query per second execution after dynamic data shuffling.

The results are summarised in Table 2 as well. There were some queries with no result for each week but the percentage is different and for week 10 none of the test queries had results. Sometimes this no-result error happens when we run many queries in a row. Therefore we considered only those queries which had some results.

5 Conclusion and Future Work

In this deliverable, we presented the final report on the dynamic data exchange. We made use of the six querying workloads from DBpedia and did dynamic data shuffling according to the new workloads. We used QpS and the average query runtime as two performance measures. The results clearly show the effectiveness of the proposed



(a) Average runtime.

(b) Queries per second.

Figure 2: Executing queries with Dynamic Data Exchange.

Train queries	No of queries	Test queries	Test query log size	Total Runtime(s)	AVG Runtime(s)	QPS
W1	63	W2	398	09.796	00.181	5.51
W3	11	W4	9999	00.053	00.027	37.74
W5	10000	W6	10000	00.010	00.002	600.00
W7	1661	W8	10000	00.098	00.025	40.82
W9	9995	W10	4494	00.000		
W11	9999	W12	9999	00.437	00.012	84.67

Table 2: Average runtime and query per second of running test query logs.

dynamic data partitioning. However, further detailed experiments are needed to draw solid conclusions.

References

[1] Akhter et al. An Empirical valuation of RDF Graph Partitioning Techniques. In *European Knowledge Acquisition Workshop*, 2018.

[2] Saleem et al. LSQ: The Linked SPARQL Queries Dataset. 2015.

[3] Saleem et al. A Fine-Grained Evaluation of SPARQL Endpoint Federation Systems. 2016.

[4] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. LargeRDFBench: A Billion Triples Benchmark for SPARQL Endpoint Federation. *Journal of Web Semantics*, 48:85–125, 2018.

[5] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. Feasible: A feature-based sparql benchmark generation framework. In *International Semantic Web Conference*, pages 52–69. Springer, 2015.

[6] Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation. *Procedia Computer Science*, 137:163–174, 2018.

[7] Schwarte et al. FedX: Optimization Techniques for Federated Query Processing on Linked Data. 2011.

[8] Claus Stadler, Muhammad Saleem, Qaiser Mehmood, Carlos Buil-Aranda, Michel Dumontier, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. LSQ 2.0: A Linked Dataset of SPARQL Query Logs. *Semantic Web*, (Preprint):1–23, 2022.