Undefined 1 (2019) 1–5
IOS Press

RDF Graph Partitioning: Techniques and Empirical Evaluation

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Adnan Akhter [a,*], Muhammad Saleem [a] and Axel-Cyrille Ngonga Ngomo [b]
[a] *AKSW, Leipzig, Germany*
E-mail: {lastname}@informatik.uni-leipzig.de
[b] *University of Paderborn, Germany*
E-mail: axel.ngonga@upb.de

Abstract.
Over the past few years, we have witnessed that the RDF data sources, both in numbers and volume have grown enormously. As the RDF datasets gets bigger, system's storage capacity becomes vulnerable and the need to improve the scalability of RDF storage and querying solutions arises. Partitioning of the dataset is one solution to this problem. There are various graph partitioning techniques exist. However, it is difficult to choose the most suitable (in terms of query performance) partitioning for a given RDF graph and application. To the best of our knowledge, there is no detailed empirical evaluation exists to evaluate the performance of these techniques. This paper presents an empirical evaluation of RDF graph partitioning techniques by using real-world datasets and real-world benchmark queries selected using the *FEASIBLE* benchmark generation framework. We evaluate the selected RDF graph partitioning techniques in terms of query runtime performances, partitioning time and partitioning imbalance. In addition, we also compare their performance with centralized storage solutions, i.e., no-partitioning at all. Our results show that the centralized storage of the complete datasets (no-partitioning) generally lead to better query runtime performance as compared to their partitioning. However, for specific cases the performance is improved with partitioning as compared to centralized solution. Hence, the general graph partitioning techniques may not lead to better performance when implied to RDF graphs. Therefore, clustered RDF storage solutions should take into account the properties of RDF and Linked Data as well as the expressive features of SPARQL queries when partitioning the given dataset among multiple data nodes.

Keywords: RDF Data, Graph Partitioning, Query Runtime Performance, Partitioning Imbalance, Partitioning Time

1. Introduction

Over the past few years, the Web of Data has increased significantly. Currently, the Linking Open Data (LOD)[1] cloud comprises around 150 billion triples from more than 2973 datasets. This includes several big datasets such as UniProt[2](over 10 billion triples) and Linked TCGA[3] (around 20 billion triples). To store and query such datasets efficiently has motivated a considerable amount of work on designing clustered triple-stores [6,7,10,11,12,13,19,21,22,27,28,33], i.e., solutions where data is partitioned among several data nodes. *Data partitioning*, by definition, is a process of logically and/or physically dividing a larger dataset into many smaller sub-datasets to facilitate better maintenance and access by improving system's availability,

*Corresponding author. E-mail: akhter@informatik.uni-leipzig.de
[1]LOD: `http://stats.lod2.eu/`
[2]UniProt: `http://www.uniprot.org/statistics/`
[3]TCGA: `http://tcga.deri.ie/`

query processing times and load balancing. There are many graph-data partitioning techniques employed by current clustered triplestores [28]. According to [14], a partitioning technique greatly affects the query runtime performance of data storage solutions. This leads to the research question of how to find the best suitable partitioning technique for a given use case. However, to the best of our knowledge, no detailed evaluation of different RDF graph partitioning techniques has been undertaken that would evaluate efficiencies of different partitioning techniques in terms of query runtime performance, partitioning imbalance and scalability.

Our previous work [2] has addressed this research question by presenting an evaluation of seven different graph partitioning techniques implied to RDF graphs. We compare these partitioning techniques in terms of query runtime performance, partitioning time and balanced load generation. In this paper we present an extended version of this work. We provide a more detailed explanation of the results. In particular, we wanted to investigate whether the general graph partitioning techniques can lead to better performance with implied to RDF graphs. To this end, we compared the performance of the selected seven partitioning techniques with no-partitioning solution.

Our overall contributions are as follows:

1. We present a comparison of seven – Predicate-Based, Subject-Based, Hierarchical, Horizontal, Total Communication Volume Minimization (TCV-Min), Min-Edgecut and Recursive-Bisection – RDF graph partitioning techniques with no-partitioning centralised storage solution in two different evaluation setups. :[4] (1) clustered RDF storage solutions in which data is distributed among different data nodes within the same machine, and (2) purely federated solutions in which data is distributed among several physically separated machines and a federation engine is used to do the query processing task.
2. To make our comparison more realistic, we make use of two real-world datasets (i.e., Semantic Web Dog Food and DBpedia), and real-world users' queries log which are generated by FEASIBLE [5] [25].
3. We used different performance measures such as query runtime performance, total partitioning time, size variations of the partitions, and number of sources selected. In addition, we performed T-Test to show significant differences in the runtime performances achieved by the selected solutions.
4. Our evaluation results revealed interesting insights such as: the performance was degraded (in general) by partitioning the data as compared to the centralized storage solution without any partitioning, the query runtime performances are greatly affected by the underlying partitioning technique, the widely used subject/Predicate-Based partitioning may not be the best candidates in all situations, and the partitioning technique that leads to the smaller number of sources selected usually leads to the better runtime performances in purely federated environment etc.

All of the data, source code, and results presented in this evaluation are available at `https://github.com/dice-group/rdf-partitioning`.

The rest of the paper is organized as follows: Section 2 defines the key concepts necessary to understand this work. Section 3 explains graph partitioning techniques used in this work. Section 4 shows complete evaluation setup followed by the evaluation results. Section 5 covers some of the previous work related to the graph partitioning. Finally, section 6 presents our conclusion.

2. Preliminaries

In this section, we define the key concepts necessary to understand the subsequent sections of this work. Some of the definitions are adopted from [16,26].

Definition 1. ***RDF Triple:*** *Assuming there are pairwise disjoint infinite sets I, B, and L (IRIs, Blank nodes, and Literals, respectively), a triple (s, p, o)* $\in (I \cup B) \times I \times (I \cup B \cup L)$ *is called an RDF triple. In this triple, s is the subject, p the predicate, and o the object.*

Definition 2 (RDF Graph)**.** *An RDF graph* G *is a set of RDF triples and can be modelled as a labelled graph (V,E,*λ*), where* $V = \{v | \exists s, p, o : (v, p, o) \in G \vee (s, p, v) \in G\}$ *is the set of vertices,* $E \subseteq \{(s, o) | (s, p, o) \in G\}$ *is the set of edges between the vertices and* $\lambda(s, o) = p$ *if (s, p, o)* $\in G$ *is the edge labeling function of* G*.*

Definition 3 (RDF Graph Partitioning Problem)**.** *Given an RDF graph* $G = (V, E)$*, divide* G *into* n *sub-graphs* $G_1, \ldots G_n$ *such that* $G = (V, E) = \bigcup_{i=1}^{n} G_i$*, where* V *is the set of all vertices and* E *is the set of all edges in the graph.*

[4] Explained in detailed in Section 4.1.2

[5] A SPARQL benchmark generation framework from queries log

Now we define the overall rank score (ref. Figure 7) and the partitioning imbalance (ref. Figure 8) we used as performance metrics to compare the selected partitioning techniques.

Definition 4 (Rank Score)**.** *Let t be the total number of partitioning techniques and b be the total number of benchmark executions used in the evaluation. Let $1 \leq r \leq t$ denote the rank number and $O_p(r)$ denote the occurrences of a partitioning technique p placed at rank r. The rank score of the partitioning technique p is defined as follows:*

$$s := \sum_{r=1}^{t} \frac{O_p(r) \times (t-r)}{b(t-1)}, 0 \leq s \leq 1$$

In our evaluation, we have a total of seven partitioning techniques (i.e., t = 7) and 10 benchmarks executions (b = 10, 4 benchmarks by FedX, 4 benchmarks by SemaGrow, and 2 benchmarks by Koral).

Definition 5 (Partitioning Imbalance)**.** *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 number of triples. The imbalance in partitions is defined as Gini coefficient:*

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

The remaining of the definitions are related to the SPARQL queries used for benchmarking (see Table 1). We assume that the reader is familiar with the basic concepts of SPARQL, including the notions of a triple pattern and basic graph pattern (BGP).[6]

We represent any basic graph pattern (BGP) of a given SPARQL query as a *directed hypergraph* (DH) [26], a generalization of a directed graph in which a hyperedge can join any number of vertices. In our specific case, every hyperedge captures a triple pattern. The subject of the triple becomes the source vertex of a hyperedge and the predicate and object of the triple pattern become the target vertices. For instance, the query (Figure 1) shows the hypergraph representation of a SPARQL query. Unlike a common SPARQL representation where the subject and object of the triple pattern are connected by an edge, our hypergraph-based representation contains nodes for all three components of the triple patterns. As a result, we can capture joins that involve predicates of triple patterns. Formally, our hypergraph representation is defined as follows [26]:

[6]See `https://www.w3.org/TR/sparql11-query/` for the corresponding definitions.

```
SELECT DISTINCT * WHERE {
?drug :description ?drugDesc .
?drug :drugType :smallMolecule .
?drug :keggCompoundId ?compound .
?enzyme :xSubstrate ?compound .
?chemReac :xEnzyme ?enzyme .
?chemReac :equation ?chemEq .
?chemReac :title ?reacTitle
}
```

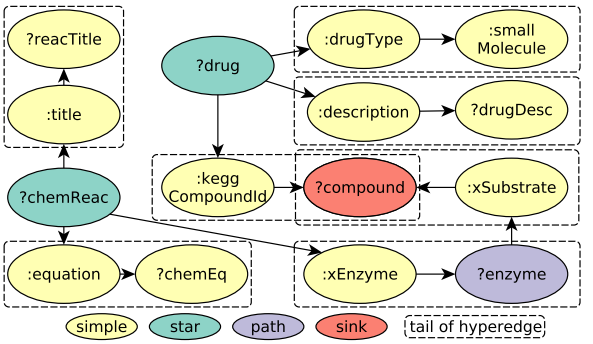


Fig. 1.: Directed hypergraph representation of a SPARQL query. Prefixes are ignored for simplicity. (Example taken from [20]).

Definition 6 (Directed Hypergraph of a BGP)**.** *The hypergraph representation of a BGP B is a directed hypergraph $HG = (V, E)$ whose vertices are all the components of all triple patterns in B, i.e., $V = \bigcup_{(s,p,o)\in B}\{s, p, o\}$, and that contains a hyperedge $(S, T) \in E$ for every triple pattern $(s, p, o) \in B$ such that $S = \{s\}$ and $T = (p, o)$.*

The representation of a complete SPARQL query as a DH is the union of the representations of the query's BGPs. Based on the DH representation of SPARQL queries, we can define the following features of SPARQL queries:

Definition 7 (Join Vertex)**.** *For every vertex $v \in V$ in such a hypergraph we write $E_{\text{in}}(v)$ and $E_{\text{out}}(v)$ to denote the set of incoming and outgoing edges, respectively; i.e., $E_{\text{in}}(v) = \{(S, T) \in E \,|\, v \in T\}$ and $E_{\text{out}}(v) = \{(S, T) \in E \,|\, v \in S\}$. If $|E_{\text{in}}(v)| + |E_{\text{out}}(v)| > 1$, we call v a* join vertex.

Definition 8 (Join Vertex Degree)**.** *Based on the DH representation of the queries the join vertex degree of*

a vertex v is $JVD(v) = |E_{in}(v)| + |E_{out}(v)|$, where $E_{in}(v)$ resp. $E_{out}(v)$ is the set of incoming resp. outgoing edges of v.

Definition 9 (Triple Pattern Selectivity)**.** *Let tp_i be a triple pattern of a SPARQL query Q and D be a dataset. Furthermore, let N be the total number of triples in D and $Card(tp_i, D)$ be the cardinality of tp_i w.r.t. D, i.e., total number of triples in D that matches tp_i, then the selectivity of tp_i w.r.t. D denoted by $Sel(tp_i, D) = Card(tp_i, D)/N$.*

3. RDF Graph Partitioning

In this section, we explain commonly used [16,18, 28,24] graph partitioning techniques by using a sample RDF graph shown in figure 2.

3.1. Horizontal Partitioning:

This is perhaps the most simplest form of partitioning which is adopted from [24] in which the whole dataset is divided into n numbers of partitions of each size. Imagine we have a dataset of 100 triples and we want to split them in ten partitions, then, the first ten triples will go to the first partitions, later two will go to the second and so on till the last ten triples go to the final partition.

Assume T be the set of all RDF triples in a dataset and n be the required number of partitions and we want to apply horizontal technique to this dataset. As explained before, the first $|T|/n$ triples will go in partition 1, the next $|T|/n$ triples will go in partition 2 and so on. In the example given in figure 2, the triples 1-4 will be assigned to the first partition (green), triples 5-8 will be assigned to the second partition (red), and triples 9-11 will be assigned to the third partition (blue).

In order to determine whether horizontal partitioning will increase overall performance of the system then we need to examine the ways in which the input queries access the dataset. If there is a significant access of a group of rows together, then Horizontal partitioning may make sense.

3.2. Subject-Based Partitioning

Subject-Based partitioning partitions a given dataset based on their subject's hash code. The idea is to group all the triples with same subject and assign them into one partition based on a hash value computed on their subjects modulo [16]. However, due to modulo operation this technique may result in high partitioning imbalance.

Consider our motivating examples given in figure 2. Three triples i.e., 3, 10 and 11 are assigned to the red partition due to the fact that these triples have same subjects, triple 7 is assigned to the blue partition, and the rest of the triples are assigned to partition green. This shows the partitioning imbalance which is 3:1:7 in our case.

In order to determine whether Subject-Based partitioning will increase overall performance of the system then again, we need to examine the ways in which the input queries access the dataset. If there is a significant access of the triples with same subjects, then Subject-Based partitioning may make sense.

3.3. Predicate-Based Partitioning

Similar to Subject-Based, Predicate-Based partitioning partitions a given dataset based on their predicate's hash code. The idea is to group all the triples with same predicate and assign them into one partition based on a hash value computed on their subjects modulo. However, due to modulo operation this technique may also result in high partitioning imbalance.

Consider our motivating examples given in figure 2. All the triples containing predicate `p1` and `p4` are assigned to the red partition, triples with predicate `p2` are assigned to the green partition, and all triples with predicate `p3` are assigned to the blue partition. Again, we can see a clear partitioning imbalance 5:4:2 in our case.

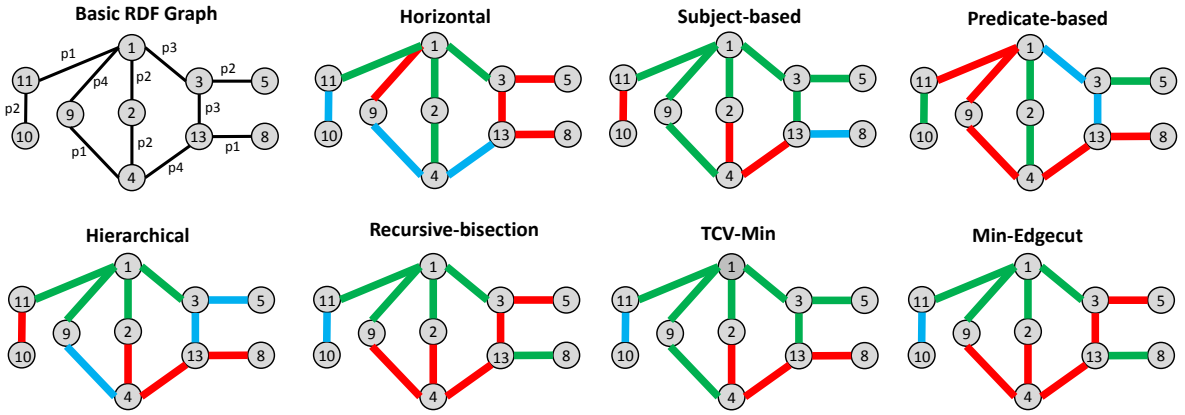
In order to determine whether Predicate-Based partitioning will increase overall performance of the system then again, we need to examine the ways in which the input queries access the dataset. If there is a significant access of the triples with same predicates, then Predicate-Based partitioning may make sense.

3.4. Hierarchical Partitioning

This partitioning is inspired by the assumption that IRIs have path hierarchy and IRIs with a common hierarchy prefix are often queried together [16]. This partitioning is based on extracting path hierarchy from the IRIs and assigning triples having the same hierarchy prefixes into one partition. For instance, the extracted path hierarchy of "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" is "org/w3/www/1999/02/22-rdf-

```
@prefix hierarchy1: <http://first/r/> . @prefix hierarchy2: <http://second/r/> .
@prefix hierarchy3: <http://third/r/> . @prefix schema: <http://schema/> .
hierarchy1:s1          schema:p1          hierarchy2:s11 .  #Triple 1
hierarchy1:s1          schema:p2          hierarchy2:s2 .   #Triple 2
hierarchy2:s2          schema:p2          hierarchy2:s4 .   #Triple 3
hierarchy1:s1          schema:p3          hierarchy3:s3 .   #Triple 4
hierarchy3:s3          schema:p2          hierarchy1:s5 .   #Triple 5
hierarchy3:s3          schema:p3          hierarchy2:s13 .  #Triple 6
hierarchy2:s13         schema:p1          hierarchy2:s8 .   #Triple 7
hierarchy1:s1          schema:p4          hierarchy3:s9 .   #Triple 8
hierarchy3:s9          schema:p1          hierarchy2:s4 .   #Triple 9
hierarchy2:s4          schema:p4          hierarchy2:s13 .  #Triple 10
hierarchy2:s11         schema:p2          hierarchy1:s10 .  #Triple 11
```

(a) An example RDF triples



(b) Graph representation and partitioning. Only node numbers are shown for simplicity.

Fig. 2.: Partitioning an example RDF into three partitions using different partitioning techniques. Partitions are highlighted in different colors.

syntax-ns/type". Then, for each level in the path hierarchy (e. g., “org", “org/w3", “org/w3/www", ...) it computes the percentage of triples sharing a hierarchy prefix. If the percentage exceeds an empirically defined threshold and the number of prefixes is equal to or greater than the number of required partitions at any hierarchy level, then these prefixes are used for the hash-Based partitioning on prefixes. In comparison to the hash-Based subject or predicate partition, this technique requires a higher computational effort to determine the IRI prefixes on which the hash is computed. In our motivating example given in figure 2, all the triples having `hierarchy1` in subjects are assigned to the green partition, triples having `hierarchy2` in subjects are assigned to the red partition, and triples having `hierarchy3` in subjects are assigned to the blue partition.

3.5. k-way Partitioning

k-way Partitioning is a multilevel graph algorithm as described in [18]. The structure of k-way algorithm consists of three phases. The first phase is coarsening where graph G = (V,E) is first coarsened down to a few thousand vertices. The second phase is partitioning where this much smaller graph is partitioned. The third and final phase is uncoarsening where this partitioned graph is projected back towards the original graph.

3.5.1. Coarsening Phase

The first phase is coarsening the graph, in which a sequence of smaller graphs $G_1, G_2, ..., G_m$ is generated from the input Graph $G_0 = (V_0, E_0)$ in such a way that $|V_0| > |V_1| > |V_2| > ... > |V_m|$.

3.5.2. Partitioning Phase

Here we are making use of three partitioning techniques out of k-way algorithm

- Recursive-Bisection Partitioning: In Recursive-Bisection, computation of a 2-way partition P_m of the graph G_m takes place, such that V_m is split into two parts and each part contains half of the vertices.
  In our motivating example given in figure 2, triples (1,2,4,7,8) are assigned to the green partition, triples (3,5,6,9,10) are assigned to the red partition, and only triple 11 is assigned to the blue partition.

- TCV-Min Partitioning: The total communication volume of a block V_i of a given graph G_m is defined as $\sum_i comm(V_i)$. TCV-Min partitioning minimizes the *total communication volume* [4].
  In our motivating example given in figure 2, triples (1,2,4,5,6,8,9) are assigned to the green partition, triples (3,7,10) are assigned to the red partition, and only triple 11 is assigned to the blue partition.

- Min-Edgecut Partitioning: Unlike TCV-Min, the objective of Min-Edgecut partitioning is to partition the vertices in such a way that the number of edges connected to them are minimized. In our motivating example given in figure 2, triples (1,2,4,7,8) are assigned to the green partition, triples (3,5,6,9,10) are assigned to the red partition, and only triple 11 is assigned to the blue partition.

3.5.3. Uncoarsening Phase

The third and last phase is uncoarsening the partitioned graph. In this phase the partition P_m of G_m is projected back to G_0 by passing through the intermediate partitions $P_{m-1}, P_{m-2}, ..., P_1, P_0$.

4. Evaluation and Results

This section consists of two parts. In the first part we will present our evaluation setup including hardware and software configurations. In the second part, we will discuss evaluation results.

4.1. Evaluation Setup

4.1.1. Hardware and software configuration:

We used Ubuntu-based machine with intel Xeon 2.10 GHz, 64 cores and 512GB of RAM for our experiments. We conducted our experiments on local copies of Virtuoso (version 7.1) SPARQL endpoints. To create TCV-Min partitions, Min-Edgecut partitions and Recursive-Bisection partitions, we used METIS 5.1.0.dfsg-2 [7]. FedX and SemaGrow are used with default configurations. We made only one change in Koral's default configuration (the number of slaves were changed from 2 to 10).

4.1.2. Partitioning Environments:

To evaluate the selected RDF graph partitioning techniques, we used two distinct partitioning environments.

Clustered RDF Storage Environment: Figure 3 shows a very generic master-slave architecture used in our clustered environment. In this environment, the give RDF dataset is distributed among several slaves (10 in our case) within the same machine as part of a single RDF storage solution. The master is responsible for task assignment. RDF storage and query processing tasks are performed by the slaves. There are many RDF storage solutions [6,7,10,11,12,13,19,21,22,27,28,33] that employ this architecture. We chose *Koral* [16] in our evaluation. The reason for choosing this platform was because it is well-integrated with the famous RDF partitioning system *METIS* [18], it is a state-of-the art distributed RDF store, and it allows the data partitioning strategy to be controlled.

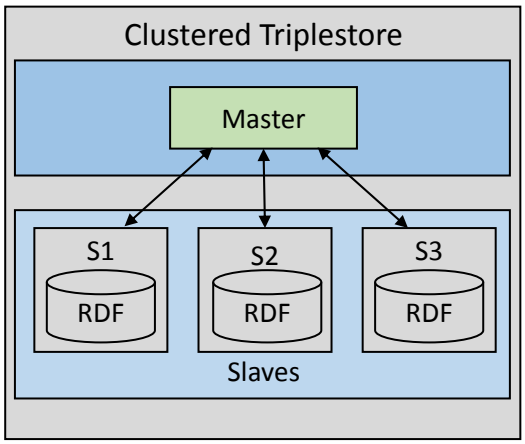


Fig. 3.: Clustered Architecture

Purely Federated Environment: In purely federated environment, the given RDF data is distributed across several machines, which are physically separated (10

[7] http://glaros.dtc.umn.edu/gkhome/metis/metis/download

in our case). There is a federation engine which is responsible for query processing task. In figure 4, the two main components (i.e., SPARQL endpoints and the federation engine) of this architecture is shown. Figure 4 also shows the general steps involved in a SPARQL query processing, in which, first a SPARQL query is being parsed to obtain the individual triple patterns in the query. In the next step, sources are selected, for which the goal is to identify the set of relevant data sources for the query (endpoints in our case). Based on the information obtained using the source selection, the federator splits the input query into multiple sub-queries. Optimizer generates an optimized sub-query execution plan and the sub-queries are forwarded to the corresponding data sources. Integrator integrates the results of the sub-queries. Finally, the integrated results are returned to the agent that issued the query. Many SPARQL endpoint federation engines [29,5,31,1,8] abide by this architecture. We chose *FedX* [29] and *SemaGrow* [5] in our evaluation. The main reason for choosing these two federation engines is because they are different in terms of query execution plans. SemaGrow is an cost-based index-assisted federation engine, while FedX is a heuristic-based index free SPARQL endpoint federation engine. Since, the query runtime performances is greatly affected by the query execution plan, therefore in order to employ the different query planners we chose two different federation engines.

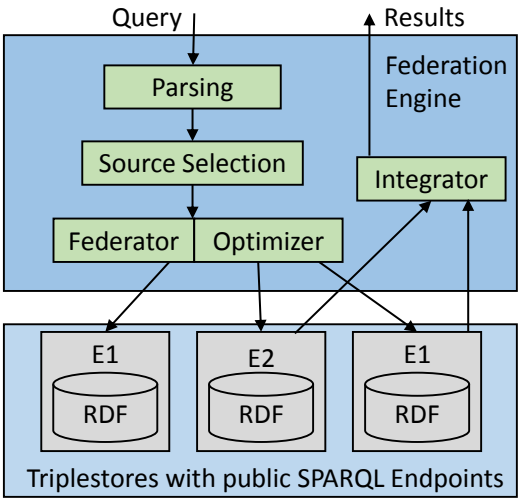


Fig. 4.: Physically Federated Architecture

4.1.3. Datasets:

We wanted to use real-world RDF datasets to evaluate our selected partitioning techniques. For this purpose, we used two real-world datasets, *Semantic Web Dog Food (SWDF) and* DBpedia 3.5.1. The reason for choosing SWDF and DBpedia is that they are used by the *FEASIBLE* [25], which is, a SPARQL benchmark generation framework to generate customized SPARQL benchmarks from the queries log of the underlying datasets. Another reason for picking these two datasets, is their sizes which are massively different in their high-level statistics: the DBpedia 3.5.1 contains 232,536,510 triples, 18,425,128 distinct subjects, 39,672 distinct predicates, and 65,184,193 distinct objects while SWDF contains 304,583 triples, 36,879 distinct subjects, 185 distinct predicates, and 95,501 distinct objects.

4.1.4. Benchmark Queries:

The benchmark queries are generated using FEASIBLE: **(1) SWDF BGP-only**, it contains a total number of 300 BGP-only SPARQL queries from the queries log using the SWDF dataset. Each query contain only single BGP, the other SPARQL features such as `OPTIONAL`, `ORDER BY`, `DISTINCT`, `UNION`, `FILTER`, `REGEX`, aggregate functions, `SERVICE`, property paths etc. are not used, **(2) SWDF fully-featured**, it contains a total number of 300 queries from the queries log using the SWDF dataset. These queries are not only single BGPs, also they may include one or more SPARQL query features (e.g., the above mentioned). **(3) DBpedia BGP-only**, it contains a total number of 300 BGP-only SPARQL queries from the queries log using the DBpedia dataset. Each query contain only single BGP. The the above mentioned features are not included. **(4) DBpedia fully-featured**, it contains a total number of 300 queries from the queries log using the SWDF dataset. These queries are not only single BGPs, also they may include one or more SPARQL query features (e.g., the above mentioned). In total, we used 1200 SPARQL queries for our evaluation selected from two different datasets. Note that we only used BGP-only benchmarks with Koral since it does not support many of the SPARQL features used in the fully-featured SPARQL benchmarks.

Table 1 shows detailed features of all aforementioned benchmark queries. Previous works [9,25,23,3] on SPARQL benchmarking show that these are important features to be considered in SPARQL querying benchmarking. Thus, a SPARQL benchmark should have sufficient diversity in these features across the different SPARQL queries included in the benchmark. Recent benchmark analysis [20] shows that the benchmark

generated by *FEASIBLE* (used in our evaluation) have the highest diversity score.

4.1.5. Number of partitions:

Inspired by [24], we created 10 partitions for each of the partitioning technique using both of the selected datasets (10 partitions for SWDF and 10 partitions for DBpedia). In Koral, we made 10 slaves each containing one partition. Both in FedX and SemaGrow, we used 10 Linux-based Virtuoso 7.1 SPARQL endpoints, each containing one partition.

4.1.6. Performance measures:

We used six performance measures to benchmark the selected partitioning techniques – partition generation time, query runtime performances, number of timeout queries (180 seconds was selected as the timeout time for each query execution [25]), overall ranking of partitioning techniques, partitioning imbalance among the generated partitions and number of sources selected for the complete benchmark execution in a purely federated environment. Additionally, The Spearman's rank correlation coefficients is also measured to ascertain the correlation between the sources selected and the query runtime in a purely federated environment.

4.2. Evaluation Results

4.2.1. Partition Generation Time:

The total time taken to generate 10 partition by each partitioning technique for both the datasets is shown in figure 5.

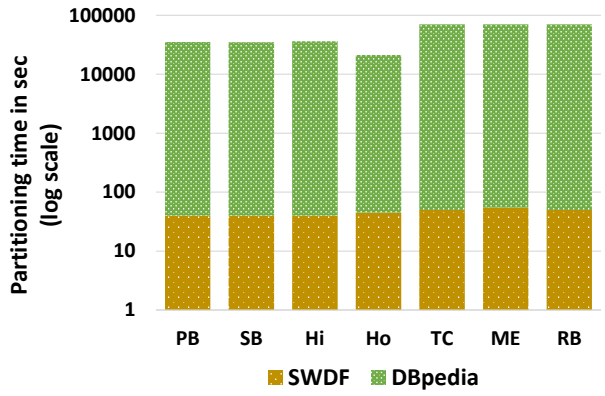


Fig. 5.: Time taken for the creation of 10 partitions.(**PB** = Predicate-Based, **SB**= Subject-Based, **Hi**= Hierarchical, **Ho** = Horizontal, **TC** = TCV-Min, **ME** = Min-Edgecut, **RB** = Recursive Bisection).

The Horizontal partitioning technique requires the smallest overall time followed by Subject-Based, Predicate-Based, Hierarchical, TCV-Min, Recursive-Bisection, and Min-Edgecut, respectively. The reason for the three k-way techniques, i.e., TCV-Min, Recursive-Bisection, and Min-Edgecut being the most time consuming in terms of partitions generation is due the their complexity in terms of the time required to perform the three-phase operation (coarsening, partitioning, and uncoarsening). After the k-way techniques, Hierarchical partitioning took most time, this is due to the extra time required to compute path hierarchies before hash function is applied. Predicate-Based and Subject-Based partitioning are the next in terms of most time consuming techniques, since both techniques, i.e., Predicate-Based and Subject-Based partitioning simply traverse each triple in the dataset and apply hash functions on the subject or predicate of the triple, therefore there is a very little difference in in total time taken to generate the partitions. The reason for Horizontal partitioning for being the least overall time consuming technique, is due to it's simplicity: it creates a range of triples and assigns them to the desired partitions without any further computations.

4.2.2. Query runtime performances:

Query runtime performances obtained by using all of our selected partitioning techniques is one of the most important result. To achieve this, we make use of all 300 queries to get (a) benchmarks execution time (including timeout queries) and (b) average query execution times (excluding timeout queries) to encapsulate the runtime performances of our partitioning techniques. Results of both, i.e., benchmarks execution time and average query execution times are taken separately for all three partitioning environments (FedX, SemaGrow and Koral). The former performance metric is measured by executing all 300 queries from each benchmark over the partitions created by the selected partitioning techniques and calculated total time taken to execute all 300 benchmark queries. An extra 180 seconds is added for each timeout query to the total benchmark execution time. For the performance metric of average query execution times, only those queries which are successfully executed within the timeout limit (180 seconds) are considered. In figure 6, the query runtime performances obtained by our selected techniques pertaining to the two aforementioned query execution metrics are presented. In figure 6(a), total execution time of complete benchmarks for the selected partitioning techniques using FedX federation engine

		SWDF		DBpedia	
		BGP-only	Fully Featured	BGP-only	Fully Featured
	#Queries	300	300	300	300
Forms (%)	SELECT	100	100	100	100
	ASK	0	0	0	0
	CONSTRUCT	0	0	0	0
	DESCRIBE	0	0	0	0
Clauses (%)	UNION	0	32.66	0	42
	DISTINCT	0	59	0	49
	ORDER BY	0	24.66	0	30.66
	REGEX	0	3	0	17.66
	LIMIT	0	40.66	0	36
	OFFSET	0	19.33	0	11.66
	OPTIONAL	0	34	0	35.66
	FILTER	0	25.66	0	59.33
	GROUP BY	0	25.33	0	0
Results	Min	1	1	1	1
	Max	10924	98732	1406396	1406396
	Mean	114.52	984.99	74033.36	39512.08
	S.D.	878.51	6421.84	186541.78	169502.32
BGPs	Min	1	1	1	1
	Max	1	14	1	14
	Mean	1	2.88	1	4.04
	S.D.	0	2.84	0	4.35
TPs	Min	1	1	1	1
	Max	5	14	6	18
	Mean	1.50	3.61	2.09	5.85
	S.D.	0.53	2.62	1.18	4.70
JV	Min	0	0	0	0
	Max	1	3	6	11
	Mean	0.49	0.71	0.92	1.57
	S.D.	0.50	0.81	1.13	2.76
MJVD	Min	0	0	0	0
	Max	5	4	5	11
	Mean	0.99	1.21	1.33	1.59
	S.D.	1.02	1.13	1.34	1.82
MTPS	Min	0	0	0	1
	Max	0.03	1	0.66	1
	Mean	3.92E-4	0.22	0.05	0.03
	S.D.	0.003	0.22	0.14	0.11
Runtime	Min	3	3	1	2
	Max	731	8008	56041	56041
	Mean	21.29	255.10	3786.02	2015.98
	S.D.	49.10	686.23	9364.26	7658.02

Table 1: Features of our benchmark queries **BGP** = Basic Graph Pattern, **TPs** = Triple Patterns, **JV** = Join Vertices, **MJVD** = Mean Join Vertices Degree, **MTPS** = Mean Triple Pattern Selectivity, **S.D.** = Standard Deviation). Runtime(ms)

is shown. The combined benchmarks execution result (all 4 benchmarks) shows that, No-Partitioning has consumed the least time (3579.8 seconds), followed by Horizontal (26538.7 seconds), Recursive-Bisection (26962.6 seconds), Subject-Based (28629.3 seconds), TCV-Min (28739.9 seconds), Hierarchical (28867.5 seconds), Min-Edgecut (30482.8 seconds) and Predicate-Based (33864.2 seconds), respectively. The total benchmark execution time of the individual benchmarks (i.e., two from SWDF and two from DBpedia3.51) can be seen from the bar stacked graphs directly. In figure 6(b), average query execution times for the selected partitioning techniques using FedX federation engine is shown. The combined average query execution result (all 4 benchmarks) shows that, No-Partitioning has the smallest average query runtime (2.99596), followed by Recursive-Bisection (5.020557271 seconds), Min-Edgecut (5.4330126 seconds), TCV-Min (5.4456308 seconds), Horizontal (5.4801338 seconds), Hierarchical (6.0390115 seconds), Subject-Based (6.5591146 seconds) and Predicate-Based (8.3071525 seconds), respectively.

In figure 6(c), total execution time of complete benchmarks for the selected partitioning techniques using SemaGrow federation engine is shown. The combined average query execution result (all 4 benchmarks) shows that, No-Partitioning has consumed the least time (3579.8 seconds), followed by Predicate-Based (27227.9 seconds), TCV-Min (28772.8 seconds), Hierarchical (28921.6 seconds), Recursive-Bisection (29983.9 seconds), Subject-Based (30012.5 seconds), Min-Edgecut (30807.5 seconds) and Horizontal (31145.9 seconds), respectively. In figure 6(d), average query execution times for the selected partitioning techniques using SemaGrow federation engine is shown. The combined average query execution result (all 4 benchmarks) shows that, Predicate-Based has the smallest average query runtime (2.857210203 seconds) followed by No-Partitioning (2.99596 seconds), Subject-Based (5.393390726 seconds), Hierarchical (5.349322361 seconds), Horizontal (7.077052279 seconds), TCV-Min (4.024567032 seconds), Min-Edgecut (5.850084384 seconds) and Recursive-Bisection (5.535637211 seconds) respectively.

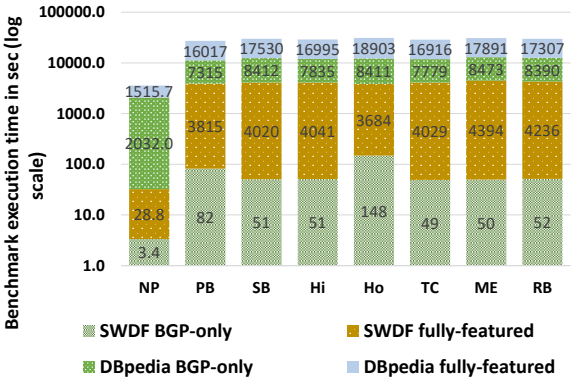
We now present the joint result of *purely federated environment* by combining the results of FedX and SemaGrow (since they both represent purely federated environment). The overall result of all 4 benchmarks using both, i.e., FedX and SemaGrow shows that, No-Partitioning has consumed the least time (3579.8 seconds), followed by Recursive-Bisection (28473.233 seconds), TCV-Min (28756.337 seconds), Horizontal (28842.264 seconds), Hierarchical (28894.5275 seconds), Subject-Based (29320.9305 seconds), Predicate-Based (30546.0905 seconds) and Min-Edgecut (30645.1825 seconds), respectively. The combined result of average query runtimes of all 4 benchmarks using both, i.e., FedX and SemaGrow shows that, No-Partitioning has the smallest average query runtime (2.99596), followed by TCV-Min (5.278097241 seconds), Recursive-Bisection (5.278097241 seconds), Predicate-Based (5.582181367 seconds), Min-Edgecut (5.641548479 seconds), Hierarchical (5.694166918 seconds), Subject-Based (5.976252639 seconds) and Horizontal (6.27859305 seconds), respectively.

In figure 6(e), total execution time of complete benchmarks for the selected partitioning techniques using Koral is shown. The combined average query execution result (2 benchmarks) shows that, the Min-Edgecut consumed the least time (16839 seconds), followed by No-Partitioning (21800 seconds), Subject-Based (34643 seconds), TCV-Min (40110 seconds), Predicate-Based (45170 seconds), Horizontal (45602 seconds), Hierarchical (53539 seconds) and Recursive-Bisection (55798 seconds), respectively.

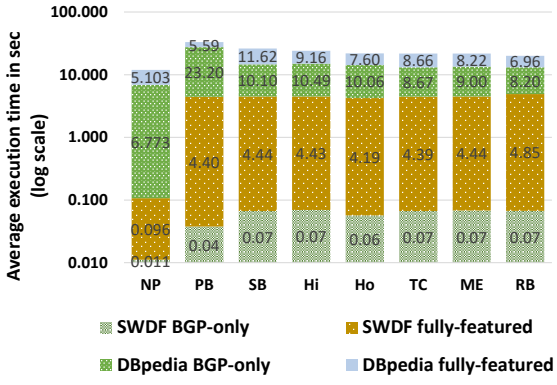
In figure 6(f), average query execution times for the selected partitioning techniques using Koral is shown. The combined average query execution result (2 benchmarks) shows that, Horizontal partitioning has the smallest average query runtime (4.393116824 seconds), followed by No-Partitioning (8.52610303 seconds), Min-Edgecut (10.48653731 seconds), Subject-Based (17.91570378 seconds), TCV-Min (25.26057554 seconds), Predicate-Based (37.66883389 seconds), Hierarchical (40.43121192 seconds) and Recursive-Bisection (554.618705 seconds), respectively.

4.2.3. Number of timeout queries:

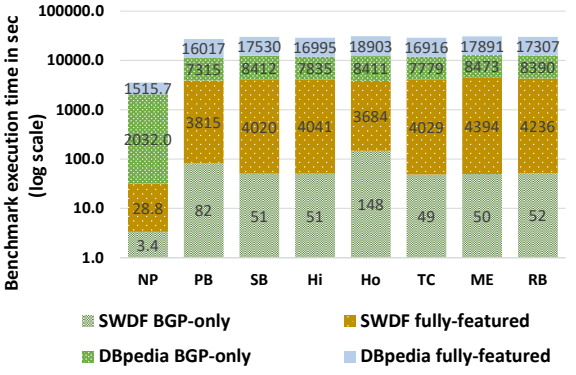
Table 2 shows the overall number of timeout queries for both of our datasets (SWDF and DBpedia) using both benchmark queries (BGP-only and Fully-featured) for each of the partitioning techniques using FedX, SemaGrow and Koral. The overall result suggests that, No-Partitioning has the smallest timeouts (0 queries) followed by Min-Edgecut (total 344 queries), followed by the Subject-Based (total 422 queries), TCV-Min (total 455 queries), Predicate-Based (total 485 queries), Horizontal (total 498 queries), Hierarchical (total 544 queries), and Recursive-Bisection (total 556 queries), respectively.



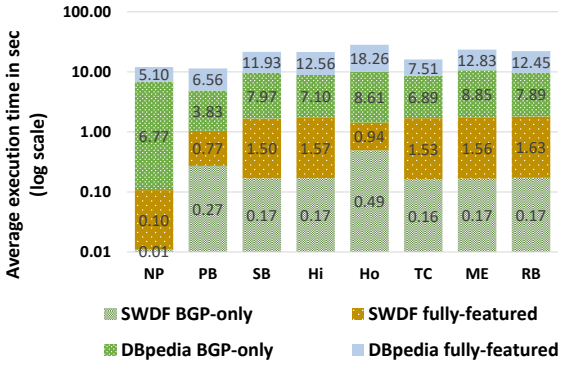
(a) FedX benchmarks execution time



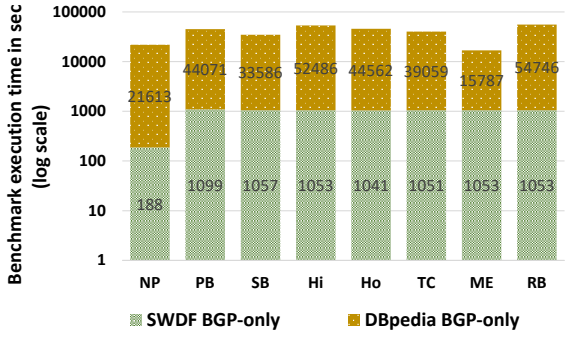
(b) FedX average query runtimes



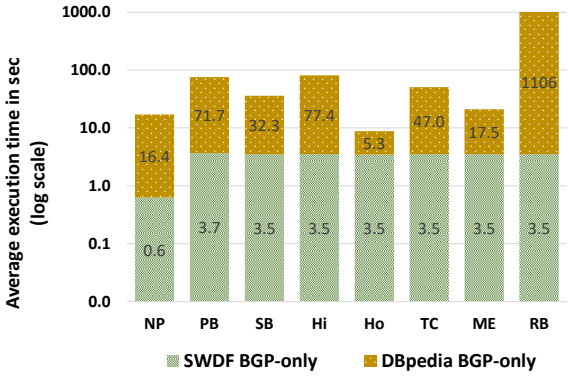
(c) SemaGrow benchmarks execution time



(d) SemaGrow average query runtimes



(e) Koral benchmarks execution time



(f) Koral average query runtimes

Fig. 6.: Total benchmarks (300 queries each) execution time including timeouts and average query runtimes excluding timeouts. (**NP** = No-Partitioning, **PB** = Predicate-Based, **SB**= Subject-Based, **Hi**= Hierarchical, **Ho** = Horizontal, **TC** = TCV-Min, **ME** = Min-Edgecut, **RB** = Recursive Bisection).

	FedX				SemaGrow				Koral	
	SWDF		DBpedia		SWDF		DBpedia		SWDF	DBpedia
Partitioning	BGP	FF	BGP	FF	BGP	FF	BGP	FF	BGP	FF
No-Partitioning	0	0	0	0	0	0	0	0	0	0
Predicate-Based	0	35	32	73	0	20	35	81	0	209
Subject-Based	0	24	29	69	0	20	35	83	0	162
Hierarchical	0	28	28	70	0	20	33	79	0	286
Horizontal	0	12	31	73	0	19	34	83	0	246
TCV-Min	0	24	35	70	0	20	33	85	0	188
Min-Edgecut	0	30	35	74	0	22	34	84	0	65
Recursive-Bisection	0	19	32	70	0	21	35	81	0	298

Table 2: Timeout queries using FedX, SemaGrow and Koral. (**FF** = Fully Featured).

4.2.4. Overall Ranking of Partitioning Techniques:

Table 3 shows the overall rank-wise results of the selected partitioning techniques based on the total benchmark execution time using all 4 benchmarks. In FedX, No-Partitioning ranked 1^{st} in all 4 benchmarks Predicate-Based partitioning ranked 2^{nd} and 3^{rd} once each, and 8^{th} twice, this shows that Predicate-based either produces the best or worst query runtime performances among the selected partitioning techniques. Subject-Based partitioning occurred mostly in the middle ranks, this shows that this technique results average runtime performances among the selected partitioning techniques. Hierarchical partitioning occurred in all, i.e., top, middle, and lower positions, this shows an unpredictable runtime performances by this technique. Horizontal partitioning has given the best results (twice good and on the other two occasions it gave the average results). TCV-Min has been very consistent by producing the third best result on three occasions. Min-Edgecut runtime performance has been on the lower side. Recursive-bisection occurred thrice at the best side of the scale, however it ranked 6^{th} once.

In SemaGrow, No-Partitioning is again ranked 1^{st} in all 4 benchmarks, most of the results using Predicate-Based partitioning have been at the good side of the scale. The query runtime performances of the Subject-Based and Hierarchical partitioning techniques are either average or at the lower sides. Horizontal partitioning has given best result only once and the lower results on the rest of three occasions. TCV-Min performance has mostly been on the high ranked side. The runtime performance of Min-Edgecut is usually on the lower side. Recursive-Bisection is also at the lower side.

In Koral, No-Partitioning ranked 1^{st} and 2^{nd} in two occasions, Predicate-Based partitioning provided below average query runtime performances. Subject-Based ranked 3^{rd} and 5^{th} one time each. Hierarchical has been on the lower side. Horizontal ranked 2^{nd} and 6^{th} one time each. TCV-Min has produced good overall results by ranking once at 3^{rd} and once at 4^{th}. Similar to TCV-Min, Min-Edgecut has also produced good query runtime performances. Recursive-Bisection occurred once at 4^{th} and once at 8^{th}. Please note that Koral ranking is based on a total of 2 (BGP-only) benchmarks because it does not support most of fully featured SPARQL queries.

4.2.5. Rank scores:

From table 3, it is difficult to find which partitioning technique has generally ranked better. To compute the rank scores (ref., definition 4) pertaining to each of the partitioning techniques presented in figure 7, we used table 3, in which, No-Partitioning results has resulted highest rank score, followed by TCV-Min, Property-based, Horizontal, Recursive-Bisection, Subject-Based, Hierarchical, and Min-Edgecut respectively.

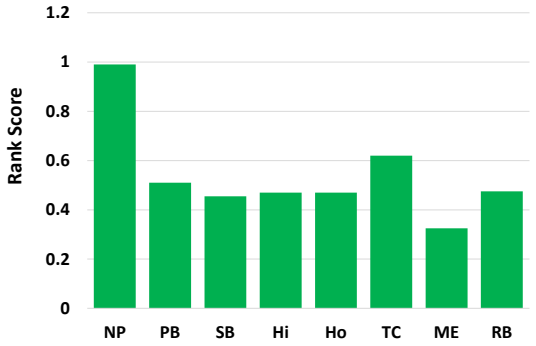


Fig. 7.: Rank scores. (**NP** = No-Partitioning, **PB** = Predicate-Based, **SB**= Subject-Based, **Hi**= Hierarchical, **Ho** = Horizontal, **TC** = TCV-Min, **ME** = Min-Edgecut, **RB** = Recursive Bisection).

	FedX								SemaGrow								Koral							
PT	1st	2nd	3rd	4th	5th	6th	7th	8th	1st	2nd	3rd	4th	5th	6th	7th	8th	1st	2nd	3rd	4th	5th	6th	7th	8th
NP	4	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
PB	0	1	1	0	0	0	0	2	0	2	1	0	0	1	0	0	0	0	0	0	1	1	0	0
SB	0	0	1	1	1	0	1	0	0	0	0	2	0	1	1	0	0	0	1	0	1	0	0	0
Hi	0	1	0	0	2	1	0	0	0	0	0	3	0	1	0	0	0	0	0	1	0	0	1	0
Ho	0	1	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	0	0	1	0	0
TC	0	0	0	3	0	1	0	0	0	1	2	0	1	0	0	0	0	0	1	1	0	0	0	0
Mi	0	0	0	1	0	0	2	1	0	0	1	0	0	0	1	2	1	0	0	1	0	0	0	0
Re	0	1	1	2	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	1	0	0	0	1

Table 3: Overall rank-wise ranking of partitioning techniques based on two benchmarks from SWDF and DBpedia each. (**NP** = No-Partitioning, **PB** = Predicate-Based, **SB**= Subject-Based, **Hi**= Hierarchical, **Ho** = Horizontal, **TC** = TCV-Min, **ME** = Min-Edgecut, **RB** = Recursive Bisection).

4.2.6. Partitioning imbalance:

Figure 8 shows the imbalance in (defined in definition 5) the values of the partitions generated by the each partitioning techniques. In which, the Horizontal portioning has resulted the least partitioning imbalance, followed by Hierarchical, Subject-Based, Min-Edgecut, Recursive-Bisection, TCV-Min and Predicate-Based respectively.

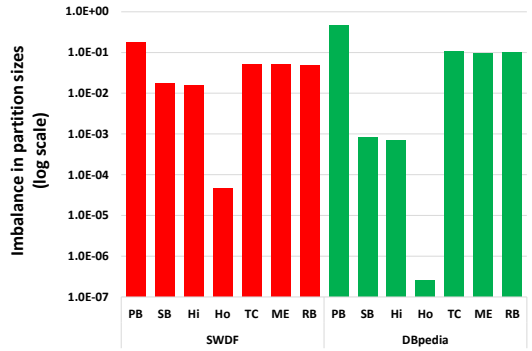


Fig. 8.: Partitioning imbalance. (**PB** = Predicate-Based, **SB**= Subject-Based, **Hi**= Hierarchical, **Ho** = Horizontal, **TC** = TCV-Min, **ME** = Min-Edgecut, **RB** = Recursive Bisection).

4.2.7. Number of sources selected:

According to [24], the number of sources selected (in our case, SPARQL endpoints) to execute a given SPARQL query by the federation engine is a key performance. Figure 9 shows the total distinct sources selected by FedX and SemaGrow. Note that the sources selection algorithm of both FedX and SemaGrow selected exactly the same sources. Generally (over 4 benchmarks) sources selection evaluation. The overall result suggests that Predicate-Based has selected the smallest number of sources, followed by Min-Edgecut, TCV-Min, Recursive-Bisection, Subject-Based, Hierarchical and Horizontal, respectively.

4.2.8. Spearman's rank correlation coefficients:

In order to show how the number of sources selected affect the query execution time, we computed the Spearman's rank correlation between the number of sources selected and the query execution time. Table 4 shows Spearman's rank correlation coefficient values for the four evaluation benchmarks (FedX and SemaGrow) for the selected partitioning techniques. The overall results shows that the number of sources selected, in general, have a positive correlation with the query execution times, i.e. the bigger the sources selected the bigger the execution time and vice versa.

4.2.9. Significance of Runtime Performances:

Section 4.2.2 shows the runtime performance of the selected partitioning techniques. In this section, we want to measure if the runtime performance improvement of on partitioning technique is significant while comparing the runtime performance of another partitioning technique. To do so, we perform the T-Test on runtime values of the partitioning techniques. Table 5 shows T-Test results of all the partitioning techniques using FedX, SemaGrow and Koral. We used the combined runtimes of all benchmarks (1200 queries for FedX and SemaGrow and 600 queries for Koral). We compared the 1st ranked partitioning technique with 2nd ranked partitioning technique, which in turn is compared with 3rd ranked partitioning techniques and so on. We use a (*) if results are significant at 1% confidence level, (**) if results are significant at 5% confidence level, and (***) if results are significant at 10% confidence level. Thus, the values inside table shows the ranking of the partitioning techniques and number of stars shows if this particular ranked is significant or not.

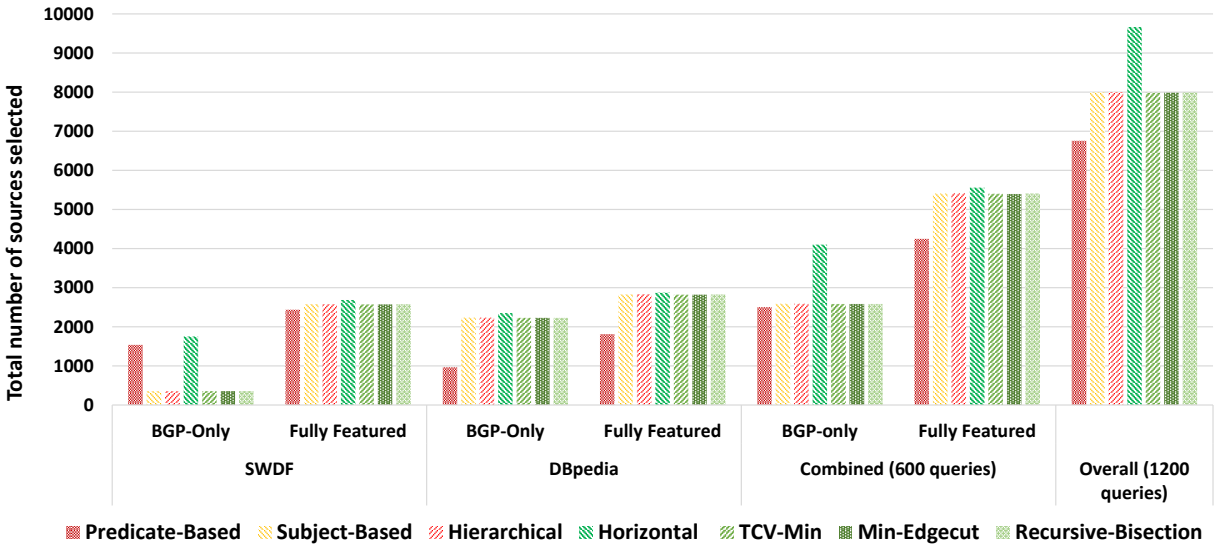


Fig. 9.: Total distinct sources selected

	Benchmark	Pred	Sub	Hierar	Horiz	TCV	Mincut	Recur	Average
FedX	DBpedia BGP-only	0.22	0.30	0.30	0.28	0.26	0.27	0.29	0.27
	DBpedia Fully-featured	0.14	0.11	0.11	0.16	0.17	0.12	0.17	0.14
	SWDF BGP-only	−0.10	0.57	0.57	0.10	0.57	0.57	0.57	0.41
	SWDF Fully-featured	0.22	0.11	0.13	0.09	0.11	0.13	0.10	0.12
S-Grow	DBpedia BGP-only	−0.02	0.11	0.10	0.06	0.09	0.30	0.29	0.13
	DBpedia Fully-featured	0.14	0.18	0.23	0.02	0.24	0.26	0.16	0.18
	SWDF BGP-only	0.23	0.64	0.64	0.65	0.66	0.64	0.64	0.59
	SWDF Fully-featured	0.07	−0.02	−0.02	−0.07	−0.02	−0.06	−0.01	−0.02
	Average	0.11	0.25	0.26	0.16	0.26	0.28	0.28	0.23

Table 4: Spearman's rank correlation coefficients between number of sources selected and query runtimes. Pred: Predicate-Based, Sub: Subject-Based, Hierar: Hierarchical, Horiz: Horizontal, TCV: TCV-Min, Mincut: Min-Edgecut, Recur: Recursive-Bisection, S-Grow: SemaGrow.
Correlations and colors: $-0.00\ldots -0.19$ very weak (●-), $0.00\ldots0.19$ very weak (●+), $0.20\ldots0.39$ weak (●+), $0.40\ldots0.59$ moderate (●+), $0.60\ldots0.79$ strong (●+).

For FedX, the ranking and significance is as follows: No-partitioning ranked 1st (significant at 1% confidence level), followed by Predicate-Based (significant at 1% confidence level), Horizontal (significant at 10% confidence level), Recursive-Bisection (significant at 10% confidence level), Subject-Based (significant at 1% confidence level), TCV-Min (significant at 1% confidence level), Hierarchical (significant at 10% confidence level), and Min-Edgecut, respectively.

For SemaGrow, the ranking and significance is as follows: No-partitioning ranked 1st (significant at 1% confidence level), followed by Predicate-Based (significant at 1% confidence level), Recursive-Bisection (significant at 10% confidence level), Min-Edgecut (significant at 10% confidence level), Hierarchical (significant at 10% confidence level), Subject-Based (significant at 10% confidence level), TCV-Min (significant at 1% confidence level), and Horizontal, respectively.

For Koral, the ranking and significance is as follows: Min-Edgecut ranked 1st (significant at 1% confidence level), followed by No-partitioning (significant at 1% confidence level), Subject-Based (significant at 10% confidence level), TCV-Min (significant at 1% confidence level), Predicate-Based (significant at 10% confi-

dence level), Horizontal (significant at 1% confidence level), Hierarchical (significant at 5% confidence level), and Recursive-Bisection, respectively.

Partitioning	FedX	SemaGrow	Koral
Predicate-Based	2^{*}	2^{*}	5^{***}
Subject-Based	5^{***}	6^{***}	3^{*}
Hierarchical	7^{***}	5^{***}	7^{**}
Horizontal	3^{***}	8	6^{*}
TCV-Min	6^{***}	7^{*}	4^{*}
Min-Edgecut	8	4^{***}	1^{*}
Recursive-Bisection	4^{***}	3^{***}	8
No-Partition	1^{*}	1^{*}	2^{*}

Table 5: T-Test results of all the partitioning techniques based on their runtime performance. * p-value is greater than 0 and smaller than 0.01, ** p-value is greater than 0.01 and smaller than 0.05, *** p-value is greater than 0.05 and smaller than 0.1

Overall, the results clearly suggest that the differences are significant (p âL'd' 0.01) for majority of the partitioning techniques. Which means that the type of partitioning technique has a significant effect on the overall query runtime performance of the distributed storage solution.

5. Related Work

In previous works, such as, [6,7,10,11,12,13,19,21, 22,27,28,33], a plethora of clustered triplestores have been designed. In this section we discuss only those literature which use RDF graph partitioning. Koral [16] a modularized distributed RDF store in which the inter-dependencies between its components are reduced to an extent that each component can be exchanged with alternative implementations. It is an open source glass box profiling system in which three RDF graph partitioning, i.e., Subject-Based, Hierarchical and Min-Edgecut are used. The performance of the partitioning techniques, in terms of query execution time, presented in [16] and [17] is similar to our results where Subject-Based has consumed least overall time followed by Hierarchical and Min-Edgecut. Both [16] and [17] used synthetic data using three aforementioned partitioning techniques, in our work, we used not just real data out of real queries but also we make use of four additional partitioning techniques (Horizontal, Predicated-Based, TCV-Min and Recursive-Bisection). in [32], a signature tree-based triple indexing scheme is proposed to store the partitions of the RDF graph efficiently. [30] provides a brief survey on RDF graph partitioning. [15] suggests that, partitioning techniques based on hashing, are more scaleable as hash values can be computed in parallel,

To the best of our knowledge, there is no detailed empirical evaluation exists to position the different RDF graph partitioning techniques based on real data and real queries in two different evaluation environments.

6. Conclusion and Future Work

In this paper, we presented an empirical evaluation of seven different RDF-graph partitioning techniques by using different environments and different systems as well as different benchmarks. We performed T-Test[8] analysis to shows significant differences in the runtime performances achieved by different partitioning techniques. Our overall results suggest that:

1. When it comes to query runtime performance evaluation, No-partitioning leads to the smallest query runtimes followed by TCV-Min, Property-Based,Horizontal, Recursive-Bisection, Subject-Based, Hierarchical, and Min-Edgecut, respectively. However, for Koral the performance of Min-Edgecut partitioning is better than no-partitioning, suggesting only intelligent (according to RDF SPARQL features) partitioning technique can lead to performance improvement. Consequently, the general graph partitioning techniques may not lead to better performance when implied to RDF graphs. Therefore, clustered RDF storage solutions should take into account the properties of RDF and Linked Data as well as the expressive features of SPARQL queries when partitioning the given dataset among multiple data nodes.
2. There is a direct relation between the number of sources selected with query runtimes. This means that a partitioning technique would generally lead to better query runtime performances when it minimizes the total number of sources selected.
3. The partitioning techniques that implement the k-way partitioning problem (TCV-Min, Min-Edgecut, and Recursive-Bisection in our case) generally need longer time to generate the desired partitions.

[8]Please see T-Test tab of the excel sheet goo.gl/fxa4cJ

4. The different ranks achieved on FedX and SemaGrow suggest that the query runtime performances of the partitioning technique are greatly dependent upon the query planner used by the underlying query processing engine.

In future, we will implement more clustered querying engines. We will also test the scalability of our partitioning techniques by using same datasets with different sizes. We will also make use of some more Big RDF datasets. When involving reasoning tasks or data updates etc, the effects of partitioning pertaining to a given use-case will also be focused.

7. Acknowledgements

This work has been supported by the project LIMBO (no. 19F2029I) and OPAL (no. 19F2028A).

References

[1] maribel acosta, maria-esther vidal, tomas lampo, julio castillo, and edna ruckhaus. anapsid: an adaptive query processing engine for sparql endpoints. In *international semantic web conference*, pages 18–34. springer, 2011.

[2] adnan akhter, axel-cyrille ngomo ngonga, and muhammad saleem. an empirical evaluation of rdf graph partitioning techniques. In *european knowledge acquisition workshop*, pages 3–18. springer, 2018.

[3] güneş aluç, olaf hartig, m tamer özsu, and khuzaima daudjee. diversified stress testing of rdf data management systems. In *international semantic web conference*, pages 197–212. springer, 2014.

[4] aydın buluç, henning meyerhenke, ilya safro, peter sanders, and christian schulz. recent advances in graph partitioning. In *algorithm engineering*, pages 117–158. springer, 2016.

[5] angelos charalambidis, antonis troumpoukis, and stasinos konstantopoulos. semagrow: optimizing federated sparql queries. In *proceedings of the 11th international conference on semantic systems*, pages 121–128. acm, 2015.

[6] orri erling and ivan mikhailov. towards web scale rdf. *proc. ssws*, 2008.

[7] luis galárraga, katja hose, and ralf schenkel. partout: a distributed engine for efficient rdf processing. In *proceedings of the 23rd international conference on world wide web*, pages 267–268. acm, 2014.

[8] olaf görlitz and steffen staab. splendid: sparql endpoint federation exploiting void descriptions. In *proceedings of the second international conference on consuming linked data-volume 782*, pages 13–24. ceur-ws. org, 2011.

[9] olaf görlitz, matthias thimm, and steffen staab. splodge: systematic generation of sparql benchmark queries for linked open data. In *international semantic web conference*, pages 116–132. springer, 2012.

[10] sairam gurajada, stephan seufert, iris miliaraki, and martin theobald. triad: a distributed shared-nothing rdf engine based on asynchronous message passing. In *proceedings of the 2014 acm sigmod international conference on management of data*, pages 289–300. acm, 2014.

[11] mohammad hammoud, dania abed rabbou, reza nouri, seyedmehdi-reza beheshti, and sherif sakr. dream: distributed rdf engine with adaptive query planner and minimal communication. *proceedings of the vldb endowment*, 8(6):654–665, 2015.

[12] steve harris, nick lamb, nigel shadbolt, et al. 4store: the design and implementation of a clustered rdf store. In *5th international workshop on scalable semantic web knowledge base systems (ssws2009)*, pages 94–109, 2009.

[13] andreas harth, jürgen umbrich, aidan hogan, and stefan decker. yars2: A federated repository for querying graph structured data from the web. In *the semantic web*, pages 211–224. springer, 2007.

[14] herodotos herodotou, nedyalko borisov, and shivnath babu. query optimization techniques for partitioned tables. In *proceedings of the 2011 acm sigmod international conference on management of data*, pages 49–60. acm, 2011.

[15] jiewen huang, daniel j abadi, and kun ren. scalable sparql querying of large rdf graphs. *proceedings of the vldb endowment*, 4(11):1123–1134, 2011.

[16] daniel janke, steffen staab, and matthias thimm. koral: a glass box profiling system for individual components of distributed rdf stores. In *ceur workshop proceedings*, 2017.

[17] daniel janke, steffen staab, and matthias thimm. impact analysis of data placement strategies on query efforts in distributed rdf stores. *journal of seb semantics*, 50:21–48, 2018.

[18] george karypis and vipin kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *siam journal on scientific computing*, 20(1):359–392, 1998.

[19] anurag khandelwal, zongheng yang, evan ye, rachit agarwal, and ion stoica. zipg: A memory-efficient graph store for interactive queries. In *proceedings of the 2017 acm international conference on management of data*, pages 1149–1164. acm, 2017.

[20] muhammad saleem, gábor Szárnyas, felix conrads, syed ahmad chan bukhari, qaiser mehmood, and axel-cyrille ngonga ngomo. how representative is a sparql benchmark? an analysis of rdf triplestore benchmarks. In *the web conference (www)*, san francisco, ca, usa, 2019 2019. acm, acm.

[21] thomas neumann and gerhard weikum. the rdf-3x engine for scalable management of rdf data. *The vldb journalâĂŤthe international journal on very large data bases*, 19(1):91–113, 2010.

[22] alisdair owens, andy seaborne, nick gibbins, et al. clustered tdb: A clustered triple store for jena. 2008.

[23] 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.

[24] muhammad saleem, yasar khan, ali hasnain, ivan ermilov, and axel-cyrille ngonga ngomo. a fine-grained evaluation of sparql endpoint federation systems. *semantic web*, 7(5):493–518, 2016.

[25] muhammad saleem, qaiser mehmood, and axel-cyrille ngonga ngomo. feasible: A feature-based sparql benchmark generation framework. In *international semantic seb conference*, pages 52–69. springer, 2015.

[26] 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.

[27] alexander schätzle, martin przyjaciel zablocki, antony neu, and georg lausen. sempala: interactive sparql query processing on hadoop. In *international semantic web conference*, pages 164–179. springer, 2014.

[28] alexander schätzle, martin przyjaciel zablocki, simon skilevic, and georg lausen. s2rdf: rdf querying with sparql on spark. *proceedings of the vldb endowment*, 9(10):804–815, 2016.

[29] 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*, pages 601–616. springer, 2011.

[30] dominik tomaszuk, łukasz skonieczny, and david wood. rdf graph partitions: A brief survey. In *international conference: beyond databases, architectures and structures*, pages 256–264. springer, 2015.

[31] xin wang, thanassis tiropanis, and hugh c davis. lhd: optimising linked data query processing using parallelisation. 2013.

[32] ying yan, chen wang, aoying zhou, weining qian, li ma, and yue pan. efficient indices using graph partitioning in rdf triple stores. In *2009 ieee 25th international conference on data engineering*, pages 1263–1266. ieee, 2009.

[33] kai zeng, jiacheng yang, haixun wang, bin shao, and zhongyuan wang. a distributed graph engine for web scale rdf data. In *proceedings of the vldb endowment*, volume 6, pages 265–276. vldb endowment, 2013.