An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL

Mohamed Ragab
Data Systems Group, University of Tartu
mohamed.ragab@ut.ee

Riccardo Tommasini
Data Systems Group, University of Tartu
riccardo.tommasini@ut.ee

Feras M. Awaysheh
Data Systems Group, University of Tartu
feras.awaysheh@ut.ee

Juan Carlos Ramos
Data Systems Group, University of Tartu
jramos@ut.ee

ABSTRACT

This paper discusses one of the most significant challenges of large-scale RDF data processing over Apache Spark, the relational schema optimization. The choice of RDF partitioning techniques and storage formats using *SparkSQL* significantly impacts query performance. The impact of the relational schemas and the underlying data storage formats is indisputable; they significantly affect the query performance. Nevertheless, the trade-offs in different configurations have not been a subject of intensive study in the literature. This paper presents an in-depth investigation for practitioners to understand such trade-offs and their best practices. It also reports on the pitfalls behind the implementation *SPARQL* optimizations over *SparkSQL*. Our experiments provide insights into these schemas' relative strengths by comparing three different partitioning techniques and four other storage formats. Our results draw a better understanding of the current State-Of-The-Art (S.O.T.A) and pave the way for a wide range of best practices and systematically tuning the performance of distributed systems to handle vast RDF data.

1 INTRODUCTION

Currently, we are witnessing an enormous amount of widely available RDF datasets [19]. Centralized RDF engines, e.g., *RDF-3X* [13] and *gStore* [26], provide native ways for processing/-querying RDF datasets with the full expressive capabilities of *SPARQL*. Yet, they can not handle large-scale RDF datasets effectively [2, 9]. The need for processing large RDF datasets calls for innovative solutions to store, analyze, and query these massive RDF datasets [2]. This call leads the community to leverage Big Data (BD) processing frameworks like `Apache Spark` [25] to process large RDF datasets [3].

BD platforms excel in the analytical processing of relational data. The literature includes several attempts that leverage such capabilities to analyze RDF data [2, 17]. In practice, utilizing BD engines for RDF relational processing requires storing RDF data using a relational schema and translating SPARQL queries into equivalent SQL ones. On the same note, BD platforms are designed to scale horizontally [7]. However, the choice of the right schema can significantly impact the performance of query processing [18]. Moreover, choosing the right partitioning technique also returns with variant query runtime performance [4]. In this regard and from a BD perspective, we cannot ignore the variety of data formats [11]. Given the complexity of the solution space, i.e., relational schema, partitioning technique, storage format, current works focus on one dimension at a time. However, the relevance of a comprehensive analysis of the trade-offs among these dimensions is of paramount importance [3] yet is still missing.

In this paper, we try to fill this research gap by experimentally evaluating SPARQL on top of SparkSQL. In particular, our analysis focuses on existing RDF relational schemas and their state-of-the-art improvements. To this end, we present a systematic and comparative evaluation of the query performance considering (i) *three* RDF partitioning techniques (most suitable for relational nature of data in Spark-SQL), i.e., Horizontal, Subject-based, and Predicate-based partitioning and (ii) $four$ different well-established storage formats, i.e., `ORC`, `CSV`, `Parquet`, and `Avro` [15, 16]. In this way, our work differs from previous ones [21, 22] that only focus on the complexity of the workloads and the size of the data.

The contribution of this paper is threefold. (i) First, it uses SparkSQL to validate the performance of RDF schema advancements (i.e. *ExtVP* and *WPT*) compared to their baseline opponents (i.e *PT*, and *VP*). (ii) Second, it empirically analyzes the effect of partitioning techniques on the ExtVP and WPT schema runtime performance. (iii) Third, it tests the effects of multiple distributed storage row and columnar-oriented file formats on HDFS. Finally, it outlines the best practices and recommendations that help in achieving the best RDF query performance. Overall, the paper findings guide the realization of next-generation large-scale RDF solutions over Apache Spark by optimizing the relational schemas.

The remainder of the paper is organized as follows: section 2 presents an overview of the required background information and key concepts necessary to understand our study. Section 3 discusses the experimental methodology. Section 4 presents the benchmarking scenario and the experimental setup. Section 5 presents the paper results, while we provide a comprehensive discussion in section 6. Section 7 presents the related work, positioning this paper in the context of other survey on RDF processing using BD frameworks. Finally, section 8 concludes the paper and presents future works.

2 BACKGROUND

In this section, we present the information that is necessary to understand the content of this paper. We assume that the reader is familiar with the RDF data model and the SPARQL query language.

2.1 Apache Spark & SparkSQL

`Apache Spark` is currently the *de-facto* BD engine [25]. It is one of the most active and widely-used large-scale data processing systems in both industry and academia [5]. It mainly adopts *in-memory* distributed computing of large scale data analytics.

SparkSQL is a relational package built on top of Apache Spark [5] with support for the SQL interface while providing capabilities for structured and semi-structured data.

2.2 RDF Relational Schema

The most intuitive approach to follow for representing RDF into a relational structure is the *Single Statement Table Schema (ST)*, which requires storing RDF datasets in a single triples table of three columns that represent components of the RDF triple, i.e., *Subject*, *Predicate*, and *Object*. This solution is the simplest, and it is commonly adopted by several existing open-source RDF triplestores, e.g., *Apache Jena*, *RDF4J*, and *Virtuoso*. However, it inevitably increases the number of required *self-joins* for long chains SPARQL query evaluation when they run on top of relational SQL systems.

Vertically Partitioned Tables Schema (VP) is an RDF storage schema proposed to mitigate the performance issues of the ST schema. It aims to speed up the queries over RDF triple stores [1]. This schema is simple to design; the RDF triples table is decomposed into a table of two columns (*Subject*, *Object*) for each unique *property* in the RDF dataset.

Extended Vertical Partitioning schema (ExtVP) is a query-driven optimization that aims at minimizing the input size of the data during query evaluation [22], inspired by the *semi-Join* reductions. In particular, ExtVP minimizes data skewness and eliminates *dangling triples* (i.e. triples that do not have a joining partner or do not contribute to any join in the SPARQL query) from the input tables. ExtVP speeds up query answering by pre-computing the possible join relations between the VP tables and materializing the results of these semi-joins as tables in the storage backend, e.g. HDFS. Particularly, for every two VP relations ExtVP relies on pre-computing semi-join reductions of *Subject-Subject* (SS), *Subject-Object* (SO), and *Object-Subject* (OS) join patterns. The output tables are reduced in size and will be used in joins instead of the original VP tables. However, one of the limitations of the ExtVP schema is the additional storage overhead of the materialized ExtVP tables in comparison to the VP schema tables (cf. Table 1).

Property (n-ary) Tables Schema (PT) is a storage schema proposed to cluster multiple RDF properties as *n-ary* table columns for the same *subject* to group entities that are similar in structure. The biggest advantage of property tables compared to a single triples table schema (ST) is that they can reduce the number of *subject-subject self-joins* that result from *star-shaped* patterns in a SPARQL query. Whereas, one of the limitations of the PT schema is that it works quite well with the highly structured RDF data. However, its performance degrades for the poorly structured ones [23]. Furthermore, typical RDF comes with diverse structures, which make it virtually hard to define an optimal layout of this schema [22]. Moreover, a poorly-selected property table layout can significantly slow down the query performance [2]. Due to its sparse-tables representation nature, PT schema also suffers from high storage overheads when a large number of predicates is present in the RDF data model [1].

Wide Property Table Schema (WPT) represents the whole RDF dataset into a single unified table [21]. Such table uses all RDF properties in the dataset as columns. It aims at extending the PT schema for optimizing *star-shaped* SPARQL queries, which are highly common in the SPARQL query workloads. Therefore, star-shaped SPARQL queries will require *no joins* to be answered. Moreover, this schema does not require any kind of clustering algorithm that is likely to produce sub-optimal schemas for an arbitrary RDF dataset. Unfortunately, WPT does not overcome all the limitations of the PT schema. Indeed, this representation can also be very sparse for poorly structured data, and it may face a large storage overhead, especially with many *multi-valued* properties existing in the RDF dataset.

2.3 RDF Data Partitioning

For RDF data processing, many partitioning techniques exist [2, 4]. In the following, we present the partitioning techniques that are suitable for our experiments on SparkSQL.

Horizontal-Based Partitioning (HP) requires dividing the RDF dataset evenly (as much as possible) on the number of machines in the cluster. In particular, we use this technique to partition the relational RDF tables of the different schemas horizontally into even *n* chunks(i.e partitions) over the cluster machines.

Subject-Based Partitioning(SBP) requires the distribution of triples into partitions according to the *hash value* computed for the RDF *subjects*. As a result, all the triples that have the same *subject* are assumed to reside on the same partition. In our scenario, we applied spark partitioning using the *subject* as the partitioning key with our different relational schema tables (i.e DataFrames).

Predicate-Based Partitioning (PBP) is similar to the SBP, it distributes triples to the various partitions based on the *hash value* computed for the *predicate*. Similarly, all the triples that have the same *predicate* are assumed to reside on the same partition. We also applied the Spark partitioning using the *predicate* as the partitioning key with our different relational schemas Dataframes.

Baseline partitioning (BP): In our experiments, we also used the baseline partitioning technique that basically depends on the native default partitioning of HDFS of the tables files over the cluster nodes. This is the technique used in the state-of-the-art works of the schema advancements [21, 22].

3 EVALUATION METHODOLOGY

In this section, we discuss the experimental methodology that we used for the reproducibility of the state-of-the-art findings [6, 21, 22] that imply some changes in the experimental artifacts, we organize our experiments as follows.

First, we assess if we can reproduce the state-of-the-art results of those schema optimizations over the baseline relational schemas performance. Thus, we performed our experiments in a setup as similar as possible to what the original authors have done [21, 22]. In this regard, we use the baseline HDFS partitioning technique. We also use *Parquet* as our baseline storage file format (grey shaded boxes cf. Figure 1).

Second, we introduce disturbing factors to our experiments, such as the different partitioning techniques, and different file formats alongside different SPARQL query shapes.

Regarding the data partitioning, we introduce the Horizontal Partitioning technique and Subject-based partitioning for the WPT and PT schema experiments.On the other hand, Horizontal, Subject and Predicate-based partitioning techniques were used for the VP and ExtVP schema experiments. We expect that these partitioning techniques will negatively impact the performance of SparkSQL when evaluating SPARQL queries due tothe distribution of the relational table across nodes. This will force more shuffling in the presence of joins. In particular, Horizontal partitioning should have a worse impact than Subject-based

Table 1: SP^{2}Bench-100M RDF relational schemata table data sizes with different file formats

SP^{2}Bench	RDF (n3)	PT	WPT	VP	ExtVP
CSV	**11GB**	~9.2MB-1.9GB -Total: **6.8GB**	**9.4GB**	8KB-1.9GB -Total: **8.3GB**	- OS (4.9GB) - SS (39GB) - SO (806MB) -Total:~**45GB**
Avro	**11GB**	980KB-416MB -Total: **1.6GB**	**1.8GB**	8KB-272MB -Total: **1.7GB**	- OS (359MB) - SS (8.8GB) - SO (331MB) -Total:~**9.5GB**
ORC	**11GB**	620KB-362MB -Total: **1.4GB**	**1.4GB**	8KB-249MB -Total: 1.5GB	- OS (243MB) - SS (7.8GB) - SO (301MB) -Total:~**8.4GB**
Parquet	**11GB**	620KB-382MB -Total: **1.5GB**	**1.7GB**	8KB-264MB -Total: **1.6GB**	- OS (319MB) - SS (8.4GB) - SO (318MB) -Total:~**9GB**

partitioning on PT and WPT schemas, and Predicated-based on (Ext)VP ones. It worth mentioning that the HP technique does not take the query shape into account and possibly place these rows in different nodes.

Regarding the storage of file formats besides the baseline Parquet, we consider an additional *columnar* one, i.e., *ORC*, and two *row-oriented* ones, i.e., *CSV* and *Avro*. We expect columnar formats to perform better for the queries with a subset of column projections, since they allow an efficient scan of tables by reading only a portion of columns [10]. In action, SP^{2}Bench has a small number of column projections across all its benchmark queries.

Finally, aiming to draft our observations, primary findings, and propose best practices, we discuss and analyze our results. Additionally, we highlight the *trade-offs* of combining all these dimensions in the discussion section.

Moreover, we aim to observe these optimizations' impact on the large SPARQL query performance on the SparkSQL engine. Mostly, we want to verify and answer the following questions:

(1) How far do RDF partitioning techniques and storage formats impact the query performance?
(2) How can we systematically analyze different relational schemas? How can these schemas effectively improved to achieve the highest performance?
(3) What are the best practices that guide the large RDF community efforts in adopting performance-oriented solutions?

4 BENCHMARK & EXPERIMENTAL SETUP

This section outlines the paper experiment setup and the used benchmark with its queries. The experimental setups (presented in Figure 1) summarizes the configuration combinations (*Relational schema*, *Partitioning*, *Storage*). The triangle with **X** represents that we have performed our experiments for 4 different relational schemas, partitioning each schema **across** 4 various relational techniques, i.e one baseline HDFS, and other 3 RDF-specific techniques. Last but not least, those schemas are stored across 4 different storage formats. In detail:

Benchmark &Dataset: In our evaluation, we used the SP^{2}Bench (SPARQL Performance Benchmark) [24]. SP^{2}Bench has a reasonable low score of data *structuredness*, making it closer to the structure of *real-world* RDF datasets [20]. So, it is valid to state that, to the best of our understanding, SP^{2}Bench meets a wide spectrum of queries and answers well the main claims we are investigating.

Data Storage: We generated a *synthetic* RDF dataset with 100*M* triples size in `Notation3` format. This scale size is enough for checking the validity of the literature findings regarding the RDF relational schemas optimizations, and maintaining the reproducibility of them in a more complex solution space.

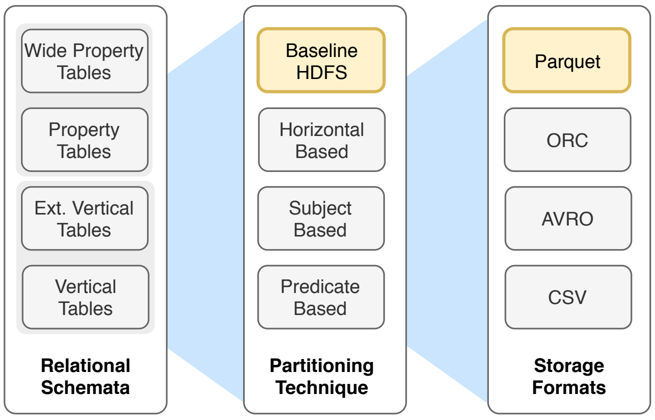


Figure 1: Experiments architecture and evaluation environment

	#Joins	#Filters	#Projections	Query Shape
Q1	3	0	1	S
Q2	8	0	10	S
Q3	1	1	1	S
Q4	7	1	2	SF
Q5	5	1	2	SF
Q6	8	3	2	SF
Q7	12	2	1	SF
Q8	10	2	1	SF
Q9	3	0	1	S (U)
Q10	0	0	2	TP (U)
Q11	0	0	1	TP

Table 2: Benchmark Queries Characteristics: Shape, i.e., [S]tar, [S]now[F]lake, or a single [T]riple[P]attern; (U) for unbounded Predicate Variable, Number of Joins, filters, and projections.

The generated *n3* RDF dataset is converted into CSV relational schemas using *Jena TDB* [1], a disk-based access repository for storing RDF datasets. We further used the *Jena ARQ* [2] for querying these TDB datasets and generating the output schemas tables in the CSV file format. Finally, these raw textual CSV documents are loaded to the HDFS. Moreover, we have used the Spark framework to write the relational schemas data tables from the CSV format into the other HDFS file formats (Avro, Parquet, and ORC). Table 1 shows the size of the generated native RDF dataset (i.e 11GB), as well as store sizes of each relational schema in the mentioned different file formats on top of HDFS. It is clearly shown, how the different relational schemas affect the input data sizes.

[1]https://github.com/apache/jena/tree/master/jena-tdb
[2]https://github.com/apache/jena/tree/master/jena-arq

In action, the PT schema has the smallest table sizes in total, followed by the VP schema, then the WPT table schema. Whereas, the largest storage overheads come with the ExtVP schema. We can also notice how the storage formats affect the sizes of the schemas significantly. In particular, columnar-oriented formats have the minimum table sizes across all the schemas. Indeed, ORC is shown to have the minimum table sizes, followed by Parquet. While, the Avro row-oriented formats have quite larger schema sizes, and CSV has the largest table sizes.

Queries: SP^{2}Bench queries have different complexities and a high *diversity* of features [20]. These queries implement meaningful requests on top of RDF data. In our experiments, we reused the SQL version of the queries associated with the SP^{2}Bench benchmark [3] for the mentioned RDF relational schemas. However, for the new relational schema advancements (e.g. ExtVP, WPT) that are missing on the benchmark website, we have manually translated these queries into SQL, and we provide all these translated queries in our project repository [4]. We have evaluated all of these 11 queries of type *SELECT*, except *Q*9, and *Q*11 which are not applicable (*'NA'*) for the PT and the WPT relational schemas. *Q*7 is also not applicable in the VP and ExtVP schemas. Notably, for generating the ExtVP tables, the default selectivity threshold of 1 has been configured [22]. Table 2 shows our benchmark queries complexities, in terms of the number of joins, filters, and projections, alongside the SPARQL query shape.

Environment Setup: Our experiments were executed on a *bare-metal* cluster of 4 machines with *CentOS-Linux* V7 OS, running on 32 cores per node processor, and 128 GB of memory per node, alongside with a high speed 2 TB *SSD* drive for each node. We used Spark V2.4 to fully support SparkSQL capabilities. In particular, our Spark cluster consists of one master node and 3 worker machines, while *Yarn* is used as the *resource manager*, which in total uses 330 GB and 84 virtual processing cores.

RDF Data Partitioning: We used Spark partitioners for partitioning the registered relational schemas tables/Spark DataFrames. This is required to persist those DataFrames on top of the HDFS default file blocks partitioning level. We use the resulting Data Frames as the input for the query engine. In our experiments, we have the baseline HDFS partitioning (grey partitioning box cf. 1). While other RDF partitioning techniques also have been tested, namely HP, SBP, and PBP approaches. These techniques depend on partitioning the tables' data horizontally across machines (i.e HP), or on the Spark key partitioning of the RDF subject or predicate (i.e SBP, PBP respectively).

Performance Evaluation measure (Latency): We used the *Spark.time* function by passing the *spark.sql(...)* query execution function as a parameter to measure the query *latency*. We run the experiments for all queries 5 times (excluding the *first cold start* run time, to avoid the *warm-up* bias, and computed an average of the other 4 run times).

5 EXPERIMENT RESULTS

In this section, we discuss our experiment results. Also, we compare the optimized relational schemas (i.e., WPT, and ExtVP) against their baseline schemas, i.e., PT, and VP, respectively, according to our methodology (cf. Section 3).

	Q1	Q2	Q3	Q4	Q5	Q6	Q8	Q10	Q11
PT	2	9	2	8	7	6	9	5	2
WPT	0	0	0	3	3	3	10	3	0

Table 3: SP2Bench queries: Number of Joins of PT vs WPT.

WPT vs. PT	Avro	CSV	ORC	Parquet
Baseline	2/9	2/9	8/9	9/9
Horizontal	2/9	3/9	6/9	6/9
Subject	2/9	2/9	6/9	6/9

Table 4: Number of queries for which WPT beats PT for data formats and partitioning techniques.

5.1 WPT VS. PT Schema Results

Table 3 shows the SP^{2}Bench queries' number of joins when translated into SQL concerning the PT and WPT schemas. Except for *Q*8 (that requires many self-joins of the WPT table), the number of joins always decreases, adopting the WPT schema. Moreover, we expect that the WPT schema query performance (i.e., in terms of *latency*) will outperform other relational schemas [6]. In this regard, the Parquet data format efficiently handles the sparsity caused by the WPT table schema —as *Null* values are efficiently ignored in this file format [21].

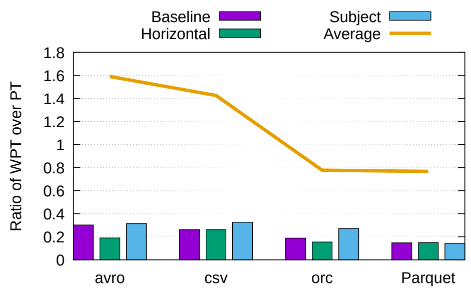
Meanwhile, Table 4 shows the overall benchmark results of the WPT performance over PT schema across all file formats (*horizontally* in the table), and across the different partitioning techniques (*vertically*). Values in this table specify the number of queries in which the WPT schema performs better than the baseline PT schema. The *green* color indicates that WPT performing the best, while the *yellow* color indicates that its performance is above 50% over PT, and the *red* means that performance is less than 50%.

Our experiment results confirm that the WPT schema performs better than the baseline PT schema in all the queries (i.e., 9 queries out of 9 queries in the benchmark) with Parquet file format, alongside using the baseline HDFS partitioning technique. Indeed, these results confirm the findings in [6, 21] assessing the reproducibility regarding the WPT schema optimization.

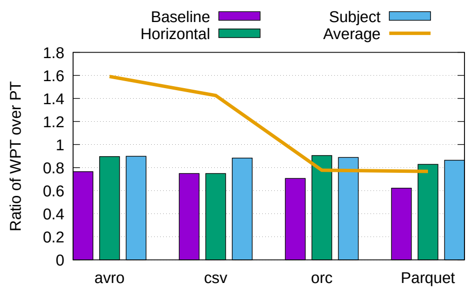
To investigate how the performance difference between the WPT and PT schemas changes, we introduce two new dimensions, i.e., various file formats and different partitioning techniques. In this regard, Table 5 shows the effect of data partitioning (*left* of the table) and storage formats (*right* of the table) considering the other new factors across all the experiments. To this extent, we have calculated the percentages as follows, for the partitioning factor's impact, we pivoted on each partitioning technique and counted the percentage of how much the WPT schema performance in SparkSQL is better than the PT schema one across all the queries while considering all the changes of the storage file formats (moving across them). We calculated the partitioning effect similarly but pivoting on the storage file format and moving across the partitioning techniques in all of queries.

Table 5 also demonstrates that in such a complex space of different relational schema, data partitioning, and storage file formats, the schema-based query optimization is not straightforward. As we can see, WPT outperforms PT schema only for 58% in the queries using only the baseline default HDFS partitioning technique regarding the storage formats, and only 78% for the

[3]http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php
[4]https://datasystemsgrouput.github.io/SPARKSQLRDFBenchmarking/



(a) Q2



(b) Q4

Figure 2: The performance of WPT over PT schema in $Q2$ and $Q4$ (values below 1 means WPT is better than PT)

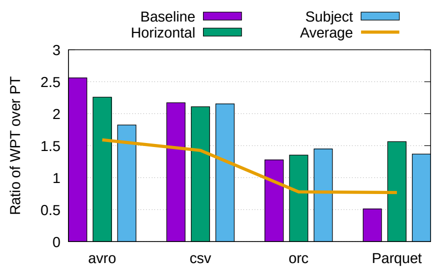


Figure 3: The performance of WPT over PT schema in Q8. values (below 1 means WPT is better than PT)

Parquet file format. The determination of this result shows the trade-off of considering alternative storage file formats and partitioning techniques alongside the experiments' query evaluation.

Regarding the storage, we can see that ORC, another columnar file format gives closer performance to our baseline columnar Parquet file format with 74%. However, the baseline Parquet is yet better, as Parquet is unlike ORC, can efficiently handle the WPT table's sparsity. Whereas, we can see that row-oriented formats have a significant negative effect on the performance of WPT. WPT schema performance is better than PT with only 22% and 25% in all Avro and CSV queries, respectively. In action, SP^2Bench queries only have one query (i.e., $Q2$) with more than 2 column projections. This justifies why column-oriented formats give better results for the WPT than the row-based ones. In general, we can state that file formats affected the generalization of the state-of-the-art results for the WPT schema.

At last, we enroll in three specific queries, namely, $Q2$, $Q4$, and $Q8$, which well exemplify our findings. We selected these queries as good representatives of our findings. There is a tremendous performance enhancement in WPT over PT in $Q2$ and $Q4$. The reason behind this refers to the number of SparkSQL joins of WPT is significantly less than the joins in PT schema (cf. Table 3). Particularly, in $Q2$ number of joins in PT (SQL-version) is 9 compared to *no-joins* in WPT schema. While in $Q4$ with PT schema, we have 8 SQL joins in comparison to 3 *self-joins* of the WPT table. Interestingly, we have more joins in WPT than the baseline PT schema in $Q8$, i.e., 10 self-joins, and 8 joins, respectively. Figure 2 (a), (b) and Figure 3 depict the performance of SparkSQL for $Q2$, $Q4$, and $Q8$ respectively under a various combination of file formats and partitioning techniques. In particular, these figures combine the ratios of WPT being better than PT in those mentioned queries. Ratios less than 1 indicate better performance of WPT over PT in that query and across the different configuration settings.

WPT/PT	Partitioning effect			Storage effect	
	Baseline_Part	58.33%		***Parquet***	77.78%
	Horizontal	47.22%		***ORC***	74.07%
	Subject-based	44.44%		***CSV***	25.93%
	Predicate-based	NA		***AVRO***	22.22%

Table 5: The effect of other partitioning techniques, and other storage formats on the reproducibility of the WPT S.O.T.A findings

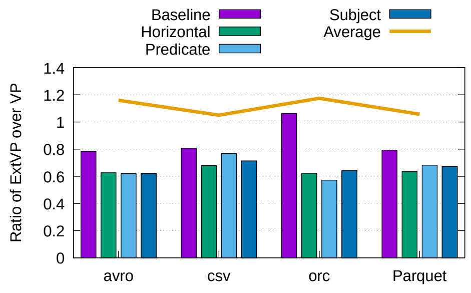
Not surprisingly, we can notice that $Q8$ is the only query that witnesses worse performance for the WPT compared to the PT schema. Figure 3 shows that most of the ratios of 'WPT over PT' is greater than 1 in the baseline-partitioned data experiments (i.e. only partitioned with HDFS), and other file formats instead of Parquet. Notably, all the results (i.e., total query runtimes) and query histograms can be found on our mentioned GitHub repository.

5.2 ExtVP VS. VP Schema Results

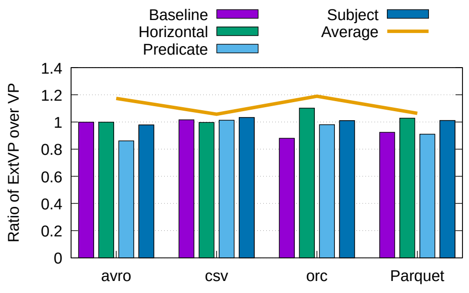
According to [22], ExtVP outperforms or at least has a similar performance to the VP schema. The reason is that queries are similar, and the number of SQL joins in the VP and ExtVP schemas are the same. This clarification is reflected in Table 6. Indeed, the performance improvement depends mainly on the percentage of reductions in the input table sizes that the ExtVP optimization might introduce out of the join correlations for each query [22]. Table 6 also presents the percentage of ExtVP reductions of the processed tables' rows for each query over the original input tables processed rows with the baseline VP tables. The *semi-join* reductions provided by the ExtVP help speeding-up the performance of SparkSQL by reducing the size of the shuffled data.

	VP	ExtVP	Input tables data Size Red.
Q1	2	2	58%
Q2	9	9	77%
Q3	1	1	59%
Q4	7	7	96%
Q5	5	5	60%
Q6	9	9	31%
Q8	9 & 1 Union	9 & 1 Union	5%
Q9	2 & 1 Union	2 & 1 Union	0%
Q10	1 Union	1 Union	0%
Q11	0	0	0%

Table 6: Number of joins and percentage of input tables sizes [Red]uctions after optimization ExtVP VS. VP.



(a) Q4



(b) Q9

Figure 4: The performance of ExtVP over VP schema in Q9. (values below 1 indicates that ExtVP is better than VP)

In more details, ExtVP optimizes specific queries according to the correlations between triple patterns in those queries [22], namely, in *Subject-to-Subject(SS)*, *Object-to-Subject(OS)*, and *Subject-to-Object(SO)* [22]. Thus, we expect some queries to give similar results to the VP schema queries (i.e., No reductions occurred in the VP tables by the ExtVP schema optimization). Notably, in our experiments, $Q9$,$Q10$, and $Q11$ do not present any input data reductions. Thus, we state that it is expected that their performance to be very close to baseline VP performance.

The same approach that has been adopted in WPT to PT schemas performance comparison is also used for evaluating the performance of ExtVP against the VP.

First, we check if our experiments' results confirm the state-of-the-art regarding the ExtVP schema optimization over the baseline VP schema performance.

Table 7 (on the right) shows the total number of queries in which the ExtVP performance is better than VP schema performance across all the benchmark queries. For our baseline HDFS partitioning technique, and with the Parquet file format, we can see that some queries do not benefit from the optimizations of the ExtVP. Indeed, 3 out of 10 queries fail to utilize the optimized ExtVP technique. The reason behind such behavior is that those queries have *unbounded* predicates that can not be optimized by the ExtVP schema [22] (see $Q9$ and $Q10$ in Table 2), or they have no effective join reductions (see $Q9$,$Q10$,$Q11$ in Table 6). The performance of these queries is a subject of discussion in detail in the next sections.

Second, similarly to what we have done for the WPT schema optimization, we now investigate how generalizable the state-of-the-art results are when we introduce different file formats partitioning techniques over the data for both the ExtVP and VP schemas.

Similarly, Table 8 shows how far the data partitioning (*left* of the table) and data formats (*right* of the table) impact the results of ExtVP in comparison to VP schema performance. Notably, this table's percentage values are also calculated similarly to how we have calculated the WPT against the PT. We pivoted on the analysis dimension of choice, i.e., file format X or partitioning technique Y, and we calculated how many times SparkSQL performs better using ExtVP than using the baseline VP approach.

Regarding the partitioning techniques' effect on ExtVP, our expectations are confirmed. In particular, we can observe that the partitioning techniques degraded the performance of ExtVP significantly. Only, 35%, and 30% of the experiments adopting Horizontal, and Subject-based partitioning respectively show a performance improvement in using ExtVP over VP. Adopting

ExtVP VS. VP	**Avro**	**CSV**	**ORC**	**Parquet**
Baseline_Part	6/10	6/10	5/10	7/10
Horizontal_Part	3/10	3/10	3/10	3/10
Predicate_Part	2/10	3/10	6/10	6/10
Subject_Part	2/10	3/10	3/10	3/10

Table 7: Comparison of ExtVP schema with the VP schema in different storage formats, and in different partitioning techniques.

ExtVP/VP	Partitioning effect			Storage effect	
	Baseline_Part	67.5%		***Parquet***	55%
	Horizontal	35%		***ORC***	45%
	Predicate-bsed	55%		***AVRO***	42.5%
	Subject-based	30%		***CSV***	42.5%

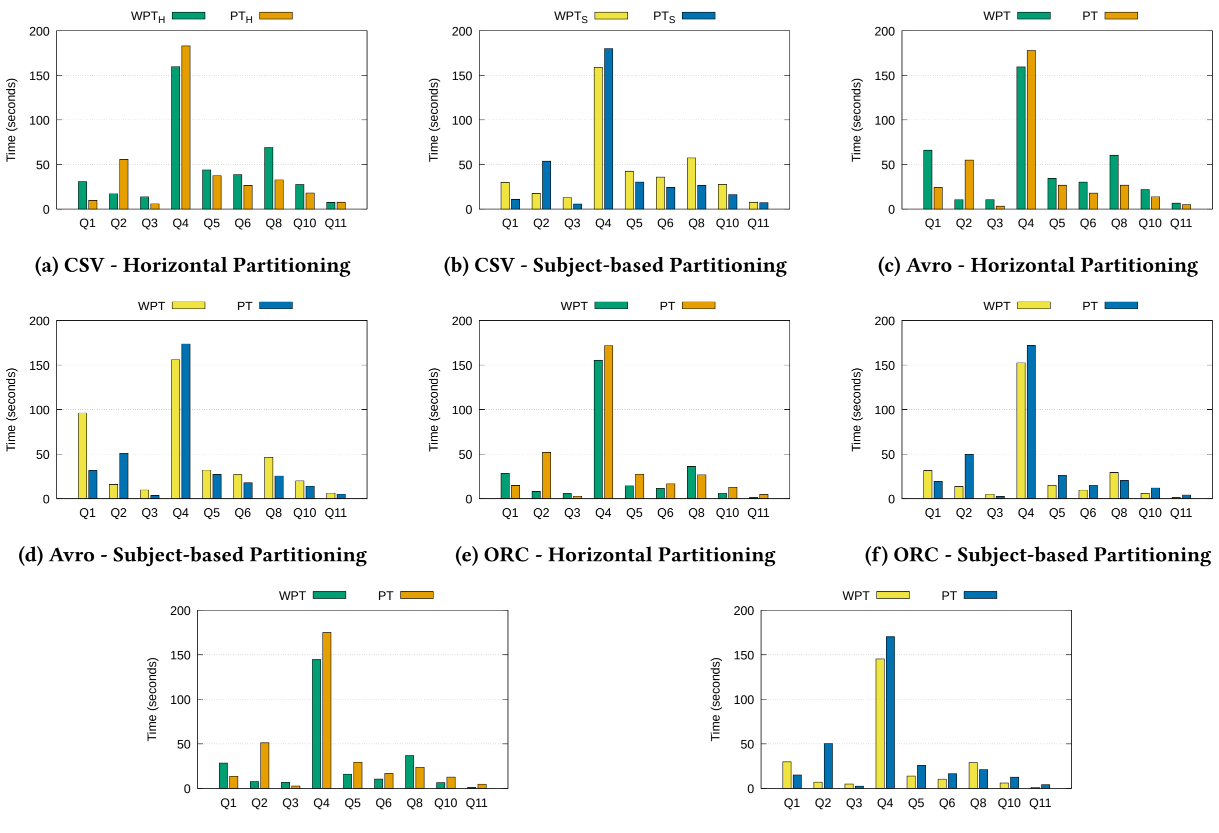
Table 8: The effect of other partitioning techniques, and other storage formats on the reproducibility of the ExtVP S.O.T.A findings

Predicate-based partitioning slightly reduces this negative effect (i.e., 55% of the queries show that performance improvement).

From Table 8, we can also see that the ExtVP schema is only outperforming the VP schema, with 67% of the queries using the baseline HDFS partitioning scenario. Thus, we can see the *trade-off* of considering various storage file formats. We can see also that the baseline Parquet file format is the one that has less impact on the overall performance for ExtVP. Indeed, in 55% of the cases where Parquet is used, ExtVP outperforms the VP performance. Additionally, the ORC columnar file format provides high performance of ExtVP over VP schema with an overall 45%. However, there is a clear difference from the Parquet file format with 10%.

On the other hand, the row-oriented formats degrade the performance of ExtVP. For only 42.5% of the experiments that adopt either Avro or CSV, ExtVP performance beats the performance of the VP schema. Such behavior is related to the number of column projections in the SP^{2}Bench queries, which are the minimum in this benchmark scenario. Thus, columnar file formats can fit such query workloads better than the row-oriented ones.

Last but not least, herein the most notable query examples are introduced, confirming our previous findings but with more innumerable details. First, $Q4$ is revealed to be the query with the most benefit with the ExtVP optimization. The reason behind this is that $Q4$ includes a high number of joins (i.e., 7 joins), and has the maximum number of input tables' rows reductions while using the ExtVP schema optimization with 96% of reduced processed rows (cf. Table 6). This query is directly followed by $Q2$ with 77%. Although $Q2$ has a higher number of table joins



(a) CSV - Horizontal Partitioning **(b) CSV - Subject-based Partitioning** **(c) Avro - Horizontal Partitioning**

(d) Avro - Subject-based Partitioning **(e) ORC - Horizontal Partitioning** **(f) ORC - Subject-based Partitioning**

(g) Parquet - Horizontal Partitioning **(h) Parquet - Subject-based Partitioning**

Figure 5: WPT Vs. PT schemata performance using different partitioning techniques and file formats

than $Q4$, the reductions in input table sizes in $Q4$ are more significant. On the other side, $Q9$, $Q10$, and $Q11$ do not benefit from the ExtVP optimization, i.e., ExtVP does not provide any input table size reductions. In particular, $Q9$ and $Q10$ have unbounded predicate variables in the original SPARQL queries. ExtVP cannot *directly* handle this type of queries[22]. While $Q11$ has only a single triple pattern, and thus it has no joins in optimizing the ExtVP optimization approach. Figures 4 (a) and (b) show the performance of SparkSQL for $Q4$ and $Q9$, respectively, under various combination of formats and partitioning techniques in the ExtVP experiments. Figure 4 (a) shows that $Q4$ is always below the line of all the other queries' average runtimes. Whereas, ExtVP does not show a remarkable difference over the VP schema in $Q9$, i.e., they show pretty close performance to each other.

In the next section, we discuss in further details the experiment findings against the current S.O.T.A regarding the superiority of ExtVP and PT.

6 DISCUSSION

The paper helps to characterize and classify the RDF schemas and their optimizations within the SparkSQL realm. It helps data architects and practitioners interested in large scale RDF better understanding the relational RDF schema's potential using different partitioning techniques and storage formats. This understanding will lead to a better selection of the most suitable and performance-optimized solution that adequately suits their case. Doing so will also accommodate better design and development of new SPARQL systems, leading to reliable RDF services with high Spark performance. Taking our experiment findings into consideration, herein, we discuss our results and give some insights on processing RDF best practices at a large scale.

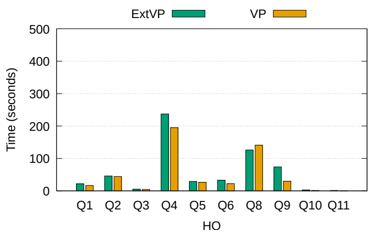
Next, we place the literature assumptions on the relational schema optimizations' superiority against our experimental findings. We follow this by recommendations to the large RDF practitioners.

6.1 Assumption: WPT always outperforms PT

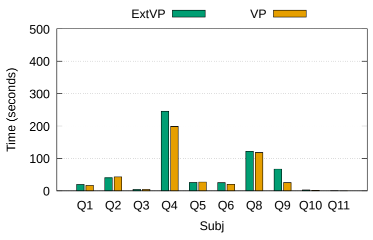
According to [6, 21], we expect that the performance of the WPT schema outperforms the PT schema, especially with the "*star-shaped*" queries. Star-shaped queries can be answered when the WPT table is queried with *no-joins* included. This assumption is because all the properties relevant to the same subject are present in the same row of the WPT table.

The state-of-the-art findings of the WPT schema are fully reproduced with the default HDFS partitioning and with using the baseline Parquet file format. That is, the performance of Spark using WPT schema for representing RDF dataset is always outperforming the baseline PT schema.

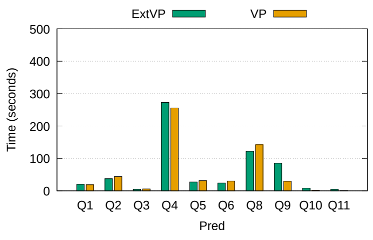
Nevertheless, our results show when we deviate from the original setup [21] introducing new experimental factors, the solution space increases in complexity. Consequently, the trade-offs between relational schema, partitioning techniques, and storage formats make the WPT optimization reproducibility not straightforward. Using other partitioning techniques alongside the baseline Parquet format affected the reproducibility of the WPT schema



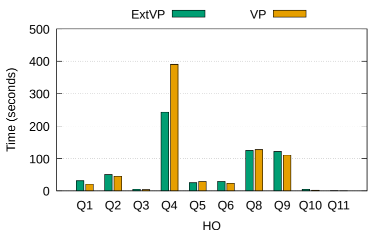
(a) Parquet-Horizontal Partitioning



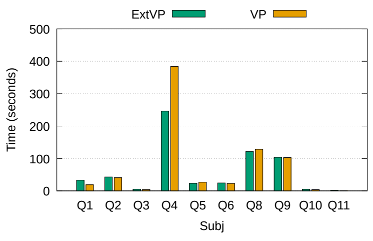
(b) Parquet-Subject-based Partitioning



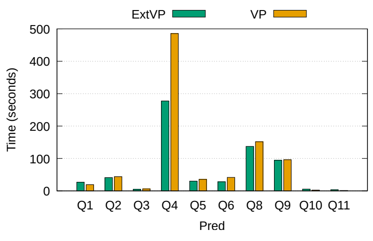
(c) Parquet-Predicate-based Partitioning



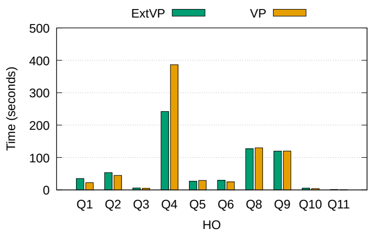
(d) ORC-Horizontal Partitioning



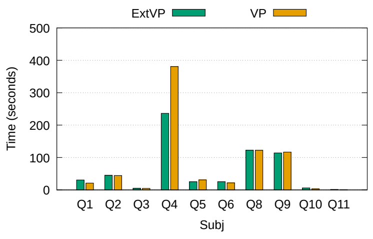
(e) ORC-Subject-based Partitioning



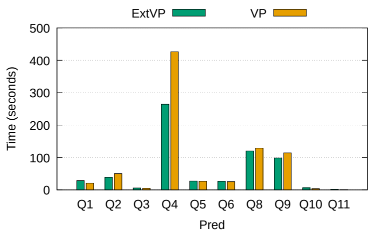
(f) ORC-Predicate-based Partitioning



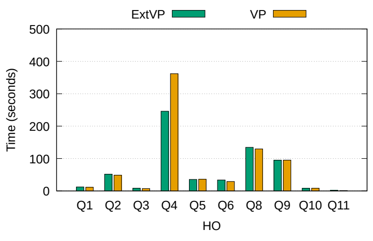
(g) Avro-Horizontal Partitioning



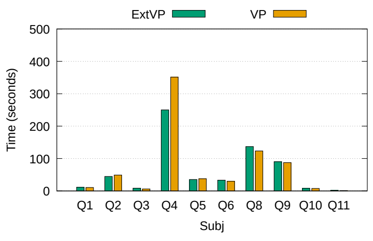
(h) Avro-Subject-based Partitioning



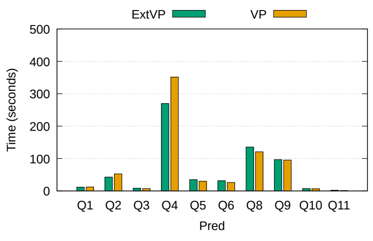
(i) Avro-Predicate-based Partitioning



(j) CSV-Horizontal Partitioning



(k) CSV-Subject-based Partitioning



(l) CSV-Predicate-based Partitioning

Figure 6: ExtVP Vs. VP schemata performance using different partitioning techniques and file formats

optimizations. Only 78% of the queries results conform with the fact that WPT is better than the PT schema (Table 5).

Figure 5 aims to analyze the schemas performance when the solution adopts different partitioning techniques and file formats. Figures 5 (a-h) show clearly the effect of partitioning techniques on the reproducibility of the WPT optimizations across all the different file formats. For instance, notably the horizontal partitioning (Figures 5 (a,c,e,g)) affected the performance of WPT, making its performance in SparkSQL worse than the baseline PT schema in most of the queries (i.e., $Q1$,$Q3$,$Q5$,$Q6$,$Q8$,$Q11$). Similarly, we can observe the negative effect of the subject-based technique on WPT schema (Figures 5 (b,d,f,h)) in the same queries.

The impact of file formats aside Parquet is even worse. Even using the baseline (HDFS) partitioning technique affects the reproducibility of the WPT schema optimizations. Overall, only 58% of the query results conforming with the fact that WPT is outperforming PT schema (Table 5). The experiments show that columnar file formats, e.g., ORC, and Parquet, are the best for representing such wide tables (WPT and PT). Columnar file formats are the best for sparse queries (i.e., queries with few column projections or columns to access) out of the wide tables. They perform better than the row-oriented file formats, e.g., CSV and Avro, which would be only better with queries that require full rows reading.

Figure 5 shows the performance degradation considering different file formats. For instance, moving from Parquet and ORC in Figures 5 (e-h) to other row-oriented file formats such as Avro and CSV in Figures 5 (a-d), we can notice the performance degradation of the queries with the WPT schema optimizations.

6.2 Assumption: ExtVP always outperforms VP

According to [22], we expect that ExtVP provides better or at least similar performance gains, as the queries are similar, and the number of SQL joins in the VP schema is equal to the ExtVP joins. Nevertheless, one should keep in mind that ExtVP improvements are mainly due to the original SPARQL query nature. It also

Table 9: Mapping the partitioning technique to the storage format best practices in WPT

	Avro	CSV	ORC	Parquet
Baseline-HDFS	**X**	**X**	✓*	✓**
Horizontal	**X**	**X**	✓	✓
Subject-based	**X**	**X**	✓	✓

Where ✓is good practice, **X** is bad practice,and ˜ has the same performance compared to PT.
* WPT had very competitive performance
** WPT had the best performance

Table 10: Mapping the partitioning technique to the storage format best practices in ExtVP

	Avro	CSV	ORC	Parquet
Baseline-HDFS	✓	✓	˜	✓*
Horizontal	**X**	**X**	**X**	**X**
Subject-based	**X**	**X**	**X**	**X**
Predicate-based	**X**	**X**	✓	✓

Where ✓is good practice, **X** is bad practice, and ˜ has the same performance compared to VP.
* ExtVP had a very competitive performance

depends on the possible reductions in the table input data size and excluding the dangling triples (rows that do not contribute to any joins) [22]. Typically, ExtVP queries are similar to the VP ones; the only difference realizes in the queried tables/DataFrames (i.e., their size reduced by ExtVP or their size are the same VP). Thus, the relational engine's performance, e.g., Spark with the ExtVP, should be equivalent or better to its performance with the VP schema.

Based on our experiments, the findings of the ExtVP schema are not fully reproduced, even considering the default HDFS partitioning and the baseline Parquet file format. Some queries do not benefit from the ExtVP optimizations ($Q9$, $Q10$, $Q11$), notable input size reductions occurred in those queries), cf. Table 6. Beyond those queries, we can confirm that the state-of-the-art results (ExtVP performs better than VP in most cases). However, our results show that the schema-based query optimization is not straightforward in such a complex solution space.

Regarding the partitioning techniques, using an alternative to the baselines technique (HDFS) affects the reproducibility of the ExtVP optimizations even if the storage format is Parquet. Only 55% of the queries results show that ExtVP is superior to the VP schema (cf. Table 8). Moreover, Figure 6 shows the effect of other RDF partitioning techniques on the reproducibility findings of the ExtVP optimization. For instance, deviating from the baseline partitioning technique to other RDF-based techniques with the same baseline Parquet, i.e., Figures 6(a-c) degrades the results of ExtVP and makes it perform worse than the baseline VP schema in several queries ($Q1$, $Q4$, $Q5$, $Q6$, $Q8$) with the Horizontal and Subject-based partitioning. The predicate-based partitioning in Figure 6(c) has a better performance with this schema, which has performance close to VP's in the previously-mentioned queries.

Similarly, using storage formats different from Parquet affects the ExtVP optimizations' reproducibility, even with the baseline (HDFS) partitioning technique. Indeed, we have only 67.5% of the queries results of ExtVP outperforming VP (cf. Table 8). Similarly, Figure 6 shows the effect of other file formats other than the baseline Parquet, i.e Figures 6 (d-l) for ORC, Avro, and CSV respectively. We can notice the queries' performance degradation with the ExtVP schema optimizations moving vertically to these other formats.

Finally, from our experiments, we observe that columnar file formats are better than the Row-oriented ones. However, the performance difference is not significant with such similar schemas. The table structure is the same table of two columns Predicate (Subject-Object) in both vertical schemas. Moreover, both schemas have not wide tables in comparison to the WPT and PT schemas. That is, these schemas will not benefit a lot from the columnar file formats. The performance gain of columnar over the row-oriented file formats is because SP2Bench queries have a few numbers of column projections. Thus, it would work better with columnar rather than row-based file formats.

6.3 Recommendations

Overall, Tables 9 and 10 provides an abstracted map of good and bad storage format and partitioning techniques.

The results in Figure 5 and Table 9, show that partitioning the WPT table has, in the majority, a negative effect on the WPT optimization, making it perform even worse than its baseline approach, i.e, the PT schema. The effect of the storage formats is more significant in the WPT optimization (cf. Tables 5, 9). Therefore, this WPT schema's storage format selection decision should be dealt with as a first-class citizen in such experiments.

The horizontal and subject-based partitioning techniques are not recommended with ExtVP optimization. However, Predicate-based still gives better results than those two other RDF partitioning techniques (cf. Tables 8 and 10). Also, columnar file formats are still recommended with the ExtVP schema optimization. However, it was noticed that the effect of the partitioning is more significant to this optimization (cf. Figure 6, Tables 8, and 10). Thus, the partitioning selection decision of this ExtVP schema should be highly considered in these experiments.

Also, our analysis yields the following recommendations

(1) With WPT, it is recommended to use the columnar storage formats rather than row-oriented ones (cf. Table 9).
(2) With the WPT schema, Parquet is yet the best columnar file format to select, it efficiently handles its sparsity.
(3) With WPT, it is recommended to use the native HDFS partitioning, rather than selecting an RDF-oriented partitioning technique.
(4) With ExtVP, the baseline HDFS partitioning is more recommended than specific RDF ones. However, larger datasets would require partitioning anyway.
(5) With ExtVP, the columnar file formats is a recommended optimization.

7 RELATED WORK

In this section, we present the related work. In particular, we focus on comparative studies that investigate the use of BD frameworks for distributed RDF processing. To the best of our knowledge, the literature includes several studies that compare partitioning techniques, relational schemas, and storage formats [2, 6, 8, 14]. However, none of these approaches focus on replicating and comparing existing optimization techniques.

Abdelaziz et al. [2] discussed several relational schemas for materializing RDF datasets. Their main goal was to assess different *native* and *non-native* RDF processing systems. However, it does not discuss the impact of different relational schemas on a specific system's performance, such as SparkSQL; nor it discusses partitioning techniques and data formats.

Arrascue et al. [6] lead an investigation on the performance of the WPT schema against alternative relational schemas, i.e., triple tables, VP, and domain-dependent tables. Additionally, they consider subject-based partitioning but limit the data formats to Parquet. The work's main finding is the flexibility of WPT for generic query shapes in contrast with other approaches and even considering partitioning. However, their exploration of the solution space is limited in terms of partitioning techniques and data formats.

Cossu et al. [8] focused on a hybrid storage approach that combines the benefits of PT and VP schemas to boost the query performance without the need for extensive loading time. Their solution, *PROST*, was able to outperform state of the art systems like S2RDF for several query shapes. Nevertheless, their exploration of partitioning techniques and data formats is limited. Additionally, they focused their work on PT and VP schemas, not considering WPT as an alternative schema that may further improve the performance.

On another side, Pham et al. results in [14] indicates that more than 95% of RDF dataset triples have tabular structure. They combine structural non-quotient and statistical methods to automatically discover and detect an emergent relational schema (in the form of property tables) in RDF datasets. A similar approach has been proposed in [12] to mitigate the limitations of the WPT and PT RDF schemata by merging the related hierarchical *characteristic sets* and provide a novel RDF relational schema. The aim of so doing is to provide a better SPARQL query evaluation.

Finally, *Akhter et al.* [4], investigated the performance of different partitioning techniques for RDF data, proposing a ranking function that helps practitioners to choose the most appropriate technique.

8 CONCLUSIONS & FUTURE WORK

The reproducibility of well-known relational RDF processing optimizations is critical to foster best practices that guide the practitioners' efforts. In this paper, we presented a comprehensive empirical evaluation using three RDF partitioning techniques and four storage formats over the distributed SparkSQL engine to cope with this limitation. Our analysis demonstrates decisively variant trade-offs using different relational schemas, data partitioning, and storage file formats against these state-of-the-art optimizations. Our experiments show significant degradation in Spark performance when partitioning by subject in the WPT and partitioning horizontally due to the vast, sparse, and large partitions of its schema table. On the same note, the storage format also affects the WPT performance, where ORC and Parquet are the most suitable representation of such configuration. Our results on ExtVP illustrate that schema-based query optimization is not straightforward using different configurations.

Future work includes extending this study by analyzing the impact of data scalability on SparQL performance. We intend to utilize other RDF benchmarks such as *WatDiv* with different types of query shapes and complexities. Our plans include investigating this area further to design a benchmark that combines query workloads with precise partitioning and storage instructions.

REFERENCES

[1] Daniel J Abadi, Adam Marcus, Samuel R Madden, and Kate Hollenbach. 2007. Scalable semantic web data management using vertical partitioning. In *VLDB*.

[2] Ibrahim Abdelaziz, Razen Harbi, Zuhair Khayyat, and Panos Kalnis. 2017. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *Proceedings of the VLDB Endowment* 10, 13 (2017), 2049–2060.

[3] Giannis Agathangelos, Georgia Troullinou, Haridimos Kondylakis, Kostas Stefanidis, and Dimitris Plexousakis. 2018. RDF Query Answering Using Apache Spark: Review and Assessment. In *34th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2018, Paris, France, April 16-20, 2018*. IEEE Computer Society, 54–59.

[4] Adnan Akhter, Axel-Cyrille Ngomo Ngonga, and Muhammad Saleem. 2018. An empirical evaluation of RDF graph partitioning techniques. In *European Knowledge Acquisition Workshop*. Springer, 3–18.

[5] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *SIGMOD Conference*. ACM, 1383–1394.

[6] Victor Anthony Arrascue Ayala, Polina Koleva, Anas Alzogbi, Matteo Cossu, Michael Färber, Patrick Philipp, Guilherme Schievelbein, Io Taxidou, and Georg Lausen. 2019. Relational schemata for distributed SPARQL query processing. In *Proceedings of the International Workshop on Semantic Big Data*. 1–6.

[7] Feras M Awaysheh, Mamoun Alazab, Maanak Gupta, Tomás F Pena, and José C Cabaleiro. 2020. Next-generation big data federation access control: A reference model. *Future Generation Computer Systems* (2020).

[8] Matteo Cossu, Michael Färber, and Georg Lausen. 2018. PRoST: Distributed Execution of SPARQL Queries Using Mixed Partitioning Strategies. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose (Eds.). OpenProceedings.org, 469–472. https://doi.org/10.5441/002/edbt.2018.49

[9] J. Huang, D. Abadi, and K. Ren. 2011. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment* 4 (2011), 1123 – 1134.

[10] Todor Ivanov and Matteo Pergolesi. 2019. The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet. *Concurrency and Computation: Practice and Experience* (2019), e5523.

[11] Todor Ivanov and Matteo Pergolesi. 2020. The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet. *Concurr. Comput. Pract. Exp.* 32, 5 (2020). https://doi.org/10.1002/cpe.5523

[12] Marios Meimaris, George Papastefanatos, and Panos Vassiliadis. 2020. Hierarchical Property Set Merging for SPARQL Query Optimization.. In *DOLAP*. 36–45.

[13] Thomas Neumann and Gerhard Weikum. 2010. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal* 19, 1 (2010), 91–113.

[14] Minh-Duc Pham, Linnea Passing, Orri Erling, and Peter A. Boncz. 2015. Deriving an Emergent Relational Schema from RDF Data. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 864–874. https://doi.org/10.1145/2736277.2741121

[15] Mohamed Ragab, Riccardo Tommasini, Sadiq Eyvazov, and Sherif Sakr. 2020. Towards making sense of Spark-SQL performance for processing vast distributed RDF datasets. In *Proceedings of The International Workshop on Semantic Big Data*. 1–6.

[16] Mohamed Ragab, Riccardo Tommasini, and Sherif Sakr. 2019. Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends. In *QuWeDa@ISWC*.

[17] Sherif Sakr. 2009. GraphREL: A Decomposition-Based and Selectivity-Aware Relational Framework for Processing Sub-graph Queries. In *DASFAA*.

[18] Sherif Sakr and Ghazi Al-Naymat. 2010. Relational processing of RDF queries: a survey. *ACM SIGMOD Record* 38, 4 (2010), 23–28.

[19] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A Boncz, et al. 2020. The Future is Big Graphs! A Community View on Graph Processing Systems. *arXiv preprint arXiv:2012.06171* (2020).

[20] Muhammad Saleem, Gábor Szárnyas, Felix Conrads, Syed Ahmad Chan Bukhari, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2019. How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks?. In *The World Wide Web Conference*. ACM, 1623–1633.

[21] Alexander Schätzle, Martin Przyjaciel-Zablocki, Antony Neu, and Georg Lausen. 2014. Sempala: Interactive SPARQL query processing on hadoop. In *International Semantic Web Conference*. Springer, 164–179.

[22] Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, and Georg Lausen. 2016. S2RDF: RDF querying with SPARQL on spark. *Proceedings of the VLDB Endowment* 9, 10 (2016), 804–815.

[23] Michael Schmidt, Thomas Hornung, Norbert Küchlin, Georg Lausen, and Christoph Pinkel. 2008. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In *International Semantic Web Conference (Lecture Notes in Computer Science)*, Vol. 5318. Springer, 82–97.

[24] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. 2009. SPˆ2Bench: A SPARQL Performance Benchmark. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*. 222–233. https://doi.org/10.1109/ICDE.2009.28

[25] Matei Zaharia, Reynold S. Xin, and Patrick Wendell et.al. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.

[26] Lei Zou, Jinghui Mo, Lei Chen, M Tamer Özsu, and Dongyan Zhao. 2011. gStore: answering SPARQL queries via subgraph matching. *Proceedings of the VLDB Endowment* 4, 8 (2011), 482–493.