Comparing Schema Advancements for Distributed RDF Querying Using SparkSQL

Mohamed Ragab, Riccardo Tommasini, and Sherif Sakr*

Institute of Computer Science, Tartu University, Tartu, Estonia
firstName.lastName@ut.ee

Abstract. Linked Data reveals the need for big semantic data processing. The underlying literature already discusses numerous attempts at leveraging the relational engines of Big Data frameworks like Apache Spark to run SPARQL queries at scale. However, the choice of a relational schema to store RDF data may significantly impact the query performance and hence various alternatives exist. In this paper, we investigate the improvement of two recent proposals, i.e., *Extended Vertically Partitioned Tables* and *Wide Property Tables*, w.r.t. the baseline approaches *Vertically Partitioned Tables* and *Property Tables*. To generalize our results, we observe how the two schemas behave together with different RDF partitioning techniques and HDFS storage data formats. We run our experiments using SparkSQL over a 100-million triples dataset generated using SP^{2}Bench.

Keywords: RDF Relational Schemata · Spark-SQL · SPARQL.

1 Introduction

[1]The Semantic Web community is investigating how to leverage frameworks like Apache Spark to run SPARQL queries at scale. Since Big Data frameworks excel in relational data analysis, several solutions have been proposed to utilize their query engines for processing large RDF graphs [1]. Intuitively, the choice of a relational schema significantly impacts the performance of query processing. For instance, the ***Single Statement Table Schema (ST)***, which prescribes to store triples using a ternary relation (subject, predicate, object), often requires many self-joins. Many alternatives to ST exist, i.e., The (i) ***Vertically Partitioned Tables Schema (VP)*** proposes to use binary relations (subject, object) for each unique *predicate* in the dataset. The (ii) ***Property Table Schema (PT)*** suggests *n-ary* relations to represent RDF triples, grouping those with the same subject.

In this paper, we aim at validating two further improvements proposed for Apache Spark, i.e., the Extended Vertically-Partitioned Table (ExtVP), which extends VP with precomputed *semi-join* tables [4] to reduce data shuffling, and the Wide Property Table (WPT) schema, which extends PT considering the

* In memoriam, (1979-2020)

whole dataset in a single table [3], and minimizes the number of joins. We performed an extensive comparative evaluation of ExtVP and WPT vs VP and PT respectively using SparkSQL. To generalize the study, we check the approaches under varying experimental conditions[2]: we tested how ExtVP and WPT perform combined with RDF-based partitioning techniques and storage formats. Partitioning techniques impact the query execution as they change data locality. On the other hand, data formats are considered by the Spark optimizer and, thus, impact the query plan. In particular, we used the following partitioning techniques (i) *Horizontal (HO)* partitioning, which divides data evenly over n equivalent chunks where n is number of machines in the cluster; (ii) *Subject-based (SB)* and (iii) *Predicate-based (PB)* partitioning, which distribute triples across the various partitions according to the *hash value* computed for the *subjects* or *predicate*, respectively. Moreover, we use two row-oriented data formats, i.e., CSV, Avro, and two column-oriented ones, i.e., ORC, and Parquet. As a baseline configuration, we chose the one of the original papers: we store data in HDFS using Parquet without any specific partitioning technique, i.e. No_Partitioning(NP).

2 Experiments

In our evaluation, we used data and queries from `SP`^{2}`Bench` (SPARQL Performance Benchmark) . We prepared the SQL versions of the `SP`^{2}`Bench` queries[3] for ExtVP and WPT[4]. We generated a *synthetic* RDF dataset with *100M* triples size in `Notation3` format, which was sufficient for unveiling differences in the query execution under different experimental conditions. We run the experiments *five* times and computed an average[5] on a four-nodes *bare-metal* cluster (master node and 3 worker machines). Each node runs has 32 cores, 128GB of RAM per node, and 2-TB *SSD* drive.

2.1 WPT VS. PT

According to [3, 2], we expect that WPT outperforms PT especially with the *"Star-Shaped"* queries, which can be answered without any join operations when using WPT schema. Table 1 compares the required number of joins for the alternative SQL translations of `SP`^{2}`Bench` queries when translated. Except for Q8 that has many self-joins of the WPT table, WPT always requires fewer joins than PT. Moreover, we expect that Spark handles efficiently the sparsity caused by the WPT schema when using Parquet data format, since it ignores *Null* values.

	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 1: Number of SQL joins in PT vs WPT.

Table 2 shows the overall benchmark results of the performance of WPT over PT across all file formats (Horizontally), and across the different partitioning

[2] https://www.nist.gov/pml/nist-technical-note-1297/nist-tn-1297-appendix-d1-terminology
[3] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B/queries.php
[4] https://github.com/DataSystemsGroupUT/SPARKSQLRDFBenchmarking
[5] We excluded the *first* run to avoid *warm-up* bias

techniques (Vertically). Values in this table specify the number of queries in which the WPT schema gives performance better than the baseline PT schema[6]. The experiments confirm that WPT outperforms the baseline PT schema in all the queries (i.e 9 queries out of 9 in the benchmark) using the baseline configuration, i.e., Parquet file format and No partitioning technique (NP). These results confirm the state-of-the-art findings [3].

	Avro	***CSV***	***ORC***	***Parquet***
NP	2/9	2/9	8/9	9/9
Ho	2/9	3/9	6/9	6/9
SP	2/9	2/9	6/9	6/9

Table 2: Number of queries for which WPT outperforms PT.

To investigate the relations between WPT and PT, we introduce different file formats and partitioning techniques. Table 3 shows the effect of data partitioning (left) and storage formats (right) considering the other new factors across all the experiments. To this extent, we calculate the percentages by grouping the experiment by partitioning technique and counting how many times WPT is better than PT across all the experiments. We calculated the storage effect in a similar way but grouping by file format.

Table 3 shows that WPT outperforms PT schema for 58% of the experiments when No partitioning technique is used. Moreover, only in 78% of the experiments, using Parquet as file format as an improvement. This unveils a trade-off between file formats, partitioning techniques, and the relational schemata. ORC, which is an alternative column-based file format, gives results that are similar to Parquet(74%). Although Parquet is even better because it efficiently handles the sparsity of the WPT schema, we can generalize the benefits of column-based file formats. Indeed, SP^{2}Bench queries only have one query with more than 2-column projections, which justifies why columnar formats give better results for the WPT than the row-based ones. Row-oriented formats have a negative impact on the WPT results. WPT outperforms PT only for 22% and 25% experiments when Avro and CSV are used, respectively. In conclusion, the state-of-the-art results for WPT cannot be reproduced in the presence of different formats and partitioning techniques.

Partitioning effect		**Storage Formats effect**	
NP	58.33%	***Parquet***	77.78%
Ho	47.22%	***ORC***	74.07%
SP	44.44%	***CSV***	25.93%
PB	NA	***AVRO***	22.22%

Table 3: Effects of partitioning techniques and storage formats on the WPT/PT comparison.

2.2 ExtVP VS. VP

In this section, we discuss the comparison of the ExtVP schema to VP schema. For the comparison, we used the same approach followed in the previous section. According to [4], we expect that ExtVP provides better or at least similar execution times of VP. The enhancement depends on the percentage of semi-join reductions of table input sizes that

	Q1	Q2	Q3	Q4	Q5	Q6	Q8	Q9	Q10	Q11
Red	58%	77%	59%	96%	60%	31%	5%	0%	0%	0%

Table 4: [Red]uctions percentage using ExtVP.

[6] *Green*: improvement always outperforms the baseline; *Yellow*: improvement outperforms the baseline in at least 50% of the cases, and *Red* means less than 50%.

the ExtVP introduces, which reduce the required data shuffling. Table 4 shows the percentage of ExtVP reductions in the processed rows for each query over the original input table processed rows. We expect queries $Q9$,$Q10$,and $Q11$ giving similar results to the VP schema as they do not present any input table reductions.

Table 5 shows the comparison between ExtVP and the baseline VP schema performance in all the SP^{2}Bench queries, for different formats (horizontally) and partitioning techniques (vertically)[5]. We observe that for queries Q9,Q10,Q11, which did not benefit from any join reductions, ExtVP does not beat VP even for the baseline configuration (NP and Parquet).

	Avro	**CSV**	**ORC**	**Parquet**
NP	6/10	6/10	5/10	7/10
Ho	3/10	3/10	3/10	3/10
PB	2/10	3/10	6/10	6/10

Table 5: Number of queries for which ExtVP outperforms VP.

Table 6 also shows how far the data partitioning (left) and data formats (right) impact the results of ExtVP in comparison to VP schema performance. Percentages are calculated in the same as in Table 3, pivoting on the dimension of choice, i.e., file format X or partitioning technique Y.

The results confirm that partitioning significantly degrades the performance of ExtVP, which outperforms VP schema in 67% of the experiments for the NP configuration. However, adopting predicate-based partitioning has a less negative impact (55%), then horizontal partitioning (35%), followed by the subject-based partitioning; the worst with only 30%.

Partitioning effect	
NP	67.5%
Ho	35%
PB	55%
SP	30%

Storage Formats effect	
Parquet	55%
ORC	45%
AVRO	42.5%
CSV	42.5%

Table 6: Effects of partitioning techniques and storage formats on the ExtVP vs VP comparison.

The small number of projections in SP^{2}Bench queries suggests that columnar file formats can fit such query workloads better than the row-oriented ones. In 55% of the cases where Parquet is used ExtVP outperforms VP, while only 45% with ORC. Nevertheless, ExtVP beats VP in only 42.5% of the experiments that adopt either Avro or CSV. In conclusion, we cannot reproduce completely the state-of-the-art results for ExtVP when different partitioning techniques or file formats are used.

3 Conclusion

In this paper we investigated the reproducibility of RDF relational optimizations within distributed Spark-SQL while introducing complex experimental solution space. The optimized relational schemata can be affected with new experimental factors such as the data partitioning or the storage data formats.

References

1. Abdelaziz, I., Harbi, R., Khayyat, Z., Kalnis, P.: A survey and experimental comparison of distributed sparql engines for very large rdf data. Proceedings of the VLDB Endowment (2017)

2. Arrascue Ayala, V.A., Koleva, P., Alzogbi, A., Cossu, M., Färber, M., Philipp, P., Schievelbein, G., Taxidou, I., Lausen, G.: Relational schemata for distributed sparql query processing. In: International Workshop on Semantic Big Data (2019)
3. Schätzle, A., Przyjaciel-Zablocki, M., Neu, A., Lausen, G.: Sempala: Interactive sparql query processing on hadoop. In: ISWC (2014)
4. Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., Lausen, G.: S2rdf: Rdf querying with sparql on spark. Proceedings of the VLDB Endowment (2016)