University of Freiburg
Faculty of Engineering, Departement of Computer Science

Master Project Report

Extending Sempala project with Extended Vertical Partition Multi Table Layout

Lis Bakalli

March, 2017

Contents

1 Introduction

This project extends the existing system of Sempala with Extended Vertical Partition layout. Sempala is a SPARQL-over-SQL-on-Hadoop system which translates SPARQL queries into SQL and executes them on *Impala*[1]. Impala is a distributed open source MPP query engine which runs on top of Hadoop. Sempala consists of two components, namely the loader and query compiler. The loader prepares the data layout from the RDF datasets, while the query compiler translates the SPARQL query into SQL[3].

The idea of Extended Vertical Partitioning (ExtVP) is to precompute a set of semi-join tables, which will be smaller in size and used as input tables to query over them. The data in *RDF* are in form of triples (*subject* S , *predicate* P, *object* O). With ExtVP we try to find correlation between S and O of triples with different P, therefore we have four types of ExtVP based on this correlation, namely SS, SO, OS and OO. ExtVP tables give the advantage in case of joins due to the use of smaller tables instead of the initial big tables. ExtVP tables contain only the data that are part of the result after the semi-join. We are ensured about this due to the use of semi-joins for creating them, which also give smaller tables or at most equal as initial ones[1].

ExtVP layout was previously implemented by Simon Skilevic as part of his Master Thesis in a system called S2RDF. S2RDF similarly to Sempala translates the SPARQL queries to SQL but executes them on Spark. ExtVP model made S2RDF perform faster than other competitors, but the downside was the high amount of time required for creation of ExtVP tables[4]. Therefore, beside implementation of ExtVP on another query engine such as Impala and comparing it with the layout on Spark, our aim was also to optimize the creation of ExtVP tables.

[1]https://impala.incubator.apache.org/

Previously Sempala has had the data layouts of Property Table and Single Table. In Property Table the data are stored in a giant table where for each subject all properties are stored in this table and objects are presented as values of these properties[3]. On the other hand, Single Table is a modification of ExtVP where the data of ExtVP are stored in a single big table[2].

The new layout of ExtVP in Sempala is evaluated and compared to the other competitor layouts and systems. In Chapter 2 we will present the implementation and optimization of ExtVP layout in both components of Sempala, namely Loader and Translator. In Chapter 3 we will discuss about the process of evaluation, the results of evaluation and comparison with competitors. In Chapter 4 we will give the conclusion regarding Extended Vertical Partitioning on Sempala.

2 Implementation

In the previous chapter we elaborated the architecture of Sempala. In order for the new layout to be added, changes had to be done in both loader and translator component of the system. The good structure of code in Sempala allowed us to extend the system with ExtVP layout without making to much changes in the existing classes. The class $ExtVPLoader.java$ was created to implement the ExtVP layout for the loader. Also changes had to be done in class $Main.java$ and other classes, where new layout options had to be integrated and previous parts of code had to be adjusted to function also with ExtVP layout.
Regarding the translator class, $ImpalaBgpExtVPMultiTable.java$ was created for implementing the ExtVP table selection and translation of SPARQL Query. Also for translator other classes had to be modified for new options of the new layout to be added. In the following sections we will discuss more about the implementation done for each of the components.

2.1 Sempala Loader

As mentioned earlier the idea behind Extended Vertical Partitioning (ExtVP) is precomputation of ExtVP tables to query over them. To compute these we first create the triple table which is the table consisting of all tuples from N-Triple data which are located in HDFS. The triple table is partitioned by predicate and stored as parquet file which is a columnar based compression storage[1]. This way when tuples with a specific predicate are read, instead of scanning the whole table, only the partition containing these tuples is scanned. Meanwhile in S2RDF these data are stored in separate tables called Vertical Partition (VP) Tables. Each VP contained all tuples with a specific predicate, this way creating for each predicate a VP table[4].

[1]https://parquet.apache.org/

After creating triple table the creation of ExtVP tables can begin. Previously in S2RDF calculations for type OO where omitted due to believe of low gain in overall result[4]. In our case all four types are calculated since we believe there is also benefit from ExtVP table of type OO.

Algorithm 1: Computation of Extended Vertical Partition Tables for type SO and OS.

input : TripleTable TT, List of Predicates LP, Selectivity Threshold SF
output: ExtVP Tables $ExtVP^{SO,OS}_{p_i,p_j}$

1 **foreach** $p_i \in LP$ **do**
2 **foreach** $p_j \in LP$ **do**
3 $ExtVP^{SO}_{p_i|p_j} \leftarrow TT_{p=p_i} \ltimes_{s=o} TT_{p=p_j}$
4 **if** $ExtVP^{SO}_{p_i|p_j} \neq \emptyset$ **then**
5 $ExtVP^{OS}_{p_j|p_i} \leftarrow TT_{p=p_j} \ltimes_{o=s} ExtVP^{SO}_{p_i|p_j}$
6 **if** $\frac{ExtVP^{SO}_{p_i|p_j}.size}{TT_{p=p_i}.size} > SF$ **then**
7 DROP $ExtVP^{SO}_{p_i|p_j}$
8 **if** $\frac{ExtVP^{OS}_{p_j|p_i}.size}{TT_{p=p_j}.size} > SF$ **then**
9 Drop $ExtVP^{OS}_{p_j|p_i}$
10 **else**
11 DROP $ExtVP^{SO}_{p_i|p_j}$
12 **if** $p_i \neq p_j$ **then**
13 $ExtVP^{OS}_{p_i|p_j} \leftarrow TT_{p=p_i} \ltimes_{o=s} TT_{p=p_j}$
14 **if** $ExtVP^{OS}_{p_i|p_j} \neq \emptyset$ **then**
15 $ExtVP^{SO}_{p_j|p_i} \leftarrow TT_{p=p_j} \ltimes_{s=o} ExtVP^{OS}_{p_i|p_j}$
16 **if** $\frac{ExtVP^{OS}_{p_i|p_j}.size}{TT_{p=p_i}.size} > SF$ **then**
17 DROP $ExtVP^{OS}_{p_i|p_j}$
18 **if** $\frac{ExtVP^{SO}_{p_j|p_i}.size}{TT_{p=p_j}.size} > SF$ **then**
19 Drop $ExtVP^{SO}_{p_j|p_i}$
20 **else**
21 DROP $ExtVP^{OS}_{p_i|p_j}$
22 **end**
23 **end**

Algorithm 1 presents the calculation for ExtVP tables of type SO and OS between two predicates p_i and p_j. As explained earlier we compute ExtVP table using the semi-join operation. For $ExtVP^{SO}_{p_i,p_j}$ we compute `LEFT SEMI JOIN` between the partition of triple table where $p = p_i$ and partition of triple table where $p = p_j$, on condition $s = o$. After calculating $ExtVP^{SO}_{p_i,p_j}$ the statistics for this table are

computed using `COMPUTE STATS`. These statistics are used later by Impala query planer for join ordering.

After computing $ExtVP^{SO}_{p_i,p_j}$ the next computation is the so called opposite ExtVP which is $ExtVP^{OS}_{p_j,p_i}$. For computing $ExtVP^{OS}_{p_j,p_i}$ instead of using again the full partitions of triple tables, we use the previously calculated $ExtVP^{SO}_{p_i,p_j}$ as the right table in the left semi join operation. This way since every tuple in $ExtVP^{SO}_{p_i,p_j}$ has found an element where $s = o$ in the previous calculation, we are assured that there will not exist a tuple of $ExtVP^{SO}_{p_i,p_j}$ that is not a partner for at least one of the tuples of the left table. This way we reduce the size of the right table, removing all unnecessary tuples which will not find joining partners. The same algorithm and idea of using the previously calculated ExtVP tables is used for all four types SO, OS, SS and OO.

After each pair of opposite ExtVP tables is calculated, the condition for selectivity is checked weather the created tables fulfill it. Tables which have a higher value of selectivity than the threshold defined by the user are dropped and the rest remain in the database.

Although the creation of ExtVP tables is executed once before and not on the fly, still the time needed to be completed in S2RDF was to much, therefore we focused in optimizing this process in order to reduce the time. In order to reduce the number of calculations we try to eliminate unnecessary calculations. First possible set of calculations which can be eliminated are those of ExtVP tables of type SS and OO for same predicates. When two predicates of ExtVP are the same, the calculations for SS and OO are skipped (e.g $ExtVP^{SS}_{p_i,p_i}$). This comes from the fact that both tables of the semi join are the same, so for every tuple of the left table there will exist at least one join partner in the right table (same tuple), leading to the same input table also as output.

Another case where we can reduce the number of calculations are those of empty tables. If we calculate an ExtVP table between two predicates which results in an empty table, based on the idea of using previously calculated ExtVP tables to calculate the opposite ExtVP table, we can conclude that also the opposite table

of the empty ExtVP table is empty. This is because the semi join between a table and an empty table is always an empty table. Therefore, calculation of the second empty table is skipped. This way for each set of empty tables half of them are not calculated at all. As we will see later in the evaluation a huge amount of ExtVP tables between predicates of datasets are empty tables, making this optimization very important in improving performance.

The calculation of a single ExtVP table between two predicates usually is done in a few milliseconds or in a few seconds. Therefore, the distribution of the workload over the working nodes of the cluster is not very high. To increase the workload on all the nodes of the cluster, we implemented an option in the loader to distribute the set of predicates for which the ExtVP tables are calculated to each of the nodes. This way determining for each working node which ExtVP tables are calculated. The total number of calculations in the general case without skipping empty tables for n predicates is equal to:

$$\sum_{i=1}^{n}(2 + 8(n - i))$$

Based on the total number of calculations, we distributed the number of predicates in the way that each of the working nodes in the cluster would get approximately the same number of calculations. This distribution of workload had a huge impact in the performance of the loader, reducing the creation time very much. However, distributing the workload based on total number of calculations is not the optimal distribution. This drawback will be seen and discussed later in the evaluation part.

In Figure 2.1 we can see the list of options offered by Sempala Loader during execution in ExtVP format. The new options added to Sempala Loader for ExtVP format are:

- **ExtVP Types (-e)**: This option allows the user to define the types of ExtVP tables that should be computed. By default all four types of ExtVP tables are computed.
- **Evaluation Mode (-em)**: Under this option, the threshold given by the user is not taken into account and all non-empty ExtVP tables that are computed

are stored.

- **List of Predicates (-lp)**: This option makes it possible for the user to set a list of predicates for which the ExtVP tables are computed. By default, ExtVP tables are computed for all predicates.
- **Predicate Partition (-pp)**: This option is used mainly for the case of workload distribution explained earlier. With this option the set of predicates to be computed by the worker node is determined.
- **Threshold (-t)**: Threshold is the value of selectivity decided by the user to define which ExtVP tables are stored. By default the value of Threshold is one.

Other functionalities added to the ExtVP layout are table of statistics created for all ExtVP types, and recovery option where in case of failures or stoppage, the creation process continues from the predicate where it stopped instead of starting from the beginning.

```
-co,--column-name-object <arg>      Overwrites the column name to use. (object)
-cp,--column-name-predicate <arg>   Overwrites the column name to use. (predicate)
-cs,--column-name-subject <arg>     Overwrites the column name to use. (subject)
-d,--database <arg>                 The database to use.
-e,--extvp-types <arg>              Formats of ExtVP to be computed. By default all four formats of
                                    ExtVP (SS/SO/OS/OO) are computed
-em,--evaluation-mode               Executes Sempala in Evaluation Mode
-f,--format <arg>                   The format to use to create the table. (case insensitive)
                                    simple_property_table: (see 'Sempala: Interactive SPARQL Query
                                    Processing on Hadoop')
                                    complex_property_table: (see Polina and Matteo's Master project
                                    paper)
                                    extvp: see Extended Vertical Partitioning, Master's Thesis:
                                    S2RDF, Skilevic Simon
                                    single_table: see ExtVP Bigtable, Master's Thesis: S2RDF,
                                    Skilevic Simon
-F,--field-terminator <arg>         The character used to separate the fields in the data.
                                    (Defaults to '\t')
-h,--help                           Print this help.
-H,--host <arg>                     The host to connect to.
-i,--input <arg>                    The HDFS location of the RDF data (N-Triples).
-k,--keep                           Do not drop temporary tables.
-L,--line-terminator <arg>          The character used to separate the lines in the data. (Defaults
                                    to '\n')
-lp,--list-of-predicates <arg>      List of predicates over which the ExtVP tables will be created.
-o,--output <arg>                   Overwrites the name of the output table.
-p,--port <arg>                     The port to connect to. (Defaults to 21050)
-P,--prefix-file <arg>              The prefix file in TURTLE format.
                                    Used to replace namespaces by prefixes.
-pp,--predicate-partition <arg>     Subset of predicates for which extvp tables to be created.
                                    Default all predicates.
-s,--strip-dot                      Strip the dot in the last field (N-Triples)
-S,--shuffle                        Use shuffle strategy for join operations
-t,--threshold <arg>                Threshold of ExtVP if ExtVP format is selected. Default (SF=1)
-u,--unique                         Detect and ignore duplicates in the input (Memoryintensive!)
-ud,--user-hdfs-directory <arg>     User's Absolut path of HDFS direcotry (/user/<name>)
```

Figure 2.1: Sempala Loader Options

2.2 Sempala Translator

Different from Single Table or Property Table layouts, in ExtVP layout the data are stored in multiple tables. The task of translator for ExtVP format is to parse the SPARQL query to SQL and select the best tables to query over them.

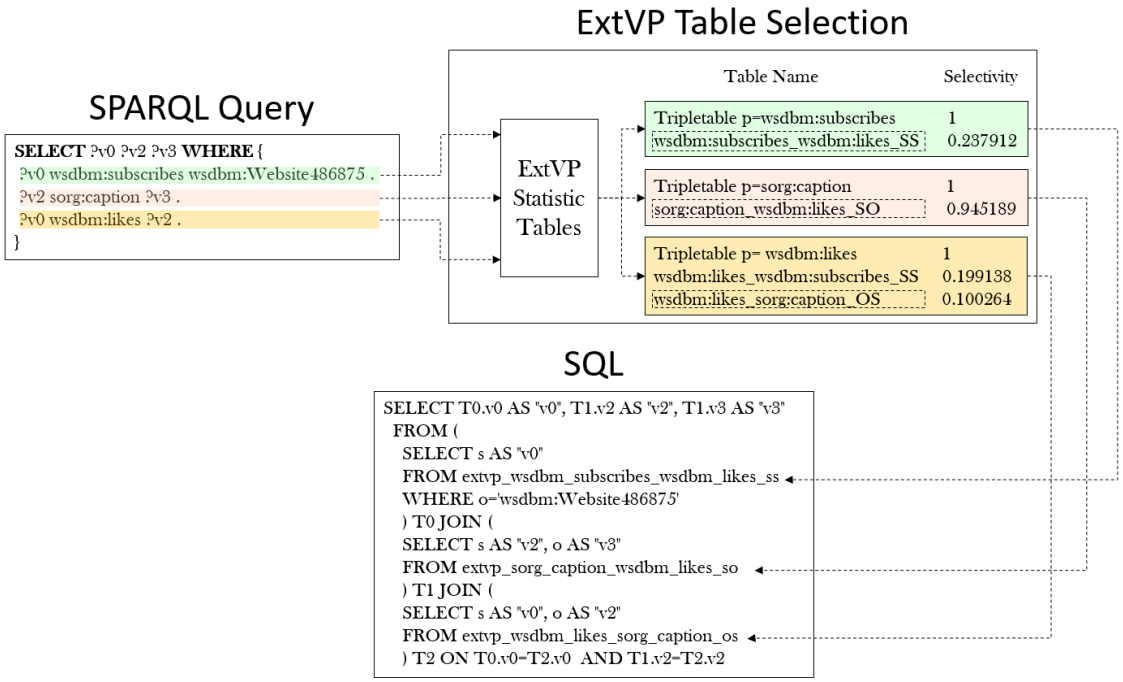

Figure 2.2: Sempala ExtVP SPARQL to SQL translator

As we can see in this example in Figure 2.2, for each triple pattern of the SPARQL query, a set of possible tables to choose from is created. This selection is implemented in $ImpalaBgpExtVPMultiTable.java$ where SPARQL query is taken together with selectivity threshold as inputs by the user. Here all triple patterns are compared between each other for possible subject-subject, subject-object, object-subject and object-object correlation.

Each of the existing correlation is translated into a possible ExtVP table. As we can see in the example, triple pattern *?v0 wsdbm:subscribes wsdbm:Website486875* has a subject-subject correlation with triple pattern *?v0 wsdbm:likes ?v2*, so this correlation is translated to table name "*extvp_wsdbm_subscribes_wsdbm_likes_SS*". For each of the table names, a query is executed over the table of statistics in the database to get the selectivity value of that table. These queries are fast and are

executed in a few milliseconds, but in cases where there are to many possible ExtVP tables to choose from, these queries add up increasing the total translation time. Since our focus was not the optimization of translation process, we did not go further into optimizing this process. One possible way of optimizing this process is to store locally the statistics of ExtVP tables once in the beginning and then get the selectivity of tables from there without querying each time.

The initial table for each of the triple patterns is the partition of the triple table corresponding to each triple pattern's predicate, which has a selectivity of one. The table with the lowest value of selectivity is selected to be used in the SQL format, but only if this value is also smaller then the threshold inputed by the user; otherwise, the triple table partition is used.

After selecting the best tables, each of the triple patterns is parsed into a SQL subquery. The join conditions between subqueries are listed based on their correlation between each other. Using the statistics which are calculated for each ExtVP table in the loader phase, Impala is able to optimize the query performance by constructing efficient query plans and join ordering.

```
-b,--benchmark                  Just print runtimes and delete results.
-c,--count                      COUNT result without storing the table.
-d,--database <arg>             The database to use.
-e,--expand                     Expand URI prefixes.
-f,--format <arg>               The database format the query is built for.
                                propertytable: (see 'Sempala: Interactive SPARQL Query Processing
                                on Hadoop')
                                complex_property_table: (see Polina and Matteo's Master project
                                paper) Impala Version
                                complex_property_table_spark: (see Polina and Matteo's Master
                                project paper) Spark Version
                                singletable: see ExtVP Bigtable, Master's Thesis: S2RDF, Skilevic
                                Simon
                                extvp: see Extended Vertical Partitioning, Master's Thesis: S2RDF,
                                Skilevic Simon
-h,--help                       Print this help.
-H,--host <arg>                 The host to connect to.
-i,--input <arg>                SPARQL query file to translate or folder containing sparql query
                                files.
-opt,--optimize                 turn on SPARQL algebra optimization
-p,--port <arg>                 The port to connect to. (Defaults to 21050)
-rn,--result_table_name <arg>   Result Table Name format if results tables are stored.
-s,--straighjoin                Executes query with Straight join. Default (Impala sets the join
                                order of tables)
-t,--threshold <arg>            Threshold of ExtVP if ExtVP format is selected. Default (SF=1)
```

Figure 2.3: Sempala Translator Options

Similar to the loader, also in Sempala translator new options added for ExtVP format:

- **Count (-c)**: This option allows the user execute the query by just counting the result and not storing them as a table. By default, if this option is not selected, Sempala translator executes the queries as CTAS queries (Create Table As Select).
- **Result Table Name (-rn)**: Allows the user to set the table name where the results are stored.
- **Straight Join (-s)**: This option allows the user to execute the query with `STRAIGHT JOIN`, this way revoking the right of Impala to change the join ordering.
- **Threshold (-t)**: Threshold is the value of selectivity decided by the user to define which ExtVP tables are allowed to be selected. By default the value of Threshold is one.

3 Evaluation

In this chapter we will present the overall evaluation of ExtVP layout in Sempala. The evaluation is done in a distributed environment which is a cluster of ten servers. These machines have the same physical resources, namely a processor Intel Xeon E5-2420 with 6 cores, 12 threads and frequency of 1.9 GHz. Each of them has 32 GB of main memory, 2 TB of hard drive and are connected through Gigabit Ethernet. The softwares used in these machines for evaluation phase are Cloudera's open-source Apache Hadoop distribution CDH version 5.10.0, with Hadoop version 2.6.0 and Impala version 2.5.0. From ten machines of the cluster, one performs as the master node and nine others as worker nodes.

Datasets which are used to evaluate the system are WatDiv (Waterloo SPARQL Diversity Test Suite)[1] and YAGO (Yet Another Great Ontology)[2]. WatDiv is used to evaluate the performance of systems that work with RDF data using SPARQL queries. WatDiv is a dataset generator which was used to create four datasets with different scaling factors (SF), namely SF10 which has 1 million triples, SF100 with 10 million triples, SF1000 with 100 million triples and SF10000 which is the biggest with 1 billion triples. WatDiv also has four sets of queries which are used to query over these datasets. These queries and their type will be discussed later in query execution part. On the other hand, YAGO is a dataset which contains real data retrieved from WordNet, Wikipedia and GeoNames.

The same datasets used for evaluation of Sempala ExtVP Multi Table are also used to evaluate S2RDF ExtVP Multi Table and Sempala ExtVP Single Table. The evaluation is separated in two parts, the loader phase evaluation where creation of

[1] http://dsg.uwaterloo.ca/watdiv/

[2] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

ExtVP tables for WatDiv and Yago is evaluated and query execution evaluation where translation and execution of queries are evaluated.

3.1 Loader Phase

In this section as mentioned earlier we will present the results of creating ExtVP tables for WatDiv and Yago dataset. In Table 3.1 we presented the number of ExtVP tables created by Sempala ExtVP Multi Table and S2RDF ExtVP Multi Table, for all four scaling factor datasets of WatDiv. As we can see here the number of ExtVP tables created by Sempala is higher than those created by S2RDF due to the calculation of type *OO* in Sempala, which was omitted by S2RDF. In this table we also see the overall number of tuples stored in database for each dataset compared to the number of tuples in the triple table. We can see that the number of tuples stored with ExtVP is approximately 18 times higher than the initial one from triple table. Also here we can see that the highest number of ExtVP tables have a scale of zero meaning that they are empty. Using the optimization we did, we calculated for each pair of empty tables only one of them, this way we achieved to skip a high number of calculations which would have ended in empty tables.

Table 3.1: WatDiv sizes of ExtVP tables and tuples for Sempala ExtVP Multi Table ($0 < SF < 1$) and competitors

Size (#tables)					
		SF10	**SF100**	**SF1000**	**SF10000**
S2RDF ExtVP Multi Table	**0 <Scale <1**	2,006	2,038	2,041	2,043
	Scale = 0	19,798	19,780	19,780	19,780
	Scale = 1	298	284	281	279
	Total ExtVP	2,092	2,124	2,127	2,129
Sempala ExtP Multi Table	**0 <Scale <1**	2,324	2,597	2,764	2,762
	Scale = 0	26,738	26,456	26,246	26,210
	Scale = 1	350	359	402	440
	Total ExtVP	2,674	2,956	3,166	3,202
Size (#tuples)					
		SF10	**SF100**	**SF1000**	**SF10000**
Sempala ExtVP Multi Table		18,183,108	184,394,601	1,867,345,461	18,741,520,645
S2RDF ExtVP Multi Table		17,996,082	179,693,549	1,801,839,621	11,997,222,766
Triple Table		1,099,883	10,916,457	109,945,228	1,091,584,535

Table 3.2: WatDiv loading times (in s) for Sempala ExtVP Multi Table and competitors

	SF10	SF100	SF1000	SF10000
S2RDF ExtVP Multi Table	10,931	18,768	45,928	184,052
S2RDF ExtVP Multi Table Opt.	1,430	2,418	9,497	60,572
Sempala ExtVP Single Table	564	855	5,179	67,080
Sempala ExtVP Multi Table	4,443	4,004	4,364	9,948

As we can see from Table 3.2 and graphically represented in Figure 3.1, for smaller datasets the optimized S2RDF version and Sempala ExtVP Single Table perform better than Sempala ExtVP Multi Table and initial version of S2RDF. But for the biggest dataset of WatDiv, Sempala ExtVP Multi Table performs from 5 to 18 times faster than the competitors.

We believe that the time needed for Sempala ExtVP Multi Table to complete the loader phase can be further reduced, if we have more information about the data which we are working with. As we mentioned earlier the distribution of workload over the nodes was done based on the total number of calculations for the dataset. This led to a difference in time required by nodes to calculate Ext tables for their partition of predicates. For example in SF10000, the node which finished first needed only 3,220 seconds to complete its set of calculations while the node which finished last needed 9,948 seconds. We can see that the distribution of the workload is not optimal. In order to have a better distribution of the workload, information about the correlation of predicates is required to be known. This way we can make sure that predicates are distributed over the nodes such that the number of empty tables which are skipped and ExtVP tables which take more time to be calculated is approximately the same in all nodes.

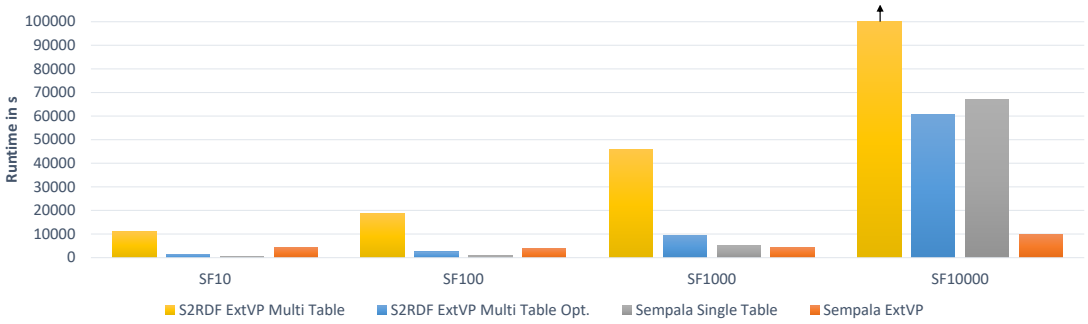

Figure 3.1: Loading times of WatDiv for Sempala ExtVP Multi Table and competitors

Table 3.3: YAGO sizes and loading times for Sempala ExtVP Multi Table ($0 < SF < 1$) and S2RDF ExtVP Multi Table

Size (# tables)		
S2RDF ExtVP Multi Table	**0 <Scale <1**	6,588
	Scale = 0	25,198
	Scale = 1	662
	Total ExtVP	7,250
Sempala ExtVP Multi Table	**0 <Scale <1**	7,273
	Scale = 0	35,104
	Scale = 1	679
	Total ExtVP	7,952
Size (# tuples)		
S2RDF ExtVP Multi Table		2,374,426,690
Sempala ExtVP Multi Table		2,816,753,986
Triple Table		244,800,042
Time (seconds)		
S2RDF ExtVP Multi Table		17,406 s
Sempala ExtVP Multi Table		11,073 s

Also in the case of YAGO dataset the times and sizes of ExtVP tables can be seen in Table 3.3. Again the numbers for size from Sempala are higher compared to S2RDF due to calculation of type *OO*. Compared to initial triple table, the number of tuples stored in ExtVP cases is approximately ten times higher. Regarding the loading time Sempala performs faster then S2RDF, but again the distribution of the workload in Sempala is not optimal due to differences between nodes. Here the fastest node completes the calculations in 6,453 seconds while the node which needs the most time finishes in 11,073 seconds. In Figure 3.2 the loading times for YAGO are presented graphically.

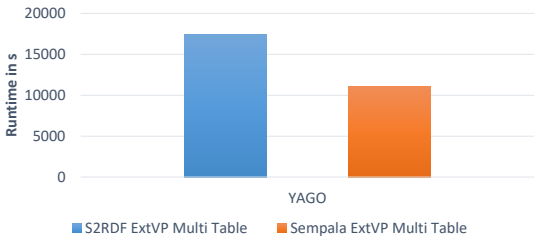

Figure 3.2: Loading times of YAGO for Sempala ExtVP Multi Table and competitors

3.2 Query Execution

In this section we will present the results achieved by Sempala ExtVP Multi Table and competitors in query execution. The evaluation is done for both datasets WatDiv and YAGO. WatDiv is evaluated for two sets of queries, namely Basic and Incremental Linear. On the other hand, YAGO is evaluated with only one set of queries.

3.2.1 WatDiv Basic

WatDiv Basic consists of four type of queries. These queries are linear queries (L), star queries (S), snowflake shaped queries (F), and complex queries (C). Figure 3.3 shows the differences in execution time between Sempala ExtVP Multi Table and competitors for largest dataset of WatDiv (SF10000) corresponding to the AM runtime which is presented in Table 3.4. We can see from the chart that ExtVP Multi Table in Spark and Impala perform faster than Single Table.

Table 3.4: WatDiv Basic (in ms), AM-X = AM of query type X, AM-T = AM of all queries

	Query	L1	L2	L3	L4	L5	AM-L	S1	S2	S3	S4	S5	S6	S7	AM-S
SF10	Sempala ExtVP Single Table	591	590	527	534	590	567	1,000	634	618	629	617	582	572	665
	S2RDF ExtVP Multi Table	164	145	97	95	140	128	478	204	180	190	211	138	141	220
	Sempala ExtVP Multi Table	514	839	500	513	866	646	591	525	863	514	509	526	483	573
SF100	Sempala ExtVP Single Table	640	621	570	575	640	609	1,019	730	661	663	668	617	606	709
	S2RDF ExtVP Multi Table	168	193	126	107	173	153	562	216	214	221	193	146	164	245
	Sempala ExtVP Multi Table	523	864	511	514	848	652	579	551	892	543	515	536	487	586
SF1000	Sempala ExtVP Single Table	947	706	866	697	670	777	1,792	910	919	831	858	886	908	1,015
	S2RDF ExtVP Multi Table	202	196	196	132	162	178	735	294	219	209	199	209	191	294
	Sempala ExtVP Multi Table	561	840	532	519	864	663	726	594	882	540	510	543	546	620
SF10000	Sempala ExtVP Single Table	4,775	2,372	4,346	2,340	1,110	2,989	7,977	4,218	2,945	2,522	2,968	3,735	4,581	4,135
	S2RDF ExtVP Multi Table	471	498	549	209	270	399	2,208	607	311	329	260	235	420	624
	Sempala ExtVP Multi Table	807	902	814	583	943	809	2,025	1,037	980	555	547	536	918	942

	Query	F1	F2	F3	F4	F5	AM-F	C1	C2	C3	AM-C	AM-T
SF10	Sempala ExtVP Single Table	742	861	767	941	767	816	912	1,009	828	916	716
	S2RDF ExtVP Multi Table	370	451	376	461	334	398	535	472	450	486	282
	Sempala ExtVP Multi Table	889	833	874	1,197	1,482	1,055	2,080	1,220	1,837	1,712	883
SF100	Sempala ExtVP Single Table	784	918	804	990	814	862	1,007	1,002	907	972	762
	S2RDF ExtVP Multi Table	393	539	385	579	398	459	577	689	688	651	337
	Sempala ExtVP Multi Table	570	884	894	1,220	1,504	1,014	2,114	1,185	2,168	1,822	895
SF1000	Sempala ExtVP Single Table	1,101	1,220	1,414	1,575	1,436	1,349	1,689	2,435	3,900	2,675	1,288
	S2RDF ExtVP Multi Table	433	642	638	692	672	615	923	1,460	2,929	1,771	567
	Sempala ExtVP Multi Table	581	950	993	981	1,804	1,061	2,016	1,347	4,024	2,462	1,022
SF10000	Sempala ExtVP Single Table	3,569	4,043	8,306	10,184	7,331	6,887	4,247	16,700	7,899	9,639	5,362
	S2RDF ExtVP Multi Table	590	1,226	1,969	1,265	2,254	1,461	2,508	2,740	16,407	7,218	1,766
	Sempala ExtVP Multi Table	918	1,424	2,192	1,551	4,007	2,018	2,500	2,327	10,499	5,108	1,803

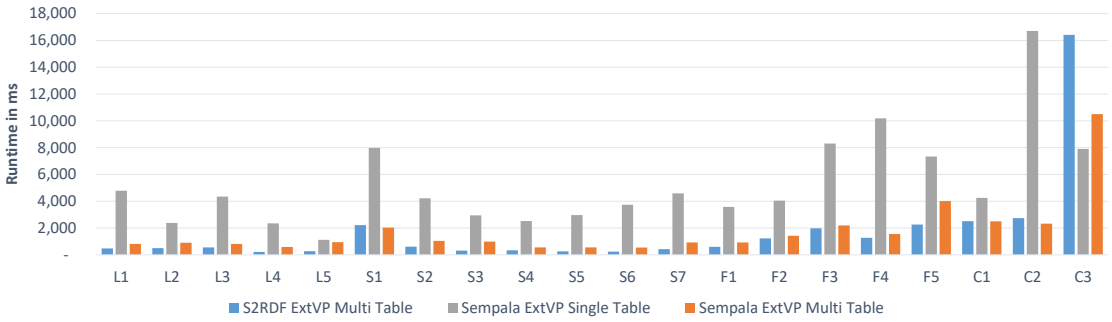

Figure 3.3: WatDiv Basic Testing (SF10000) of Sempala ExtVP Multi Table and competitors

Comparing ExtVP Multi Table in Spark with Impala we see that execution on Spark is in most of the query types faster compared to execution on Impala, except the complex queries where Impala is faster then Spark.

3.2.2 WatDiv Incremental Linear

The other set of queries of WatDiv dataset is Incremental Linear (IL). These queries are divided into three types, namely IL-1, IL-2 and IL-3. Each of the query types consist of queries with 5 triple patterns and increasing linearly up to queries with 10 triple patterns. The difference between three types of IL queries stands in the way the queries are bounded, namely by user, retailer or unbounded.

Table 3.5: WatDiv IL (in ms), AM-IL-X = AM of query type X, AM-Y = AM of length Y queries

	Query	IL-1-5	IL-1-6	IL-1-7	IL-1-8	IL-1-9	IL-1-10	AM-IL-1	IL-2-5	IL-2-6	IL-2-7	IL-2-8	IL-2-9	IL-2-10	AM-IL-2
SF10	Sempala ExtVP Single Table	888	1,074	1,165	994	1,037	1,088	1,041	782	858	900	966	1,027	1,078	935
	S2RDF ExtVP Multi Table	307	360	390	493	503	701	459	453	334	388	466	495	660	466
	Sempala ExtVP Multi Table	1,338	1,656	1,947	2,243	2,549	2,856	2,098	1,411	1,649	1,949	2,254	2,551	2,856	2,111
SF100	Sempala ExtVP Single Table	2,953	4,067	4,709	2,417	2,199	2,312	3,110	1,493	1,723	1,698	1,798	1,845	1,901	1,743
	S2RDF ExtVP Multi Table	558	604	711	834	875	1,299	814	969	594	654	795	911	1,047	828
	Sempala ExtVP Multi Table	1,731	1,938	2,279	2,605	2,936	3,278	2,461	2,786	2,048	2,127	2,421	2,741	3,057	2,530
SF1000	Sempala ExtVP Single Table	14,118	18,865	22,838	11,773	10,596	10,654	14,807	10,913	8,888	8,755	8,499	8,974	8,849	9,146
	S2RDF ExtVP Multi Table	1,724	1,745	1,965	2,029	2,185	2,643	2,048	4,944	1,869	1,980	2,114	2,382	2,413	2,617
	Sempala ExtVP Multi Table	3,777	3,481	3,896	2,567	2,144	2,158	3,003	7,032	6,577	2,980	3,316	3,633	3,970	4,584
SF10000	Sempala ExtVP Single Table	235,865	148,567	178,891	148,158	144,950	176,364	172,133	59,607	125,678	84,605	75,862	113,046	111,570	95,061
	S2RDF ExtVP Multi Table	12,543	12,252	15,062	15,003	15,478	16,124	14,410	41,188	13,276	14,182	15,261	16,313	13,922	19,024
	Sempala ExtVP Multi Table	9,712	11,374	12,549	12,044	12,254	12,339	11,712	13,375	36,831	9,185	5,739	10,194	6,078	13,567
	Query	IL-3-5	IL-3-6	IL-3-7	IL-3-8	IL-3-9	IL-3-10	AM-IL-3	AM-5	AM-6	AM-7	AM-8	AM-9	AM-10	
SF10	Sempala ExtVP Single Table	9,465	12,070	2,696	67,921	4,431	4,861	16,907	3,712	4,667	1,587	23,294	2,165	2,342	
	S2RDF ExtVP Multi Table	382	783	732	10,698	1,118	1,241	2,492	381	492	503	3,885	705	868	
	Sempala ExtVP Multi Table	1,359	1,774	2,015	2,560	3,454	3,874	2,506	1,369	1,693	1,970	2,352	2,851	3,195	
SF100	Sempala ExtVP Single Table	96,188	125,219	22,903	714,894	37,645	41,267	173,019	33,545	43,669	9,770	239,703	13,896	15,160	
	S2RDF ExtVP Multi Table	855	2,766	1,978	36,443	3,608	3,244	8,149	794	1,322	1,114	12,691	1,798	1,863	
	Sempala ExtVP Multi Table	2,076	3,876	3,238	5,392	14,956	16,104	7,607	2,198	2,621	2,548	3,473	6,878	7,480	
SF1000	Sempala ExtVP Single Table	479,772	620,307	107,386	3,583,029	213,286	230,228	872,335	168,268	216,020	46,326	1,201,100	77,619	83,243	
	S2RDF ExtVP Multi Table	4,474	12,188	8,552	178,514	13,411	13,405	38,424	3,714	5,267	4,166	60,886	5,993	6,154	
	Sempala ExtVP Multi Table	22,324	25,133	9,020	82,028	14,362	15,395	28,043	11,044	11,730	5,299	29,304	6,713	7,174	
SF10000	Sempala ExtVP Single Table	2,167,746	2,854,454	761,754	2,900,000	2,900,000	2,900,000	9,093,967	821,073	1,042,900	341,750	14,275,639	1,085,322	1,155,639	
	S2RDF ExtVP Multi Table	29,590	87,525	102,971	2,068,100	158,595	141,940	431,454	27,774	37,684	44,072	699,454	63,462	57,329	
	Sempala ExtVP Multi Table	98,241	97,738	71,111	362,596	97,358	114,627	140,279	40,443	48,648	30,948	126,793	39,935	44,348	

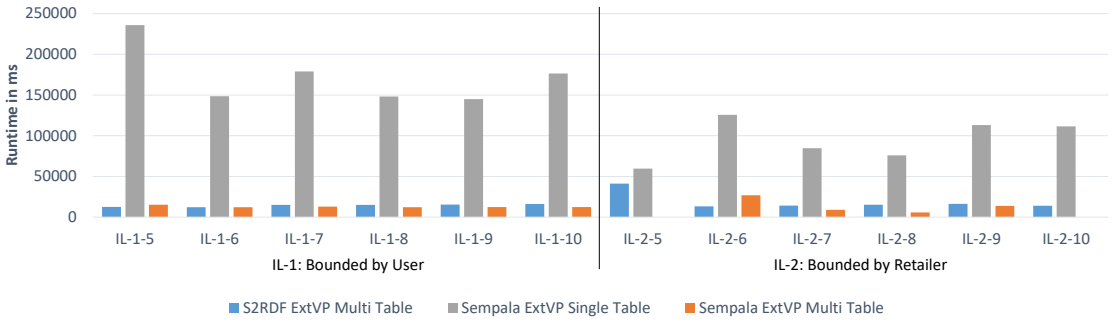

Figure 3.4: WatDiv IL-1 and IL-2 runtime in ms (SF10000) of Sempala ExtVP Multi Table and competitors

In Table 3.5 the execution of IL queries for all WatDiv datasets based on scaling factor are presented. Each query is executed ten times for ten different users or retailers in cases when queries are bounded. The AM of ten executions is written in the table.

We can see from the table that ExtVP Multi Table model outperforms the Single Table model. Comparing the ExtVP model on Spark and Impala, we see that execution on Spark is faster than execution on Impala for smaller datasets, but in the case of largest dataset the execution on Impala performs better than that on Spark. An exception for all datasets is query IL-3-8 which is query with the highest number of tuples returned as result. The reason why this query performs better in Impala is that for type IL-3 of queries, which are unbounded, Sempala lets the Impala query planer to decide the join ordering using statistics of ExtVP tables. Based on the type of queries, also IL-1 which are bounded by the user and IL-2 which are bounded by the retailer, performed better using the straight join compared to Impala join ordering. The execution of IL-1 and IL-2 without straight join and execution of IL-3 with straight join performed many times slower.

Another reason why Sempala performs better then S2RDF for the biggest dataset is the reason that here the results are not stored in a table. If we compare the case of Sempala storing the result tuples in table and the case of Sempala counting the result, we see that storing the results needs more time than counting. This difference is mostly noticed in type IL-3 were the results contain up to 25 million tuples. The reason behind this we believe stands in the use of hive metastore by Impala. Since Impala uses the hive metastore to store its table definitions and store the data, this makes the process of writing the result in table much more slower than counting

the result tuples. In contrast from Impala, Spark can store the result as a parquet file directly to Hdfs. This feature allows S2RDF to perform better than Sempala in storing the result tuples in table, but in the case of counting Sempala performs much faster then S2RDF. The runtime for Incremental Linear queries for largest dataset (SF10000) is presented graphical in Figure 3.4 for IL-1 and IL-2, and in Figure 3.5 for IL-3.

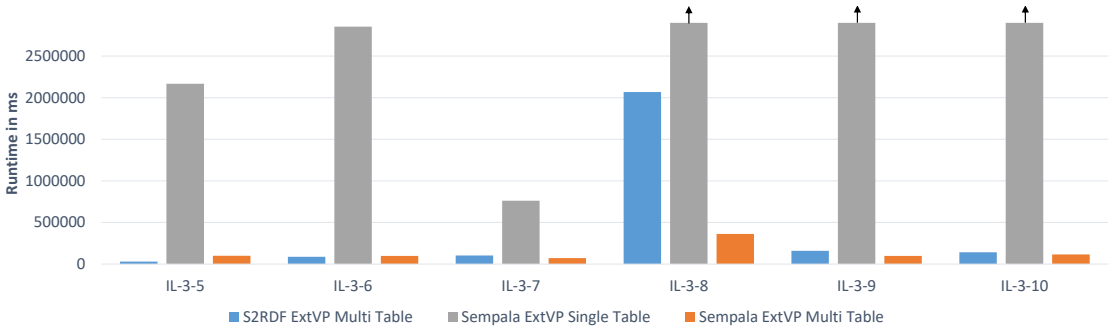

Figure 3.5: WatDiv IL-3 runtime in ms (SF10000) of Sempala ExtVP Multi Table and competitors

3.2.3 YAGO

For YAGO dataset there exist only one set of queries for evaluation. This set consists of 15 queries. Also for YAGO as for other datasets queries are run multiple times and the AM of execution time is presented in Table 3.6. We can see from the table and graphical representation in Figure 3.6, that Sempala performs better in some queries and S2RDF in some other. What can be observed easily is the difference in performance for query Q5 and Q9 in Sempala. For these two queries the difference between Sempala and S2RDF is very high. For query Q5 which is a SPARQL query with Filter, the difference may come from the fact that Sempala does not perform the filtering in the first join to reduce the number of output. While for table Q9 a problem in the creation of an ExtVP table which is used by this query is believed to derive this delay in execution. This problem with some ExtVP tables in YAGO is observed also in cases where these queries are executed with different threshold.

Table 3.6: YAGO (in ms)

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
S2RDF ExtVP Multi Table	1,612	1,808	905	798	2,410	380	871	895	25,504	919	2,919	2,382	2,145	2,895	1,326
Sempala ExtVP Multi Table	834	1,031	1,245	563	21,102	402	463	709	90,134	639	1,363	4,618	4,233	1,472	524

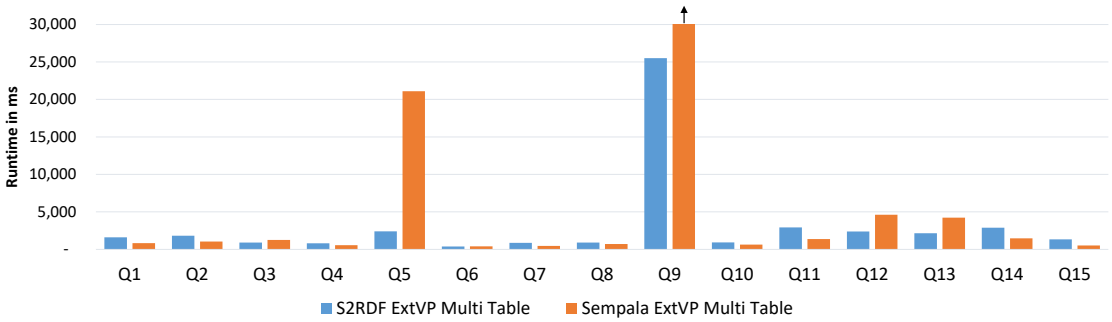

Figure 3.6: YAGO runtime in ms of Sempala ExtVP Multi Table and competitors

In some queries, specifically query Q5, the use of ExtVP tables did not give reduce the time of query execution, on contrary performed slower then using the complete corresponding partition of the Triple Table. This problem occurred in use of a specific table which led to believe that this slow performance was linked to this table.

4 Conclusion

The aim of this project was to optimize and implement Extended Vertical Partition layout in existing system of Sempala. Due to optimization done in the algorithm for creation of ExtVP tables in loader phase and distribution of workload over the cluster, we were able to reduce the creation time of ExtVP tables for approximately 20 times compared to S2RDF. In the evaluation of query execution we saw that Sempala ExtVP Multi Table is comparable with S2RDF, with both systems performing better then the other for specific queries and outperfoming Sempala ExtVP Single Table, but in cases when the results are stored, Sempala is outperformed by S2RDF.

We saw earlier that the amount of tuples stored in ExtVP tables is 18 times higher than the number of tuples stored in initial Triple Table. But we also observed that using ExtVP layout reduced the execution time of queries. ExtVP layout is a trade-off between performance and size of data stored. This can be seen also in the graphs presented in Figure 4.1, where we see that with the increase of SF Threshold from 0 where no ExtVP tables are stored, to 1 where all ExtVP tables are stored, the amount of data is increased but also the execution time is reduced. The impact of threshold changes for the type of queries and datasets, but in overall after threshold value of 0.5 there isn't any big difference in execution time.

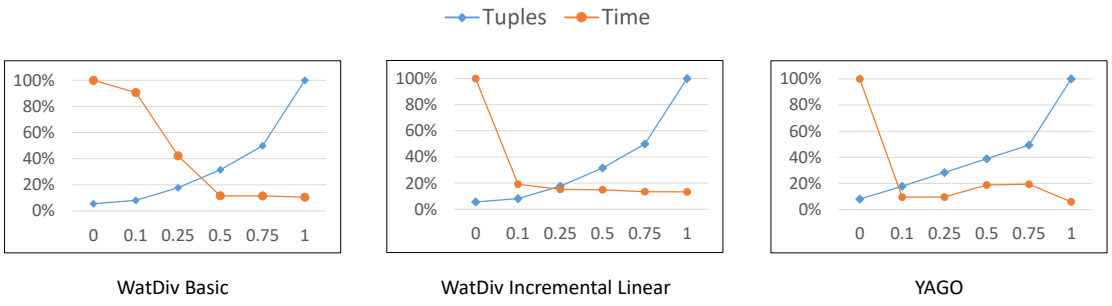

Figure 4.1: Relative query runtimes and data sizes in relation to increasing SF threshold values

Bibliography

[1] P. A. Bernstein and D.-M. W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981.

[2] Manuel Schneider. Master projekt bericht: Erweiterung des projektes sempala um das datenformat single table.

[3] A. Schätzle, M. Przyjaciel-Zablocki, A. Neu, and G. Lausen. Sempala: Interactive sparql query processing on hadoop. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, and C. Goble, editors, *The Semantic Web – ISWC 2014*, volume 8796 of *Lecture Notes in Computer Science*, pages 164–179. Springer International Publishing, Cham, 2014.

[4] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen. S2rdf. *Proceedings of the VLDB Endowment*, 9(10):804–815, 2016.