# SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions

Olaf Görlitz and Steffen Staab

Institute for Web Science and Technology
University of Koblenz-Landau, Germany
{goerlitz,staab}@uni-koblenz.de
http://west.uni-koblenz.de/

**Abstract.** In order to leverage the full potential of the Semantic Web it is necessary to transparently query distributed RDF data sources in the same way as it has been possible with federated databases for ages. However, there are significant differences between the Web of (linked) Data and the traditional database approaches. Hence, it is not straightforward to adapt successful database techniques for RDF federation. Reasons are the missing cooperation between SPARQL endpoints and the need for detailed data statistics for estimating the costs of query execution plans. We have implemented SPLENDID, a query optimization strategy for federating SPARQL endpoints based on statistical data obtained from voiD descriptions [1].

## 1 Introduction

A large amount of structured data is freely available on the web and can be accessed by machines in various ways. RDF [11] and the SPARQL [8] query language are essential parts of the evolving Web of Data. Moreover, the Linked Data principles[1] give guidelines on how to interconnect different datasets. However, answering complex queries across different RDF data sources, like in federated databases, is not trivial to implement. Two different paradigms are typically applied. The *data warehousing* approach loads all data sets into one large repository and executes queries efficiently employing optimized index structures. But changes in the original data are not easily accounted for. *Distributed Query Processing*, on the other hand, executes queries on the actual distributed data sources and aggregates the returned results. However, query planning requires a-priori knowledge about the data sources to judge whether a data source can return results for a query and to apply sophisticated query optimization techniques, as known from traditional databases.

Statistical information can be extracted from RDF data dumps. But dumps may not always be available, e.g. due to legal issues, or frequent statistics recalculation may become expensive for changing data. Instead, we exploit the *Vocabulary of Interlinked Datasets* [1] (VOID) which already incorporates statistical information. However, there is a trade-off between the compactness of a data source representation and the level of precision for communicating statistical details that may be expensive to built, store

[1] http://esw.w3.org/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

and maintain. VOID is positioned at the sparse end of a data description vocabulary. It was not initially designed for being used for query optimization. But we believe that this is a reasonable choice. However, it requires to rethink existing query federation mechanisms in order to provide optimal source selection and query routing efficiency.

We present SPLENDID, a query federation strategy for SPARQL endpoints. In contrast to other existing federation approaches that assume an arbitrary level of detail for statistics-based source selection, query optimization and query execution, SPLENDID solely relies on VOID statistics. Thus, we can integrate virtually any RDF data source found in the Semantic Web.

## 2 Scenario

Researchers in the life science domain have numerous databases at hand which contain detailed information about pathways, genes, proteins, drugs and so forth. Data integration is an active research area [5] and some of these datasets are also available as RDF [2] with links to other datasets. Imagine a researcher is looking for pain relieving drugs similar to Acetaminophen. The search request can be formulated as "drugs based on Enzyme Cytochrome P450 3A4". In order to obtain answers for this request data sources like Kegg, ChEBI, Drugbank, and UniProt have to be examined. But the researcher does not want to do this by hand. Instead, he uses a search tool which translates the request into a SPARQL query and processes it transparently across all known life science datasets. The results are aggregated and presented to the researcher as if there were retrieved from one large database. This is possible as all the datasets offer access to their data via SPARQL endpoints and return results as RDF. Information about the data in each dataset is provided as VOID descriptions [1] (c.f. Fig. 1) such that the search tool can determine which data sources to contact for specific information. Since RDF, SPARQL and VOID are core technologies of the Semantic Web, the data federation is applicable for other domains as well.

```
:ChEBI a void:Dataset ;

# general information
dcterms:title "ChEBI" ;
dcterms:description "Chemical␣Entities" ;
foaf:homepage
    <http://chebi.bio2rdf.org/>
void:sparqlEndpoint
    <http://chebi.bio2rdf.org/sparql> ;

# simple data statistics:
void:triples        "7325744" ;
void:entities         "50477" ;
void:properties          "28" ;
void:distinctSubjects "50477" ;
void:distinctObjects "772138" ;
```

```
# entity count per concept
void:classPartition [
    void:class chebi:Compound ;
    void:entities "50477" .
] ;

# triple count per predicate
void:propertyPartition [
    void:property bio:formula ;
    void:triples "39555" ;
] , [
    void:property bio:image ;
    void:triples "34055" ;
] , [
    ...
] .
```

**Fig. 1.** VOID description excerpt for the ChEBI dataset containing general information and statistical data, like total triple count and number of occurrences of predicates and instances. (namespace are omitted for better readability, see http://void.rkbexplorer.com/ for more examples)

## 3 Related Work

One of the first RDF federation appraoches was presented by Stuckenschmidt et al.[19]. A path index is created for the graph structure of the data sets and then used for the query optimization to match longest paths in the query. More complex structures than paths are not supported. Harth et al. [9] use a *QTree* for indexing the content of many data sources. The QTree is initialized with triples from seed data sources which are hashed by subject, predicate, and object into buckets along the three dimensions of the QTree. Source selection and ranking is done by identifying all QTree buckets matching a query's triple and join patterns. Ladwig and Tran [10] apply full indexing for all triples and the join combination of triples similar to [12, 21]. This allows for accurate source selection and result size estimation. However, all of the aforementioned approaches require raw triples to be indexed. Hence, they are not applicable for scenarios where access to statistical data is restricted to VOID descriptions.

DARQ [13] employs hand crafted data source descriptions similar to VOID. Besides basic triple and entity counts it also allows for defining average selectivity estimates for combinations of subject, predicate, and object and includes restrictions (so called capabilities) on satisfiable subject, object values. FedX [17] focuses on efficient query execution techniques using chunked semi-joins. It does not use any precomputed statistics for query optimization but solely relies on join order heuristics. Source selection is based on SPARQL ASK queries and the maintainance of a ASK history. Recent work by Buil-Arada et. al [4] investigates the complexity and optimization of SPARQL 1.1 federation[2] queries where data sources are already assigned to query expressions. Networked Graphs [14] integrate distributed RDF data sources via a declarative SPARQL based view mechanism.

## 4 SPLENDID: SPARQL Endpoint Federator

The main components of SPLENDID are the *Index Manager*, the *Query Optimizer*, and the *Query Executor* (c.f. Fig. 2). The query parser transforms the textual representation of a SPARQL query into an abstract syntax tree, which can be handled by the query optimizer. The result of the query optimization is a query execution plan which is processed by the query executor in order to retrieve and join the result tuples. The query parser, the query executor, and the result serializer in SPLENDID are based on standard methods and will not be discussed in detail.

### 4.1 Index Manager

The statistics of VOID descriptions are aggregated in a local index by the Index Manager. General information, like triple count, the number of distinct predicates, subjects, and objects are stored as attributes for every SPARQL endpoint. The statistical information for every predicate and type are organized in inverted indexes $I_p : \{(p, \{(d_i, c_i)\})\}$ and $I_\tau : \{(\tau, \{(d_i, c_i)\})\}$ which map predicates and types to a set of tuples containing the data source $d$ and the number of occurrences in the data source.

[2] http://www.w3.org/TR/sparql11-federated-query/

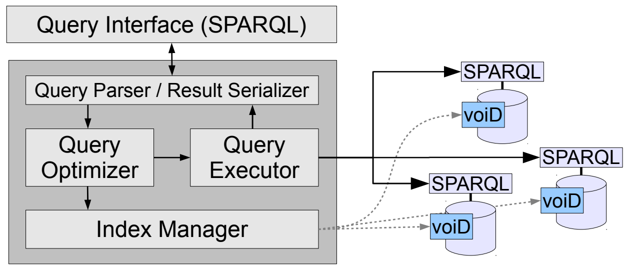

**Fig. 2.** Architecture of the SPLENDID Federator.

## 4.2 Query Optimizer

The SPLENDID query optimizer transforms a given query into a semantically equivalent query that exhibits low costs in terms of processing time and communication overhead. Three optimization steps are applied 1) query rewriting, 2) data source selection, and 3) cost-based join order optimization. Query rewriting is an optimization of the logical tree structure of a query, e.g. heuristics are used to split up complex filter expressions and relocate them close to operators which produce bindings for the filtered variables. In the following the second and third step will be discussed in more detail.

**Data Source Selection** Each triple pattern in a query may potentially be answered by different data sources. Hence, we need to identify all SPARQL endpoints which can return results for them. First, each triple pattern with a bound predicate is mapped to a set of data sources using the index $I_p$. Triple patterns which have rdf:type as predicate and a bound object variable are mapped by using the index $I_\tau$. For triple patterns with unbound predicates we assign all data sources as there is no further information available from the VOID descriptions.

*Refining selected data sources* The precision of the source selection is important. Requesting data from wrongly identified data sources, which can not return any results, is expensive in terms of network communication and query processing cost. For example, the predicate rdfs:label may occur in almost all data sources, whereas the triple pattern (?x rdfs:label ”ID_1652”) may only be matched by one data source. Hence, for triple patterns with bound variables which are not covered in the VOID statistics we send a SPARQL ASK query including the triple pattern to all pre-selected data sources and remove sources which fail the test. This pruning of data sources before the actual join order optimization is more efficient than accepting no results for regular SPARQL SELECT queries. Algorithm 1 shows in detail how the source selection is done.

*Building Sub Queries.* Triple patterns must be sent to all selected data sources independently, even if a group of triple patterns shares exactly the same set of sources. This ensures that results for individual triple patterns can be joined across data sources. However, if a source is exclusively selected for a set of triple patterns all of them can be combined into a single sub query. This is termed *exclusive groups* in FedX [17]. Another option for pattern grouping exists for triple patterns with the predicate owl:sameAs and

**Algorithm 1** Source Selection for triple patterns using VOID and `ASK` queries.

```
Require: $I_p$; $I_\tau$; $D = \{d_1, \ldots, d_m\}$; $T = \{t_1, \ldots, t_n\}$ // indexes, data sources, and triple patterns
 1: for each $t_i \in T$ do
 2:     $sources = \emptyset$
 3:     $s = subj(t_i)$; $p = pred(t_i)$; $o = obj(t_i)$
 4:     if ! $bound(p)$ then
 5:         $sources = D$ // assign all sources for unbound predicate
 6:     else
 7:         if $p =$ rdf:type $\wedge$ $bound(o)$ then
 8:             $sources = I_\tau(o)$
 9:         else
10:             $sources = I_p(p)$
11:         end if
12:     end if
13:     // prune selected sources with ASK queries
14:     if ! $bound(p) \vee bound(s) \vee bound(o)$ then
15:         for each $d_i \in sources$ do
16:             if $ASK(d_i, t_i) \neq true$ then
17:                 $sources = sources / \{d_i\}$
18:             end if
19:         end for
20:     end if
21: end for
```

unbound subject variable. Under the assumption that all data sources define owl:sameAs links for their own data, we can combine triple patterns which contain the same unbound variable as defined as the subject of the owl:sameAs pattern, e.g. variable $?y$ in { ?x foaf:knows ?y . ?y owl:sameAs ?z }. The sameAs optimization can only be enabled manually, if no 3rd party dataset with external owl:sameAs links is included in the federation.

**Join Order Optimization** SPLENDID employs Dynamic Programming [18], a flexible optimization strategy often used in traditional relational databases, to optimize the join order of SPARQL basic graph patterns. Using the sub queries generated by the source selection step, all possible physical query execution plans are iterated and inferior plans are pruned based on an overall cost estimate for executing all operators of a query plan. Query execution plans can have different tree structures. We prefer bushy trees since it has been shown in [20] that they are a good choice for SPARQL queries.

**Join Implementation** We consider two different join execution strategies, 1) requesting result tuples for the join arguments in parallel from the SPARQL endpoints to join them locally, and 2) using the results of the first join argument to substitute unbound variables in the second join argument with a repeated evaluation for every binding. The first approach is well suited for retrieving two small result sets which can be joined locally, while the second one can significantly reduce the network overhead if one result set is large and the selectivity of the join variable is high. We have implemented the first strategy with hash joins (SPLENDID$_\mathrm{H}$) and the second one with bind joins [7] (SPLENDID$_\mathrm{B}$).

**Cost Function** In order to compare two equivalent query execution plans with different join order and different physical join operators we need to calculate the total execution cost. Since the network communication has the highest impact on the overall cost, our cost model currently only includes the cost for sending queries to a SPARQL endpoint and the cost for receiving the results. For simplification, we assume that the size of all queries is the same, i.e. they require the same number of packages to be transmitted over the network, and all result tuples are considered to be of the same average size as well. Formally, we define the transfer cost for hash join and bind join as follows

$$tc(q_1 \bowtie_H q_2) = card(q_1) \cdot c_{rt} + card(q_2) \cdot c_{rt} + 2 \cdot c_{sq} \tag{1}$$
$$tc(q_1 \bowtie_B q_2) = card(q_1) \cdot c_{rt} + card(q_1) \cdot c_{sq} + card(q_2') \cdot c_{rt} \tag{2}$$

The cost for sending a SPARQL query is $c_{sq}$ and the cost for receiving a single result tuple is $c_{rt}$. The number of result tuples which will be returned for a query $q$ is defined by $card(q)$. For the bind join the result size of the second join argument is reduced by the bindings of the first join argument which is expressed by $card(q_2')$.

**Cardinality Estimation** The reliability of a query's estimated processing cost mainly depends on the accuracy of the result cardinality estimation. Although detailed statistics yield better estimates, we can also observe that estimation errors are growing with the number of joins. In order to estimate the cardinality of a basic graph pattern we first need to estimate the cardinality of all individual triple patterns and then calculate the join cardinality.

*Single Triple Pattern* Due to the restriction of VOID statistics to predicates and types we can only determine the exact cardinality for triple patterns with bound predicate or for triple pattern with rdf:type and bound object. The cardinality estimation of all other variations of bound variables relies on estimates as follows.

$$\begin{aligned}
card_d(?,p,?) &= card_d(p) & card_d(s,?,?) &= |d| \cdot sel.s_d \\
card_d(s,p,?) &= card_d(p) \cdot sel.s_d(p) & card_d(?,?,o) &= |d| \cdot sel.o_d \\
card_d(?,p,o) &= card_d(p) \cdot sel.o_d(p) & card_d(s,?,o) &= |d| \cdot sel.s_d \cdot sel.o_d
\end{aligned}$$

The cardinality $card_d(p))$ of a predicate $p$ in a data source $d$ is the number of triples in $d$ which contain $p$ as the predicate. The number of all triples in data source $d$ is $|d|$. For bound subjects and bound objects we use their average selectivity in combination with a bound predicate $sel.s_d(p)$ and $sel.o_d(p)$ or without a bound predicate $sel.s_d$ and $sel.o_d$. The average selectivity is defined as the fraction of triples with the same subject or object if subjects and objects were uniformly distributed and independent from the predicate.

*Pattern Groups* Star shaped query pattern are common in SPARQL queries and typically match subjects with certain attributes. Adding a triple pattern with the same subject introduces another restriction on the subject but depending on the object variable, whether it is bound or not, the result size can decrease or increase. Consider the two following queries with two triple patterns each.

| | |
|---|---|
| ?*x* :lives_in "Berlin" . | ?*x* :lives_in "Berlin" . |
| ?*x* :has_name "John" | ?*x* :has_friend ?*y* |

In the first case the result size is reduced by the second pattern whereas in the second case the result size increases. To capture this behavior we handle triple patterns with the same subject separately. First, all triple patterns with the same subject are grouped. Then we take the minimum cardinality of all patterns with a bound object. The remaining patterns with an unbound object are simply multiplied with the minimum value and the average selectivity of the subject.

$$card_d(T) = min(card_d(T_{bound})) \cdot \prod (sel.s_d \cdot card_d(T_{unbound}))$$

The combination of triple patterns with different subjects is a join in the common sense and will be computed based on the join cardinality.

*Combine Results of Multiple Sources* The results obtained for a sub query which is evaluated on multiple data sources need to be combined. For simplicity we assume that all data sources return distinct result tuples. Hence, the cardinality across multiple data sources is the Union of the individual cardinalities.

*Join Cardinality* We compute the join cardinality as

$$card(q1 \bowtie q2) = card(q_1) \cdot card(q_2) \cdot sel_{\bowtie}(q1, q2)$$

where the $sel_{\bowtie}$ is the join selectivity of the two input relations. The join selectivity defines how many bindings of one relation match with bindings of the other relation. It is a reduction factor which depends on the selectivity of the join variable in both datasets. We use the average selectivity of the join variable as the join selectivity.

## 5 Evaluation

The goal of the evaluation is to show that SPLENDID is able to achieve good query execution performance for real world federation scenarios. We use FedBench [15] as evaluation infrastructure[3] and analyze query execution times for different queries and settings. We also compare SPLENDID with other federation implementations.

### 5.1 Benchmark Setup

RDF benchmarks like BSBM [3] and SP2B [16] are mainly designed for the evaluation of triple stores which keep their data in a single large repository. Although a benchmark

[3] http://code.google.com/p/fbench/

**Table 1.** FedBench datasets used for the evaluation.

| Data Set | version | #triples | #subjects | #pred. | #objects | #types | #links |
|---|---|---|---|---|---|---|---|
| DBpedia subset[1] | 3.5.1 | 43.6M | 9.50M | 1063 | 13.6M | 248 | 61.5k |
| GeoNames | 2010-10-06 | 108M | 7.48M | 26 | 35.8M | 1 | 118k |
| LinkedMDB | 2010-01-19 | 6.15M | 694k | 222 | 2.05M | 53 | 63.1k |
| Jamendo | 2010-11-25 | 1.05M | 336k | 26 | 441k | 11 | 1.7k |
| New York Times | 2010-01-13 | 335k | 21.7k | 36 | 192k | 2 | 31.7k |
| SW Dog Food | 2010-11-25 | 104k | 12.0k | 118 | 37.5k | 103 | 1.6k |
| KEGG | 2010-11-25 | 1.09M | 34.3k | 21 | 939k | 4 | 30k |
| ChEBI | 2010-11-25 | 7.33M | 50.5k | 28 | 772k | 1 | - |
| Drugbank | 2010-11-25 | 767k | 19.7k | 119 | 276k | 8 | 9.5k |
| DBpedia subset | 3.5.1 | 43.6M | 9.50M | 1063 | 13.6M | 248 | 61.5k |

[1] includes the ontology, infobox types plus mapped properties, titles, article categories with labels, Geo coordinates, images, SKOS categories, and links to New York Times and Linked Geo Data.

dataset may be split up, it is difficult to create partitions which resemble the characteristics of real world datasets. Moreover, as shown in [6] there are significant differences concerning the *structuredness* of artificial data sets and real world data sets. Hence, the authors recommend not to use artificial datasets for such evaluations. To the best of our knowledge the recently published FedBench [15] is the only real federation benchmark which was explicitly designed for this purpose.

The FedBench datasets are carefully chosen, with respects to size, diversity, number of interlinks, etc. The benchmark queries resemble typical requests on these datasets and their structure ranges from simple star and chain queries to complex graph patterns. All queries cover at least two different data sources. Table 1 gives details about the size of the data sets along with some statistical information. The number of sources which contribute results to a query and the number of result tuples is shown in following table.

| | CD1 | CD2 | CD3 | CD4 | CD5 | CD6 | CD7 | LS1 | LS2 | LS3 | LS4 | LS5 | LS6 | LS7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #sources | 2 | 2 | 5 | 5 | 5 | 4 | 5 | 2 | 4 | 2 | 2 | 3 | 3 | 3 |
| #results | 90 | 1 | 2 | 1 | 2 | 11 | 1 | 1159 | 333 | 9054 | 3 | 393 | 28 | 144 |

Due to the unpredictable availability and latency of the original SPARQL endpoints of the benchmark dataset we used local copies of them which were hosted on five 64bit Intel(R) Xeon(TM) CPU 3.60GHz server instances running Sesame 2.4.2 with each instance providing the SPARQL endpoint for one life science and for one cross domain dataset. The evaluation was performed on a separate server instance with 64bit Intel(R) Xeon(TM) CPU 3.60GHz and a 100Mbit network connection.

**Life Science Query 5:**

```
SELECT ?drug ?keggUrl ?chebiImage WHERE {
   ?drug rdf:type drugbank:drugs .
   ?drug drugbank:keggCompoundId ?keggDrug .
   ?keggDrug bio2rdf:url ?keggUrl .
   ?drug drugbank:genericName ?drugBankName .
   ?chebiDrug purl:title ?drugBankName .
   ?chebiDrug chebi:image ?chebiImage .
}
```

**Cross Domain Query 3:**

```
SELECT ?pres ?party ?page WHERE {
   ?pres rdf:type dbpedia-owl:President .
   ?pres dbpedia-owl:nationality dbpedia:United_States .
   ?pres dbpedia-owl:party ?party .
   ?x nytimes:topicPage ?page .
   ?x owl:sameAs ?pres
}
```

**Fig. 3.** Example FedBench Queries

## 5.2 Evaluation of Source Selection

As mentioned before, the accuracy of the source selection has a large influence on the query processing time. Therefore, we investigated how the information from the VOID descriptions effect the accuracy of the source selection. For each query, we look at the number of sources selected and the resulting number of requests to the SPARQL endpoints. We tested three different source selection approaches, based on 1) predicate index only (no type information), 2) predicate and type index, and 3) predicate and type index and grouping of sameAs patterns as described in Section 4.2.

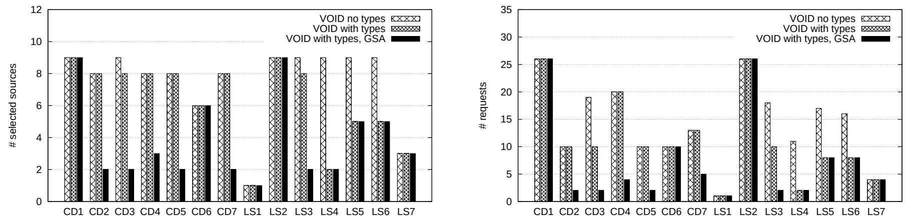

**Fig. 4.** Number of selected data sources (left) and number of SPARQL endpoints requests (right) when using VOID statistics with or without type information, grouping sameAs patterns (GSA).

Figure 4 shows that for the life science queries the number of selected sources is reduced if type information is available. For the cross domain queries it has no effect as they do not contain any triple pattern with rdf:type (except for CD3). In contrast, the grouping of sameAs patterns is most effective for the cross domain queries since the life science queries do not include sameAs triples (except for LS3). Queries CD1 and LS2 contain a triple pattern with no bound variable. Hence, all sources are selected. From the results, we conclude that type information is important and should always be included in VOID descriptions. In addition, the number of requests is reduced significantly, if sameAs links are not provided by 3rd party ”link sets” but included in the original datasets. For the FedBench datasets we can safely apply sameAs grouping without sacrificing the result completeness.

## 5.3 Evaluation of Query Optimization

We measured the query evaluation times for the different optimizer configurations to see how the use of hash join, bind join, and the sameAs grouping affects the overall performance of the query federation. To ensure a consistent evaluation setup, we used the source selection based on SPARQL ASK queries. All queries were evaluated ten times with a two minute timeout. The average query evaluation time is shown in Fig. 5.

First note that there are query execution plans which did not finish within the time limit. Generally, we can observe that the bind join has the shortest query evaluation times for all cross domain queries and for about half of the life science queries. But there are also three query execution plans using hash joins which perform best for the life science queries LS1, LS5, and LS7. One reason for the good results of the bind join are large intermediate result sets, which are produced by the queries and processed most

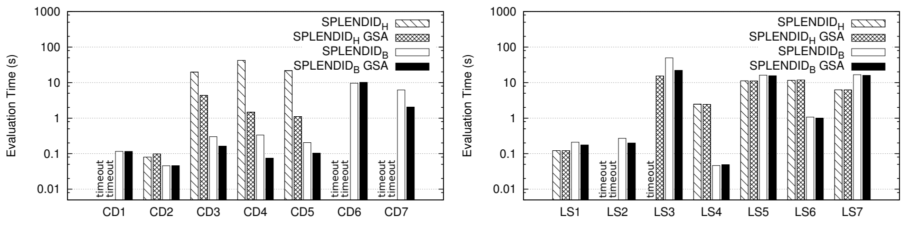

**Fig. 5.** Query evaluation time for cross domain (CD) and life science (LS) queries. The optimization employs either bind join (B) or hash join (H), and groups sameAs patterns (GSA).

efficiently if they are not transmitted completely over the network. Currently, SPLENDID does not include optimizations of the actual query execution. Hence, hash joins could benefit in the future if an efficient parallel retrieval of result tuples would be implemented. Note also the reduced evaluation time from grouping sameAs patterns in the cross domain queries, which is most significant for CD3-CD5.

In general we see that neither bind join nor hash join is superior to the other. The combination of both join implementations in one query execution plan should theoretically yield the best optimization results. As we can see in Fig. 6, this is true for the cross domain dataset, as all query plan achieve the best performance compared to Fig. 5. But we also see that for the life science queries there is still space for improvements.

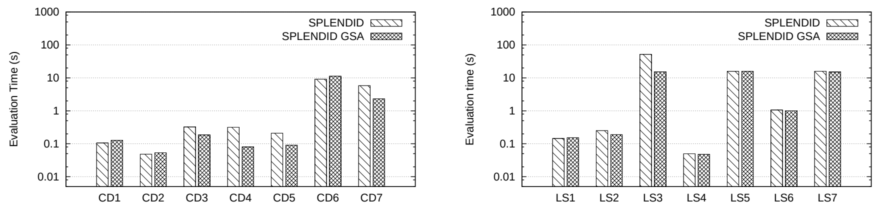

**Fig. 6.** Query evaluation time for cross domain (CD) and life science (LS) queries when combining bind join and hash join in query execution plans, and grouping of sameAs patterns (GSA).

## 5.4 Comparison with other Federation Approaches

SPLENDID was compared to other state-of-the-art SPARQL federation approaches, namely Sesame's AliBaba, DARQ, and the recently published FedX. AliBaba and FedX use heuristics to find the best join order whereas DARQ and SPLENDID use statistical information and optimize query plans based on dynamic programming. Only SPLENDID utilizes hash joins. As Fig. 7 clearly shows, SPLENDID and FedX return results for all queries. AliBaba and DARQ fail to return results for six out of the 14 queries for different reasons. AliBaba generates malformed sub queries for CD3, CD5, LS6, and LS7. DARQ can not handle the unbound predicate in CD1 and LS2. For CD3 and CD5 DARQ opens too many connections to GeoNames. All other unsuccessful queries take longer than the time limit of five minutes. Overall, FedX has the best query evaluation performance. The reason is its novel and efficient query execution based on block transmission of result tuples and parallelization of joins. However, there is only a significant difference between FedX and SPLENDID for CD6, CD7, LS3, LS5-7. For the other queries SPLENDID is close to FedX and for CD3 and CD4 even slightly faster, which indicates that SPLENDID, indeed, generates better query execution plans.

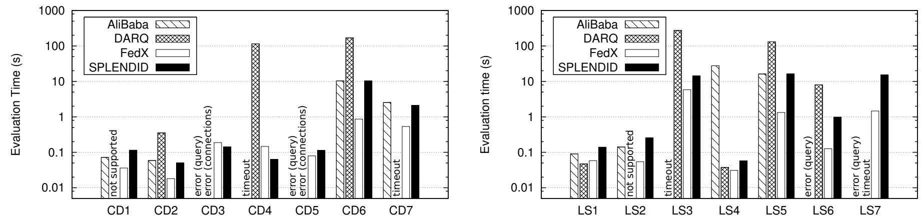

**Fig. 7.** Comparing the query evaluation time for state-of-the-art SPARQL endpoint federation approaches, i.e. Sesame AliBaba, DARQ, FedX, and SPLENDID, using the FedBench cross domain (CD) and life science (LS) queries.

## 6 Conclusion

SPLENDID allows for transparent query federation over distributed SPARQL endpoints. In order to achieve a good query execution performance, data source selection and query optimization is based on basic statistical information which is obtained from VOID descriptions. The utilization of open semantic web standards, like VOID and SPARQL endpoints, allows for flexible integration of various distributed and linked RDF data sources. We have described in detail the implementation of the data source selection and the join order optimization. The evaluation shows that our approach can achieve good query performance and is competitive compared to other state-of-the-art federation implementations.

In our analysis of the source selection we came to the conclusion that at least predicate and type statistics should be included in VOID description for RDF datasets. The use of 3rd party sameAs links, however, can significantly increase the number of requests and thus, hamper the efficiency of query execution plans. The comparison of the two employed physical join implementations has shown that the network overhead plays an important role. Both hash join and bind join can significantly reduce the query processing time for certain types of queries. With SPLENDID we also like to advocate the adoption of VOID statistics for Linked Data.

As next steps, we plan to investigate whether VOID descriptions can easily be extended with more detailed statistics in order to allow for more accurate cardinality estimates and, thus, better query execution plans. On the other hand, the actual query execution has not yet been optimized in SPLENDID. Therefore, we plan to integrate optimization techniques as used in FedX. Moreover, the adoption of the SPARQL 1.1 federation extension will also allow for more efficient query execution.

*Acknowledgments* The research presented in this paper has been supported by the German BMBF in the project CollabCloud. http://collabcloud.de/

## References

1. K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets – On the Design and Usage of voiD, the "Vocabulary Of Interlinked Datasets". In *Proceedings of the Linked Data on the Web Workshop*, Madrid, Spain, 2009.
2. F. Belleau, M.A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716, 2008.

3. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.
4. C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and Optimization of the SPARQL 1.1 Federation Extension. In *8th Extended Semantic Web Conference*, Heraklion, Greece, 2011.
5. K. Cheung, H. R. Frost, M. S. Marshall, E. Prud'hommeaux, M. Samwald, J. Zhao, and A. Paschke. A journey to Semantic Web query federation in the life sciences. *BMC bioinformatics*, 10 Suppl 1, January 2009.
6. S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, page 145, New York, New York, USA, 2011.
7. L.M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing Queries across Diverse Data Sources. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 276–285, Athens, Greece, 1997.
8. S. Harris and A. Seaborne. SPARQL Query Language 1.1, W3C Working Draft 26 January 2010. http://www.w3.org/TR/sparql11-query/.
9. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K-U. Sattler, and J. Umbrich. Data Summaries for On-Demand Queries over Linked Data. In *Proceedings of the 19th International World Wide Web Conference*, pages 411–420, Raleigh, NC, USA, 2010.
10. G. Ladwig and T. Tran. Linked Data Query Processing Strategies. In *Proceedings of the 9th International Semantic Web Conference*, pages 453–469, 2010.
11. F. Manola and E. Miller. RDF Primer, W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-primer/.
12. T. Neumann and G. Weikum. RDF-3X: a RISC-style Engine for RDF. In *Proceedings of the 34th International Conference on Very Large Data Bases*, pages 647–659, Auckland, New Zealand, 2008.
13. B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *Proceedings of the 5th European Semantic Web Conference*, pages 524–538, Tenerife, Canary Islands, Spain, 2008.
14. S. Schenk and S. Staab. Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web. In *Proceeding of the 17th International World Wide Web Conference*, pages 585–594, Beijing, China, 2008.
15. M. Schmidt, O. Görlitz, P. Haase, A. Schwarte, G. Ladwig, and T. Tran. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In *Proceedings of the 10th International Semantic Web Conference*, Bonn, Germany, 2011.
16. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP$^2$Bench: A SPARQL Performance Benchmark. In *Proceedings of the 25th International Conference on Data Engineering*, pages 222–233, Shanghai, 2009.
17. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *Proceedings of the 10th International Semantic Web Conference*, Bonn, Germany, 2011.
18. P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 23–34, Boston, MA, USA, 1979.
19. H. Stuckenschmidt, R. Vdovjak, G-J. Houben, and J. Broekstra. Index Structures and Algorithms for Querying Distributed RDF Repositories. In *Proceedings of the 13th International World Wide Web Conference*, pages 631–639, New York, NY, USA, 2004.
20. M.E. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *7th Extended Semantic Web Conference*, pages 228–242, Heraklion, Crete, Greece, 2010. Springer.
21. C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *Proceedings of the 34th International Conference on Very Large Data Bases*, pages 1008–1019, Auckland, New Zealand, 2008.