Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090607

Federated SPARQL Basic Graph Pattern Optimization using Links over the Web of Linked Data

Xuejin Li[1] *and Zhendong Niu*[1,2,3,*]

[1] School of Computer Science and Technology, Beijing Institute of Technology, 100081 Beijing, China
[2] Information School, University of Pittsburgh,15260 Pennsylvania, USA
[3] Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application, Beijing Institute of Technology, Beijing 100081, China

Received: 7 Feb. 2015, Revised: 7 Apr. 2015, Accepted: 8 Apr. 2015
Published online: 1 Nov. 2015

Abstract: The fast growth of the web of linked data raises new challenges for distributed query processing. Different from traditional federated databases, linked data sources cannot cooperate with each other. Hence, sophisticated optimization techniques are necessary for efficient query processing. In this paper, we formalize the problem of Basic Graph Pattern (BGP) optimization for federated SPARQL queries over the Web of Linked Data. We define and analyze the characteristics of source selection for links based static BGP optimization. The classes of bound subject and object associated with bound predicates of triple patterns are first used to select the set of relevant sources. Then links between linked data are used to prune the relevant sources of triple patterns. With the FedBench benchmark, we evaluate the performance of our approach of source selection for FedBench queries. The results of the evaluation show the feasibility of our approach.

Keywords: federated query processing, SPARQL, the Web of Data

1 Introduction

With the wide adoption of linked data principles, the World Wide Web has evolved from a global information space of linked documents to one where both documents and data are linked[10]. A large amount of structural data on the Web enable new types of applications which can aggregate data from different data sources and integrate fragmentary information from multiple sources to achieve a more complete view. Answering queries across multiple distributed Linked Data sources is a key challenge for developing this kind of applications.

Federated querying over the distributed data sources is called *virtual data integration*. User queries are decomposed into several sub-queries that are distributed to autonomous data sources which execute these sub-queries and return the results which are integrated locally. There are a high number of links in the Web of Linked Data. Yet, so far, only little attention has been paid to the effect of links between linked data on federated querying.

In this paper, we focus on *Basic Graph Pattern* (BGP) optimization for federated SPARQL queries over the Web of Linked Data. In SPARQL, a BGP is a set of triple patterns where a triple pattern is like an RDF triple except that each of the subject, predicate and object may be concrete (i.e. bound) or variable (i.e. unbound). Basic Graph Patterns are fundamental to SPARQL queries and other complex graph patterns can be constructed by BGP using SPARQL logical operators(UNION, OPTIONAL).

Listing 1: Example SPARQL query with one single BGP

```
SELECT ?Drug WHERE {
1 ?Drug rdf:type dbpedia-owl:Drug .
2 ?y owl:sameAs ?Drug .
}
```

The problem we are going to tackle in this paper is best explained by a simple example. Consider the BGP displayed in Listing 1 which represents a BGP of a SPARQL query executed over RDF data describing the life science domain. Assume that D_1, D_2 and D_3 are three data sources over the Web of Linked Data, the first triple

* Corresponding author e-mail: zniu@bit.edu.cn

pattern is relevant to D_1 and D_2 and the second triple pattern is relevant to D_3. Typically, the final query answers are obtained by the following steps: firstly, the first triple pattern is respectively matched on D_1 and D_2 which respectively produce the result set S_1 and S_2, the second triple pattern is matched on D_3 which produces the result set S_3; then, the final query result set S is generated by integrating these intermediate query results: $S = (S_1 \cup S_2) \cap S_3$. If we have known that there are not links between D_2 and D_3 by previous statistic information of D_1, D_2 and D_3, we can get the inference: $S_2 \cap S_3 = \emptyset$. Therefore, the remote request to D_2 is useless. We can get the final query result set S only by joining S_1 and S_3.

This paper is an extension of our work presented in [26]. In this paper, we presented the link-aware approach for source selection in more detail. Besides, we also study the join reordering of sub-queries and the way to execute distributed join operations in federated SPARQL queries over the Web of Linked Data.

Our main contribution presented in this paper is fourfold. (1)We formalize the problem of Basic Graph Pattern (BGP) optimization for federated SPARQL queries over the Web of Linked Data. (2)We propose an efficient approach of source selection. (3)We design an efficient approach for executing distributed join operations. (4)We perform comprehensive simulation study based on FedBench benchmark to evaluate our approaches.

The remainder of this paper is structured as follows. In Section 2 we review related work. Section 3 we present the background knowledge. Section 4 describes the statistical model and the approach of source selection based on it. The execution of join operations is presented in Section 5. An evaluation of our approach is given in Section 6. Finally, we conclude and discuss future directions in Section 7.

2 Related Works

Related work can be divided into two main categories: (a) source selection (b) query optimization.

2.1 Source Selection

DARQ [1] extends the popular query processor Jena ARQ to an engine for federated SPARQL queries. It requires users to explicitly supply a configuration file which enables the query engine to decompose a query into sub-queries and optimize joins based on predicate selectivity. Stuckenschmidt [2] presents an index structure called source index hierarchy which is used to determine information sources that contain instances of a particular schema path. Given a predicate path in a dataset, an index hierarchy is constructed, where the source index of the indexed path is the root element. Both two approaches require predicates of triple patterns to be bound. SemWIQ [3] requires all subjects of triple patterns to be variables and for each subject variable its type must be explicitly or implicitly defined. Additional information (another triple pattern or DL constraints) is needed to tell the type for the subject of a triple pattern. It uses these additional information and extensive RDF statistics to decompose the original user query. These requirements limit the variety of user queries.

In other cases, users are required to provide additional information to determine the relevant data sources. For instance, [4] theoretically describes a solution called Distributed SPARQL for distributed SPARQL query on the top of the Sesame RDF repository. Users are required to determine which SPARQL endpoint the sub-queries should be sent to by the GRAPH graph pattern. The association between graph names and respective SPARQL endpoints at which they reside is explicitly described in a configuration file. The W3C SPARQL working group has defined a federation extension for SPARQL 1.1 [5]. However, remote SPARQL queries require the explicit notion of endpoint URIs. The requirement of additional information imposes further burden on the user. On the other hand, the proposed approach hardly imposes any restrictions on user queries.

Recently, several attempts have been made to do source selection without local statistics. FedX [6] asks all known data sources by SPARQL ASK query form whether they contain matched data for each triple pattern presented in a user query. FedSearch [14] is based on FedX and extends it with sophisticated static optimization strategies. If the amount of known data sources is very large(it is common in an open setting), the query performance may leave much to be desired. SPLENDID [7] relies on the VOID [22] descriptions existing in remote data sources. However, a VOID description is not an integral part of Linked Data principles [8].

The link-aware source selection approach was first proposed by Stuckenschmidt [2]. They use predicate path index hierarchies of datasets for source selection. This approach requires predicates of triple patterns to be bounded, and then limit the variety of user queries. Another link-aware source selection approach is presented in [13]. It decomposes original queries based on general statistical models which form a local web of linked classes. Its statistical models are class-based, and the statistical models presented in this paper is property-based which can flexibly make a compromise between answer completeness and the time performance of query engines. Acosta et al. [24] present ANAPSID, an adaptive query engine that adapts query execution schedulers to SPARQL endpoints data availability and run-time conditions. Montoya et al. [25] extend the ANAPSID framework by an approach for link-aware source selection. Yet, this extension is based on evaluating namespaces and sending ASK queries to data sources at runtime. HiBISCuS [23] is an efficient hypergraph based source selection approach for SPARQL query federation

over multiple SPARQL endpoints. It identifies links between linked data based on authorities of URIs. URIs of classes defined by same data sources generally have the same authorities. Compared to our approach, HiBISCuS is more generic and usually over-estimate the set of sources capable for answering a query.

2.2 Query Optimization

Research on query optimization has a long history in the area of database systems. Concepts in these research areas have been adopted to optimize queries on local RDF stores. OptARQ [15] reorders triple patterns in SPARQL queries based on their selectivity. Hartig [16] adapted the query graph model (QGM) for SQL queries to represent SPARQL queries. Based on SQGMs, SPARQL queries are rewritten for optimization purpose. Due to the triple nature of RDF data, optimization for queries on local repositories has also focused on the use of specialized indices to accelerate the join operations, e.g. [17].

In [1] Quilitz et.al have adopted some of existing techniques from relational systems to federated SPARQL queries. They present a cost based optimization for join ordering. However, their estimation on the result size of joins is inaccurate by simply setting the selectivity factor for the join attributes to a constant. Because unbound queries generally returning a large result set, other join implementations are proposed as an alternative to local nested-loop implementation of joins, such as pipeline join [18] and semijoin [4]. While pipeline may produce too many concurrent access to remote data sources, semijoin tends to lead to program errors due to long query strings. In this paper, we propose a novel way, called range join, to execute join operations. By a factor, it can reach a compromise between the amount of network traffic sending to and downloading from remote data sources.

3 PRELIMINARY

For a better understanding, we develop the theory by means of the example BGP displayed in Listing 2 (i.e. the WHERE clause of the FedBench Benchmark [6] Life Science Query 5).

Listing 2: Example SPARQL query with one single BGP

```
1 ?drug rdf:type drugbank:drugs .
2 ?drug drugbank:keggCompoundId ?keggDrug .
3 ?keggDrug bio2rdf:url ?keggUrl .
4 ?drug drugbank:genericName ?drugBankName .
5 ?chebiDrug dc:title ?drugBankName .
6 ?chebiDrug bio2rdf:image ?chebiImage .
```

A BGP in SPARQL is defined as follows:

Definition 1.*A BGP is represented to be a graph G as a set $\mathscr{G}$ of undirected connected graphs. For each pair $(g_i, g_j) \in \mathscr{G}$, g_i and g_j are disconnected. A graph g is an ordered pair (V,E), where V is a set of distinct triple patterns and E is a set of distinct triple pattern pairs. For each pair $(v_i, v_j) \in E$, $v_i \in V, v_j \in V$, v_i and v_j share at least one variable.*

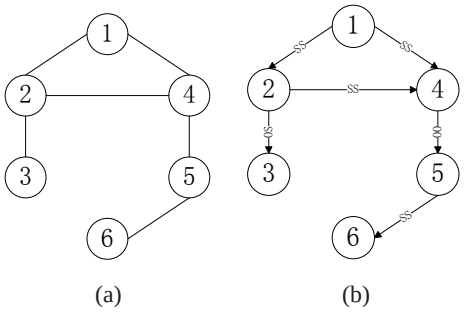


Fig. 1: Undirected connected graph $g_1 \in \mathscr{G}$ (a) and DAG d_g for Listing 2 (b)

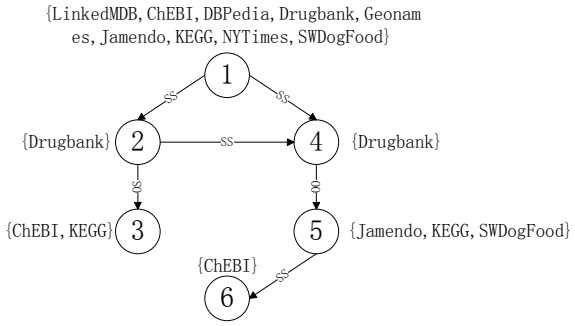
In Figure 1(a), we display the undirected connected graph $g_1 \in \mathscr{G}$ for the BGP in Listing 2. As the BGP triple patterns in Listing 2 are (transitively) joined, the graph G has only one component, thus $\mathscr{G}$ contains only the connected graph g_1. Note, that the numbers used for the nodes of g_1 in Figure 1(a) correspond to the numbers of the triple patterns of the BGP in Listing 2.

For a BGP B, links between linked data obviously do not affect the source selection of pairwise disconnected graphs $(g_i, g_j) \in \mathscr{G}$; the execution order of pairwise disconnected graphs $(g_i, g_j) \in \mathscr{G}$ also does not affect query performance as the overall result set corresponds to the Cartesian product of the result sets for g_i and g_j . Therefore, we can reduce the optimization problem for B to the optimization of each $g \in \mathscr{G}$. In the following, we focus on the optimization of connected graphs $g \in \mathscr{G}$.

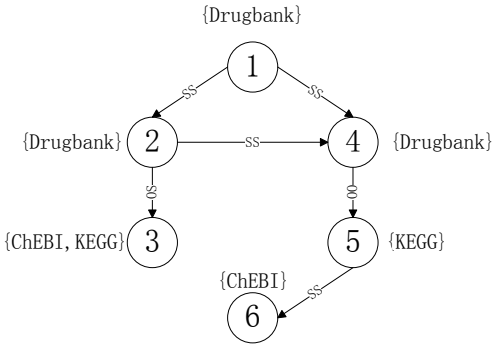
Definition 2.*An execution plan p_g for $g \in \mathscr{G}$ is a well defined order for the nodes of g. The set $\mathscr{P}_g$ is the execution plan space of $g \in \mathscr{G}$. An execution plan $p_g \in \mathscr{P}_g$ is an element of the space.*

An execution plan $p_g \in \mathscr{P}_g$ can be represented as a directed acyclic graph (DAG). We define $\mathscr{D}_g$ as the set of directed acyclic graphs for the execution plans in $\mathscr{P}_g$. Each DAG $d_g \in \mathscr{D}_g$ represents one or more execution plans p_g of an undirected connected graph $g \in \mathscr{G}$.

In Figure 1(b) we show the DAG corresponding to the execution plan p_g which executes the BGP of Listing 2 top-down, i.e. the triple patterns are evaluated in the same order as they are listed in Listing 2. For any two nodes $(v_i, v_j) \in d_g$, there is a directed path between v_i and v_j, if (1) the triple patterns corresponding to v_i and v_j are joined and (2) v_i is executed first in the execution plan. There is a clear relationship between the set $\mathscr{P}_g$ of execution plans p_g and the set $\mathscr{D}_g$ of directed acyclic



(a)



(b)

Fig. 2: The SRG $u \in \mathscr{U}$ of d_g shown in Figure 1(b) (a) and The Pruning Result of $u \in \mathscr{U}$ (b)

graphs d_g. More formally, we can state the following function $f : p_g \rightarrow d_g$ that is injective and not surjective. Thus, an execution plan p_g can be mapped uniquely to a DAG d_g, whereas a DAG d_g is an abstraction for one or more execution plans p_g.

Definition 3.*If each node (triple pattern) in the graph $d_g \in \mathscr{D}_g$ is attached a set of its relevant data sources, the new graph is called the Source Relevance Graph (SRG) of d_g, written as $s(d_g)$.*

We define $\mathscr{S}(g)$ as the set of source relevance graphs for the execution plans in $\mathscr{P}_g$; $M(s(d_g))$ as the set produced by matching triple patterns on their respective relevant data sources and joining the matched data locally. The Figure 2(a) shows the SRG of d_g shown in Figure 1(b). For the sake of simplicity, we use predicate based approach to determine relevant data sources of triple patterns where FedBench datasets are taken as the set of known data sources. If the bound predicate of a triple pattern is used by a data source, then the triple pattern and the data source are relevant.

Table 1: Statistics for single data source

No.	Field	Descriptions
1	predicate	a predicate in a data source
2	domain	a set of classes (the domain of the predicate)
3	range	a set of classes (the range of the predicate)

Table 2: Links in one single data source

No.	Field	Descriptions
1	linkType	the type of a link
2	startPredicate	the start predicate
3	endPredicate	the end predicate
4	intersection	the intersection of the class set

Table 3: Links between data sources

No.	Field	Descriptions
1	linkType	the type of a link
2	startPredicate	the start predicate
3	endPredicate	the end predicate
4	intersection	the intersection of the class set
4	startDS	the data source of the start point of the link
4	endDS	the data source of the end point of the link

4 Source Selection

Given a graph $g \in \mathscr{G}$, the matched data of g can be obtained by the following steps. Firstly, the SRG of g is constructed using previous statistics of known data sources. Then, triple patterns are respectively matched over their relevant data sources; the matched data for one single triple pattern is the union of all matched data from its relevant data sources. Finally, all matched data of triple patterns are joined into the matched data of g locally. Because that join operations are executed on the local mediator, this process generally needs a large number of network traffic and remote requests.

Source selection and distributed join operations are two key factors affecting the number of network traffic and remote requests. The approach for executing distributed join operations is presented in Section 5. Our approach for source selection includes two stages. The first stage has the aim of selecting relevant data sources for each single triple pattern, i.e. determining data sources that may contain matched data of one single triple pattern. The second stage has the aim of selecting relevant data sources for the whole BGP, i.e. excluding relevant data sources of triple patterns that potentially have no contributions for the final query answers.

4.1 Statistics

The need for customized summary statistics of RDF data for our purpose of BGP optimization is motivated by at least the following two arguments. Firstly, the relevant data source set of one single triple pattern is selected, i.e. constructing $\mathscr{S}(g)$ of $g \in \mathscr{G}$. Secondly, the SRGs of $s(d_g) \in \mathscr{S}(g)$ are pruned. For this purpose, we build three kinds of statistics for each known data source.

In Table 1 we show the statistics for single data source. Firstly, the property set P of a dataset is collected. Then, for each property $p \in P$, we compute its domain D and range R: (1)The subject of p is a URI. Its classes are generally defined in the data source hosting it. We compute the domain of p by the SPARQL query: SELCET different(?d) WHERE {?s p ?o. ?s rdf:type ?d}. (2)The object of p is a URI or Literal. Its classes are defined in the data source hosting it or other data sources. The range of p is computed by the SPARQL query: SELCET different(?r) WHERE {?s p ?o. OPTIONAL{?o rdf:type ?r}}. If the object of p is a URI and its classes are not defined by the current data source, the its classes are computed by dereferencing the URI. (3) URIs which classes not explicitly defined by rdf:type are assigned with the common class rdfs:Resource. For literals, the more specific the classes (data types) are divided, the more accurate the source selection will be. In this paper, we just assign literals with the common class rdfs:Literal. Finally, a *predicate tuple* D,p,R is constructed for each $p \in P$.

For each pair $((D_1,p_1,R_1),(D_2,p_2,R_2))$, if one of $\frac{2|D_1 \cap D_2|}{|D_1|+|D_2|} \geq \alpha$, $\frac{2|D_1 \cap R_2|}{|D_1|+|R_2|} \geq \alpha$, $\frac{2|R_1 \cap D_1|}{|R_1|+|D_1|} \geq \alpha$ and $\frac{2|R_1 \cap D_2|}{|R_1|+|D_2|} \geq \alpha$ is true, a link with a corresponding type is built, where α is a positive real number denoting the overlap ratio of two sets. The information of links in one single data source and links between data sources is respectively shown in Table 2 and Table 3.

4.2 Source Selection Approach

Given a BGP B, we first select relevant data sources for each triple pattern. The only available information for this process is the bound parts (non-variable) of triple patterns. According to the statistical model shown in Table 1, we tackle the bound subjects and objects of triple patterns as follows: (1)For a URI, its classes can be obtained by directly dereferencing it. If the classes of the URI can not be obtained, rdfs:Resource is the default one.(2)For a Literal, we assign it with a common class rdfs:Literal. Next, the set of relevant data source is determined by matching these information on statistics of known data sources.

Assume that $t:(s_t,p_t,?o_t)$ is a triple pattern with a bound subject and predicate, S_t is the set of classes of s_t, a tuple pattern $up = (S_t,p_t,?o)$ is constructed. The predicate tuple $u = (D,p,R)$ of DS matches up if and only if $p_t = p$ and $\frac{2|D \cap S_t|}{|D|+|S_t|} \geq \alpha$. The factor α has the same meaning presented above subsection. If up and u are matched then t and DS are relevant.

4.3 Pruning Algorithm

Given the set $\mathscr{S}(g)$ of $g \in \mathscr{G}$, each $s(d_g) \in \mathscr{S}(g)$ is pruned by removing potentially having no contributions to the matched data of g. In Algorithm 1, we provide the pseudo-code for the core optimization algorithm. The algorithm tests for each relevant data source of a triple pattern whether it has contributions to the results that the triple pattern join with other triple patterns (line 1-20); if all test results for a relevant data source of a triple pattern are false then removing the data source from the relevant data source set of the triple pattern (line 16-18). If there are no other triple patterns can produce results by joining with a triple pattern on a data source, then the data source is excluded from the relevant data source set of the triple pattern (21-23). Note that if the relevant data source set of any nodes in $s(d_g)$ becomes empty after pruning then removing $s(d_g)$ from $\mathscr{S}_g$. The algorithm returns the pruned $s(d_g)$ as the result.

Algorithm 1 Prune the Source Relevance Graph the $s(d_g)$ of $p_g \in \mathscr{P}_g$

1: $s(d_g) \leftarrow GenerateDAG(p_g)$
2: $s'(d_g) \leftarrow Copy(s(d_g))$
3: **for** i=1 to size(p_g) **do**
4: $v_i \leftarrow Node(p_g[i])$
5: $startSet \leftarrow GetRelevantSources(v_i)$
6: $E \leftarrow OutgoingEdges(v_i)$
7: **for** each $ss \in startSet$ **do**
8: $endTemp \leftarrow \varnothing$
9: **for** each $e \in E$ **do**
10: $v_j \leftarrow TargetNode(e)$
11: $tpLink \leftarrow GetLinkType(e)$
12: $endSet \leftarrow GetRelevantSources(v_j)$
13: $ess = GetDataSourcesByLink(endSet,tpLink,ss)$
14: $VistSources(v_j,ess)$
15: $endTemp \leftarrow endTemp + ess$
16: **end for**
17: **if** $endTemp = \varnothing$ **then**
18: $RemoveRelevantSource(s'(d_g),v_i,ss)$
19: **end if**
20: **end for**
21: **end for**
22: **for** each $v \in Nodes(s'(d_g))$ **do**
23: $RemoveNotVistedSources(v)$
24: **end for**
25: **return** $s'(d_g)$

In Figure 2(b) we show the pruned $s(d_g) \in \mathscr{S}(g)$ based on links between FedBench datasets according to algorithm 1. For the edge (1,2) and (1,4), a type *SS* link only exists in Drugbank with respect to relevant data sources of node 1, 2 and 4. Hence, only Drugbank is considered to be relevant for node 1. For the edge (4,5), Drugbank has the type *OO* links only towards KEGG. Therefore, Jamendo and SWDogFood are removed from the relevant data source set of the node 5. $M(s(d_g))$ is equal to $M(s'(d_g))$. However, the network traffic concerning $M(s(d_g))$ is far more than the one concerning $M(s'(d_g))$.

5 Optimization for Join Operations

Query optimization is a fundamental and crucial subtask of query execution in database management systems. While query optimization for a single database focus on join reordering of triple patterns, query optimization in distributed environments is a more sophisticated task. The optimization goal is to find the execution plan which is expected to return the result with minimum remote requests and network traffic.

Given a set $\mathscr{S}(g)$ and for each $s(d_g) \in \mathscr{S}(g)$ has been pruned, the matched data $M(g)$ of g is the union of $M(s(d_g)$ for all $s(d_g) \in \mathscr{S}(g)$. The aim of the optimization for join operations is to find the optimal $p_g \in \mathscr{P}_g \rightarrow d_g$ to compute the $M(s(d_g)$ with minimum cost w.r.t. network traffic.

5.1 Join Reordering

To minimize costs of network traffic, if a group of triple patterns share a set of relevant data sources then these triple patterns should be sent as a whole to the respective relevant data sources, i.e. the idea of the *exclusive groups* presented in [6]. Hence, the task for join reordering is to find the optimal order for joining a set of groups of triple patterns.

Algorithm 2 Join Order Optimization

```
1:  GS ← {gs_1, gs_2, ..., gs_n}
2:  orderedGS ← ∅
3:  while GS ≠ ∅ do
4:      mincost ← +∞
5:      for each gs ∈ GS do
6:          cost ← joinCost(gs, preGS)
7:          if cost < mincost then
8:              temp ← gs
9:              mincost ← cost
10:         end if
11:     end for
12:     orderedGS ← append(orderedGS, temp)
13:     GS ← GS − {temp}
14: end while
15: return orderedGS
```

In Algorithm 2, we provide the pseudo-code for the join order optimization algorithm. The algorithm is a variation of the variable counting technique proposed in [12]. It determines the group of triple patterns with lowest cost from the remaining items (line 4-11) and appends it to the tail of the ordered list (line 12). For cost estimation (line 6) the number of free variables is counted considering already bound variables, i.e., the variables that are bound through a join argument that is already ordered in the ordered list.

5.2 Join Execution

By computing the joins through buffering the obtained variable binding sets and sends them in a batch as value constraints to remote SPARQL endpoints, i.e., as a distributed semijoin, it is possible to reduce the number of requests and network traffic. The range of a join variable binding set is divided into some equal-width section. Not all bound values of the join variable but the breakpoints of the range are used to construct value constraints. The overall idea of this optimization is to provide a way to reach a compromise between the amount of network traffic sending to and downloading from remote data sources. We propose the *range join* technique and discuss the technical insights below.

In the following, we illustrate range join processing for the join triple pattern $t_1 : (s_1 \ p_1 \ ?o_1)$ and $t_2 : (?o_1 \ p_2 \ ?o_2)$. For the example, t_1 is first executed and the bound value sequence of $?o_1$ is $B = (b_1, b_2, ..., b_n)$ where $b_1 \leq b_2 \leq \cdots \leq b_n$; the set B is divided into m sections: $((b_1, \ldots, b_{n/m}), (b_{n/m}, \ldots, b_{2n/m}), \ldots, (b_{(m-1)n/m}, \ldots, b_n))$; the value constraints of t_2 are constructed as $(b_1 \leq ?o_1 \leq b_{n/m})$ OR $(b_{n/m} < ?o_1 \leq b_{2n/m})$ OR ... OR $(b_{(m-1)n/m} < ?o_1 \leq b_n)$; Finally, t_2 attached with the value constraints is sent to its relevant data sources.

Obviously, the range join is equal to the original implementation of semijoin [4] when $m = 1$ and is equal to the pipeline join [18] when $m = n$. Therefore, we can adjust the value of m to reach a compromise between the amount of network traffic sending to and downloading from remote data sources. The selection of the optimal value of m is a little difficult. It needs the information of data distribution in data sources. For our setup, we just set the value of m to 10.

6 EXPERIMENTAL STUDY

We have developed a prototype system(LDMS[3]) implementing the proposed approaches and conducted an experimental study to empirically analyze the effectiveness of it compared with several existing federated SPARQL query systems.

Our evaluation is based on FedBench[4][9]. In contrast to other SPARQL benchmarks[19,20], FedBench focus on testing and analyzing the performance of **federated** query processing strategies on semantic data. It includes two subsets of data sources in the Linked Data cloud: Cross Domain(DBpedia, NYTimes, LinkedMDB, Jamendo, GeoNames) and Life Sciences(KEGG, Drugbank, ChEBI, DBpedia). For each data set, it defines seven queries. The overview of the data sets is shown in Table 4(a) in terms of number of triples(#Triples), size of statistical models and time taken to create them in hh:mm:ss. Queries are shown in Table 4(b) in terms of

[3] LDMS is available as Java source code(eclipse project) from the SVN repository: https://svn.code.sf.net/p/semwldms/code/LDMS/trunk

[4] FedBench can be downloaded at http://code.google.com/p/fbench/

Table 4: FedBench datasets and queries used for the evaluation.

(a)

Dataset	#Triples	SM Size	SM Time
DBpedia	43.6M	332KB	01:01:18
NYTimes	335k	10KB	00:00:27
LinkedMDB	6.15M	36KB	00:07:36
Jamendo	1.05M	3KB	00:00:12
Geo Names	108M	7KB	02:04:47
SW DogFood	104k	64KB	00:00:30
KEGG	1.09M	4KB	00:01:30
Drugbank	767k	19KB	00:01:12
ChEBI	7.33M	3KB	00:04:12

(b)

Query	#BGPs	#Patterns	#Results
CD1	2	3	90
CD2	1	3	1
CD3	1	5	2
CD4	1	5	1
CD5	1	4	2
CD6	1	4	11
CD7	1	4	1
LS1	2	2	1159
LS2	2	3	333
LS3	1	5	9054
LS4	1	7	3
LS5	1	7	393
LS6	1	5	28
LS7	2	5	144

number of BGPs and patterns in the WHERE clause and size of results.

The data server was set up using OpenRDF Sesame framework which provides a query service (SPRAQL endpoint) for each data source. Benchmark datasets simulated on the same physical host and were respectively loaded as a single repository with the type of Sesame Native Store. The prototype system(i.e. test client) was on a Windows XP with two Dual-Core Intel Xeon processors (2.8 GHz) and 3GB memory. The server was running a 64 Bit Debian Linux Operation System with two Intel Xeon CPU E7530 processors (each with twelve cores at 2 GHz), 32 GB main memory. The statistical models for data sources were loaded into memory when starting the system.

6.1 Evaluation of Source Selection

First, we show the result of evaluation of source selection. Assume that N and N_e respectively are the total number of sources in the data and the number of sources that are actually accessed for answering a query, we define $\frac{N_e}{N}$ is the precision for source selection. The precision measures the quality of source selection. In other words, the precision gives an idea that how much costs we take for answering a query when not losing the recall of results. The source selection factor α was set to 1, i.e. a link between two predicate tuples is constructed when the corresponding two sets are equal.

Table 5: Comparison of precision (%) for source selection

Query	DARQ	SPLENDID	FedX	ANAPSID	HiBISCuS	LDMS
CD1	10	10	30	10	75.6	75.6
CD2	33.7	100	100	33.7	100	100
CD3	25	100	12.5	33	75.6	100
CD4	16.7	100	1.56	12.5	50	100
CD5	33.3	100	12.5	25	100	100
CD6	33.3	5.56	5.56	12.5	75.6	100
CD7	33.3	25	3.125	37.5	50	100
LS1	100	100	100	100	100	100
LS2	10	10	10	12.5	25	50
LS3	25.6	100	12.5	25	95	100
LS4	51.5	100	100	13.5	100	100
LS5	26.5	12.5	12.5	16.2	100	100
LS6	25	12.5	12.5	36.6	75.6	100
LS7	35	75	37.5	35	100	100
Avg.	**23.78**	**67.9**	**31.77**	**28.79**	**80.17**	**94.69**

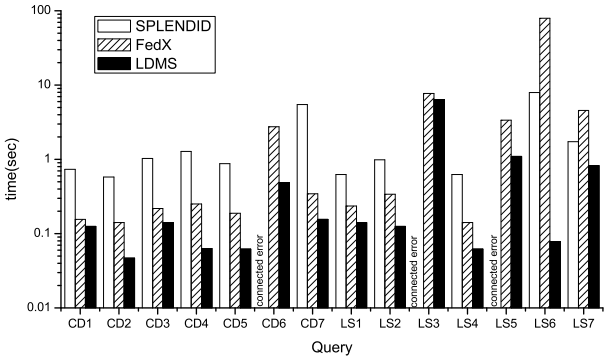


Fig. 3: The Comparison of Time Performance with other state-of-the-art Federated SPARQL Query Systems

Table 5 shows the precision for FedBench queries. We observe an avg. precision of 94.69% for LDMS. It means that only 5.31% useless remote requests during answering queries by LDMS. The high precision shows that our approach is very well suited to prune the space of execution plans for all queries. LDMS and HiBISCuS take links over the Web of Linked Data into account and achieve higher precisions than other systems not doing this task. The evaluation results also show that links between data are important factors for source selection when tackling distributed queries on triple-style data (e.g. linked data).

6.2 Evaluation of Execution Time

In this section, we evaluate the time performance of LDMS that implement approaches presented in this paper, and compare with SPLENDID and FedX. Besides of one warm execution, these three systems execute all queries five times and the average time is used for comparisons. The difference in these three systems are technologies of source selection and join optimization. FedX uses a runtime source selection and heuristics to reordering join operations. Both SPLENDID and LDMS use statistical information to select relevant sources for triple patterns and optimize query plans. While SPLENDID implements both nested loops join and pipeline join, The technologies

used by FedX for distributed join operations depend on bound join. LDMS reorders joins based on the result of cardinality estimation of sub-queries and executes join operations in the way of range join.

The results of the experiment is shown in Figure 3. From the Figure 3, we can see that the time performance for FedX and LDMS is comparable for the query LS3. It is clear that this query has more results than other queries , and the cost of query decomposition account for very small proportion of the overall time cost. For CD6, LS3 and LS5, SPLENDID encounters connection errors due to opening too many connections to data sources. SPLENDID produces query plans based on pre-statistics and prunes them using additional ASK queries. It leads to a lower time performance for some queries, i.e. CD1-5, CD7, LS1-2, LS4. The lack of FedX is that it may generate many useless execution plans for some queries. In some query plans when evaluating query LS6, the first sub-query evaluated has non-empty result set. Many intermediate results transferred to local federator has no contributions to the final query results because the join operation between them and the next sub-query produces an empty set.

The approach of source selection presented in this paper prunes a lot of data sources potentially having no contributions for the final query answers, such that avoiding many useless remote requests and network traffic. Furthermore, the range join can decrease the number of uploading data needed for join operations. Though the range join may increase the number of downloading data, the downloading speed is generally faster than the uploading speed for most of data sources. Hence, LDMS is faster than other systems for most of queries.

7 Conclusions

We have presented an approach for evaluating SPARQL queries over the Web of Linked Data. Source selection and distributed join operations are key factors concerning performance of linked data query engines. We presented a link-aware source selection approach and a novel way to execute distributed join operations. We evaluated our source selection approach against DARQ, SPLENDID, FedX, ANAPSID and HiBISCuS. The evaluation shows that the precision of source selection of the first four systems is improved significantly on average. We also compare the time performance with SPLENDID and FedX. The results of evaluation show that LDMS is faster than other two systems for most of queries.

The source selection approach presented in this paper is similar with the one proposed in [13]. The difference between them lies in the type of their statistics model. The former is based on a property link graph and the later is based on a class link graph. From the evaluation results we can see that the link-aware source selection strategies are better than others with respect to the accuracy of the source selection. Due to the nature of the linked data, the link relationship between resources and between data sources is very important in the query processing.

Besides, distribution join operations are in the center for most of query engines for distributed data. The range join presented in this paper is a novel approach and the experiments show its effectiveness.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Project No. 61370137), the International Corporation Project of Beijing Institute of Technology (No. 3070012221404) and the 111 Project of Beijing Institute of Technology.
The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] B. Quilitz, U. Leser, Querying distributed RDF data sources with SPARQL, Proceedings of the 5th European Semantic Web Conference (ESWC), 524-538 (2008).
[2] H. Stuckenschmidt, R. Vdovjak, G.J. Houben, et al. Index structures and algorithms for querying distributed RDF repositories, Proceedings of the 13th international conference on World Wide Web, 631-639 (2004).
[3] A. Langegger, W. Wöß, M. Blöchl, A semantic web middleware for virtual data integration on the web, Proceedings of the 5th European Semantic Web Conference (ESWC), 493-507 (2008)
[4] J. Zemanek, S. Schenk, V. Svatek, Optimizing SPARQL Queries over Disparate RDF Data Sources through Distributed Semi-Joins, International Semantic Web Conference (Posters & Demos), 2008.
[5] S.H Garlik, A. Seaborne, E, Prudhommeaux, SPARQL 1.1 query language, World Wide Web Consortium, 2013.
[6] A. Schwarte, P. Haase, K. Hose, et al, FedX: Optimization techniques for federated query processing on linked data, The Semantic WebCISWC 2011, Springer Berlin Heidelberg, 601-616 (2011).
[7] O. Görlitz, S. Staab, SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions, COLD, 782 (2011).
[8] T. Berners-Lee, Design issues: Linked data (2006), http://www.w3.org/DesignIssues/LinkedData.html.
[9] M. Schmidt, O. Görlitz, Haase, Peter et al, FedBench: a benchmark suite for federated semantic data query processing, The Semantic Web–ISWC 2011, 585-600 (2011).
[10] C. Bizer, T. Heath and T. Berners-Lee, Linked data-the story so far, International Journal on Semantic Web and Information Systems (IJSWIS), Volume 5, IGI Global, 2009.
[11] C. Bizer, A. Schultz, Benchmarking the performance of storage systems that expose SPARQL endpoints, World Wide Web Internet And Web Information Systems, 2008.

[12] M. Stocker, A. Seaborne, A. Bernstein, et al, SPARQL basic graph pattern optimization using selectivity estimation, Proceedings of the 17th international conference on World Wide Web, 595-604 (2008).
[13] X. Li, Z. Niu, C. Zhang, Towards Efficient Distributed SPARQL Queries on Linked Data, Algorithms and Architectures for Parallel Processing, Springer International Publishing, 259-272 (2014).
[14] A. Nikolov, A. Schwarte, C. Htter, FedSearch: Efficiently Combining Structured Queries and Full-Text Search in a SPARQL Federation, The Semantic WebCISWC 2013. Springer Berlin Heidelberg, 427-443 (2013).
[15] A. Bernstein, C. Kiefer, M. Stocker, Optarq: A sparql optimization approach based on triple pattern selectivity estimation, Citeseer, 2007.
[16] O. Hartig, R. Heese, The SPARQL query graph model for query optimization, The Semantic Web: Research and Applications. Springer Berlin Heidelberg, 564-578 (2007).
[17] A. Harth, S. Decker, Optimized index structures for querying RDF from the web, Web Congress, 2005.
[18] O. Hartig, C. Bizer, J.C. Freytag, Executing SPARQL queries over the web of linked data, ISWC 2009, Springer Berlin Heidelberg, 293-309 (2009).
[19] C. Bizer, A. Schultz, The berlin sparql benchmark, International Journal on Semantic Web and Information Systems (IJSWIS) **5**, 1-24 (2009).
[20] M. Morsey, J. Lehmann, S. Auer, et al, DBpedia SPARQL benchmarkCperformance assessment with real queries on real data, ISWC 2011. Springer Berlin Heidelberg, 454-469 (2011).
[21] L. Günter, T, Thanh, Linked data query processing strategies, ISWC 2010, Springer, 453-469 (2010).
[22] K. Alexander, M. Hausenblas, Describing linked datasets-on the design and usage of void, the vocabulary of interlinked datasets, In Linked Data on the Web Workshop, in conjunction with 18th International World Wide Web Conference, 2009.
[23] M. Saleem, A.C.N. Ngomo, Hibiscus: Hypergraph-based source selection for sparql endpoint federation, The Semantic Web: Trends and Challenges, Springer International Publishing, 176-191 (2014).
[24] M. Acosta, M.E. Vidal, T. Lampo, et al, ANAPSID: An adaptive query processing engine for SPARQL endpoints, The Semantic WebCISWC 2011. Springer Berlin Heidelberg, 18-34 (2011).
[25] G. Montoya, M .E. Vidal, M. Acosta, A Heuristic-Based Approach for Planning Federated SPARQL Queries, COLD, 905 (2012).
[26] X. Li, Z. Niu, C. Zhang, et al, LAW: Link-AWare Source Selection for Virtually Integrating Linked Data, Technologies and Applications of Artificial Intelligence. Springer International Publishing, 239-248 (2014).

Xuejin Li received the MS degree from Beijing Institute of Technology. He is currently a PhD Student in the School of Computer Science and Technology at Beijing Institute of Technology, China. His research interests are in the areas of semantic web, linked data, distributed query processing.

Zhendong Niu received his PhD degree from Beijing Institute of Technology. He is currently a Professor in the School of Computer Science and Technology at Beijing Institute of Technology and an Adjunct Research Professor in the Information School at University of Pittsburgh. His research interests include computer software architecture, intelligent education system, digital library and neural information system.