Implementation of Federated Query Processing on Linked Data

Yuchao Zhou, Suparna De and Klaus Moessner, *Member IEEE*

***Abstract*—As the number of Linked Data sets increases with more and more interconnections defined between them, querying a single data set is no longer enough for users who need data from mixed domains. The requirement to query data from different data sets motivates the research into federated queries. Network latency is one of the key factors which affect the performance of a federated query. The influence of network latency can be minimised by decreasing the number of remote requests, which is related to the number of joins. In this paper, we provide a mechanism for federated querying based on subject and sameAs grouping techniques. Exploiting the benefits of proposed grouping methods, the number of joins during a federated query has been reduced, thus improving the performance of the entire query. We have evaluated our approach against other existing approaches, using an existing benchmark suite and found that our approach performs better than comparable approaches for queries that are not highly selective.**

***Index Terms*—Linked Data, Federated Query Processing**

I. Introduction

FOLLOWING the rules [1] designed by Tim Berners-Lee, and with increasing numbers of data sets being provided by data providers, the Linked Data concept is moving towards the realisation of the Web of Data vision. The Linked Data concept allows data providers to publish data on the Semantic Web using URIs and RDF [2] and to connect related data together. Linking data to other sources enables users to obtain more information relating to a particular data item by exploring the links across different concepts and domains. To date, the most popular standard for publishing Linked Data is RDF. Meanwhile, SPARQL [3], the query language for RDF, is also becoming a de-facto standard on Linked Data, with data sets such as DBpedia [4], LinkedMDB [5], providing SPARQL endpoints for querying. Based on RDF and SPARQL, Linked Data allows users to request data with a query to the concerned data set through its endpoint. However, due to the domain limitation of one data set, querying a single data set is insufficient for users who want data from mixed domains. Thus, federated querying mechanisms are required that can query data across various data sets.

There are two major trends for federated querying. One is to build a data warehouse that collects data from different data sources in advance so that the federated query can be answered directly by the data warehouse endpoint. The other is distributed query processing, which decomposes the federated query into sub-queries in order to detect which data set can answer which sub-query, then sends the sub-queries to the relevant endpoints and finally combines the results. Due to unavoidable network latency, executing a query over the Internet is time consuming. The data warehousing approach has a better query performance because it executes the query locally. Moreover, the data in the data warehouse can be restructured for a faster response when collected from various data sources. However, updates of original data sources are hard to be detected by data warehouse. Therefore, the data in data warehouse are not guaranteed to be up to date. On the other hand, distributed query processing query live data sets so that it guarantees that the most recent data is retrieved. Furthermore, it is flexible to be extended to new data sets as compared to the data warehousing approach. Moreover, the data warehouse can be treated as one of the data sets being queried by the distributed query processing approach. However, query optimisation techniques are required before the query execution for efficient query processing.

In this paper we focus on the distributed query processing method and extend it with optimisation techniques of subject and sameAs grouping. An implementation and a comparison with other comparable approaches is presented, which concludes that our approach achieves a better performance for queries that are not highly selective by reducing the number of join operations.

The remainder of the paper is organised as follows: a classification of distributed federated query processing methods is presented in Section II, followed by a review of existing work in Section III. The design and implementation of our approach is explained in Section IV. Evaluation results and discussions are presented in Section V, followed by conclusion and directions for future work in Section VI.

II. Classification of Distributed Federated Query Processing Strategies

Ladwig and Tran [6] classify distributed federated query processing based on three main strategies: top-down, bottom-up, and mixed strategy. For the bottom-up strategy,

Manuscript submitted on July 17, 2013. This article describes work undertaken in the context of the iCore ("Internet Connected Objects for Reconfigurable Ecosystems") project (http://www.iot-icore.eu/). iCore is an EU Integrated Project funded within the European 7th Framework Programme, contract number: 287708.

Yuchao Zhou, Suparna De and Klaus Moessner are with the Centre for Communication Systems Research, University of Surrey, Guildford, Surrey, GU2 7XH United Kingdom. (phone: 44-1483-686177; fax: 44-1483-686011; e-mail: {yuchao.zhou, S.De, K.Moessner}@surrey.ac.uk).

new data sets are discovered during query execution. For the top-down strategy, the complete information of data set is known before querying. The information includes endpoints, VoID, and statistics of data in data sets, which support source selection and optimisation. For the mixed strategy, part of data sets' information is known and other unknown data sets are discovered during execution. As the authors in [6] also make an evaluation of the strategies and conclude that the top-down strategy achieves better performance than the others, we focus on this particular strategy.

For the top-down strategy, some components are necessary. Federated query statements contain triples (a triple as applied in this article indicates an RDF triple as well as its corresponding statement in SPARQL, which is in the form of subject, predicate, and object) that can be attributed to different data sets. The federated query should be decomposed into triples so that they can be sent to the relevant data sets. So query decomposer is necessary. Meanwhile, source selection is required to derive the relevant data sets. The next one is a join ordering component that reorders joins of the decomposed sub-queries or that of results tables. Join ordering is important for federated querying because the cost of first join determines the cost of the entire query mostly. The final one is a query execution component which is used to get results.

The top-down strategy can be classified into two techniques: pre-join and post-join. Pre-join is the mechanism that starts with a triple and then uses bind-join [7] or semi-join [8] to map the common variable of next triple with each intermediate result to form subsequent joins. In contrast, post-join executes each triple pattern individually, and combines the results after all intermediate results have been obtained. Due to network latency and huge amounts of data to be transported over the Linked Data cloud, the time involved in query execution is much higher than that in query statement pre-processing and post-processing. So query optimisation techniques always add pre-processing components to reduce the cost of federated query execution. Most approaches (DARQ [9], FedX [10], SPLENDID [11]) focus on the join ordering, which estimates the cost of join orders and selects the plan of join order of the minimal cost in order to get a better performance. This technique also requires a cost estimate component. The precision of the cost estimation affects the performance of the federated query. FedX [10] introduces a group component, which groups some triples in a query first such that the number of joins decreases. As the cost of a join contains both intermediate processing as well as the execution time for the join, reducing the number of joins does improve the performance of the federated query.

III. Related Work

A number of the existing works implementing the top-down strategy include DARQ [9], FedX [10], and SPLENDID [11]. We also review SPARQL 1.1 [12] as an approach for federated querying.

DARQ handles the steps from parsing to query planning to optimisation to query execution. Specifically, it first decomposes the query and builds sub-queries from each triple. Then service description, which is pre-processed description information of data sets, is applied, to choose which data set should be employed to execute the sub-queries. Matching according to service description is based on predicates, so DARQ only supports queries with bound predicates triples. In the optimisation part, DARQ provides logical and physical optimisations including query rewriting, as well as nested loop join. As it executes a triple as a join, the number of joins is not optimised. Therefore, even though DARQ provides some optimisation mechanisms, it does not perform efficiently.

In contrast to DARQ, FedX uses the SPARQL ASK query to select the data sets. ASK query is a request sent to the endpoint asking whether the corresponding data set can answer one or several triples that wrapped in the query. TRUE/FALSE value will return. So FedX does not require pre-storing the related description information such as predicates used in a data set to specify whether the query can be answered by that data set. Moreover, the approach using ASK query allows more query statements to be supported, not only queries with bound predicate triples. In the processing step, it also separates the query into triples, however, it adds an exclusive group component that groups the triples that return a TRUE value from the ASK query from only one data set, which means triples that can be attributed to only one data set are grouped together. The grouped triples are formed into a sub-query to be sent to the relevant data set together. By doing this, both the number of remote requests and the number of joins is reduced compared to DARQ. An evaluation of FedX against DARQ [10] shows an improvement of query performance for most queries.

FedX provides a heuristic method for cost estimation of every sub-query which supports plan of join ordering, whereas SPLENDID applies VoID [13] towards the goal of more accurate cost estimation and fine source selection. SPLENDID provides two steps of source selection which uses VoID to decide the source of bound predicate triples at first, then determines the source of unbound predicates with the ASK query. Regardless of the cost of network communication, the cost of sending queries and receiving results are considered during cost estimation. Although SPLENDID could give a precise estimate for bound predicates, it estimates the cost of bound subjects and objects according to their average selectivity which is based on the assumption that they are uniformly distributed and independent from the predicate. In queries where the distributions of subjects and objects depend on the data set, this may result in inaccurate estimations.

The above implementations are based on SPARQL 1.0. SPARQL 1.1, the version focusing on expressing queries across diverse data sets, has been approved as a W3C recommendation recently. The details of how SPARQL 1.1 performs a federated query are presented in [12].

IV. Design and Implementation

A. *Underlying Assumptions*

In [14], the authors state how to publish Linked Data on the Web. According to [14], the subject of a RDF triple is used to identify resource of a dataset. The predicate indicates the relation between subject and object. The object is the value of the property and can be literal value as well as the URI of another resource. Meanwhile they state that URIs should better begin with the domain name of a dataset, not that of the others For example, URIs of data in DBpedia always start with http://dbpedia.org/ and ones in LinkedMDB start with http://data.linkedmdb.org/. We assume that all the Linked Data sets apply the above statements. The following assumptions are defined for our implementation.

Assumption 1: Data sets define their own data/resources as the triple subject and the data/resources of another data set as the triple object.

This assumption can be verified with similar observations made in the current literature [14], where it is recognised that the most valuable RDF links are those that connect a resource to external data published by other data sources, with the technical realisation taking the subject URI from one data source and the object URI from another data source.

Assumption 2: In a query triples with the same subject can be only contained in the same data set.

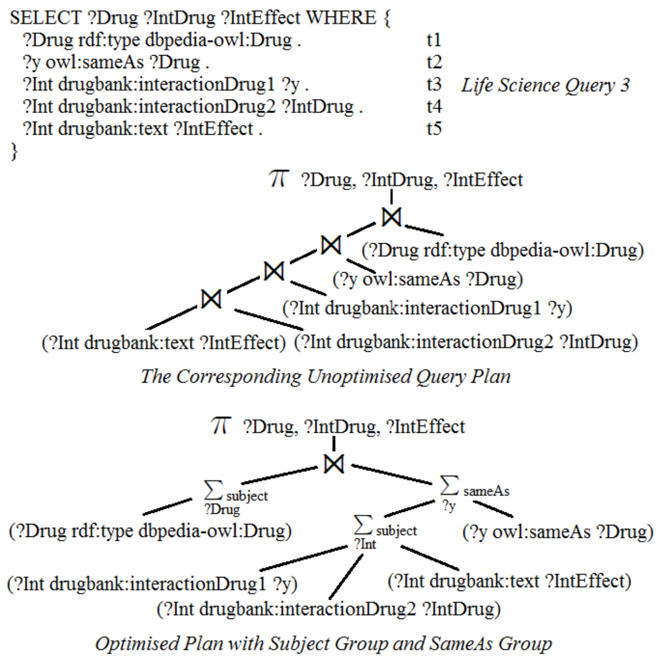
This assumption relates to the provenance of data. Following on from Assumption 1, data sets usually define their own data as subjects in RDF triples, with the subject having a URI beginning with the domain name of the data set. Since the URI of the subject is unique and belongs to the data set, the triples with the same subject can only contained in one given data set.

Assumption 3: In a federated query there are always some triples that constrain the subject of the link triple which links data in one data set to another.

Excluding the query that finds out all triples that link to other data sets: '*select * where {?data_in_this_data_set owl:sameAs ?data_in_other_data_set},*' for the link triple, '?sub owl:sameAs ?obj,' a federated query always gives some triples that constrain the '?sub' so that to know some details of '?sub' and to impose constraints on the '?sub' in a data set. As predicate 'owl:sameAs' is commonly used in Linked Data sets and the subject '?sub' decides which data set the triple is in, the link triple with predicate 'owl:sameAs' and triples constraining the subject must be in one data set. So combining 'owl:sameAs' triple with triples that '?sub' belongs to and sending them together to get results does not change the number of the results.

B. Overall Process

The design of our proposed federated query processing approach is described as follow. The federated query statements are first parsed and decomposed into triples. Triples which share the same subject are grouped together. Then the triple with predicate 'owl:sameAs,' which links one data set to another, is grouped with triples which contain its subject. These two grouping mechanisms are based on the above assumptions. Next, the grouped triples are wrapped in an ASK query which is sent to all data sets to test whether they can provide results. For each data set, the triples returning TRUE values from ASK queries are wrapped in a SELECT query and sent to the relevant endpoint to get results (Execution phase). After all intermediate results returned from data sets, the final results are generated by



```
SELECT ?Drug ?IntDrug ?IntEffect WHERE {
  ?Drug rdf:type dbpedia-owl:Drug .                t1
  ?y owl:sameAs ?Drug .                            t2
  ?Int drugbank:interactionDrug1 ?y .              t3   Life Science Query 3
  ?Int drugbank:interactionDrug2 ?IntDrug .        t4
  ?Int drugbank:text ?IntEffect .                  t5
}
```

π ?Drug, ?IntDrug, ?IntEffect
⋈
(?Drug rdf:type dbpedia-owl:Drug)
(?y owl:sameAs ?Drug)
(?Int drugbank:interactionDrug1 ?y)
(?Int drugbank:text ?IntEffect)
(?Int drugbank:interactionDrug2 ?IntDrug)
The Corresponding Unoptimised Query Plan
π ?Drug, ?IntDrug, ?IntEffect
∑ subject ?Drug
∑ sameAs ?y
∑ subject ?Int
(?Drug rdf:type dbpedia-owl:Drug)
(?y owl:sameAs ?Drug)
(?Int drugbank:interactionDrug1 ?y)
(?Int drugbank:text ?IntEffect)
(?Int drugbank:interactionDrug2 ?IntDrug)
Optimised Plan with Subject Group and SameAs Group

Fig. 1. Life Science Query 3, the Corresponding Unoptimised Query Plan, and Optimised Execution Plan with Subject Group and SameAs Group (∑). From the figure, we could see that the number of joins is reduced by applying our optimisation techniques.

performing table joins of the intermediate results (Table Join phase).

Following the assumptions, the query processing flow can be best explained with an example available in [15]. For all drugs listed in DBpedia, the query finds all drugs in drugbank they interact with, along with an explanation of the interaction. It is the Life Science Query 3 shown in Fig. 1. The following steps will be performed:

1) The query will be first separated into triples, as depicted above: t1, t2, t3, t4, t5.
2) Triples will be grouped by subject first, i.e. the triples which share a subject are grouped together into the following groups: g1 {t1}, g2 {t2}, g3 {t3, t4, t5}. This is depicted in Fig. 1 by the □ operator with the corresponding subject subscript.
3) g2 {t2} is detected that only contains one triple with predicate owl:sameAs. Its subject '?y' is identified and grouped with g3, which includes the variable ?y. The resultant groups thus are: g1 {t1}, g2 {t2, t3, t4, t5}. The □ operator with sameAs subscript in Fig. 1 depicts this step.
4) Groups of triples will be first wrapped in ASK queries and be sent to all data sets to test which data set can answer which group. Then the group whose ASK query returning a TRUE value is wrapped in a SELECT query and sent to get the answer. In this example, g1 will be sent to DBpedia, whereas g2 be sent to drugbank.
5) Finally, two intermediate results tables are joined together, which is presented by the ⋈ operator in Fig. 1.

Apart from the component which controls the entire processing flow and results' visualisation, there are three main parts of our proposed approach: Source Selection, Group, and Join.

C. Source Selection

SPARQL enables ASK query to return TRUE/FALSE values depending on whether the data set contains the triples that can match the ones in the ASK query. ASK query is used for source selection in our approach. By wrapping the grouped triple patterns in an ASK query and sending to a certain data set, we can determine whether the data set is able to contribute to the results. Thus which data set to query for which part of the triples is decided. This mechanism avoids the need for having description information for data sets. Moreover a new endpoint of a data set could be added without having to know the statistics of the data in the data set, which allows the approach to be extended to more data sets.

D. Group

On the basis of the assumptions we outlined in Section IV-A, we perform subject and sameAs grouping as part of our query optimisation techniques.

Subject grouping is a mechanism that groups the triples which share the same subject. From Assumption 2, we know that triples that share a subject must be in the same data set. Hence we can group them together to get the results without affecting the number of the query results.

Based on Assumption 3, the sameAs grouping technique combines the link triple which has the predicate of ‘owl:sameAs’ and is not grouped with other triples after subject grouping with the triples which constrain the subject of the link triple. The goal of the sameAs group is to avoid a large amount of data to be transported over the network which could be caused by sending a single sameAs triple, such as *‘?subject owl:sameAs ?object’*, to all the data sets. The sameAs triple already grouped with other triples by subject group is ignored in this step. Mostly the triple with predicate ‘owl:sameAs’ is always the link from one data set to another because it is pointless to define the same data using two different URIs in one data set. By sameAs grouping, one join of ‘owl:sameAs’ triple is merged into another join, so the total number of joins is reduced.

E. Join

Since the results of a federated query are constructed from the intermediate results obtained from different data sets, a join step is required. One way of achieving a join implementation is to send grouped triples one by one, with the subsequent triples being changed according to the intermediate results from the previous join. Bind-join is used to realise this mechanism. Another method of join implementation is to send the grouped triples together, when all the intermediate results have been obtained, they are joined by the column with the same value. The first method is called pre-join as the join plan is done before the query execution whereas the second is called post-join as the join is performed at the end of query execution.

For pre-join, there should be a component that estimates cost of each triple group so that the joins can be ordered for minimum cost of entire query. By selecting first join properly, the number of returned results is minimised for each sub-query. However, as the sub-queries are executed sequentially, the time involved for the entire query increases when the number of joins increases.

For post-join, the sub-queries are executed in parallel, the cost estimation component is not necessary and the number of joins does not affect the entire processing time a lot. However, the performance of the query can deteriorate when there are a huge number of intermediate results for the sub-queries. To deal with this problem, LIMIT and OFFSET parameters of SPARQL are employed. The relevant query is first sent with a set LIMIT parameter to retrieve the first set of results. The query is then repeated by setting the OFFSET parameter to the proper count to get the next set of results from the relevant endpoint.

F. Implementation

We provide an implementation of the query processing model in Java based on Jena [16], an external Java toolkit for RDF and SPARQL. Query parsing is done in Jena automatically when the query string is interpreted into query instance. Then the formatted query is separated into triples. As described in Section IV-D, the subjects of triples are detected and triples which share the same subject are grouped together, this is the subject group. After that, if the triple whose predicate is ‘owl:sameAs’ is not grouped with anyone else in subject group, it will be grouped with triples that contain the subject of the triple. Next, the grouped triples will be wrapped in an ASK query and sent to all data sets to test whether the data set can return results for the triples. Multithreading is applied to send ASK queries in parallel, thus the time used by remote requests is reduced. In the following step, which sends grouped triples to corresponding data sets in order to get intermediate results, multithreading is used for decreasing execution time as well. The intermediate results are stored in table instances. Finally, the table with minimal rows is selected to be the basic table. All other tables are joined with basic table one by one.

V. Evaluation Results

In this section, we perform an evaluation of our implementation introduced above. The evaluation is performed on the FedBench [17] benchmark suite, which is an evaluation platform for federated query processing on Linked Data. It can be implemented on remote endpoints as well as on local repositories. By making a bridge from a federated query application to FedBench, FedBench can evaluate efficiency and effectiveness of the application. The evaluation results will be stored in local file. In this paper, we use the query processing time as a metric to evaluate the implementation. As not all Linked Data sets provide publically available endpoints, we have to setup local repositories using Apache Tomcat [18] and Openrdf Sesame [19]. From evaluation already done in FedX[10] and SPLENDID[11], we find that FedX performs the best among all reviewed approaches. Hence we evaluate against it.

The evaluation was performed on a laptop with 2.50 GHz processor, 4.00GB RAM and 64-bit Window 7 operating system. The queries used are the life science domain queries from the FedBench suite, (available at http://code.google.com/p/fbench/wiki/Queries). The results, as shown in Fig. 2 are average values from five runs of each query.

Fig. 2, LS1 consists of only an UNION operator without any

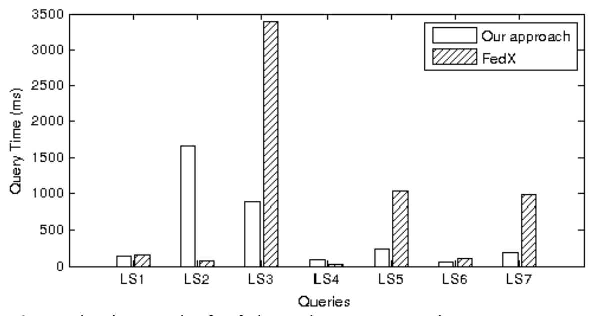


Fig. 2. Evaluation results for federated query processing.

join operation involved. Thus, both approaches give similar performance. LS2 has only one join with one selective triple and one non-selective one. In this case, FedX performs better because it chooses the selective triple as first join, which will return a small number of intermediate results, so that the pre-processing involved in generating the next sub-query is less time consuming. Moreover, the next set of results is restricted by the sub-query. The time used by our approach shows a dramatic increase because of the non-selective triple, which is a kind of generic '?s ?p ?o' triple requesting all data triples from the endpoint. All data sets can potentially contribute to the final results. The triple will be sent to every data set to get results. It takes considerable time getting a large volume of intermediate results from all data sets. LS3, LS5, and LS7 are the queries with two or three joins and over 1000 results for each sub-query. The grouping optimisation techniques we applied reduce the number of join operations by one compared to FedX, resulting in a better performance. LS4 and LS6 are queries with one highly selective triple group, which returns less than 50 results, and one less selective triple group, which would get a large number of results if the query with this triple group is sent to concerned data set. FedX performs better in LS4 as the number of joins is the same after grouping compared to our approach. However, our approach outperforms FedX in LS6 by reducing the number of joins.

To summarise, FedX performs well for queries with one highly selective triple, whose results number less than 50, such as LS2, LS4, and LS6. Our method outperforms FedX only when the number of joins can be reduced more than FedX after grouping. However, our method shows a better performance for the queries without a highly selective triple, such as LS3, LS5, and LS7.

VI. Conclusion

In summary, our main contributions are the optimisation techniques involving subject group and sameAs group which decrease the number of joins and reduce the number of queries sent to remote endpoints. The assumptions underlying our proposed method relate to provenance of data and can take into account distributed or replicated data sets. Our approach would also benefit from techniques being developed in the Information Centric Network (ICN) research area, which allows late binding to location.Furthermore we provide an implementation based on the post-join ordering technique which shows good performance for both small and large number of joins.

From the experimental results, we find that a post-join based implementation is not always the most efficient. We intend to extend our work to a hybrid method to achieve a good performance for all types of federated queries. This would involve sending the post-join query first with a LIMIT parameter. Once the first part of the results is obtained, the cost of each sub-query will be determined in order to plan the join ordering with precise costs for the rest of results. In the last step, the resulting pre-join query will be sent to get the final answers.

Acknowledgment

The authors would like to thank Olaf Görlitz and Andreas Schwarte for their valuable support.

References

[1] T. Berners-Lee, Linked Data – Design Issues, Jul. 2006, retrieved Jun. 2009, available: http://www.w3.org/DesignIssues/LinkedData.html.

[2] RDF Working Group, "RDF - Semantic Web Standards," 2004. [Online]. Available: http:// www.w3.org/RDF/.

[3] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," 15 Jan 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/.

[4] DBpedia, [Online]. Available: http://dbpedia.org/About.

[5] LinkedMDB, [Online]. Available: http://linkedmdb.org/.

[6] G. Ladwig and T. Tran, "Linked data query processing strategies," *The Semantic Web–ISWC 2010*, pp. 453–469, 2010.

[7] L. Haas, D. Kossmann, E. Wimmers, and J. Yang, "Optimizing queries across diverse data sources," 1997.

[8] J. Zemanek, S. Schenk, and V. Svatek, Optimizing SPARQL Queries over Disparate RDF Data Sources through Distributed Semi-Joins. In *ISWC 2008 Poster and Demo Session Proceedings*. CEUR-WS, 2008.

[9] B. Quilitz and U. Leser, "Querying distributed RDF data sources with SPARQL," *The Semantic Web: Research and Applications*, 2008.

[10] A. Schwarte, P. Haase, and K. Hose, "FedX: Optimization techniques for federated query processing on linked data," *The Semantic Web–ICWC 2011,* 2011.

[11] O. Görlitz and S. Staab, "SPLENDID: SPARQL endpoint federation exploiting VOID descriptions," *Proceedings of the 2nd International Workshop on Consuming Linked Data*, 2011.

[12] S. Harris, A. Seaborne and E. Prud'hommeaux, "SPARQL 1.1 Query Language," March 2013. [Online]. Available: http://www.w3.org/TR/sparql11-query/.

[13] K. Alexander and M. Hausenblas, "Describing linked datasets-on the design and usage of void, the 'vocabulary of interlinked datasets,'" *In Linked Data on the Web Workshop*, 2009.

[14] C. Bizer, R. Cyganiak, T. Heath, How to Publish Linked Data on the Web, [Online]. Available: http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/

[15] fbench, "FedBench - A Benchmark Suite for Federated Semantic Data Query Processing," [Online]. Available: http://code.google.com/p/fbench/.

[16] Apache Jena, January 2004. [Online]. Available: http://jena.apache.org/.

[17] M. Schmidt, O. Görlitz, and P. Haase, "FedBench: a benchmark suite for federated semantic data query processing," *The Semantic Web–ISWC 2011*, pp. 585–600, 2011.

[18] Apache Tomcat, [Online]. Available: http://tomcat.apache.org/.

[19] openRDF Sesame, [Online]. Available: http://www.openrdf.org/.