An Evaluation of Approaches to Federated Query Processing over Linked Data

Peter Haase
fluid Operations
Altrottstr. 31
69190 Walldorf, Germany
peter.haase@fluidops.com

Tobias Mathäß
fluid Operations
Altrottstr. 31
69190 Walldorf, Germany
tobias.mathaess
@fluidops.com

Michael Ziller
fluid Operations
Altrottstr. 31
69190 Walldorf, Germany
michael.ziller@fluidops.com

ABSTRACT

The Web has evolved from a global information space of linked documents to a web of linked data. The Web of Data enables answering complex, structured queries that could not be answered by a single data source alone. While the current procedure to work with multiple, distributed linked data sources is to load the desired data into a single RDF store and process queries in a centralized way against the merged data set, such an approach may not always be practically feasible or desired.

In this paper, we analyze alternative approaches to federated query processing over linked data and how different design alternatives affect the performance and practicality of query processing. To this end, we define a benchmark for federated query processing, comprising a selection of data sources in various domains and representative queries. Using the benchmark, we perform experiments with different federation alternatives and provide insights about their advantages and disadvantages.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing, Distributed databases*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation*

General Terms

Design, Measurement, Performance

1. INTRODUCTION

In recent years the Web has evolved from a global information space of linked documents to a web of linked data [3]. The number of data sources and the amount of data published has been exploding, covering diverse domains such as people, companies, publications, popular culture and online communities, the life sciences genes, governmental and statistical data, and many more.

The Web of Data also opens up new possibilities for generic as well as domain-specific applications. One such possibility is the ability to answer complex, structured queries that could not be answered by a single data source alone.

The current procedure to work with multiple, distributed linked data sources is to load the desired data (in the form of RDF dumps) into a single RDF store and process queries in a centralized way against the merged data set. However, such an approach may not always be practically feasible or desired. For example, in some cases a complete dump of the data sources may not be available, instead the data source may only be accessible via a SPARQL endpoint. In the case of frequently changing data sources, the synchronization with the centralized store becomes a problem.

An alternative approach is that of federated query processing, in which a query against a federation of data sources is split into queries that can be answered by the individual data sources and the results are merged by the federator. The federation is transparent to the end user, i.e. the federated data sources can be queried as if they were a single RDF graph. Sometimes this approach is therefore referred to as *virtual integration*. Approaches to federated query processing over linked data are still in their infancy. Some first proposals exist, none of them have been used on a larger scale. We expect that this will change in the next years. What is currently missing is a clear understanding what use cases and requirements to federations are, and how different design alternatives affect the performance and practicality of query processing.

For the problem of query processing against single data sources various benchmarks have been proposed (including LUBM [6], the Berlin SPARQL benchmark [4], and SP^2Bench [12, 13]). Based on these benchmarks, insights about the pros and cons of the various alternatives for storing and querying RDF have been achieved.

We believe that a benchmark for federated query processing will be similarly useful. With this paper, we make an important step towards such a benchmark. To this end, the contributions of the paper are as follows:

- We discuss alternatives to the integration of and the query processing over multiple data sources.
- We define a benchmark for federated query processing, comprising a selection of data sources in various domains and representative queries.

I-SEMANTICS 2010 September 1-3, 2010 Graz, Austria

- Using the benchmark, we perform experiments with different federation alternatives and provide insights about their advantages and disadvantages.

In the end, we want to enable and assist application developers in choosing the right architecture and the right storage system for their requirements and constraints.

The remainder of the paper is structured as follows: In Section 2, we present different alternatives to query processing over multiple data sources. In the subsequent section, we introduce our benchmark: data sets and queries in Section 3, performance measures and experimental results in Section 4. We discuss related work in Section 5. We conclude and discuss future work in Section 6.

2. APPROACHES TO FEDERATION

In this section, we discuss different approaches to federation and integration of linked data sources. Generally, in data integration and query answering over distributed sources we can distinguish two broad classes of approaches:

- *Warehousing*, where all data is collected in advance, preprocessed and stored in a central database, and queries are evaluated against this central database, and
- *Distributed Query Processing*, where queries are evaluated against the distributed sources by splitting the query in appropriate subqueries and combining the results from the remote sources.

In practice, there are of course many variants and hybrids of these classes of approaches. We will focus on three variants relevant in the context of dealing with distributed sources on the Web of Data. The selection of presented variants is not intended to be exhaustive in all dimensions, but rather to illustrate fundamentally different design alternatives.

As we will see later, all of these approaches have their advantages and disadvantages; the decision for one or the other will have to depend on the requirements and constraints of the specific application. To illustrate this, consider the following two use cases for integrating a set of data sources.

1. *Ad hoc data analysis:* Here, the end user wants to quickly explore a set of data sources, build federations on the fly, and potentially even select a subset of available data sources on a per-query-basis.
2. *Linked data portals:* Here, a carefully selected set of data sources is preprocessed and aggregated centrally in a portal. A primary goal is to enable efficient and reliable access to the data for a large user base.

These examples illustrate a range of different requirements on the side of the data consumer:

- *Selection of data sources*: How large is the set of data sources? Does the set change over time?
- *Lifetime of the federation*: How long is the federation intended to exist? Is it short-lived, or is it permanent?
- *Characteristics of the queries*: What types are to be processed, what is the frequency of the queries?
- *Updates*: Are updates to the data required?
- *Processing capabilities*: Does the consumer (client) have the computational and storage resources for query processing? Or should the processing be pushed to the data providers?

Similarly, we can identify differences on the side of data source providers that act as constraints to the client:

- *Dynamics*: How frequently do the data sources change?
- *Access*: How are the data sources made accessible? As RDF dump, as dereferencable HTTP URIs, or as a SPARQL endpoint?
- *Interfaces*: What kind of interfaces are exposed?
- *Service Levels*: What kind of guarantees or restrictions are made with respect to performance, response times, etc.?

In the following, we discuss three possible realizations of federating multiple distributed data sources (c.f. Figure 1).

Integration in a central repository.

In this scenario (Figure 1a), all data is physically loaded from dumps of the data sources into a central repository. Queries are evaluated against this single repository. In this sense, we do not have a physical federation, however, we can consider this approach as a logical federation when managing the data as named graphs, i.e. maintaining information about the data source as context information.

Clearly, a disadvantage of this approach is that the complete data needs to be downloaded in the form of an RDF dump and loaded into the repository upfront. From the perspective of a single application or user, way too much unnecessary data is collected, as for a given query typically only a small subset of the data would be needed. The load times can be significant for large data sources, such that this approach is impractical for ad-hoc federation. On the contrary, this approach typically allows the fastest query time since 1) no network communication is needed and 2) queries can be highly optimized as complete knowledge about all data sources is available centrally.

Further, updates to the data are easily manageable, although the updates only affect the centralized copy and are not propagated back to the original sources.

Federation over multiple single repositories.

In this scenario (Figure 1b), the data from the individual data sources is managed in separate repositories, one for each data source. As in the prior scenario, the single repositories can be populated from RDF dumps. This loading can happen either on the side of the federator (consumer), or on the side of the data provider, who may provide the repositories in addition or as an alternative to RDF dumps. The single repositories are federated in a federation layer, which splits a query into sub-queries against the single repositories and merges the results. The federator directly uses the native API of the single repositories. Such a federation layer is e.g. available in the Sesame framework. To the end user, the federation is completely transparent, i.e. queries can be asked in the same way as with a single repository.

As in the previous scenario, there is a cost in loading the data from the dumps. However, the loading can be done beforehand, such that an ad hoc federation can be built

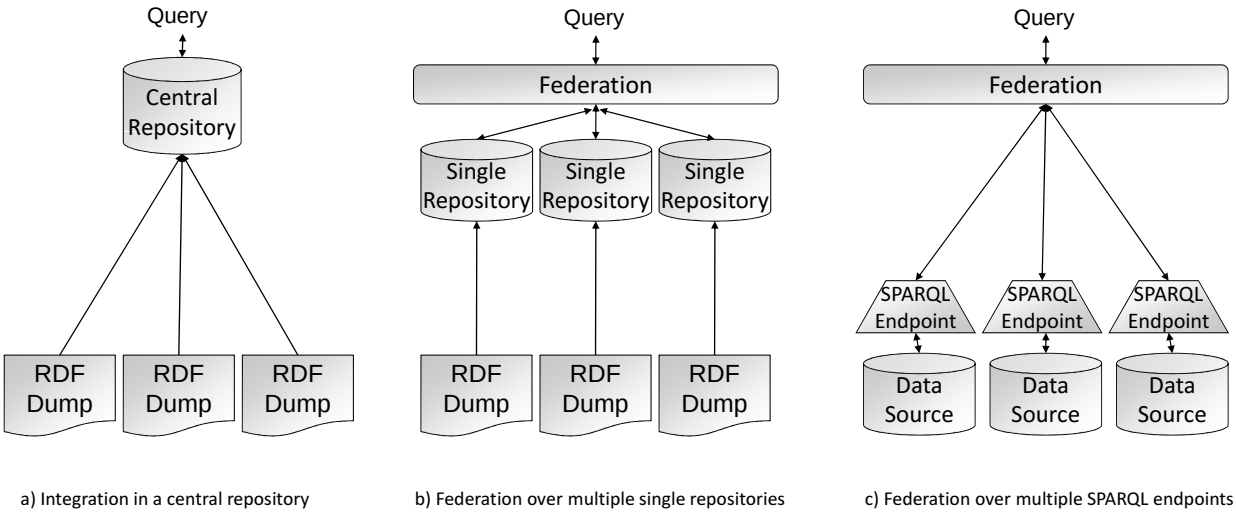

Figure 1: Approaches to federation

by simply adding an additional member repository to the federation. Certainly, for query processing there is a performance penalty incurred by the additional federation layer. However, as the federation layer has direct access to the repository along with statistics needed for query optimization, the performance overhead may be low.

Updates in this federated scenario are difficult. Typically, such configurations are restricted to read-only access, or write-access limited to one member of the federation.

Federation over multiple SPARQL endpoints.

As in the previous scenario, in this scenario (Figure 1c) multiple repositories are accessed via a federation layer. However, the access is realized via SPARQL endpoints of the data providers. Loading of the data is thus not needed, an ad hoc federation can be built by simply adding an additional SPARQL endpoint to the federation. On the other hand, query processing becomes more difficult: The access via a SPARQL endpoint is much more restricted than with a native interface, essential statistical information needed for query optimization is typically not accessible. Further, the access is restricted to read-only (although a proposal for SPARQL-Update exists).

3. BENCHMARK DATA SETS AND QUERIES

In this section we present the data sets and queries defined for our benchmark. All data sets and queries are available at `http://code.google.com/p/fbench/`. A primary goal of this benchmark is to rely on real life data. We therefore decided to use subsets of the actual Linked Open Data cloud[1], instead of using synthetic data sets.

The data sources published in the Linking Open Data initiative vary greatly in their coverage, domain-specificity, depth, etc. Some data sources (such as DBpedia) are cross domain and very broad in coverage. Other data sources, in particular scientific data, are very domain specific. We therefore decided to select as data sets two subsets of data sources in the Linked Data cloud: one data set that spans many different domains of general interest, and one domain specific data set in the Life Sciences domain.

For these two data sets, we targeted to have a good coverage of data sources of a) different size, b) different domain coverage, c) different complexity of the schemas. Further, we selected only data sources that are accessible both in the form of an RDF dump as well as via a SPARQL endpoint.

3.1 Data Set 1: Cross-domain

As data sources for the domain-independent part of the benchmark, we chose a set of data sources spanning various domains of popular interest, including news, movies, music, geography. In more detail, the used data sources are a subset of DBpedia containing the infoboxes and the instance types, GeoNames containing geographical data about persons, locations, organizations etc., LinkedMDB, a movie and actor database, the music database Jamendo and a data repository containing news from the New York Times. Figure 2 a) shows how these data sources are interrelated. Most of the links between data sources are `owl:sameAs` links.

3.2 Data Set 2: Life Sciences

For the domain-specific part of the benchmark, we chose the life sciences domain, integrating various data sources dealing with chemical compounds, reactions and drugs. In detail, the used data sources are Drugbank, a database containing information about drugs, their composition and their interactions with other drugs, a subset of the Kyoto Encyclopedia of Genes and Genomes (KEGG) which contains further information about chemical compounds and reactions with a focus on information relevant for geneticists, and the Chemical Entities of Biological Interest (ChEBI) database which describes the life sciences domain from a more chemical point of view. Additionally, we added the DBpedia subset from the cross-domain data set, which also

[1] `http://linkeddata.org/`

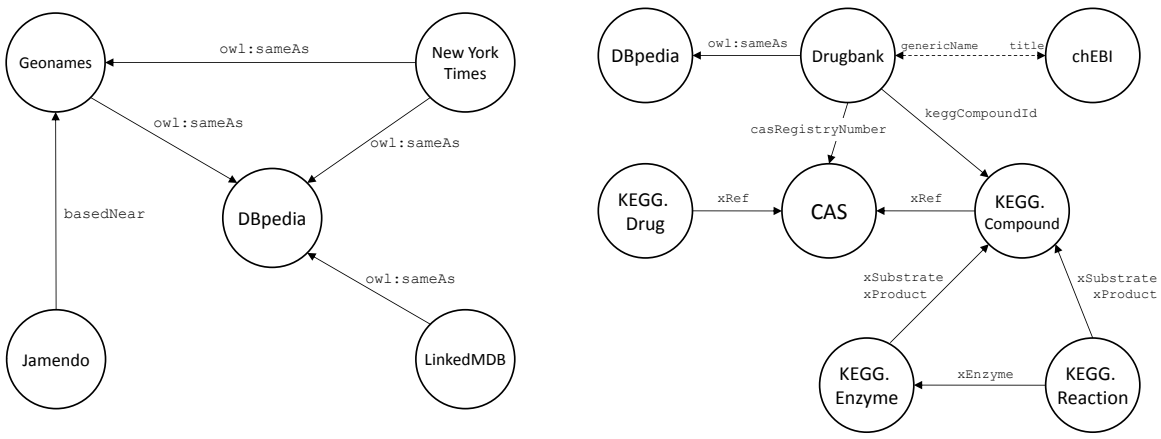

Figure 2: Data Sets: a) Cross-domain, b) Life Sciences Domain

contains data from the life sciences and, additionally, acts as a nucleus and central hub for links across domains. Figure 2 b) shows the links between these data sources. Since KEGG is published in a number of separate modules, we considered these modules as autonomous data sources. The dotted arrow between Drugbank and ChEBI symbolizes that there are no direct links in the RDF graph, but that a join is possible via common values of data properties. The Chemical Abstracts Service (CAS) data source acts as an element that enables a join between KEGG and Drugbank, as both refer to entities in the CAS data source.

Table 1: Benchmark Datasets

Data set	**coverage**	**triples**
Cross domain		**91,402k**
DBpedia (subset)	generic	14,000k
NYTimes News	news	206k
GeoNames	geography	70,000k
Jamendo	music	1,048k
LinkedMDB	movies	6,148k
Life sciences domain		**20,379k**
DBpedia (subset)	generic	14,000k
KEGG Compounds	compounds	178k
KEGG Enzymes	enzymes	685k
KEGG Drugs	drugs	117k
KEGG Reactions	reactions	110k
Drugbank	drugs and structures	517k
ChEBI	chemical entities	4,772k

Table 1 shows statistics of the data sets, including their size in number of triples.

3.3 Benchmark Queries

For both data sets, we defined a set of queries. In principle, there are two options for the design of benchmark query mixes [5]: a) Design the queries to test specific features of the query language, or b) base the queries on the specific requirements of a real world use case. We follow a combination of the above. Just as with the data sets, queries should be realistic and represent real life use cases. At the same time, we designed the queries to vary in important characteristics. However, we did not aim at completeness with respect to all features of the SPARQL query language (which is addressed by other benchmarks), but focus on aspects that are relevant in the context of query processing over multiple data sources, in particular: 1) the number of data sources involved, 2) the complexity of the joins, 3) the types of links used to join the data sources, and 4) the query result size.

In the following, we list the queries for each data set, and explain important characteristics. Queries Q1.1 – Q1.7 are queries for the cross-domain data sets, queries Q2.1 – Q2.7 are for the life sciences domain.

Q 1.1: Find all information about Barack Obama.

```
SELECT ?predicate ?object WHERE {
   { dbpedia:Barack_Obama ?predicate ?object }
  UNION
   { ?subject owl:sameAs dbpedia:Barack_Obama .
     ?subject ?predicate ?object } }
```

This query finds all outgoing properties of a specific entity (in `DBpedia`) and those of all entities linked via `owl:sameAs` from all datasources available.
It returns the union of all properties of any of these entities. In our data set, `DBpedia` and `NYTimes` will contribute to the answer.

Q 1.2: Return Barack Obama's party membership and news pages

```
SELECT ?party ?page WHERE {
   dbpedia:Barack_Obama dbpedia-owl:Person/party ?party .
   ?x nytimes:topicPage ?page .
   ?x owl:sameAs dbpedia:Barack_Obama }
```

This simple query is similar to Q1.1, but is more selective as it specifically asks for specific properties in the `NYTimes` and `DBpedia` data sources. It requires a join via an `owl:sameAs` property.

Q 1.3: Return for all US presidents their party membership and news pages about them

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

This query is very similar to Q 1.2, but less selective, since the person asked for is variable and not bound to the entity Barack Obama.

Q 1.4: Find all news about actors starring in a movie with name Tarzan

```
SELECT ?actor ?news WHERE {
   ?film purl:title 'Tarzan' .
   ?film linkedMDB:actor ?actor .
   ?actor owl:sameAs ?x.
   ?y owl:sameAs ?x .
   ?y nytimes:topicPage ?news }
```

This query performs a two-way join via `owl:sameAs` links between `LinkedMDB` and the `NYTimes` data source. Both data sources do not link to each other, but instead both provide `owl:sameAs` links to `DBpedia`.

Q 1.5: Find the director and the genre of movies directed by Italians

```
SELECT ?film ?director ?genre WHERE {
   ?film dbpedia-owl:Film/director ?director.
   ?director dbpedia-owl:Person/nationality dbpedia:Italy .
   ?x owl:sameAs ?film .
   ?x linkedMDB:genre ?genre }
```

This query performs a join via an `owl:sameAs` between `LinkedMDB` and`DBpedia`.

Q 1.6: Find all musical artists based in Germany

```
SELECT ?name ?location ?news WHERE {
   ?artist foaf:name ?name .
   ?artist foaf:based_near ?location .
   ?location geonames:parentFeature ?germany .
   ?germany geonames:name 'Federal Republic of Germany' }
```

This query performs a join via the property `geonames:basedNear` which links entities in `Jamendo` with `GeoNames`.

Q 1.7: Find all news about locations in the state of California

```
SELECT ?location ?news WHERE {
   ?location geonames:parentFeature ?parent .
   ?parent geonames:name 'California'  .
   ?y owl:sameAs ?location.
   ?y nytimes:topicPage ?news }
```

This query performs a join via a direct `owl:sameAs` link from `GeoNames` and `NY Times`.

Q 2.1: Find all drugs from `Drugbank` and `DBpedia` with their melting points

```
SELECT ?drug ?melt WHERE {
   { ?drug drugbank:meltingPoint ?melt . }
  UNION
   { ?drug dbpedia:meltingPoint ?melt . } }
```

This query is a simple union of two partial results which can both be evaluated on single datasources, in this case `Drugbank` and `DBpedia` without a join.

Q 2.2: Find all properties of Caffeine in Drugbank. Find all entities from all available databases describing Caffeine, return the union of all properties of any of these entities.

```
SELECT ?predicate ?object WHERE {
   { drugbank:DB00201 ?predicate ?object . }
  UNION
   { drugbank:DB00201 owl:sameAs ?caff .
     ?caff ?predicate ?object . } }
```

This query returns all outgoing properties of a specific entity (drugbank:DB00201, denoting Caffeine) and those of all entities linked via `owl:sameAs` from all data sources available.

Q 2.3: For all drugs in DBpedia, find all drugs they interact with, along with an explanation of the interaction

```
SELECT ?Drug ?IntDrug ?IntEffect WHERE {
   ?Drug rdf:type dbpedia-owl:Drug .
   ?y owl:sameAs ?Drug .
   ?Int drugbank:interactionDrug1 ?y .
   ?Int drugbank:interactionDrug2 ?IntDrug .
   ?Int drugbank:text ?IntEffect . }
```

This query performs a join via `owl:sameAs` between `DBpedia` and the `Drugbank` data source.

Q 2.4: Find all the equations of reactions related to drugs from category Cathartics and their drug description

```
SELECT ?drugDesc ?cpd ?equation WHERE {
   ?drug drugbank:drugCategory drugbank:cathartics .
   ?drug drugbank:keggCompoundId ?cpd .
   ?drug drugbank:description ?drugDesc .
   ?enzyme kegg:xSubstrate ?cpd .
   ?enzyme rdf:type kegg:Enzyme .
   ?reaction kegg:xEnzyme ?enzyme .
   ?reaction kegg:equation ?equation . }
```

This query performs a join via a common entity in relation to a drug from `Drugbank` and to an enzyme from `KEGG Enzymes`.

Q 2.5: Find all drugs from Drugbank, together with the URL of the corresponding page stored in KEGG and the URL to the image derived from ChEBI

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

Since `drugbank:genericName` and `purl:title` are attributes, this query performs a join via triples from `Drugbank` and `ChEBI` sharing a literal as their object. The join between `Drugbank` and `KEGG Drugs` is performed via the `owl:sameAs` property.

Q 2.6: Find KEGG drug names of all drugs in Drugbank belonging to category Micronutrient.

```
SELECT ?drug ?title WHERE {
   ?drug drugbank:drugCategory drugbank-owl:micronutrient> .
   ?drug drugbank:casRegistryNumber ?id .
   ?keggDrug rdf:type kegg:Drug .
   ?keggDrug kegg:xRef ?id .
   ?keggDrug purl:title ?title . }
```

The join via `drugbank:cas-number` is expensive, since most drugs in `KEGG Drugs` have many outgoing properties `kegg:xref` that have to be checked for equality with the one passed from `Drugbank`

Q 2.7: Find all drugs that affect humans and mammals. For those having a description of their biotransformation, also return this description. Show only those whose mass starts with a number larger then 5

```
SELECT ?drug ?transform ?mass WHERE {
   ?drug drugbank:affectedOrganism  'Humans and other mammals'.
   ?drug drugbank:casRegistryNumber ?cas .
   ?keggDrug kegg:xRef ?cas .
   ?keggDrug kegg:mass ?mass
      FILTER ( ?mass > '5' )
      OPTIONAL { ?drug drugbank:biotransformation ?transform . } }
```

Additional to the join via the `drugbank:casRegistryNumber`, this query contains a `FILTER` and an `OPTIONAL` variable.

4. BENCHMARK EXPERIMENTS

In this section we report on the benchmark experiments performed. We first present the performance measures, describe the benchmark setup, and then present and discuss the experimental results.

4.1 Performance Measures

We use mainly two metrics – load time and query time – to capture the tradeoffs of the different setups. These metrics should of course always be considered together with the qualitative analysis of the advantages and disadvantages, as discussed in Section 2.

Load time.

We measure the time needed to load the RDF dumps of the data sources into the repositories. This measure is only applicable for setups a) and b) and does not apply to a federation of remote SPARQL endpoints.

It should be noted that in addition to the time to *load* an RDF dump into the repository, also the time to *download* the dumps is of relevance. As this time will mainly depend on the available network connection and will hardly be reproducible, we do not report this time explicitly in the results. Instead, the size of these data sets can be considered as an indicator.

Query time.

For each query, we measure the time for its evaluation in a particular setup. The time refers to time for the evaluation including an iteration over the entire result set. This iteration has to be included as results may be delivered in a streaming manner.

4.2 Benchmark Setup

We compare three different setup configurations, each implementing one of the approaches to federation introduced in Section 2. The goal here was not the evaluation of particular systems, instead the focus was on the comparison of different architectural design alternatives. We therefore compared solutions within the same framework: the OpenRDF Sesame framework. Sesame provides a flexible and extensible architecture that allows to use different implementations of its Repository API. All three setup are realized with off-the-shelf implementations and work with the data sources as they are provided.

For the benchmark, we used Sesame in version 2.3.1. The individual setups have been realized as follows:

- **Setup a)** *Integration in a central repository:* Sesame offers different repository implementations. We used the Sesame Native store with its default configuration.
- **Setup b)** *Federation over multiple single repositories:* We again used the Sesame Native store as implementation for the single repositories. The federation layer has been realized using the Federation SAIL provided by AliBaba[2] as an extension of Sesame 2.3.1. We used AliBaba in version 2.0-alpha3. The single repositories as well as the Federation SAIL run on the same node.
- **Setup c)** *Federation over multiple SPARQL endpoints:* We again used the Federation SAIL of AliBaba, in combination with the SPARQL repository implementation, which allows transparent read access to repositories providing a SPARQL endpoint. As SPARQL endpoints, we used the endpoints as provided with the data sources.

The experiments have been performed on a server with two Intel Xeon CPU 5160 processors (each with two cores at 3 GHz), 20 GB main memory, running Windows Server 2003 Enterprise 64 Bit, Service Pack 2. We used the 64 Bit Java Runtime Environment 6.

All query mixes in each setup have been run 10 times. To exclude the influence of cold start effect, for each setup we ran a "ramp-up query" mix upfront (in a similar way as the Berlin SPARQL benchmark).

[2]Details about the federation approach and optimizations in AliBaba can be found at `http://wiki.aduna-software.org/confluence/display/SESDOC/Federation`.

Table 2: Load times (in hh:mm:ss)

Data source	Load time		
	Setup a)	Setup b)	Setup c)
Cross domain			
DBpedia	39:28:14	42:17	n/a
Geonames		5:56:20	
LinkedMDB		22:11	
Jamendo		1:57	
NY Times		0:34	
Life sciences			
DBpedia	59:34	42:17	n/a
KEGG Compounds		0:30	
KEGG Enzymes		1:43	
KEGG Drugs		0:26	
KEGG Reactions		0:21	
Drugbank		2:01	
ChEBI		13:09	

Table 3: Query times (in milliseconds)

Query	Query time (in ms)		
	Setup a)	Setup b)	Setup c)
Cross domain			
Q 1.1	4	< 1	5,973
Q 1.2	2	< 1	3,998
Q 1.3	110	370	timeout
Q 1.4	9	31,860	13,596
Q 1.5	19	15,839	timeout
Q 1.6	100	2,613	timeout
Q 1.7	4,751	17,222	timeout
Life sciences			
Q 2.1	49	41	1,828
Q 2.2	12	9	3,772
Q 2.3	1,409	1,348	timeout
Q 2.4	1	1,440	12,235
Q 2.5	225	1,351	timeout
Q 2.6	23	9,872	timeout
Q 2.7	307	16,395	timeout

4.3 Results and Discussion

Tables 2 and 3 show the results for our two metrics load time and query time. The results for the load times do not come as a surprise, as they have been reported in similar forms in other benchmarks [4]. It should however be noted that due to the non-linear dependence of the load time from the number of triples, loading the individual data sources into multiple separate repositories may be significantly faster than loading the entire data set into a single central repository. For example, the cross-domain data set with 91 million triples seems to hit a natural limit of the Sesame native store: It took more than 39 hours to load into a single repository, as opposed to roughly seven hours for the sum of the individual data sources. This is in line with observations from the Berlin SPARQL Benchmark, which reported more than 3 days for loading 100 million triples into a Sesame native store, as opposed to 3 minutes for 1 million triples (i.e. an increase by a factor of $\tilde{1}500$ in load time vs. a factor 100 in number of triples.). It should also be noted that loading the data sets is not applicable for setup c) running against existing SPARQL endpoints.

Table 3 shows the values for the average query times over 10 runs (after ramp-up) for a given query and setup. The timeout limit was set to one minute per query.

We can make some interesting observations. Expectedly, on average the setup a) shows the best query performance, as all data is local and the query optimizer has complete knowledge over the centralized data sets. For the federation over multiple repositories in setup b), we observe – in most cases – significantly larger query times. However, in certain cases a performance penalty does not occur, in some case the federation in setup b) actually performs better than the centralized processing. This is the case for very simple queries (the first queries for each data set, i.e. Q1.1, Q1.2, Q2.1, Q2.2, Q2.3). The federation here actually provides an effective means for parallelization: The simple, very selective triple patterns in these queries can be evaluated in parallel by the members of the federation, result merging requires only a simple union of the individual results. The situation is different for queries involving more complex joins. Depending on the complexity of the join, the performance penalty can be between one and three orders of magnitude. The most important factors here are the join order optimization and the push-down of selective predicates to the federation members. In principle, this requires the exploitation of statistics about the data sources, which in the current implementation of the Federation SAIL are only available to a limited extent. As a consequence, in some cases the entire extensions of certain predicates need to be retrieved from the members of the federation, and expensive joins need to be performed within the Federation SAIL.

The situation with the federation over SPARQL endpoints (setup c) is in principle similar, but in addition we incur the performance penalty by the network communication. As a result, even for the simple, very selective queries we observe response times of >1s. Less selective predicates require the retrieval of entire extensions of certain predicates (e.g. asking for all `owl:sameAs` statements) and a join afterwards. In most cases, this results in a timeout. An interesting case is query Q1.4, which is an exception as it performs better in setup c) than in setup b). Q1.4 involves a complex two-way join (via two `owl:sameAs` statements), and as a result requires extensive reads (much of it from disk) by the federation members. In the case of setup 2), all federation members share a disk and thus an I/O bottleneck, whereas in the case of setup c), much of the processing is pushed to the SPARQL endpoints, without a central bottleneck.

Another interesting observation is that the query times for the cross-domain data set and the queries for the life sciences data set show very similar behavior across all setups for similarly structured queries.

5. RELATED WORK

We discuss two dimensions of related work: approaches to federated query processing and benchmarks for query processing over RDF data.

Federated query processing.

Federated query processing over heterogeneous databases has been a long studied topic in database research [8, 14]. The problem of query processing over distributed RDF data sources has recently been addressed in a number of works.

[15] presented a first approach for querying distributed

RDF data sources, introducing index structures, a cost model and algorithm for query answering. The approach was implemented as a Mediator SAIL within the Sesame Framework, which can be seen as a predecessor of the Federation SAIL considered in our work.

DARQ [10] is an engine for federated SPARQL queries that provides transparent query access to multiple SPARQL endpoints. To speed-up query processing, DARQ uses cost-based query optimization based on statistical information provided in the form of service descriptions by the SPARQL endpoints. While there are proposals to standardize such descriptions [1], they are still not provided by currently available SPARQL endpoints.

Networked Graphs [11] follow a view-based approach to integration. They provide a formalism allowing users to define RDF graphs by using views on other graphs. The relationships between graphs are described declaratively using SPARQL queries and an extension of the SPARQL semantics. Networked Graphs can be evaluated in a distributed setting, yet, they also require extensions to existing languages that are not commonly supported.

As both DARQ and Networked Graphs require proprietary extensions to existing languages and protocols, we did not consider them in our selection of benchmarked systems. Instead we initially focused on architectural alternatives, evaluated within the same software framework.

With respect to query optimization and statistics for data sources, [9] introduces a cost model for querying distributed RDF. With a similar goal, [7] presents an approximate index structure summarising graph-structured content of data sources and provides an algorithm for query answering over such indexed linked data. We believe that providing such statistical information and data summaries will be essential to enable query optimization and make federated query processing practically possible. Our benchmark may serve as a basis for comparing the different models.

Benchmarks.

Several benchmarks have been proposed to evaluate various aspects of RDF query processing over *single* data sources. These include LUBM [6], the Berlin SPARQL benchmark [4], and SP^2Bench [12, 13]. For the problem of query processing over multiple distributed data sources in the context of linked data, no benchmarks have been proposed yet. To the best of our knowledge, our work is the first step in this direction. Clearly, our benchmark definition takes the best practices and experiences from existing benchmarks into account, but focuses on the aspects relevant for dealing with distributed RDF sources.

6. CONCLUSIONS AND FUTURE WORK

The ability to integrate and ask complex queries over the vast amount of linked data published on the Web bears unprecedented opportunities and challenges at the same time. Given the current state-of-affairs, it does not come as a surprise that existing approaches to integration (e.g. in linked open data portals) follow a warehousing approach with a centralized database. Centralized RDF stores have been highly optimized and allow sub-second response times even for complex queries over data sets with >1 billion triples from multiple sources (c.f. the Linked Data Semantic Repository `http://ldsr.ontotext.com/`).

However, we have argued that this warehousing approach is not always feasible or desired, and that for certain requirements and use cases other approaches to federation may be better suited. Clearly, in the context of Linked Open Data, federation is far from trivial. Unlike in an enterprise scenario, the nodes exhibit a very extreme case of autonomy - everybody is free to publish data as he pleases, there are no performance/reliability guarantees.

In this paper, we have analyzed various design choices for federating distributed data sources and discussed their advantages and disadvantages qualitatively. To be able to quantitatively analyze different approaches to federation, we have defined a benchmark, comprising a selection of real life data sources as subsets from the Linked Open Data cloud, along with queries, and evaluation measures.

Based on this benchmark, we have compared and analyzed three implementations of different approaches to federation. Not surprisingly, the centralized approach to query processing shows the best performance. The performance penalty incurred by a federation layer can be several orders of magnitude for complex queries. Federated query processing over SPARQL endpoints is only practical for rather simple queries.

On a more detailed level, we can gain insights about the influence of various effects, including the ability to parallelize, to push operations to remote nodes, the availability of statistics for query optimization, the existence of central bottlenecks, etc. E.g., managing the individual data sources as members in a federation provides a natural way to partition the data, and at least for simple queries the advantages of being able to parallelize certain operations outweigh the disadvantages of the more complex join processing.

It is also clear that a practical federation approach – aimed at supporting complex structured queries that involve joins over multiple data sources – requires knowledge about the remote sources in the form of appropriate statistics and index. Possible approaches include schema-level indexes and data summaries (See the index structures proposed by [7]). Given that the techniques for federated processing in the context of linked open data are still in their infancy on the one hand, and the abundance of work in the context of federated databases on the other hand, we believe that there is great potential for improvements.

In summary, we would like to point out that there is not one best solution for integrating or federating data. For specific requirements, one will need to decide which criteria and constraints are most important, and which solution is best suited. We hope that the results of this analysis will help developers in making such decisions.

Beside the ability to evaluate specific approaches and systems, our benchmark may also provide insights for principles how to publish linked data on the web such that query processing over multiple distributed sources becomes feasible. One important question is how data sources should be interlinked. Using `owl:sameAs` is an expensive mechanism especially in a federated setting. If a central hub like DBpedia is used to establish links between two data sources, even a two-way join is introduced. Reusing URIs appears much more effective from the perspective of query processing.

Future Work.

Clearly, our experimental analysis is not (and does not intend to be) complete with respect to covering all existing solutions enabling federation and integration of RDF

data. To this end, we invite others to evaluate their systems against this benchmark and to provide feedback for improvements. The entire benchmark, including data sets, queries, and the evaluation framework itself is publicly available at `http://code.google.com/p/fbench/`. Ideally, the benchmark will be extended and maintained as a community effort.

Finally, an interesting direction for future work would also be to extend the benchmark towards reasoning, i.e. to consider entailment regimes beyond basic graph matching.

Acknowledgments

Research reported in this paper was partially supported by the German BMBF in the project CollabCloud.
`http://collabcloud.de/`

7. REFERENCES

[1] Sparql 1.1 service description, w3c editor's draft 26 january 2010.

[2] *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in Computer Science*. Springer, 2008.

[3] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[4] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[5] Jim Gray. Database and transaction processing performance handbook. In Jim Gray, editor, *The Benchmark Handbook*. Morgan Kaufmann, 1993.

[6] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[7] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *WWW*, pages 411–420. ACM, 2010.

[8] Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[9] Lyndon Nixon Philipp Obermeier. A cost model for querying distributed rdf-repositories with sparql. In *Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense Tenerife, Spain, June 2, 2008.*, 2008.

[10] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *ESWC* [2], pages 524–538.

[11] Simon Schenk and Steffen Staab. Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In *WWW*, pages 585–594. ACM, 2008.

[12] Michael Schmidt, Thomas Hornung, Norbert Küchlin, Georg Lausen, and Christoph Pinkel. An experimental comparison of rdf data management approaches in a sparql benchmark scenario. In *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2008.

[13] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. Sp2bench: A sparql performance benchmark. In *ICDE*, pages 222–233. IEEE, 2009.

[14] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[15] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben, and Jeen Broekstra. Index structures and algorithms for querying distributed rdf repositories. In *WWW*, pages 631–639. ACM, 2004.