RDFINT: A Benchmark for Comparing Data Warehouse with Virtual Integration Approaches for Integration of RDF Data

Samson Oni*, Kajal Pansare*, Sukrit Singh Arneja*, Zhiyuan Chen*, Adina Crainiceanu†, and Don Needham†
* *University of Maryland Baltimore County, Baltimore, MD*
Email: {soni5, kajalpansare, sarneja1, zhchen}@umbc.edu
†*US Naval Academy, Annapolis, MD*
Email: {adina, needham}@usna.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378131

***Abstract*—Users often need to integrate large amounts of RDF data from multiple sources. Although there has been a lot of work on data cleaning and integration for structured data, relatively little work has been done for RDF data. We consider two different approaches to data integration: 1) a traditional data warehouse approach where relevant RDF data is extracted from different sources and then integrated in a data warehouse; 2) a virtual integration approach where RDF data still resides at each source and data integration happens when the data is queried, through a mediator that coordinates with wrappers at each source. It is often unclear how to choose the appropriate approach given an application scenario. This paper proposes RDFINT, a benchmark to compare these two approaches for integrating RDF data. We describe typical data integration operations, metrics that can be used to compare these two approaches, and factors that affect these metrics. We also report preliminary results of an implementation of these two approaches using the Apache Jena Fuseki framework.**

I. Introduction

The Resource Description Framework (RDF) [1] provides a set of specifications for describing resources and relationships between them. Semantic queries for RDF data can be written in the SPARQL language [2]. Billions of RDF triples are now available on the internet. In practice, RDF data is often distributed at many different sources so the data needs to be integrated and cleaned before it can be used.

There are two popular approaches to integrate RDF data. The first is the traditional data warehouse approach where relevant RDF data is extracted from the original sources, transformed, cleaned, and then integrated at one place. The second is a virtual integration approach where RDF data remains in the original sources and data integration happens at the time the data is being queried. In this approach, a mediator coordinates with wrappers at each source to map the query to local sources, combine results, and perform further data cleaning operations over the results if needed (e.g., to remove duplicates). In the data warehouse approach, more work is required up-front, when data is integrated into the warehouse, but then queries can be processed faster as all the data is stored in one place. On the other hand, using a virtual integration approach, queries might be slower, but have a more up-to-date answer, since data is queried directly from the original sources. It is often unclear how to choose the appropriate approach given an application scenario.

In this paper we propose RDFINT (RDF Integration Benchmark), a benchmark to be used in comparing a data warehouse approach with a virtual integration approach for integrating RDF data. Given a scenario where multiple sources produce data and queries are issued over the union of all of the data, we compare a data warehouse approach where all data is transformed and stored in one centralized database, versus a virtual integration approach where data is stored at the local sources and data integration is implemented by rewriting queries and/or post-processing of query results. This paper describes the following contributions:

- We propose measures of performance relevant to comparing the data warehouse approach and the virtual integration approach.
- We determine some of the factors that affect the performance of a data warehouse approach and the virtual integration approach.
- We provide a prototype implementation for a benchmark to compare the data warehouse and virtual integration approach. The benchmark includes a data generator and a set of data integration operations and their sample implementations in both approaches.
- Finally, we provide preliminary experimental evaluation comparing the warehouse approach with the virtual data integration approach using Apache Jena Fuseki Framework [3].

II. Metrics, Factors, and Data Integration Operators Used in The Benchmark

We propose the following metrics to quantify the differences between different RDF data integration approaches:

(M1) Time for ETL operations including time to load historical data (the initial load), time to load new data (incremental load), and time to address ontology mismatch and various data quality issues.

(M2) Scalability of the system with respect to the number of triples and the number of sources.

(M3) Completeness of the query results when a certain fraction of source data gets updated. Completeness

means any correct result is being returned.

(M4) Usability, including ease of implementation, required skill levels, etc.

We also varied the following factors that affect the performance of the data warehouse approach and the virtual integration approach.

(F1) Size of the data sets
(F2) The number of data sources
(F3) Fraction of the data being updated
(F4) Characteristics of the data sets (e.g., fraction of duplicates)
(F5) Network delay, which has an impact on the performance of the virtual integration approach.

We consider the following data integration and data quality issues that have to be solved for data integration in either approach:

(I1) Existence of exact duplicates of some RDF triples between multiple data sources
(I2) Existence of approximate duplicates between data sources where different resource IDs or IRIs refer to the same conceptual or real-life resource
(I3) Missing values for the object or subject of some triples
(I4) Ontology mismatch, for example when the same concept is expressed by different predicates in different data sources.

III. Data Generator

We modified the data generator used in Lehigh University Benchmark (LUBM) [4] to fit the purposes of this benchmark. This provides several benefits: 1) LUBM is a well known benchmark for RDF storage and query processing so many users are already familiar with it; 2) the LUBM data model is relatively easy to understand; 3) LUBM provides a set of test queries with sufficient variety so we can reuse some of its test queries as well.

The LUBM benchmark generates data of different universities, where each university contains a number of departments, professors, students, and courses. Since data integration requires data from multiple sources, we consider each university's data as a data source.

Data integration also needs to address a number of data quality issues. We modified the data generator to simulate the data quality issues (I1)-(I4) introduced in Section II:

- Exact duplicates: a certain fraction of randomly selected triples are duplicated in the data.
- Approximate duplicates: we generate a certain fraction of professors at different universities who have different IRIs, but they as likely the same person as they have the same name, and graduated from the same institutions, both for their graduate and undergraduate degrees. In real life, such cases may happen when a professor moves from one university to another.
- Missing values: emails and phone numbers of some professors are randomly selected and replaced with blank nodes.
- Ontology mismatch: we generate a few ontology mismatches, including in one university the predicate used to represent a student takes a class is called "takesClass" and in another university it is called "takesCourse".

IV. Virtual Integration Approach

We discuss now the implementation of the virtual integration approach. We assume that each data source exposes its data through a SPARQL query endpoint. In the virtual integration approach, a mediator issues SPARQL federated queries [2] to do most of the integration tasks. The queries contain fragments that are sent to different SPARQL query endpoints and the mediator combines the results through join or union. Each fragment has an associated `SERVICE` keyword to specify which source will receive this fragment. Note that different sources may have data in different schema so the mediator needs to map the query fragment to the schema at the corresponding source.

Next we describe how to address the data quality issues (I1)-(I4) listed in Section II using the virtual integration approach.

(I1) Exact duplicates: The removal of exact duplicates can be implemented by using the `DISTINCT` keyword in SPARQL. For example below is test query 1 in LUBM benchmark.

```
PREFIX rdf: <http://www.w3.org/1999
/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004
/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:GraduateStudent .
  ?X ub:takesCourse
http://www.Department0.University0.edu
/GraduateCourse0 .}
```

Suppose there are two data sources, one local and one with a SPARQL endpoint at IP 192.168.56.103,

The mediated query will be rewritten as

```
PREFIX rdf: <http://www.w3.org/1999
/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2
/2004/0401/univ-bench.owl#>
SELECT DISTINCT ?X
WHERE
{{?X rdf:type ub:GraduateStudent .
  ?X ub:takesCourse
http://www.Department0.University0.edu
/GraduateCourse0}
```

```
Union
{SERVICE <http://192.168.56.103:3030/lubm>
{?X rdf:type ub:GraduateStudent .
  ?X ub:takesCourse
http://www.Department0.University0.edu
/GraduateCourse0}}}
```

(I2) Approximate duplicates: The approximate duplicates can be eliminated in the post-processing step if there is a clear definition of what the approximate duplicates are. Suppose a query returns professors in different universities. Once the original query is executed, the mediator executes a SPARQL query that returns pairs of professors that have the same name, and the same institution where they get undergraduate and graduate degree from, as shown below. The mediator can then compare these pairs of professors to the returned results to eliminate duplicates.

```
PREFIX rdf: <http://www.w3.org/1999
/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/
2004/0401/univ-bench.owl#>
SELECT ?Professor ?Professor2
WHERE
{ ?Professor hasName ?N .
?Professor undergraduateDegreeFrom ?U1 .
?Professor graduateDegreeFrom ?U2
 SERVICE <http://192.168.56.103:3030/lubm>
{?Professor2 hasName ?N .
?Professor2 undergraduateDegreeFrom ?U1 .
?Professor2 graduateDegreeFrom ?U2
}}
```

(I3) Missing values: In RDF, missing values can be represented as blank nodes (i.e., nodes with no IRI or literal). Blank nodes can be filtered out by adding a `FILTER` predicate to the SPARQL query. For example, if we want to make sure a variable ?Y is not blank, we can add the following in the WHERE clause "FILTER (!isBlank(?Y))".

(I4) Ontology mismatch: The mediator can handle ontology mismatch cases between different sources by mapping the local schema to a global schema (the local as view approach) or vice versa (the global as view approach). For example, if the predicate "takesClass" is mapped to the predicate "takesCourse" in a local schema, the mediator can replace the former with the latter in the query fragment sent to that source.

V. Data Warehouse Approach

The data warehouse approach first extracts data from multiple sources, then transforms data to address data quality issues, and finally loads data into the data warehouse. Next we describe a possible implementation for the data integration issues (I1)-(I4) described in Section II and a new issue specific to this approach, (I5) which is detecting changes in the original data sources.

(I1) Exact duplicates: Exact duplicates are automatically removed when RDF data is loaded into a RDF triple store as most triple stores build a primary index on the combination of subject, predicate, and object of each triple so duplicate triples will be automatically rejected.

(I2) Approximate duplicates: Approximate duplicates can be removed at transform step by executing a similar query as in the virtual integration approach. For example, to remove professors with same name, and same undergraduate and graduate institutions, a query can be executed to return such pairs of professors as in Section IV. The only difference is that data is already extracted to the same place so there is no need for the SERVICE keyword.

(I3) Missing values: The data warehouse approach uses the same FILTER predicate as the virtual integration approach to detect blank nodes. Triples with blank nodes are typically removed.

(I4) Ontology Mismatch: The data warehouse approach addresses ontology mismatch during the transform phase. This often requires using a scripting or programming language other than SPARQL. For example, the "takesClass" predicate can be replaced with "takesCourse".

(I5) Detecting changes: There are two possible scenarios. In the first scenario, each data source periodically sends a file that contains incremental update to the data warehouse. In this case the new data just need to be loaded into the data warehouse. In the second scenario, data sources will not notify the data warehouse. Instead someone needs to periodically visit each data source and detect changes in source data. Suppose each source stores data in `ttl` files. The following SPARQL query checks can be used to detect triples in the first file but not in the second file.

```
PREFIX rdf: <http://www.w3.org/1999
/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/
2004/0401/univ-bench.owl#>
SELECT *
    FROM NAMED <a.ttl>
    FROM NAMED <b.ttl>
 WHERE {
  GRAPH :a.ttl { ?s ?p ?o }
  FILTER NOT EXISTS
  { GRAPH :b.ttl { ?s ?p ?o } }
}
```

VI. Related Work

Lehigh University Benchmak (LUBM) [4] measures SPARQL query performance for centralized RDF triple stores and is the basis for the RDFINT benchmark proposed in this paper. BSBM [5], SP2Bench [6], and DBPSB [7] are other benchmarks for centralized RDF triple stores. FedBench [8] and LargeRDFBench [9] are benchmarks for federated RDF stores, but they do not consider data

Data Set	University0 (Triples)	University1 (Triples)	University2 (Triples)	Load time (Data warehouse)	Load time (Virtual integration)
Small	103,076	134,070	145,696	10 sec	2.6 sec
Medium	1,527,819	1,623,465	1,591,536	76.8 sec	58.4 sec
Large	20,196,566	19,820,895	20,320,636	1159 sec	601 sec

Table I
DATA SIZES AND INITIAL LOAD TIME

integration. There are several benchmarks to measure data integration for relational data [10] but there is relatively little work on benchmarking data integration of RDF data. Linked Open Data Integration Benchmark (LODIB) [11] is a benchmark for data integration of RDF data but it focuses only on structure translation between different data sources and does not consider comparison between the two data integration approaches. It also does not consider a wider range of data cleaning operations such as detecting nontrivial duplicates. In [12] authors discuss issues that arise when integrating data from different RDF triple stores to support drug discovery, but there is no comparison between a federated approach versus a warehouse approach.

VII. PRELIMINARY RESULTS

A. Experimental Setup

The experiments for the virtual integration approach were performed using up to four virtual machines (VMs), each representing a data source in a real-world network. Each VM ran Ubuntu 18.0.4 (64bit), 8GB RAM, and 100GB hard disk. For the data warehouse experiment we used one of the VMs but we increased the RAM to 16GB RAM. The host machine had 64 GB RAM and 2 TB hard disk running Windows 10. We used Apache Jena Fuseki as the RDF triple stores.

For our experiments we generated three data sets: Small (approximately 100,000 to 150,000 triples per data source), Medium (approximately 1.5 million triples per data source), and Large (approximately 20 million triples per data source). The exact number of triples is shown in Table I.

In the experiments we varied the factors (F1)-(F5) introduced in Section II. The size of the data sets (F1) was Small, Medium, or Large as defined above, the number of sources (F2) was varied from 2 to 4, the fraction of the data being updated (F3) was an additional 1%, 3%, 5% of new data being added to each data source. As characteristics of the data (F4) we varied the fraction of exact duplicates from 5%, 10% to 20% of the total triples. We simulated a network delay between sources (F5) of 0 ms, 50 ms (typical US east coast to west coast delay), 100 ms (typical US to Europe delay), and 200 ms (typical US to Asia delay).

To simulate the data integration and quality issues introduced in Section II, we modified the LUBM data generator to generate exact duplicates (I1) as described above (F4). About 15 approximate duplicates (I2) were generated as professors from different universities that have the same

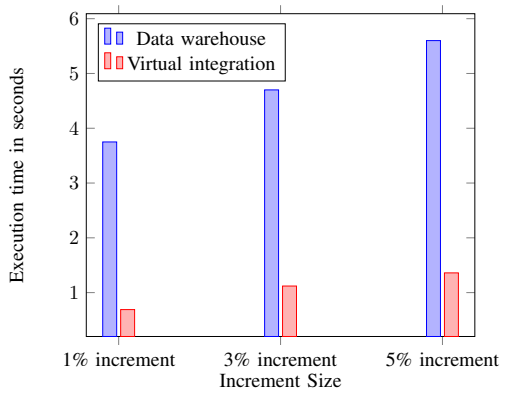

Figure 1. Incremental load time for Medium data

name, undergraduate and graduate institutions. About 10-20 blank nodes (I3) were also generated as email or phone numbers of professors in each source. The structure change (I4) happens in one of the sources.

For each experiment, we varied one of the factors and kept the others at the default values. The default setting uses Medium data set, 3 sources, no network delay, and 20% exact duplicates. Each experiment was repeated 6 times and the average time was reported.

Next we report the experimental results of evaluating the data warehouse and the virtual integration approaches.

B. Experimental Results

Load Time: The initial load time is reported in Table I. Figure 1 reports the time to load 1%, 3%, and 5% increment of the Medium data set in the data warehouse approach. For the virtual integration approach, we report the load time for the data source with the most triples, as all sources can load their data in parallel. The results show that the load time for data warehouse approach is much higher, which is expected as in virtual integration approach all sources can load their data in parallel. However the incremental load time is still much smaller than initial load time.

Removing exact duplicates: Only virtual integration needs to explicitly remove exact duplicates, since the data warehouse automatically removes duplicates when data is inserted. We selected 4 LUBM queries: Query 1, 3, 4, and 9. These queries have various degree of complexity and selectivity. In virtual integration approach, these queries are rewritten to return results from multiple sources.

Figure 2 reports execution time of all four queries with or without the use of DISTINCT keyword, for the default setting. The results show that for LUBM queries 1, 3, and 4, the execution time with or without DISTINCT is almost the same. Note that the difference of these two execution times is the overhead to remove exact duplicates. So there is very little overhead to filter out exact duplicates. For query 9, the overhead is about 0.5 seconds. It is higher than other queries because Q9 is not selective and returns many results so it takes more time to remove duplicates.

Figure 2. Execution time of 4 Queries with and without distinct for Medium data, 3 sources, no delay

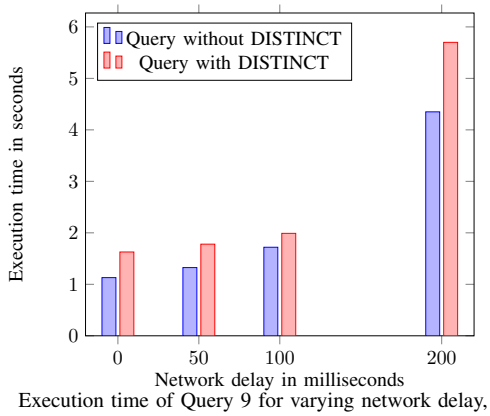

Figure 3. Execution time of Query 9 for varying network delay, Medium data, 3 sources

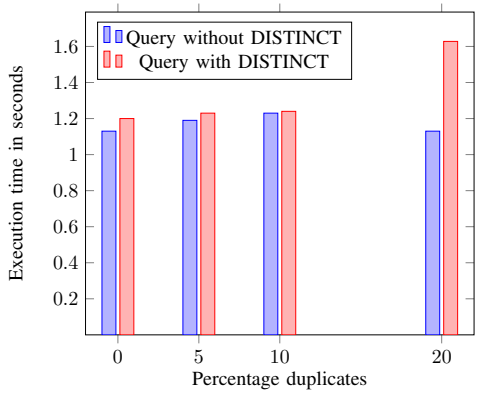

Figure 4. Execution time of Query 9 for varying duplicates, Medium data, 3 sources, no delay

Figure 3 shows the execution time for query 9 (the most expensive one) when we vary the network delay. The results show that execution time increases with network delay, and the overhead of removing exact duplicates also increases. The overhead of removing duplicates is still around 1.4 seconds with 200 ms delay.

Figure 4 reports time of query 9 when we vary the fraction of duplicates between data sources. The results show that the overhead of removing exact duplicates is quite small when fraction of duplicates is no more than 10%.

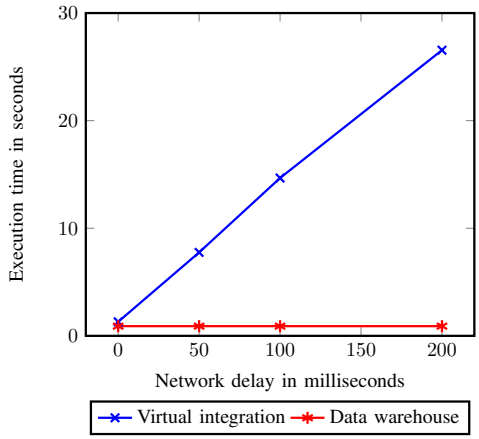

Figure 5. Execution time of removing approximate duplicates varying network delay, Medium data, 3 sources

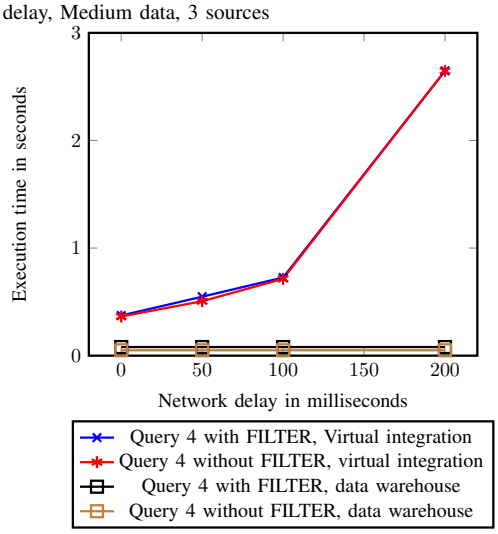

Figure 6. Execution time of query 4 with or without the FILTER predicate for various network delay, Medium data, 3 sources

The overhead does increase when the fraction of duplicates reaches 20%.

Overall, the overhead of removing exact duplicates using a virtual integration approach is acceptable.

Removing approximate duplicates: Both virtual integration and data warehouse approaches used a SPARQL query (details in Section IV) to find pairs of professors who are from different sources but have the same name, graduate degree and undergraduate degree institutions.

Figure 5 reports execution time of the query to find approximate duplicates for default setting (3 sources) with various network delay. The time for data warehouse is a flat line as data is stored in one place.

The results show that it takes about 1 second for the data warehouse approach to remove approximate duplicates. The time for virtual integration approach is higher, approaching 27 seconds under 200 ms delay. This is expected as the SPARQL query needs to perform expensive joins between different sources. However, the execution time increases

linearly with network delay.

Note that we tested the case when all approximate duplicates are from different sources. If approximate duplicates are from the same source then the cost of the SPARQL query will be lower as the joins will be done locally.

Filtering missing values: We ran LUBM Query 4 which returns professors in a specific department as well as their emails and phone numbers. We ran both the original version and a version using a FILTER predicate to remove blank nodes. The difference of execution time indicates the overhead of removing blank nodes.

Figure 6 reports execution time of Query 4 with or without the FILTER predicate for various network delays. The execution time for data warehouse approach is also reported as two flat lines because network delay has no impact on the data warehouse. The results show that there is very little overhead to filter out blank nodes in virtual integration approach. In data warehouse approach, the time of running the query with or without filtering is very low and almost identical as well.

Ontology Mismatch: It took approximately 11 seconds for the data warehouse approach to replace the "takesClass" predicate with the "takesCourse" predicate for the Medium data set. For virtual integration approach, the correct predicate was used for each data source.

Impact of data size and number of data sources: Figure 7 reports execution time of Query 9 with and without DISTINCT when we vary the number of sources from 1 to 4 over the Medium data with a network delay of 200 ms. The results show that overhead of removing exact duplicates (the differences of execution time with and without distinct) scales almost linearly with number of sources. For example, the overhead is about 0.33 seconds in one source and about 1.4 seconds for 4 sources.

Figure 8 reports execution time of Query 9 with or without use of keyword DISTINCT for various sized data for 3 sources and 200 ms network delay. The results show that the execution time of Query 9 with or without distinct grows almost linearly as data size increases (note that Large data is approximately 100 times of Small and 10 times of Medium). More importantly, the overhead of removing exact duplicates is around 1-2 seconds for all data sizes. So the virtual integration approach is scalable with respect to data size.

Completeness of results: For the data warehouse approach, we ran LUBM Query 6, which returns all students, on the original Medium data set as well as original data plus 1%, 3%, and 5% additional data. Table II shows the number of results returned by the query. Clearly, more results are returned in data sets with incremental updates.

Overall, virtual integration adds some overhead at run time to address data quality issues compared to data warehouse approaches. However this approach gives complete and real time results while the data warehouse solution's

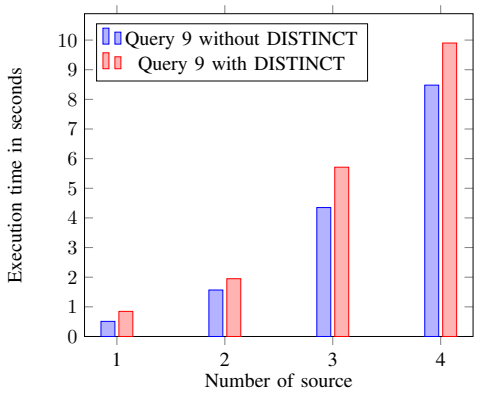

Figure 7. Execution time of Query 9 varying number of sources on Medium data, 200ms delay

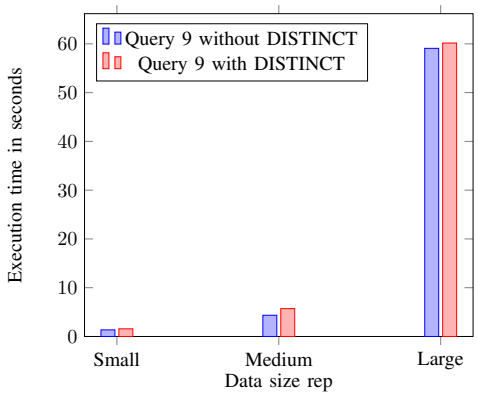

Figure 8. Execution time of Query 9 on various data sets with 200ms delay, 3 sources

results are based on last batch update in the data warehouse. So if data sources are updated more frequently than the data warehouse, the data warehouse solution could give out-of-date and incomplete results.

Usability: We found that virtual integration approach requires more knowledge of federated SPARQL queries and knowledge about different data sources, including their service endpoint and schema. So usability-wise, data warehouse approach is better than virtual integration approach.

VIII. Conclusion

We proposed RDFINT, a benchmark for comparing virtual integration and data warehouse approaches to integrating RDF data. This benchmark modifies well-known LUBM benchmark data generator and test queries, and implements several data integration operations. It can be used to compare the above two data integration approaches over RDF data.

Original data	Original + 1% increment	Original + 3% increment	Original + 5% increment
259,965	262,745	268,000	273,386

Table II
TOTAL NUMBER OF TRIPLES RETURNED FROM QUERY 6 ON MEDIUM DATA ON DIFFERENT INCREMENTS

Preliminary results showed that virtual integration does pay some overhead at run time, but has the benefit of result completeness. We also found that many data integration/cleaning operations can be implemented using simple query rewriting in virtual integration approach. The overhead of these operators is often quite small compared to cost of the original queries. Most data integration operations also scale linearly with data size and number of sources. As future work we plan to include more comprehensive set of data integration operations in the benchmark and conduct more comprehensive experiments.

IX. Acknowledgement

This work was partially supported by Office of Naval Research grants N00014-18-12452 and N00014-19-WX00152.

References

[1] RDF, "http://www.w3.org/rdf/."

[2] SPARQL 1.1 Overview, "https://www.w3.org/tr/sparql11-overview/."

[3] Apache Jena, "Apache jena: A free and open source java framework for building semantic web and linked data applications," accessed June 2, 2020. [Online]. Available: https://jena.apache.org/

[4] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semantics*, vol. 3, no. 2-3, pp. 158–182, Oct. 2005.

[5] C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 2, pp. 1–24, 2009.

[6] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "sp^2bench: A sparql performance benchmark," in *2009 IEEE 25th International Conference on Data Engineering*, 2009, pp. 222–233.

[7] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo, "Dbpedia sparql benchmark – performance assessment with real queries on real data," in *International Semantic Web Conference*, ser. ISWC 2011, 2011, pp. 454–469.

[8] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran, "Fedbench: A benchmark suite for federated semantic data query processing," in *International Semantic Web Conference*. Springer, 2011, pp. 585–600.

[9] M. Saleem, A. Hasnain, and A.-C. Ngonga Ngomo, "Largerdfbench: A billion triples benchmark for sparql endpoint federation," *Journal of Web Semantics*, vol. 48, pp. 85 – 125, 2018.

[10] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield, "Tpc-di: the first industry benchmark for data integration," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1367–1378, 2014.

[11] C. R. Rivero, A. Schultz, C. Bizer, and D. Ruiz Cortés, "Benchmarking the performance of linked data translation systems," in *LDOW 2012: WWW2012 Workshop on Linked Data on the Web (2012),*. CEUR-WS, 2012.

[12] A. Hasnain, M. R. Kamdar, P. Hasapis, D. Zeginis, C. N. Warren, H. F. Deus, D. Ntalaperas, K. Tarabanis, M. Mehdi, and S. Decker, "Linked biomedical dataspace: Lessons learned integrating data for drug discovery," in *International Semantic Web Conference*, ser. ISWC 2014, 2014, pp. 114–130.