LUBM: A Benchmark for OWL Knowledge Base Systems

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin

Computer Science & Engineering Department
Lehigh University,
19 Memorial Drive West, Bethlehem, PA18015, USA
Tel: (610) 758-4719
Fax: (610)758-4096
{yug2, zhp2, heflin}@cse.lehigh.edu

This is a revised and extended version of the paper "An Evaluation of Knowledge Base Systems for Large OWL Datasets" presented at ISWC2004.

Abstract

We describe our method for benchmarking Semantic Web knowledge base systems with respect to use in large OWL applications. We present the Lehigh University Benchmark (LUBM) as an example of how to design such benchmarks. The LUBM features an ontology for the university domain, synthetic OWL data scalable to an arbitrary size, fourteen extensional queries representing a variety of properties, and several performance metrics. The LUBM can be used to evaluate systems with different reasoning capabilities and storage mechanisms. We demonstrate this with an evaluation of two memory-based systems and two systems with persistent storage.

Keywords

Semantic Web, Knowledge Base System, Lehigh University Benchmark, Evaluation

1. Introduction

Various knowledge base systems (KBS) have been developed for storing, reasoning and querying Semantic Web information. They differ in a number of important ways. For instance, many KBSs are main memory-based while others use secondary storage to provide persistence. Another key difference is the degree of reasoning provided by the KBS. Some KBSs only support RDF/RDFS [44] inferencing while others aim at providing extra OWL [10] reasoning.

In this paper, we consider the issue of how to choose an appropriate KBS for a large OWL application. Here, we consider a large application to be one that requires the processing of megabytes of data. Generally, there are two basic requirements for such systems. First, the enormous amount of data means that scalability and efficiency become crucial issues. Secondly, the system must provide sufficient reasoning capabilities to support the semantic requirements of the application.

It is difficult to evaluate KBSs with respect to these requirements. In order to evaluate the systems in terms of scalability, we need Semantic Web data that are of a range of large sizes and commit to semantically rich ontologies. However, there are few such datasets available on the current Semantic Web. As for the reasoning requirement, the ideal KBS would be sound and complete for OWL Full. Unfortunately, increased reasoning capability usually means an increase in data processing time and/or query response time as well. Given this, one might think that the next best thing is to choose a KBS that is just powerful enough to meet the reasoning needs of the application. However, this is easier said than done. Many systems are only sound and complete for unnamed subsets of OWL. For example, most description logic (DL) reasoners are complete for a language more expressive than OWL Lite (if we ignore datatypes) but, ironically, less expressive than OWL DL. As such, although some systems are incomplete with respect to OWL, they may still be useful because they scale better or respond to queries more quickly. Therefore, to evaluate the existing KBSs for the use in those OWL applications, it is important to be able to measure the tradeoffs between scalability and reasoning capability.

In light of the above, there is a pressing need for benchmarks to facilitate the evaluation of Semantic Web KBSs in a standard and systematic way. Ideally we should have a suite of such benchmarks, representing different workloads. These benchmarks should be based on well-established practices for benchmarking databases [3, 4, 8, 33], but must be extended to support the unique properties of the Semantic Web. As a first step, we have designed a benchmark that fills a void that we consider particularly important: extensional queries over a large dataset that commits to a single ontology of moderate complexity and size. Named the Lehigh University Benchmark (LUBM), this benchmark is based on an ontology for the university domain. Its test data are synthetically generated instance data over that ontology; they are random and repeatable and can be scaled to an arbitrary size. It offers fourteen test queries over the data. It also provides a set of performance metrics used to evaluate the system with respect to the above mentioned requirements.

A key benefit of our benchmarking approach is that it allows us to empirically compare very different knowledge base systems. In order to demonstrate this, we have conducted an experiment using three different classes of systems: systems supporting RDFS reasoning, systems providing partial OWL Lite reasoning, and systems that are complete or almost complete for OWL Lite. The representatives of these classes that we chose are Sesame, DLDB-OWL, and OWLJessKB, respectively. In addition to having different reasoning capabilities, these systems differ in their storage mechanisms and query evaluation. We have evaluated two memory-based systems (memory-based Sesame and OWLJessKB) and two systems with persistent storage (database-based Sesame and DLDB-OWL). We describe how we have settled on these systems and set up the experiment in the benchmark framework. We show the experimental results and discuss the performance of the systems from several different aspects. We also discuss some interesting observations. Based on that, we highlight some issues with respect to the development and improvement of the same kind of systems, and suggest some potential ways in using and developing those systems.

This work makes two major contributions. First, we introduce our methodology for benchmarking Semantic Web knowledge base systems and demonstrate it using the LUBM as a specific product. We believe our work could benefit the Semantic Web community by inspiring and guiding the development of other benchmarks. Secondly, the experiment we have done could help developers identify some important directions in advancing the state-of-the-art of Semantic Web KBSs.

The outline of the paper is as follows: Section 2 elaborates on the LUBM. Section 3 describes the aforementioned experiment. Section 4 talks about related work. Section 5 considers issues in applying the benchmark. We conclude in Section 6.

2. Lehigh University Benchmark for OWL

As mentioned earlier, this paper presents a benchmark that is intended to fill a void that we considered particularly important. The benchmark was designed with the following goals in mind:

1) *Support extensional queries*. Extensional queries are queries about the instance data over ontologies. There already exist DL benchmarks that focus on intensional queries (i.e., queries about classes and properties). However, our conjecture is that the majority of Semantic Web applications will want to use data to answer questions, and that reasoning about subsumption will typically be a means to an end, not an end in itself. Therefore, it is important to have benchmarks that focus on this kind of query.
2) *Arbitrary scaling of data***.** We also predict that data will by far outnumber ontologies on the Semantic Web of the future. Here, we use the word "data" to refer to assertions about instances, in other words what is referred to as an ABox in DL terminology. In order to evaluate the ability of systems to handle large ABoxes we need to be able to vary the size of data, and see how the system scales.
3) *Ontology of moderate size and complexity***.** Existing DL benchmarks have looked at reasoning with large and complex ontologies, while various RDF systems have been evaluated with regards to various RDF Schemas (which could be considered simple ontologies). We felt that it was important to have a benchmark that fell between these two extremes. Furthermore, since our focus is on data, we felt that the ontology should not be too large.

It should be noted that these goals choose only one point in the space of possible benchmarks. We recognize that some researchers may not agree with the importance we placed on these goals, and would suggest alternative benchmarks. We encourage these researchers to consider the lessons learned in this work and develop their own benchmarks.

Given the goals above, we designed the Lehigh University Benchmark. The first version [14] of this benchmark was used for the evaluation of DAML+OIL [9] repositories, but has since been extended to use an OWL ontology and dataset. We introduce the key components of the benchmark suite below.

2.1. Benchmark Ontology

The ontology used in the benchmark is called Univ-Bench. Univ-Bench describes universities and departments and the activities that occur at them. Its predecessor is the Univ1.0 ontology[1], which has been used to describe data about actual universities and departments. We chose this ontology expecting that its domain would be familiar to most of the benchmark users.

We have created an OWL version of the Univ-Bench ontology[2]. The ontology is expressed in OWL Lite, the simplest sublanguage of OWL. We chose to restrict the ontology (and also the test data) to OWL Lite since efficient reasoning systems exist for that language, for example, Racer [17] and Pellet [42]. Note that neither of these systems is complete for OWL DL.

[1] http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml

[2] http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl

For benchmark purposes, we have intentionally included in the ontology specific language features. For instance, originally the Univ1.0 ontology states that GraduateStudent is a subclass of Student. In creating the Univ-Bench ontology, we have replaced that definition with what is shown in Fig. 1 using a restriction. As a result, the subclass relationship between both the classes GraduateStudent and Student must be inferred using OWL semantics. Moreover, sometimes an inference could be made in multiple ways. To allow emphasis on description logic subsumption, we have made some domain constraint changes. For example, we have removed the domain constraint (to the class Student) of the property takesCourse so that no individuals of GraduateStudent in the benchmark data can be inferred as an instance of Student without the inference of the subsumption relationship between both classes from the ontology.

OWL Code

```
<owl:Class rdf:ID="GraduateCourse">
   <rdfs:label>graduate level Courses</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Course" />
</owl:Class>
<owl:Class rdf:ID="GraduateStudent">
   <rdfs:label>graduate student</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Person" />
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#takesCourse" />
         <owl:someValuesFrom>
            <owl:Class rdf:about="#GraduateCourse" />
         </owl:someValuesFrom>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Student">
   <rdfs:label>student</rdfs:label>
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Person" />
      <owl:Restriction>
         <owl:onProperty rdf:resource="#takesCourse" />
         <owl:someValuesFrom>
            <owl:Class rdf:about="#Course" />
         </owl:someValuesFrom>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Simplified DL Syntax Description

$$\text{GraduateCourse} \sqsubseteq \text{Course}$$

$$\text{GraduateStudent} \sqsubseteq \text{Person} \sqcap \exists\text{takesCourse.GraduateCourse}$$

$$\text{Student} \equiv \text{Person} \sqcap \exists\text{takesCourse.Course}$$

Fig. 1. Definition of the classes GraduateStudent and Student

The ontology currently defines 43 classes and 32 properties (including 25 object properties and 7 datatype properties). It uses OWL Lite language features including *inverseOf*, *TransitiveProperty*, *someValuesFrom* restrictions, and *intersectionOf*. According to a study by Tempich and Volz [34], this ontology can be categorized as a "description logic-style" ontology, which has a moderate number of classes but several restrictions and properties per class.

2.2. Data Generation and OWL Datasets

Test data of the LUBM are extensional data created over the Univ-Bench ontology. For the LUBM, we have adopted a method of synthetic data generation. This serves multiple purposes. As with the Wisconsin benchmark [3, 4], a standard and widely used database benchmark, this allows us to control the selectivity and output size of each test query. However, there are some other specific considerations:

1) We would like the benchmark data to be of a range of sizes including considerably large ones. It is hard to find such data sources that are based on the same ontology.
2) We may need the presence of certain kinds of instances in the benchmark data. This allows us to design repeatable tests for as many representative query types as possible. These tests not only evaluate the storage mechanisms for Semantic Web data but also the techniques that exploit formal semantics. We may rely on instances of certain classes and/or properties to test against those techniques.

Data generation is carried out by UBA (Univ-Bench Artificial data generator), a tool we have developed for the benchmark. We have implemented the support for OWL datasets in the tool. The generator features random and repeatable data generation. A university is the minimum unit of data generation, and for each university, a set of OWL files describing its departments are generated. Instances of both classes and properties are randomly decided. To make the data as realistic as possible, some restrictions are applied based on common sense and domain investigation. Examples are "a minimum of 15 and a maximum of 25 departments in each university", "an undergraduate student/faculty ratio between 8 and 14 inclusive", "each graduate student takes at least 1 but at most 3 courses", and so forth. A detailed profile of the data generated by the tool can be found on the benchmark's webpage.

The generator identifies universities by assigning them zero-based indexes, i.e., the first university is named University0, and so on. Data generated by the tool are exactly repeatable with respect to universities. This is possible because the tool allows the user to enter an initial seed for the random number generator that is used in the data generation process. Through the tool, we may specify how many and which universities to generate.

Finally, as with the Univ-Bench ontology, the OWL data created by the generator are also in the OWL Lite sublanguage. As a consequence, we have had to give every individual ID appearing in the data a type and include in every document an ontology tag (the *owl:Ontology* element)[3].

2.3. Test Queries

The LUBM currently offers fourteen test queries, one more than when it was originally developed. Readers are referred to Appendix 1 for a list of these queries. They are written in SPARQL [28], the query language that is poised to become the standard for RDF. In choosing the queries, first of all, we wanted them to be realistic. Meanwhile, we have mainly taken into account the following factors:

1) *Input size*. This is measured as the proportion of the class instances involved in the query to the total class instances in the benchmark data. Here we refer to not just class instances explicitly expressed but also those that are entailed by the knowledge base. We define the input size as large if the proportion is greater than 5%, and small otherwise.
2) *Selectivity.* This is measured as the estimated proportion of the class instances involved in the query that satisfy the query criteria. We regard the selectivity as high if the proportion is lower than 10%, and low otherwise. Whether the selectivity is high or low for a query may depend on the dataset used. For instance, the selectivity of Queries 8, 11 and 12 is low if the dataset contains only University0 while high if the dataset contains more than 10 universities.
3) *Complexity.* We use the number of classes and properties that are involved in the query as an indication of complexity. Since we do not assume any specific implementation of the repository, the real degree of complexity may vary by systems and schemata. For example, in a relational database, depending on the schema design, the number of classes and properties may or may not directly indicate the number of table joins, which are significant operations.
4) *Assumed hierarchy information.* This considers whether information from the class hierarchy or property hierarchy is required to achieve the complete answer. (We define completeness in next subsection).
5) *Assumed logical inference.* This considers whether logical inference is required to achieve the completeness of the answer. Features used in the test queries include subsumption, i.e., inference of implicit subclass relationship, *owl:TransitiveProperty*, *owl:inverseOf*, and realization, i.e., inference of the most specific concepts that an individual is an instance of. One thing to note is that we are not benchmarking complex description logic reasoning. We are concerned with extensional queries. Some queries use simple description logic reasoning mainly to verify that this capability is present.

We have chosen test queries that cover a range of properties in terms of the above criteria. At the same time, to the end of performance evaluation, we have emphasized

[3] In OWL, the notion of the term ontology differs from that in the traditional sense by also including instance data [32].

queries with large input and high selectivity. If not otherwise noted, all the test queries are of this type. Some subtler factors have also been considered in designing the queries, such as the depth and width of class hierarchies[4], and the way the classes and properties chain together in the query.

2.4. Performance Metrics

The LUBM consists of a set of performance metrics including load time, repository size, query response time, query completeness and soundness, and a combined metric for query performance. Among these metrics: the first three are standard database benchmark metrics—query response time was introduced in the Wisconsin benchmark, and load time and repository size have been commonly used in other database benchmarks, e.g. the OO1 benchmark [8]; query completeness and soundness and the combined metric are new metrics we developed for the benchmark. We address these metrics in turn below.

Load Time

In a LUBM dataset, every university contains 15 to 25 departments, each described by a separate OWL file. These files are loaded to the target system in an incremental fashion. We measure the load time as the stand alone elapsed time for storing the specified dataset to the system. This also counts the time spent in any processing of the ontology and source files, such as parsing and reasoning.

Repository Size

Repository size is the resulting size of the repository after loading the specified benchmark data into the system. Size is only measured for systems with persistent storage and is calculated as the total size of all files that constitute the repository. We do not measure the occupied memory sizes for the main memory-based systems because it is difficult to accurately calculate them. However, since we evaluate all systems on a platform with a fixed memory size, the largest dataset that can be handled by a system provides an indication of its memory efficiency.

Query Response Time

Query response time is measured based on the process used in database benchmarks. To account for caching, each query is executed for ten times consecutively and the average time is computed. Specifically, the benchmark measures the query response time as the following:

```
For each test query
    Open the repository
        Execute the query on the repository consecutively for 10 times and compute the average response time. Each time:
```

[4] We define a class hierarchy as deep if its depth is greater than 3, and as wide if its average branching factor is greater than 3.

```
        Issue the query, obtain the result set, traverse that set sequentially, and
        collect the elapsed time
    Close the repository
```

Query Completeness and Soundness

We also examine query completeness and soundness of each system. In logic, an inference procedure is complete if it can find a proof for any sentence that is entailed by the knowledge base. With respect to queries, we say a system is complete if it generates all answers that are entailed by the knowledge base, where each answer is a binding of the query variables that results in an entailed sentence. However, on the Semantic Web, partial answers will often be acceptable. So it is important not to measure completeness with such a coarse distinction. Instead, we measure the degree of completeness of each query answer as the percentage of the entailed answers that are returned by the system. Note that we request that the result set contains unique answers.

In addition, as we will show in the next section, we have realized that query soundness is also worthy of examination. With similar argument to the above, we measure the degree of soundness of each query answer as the percentage of the answers returned by the system that are actually entailed.

Combined Metric (CM)

The benchmark also provides a metric for measuring query response time and answer completeness and soundness in combination to help users better appreciate the potential tradeoff between the query response time and the inference capability and the overall performance of the system. Such a metric can be used to provide an absolute ranking of systems. However, since such a metric necessarily abstracts away details, it should be used carefully. In order to allow the user to place relative emphasis on different aspects, we have parameterized the metric. Changing these parameters can result in a reordering of the systems considered. Thus, this metric should never be used without careful consideration of the parameters used in its calculation. We will come back to this in Section 5.

First, we use an F-Measure [30, 24] like metric to compute the tradeoff between query completeness and soundness, since essentially they are analogous to recall and precision in Information Retrieval respectively. In the formula below, C_q and S_q ($\in [0, 1]$) are the answer completeness and soundness for query q. β determines the relative weighting between S_q and C_q.

$$F_q = \frac{(\beta^2 + 1) * C_q * S_q}{\beta^2 * C_q + S_q}$$

Next, we define a query performance metric for query q:[5]

$$P_q = \frac{1}{1 + e^{a * T_q / N - b}}$$

[5] We introduce a different function from that of our original work [16], since we find that the current one could map the query time (possibly from zero to infinity) to the continuous range of [0, 1] in a more natural way, and could give us a more reasonable range and distribution of query performance evaluation of the systems.

This is an adaptation of the well known sigmoid function. In the formula, T_q is the response time (ms) for query q and N is the total number of triples in the dataset concerned. To allow for comparison of the metric values across datasets of different sizes, we use the response time per triple (i.e. T_q/N) in the calculation. a is used to control the slope of the curve. b shifts the entire graph. In this evaluation, we have chosen a of 500 and b of 5 so that 1) $P_q(0)$ is close enough to one (above 0.99); 2) A response time of one second for 100,000 triples (roughly the size of the smallest dataset used in our test) will receive the P_q value of 0.5. These values appear to be reasonable choices for contemporary systems. However, as the state-of-the-art progresses and machines become substantially more powerful, different parameter values may be needed to better distinguish between the qualities of different systems. Fig. 2 illustrates the function for the triple size of one million.

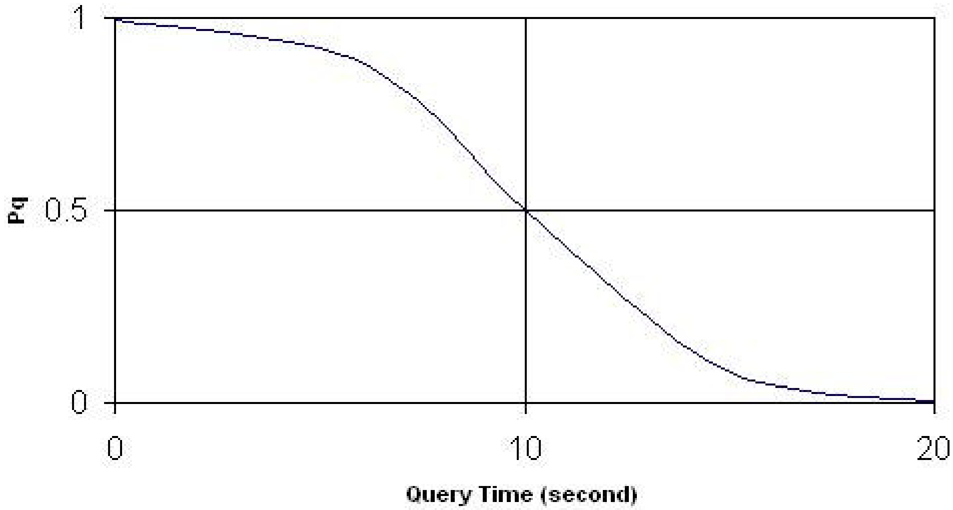

Fig. 2. P_q function (a=500, b=5, N=1,000,000)

As will be shown in the next section, some system might fail to answer a query. In that case, we will use zero for both F_q and P_q in the calculation.

Lastly, we define a composite metric CM of query response time and answer completeness and soundness as the following, which is also inspired by F-Measure:

$$CM = \sum_{q=1}^{M} w_q * \frac{(\alpha^2 + 1) * P_q * F_q}{\alpha^2 * P_q + F_q}$$

In the above, M is the total number of test queries. w_q ($\sum_{q=1}^{M} w_q = 1$) is the weight given to query q. α determines relative weighting between F_q and P_q. Generally speaking, the metric will reward those systems that can answer queries faster, more completely and more soundly.

2.5. Benchmark Architecture

Fig. 3 depicts the benchmark architecture. We prescribe an interface to be instantiated by each target system. Through the interface, the benchmark test module requests operations on the repository (e.g. open and close), launches the loading process, issues queries and obtains the results. Users inform the test module of the target systems and test queries by defining them in the KBS specification file and query definition file respectively. It needs to be noted that queries are translated into the query language supported by the system prior to being issued to the system. In this way, we want to eliminate the effect of query translation on query response time. The translated queries are fed to the tester through the query definition file. The tester reads the lines of each query from the definition file and passes them to the system.

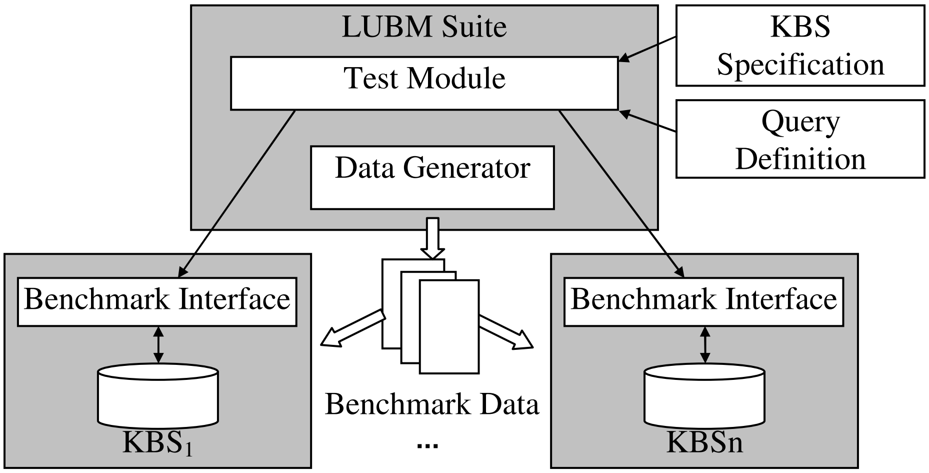

Fig. 3. Architecture of the benchmark

The benchmark suite is accessible at http://swat.cse.lehigh.edu/projects/lubm/index.htm.

3. An Evaluation Using the LUBM

In order to demonstrate how the LUBM can be used to evaluate very different knowledge base systems, we describe an experiment. We discuss how we selected the systems, describe the experiment conditions, present the results, and discuss them.

3.1. Selecting Target Systems

In this experiment, we wanted to evaluate the scalability and support for OWL Lite in various systems. In choosing the systems, first we decided to consider only non-commercial systems. Moreover, we did not mean to carry out a comprehensive evaluation of the existing Semantic Web KBSs. Instead, we wanted to evaluate systems with a variety of characteristics. Since a key point of this work is how to evaluate systems with different capabilities, we wanted to evaluate representative systems at distinct points in terms of OWL reasoning capability. Specifically, we considered three degrees of supported reasoning in the systems under test, i.e. RDFS reasoning, partial OWL Lite reasoning, and complete or almost complete OWL Lite reasoning. Finally, we believe a practical KBS must be able to read OWL files, support incremental data loading, and provide programming APIs for loading data and issuing queries. As a result, we have settled on four different knowledge base systems, including two implementations of Sesame, DLDB-OWL, and OWLJessKB. Below we briefly describe the systems we have considered along with the reason for choosing or not choosing them.

RDFS Reasoning

For this category of systems, we have considered RDF repositories including Jena, Sesame, ICS-FORTH RDFSuite [1], Kowari [40], and so on. Since Jena and Sesame are currently the most widely used, we focused on these two systems.

Jena [37] is a Java framework for building Semantic Web applications. Jena currently supports both RDF/RDFS and OWL. We have done some preliminary tests on Jena (v2.1) (both memory-based and database-based) with our smallest dataset (cf. Appendix 3). Compared to Sesame, Jena with RDFS reasoning was much slower in answering nearly all the queries. Some of the queries did not terminate even after being allowed to run for several hours. The situation was similar when Jena's OWL reasoning was turned on.

Due to Jena's limitations with respect to scalability, we decided to evaluate Sesame instead. Sesame [5] is a repository and querying facility based on RDF and RDF Schema. It features a generic architecture that separates the actual storage of RDF data, functional modules offering operations on those data, and communication with these functional modules from outside the system. Sesame supports RDF/RDF Schema inference, but is an incomplete reasoner for OWL Lite. Nevertheless, it has been used on a wide number of Semantic Web projects. Sesame can evaluate queries in RQL [22], SeRQL [45], and RDQL [31]. We evaluate two implementations of Sesame, main memory-based and database-based.

Partial OWL Lite Reasoning

KAON [39] is an ontology management infrastructure. It provides an API for manipulating RDF models. The suite also contains a library of KAON Datalog Engine, which could be used to reason with the DLP fragment of OWL. However, that functionality is not directly supported by the core of KAON, i.e., its APIs.

Instead, we have selected DLDB-OWL [27], a repository for processing, storing, and querying OWL data. The major feature of DLDB-OWL is the extension of a relational database system with description logic inference capabilities. Specifically, DLDB-OWL uses Microsoft Access® as the DBMS and FaCT [19] as the OWL reasoner. It uses the reasoner to precompute subsumption and employs relational views to answer extensional queries based on the implicit hierarchy that is inferred. DLDB-OWL uses a language in which a query is written as a conjunction of atoms. The language syntactically resembles KIF [13] but has less expressivity. Since DLDB uses FaCT to do TBox reasoning only, we consider it as a system that provides partial OWL Lite reasoning support. Note, FaCT and FaCT++ [35] do not directly support ABox reasoning (although it is possible to simulate it), so use of them as stand-alone systems is inappropriate for the extensional queries used by our benchmark.

Complete or Almost Complete OWL Lite Reasoning

OWL reasoning systems such as Racer, Pellet and OWLJessKB fall into this category. The first two systems, like FaCT, are based on the tableaux algorithms developed for description logic inferencing, and additionally support ABox reasoning. Since DLDB-OWL uses FaCT as its reasoner, we have decided to choose a system with a different reasoning style. Thus we have settled on OWLJessKB.

OWLJessKB [41], whose predecessor is DAMLJessKB [23], is a memory-based reasoning tool for description logic languages, particularly OWL. It uses the Java Expert System Shell (Jess) [38], a production system, as its underlying reasoner. Current functionality of OWLJessKB is close to OWL Lite plus some. Thus we have chosen OWLJessKB and we evaluate it as a system that supports most OWL entailments.

Although we have decided not to evaluate Racer here, we want to mention that Haarslev et al. [18] have conducted their own evaluation using the LUBM. They developed a new query language called nRQL, which made it possible to answer all of the queries in our benchmark. The results showed that Racer could offer complete answers for all the queries if required (they have tested Racer on Queries 1 through 13). However, since it has to perform ABox consistency checking before query answering, Racer was unable to load a whole university dataset. As a result, they have only loaded up to 5 departments using Racer (v. 1.8) on a P4 2.8GHz 1G RAM machine running Linux.

3.2. Experiment Setup

System Setup

The systems we test are DLDB-OWL (04-03-29 release), Sesame v1.0, and OWLJessKB (04-02-23 release). As noted, we test both the main memory-based and database-based implementations of Sesame. For brevity, we hereafter refer to them as Sesame-Memory and Sesame-DB respectively. For both of them, we use the implementation with RDFS inference capabilities. For the later, we use MySQL (v4.0.16) as the underlying DBMS since, according to a test by Broekstra and Kampman [5], Sesame performs significantly better than using the other DBMS PostgreSQL [43]. The DBMS used in DLDB-OWL is MS Access® 2002. We have created a wrapper over each system as an interface to the test module.

We should point out that we did not optimize any of the systems for purpose of the experiment. This is for two reasons. First, most Semantic Web KBSs do not provide the level of customization typically found in databases. This is true even for KBSs that use a DBMS; such systems typically make decisions for the user with respect to table layout and index creation. Second, we feel that the variability of Semantic Web data will make it more difficult to optimize for any specific application. However, if the benchmark is being used to evaluate systems for a well-defined application, then we encourage users of the benchmark to use all tools available to optimize the system so that they may conduct an evaluation that better reflects the way the system will behave operationally.

Datasets

To identify the dataset, we use the following notation in the subsequent description:

LUBM(*N, S*): The dataset that contains *N* universities beginning at University0 and is generated using a seed value of *S*.

We have created 5 sets of test data[6]: LUBM(1, 0), LUBM(5, 0), LUBM(10, 0), LUBM(20, 0), and LUBM(50, 0), which contain OWL files for 1, 5, 10, 20, and 50 universities respectively, the largest one having over 6,800,000 triples in total. To our knowledge, prior to this experiment, Sesame has been tested with at most 3,000,000 statements. We have easily exceeded that by virtue of the data generation tool. Note that in counting the RDF triples in a dataset, we count those from the ontology only once and for others, we count duplicate triples multiple times.

Query Test

As mentioned earlier, the queries are expressed in SPARQL. However, many Semantic Web KBSs were designed before SPARQL began to take shape. As such, none of the selected systems currently support SPARQL. Therefore, we manually translate the fourteen queries into a language supported by each system: RQL for Sesame, Jess for OWLJessKB, and a KIF-like query language for DLDB-OWL. The translated queries form the query definition file for that system. Note, with one exception explained below, we do not try to optimize the queries. Instead, we expect that queries are issued by naïve users and any query optimization must be a function of the KBS.

From preliminary experiments we discovered that the ordering of statements within a Jess query can affect the response time of OWLJessKB to that query. In fact, a direct translation of the benchmark queries resulted in poor performance from OWLJessKB. Even with the one-university dataset it ran out of memory for some queries (e.g., Query 2). However, we were able to reduce the response time by simply reordering the statements so that arbitrary property-related statements precede all type-related statements in each query. Since this is an optimization that could be trivially automated, we decided to make this one exception to our rule above.

As another detail, OWLJessKB needs two separate phases to perform a query: define it and execute it. In the original DAML version of our experiment [15] we predefined the patterns for each test query prior to loading any data. However, we have subsequently learned that this could lead to worse performance of OWLJessKB. Since we have found no guidance as to when to or not to use such kinds of patterns, we will show the results of OWLJessKB with both settings in the subsequent discussion. When distinction is needed, we will refer to them as OWLJessKB-P and OWLJessKB-NP respectively.

Query response time is collected in the way defined by the benchmark. Note that instead of providing a result set that can be iterated through, Sesame returns data one-at-a-time in streams and calls back user specified functions upon each result item. Thus we regard those call backs as the result traverse that is required by the benchmark, and count them in the query time instead.

[6] The version of the data generator is UBA1.6.

Test environment

We have done the test on a desktop computer. The environment is as follows:

1.80GHz Pentium 4 CPU;
256MB of RAM; 80GB of hard disk
Windows XP Professional OS;
Java SDK 1.4.1; 512MB of max heap size

In order to evaluate OWLJessKB, we needed to adjust this configuration slightly. With the standard setting for max heap size in Java, the system failed to load the one-university dataset due to out of memory errors. As a workaround, we increased the maximum heap size to 1GB, which requests a large amount of virtual memory from the operating system. This change allowed OWLJessKB to properly load the dataset.

3.3. Results and Discussions

3.3.1. Data Loading

Table 1. Load time and repository sizes

	Dataset	File #	Total Size (MB)	Triple #	Load Time (hh:mm:ss)	Repository Size (KB)
DLDB-OWL	LUBM (1, 0)	15	8.6	103,397	00:05:43	16,318
Sesame-DB					00:09:02	48,333
Sesame-Memory					00:00:13	-
OWLJessKB-P					03:16:12	-
OWLJessKB-NP					02:19:18	-
DLDB-OWL	LUBM (5, 0)	93	54.2	646,128	00:51:57	91,292
Sesame-DB					03:00:11	283,967
Sesame-Memory					00:01:53	-
OWLJessKB					-	-
DLDB-OWL	LUBM (10, 0)	189	110.6	1,316,993	01:54:41	184,680
Sesame-DB					12:27:50	574,554
Sesame-Memory					00:05:40	-
OWLJessKB					-	-
DLDB-OWL	LUBM (20, 0)	402	234.8	2,782,419	04:22:53	388,202
Sesame-DB					46:35:53	1,209,827
Sesame-Memory					-	-
OWLJessKB					-	-
DLDB-OWL	LUBM (50, 0)	999	583.6	6,890,933	12:37:57	958,956
Sesame-DB					-	-
Sesame-Memory					-	-
OWLJessKB					-	-

Table 1 shows the data loading time for all systems and the on-disk repository sizes of DLDB-OWL and Sesame-DB. Fig. 4 depicts how the load time grows as the dataset size increases and compares the repository sizes of the two database-based systems.

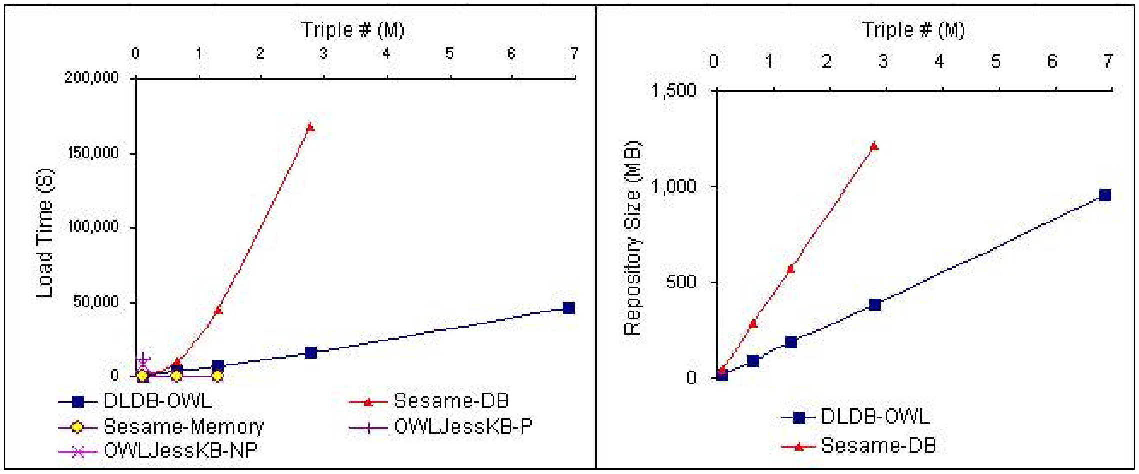

Fig. 4. Load time and repository sizes. The left hand figure shows the load time. The right hand figure shows the repository sizes of the database-based systems.

The test results have reinforced scalability as an important issue and challenge for Semantic Web knowledge base systems. One of the first issues is how large of a dataset each system can handle. As expected, the memory-based systems did not perform as well as the persistent storage systems in this regard. OWLJessKB, could only load the 1-university dataset, and took over 15 times longer than any other system to do so. On the other hand, we were surprised to see that Sesame-Memory could load up to 10 universities, and was able to do it in 5% of the time of the next fastest system. However, for 20 or more universities, Sesame-Memory also succumbed to memory limitations.

Using the benchmark, we have been able to test both Sesame-Memory and Sesame-DB on larger scale datasets than what has been reported so far. The result reveals an apparent problem for Sesame-DB: it does not scale in data loading, as can be seen from Fig. 4. As an example, it took over 300 times longer to load the 20-university dataset than the 1-university dataset, although the former set contains only about 25 times more triples than the later. We extrapolate that it will take Sesame-DB over 3 weeks to finish loading the 50-university dataset. Therefore, we have decided not to do a test that consumes so many resources.

In contrast, DLDB-OWL displays good scalability in data loading. We suspect the different performance of the two systems is caused by the following two reasons. First, to save space, both DLDB-OWL and Sesame map resources to unique IDs maintained in a table. When a resource is encountered during the data loading, they will look up that table to determine if it has not been seen before and needs to be assigned a new ID. As mentioned in [27], querying the ID table every time is very likely to slow down the data loading as the data size grows. In its implementation, Sesame also assigns every literal an ID, while DLDB-OWL stores literals directly in the destination tables, which means Sesame has to spend even more time on ID lookup. Moreover, in order to improve performance, DLDB-OWL caches resource-ID pairs during current loading.

A second reason for the performance difference is related to the way Sesame performs inference. Sesame is a forward-chaining reasoner, and in order to support statement deletions it uses a truth maintenance system to track all deductive dependencies between statements. As Broekstra and Kampman [6] shows, this appears to affect the performance significantly if there are many inferred statements or the dataset is fairly large. We should note that this scalability problem was not as noticeable in our previous study involving a DAML+OIL benchmark [15]. We believe this is because the current benchmark ontology replaced all occurrences of *daml:domain* with *rdfs:domain*. Unlike *daml:domain*, the use of *rdfs:domain* triggers additional forward-chaining inferences in Sesame.

3.3.2. Query Response Time

Readers are referred to Appendix 2 for a complete list of query test results including query response time, number of answers, and query completeness. Fig. 5 and Fig. 6 compares using graphs the query response time of the systems from two different views. Fig. 5 compares the performance of all the queries with respect to each dataset while Fig. 6 compares the query response time across all the datasets with respect to each query. Note, in Fig. 5 we use an exponential scale and have reordered the queries to facilitate presentation. In general, the faster queries are to the left, while the slower ones are to the right.

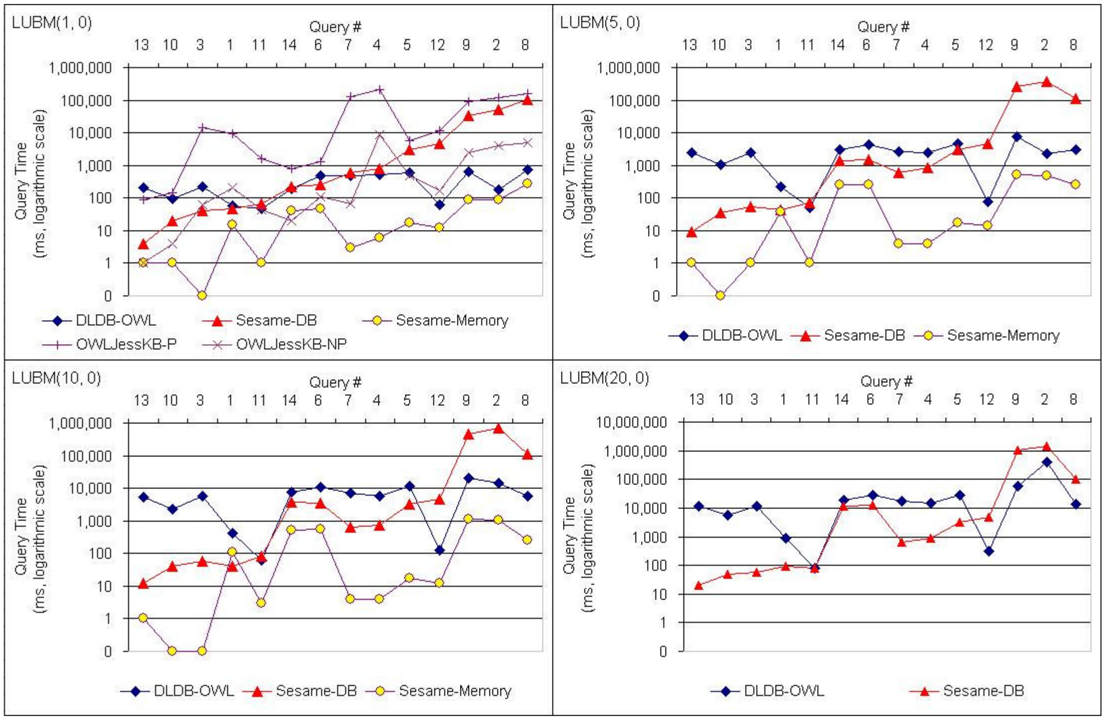

Fig. 5. Query response time comparison with respect to each dataset (up to 20 universities)

In terms of query, the results also lead to some scalability and efficiency concerns. Sesame-DB was very slow in answering some queries (even for one university), including Queries 2, 8, and 9. As for DLDB-OWL, it is the only system that has been tested with the largest dataset. But one concern is that when it comes to the larger datasets, especially the 50-university set, DLDB-OWL's query time no longer appears linear for

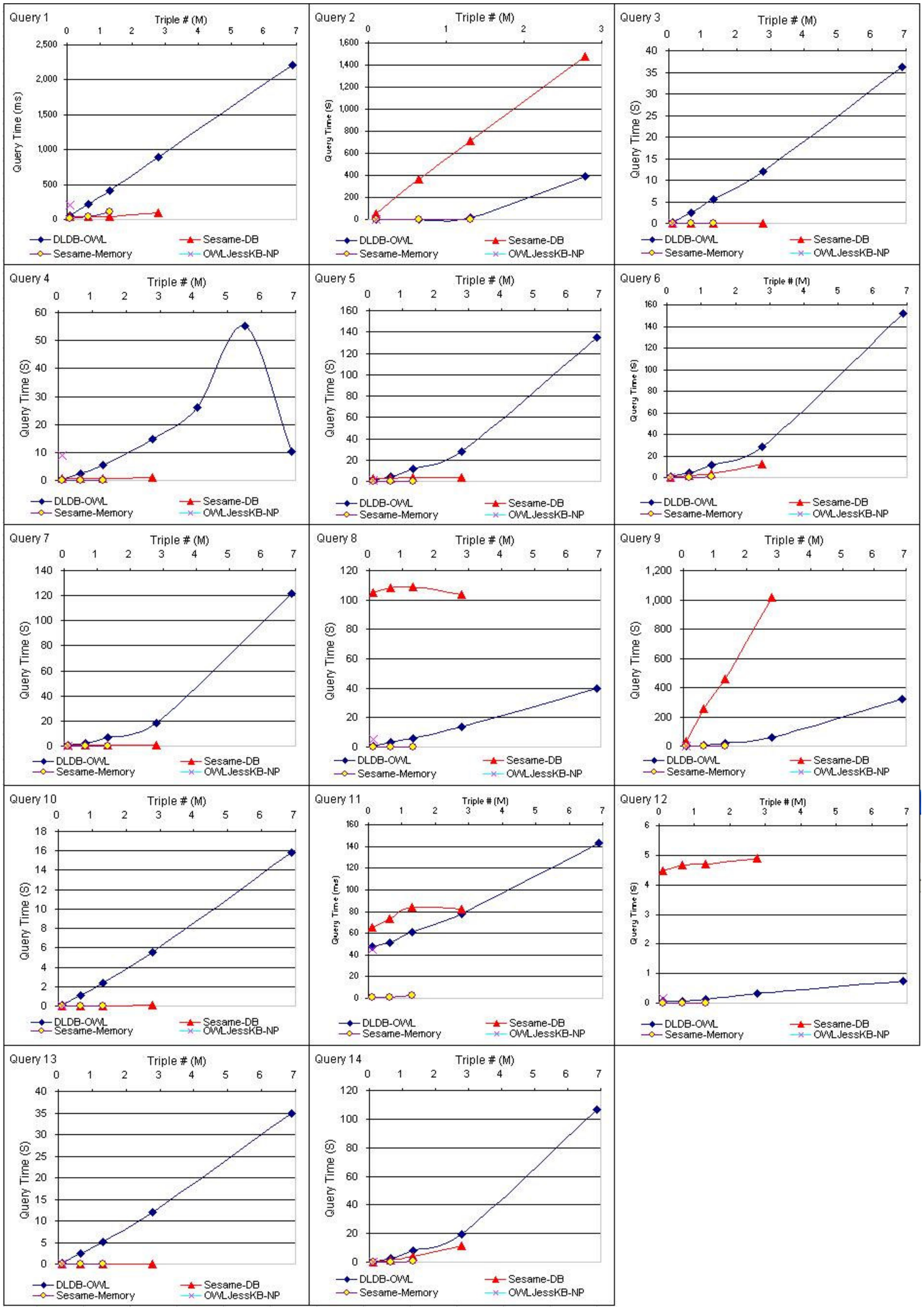

Fig. 6. Query response time comparison between DLDB-OWL, Sesame-DB, Sesame-Memory, and OWLJessKB-NP with respect to each query

some queries, i.e., Queries 2, 5, 6, 7, 9, and 14. Moreover, it failed to answer Query 2 on the 50-univeristy dataset after MS Access ran out of temporary space. Regarding OWLJessKB, compared to the performance of its predecessor DAMLJessKB in [15], OWLJessKB improves its query time greatly at the sacrifice of much longer load time. Nonetheless, when OWLJessKB is queried with pre-defined patterns it is still the slowest in answering thirteen of the queries. It responds to the queries much faster when such patterns are not used, however is still outperformed by one of the database-based systems for quite a few queries. Compared to other systems, Sesame-Memory is the fastest in answering almost all the queries. It is also the fastest in data loading. This suggests that it might be the best choice for data of small scale if persistent storage and OWL inference is not required.

We have observed that those queries for which Sesame-DB's performance goes down dramatically are common in that they do not contain a specific URI as a subject or object in the statements. On the other hand, Sesame-DB shows a nice property in answering some other queries like Queries 3, 4, 5, 7, and 8: there was no proportional increase in the response time as the data size grows. We have also noticed a common feature of these queries, i.e., they have constant number of results over the test datasets. Whether these are the causes or coincidences is a subject for future work.

Another thing that has drawn our attention is that, unlike the other queries, DLDB responds to Query 4 on the 50-university dataset even faster than on the 20-university set. We have redone the test several times and confirmed that the phenomenon is repeatable. We have ruled out the following possible causes for this behavior: the query does not terminate due to an error; the result set is not incorrect; there are no outliers in the individual runs of the query; and there is no integer overflow in our measurement of query response time. In addition, we have run the query on another two datasets consisting of 30 universities and 40 universities respectively (we have added these two data points to the graph for Query 4 in Fig. 6). It turned out that query time monotonically increases for these two datasets, which makes the 50-university evaluation appear to be an even stranger outlier. We currently suspect the phenomenon is due to some optimization performed by MS Access. We intend to do a further investigation in the future.

This initial research leads to a number of topics for future investigation. One is to explore the potential relationship between query types and performance of the system. As shown here, different systems seem to be optimized for different types of queries. Is it possible to develop a system that efficiently support all queries? As a related issues, what are the major categories of queries? Finally, what is the best design for interacting with a DBMS. Sesame-DB implements the main bulk of the evaluation in its RQL query engine, but has another query engine for the language SeRQL that pushes a lot of the work down to the underlying DBMS. DLDB-OWL directly translates as much of the query for the database. Further work should be done to investigate how these design differences as well as the underlying DBMS used impact performance.

3.3.3. Query Completeness and Soundness

It was noted before that we have chosen the benchmark test queries according to several criteria. In addition, we have made effort to make those queries as realistic as possible.

In other words, we want these queries to represent, to some extent, those in the real world. We are very interested in seeing what queries can be answered by each system.

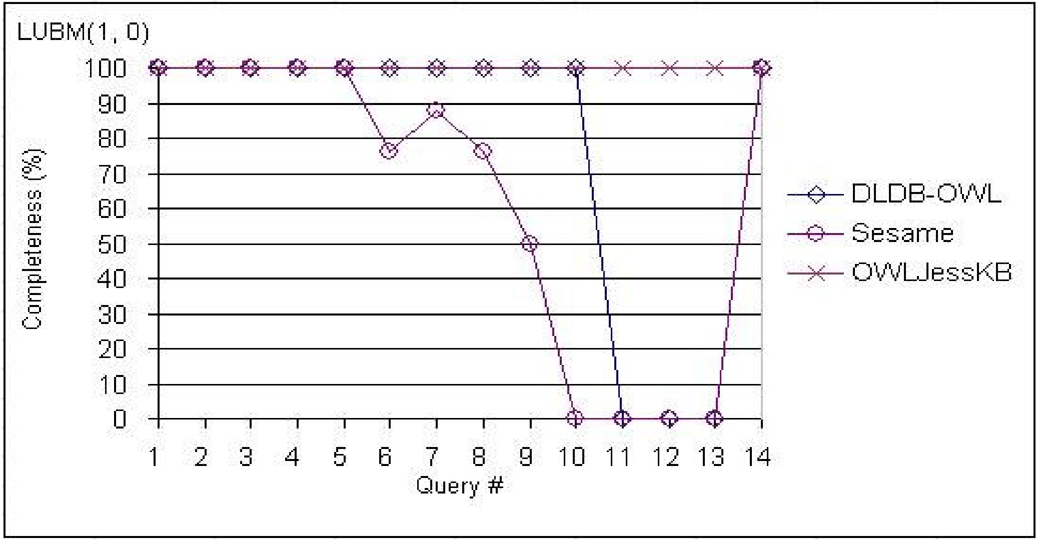

Fig. 7. Query completeness comparison. We show the results of only the first dataset since there are only minor or no differences between the five datasets.

Fig. 7 depicts the comparison of query completeness between the systems. As mentioned, Sesame is able to address RDF/RDFS semantics while DLDB-OWL and OWLJessKB integrate extra OWL inference capability. As the results turned out, all systems could answer Queries 1 through 5 and Query 14 completely. As we expected, DLDB-OWL was able to find all the answers for Queries 6 to 10, which requires subsumption inference in order to get complete results, while Sesame could only find partial or no answers for them. It is interesting to notice that DLDB-OWL and Sesame found complete answers for Query 5 in different ways: DLDB-OWL made use of subsumption, while Sesame, although not able to figure out the subsumption, used an *rdfs:domain* restriction to determine the types of the individuals in the dataset and thus achieved the same result. OWLJessKB could find all the answers for every query, and was the only system to answer Queries 11 and 13 completely, which assume *owl:TransitiveProperty* and *owl:inverseOf* inference respectively. Nevertheless, we have discovered that OWLJessKB made unsound inferences with respect to some queries. Specifically, it returned incorrect answers to Queries 4, 6, 8, and 12 because it incorrectly inferred that Lecturer is a Professor, Employee a Student, and Student a Chair. Our investigation has shown that this is due to OWLJessKB's incorrect handling of *intersectionOf*. We list in Table 2 the soundness of OWLJessKB for each query.

Table 2. Query soundness of OWLJessKB.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Correct answers /Total answers	4/4	0/0	6/6	34/41	719/719	7790/8330	67/67	7790/8330	208/208	4/4	224/224	15/540	1/1	5916/5916
Soundness	100	100	100	83	100	94	100	94	100	100	100	3	100	100

3.3.4. Combined Metric Values

We have calculated the combined metric value (cf. Section 2.4) of each target system with respect to each dataset. In the evaluation, we set both β and α to 1, which means we equally weight query completeness against soundness, and query response time against completeness and soundness as well. In addition, we equally weight the queries too. Fig. 8 shows the results.

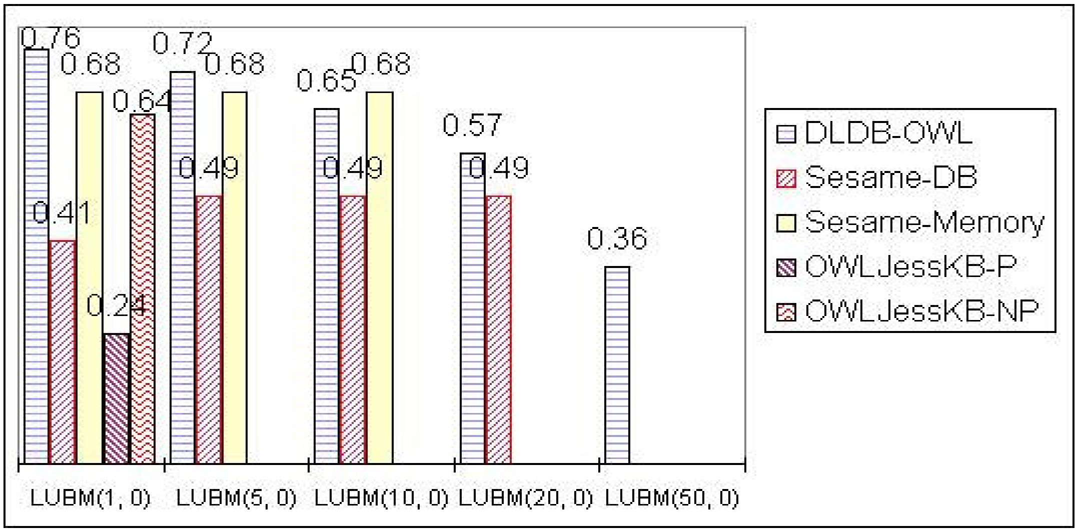

Fig. 8. CM values (*a*=500, *b*=5, *α*=1, *β*=1, $w_i=w_j$ (i,j=1,…,14))

We find that these numerical results are helpful for us to appreciate the overall query performance of each system. The higher values of Sesame-Memory than Sesame-DB for the first three datasets and than DLDB-OWL for the 10-university set again suggest that it is a reasonable choice for small scale application if persistent storage is not required, particularly if completeness is not significant. DLDB-OWL achieves higher scores across all the datasets than the other database-based system Sesame-DB. This helps us believe that its extra inference capability is not counterproductive. Although DLDB-OWL is the only system that could handle the largest dataset, the relatively low value it gets for that dataset reveals room for improvement. OWLJessKB-NP receives a comparable value to Sesame-Memory for the smallest dataset. However, the extremely long load time of OWLJessKB and its failure to load larger datasets emphasize the need for performance improvement in that regard. Moreover, the significant difference in scores for OWLJessKB-P and OWLJessKB-NP suggests the necessity of a set of query writing guidelines for that system.

4. Related Work

To the best of our knowledge, the Lehigh University Benchmark is the first benchmark for expressive Semantic Web knowledge base systems. Magkanaraki et al. [25] have conducted a statistical analysis about the size and morphology of RDF schemata. However their work does not provide a benchmark for evaluating a repository. Alexaki et al.

[2] have developed some benchmark queries for RDF, however, these are mostly intensional queries, while our work is concerned with extensional queries for OWL.

Tempich and Volz [34] have done some preliminary work towards a benchmark for Semantic Web reasoners. Though their benchmark is still under construction, they analyze the publicly available ontologies and report them to be clustered into three categories. According to the characteristics of each category, our Univ-Bench ontology happens to be a synthetic "description logic-style" ontology, which has a moderate number of classes but several restrictions and properties per class. Therefore we argue that our evaluation represents at least a considerable portion of the real world situations. The other two categories are terminological ontologies and database schema-like ontologies. We are currently working on extending our benchmark suite to those two categories.

García-Castro and Gómez-Pérez [12] describe a benchmarking methodology for the WebODE ontology engineering tool. They generated workloads in different sizes and tested the performances of the API's method calls. Although their work is very useful in evaluating ontological tools (e.g., editors, alignment tools, etc.), it provides less information on how to benchmark KBSs with respect to extensional queries against large scale instance data. Therefore, our work can be seen as complementary to theirs.

Maynard et al. [26] details the goals, requirements and processes of benchmarking ontology tools. It provides a good account of how to organize, implement and report a benchmark suite from the perspectives of software engineering and project management. When describing the high level requirements in evaluating ontology-based reasoning tools, they identified two use-cases which have already been covered in the LUBM: concept reasoning and instance retrieval. While this report can serve as a general guideline for building benchmark suite, our benchmark can be seen as an implementation that follows those guidelines with emphasis on the issue of scalability.

Our work borrows heavily from existing database benchmarks, including the Wisconsin benchmark [3, 4], the OO1 benchmark [8], and the SEQUOIA 2000 benchmark [33]. They are all DBMS-oriented benchmarks and storage benchmarks (vs. visualization benchmarks). The LUBM shares in spirit with them methodology and rationale in terms of the use of synthetic data, some criteria for choosing test queries, and three of the performance metrics. However, our benchmark is tailored to the evaluation of OWL knowledge base systems and thus has many unique features. Particularly as shown in the previous sections, the benchmark contains an ontology, datasets, test queries and criteria that reflect special concepts, structures and concerns in the Semantic Web area such as classes and properties, logical completeness vs. system performance, etc. Moreover, our benchmark is intended to work with any OWL repository, not just database systems.

Some attempts have been made by Elhaik et al. [11] and Horrocks and Patel-Schneider [21] to benchmark description logic systems. The emphasis of this work is to evaluate the reasoning algorithms in terms of the tradeoff between expressiveness and tractability in description logic. Our benchmark is not a description logic benchmark. We are more concerned about the issue of storing and querying large amounts of data that are created for realistic Semantic Web systems. In [11] and [21], the benchmark data are TBoxes and/or ABoxes, which can essentially be viewed as the counterparts of the ontology and the datasets in our benchmark respectively. In [21] they use both artificial and realistic TBoxes and use synthetic ABoxes. But the ABoxes in the test are of fixed sizes. In contrast, our benchmark data can scale to arbitrary size. The ABox is randomly generated in [11]. However, unlike our benchmark data, the ABox is not customizable and repeatable. They also generate the TBox randomly while our benchmark is based on a realistic ontology.

The W3C Web Ontology Working Group provides a set of OWL test cases [7]. They are intended to provide examples for, and clarification of, the normative definition of OWL and focus on the completeness and soundness with respect to individual features. Our benchmark complements these tests. While these tests determine the capability and correctness of a system, our benchmark evaluates the performance of the system from different perspectives such as data loading, extensional queries and scalability.

5. Applying the Benchmark

It should be pointed out that we believe the performance of any given system will vary depending on the ontology and data used to evaluate it. Thus the LUBM is not meant to be an overall Semantic Web KBS benchmark. It is a benchmark limited to a particular domain represented by the ontology it uses. Although as we argued in the previous section, the ontology could represent the characteristics of a considerable portion of real domains, the benchmark does not guarantee the same results for a different domain. For instance, with an RDF/RDFS-style ontology and instance data, we expect that there would be some different results for Sesame than that of this experiment. Therefore, we strongly suggest that the user who is considering using the benchmark first assess the benchmark ontology to see if it is close enough to his operational domain.

In addition, we encourage the benchmark user to evaluate the systems by looking at all the details of the benchmark results with respect to every performance metric. Especially, the combined metric is only intended to be used for a rough guideline, e.g. to sort the systems under test into tiers. It should not be considered as a replacement for the other performance metrics. Moreover, we remind the user to make use of the weights in the combined metric to reflect his emphasis on different aspects. To give readers a flavor of this, suppose we want to put more emphasis on RDFS queries. To do this, we could adjust the weights given to the benchmark queries. Here, we show an example of the extreme case: we equally weight the queries that do not require any inference or only require RDFS inference (i.e. Queries 1-5, 14) and we assign zero to the weights of the other queries that require extra OWL reasoning. Fig. 9 shows the result. As can be seen, the new result is quite different from that in Fig. 8, especially in the comparison between Sesame and DLDB-OWL.[7]

[7] Note that the actual CM values Sesame-Memory received were very close to 1. Numbers shown in the figure are rounded off to the two decimal places.

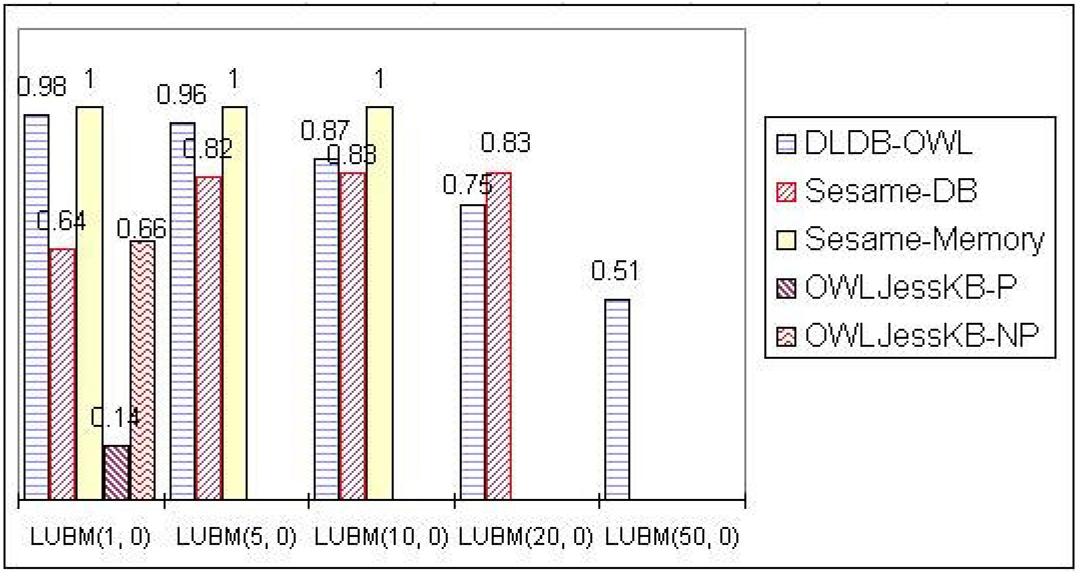

Fig. 9. *CM* values (*a*=500, *b*=5, *α*=1, *β*=1, only Queries 1-5, 14 are considered)

6. Conclusions and Future Work

We presented our work on evaluating knowledge base systems (KBS) with respect to use in large OWL applications. We have developed the Lehigh University Benchmark (LUBM) as the first in an eventual suite of benchmarks that would standardize and facilitate such evaluations. In the LUBM, the Univ-Bench ontology models the university domain in the OWL language and offers necessary features for evaluation purposes. The OWL datasets are synthetically created over the ontology. The data generated are random and repeatable, and can scale to an arbitrary size. Fourteen test queries are chosen to represent a variety of properties including input size, selectivity, complexity, assumed hierarchy information, assumed logical inference, amongst others. A set of performance metrics are provided, which include load time and repository size, query response time, query completeness and soundness, and a combined metric for evaluating the overall query performance. The LUBM is intended to be used to evaluate Semantic Web KBSs with respect to extensional queries over a large dataset that commits to a single realistic ontology.

We have demonstrated how the LUBM can be used to conduct an evaluation of four very different systems, including two memory–based systems (OWLJessKB and memory-based Sesame) and two systems with persistent storage (database-based Sesame and DLDB-OWL). We tested those systems with 5 sets of benchmark data. To our knowledge, no OWL experiment has been done with the scale of data used here. The smallest data size used consists of 15 OWL files totaling 8MB, while the largest data size consists of 999 files totaling 583MB.

This initial experiment had a number of interesting results. First, we discovered that the performance of OWLJessKB is extremely sensitive to the ordering of the query terms, and the system is unsound for OWL files that use *owl:intersectionOf*. Second, Sesame's load time increases exponentially with the size of data loaded. Third, as expected we showed that persistent storage systems could handle larger data sizes than memory-based systems, but we were surprised to discover that Sesame memory could handle up to 110 MB of input data. Finally, we found that DLDB-OWL and Sesame database were each better at different queries.

We caution against using the results of this experiment to make universal pronouncements about the relative quality of different KBSs. As stated earlier, we believe that different systems will be better suited for different applications. One must determine if the benchmark is similar enough to a desired workload in order to make predictions about how well it will perform in any given application. Still, we think we can extrapolate some general guidelines even from this simple experiment. Of the systems tested, DLDB-OWL is the best for large datasets where an equal emphasis is placed on query response time and completeness, however the system's performance declines as we approach datasets in the 500 MB range. Sesame-Memory is the best when the size is relatively small (e.g., 1 million triples) and only RDFS inference is required; while for a somewhat larger dataset (e.g., between 1 and 3 million triples), Sesame-DB may be a good alternative. OWLJessKB is the best for small datasets when OWL Lite reasoning is essential, but only after its unsoundness has been corrected. More detailed conclusions will require larger experiments that evaluate more systems and apply different benchmarks.

There are two logical next steps to this work. The first is to continue to develop different benchmarks that will evaluate different types of workloads. The benchmarks should contain ontologies that range in size and expressivity, have different profiles of instance data, and define different types of queries. Of course, determining the most appropriate set of benchmarks for the Semantic Web is still an open question; since the Semantic Web is still in its infancy, it is hard to predict how it will be used. A related question is how to rapidly generate scalable data that is representative of a given application. We are currently exploring an approach that learns a statistical model of a relatively small set of real-world data, and then uses this model to generate synthetic data. In addition to developing new benchmarks, we will apply the benchmarks to a more thorough set of systems. For example, it would be informative to compare DLDB-OWL to systems with similar capabilities. In particular, these systems include the BOR extension to Sesame (which integrates it with a DL reasoner), the KAON DLP extension (which transforms OWL ontologies into disjunctive logic programs), and the Instance Store [20] (another tool that combines a database and FaCT). Another useful evaluation would compare memory-based systems that support complex reasoning, including OWLIM (a Sesame SAIL that supports in-memory OWL Lite reasoning), Hoolet [36] (an OWL DL reasoner that uses the Vampire [29] theorem prover), and Jena with the OWL Lite rule set enabled. We believe that such investigations can drive research in practical, scalable reasoning systems, and that by examining the relative strengths and weaknesses of different systems, we will learn how to build a better OWL KBS.

Acknowledgements

We thank the anonymous reviewers of this paper for their careful reading and invaluable comments. In particular, we thank two reviewers for suggestions on how to design future benchmarks and on particular systems that we should evaluate.

Some of the material in this paper is based upon work supported by the Air Force Research Laboratory, Contract Number F30602-00-C-0188 and by the National Science Foundation (NSF) under Grant No. IIS-0346963. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or NSF.

Appendix 1: Test Queries

We herein describe each query in the SPARQL language. Following that we describe the characteristics of the query.

Query1

```
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}
```

This query bears large input and high selectivity. It queries about just one class and one property and does not assume any hierarchy information or inference.

Query2

```
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, ?Y, ?Z
WHERE
    {?X rdf:type ub:GraduateStudent .
     ?Y rdf:type ub:University .
     ?Z rdf:type ub:Department .
     ?X ub:memberOf ?Z .
     ?Z ub:subOrganizationOf ?Y .
     ?X ub:undergraduateDegreeFrom ?Y}
```

This query increases in complexity: 3 classes and 3 properties are involved. Additionally, there is a triangular pattern of relationships between the objects involved.

Query3

```
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:Publication .
     ?X ub:publicationAuthor
          http://www.Department0.University0.edu/AssistantProfessor0}
```

This query is similar to Query 1 but class Publication has a wide hierarchy.

Query4

```
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, ?Y1, ?Y2, ?Y3
WHERE
    {?X rdf:type ub:Professor .
     ?X ub:worksFor <http://www.Department0.University0.edu> .
     ?X ub:name ?Y1 .
     ?X ub:emailAddress ?Y2 .
     ?X ub:telephone ?Y3}
```

This query has small input and high selectivity. It assumes *subClassOf* relationship between Professor and its subclasses. Class Professor has a wide hierarchy. Another feature is that it queries about multiple properties of a single class.

Query5

```
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:Person .
     ?X ub:memberOf <http://www.Department0.University0.edu>}
```

This query assumes subClassOf relationship between Person and its subclasses and *subPropertyOf* relationship between memberOf and its subproperties. Moreover, class Person features a deep and wide hierarchy.

Query6

```
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:Student}
```

This query queries about only one class. But it assumes both the explicit *subClassOf* relationship between UndergraduateStudent and Student and the implicit one between GraduateStudent and Student. In addition, it has large input and low selectivity.

Query7

```
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, ?Y
```

```
WHERE
      {?X rdf:type ub:Student .
        ?Y rdf:type ub:Course .
        ?X ub:takesCourse ?Y .
        <http://www.Department0.University0.edu/AssociateProfessor0>,
            ub:teacherOf, ?Y}
```

This query is similar to Query 6 in terms of class Student but it increases in the number of classes and properties and its selectivity is high.

Query8

```
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, ?Y, ?Z
WHERE
      {?X rdf:type ub:Student .
        ?Y rdf:type ub:Department .
        ?X ub:memberOf ?Y .
        ?Y ub:subOrganizationOf <http://www.University0.edu> .
        ?X ub:emailAddress ?Z}
```

This query is further more complex than Query 7 by including one more property.

Query9

```
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, ?Y, ?Z
WHERE
      {?X rdf:type ub:Student .
        ?Y rdf:type ub:Faculty .
        ?Z rdf:type ub:Course .
        ?X ub:advisor ?Y .
        ?Y ub:teacherOf ?Z .
        ?X ub:takesCourse ?Z}
```

Besides the aforementioned features of class Student and the wide hierarchy of class Faculty, like Query 2, this query is characterized by the most classes and properties in the query set and there is a triangular pattern of relationships.

Query10

```
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:Student .
        ?X ub:takesCourse
            <http://www.Department0.University0.edu/GraduateCourse0>}
```

This query differs from Query 6, 7, 8 and 9 in that it only requires the (implicit) *subClassOf* relationship between GraduateStudent and Student, i.e., *subClassOf* relationship between UndergraduateStudent and Student does not add to the results.

Query11

```
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:ResearchGroup .
      ?X ub:subOrganizationOf <http://www.University0.edu>}
```

Query 11, 12 and 13 are intended to verify the presence of certain OWL reasoning capabilities in the system. In this query, property subOrganizationOf is defined as transitive. Since in the benchmark data, instances of ResearchGroup are stated as a sub-organization of a Department individual and the later suborganization of a University individual, inference about the subOrgnizationOf relationship between instances of ResearchGroup and University is required to answer this query. Additionally, its input is small.

Query12

```
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, ?Y
WHERE
     {?X rdf:type ub:Chair .
      ?Y rdf:type ub:Department .
      ?X ub:worksFor ?Y .
      ?Y ub:subOrganizationOf <http://www.University0.edu>}
```

The benchmark data do not produce any instances of class Chair. Instead, each Department individual is linked to the chair professor of that department by property headOf. Hence this query requires realization, i.e., inference that that professor is an instance of class Chair because he or she is the head of a department. Input of this query is small as well.

Query13

```
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:Person .
      <http://www.University0.edu> ub:hasAlumnus ?X}
```

Property hasAlumnus is defined in the benchmark ontology as the inverse of property degreeFrom, which has three subproperties: undergraduateDegreeFrom, mastersDegreeFrom, and doctoralDegreeFrom. The benchmark data state a person as an alumnus of a university using one of these three subproperties instead of hasAlumnus. Therefore, this query assumes *subPropertyOf* relationships between degreeFrom and its subproperties, and also requires inference about *inverseOf*.

Query14

```
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:UndergraduateStudent}
```

This query is the simplest in the test set. This query represents those with large input and low selectivity and does not assume any hierarchy information or inference.

Appendix 2: Query Test Results

Table 3 lists query test results including query response time, number of answers and completeness.

Appendix 3: Initial Test Results of Jena

Tables 4 to 6 show the initial test results of Jena (v2.1). We have tested Jena both based on the main memory (Jena-Memory) and using a MySQL database backend (Jena-DB). The benchmark queries were expressed in RDQL. We have tested Jena only with the smallest dataset. Unsurprisingly, when its RDFS reasoning was turned on, Jena's performance was exactly the same as Sesame's in terms of query completeness and soundness. However, Jena was much slower in answering most of the queries than Sesame. For some of the queries, Jena did not terminate even after being allowed to run for several hours. In this experiment we have used a timeout of 2 hours.

When Jena was used with its OWL inferencing, it could answer even smaller number of queries within the time limit. We speculate that the poor performance of Jena is because its rule-based reasoners are less optimized for a semantically complex ontology like Univ-Bench.

In order to investigate its query completeness and soundness with respect to the test queries, we have tested Jena with OWL reasoning on a single department file. This has allowed Jena to answer more queries within a reasonable time and noticeably, Jena could answer all those queries (including Queries 11-13) completely and correctly.

Table 3. Query test results [8]

Query	System & Dataset / Metrics	LUBM(1,0)					LUBM(5,0)			LUBM(10,0)			LUBM(20,0)		LUBM (50,0)
		DLDB-OWL	Sesame-DB	Sesame-Memory	OWL JessKB-P	OWL JessKB-NP	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	DLDB-OWL
1	Time(ms)	59	46	15	9203	200	226	43	37	412	40	106	887	96	2211
	Answers	4	4	4	4		4	4	4	4	4	4	4	4	4
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
2	Time(ms)	181	51878	87	116297	3978	2320	368423	495	14556	711678	1068	392392	1474664	failed
	Answers	0	0	0	0		9	9	9	28	28	28	59	59	-
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	-
3	Time(ms)	218	40	0	13990	164	2545	53	1	5540	59	0	11956	56	36160
	Answers	6	6	6	6		6	6	6	6	6	6	6	6	6
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
4	Time(ms)	506	768	6	211514	8929	2498	823	4	5615	762	4	14856	881	10115
	Answers	34	34	34	34*		34	34	34	34	34	34	34	34	34
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
5	Time(ms)	617	2945	17	5929	475	4642	3039	17	11511	3214	17	27756	3150	135055
	Answers	719	719	719	719		719	719	719	719	719	719	719	719	719
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
6	Time(ms)	481	253	48	1271	112	4365	1517	251	11158	3539	543	28448	12717	151904
	Answers	7790	5916	5916	7790*		48582	36682	36682	99566	75547	75547	210603	160120	519842
	Completeness	100	76	76	100		100	76	76	100	76	76	100	76	100
7	Time(ms)	478	603	3	128115	67	2639	606	4	7028	634	4	18073	657	121673
	Answers	67	59	59	67		67	59	59	67	59	59	67	59	67
	Completeness	100	88	88	100		100	88	88	100	88	88	100	88	100
8	Time(ms)	765	105026	273	164106	4953	3004	108384	262	5937	108851	264	13582	103779	39845
	Answers	7790	5916	5916	7790*		7790	5916	5916	7790	5916	5916	7790	5916	7790
	Completeness	100	76	76	100		100	76	76	100	76	76	100	76	100
9	Time(ms)	634	34034	89	87475	2525	7751	256770	534	19971	460267	1123	57046	1013951	323579
	Answers	208	103	103	208		1245	600	600	2540	1233	1233	5479	2637	13639
	Completeness	100	50	50	100		100	48	48	100	49	49	100	48	100
10	Time(ms)	98	20	1	141	4	1051	36	0	2339	40	0	5539	50	15831
	Answers	4	0	0	4		4	0	0	4	0	0	4	0	4
	Completeness	100	0	0	100		100	0	0	100	0	0	100	0	100
11	Time(ms)	48	65	1	1592	45	51	73	1	61	84	3	78	82	143
	Answers	0	0	0	224		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
12	Time(ms)	62	4484	12	11266	162	78	4659	14	123	4703	12	310	4886	745
	Answers	0	0	0	15*		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
13	Time(ms)	200	4	1	90	1	2389	9	1	5173	12	1	11906	21	34854
	Answers	0	0	0	1		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
14	Time(ms)	187	218	42	811	20	2937	1398	257	7870	3831[9]	515	19424	11175	106764
	Answers	5916	5916	5916	5916		36682	36682	36682	75547	75547	75547	160120	160120	393730
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100

[8] The numbers marked with * do not count any incorrect answers returned by the system (refer to Table 2)

[9] This is an adjusted value from the original experiment [16], in which the query time was much longer. This was because the OS was increasing virtual memory at the time of the query. We have updated the result without the affect of that operation.

Table 4. Load time of Jena

	Dataset	Load Time (hh:mm:ss)
Jena-Memory (RDFS reasoning)	LUBM (1, 0)	00:00:12
Jena-DB (RDFS reasoning)		00:30:45
Jena-Memory (OWL reasoning)		00:00:13
Jena-DB (OWL reasoning)		00:32:27

Table 5. Query response time of Jena.

Query	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Jena-Memory (RDFS)	160	time-out	215	51	585	215	272951	time-out	time-out	209	14	4	203	220
Jena-DB (RDFS)	5715	time-out	13110	2860	24356	479	timeout	time-out	time-out	11562	536	1048	11731	2095
Jena-Memory (OWL)	time-out	time-out	3929	time-out	time-out	time-out	timeout	time-out	time-out	timeout	time-out	timeout	timeout	251
Jena-DB (OWL)	time-out	time-out	52818	time-out	time-out	time-out	timeout	time-out	time-out	timeout	time-out	timeout	timeout	2289

Table 6. Query completeness and soundness of Jena.

Query		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Jena (RDFS, one university)	Completeness	100	n/a	100	100	100	76	88	n/a	n/a	0	0	0	0	100
	Soundness	100		100	100	100	100	100			100	100	100	100	100
Jena (RDFS, one department)	Completeness	100	100	100	100	100	78	88	78	38	0	0	0	0	100
	Soundness	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Jena (OWL, one department)	Completeness	100	n/a	100	100	100	100	n/a	n/a	n/a	n/a	100	100	100	100
	Soundness	100		100	100	100	100					100	100	100	100

n/a: not applicable due to timeout

References

[1] S. Alexaki et al. The RDFSuite: Managing Voluminous RDF Description Bases. In Proc. of the 2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with the Tenth International World Wide Web Conference (WWW10), 2001.

[2] S. Alexaki et al. On Storing Voluminous RDF Description: The case of Web Portal Catalogs. In Proc. of the 4th International Workshop on the Web and Databases, 2001.

[3] D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking Database Systems, a Systematic Approach. In Proc. of the 9th International Conference on Very Large Data Bases, 1983.

[4] D. Bitton. and C. Turbyfill. A Retrospective on the Wisconsin Benchmark. In Readings in Database Systems, Second Edition, Morgan Kaufman 1994, pp180-299.

[5] J. Broekstra and A. Kampman. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proc. of the 1st International Semantic Web Conference (ISWC2002), 2002.

[6] J. Broekstra and A. Kampman. Inferencing and Truth Maintenance in RDF Schema: exploring a naive practical approach. In Workshop on Practical and Scalable Semantic Systems (PSSS), 2003.

[7] J.J. Carroll and J.D. Roo ed. OWL Web Ontology Test Cases, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-owl-test-20040210/

[8] R.G.G. Cattell. An Engineering Database Benchmark. In the Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufman 1991, pp247-281.

[9] D. Connolly et al. DAML+OIL (March 2001) Reference Description. http://www.w3.org/TR/daml+oil-reference

[10] M. Dean and G. Schreiber ed. OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[11] Q. Elhaik, M-C Rousset, and B. Ycart. Generating Random Benchmarks for Description Logics. In Proc. of the 1998 Description Logic Workshop (DL'98), 1998.

[12] R. García-Castro and A. Gómez-Pérez. A Benchmark Suite for Evaluating the Performance of the WebODE Ontology Engineering Platform. In Proc. of the 3rd International Workshop on Evaluation of Ontology-based Tools, 2004.

[13] M. Gensereth and R. Fikes. Knowledge Interchange Format. Stanford Logic Report Logic-92-1, Stanford Univ. http://logic.standford.edu/kif/kif.html

[14] Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL Repositories. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), 2003.

[15] Y. Guo, Z. Pan, and J. Heflin. Choosing the Best Knowledge Base System for Large Semantic Web Applications. In Proc. of the 13th International World Wide Web Conference (WWW2004) - Alternate Track Papers & Posters, 2004.

[16] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In Proc. of the 3rd International Semantic Web Conference (ISWC2004), 2004.

[17] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. In Workshop on Evaluation on Ontology-based Tools, the 2nd International Semantic Web (ISWC2003), 2003.

[18] V. Haarslev, R. Möller, and M. Wessel. Querying the Semantic Web with Racer + nRQL. In Proc. of the Workshop on Description Logics 2004 (ADL2004).

[19] I. Horrocks. The FaCT System. In Automated Reasoning with Analytic Tableaux and Related Methods International Conference (Tableaux'98), 1998.

[20] I. Horrocks et al. The instance store: DL reasoning with large numbers of individuals. In Proc. of the 2004 Description Logic Workshop (DL2004), 2004.

[21] I. Horrocks and P. Patel-Schneider. DL Systems Comparison. In Proc. of the 1998 Description Logic Workshop (DL'98), 1998.

[22] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In Proc. of the Eleventh International World Wide Web Conference (WWW'02), 2002.

[23] J.B. Kopena and W.C. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), 2003.

[24] D.D. Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. SIGIR Forum, 29(2), 13-19, 1995.

[25] A. Magkanaraki et al. Benchmarking RDF schemas for the Semantic Web. In Proc. of the 1st International Semantic Web Conference (ISWC2002), 2002.

[26] D. Maynard et al. D2.1.4: Definition of a Methodology, General Criteria, and Benchmark Suites for Benchmarking Ontology Tools. EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB, 2005.

[27] Z. Pan and J. Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In Workshop on Practical and Scalable Semantic Systems, the 2nd International Semantic Web Conference (ISWC2003), 2003.

[28] E. Prud'hommeaux and A. Seaborne ed. SPARQL Query Language for RDF, W3C Working Draft 19 April 2005. http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050419/

[29] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. AI Communications, 15(2-3):91–110, 2002.

[30] C. J. van Rijsbergen. Information Retireval. Butterworths, London, 1979.

[31] A. Seaborne. RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[32] M.K. Smith, C. Welty, and D.L. McGuinness ed. OWL Web Ontology Language Guide, W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[33] M. Stonebraker et al. The SEQUIOA 2000 Storage Benchmark. In Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, 1993.

[34] C. Tempich and R. Volz. Towards a benchmark for Semantic Web reasoners–an analysis of the DAML ontology library. In Workshop on Evaluation on Ontology-based Tools, the 2nd International Semantic Web Conference (ISWC2003), 2003.

[35] FaCT++. http://owl.man.ac.uk/factplusplus/

[36] Hoolet. http://owl.man.ac.uk/hoolet/

[37] Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net/

[38] Jess: the Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/jess

[39] KAON: The KArlsruhe ONtology and Semantic Web tool suite. http://kaon.semanticweb.org/

[40] Kowari. http://www.kowari.org/

[41] OWLJessKB: A Semantic Web Reasoning Tool. http://edge.cs.drexel.edu/assemblies/software/owljesskb/

[42] Pellet OWL Reasoner. http://www.mindswap.org/2003/pellet/

[43] PostgreSQL. http://www.postgresql.org

[44] Resource Description Framework (RDF). http://www.w3.org/RDF/

[45] The SeRQL query language. http://www.openrdf.org/doc/users/ch06.html