OWLIM – a Pragmatic Semantic Repository for OWL

Atanas Kiryakov[1], Damyan Ognyanov[1], Dimitar Manov[1]

[1] Ontotext Lab, Sirma Group Corp.
135 Tsarigradsko Chaussee, Sofia 1784, Bulgaria
{naso,damyan,mitac}@sirma.bg
http://www.ontotext.com

Abstract. OWLIM is a high-performance Storage and Inference Layer (SAIL) for Sesame, which performs OWL DLP reasoning, based on forward-chaining of entilement rules. The reasoning and query evaluation are performed in-memory, while in the same time OWLIM provides a reliable persistence, based on N-Triples files. This paper presents OWLIM, together with an evaluation of its scalability over synthetic, but realistic, dataset encoded with respect to PROTON ontology. The experiment demonstrates that OWLIM can scale to millions of statements even on commodity desktop hardware. On an almost-entry-level server, OWLIM can manage a knowledge base of 10 million explicit statements, which are extended to about 19 millions after forward chaining. The upload and storage speed is about 3,000 statement/sec. at the maximal size of the repository, but it starts at more than 18,000 (for a small repository) and slows down smoothly. As it can be expected for such an inference strategy, delete operations are expensive, taking as much as few minutes. In the same time, a variety of queries can be evaluated within milliseconds. The experiment shows that such reasoners can be efficient for very big knowledge bases, in scenarios when delete operations should not be handled in real-time.

1 Introduction

The Semantic Web requires scalable high-performance storage and reasoning infrastructure in order to mach the expectations for a hype of ontologies and structured metadata. The major challenge towards building such infrastructure is the expressivity of the underlying standards: RDF(S), [6], and OWL, [2]. Although RDF(S) is a simple Knowledge Representation (KR) language, it is already a challenging task to implement a repository for it, which provides performance and scalability comparable even to entry-level relational database management systems (RDBMS). Going up the stairs of the Semantic Web stack, the challenges for the repository engineers are getting more and more serious. Even the simplest dialect of OWL (Lite) is a description logic (DL) with no obvious algorithms allowing for efficient inference and query answering over reasonably scaled knowledge bases (KB).

Logical programming (LP) is a common name used for rule-based logical dialects and systems, such as PROLOG, Datalog, and Flora. OWL DLP is emerging as a new dialect, offering a promising compromise between expressive power, efficient reason-

ing, and compatibility. It is defined in [3] as the intersection of the expressivity of OWL DL and LP, which is more clearly layered on top of RDFS.

The two principle strategies for rule-based inference are, as follows:

- **Forward-chaining**: to start from the known facts and to perform inference in an inductive fashion. The goals could be different: to answer a particular query; to infer a particular sort of knowledge (e.g. the class taxonomy).
- **Backward-chaining:** to start from a particular fact or a query and to verify it or get all possible results, using deductive reasoning. In essence, the reasoner decomposes the query or the fact into alternative or simpler facts, which are available in the KB or can be further, recursively, transformed.

Let us imagine a repository, which performs total forward-chaining; after each update to the KB, the *inferred closure*[1] is computed and made available for query evaluation or retrieval. This strategy is known as *materialization.* The advantages and disadvantages of this approach are discussed at length in [1].

This paper presents the implementation of OWLIM – a semantic repository, based on full materialization, providing support for a fraction of OWL, close to OWL DLP. We present a benchmark for examining the performance of the repository, report the results, and discuss the advantages and disadvantages of the approach taken.

Section 2 presents OWLIM with its specifics, optimizations, and limitations. Section 3 presents the ontology and the dataset that we use as a basis for the benchmark, described in section 4. The fifth section presents the results from the experiment; section 6 comments on related work and section 7 concludes the paper.

2 OWLIM

OWLIM is the short name of the `OWLMemSchemaRepository` SAIL (Storage and Inference Layer) for Sesame[2], which supports partial reasoning over OWL DLP. It is an in-memory reasoning implementation; the latter means that the full content of the repository is loaded and maintained in the main memory, which allows for efficient retrieval and query answering. Although the reasoning is handled in-memory, this SAIL offers a relatively comprehensive persistency and backup strategy.

Technically, OWLIM v.2.0 is an extension of the `RdfSchemaRepository` SAIL of Sesame v.1.2.1. Thus, the results reported represent an evaluation of a tuning of Sesame and can be considered indicative for its robustness. More information related to various aspects of its specification, architecture, and implementations can be found in [1]. The modifications in OWLIM can be summarized as follows:

- The set of entilement rules supports some OWL primitives (see section 2.1);
- Concurrent multi-thread inference: it delivers serious improvements of the inference speed for machines with multiple processors or Hyper-Threading;
- The persistence implementation is derived from `RdfSchemaRepositoryV2` (underlying OWLIM v.1.0). It was further optimized, as a serious performance bottleneck in the file operations was removed.

[1] "Inferred closure" is the extension of a KB with all the facts which could be inferred from it.

[2] One of the most popular Semantic Web repositories, http://www.openrdf.org, [1].

- Optimizations speeding-up the delete operation – a number of improvements were made to the implementation of `RdfSchemaRepository`.

Along with the SAIL, the distribution of OWLIM also contains a custom RMI factory, which allows the remote access to the SAIL layer. OWLIM, together with its documentation can be downloaded from http://www.ontotext.com/owlim.

2.1 Reasoning and Language Support

OWLIM uses for reasoning the in-memory rule entilement engine of `RdfSchemaRepository`. The engine can be configured with a set of inference rules, which determines the supported semantic. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are RDF statements, which can contain free variables. The rule head contains one or more consequences, each of which is an RDF statement, without free variables, not already introduced in the body.

As a basis, `RdfSchemaRepository` is configured with a set of rules which cover the model-theoretic semantics of RDFS, as defined in [5]. More details about the implementation and motivation can be found in [1]. OWLIM extends these rules with a set, which support the following OWL constructs: `SymmetricProperty`, `TransitiveProperty`, `inverseOf`, `equivalentProperty`, `sameAs`, `FunctionalProperty`, `InverseFunctionalProperty`. The full list of the axioms can be found in the OWLIM documentation.

We are currently investigating the extension of the set of rules to make possible full support of OWL DLP. Extensions towards more expressive LP fragments are straightforward, taking the rule-based inference engine used.

2.2 Persistence

The persistency of OWLIM is implemented through N-Triples files. The repository can be spread into several files. All but one of these files are considered read-only; there is a single file (let us name it `persist`) that is considered both an input for loading and a target, where new statements are stored.

The backup strategy implemented, ensures that no loss of newly asserted triples can occur in cases of power failure or abnormal termination – the detailed description is presented in OWLIM's documentation. Although relatively simple, this strategy had proven to be very efficient and reliable, through the couple of years for which `RdfSchemaRepositoryV2` and OWLIM has been used as a semantic repository for different applications of the KIM platform, http://www.ontotext.com/kim.

2.3 Limitations

The limitations of OWLIM are related to its reasoning strategy. In general, the expressivity of the language supported cannot be extended in the direction of DL. The rule-engine behind OWLIM is limited in expressivity by the Horn logic.

The "total materialization" strategy has its obvious drawbacks, as discussed in [1] (section 6). For specific ontologies and KBs, the count of the implicit statements can appear to grow rapidly[3]. What is even more important, the delete operation is really slow, which means that OWLIM is not suitable for applications where removal of data is a typical transaction.

The most obvious disadvantage of the in-memory reasoning is that the size of the KB, which can be handled, is limited by the size of the available RAM. Considering the currently available commodity hardware, OWLIM can handle millions of statements on desktop machines and above ten millions on an almost-entry-level server.

3 Ontology and Dataset

We took the PROTON light-weight upper-level ontology as a basis for our experiment. It contains about 300 classes and 100 properties, providing coverage of the general concepts necessary for a wide range of tasks, with special focus on named entities and concrete domains (i.e. people, organizations, locations, numbers, dates, addresses). The ontology is encoded in a fragment of OWL DLP. It is split into four modules: System, Top, Upper, and KM. The PROTON ontology itself and related documents can be found at http://proton.semanticweb.org.

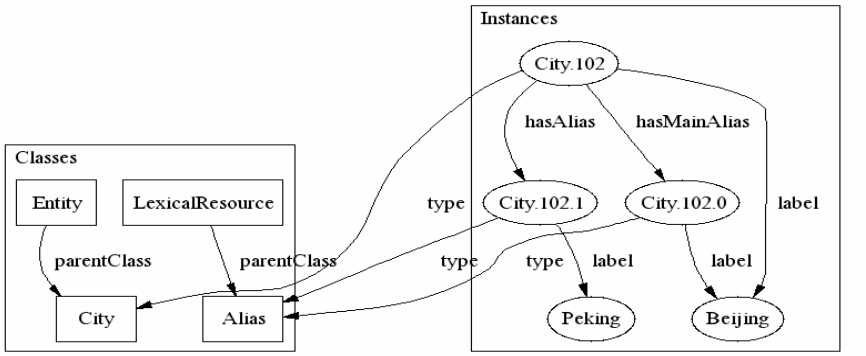


Fig. 1. Sample Representation of an Entity Description

PROTON is also heavily used within the KIM platform. As a start, part of KIM is the so-called World Knowledge Base (WKB), which consist of thousands of entity (instance) descriptions. Each entity is described by its most specific type, aliases, attributes (e.g. the latitude of a `Location`), and relations (e.g. `subRegionOf` of another `Location`). A simplified schema of the entity representation is demonstrated in Fig. 1. WKB is populated with entities of general importance, which serve as a seed for KIM to perform automatic semantic annotation of text and ontology

[3] Still, for many real-live scenarios the amount of implicit statements is comparable to this of the explicit ones – for instance KIM and the examples available in section 6 of [1].

population. The full variant of WKB consists of more than 200,000 entities, which have been gathered semi-automatically out of a big number of publicly available data-sources. The small variant of WKB, which is used here, contains about 40,000 entities, of which about 8,000 organizations (mostly big companies, with their industry codes, associated through the `activeInSector` property); 6,000 persons; 12,000 locations, including continents, global regions, and countries with their provinces, 4,400 cities, oceans, seas, etc. Each location has geographic coordinates and several aliases (including different language variants), as well as co-positioning relations (e.g. `subRegionOf`). More information about the compilation of WKB and its usage in KIM can be found at http://www.ontotext.com/kim/docs.html.

4 Benchmark organization

The experiment loads initially non-synthetic ontology and instance data. Then it starts uploading more quasi-real knowledge in transactions of about 10,000 statements each. Upload transactions are performed until either the target upload transactions count or the memory limit is reached. After each ten uploads we measure, as follows:

- The upload speed, in terms of explicit statements per second;
- The speed for evaluation of two queries. Each of these queries is evaluated 10 times, for the sake of accuracy. As long as OWLIM evaluates all the queries in-memory, caching effects cannot be expected.

Delete transactions are performed after each 100 upload transactions. We remove a single statement, the one stating the label of last created city. The important point here is that under the current implementation, any delete transaction causes invalidation of the inferred closure, so, full inference on top of the current content of the repository takes place. In other words, such transactions indicate the time for calculation of the full inferred closure, over repository of the corresponding size.

4.1 Initialization

The benchmark application starts OWLIM with all following files, given as "background" knowledge, loaded in the repository during initialization:

- `owl.rdfs` – the standard OWL schema;
- `protons.owl`, `protont.owl`, `protonu.owl`, `protonkm.owl` – the four modules of PROTON ontology. The namespace prefixes used below are respectively: `psys`, `ptop`, `pupp`, `protonkm`.;
- `kimso.owl` and `kimlo.owl` – a couple of small KIM specific ontologies, defining auxiliary primitives such as `hasAlias`;
- `wkb.nt` – the small version of KIM's WKB in N-Triples format, which accounts for most of the volume of the initial repository.

After loading the above files, OWLIM is in the state in which it is used in the KIM platform; this is to say we start with ontology and KB used in a real application. The explicit statements in the repository are about 0.5 millions – this figure can be seen as a starting point at the charts in section 5. The inferred closure adds about 0.5 million

statements – which means, that due to the total materialization strategy of OWLIM, it holds at this stage about 1 million statements in memory.

4.2 Upload Transactions

Each upload transaction adds one new city and a number of other entities related to it. We created a new instance of `pupp:City`, which is linked as sub-region of an arbitrarily chosen province from the WKB. Due to the fact that WKB covers all the provinces (or states) for most of the countries in the world, this strategy guarantees: (i) connectivity between the real KB and the synthetically generated one; (ii) some inference takes place, at least for the closure of the transitive sub-region relation, considering that provinces are "nested" in countries, which are nested in regions and continents; and (iii) good spread of the synthetically generated cities all over the globe.

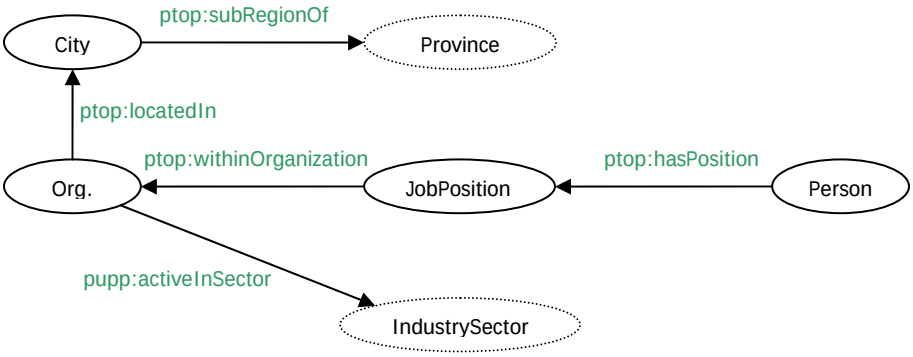


Fig. 2. Synthetic Description of City and Related Entities

Ten new organizations are created, and related to the synthetic city. The organizations are related to an arbitrarily selected industry sector from the WKB, in order to extend the connectivity of the synthetic data with the real KB. 38 new instances of `ptop:Person` are created and related for each organization (i.e. 380 per city) and related to it through positions. The pattern of entities created within a transaction can be seen at Fig. 2. Notice that the province and the industry sector are dashed to indicate that those are not newly created, but just randomly chosen from WKB. For each of the new entities, we generate their aliases (one of which is the main alias) and a label, in the manner depictured at Fig. 1.

The number of persons per organization is tuned so that the description of the city and its related entities consists of 10,033 statements. It turns out that the upload of a city description causes extension of the inference closure with about 9,000 implicit statements. Measuring the speed after each 10 upload transactions, means measuring the upload and storage of about 100,000 explicit statements and the inference of roughly the same amount of implicit ones.

4.3 Queries

We made two test queries, a simple and a more complex one. Both queries are encoded in SeRQL, which is a language developed for Sesame, as an improvement of RQL. A description of SeRQL can be found in section 3.5 of [1].

Query 1:

```
select OL,CL
from   {C} rdf:type {pupp:City},
       {C} ptop:subRegionOf {wkb:Continent_T.4},
       {O} ptop:locatedIn {C},
       {O} rdf:type {ptop:Organization},
       {C} rdfs:label {CL},
       {O} rdfs:label {OL}
```

Query 2:

```
select distinct PL,OL,SL,CL
from   {S} pupp:subSectorOf {wkb:SIC_H},
       {O} pupp:activeInSector {S},
       {O} ptop:locatedIn {C},
       {C} ptop:subRegionOf {wkb:Continent_T.4},
       {O} rdf:type {pupp:City},
       {Pos} ptop:withinOrganization {O},
       {Pos} ptop:holder {Pos},
       {Per} kimso:hasAlias {A} rdfs:label {AL},
       {C} rdfs:label {CL},
       {S} rdfs:label {SL},
       {Per} rdfs:label {PL},
       {O} rdfs:label {OL}
where  AL like "*son*" ignore case
```

The first query lists all the names of organizations in Europe (identified through its URI in WKB – `wkb:Continent_T.4`). It involves joining of a pattern of 6 statements. The results, indicate that the transitive closure of `ptop:subRegionOf` has been calculated, as there were not cities declared explicitly to be part of Europe.

The second query returns the list of all people having "son" as part of one of their aliases and working for organization in Europe, which are active in sub-sector of "Finance, Insurance, And Real Estate" (identified in WKB with `wkb:SIC_H`). This query involves a pattern of 12 statements and a literal constraint, which does not allow for easy indexing and optimization (without a full-text search system built in). In addition to what is already checked with Query 1, the results of the second query indicate that `pupp:subSectorOf` is closed properly (usually, the organizations are active in a more specific sector below `wkb:SIC_H`). It also checks the inference of `ptop:holder` on the basis of its inverse property `ptop:hasPosition`.

The number of results returned by both queries grows proportionally to the size the repository. For instance, the number of results returned after adding 200 cities (i.e. for a repository of size 2.5 million statements) is respectively 1042 for Query 1 and 3714 for Query 2. Considering that the benchmark application fetches all the results from the repository and the cardinality of the results, this test provides also an indication for the speed of fetching results.

4.4 Test Hardware

We had measured the performance of the benchmark application on equipment of different scale and specificity, as presented in Table 1. Here follow some comments on the configurations:

- 2Opt5GB and 2Opt3GB are servers in the price range 2000-6000 EURO;
- 2Xeon1GB is a high-end workstation, PM512MB is a notebook;
- The amount of RAM indicated in the tables is the RAM given as maximum heap constraint to the Java virtual machine (JVM); all the machines have more physical memory than this.

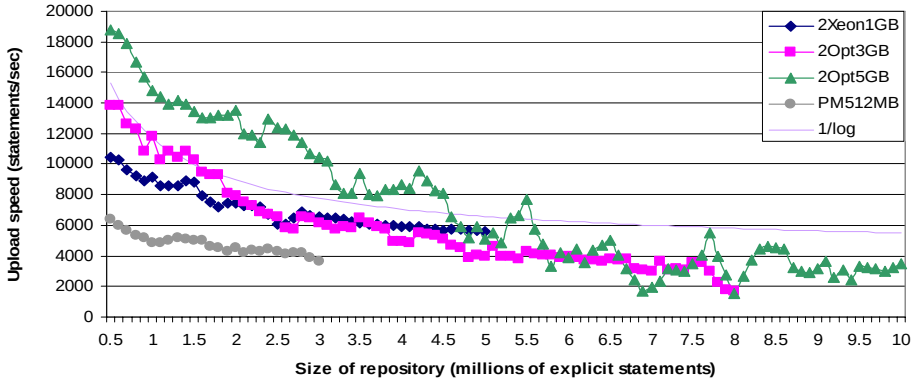
Table 1. Hardware and software configurations for the different runs

Name	Configuration	RAM	JDK
2Opt5GB	2xOpteron 2.4GHz, DDR400, Red Hat Linux v.3	5GB	JDK 1.5 64-bit
2Opt3GB	2xOpteron 1.4GHz, DDR333, Win 2003 64-bit	3GB	JDK 1.5 64-bit
2Xeon1GB	2xXeon 2.4GHz, DDR333, Win XP	1GB	JDK 1.5 32-bit
PM512MB	Pentium M 1.6GHz, DDR266, Win XP	0.5GB	JDK 1.5 32-bit

5 Results

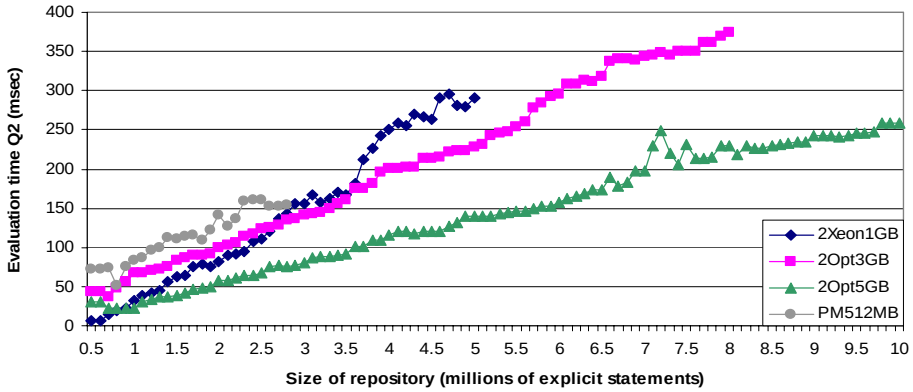
We present the results below in three charts, demonstrating: upload speed, query evaluation time for Query 2, and delete time. The chart of the diagram for Query 1, shows that the dependency is the same as for Query 2, but less steep, i.e. that the evaluation time grows in a similar but "slower" relation to the size of the repository.

Some of the data has been manually "polished" in order to clear distortions caused by garbage collection or known defects of the benchmark application. In all of these cases fixes were made so that (i) less than 10% of the data in each of the series were affected and (ii) the trends were respected as much as possible.



The upload speed of the fastest server starts at almost 19,000 statements/second (st/sec) and slows down to about 3,000 st/sec. Within 5GB of RAM this machine was

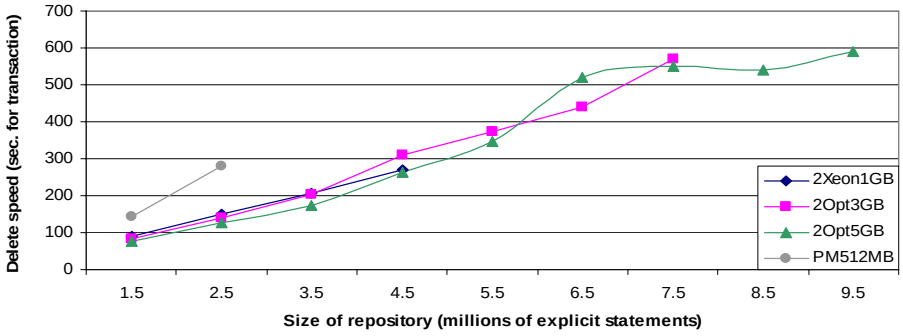
able to manage 10 million statements. The slowest machine starts at bit more than 6,000 st/sec and slows down to about 3,500 st/sec at 3 million statements.
The upload speed measurements needed to be smoothened, to provide clear tendencies. The chart presents the average of three consequent values in the actual series. The diagram demonstrates that the speed is falling down in reciprocal logarithmic dependency with respect to the size of the repository.



Both queries are evaluated within milliseconds. Two of the machines evaluated Query 2 for less than 30 ms with the initial size of the repository; 2Opt5GB needed ten times more (258 ms) time when the repository grew up to 10 million statements. Even the slowest machine, evaluates the complex query respectively in the range between 77 and 155 msec.

The dependency between the evaluation time and the size of the repository seems linear, which can be explained with the fact that the repository does not use any indices, which can let it behave in a more logarithmical fashion.

The delete operations are slow, as they perform full inferrence over the repository. The chart with the time necessary for completion of delete transaction demonstrates this. It takes between 77 and 144 seconds for the machines in this experiment to acomplish a delete opeation on top of a repository of size 1.5 millions statements. The time grows in an almost linear dependency to reach 10 minutes for a repository with 10 million statements.



The same performance experiments has been also conducted with a "pure", unoptimized, version of `RdfSchemaRepository` SAIL of Sesame v.1.2.1, which we name below Sesame-Memory. For the sake of fair comparison, the later has been extended with the persistence mechanisms of OWLIM. OWLIM v.2.0 uses the in-memory representation of Sesame, so, there were no deviations in the scalability (mostly determined by the amount of RAM available). Sesame-Memory was running on a smaller set of rules – only those for RDFS, without the OWL supporting ones in OWLIM; the inference, and the query results, of Sesame-Memory were incomplete in this test. What was compared was the speed of the inference engines. Sesame-Memory appeared 30-50% faster than OWLIM, which can be explained with the simpler set of rules. Apart from the benefit from checking fewer rules, the size of the repository was actually smaller for Sesame-Memory, because fewer implicit statements are generated and later on involved in further reasoning. With the growth of the repository, however, OWLIM is slow reducing the speed difference, which is an evidence, that the multi-threaded inference engine delivers growing improvement to the performance with the growth of the repository.

6 Related Work

For the purposes of this evaluation, we inspected two related studies. They both provide interesting approaches with respect to a more comprehensive benchmarking.

SWETO, [7], is a large scale dataset for testing algorithms for discovery of semantic associations, developed in the LSDIS Lab, University of Georgia. The schema component of the ontology reflects the types of entities and relationships available explicitly (and implicitly) in Web sources. It is populated through knowledge extractors. SWETO is intended for ontology tools benchmarking purposes; its version (1.3) is populated with well over 800,000 entities and over 1.5 million relationships. It is available as OWL files of total size about 250 MB. Overall, the SWETO dataset is similar in nature and origin to the ontologies and datasets used in this evaluation.

The Lehigh University evaluation, [4], is one of the most comprehensive benchmark experiments published recently. It evaluates the upload and query performance of 4 systems: memory-based Sesame, database-based Sesame, DLDB-OWL, and OWLJessKB. The benchmark was made with a relatively simple ontology about the organizational structure of a university and with synthetically generated datasets – for each university, a number of departments and employees with descriptions and relations among them were generated. The performance of the systems was measured in the course of incrementally increasing the number of the universities (from 1 to 50). The smallest set is 8MB and the largest one (for 50 universities) is 583MB, described in 1000 owl files, with a total of about 6 million statements. The main conclusions are: DLDB is the best system for very large data sets where an equal emphasis is placed on query response time and completeness. Sesame-Memory is the best when the size is relatively small (e.g., 1 million triples) and only RDFS inference is required.

The evaluation reported here is very similar to the on in [4], but simpler – we evaluate a single repository with a much smaller set of queries. Note that OWLIM

extends a later version what is called in [4] Sesame-Memory. Further, the evaluation there was made on a desktop machine with 512MB of RAM. Here we demonstrated that Sesame can scale much further, given server hardware. Our short-term plans include evaluation of OWLIM with respect to the Lehigh University dataset and queries, to allow for easy comparison with the results of the other systems.

7 Conclusion

This paper reported on the scalability and performance of a particular reasoning schema and implementation. We had presented OWLIM – a Sesame-based repository, which offers N-Triples persistence and supports a dialect close to OWL DLP through forward-chaining in-memory reasoning and full total materialization.

The evaluation can be summarized as follows: OWLIM can handle millions of statements on commodity hardware. The scale varies from a couple of millions on desktop machines to ten million statements on a server with few GB of RAM. The upload speed is in the range of thousands of statements/second and decreases in a sort of logarithmic dependency to the size of the repository. The delete operation is slow, but the query evaluation is very fast – even complicated queries pass in milliseconds.

We develop OWLIM in a number of directions. The applicability of the total materialization strategy should be evaluated to support more expressive languages. The rule engine can be optimized to allow for faster operation in less memory – ongoing experiments provide evidence that a ten-fold increase of the inference speed is achievable. Beyond these "gradual" improvements, we are working on a next generation repository, which should allow for efficient integration with databases and full-text search engines. We investigate strategies combining forward- and backward-chaining, without loosing the essential advantage of the inductive approaches, namely, that they allow for more straightforward RDBMS-like query optimizations.

8 References

1. Broekstra, J. (2005). *Storage, Querying and Inferencing for Semantic Web Languages*. Ph.D. Thesis, VU Amsterdam, ISBN 90 9019 2360. http://www.cs.vu.nl/~jbroeks/#pub
2. Dean, M; Schreiber, G. – editors; (2004). *OWL Web Ontology Language Reference.* W3C Recommendation, 10 Feb. 2004. http://www.w3.org/TR/owl-ref/
3. Grosof, B; Horrocks, I; Volz, R; Decker, St. (2003). *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WWW2003, May 2003.
4. Guo, Y; Pan, Z; and Heflin, J. (2004). *An Evaluation of Knowledge Base Systems for Large OWL Datasets.* CS and Engineering Department, Lehigh University, Bethlehem, PA18015, USA, http://www.lehigh.edu/~yug2/iswc2004-benchmark.PDF
5. Hayes, P. (2004). *RDF Semantics*. W3C Recommendation, 10 February 2004.http://www.w3.org/TR/2004/REC-rdf-mt-20040210/
6. Klyne, G; Carroll, J. J. (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax.* W3C Recom. 10 Feb, 2004. http://www.w3.org/TR/rdf-concepts/
7. LSDIS and the University of Georgia. (2004). *Semantic Web Technology Evaluation Ontology (SWETO).* http://lsdis.cs.uga.edu/Projects/SemDis/sweto/