Minerva: A Scalable OWL Ontology Storage and Inference System

Jian Zhou[1], Li Ma[2], Qiaoling Liu[1], Lei Zhang[2], Yong Yu[1], and Yue Pan[2]

[1] APEX Data and Knowledge Management Lab,
Department of Computer Science and Engineering,
Shanghai Jiao Tong University,
200240 Shanghai, China.
{priest, lql, yyu}@apex.sjtu.edu.cn

[2] IBM China Research Lab,
100094 Beijing, China
{malli, lzhangl, panyue}@cn.ibm.com

Abstract. With the increasing use of ontologies in Semantic Web and enterprise knowledge management, it is critical to develop scalable and efficient ontology management systems. In this paper, we present Minerva, a storage and inference system for large-scale OWL ontologies on top of relational databases. It aims to meet scalability requirements of real applications and provide practical reasoning capability as well as high query performance. The method combines Description Logic reasoners for the TBox inference with logic rules for the ABox inference. Furthermore, it customizes the database schema based on inference requirements. User queries are answered by directly retrieving materialized results from the back-end database. The effective integration of ontology inference and storage is expected to improve reasoning efficiency, while querying without runtime inference guarantees satisfactory response time. Extensive experiments on University Ontology Benchmark show the high efficiency and scalability of Minerva system.

1 Introduction

The rapid growing information volume in World Wide Web and enterprise intranet makes it difficult to access and maintain the information required by users. Semantic Web, the next generation web, aims to provide easier information access and usability by exploiting machine understandable metadata. In recent years, ontology, which enables a shared, formal, explicit and common description of a domain knowledge, has been recognized to play an important role in Semantic Web and enterprise knowledge management. W3C has recommended two standards for publishing and sharing ontologies on the World Wide Web: RDF/RDFS [1] and OWL [2]. OWL builds on top of RDF/RDFS and adds more vocabularies for describing properties and classes, which improves expressiveness but increases reasoning complexity.

The logical foundation of OWL is Description Logic(DL) [3] which is a decidable fragment of First Order Logic(FOL). Therefore, inference of OWL ontologies can be handled by DL reasoners. A DL knowledge base consists of two components, a TBox and an ABox. The TBox describes the terminology, while the ABox contains assertions about individuals. Correspondingly, the DL reasoning includes TBox reasoning(i.e., reasoning with concepts) and ABox reasoning(i.e., reasoning with individuals). It is demonstrated in [4, 5] that DL reasoners are able to cope with TBox reasoning of real world ontologies. But the extremely large number of instances of real ontologies makes it difficult for DL reasoners to deal with ABox reasoning. It is critical to develop scalable and efficient ontology management systems.

This paper presents Minerva, a storage, inference and querying system for large-scale OWL ontologies on top of relational databases. It aims to meet scalability requirements of real applications and provide practical reasoning capability as well as high query performance. Minerva is a component of the IBM Integrated Ontology Development Toolkit(IODT) which is publicly available at [6]. Figure 1 shows graphical user interface of Minerva. Using Minerva, one can store multiple large-scale ontologies in different ontology stores, issue SPARQL [7] queries and obtain results listed in tables or visualized as RDF graphs.

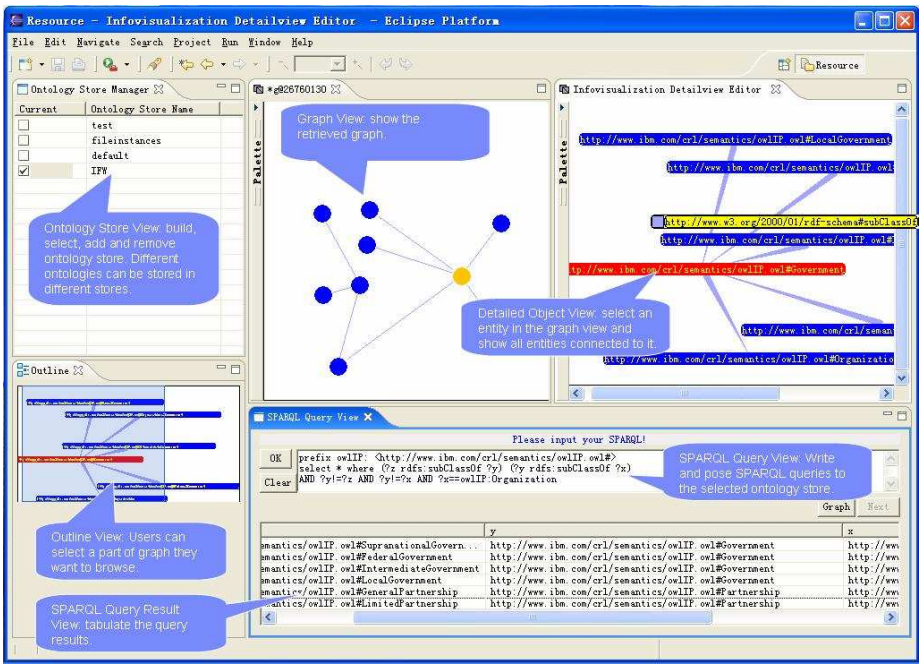

Fig. 1. The Graphical User Interface of Minerva

In order to achieve high system performance and provide practical inference capability, we combine DL reasoners for the TBox inference with logic rules for the ABox inference. Our method is based on the mapping theory between Description Logic and Logic Programs [8]. It is proved that Description Horn Logic(DHL) ontologies can be translated into a set of logic programs(i.e., logic rules) without loss of semantics. The TBox precomputation by DL reasoners ensures complete and sound inference on classes and properties within OWL-DL. The logic rules translated from DHL implement practical ABox inference, since DHL covers RDFS semantics (except from the recursive meta model) and most practical OWL semantics [8]. Particularly, we customize the relational database schema based on the translated logic rules for efficient inference. Both the TBox and ABox inference results are materialized in the database so that SPARQL queries can be evaluated efficiently. Extensive experiments on University Ontology Benchmark [9] show the high efficiency and scalability of Minerva system.

The rest of this paper is organized as follows. Section 2 gives an overview of Minerva. Detailed storage design, inference and query processing is described in Section 3. Evaluation and results are reported in Section 4. Related work is discussed in Section 5 and Section 6 concludes the paper.

2 Overview

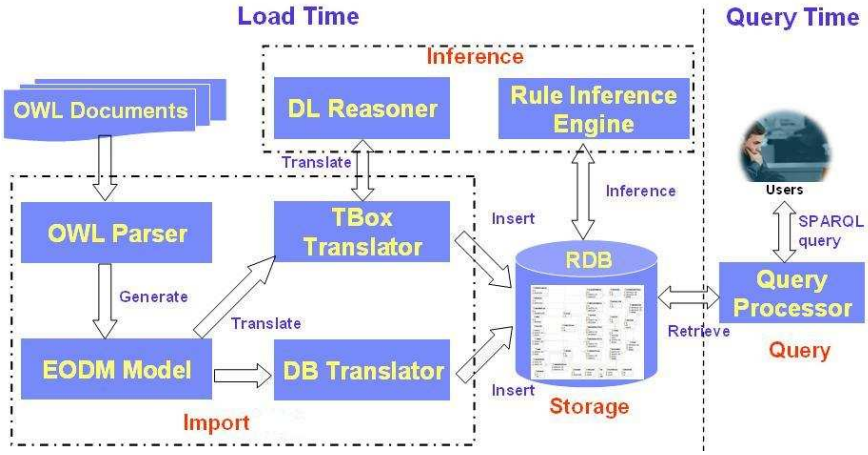

Fig. 2. The Component Diagram of Minerva

Figure 2 shows the component diagram of Minerva. Minerva is comprised of Import Module, Inference Module, Storage Module (it is an RDBMS schema) and Query Module.

– Import Module. The import module consists of an OWL parser and two translators. The parser parses OWL documents into an in-memory EODM model(EMF ontology definition metamodel) [6][3], and then the DB translator populates all ABox assertions into the back-end database. The function of the TBox translator is twofold, one is to populate all TBox axioms into a DL reasoner and the other is to obtain inference results from the DL reasoner and insert them into the database.

– Inference Module. A DL reasoner and a rule inference engine compose the inference module. Firstly, the DL reasoner infers complete subsumption relationships between classes and properties. Then, the rule engine conducts ABox inference based on the DLP rules. Currently, the inference rules are implemented using DB SQL statements. Besides our developed structural subsumption algorithm [6], Minerva can use well-known RACER [10] and Pellet [11] for TBox inference via DIG interface.

– Storage Module. It is intended to store both original and inferred assertions by the DL reasoner and the rule inference engine. Since inference and storage are considered as an inseparable component in a complete storage and query system for ontologies, we design a specific RDBMS schema to effectively support ontology inference. Currently, Minerva can take IBM DB2, Derby (http://incubator.apache.org/derby/) and HSQLDB (http://www.hsqldb.org/) as the back-end database.

– Query Module. The query language supported by Minerva is SPARQL [7]. User SPARQL queries are answered by directly retrieving inferred results from the database using SQL statements. There is no inference during the query answering stage because the inference has already been done at the time of loading data. Such processing is expected to improve the query response time.

In summary, Minerva combines a DL reasoner and a rule engine for ontology inference, materializes all inferred results into a database. The database schema is well designed to effectively support inference and SPARQL queries are answered by direct retrieval from the database. More details about inference and storage are described in next section.

3 Inference, Storage and Querying

3.1 Inference

Grosof et al. [8] defined a new intermediate knowledge representation contained within the intersection between Description Logic(DL) and Logic Programs(LP):

[3] EODM is an implementation of OMG's Ontology Definition Metamodel (http://www.omg.org/cgi-bin/doc?ad/2003-3-40) on Eclipse Modeling Framework(EMF) (http://www.eclipse.org/emf).

Description Logic Programs(DLP), and the closely related Description Horn Logic(DHL). DLP is the LP-correspondent of DHL ruleset. The definition of DHL and DLP makes it practicable to do efficient reasoning of large-scale ontology using the rule inference engine. Considering most real OWL-DL ontologies are more complex than DHL, we extend the original DHL axioms to support OWL-DL-complete[4] TBox inference. More precisely, we use a DL reasoner to obtain all class and property subsumption relationships, instead of supporting only DHL axioms. Note that we decompose the complex class descriptions into instantiations of class constructors, assign a new URI to each instantiation and ask the DL reasoner for inference as well. For ABox reasoning, Minerva implements all DLP rules based on the Meta Mapping approach [12]. The Meta Mapping converts all concept and property instances into facts of two predicates `TypeOf` and `Relationship`, and ontology axioms into facts of some predefined predicates(e.g., `SubClassOf` and `SubPropertyOf`). Consequently, there are a fixed number of predefined predicates, reflecting the vocabulary of OWL-DL. Based on these predicates, only a constant rule set is required to cover the semantics of the ontology.

Table 1. The set of rules that cover all DHL axioms. (Rel stands for Relationship, Type stands for TypeOf)

DHL Axioms	Corresponding rule
Rel-Rel Layer(Group 1):	
$P \sqsubseteq Q$	$\mathrm{Rel}(x, Q, y)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{SubPropertyOf}(P, Q)$.
$P \equiv Q^-$	$\mathrm{Rel}(y, Q, x)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{InversePropertyOf}(P, Q)$.
$P^+ \equiv P$	$\mathrm{Rel}(x, P, z)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{Rel}(y, P, z)$, $\mathrm{Transitive}(P)$.
$P \equiv P^-$	$\mathrm{Rel}(y, P, x)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{Symmetric}(P)$.
Rel-Type Layer(Group 2):	
$\top \sqsubseteq \forall P^-.D$	$\mathrm{Type}(x, D)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{Domain}(P, D)$.
$\top \sqsubseteq \forall P.D$	$\mathrm{Type}(y, D)$:- $\mathrm{Rel}(x, P, y)$, $\mathrm{Range}(P, D)$.
Type-Type Layer(Group 3):	
$C \sqsubseteq D$	$\mathrm{Type}(x, D)$:- $\mathrm{Type}(x, C)$, $\mathrm{SubClassOf}(C, D)$.
$\exists R.D \sqsubseteq C$	$\mathrm{Type}(x, C)$:- $\mathrm{Rel}(x, R, y)$, $\mathrm{Type}(y, D)$, $\mathrm{SomeValuesFrom}(C, R, D)$.
$C \sqsubseteq \forall R.D$	$\mathrm{Type}(y, D)$:- $\mathrm{Rel}(x, R, y)$, $\mathrm{Type}(x, C)$, $\mathrm{AllValuesFrom}(C, R, D)$.
$D_1 \sqcap D_2 ... \sqcap D_n \sqsubseteq C$	$\mathrm{Type}(x, C)$:- $\mathrm{Type}(x, D_1)$, $\mathrm{IntersectionMemberOf}(D_1, C)$,..., $\mathrm{Type}(x, D_n)$, $\mathrm{IntersectionMemberOf}(D_n, C)$.

The DLP rules obtained by the mapping of DHL axioms are listed in Table 1. These rules can be directly handled by deductive databases. Here, we make use of mature relational database to store large-scale ontologies. In order to leverage optimization technologies and scalability of RDBMS as much as possible,

[4] DL reasoners implement sound and complete reasoning algorithms that can effectively handle the DL fragment of OWL.

we enforce DLP rules using SQL statements on the underlying RDBMS as the implementation of a rule inference engine. [13] shows the semantics of logic programs can be interpreted by the fixed point semantics with respect to Herbrand Models. So we can iteratively execute these rules until no new assertions can be made to obtain the fixed point. The inferred results are materialized in the database so that queries can be evaluated efficiently. Our approach is to trade space for time.

Firstly we use the DL reasoner to calculate the `SubClassOf` relationships between classes and `SubPropertyOf` relationships between properties. The results of this precomputation are stored in the database tables and used by the subsequent rule inference. Rules for ABox inference are categorized into three groups based on their dependency so that rules in group i cannot be fired by rules in group $j (j \geq i)$. This effectively reduces inference costs using SQL statements. Rules in each group will be recursively executed until no new results can be generated. Then the rule engine will proceed to the next group of rules.

The TBox precomputation by DL reasoners ensures complete and sound OWL-DL inference on classes and properties, which can not be covered by DLP rules. For example, if we have axioms $\{Mother \equiv Woman \sqcap \exists hasChild.Person$, $Parent \equiv Person \sqcap \exists hasChild.Person, Woman \sqsubseteq Person\}$, the implicit relationship that $Mother$ is a subclass of $Parent$ cannot be derived by the DLP rules but will be found by a DL reasoner. After the full TBox reasoning, our rule engine provides complete and sound ABox inference with respect to the semantics of DHL, which covers RDFS semantics (except from the recursive meta model) and most practical OWL semantics [8]. Therefore, our method provides practical inference capability for real applications.

3.2 Storage on Relational Databases

Two best-known ontology toolkits, Jena [14] and Sesame [15], have provided supports for ontology persistent storage on relational database. They persist OWL ontologies as a set of RDF triples and do not consider specific processing for complex class descriptions generated by class constructors(boolean combinators, various kinds of restrictions, etc). [16] proposed to store OWL restrictions in a separate table for ease of representation. However, they did not explain and discuss the effect of their schema on inference in-depth.

The highlight of our database schema is that all predicates in the DLP rules have corresponding tables in the database. Therefore, these rules can be easily translated into sequences of relational algebra operations. For example, rule Type(x, C) :- Rel(x, R, y), Type(y, D), SomeValuesFrom(C, R, D) in Table 1 has four predicates in the head and body, resulting in three tables: `Relationship`, `Typeof` and `SomeValuesFrom`. It is highly straightforward to use SQL statements to execute this rule. We just need to use simple SQL select and join operations among these three tables. The effective integration of ontology inference and storage is expected to significantly reduce inference costs.

We categorize tables of the database schema into 4 types: atomic tables, TBox axiom tables, ABox fact tables and class constructor tables, which are shown in

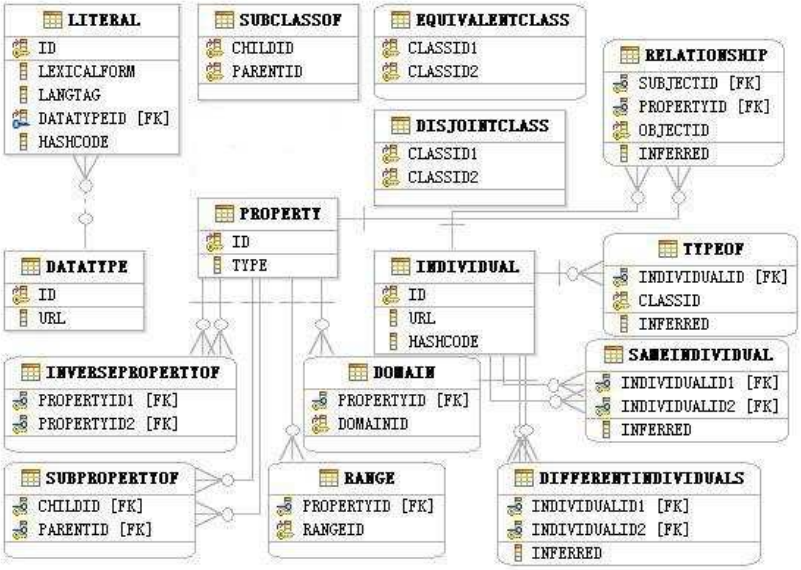

Fig. 3. The relational schema of Atomic, TBox axiom and ABox fact tables

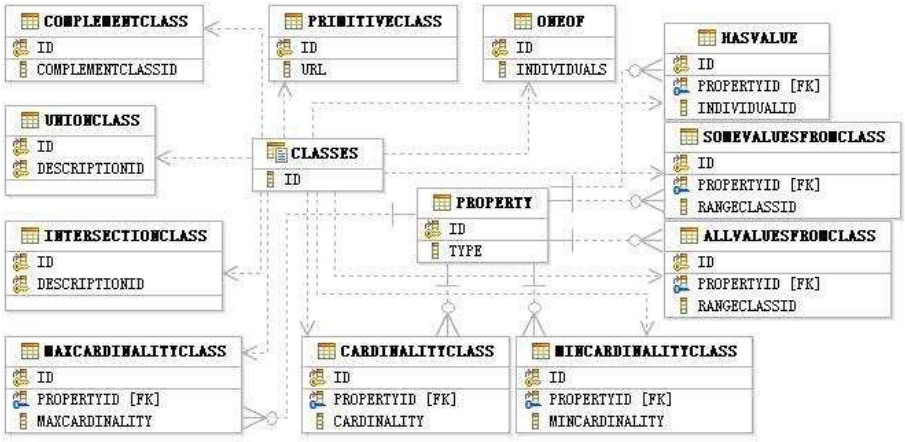

Fig. 4. The relational schema of class constructor tables

Figures 3 and 4. The atomic tables include: `PrimitiveClass` (in Figure 4), `Property`, `Datatype`, `Individual` and `Literal`. These tables encode the URI with an integer value(the ID column), which reduces the overhead caused by the long URI to a minimum. The hashcode column for URI in `Individual` and `Literal` tables is used to speed up search on individuals and literals. The `Property` table stores URI as well as its characteristics(symmetric, transitive, etc).

There are two important kinds of ABox facts: `TypeOf` and `Relationship`. [17] discussed the advantages of 'Vertical Table'(storing all data in one table with index on the type) in terms of manageability and flexibility to 'Binary Table'(a table for each class and property). Therefore, we adopt 'Vertical Table' to store ABox facts. Tables `SubClassOf`, `SubPropertyOf`, `Domain`, `Range`, `DisjointClass`, `InversePropertyOf` are used to store TBox axioms.

The most distinguishing part of our design is class constructor tables in Figure 4. We decompose the complex class descriptions into instantiations of class constructors, assign a new ID to each instantiation and store it in the corresponding class constructor table. Take the axiom $Mother \equiv Woman \sqcap \exists hasChild.Person$ as an example, we first define S_1 for $\exists hasChild.Person$ in `SomeValuesFrom` table. Then I_1 standing for the intersection of $Woman$ and S_1 will be defined in the `IntersectionClass` table. At last, $\{Mother \sqsubseteq I_1, I_1 \sqsubseteq Mother\}$ will be added to the `SubClassOf` table. Such a design is motivated by making the semantics of complex class description explicit. In this way, all class nodes in the OWL subsumption tree are materialized in database tables, and rule inference can thus be easier to implement and faster to execute using SQL statements. Also, a view `Classes` is defined to provide an overall view of both named and anonymous classes in OWL ontology.

As introduced in previous section, Minerva materializes all inferred results in the database. Therefore, we have to propose effective methods for ontology update.

1. Addition of TBox Axioms. When new TBox axioms are added, Minerva will send them and the original TBox together to a DL reasoner. Then, we can obtain newly-inferred TBox Axioms and store them into the database. Finally, the rule engine will do ABox inference with only the newly-added and newly-inferred TBox axioms.
2. Addition of ABox assertions. Currently, two kinds of methods for ABox update are supported. The first approach is to add only one assertion at one time. Minerva will determine rules to be fired based on the premise of all ABox rules and obtain inferred assertions using these rules. Then, the inferred assertions are processed one by one in the same manner until no new assertion can be inferred. Another way is relatively straightforward. It just re-runs all ABox inference rules and newly-inferred assertions are materialized into the database.
3. Deletion of TBox Axioms. When some TBox axioms need to be deleted, Minerva will delete all inferred results and redo both TBox and ABox inference, just like populating a new ontology. Obviously, such an update is expensive.

But fortunately, ontologies do not change frequently in real applications and thus deletion of TBox Axioms occurs rarely.

4. Deletion of ABox assertions. When deleting an assertion, Minerva first obtains all assertions inferred from this assertion. Then, it runs ABox rules to check whether each of those assertions could be inferred from other existing assertions. By this way, we make sure the safe deletion of an assertion.

3.3 Querying

Recently, W3C has worked out a query language SPARQL [7] for RDF retrieval. The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, which is like an RDF triple but with the possibility of variables in any positions. Minerva has implemented the basic query features of SPARQL, but class expressions are not supported in user queries.

Our query answering algorithm is to simply retrieve results from the database including both original assertions and inferred facts. The query answering module consists of a SPARQL query parser and a SQL translator. In fact, every $x_i : C$ pattern can be translated into a select operation on `TypeOf` table, while every $< x_i, x_j >: R$ pattern can be translated into a select operation on `Relationship` table. The translator uses join and union operations on the basic triple selections to build a complete SQL statement and obtain final results. That is, we make effective use of the well-optimized SQL query engine for SPARQL evaluation. This makes Minerva practicable for concurrent queries in various real applications.

4 Evaluation

4.1 Experiment settings

Experiments are designed to evaluate scalability, efficiency and inference capability of Minerva. We compare our system with OWLIM [18] and DLDB-OWL [19]. These two systems are chosen because it is reported in [20] that DLDB and Sesame(OWLIM is an extension of Sesame) have better performance than other systems in general. DLDB [19] uses the DL reasoner to precompute class and property hierarchies, and employs relational views to answer extensional queries. Its ABox inference mainly supports instance membership reasoning.

Evaluation is conducted on University Ontology Benchmark(UOB) [9], which is extended from the well-known Lehigh University Benchmark(LUBM) [20]. The UOB extends the LUBM in terms of two aspects: 1) include both OWL-Lite and OWL-DL ontologies covering a complete set of OWL-Lite and OWL-DL constructors respectively. 2) add necessary properties to build effective instance links (hence reasoning requirements) and improve instance generation methods accordingly. The UOB consists of university domain ontologies, customizable and repeatable synthetic data, a set of test queries and corresponding answers. In our experiments, we create 3 test sets: OWL Lite-1, OWL Lite-5, OWL Lite-10(The parameter denotes the number of universities). The number of triples is about 220000 for Lite-1, 1100000 for Lite-5 and 2200000 for Lite-10.

There are 13 queries in the benchmark which cover most features of OWL-Lite. The details of all queries can be found in [9]. Here, the evaluation metrics used in [20] are adopted for comparison:

1. *Load time*. The time for storing the benchmark data to the repository, including time for parsing OWL files and reasoning.
2. *Query Response time*. The time for issuing the query, obtaining the result set and traversing the set sequentially.
3. *Completeness and Soundness*. Completeness means the system generates all answers that are entailed by the knowledge base and soundness means all generated answers are correct.

Here, the load time is an average of three times of experiments and the query response time is an average of ten times of experiments. Experiments are conducted on a PC with Pentium IV CPU of 2.66 GHz and 1G memory, running Windows 2000 professional with Sun Java JRE 1.4.2 (JRE 1.5.0 for OWLIM) and Java VM memory of 512M. The version of OWLIM and DLDB we evaluated are v2.8.2 (http://www.ontotext.com/owlim/) and DLDB-OWL (http://swat.cse.lehigh.edu/downloads/dldb-owl.html) respectively.

4.2 Results

Load Time Table 2 compares the load time of three systems. We can see that OWLIM can load the smallest data set OWL Lite-1 using only 29 seconds. It is fastest among these systems. When loading OWL Lite-5 and OWL Lite-10, it reported "Out of Memory" error. In fact, we have also tested other memory-based systems, e.g. RACER [10]. They cannot load the smallest data set OWL Lite-1 which includes about 220,000 triples, because of the memory limitation. The results strongly support our understanding that database technologies should be used to deal with large-scale ontology storage. The average load time of DLDB is less than Minerva's. The difference mainly lies in the inference capabilities and methods of the two systems. DLDB makes use of FaCT for TBox inference and supports a small subset of OWL-Lite in ABox, mainly membership inference based on `SubClassOf` axiom. Minerva implements inference of DLP rules and covers most of OWL-Lite. DLDB constructs views based on inferred class hierarchy information to implement ABox inference, whereas Minerva needs to materialize inferred results by DLP rules in database. Besides additional time for more reasoning, Minerva takes some time to insert inferred results into database. This makes Minerva slower than DLDB to load data. Note that the time in Table 2 includes the reasoning time as these three systems do inference at load time. The time needed for inference in Minerva is about 30%-40% of the load time.

Completeness and Soundness The three systems can answer all queries soundly. That is, they do not return wrong answers for any query. So we only need to check their completeness. Table 3 shows the results. Compared with previous version, OWLIM v2.8.2 can answer all queries correctly. In this new release,

Table 2. Load Time (the unit is second)

	OWL Lite-1	OWL Lite-5	OWL Lite-10
Minerva	868	5469	9337
DLDB	428	1945	3904
OWLIM	29	N/A	N/A

more rules are added and inference is made configurable. As is known, OWL-Lite and OWL-DL reasoning cannot be implemented only by rules. That is, OWLIM can conduct only partial OWL-DL TBox inference. This is different from DLDB and Minerva which depend on a DL reasoner for complete TBox inference. Coincidentally, the UOB does not contain a query that needs subsumption inference not covered by existing OWLIM rules. The inference capability of DLDB is relatively weak that it gives 100% complete answers to only 3 queries. Minerva is able to completely process 12 out of 13 queries. Inference on `minCardinality` needed by query 13 is not currently supported in Minerva. As described in Section 3, Minerva makes effective use of DLP rules for ABox inference. Similar to OWLIM, Minerva can add more rules to enhance its ABox inference. In fact, we are currently working on this improvement.

Table 3. Query Completeness (Qi stands for the ith query and the real number denotes $\frac{|Answer_{system} \bigcap Answer_{correct}|}{|Answer_{correct}|}$)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13
Minerva	1	1	1	1	1	1	1	1	1	1	1	1	0.67
DLDB	1	0.82	1	1	0	0	0	0	0	0.83	0	0.2	0.56
OWLIM	1	1	1	1	1	1	1	1	1	1	1	1	1

Query Response Time Figure 5 shows quantitative comparison on query response time on data sets of different sizes among these systems. The first graph compares performance of three systems on data set OWL Lite-1. OWLIM performs the best in general because its queries are evaluated in memory. Both DLDB and Minerva leverage relational database for query evaluation, which needs expensive IO access to hard disk. On the other hand, DLDB and Minerva have better scalability. This is more or less benefited from the scalability of RDBMS. As OWLIM fails to load other two larger data sets, we do not show its performance curve with the increasing data size. An interesting phenomenon we observed in experiments is about query evaluation of OWLIM. Query 4 includes a four-triple constraint, {?y rdf:type benchmark:Faculty . ?y benchmark:isMemberOf <http://www.Department0.University0.edu> . ?x rdf:type benchmark:Publication . ?x benchmark:publicationAuthor ?y}. If we exchange

the order of the 2nd and 3rd triples in the constraint, the response time will increase to 13726ms from only 626ms. That is because OWLIM uses triple patterns in the constraints to filter out irrelevant results. When the most selective triple patterns are at the end of the query, the filtering process would be time-consuming. However, DLDB and Minerva avoid this problem by leveraging query optimization technologies of RDBMS. The second and third graph show the performance of DLDB and Minerva on different data sets. We observed that the query time of Minerva never exceeds 2 seconds, which makes Minerva qualified for practical applications. Also, we found that query response time of Minerva scales well with the data size. The test results of DLDB show that its query time dramatically grows with the increase of the data size and its performance is not as good as Minerva's. DLDB uses class views which is built based on inferred class hierarchy at load time to retrieve instances at query time. DLDB's view query[7] needs to execute union operations in runtime for retrieval. In contrast, Minerva materializes all inferred results, and uses select operations on pre-built index in most cases instead of union operations. This results in less computational costs.

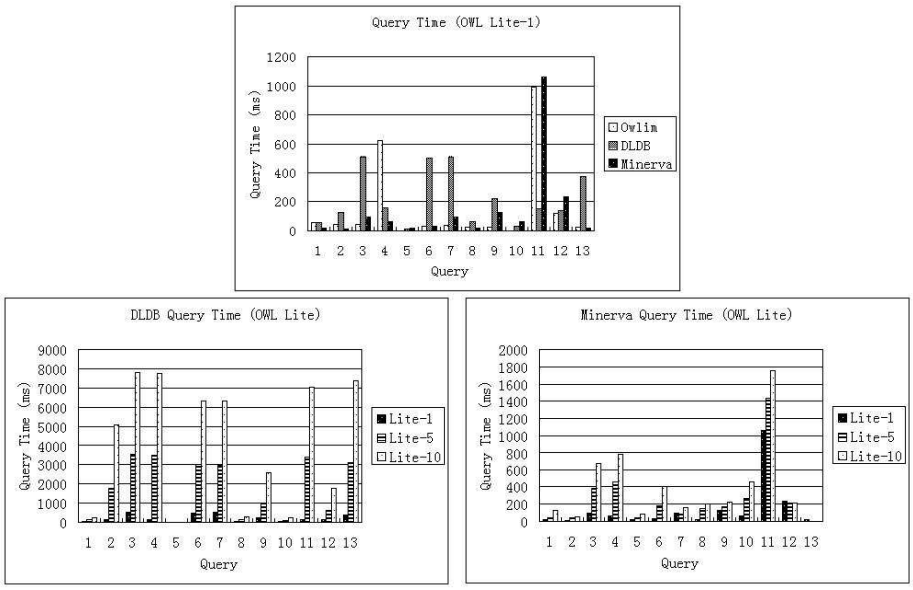

Fig. 5. The average query response time.

[7] Note that a view is equivalent to a query in relational database.

4.3 Discussions

Based on the above results and analysis, we can draw a number of conclusions as well as find that some issues need to be further investigated.

1. The proposed method for inference is a combination of the DL reasoner and a set of DLP rules corresponding to DHL semantics. It promises that our inference on DHL (a subset of OWL-DL) ontologies is sound and complete as well as that the complete subsumption relationship among classes and properties can be made explicit. Now, we intend to add more rules for ABox inference so that Minerva can handle more expressive ontologies. Also, we are focusing on how to support inference and querying on datatypes (e.g., integers, doubles).
2. Experiments show the high scalability and desirable query optimization of DLDB and Minerva. In fact, this benefits mainly from the underlying relational database. DLDB uses MS Access database and Minerva is built on IBM DB2. In our experiments, we did not change DLDB's back-end database to DB2 as it is reported in [19] that DLDB achieves high performance with default Access database. Further work is to investigate the impact of the underlying RDBMS on the performance of ontology repositories. OWLIM made a significant and meaningful attempt to build native ontology repository and the results are promising. Like the development of native XML storage systems, more efforts are needed for native ontology persistent storage including storage model, query caching and optimization.
3. In Section 3.2, we discussed the ontology update problem. Our method for TBox deletion is expensive though TBox axioms are not deleted frequently. Currently, we are working on an incremental update method for the deletion of TBox axioms.

5 Related Work

Some ontology storage and inference systems have been developed in the past several years. For the sake of efficient storing and querying data with high scalability, there is a trend toward extending RDBMS with OWL inference capabilities, e.g. DLDB [19], Sesame [15], and InstanceStore [21]. Detailed comparisons with DLDB and OWLIM has been reported in previous section.

Sesame is a well-known system which provides efficient storage and expressive querying of large quantities of metadata in RDF/RDFS. In order to support OWL ontology management, Sesame extends its rules. But its simple extension cannot guarantee the inference completeness. OWLIM [18] is another extension for Sesame, which provides a reliable persistence based on N-Triples files. However, its reasoning and query evaluation are performed in memory, which makes it less suitable to handle large numbers of instances in real world ontologies.

InstanceStore [21] implements a restricted form of ABox reasoning on RDBMS. More precisely, it provides sound and complete reasoning on role-free ABox.

However, role-free ABox does not include role assertions which describe the relationships between individuals. This guarantees its high efficiency but limits its use in real applications needing role inference.

KAON2 [22] is a successor to the KAON [23] project, an open-source ontology management infrastructure which pays special attention to scalable and efficient reasoning with ontologies. Whereas KAON used a proprietary extension of RDFS, KAON2 is based on OWL-DL and F-Logic. Reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program, thus allowing to apply well-known deductive database techniques, such as magic sets or join-order optimizations. ABox assertions can be stored in a relational database (RDBMS), but not all the TBox and ABox inference results are materialized in the database as Minerva.

6 Conclusion and Future Work

This paper presented an RDBMS-based storage and inference system for large-scale OWL ontologies. DL reasoner for the TBox reasoning and rule-based algorithms for the ABox reasoning are combined appropriately. Based on the theoretically proved mapping from Description Logic to Logic Programs [8], we can claim that our system is sound and complete on DHL ontologies. By calculating the subsumption relationship between classes and properties with the DL reasoner, we achieved complete class and property hierarchies and further improved inference capability of Minerva. Extensive experimental results showed the high efficiency and scalability of Minerva.

7 Acknowledgements

The authors would like to thank GuoTong Xie, Yang Yang, Jing Lu, Sheng-Ping Liu, Lei Li for their helpful discussions and constructive comments, Atanas Kiryakov and Damyan Ognyanov of OntoText Lab for their great help on evaluation.

References

1. Brickley, D., Guha, R., eds.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. (2004)
2. Bechhofer, S., van Harmelen, Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A., eds.: OWL Web Ontology Language Reference. W3C Recommendation. (2004)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications., Cambridge University Press (2003)
4. Haarslev, V., Möller, R.: High Performance Reasoning with Very Large Knowledge Bases. In: DL. (2000)
5. Horrocks, I.: FaCT and iFaCT. In: DL. (1999)

6. : IBM's Integrate Ontology Development Toolkit. (http://www.alphaworks.ibm.com/tech/semanticstk)
7. Prud'hommeaux, E., Seaborne, A., eds.: SPARQL Query Language for RDF. W3C Working Draft. (2005)
8. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: WWW. (2003) 48–57
9. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards A Complete OWL Ontology Benchmark. In: To appear in European Semantic Web Conference. (2006)
10. Haarslev, V., Möller, R.: RACER System Description. In: Automated Reasoning, First International Joint Conference, IJCAR 2001. (2001)
11. Sirin, E., Parsia, B.: Pellet: An OWL DL Reasoner. In: DL. (2004)
12. Weithöner, T., Liebig, T., Specht, G.: Storing and Querying Ontologies in Logic Databases. In: Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003. (2003)
13. Beeri, C.: Logic Programming and Databases. In: ICLP. (1990)
14. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: (Alternate Track Papers & Posters) WWW. (2004)
15. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: ISWC. (2002) 54–68
16. Das, S., Chong, E.I., Eadon, G., Srinivasan, J.: Supporting Ontology-Based Semantic matching in RDBMS. In: (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases. (2004)
17. Agrawal, R., Somani, A., Xu, Y.: Storage and Querying of E-Commerce Data. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases. (2001)
18. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - a pragmatic semantic repository for OWL. In: Proceedings of the 2005 International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2005). (2005)
19. Pan, Z., Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: PSSS1 - Proceedings of the First International Workshop on Practical and Scalable Semantic Systems. (2003)
20. Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: ISWC. (2004)
21. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL Reasoning with Large Numbers of Individuals. In: DL. (2004)
22. Motik, B., Sattler, U.: Practical DL reasoning over large ABoxes with KAON2, available at http://kaon2.semanticweb.org/. (2006)
23. KAON: (http://kaon.semanticweb.org/)