DLDB2: A Scalable Multi-Perspective Semantic Web Repository

Zhengxiang Pan, Xingjian Zhang, Jeff Heflin
Department of Computer Science and Engineering, Lehigh University
19 Memorial Dr. West, Bethlehem, PA 18015, U.S.A.
{zhp2, xiz307, heflin}@cse.lehigh.edu

Abstract

A true Semantic Web repository must scale both in terms of number of ontologies and quantity of data. It should also support reasoning using different points of view about the meanings and relationships of concepts and roles. Our DLDB2 system has these features. Our system is sound and complete on a sizable subset of Description Horn Logic when answering extensional conjunctive queries, but more importantly also computes many entailments from OWL DL. By delegating TBox reasoning to a DL reasoner, we focus on the design of the table schema, database views, and algorithms that achieve essential ABox reasoning over an RDBMS. We evaluate the system using synthetic benchmarks as well as real-world data and queries.

1 Introduction

The evolving Semantic Web extends the existing Web with structure for data and a mechanism to specify formal, logic based and shareable semantics. It does so via the use of ontologies, which are expressed in the Web Ontology Language OWL. The formal semantics of Semantic Web data can be used to make inferences (logical deduction) and develop powerful query systems. The Semantic Web is growing and clearly scalability is an important requirement for Semantic Web systems. Furthermore, the Semantic Web is an open and decentralized system where different parties can and will, in general, adopt different ontologies. Thus, merely using ontologies, does not reduce heterogeneity: it just raises heterogeneity problems to a different level. Without some form of alignment, the data that is described in terms of one ontology will be inaccessible to users that ask questions in terms of another ontology. Our theory of perspective provides a framework to integrate data sources using different ontologies. Based upon this framework, we built DLDB2, a scalable Semantic Web knowledge system that allows queries from different points of view.

In what follows, we first introduce ontology perspectives. We present the system's architecture, design and implementation with a focus on how the reasoning is achieved and perspectives are supported. We then evaluate the system using benchmarks as well as real-world multi-ontology data sets and queries.

2 Ontology Perspectives

In prior work, we have defined ontology perspectives which allows the same set of data sources to be viewed from different contexts, using different assumptions and background information [4]. That work also presents a model theoretic description of perspectives. In this section, we set aside the versioning issues of that paper and introduce some essential definitions.

Each perspective is based on an ontology, hereafter called the basis ontology or base of the perspective. By providing a set of terms and a standard set of axioms, an ontology provides a shared context. Thus, data sources that commit to the same ontology have implicitly agreed to share a context. When it makes sense, we also want to maximize integration by including data sources that commit to different ontologies.

We now provide informal definitions to describe our model of the Semantic Web. A Semantic Web space $\mathcal{W}$ is a pair $\langle \mathcal{O}, \mathcal{S} \rangle$, where $\mathcal{O}$ is a set of ontologies and $\mathcal{S}$ is a set of data sources. An ontology O in $\mathcal{O}$ is a four-tuple $\langle \mathcal{C}, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$, where $\mathcal{C}$ is a set of concepts; $\mathcal{R}$ is a set of roles; $\mathcal{T}$ is a TBox that consists of a set of axioms; $\mathcal{E} \subset \mathcal{O}$ is the set of ontologies that are extended by O.

An ontology defines a set of concepts and a set of roles, the union of which is referred to as its vocabulary. It also contains a set of axioms, which is called the TBox (a standard term in description logics, short for terminological box). An ontology can extend another, which means that it adds new vocabulary and or axioms. Extension is sometimes referred to as inclusion or importing.

An ancestor of an ontology is an ontology extended either directly or indirectly by it. If O_2 is an ancestor of O_1, we write $O_2 \in anc(O_1)$. Note the ancestor function re-

turns the extension closure of an ontology, which does not include the ontology itself.

For ontology extension to have its intuitive meaning, all models of an ontology should also be models of every ontology extended by it. Here we assume that the models of T are as typically defined for OWL, for example see [5]. We now define the semantics of a data source.

Definition 1 *A data source s in $\mathcal{S}$ is a pair $\langle \mathcal{A}, O \rangle$, where $\mathcal{A}$ is a ABox that consists of a set of formulas and $O \in \mathcal{O}$ is the ontology that s commits to.*

When a data source commits to an ontology, it has agreed to the terminology and definitions of the ontology. It means that for data source $s = \langle \mathcal{A}_s, O_s \rangle$, all the concepts and roles that are referenced in $\mathcal{A}_s$ should be either from the ontology O_s or $anc(O_s)$. Thus every interpretation that is a model of data source must also be a model of the ontology for that data source.

We now define an ontology perspective model of a Semantic Web space. This definition presumes that each ontology can be used to provide a different viewpoint of the Semantic Web.

Definition 2 (Ontology Perspective Model) *An interpretation $\mathcal{I}$ is an ontology perspective model of a semantic web space $\mathcal{W} = \langle \mathcal{O}, \mathcal{S} \rangle$ based on $O \in \mathcal{O}$ (written $\mathcal{I} \models_O W$) iff: 1) $\mathcal{I}$ is a model of O and 2) for each $s = \langle \mathcal{A}_s, O_s \rangle \in \mathcal{S}$ such that $O_s = O$ or $O_s = anc(O)$, $\mathcal{I}$ is a model of s.*

Based on this definition, entailment is defined in the usual way, where $\mathcal{W} \models_O \phi$ is read as "$\mathcal{W}$ O-entails ϕ".

Theoretically, each O-entailment relation (perspective) represents a set of beliefs about the state of the world, and could be considered a knowledge base. Thus, the answer to a semantic web query must be relative to a specific perspective. We now define a Semantic Web query.

Definition 3 *Given a Semantic Web Space $\langle \mathcal{O}, \mathcal{S} \rangle$, a Semantic Web query is a pair $\langle O, \rho \rangle$ where $O \in \mathcal{O}$ is the base ontology of the perspective and ρ is a conjunction of query terms q_1,....., q_n. Each query term q_i is of the form x:c or $\langle x, y \rangle$:r, where c is an atomic concept and r is an atomic role from O or ancestor of O and x, y are either individual names or existentially quantified variables.*

An answer to the query $\langle O, \rho \rangle$ is θ iff for each q_i, $\mathcal{W} \models_O \theta q_i$ where θ is a substitution for the variables in ρ.

Given the model described above, we can define a special class of ontologies called mapping ontologies. A mapping ontology between a sequence of ontologies O_i, ... ,$O_n$introduces no vocabulary of its own but extends O_i , ... , O_n and contains the axioms that map their vocabularies.

We argue that our perspectives have at least two advantages over traditional knowledge representation languages. First, the occurrence of inconsistency is reduced compared to using a global view, since only a relevant subset of the Semantic Web is involved in processing a query. Even if two ontologies have conflicting axioms, inconsistency would not necessarily happen in most perspectives other than the ones that are based on the common descendants of the conflicting ontologies. Second, the integration of information resources is flexible, i.e. two data sources can be included in the same perspective as long as the ontologies they commit to are both being extended by a third ontology.

3 DLDB2 System

3.1 Architecture

DLDB2 extends DLDB with additional reasoning capabilities and support for perspectives as defined in section 2. DLDB2 is a knowledge base system that combines a relational database management system with additional capabilities for partial OWL reasoning. It is freely available as an open source project under the HAWK framework [1].

The DLDB2 core consists of a Load API and a Query API implemented in Java. Any DL Implementation Group (DIG) compliant DL reasoner and any SQL compliant RDBMS with a JDBC driver can be plugged into DLDB2. This flexible architecture maximizes its customizability and allows reasoners and RDBMSs to run as services or even clustered on multiple machines.

It is known that the complexity of complete OWL DL reasoning is NEXPTime-complete. Our pragmatic approach is to trade some completeness for performance. The overall strategy of DLDB2 is to find the ideal balance of precomputation of inference and run-time query execution via standard database operations. Whenever DLDB2 loads a Semantic Web document, it separates TBox from instance data. TBoxes are first processed by the DL reasoner in order to compute implicit subsumptions and the results are used to create tables and views in the database. Instance data are directly loaded into the database tables. The consideration behind this approach is that DL reasoners are optimized for reasoning over ontologies, as opposed to instance data.

3.2 Loading Ontologies and Data

When DLDB2 loads an ontology, it uses Algorithm 1 to process the TBox. Note if there are instance data in the ontology, they are processed in the same way as if they come from a data source which commits to this ontology.

First the system creates tables in the database for atomic classes and roles. There are straightforward correspondences between database relations and datalog relations,

[1] http://swat.cse.lehigh.edu/downloads/hawk.html

the tables are extensional relations and the database views are intensional relations. For convenience, we use datalog-style rules to describe the handling of axioms. However, it is important to note that only acyclic datalog programs (those with non-recursive rules) can be directly translated to database view definitions.

Intuitively, predicates represent classes should be unary and predicates represent roles (in DL) should be binary. In our design, they both have two additional arguments. We use $B_{table}(x, s, o)$ and $P_{table}(x, y, s, o)$ to represent class B and role P respectively. Argument s is the source document in which a fact is stated. Argument o is the ontology to which the source document commits. Their purpose is to support perspectives described in section 2.

The system also creates database views for classes and properties in order to capture the instances through subsumptions. It does so not only for vocabularies defined in the current ontology, but also for the vocabularies in its ancestors. To differentiate multiple views of the same class (role) introduced by different ontologies, the name of a view has to include the identification of the ontology who owns it. In other words, the argument o is no longer for the predicates representing views. Instead, the names of such predicates have already taken the ontology into account. We use $B^o_{view}(x, s)$ and $P^o_{view}(x, y, s)$ to represent the view of class B and the view of role P from the perspective of ontology o respectively. This design enables DLDB2 to support perspectives and allows different perspectives to have different sets of axioms.

Next the system encodes and transfers the ontology and its ancestors to a DL reasoner through the DIG interface. The DL reasoner performs a satisfiability check and computes subsumptions. Then the system asks the DL reasoner to return the parents of each atomic class (role) and the equivalents of each class (role).

Note complex class expressions are not included in the results we get from DL reasoner through DIG. This is why we take two steps to combine the original TBox with the inferred axioms from the reasoner. First, we apply the Lloyd-Topor transformation to rewrite rules with conjunctions in the head and disjunctions in the body as described in [2]. Second, we merge T_{inf} and T into T' and eliminate redundant rules.

Then subsumption axioms in T' are translated into datalog rules if they fall into horn logic. Atomic classes or roles in these axioms are directly translated into datalog relations. Those complex class or role expressions are translated into a conjunction of datalog atoms by recursively calling the translation function defined in Table 1, where B denotes an atomic class, C, C_1 and C_2 denote classes, P denotes an atomic role and R denotes a role expression. Each equivalence axiom is translated into two datalog rules in a standard way: to prevent cycles, extensional relations instead of intensional relations are used in the body.

Translation input	**Translate to**
Trans(B, O, x, s)	$B^o_{view}(x, s)$
Trans($C_1 \sqcap C_2$, O, x, s)	Trans(C_1,O, x, s1), Trans(C_2,O, x, s2)
Trans($\exists R.C$, O, x, s)	Trans(R, O, x, y, s1), Trans(C, O, y, s2) [2]
Trans(P, O, x, y, s)	$P^o_{view}(x, y, s)$
Trans(R^-, O, x, y, s)	Trans(R, O, y, x, s)

Table 1. Translation Function

Currently DLDB2 does not support universal restrictions, thus axioms that contain such construct are ignored as well as other axioms which are out of the scope of Description Horn Logic (DHL) as defined in [2]. Finally, all the datalog rules resulting from this algorithm are translated into database view definitions. Relations have already been mapped to tables and views. Conjunctions in the rule bodies correspond to joins in SQL. Multiple rules having the same head are combined using "UNION" in SQL.

It is worth noting that although the datalog rules implemented in the DLDB2's relational database system correspond to a subset of DHL, DLDB2 does support reasoning on DL ontologies richer than DHL. For example, the axioms $A \sqsubseteq B \sqcup C$ and $A \sqsubseteq \neg B$ are both beyond the expressiveness of DHL. However, in line 6 of Algorithm 1, the DL reasoner will compute and return $A \sqsubseteq C$ assuming A and C are both atomic classes.

In general, data loading in DLDB2 is straight-forward. Each *rdf:type* triple inserts a row into a class table, while each triple of other predicates inserts a row into a role table corresponding to the predicate. If a data document imports multiple ontologies, the 'o' argument is decided by the ontology that introduces the term that the table corresponds to. However, DLDB2 materializes certain inferences at data load time, as discussed in the next section.

3.3 Special ABox Reasoning

This subsection focuses on precomputations that simplify reasoning at query time. These ABox reasoning routines along with the TBox reasoning provided by the the DL reasoner make the system complete on a significant subset of OWL DL.

OWL does not make the unique names assumption, which means that different names do not necessarily imply different objects. Given that many individuals contribute to the Web, it is highly likely that different IDs will be used to refer to the same object. Such IDs are said to be equal. A Semantic Web knowledge base system thus needs an inference mechanism that actually treats them as one object.

[2] y should be a brand new variable everytime the translation function is called.

Algorithm 1 Process TBox of an ontology

LOADONTOLOGY($O = \langle \mathcal{C}, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$)

1: **for** each atomic class B in $\mathcal{C}$ or $\mathcal{C}_i$ where $O_i = \langle \mathcal{C}_i, \mathcal{R}_i, \mathcal{T}_i, \mathcal{E}_i \rangle \in anc(O)$ **do**
2: $B^o_{view}(x, s) : -B_{table}(x, s, o), o \in anc(O)$
3: **for** each atomic class P in $\mathcal{R}$ or $\mathcal{R}_i$ where $O_i = \langle \mathcal{C}_i, \mathcal{R}_i, \mathcal{T}_i, \mathcal{E}_i \rangle \in anc(O)$ **do**
4: $P^o_{view}(x, y, s) : -P_{table}(x, y, s, o), o \in anc(O)$
5: Tell the TBox of O and its ancestors to a DL reasoner
6: For each atomic class (role), ask the DL reasoner for its parents and equivalents, add resulting axioms to T_{inf}
7: Apply Llyod-Topor transformation to axioms in T
8: $T' = T_{inf} \cup T$
9: **for** each axiom of the form $C \sqsubseteq D \in T'$ such that D is atomic class that doesn't participate in a cycle of rules **do**
10: $D^o_{view}(x, s) : -Trans(C, O, x, s)$
11: **for** each axiom of the form $R \sqsubseteq P \in T'$ such that P is atomic role that doesn't participate in a cycle of rules **do**
12: $P^o_{view}(x, y, s) : -Trans(R, O, x, y, s)$
13: **for** each pair of atomic classes C and D such that $C \equiv D \in T'$ **do**
14: $D^o_{view}(x, s) : -C_{table}(x, s, o), o \in anc(O)$
15: $C^o_{view}(x, s) : -D_{table}(x, s, o), o \in anc(O)$
16: **for** each pair of atomic roles P and R such that $P \equiv R \in T'$ **do**
17: $P^o_{view}(x, y, s) : -R_{table}(x, y, s, o), o \in anc(O)$
18: $R^o_{view}(x, y, s) : -P_{table}(x, y, s, o), o \in anc(O)$

Usually, equality is encoded in OWL as (a *owl:sameAs* b), where a and b are URIs.

In DLDB2, each unique URI is assigned a unique integer id in order to save storage space and improve the query performance (via faster joins on integers than strings). Our approach to equality is to designate one id as the canonical id and globally substitute the other id with this canonical id in the knowledge base. The advantage of this approach is that there is effectively only one system identifier for the (known) individual, nevertheless that identifier could be translated into multiple URIs. Since reasoning in DLDB2 is based on these identifiers instead of URIs, the existing inference and query algorithms do not need to be changed to support equalities.

However, in many cases, the equality information is found much later than the data that it "merges". Thus, each URI is likely to have been already used in multiple assertions. Finding those assertions is especially difficult given the table design of DLDB2, where assertions are scattered into a number of tables. It is extremely expensive to scan all the tables in the knowledge base to find all the rows that use a particular id, especially if you consider that the number of tables is equal to the number of classes plus the number of properties. For this reason, We use auxiliary tables to keep track of the tables that each id appears in.

Often times, the knowledge on equality is not given explicitly. Equality could result from inferences across documents: *owl:FunctionalProperty* , *owl:maxCardinality* and *owl:InverseFunctionalProperty* can all be used to infer equalities. DLDB2 is able to discover equality on individuals using a simple approach. If two URIs have the same value for an *owl:InverseFunctionalProperty*, they are regarded as representing the same individual. A naive approach is to test for this event every time a value is inserted into an inverse functional property table. However, this requires a large number of queries and potentially a large number of update operations. In order to improve the throughput of loading, we developed a more sophisticated approach which queries the inverse functional property table periodically during the load. This happens after a certain number of triples have been loaded into the system. The specific interval is specified by users based upon their application requirements and hardware configurations (we used 1.5 million in our experiment). This approach not only reduces the number of database operations, but also speeds up the executions by bundling a number of database operations as a stored procedure. DLDB2 also supports the same approach on *owl:FunctionalProperty*.

One of the ABox reasoning tasks is to infer implicit property assertions through the transitive property. This task can be regarded as computing a transitive closure over a directed acyclic graph. Transitive closure is typically a problematic operation for an RDBMS. Nevertheless a number of algorithms have been proposed and some systems even support a built-in operation. In our system, we must also address how these algorithms interact with other reasoning. For example, the property *isIn* is transitive, whereas its two subproperties: *isInState* and *isInRegion*, are not (and should not be) transitive.

Our solution is to periodically run an algorithm that joins the view (not the table) on the transitive property iteratively until a fixed point is reached. This algorithm also takes care of the perspectives, which allows different ontologies to independently describe a property as transitive or not.

3.4 Query Answering

The query API of DLDB2 receives queries from users, executes query operations and returns query results to them. This API currently supports SPARQL encodings of conjunctive queries as defined in section 2. During execution, predicates and variables in the query are substituted by table names and field names through translation. Depending on the perspective being selected, the table names are further substituted by corresponding database view names. Finally,

a standard SQL query sentence is formed and sent to the database via JDBC. Then the RDBMS processes the SQL query and returns appropriate results.

Since we build the class and property hierarchy when loading the ontology, there is no need to call the DL reasoner at query time. The results returned by the RDBMS can be directly served as the answer to the original query. We think this approach makes the query answering system much more efficient than conducting DL reasoning at query time. To improve the query performance. DLDB2 system independently maintains indexes without the intervention from database administrators.

4 Related Work

The C-OWL work [1] proposed that ontologies could be contextualized to represent local models from a view of a local domain. The authors suggested that each ontology is an independent local model. If some other ontologies' vocabularies need to be shared, some bridge rules should be appended to the ontology which extends those vocabularies. Compared to C-OWL, our perspective approach also provides multiple models from different views without modifying the current Semantic Web languages.

In the past few years there has been a growing interest in the development of systems that will store and process large amount of Semantic Web data. The general design goal of these systems is often similar to ours, in the sense that they use some database systems to gain scalability while supporting as much inference as possible by processing and storing entailments. However, most of these systems emphasize RDF and RDF(S) data at the expense of OWL reasoning. Some systems resemble the capabilities of DLDB2, such as KAON2 [6], which uses a novel algorithm to reduce OWL DL into disjunctive datalog programs. OWLIM [7] uses a rule engine to support a limited OWL-Lite reasoning. Minerva [10] uses DL reasoner to do TBox reasoning and a rule engine to do ABox reasoning. It is designed to be sound and complete on DHL (a subset of OWL-DL) ontologies [10]. Unlike DLDB2, both OWLIM and Minerva chose a "vertical" table design, in which all data is stored in a single "triple table". To the best of our knowledge, none of the systems above support queries from different perspectives, though BigOWLIM [9] has been reported to support 1 billion triples of artificially generated data with high performance hardware.

5 Evaluation

5.1 Performance on Benchmarks

We evaluated DLDB2 using LUBM [3] and UOBM [8]. UOBM extends LUBM with additional reasoning requirements and links between data. The evaluation is done on a desktop with P4 2.9G CPU and 1G memory running Windows XP professional. DLDB2 is configured to use Racer Pro 1.9 as DL reasoner and MySQL 5.0 community version. For comparison, we also tested Minerva (version 1.1.2) on the same machine. IBM DB2 Express-C was used as Minerva's backend database.

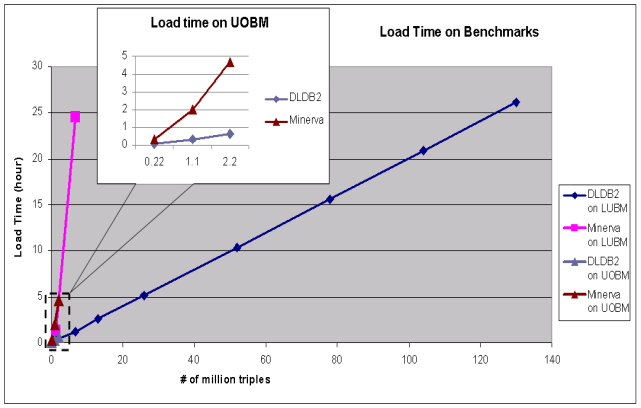

Figure 1. Load time on Benchmarks

The main diagram of Figure 1 shows the load times on LUBM. The datasets were LUBM(x,0) for $x \in \{10, 50, 100, 200, 400, 600, 800, 1000\}$. It shows that DLDB2 exhibits linear behavior in the 0 to 130 million triples range. In comparison, Minerva could only finish LUBM(50,0) within the same time period.

The smaller diagram on Figure 1 shows the load time of DLDB2 and Minerva on UOBM. Note UOBM only provides datasets in three sizes and has no publicly available data generator. The largest dataset, Lite-10 has 2.2 million triples. DLDB2 is faster on loading than Minerva, which materializes all inferred facts at load time.

Figure 2 shows the query response time of DLDB2 and Minerva on UOBM. DLDB2 is faster than Minerva on 6 queries: query 1,2,5,7,9, and 11. Most of the queries can be finished under 20 seconds. We noticed the query response time of query 3 and query 6 are increasing drastically as the size of the knowledge base increases. They both involve joining two views with a large number of records. We suspect this is due to the performance of underlying RDBMS, as it is known that MySQL has serious performance issues on joining views. Query 10 is also very slow on larger datasets as it requires a large view to join with itself.

DLDB2 is complete on all the queries in LUBM. As shown in Table 2, DLDB2 is complete on 10 out of the 13 queries in UOBM lite. Two of the queries (query 2 and 9) are not complete due to universal restrictions and one of them (query 13) is due to cardinalities. In terms of completeness on ABox reasoning, DLDB2 is close to DHL

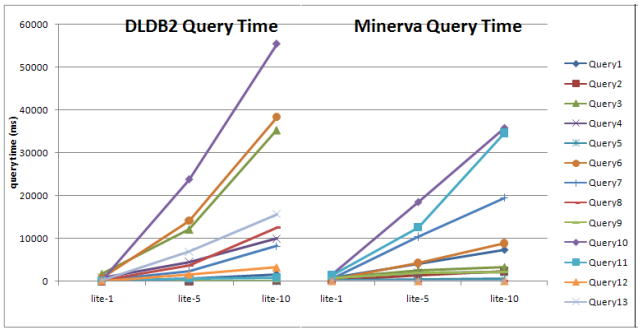

Figure 2. Query Response Time on UOBM

	Q1	Q2	Q3-8	Q9	Q10-12	Q13
DLDB2	100%	95%	100%	0%	100%	80%
Minerva	100%	100%	100%	100%	100%	61%

Table 2. Completeness on Lite-5

since it implements all the datalog rules except universal restrictions. On the other hand, DLDB2 supports reasoning on individual equalities. In addition, DLDB2 uses a DL reasoner to compute TBox entailments. Thus DLDB2 supports some subset of description logics that is beyond DHL. This is exemplified by query 13 of UOBM, where TBox reasoning makes DLDB2 more complete than Minerva.

Note DLDB2 supports queries across multiple ontologies, but currently no standard benchmark evaluates this capability. Next we examine the performance of DLDB2 using real-world data that commits to multiple ontologies. We also evaluates the use of mapping ontologies and perspectives.

5.2 Multi-ontology Evaluation

In order to evaluate our system in a more realistic setting, we loaded 24 million triples that commit to 9 different

Data	Description	# of Triples
Geographic Data	data of states and regions in US	578
Citeseer	online publications	7,430,380
DBLP	peer-reviewed publications	16,073,209
Census Data	populations by states	314
NSF Awards	50 recent award of each state	460,246
University	US universities from wikipedia	16,767
107th Congress	members of 107th Congress	2,628
Bill Data	politician bills of 107th Congress	75,711
AIGP	AI researchers with their degree info and advisors	32,716
Total		24,092,549

Table 3. Data sources

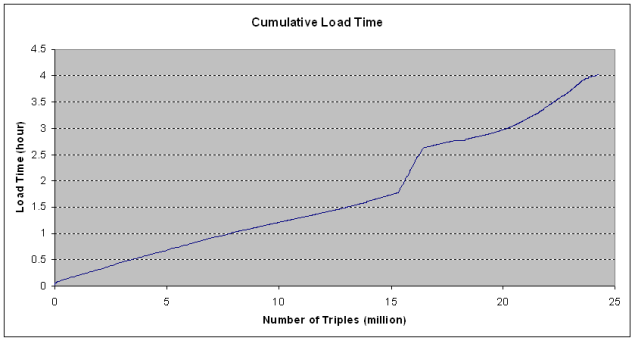

Figure 3. Cumulative Load Time

ontologies. This data set was extracted from various Internet resources on academic research or e-government domain and was converted to OWL. Table 3 summarizes the data sources in the sequence we loaded the data. This evaluation is done on a SUN workstation with dual 2G CPU and 10G memory. The settings of database and DL reaoner are the same as the last section. Figure 3 shows the cumulative load time [3]. Note the steep slope between 15.3 and 16.4 million triples corresponds to loading of some files from DBLP data. The exact reason for this steepness is yet unknown, but the slope of the curve above that area indicates that it is not the beginning of exponential behavior.

Based on our notion of query in Definition 3, DLDB2 allows users to choose the perspective associated with each query, i.e. the mappings they approve. In our evaluation, we first create the mappings between pairs of related ontologies, and then map them to popular ontologies like FOAF and Dublin Core. Totally, we created 16 mappings with 46 axioms. We also created 4 common perspectives such as "academic" and "e-government"; each perspective extends (imports) some mappings of interest. Additionally, we created 169,023 *owl:sameAs* statements to map URIs from different sources but refer to the same individual. Given these mappings, we could now ask more interesting queries. The following are some examples.

Query 1. Politician bill by subject and population.
Someone may be curious about the topics of bills sponsored by congressmen and the size of their states. For example, consider a query to find politician bills related to "energy" that are sponsored by a politician from a state with population less than 700,000. We use the "e-government" perspective to issue the query and find results like James Jeffords from Vermont sponsored a bill "Renewable Energy and Energy Efficiency".

Query 2. Academic influence of a researcher.
One way to get a sense of the academic influence of a re-

[3]The total number of triples in Figure 3 is larger than that in Table 3 because of the ontology, mapping and owl:sameAs statements.

	Response time (sec)	# of results
Query 1	0.512	6
Query 2	0.622	180
Query 3	20.808	1140

Table 4. Results of Queries

searcher is to look at the journals their academic descendants publish in. A sample query is to find the journals in which papers are published by the people who are influenced by Eugene Charniak. We use the "academic" perspective to issue the query. The query results show academic descendants such as James Hendler and Kutluhan Erol, and include journals such as Computational Linguistics, Cognitive Science, Neural Networks, Parallel Computing and the Journal of Web Semantics.

Query 3. Correlation between Congressional interests and university funding.

Someone wants to see if states that have congressmen with specific interests in technology tend to get funding for their large universities. This query is to find universities (with more than 10,000 undergraduate students) that have received NSF awards, which are located in the state of congressmen who sponsored bills on "technology". We use the "all" perspective that includes all mappings across academic and e-government to issue the query.

The results of the above queries are summarized in Table 4. Note the mappings that integrate different sources are critical to answer these queries. At load time, DLDB2 creates views based on the map ontologies and perspectives. At query time, the appropriate views are queried to provide answers from the chosen perspective. As we can see that the third query is much slower than the first two, which are under 1 second. The reason is that the first two are about a specific person or a small set of states, while the third one is not particularly selective and involves joins on larger data sets.

6 Conclusion and Future Work

In this paper we present our DLDB2 system, which takes advantage of the scalability of relational databases and the inference capability of description logic reasoners. Our evaluation shows that DLDB2 scales well when loading up to 130 million triples. It also finishes most queries under 20 seconds on smaller data sets. Based on ontology perspectives which use existing language constructs, DLDB2 can support queries from different view points. Real-world data using multiple ontologies and realistic queries show that DLDB2 has achieved this capability without any significant performance degradation.

Although we believe our work is a first step in the right direction, we have discovered many issues that remain unsolved. First, to ensure the freshness of data, we plan to support efficient updates on documents. Second, we will investigate query optimization techniques that can improve the query response time. Third, we will formally define the language that our system is complete on and theoretically prove the system's completeness.

7 Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. IIS-0346963. The authors would like to thank developers of Racer Pro for their software license. Graduate students Abir Qasem, Ameet Chitnis and Fabiana Prabhakar also contributed to the system described here.

References

[1] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Proc. of the 2003 Int'l Semantic Web Conf. (ISWC 2003)*, LNCS 2870, pages 164–179. Springer, 2003.

[2] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of WWW2003*, Budapest, Hungary, May 2003. World Wide Web Consortium.

[3] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[4] J. Heflin and Z. Pan. A model theoretic semantics for ontology versioning. In *Proc. of the 3rd International Semantic Web Conference*, pages 62–76, 2004.

[5] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logics satisfiability. In *Proceedings of the Second International Semantic Web Conference*, pages 17–29, 2003.

[6] U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ description logic to disjunctive datalog programs. In *Proc. of the 9th International Conference on Knowledge Representation and Reasoning*, pages 152–162, 2004.

[7] A. Kiryakov. OWLIM: balancing between scalable repository and light-weight reasoner. In *Developer's Track of WWW2006*, 2006.

[8] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *ESWC*, pages 125–139, 2006.

[9] D. Ognyanoff, A. Kiryakov, R. Velkov, and M. Yankova. A scalable repository for massive semantic annotation. Technical Report D2.6.3, SEKT project, 2007.

[10] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, , and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In *Proc. of Asia Semantic Web Conference (ASWC)*, pages 429–443, 2006.