A Semantic Web Knowledge Base System that Supports Large Scale Data Integration

Zhengxiang Pan, Yingjie Li and Jeff Heflin

Department of Computer Science and Engineering, Lehigh University
19 Memorial Dr. West, Bethlehem, PA 18015, U.S.A.
{zhp2, yil308, heflin}@cse.lehigh.edu

Abstract. A true Semantic Web knowledge base system 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. We present our DLDB3 system that supports large scale data integration, and is provably sound and complete on a fragment of OWL DL when answering extensional conjunctive queries. 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. The ABox inferences from cyclic axioms are materialized at load time, while other inferences are computed at query time. Instance data are directly loaded into the database tables. We evaluate the system using synthetic benchmarks and compare performances with other systems. We also validate our approach on data integration using multiple ontologies and data sources.

1 Introduction

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 ontology, perspective semantics provides a framework to integrate data sources using different ontologies. This framework uses only standard OWL axioms and hence would not add any additional reasoning complexity to the knowledge base system. Another unique feature of this framework is that different viewpoints regarding the integration (or mapping) could coexist in one knowledge base system, even when they are contradictory. We think this is important for the Semantic Web, which is inherently distributed and inconsistent.

Based upon this framework, we built DLDB3, a scalable Semantic Web knowledge base system that allows queries from different points of view. DLDB3 has major improvements over DLDB2 [16], including a novel approach to handle cyclic (recursive) axioms in relational databases. Although relational databases are optimized for scalable query answering, they usually require special purpose algorithms to handle recursive queries [17]. Our approach identifies cyclic axioms that cannot be directly handled by

precomputed database views and materializes the inferences entailed by these axioms. Our resulting system is complete for 10 of 14 queries in UOBM DL benchmark and all of the LUBM [8] queries, and can load 130 million triples with 24 hours.

The kinds of cycles that lead to incompleteness in a database view approach like DLDB include transitive axioms and property restrictions where the same class appears on both sides of a general concept inclusion axioms (e.g. $\exists P.C \sqsubseteq C$). Wang et al. [20] surveyed 1275 ontologies and only found 39 (3%) that contained a transitive property. In our own analysis of Swoogle's [5] Semantic Web data collection, we could not find any cycles that involves an existential restriction. Only 1.6% out of 16285 ontologies define transitive properties. The number of transitive properties defined in all ontologies is 459, merely 1.4% of the total number of properties. Given that such cycles are rare, we believe that they should be handled as special cases, and our RDBMS-based architecture should be preserved. We hypothesize that selectively materializing these axioms will only lead to minor increases in load time and overall repository size using our approach. In prior work, we added materialization of transitive properties to DLDB2 [16]. In this paper we generalize this to materialize all other cyclic axioms. A key element to this approach is ensuring that the materialized data only appears in the appropriate perspectives.

In what follows, we first introduce ontology perspectives. We then present the DLDB3 system's reasoning algorithms, architecture, design and implementation with a focus on how the reasoning is achieved and perspectives are supported. Finally we evaluate the system using benchmarks as well as 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 [9]. 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. Note 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 returns 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 $\mathcal{T}$ are described by the semantics of OWL, for example see [10]. We now define the semantics of a data source.

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

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 $anc(O_s)$.

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 $\mathcal{W} = \langle \mathcal{O}, \mathcal{S} \rangle$, 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.

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, the inconsistency would only be propagated to perspectives based on 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 A Scalable Algorithm for Semantic Web Query Answering

Our approach was inspired and based on the work of Description Horn Logic (DHL)[7], a fragment of DL that can be translated into logic programming. Although the DHL

work has a general description on how the datalog programs can be implemented on relational databases, details or working algorithms are not present, especially for handling (cyclic) recursive rules. To our best knowledge, none of the publicly available Semantic Web knowledge base systems has took this route of combining datalog programs with relational databases. Although deductive databases directly implement datalog, their techniques are currently not mature enough for handling large scale data.

3.1 Defining the Language

The DHL language and its mapping to other formalisms has been described in detail in [7]. Here we provide a quick summary for the convenience of discussion. Formally, in DHL:

Concepts are defined as	Roles are defined as
$D ::= A \mid D_1 \sqcap D_2 \mid \forall R.D$	$R ::= P \mid P^-$
$C ::= A \mid \exists R.C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2$	Where P denotes atomic role.
Where A denotes atomic concept.	
The axioms have form:	The assertions have form:
$C \sqsubseteq D$	$a : C$
$R_1 \sqsubseteq R_2$	$(a, b) : R$
$R^+ \sqsubseteq R$ means R is a transitive property	where a, b are named individuals.
$Func(R)$ means R is a functional property [1]	

Translation input	**Translate to**
Trans(A, x)	$A(x)$
Trans($C \sqsubseteq D, x$)	Trans(C, x)→ Trans(D, x)
Trans($C_1 \sqcap C_2$, x)	Trans(C_1, x) ∧ Trans(C_2, x)
Trans($C_1 \sqcup C_2$, x)	Trans(C_1, x) ∨ Trans(C_2, x)
Trans($\exists R.C$, x)	Trans(R, x, y) ∧ Trans(C, y)
Trans($\forall R.D$, x)	Trans(R, x, y) → Trans(D, y)
Trans($R_1 \sqsubseteq R_2$, x, y)	Trans(R_1, x, y) → Trans(R_2, x, y)
Trans($R^+ \sqsubseteq R$)	Trans(R, x, y) ∧ Trans(R, y, z) → Trans(R, x, z)
Trans(P^-, x, y)	Trans(P, y, x)
Trans(P, x, y)	P(x, y)

Table 1. Translation Function from DL axioms to Horn rules

The DL constructors and axioms can be recursively mapped to First Order Logic rules. By applying Lloyd-Topor transformation to rewrite rules with conjunctions in the

[1] This feature will be handled separately in our system, and will not be translated into horn rules.

head and disjunctions in the body, we can ensure the resulting rules are all horn rules. The equivalences can be rewritten into two subsumptions.

3.2 Reasoning

Our approach uses a relational database to store, manage and retrieve instance data. For each predicate P (class or property) that is defined in an Ontology, we create a dedicated table P_{table} to store the explicit facts. In order to reflect the perspective, we use P^O_{table} to represent the explicit or materialized facts of P that committed to ontology O or $anc(O)$. Note in actual implementation, P^O_{table} doesn't have to be a real table in the database, it can be implemented as a filter of ontology commitment on top of P_{table}. When facts are stored as tuples in the database, their "provenance" or "source" information are also preserved in the form of the identifier of the ontology they commit to. We use P^O_{view} to represent all instances of P, explicit or implicit, from the perspective based on O. For convenience, we define extensional disjunctive normal form.

Definition 4 *A logical formula in first order logic is in extensional disjunctive normal form (E-DNF) if and only if it is a disjunction of one or more conjunctions of one or more atoms, where all predicates correspond to extensional tables.*

Algorithm 1 shows how the reasoning is conducted in our approach. First, we convert axioms in a given ontology into a set of horn rules using the translation function defined above. Note the DL axioms can be reasoned and enriched by a DL reasoner, but it is not necessary in our algorithm. Then for a set of horn rules with a common head, we convert them into a single FOL rule, where the left hand side is in E-DNF. This conversion is processed recursively by applying Modus Pones in a reverse direction until all predicates are primitive (correspond to extensional tables). Next, we separate acyclic portion of disjuncts from the cyclic portion. For acyclic rules, we create view for the head's predicate using straightforward translation from rule to SQL, where conjunctions correspond to joins on common variables and disjunctions correspond to UNION in SQL. Each ontology has a distinct view for each predicate defined by it or one of its ancestors. Periodically when data is being loaded, we use Algorithm 2 to handle cyclic rules left from Algorithm 1. During the computation, we materialize new tuples in the tables so that the computation does not need to be invoked at query time. The materialization would also set the ontology commitment information to be the ontology that invokes this round of computation, such that these "derived" facts can be correctly managed using perspectives. When answering extensional conjunctive query $\langle O, \rho \rangle$ as defined in section 2, each predicate P is directly translated into a view P^O_{view} that not only contains the implicit facts through subsumptions, but also the explicit and materialized facts in the underlying table.

Theorem 1 *Given a knowledge base consists of ontologies and data sources in DHL, the reasoning algorithms described above is sound and complete w.r.t any Semantic Web Query $\langle O, p \rangle$.*

PROOF. (Sketch) When we load an ontology O, the axioms of O and $anc(O)$ are used to generate database views for the perspective based on O. This is consistent with the

Algorithm 1 Load an ontology into the knowledge base

LOADONTOLOGY(O)

Translate axioms in O and $anc(O)$ into a set of horn rules H
Initialize sets F, F*, each holds a set of FOL rules
for each predicate P in H **do**
    Convert the set of horn rules whose heads' predicate is P to L, where the body of L is a E-DNF FOL formula
    for each disjunct B in L such that one of the predicates is P but with different arguments **do**
        remove B from L and add B as a disjunct into the body of L^* where the head of L^* is also P
        $F = F \sqcup L$, $F^* = F^* \sqcup L^*$
    end for
end for
for each FOL rule $L \in F$ **do**
    create view P^O_{view} as a SQL expression that joins the conjuncts and unions the disjuncts. Each predicate A in the body of L is translated into A^O_{table}.
end for

Algorithm 2 Fix point computation of cyclic rules

FIXPOINTCOMPUTE(F^*, O)

repeat
    for each FOL rule L^* in F^*, where the head's predicate is P **do**
        Translate the body of L^* into SQL query q, where each predicate A are replaced by views A^O_{view}
        Execute q, add new tuples into P^O_{table}
    end for
until A fix point has reached, which means none of the iterations generates new tuples

Ontology Perspective Model in 2. It has been shown in [7] that the translation from DL axioms in DHL to horn rules preserves semantic equivalence. Further on, the conversion to extensional disjunctive normal form in essence is backward chaining and syntactical rewriting of rules by applying Modus Pones in a reverse direction. This kind of conversion does not alter their semantics and logical consequences in FOL. The separation of acyclic rules from cyclic rules is a variation of Lloyd-Topor transformation for disjunctions in the body. Thus again, the changes are only syntactic. For the acyclic rules, their correspondence in SQL has been shown in previous work such as [19]. For the cyclic rules, the correctness of fix point algorithms has also been proved in [19]. To summarize, the query on each database view of a predicate A would get the exact same set of facts as a sound and complete reasoning algorithm would infer for $A(x)$ since the algorithms behind the view exercise all and only the axioms in the knowledge base that the perspective represents.

4 Implementation of DLDB3 System

4.1 Architecture

DLDB3 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 [2].

The DLDB3 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 DLDB3. 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 DLDB3 is to find the ideal balance of precomputation of inference and run-time query execution via standard database operations. The consideration behind this approach is that DL reasoners are optimized for reasoning over ontologies, as opposed to instance data.

Following our algorithm described in the last section, creating tables corresponds to the definition of classes or properties in ontology. Each class and property has a table named using its URI.

Normally, the 'sub' and 'obj' fields are foreign keys from the 'ID' field of the class tables that are the domain or range of the property, respectively. However, if the property's range is a declared data type, then the 'object' field is of the corresponding data type (RDF and OWL use XML Schema data types). Currently DLDB3 supports integer, float and date in addition to string.

DLDB3's table schema is different from the vertical (also called "schema-oblivious") approach [18], which uses a single table for storing both RDF/S schemata and resource descriptions under the form of triples (subject-predicate-object). Note, when new ontologies are loaded, the correspondent tables are created for their classes and properties.

We think the table design of DLDB3 has two advantages over the vertical approach. First, it will contain smaller tables than the vertical scheme. Second, it preserves the data types of the properties and hence can directly and efficiently answer queries involving numeric comparison, such as finding individuals whose ages are under 21. Compared to the traditional relational model, where properties correspond to columns, multi-valued properties (attributes) in DLDB3 don't need to be identified at ontology design time.

In addition to the basic table design, some details should be taken into account when implementing the database schemas for the system. First, we need a scheme for naming the class and property tables. Note, using the full URI will not work, because these URIs often exceed the RDBMS's limits on the length of table names. However, the local name is also insufficient because many ontologies may include the same local name. So we assign a unique sequence number to each loaded ontology. Then each table's name is a class or property's local name plus its ontology's sequence number. This is supported by an extra table:

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

`ONTOLOGIES_INDEX(Url, SeqNum)`
that is used to register and manage all the ontologies in the knowledge base. The sequence number will be assigned by the system when an ontology is first loaded into the database.

Since each row in the class tables corresponds to an instance of the class, an 'ID' field is needed here to record the ID of the instances. The rest of the data about an instance is recorded using the table per property (a.k.a. decompositional) approach. Each instance of a property must have a subject and an object, which together identify the instance itself. Thus the 'sub' and 'obj' fields are set in each table for property.

Sometimes it is important to know which document a particular statement came from, or how many documents contain a particular statement. We support this capability by including a 'Src' field in each class and property table. Together with other fields, it serves as the multiple-field primary key of the table. In other words, the combination of all the information of one record identifies each instance stored in the knowledge base. In order to support perspectives described in section 2, the ontology to which the individual data commits is also needed. Thus an 'Onto' field is include in the tables. This field is used to record the committed ontology from Algorithm 1 and 2. An example of class and property tables might be:
`STUDENT:1(Id, Src, Onto)`
`TAKESCOURSE:1(Sub, Obj, Src, Onto)`

In order to shrink the size of the database and hence reduce the average query time, DLDB assigns each URI a unique numeric ID in the system. We use a table:
`URI_INDEX(Uri, Id)`
to record the URI-ID pairs. Thus, for a particular resource, its URI is only stored once; its corresponding ID number will be supplied to any other tables. Since the typical URI is often 20-50 characters long and the size of an integer is only 4 bytes in the database, this leads to a significant savings in disk space. Furthermore, query performance is also improved due to the faster joins on integers instead of strings and the reduced table size. By discriminating the DataType properties and ObjectType properties, the literals are kept in their original form without being substituted by ID numbers. The reason why we don't assign IDs to literals is 1) literals are less likely to be repeated in the data; and 2) literals are less likely to be joined with other tables because they are never used as database keys.

Unsurprisingly, the tradeoff of doing the URI-ID translation is an increase in load time. We use a hash table to cache URI-ID pairs found recently during the current loading process. Since URIs are likely to repeat in one document or neighboring documents, this cache saves a lot of time by avoiding lookup queries when possible.

When DLDB3 loads an ontology, it uses Algorithm 1 to do reasoning. 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.

It is worth noting that DLDB3 uses a DL reasoner to make inferences on DL axioms before these axioms are translated into horn rules. In result, although the horn rules implemented in the DLDB3's relational database system correspond to a subset of DHL, DLDB3 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,

the DL reasoner will compute and return $A \sqsubseteq C$ assuming A and C are both atomic classes. Unfortunately, it is difficulty to characterize this expressivity formally since some other axioms involving disjunction or negation are not supported.

In general, data loading in DLDB3 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 value of the 'Onto' field is decided by the ontology that introduces the term that the table corresponds to. However, DLDB3 materializes certain inferences at data load time, as discussed in the next sections.

4.2 Inference on Individual Equality

This subsection focuses on precomputations that simplify reasoning at query time. These ABox reasoning routines along with the rule-based reasoning algorithm 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. Usually, equality is encoded in OWL as (a *owl:sameAs* b), where a and b are URIs.

In DLDB3, 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 DLDB3 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 DLDB3, 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. DLDB3 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. DLDB3 also supports the same approach on *owl:FunctionalProperty*.

4.3 Handling Different Kinds of Cyclic Axioms

Although Algorithm 1 deals with cyclic axioms in general, our implementation handles two categories differently. Class and Property Equalities is a form of cyclic axioms. However, they do not need fix point computation in our algorithm since they are solved during the atom expansion process. The fundamental difference between these equalities and other forms of cyclic axioms such as the transitive property is that they do not involve self-joinings or iterative procedures. They simply require that we synchronize the subsumptions between two equivalent terms. For named classes and properties, this synchronization has been taken care of by the DL reasoner.

In actual implementation, transitive properties are obvious cyclic axioms and hence they do not need to be identified by translating into FOL rules. Our solution is to periodically run an algorithm that self-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. The other forms of cyclic axioms are handled by repeating iterations until no new tuples are generated. For new tuples generated during the iterations, their 'onto' field is set to the value of the ontology that invokes this round of fix point computation (as shown in Algorithm 2).

4.4 Special Handling on Domain and Range

Normally, DHL allows universal restrictions on the right hand side. Domain and range axioms are both special cases of universal restriction ($\top \sqsubseteq \forall P.C$ and $\top \sqsubseteq \forall P^{-}.C$, respectively). The reasoning algorithm would include such rules and their corresponding SQL expressions in the view definition of classes. Our initial implementation experience shows this kind of axioms can lead to efficiency issues at query time. In particular, when the property involved has many triples, this leads to inference of class membership for a large number of instances. However, since OWL-DL requires that every instance has a type, these inferences are possibly redundant with explicit information on the knowledge base. Note, explicit universal restrictions that involve specific classes on the left hand side usually are not as bad as domain and range, simply because the join would reduce the number of facts that need to be compared with existing facts.

In order to improve the query efficiency, DLDB3 handles domain and range axioms at load time. Following the same translation method that translate the Horn rules into SQL expressions, we execute these expressions periodically during load time and effectively materialize the new facts into the corresponding tables. Then at query time, there is no need to invoke the inferences for domain and range axioms. Our initial analysis

has shown that this special treatment for domain and range axioms can improve the overall performance of the system, considering the same domain or range class could be queried over and over. Note this special handling would not alter the soundness and completeness of the reasoning algorithm.

4.5 Query Answering

The query API of DLDB3 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. DLDB3 system independently maintains indexes without the intervention from database administrators.

5 Related Work

The C-OWL work [2] 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.

Our work differs from deciding the conservative extensions [14] in that we do not study the characteristics of the logical consequences of integrating two ontologies. Instead, we focus on how to efficiently integrate data that commits to those ontologies assuming that the logical consequences have been explored and approved by the user.

In order to improve the scalability of ABox reasoning, a number of "tractable fragments" of OWL have been proposed and studied. Compare to DHL, DL-Lite [3] supports a limited form of existential restrictions on both sides of class inclusion. It also supports negation on classes and properties. However, it does not support transitive properties. EL++ [1], on the other hand, supports qualified existential restrictions and negation on classes but does not support inverse properties. Both DL-lite and EL++ do not support universal restrictions. In terms of expressiveness, DHL is more or less close to those subsets of OWL DL. It is noteworthy that in the upcoming W3C recommendation OWL 2, EL++ and DL-Lite are the bases of EL profile and QL profile respectively.

In recent 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 DLDB3, such as KAON2 [11], which uses a novel algorithm to reduce OWL DL into disjunctive datalog programs. SwiftOWLIM [12] uses a rule engine to support a limited OWL-Lite reasoning. SOR [13] uses DL reasoner to do TBox reasoning and a rule engine to do ABox reasoning. SHER [6] aims at scalable ABox reasoning using a summary of ABox. To the best of our knowledge, none of the systems above supports queries from different perspectives.

6 Evaluation

6.1 Performance on Benchmarks

We evaluated DLDB3 using LUBM [8] and UOBM (DL)[15]. UOBM extends LUBM with additional reasoning requirements and links between data. The evaluation is done on a desktop with P4 3.2G CPU and 3G memory running Windows XP professional. We configured DLDB3 to use Pellet 2.0 as its DL reasoner and MySQL 5.0 community version as its backend RDBMS. For comparison, we also tested SHER, SOR (version 1.5), KAON2 and SwiftOWLim (v2.9) on the same machine. IBM DB2 Express-C V9.5 was used as SHER and SOR's backend database.

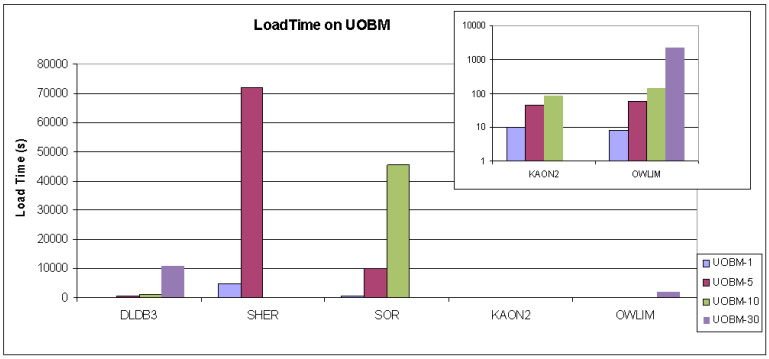

Fig. 1. Load time on Benchmarks

Note UOBM only provides datasets in four sizes and has no publicly available data generator. The largest dataset, UOBM-30 has 6.6 million triples. In our experiment, DLDB3 can load 130 million triples from LUBM(1000,0) with 24 hours and be complete on all the queries in LUBM.

The smaller diagram on Figure 1 shows the load time of KAON2 and SwiftOWLim on UOBM. These two systems are memory based so that they are fast at loading. However, their scalability are limited by the memory size. KAON2 could not finish reasoning

on UOBM-10 in our experiment. SHER, SOR and DLDB3 all use a backend RDBMS and hence would scale better than memory based systems. DLDB3 is faster on loading than SOR and SHER. It is reasonable for SOR since it materializes all inferred facts at load time. For reasons that we cannot explain, SHER failed to load datasets larger than UOBM-5. SOR did not finish the loading of UOBM-30 within a 72 hours period in our experiment.

Figure 2 shows the average query response time (average on 14 queries) for all systems on UOBM. DLDB3 is faster than KAON2 and SHER on all datasets, and keeps up with SwiftOWLim as the size of the knowledge base increases. The standard deviation on query response times gives no surprise: DLDB3 has higher variation than SwiftOWLim. For DLDB3, all the queries can be finished under 4 minutes across the datasets.

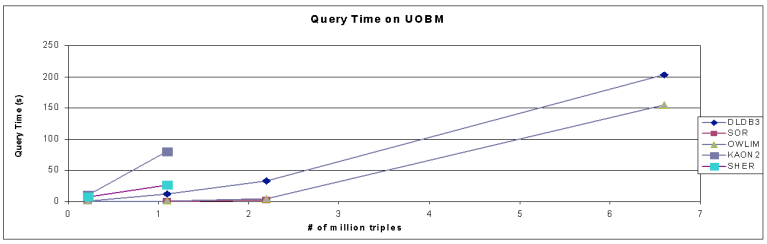

Fig. 2. Query Response Time on UOBM

As shown in Table 2, DLDB3 is complete on 11 out of the 14 queries in UOBM. Two of the queries (Q12 and Q14) are not complete due to cardinalities. Another query (Q13) involves inference using negation. SwiftOWLim is incomplete on only one query (Q13); SOR is incomplete on two queries (Q12 and Q14) and SHER is only complete on 6 queries.

	Q1 - Q11	Q12	Q13	Q14
DLDB3	100%	80%	96%	0%
SOR	100%	63%	100%	63%
SwiftOWLim	100%	100%	96%	100%

Table 2. Completeness on UOBM-1

6.2 Multi-ontology Evaluation

Both LUBM and UOBM only have a single ontology in their test dataset, which means they cannot be used to test the system's capability on data integration. In addition,

both benchmark ontologies contain no cycles besides transitive properties. In order to empirically validate our implementation on perspectives and cyclic axiom handling, we used a synthetic data generator to generate a multi-ontology test dataset. The details about the data generator is described in [4]. The dataset we chose features 10 domain ontologies and 10 mapping ontologies that map between the domain ontologies, the expressivity of the ontologies was limited to DHL. There are 20,000 data files and about 1 million triples in total. 10 random generated queries associated with one particular ontology were picked as test query. Note the ontologies in this dataset have a number of cyclic axioms, some of them form cycles across ontologies.

The experiment set-up was the same as the UOBM benchmark described above. KAON2 and DLDB3 were tested using this multi-ontology dataset. Since KAON2 is proved to be sound and complete on DHL, the results of KAON2 is used as reference. In order to verify the correctness of DLDB3, each test queries was issued using different perspectives. For a query $\langle O, \rho \rangle$, the reference result sets were collected by loading the ontologies and data sources that are included in the perspective model based on O (see definition 2) into KAON2, and then issue the conjunctive query ρ to KAON2.

All the query results from DLDB3 match the references from KAON2. This experiment has shown that DLDB3 is sound and complete w.r.t the logical consequences defined in section 2, and correctly implements the algorithm that handles cyclic axioms.

We also did some initial analysis on the scalability of the perspective approach. Figure 3 shows that as the number of ontologies included (though mappping) in the perspective increases, the query response time would increase. However, more ontologies bring more results to the query, which at large extent justifies the increase of response time.

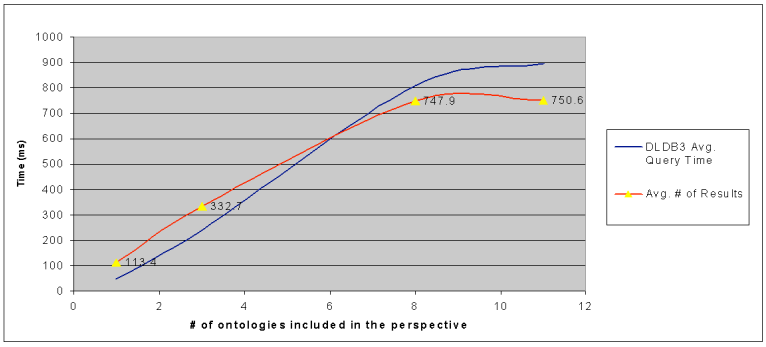

Fig. 3. Number of Ontologies V.S. Avg. Number of Query Results

On the other hand, as shown in Figure 4 the depth of the mapping (the maximum length of the mapping chain from the query term to the term that data commits to) only has small impact on the query response time. Again, when compared with the number of results, the increase of query response time is justified. Overall, we have seen that the

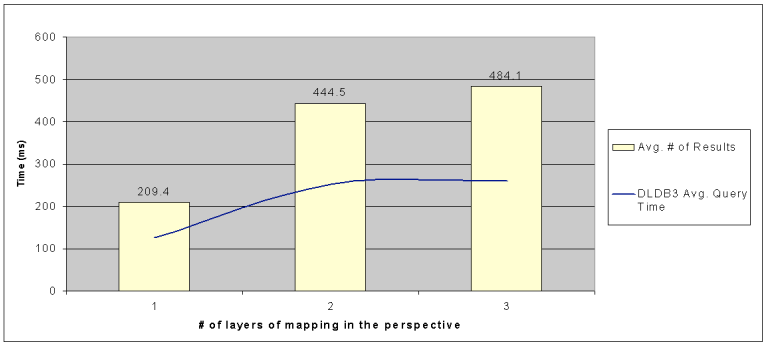

Fig. 4. Depth of Mapping V.S. Avg. Number of Query Results

query performance of DLDB3 would degrade gracefully as the depth or breadth of the perspective increases, but largely due to the gaining of new results through integration.

7 Conclusion and Future Work

In this paper we present our DLDB3 system, which takes advantage of the scalability of relational databases and the inference capability of description logic reasoners. Our scalable querying answering algorithm is guaranteed to be sound and complete on DHL. Our evaluation shows that DLDB3 scales well both in load and query comparing to other systems. It has achieved a good balance between scalability and query completeness. Based on ontology perspectives which use existing language constructs, DLDB3 can support queries from different view points. Real-world data using multiple ontologies and realistic queries show that DLDB3 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 evaluate our system's capability on perspectives more extensively and comprehensively using real-world datasets.

8 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 Tim Finnin of UMBC for providing access to the Swoogle's index of URLs. Graduate students Abir Qasem also contributed to the evaluations in this paper.

References

1. F. Baader, S. Brand, and C. Lutz. Pushing the el envelope. In *In Proc. of IJCAI 2005*, pages 364–369. Morgan-Kaufmann Publishers, 2005.
2. 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.
3. D. Calvanese, D. Lembo, M. Lenzerini, and R. Rosati. Tailoring owl for data intensive ontologies. In *In Proc. of the Workshop on OWL: Experiences and Directions*, 2005.
4. A. Chitnis, A. Qasem, and J. Heflin. Benchmarking reasoners for multi-ontology applications. In *In Proc. of Workshop on Evaluation of Ontologies and Ontology-Based Tools*. ISWC 07, 2007.
5. L. Ding, T. Finin, A. Joshi, Y. Peng, R. Pan, and P. Reddivari. Search on the semantic web. *IEEE Computer*, 10(38):62–69, October 2005.
6. J. Dolby, A. Fokoue, A. Kalyanpur, L. Ma, E. Schonberg, K. Srinivas, and X. Sun. Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In *International Semantic Web Conference*, pages 403–418, 2008.
7. 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.
8. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
9. 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.
10. 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.
11. 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.
12. A. Kiryakov. OWLIM: balancing between scalable repository and light-weight reasoner. In *Developer's Track of WWW2006*, 2006.
13. J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. Sor: a practical system for ontology storage, reasoning and search. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1402–1405. VLDB Endowment, 2007.
14. C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *In Proc. of IJCAI-2007*, pages 453–459. AAAI Press, 2007.
15. 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.
16. Z. Pan, X. Zhang, and J. Heflin. Dldb2: A scalable multi-perspective semantic web repository. In *Web Intelligence*, pages 489–495, 2008.
17. G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory Pract. Log. Program.*, 8(2):129–165, 2008.
18. Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Proc. of the 4th International Semantic Web Conference*, pages 685–701, 2005.
19. J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, Rockville, MD, 1988.
20. T. D. Wang, B. Parsia, and J. Hendler. A survey of the web ontology landscape. In *Proc. of the 5th Int. Semantic Web Conference (ISWC 2006), Athens, Georgia*, 2006.