Document-Centric Query Answering for the Semantic Web

Yuanbo Guo[1] Jeff Heflin
Computer Science & Engineering Department, Lehigh University
yug2@lehigh.edu *heflin@cse.lehigh.edu*

Abstract

In this paper, we propose document-centric query answering, a novel form of query answering for the Semantic Web. We discuss how we have built a knowledge base system to support the new queries. In particular, we describe the key techniques used in the system in order to address scalability issues. In addition, we show encouraging experimental results.

1. Introduction

The Semantic Web [1] is a major endeavor to enhance the current Web with semantically rich content. OWL [2] is a Semantic Web standard language for representing domain ontologies on the Web. OWL Lite [2] is a well studied sublanguage of OWL. A salient feature of OWL Lite is that it logically corresponds to the description logic $\mathcal{SHIF}(\mathbf{D})$ [6]. Thus, from the logical point of view, an OWL Lite document can be seen as a description logic (DL) knowledge base, written $<\mathcal{T},\mathcal{A}>$, which consists of the TBox $\mathcal{T}$, i.e., the ontology, and the ABox $\mathcal{A}$, i.e., instance data that commit to the ontology.

In this work, we have built a knowledge base system for querying OWL documents. We focus on OWL Lite and conjunctive queries on ABoxes that commit to a single TBox. We are interested in two typical types of ABox statements: 1) A concept assertion a:C states that the individual a belongs to the concept C. 2) A role assertion $<a,b>$:R states that individuals a and b have relationship with respect to the role R, which is a binary relation.

Like most of the contemporary OWL knowledge base systems, our system is a centralized store of OWL statements and provides reasoning and querying services. However, what distinguishes our system from others is that we have introduced a novel form of query answering. We call it document-centric query answering.

To explain the idea, suppose the system stores a set of OWL documents, $D=\{D_1, D_2, D_3, D_4\}$. In the usual way, D will be treated as a knowledge base that is the conjunction of all the documents in D. Then, we can pose queries on that knowledge base, for example, we can ask whether an assertion is entailed by the knowledge base.

In contrast, for document-centric query answering, our system does not handle its knowledge base as a monolithic one. Rather, the system may provide different answers depending on which portion of knowledge base, i.e., subset of documents, is used. Currently, the system is able to answer two new kinds of queries: 1) A ***document entailment* (*DE*) *query*** is concerned with a specific subset of D: Is that subset consistent and if so what statements are entailed by that subset? 2) A ***document provenance* (*DP*) *query*** asks for the smallest consistent document sets that entail specific statements.

Document entailment queries and document provenance queries can be seen as extensions to basic OWL queries by adding the element of documents. We can further divide these queries according to the genre of the basic query in question. Traditionally, there are two major types of ABox queries, i.e., boolean queries and retrieval queries. Accordingly, we can derive four sub-types of document-centric queries, as below.

1) A ***boolean document entailment query***, BDE(q_b, D_{sub}), where q_b is a boolean ABox query and $D_{sub} \subseteq D$, asks if D_{sub} is consistent and D_{sub} entails q_b.

E.g.: BDE(a:*Student*, $\{D_1,D_2\}$) Answer: *true* or *false*

2) A ***boolean document provenance query***, BDP(q_b, ?d), where q_b is a boolean ABox query and d a variable for document set, searches for all the minimal consistent subsets of D that entail q_b.

E.g.: Query: BDP(a:*Student,* ?d)
Answer: $\{d/\{D_1,D_2\}, d/\{D_3\}\}$

3) A **retrieval document entailment query**, RDE(q_r, D_{sub}), where q_r is a retrieval ABox query and $D_{sub} \subseteq D$, retrieves all the individuals that satisfy q_r in the context of D_{sub}, given D_{sub} is consistent.

E.g.: Query: RDE(<x>←x:*Student*, $\{D_1,D_2\}$)
Answer: {<x/*Student*1>, <x/*Student*2>}

4) A **retrieval document provenance query**, RDP(q_r, ?d), where q_r is a retrieval ABox query and d a variable for document set, retrieves all the individuals that satisfy q_r and in addition the minimal consistent subsets of D that support each answer.

E.g.: Query: RDP(<x>←<a,x>:*worksFor*, ?d)
Answer: {<<x/Company1>, d/$\{D_1\}$>,
<<x/University2>, d/$\{D_2, D_3\}$>}

Document-centric queries reflect an important nature of the Semantic Web, i.e., different sources may say different things about the same objects. By supporting these queries, the system allows its users to perceive the same set of data from different views, a view being represented by a subset of documents in the knowledge base.

[1] This work is part of the author's doctoral dissertation work. The author is currently employed by Microsoft Corporation.

To better motivate the queries, let us look at Fig. 1, which shows all the combinations of documents in D except the empty set. Essentially, a document entailment query is a query asked upon a specific combination. On the other hand, to answer a document provenance query, we potentially have to explore all the combinations (thus the size of the search space is $2^{|D|}$-1).

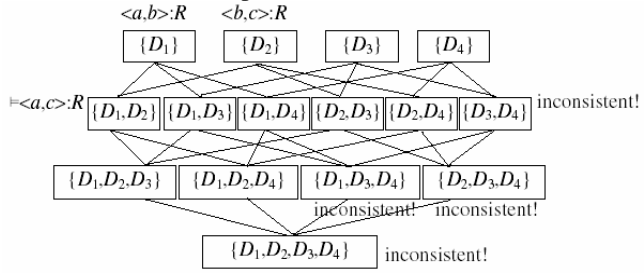

Fig. 1. Working space for document-centric queries.

Furthermore, we want to point out two important facts. First, multiple OWL documents may be combined to form a conclusion that is not entailed by any of the documents alone. As shown in Fig. 1, suppose R is a transitive role and D_1 and D_2 state $\langle a,b\rangle$:R and $\langle b,c\rangle$:R, respectively, then we can infer $\langle a,c\rangle$:R from $\{D_1,D_2\}$.

The other fact is that OWL is monotonic. This means if a statement is entailed by a set of documents, it is also entailed by any superset of that set; likewise, if a document set is inconsistent, each of its supersets will be inconsistent too. In Fig. 1, say $\{D_3,D_4\}$ are inconsistent, then $\{D_1,D_3,D_4\}$, $\{D_2,D_3,D_4\}$ and $\{D_1,D_2,D_3,D_4\}$ are inconsistent too. Note that it is important to identify and avoid use of inconsistent information since, as with classic logics, every statement can be deduced from an inconsistent OWL knowledge base.

Next we describe how we have built the system in order to support document-centric queries.

2. Preprocessing Documents

Answering OWL queries requires reasoning in order to guarantee completeness. Now that document-centric queries build upon basic ABox queries, our system relies on an external reasoner to fulfill those basic tasks for OWL knowledge bases, such as consistency checking, TBox classification (i.e., inference of the subsumption relationships between concepts and roles), ABox realization (i.e., inference of the most specific concepts each individual belongs to), and answering basic conjunctive ABox queries.

However, a great challenge here is scalability. We know that OWL reasoning is highly expensive (OWL Lite has worst-case ExpTime complexity). Furthermore, the new form of document-centric queries we proposed poses an even higher requirement for scalability, given its exponential search space as we showed in Fig. 1.

Given that, we recognize it is important to be able to reuse of the results of document processing, especially the expensive reasoning process. Therefore, instead of doing all the necessary reasoning at query time, we have decided to adopt a strategy that preprocesses the documents during load time and caches selected results. Roughly speaking, when the system loads documents into its knowledge base, it will try to process those subsets of documents that are consistent and performs reasoning on each combination. Based on the results, i.e., the statements inferred from each combination, the system identifies and records the minimal consistent subsets that support each of those statements. It will also record those minimal subsets of documents that are found inconsistent. The effect of this preprocessing is that the cached results will allow us to avoid reasoning on the documents every time when answering a query.

In addition, in order to reduce the number of document sets we have to process, we have introduced partitioning based approaches for identifying documents that are independent of each other with respect to the queries in question so that we can skip processing their combinations.

In the following subsections, we explain the key elements of the preprocessing approach. Before that, for brevity, we introduce an alias for "minimal consistent set", i.e., "context". A set of OWL documents is a ***context*** of a statement if it is a minimal consistent set that entails that statement. Note a statement may have multiple contexts.

2.1. Representing Contexts and Inconsistencies

The first question is how to keep track of the contexts identified for each statement during preprocessing. To streamline this, we have built our approach on the concept of the assumption-based truth maintenance system (ATMS). An ATMS is designed to readily tell the problem solver what is true given the currently made assumptions and work efficiently when the problem solver needs to explore multiple assumption sets simultaneously. The concept of ATMS fits our work if we regard a subset of documents as an assumption of the system, and the inferred statements from that document set as what is true under the assumption. Hence, we can make use of an ATMS to represent the contexts of statements.

A straightforward way of using the ATMS in our system is to use the ATMS as is, which will be to represent each statement with an ATMS node and use the ATMS network to record the justifications, i.e., how statements or inconsistencies are derived from others. However, there are several reasons that have prevented us from adopting this approach. First is the scalability consideration. Considering the scale of data available on the Semantic Web, it is impossible to represent each statement individually, not to mention all of their

inference relationships. Second, we assume that documents are all or nothing, i.e., we trust either the whole content of a document or none of it. Third, we use the OWL reasoner as a black box and we cannot easily determine the exact justifications at the statement level. We instead must determine them at the document level.

In light of the above, we have decided to use the ATMS in an unconventional way. We use an assumption node to represent a document and the node is seen as representing the set of statements entailed by the document. For the derived nodes, we use them to represent a set of statements as opposed to a statement: Each of the nodes represents the set of statements that are entailed by the document set in the environment of that node. Given this, a justification, $n_1,n_2,\ldots,n_m \Rightarrow n$, on the ATMS, is interpreted as: the conjunction of the statements represented by $n_1,n_2,\ldots,n_m$ implies the statements represented by n. Note again that it is possible that the union of multiple statement sets entails something that is not entailed by any of the statement sets individually. As a special case, a justification $n_1,n_2,\ldots,n_m \Rightarrow \bot$ conveys the information that the conjunction of the statements represented by $n_1,n_2,\ldots,n_m$ has led to a contradiction and thus the document set that is computed as the environment of the contradiction node is inconsistent. Fig. 2 is an example ATMS for four documents, D_1, D_2, D_3, and D_4, among which $\{D_1, D_2\}$ and $\{D_1, D_3\}$ are inconsistent.

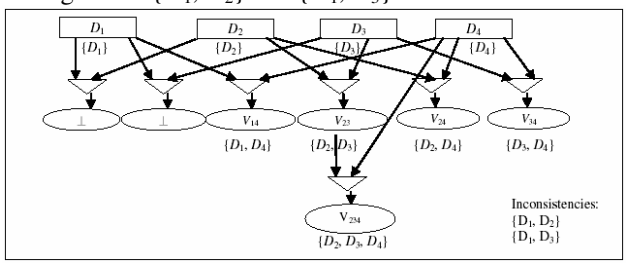

Fig. 2. An example ATMS with the unconventional use.

As another remark, given the above design, the ATMS we use bears some special features that are beneficial to the system's performance. First, every node except the contradiction node has one and only one environment. Moreover, since the environment of each of these nodes is just the document set the node represents, the justification relationships between these nodes become trivial. As such, in the implementation, we do not need to actually record the justifications between nodes. Note however that the full structure of the ATMS may be necessary for future extensions, for instance, when ABoxes commit to different TBoxes.

2.2. Storing Statements and Their Contexts

Now that we do not store statements directly on the ATMS, we need to store them elsewhere together with the links to their contexts represented on the ATMS. To make the system more scalable, we do not try to store the deductive closure of a knowledge base, i.e., all the entailments by the knowledge base. We observe that once the results of TBox classification and ABox realization are obtained, a simple semantic network suffices for answering a query about instances of a concept, given the concept is an atomic concept, or about instances of a role.

A semantic network is a graphical notation used for representing individual objects, categories of objects, and relations between objects. One important feature of the semantic network which we want to leverage is that its inference is much simpler than logic-based knowledge representation formalisms. Moreover, in the semantic network, we only store the subsumption relationships ($\sqsubseteq$) that are not redundant (e.g., $C_1 \sqsubseteq C_3$ is redundant given $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_3$) and the most specific concept instances and role instances (e.g., $a{:}C_1$ is more specific than $a{:}C_2$ given $C_1 \sqsubseteq C_2$). (Note this information can be obtained from the reasoner.) As we will show later, this allows us to replace the expensive OWL reasoning with a much simpler semantic network inference-like procedure during query answering.

To answer the queries, we also need to store context information in the semantic network. As a result, what is stored can be seen as a semantic network whose links are "annotated" by the contexts, as Fig. 3 depicts. Specifically, on the semantic network, we use nodes for representing concepts, roles and individuals, and arcs for *subclassOf* and *type* relationships. For each arc, its label shows the relationship it represents as well as the contexts for that relationship. Recall we use the ATMS to represent those document sets that are contexts of some statements. Thus a context annotation on an arc is just the ATMS node representing the context. In the figure, we use Γ_D to denote the ATMS node for D, D being a document or document set.

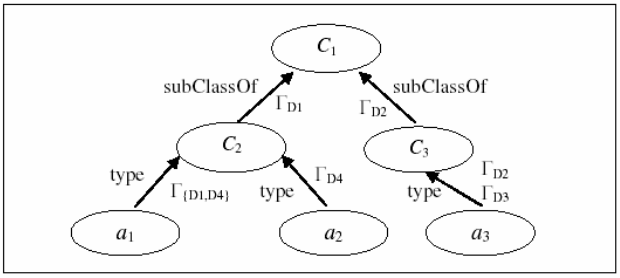

Fig. 3. An "annotated" semantic network.

It is noteworthy that what we are caching during preprocessing are selected reasoning results on the documents. We are not trying to precompute answers to particular queries. In addition, as shown above, we only store inferences that are minimal but sufficient for answering queries. For example, in the case of Fig. 3, we will store the statements $a_1{:}C_2$ and $C_2 \sqsubseteq C_1$, but we do not store the answer to a query such as BDE($a_1{:}C_1$, $\{D_1\}$).

Instead, the answer is computed at query time through the simple semantic network-like inference. In this way, we are trying to find a balance between doing some precomputation at loading time in order to save query time while controlling storage requirements.

2.3. Partitioning Documents

Our preprocessing approach aims at simplifying query answering by moving processing of documents from query time to document loading time. Now another key question facing us is how to reduce the space for preprocessing, in other words, to cut down the number of document combinations that need to be preprocessed. To this end, if we could in advance identify subsets of documents that might be logically dependent with respect to the reasoning in question, then we only need to focus on those subsets. This can be regarded as partitioning the documents, in the sense that those documents in the same partition are deemed dependent while those not are independent. Formally, we call this an independent document partitioning, as defined below.

Def. 1 (Independent Document Partitioning). *Given a set of common ontology OWL documents, S,* $\{S_i\}_{i\leq n}$ *is an independent partitioning of S iff 1)* $S = \cup_i S_i$*; 2) If* $S_{sub} \subseteq S$ *is a minimal set such that* $S_{sub} \models \varphi$ *for a concept assertion or role assertion* φ*, then there exists* S_i *such that* $S_i \supseteq S_{sub}$*.*

Fig. 4 illustrates how independent document partitioning may help reduce the amount of work for our system. Suppose $\{D_1,D_2,D_3,D_4\}$ could be partitioned to $\{\{D_1,D_2\},\{D_2,D_3,D_4\}\}$, then we only need to reason with $\{D_1,D_2\}$ and $\{D_2,D_3,D_4\}$ in the partition set and also their subsets (shown in grey color in the figure). This, trivially according to the above definition, will still allow us to correctly identify the contexts of any possible inference of concept assertion or role assertion.

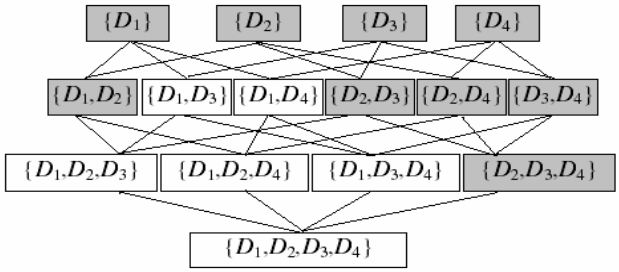

Fig. 4. How independent document partitioning may help reduce processing of documents.

The core of our approach for independent document partitioning is an algorithm for partitioning a set of OWL assertions into knowledge bases that are logically independent of each other with respect to the inference of concept assertions and role assertions. The main features of the approach include: a reasonable tradeoff between the complexity of determining partitions and the granularity of partitioning; worst-case polynomial time complexity of partitioning; and the ability to handle problems that are too large for main memory. Due to lack of space, we omit the detailed description and refer readers to [4, 5].

3. Answering Queries

Now we show how we answer document-centric queries based on the preprocessing results. We can consider that, after preprocessing, the knowledge base of the system consists of the semantic network (SN for abbreviation) and the ATMS. Due to lack of space, we will only give the algorithm for answering retrieval document provenance queries (RDP), the most complex type.

In order to facilitate the discussion, we start from a special case where the query contains a single query term about intances of a concept. Function EXECUTE-ATOMIC-RDP shows the algorithm. To answer the query on the knowledge base *kb*, we first search the semantic network for all the instances of *C* (see RETRIEVE-INSTANCES-AND-CONTEXTS) and add the instances together with their supporting contexts to the result. Then we try to combine the information about concept subsumptions and concept assertions. We look for all the subclasses of *C* in the semantic network and the associated contexts. For each found subsumption $C_{sub} \sqsubseteq C$, we repeat RETRIEVE-INSTANCES-AND-CONTEXTS on C_{sub} and, for each context in the result (i.e., the context for a concept membership), we combine it with each of the contexts for the subsumption. Note that, in the final result, we guarantee that, for every concept membership, only those minimal document sets that support the membership are returned (see INSERT-RDP-ANSWER).

---

function EXECUTE-ATOMIC-RDP(*kb*, RDP(<*x*>←*x*:*C*, ?*d*)) **return** a set of assignments for *x and d.* [2]
begin
  result := RETRIEVE-INSTANCES-AND-CONTEXTS(*kb*, *C*, ∅)
  kb.SN: search for all the subclasses of *C* and the associated context nodes
  for each found subsumption $C_{sub} \sqsubseteq C$ and its context node *n* **do**
    env := *kb*.ATMS: GET-ENVIRONMENT(*n*)
    s := RETRIEVE-INSTANCES-AND-CONTEXTS(*kb*, C_{sub}, *env*)
    for each element *e* in *s* **do**
      INSERT-RDP-ANSWER(*result*, *e*)
    end for
  end for
  return *result*

---

[2] For simplicity, hereafter we ignore "*x*/" and "*d*/" in the assignments for *x* and *d*.

end
function RETRIEVE-INSTANCES-AND-CONTEXTS (*kb*, *C*, *set*)
return: the set of every pair of <*a*, *s*> where *a*:*C* is in the semantic network and *s* the union of *set* and one of the contexts of *a*:*C*.
begin
  result := ∅
  kb.SN: search for all the concept assertions of *C* and their context nodes
  for each found concept assertion *a*:*C* and its context node *n* **do**
    env := *kb*.ATMS: GET-ENVIRONMENT(*n*)
    result := *result* ∪ {<<*a*>, *env* ∪ *set*>}
  end for
  return *result*
end
procedure INSERT-RDP-ANSWER(*set*, <*α*, *context*>)
{This procedure guarantees that *set*, which holds a set of answers to an RDP query, only contains minimal document sets for each set of retrieved individuals.}
begin
  if there exists no <*α*, *s*>∈*set* s.t. *s*⊆*context* **then**
    for each <*α*, *s*>∈*set* s.t. *s*⊇*context* **do**
      set := *set*\{<*α*, *s*>} //remove <*α*, *s*> from *set*
    end for
    set := *set* ∪ {<*α*, *context*>}
  end if
end

In case the query term is about instances of a role, e.g., <*a*,*b*>:*R* or *x*,*y*←<*x*,*y*>:*R*, the query can be done in exactly the same way, except that it is the role subsumptions and role assertions (as opposed to the concept subsumptions and concept assertions) which we will search the semantic network for.

Now let us look at the case where the query consists of multiple conjunctive terms. We can build upon the above algorithm by breaking the query into single terms. EXECUTE-CON-RDP shows the algorithm. Here one complication we need to deal with is that, since the query contains variables, how variables appear across query terms will affect the combination of the results on each query term. To this end, the algorithm uses the natural join to combine the variable bindings.

---

function EXECUTE-CON-RDP(*kb,* RDP(q_r, ?*d*))
input: *kb*: the knowledge base.
  q_r: a retrieval conjunctive ABox query <x_1,…,x_m> ← $q_1 \wedge \ldots \wedge q_n$.
return: a set of pairs of an assignment for x_1,…,x_m and an assignment for *d*.
begin
  if n = 1 **then return** EXECUTE-ATOMIC-RDP(*kb,* RDP(q_r, ?*d*))
  for every query term q_i **do**
    $result_i$:= EXECUTE-ATOMIC-RDP(*kb,* RDP(<x_{i1},…,x_{ik}> ← q_i, ?*d*))
    (where x_{i1},…,x_{ik} are variables appearing in q_i)
    if $result_i$ = ∅ **then return** ∅
    Create an initially empty relation $r_i(R_i)$, where schema R_i is a list of the variable names in q_i, i.e., x_{i1},…,x_{ik}, plus the column *context_i*
    for each answer *t*, i.e., a pair <<a_{i1},…,a_{ik}>, *context*>, in $result_i$ **do**
      insert *t* into r_i
    end for
  end for
  $r = r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$, the natural join of r_i (i≤n)
  result := ∅
  for each tuple (a_1,…,a_m,c_1,…,c_n) in *r*, where a_i is the binding for x_i and c_i the value for column *context_i*, **do**
    INSERT-RDP-ANSWER(*result*, <<a_1,…,a_m>, $c_1 \cup \ldots \cup c_n$>)
  end for
  return *result*
end

Here we give an example to help explain the algorithm. Consider the query RDP(<*x*,*y*> ← *Student*(*x*) ∧ *advisorOf*(*y*,*x*), ?*d*). Suppose the results on the individual query terms, i.e., RDP(<*x*> ← *Student*(*x*), ?*d*) and RDP(<*x*,*y*> ← *advisorOf*(*y*,*x*), ?*d*), are {<(*Student*$_1$), {D_1}>, <(*Student*$_2$), {D_1}>} and {<(*Faculty*$_1$, *Student*$_1$), {D_2}>, <(*Faculty*$_1$, *Student*$_3$), {D_2}>}, respectively. The two corresponding relations holding these results are shown below. Note that each relation has an extra column holding the contexts, in other words, the bindings for *d*. Therefore, when we join both relations based on the same variable name conditions, these context columns will be included in the joined relation as well. Then for each tuple in the resulting relation, we will combine the contexts by doing set operations (INSERT-RDP-ANSWER) in order to obtain the correct context for each variable assignment. For this example, the final result will then be {<(*Student*$_1$, *Faculty*$_1$), {D_1,D_2}>}.

Ex. 1. query: RDE(<*x*,*y*> ← *Student*(*x*) ∧ *advisorOf*(*y*,*x*), ?*d*)

result of RDP(<*x*> ← *Student*(*x*), ?*d*)

x	*context_1*
Student$_1$	{D_1}
Student$_2$	{D_1}

result of RDP(<*x*,*y*> ← *advisorOf*(*y*,*x*), ?*d*)

y	*x*	*context_2*
Faculty$_1$	*Student*$_1$	{D_2}
Faculty$_1$	*Student*$_3$	{D_2}

joined relation

x	*y*	*context_1*	*context_2*
Student$_1$	*Faculty*$_1$	{D_1}	{D_2}

final result: {<(*Student*$_1$, *Faculty*$_1$), {D_1,D_2}>}

4. Implementation and Empirical Evaluation

The implementation of the system is based upon a MySQL relational database. The main contents stored in the database are: the original OWL assertions; the semantic network that stores the reasoning results; and the document partitions. In addition, the ATMS is implemented as a serializable object. Moreover, we use Pellet [7] as the external reasoner.

We have conducted an evaluation using the real world FOAF (The Friend of a Friend) data and the synthetically generated Lehigh University Benchmark (LUBM) data. Both kinds of data have been frequently used in the Semantic Web community. The test environment is as follows: 2.80GHz Pentium 4 CPU; 1GB of RAM; 80GB of hard disk; Windows XP OS; Java SDK 1.5.

FOAF (www.foaf-project.org) is a Web community-driven project for creating Semantic Web data that describe people and basic social networks between them. In our test, we have used over 27,000 real world FOAF documents, thanks to Tim Finin from UMBC who has provided us with the URL index of Swoogle [3], the well-known Semantic Web search engine. In order to test the scalability of the system, we have also randomly selected three subsets of those documents. Thus we have four test sets in total. As the top part of Table 1 shows, these documents have a small average number of triples (i.e., statements). This is understandable if we consider a FOAF document as a Semantic Web homepage for a person.

The second part of Table 1 shows the data processing time for each test set, including the total time, time for loading the documents (into the database), time for partitioning the documents, and time for processing the documents and document combinations. Also Fig. 5 depicts how these times grow with respect to the number of documents as well as the number of triples in the test set. It turns out that the system scales fairly well in terms of both views.

Also we want to note that among the tasks performed on data, loading and reasoning with single documents are indispensable for every Semantic Web knowledge base system. Thus we could roughly regard the other tasks, i.e., partitioning documents and processing document combinations, as the overhead of our system in order to support document-centric queries. Then, take the largest test set as an example, these extra tasks accounts for only about 14% of the total time in the case of FOAF.

Table 1 also includes some interesting statistics about document partitions. For example, for all the test sets, the number of partitions (size>1) is far less than the total number of documents. In addition, the average number of documents contained in a partition is very small (≤5). This clearly demonstrates the utility of partitioning.

Table 1. Test results on FOAF data – preprocessing.

		Test Set 1	Test Set 2	Test Set 3	Test Set 4
Statistics of Test Sets	# of Documents	6,935	13,724	20,532	27,392
	# of Triples	162K	342K	505K	672K
	Average Size of Documents (in triples)	23	25	25	25
	Min Size of Documents (in triples)	1	1	1	1
	Max Size of Documents (in triples)	8,275	1,987	8,275	8,275
Document Processing Time	Total Time (Seconds)	925	1,897	3,292	4,531
	Load Time	359	759	1,355	1,856
	Partitioning Time	85	166	289	407
	Time for Processing Single Documents	480	957	1,536	2,062
	Time for Processing Combinations (size>1)	1	15	112	206
Statistics of Document Partitions	# of Partitions (size>1)	6	13	27	34
	Average Size of Partitions (size>1) (# of documents)	2	3	4	5
	Max Size of Partitions (size>1) (# of documents)	3	5	15	18
Statistics of Document Processing	# of Combinations (size>1) Processed	9	104	837	1,558
	# of Minimal Inconsistent Subsets of Documents (including size=1)	0	4	6	12
	Max Size of the Minimal Consistent Subsets of Documents that have New Entailments	2	3	3	3

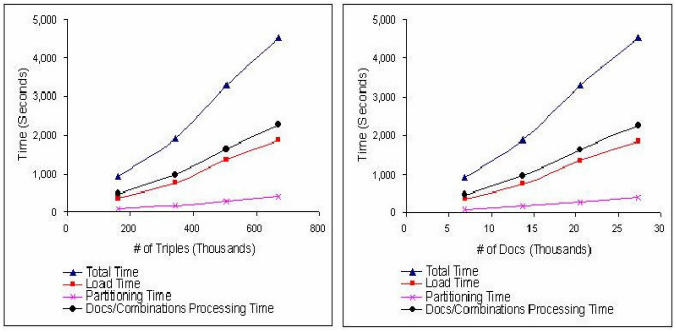

Fig. 5. Time for processing FOAF data.

We have also done an evaluation on query answering. Since our system adopts a strategy that moves the bulk of work from query time to document loading time, we are not surprised to see that the system responds to the test queries fairly quickly. Given this, we will only present the results with the largest set, i.e., Test Set 4, as shown in Table 2. Note that here, for brevity, we use anonymous names (e.g., *name1*) as opposed to the real names of people. Likewise for URIs and document IDs. This suffices to show readers the patterns of the queries.

One thing stands out is that it takes the system the longest time to answer the two retrieval document

provenance (RDP) queries. This is however not surprising, considering the nature of an RDP query: An RDP query does not offer a specific context for soliciting answers, and in addition, its basic query retrieves individuals rather than checking a specific instance of a concept or role.

Table 2. Query results on FOAF Test Set 4 (27K documents, 672K triples).

Test Query	Query Time (ms)	Answer/ # of Answers	Note
BDE(*uri1*:*Person*⋀<*uri1*,*name1*>:*name*, {D_1,D_2})	1	*false*	
BDE(*uri1*:*Person*⋀<*uri1*,*name1*>:*name*, {D_1,D_2,D_3})	<1	*true*	
BDP(*uri1*:*Person*⋀<*uri1*,*name1*>:*name*, ?*d*)	2	1	the answer is {D_1,D_2,D_3}
BDP(*uri1*:*Person*⋀<*uri1*,*name1*>:*name*⋀ <*uri1*,*nick1*>:*nick*, ?*d*)	3	0	
RDE(<*x*,*y*>←*x*:*Person*⋀<*x*,*y*>:*name*, {D_1,D_2})	14	1	
RDE(<*x*,*y*>←*x*:*Person*⋀<*x*,*y*>:*name*, {D_1,D_2,D_3})	2	3	{D_1,D_2,D_3} entails two more answers than {D_1,D_2}
RDP(<*x*,*y*>←*x*:*Person*⋀<*x*,*y*>:*name*, ?*d*)	13,337	1,151	
RDP(<*x*,*y*>←*x*:*Person*⋀<*x*,*y*>:*name*⋀ <*x*,*z*>:*nick*, ?*d*)	18,672	1,320	

As we know, document entailment queries specify a set of documents. In order to see how the system performs as the number of documents in the specified set increases, we have also evaluated the system on the above boolean document entailment queries (BDE) and retrieval document entailment queries (RDE) with a range of document set sizes. (Note that we do not include the largest set since it is inconsistent.) Table 3 shows the result. As we can see, the system still scales well.

Table 3. Query results on FOAF Test Set 4 - document entailment queries.

Test Query		Size of *S*			
		100	1000	10000	20000
BDE(*uri1*:*Person*⋀<*uri1*,*name1*>:*name*, *S*)	Query Time (ms)	2	2	17	42
	Answer	*false*	*false*	*false*	*false*
RDE(<*x*,*y*>←*x*:*Person*⋀<*x*,*y*>:*name*, *S*)	Query Time (ms)	31	145	1,129	2,226
	# of Answer	1	19	432	663

Compared to FOAF, the LUBM data have much larger average sizes of documents but relatively small numbers of documents, e.g., the largest test set contains over 2.7 million statements in about 4,800 documents. The results demonstrate that the system also scales quite well in terms of both preprocessing and query answering. Due to lack of space, we will not present the detailed results here and refer users to [4].

5. Conclusion and Future Work

We have proposed document-centric query answering, a novel form of query answering for the Semantic Web. The new queries extend basic Semantic Web queries in order to better reflect the open and distributed nature of the Semantic Web, and at the same time, pose higher requirements for scalability. In building the system for answering those queries, we have developed several key techniques, including preprocessing the documents and caching selected reasoning results in order to simplify querying and a partitioning based approach for reducing the working space for preprocessing. Experiments have demonstrated good scalability of the system. In the future, we plan to extend our approach to support OWL DL, a more expressive sublanguage of OWL and to support the case where ABoxes commit to multiple TBoxes. Also we intend to build proof-of-concept applications upon the query answering system, e.g., in the domain of trust.

References

[1] Berners-Lee, T., Hendler, J., & Lassila, O., "The Semantic Web", *Scientific American* 284(5), 2001, pp. 34–43.

[2] Dean, M. & Schreiber, G. (Eds.), *OWL Web Ontology Language Reference*, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/RECowl-ref-20040210/

[3] Ding, L., Finin, T., Joshi, A., Peng, Y., Pan, R., & Reddivari, P., "Search on the Semantic Web", *IEEE Computer* 10(38), 2005, pp. 62–69.

[4] Guo, Y., *Reasoning and Querying the Semantic Web – a Document-Centric Perspective*, PhD dissertation, Lehigh University, 2007.

[5] Guo, Y. & Heflin, J., "A Scalable Approach for Partitioning OWL Knowledge Bases", in *Proc. of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems* (*SSWS2006*).

[6] Horrocks, I. & Patel-Schneider, P.F., "Reducing OWL entailment to description logic satisfiability", *Journal of Web Semantics* 1(4), 2004, pp. 345–357.

[7] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., & Katz, Y., "Pellet: A practical OWL-DL reasoner", *Journal of Web Semantics* (to appear).