# A Scalable Indexing Mechanism for Ontology-Based Information Integration

Yingjie Li[1], Abir Qasem[2], Jeff Heflin[1]
[1]Department of Computer Science and Engineering, Lehigh University
19 Memorial Dr. West, Bethlehem, PA 18015, U.S.A.
{yil308, heflin}@cse.lehigh.edu
[2]Department of Computer Science, Bridgewater College
402 East College Street, Bridgewater, VA 22812, U.S.A.
aqasem@bridgewater.edu

## Abstract

*In recent years, there has been an explosion of publicly available RDF and OWL web pages. Some of these pages are static text files, while others are dynamically generated from large knowledge bases. Typically, these pages are small, heterogeneous and prone to change frequently. Although centralized knowledge bases and/or triple stores can be used to collect and query large volumes of heterogeneous Semantic Web data, such systems will rapidly become stale when the sources are highly dynamic. An approach based on information integration can address this problem, but poses new questions with respect to how to effectively index the data sources. We propose to adapt a query reformulation algorithm and combine it with an information retrieval inspired index in order to select all sources relevant to a query. We treat each RDF document as a bag of URIs and literals and build an inverted index. Our system first reformulates the user's query into a set of subgoals and then translates these into Boolean queries against the index in order to determine which sources are relevant. Finally, the selected data sources and the relevant ontology mappings are used in conjunction with a description logic reasoner to provide an efficient query answering solution for the Semantic Web. We have evaluated our system using ontology mappings and ten million real world data sources.*

## 1 Introduction

Semantic Web researchers envision a world-wide distributed architecture where data and computational resources will easily interoperate to coordinate complex tasks such as reasoning and answering queries. Due to the decentralized nature of the Semantic Web, it is inevitable that different communities of users or software developers will use their own ontologies to describe semantic data sources. Many of these sources are text documents represented in the standard formats for the Semantic Web: RDF and OWL. Most large semantic web knowledge bases (like DBPedia) frequently follow Linked Open Data [1] guidelines and expose their contents via dynamically generated web pages about each resource contained within. Unlike traditional applications in distributed databases, these semantic web data sources are often small (containing fewer than 50 triples), heterogeneous (committing to many different schemas or ontologies) and are prone to change frequently. Sindice, a popular semantic web index, reports that it has indexed over four billion pieces of information from over 60 million documents. Although recent research has led to the development of knowledge bases (KBs) and/or triple stores that can answer queries over collections of this size, centralized knowledge bases will become stale unless they are frequently reloaded with changed data. Furthermore, since the majority of these systems materialize inferences, it is difficult for them to accommodate deleted information. The field of information integration has developed algorithms for querying distributed, heterogeneous databases and thus is appropriate when data is dynamic. However, this work often assumes a small number of total data sources, and that human-generated summaries of the content of each are available.

In this paper we build upon our earlier work in ontology-based information integration [10]. In that work, we defined relevance files - additional meta-descriptions of data sources that indicate not only what specific terms are used from which ontologies, but also placed restrictions on the resources used in the triples. We assumed that a set of ontology alignments, in the form of mapping ontologies, were available to resolve heterogeneity between the sources. We then defined an algorithm that given the rel-

[1]http://linkeddata.org/

evance files, mapping ontologies, and a query, could identify which sources are potentially relevant in the sense that their descriptions could apply to source that has relevant data (since the relevance files are abstractions of the original data sources, we can only know if the source is truly relevant by directly accessing it). Note, in this process, it is essential that we do not miss relevant sources that commit to ontologies different from the ones used in the query. For this reason, we adapted the PDMS information integration algorithm [4] to reformulate the query into alternatives before comparing it to the relevance files. Note, since we assume that our data sources are files instead of databases, instead of sending rewritten queries to them, we load all of the selected files into a knowledge base system and apply the original query to this KB. This has the advantage that our algorithm can be complete and terminate, even when there are cycles in the mapping axioms. The most significant drawbacks to this original approach are: the reliance on human-generated relevance statements for each source, and the inability of the relevance language to satisfactorily summarize sources like people's home pages.

We make two technical contributions in this paper. First, we combine our previous reformulation algorithm with an index inspired by the field of information retrieval. Specifically, we abstract each RDF document as a bag of URIs and literals and build an inverted index over these entities. This index is then used in place of relevance files when selecting potentially relevant sources. Second, we conduct a number of experiments to evaluate the characteristics of our system, and demonstrate that it improves upon its predecessor not only by allowing fully automatic index generation, but also by being more selective while retaining completeness.

The rest of the paper is organized as follows: Section 2 formally defines the source selection problem. In section 3, we first describe the overall architecture of our system and then describe specific details about our index based source selection algorithm. Section 4 presents the experiments that we have conducted to evaluate the algorithm. Finally, section 5 concludes and discusses future work.

## 2 Problem Definition

In this section we formally define the source selection problem in the context of the Semantic Web.

The Semantic Web can be viewed a set of ontologies. According to the official OWL description [8], an ontology can have classes, properties and individuals that are described by these classes and properties. However, for convenience of analysis we decided to separate ontologies (i.e. the class/property definitions and axioms that relate them) and data sources (assertions of class membership or property values). In the discussion that follows, we use $C$ to refer to the set of all classes, $P$ to refer to the set of all properties, $D$ to refer to the set of all individuals, and $U$ to refer to the set of document identifiers (URLs in the case of OWL) in the Semantic Web.

We can say an ontology is some subset of $C \sqcup P$ (and possibly a set of axioms that relate them). A data source is a set of assertions of the form $a : c \in C$ or $< a, b >: p \in P$ where $a$ and $b$ are names that denote individuals. Then, we introduce two functions. First, $o$ is an ontology function that maps $\mathcal{U}$ to a set of ontologies. Second, $s$ is a source function maps $\mathcal{U}$ to a set of data sources.

**Definition 1 (Semantic Web Space)** *A Semantic Web Space SWS is a tuple $\langle \mathcal{U}, o, s \rangle$.*

We define the satisfaction of $o(u)$, $s(u)$ per the official OWL semantics document [8].

**Definition 2 (Satisfaction)** *An interpretation $\mathcal{I}$ satisfies a Semantic Web Space $\langle \mathcal{U}, o, s \rangle$, iff for each u $\in \mathcal{U}$, $\mathcal{I}$ satisfies o(u) and s(u).*

A knowledge base entails (written $\models$) a set of logical sentences $\alpha$ iff every interpretation that satisfies the knowledge base also satisfies $\alpha$. The notion of entailment of a SWS is defined as following.

**Definition 3 (Semantic Web Space Entailment)** *Given a set of description logic sentences $\alpha$, SWS $\models \alpha$ iff every interpretation that satisfies SWS also satisfies $\alpha$.*

Our query language is based on the conjunctive query language for DLs that has been proposed by Horrocks $et\ al.$[6]. This query language overcomes the inadequacy of description logic languages in forming extensional queries.

**Definition 4 (Conjunctive Query Form)** *A conjunctive query has the form $Q\left(\overline{X}\right)$:- $B_1\left(\overline{X}_1\right), \ldots, B_n\left(\overline{X}_n\right)$ where $\overline{X}$ is a vector of variables and/or individuals and each $B_i$ is a unary or a binary atom representing a concept or role term respectively.*

Furthermore, this kind of query corresponds to the most common SPARQL queries. We refer to the left hand side of :- as the head of the query and the right hand side as the body of the query. The variables that appear in the head must appear also in the body and are universally quantified. Such variables are called distinguished variables and describe the form of a query's answers. All other variables in the query are called non-distinguished variables and are existentially quantified. For a given query Q and substitution $\theta$, we use Q$\theta$ as a shorthand for $\mathrm{B}_1\theta \wedge \mathrm{B}_2\theta \ldots \wedge \mathrm{B}_n\theta$.

**Definition 5 (Answer Set)** *Given a Semantic Web Space SWS, an answer set $\mathcal{A}$ to a query Q is the set of all substitutions $\theta$ for all distinguished variables in Q such that: SWS $\models Q\theta$.*

All variables in a query should be mapped to individuals explicitly introduced in the data sources. Finally, we will give the source relevance definition. In this definition, the term index is a function $I : T \rightarrow \mathcal{P}(U)$, where $T = C \sqcup P \sqcup R \sqcup L$, $R$ is the set of all constant URIs, $L$ is the set of the terms in all literals, and $\mathcal{P}(U)$ is the power set of $U$.

**Definition 6 (Source Relevance)** *Given a conjunctive query $Q$, a term index $I$ (this index is which terms appear in which data sources), a set of document identifiers $U$, and an ontology function $o$, a source $u$ is potentially relevant to $Q$ iff $\exists$ a source function $s$ that conforms to $I$ where $< U, o, s > \models Q\theta$ and $< U, o, s - u > \not\models Q\theta$.*

*Note, $s$ conforms to $I$ means that $\forall t \in T$, $u \in I(t)$ iff $t \in terms(s(u))$, where the function terms is defined as $terms : ABox \rightarrow T$, and $s - u$ denotes a function f(x) as follows:*

$$f(x) = \begin{cases} s(x) & if\ x \neq u \\ \emptyset & otherwise \end{cases}$$

## 3 Index-based Source Selection

In this section we first introduce our system architecture, then propose our index-based source selection algorithm.

### 3.1 System Architecture

As depicted in Figure 1, our system consists of three main components: GUI, OBII-IR, and distributed data sources. We assume that each data source commits to one or more OWL domain ontologies from the set $\{O_1, ..., O_n\}$. We use OWL axioms to describe a map between a pair of related ontologies. The choice of OWL to articulate the alignments make these maps shareable via the Web.

Periodically, the Indexer is run to create an inverted index for all of the data sources. The details of this process are described in Section 3.2. A user query is input via a graphical user interface (GUI) and converted into SPARQL. The Reformulator takes the SPARQL query as an input and uses the domain and mapping ontologies to determine all possible subgoals for the query. Specifically, a subquery expressed in terms of a particular domain ontology is translated into queries in terms of other domain ontologies. The Selector takes the reformulated subgoals as inputs, issues a Boolean query on the index, and returns a set of source identifiers (e.g., URLs). The Loader reads the selected sources and inputs them into a description logic (DL) reasoner, which is then queried to produce results that are returned to the GUI. Since the selected sources are loaded in their entirety into a reasoner, any inferences due to a combination of these selected sources will also be computed by the reasoner. This process of source selection effectively prunes irrelevant data available on the Semantic Web from consideration by the reasoner.

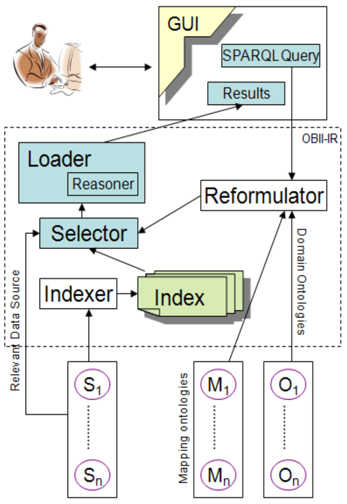


**Figure 1. System Architecture with arrows showing the flow of information when processing a query**

Given a conjunctive query, our Reformulator adapts the OBII-GNS reformulation algorithm [11]. OBII-GNS is complete for data sets with domain and map ontologies expressed in OWLII, a sublanguage of OWL defined below:

**Definition 7** *The syntax of OWLII consists of DL axioms of the forms $C \sqsubseteq D$, $A \equiv B$, $P \sqsubseteq D$, $P \equiv Q$, $P \equiv Q^-$, where $C$ is an $L_a$ class, $D$ is an $L_c$ class, $A$, $B$ are $L_{ac}$ classes and $P$, $Q$ are properties. $L_{ac}$, $L_a$ and $L_c$ are defined as:*

- *$L_{ac}$ is a DL language where $A$ is an atomic class and $i$ is an individual. If $C$ and $D$ are classes and $R$ is a property, then $C \sqcap D$, $\exists R.C$ and $\exists R.\{i\}$ are also classes.*
- *$L_a$ includes all classes in $L_{ac}$. Also, if $C$ and $D$ are classes then $C \sqcup D$ is also a class.*
- *$L_c$ includes all classes in $L_{ac}$. Also, if $C$ and $D$ are classes then $\forall R.C$ is also a class.*

Like the PDMS adaptation for OBII, OBII-GNS applies global-as-view (GAV) and local-as-view (LAV) expansions to subgoals. The algorithm maintains an open list of nodes to be expanded and a closed list of nodes that have been already expanded instead of creating children nodes in a tree as OBII-PDMS did. At the beginning, the open list is initialized with goal nodes created from the subgoals of the query. During expansion, new expanded nodes are added to the open list and already expanded nodes are added to the closed list. This process is repeated until the open list become empty. OBII-GNS has been shown to be faster than the OBII-PDMS, mostly due to its ability to save on book-

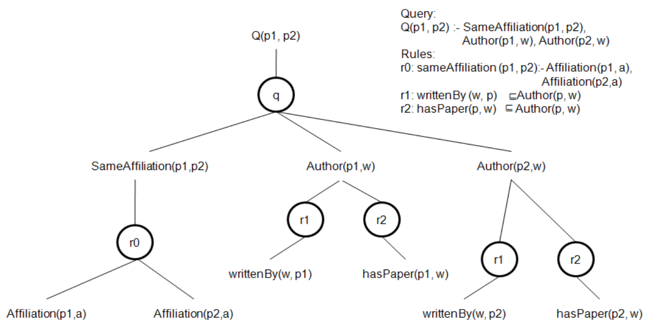
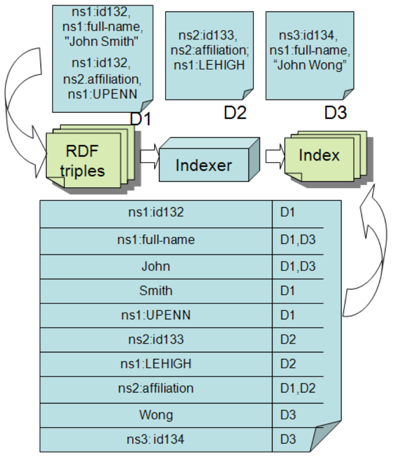

Query:
Q(p1, p2) :- SameAffiliation(p1, p2), Author(p1, w), Author(p2, w)
Rules:
r0: sameAffiliation (p1, p2):- Affiliation(p1, a), Affiliation(p2,a)
r1: writtenBy (w, p) ⊑Author(p, w)
r2: hasPaper(p, w) ⊑Author(p, w)
Q(p1, p2)
q
SameAffiliation(p1,p2)
Author(p1,w)
Author(p2,w)
r0
r1
r2
r1
r2
Affiliation(p1,a)
Affiliation(p2,a)
writtenBy(w, p1)
hasPaper(p1, w)
writtenBy(w, p2)
hasPaper(p2, w)

**Figure 2. Reformulation tree for an sample query**


ns1:id132, ns1:full-name, "John Smith"
ns1:id132, ns2:affiliation, ns1:UPENN
D1
ns2:id133, ns2:affiliation; ns1:LEHIGH
D2
ns3:id134, ns1:full-name, "John Wong"
D3
RDF triples
Indexer
Index

| | |
|---|---|
| ns1:id132 | D1 |
| ns1:full-name | D1,D3 |
| John | D1,D3 |
| Smith | D1 |
| ns1:UPENN | D1 |
| ns2:id133 | D2 |
| ns1:LEHIGH | D2 |
| ns2:affiliation | D1,D2 |
| Wong | D3 |
| ns3:id134 | D3 |

**Figure 3. RDF inverted index**

keeping and avoid redundant expansions. The time complexity of OBII-GNS is polynomial.

To illustrate the reformulation, Figure 2 shows an example for a simple query. We begin with the query, $Q$, which asks for researchers who have worked at the same affiliation and also coauthored a paper. $Q$ is expanded into its three subgoals, each of which appears as a goal node. The SameAffiliation is involved in a GAV rule ($r0$), hence we expand the SameAffiliation goal node with the rule $r0$ and its children are two goal nodes of the Affiliation relation (each specifying the affiliations an individual researcher works at).

The Author relation is involved in two LAV rules ($r1$ and $r2$), hence we expand Author($p1$, w) and Author($p2$, w) with these two rules and their children become two nodes of the writtenBy and the hasPaper respecitvely. Finally, after this process, the orginial query $Q$ will be reformulated as the set of relations consisting of Affiliation, writtenBy and hasPaper.

## 3.2 Index-based Source Selection Algorithm

Our Index-based Source Selection Algorithm is inspired by the IR techniques. It is well-known that IR approaches treat each document as a bag of words. IR systems typically use an inverted index, where each term is an entry to an index that contains a posting list of documents that contain the term.

Unlike typical document collections, the units in semantic web documents are triples not words. Each triple consists of three parts: subject, predicate and object. Both the subject and predicate are always URIs, but the object could be a URI or a Literal / datatype value. If we let $U$ be the set of URIs and $L$ be the set of Literals, then an RDF document $d$ has the form $d \sqsubseteq U \times U \times (U \sqcup L)$.

We treat an RDF document as a bag of URIs and Literals and create an inverted index. In this process, the most critical point is to determine the entry terms for an RDF document. In order to identify such terms, we must first tokenize each RDF document. In our system, we will use the full URIs of subjects, predicates and objects as our tokens. When the object is a literal, we will process it as a free-text format and index its literal value. In this way, our system can support queries that involve partial string matches. Thus, the terms of the document can be formally expressed as following:

$terms(d) \equiv \{x| < s, p, o > \in d \wedge [x \equiv s \vee x \equiv p \vee (o \in U \wedge x \equiv o) \vee (o \in L \wedge x \in lit-terms(o))]\}$

where lit-terms() is a function that extracts terms from literals, and may involve typical IR techniques such as stemming and stopwords. The dictionary of our system is then $\bigcup_{d \in D} terms(d)$. We have implemented an algorithm BUILD-INDEX that takes an RDF document $d$ and its identifier (e.g., a URL) as input, constructs a virtual document consisting of $terms(d)$, and then indexes this document using Lucene, an open source information retrieval library. During this process all local names and qualified names are resolved to full URIs. An example inverted index for three RDF documents is given in Figure 3.

Our new source selection algorithm is described in Algorithm 1. This algorithm is mainly to construct index-compatible Boolean queries and then identify potentially relevant data sources using these queries. Inside, the core function is the Boolean query construction. In this process, we need to further explain the process of class membership in a query. After query reformulation, the class membership query would be transformed into a goal node of $ns : Class(x)$ (suppose '$ns$' stands for the namespace and 'Class' stands for the name of one domain class defined in '$ns$'). Here, we need to process one special term $rdf : type$, which does not explicitly appear in our reformulation results. Therefore, when we are constructing our Boolean queries, we need to expand the class membership query into the triple pattern with the form "$ns : Class$ AND $rdf : type$". For example, consider a query that has been reformulated into the goals $\{u : Professor(x), u : teaches(x, cs : proglang), j : works-at(x, y)\}$. To use our Boolean inverted index, we need to issue a Boolean query ($u : Professor$ AND $rdf : type$) OR ($u : teaches$ AND $cs : proglang$) OR ($j : works-at$).

**Algorithm 1** Source selection by index

```
SELECTED-BY-INDEX(RQN: Reformulated Query Nodes)

 1: $sources \leftarrow \emptyset$
 2: for each $n \in RQN$ do
 3:   if $n$ $typeOf$ $Unary\_Node$ then
 4:     $qterm \leftarrow$ "($rdf : type$ AND"+$n.predicate$+")"
 5:   else
 6:     $qterm \leftarrow$ "(" + $n.predicate$
 7:     if $n.subject$ $typeOf$ $Constant$ then
 8:       $qterm \leftarrow qterm$+" AND "+$n.subject$
 9:     if $n$ $typeOf$ $owl : ObjectProperty$ then
10:       if $n.object$ $typeOf$ $Constant$ then
11:         $qterm \leftarrow qterm$+" AND "+$n.object$
12:     else
13:       if $n$ $typeOf$ $owl : DatatypeProperty$ then
14:         $lterms \leftarrow lit-terms(n)$
15:         for each $lterm \in lterms$ do
16:           $qterm \leftarrow qterm$ + " AND $lterm$"
17:   $boolean\_query.add(qterm$, ") OR")
18: $sources \leftarrow askIndex(INDEX, boolean\_query)$
19: return $sources$
```

**Theorem 1** *The index-based source selection algorithm is complete in that given a set of simple domain ontologies, OWLII map ontologies, and a term index over the data source documents, it will identify exactly the potentially relevant sources for the given conjunctive query.*

**Proof(sketch)**: Assume in the semantic web space, $C$ refers to the set of all classes, $P$ refers to the set of all properties, $R$ refers to the set of all constant URIs, $L$ refers to the set of all literal terms, $V$ refers to the set of all variables, and $T$ refers to the set of terms indexed by the term index $I$.

As shown in [9], the OBII-GNS algorithm is complete. Therefore, the subgoals identified by OBII-GNS are complete. Given a subgoal in form of $p(x, y)$ or $c(x)$ where $p \in P$, $c \in C$, $x/y \in R \sqcup V \sqcup L$ and a source function $s$, suppose a source $s(u)$ has a triple that unifies with $p(x, y)$ or $c(x)$. If $t_u = terms(s(u))$ [2], then $p/c \in t_u$, and $I(p/c) = u$. For $x/y$, if $x/y \in R$, then $x/y \in t_u$, and $I(x/y) = u$. Similarly, if $x/y \in L$, then $lit-terms(x/y) \in t_u$, and $I(x/y) = u$. Thus, a query reformulated by Algorithm 1 against $I$ must return $s(u)$. Therefore, our index-based source selection algorithm is complete.

[2]where $terms(s(u))$ is as defined in Definition 6.

## 4 Evaluation

To evaluate our system, we have conducted two experiments: a comparison with OBII-GNS and a comparison with Sindice. The first is to evaluate our system's source selectivity. This evaluation is done on a desktop with P4 2.6G CPU and 3G memory running Windows XP professional. The second is to evaluate our system's scalability. This evaluation is done on a workstation with Xeon 2.93G CPU and 6G memory running UNIX. In the indexer implementation, we currently use Lucene as our index builder. In all cases, we use KAON2 as our Reasoner.

### 4.1 Source Selectivity Evaluation

Our first experiment attempts to demonstrate that the IR-inspired index is superior to the relevance file indices are our prior work. We will use IR and REL, respectively to distinguish these two approaches. In this experiment we use the GNS algorithm for reformulation, but change the Indexer and Selector. Since the REL approach requires the generation of relevance files, we chose to use a synthetic data set for this evaluation, similar to those used in earlier evaluations of OBII-PDMS and OBII-GNS.

In order to make the synthetic data set much closer to real world data, we modified the original data generator. First, we ensure that each generated file is a connected graph, which more accurately reflects most real-world RDF files. Second, we randomly sampled 200 semantic web documents and determined that the average number of triples in each result document is around 54.015 with the standard deviation 163.9148; in the interest of round numbers, we set the average number of triples in a generated document to be 50. We conducted experiments with 50 ontologies, 1000 data source sources, and a diameter of 6, meaning that longest sequence of mapping ontologies between any two domain ontologies was six.

In this experiment, our IR index size is 10.8MB. The time to construct the IR index is 5,094ms, while it takes 14,593ms to construct the REL index. We issue 200 random queries and compute averages for all metrics mentioned here. Figure 4 displays the average number of results and average number of selected sources for each query. Observe that IR is more selective than REL in source selection but the query answers are still guaranteed to be the same. In this result, we select 20-25% fewer sources than in the REL method without losing any completeness.

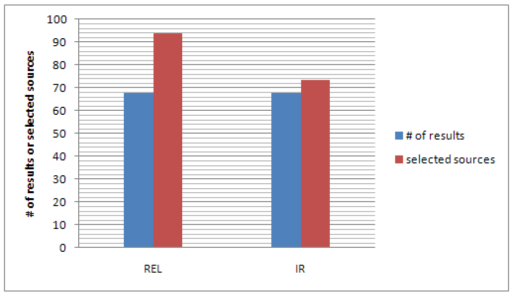


**Figure 4. Source selection of IR and REL**

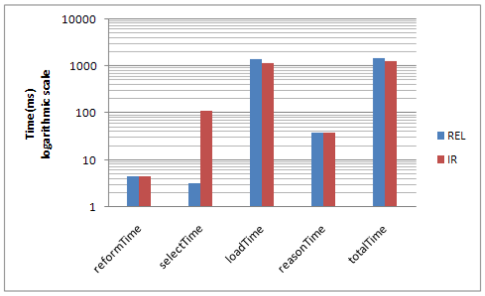


**Figure 5. Response time of IR and REL**

Figure 5 compares the response time of both systems, and breaks out the time to reformulate the query (reformTime), time to select sources (selectTime), time to load sources from local disk files (loadTime) and time spent by the KAON2 reasoner (reasonTime). Here, y axis is in logarithmic scale. The key observation here is that the totalTime of IR is $\sim$ 10% smaller than that of REL. The reason is that in both systems, loading sources is the dominant system cost, so fewer sources selected result in big gains. It should be mentioned the IR system has a worse select time than REL. This is mainly because the REL system uses a memory-based index, while IR uses a disk-based index to achieve greater scalability.

## 4.2 Scalability Evaluation

In this section, we will evaluate our system's scalability by using set of real world data sources. We choose a subset of the Billion Triple Challenge (BTC) 2009 data set, focusing on four collections as summarized in Table 1. The total number of triples in this dataset is 73,889,151. We created local N3 versions of the original files from the BTC by using the provenance information, result in 21,008,285 documents. The documents confirm our earlier claims about small size, varying from roughly 5 to 50 triples each.

Based on this dataset, we have designed 4 queries to evaluate our system (Table 2). The # of reformuated query

| Data sources | Ontology namesapce | # of Triples |
|---|---|---|
| http://data.semanticweb.org/ | swrc | 174,816 |
| http://sws.geonames.org/ | geonames | 14,866,924 |
| http://dbpedia.org | dbpedia | 48,694,372 |
| http://dblp.rkbexplorer.com | akt | 10,153,039 |
| Total | | 73,889,151 |

**Table 1. Data sources**

| Query | Query string | # of Reformulated query terms |
|---|---|---|
| 1 | ?person dbpedia:name "James A. Hendler" | 6 |
| 2 | ?paper swrc:author swrc:abir-qasem ?paper swrc:author swrc:jeff-heflin | 4 |
| 3 | ?person swrc:affiliation swrc:lehigh-university | 5 |
| 4 | ?person akt:full-name "Jeff Heflin" ?person swrc:affiliation ?org | 11 |

**Table 2. Test queries**

terms for each query are determined by the mapping ontologies and local axioms defined for the selected data sources. Figure 6 shows the performance of our system for answering these four queries. Table 3 shows the source selectivity of our system by triple level and document level.

Through the above experimental results, the first observation is that our index based mechanism is quite selective for our designed queries in both terms of number of triples selected and number of documents selected, as shown by Table 3. Our metrics mainly focus on the triple selectiviy and the document selectivity. The triple/document selectiviy is the ratio of the number of selected triples/documents over the total number of the triples/documents. In this result, both our triple and document selectivity are less than 0.1% of all triples and documents collected. The second observation is that our system can scale well to real world data with reasonable reformTime, selectTime, loadTime and reasoning time, as shown in Figure 6 (in logarithmic scale). The third observation is that our system performs better for queries Q1, Q2 and Q3 with selective terms such as "James A. Hendler" and "Jeff Heflin". This is because these terms make our system select less sources. For those queries without selective terms such as Q4 having a triple with two variables, our current system would perform worse even when no answers are returned. We will address this problem in furture work.

In addition, we compared the selector component of our system with Sindice. The motivation is to see how effective a local "bag of URIs" index is vs. querying a remote semantic web search engine to retrieve the relevant triples. Here, Sindice is used as an alternative for the selector, but the reformulator and loader are kept the same. After reformulation, the query would be transformed into Sindice

| Query | **# of Results** | **# of Selected triples** | **# of Selected documents** |
|---|---|---|---|
| 1 | 142 | 715 | 143 |
| 2 | 2 | 46 | 9 |
| 3 | 15 | 163 | 20 |
| 4 | 16 | 25342 | 5069 |

**Table 3. Source selectivity**

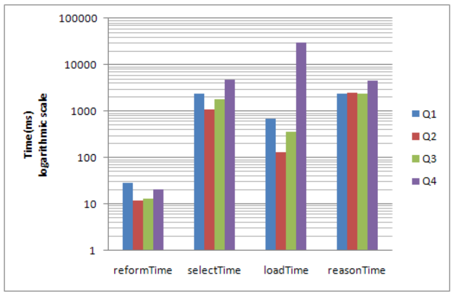


**Figure 6. Performance of IR**

compatible triple query patterns by using Sindice API to select relevant data sources. We mainly did comparison on their source selection time, as shown in Figure 7.

From Figure 7, we can conclude that our system performs faster than Sindice. One possible reason is that our index based mechanism is by local disk access different from Sindice's web access. In the future, we will expand our sytem to the web access to do further comparison with Sindice.

In this experiment, our index construction time is $\sim 58$ hours and its size is $\sim 18$GB. Each document takes $\sim 10ms$ on average. The Lucence configurations are 1500MB for RAMBufferSize and 1000 for MergeFactor, which are the best tradeoff between index building and searching for our scalability.

Since our algorithm does not yet select all relevant sources with sameAs information, we assume an environment where any relevant sameAs information is already supplied to the reasoner. We do this by initializing the KB with the necessary sameAs statements.

## 5 Related Work

Currently, there are mainly two areas of work related with our paper: Information Integration and RDF indexes.

In Information Integration, Haase and Motik developed a mapping system for OWL that involves relating conjuctive queries [3]. However, since this effectively adds rules to OWL, it is undecidable and as they need to introduce restrictions to achieve decidability. They do not explicitly address the issue of distributed data, and provide no means of

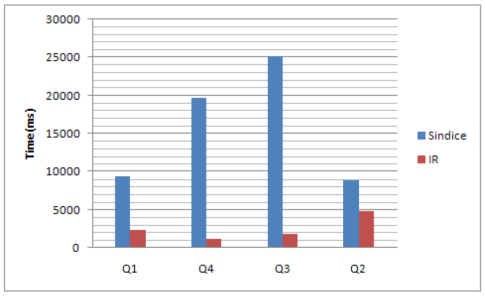


**Figure 7. Souce selection time of IR and Sindice**

indexing the relevant sources. Peer-to-peer (P2P) semantic web systems like Bibster [2] and SomeWhere [1] address the distributed nature of the Web, but each is insufficient to address the problem described in this paper. Peers in Bibster might have different data, but use the same ontologies. Peers in SomeWhere can have different ontologies, but the data consists only of category information (from an RDF point of view, this means all triples use the rdf:type predicate). While this is useful for sharing bookmarks, it is not very useful for question answering. Liarou et al. use Distributed Hash Tables (DHT) to provide answers to continuous, conjunctive queries issued to a network of nodes with RDF data [7]. However, like Bibster, they do not address the schema mapping issue and therefore only work in a single ontology environment. All of these P2P systems have a drawback in that each node must install system specific P2P software, presenting a barrier to adoption. Stuckenschmidt $et\ al.$ presented an architecture for querying distributed RDF repositories by extending the Sesame system, and propose an index structure as well as algorithms for query processing and optimization in a distributed context [12]. They use a hierarchy of path indexes that indicate which sources have information on which query paths. A major drawback of this work is that it does not consider schema heterogeneity. Hermes is probably the work that most closely addresses our problems [13]. Heremes translates a keyword query provided by the user into a federated query and then decomposes this into separate SPARQL queries that are issued to web data sources. A number of indexes are used, including a keyword index, mapping index, and structure index. The most significant drawback to the approach is that it does not account for rich schema heterogeneity (mappings are basically of the subclass/equivalent class variety).

In RDF indexes, Harth and Decker propose that six indexes can cover the possible access patterns for triples (depending on which values are known in a triple pattern) [5]. This allows for quick access to queries about a single statement, but is not as effective when used with conjunctive queries. Hexastore attempts to achieve scalability by replicating each triple six times: one for each sorting order of subject, predicate and object [15]. It has been demonstrated that this strategy results in good response time for conjunctive queries. The major disadvantages of both of these approaches are that they rely on centralized knowledge bases and that the indexes (or replication) are quite expensive in terms of space. The goal of this paper is to leave the original data at its source and to have compact local representations that help us locate this data. GRIN is a novel index developed specifically for graph-matching queries in RDF [14]. This index identifies selected central vertices and identifies the distance of other nodes from these vertices. This index is more compact than the two previously mentioned indexing approaches, but it still is not clear how it could be adapted for a distributed context.

## 6 Conclusions and Future Work

In this paper, we have proposed a scalable IR-inspired indexing mechanism for ontology-based information integration. According to this method, we treat each RDF as a bag of URIs and literals and build an inverted index over them. By using this index together with the query reformulation results, data sources that are potentially relevant with user's query can be selected. The selected data sources and the relevant ontology mappings are used in conjunction with a DL reasoner to provide a Semantic Web query answering solution that is better equipped to deal with dynamic data than a centralized knowledge base. We have demonstrated that our new system is better than our prior work in that not only does it allow automated index creation, but it also prunes more irrelevant sources without losing completeness, and has a 10% improvement in response time. We have also shown that the system scales well, allowing queries against 20 million heterogeneous data sources to complete in seconds.

However, there is still significant room for improvement. First, our current algorithm's performance will decline significantly when there are triple patterns with variables in both the subject and object position, because this amounts to retrieving all sources that mention a specific predicate. We intend to develop an optimizer that will find the most selective triple patterns and use the sources found by these to further constrain other triple patterns. Second, real semantic web sources often use the different identifiers for the same entity. Our algorithm needs to be adapted to locate relevant owl:sameAs statements; this must necessarily be an iterative process in order to find the transitive closure of relevant owl:sameAs statements. We believe that solving such problems will lead to a pragmatic solution for querying a large, distributed, and ever changing Semantic Web.

## References

[1] P. Adjiman, P. Chatalic, F. Goasdou, M. c. Rousset, and L. Simon. Somewhere in the semantic web. In *Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR, Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[2] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Olko, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *International Semantic Web Conference*, pages 122–136, 2004.

[3] P. Haase and B. Motik. A mapping system for the integration of owl-dl ontologies. In *IHIS '05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 9–16, New York, NY, USA, 2005. ACM.

[4] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. *Data Engineering, International Conference on*, page 505, 2003.

[5] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *LA-WEB '05: Proceedings of the Third Latin American Web Congress*, page 71, Washington, DC, USA, 2005. IEEE Computer Society.

[6] I. Horrocks and S. Tessaris. A conjunctive query language for description logic Aboxes. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[7] E. Liarou, S. Idreos, and M. Koubarakis. Continuous rdf query processing over dhts. pages 324–339. 2008.

[8] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language semantics and abstract syntax. Recommendation, February 2004. http://www.w3.org/TR/owl-semantics/.

[9] A. Qasem. *Query-based Selection and Integration of Semantic Web Data Sources*. Dissertation, Computer Science and Engineering, Lehigh University, 2009.

[10] A. Qasem, D. A. Dimitrov, and J. Heflin. Efficient selection and integration of data sources for answering semantic web queries. *International Conference on Semantic Computing*, pages 245–252, 2008.

[11] A. Qasem, D. A. Dimitrov, and J. Heflin. Goal node search for semantic web source selection. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 566–569, 2008.

[12] H. Stuckenschmidt, R. Vdovjak, G.-J. Houben, and J. Broekstra. Index structures and algorithms for querying distributed rdf repositories. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 631–639, New York, NY, USA, 2004. ACM.

[13] T. Tran, H. Wang, and P. Haase. Hermes: Data web search on a pay-as-you-go integration infrastructure. *Web Semant.*, 7(3):189–203, 2009.

[14] O. Udrea, A. Pugliese, and V. S. Subrahmanian. Grin: A graph based rdf index. In *AAAI*, pages 1465–1470, 2007.

[15] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, pages 1008–1019, 2008.