Sindice.com: Weaving the Open Linked Data

Giovanni Tummarello, Renaud Delbru, and Eyal Oren

Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland

Abstract. Developers of Semantic Web applications face a challenge with respect to the decentralised publication model: where to find statements about encountered resources. The "linked data" approach, which mandates that resource URIs should be de-referenced and yield metadata about the resource, helps but is only a partial solution. We present Sindice, a lookup index over resources crawled on the Semantic Web. Our index allows applications to automatically retrieve sources with information about a given resource. In addition we allow resource retrieval through inverse-functional properties, offer full-text search and index SPARQL endpoints.

1 Introduction

The Semantic Web can be seen as a large knowledge-base formed by sources that serve information as RDF files or through SPARQL endpoints. A fundamental feature of the Semantic Web is that the graphs are decentralised: it has no single knowledge-base of statements but instead anyone can contribute statements by making them available in a public web space. These sources might have nothing in common, but by using shared identifiers (URIs) and shared terms, their information can be merged to provide useful services to both humans and software clients.

This decentralised nature of the Semantic Web, much like that of the Web, is one of its most fascinating characteristics. But for developers of Semantic Web applications, automatically finding relevant sources of information is a big challange: *how and where to find statements about certain resources*?

This paper introduces Sindice, a scalable online service that addresses exactly this question. Sindice crawls the Semantic Web and indexes the resources encountered in each source. A simple API then offers to Semantic Web application developers the ability to automatically locate relevant data sources and integrate the data from these sources into their applications.

As shown in Fig. 1, Sindice collects RDF documents from the Semantic Web and indexes these on resource URIs, IFPs and keywords. The figure shows some example RDF documents that mention Tim Berners-Lee, either by using his URI directly or by using inverse functional properties (IFPs) that uniquely identify him. Sindice offers a user interface through which human users can find these documents, based on keywords, URIs, or IFPs. More importantly, Sindice allows Semantic Web agents and clients such as Disco[1] to retrieve and integrate these

[1] `http://www4.wiwiss.fu-berlin.de/rdf_browser/`

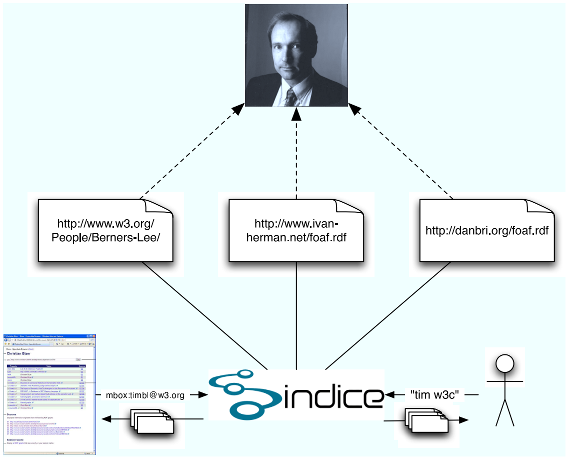

http://www.w3.org/ People/Berners-Lee/
http://www.ivan-herman.net/foaf.rdf
http://danbri.org/foaf.rdf
mbox:timbl@w3.org
sindice
"tim w3c"

Fig. 1. Linking disparate information on the Semantic Web

results into a unified information corpus for their users. Note that Sindice may return sources regardless of whether they reference each other; Sindice may also return documents that use different identifiers for the same concept, using inverse functional properties for consolidation.

1.1 Motivation: Data and Linked Data

The amount of semantically structured data available on the Semantic Web has recently grown considerably. Large and important data collections, e.g. DBLP, Wikipedia, CiteSeer, SwissProt, Geonames, are now available as retrievable RDF datasets or SPARQL query endpoints. Projects such as Bio2RDF (Belleau *et al.*, 2007) are providing real time translation and harmonization of identifiers over vast amounts of large bioscience databases. Moreover, there is a clear trend toward embedding more semantic information in conventional web pages with tecniques such as GRDDL and RDFa.

These developments make the Semantic Web a practical reality in terms of open availability of significant data. But availability of data and syntactic compatibility (e.g. RDF) is just a first step toward implementing the vision of the Semantic Web as an open and world-wide distributed source of knowledge. The next step is a Semantic Web of combined and interconnected datasets, or as an alternative, of client applications which can see such data as interconnected. Interlinked datasets with common vocabularies are not yet widespread, as shown by Ding and Finin (2006) in their study of maintenance and reuse of Semantic Web ontologies.

Several projects[2] promote the interlinked nature of data, whose main principles are, (i) that all items should be identified using URI references (instead

[2] `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

of blank nodes); (ii) that all URI references should be resolvable on the Web to RDF descriptions; and (iii) that every RDF triple should be interpreted as a hyperlink to be followed by Semantic Web browsers and crawlers (Berners-Lee, 2006; Sauermann *et al.*, 2007; Miles *et al.*, 2006).

The linked data approach relates a resource URI to a resolvable web address making the creator of the identifier the natural “official” source of information about the resource. On the one hand this approach creates the preconditions for successful Semantic Web crawling of the Semantic Web by applications and spiders. Also, it fits scenarios where entities have a clear official ownership such as personal FOAF profiles.

But on the other hand, the linked data approach alone is not sufficient to locate all relevant information about a resource, exactly because it only leads to “official”information. As a parallel in current Web search, we may consider a user looking for information about a particular mobile phone: not only the information linked from the producer’s homepage is interesting but also other opinions from the audienc at large. To aggregate, relate and link disparate sources together that provide information about the same resources, we need for a crawled overview of information from the whole (Semantic) Web.

1.2 Usage scenario

Sindice, online at `http://sindice.com`, allows its users to find documents with statements about particular resources. Sindice is in the first place not an end-user application, but a service to be used by any decentralised Semantic Web client application to locate relevant data sources. As an application service Sindice can be accessed through its Web API, for human testing and debugging we also offer an HTML front-end.

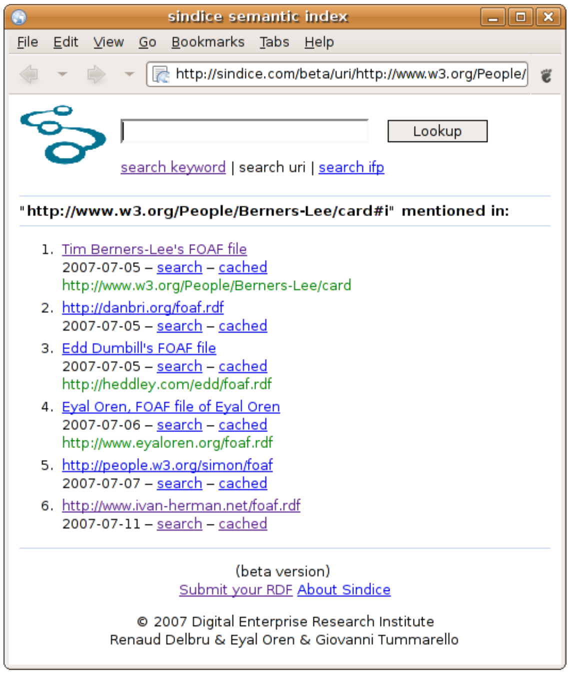
Fig. 2 displays the results of searching for the URI of Tim Berners-Lee as displayed on the HTML interface. The application interface returns the same results but in various machine-processable formats such as RDF, XML, JSON and plain text, an example is shown in Listing 1.1. In this example, several documents are returned, each of which mentions Tim Berners-Lee’s URI. The results are ranked in order of general relevance and some further information is given to enable users to choose their preferred source.

Listing 1.1. Documents mentioning Tim Berners-Lee (RDF/XML)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Description rdf:about="http://www.w3.org/People/Berners-Lee/card#i">
   <rdfs:seeAlso rdf:resource='http://www.w3.org/People/Berners-Lee/card'/>
   <rdfs:seeAlso rdf:resource='http://danbri.org/foaf.rdf'/>
   <rdfs:seeAlso rdf:resource='http://heddley.com/edd/foaf.rdf'/>
   <rdfs:seeAlso rdf:resource='http://www.eyaloren.org/foaf.rdf'/>
   <rdfs:seeAlso rdf:resource='http://people.w3.org/simon/foaf'/>
   <rdfs:seeAlso rdf:resource='http://www.ivan-herman.net/foaf.rdf'/>
</rdf:Description>

</rdf:RDF>
```


sindice semantic index
File Edit View Go Bookmarks Tabs Help
http://sindice.com/beta/uri/http://www.w3.org/People/
Lookup

search keyword | search uri | search ifp

"http://www.w3.org/People/Berners-Lee/card#i" mentioned in:

1. Tim Berners-Lee's FOAF file
   2007-07-05 – search – cached
   http://www.w3.org/People/Berners-Lee/card
2. http://danbri.org/foaf.rdf
   2007-07-05 – search – cached
3. Edd Dumbill's FOAF file
   2007-07-05 – search – cached
   http://heddley.com/edd/foaf.rdf
4. Eyal Oren, FOAF file of Eyal Oren
   2007-07-06 – search – cached
   http://www.eyaloren.org/foaf.rdf
5. http://people.w3.org/simon/foaf
   2007-07-07 – search – cached
6. http://www.ivan-herman.net/foaf.rdf
   2007-07-11 – search – cached

(beta version)
Submit your RDF About Sindice

© 2007 Digital Enterprise Research Institute
Renaud Delbru & Eyal Oren & Giovanni Tummarello

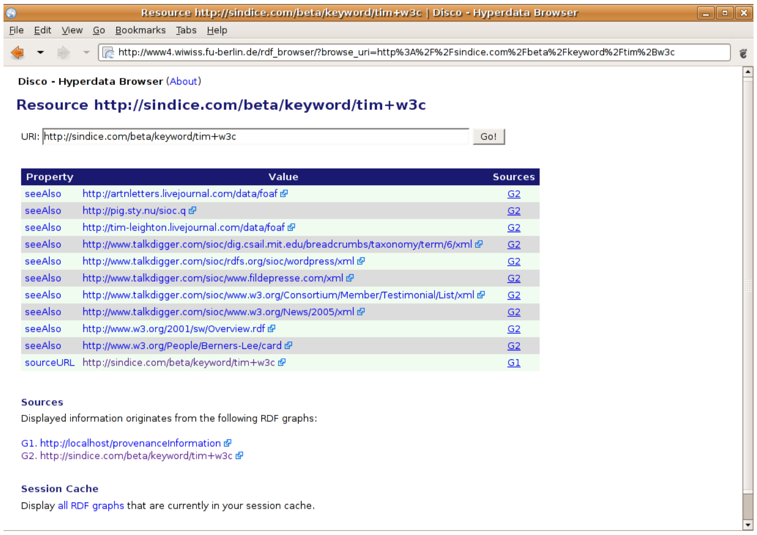
Fig. 2. Searching for documents mentioning Tim Berners-Lee (Web interface)

Sindice enables Semantic Web clients such as Piggy Bank (Huynh *et al.*, 2007) or Tabulator (Berners-Lee *et al.*, 2006) to find documents with information about a given resource, identified through an explicit URI, an inverse functional property or a keyword search. This capability fits well on top of many existing Semantic Web clients. The immediate use for Sindice inside such clients is to enable a "find out more" button, to be shown next to the available information about a resource.

Upon pressing that button, the client would contact Sindice for a ranked list of documents with more information about the resource. The user would be presented with a ranked list of these documents including a human-readable source description. The user could then choose the sources of interest (or those considered trustworthy), after which the client application could import the information from these documents. The user could maybe also select to "always consider these domains as providers of good information" to allow fully automated information import during subsequent lookups.

For clients that implement the linked data principles, integration with Sindice is trivial. Sindice behaves as a "good citizen" of the linked data Web: it provides all results as RDF that themselves follow the "linked data" principles. For example, Fig. 3 shows results from Sindice in the Disco[3] Semantic Web browser: all resulting documents are browseable resources themselves and can easily be followed.

[3] `http://www4.wiwiss.fu-berlin.de/rdf_browser/`


Resource http://sindice.com/beta/keyword/tim+w3c | Disco - Hyperdata Browser
File
Edit
View
Go
Bookmarks
Tabs
Help
http://www4.wiwiss.fu-berlin.de/rdf_browser/?browse_uri=http%3A%2F%2Fsindice.com%2Fbeta%2Fkeyword%2Ftim%2Bw3c
Go!

Disco - Hyperdata Browser (About)

Resource http://sindice.com/beta/keyword/tim+w3c

URI: http://sindice.com/beta/keyword/tim+w3c

Property	Value	Sources
seeAlso	http://artnletters.livejournal.com/data/foaf	G2
seeAlso	http://pig.sty.nu/sioc.q	G2
seeAlso	http://tim-leighton.livejournal.com/data/foaf	G2
seeAlso	http://www.talkdigger.com/sioc/dig.csail.mit.edu/breadcrumbs/taxonomy/term/6/xml	G2
seeAlso	http://www.talkdigger.com/sioc/rdfs.org/sioc/wordpress/xml	G2
seeAlso	http://www.talkdigger.com/sioc/www.fildepresse.com/xml	G2
seeAlso	http://www.talkdigger.com/sioc/www.w3.org/Consortium/Member/Testimonial/List/xml	G2
seeAlso	http://www.talkdigger.com/sioc/www.w3.org/News/2005/xml	G2
seeAlso	http://www.w3.org/2001/sw/Overview.rdf	G2
seeAlso	http://www.w3.org/People/Berners-Lee/card	G2
sourceURL	http://sindice.com/beta/keyword/tim+w3c	G1

Sources

Displayed information originates from the following RDF graphs:

G1. http://localhost/provenanceInformation
G2. http://sindice.com/beta/keyword/tim+w3c

Session Cache

Display all RDF graphs that are currently in your session cache.

Fig. 3. Application integration: Sindice results in the Disco browser

While Sindice supports, uses, and promotes the linked data model (namely in its crawling and ranking), it also supports locating information about URIs that are not URLs and cannot be de-referenced such as telephone numbers or ISBN numbers. But most importantly, Sindice helps locating statements about resources made outside their "authoritative" source.

1.3 Design principles

Sindice only acts as locator of RDF resources, returning pointers to remote data sources, and not as a query engine. Sindice is thus conceptually more close to standard Web search engines but with specific Semantic Web concepts, procedures and metrics, rather than to Semantic Web search engines such as SWSE (Hogan *et al.*, 2007) or Swoogle (Finin *et al.*, 2005) which in general aim at providing general query capabilities over the collections of all the Semantic Web statements.

By only providing pointers to sources, Sindice can avoid many of the complex issues which Semantic Web search engines must face. These issues include trust, global entity consolidation policies, voluntary or involuntary denial of services by queries of excessive complexity or on excessive data, etc. While difficult at global scale, such problems are easier to handle on the application level, e.g. by using direct user interaction, domain-specific policies or heuristics. Sindice's design supports applications that give the user full control over the considered data sources: only the information from explicitly appointed sources is used.

For this class of applications, a simple API as offered by Sindice is probably most of what is needed to connect to the global Semantic Web, without relegating control over which data sources to consider and which to ignore.

2 Sindice architecture

This section introduces the functional and non-functional requirements on the Sindice architecture and analyses whether building such a service is technically feasible on commodity hardware.

2.1 Requirements

The requirements for Sindice can be divided in functional and non-functional requirements. In terms of base functionality, Sindice offers three services to client applications: (i) it parses files and SPARQL endpoints while crawling or when "pinged" explicitly; (ii) it looks up resources (identified by their URI or by a combination of an inverse-functional property and identifying value) and returns URLs of RDF documents where these resources occur; and (iii) it searches full-text descriptions and returns the URLs of sources in which these resources occur. To fulfil these requirements, the abstract Sindice API thus consists of four methods:

- `index(url) => nil`: parses and indexes document or SPARQL endpoint at given URL,
- `lookup(uri) => url[]`: looks up a resource with given URI, returns a ranked list of sources in which that resource occurs,
- `lookup(ifp, value) => url[]`: looks up a resource uniquely identified with property-value pair, returns a ranked list of sources in which that resource occurs,
- `lookup(text) => url[]`: looks up a textual query, returns a ranked list of sources in which the given terms occur.

Additionally, we have three non-functional design requirements. First, we want to minimise the index size, so as to allow indexing of the whole (current) Semantic Web on a single commodity node without networked storage or disk-arrays. Secondly, we want to minimise lookup times, so as to allow applications to use Sindice by default to lookup more information for any encountered resource. Thirdly, we want to allow continuous live updates of the index so as to keep the index up-to-date.

2.2 Architecture design

The architecture consists of several independent components that operate in several pipelines to achieve crawling, indexing, and querying. Each pipeline will be discussed in detail in Section 3. Here we briefly introduce the overall architecture, as shown in Figure 4.

The Web frontend is the main entry point, divided in a user interface for human access and an HTTP API for machine access. Then, there are several components for crawling and indexing RDF documents. The crawler autonomously harvests RDF data from the Web and adds it to the indexing queue. If pinged (through the human interface or the API) to parse new documents, these are also added to the queue. The gatekeeper evaluates each entry in the queue and decides whether,

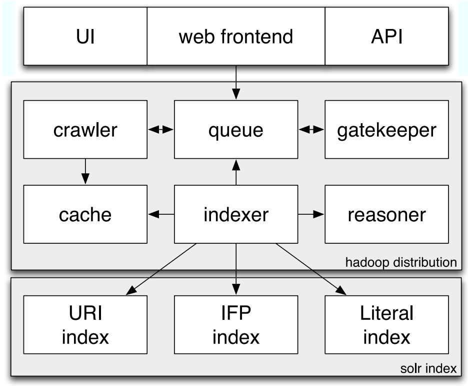


Fig. 4. Sindice architecture

and with which priority, we want to index it, based on whether we have seen the document before, its last modification date, its content digest, etc. The indexer extracts URIs, IFPs and keywords from each document (using the reasoner for IFP extraction) and adds these to their respective index. During lookup, the interface components only need to pass the queries to the relevant index, gather the results, and generate the required output such as HTML pages with appropriate layout.

The three indices store occurrences of resource URIs, resource IFPs and literals in RDF documents. The URI index contains an entry for each resource URI that lists the document URLs where this resource occurs. The IFP index is similar, except that instead of explicit resource URIs, the uniquely identifying pair $(property, value)$ is used as index key, again pointing to a list of document URLs where this pair occurs. This index allows lookup of resources with different URIs that actually identify the same real-world thing. The literal index contains an entry for each token (extracted from the literals in the documents), again pointing to a list of document URLs.

In designing the index, we optimise for disk space and lookup times. Since the only required access pattern is from resource to mentioning sources, an inverted index of URI occurrences in documents is a natural structure. In general, lookup on such an index can be performed in almost constant time over the size of the index.

2.3 Feasability

Before detailing the internals of Sindice, we analyse its feasibility. We analyse a representative sample of Semantic Web data and analyse graph-theoretical properties that allow us to predict the required index size using inverted index structures.

As an indication of required index size, we aim to store, on a single commodity machine, at least a billion unique resources. To predict the project index size for indexing such resources, we have crawled Semantic Web data for around four weeks and collected some 3.2 million unique resources. Our crawl seems a representative collection of Semantic Web data: the SWSE search engine currently contains around 11 million unique URIs excluding blank nodes[4], whereas Ding and Finin (2006) estimated the Semantic Web to contain around 10 million documents in August 2006.

The required index space in terms of indexed resources depends primarily on how often each resource is mentioned, i.e. the ratio between resources (URIs) and documents (URLs). We analysed how often the same URIs where mentioned across different documents, which is plotted in Fig. 5. The left graph shows a zoomed result in log-scale, the right graph shows the complete data in log-log-scale. These graphs demonstrate that distribution (reuse) of URIs over documents follow a power-law and therefore exhibit scale invariance. This scale-free property means that the ratio of URIs/URLs will remain constant as the Semantic Web grows which means that we can estimate the average number of times a resource will be mentioned independent of the size of the Semantic Web.

The power law found in URI occurrences is not surprising since choosing URIs is a social process and thus prone to properties such as preferential attachment that result in scale-free networks. Our result corresponds with an earlier analysis (Ding and Finin, 2006) showing other properties of Semantic Web graphs to follow power-law behaviour as well

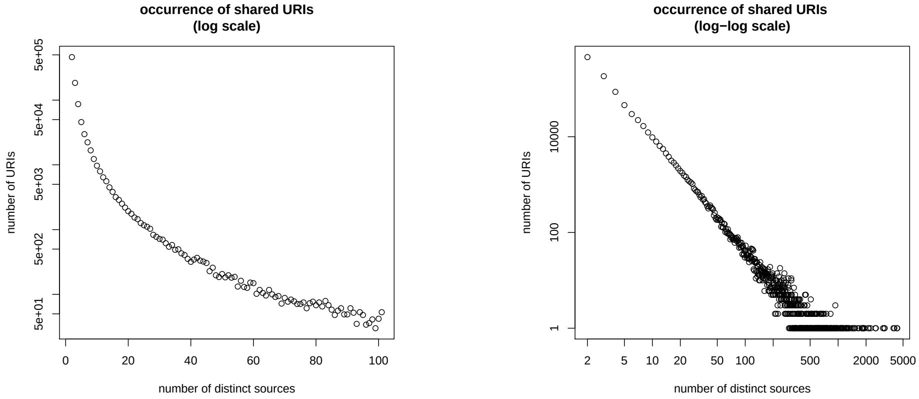


Fig. 5. Occurrence of same resources (URIs) in different documents

With respect to feasability, the index size of our crawl on the simple persistent hashtable of URI occurrences was around 2.5GB for 3.2 million URIs. Given the scale-invariance of the URI/URL ratio we can extrapolate from this data and estimate to need around 785 bytes per resource; indexing a billion unique resources would thus require around 785GB, an ordinary capacity for commodity harddisks.

[4] personal communication with A. Hogan

3 Inside Sindice

The Sindice service indexes RDF graph and then enables users or Semantic Web applications to find the location of an sources through querying. We conceptualise these two tasks by two pipelines: an *indexing* pipeline and a *querying* pipeline.

3.1 Indexing pipeline

The indexing pipeline performs a sequence of tasks, described in Fig. 6, to index an RDF graph.

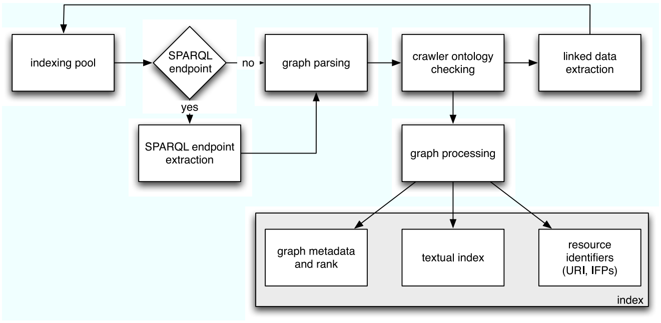


Fig. 6. Overview of the indexing pipeline

Scheduler The indexing pipeline takes as input an RSS feed or an external ping that specifies an RDF graph location. The URL, corresponding to the RDF graph location, is injected into the scheduler. The scheduler acts as a collector of URLs waiting to be processed and avoid overloading Web servers. The scheduler maintains a small hashtable with metadata for each visited source (e.g. visiting time and hash). We only revisit sources after a threshold waiting time and we only reparse the source's response if the content hash has changed.

Graph extraction The URLs in the scheduler can be of two types: RDF resources and SPARQL end points. In the first case, we retrieve the file and send it to the parser. In the case of a SPARQL endpoint, we send to the database one or more queries to extract its full content.

Graph parsing Every time an RDF graph is retrieved successfully, it is sent to the graph parser. The parser first verifies the validity of the RDF graph. It then extracts all URIs from the graph and injects them into the scheduler, following the linked data principle to treat every URI as a hyperlink to more information.

Graph processing The graph processor then extracts and indexes the full-text and all resource identifiers in the graph. We also extract indirect identifiers, namely pairs (p, o) for all inverse functional properties that are mentioned in the graph. Extracting these IFPs requires two additional steps: the recursive expansion of the graph with all its schema information, followed by OWL inferencing to identify all IFPs in these schemas (described below).

Finding inverse functional properties A graph typically does not explicitly mention which of its properties are inverse functional; rather, the graph would refer to one or more schema definitions, which may recursively depend on other schemas.

We therefore proceed as follows: we fetch the definition of each property, following the linked data principle, by de-referencing its URI and importing the returned definition into the RDF graph. This "schema fetching" task is repeated recursively until fixpoint. At this point we perform inferencing using the OWL "ter Horst" fragment (ter Horst, 2005). This fragment is expressive enough for finding inverse functional properties and has efficient entailment prodedures. After this reasoning step, we query the combined model for IFPs and proceed with the creation of identifiers as explained above.

Reasoning cache To improve the graph processing we cache the reasoner's output. Since reasoning over a large schema to find all inverse functional properties is computationally expensive it could quickly form an indexing bottleneck. However, the reasoning results need to be separated for each document, to prevent malicious users from "infecting" results on a global scale by defining a property to be inverse functional (while it is not).

The graph processor therefore uses a caching system based on the set of properties explicitly used in a graph. For example, if a FOAF profile mentions some properties and classes such as `foaf:name`, `foaf:Person`, we only perform the graph reasoning if that exact set of properties and classes has not occurred before. Then, after graph expansion and identifying the set of IFPs (e.g. `foaf:mbox` and `foaf:personalHomepage`) we cache the found IFPs using the set of properties and classes in the original document as cache key.

3.2 Querying pipeline

The querying pipeline is on the other hand splitted into three main stages: the index retrieval, the ranking phase and the result generation.

Index retrieval The query is looked up in the inverted index, which can be implemented either as an on-disk hashmap or in an information retrieval engine. The list of results is cached for later reuse, and is invalidated daily to keep the result up-to-date.

Ranking phase After index retrieval the results are ranked according to various metrics. Since for popular resources our index could easily return many hundreds or thousands sources, providing a ranked list is crucial.

We have designed a ranking function that requires only little metadata for each source and is relatively fast to compute; it does not construct a global ranking of all sources but ranks sources based on their own metadata and external ranking services. The following metadata values for each source are computed and their value combined using an unweighted average:

- Hostname: we prefer sources whose hostname is the same as the resource's hostname, in support of the linked data paradigm. For example, we consider that more information on the resource `http://eyaloren.org/foaf.rdf#me` can be found at the source `http://eyaloren.org/foaf.rdf` than at an arbitrary "outside" source, such as `http://g1o.net/g1ofoaf.rdf`.
- External rank: we prefer sources hosted on sites which rank high using traditional Web ranking algorithms. These rankings can be purchased or can be estimated using various techniques (Davis and Dhillon, 2006).
- Relevant sources: we prefer sources that share rare terms (URIs, IFPs, keywords) rather than common terms with the requested terms. This relevance metric is comparable to the TF/IDF relevance metric (Frakes and Baeza-Yates, 1992) in information retrieval.

Result generation Once the resulting data sources are ranked into order of importance, Sindice can export them into different syntaxes, such as the HTML Web interface, RDF, XML, JSON, and plain text.

3.3 Crawling ontology

On February the 2nd 2007, Geonames experienced what has been called "the first distributed denial of service on the Semantic Web"[5]. What happened, however, was not due to malicious behaviour but rather due a Semantic Web Crawler simple a bit too fast in following the linked data paradigm and resolving each of the 6.4 million URL/URI on the server. As the result of each one of these calls is a query, the load on the server was in the end very high leading to the denial of service. It has been suggested that this could have been avoided if the geoname database RDF dump, which was in fact being made available, had been imported as an alternative to full site crawling. But how could the spider have known that?

Our crawling ontology[6] helps Semantic Web spiders and clients alike in indexing and operating over large quantity of Semantic Linked data. Through this ontology, a site administrator can avoid denial of services and ensure that the data will be used in the form which is optimal for the task.

The crawling ontology augments the existing `robots.txt` protocol; all relevant statements should be located in a `srobots.rdf` file, on the site root as done in the `robots.txt` file. The main class to be used is the `DataEquivalenceStatement`, which states that data represented in three possible ways (linked data, SPARQL endpoint, or data dump) is equivalent, allowing the client to choose one of these representations

[5] `http://geonames.wordpress.com/2007/02/03/friendly-fire-semantic-web-crawler-ddos/`

[6] `http://sindice.com/srobotsfile`

To follow the protocol, a client retrieving a resource URL should first check for the traditional robots.txt; next, it should check for the existence of srobot.rdf, to see whether the URL is maybe part of a larger dataset that can be downloaded instead. To increase awareness of the srobot.rdf file, a link to it can be also provided as a ”see also” statement both in the RDF data dump as a description of the file location URL itself and in the RDF returned by resolving the linked data URLs. Multiple Data Equivalence Statements can be used as necessary for different data sets served by the same host.

At the time of writing, we are working on an updated version of this ontology after feedback from RDF data providers. The new version of this ontology[7] covers the material discussed above as an extension of the Sitemap protocol[8] for web document discovery.

4 Related work

We are aware of two Semantic Web search engines that index the Semantic Web by crawling RDF documents and then offer a search interface over these documents. SWSE[9] crawls not only RDF documents but also “normal” HTML Web documents and RSS feeds and converts these to RDF (Harth *et al.*, 2007; Hogan *et al.*, 2007). SWSE stores all triples found in the crawling phase including their provenance and offers rich queries, comparable to SPARQL, over these quads. Similarly, Swoogle (Finin *et al.*, 2005) crawls and indexes the Semantic Web data found online.

Some differences between these engines and Sindice have already been highlighted during this paper. To the best of our knowledge, none of these engines seem to display continuous crawling capabilities, probably due to the cost and complexity of updating an index which can answer relational queries. Also, although SWSE allows IFP lookups in its query capabilities, it does not perform reasoning to extract these IFPs but instead extracts only several hardcoded properties. Finally, none of these engines provide indexing based on “linked data” paradigm reasoning, SPARQL endpoint indexing and the ability to index large repositories consciously through the Sitemap extention.

Table 1 shows an overall comparison of our approach against on the one hand traditional Web search engines such as Google or Yahoo! and on the other hand Semantic Web (SW) search engines such as SWSE or Swoogle. Whereas traditional Web search focuses on document retrieval for HTML documents, and SW search focuses on building a global database of retrieved triples, we provide a document retrieval service for RDF documents. Sindice is thus conceptually close to traditional Web search engines but employs different ways of finding documents and indexes more than only natural language texts).

Finally, our service is related to http://pingthesemanticweb.com, which maintains a list of recently updated documents and currently lists over seven

[7] http://sw.deri.org/2007/07/sitemapextension/

[8] http://www.sitemaps.org/protocol.html

[9] http://swse.deri.org/

	Web search	SW search	Sindice
focus	document retrieval	global database	SW document retrieval
orientation	Web documents	triples/quads	RDF documents
URI lookup	–	+	+
IFP lookup	–	±	+
scalability	+	±	+
full queries	-	+	-
SPARQL indexing	-	-	+

Table 1. Approaches in (Semantic) Web information retrieval

million RDF documents. The service does not perform indexing and does not allow lookups over its content, it does offer a periodical dump of updated documents. We see the service as a companion to Sindice and we in fact use it as input to our crawling pool.

5 Conclusion

We have presented Sindice, a public API to locate the sources of Semantic Web annotations, be these RDF files or SPARQL endpoints. By committing to this simple service Sindice provides a much needed interweaving framework for the Semantic Web achieving very high scalability while maintaining overall neutrality on issues such as trust, reputation, ontologies and identifiers. Such neutrality is key to the use of Sindice regardless of the needs and purpose of the Semantic Web client: clients will be free to chose their sources according to their preferences, requirements and possibly aided by the assistance and supervision of the end user.

Apart from the Sindice service as such, the contributions of this paper are: a crawling ontology which can avoid the high inefficiency of blindly navigating sites that support the "linked data" paradigm; a strategy for calculation of IFPs as found in crawled RDF files ; a simple strategy to locate human-readable descriptions of RDF files and their web ranking; a result strategy strategy and finally the discovery of a power-law behaviour regarding URI reuse on the Semantic Web.

In future work, we aim to provide a Sindice client-side library that implements useful procedures such as smushing or equality reasoning, using "linked data" practices and involving potentially many Sindice calls. We are also working on a distributed index, and potentially a distributed pinging architecture as well. Finally, Sindice services could be provided over UDP, removing the time needed for TCP handshake to provide a more responsive service.

Acknowledgements This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694.

References

F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, *et al.* Bio2RDF: Towards a mashup to build bioinformatics knowledge system. In *Proceedings of the WWW*

Workshop on Health Care and Life Sciences Data Integration for the Semantic Web. 2007.

T. Berners-Lee. *Linked Data*. W3C Design Issues, 2006.

T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, *et al.* Tabulator: Exploring and analyzing linked data on the Semantic Web. In *Proceedings of the ISWC Workshop on Semantic Web User Interaction*. 2006.

J. V. Davis and I. S. Dhillon. Estimating the global pagerank of web communities. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 116–125. 2006.

L. Ding and T. Finin. Characterizing the Semantic Web on the web. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2006.

T. W. Finin, L. Ding, R. Pan, A. Joshi, *et al.* Swoogle: Searching for knowledge on the semantic web. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 2005.

W. B. Frakes and R. A. Baeza-Yates, (eds.) *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992.

A. Harth, J. Umbrich, and S. Decker. Multicrawler: A pipelined architecture for crawling and indexing Semantic Web data. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2007.

A. Hogan, A. Harth, and S. Decker. Reconrank: A scalable ranking method for semantic web data with context. In *Second International Workshop on Scalable Semantic Web Knowledge Base Systems*. 2006.

A. Hogan, A. Harth, J. Umbrich, and S. Decker. Towards a scalable search and query engine for the web. In *Proceedings of the International World-Wide Web Conference*. 2007. Poster presentation.

H. J. ter Horst. Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *Proceedings of the International Semantic Web Conference (ISWC)*, pp. 668–684. 2005.

D. Huynh, S. Mazzocchi, and D. Karger. Piggy bank: Experience the Semantic Web inside your web browser. *Journal of Web Semantics*, 5(1):16–27, 2007.

A. Miles, T. Baker, and R. Swick, (eds.) *Best Practice Recipes for Publishing RDF Vocabularies*. W3C Working Draft, 2006. `http://www.w3.org/TR/swbp-vocab-pub/#redirect`.

L. Sauermann, R. Cyganiak, and M. Völkel. Cool URIs for the Semantic Web. Tech. Rep. TM-07-01, DFKI, 2007.