Sindice.com: A Document-oriented Lookup Index for Open Linked Data

EYAL OREN
The Network Institute
Vrije Universiteit Amsterdam
RENAUD DELBRU, MICHELE CATASTA, RICHARD CYGANIAK
Digital Enterprise Research Institute
National University of Ireland, Galway
HOLGER STENZHORN
Institute of Medical Biometry and Medical Informatics, University Medical Center Freiburg, Stefan-Meier-Str. 26, 79104 Freiburg, Germany
GIOVANNI TUMMARELLO
Digital Enterprise Research Institute
National University of Ireland, Galway

Developers of Semantic Web applications face a challenge with respect to the decentralised publication model: how and where to find statements about encountered resources. The "linked data" approach mandates that resource URIs should be de-referenced to return resource metadata. But for data discovery linkage itself is not enough, and crawling and indexing of data is necessary. Existing Semantic Web search engines are focused on database-like functionality, compromising on index size, query performance and live updates. We present Sindice, a lookup index over resources crawled on the Semantic Web. Our index allows applications to automatically locate documents containing information about a given resource. In addition, we allow resource retrieval through uniquely identifying inverse-functional properties, offer a full-text search and index SPARQL endpoints. Finally we introduce an extension to the sitemap protocol which allows us to efficiently index large Semantic Web datasets with minimal impact on the data providers.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

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

This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694. Holger Stenzhorn is also affiliated with the Digital Enterprise Research Institute in Galway, Ireland.

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 challenge: *how and where to find statements about certain resources*?

This article 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.

Sindice collects RDF documents from the Semantic Web and indexes these on resource URIs, inverse functional properties (IFPs) and keywords. 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 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: an interlinked web of data

The amount of structured data available on the Semantic Web which could support our scenario has recently grown considerably. Large and important data collections such as DBLP[2], Wikipedia article links[3] and infoboxes[4], UniProt[5] and Geonames[6], are now available as retrievable RDF datasets or as SPARQL endpoints. Projects such as Bio2RDF [Belleau et al. 2007] provide real-time mapping and consolidation of identifiers over vast amounts of bioscience databases. Moreover, there is a trend towards embedding RDF in conventional web pages using techniques such as GRDDL[7] and RDFa[8].

These developments make the Semantic Web a practical reality in terms of open availability of significant data. But syntactic compatibility through RDF of data is only a first step toward the vision of the Semantic Web as an open and world-wide distributed source of knowledge. The Semantic Web enables applications to integrate data from distributed sources at runtime, by simply merging the graphs in the data sources. Due to the reuse of the same global identifiers across the graphs, the distributed sub-graphs can be connected to form a single, virtual, graph. But to combine data from multiple sources, application developers face two challenges, due to the decentralised publishing infrastructure of the Semantic Web: first, they need to discover the data sources that they want to integrate into their application;

[1] http://www4.wiwiss.fu-berlin.de/rdf_browser/
[2] http://www4.wiwiss.fu-berlin.de/dblp/
[3] http://labs.systemone.at/wikipedia3
[4] http://dbpedia.org
[5] http://www.isb-sib.ch/~ejain/rdf/
[6] http://geonames.org
[7] http://www.w3.org/TR/grddl/
[8] http://www.w3.org/TR/xhtml-rdfa-primer/

secondly, in order to combine the graphs, they need to identify common resources across the sub-graphs.

For the first challenge, applications could crawl the Semantic Web by Interpreting resource identifiers as hyperlinks that can be followed, as mandated by the "linked data" approach[9]. The main principles of linked data are, (i) that all items should be identified using URI references (instead 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 linkage alone is not sufficient since it only discovers sources recursively linked to by the original data owner [McBryan 1994]. A more inclusive discovery process which aggregate, relates and links disparate sources together that provide information about the same resources requires crawling and indexing of Semantic Web data [Ding et al. 2004].

For the second challenge, identifying common resources inside data sources to connect the sub-graphs, application developers can rely on shared reuse of URIs (which identify a resource directly) and on OWL's inverse functional properties (which uniquely identify a resource indirectly). However, interlinked datasets through reuse of common vocabulary and shared URIs are not yet widespread, as shown by Ding and Finin [2006] in their study of maintenance and reuse of Semantic Web ontologies. Also, performing the required OWL reasoning to find the inverse functional properties in a graph can be computationally expensive and prevent efficient indexing. Again, since this discovery and identification process is needed in every Semantic Web client that integrates online data sources, application developers would benefit from an architecture providing these as a service.

To address both challenges in discovering decentralised Semantic Web data, an architectural element is required that creates the missing links and allows Semantic Web clients to find and integrate independent sub-graphs into a single, virtual, large graph. This element is offered by Sindice, our lookup service for source discovery.

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. 1 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

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

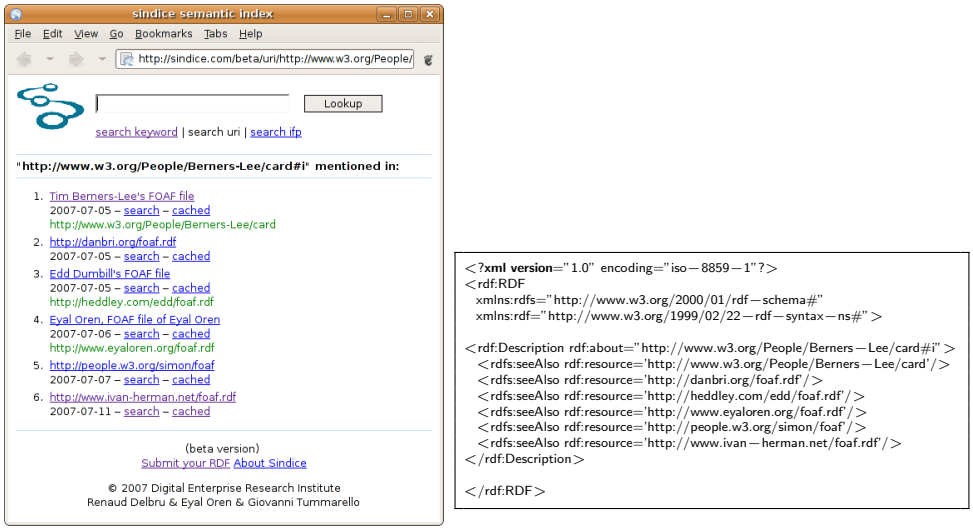

Fig. 1. Documents mentioning Tim Berners-Lee, in the Web interface (left) and in the API (right)

text. We can see that several documents are returned that each mention his URI. For each result some further information is given (human description, date of last update) to enable users to choose the best suitable source. The results are ranked in order of relevance.

Sindice enables Semantic Web clients such as Piggy Bank [Huynh et al. 2007], DBin [Tummarello et al. 2006] and 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 information currently known 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. 2 shows results from Sindice in the Disco[10] Semantic Web browser: all resulting documents are browseable resources themselves and can easily be followed.

While Sindice supports, uses, and promotes the linked data model (namely in its crawling and ranking), it also supports locating information about URIs that

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

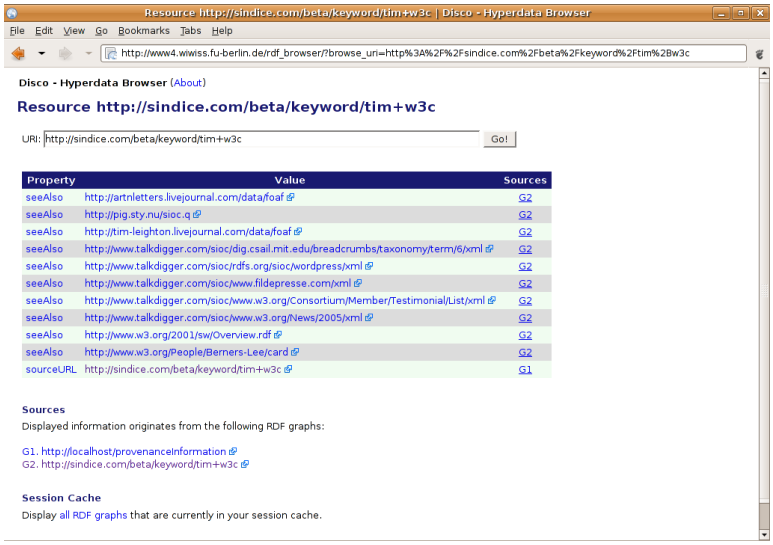

Fig. 2. Application integration: Sindice results in the Disco browser

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 Outline

The rest of this article will proceed as follows. Section 2 introduces Sindice's design principles: to act only as document locator, to be document-oriented, and to offer continuous live updates over the index. Section 3 introduces the requirements and architecture and discusses the technical feasibility of implementing the requirements while adhering to our design principles, while Section 4 introduces the three operational pipelines for crawling, indexing, and querying that form the core of our architecture. Section 6 describes the overall set of tools that Sindice offers to the data publishers. These include a validation service to ensure data is published well-interlinked and introduces an extension to the robots protocol to ensure published data will be found and indexed by Semantic crawlers. We then discuss related work in Section 7 and conclude in Section 8.

2. DESIGN PRINCIPLES

During the design of Sindice, or in general, of a component for discovering Semantic Web data, several decisions need to be made with respect to for example query functionality, inferencing support, and physical index distribution. This section explores each of these design decisions in general and explains which choices we have made in the design of Sindice, influenced by the specifics of our use case: supporting application developers in locating relevant data sources with common resources and allowing them to connect these sub-graphs.

Query functionality. The offered query functionality of any Semantic Web index is a trade-off between expressiveness and performance. On the one end, a minimum

query functionality would be simple term lookups, similar to the basic model of current Web search engines [Broder 2002]. On the other hand, a maximal query functionality would be similar to GraphLog-like [Consens and Mendelzon 1990] or Datalog-like queries [Ullman 1989] over the graph of triples.

This trade-off is influenced by one's view of the relation between the Web and the Semantic Web. When regarding the Semantic Web as a large collection of RDF documents, it would be natural to index these documents through information retrieval techniques such as inverted indices [Salton and McGill 1983; Manning et al. 2008] and to offer simple lookup queries over the indexed terms. When regarding the Semantic Web as a large connected space of triples, it would be natural to index these triples through database techniques and to offer conjunctive queries over the full set of triples.

In Sindice, we take the first approach: we index all identifiers in (URIs and IFPs) and literal words in the graph, allow lookups over these identifiers and return pointers to sources that mention these terms. For our scenario only lookups over common identifiers are required: after providing clients with pointers to relevant data sources, the clients can process the triples in these sources themselves.

Inferencing support. Since the data that we are indexing has a particular and well-defined semantics, again a trade-off must be made regarding the offered inferencing support: through inferencing rules such as the RDF, RDFS, or OWL entailment regimes [Hayes 2004; McGuinness and van Harmelen 2004], more precise and more complete answers can be given, in exchange for additional computational costs during document processing or during query answering.

Focusing only on the indexing of resource identifiers (URIs and IFPs), the relevant inferences come from the OWL vocabulary that defines implicit equality through (inverse) functional properties. The exact set of identifiers that we can extract from each document depend on the properties and schemas used in the document: for example, the usage of an inverse functional property would result in an indirect identifier. To correctly compute all identifiers in the document we should therefore not only perform the inference computations but first recursively fetch and import all the referenced schemas.

Access ethics. Similar to traditional Web crawling, collecting and indexing data from Web sites involves ethical issues with regard to the behaviour of the crawler. Crawling should be done in accordance with accepted ethics of "good behaviour" [Thelwall and Stuart 2006]. An element of such ethics, related to privacy and crawling costs, is partially addressed by the robots exclusion protocol[11], which enables site owners to partially disallow crawling on their site and is adhered to by all crawlers of commercial search engines. Since crawling can incur considerable costs for data providers, due to bandwidth cost and a degradation or even denial of service to other clients, crawlers should respect accepted standards. Micropayments have been suggested to secure proper crawler behaviour [Krogh 1996]; crawlers could also share a portion of the economical value gained from the crawled data with the data provider [Thelwall and Stuart 2006], through advertisements or click-through compensation.

[11] `http://www.robotstxt.org/wc/exclusion.html`

In Sindice we try to respect the privacy and resource limits of data publishers by adhering to the robots protocol and by our extension of the Sitemap protocol allowing data publishers to indicate how to best crawl their site and its data: which data to ignore, which data to index iteratively and where to find large collected dumps of data.

Data ownership. In terms of data access, further consideration must be given to issues such as copyright and privacy. These issues are also relevant in traditional Web crawling, but are more pertinent here due to the higher economic value of Semantic Web data. In terms of copyright and data ownership, some argue that Web data is in the public domain and can thus be crawled freely while others suggest that data should only be aggregated, indexed and analysed with the informed consent of its owners [van Wel and Royakkers 2004; Lin and Loui 1998; Tavani 1999]. More important is the copyright and other intellectual property rights of the data publishers; although the notion of "fair use" allows usage and indexing of copyrighted material for purposes such as news reporting or research [Landes and Posner 2003; Depoorter and Parisi 2002; Erickson 2003], major economic value gained by crawling and indexing copyrighted content might not be considered to fall under fair use, as in the Napster case [Klein et al. 2002] or the case around Google's indexing of Belgian newspapers [van Asbroeck and Cock 2007].

To properly address the ownership rights of data providers, one could employ a "push" model instead of the traditional "pull" model of Web crawlers. If data providers are offered a way to notify the indexing service of new data, they can explicitly give their consent to the indexing of that data and could attach appropriate clauses for the use of that data. In Sindice we currently rely on a combination of a "pull" approach and a "push" approach: we crawl data providers but we also use offer a notification service through which data providers can explicitly ask us to index their data.

Another design possibility with respect to data ownership, is to let the data providers index their own data and submit the compressed index to our centralised index. In that manner, data providers can make use of the lookup index to advertise their data to potential customers while retaining control over which information is disclosed to these clients. We have not implemented this model in Sindice yet but are investigating it for future work: when Semantic Web data increases in value, data providers will want to advertise it without publishing it for free.

Index construction. Considering the scale and growth rate of most Web data, indexing all Semantic Web data will result in very large indices. Our current index and short-term estimates (discussed in Section 3.3) show that projected sizes will fit on one commodity machine for some while. But since Semantic Web data will likely continue to grow exponentially [Finin and Ding 2006], distributed processing and indexing will be required; our parallel and distributed architecture is designed to do so.

To index Semantic Web data we need to consider three types of limited resources: storage space, processing power and network bandwidth. Network bandwidth is required to crawl and fetch Semantic Web documents, processing power in order to perform the OWL reasoning and extract the relevant identifiers. Storage space is required for holding the indexed data until query time but also potentially to

store all encountered Semantic Web data to later perform off-line, in-depth, data analysis.

In terms of network bandwidth and processing power, one possibility for decentralisation is to let data providers index their own data and to send us the indexed data, as discussed above. Since indices are typically at least one order of magnitude smaller than the raw data, local index construction would save network consumption and share the processing costs over all data providers (which would probably need to be compensated for constructing the index). As mentioned before, we have not yet explored this possibility.

In Sindice, to be able to construct and store the index in a scalable manner, we cluster our machines into a parallel architecture with shared storage space, which allows us to address the scarcity of processing power and storage through simple scaling of commodity hardware. Sindice uses the Hadoop[12] parallel architecture, which is inspired by the job management architecture MapReduce [Dean and Ghemawat 2004], the distributed lightweight database Bigtable [Chang et al. 2006] and the distributed filesystem GFS [Ghemawat et al. 2003].

3. SINDICE ARCHITECTURE

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

3.1 Functional 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,

[12] `http://lucene.apache.org/hadoop`

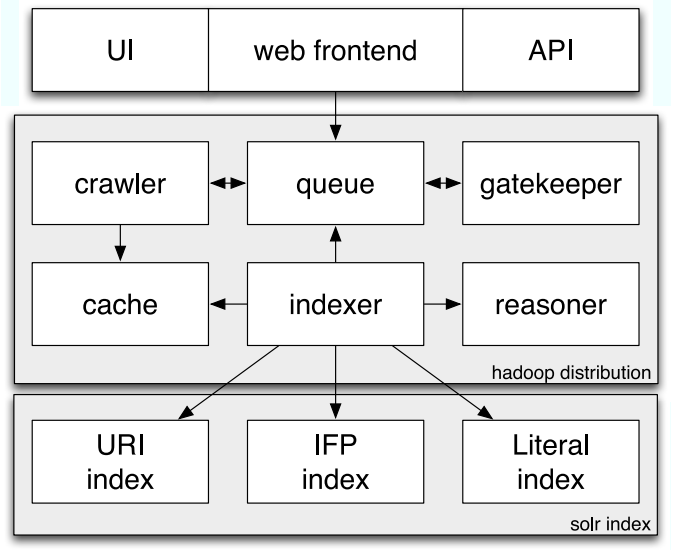

Fig. 3. Sindice architecture

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.

3.2 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 4. Here we briefly introduce the overall architecture, as shown in Figure 3.

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, 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.

All components except the frontend and the actual index are distributed over arbitrary parallel nodes using Hadoop[13], a parallel architecture based on the MapReduce paradigm [Dean and Ghemawat 2004].

[13] `http://lucene.apache.org/hadoop`

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. Lookups that return a large list of documents cause longer query times, especially ontology classes such as `foaf:Person` or `sioc:Forum` that appear in many documents (in membership definitions). The straightforward solution is to either eliminate these occurrences as stopwords or to return only a limited set of results (top k).

Technically these indices have been implemented both as an on-disk persistent hashtable, mapping from resource URIs to mentioning documents, and in the Solr[14] information retrieval engine. The on-disk hashtable is conceptually simple but less efficient in practise because it lacks optimisations, such as distributed indexing, efficient sort algorithms, index compression, and caching, that are standard in information retrieval engines.

3.3 Feasibility

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. We can then cluster and replicate such index structures to provide service in a round-robin fashion.

As an indication of required index size, informed by earlier results on the size of the Semantic Web [Ding and Finin 2006], 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.

Compared to other crawls, ours seems a representative collection of Semantic Web data. For example, the SWSE search engine contained around 11 million unique URIs in May 2006[15], whereas Ding and Finin [2006] estimated the Semantic Web to contain around 10 million documents in August 2006 (although the first counts URIs and the other counts documents, they are in the same order of magnitude).

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). This ratio is important because inverted indices are compressed

[14] `http://lucene.apache.org/solr`
[15] personal communication with A. Hogan

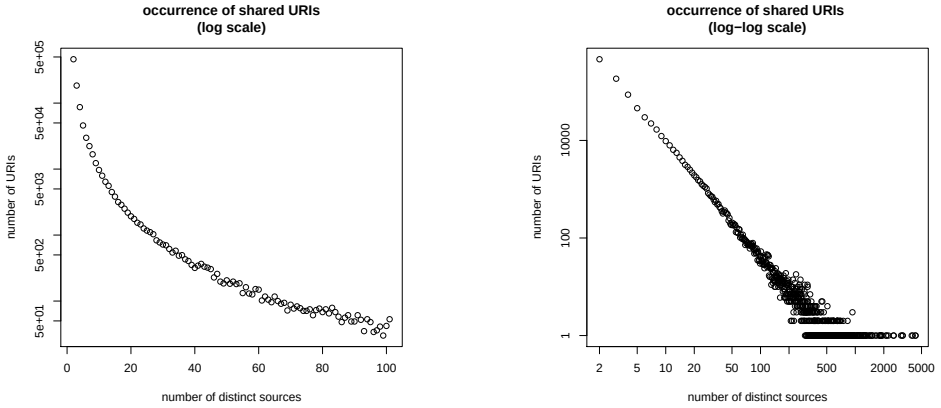

Fig. 4. Occurrence of same resources (URIs) in different documents

for optimised storage (and for result caching) and the compression depends on the occurrence distribution of URIs. In ordinary text documents, words are estimated to follow a Zipfian distribution (a type of power law) [Li 1992; Zipf 1932].

We analysed how often the same URIs where mentioned across different documents, which is plotted in Fig. 4. 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 terms (URIs) to use in an RDF description is a social process and thus prone to properties such as preferential attachment that result in scale-free networks. Also, our result corresponds with earlier analysis that showed that many other properties of Semantic Web graphs also follow power-law behaviour [Ding and Finin 2006].

As we can see, URIs on the Semantic Web seem to follow a similar distribution to words in natural-language documents [Li 1992; Zipf 1932]. Because of this similarity between term distribution of URI occurrence in Semantic Web documents and term distribution of words in natural-language documents, we expect that applying existing information retrieval techniques on Semantic Web data will exhibit similar scalability behaviour as on ordinary documents. In Section 5 we empirically evaluate the scalability of our index structure using real-world data.

4. OPERATIONAL PIPELINES

As explained in the previous section, Sindice consists of three major tasks: crawling the Semantic Web, indexing found RDF documents, and allowing lookup queries to locate documents mentioning a resource. Each of these tasks can be broken into processing pipelines: a *crawling* pipeline, an *indexing* pipeline and a *querying* pipeline.

4.1 Crawling pipeline

We aim to provide a live view of the HTTP-retrievable Semantic Web, to reward and encourage people that publish RDF files by allowing them to quickly observe their information being reused. Crawling and finding those documents is the goal of the crawling pipeline which is devoted to discovering new RDF documents and refreshing previously indexed sources. We focus on maximising index quality while adapting to constrained network resources and minimising the impact on remote servers.

4.1.1 *Source verification.* The first stage of the pipeline is a pool of unverified starting points. This pool is continuously being fed by direct user submission via the Sindice user interface and API, via pings from other services such as `pingthesemanticweb.com` and by finding new source candidates at many stages of the crawling process itself.

The first step of the crawling pipeline is to decide whether sources in the *unverified sources pool* indeed contain Semantic Web data, by being either an RDF document or a SPARQL endpoint: if they do, they are moved to the *verified sources pool*. If sources do not respond or give errors, they are removed from the pool. If the source is not an RDF source, we verify whether it links to RDF data in various ways. We currently follow two approaches to discover such linked RDF data: through an HTML "meta" link, as is current practise to link from e.g. personal homepages to FOAF profiles, or through an ordinary HTML "href" link pointing to documents with an ".rdf" extension.

A similar processing algorithm is used to periodically refresh indexed sources. The handling of these sources is more conservative: locations that timeout or return errors are ignored but left in the pool and retried at a later time.

4.1.2 *Source scheduler.* Sources from the *verified resources pool* are selected and moved into the indexing pipeline by the scheduler. The scheduler tackles a common problem in Web crawling [Pant et al. 2004; Aggarwal et al. 2001]: deciding which sources to visit next. Scheduling is based on a notion of fairness to avoid starving a source because other sources are continuously indexed first, on avoiding getting "stuck" in large isolated datasets such as LiveJournal or DBpedia, and on preventing server overload of data providers.

For each previously visited source, the scheduler stores metadata such as visiting time and content hash. We only revisit sources after a threshold waiting time and only if the content has been modified, detected through an HTTP "last-modified-since" header if provided by the server or through a content hash otherwise.

4.2 Indexing pipeline

The indexing pipeline performs a sequence of tasks, described in Fig. 5, to index an RDF graph. 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.

Then, the parser checks if the graph contains some specific crawler ontology rules (detailed in Sect. 6.1). For example, a crawler ontology rule can indicate that

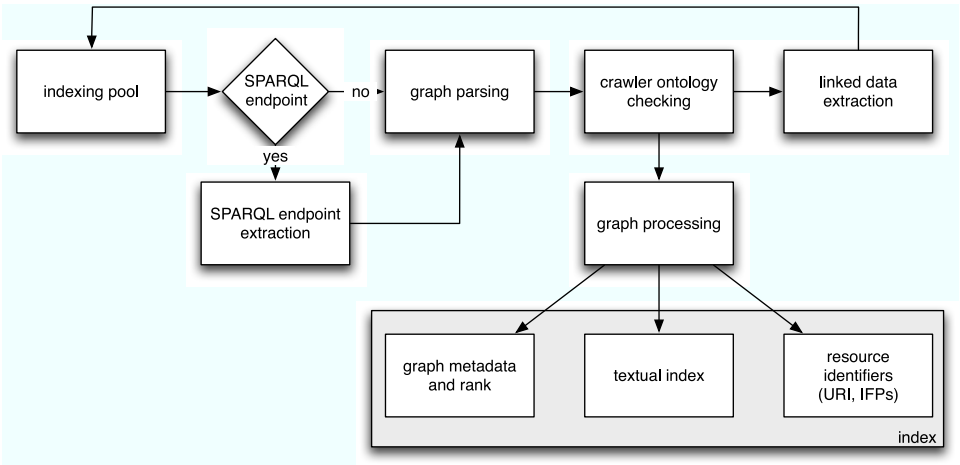

Fig. 5. Overview of the indexing pipeline

another graph should be indexed instead (e.g. a file containing a complete dump) in which case the process is stopped and the new URL is instead added to the indexing queue. If the crawler ontology permits indexing the graph, or if no crawler rules are specified, the graph is sent to the "graph processor".

The graph processor then extracts and indexes all full-text (tokenised) and one or more identifiers for each resource in the graph. These resource identifiers can be explicit URIs that directly identify the resource: these are straightforward to extract. We also extract indirect identifiers, namely pairs (p, o) for all inverse functional properties that are mentioned in the graph, which allows searching for blank nodes which do not have an explicit identifier.

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). After inferencing, the graph processor extracts graph metadata, such as a human-readable description (`rdfs:label`, `dc:title` and all their subproperties) and the total size of the graph.

4.2.1 *Finding IFPs.* 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 in turn could define certain properties as inverse functional properties. But these schemas may in turn depend on other schemas, so we need to recursively fetch each schema definition until fixpoint. Also, inverse functional properties need not be defined explicitly, but could be derived from various RDFS and OWL inferencing rules such as subproperty relations or cardinality restrictions. We therefore also need to perform reasoning over the recursively fetched schema definitions.

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] as implemented in the OWLIM reasoner [Kiryakov et al.

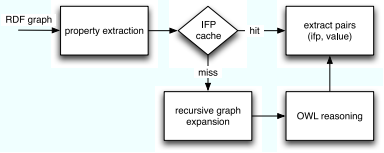

Fig. 6. Details of the graph processing details

2005]. This fragment is expressive enough for finding inverse functional properties and has efficient entailment procedures. After this reasoning step, we query the combined model for IFPs and proceed with the creation of identifiers as explained above.

We have to perform this reasoning for each graph, not because the schemas are likely to change continuously, but because in the RDF(S) semantics every statement we find in every document might influence whether a property is determined to be inverse functional. Thus, a malicious user could define some common property such as `foaf:name` to be inverse functional (which it is not) and derail the indexing procedure. To prevent this we need to treat each document separately and perform the reasoning only on this document plus all the "imported" (referenced) schemas (by using a schema element one is assumed to trust and agree with the schema definition).

4.2.2 *Reasoning cache.* To improve the graph processing we apply two caches in the reasoning process, one caches the reasoner's input and the other the reasoner's output. With regards to the input, we need to recursively fetch all schema definitions as described above. An RDF graph can use many different schemas and the number of property definitions to import can be large. Since even fetching these schemas can cost significant time delays, we cache all schemas which were successfully fetched.

With respect to the reasoner's input cache, a complication involves the use of HTTP 303 redirection. Many schemas are published using the recipe [Miles et al. 2006] that suggests that, when receiving an HTTP request for a schema element, such as `foaf:name`, the server should redirect the client, using the HTTP 303 code, to the complete schema specification, in this case `http://xmlns.com/foaf/spec`. But, since the HTTP specification[16] does not allow caching for 303 redirects, we have to request these schema elements repeatedly, only to be redirected continuously. In order to resolve this unfortunate side-effect of the recipe, it was necessary to implement our own HTTP cache system that allow the caching of the 303 redirects.

The second form of caching involves the reasoning results. Reasoning over a large schema to find all inverse functional properties is computationally expensive and can quickly form an indexing bottleneck. As explained above, we have to separate the reasoning tasks for each document, to prevent malicious users from "infecting" results on a global scale.

To minimise reasoning time without compromising on safety, the graph processor has a reasoning caching system based on the set of properties explicitly used in a

[16] `http://www.w3.org/Protocols/rfc2616/`

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.

4.3 Querying pipeline

The querying pipeline is run when users or clients perform lookups on the index. The querying pipeline is split into three stages: the index retrieval, the ranking phase and the result generation. The first step retrieves the relevant results, the second step orders these according to several scoring metrics, and the third one generates the output requested by the client (e.g. HTML for the Web interface, RDF, XML, JSON or plain text for access through the API).

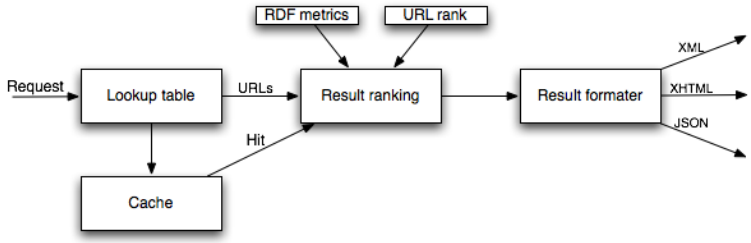

Fig. 7. Overview of the querying pipeline

If the query has been cached in a previous lookup, we skip the retrieval and ranking step and proceed directly to output generation. Otherwise 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 as explained before. The list of results is cached for later reuse, and is invalidated daily to keep the result up-to-date.

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; in contrast to more optimal ranking functions such as HITS [Kleinberg 1999], PageRank [Brin and Page 1998], or ReconRank [Hogan et al. 2006], it does not construct a global ranking of all 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. When the inverted index is implemented on top of the information retrieval engine, we can thus reuse the built-in scoring function for this part of overall score. In addition to ordinary information retrieval rankings, we follow the "linked data" paradigm and prefer sources whose hostname corresponds to the resource's hostname. For example, we consider that more information on the resource `http:`

index type	index size
URI index	14Gb
Literal index	9.9Gb
IFP index	205Mb

Table I. Index size of Sindice.com for 26.6m RDF documents

`//eyaloren.org/foaf.rdf#me` can be found at the source `http://eyaloren.org` than at "outside" sources such as `http://dbpedia.org`.

5. EVALUATION

Our Sindice service has been running live for several months in which we have gathered some real-world statistics. We have also performed several controlled experiments to empirically establish the growth rate and scalability of our index structure. Table I shows the details of our index in December 2007. At that point, around 26.6 million RDF documents had been indexed; the URI index used around 14Gb of data, the literal index some 10Gb of data and the IFP index around 200Mb. The IFP index is smallest since IFPs are used much less frequently in documents than direct URIs.

We have experimentally evaluated index construction by incrementally indexing up to 6m RDF documents which were taken as uniform random samples from the combination of the Geonames, DBpedia, DBLP and UniProt datasets. Together, as shown in Figure 8, these documents contained around 104m RDF triples which consumed around 15Gb of RDF/XML. Also, we measured the number of (distinct) URIs and literal words mentioned in these documents, as shown on the left side of Figure 9.

We indexed each of these documents and measured, for every 5k indexed documents, the total amount of triples indexed, the total size in bytes of RDF data indexed and the total size of the resulting index, both for the URI and for the literal indices. The right side of Figure 9 shows the resulting size and growth rate of the URI and literal indices for these data collections. Due to the particulars of these datasets, the literal index uses more disk-space than the URI index (since DBLP and DBpedia contain large amounts of natural-language texts). The "dips" in the graph which occur approximately 500k documents are due to Solr's periodic disk-space optimisation. More importantly, one can clearly observe that that both

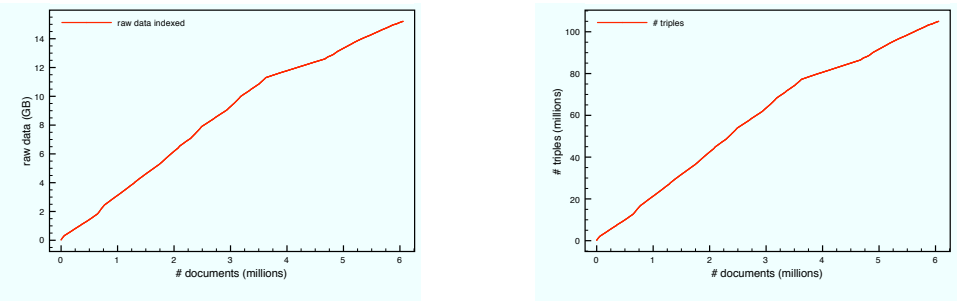

Fig. 8. Size (bytes of RDF/XML and triples) of combined samples

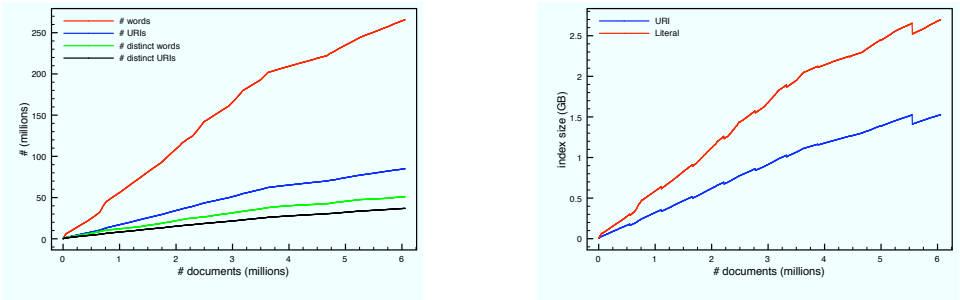

Fig. 9. Amount of URIs and literals (left) and resulting index size (right) in combined samples

indices grow completely linearly, as we had already predicted from the powerlaw behaviour observed in Section 3.3. This proves our important hypothesis that, because the distribution of URIs in Semantic Web documents is similar to that of words in natural-language documents, the same indexing technique may be used and will lead to similar scalibility results. These results allow us to confidently extrapolate existing results on document indexing to Semantic Web documents.

In the whole indexing pipeline (including fetching, parsing, reasoning and index updates) around 20% of CPU time is spent with crawling, parsing and splitting. The rest of the CPU time is mostly spent on reasoning, index updates are quick and mostly IO-bound. Network latency has not proven to be a big concern since the large majority of our data originates from downloading large data dumps instead of Web crawls (see Section 6.3). The main computational bottle-neck in our architecture is the reasoning step. Our reasoning implementation, including the reasoning cache, can currently process around 40–45 docs/s on seven parallel reasoning nodes, which is on average around 6 docs/s per node. In contrast, an earlier implementation of the cache could process less than 1 doc/s, whereas existing reasoners without cache require 1–2 minutes per document.

6. DATA PUBLICATION TOOLS

Sindice assists data producers by providing a set of tools and clear specifications to ensure that their data will be indexed correctly. The two main components are the proposed Semantic Sitemap standard, and the “linked data” validator.

6.1 Semantic Sitemaps

The semantic sitemap communicates precisely how and where Semantic Web data is published on an authoritative domain such as UniProt. The domain operator can publish a dump of all their RDF documents and use the sitemap to clearly communicate where the dump is and what it contains. Providing a single data dump has many benefits for both the data publisher and the data consumer (crawler). On one hand, the consumer will decrease crawling time to gathering the complete data collection. On the other hand, the publisher ensures that all his data is indexed while avoiding server overload by an “army” of crawlers.

In this section, we focus on the actual gathering of a given dataset. We show the benefits provided by the semantic sitemap by comparing the approach of data

Listing 1. Example dataset definition for DBpedia

```
<sc:dataset>
  <!-- Dataset description -->
  <sc:datasetLabel>DBpedia.org dataset</sc:datasetLabel>
  <changefreq>monthly</changefreq>

  <!-- Alternate access methods -->
  <sc:linkedDataPrefix>http://dbpedia.org/resource/</sc:linkedDataPrefix>
  <sc:sparqlEndpoint>http://dbpedia.org/sparql</sc:sparqlEndpoint>
  <sc:dataDump>http://dbpedia.org/downloads/articles.nt.gz</sc:dataDump>
  <sc:dataDump>http://dbpedia.org/downloads/infoboxes.nt.gz</sc:dataDump>
  <sc:dataDump>http://dbpedia.org/downloads/categories_skos.nt.gz</sc:dataDump>
  <!-- ... more dumps -->

  <!-- Some example URIs -->
  <sc:sampleURI>http://dbpedia.org/resource/Berlin</sc:sampleURI>
  <sc:sampleURI>http://dbpedia.org/resource/Pulp_Fiction_(film)</sc:sampleURI>
  <sc:sampleURI>http://dbpedia.org/resource/DBpedia</sc:sampleURI>
</sc:dataset>
```

crawling to the dump processing in an experimental scenario (i.e., how a can a semantic search engine index the full UniProt dataset). Then, we show how to efficiently process a data dump. The dump processing is of central importance, especially when it is very large in terms of file size and number of triples.

For a semantic search engine, gathering a large collection of RDF documents, such as the UniProt dataset, becomes a major problem: The issue in here is that a semantic web search engine cannot execute multiple crawlers concurrently on a single domain because otherwise it would possibly overload the server infrastructure of the data publisher (and in turn be banned). So to behave properly, a well-behaving crawler should wait at least one second between requests to the same server. Taking into account that the UniProt dataset contains 6.4m RDF documents in total, crawling its full content would take approximately 2.5 months using this maximum crawling speed. Such crawling duration is not acceptable for a semantic search engine. In the case of a data dump, the crawling time is reduced to the download time of the dump file(s).

6.2 Semantic Sitemap details

Semantic sitemaps are based on the notion of a *dataset*. A dataset, in this context, is a set of RDF statements that describe resources identified by URIs hosted on the domain where a sitemap file is published. A dataset can be accessed in several manners: as *linked data* (resolving resource URIs with a given URI prefix yields the resource description in RDF), as SPARQL *endpoints* (querying the dataset by sending SPARQL queries to a given service URI), or as *RDF dumps* (the entire dataset is available as a single RDF file or as a set of smaller files, potentially very large and usually compressed). A partial dataset definition for DBpedia[17] is shown in Listing 1. In this example, the dataset is available through each of the access methods discussed above.

When processing sites, Sindice checks the `robots.txt` file for a potential sitemap announcement. If a datadump is indeed available, the dump will be used instead instead of crawling the dataset. At the time of writing, a draft specification for

[17] `http://dbpedia.org/`

dataset	processing time	1 node	2 nodes	4 nodes
uniref.nt (272M triples)	parsing	1:17:01	1:19:30	1:19:10
	sorting	10:04:35	5:18:55	2:56:54
uniprot.nt (1,456M triples)	parsing	–	–	6:19:28
	sorting	–	–	16:36:26

Table II. UniProt processing with varying cluster size

Semantic Sitemaps is available for public review and comments. Initial feedback is positive, with early adoption by several major RDF data producers.

6.3 Semantic sitemap processor

To process large data dumps provided by the publisher through the semantic sitemap extension, we implemented a semantic sitemap processor that, like the rest of the Sindice architecture, uses the parallel Hadoop architecture to distribute data processing over many commodity nodes.

We evaluate our processing architecture using the large biological UniProt dataset. In our experimental setting, we used a "mini" cluster of four AMD Opteron 2.2Ghz machines with 4Gb of RAM each. We first compared a smaller subset of UniProt (uniref.nt, 272M triples) on different cluster size and then processed the full dataset (uniprot.nt, 1,456M triples) using the complete cluster as it could not be practically processed on a single machine. For technical reasons, the parsing step was always performed on a single machine, the splitting and sorting distributed over one, two, and four machines. Preliminary results are shown in Table II. As can be seen, parallel processing of large dumps is possible and decreases processing time almost linearly. We are also working on distributed parsing to further minimise the overall processing time.

6.4 Linked data validator

Sindice's linked data validator enables users and data providers to validate the linked data of a given RDF document. The Sindice validator differs from existing RDF validators, such as the W3C's RDF validator[18], which only validate the document's syntax. In addition, our validator ensures (recursively) that (i) all URIs of properties and classes link to a valid RDF document that can be found when dereferencing the resource URI, (ii) that these schema concepts are actually defined in that dereferenced RDF document, and (iii) that the document has a human-readable description. Checking such a broader notion of linked data "validity" is crucial since badly linked data can lead to incomplete reasoning and thus to an incomplete set of identifiers or human description for a given RDF document.

Common errors encountered include URIs that point to non-existent locations (Wordnet) or to documents that do not contain the any RDF data. For example, in our experience, most currently published FOAF files contain such "linked data errors" because several RSS, XML Schema and Wordnet concepts from the FOAF schema definition are not available at their URI location (404 error).

[18] `http://www.w3.org/RDF/Validator/`

	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 III. Approaches in (Semantic) Web information retrieval

7. 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[19] 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.

Most of the differences between these engines and Sindice have been highlighted during the discussion of this paper. To the best of our knowledge, however, 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, no full reasoning is performed to extract IFPs but instead extracts only several hard-coded properties as IFPs. 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 Semantic Sitemap.

Table III 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 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 input to our crawling pool.

8. CONCLUSION

We have presented Sindice, an indexing infrastructure with a Web front-end and a public API to locate Semantic Web data sources such as RDF files and SPARQL endpoints. The Sindice service has been deployed on `http://sindice.com`, is ac-

[19] `http://swse.deri.org/`

tively indexing Semantic Web data and has attracted much community interest. Sindice supports application developers in discovering relevant data sources to integrate in their application. Lookups for data sources can be performed directly using resource URIs, indirectly through uniquely-identifying inverse functional properties, and using full-text search over the literals.

To achieve scalability, the Sindice architecture is based on Hadoop, an existing parallel processing infrastructure, allowing us to efficiently distribute index construction over an arbitrary number of nodes. The inverted index itself has been implemented twice, once directly using on-disk persistent hashtables and once using Solr, an off-the-shelf information retrieval index.

We have shown experimentally that the ratio of URIs/URLs (the term distribution of Semantic Web identifiers over documents) follows a power law and thus exhibits scale invariance, allowing us to confidently estimate the required data storage as a function of the number of indexed documents. Also, since this distribution is crucially similar to term frequency distribution in natural language, this finding allows us to confidently predict that existing information retrieval techniques and optimisations can be used for indexing Semantic Web identifiers.

Areas for future investigation include: optimising data processing by adjusting our indexing algorithms to better utilise locality of data in the parallel architecture [Dean and Ghemawat 2004; Lämmel 2007]; improving the crawling scheduler with existing algorithms for source scheduling [Aggarwal et al. 2001; Pant et al. 2004] and duplicate detection [Bar-Yossef et al. 2007; Broder et al. 1997]; and continuing investigation into ranking techniques that combine information-retrieval and graph-based approaches [Hogan et al. 2006; Kleinberg 1999].

Possibly the most innovative aspect of Sindice is its philosophy: an interweaving framework for the Semantic Web that achieves 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, since it empowers users and their clients with the freedom to choose their sources according to their preferences and requirements.

REFERENCES

Aggarwal, C. C., Al-Garawi, F., and Yu, P. S. 2001. Intelligent crawling on the World Wide Web with arbitrary predicates. In *Proceedings of the International World-Wide Web Conference*. 96–105.

Bar-Yossef, Z., Keidar, I., and Schonfeld, U. 2007. Do not crawl in the DUST: Different URLs with similar text. In *Proceedings of the International World-Wide Web Conference*. 111–120.

Belleau, F., Nolin, M.-A., Tourigny, N., Rigault, P., and Morissette, J. 2007. 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*.

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

Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., and Sheets, D. 2006. Tabulator: Exploring and analyzing linked data on the Semantic Web. In *Proceedings of the ISWC Workshop on Semantic Web User Interaction*.

Brin, S. and Page, L. 1998. Anatomy of a large-scale hypertextual web search engine. In *Proceedings of the International World-Wide Web Conference*.

Broder, A. 2002. A taxonomy of web search. *SIGIR Forum 36,* 2, 3–10.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. 1997. Syntactic clustering of the web. *Computer Networks 29,* 8-13, 1157–1166.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. 2006. Bigtable: a distributed storage system for structured data. In *Proceedings of the Symposium on Operating Systems Design & Implementation*. 15–15.

Consens, M. P. and Mendelzon, A. O. 1990. Graphlog: A visual formalism for real life recursion. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*. 404–416.

Dean, J. and Ghemawat, S. 2004. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the Symposium on Operating Systems Design & Implementation*. 137–147.

Depoorter, B. and Parisi, F. 2002. Fair use and copyright protection: a price theory explanation. *International Review of Law and Economics 21,* 4, 453–473.

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

Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. 2004. Swoogle: A search and metadata engine for the Semantic Web. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*. 652–659.

Erickson, J. S. 2003. Fair use, DRM, and trusted computing. *Communications of the ACM 46,* 4, 34–39.

Finin, T. and Ding, L. 2006. Search engines for semantic web knowledge. In *XTech*.

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

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

Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. The Google file system. 29–43.

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

Hayes, P., Ed. 2004. *RDF Semantics*. W3C Recommendation.

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

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

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

Kiryakov, A., Ognyanov, D., and Manov, D. 2005. OWLIM – a pragmatic semantic repository for OWL. In *Proceedings of the Conference on Web Information Systems Engineering (WISE) Workshops*. 182–192.

Klein, B., Lerner, A., and Murphy, K. 2002. The economics of copyright "fair use" in a networked world. *The American Economic Review 92,* 2, 205–208.

Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 604–632.

Krogh, C. 1996. The rights of agents. In *Intelligent agents II, Agent theories, architectures and languages*, M. Wooldridge, J. P. Muller, and M. Tambe, Eds. Springer-Verlag, 1–16.

Lämmel, R. 2007. Google's MapReduce programming model – revisited. *Sci. Comput. Program.*. Accepted for publication.

Landes, W. M. and Posner, R. A. 2003. *The Economic Structure of Intellectual Property Law*. Harvard University Press.

Li, W. 1992. Random texts exhibit Zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory 38,* 6, 1842–1845.

Lin, D. and Loui, M. C. 1998. Taking the byte out of cookies: Privacy, consent, and the web. *28,* 2, 39–51.

Manning, C., , Raghavan, P., and Schütze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press. To be published.

McBryan, O. A. 1994. GENVL and WWWW: Tools for taming the Web. In *Proceedings of the International World-Wide Web Conference*.

McGuinness, D. L. and van Harmelen, F., Eds. 2004. *OWL Web Ontology Language*. W3C Recommendation.

Miles, A., Baker, T., and Swick, R., Eds. 2006. *Best Practice Recipes for Publishing RDF Vocabularies*. W3C Working Draft.

Pant, G., Srinivasan, P., and Menczer, F. 2004. Crawling the web. In *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*. Springer-Verlag.

Salton, G. and McGill, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill.

Sauermann, L., Cyganiak, R., and Völkel, M. 2007. Cool URIs for the Semantic Web. Tech. Rep. TM-07-01, DFKI. 2.

Tavani, H. T. 1999. Informational privacy, data mining, and the internet. *Ethics and Information Technology 1,* 2, 137–145.

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

Thelwall, M. and Stuart, D. 2006. Web crawling ethics revisited: Cost, privacy, and denial of service. *57,* 13, 1771–1779.

Tummarello, G., Morbidoni, C., and Nucci, M. 2006. Enabling Semantic Web communities with DBin: An overview. In *Proceedings of the International Semantic Web Conference (ISWC)*. 943–950.

Ullman, J. D. 1989. *Principles of Database and Knowledge Base Systems*. Vol. 2. Computer Science Press.

van Asbroeck, B. and Cock, M. 2007. Belgian newspapers v Google News: 2-0. *Journal of Intellectual Property Law & Practice 2,* 7, 463–466.

van Wel, L. and Royakkers, L. 2004. Ethical issues in web data mining. *Ethics and Information Technology 6,* 2, 129–140.

Zipf, G. K. 1932. *Selective Studies and the Principle of Relative Frequency in Language*. MIT Press.