Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine

Aidan Hogan [a], Andreas Harth [b], Jürgen Umbrich [a], Sheila Kinsella [a], Axel Polleres [a], Stefan Decker [a]

[a] *Digital Enterprise Research Institute, National University of Ireland, Galway*
[b] *AIFB, Karlsruhe Institute of Technology, Germany*

Abstract

In this paper, we discuss the architecture and implementation of the Semantic Web Search Engine (SWSE). Following traditional search engine architecture, SWSE consists of crawling, data enhancing, indexing and a user interface for search, browsing and retrieval of information; unlike traditional search engines, SWSE operates over RDF Web data – loosely also known as Linked Data – which implies unique challenges for the system design, architecture, algorithms, implementation and user interface. In particular, many challenges exist in adopting Semantic Web technologies for Web data: the unique challenges of the Web – in terms of scale, unreliability, inconsistency and noise – are largely overlooked by the current Semantic Web standards. Herein, we describe the current SWSE system, initially detailing the architecture and later elaborating upon the function, design, implementation and performance of each individual component. In so doing, we also give an insight into how current Semantic Web standards can be tailored, in a best-effort manner, for use on Web data. Throughout, we offer evaluation and complementary argumentation to support our design choices, and also offer discussion on future directions and open research questions. Later, we also provide candid discussion relating to the difficulties currently faced in bringing such a search engine into the mainstream, and lessons learnt from roughly six years working on the Semantic Web Search Engine project.

Key words: Web search, semantic search, RDF, Semantic Web, Linked Data

1. Introduction

Offering a minimalistic and uncluttered user interface, a simple keyword-based user-interaction model, fast response times, and astute prioritisation of results, Google [18] has become the yard-stick for Web-search, servicing approximately 64.6% of traditional Web search queries [2] over billions of Web documents. Arguably, Google reaches the *imminent* limit of providing the best possible search over the largely HTML data it indexes. However, from the user perspective, the core Google engine (here serving as the archetype for traditional HTML search engines, such as Yahoo, MSN/Bing, AOL, Ask, etc.) is far from the consummate Web search solution:

[2] Statistics taken from Nielsen MegaView Search for ∼11 g searches recorded in Aug. 2009: cf. `http://searchenginewatch.com/3634991`

Google does not *typically* produce direct answers to queries, but instead typically recommends a selection of related documents from the Web. We note that in more recent years, Google has begun to provide direct answers to prose queries matching certain common templates – for example, "population of china" or "12 euro in dollars" – but again, such functionality is limited to a small subset of popular user queries. Furthermore, Google now provides individual and focussed search interfaces over images, videos, locations, news articles, books, research papers, blogs, and realtime social media – although these tools are inarguably powerful, they are limited to their respective domain.

In the general case, Google is not suitable for complex information gathering tasks requiring aggregation from multiple indexed documents: for such tasks, users must manually aggregate tidbits of pertinent information from various recommended *heterogeneous* sites, each such site presenting information in its own formatting and using its own navigation system. In effect, Google's limitations are predicated on the lack of structure in HTML documents, whose machine interpretability is limited to the use of generic markup-tags mainly concerned with document rendering and linking. Although Google arguably makes the best of the limited structure available in such documents, most of the real content is contained in prose text which is inherently difficult for machines to interpret. Addressing this inherent problem with HTML Web data, the Semantic Web movement provides a stack of technologies for publishing machine-readable data on the Web, the core of the stack being the Resource Description Framework (RDF).

Using URIs to name things – and not just documents – RDF offers a standardised and flexible framework for publishing structured data on the Web (i) such that data can be linked, incorporated, extended and re-used by other RDF data across the Web, (ii) such that heterogeneous data from independent sources can be automatically integrated by software-agents, and (iii) such that the meaning of data can be well-defined using lightweight ontologies described in RDF using the RDF Schema (RDFS) and Web Ontology Langauge (OWL) standards.

Thanks largely to the "Linked Open Data" project [14] – which has emphasised more pragmatic aspects of Semantic Web publishing – a rich lode of open RDF data now resides on the Web: this "Web of Data" includes content exported from, for example: Wikipedia, the BBC, the New York Times, Flickr, Last.fm, scientific publishing indexes, biomedical information and governmental agencies. This precedent raises an obvious question: assuming large-scale adoption of high-quality RDF publishing on the Web, could a search engine indexing RDF feasibly improve upon current HTML-centric engines? Theoretically at least, such a search engine could offer advanced querying and browsing of structured data with search results automatically aggregated from multiple documents and rendered directly in a clean and consistent user-interface, thus reducing the manual effort required of its users. Indeed, there has been much research devoted to this topic, with various incarnations of (mostly academic) RDF-centric Web search engines emerging – e.g., Swoogle, Falcons, Watson, Sindice – and in this paper, we present the culmination of over six years research on another such engine: the "Semantic Web Search Engine" (SWSE) [3].

Indeed, the realisation of SWSE has implied two major research challenges:

(i) the system must **scale** to large amounts of data; and

(ii) the system must be **robust** in the face of heterogeneous, noisy, impudent, and possibly conflicting data collected from a large number of sources.

Semantic Web standards and methodologies are not naturally applicable in such an environment; in presenting the design and implementation of SWSE, we show how standard Semantic Web approaches can be tailored to meet these two challenging requirements, often taking cues from traditional information retrieval techniques.

As such, we present the core of a system which we demonstrate to provide scale, and which is distributed over a cluster of commodity hardware. Throughout, we focus on the unique challenges of applying standard Semantic Web techniques and methodologies, and show why the consideration of the source of data is an integral part of creating a system which must be tolerant to Web data – in particular, we show how Linked Data principles can be exploited for such purposes. Also, there are many research questions still very much open with respect to the direction of the overall system, as well as improvements to be made in the individual components; we discuss these as they arise, rendering a road-map of past, present and possible future research in the area of Web search over RDF data.

---

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

More specifically, in this paper we:

– present the architecture and modus-operandi of our system for offering search and browsing over RDF Web data (Section 2);
– present high-level related work in RDF search engines (Section 3);
– present core preliminaries required throughout the rest of the paper (Section 4);
– detail the design, distributed implementation and evaluation of the offline index building components, including *crawling* (Section 5), *consolidation* (Section 6), *ranking* (Section 7), *reasoning* (Section 8), and *indexing* (Section 9);
– summarise the runtimes for each of the offline tasks (Section 10);
– detail the design, distributed implementation and evaluation of the runtime components, including our (lightweight) query-processor (Section 11) and user-interface (Section 12);
– conclude with discussion of future directions, open research challenges and current limitations of Web search over RDF data (Sections 13–14).

2. System Overview

In this section, we outline the functionality of the SWSE system. We begin with an overview of the functionality offered to end users (Section 2.1). Thereafter we present the high-level SWSE architecture (Section 2.2) and describe the distributed framework upon which our components operate (Section 2.3). Wrapping up this section, we detail the software and hardware environments used for the experiments detailed herein (Section 2.4).

2.1. *Application Overview*

To put later discussion into context, we now give a brief overview of the lightweight functionality of the SWSE system; please note that although our methods and algorithms are tailored for the specific needs of SWSE, many aspects of their implementation, design and evaluation apply to more general scenarios.

Unlike prevalent document-centric Web search engines, SWSE operates over structured data and holds an entity-centric perspective on search: in contrast to returning links to documents containing specified keywords [18], SWSE returns data representations of real-world entities. While current search engines such as Google, Bing and Yahoo re-turn search results in different domain-specific categories (`Web`, `Images`, `Videos`, `Shopping`, etc.), data on the Semantic Web is flexibly typed and does not need to follow pre-defined categories. Returned objects can represent people, companies, cities, proteins – anything people care to publish data about.

In a manner familiar from traditional Web search engines, SWSE allows users to specify keyword queries in an input box and responds with a ranked list of result snippets; however, the results refer to entities not documents. A user can then click on an entity snippet to derive a detailed description thereof. The descriptions of entities are automatically aggregated from arbitrarily many sources, and users can cross-check the source of particular statements presented; descriptions also include inferred data – data which has not necessarily been published, but has been derived from the existing data through reasoning. Users can subsequently navigate to related entities, as such, browsing the Web of Data.

Along these lines, Figure 1 shows a screenshot containing a list of entities returned as a result to the keyword search "`bill clinton`" – such results pages are familiar from HTML-centric engines, with the addition of result types (e.g., `DisbarredAmericanLawyers`, `AmericanVegitarians`, etc.). Results are aggregated from multiple sources. Figure 2 shows a screenshot of the focus (detailed) view of the `Bill Clinton` entity, with data aggregated from 54 documents spanning six domains (`bbc.co.uk`, `dbpedia.org`, `freebase.com`, `nytimes.com`, `rdfize.com` and `soton.ac.uk`), as well as novel data found through reasoning.

2.2. *System Architecture*

The high-level system architecture of SWSE loosely follows that of traditional HTML search engines [18]. Figure 3 details the pre-runtime architecture of our system, showing the components involved in achieving a local index of RDF Web data amenable for search. Like traditional search engines, SWSE contains components for crawling, ranking and indexing data; however, there are also components specifically designed for handling RDF data, namely the consolidation component and the reasoning component. The high-level index building process is as follows:
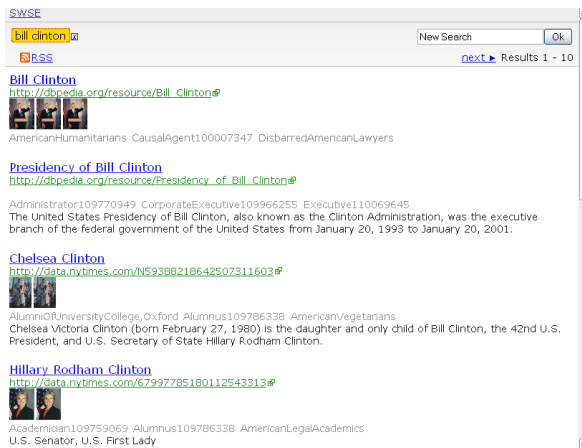– the crawler accepts a set of seed URIs and re-

Fig. 1. Results view for keyword query `Bill Clinton`

Fig. 2. Focus view for entity `Bill Clinton`

trieves a large set of RDF data from the Web;
– the consolidation component tries to find synonymous (i.e., equivalent) identifiers in the data, and canonicalises the data according to the equivalences found;
– the ranking component performs links-based analysis over the crawled data and derives scores indicating the importance of individual elements in the data (the ranking component also considers URI redirections encountered by the crawler when performing the links-based analysis);
– the reasoning component materialises new data which is implied by the inherent semantics of the input data (the reasoning component also requires URI redirection information to evaluate the trustworthiness of sources of data);
– the indexing component prepares an index which supports the information retrieval tasks required by the user interface.

Subsequently, the query-processing and user-interface components service queries over the index built in the previous steps.

Our methods follow the standards relating to RDF [92], RDFS [65] and OWL [115], and leverage the Linked Data principles [9] which state how RDF should be published on the Web. As such, our methods should be 100% precise (aka. *sound*) with respect to data *correctly* published according to these documents, but we note that oftentimes, the noise inherent in heterogenous RDF Web data may create unintended results. We characterise these problems as they occur, although we know of no method for accurately determining the amount of incorrect or unwanted results generated by these tasks – we note that such considerations may also be subjective (for example, see [54]).
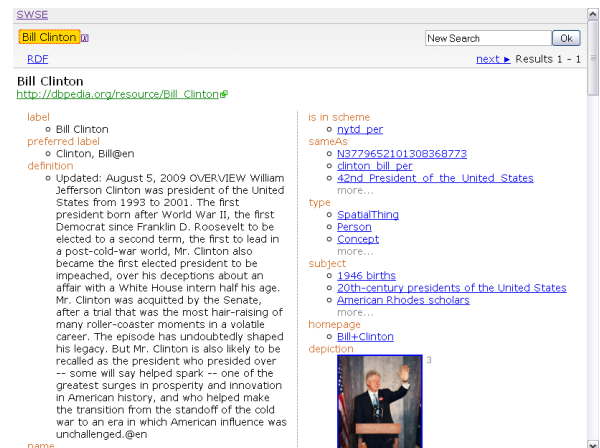
We will detail the design and operation of each of the components in the following sections, but beforehand, we present the distribution framework upon which all of our components are implemented.

2.3. Distribution Abstraction

In order to scale, we deploy each of our components over a distributed framework which we now briefly describe; Figure 4 illustrates the distributed operations possible in our framework. The framework is based on a shared nothing architecture [116] and consists of one master machine which orchestrates the given tasks, and several slave machines which perform parts of the task in parallel.

The master machine can instigate the following distributed operations:
– **scatter:** partition a file into chunks given some local *split* function, and send the chunks to individual machines – usually only used to initialise a task and seed the slave machines with an initial set of data for processing;
– **run:** request the parallel execution of a task by the slave machines – such a task either involves processing of some local data (embarrassingly parallel execution), or execution of the **co-ordinate** method by the slave swarm;
– **gather:** gathers chunks of output data from the slave swarm and performs some local *merge* function over the data – this is usually performed to create a single output file for a task, or more usually to gather global knowledge required by all slave machines for a future task;
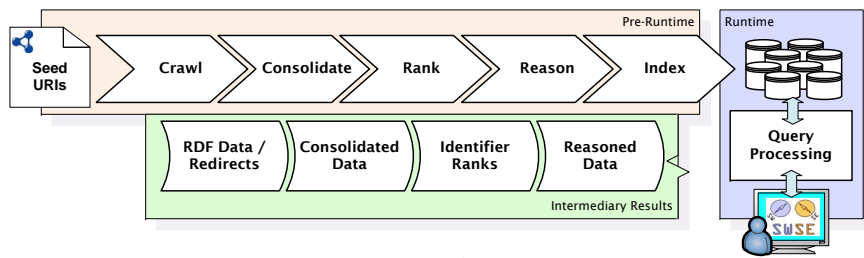
Fig. 3. System Architecture

– **flood:** broadcast global knowledge required by all slave machines for a future task.

The master machine is intended to disseminate input data to the slave swarm, to provide the control logic required by the distributed task (commencing tasks, co-ordinating timing, ending tasks), to gather and locally perform tasks on global knowledge which the slave machines would otherwise have to replicate in parallel, and to transmit globally required knowledge. The master machine can also be used to compute the final result for a given distributed task; however, the end goal of our distributed framework is to produce a distributed index over the slave machines, thus this task is never required in our system.

The slave machines, as well as performing tasks in parallel, can perform the following distributed operation (on the behest of the master machine):

– **co-ordinate:** local data on each machine is partitioned according to some *split* function, with the chunks sent to individual machines in parallel; each machine also gathers the incoming chunks in parallel using some *merge* function.

The above operation allows slave machines to partition and disseminate intermediary data directly to other slave machines; the **co-ordinate** operation could be replaced by a pair of **gather**/**scatter** operations performed by the master machine, but we wish to avoid the channelling of all such intermediary data through one machine.

We note that our framework resembles the MapReduce framework [31], with **scatter** loosely corresponding to the Map operation, and **gather** loosely corresponding to the Reduce operation; similarly, the *split* function corresponds loosely to the Partition function, and the **co-ordinate** function loosely corresponds to the Shuffle operation in the MapReduce setting.

2.4. *Software Environment/Hardware*

We instantiate this architecture using the standard Java Remote Method Invocation libraries as a convenient means of development given our Java code-base.

All of our evaluation is based on nine machines connected by Gigabit ethernet [4], each with uniform specifications; viz.: 2.2GHz Opteron x86-64, 4GB main memory, 160GB SATA hard-disks, running Java 1.6.0_12 on Debian 5.0.4. Please note that much of the evaluation presented in this paper assumes that the slave machines have roughly equal specifications in order to ensure that tasks finish in roughly the same time, assuming even data distribution.

We currently do not consider more advanced topics in our architecture – such as load-balancing (with the exception of evenly distributing data), replication, uptime and counteracting hardware failure – and discussion of these fall outside of the current scope.

3. Related Work

In this section, we give an overview of related work, firstly detailing distributed architectures for Web search (Section 3.1), then discussing related systems in the field of "Hidden Web" and "Deep Web" (Section 3.2), and finally describing current systems that offer search and browsing over RDF Web data (Section 3.3) – for a further survey of the latter, cf. [127]. *Please note that we will give further detailed related work in the context of each component throughout the paper.*

3.1. *Distributed Web Search Architectures*

Distributed architectures have long been common in traditional information-retrieval based Web search engines, incorporating distributed crawling, ranking, indexing and query-processing components. Although all mainstream search engines are based on distributed architectures, details are not

---

[4] We observe, e.g., a max FTP transfer rate of 38MB/sec between machines.
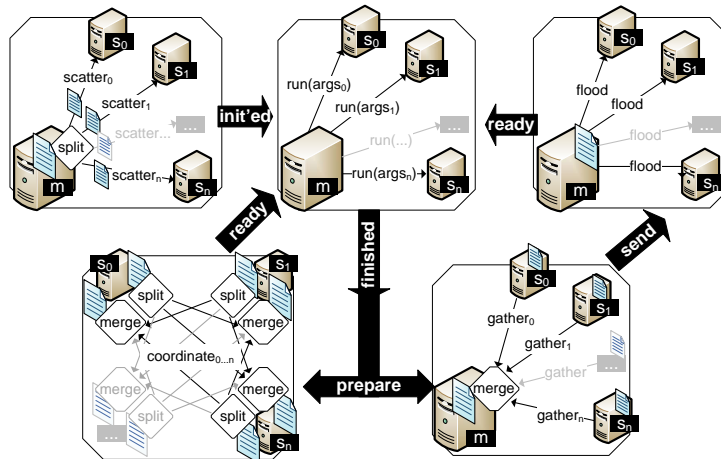
Fig. 4. Distribution Methods Architecture

commonly published. Again, one of the most well-known search engine architectures is that previously described for the Google search engine [18]. More recent publications relating to the Google architecture relate to the MapReduce framework previously alluded to [31], and to the underlying BigTable [23] distributed database system.

Similar system architectures have been defined in the literature, including WebBase [70] which includes an incremental crawler, storage manager, indexer and query processor; in particular, the authors focus on hash- and log-based partitioning for storing incrementally-updated vast repositories of Web documents. The authors of [94] also describe a system for building a distributed inverted-index over a large corpus of Web pages, for subsequent analysis and query-processing: they employ an embedded distributed database system.

Much of the work presented herein is loosely inspired by such approaches, and thus constitutes an adaptation of such works for the purposes of search over structured data. Since we consider replication, fault tolerance, incremental indexing, etc., currently out of scope, many of our techniques are more lightweight than those discussed.

3.2. *Hidden Web/Deep Web Approaches*

So called "Hidden Web" or "Deep Web" approaches [21] are predicated on the premise that a vast amount of the information available on the Web is veiled behind sites with heavy dynamic content, usually backed by relational databases. Such information is largely impervious to traditional crawl-ing techniques since content is usually generated by means of bespoke flexible queries; thus, traditional search engines can only skim the surface of such information [66]. In fact, such data-rich sources have lead to early speculative work on entity-centric search [28].

Approaches to exploit such sources heavily rely on manually constructed, site-specific wrappers to extract structured data from HTML pages [21], or to communicate directly with the underlying database of such sites [24]. Some works have also looked into automatically crawling such hidden-Web sources, by interacting with forms found during traditional crawls [112]; however, this approach is "task-specific" and not appropriate for general crawling.

The Semantic Web may represent a future direction for bringing deep-Web information to the surface, leveraging RDF as a common and flexible data-model for exporting the content of such databases, leveraging RDFS and OWL as a means of describing the respective schemata, and thus allowing for automatic integration of such data by Web search engines. Efforts such as D2R(Q) [13] seem a natural fit for enabling RDF exports of such online databases.

3.3. *RDF-centric Search Engines*

Early prototypes using the concepts of ontologies and semantics on the Web include Ontobroker [32] and SHOE [67], which can be seen as predecessors to standardisation efforts such as RDFS and OWL, describing how data on the Web can be given in structured form, and subsequently crawled, stored,

6

inferenced and queried over.

Swoogle[5] offers search over RDF documents by means of an inverted keyword index and a relational database [38]. Swoogle calculates metrics that allow ontology designers to check the popularity of certain properties and classes. In contrast to SWSE, which is mainly concerned with entity search over instance data, Swoogle is mainly concerned with more traditional document-search over ontologies.

WATSON[6] provides a similar effort to provide keyword search facilities over Semantic Web documents, but additionally provides search over entities [114,29]. However, they do not include components for consolidation or reasoning, and seemingly instead focus on providing APIs to external services.

Sindice[7] is a registry and lookup service for RDF files based on Lucene and a MapReduce framework [103]. Sindice originally focussed on providing an API for finding documents which reference a given RDF entity or given keywords – again, document-centric search. More recently however, Sindice has begun to offer entity search in the form of Sig.Ma[8] [120]. However, Sig.ma maintains a one-to-one relationship between keyword search and results, representing a very different user-interaction model to that presented herein.

The Falcons Search engine[9] offers entity-centric searching for entities (and concepts) over RDF data [27]. They map certain keyword phrases to query relations between entities, and also use class hierarchies to quickly restrict initial results. Conceptually, this search engine most closely resembles our approach. However, there are significant differences in how the individual components of SWSE and Falcons are designed and implemented. For example, like us, they also rank entities, but using a logarithm of the count of documents in which they are mentioned – we employ a links-based analysis of sources. Also, Falcons supports reasoning involving class hierarchies, whereas we apply a more general rule based approach, applying a scalable subset of OWL 2 RL/RDF rules. Such differences will be discussed further throughout this paper, and in the context of the individual components.

Aside from the aforementioned domain-agnostic search systems, we note that other systems focus on exploiting RDF for the purposes of domain-specific querying; for example, the recent GoWeb system[10] demonstrates the benefit of searching structured data for the biomedical domain [36]. However, in catering for a specific domain, such systems do not target the same challenges and use-cases as we do.

4. Preliminaries

Before we continue, we briefly introduce some standard core notation used throughout the paper – relating to RDF terms (constants), triples and quadruples – and also discuss Linked Data principles. Note that in this paper, we will generally use bold-face to refer to infinite sets: e.g., $\mathbf{G}$ refers to the set of all triples; we will use calligraphic font to denote a subset thereof: e.g., $\mathcal{G}$ is a particular set of triples, where $\mathcal{G} \subset \mathbf{G}$.

4.1. *Resource Description Framework*

The Resource Description Framework provides a structured means of publishing information describing entities through use of RDF terms and RDF triples, and constitutes the core data model for our search engine. In particular, RDF allows for optionally defining names for entities using URIs and allows for subsequent re-use of URIs across the Web; using triples, RDF allows to group entities into named classes, allows to define named relations between entities, and allows for defining named attributes of entities using string (literal) values. We now briefly give some necessary notation.

RDF Constant Given a set of URI references $\mathbf{U}$, a set of blank nodes $\mathbf{B}$, and a set of literals $\mathbf{L}$, the set of *RDF constants* is denoted by $\mathbf{C} = \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$. The set of blank nodes $\mathbf{B}$ is a set of existensially quantified variables. The set of literals is given as $\mathbf{L} = \mathbf{L}_p \cup \mathbf{L}_d$, where $\mathbf{L}_p$ is the set of *plain literals* and $\mathbf{L}_d$ is the set of *typed literals*. A typed literal is the pair $l = (s, d)$, where s is the lexical form of the literal and $d \in \mathcal{U}$ is a datatype URI. The sets $\mathbf{U}$, $\mathbf{B}$, $\mathbf{L}_p$ and $\mathbf{L}_t$ are pairwise disjoint.

Please note that in this paper, we treat blank nodes as their skolem versions: i.e., not as existential variables, but as denoting their own syntactic form. We also ensure correct merging of RDF graphs [65] by using blank-node labels unique for a given source.

For URIs, we use namespace prefixes in this paper as common in the literature – the full URIs can be retrieved from the convenient `http://prefix.cc` service. For space reasons, we sometimes denote `owl:` as the default namespace.

RDF Triple   A triple $t = (s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ is called an *RDF triple*. In a triple (s, p, o), s is called subject, p predicate, and o object.

RDF Graph   We call a finite set of triples an *RDF graph* $\mathcal{G} \subset \mathbf{G}$ where $\mathbf{G} = (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$.

RDF Entity   We refer to the referent of a URI or blank-node as an *RDF entity*, or commonly just entity.

4.2. *Linked Data, Data Sources, Quadruples, and Dereferencing*

In order to cope with the unique challenges of handling diverse and unverified Web data, many of our components and algorithms require inclusion of a notion of provenance: consideration of the source of RDF data found on the Web. Tightly related to such notions are the Linked Data Best Practices (here paraphrasing [9]):

LDP1   use URIs to name things;

LDP2   use HTTP URIs so that those names can be looked up;

LDP3   provide useful structured information when a look-up on a URI is made – loosely, called *dereferencing*;

LDP4   include links using external URIs.

In particular, within SWSE, these best-practices form the backbone of various algorithms designed to interact and be tolerant to Web data.

We must thus extend RDF triples with *context* to denote the source thereof [53,58]. We also define some relations between the identifier for a data source, and the graph it contains, including a function to represent HTTP redirects prevalently used in Linked Data for **LDP3** [9].

Data Source   We define the *http-download* function $\mathsf{get} : \mathbf{U} \to 2^{\mathbf{G}}$ as the mapping from a URI to an RDF graph it may provide by means of a given HTTP lookup [47] which directly returns status code `200 OK` and data in a suitable RDF format. [11] We define

the set of *data sources* $\mathbf{S} \subset \mathbf{U}$ as the set of URIs $\mathbf{S} = \{s \in \mathbf{U} \mid \mathsf{get}(s) \neq \emptyset\}$. We define the *reference function* $\mathsf{refs} : \mathbf{C} \to 2^{\mathbf{S}}$ as the mapping from an RDF term to the set of data sources that mention it.

RDF Triple in Context/RDF Quadruple   A pair (t, c) with a triple $t = (s, p, o)$, $c \in \mathbf{S}$ and $t \in \mathsf{get}(c)$ is called a *triple in context c*. We may also refer to (s, p, o, c) as an *RDF quadruple* or quad q with context c.

HTTP Dereferencing   We define *dereferencing* as the function $\mathsf{deref} : \mathbf{U} \to \mathbf{U}$ which maps a given URI to the identifier of the document returned by HTTP lookup operations upon that URI following redirects (for a given finite and non-cyclical path) [47], or which maps a URI to itself in the case of failure. Note that we do not distinguish between the different `30x` redirection schemes, and that this function would involve, e.g., stripping the fragment identifier of a URI [11]. Note that all HTTP level functions $\{\mathsf{get}, \mathsf{refs}, \mathsf{deref}\}$ are set at the time of the crawl, and are bounded by the knowledge of our crawl: for example, refs will only consider documents accessed by the crawl.

5. Crawling

We now begin the discussion of the first component required for building the index, and thus for retrieving the raw RDF documents from the Web: that is, the *crawler*. Our crawler starts with a set of seed URIs, retrieves the content of URIs, parses and writes content to disk in the form of quads, and recursively extracts new URIs for crawling. We leverage Linked Data principles (see Section 4.2) to discover new sources, where following **LDP2** and **LDP3**, we consider all `http:` protocol URIs extracted from an RDF document as candidates for crawling.

Like traditional HTML crawlers, we identify the following requirements for crawling:

– **Politeness:** The crawler must implement politeness restrictions to avoid hammering remote servers with dense HTTP GET requests and to abide by policies identified in the provided `robots.txt` files [12].

– **Throughput:** The crawler should crawl as many URIs as possible in as little time as is possible within the bounds of the politeness policies.

---

[11] $2^{\mathbf{G}}$ refers to the powerset of $\mathbf{S}$.

[12] `http://www.robotstxt.org/orig.html`

– **Scale:** The crawler should employ scalable techniques, and on-disk indexing as required.
– **Quality:** The crawler should prioritise crawling URIs it considers to be "high quality".

Thus, the design of our crawler is inspired by related work from traditional HTML crawlers. Additionally – and specific to crawling structured data – we identify the following requirement:

– **Structured Data:** The crawler should retrieve a high percentage of RDF/XML documents and avoid wasted lookups on unwanted formats: e.g., HTML documents.

Currently, we crawl for RDF/XML syntax documents – RDF/XML is still the most commonly used syntax for publishing RDF on the Web, and we plan in future to extend the crawler to support other formats such as RDFa, N-Triples and Turtle.

The following algorithm details the operation of the crawler, and will be explained in detail throughout this section.

5.1. *High-level Approach*

Our high-level approach is to perform breath-first crawling, following precedent set by traditional Web crawlers (cf. [15] [69]): the crawl is conducted in rounds, with each round crawling a *frontier*. On a high-level, Algorithm 1 represents this round-based approach applying ROUNDS number of rounds. The frontier comprises of seed URIs for round 0 (Algorithm 1, Line 1), and thereafter with novel URIs extracted from documents crawled in the previous round (Algorithm 1, Line 21). Thus, the crawl emulates a breadth-first traversal of inter-linked Web documents. (Note that the algorithm is further tailored according to requirements we will describe as the section progresses.)

As we will see later in the section, the round-based approach fits well with our distributed framework, allowing for crawlers to work independently for each round, and co-ordinating new frontier URIs at the end of each round. Additionally, [99] show that a breadth-first traversal strategy tends to discover high-quality pages early on in the crawl, with the justification that well-linked documents (representing higher quality documents) are more likely to be encountered in earlier breadth-first rounds; similarly, breadth first crawling leads to a more diverse dataset earlier on, rather than a depth-first approach which may end up traversing deep paths within a given site. In [88], the authors justify a rounds-based ap-

---

Algorithm 1 Algorithm for crawling

Require: SEEDS, ROUNDS, PLD-LIMIT, MIN-DELAY
1: $frontier \leftarrow$ SEEDS
2: $pld_{0...n} \leftarrow$ new $queue$
3: $stats \leftarrow$ new $stats$
4: **while** $rounds + 1 <$ ROUNDS **do**
5:    put $frontier$ into $pld_{0...n}$
6:    **while** $depth + 1 <$ PLD-LIMIT **do**
7:       **for** $i = 0$ to n **do**
8:          prioritise(pld_i, $stats$)
9:       **end for**
10:      $start \leftarrow$ current_time()
11:      **for** $i = 0$ to n **do**
12:         $cur_i =$ calculate_cur(pld_i, $stats$)
13:         **if** $cur_i >$ random([0,1]) **then**
14:            get uri from pld_i
15:            $uri_{\mathsf{deref}} = \mathsf{deref}(uri)$
16:            **if** $uri_{\mathsf{deref}} = uri$ **then**
17:               $\mathcal{G} = \mathsf{get}(uri)$
18:               output $\mathcal{G}$
19:               $U_{\mathcal{G}} \leftarrow$ URIs in $\mathcal{G}$
20:               $\overline{U_{\mathcal{G}}} \leftarrow$ prune blacklisted from $U_{\mathcal{G}}$
21:               add unseen URIs in $\overline{U_{\mathcal{G}}}$ to $frontier$
22:               update $stats$ wrt. $\overline{U_{\mathcal{G}}}$
23:            **else**
24:               **if** uri_{deref} is unseen **then**
25:                  add uri_{deref} to $frontier$
26:                  update $stats$ for uri_{deref}
27:               **end if**
28:            **end if**
29:         **end if**
30:      **end for**
31:      $elapsed \leftarrow$ current_time() - $start$
32:      **if** $elapsed <$ MIN-DELAY **then**
33:         wait(MIN-DELAY $- elapsed$)
34:      **end if**
35:   **end while**
36: **end while**

---

proach to crawling according to observations that writing/reading concurrently and dynamically to a single queue can become the bottleneck in a large-scale crawler.

5.1.1. *Incorporating Politeness*

The crawler must be careful not to bite the hands that feed it by hammering the servers of data providers or breaching policies outlined in the provided `robots.txt` file [118]. We use pay-level-domains [88] (PLDs; a.k.a. "root domains"; e.g.,

`bbc.co.uk`) to identify individual data-providers, and implement politeness on a per-PLD basis. Firstly, when we first encounter a URI for a PLD, we cross-check the `robots.txt` file to ensure that we are permitted to crawl that site; secondly, we implement a "minimum PLD delay" to avoid hammering servers, viz.: a minimum time-period between subsequent requests to a given PLD. This is given by MIN-DELAY in Algorithm 1.

In order to accommodate the min-delay policy with minimal effect on performance, we must refine our crawling algorithm: large sites with a large internal branching factor (large numbers of unique intra-PLD outlinks per document) can result in the frontier of each round being dominated by URIs from a small selection of PLDs. Thus, naïve breadth-first crawling can lead to crawlers hammering such sites; conversely, given a politeness policy, a crawler may spend a lot of time idle waiting for the min-delay to pass.

One solution is to reasonably restrict the branching factor [88] – the maximum number of URIs crawled per PLD per round – which ensures that individual PLDs with large internal fan-out are not hammered; thus, in each round of the crawl, we implement a cut-off for URIs per PLD, given by PLD-LIMIT in Algorithm 1.

Secondly, to ensure the maximum gap between crawling successive URIs for the same PLD, we implement a per-PLD queue (given by $pld_{0...n}$ in Algorithm 1) whereby each PLD is given a dedicated queue of URIs filled from the frontier, and during the crawl, a URI is polled from each PLD queue in a round-robin fashion. If all of the PLD queues have been polled before the min-delay is satisfied, then the crawler must wait: this is given by Lines 31–34 in Algorithm 1. Thus, the minimum crawl time for a round – assuming a sufficiently full queue – becomes MIN-DELAY * PLD-LIMIT.

5.1.2. *On-disk Queue*

As the crawl continues, the in-memory capacity of the machine will eventually be exceeded by the capacity required for storing URIs [88]. Performing a stress-test, we observed that with 2GB of JAVA heap-space, the crawler could crawl approx. 199 k URIs (additionally storing the respective frontier URIs) before throwing an out-of-memory exception. In order to scale beyond the implied main-memory limitations of the crawler, we implement on-disk storage for URIs, with the additional benefit of maintaining a persistent state for the crawl and thus offering a "continuation point" useful for extension of an existing crawl, or recovery from failure.

We implement the on-disk storage of URIs using Berkeley DB which comprises of two indexes – the first provides lookups for URI strings against their status (polled/unpolled); the second offers a key-sorted map which can iterate over unpolled URIs in decreasing order of inlink count. The inlink count reflects the total number of documents from which the URI has been extracted thus far; we deem a higher count to roughly equate to a higher priority URI.

The crawler utilises both the on-disk index and the in-memory queue to offer similar functionality as above. The on-disk index and in-memory queue are synchronised at the start of each round:

(i) links and respective inlink counts extracted from the previous round (or seed URIs if the first round) are added to the on-disk index;

(ii) URIs polled from the previous round have their status updated on-disk;

(iii) an in-memory PLD queue is filled using an iterator of on-disk URIs sorted by descending inlink count.

Most importantly, the above process ensures that only the URIs active (current PLD queue and frontier URIs) for the current round must be stored in memory. Also, the process ensures that the on-disk index stores the persistent state of the crawler up to the start of the last round; if the crawler unexpectedly dies, the crawl can be resumed from the start of the last round. Finally, the in-memory PLD queue is filled with URIs sorted in order of inlink count, offering a cheap form of intra-PLD URI prioritisation (Algorithm 1, Line 8).

5.1.3. *Multi-threading*

The bottle-neck for a single-threaded crawler will be the response times of remote servers; the CPU load, I/O throughput and network bandwidth of a crawling machine will not be efficiently exploited by sequential HTTP GET requests over the Web. Thus, crawlers are commonly multi-threaded to mitigate this bottleneck and perform concurrent HTTP lookups. At a certain point of increasing the number of lookup threads operating, the CPU load, I/O load, or network bandwidth becomes an immutable bottleneck; this becomes the optimal number of threads.

In order to find a suitable thread count for our particular setup (with respect to processor/network bandwidth), we conducted some illustrative small-scale experiments comparing a machine crawling with the same setup and input parameters, but with an exponentially increasing number of threads: in particular, we measure the time taken for crawling 1,000 URIs given a seed URI [13] for 1, 2, 4, 8, 16, 32, 64, and 128 threads. Also, to alleviate the possible effects of remote caching on our comparison of increasing thread counts, we pre-crawled all of the URIs before running the benchmark.

For the different thread counts, Figure 5 overlays the total time taken in minutes to crawl the 1,000 URIs, and also overlays the average percentage CPU *idle* time. [14] Time and CPU% idle noticeably have a direct correlation. As the number of threads increases up until 64, the time taken for the crawl decreases – the reduction in time is particularly pronounced in earlier thread increments; similarly, and as expected, the CPU idle time decreases as a higher density of documents are retrieved and processed. Beyond 64 threads, the effect of increasing threads becomes minimal as the machine reaches the limits of CPU and disk I/O throughput; in fact, the total time taken starts to increase – we suspect that contention between threads for shared resources affects performance. Thus, we settle upon 64 threads as an approximately optimal figure for our setup.
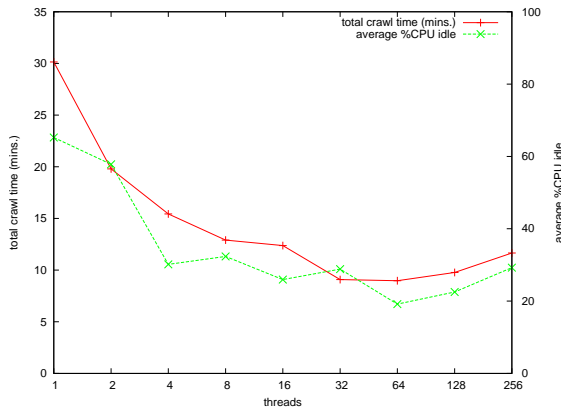

Fig. 5. Total time (mins.) and average percentage of CPU *idle* time for crawling 1,000 URIs with a varying number of threads

---

[13] http://sw.deri.org/~aidanh/foaf/foaf.rdf

[14] Idle times are measured as (100 - %CPU Usage), where CPU usage is extracted from the UNIX command `ps` taken every three seconds during the crawl.

5.1.4. *Crawling RDF/XML*

Since our architecture is currently implemented to index RDF/XML, we would feasibly like to maximise the ratio of HTTP lookups which result in RDF/XML content; i.e., given the total HTTP lookups as L, and the total number of downloaded RDF/XML pages as R, we would like to maximise the *useful ratio*: $ur = R/L$.

In order to reduce the amount of HTTP lookups wasted on non-RDF/XML content, we implement the following heuristics:

(i) firstly, we blacklist non-`http` protocol URIs;

(ii) secondly, we blacklist URIs with common file-extensions that are highly unlikely to return RDF/XML (e.g., `html`, `jpg`, `pdf`, etc.) following arguments we previously laid out in [121];

(iii) thirdly, we check the returned HTTP header and only retrieve the content of URIs reporting `Content-type: application/rdf+xml`; [15]

(iv) finally, we use a *credible useful ratio* when polling PLDs to indicate the probability that a URI from that PLD will yield RDF/XML based on past observations.

Our third heuristic involves rejecting content based on header information; this is perhaps arguable in that previous observations [76] indicate that 17% of RDF/XML documents are returned with a `Content-type` other than `application/rdf+xml`. Thus, we automatically exclude such documents from our crawl; however, here we put the onus on publishers to ensure correct reporting of `Content-type`.

With respect to the fourth heuristic above, we implement an algorithm for selectively polling PLDs based on their observed useful ratio; since our crawler only requires RDF/XML, we use this score to access PLDs which offer a higher percentage of RDF/XML more often. Thus, we can reduce the amount of time wasted on lookups of HTML documents and save the resources of servers for non-RDF/XML data providers.

The credible useful ratio for PLD i is derived from the following credibility formula:

$$cur_i = \frac{rdf_i + \mu}{total_i + \mu}$$

where rdf_i is the total number of RDF documents returned thus far by PLD i, $total_i$ is the total number of lookups performed for PLD i excluding redi-

---

[15] Indeed, one advantage RDF/XML has over RDFa is an unambiguous MIME-type useful in such situations

rects, and μ is a "credibility factor". The purpose of the credibility formula is to dampen scores derived from few readings (where $total_i$ is small) towards the value 1 (offering the benefit-of-the-doubt), with the justification that the credibility of a score with few readings is less than that with a greater number of readings: with a low number of readings ($total_i \ll \mu$), the cur_i score is affected more by μ than actual readings for PLD i; as the number of readings increases ($total_i \gg \mu$), the score is affected more by the observed readings than the μ factor. Note that we set this constant to 10. [16]

Example 1 If we observe that PLD $a = $ `deri.org` has returned 1/5 RDF/XML documents and PLD b $= $ `w3.org` has returned 1/50 RDF/XML documents, and if we assume $\mu = 10$, then $cur_a = (1 + \mu)/(5 + \mu) = 0.7\overline{3}$ and $cur_b = (1 + \mu)/(50 + \mu) = 0.18\overline{3}$. We thus ensure that PLDs are not unreasonably punished for returning non-RDF/XML documents early on (i.e., are not immediately assigned a cur of 0. $\diamond$

To implement selective polling of PLDs according to their useful ratio, we simply use the cur score as a probability of polling a URI from that PLD queue in that round (Algorithm 1, Lines 12–13). Thus, PLDs which return a high percentage of RDF/XML documents – or indeed PLDs for which very few URIs have been encountered – will have a higher probability of being polled, guiding the crawler away from PLDs which return a high percentage of non RDF/XML documents.

We evaluated the useful ratio scoring mechanism on a crawl of 100k URIs, with the scoring enabled and disabled. In the first run, with scoring disabled, 22,504 of the lookups resulted in RDF/XML (22.5%), whilst in the second run with scoring enabled, 30,713 lookups resulted in RDF/XML (30.7%). Table 1 enumerates the top 5 PLDs which were polled and the top 5 PLDs which were skipped for the crawl with scoring enabled, including the useful ratio (ur – the unaltered ratio of useful documents returned to non-redirect lookups) and the weighted useful ratio score (cur). The top 5 polled PLDs were observed to return a high-percentage of RDF/XML, and the top 5 skipped PLDs were observed to return a low percentage of RDF.

PLD	polled	skipped	ur	cur	% polled
top five polled					
`linkedmdb.org`	**1,980**	0	1	1	100
`geonames.org`	**1,971**	26	0.996	0.996	98.7
`rdfabout.com`	**1,944**	0	1	1	100
`fu-berlin.de`	1,925	74	0.947	0.949	96.3
`bbc.co.uk`	**1,917**	33	0.982	0.982	98.3
top five skipped					
`deri.ie`	233	**1,814**	0.013	0.054	11.4
`megginson.com`	196	**1,795**	0	0.049	9.8
`xbrl.us`	206	**1,785**	0	0.046	10.3
`wikipedia.org`	203	**1,784**	0	0.051	10.2
`uklug.co.uk`	215	**1,776**	0	0.044	10.7

Table 1
Useful ratio (ur) and credible useful ratio (cur) for the top five most often polled/skipped PLDs

5.2. Distributed Approach

We have seen that given a sufficient number of threads, the bottleneck for multi-threaded crawling becomes the CPU and/or I/O capabilities of one machine; thus, by implementing a distributed crawling framework balancing the CPU workload over multiple machines, we expect to increase the throughput of the crawl. We apply the crawling to our framework as follows:

(i) **scatter**: the master machine scatters a seed list of URIs to the slave machines, using a hash-based *split* function;

(ii) **run**: each slave machine adds the new URIs to its frontier and performs a round of the crawl, writing the retrieved and parsed content to the local hard-disk, and creating a frontier for the next round;

(iii) **co-ordinate**: each slave machine then uses the `split` function to scatter new frontier URIs to its peers.

Steps (ii) & (iii) are recursively applied until Rounds has been fulfilled. Note that in Step (ii), we adjust the Min-Delay for subsequent HTTP lookups to a given PLD value by multiplying the number of machines: herein, we somewhat relax our politeness policy (e.g., no more than 8 lookups every 4 seconds, as opposed to 1 lookup every 0.5 seconds), but deem the heuristic sufficient assuming a relatively small number of machines and/or large number of PLDs.

In order to evaluate the effect of increasing the number of crawling machines within the framework, we performed a crawl performing lookups on 100k URIs on 1, 2, 4 and 8 machines using 64 threads. The results are presented in Table 2, showing num-

#machines	1	2	4	8
mins	360	156	71	63
%delay	1.8	10	81.1	94.6

Table 2
Time taken for a crawl performing lookups on 100 k URIs, and average percentage of time each queue had to enforce a politeness wait, for differing numbers of machines

ber of machines, number of minutes taken for the crawl, and also the percentage of times that the in-memory queue had to be delayed in order to abide by our politeness policies. There is a clear increase in the performance of the crawling with respect to increasing number of machines. However, in moving from four machines to eight, the decrease in time is only 11.3%. With 8 machines (and indeed, starting with 4 machines), there are not enough active PLDs in the queue to fill the adjusted min-delay of 4 seconds (8*500 ms), and so the queue has a delay hit-rate of 94.6%.

We term this state *PLD starvation*: the slave machines do not have enough unique PLDs to keep them occupied until the MIN-DELAY has been reached. Thus, we must modify somewhat the end-of-round criteria to reasonably improve performance in the distributed case:

– firstly, a crawler can return from a round if the MIN-DELAY is not being filled by the active PLDs in the queue – the intuition here being that new PLDs can be discovered in the *frontier* of the next round;

– secondly, to ensure that the slave machines don't immediately return in the case that new PLDs are not found in the *frontier*, we implement a PLD-LIMIT which ensures that slave machines don't immediately return from the round;

– finally, in the case that one slave crawler returns from a round due to some stopping criteria, the master machine will request that all other slave machines also end their round such that machines do not remain idle waiting for their peers to return.

The above conditions help to somewhat mitigate the effect of PLD starvation on our distributed crawl; however, given the politeness restriction of 500 ms per PLD, this becomes a hard-limit for performance independent of system architecture and crawling hardware, instead imposed by the nature of the Web of Data itself. Also, as a crawl progresses, active PLDs (PLDs with unique content still to crawl) will become less and less, and the performance of the distributed crawler will approach that of a single-machine crawl. As Linked Data publish-

ing expands and diversifies, and as the number of servers offering RDF content increases, better performance would be observed for distributed crawling on larger numbers of machines: for the moment, we observe that 8 machines currently approaches the limit of performance given our setup and policies.

5.3. *Full-Scale Evaluation*

To perform scale-up experiments for the crawler – and indeed to achieve a large dataset for evaluation of later components – we ran the crawler continuously for 52.5 h on 8 machines from a seed list of ∼8 m URIs extracted from an old dataset with *cur* scoring enabled.[17] In that time, we gathered a total of 1.118 g quads, of which 11.7 m were duplicates (∼1% – representing duplicate triples being asserted in the same document); we provide a selection of statistics characterising the dataset in Appendix A.

We observed a mean of 140 m quads per machine and an average absolute deviation of 1.26 m across machines: considering that the average absolute deviation is ∼1% of the mean, this indicates near optimal balancing of output data on the machines.

The crawl attempted 9.206 m lookups, of which 448 k (4.9%) were for `robots.txt` files. Of the remaining 8.758 m attempted lookups, 4.793 m (54.7%) returned response code `200 Okay`, 3.571 m (40.7%) returned a redirect response code of the form `3xx`, 235 k (2.7%) returned a client error code of the form `4xx` and 95 k (1.1%) returned a server error of the form `5xx`; 65 k (0.7%) were disallowed due to restrictions specified by the `robots.txt` file. Of the 4.973 m lookups returning response code `200 Okay`, 4.022 m (83.9%) returned content-type `application/rdf+xml`, 683 k (14.3%) returned `text/html`, 27 k (0.6%) returned `text/turtle`, 27 k (0.6%) returned `application/json`, 22 k (0.4%) returned `application/xml`, with the remaining 0.3% comprising of relatively small amounts of 97 other content-types. Of the 3.571 m redirects, 2.886 m (80.8%) were `303 See Other` as used in Linked Data to disambiguate general resources from information resources, 398 k (11.1%) were `301 Moved Permanently`, 285 k (8%) were `302 Found`, 744 (∼0%) were `307 Temporary Redirect` and 21 (∼0%) were `300 Multiple Choices`. In summary, of the non-`robots.txt` lookups, 40.7% were redirects and

---

[17] The crawl was conducted in late May, 2010

45.9% were `200 Okay`/`application/rdf+xml` (as rewarded in our *cur* scoring mechanism).

An overview of the total number of URIs crawled per each hour is given in Figure 6; in particular, we observe a notable decrease in performance as the crawl progresses. In Figure 7, we give a breakdown of three categories of lookups: `200 Okay`/RDF/XML lookups, redirects, and *other* – again, our *cur* scoring views the latter category as wasted lookups. We note an initial decrease in the latter category of lookups, which then plateaus and varies between 2.2% and 8.8%.

During the crawl, we encountered 140 k PLDs, of which only 783 served content under `200 Okay`/`application/rdf+xml`. However, of the non-`robots.txt` lookups, 7.748 m (88.5%) were on the latter set of PLDs: on average, 7.21 lookups were performed on each PLD which returned *no* RDF/XML, whereas on average, 9,895 lookups were performed on each PLD which returned *some* RDF/XML. Figure 8 gives the number of active and new PLDs per crawl hour, where 'active PLDs' refers to those to whom a lookup was issued in that hour period, and 'new PLDs' refers to those who were newly accessed in that period; we note a high increase in PLDs at hour 20 of the crawl, where a large amount of 'non-RDF/XML PLDs' were discovered. Perhaps giving a better indication of the nature of PLD starvation, Figure 9 renders the same information for only those PLDs who return some RDF/XML, showing that half of said PLDs are exhausted after the third hour of the crawl, that only a small number of new 'RDF/XML PLDs' are discovered after the third hour (between 0 and 14 each hour), and that the set of active PLDs plateaus at ~50 towards the end of the crawl.

5.4. *Related Work*

Parts of our architecture and some of our design decisions are influenced by work on traditional Web crawlers; e.g., the IRLBot system of Lee et al. [88] and the distributed crawler of Boldi et. al. [15].

The research field of focused RDF crawling is still quite a young field, with most of the current work based on the lessons learnt from the more mature area of traditional Web crawling. Related work in the area of focused crawling can be categorised [7] roughly as follows:
- *classic focused crawling*: e.g., [22] uses primary link structure and anchor texts to identify pages

about a topic using various text similarity of link analysis algorithms;
- *semantic focused crawling*: is a variation of classical focused crawling but uses conceptual similarity between terms found in ontologies [41,40]
- *learning focused crawling*: [37,107] uses classification algorithms to guide crawlers to relevant Web paths and pages.

However, a major difference between these approaches and ours is that our definition of high quality pages is not based on topics or ontologies, but instead on the content-type of documents.

With respect to crawling RDF, the Swoogle search engine implements a crawler which extracts links from Google, and further crawls based on various – sometimes domain specific – link extraction techniques [38]; like us, they also use file extensions to throw away non-RDF URIs. In [27], the authors provide a very brief description of the crawler used by the FalconS search engine for obtaining RDF/XML content; interestingly, they provide statistics identifying a power-law type distribution for the number of documents provided by each pay-level domain, correlating with our discussion of PLD-starvation. In [114], for the purposes of the WATSON engine, the authors use Heritrix [18] to retrieve ontologies using Swoogle, Google and Protégé indexes, and also crawl by interpreting `rdfs:seeAlso` and `owl:imports` as links – they do not exploit the dereferencability of URIs popularised by Linked Data. Similarly, the Sindice crawler [103] retrieves content based on a *push* model, crawling documents which pinged some central service such as PingTheSemanticWeb [19]; they also discuss a PLD-level scheduler for ensuring politeness and diversity of data retrieved.

However, none of the above RDF crawlers provide significant analysis of performance, nor do they discuss a distribution model for crawling. Also for example, none discuss punishing/rewarding PLDs based on previous experience of the ratio of RDF/XML content retrieved therefrom.

5.5. *Future Directions and Open Research Questions*

From a pragmatic perspective, we would prioritise extension of our crawler to handle arbitrary RDF formats – especially the RDFa format which is growing in popularity. Such an extension may

---

[18] http://crawler.archive.org/
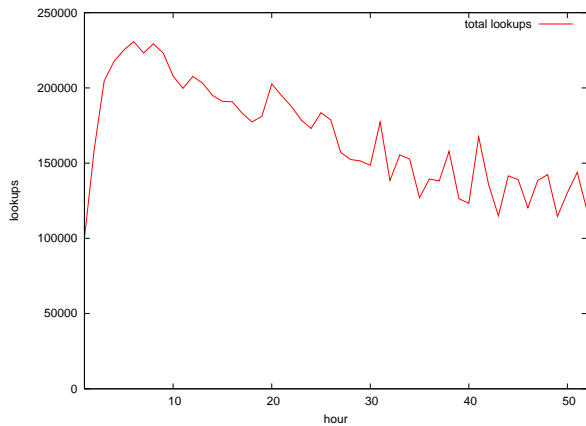[19] http://pingthesemanticweb.com

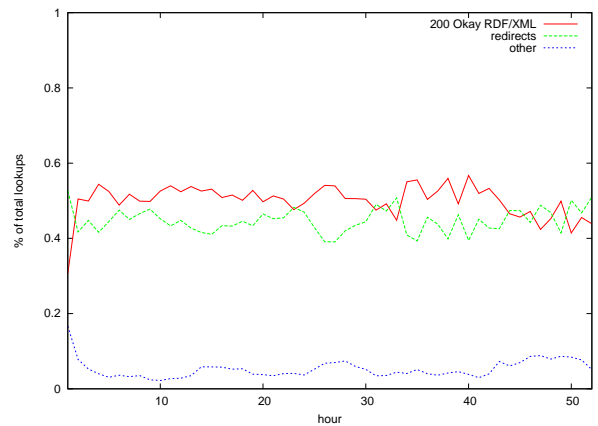Fig. 6. Number of HTTP lookups per crawl hour.

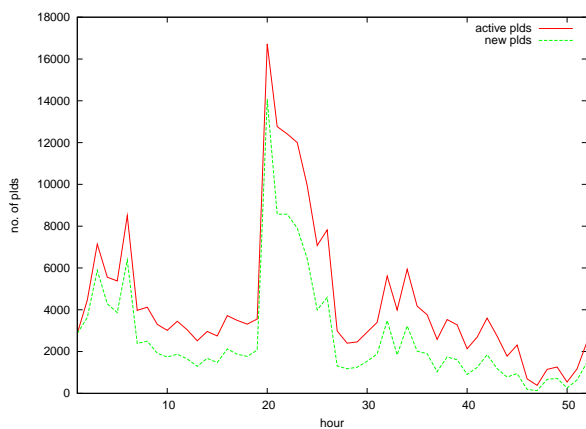Fig. 7. Breakdown of HTTP lookups per crawl hour.

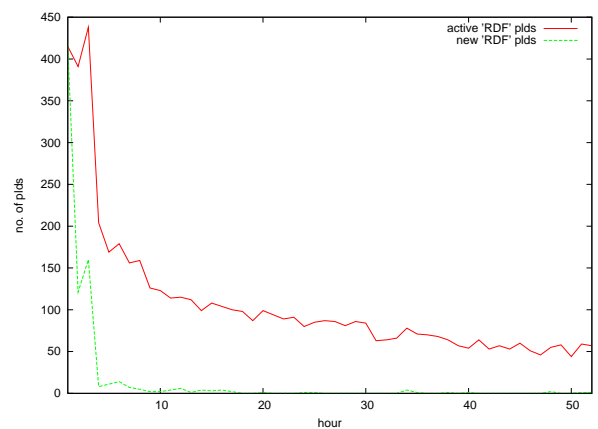Fig. 8. Breakdown of PLDs per crawl hour.

Fig. 9. Breakdown of RDF/XML PLDs per crawl hour.

mandate modification of the current mechanisms for ensuring a high percentage of RDF/XML documents: for example, we could no longer blacklist URIs with a `.html` file extension, nor could we rely on the `Content-type` returned by the HTTP header (unlike RDF/XML, RDFa does not have a specific MIME-type).

Along these lines, we could perhaps also investigate extraction of structured data from non-RDF sources; these could include Microformats, metadata embedded in documents such as PDFs and images, extraction of HTML meta-information, HTML scraping, etc. Again, such a process would require re-visitation of our RDF-centric focused crawling techniques.

The other main challenge posed in this section is that of PLD starvation; although we would expect this to become less of an issue as the Semantic Web matures, it perhaps bears further investigation. For example, we have yet to fully evaluate the trade-off

between small rounds with frequent updates of URIs from fresh PLDs, and large rounds which persist with a high delay-rate but require less co-ordination. Also, given the inevitability of idle time during the crawl, it may be practical from a performance perspective to give the crawler more tasks to do in order to maximise the amount of processing done on the data, and minimise idle time.

Finally, we have not discussed the possibility of incremental crawls: choosing URIs to recrawl may lead to interesting research avenues. Besides obvious solutions such as HTTP caching, URIs could be re-crawled based on, e.g., detected change frequency of the document over time, some quality metric for the document, or how many times data from that document was requested in the UI. More practically, an incremental crawler could use PLD statistics derived from previous crawls, and the HTTP headers for URIs – including redirections – to achieve a much higher ratio of lookups to RDF documents returned.

15

Such considerations would largely countermand the effects of PLD starvation, by reducing the amount of lookups the crawler needs in each run.

6. Entity Consolidation

In theory, RDF enables excellent data-integration over data sourced from arbitrarily many sources – as is the case for our corpora collected by our crawler. However, this integration is premised on the widespread sharing and re-use – across all sources – of URIs for specific entities. In reality, different RDF documents on the Web published by independent parties often speak about the same entities using different URIs [75];[20] to make matters worse, RDF allows for the definition of anonymous entities – entities identified by a blank node – without a prescribed URI.

As an example, in our 1.118 g statement Linked Data corpus we found 23 different URIs identifying the person Tim Berners-Lee – these identifiers spanned 9 different PLDs.[21] Now, given a keyword query for "`tim berners lee`", the data using each of the 23 different identifiers would be split over 23 different results, even though they all refer to the same entity.

Offering search and querying over a raw RDF dataset collected from the Web would thus entail many duplicate results referring to the same entity, emulating the current situation on the HTML Web where information about different resources is fragmented across source documents. Given a means of identifying *equivalent entities* in RDF data – entities representing the same real-world individual but identified incongruously – would enable the merging of information contributions on an entity given by heterogeneous sources without the need for consistent URI naming of entities.

In fact, OWL [115] provides some standard solutions to such problems. Firstly, OWL defines the `owl:sameAs` property which is intended to relate two equivalent entities; the property has symmetric, transitive and reflexive semantics as one would expect. Many sources on the Web offer `owl:sameAs` links between entities described locally and equivalent entities described remotely.

Further, OWL provides some other mechanisms for discovering implicit `owl:sameAs` relations in the absence of explicit relations: the most prominent such example is provision of the class `owl:Inverse-FunctionalProperty`, which defines a class of properties whose value uniquely identifies an entity. One example of an inverse-functional property would be an ISBN property, where ISBN values uniquely identify books. If two entities share the same ISBN value, a same-as relation can be inferred between them. Using OWL, same-as relations can also be detected using `owl:FunctionalProperty`, `owl:-maxCardinality`, and `owl:cardinality` (and, now in OWL2RL using `owl:maxQualifiedCardinality` and `owl:qualifiedCardinality`): however, the recall of inferences involving the latter OWL constructs are relatively small [77] and thus considered out of scope here.

In [75], we provided a simple batch-processing based approach for deriving `owl:sameAs` relations between individuals using inverse-functional properties defined in the data. However, in our experience, the precision of such inferences can be quite poor. As an example, in [75] we found 85,803 equivalent individuals to be inferable from a Web dataset through the incongruous values `08445a31a78661b5c746feff39a9db6e4e2cc5cf` and `da39a3ee5e6b4b0d3255bfef95601890afd80709` for the prominent inverse-functional property `foaf:mbox_sha1sum`: the former value is the sha1-sum of an empty string and the latter is the sha1-sum of the '`mailto:`' string, both of which are erroneously published by online Friend Of A Friend (FOAF – a very popular vocabulary used for personal descriptions) exporters.[22] Aside from obvious pathological cases – which can of course be blacklisted – publishers commonly do not respect the semantics of inverse-functional properties [76].

More recently, in [72] we showed that we could find 1.31x sets of equivalent identifiers by including reasoning over inverse-functional properties and functional-properties, than when only considering explicit `owl:sameAs`. These sets contained 2.58x more identifiers. However, we found that the additional equivalences found through such an approach were mainly between blank-nodes on domains which do not use URIs to identify resources, common for older FOAF exporters: we found a 6% increase in URIs involved in an equivalence. We again observed

that the equivalences given by such an approach tend to offer more noise than when only considering explicit `owl:sameAs` relations.

In fact, the performance of satisfactory, high-precision, high-recall entity consolidation over large-scale Linked Data corpora is still an open research question. At the moment, we rely on `owl:sameAs` relations which are directly asserted in the data to perform consolidation, and this section briefly outlines our distributed approach, provides performance evaluation for the algorithm, and provides some insights into the fecundity of such an approach – with respect to finding equivalence – over Linked Data.

6.1. *High-level Approach*

The overall approach involves two scans of the main body of data, with the following high-level steps:

(i) `owl:sameAs` statements are extracted from the data: the main body of data is scanned once, identifying `owl:sameAs` triples and buffering them to a separate location;

(ii) the transitive/symmetric closure of the `owl:sameAs` statements are computed, inferring new `owl:sameAs` relations;

(iii) for each set of equivalent entities found (each *equivalence class*), a *canonical identifier* is chosen to represent the set in the consolidated output;

(iv) the main body of data is again scanned and consolidated: identifiers are rewritten to their canonical form – we do not rewrite identifiers in the predicate position, objects of `rdf:type` triples, or literal objects.

In previous work, we have presented two approaches for performing such consolidation; in [75], we stored `owl:sameAs` in memory, computing the transitive/symmetric closure in memory, and performing in-memory lookups for canonical identifiers in the second scan. In [77], we presented a batch-processing technique which uses on-disk sorts and scans to execute the `owl:sameAs` transitive/symmetric closure, and the canonicalisation of identifiers in the main body of data. The former approach is in fact much faster in that it reduces the amount of time consumed by hard-disk I/O operations; however, the latter batch-processing approach is not limited by the main-memory capacity of the system. Either approach is applicable with our

consolidation component (even in the distributed case); however, since for the moment we only operate on asserted `owl:sameAs` statements (we found $\sim$12 m `owl:sameAs` statements in our full-scale crawl, which we tested to be within our 4GB in-memory capacity using the flyweight pattern), for now we apply the faster in-memory approach.

Standard OWL semantics mandates duplication of data for all equivalent terms by the semantics of replacement (cf. Table 4, [51]). However, this is not a practical option at scale. Firstly, the amount of duplication will be quadratic with respect to the size of an equivalence class – as we will see in Section 6.3, we find equivalence classes with 8.5 k elements. If one were to apply transitive, reflexive and symmetric closure of equivalence over these identifiers, we would produce $8.5k^2 = 72.25m$ `owl:sameAs` statements alone; further assuming an average of 6 unique quads for each identifier – 51 k unique quads in total – we would produce a further 433.5 m repetitive statements by substituting each equivalent identifier into each quad. Secondly, such duplication of data would result in multitudinous duplicate results being presented to end-users, with obvious impact on the usability of the system.

Thus, the practical solution is to abandon standard OWL semantics and instead *consolidate* the data by choosing a canonical identifier to represent the output data for each equivalence class. Canonical identifiers are chosen with preference of URIs over blank-nodes, and thereafter we arbitrarily use a lexicographical order – the canonical identifiers are only used internally to represent the given entity. [23] Along these lines, we also preserve all URIs used to identify the entity by outputting `owl:sameAs` relations to and from the canonical identifier (please note that we do not preserve redundant blank-node identifiers which are only intended to have a local scope, have been assigned arbitrary labels during the crawling process, and are not subject to re-use), which can subsequently be used to display all URIs originally used to identify an entity, or to act as a "redirect" from an original identifier to the canonical identifier containing the pertinent information.

In the in-memory map structure, each equivalence class is assigned a canonical identifier according to the above ordering. We then perform a second scan of the data, rewriting terms according to canonical

---

[23] If necessary, the ranking algorithm presented in the next section could be used to choose the most popular identifier as the canonical identifier.

identifiers. Please note that according to OWL Full semantics, terms in the predicate position and object position of `rdf:type` triples should be rewritten (referring to term positions occupied by properties and classes respectively in membership assertions; again, cf. Table 4, [51]). However, we do not wish to rewrite these terms: in OWL, equivalence between properties can instead be specified by means of the `owl:equivalentProperty` construct, and between classes as the `owl:equivalentClass` construct.[24] We omit rewriting class/property terms in membership assertions, handling inferences involving classes/properties by alternate means in Section 8.

Thus, in the second scan, the subject and object of non-`rdf:type` statements are rewritten according to the canonical identifiers stored in the in-memory map, with rewritten statements written to output. If no equivalent identifiers are found, the statement is buffered to the output. When the scan is complete, `owl:sameAs` relations to/from canonical URIs and their equivalent URIs are appended to the output. Consolidation is now complete.

6.2. *Distributed Approach*

Assuming that the target-data of the consolidation is arbitrarily and preferably evenly split over multiple machines – as is the result of our crawling component – we can apply the consolidation process in a distributed manner as follows:

(i) **run**: `owl:sameAs` statements are extracted from the data in parallel on each slave machine;

(ii) **gather**: the `owl:sameAs` statements are gathered onto the master machine, which computes the transitive/symmetric closure over them, and chooses the canonical identifiers;

(iii) **flood**: the closed `owl:sameAs` statements and chosen canonical identifiers are sent (in their entirety) to the slave machines;

(iv) **run**: in parallel, the slave machines scan the main body of data, rewriting identifiers to their canonical form and outputting canonicalised triples to a new file.

In the above process, only the `owl:sameAs` statements need be transferred between machines. The

---

[24] As an example of naïve usage of `owl:sameAs` between classes and properties on the Web, please see: `http://colab.cim3.net/file/work/SICoP/DRMITIT/DRM_OWL/Categorization/TaxonomyReferenceModel.owl`

#machines	1	2	4	8
mins	12	6.2	3.2	1.8

Table 3
Time taken for consolidation of ∼31.3 m statements for differing numbers of machines

more expensive on-disk scans can be conducted in parallel, and thus we would reasonably expect near-linear scale with respect to the number of machines for consolidation over a fixed dataset – the assumptions being that the data has been pre-distributed, that the proportion of `owl:sameAs` statements is relatively small compared to the main body of data, and that the dataset is relatively large compared with the number of machines (all of which apply in our setting).

In order to evaluate the above claim, we ran a small scale experiment over 1, 2, 4 and 8 machines using a dataset of 31.3 m statements extracted from one of the 100 k URI crawls from the previous section. The dataset contained 24.9 k `owl:sameAs` statements. Table 3 presents the total time taken for each experiment, where in particular, performance appears to be a near-linear function on the number of machines.

6.3. *Full-scale Evaluation*

Using the above setup, we ran consolidation over our full-scale (1.118 g) RDF crawl with one master and 8 slave machines. The entire consolidation process took 63.3 min.

The first scan extracting `owl:sameAs` statements took 12.5 min, with an average idle time for the servers of 11 s (1.4%) – i.e., on average, the slave machines spent 1.4% of the time idly waiting for peers to finish. Transferring, aggregating and loading the `owl:sameAs` statements on the master machine took 8.4 min. In total, 11.93 m `owl:sameAs` statements were extracted; 2.16 m equivalence classes were found, containing 5.75 m terms – an average of 2.65 elements per equivalence class. Figure 10 presents the distribution of sizes of the equivalence classes, where the largest equivalence class contains 8,481 equivalent entities and 1.6 m (74.1%) equivalence classes contain two equivalent identifiers.

Table 4 shows the canonical URIs for the largest 5 equivalence classes, and whether the results were verified as correct/incorrect by manual inspection. Indeed, results for class **1** and **2** were deemed incorrect due to overly-liberal use of `owl:sameAs` for linking drug-related entities in the DailyMed and

LinkedCT exporters. [25] Results **3** and **5** were verified as correct consolidation of prominent Semantic Web related authors, resp.: `Dieter Fensel` and `Rudi Studer` – authors are given many duplicate URIs by the RKBExplorer co-reference index. [26] Result **4** contained URIs from various sites generally refering to the United States, mostly from DB-Pedia and LastFM. With respect to the DPPedia URIs, these (i) were equivalent but for capitilisation variations or stop-words, (ii) were variations of abbreviations or valid synonyms, (iii) were different language versions (e.g., `dbpedia:États_Unis`), (iv) were nicknames (e.g., `dbpedia:Yankee_land`), (v) were related but not equivalent (e.g., `dbpedia.org:American_Civilization`), (vi) were just noise (e.g., `dbpedia:LOL_Dean`).

Besides the largest equivalence classes – which we have seen are prone to errors perhaps due to the snowballing effect of the transitive and symmetric closure – we also randomly sampled 100 equivalent sets and manually checked for errors based on label (as an intuitive idea of what the identifier refers to) and type. We verified that all 100 were correct (or, more accurately, were not obviously incorrect). [27]

The second scan rewriting the data according to the canonical identifiers took in total 42.3 min, with an average idle time of 64.7 s (2.5%) for each machine at the end of the round. The slower time for the second round is attributable to the extra overhead of re-writing the data to disk, as opposed to just reading. Identifiers in 188 m positions of the 1.118 g statements were rewritten.

From a overall performance perspective, we note that 86.6% of the time is spent in parallel execution, and during that time, peers are idle for <2.3% of total parallel execution time without any bespoke load-balancing. However, we note that aggregating and loading the same-as statements on the master machines is somewhat slow (13.3% of computation
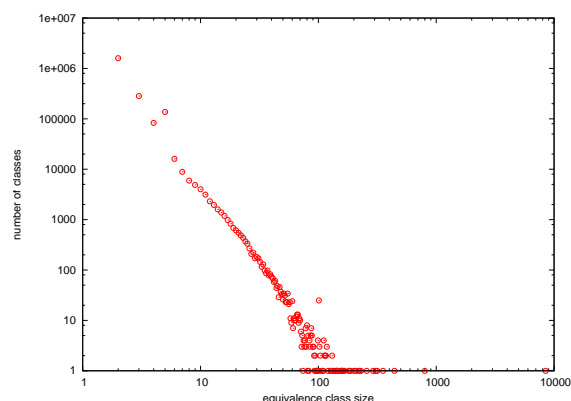

Fig. 10. Distribution of sizes of equivalence classes (log/log scale)

time) where the slave swarm – the bulk of processing power – is idle for this period; also, this becomes the lower bound on computation time for increasing machines. However, in absolute terms, we deem 8.4 min not to be so significant for off-line processing.

6.4. Related Work

Entity consolidation has an older related stream of research relating largely to databases, with work under the names of record linkage, instance fusion, and duplicate identification; cf. [102,95,25] and a survey at [42]. Due to the lack of formal specification for determining equivalences, these older approaches are mostly concerned with probabilistic methods.

With respect to RDF, Bouquet et al. [17] motivate the problem of (re)using common identifiers as one of the pillars of the Semantic Web, and provide a framework for mapping heterogeneous identifiers to a centralised naming-scheme for re-use across the Web – some would argue that such a centralised service would not be in-tune with the architecture or philosophy of the Web.

The Sindice and Sig.ma search systems internally uses inverse-functional properties to find equivalent identifiers [103,120] – Sindice uses reasoning to identify a wider range of inverse-functional properties [103]. Online systems RKBExplorer [50] [28], <sameAs> [29] and ObjectCoref [30] offer on-demand querying for `owl:sameAs` relations found for a given input URI, which they internally compute and store; as previously alluded to, the former publish

---

[25] Please see `http://groups.google.com/group/pedantic-web/browse_thread/thread/ad740f7052cc3a2d` for Pedantic Web discussion on this issue – we contacted the publishers to request a fix.

[26] For example, see the co-reference results given by `http://www.rkbexplorer.com/sameAs/?uri=http://acm.rkbexplorer.com/id/person-53292-22877d02973d0d01e8f29c7113776e7e`, which at the time of writing correspond to 436 out of the 443 equivalent URIs found for `Dieter Fensel`.

[27] Many were simple 'syntactic' equivalences from the `opiumfield.com` LastFM data exporter; for reference, we've published the 100 sets at `http://aidanhogan.com/swse/eqcs-sample-100.txt`.

[28] `http://www.rkbexplorer.com/sameAs/`

[29] `http://sameas.org/`

[30] `http://ws.nju.edu.cn/objectcoref/`

#	Canonical Term (Lexically Lowest in Equivalence Class)	Size	Correct?
1	http://bio2rdf.org/dailymed_drugs:1000	8,481	✗
2	http://bio2rdf.org/dailymed_drugs:1042	800	✗
3	http://acm.rkbexplorer.com/id/person-53292-22877d02973d0d01e8f29c7113776e7e	443	✓
4	http://agame2teach.com/#ddb61cae0e083f705f65944cc3bb3968ce3f3ab59-ge_1	353	✓/✗
5	http://acm.rkbexplorer.com/id/person-236166-1b4ef5fdf4a5216256064c45a8923bc9	316	✓

Table 4

 Largest 5 equivalence classes

`owl:sameAs` relations for authors and papers in the area of scientific publishing.

The authors of [124] present *Silk*: a framework for creating and maintaining inter-linkage between domain-specific RDF datasets; in particular, this framework provides publishers with a means of discovering and creating `owl:sameAs` links between data sources using domain-specific rules and parameters. Thereafter, publishers can integrate discovered links into their exports, enabling better linkage of the data and subsequent consolidation by data consumers: this framework goes hand-in-hand with our approach, producing the `owl:sameAs` relations which we consume.

In [54], the authors discuss the semantics and current usage of `owl:sameAs` in Linked Data, discussing issues relating to *identity*, and providing four categories of `owl:sameAs` usage to relate entities which are closely related, but for which the semantics of `owl:sameAs` – particularly substitution – does not quite hold.

6.5. *Future Directions and Open Research Questions*

In this section, we have focused on the performance of what we require to be a distributed and scalable consolidation component. We have not presented analysis of the precision or recall of such consolidation – such evaluation is difficult to achieve in practice given a lack of gold-standard, or suitable means of accurately verifying results. The analysis of the precision and recall of various scalable consolidation methods on current Web data would represent a significant boon to research in the area of querying over Linked Data.

We are currently investigating statistical consolidation methods, with particular emphasis on extracting some notion of the quality or trustworthiness of derived equivalences [79]. Presently, we try to identify "quasi-inverse-functional" and "quasi-functional" properties (properties which are useful for distinguishing identity) using statistical analysis of the input data. We then combine shared property/value pairs for entities and derive a fuzzy value representing the confidence of equivalence between said entities. However, this preliminary work needs further investigation – including scalability and performance testing, and integration with more traditional reasoning-centric approaches for consolidation – before being included in the SWSE pipeline.

A further avenue for research in the same vein is applying "disambiguation", or attempting to assert that two entities cannot (or are likely not) to be equivalent using statistical approaches or analysis of inconsistencies in reasoning: disambiguation would allow for increasing the precision of the consolidation component by quickly removing "obvious" false positives.

Again, such approaches would likely have a significant impact on the quality of data integration possible in an engine such as SWSE operating over RDF Web data.

7. Ranking

Ranking is an important mechanism in the search process with the function of prioritising data elements. Herein, we want to quantify the importance of consolidated entities in the data, such that can be used for ordering the presentation of results returned when users pose a keyword query (e.g., see Figure 1), such that the most "important" results appear higher in the list. (Note that we will combine these ranking scores with relevance scores later in Section 9.)

As such, there is a significant body of related work on link-based algorithms for the Web (seminal works include [106,85]). A principal objective when ranking on the Web is rating popular pages higher than unpopular ones – further, ranks can be used for performing top-k processing, allowing the search engine to retrieve and process small segments of results ordered by their respective rank. Since we share similar goals, we wish to leverage the benefits of links-based analysis, proven for the HTML Web,

for the purposes of ranking Linked Data entities. Along these lines, we identify the following requirements for ranking Linked Data, which closely align with those of HTML-centric ranking schemes:

- the methods should be **scalable**, and applicable in scenarios involving large corpora of RDF;
- the methods should be **automatic** and **domain-agnostic**, and not inherently favouring a given domain or source of data;
- the methods should be **robust** in the face of spamming.

With respect to ranking the entities in our corpus in a manner sympathetic with our requirements, we further note the following:

- on the level of triples (data level), publishers can provide arbitrary information in arbitrary locations using arbitrary identifiers: thus, to discourage low-effort spamming, the source of information must be taken into account;
- following traditional link-based ranking intuition, we should consider links from one source of information to another as a 'positive vote' from the former to the latter;
- in the absence of sufficient source-level ranking, we should infer links between sources based on usage of identifiers on the data level, and some function mapping between data-level terms and sources;
- data providers who reuse identifiers from other sources should not be penalised: their data sources should not lose any rank value.

In particular, our methods are inspired by Google's PageRank [106] algorithm, which interprets hyperlinks to other pages as positive votes. However, PageRank is generally targeted towards hypertext documents, and adaptation to Linked Data sources is non-trival, given that the notion of a hyperlink (interpreted as a vote for a particular page) is missing: Linked Data principles mandate *implicit* links to other Web sites or data sources through re-use of dereferenceable URIs. Also, the unit of search is no longer a document, but an entity.

In previous work [61], we proposed a scalable algorithm for ranking structured data from an open, distributed environment, based on a concept we term *naming authority*. We re-introduce select important discussion from [61] and extend here by implementing the method in a distributed way and re-evaluating with respect to performance.

7.1. *High-level Approach*

Although we wish to rank entities, our ranking algorithm must consider the source of information to avoid low-effort data-level spamming. Thus, we must first have a means of ranking source-level identifiers and thereafter can propagate such ranks to the data-level.

In order to leverage existing links-based analysis techniques, we need to build a graph encoding the interlinkage of Linked Data sources. Although one could examine use of, e.g., `owl:imports` or `rdfs:seeAlso` links, and interpret them directly as akin to a hyperlink, the former is used solely in the realm of OWL ontology descriptions and the latter is not restricted to refer to RDF documents; similarly, both ignore the data-level linkage that exists by means of **LDP4** (include links using external URIs). Thus, we aim to infer source-level links through usage of data-level URIs in the corpus.

To generalise this idea, we previously defined the notion of "naming authority" for identifiers: a naming authority is a data source with the power to define identifiers of a certain structure [61]. Naming authority is an abstract term which could be applied to a knowable provenance of a piece of information, be that a document, host, person, organisation or other entity. Data items which are denoted by unique identifiers may be reused by sources other than the naming authority.

Example 2 With respect to Linked Data principles (see Section 4.2), consider for example the data-level URI `http://danbri.org/foaf.rdf#danbri`. Clearly the owner(s) of the `http://www.danbri.org/foaf.rdf` document (or, on a coarser level, the `danbri.org` domain) can claim some notion of 'authority' for this URI: following **LDP4**, the usage of the URI on other sites can be seen as a vote for the respective data source. We must also support redirects as commonly used for **LDP3** – thus we can reuse the `deref` function given in Section 4.2 as a function which maps an arbitrary URI identifier to the URI of its naming authority document (or to itself in the absence of a redirect). ◇

Please note that there is no obvious function for mapping from literals to naming authority, we thus omit them from our source-level ranking (one could consider a mapping based on datatype URIs, but we currently see no utility in such an approach). Also, blank nodes may only appear in one source document and are not subject to re-use: although

one could reduce the naming authority of a blank-node to the source they appear in, clearly only self-links can be created.

Continuing, we must consider the granularity of naming authority: in [61], we discussed and contrasted interpretation of naming authorities on a document level (e.g., `http://www.danbri.org/foaf.rdf`) and on a PLD level (`danbri.org`). Given that the PLD-level linkage graph is significantly smaller than the document-level graph, the overhead for aggregating and analysing the PLD-level graph is significantly reduced, and thus we herein perform ranking at a PLD-level.

Please note that for convenience, we will assume that PLDs are identified by URIs; e.g. (`http://danbri.org/`). We also define the convenient function $\mathsf{pld} : \mathbf{U} \to \mathbf{U}$ which extracts the PLD identifier for a URI (if the PLD cannot be parsed for a URI, we simply ignore the link) – we may also conveniently use the function $\mathsf{plds} : 2^{\mathbf{U}} \to 2^{\mathbf{U}}$ for sets of URIs.

Thus, our ranking procedure consists of the following steps:

(i) Construct the *PLD-level naming authority graph*: for each URI u appearing in a triple t in the input data, create links from the PLDs of sources mentioning a particular URI to the PLD of that URI: $\mathsf{plds}(\mathsf{refs}(u)) \to \mathsf{pld}(\mathsf{deref}(u))$.

(ii) From the naming authority graph, use the PageRank algorithm to derive scores for each PLD.

(iii) Using the PLD ranks, derive a rank value for terms in the data, particularly terms in $\mathbf{U} \cup \mathbf{B}$ which identify entities.

7.1.1. *Extracting Source Links*

As a first step, we derive the naming authority graph from the input dataset. That is, we construct a graph which encodes links between data source PLDs, based on the implicit connections created via identifier reuse.

Given PLD identifiers $p_i, p_j \in \mathbf{U}$, we specify the naming authority matrix A as a square matrix defined as:

$$a_{i,j} = \begin{cases} 1 & \text{if } p_i \neq p_j \text{ and } p_i \text{ uses an identifier} \\ & \qquad \text{with naming authority } p_j \\ 0 & \text{otherwise} \end{cases}$$

This represents an $n \times n$ square-matrix where n is the number of PLDs in the data, and where the element at (i, j) is set to 1 if $i \neq j$ and PLD i mentions a URI which leads to a document hosted by PLD j.

As such, the naming authority matrix can be arbitrarily derived through a single scan over the entire dataset. Note that we (optionally, and in the case of later evaluation) do not consider URIs found in the predicate position of a triple, or the object position of an `rdf:type` triple, in the derivation of the naming authority graph, such that we do not want to overly inflate scores for PLDs hosting vocabularies: we are concerned that such PLDs (e.g., `w3.org`, `xmlns.org`) would receive rankings orders of magnitude higher than their peers, overly inflating the ranks of arbitrary terms appearing in that PLD; further, users will generally not be interested in results describing the domain of knowledge itself [61].

7.1.2. *Calculating Source Ranks*

Having constructed the naming authority matrix, we now can compute scores for data sources. For computing ranking scores, we perform a standard PageRank calculation over the naming authority graph: we calculate the dominant eigenvector of the naming authority graph using the Power iteration while taking into account a damping factor (see [106] for more details).

7.1.3. *Calculating Identifier Ranks*

Based on the rank values for the data sources, we now calculate the ranks for individual identifiers. The rank value of a constant $c \in \mathbf{C}$ is given as the summation of the rank values of the PLDs for the data sources in which the term occurs:

$$idrank(c) = \sum_{\mathsf{pld} \in \mathsf{plds}(\mathsf{refs}(c))} sourcerank(pld)$$

This follows the simple intuition that the more highly-ranked PLDs mentioning a given term, the higher the rank of that term should be. [31] Note again – and with similar justification as for deriving the named authority graph – we do not include URIs found in the predicate position of a triple, or the object position of an `rdf:type` triple in the above summation for our evaluation. Also note that the ranking for literals may not make much sense depending on the scenario – in any case, we currently do not require ranks for literals.

---

[31] This generic algorithm can naturally be used to propagate PLD/source-level rankings to any form of RDF artefact, including triples, predicates, classes, etc.

7.1.4. *User Evaluation*

Herein, we summarise the details of our user evaluation, where the full details are available in [61]. We conducted a study asking 10–15 participants to rate the ordering of SWSE results given for five different input keyword queries, including the evaluators own name. We found that our method produced preferable results (with statistical significance) for ranking entities than the baseline method of implementing PageRank on the RDF node-link graph (an approach which is similar to existing work such as ObjectRank [6]). Also, we found that use of the PLD-level graph and document-level graph as input for our PageRank calculations yielded roughly equivalent results for identifier ranks in our user evaluation.

7.2. *Distributed Approach*

We now discuss our approach for applying the ranking analysis over our distributed framework – we again assume that the input data are evenly distributed across the slave machines:

(i) **run**: each slave machine scans its segment of the data in parallel, and extracts PLD level links;

(ii) **gather**: the master machine gathers the PLD graph from the slave machines, aggregates the links, executes the PageRank algorithm, and derives the scores for each PLD;

(iii) **flood**: the master machine sends the PLD scores to all machines;

(iv) **run**: the slave machines calculate and summate the identifier-ranks given the PLD scores and their local view on the segment of data, outputting and sorting/uniquing $(id, pld, pldrank)$ tuples;

(v) **gather**: the master machine must now gather the identifier-ranks from the slave machines, and aggregate the scores – importantly, rank contributions for a given identifier from a given PLD must be uniqued across machines, and so the sorted $(id, pld, pldrank)$ tuples streamed from the slave machines are merge-sorted/uniqued with $pldrank$ values subsequently summated for each id.

We again performed some smaller-scale experiments to illustrate the performance advantages of distribution for our ranking methods, demonstrating again over 31.3 m statements with 1, 2, 4, and 8 machines. Table 5 presents the results. The dis-

#machines	1	2	4	8
mins	19.7	11	6.3	4.3

Table 5
Time taken for ranking of 31.3 m statements for differing numbers of machines

tribution exhibits near-linear scale with respect to the number of machines, with the most expensive tasks being run in an embarrassingly parallel fashion – the non-linear aspect ($\sim$2 min) is given by initialisation costs and the **gather** and **flood** operations dealing with the aggregation, preparation and analysis of globally-required knowledge (viz.: aggregating and calculating the PLD and identifier ranks on the master machine). We will see more detailed evaluation in the following.

7.3. *Full-Scale Evaluation*

We now discuss the results of applying the ranking procedure over our full-scale crawl (1.118b statements) over 8 machines. We extract the PLD graph from the unconsolidated data: to derive said graph as it was natively found on the Web, unaffected by the consolidation process; and apply identifier ranking over the consolidated data: to ensure that the identifier ranks were aggregated correctly for canonicalised identifiers in the consolidated data, thus representing ranks for entities derived from all sources in which all of the original referent identifiers appeared. This, however, would have minimal effect on the performance evaluation presented.

The entire process took 126.1 min.

Roughly 2 min were spent loading redirects information on the slave machines. The PLD-level graph – detailed below – was extracted in parallel in 27.9 min, with an average idle time of 35 s (2% of total task time) for machines waiting for their peers to finish. The PLD graph consisted of 566 k irreflexive links between 507 k PLDs (due to the crawling process, all PLDs enjoyed at least one in-link), with an average indegree of 1.118; on the other hand, only 704 PLDs offered outlinks (roughly speaking, those PLDs which offered RDF/XML content), with an average out-degree of 804.6. The significant variance in indegree/outdegree is attributable to those small number of PLDs offering RDF/XML content offering a large number of links to those PLDs from which we did not find RDF/XML content. Along these lines, 5.1 k links were to PLDs which themselves contained outlinks, roughly equating to an average indegree of 7.25 for those PLDs hosting RDF/XML

content.

The PageRank calculation – with ten iterations performed in memory – took just over 25 s. Table 6 presents the top 5 ranked PLDs. The PLDs `identi.ca` and `status.net` host services for users of the micro-blogging platform StatusNet, linking between user profiles exported in FOAF [32]; `status.net` had 108 unique inlinking PLDs and 999 outlinks, and `identi.ca` had 179 inlinks and 36.6 k outlinks. The `geonames.org` domain – a prominent RDF publisher of geographical data – had 167 inlinks and 9 outlinks. In fourth place, `ldodds.com` had 98 inlinks (and 52 outlinks) mostly through the `admin:generatorAgent` property from FOAF-a-matic [33] generated FOAF files. The PLD `w3.org` enjoyed 116 inlinks, and provided 658 outlinks; inlinks were diverse in nature, but included some use of core RDF/RDFS/OWL terms in non-class/property positions as discussed previously (e.g., statements of the form `?s rdfs:range rdfs:Resource` given in vocabularies), as well as, for example, links to Tim Berners-Lee's personal FOAF profile.

It is worth remembering that a higher inlink count does not necessarily imply a higher PageRank – the outdegree of those inlinking nodes is also important: indeed, it seems that at the very top of our ranking table, PLDs (such as `ldodds.com`) are highly rewarded for offering a means of exporting or publishing often simple RDF/XML on many external sites, not necessarily for hosting high-quality RDF/XML themselves, where they benefit from being linked from a single document on many low-ranked PLDs with low outdegree. More prestigious RDF publishing domains share a similar indegree from higher ranked PLDs, but the inlinking PLDs themselves have a much higher outdegree, and thus split their vote more evenly. For example, the prominent Linked Data publisher `dbpedia.org` [5] was ranked 9th with 118 inlinks – an inlink count which is comparable with some of the top ranked PLDs, but in this case the inlinks generally spread their rank more evenly: e.g., the median outdegree of the PLDs linking to `ldodds.com` was 6, whereas the median outdegree of PLDs linking to `dbpedia.org` was 19. That said, we are still satisfied by the rankings, and are encouraged by the user-evaluation from [61]. We are reluctant to amend our PLD ranking algorithm

for the purposes of punishing the aforementioned PLDs, although such practices may eventually be required to counter-act deliberate link-farming: we are not yet at that stage. Indeed, part of the problem could be attributable to the relatively small number of PLDs (and thus parties) hosting significant RDF/XML content. As Linked Data publishing grows in popularity and diversity, we would expect more PLDs with higher rates of inter-linkage, hopefully enabling more high-quality results from our link-analysis techniques.

Extracting the identifier rank tuples in parallel took 67.6 min– the slower time is associated with sorting and uniquing tuples which encode $(id, pld, pldrank)$ – with an average idle time of 86 s (2.1% of total task time). Locally aggregating and summating the ID ranks took 27.7 min. [34] Table 6 also gives the top 5 ranks for consolidated entities. The top result refers to "Sir Tim Berners-Lee"; the second result refers to "Dan Brickley", co-founder of the FOAF vocabulary and a prominent member of the Semantic Web community; the third and fifth results are commonly referenced in StatusNet exporters; the fourth result is the URL for the FOAF-a-Matic generator previously discussed.

With respect to performance, 77.3% of the computation time is spent in parallel execution, of which, on average 2.1% of time is spent idle by slave machines. In total, 28.6 min execution time is spent on the master machine, the majority of which is spent calculating PLD ranks and aggregating ID ranks.

7.4. Related Work

There have been numerous works dedicated to comparing hypertext-centric ranking for varying granularity of sources. Najork et al. [100] compared results of the HITS [85] ranking approach when performed on the level of document, host and domain granularity and found that domain granularity returned the best results: in some cases PLD-level granularity may be preferable to domain or host-level granularity because some sites like LiveJournal (which export vast amounts of user profile data in the Friend Of A Friend [FOAF] vocabulary) assign subdomains to each user, which would result in

---

[32] For example, see inter-linkage between `http://rant.feebleforce.com/dantheman/foaf` and `http://identi.ca/methoddan/foaf`

[33] `http://www.ldodds.com/foaf/foaf-a-matic`

[34] Please note that we have further optimised the algorithm presented in [61] using LRU caching of seemingly costly PLD extraction methods, duplicate detection for extracted links, and other low-level improvements; hence, we see increased performance.

#	PLD	Term
1	status.net	http://identi.ca/user/45563
2	identi.ca	http://identi.ca/user/226
3	geonames.org	http://update.status.net/
4	ldodds.com	http://www.ldodds.com/foaf/foaf-a-matic
5	w3.org	http://update.status.net/user/1#acct

Table 6
Top 5 ranked PLDs and terms

large tightly-knit communities if domains were used as naming authorities. Previous work has performed PageRank on levels other than the page level, for example at the more coarse granularity of directories, hosts and domains [82], and at a finer granularity such as logical blocks of text [20] within a page.

There have been several methods proposed to handle the task of ranking Semantic Web data.

Swoogle ranks documents using the OntoRank method, a variation on PageRank which iteratively calculates ranks for documents based on references to terms (classes and properties) defined in other documents. We generalise the method described in [39] to rank entities, and perform links-analysis on the PLD abstraction layer.

ObjectRank [6] ranks a directed labelled graph using PageRank using "authority transfer schema graphs", which requires manual weightings for the transfer of propagation through different types of links; further, the algorithm does not include consideration of the source of data, and is perhaps better suited to domain-specific ranking over verified knowledge.

We note that Falcons [27] also rank the importance of entities (what they call "objects"), but based on a logarithm of the number of documents in which the object is mentioned.

In previous work, we introduced ReConRank [74]: an initial effort to apply a PageRank-type algorithm to a graph which unifies data-level and source-level linkage. ReConRank does take data provenance into account: however, because it simultaneously operates on the object graph, it is more susceptible to spamming than the presented approach.

A recent approach for ranking Linked Data called Dataset rankING (DING) [35] – used by Sindice – holds a similar philosophy to ours: they adopt a two-layer approach consisting of an entity layer and a dataset layer. However, they also apply rankings of entities within a given dataset, using PageRank (or optionally link-counting) and unsupervised link-weighting schemes, subsequently combining dataset and local entity ranks to derive global entity ranks.

Because of the local entity ranking, their approach is theoretically more expensive and less flexible than ours, but would offer better granularity of results – less entity results with the same rank. However, as we will see later, we will be combining global entity-ranks with keyword-query specific relevance scores, which mitigates the granularity problem.

There are numerous other loosely related approaches, which we briefly mention: SemRank [3] ranks relations and paths on Semantic Web data using information-theoretic measures; AKTiveRank [1] ranks ontologies based on how well they cover specified search terms; Ontocopi [2] uses a spreading activation algorithm to locate instances in a knowledge base which are most closely related to a target instance; the SemSearch system [89] also includes relevance ranks for entities according to how well they match a user query.

7.5. *Future Directions and Open Research Questions*

Ranking in Web search engines depends on a multitude of factors, ranging from globally computed ranks to query-dependent ranks to location, preferences, and history of the searcher. Factoring additional signals into the ranking procedure is area for further research, especially in the face of complex database-like queries and results beyond the simple list of objects. For example, we have already seen that we exclude predicate and class identifiers from the ranking procedure, in order not to adversely affect our goal of ranking entities (individuals) in the data; specific modes and display criteria of the UI may require different models of ranks, providing multiple contextual ranks for identifiers in different roles – e.g., creating a distinctive ranking metric for identifiers in the role of predicates, reflecting the expectations of users given various modes of browsing.

Another possibly fruitful research topic relates to the question of finding appropriate mathematical representations of directed labelled graphs, and appropriate operations on them [60,48]. Most of the

current research in ranking RDF graphs is based around the directed graph models borrowed from hypertext ranking procedures. A bespoke mathematical model for RDF (directed, labelled, and named) graphs may lead to a different view on possible ranking algorithms.

Finally, the evaluation of link-based ranking as a indicator of trustworthiness would also be a interesting contribution; thus far, we have evaluated the approach according to user evaluation reflecting preference for the prioritisation of entity results in the UI. However, given that we also consider the source of information in our ranking, we could see if there was a co-occurrence, for example, of poorly-ranked PLDs and inconsistent data. Such a result would have impact for the reasoning component, presented next, and some discussion is provided in the respective future work section to follow.

8. Reasoning

Using the Web Ontology Language (OWL) and the RDF Schema language (RDFS), instance data (i.e., assertional data) describing individuals can be supplemented with structural data (i.e., terminological data) describing classes and properties, allowing to well-define the domain of discourse and ultimately provide machines a more sapient understanding of the RDF data. As such, numerous vocabularies have been published on the Web of Data, encouraging re-use of terms for prescribed classes and properties across sources, and providing formal RDFS/OWL descriptions thereof – for a breakdown of the most instantiated terms and vocabularies, we refer the reader to Appendix A.

We have already seen that OWL semantics can be used to automatically aggregate heterogeneous data – using `owl:sameAs` relations and, e.g., the `owl:InverseFunctionalProperty` to derive said – where the knowledge is fractured by use of discordant identifiers.[35] However, RDFS and OWL descriptions in the data can be further exploited to infer new statements based on the terminological knowledge and provide a more complete dataset for query answering, and to automatically translate

data from one conceptual model to another (where appropriate mappings exist in the data).

Example 3 In our data, we find 43 properties whose memberships can be used to infer a `foaf:page` relationship between a resource and a webpage pertaining to it. These include specialisations of the property within the FOAF namespace itself, such as `foaf:homepage`, `foaf:weblog`, etc., and specialisations of the property outside the FOAF namespace, including `mo:wikipedia`, `rail:arrivals`, `po:microsite`, `plink:rss`, `xfn:mePage`, etc. All such specialisations of the property are related to `foaf:page` (possibly indirectly) through the `rdfs:subPropertyOf` relation in their respective vocabulary. Similarly, *inverses* of `foaf:page` may also exist, where in our corpus we find that `foaf:topic` relates a webpage to a resource it pertains to. Here, `foaf:topic` is related to `foaf:page` using the built-in OWL property `owl:inverseOf`. Thus, if we know that:

```
ex:resource mo:wikipedia ex:wikipage .
mo:wikipedia rdfs:subPropertyOf foaf:page .
foaf:page owl:inverseOf foaf:topic .
```

we can *infer* through *reasoning* that:

```
ex:resource foaf:page ex:wikipage .
ex:wikipage foaf:topic ex:resource .
```

In particular, through the RDFS and OWL definitions given in the data, we infer a new fact about the entities `ex:resource` and `ex:wikipage`. (Note that reasoning can also apply over class memberships in a similar manner.) ◇

Applying RDFS and OWL reasoning at large scale has only recently become a more mature area of research [77,123,126,84,122,78], with most legacy works focussing on more expressive logics and theoretical considerations such as computational complexity and completeness, demonstrating evaluation typically over clean and domain-specific datasets, and relying largely on in-memory computation or database technologies. Most works do not discuss the application of reasoning to open Web data (we leave detailed related work to 8.4).

We thus identify the following requirements for large-scale RDFS and OWL reasoning over Web data:

- **pre-computation:** the system should pre-compute inferences to avoid the runtime expense of backward-chaining, such that could negatively impact upon response times;
- **reduced output:** the system should not produce

---

[35] Note that in this paper, we deliberately decouple consolidation and reasoning, since in future work, we hope to view the unique challenges of finding equivalent identifiers as separate from those of inferencing according to terminological data presented here.

so many inferences that it over-burdens the consumer application;

– **scalability:** the system should scale near-linearly with respect to the size of the Linked Data corpus;
– **Web tolerant:** the system should be tolerant to noisy and possibly inconsistent data on the Web;
– **domain agnostic:** the system should be applicable over data from arbitrary domains, and consider non-core Web ontologies (ontologies other than RDF(S)/OWL) as equals.

In previous work [77], we introduced the Scalable Authoritative OWL Reasoner (SAOR) system for performing large-scale materialisation using a rule-based approach over a fragment of OWL, according to the given requirements. We subsequently generalised our approach, extended our fragment to a subset of OWL 2 RL/RDF, and demonstrated distributed execution in [78]. We now briefly reintroduce important aspects from that work, focussing on discussion relevant to the SWSE usecase. *For a more thorough treatment and formalisations relating to the following discussions, we refer the interested reader to [78] and earlier work in [77] – herein, our aim is to sketch our methods, particularly by means of examples.*

8.1. *High-level Approach*

Firstly, we choose a rule-based approach since it offers greater tolerance in the inevitable event of inconsistency than Description Logics based approaches – indeed, consistency cannot be expected on the Web (cf. [76] for our discussion on reasoning issues in Linked Data). Secondly, rule-based approaches offer greater potential for scale following arguments made in [46]. Finally, many Web ontologies – although relatively lightweight and inexpressive – are not valid DL ontologies: for example, FOAF defines the data-type property `foaf:mbox_sha1sum` as inverse-functional, which is disallowed in OWL DL – in [8] and [125], the authors provided surveys of Web ontologies and showed that most are in OWL Full, albeit for largely syntactic reasons.

However, there does not exist a standard ruleset suitable for application over arbitrary Web data – we must compromise and deliberately abandon completeness, instead striving for a more pragmatic form of reasoning tailored for the unique challenges of Web reasoning. [36] In [77] we discussed the tailoring of a non-standard OWL ruleset – pD* given by

ter Horst in [117] – for application over Web data. More recently, OWL 2 has become a W3C Recommendation, and interestingly from our perspective, includes a standard rule-expressible fragment of OWL, viz.: OWL 2 RL [51]. In [73], we presented discussion on the new ruleset from the perspective of application over Web data, and showed that the ruleset is not immediately amenable to the requirements outlined, and still needs amendment for our purposes.

Herein, we follow on from discussion in [73] and implement a fragment of OWL 2 RL/RDF: we present our ruleset in Appendix B and now briefly discuss how we tailored the standard ruleset [51] for our purposes. [37]

Firstly, we do not apply rules which specifically infer what we term as "tautological statements", which refer to syntactic RDFS and OWL statements such as `rdf:type rdfs:Resource` statements, and reflexive `owl:sameAs` statements – statements which apply to every term in the graph. Given n rules which infer such statements, and t unique terms in the dataset, such rules would burden the consumer application with $t * n$ largely jejune statements – in fact, we go further and filter such statements from the output.

Secondly, we identified that separating terminological data (our T-Box) [38] that describes classes and properties from assertional data (our A-Box) that describes individuals could lead to certain optimisations in rule execution, leveraging the observation that only <1% of Linked Data is terminological, and that the terminological data is the most frequently accessed segment for OWL reasoning [77]. We used such observations to justify the identification, separation, and provision of optimised access to our T-Box, storing it in memory.

Thirdly, after initial evaluation of the system at scale encountered a puzzling deluge of inferences, we discovered that incorporating the source of data into

---

[36] For interesting discussion on the often infeasible nature

of sound and complete reasoning and alternative metrics for reasoners, please see [71].

[37] Please note that we do not consider rules which infer an inconsistent (have a `false` consequent), and consider equality reasoning separately in the consolidation component: thus we do not support any of the rules in rule group $\mathcal{R}2$ as defined in [73] – also of note, we co-incidently do not supported any rules which use the new OWL 2 constructs, since they require A-Box joins which – as we will justify herein – our system currently does not support.

[38] For example, we consider the triples `mo:wikipedia rdfs:subPropertyOf foaf:page .` and `foaf:page owl:inverseOf foaf:topic .` to be terminological.

the reasoning algorithm is of utmost importance; naïvely applying reasoning over the merge of arbitrary RDF graphs can lead to unwanted inferences whereby third parties redefine classes and properties provided by popular ontologies [77]. For example, one document [39] defines `owl:Thing` to be a member of 55 union classes, another defines nine *properties* as the domain of `rdf:type` [40], etc. We counter-act such behavior by incorporating the analysis of authoritative sources for classes and properties in the data.

We will now discuss the latter two issues in more detail, but beforehand let us treat some preliminaries used in this section. [41]

8.1.1. *Reasoning Preliminaries*

We briefly reintroduce some notions formalised in [77,78]; for brevity, in this section, we aim to give an informative and informal description of terms, and refer the interested reader to [77,78] for a more formal description thereof.

Generalised Triple   A *generalised triple* is a triple where blank-nodes and literals are allowed in all positions [51]. Herein, we assume generalised triples internally in the reasoning process and post-filter non-RDF statements from the output.

Meta-class   Informally, we consider a *meta-class* as a class specifically of classes or properties; i.e., the members of a meta-class are themselves either classes or properties. Herein, we restrict our notion of meta-classes to the set defined in RDF(S) and OWL specifications, where examples include `rdf:Property`, `rdfs:Class`, `owl:-Class`, `owl:Restriction`, `owl:DatatypeProperty`, `owl:FunctionalProperty`, etc.; `rdfs:Resource`, `rdfs:Literal`, e.g., are not meta-classes.

Meta-property   A *meta-property* is one which has a meta-class as its domain; again, we restrict our notion of meta-properties to the set defined in RDF(S) and OWL specifications, where examples include `rdfs:domain`, `rdfs:subClassOf`, `owl:has-Key`, `owl:inverseOf`, `owl:oneOf`, `owl:onProperty`,

owl:unionOf, etc.; `rdf:type`, `owl:sameAs`, `rdfs:-label`, e.g., do *not* have a meta-class as domain.

Terminological Triple   We define the set of *terminological triples* as the union of the following sets of generalised triples:
  (i) triples with `rdf:type` as predicate and a meta-class as object;
  (ii) triples with a meta-property as predicate;
  (iii) triples forming a *valid* RDF list whose head is the object of a meta-property (e.g., a list used for `owl:unionOf`, `owl:intersectionOf`, etc.).

Example 4   The triples:

```
mo:wikipedia rdfs:subPropertyOf foaf:page .
     foaf:page owl:inverseOf foaf:topic .
```

are considered terminological, whereas the following are not:

```
ex:resource mo:wikipedia ex:wikipage .
ex:resource rdf:type rdfs:Resource .
```

◇

Triple Pattern, Basic Graph Pattern   A *triple pattern* is a generalised triple where variables from the infinite set **V** are allowed in all positions. We call a set (to be read as conjunction) of triple patterns a *basic graph pattern*. Following standard notation, we prefix variables with '?'. We say that a triple is a *binding* of a triple pattern if there exists a mapping of the variables in the triple pattern to some set of RDF constants such that, subsequently, the triple pattern equals the triple; we call this mapping *variable binding*. The notion of a binding for a graph pattern follows naturally.

Terminological/Assertional Pattern   We refer to a *terminological -triple/-graph pattern* as one what can only be bound by a terminological triple or, resp., a set thereof. An *assertional pattern* is any pattern which is not terminological.

Inference Rule   We define an *inference rule r* as the pair $(\mathcal{A}nte, \mathcal{C}on)$, where the *antecedent* $\mathcal{A}nte$ and the *consequent* $\mathcal{C}on$ are basic graph patterns [110], all variables in $\mathcal{C}on$ are contained in $\mathcal{A}nte$, and if $\mathcal{A}nte$ is non-empty, at least one variable must co-exist in $\mathcal{A}nte$ and $\mathcal{C}on$. Every unique match – in the union of the input and inferred data – for the graph pattern $\mathcal{A}nte$ leads to the inference of $\mathcal{C}on$ with the respective variable bindings. Rules with empty $\mathcal{A}nte$ can

---

[39] http://lsdis.cs.uga.edu/~oldham/ontology/wsag/wsag.owl
[40] http://www.eiao.net/rdf/1.0
[41] Please note that we largely re-use the definitions provided in [77], which are required here for further discussion of our reasoning approach.

be used to model axiomatic statements which hold for every graph. Herein, we use SPARQL-like syntax to represent graph-patterns, and will typically formally write inference rules as $\mathcal{A}nte \Rightarrow \mathcal{C}on$.

Example 5 The OWL 2 RL/RDF rule **prp-spo1** (Table B.3) supports inferences for `rdfs:subPropertyOf` with the following rule:

$?p_1$ `rdfs:subPropertyOf` $?p_2$. $?x$ $?p_1$ $?y$. $\Rightarrow$ $?x$ $?p_2$ $?y$.

where the antecedent $\mathcal{A}nte$ consists of the two patterns on the left side of $\Rightarrow$, and the consequent $\mathcal{C}on$ consists of the pattern on the right side of $\Rightarrow$. This can be read as an IF–THEN condition, where if data matching the patterns on the left are found, the respective bindings are used to infer the respective pattern on the right. ◇

8.1.2. *Separating Terminological Data*

Given the above preliminaries, we can now define our notion of a $\mathcal{T}$-*split inference rule*, whose antecedent is split into two: one part which can only be matched by terminological data, and one which can be matched by assertional data.

Definition 1 ($\mathcal{T}$-split inference rule) *Let r be the rule $(\mathcal{A}nte, \mathcal{C}on)$. We define the $\mathcal{T}$-split version of r as the triple $(\mathcal{A}nte_\mathcal{T}, \mathcal{A}nte_\mathcal{G}, \mathcal{C}on)$, where $\mathcal{A}nte_\mathcal{T}$ is the set of terminological patterns in $\mathcal{A}nte$ and $\mathcal{A}nte_\mathcal{G}$ is given as all remaining antecedent patterns: $\mathcal{A}nte \setminus \mathcal{A}nte_\mathcal{T}$.*

We generally write $(\mathcal{A}nte_\mathcal{T}, \mathcal{A}nte_\mathcal{G}, \mathcal{C}on)$ as $\underline{\mathcal{A}nte_\mathcal{T}}\mathcal{A}nte_\mathcal{G} \Rightarrow \mathcal{C}on$, identifying terminological patterns by underlining.

Example 6 Take the rule **prp-dom** (Table B.3):

$\underline{?p\ \texttt{rdfs:domain}\ ?c}$. $?x\ ?p\ ?y$. $\Rightarrow ?y\ \texttt{rdf:type}\ ?c$.

The terminological (underlined) pattern can only be matched by triples who have `rdfs:domain` – a meta-property – as predicate, and thus must be terminological. The second pattern can be matched by non-terminological triples and so is considered an assertional pattern. ◇

Given the general notion of terminological data, we can constrain our *T-Box* (Terminological-Box) to be the set of terminological triples present in our input data that match a terminological pattern in our rules – intuitively, our T-Box represents the descriptions of classes and properties required in our ruleset; e.g., if our ruleset is RDFS, we do not include OWL terminological triples in our T-Box. We define our *closed T-Box* – denoted $\mathcal{T}$ – as the set of terminological triples derived from the input, and the

result of exhaustively applying rules with no assertional patterns (axiomatic and 'schema-level' rules) up to a *least fixpoint* [78]. Again, our 'A-Box' is the set of all statements, including the *T-Box* and inferred statements.

When applying a $\mathcal{T}$-*split inference rule*, $\mathcal{A}nte_\mathcal{T}$ is strictly only matched by our closed T-Box. Thus, in our reasoning system, we have a well defined distinction between T-Box and A-Box information, reflected in the definition of our rules, and the application of rules over the T-Box split data. This decoupling of T-Box and A-Box allows for incorporating the following optimisations:

(i) knowing that the T-Box is relatively small and is the most frequently accessed segment of crawling – e.g., all of the rules in Appendix B require terminological knowledge – we can store the T-Box in an optimised index;

(ii) we can identify optimised $\mathcal{T}$-split rules as those with low assertional-arity – namely, rules which do not require joins over a large A-Box can be performed in an optimal and scalable manner;

(iii) we will later use the separation of the T-Box as an integral part of our distributed approach.

With respect to the first possible optimisation, at the moment we store the entire T-Box in memory, but on-disk indices can be employed as necessary. [42] We will refer in Section 8.2 to the third optimisation avenue.

With respect to the second optimisation, in [77], we showed that rules involving more than one assertional pattern (i.e., requiring a join operation over the large A-Box) were in practice difficult to compute at the necessary scale. Thus, we categorised rules according to the *assertional arity* of their antecedent; i.e., the number of assertional patterns in the antecedent. In [73], we performed similar categorisation of OWL 2 RL/RDF rules.

In Appendix B, we provide the ruleset we apply over our Linked Data corpus: the rules are categorised according to the arity of assertional/terminological antecedent patterns, showing rules with no antecedent (axiomatic triple rules) in Table B.1 (we denote this subset of rules $\mathcal{R}^\emptyset$), rules with *only* terminological patterns answerable entirely from our T-Box in Table B.2 ($\mathcal{R}^{\mathcal{T}\emptyset}$), and rules

---

[42] We would expect an on-disk index with heavy caching to work well given the distribution of classes and properties in the data – i.e., we would expect a high cache hit rate.

with *some* terminological patterns and *precisely one* assertional pattern in Table B.3 ($\mathcal{R}^{\mathcal{TG}^1}$).[43]

Not shown are rules with multiple assertional patterns;[44] we currently only apply reasoning over rules with less than one assertional pattern using an optimised approach – provided at a high-level in Algorithm 2 – which consists mainly of two scans as follows:

(i) to commence, we apply rules with no antecedent (Table B.1), inferring axiomatic statements [Lines 1–2];

(ii) we then run the first scan of the data, identifying terminological knowledge found in the data, and separating and indexing the data in our in-memory T-Box [Lines 3–10];

(iii) using this T-Box, we apply rules which only require T-Box knowledge (Table B.2), deriving the closed T-Box [Lines 11–14];

(iv) the second scan sequentially joins individual A-Box statements with the static in-memory T-Box, including recursively inferred statements (Table B.3) [Lines 15–28].

Note that Algorithm 2 assumes that the input ruleset does not contain rules with multiple assertional patterns. The *applyRules* method for the T-Box can be considered equivalent to standard semi-naïve evaluation. We omit from the algorithm some optimisations, such as a fixed-size LRU cache which removes duplicate A-Box inferences appearing in a given locality – although our data is unsorted, it is grouped by document and we thus expect locality in the repetition of many inferences.

We call the above reasoning approach "partial indexing" in that only a subset of the data need be indexed: in the above version, rules without A-Box joins are not supported so we need only index the T-Box. In [78], we give a more general partial-indexing algorithm which supports A-Box joins: we showed the approach to be sound with respect to standard exhaustive rule-application (e.g., semi-naïve evaluation) and also complete with the condition that a

---

Algorithm 2 Algorithm for reasoning
Require: *INPUT FILE*: $\mathcal{G}$, *RULES*: $\mathcal{R}$
1: $AXI \leftarrow$ axiomatic triples
2: **output** (AXI)
3: $\mathcal{T} \leftarrow \{\}$
4: **for** $t \in \mathcal{G} \cup AXI$ **do**
5:    **for** $r \in \mathcal{R}$ **do**
6:       **if** $r.\mathcal{A}nte^\mathcal{T}$ needs t **then**
7:          $\mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
8:       **end if**
9:    **end for**
10: **end for**
11: $\mathcal{R}^{\mathcal{T}\emptyset} \leftarrow \{r \in \mathcal{R} \mid r.\mathcal{A}nte^\mathcal{T} \neq \emptyset, r.\mathcal{A}nte^\mathcal{G} = \emptyset\}$
12: $\mathcal{T}^{new} \leftarrow applyRules\,(\mathcal{T}, \mathcal{R}^\mathcal{T})$
13: **output** ($\mathcal{T}^{new}$)
14: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}^{new}$
15: $\mathcal{R}^\mathcal{G} \leftarrow \{r \in \mathcal{R} \mid r.\mathcal{A}nte^\mathcal{G} \neq \emptyset\}$
16: **for** $t \in \mathcal{G} \cup AXI \cup \mathcal{T}^{new}$ **do**
17:    $\mathcal{G}_t \leftarrow \{t\}$
18:    **for** new triple $t_n \in \mathcal{G}_t$ **do**
19:       **for** $r \in \mathcal{R}^\mathcal{G}$ **do**
20:          **if** $\exists$ binding $b_\mathcal{G} \mid b_\mathcal{G}(r.\mathcal{A}nte^\mathcal{G}) = t_n$ **then**
21:             **for** $\forall b_\mathcal{T} \mid b_\mathcal{T}(b_\mathcal{G}(r.\mathcal{A}nte^\mathcal{T})) \subseteq \mathcal{T}$ **do**
22:                $\mathcal{G}_t \leftarrow \mathcal{G}_t \cup b_\mathcal{T}(b_\mathcal{G}(r.\mathcal{C}on))$
23:             **end for**
24:          **end if**
25:       **end for**
26:    **end for**
27:    **output** ($\mathcal{G}_t \setminus \{t\}$)
28: **end for**

---

rule requiring assertional knowledge does not infer terminological triples (our T-Box is static and will not be updated). In general, the partial-indexing approach is suitable when only a small subset of the data need be indexed: the more data that needs to be indexed, the more inefficient the approach becomes – e.g., standard semi-naïve evaluation over a full index would perform better (we refer the interested reader to [78] for more discussion).

In [77], we demonstrated scalable means of processing A-Box joins using static join-indices: however, these indices performed poorly due to recursion of inferences – they efficiently generated many inferences, but took a long time to reach a fixpoint. Further, we then showed that 99.7% of inferences occurred through pD* rules with zero or one assertional patterns;[45] we thus proposed that such rules

---

[43] Briefly to explain our notation for rule categorisation: $\mathcal{R}^\emptyset$ refers to rules with no antecedent; e.g., $\mathcal{R}^\mathcal{G}$ refer to rules with *some* assertional patterns; e.g., $\mathcal{R}^{\mathcal{T}\emptyset}$ refers to rules with *only* terminological patterns; finally, we may denote a constant arity of patterns where, e.g., $\mathcal{R}^{\mathcal{TG}^1}$ refers to rules with some terminological patterns and *exactly one* assertional pattern.
[44] Coincidentally, none of our rules have only assertional patterns and no terminological patterns ($\mathcal{R}^{\emptyset\mathcal{G}}$) – such rules in OWL 2 RL/RDF are concerned with `owl:sameAs` inferencing for which we presented a bespoke approach in the previous section.

[45] Please note that this was over a dataset of 147m statements, and used a similar canonicalisation approach to equality which otherwise would have produced quadratic infer-

cover the majority of inferencing mandated by the lightweight Web vocabularies whose terms are commonly used on the Web.

Avoiding expensive intra-A-Box joins, we instead performing reasoning at roughly the cost of two sequential scans of the input data, and the cost of writing the inferred statements to disk [77].

8.1.3. *Template Rule Optimisations*

One important aspect in the efficiency of the partial indexing approach is how the T-Box is indexed: the T-Box will have to serve – in our scenario – billions of sequential lookups. Originally in [77], we described a T-Box indexing approach optimised specifically for the pD*-inspired ruleset at hand, including hard-coded meta-property links between classes and properties, and hard-coded encoding of meta-class membership. In [78], we looked to generalise the T-Box indexing: the main intuition behind the optimisation was to pre-bind the T-Box patterns in the rules before accessing the A-Box. This follows the precedent of *template rules* as discussed in the RIF working group [113] and used in the DLEJena [93] system, where terminological patterns in the rules are substituted by terminological data, producing a set of rules which by themselves encode the T-Box.

Example 7 Given the T-Box

$$\mathcal{T} = \{ \texttt{foaf:homepage rdfs:subPropertyOf}$$
$$\texttt{foaf:isPrimaryTopicOf . foaf:homepage}$$
$$\texttt{rdfs:subPropertyOf foaf:page . } \}$$

and a rule r as follows

$$\frac{\texttt{?p}_1 \texttt{ rdfs:subPropertyOf ?p}_2 \texttt{ . ?x ?p}_1 \texttt{ ?y .}}{\Rightarrow \texttt{?x ?p}_2 \texttt{ ?y .}}$$

Then we can produce the following *templated rules*:

$\{$?x `foaf:homepage` ?y . $\Rightarrow$?x `foaf:isPrimaryTopicOf` ?y . ;
?x `foaf:homepage` ?y . $\Rightarrow$?x `foaf:page` ?y . $\}$

$\diamond$

Since we restrict our ruleset to exclude rules with multiple assertional patterns, the set of template rules we produce all contain a single antecedent pattern which should be efficiently applicable. However, as we showed in [78], we need to amend our partial-indexing algorithm for templated rules: the templating process may create a prohibitively large set of rules to apply in the brute force manner of Algo-

rithm 2 (Line 15). Thus, we applied optimisations tailored for the templated rules.

Firstly we merged rules with compatible antecedents (antecedents which can be made equivalent by a variable rewriting function [78]); we give a brief example.

Example 8 In the previous example, we would *merge* the two templated rules to form the new rule:

?x `foaf:homepage` ?y .
$\Rightarrow$?x `foaf:isPrimaryTopicOf` ?y . ?x `foaf:page` ?y .

$\diamond$

However, even with merging, we may still have too many rules for brute force application.

Thus, we proposed a rule index which takes a triple and returns rules which have a pattern which could be bound by that triple. The index must perform $2^3 = 8$ lookups for each possible triple pattern. For example, given a triple:

`ex:aidan foaf:homepage ex:index.html .`

the index would return the previous merged rule for the pattern (? `foaf:homepage` ?).

One rule application may lead to another. Thus, we also included the notion of a graph in our rule index; our index stores a linked list of rules where one rule links to dependant rules. This allows us to avoid repetitive lookups on our rule index, instead following the rule graph to find the recursive rules to fire.

Example 9 Take the rules

?x `foaf:homepage` ?y .
$\Rightarrow$?x `foaf:isPrimaryTopicOf` ?y . ?x `foaf:page` ?y .

and

?x `foaf:isPrimaryTopicOf` ?y .
$\Rightarrow$?y `foaf:primaryTopicOf` ?x .

When the first rule fires, the second rule will also fire. Thus, we encode a link from the first rule to the second. $\diamond$

We additionally label the links from one rule to another: given that a rule may have multiple consequent patterns – esp. as the result of merging – we label the dependency link with the index of the consequent pattern(s) that given the dependency. During the reasoning process, we then know which rule is dependent on which particular inferred statement.

Finally, we also investigated one more template-rule optimisation in [78]: we distinguish *strong dependencies* between rules where the inference of one rule *will* necessarily lead to the inference of another. This is commonly the case for rules with only one an-

---

ences. This was a surprising result given the quadratic nature, for example, of transitive reasoning, which theoretically should make full materialisation infeasible – roughly 300k transitive inferences were made.

tecedent pattern (rules which do not require A-Box joins). Thus, we can actually "saturate" the rules according to strong dependencies and prune links from the graph.

Example 10 Take the same two rules as the previous example; we can saturate the first rule to create:

$$?x \text{ foaf:homepage } ?y .$$
$$\Rightarrow ?x \text{ foaf:isPrimaryTopicOf } ?y . ?x \text{ foaf:page } ?y .$$
$$?y \text{ foaf:primaryTopicOf } ?x .$$

Note that the second rule remains in the index, but the first rule is no longer dependent on it, and so we prune the link from the first rule to the second. ⋄

Although the saturisation technique reduces the number of rule applications necessary during reasoning, we found that applying saturated rules produced more initial duplicates which put more load on our LRU cache: for example, consider if the triples

```
ex:aidan foaf:isPrimaryTopicOf ex:index.html .

ex:index.html foaf:primaryTopicOf ex:aidan .
```

reside in the cache before the above rule is applied to the statement

```
ex:aidan foaf:homepage ex:index.html .
```

Without saturated rules, the second rule would not fire and the duplicate `foaf:primaryTopicOf` triple would not be produced. We found this particularly expensive for saturated domain rules which inferred, for example, membership of `foaf:Person` and all of its subclasses: generally, the class memberships would already have been inferred by other means.

For reference, we give the partial indexing approach with the template rule optimisations in Algorithm 3. Note that the algorithm is identical to Algorithm 2 until the closed T-Box is derived; also, we omit the saturisation process for reasons discussed, and we again use an LRU cache for duplicates as before (not shown in the algorithm).

8.1.4. *Authoritative Reasoning*

In order to curtail the possible side-effects of open Web data publishing, we include the source of data in inferencing. Our methods are based on the view that a publisher instantiating a vocabulary's term (class/property) thereby accepts the inferencing mandated by that vocabulary and recursively referenced vocabularies for that term. Thus, once a publisher instantiates a class or property from a vocabulary, only that vocabulary and its references should influence what inferences are possible through that instantiation.

Algorithm 3 Reasoning with templated rules

Require: *INPUT FILE*: $\mathcal{G}$, *RULES*: $\mathcal{R}$
1: **derive** AXI, $\mathcal{T}$, $\mathcal{T}^{new}$ as in Algorithm 2
2: $\mathcal{R}^{\emptyset \mathcal{G}} \leftarrow \{r \in \mathcal{R} \mid r.Ante^{\mathcal{G}} \neq \emptyset, r.Ante^{\mathcal{T}} = \emptyset\}$
3: $\mathcal{R}^{\mathcal{T}\mathcal{G}} \leftarrow \{r \in \mathcal{R} \mid r.Ante^{\mathcal{G}} \neq \emptyset, r.Ante^{\mathcal{T}} \neq \emptyset\}$
4: $\mathcal{R}^{\mathcal{T}\mathcal{G}\prime} = template(\mathcal{R}^{\mathcal{T}\mathcal{G}}, \mathcal{T})$
5: $\mathcal{R}^{\mathcal{T}\mathcal{G}\prime\prime} = merge(\mathcal{R}^{\mathcal{T}\mathcal{G}\prime})$
6: $\mathcal{R}^{index} = linkedRuleIndex(\mathcal{R}^{\mathcal{T}\mathcal{G}\prime\prime} \cup \mathcal{R}^{\emptyset \mathcal{G}})$
7: **for** $t \in \mathcal{G} \cup AXI \cup \mathcal{T}^{new}$ **do**
8: $\quad \mathcal{R}\mathcal{G}_t \leftarrow \{(r, t) \mid r \in \mathcal{R}^{index}.lookup(t)\}$
9: $\quad$ **for** new rule/triple pair $(r_j, t_k) \in \mathcal{R}\mathcal{G}_t$ **do**
10: $\quad\quad$ **for** $t_m \in r_j.apply(t_k)$ **do**
11: $\quad\quad\quad \mathcal{R}\mathcal{G}_t \leftarrow \mathcal{R}\mathcal{G}_t \cup \{(r_l, t_m) \mid r_l \in r_j.link(t_k)\}$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad$ **output** (unique triples in $\mathcal{R}\mathcal{G}_t$)
15: **end for**

In order to do so, we again leverage Linked Data best-practices: in this case, particularly **LDP2** and **LDP3** – use HTTP URIs and offer an entity description at the dereferenced document. Similarly to the ranking procedure, we follow the intuition that the document returned by resolving a URI is authoritative for that URI, and the prerogative of that document on that URI should have special consideration. More specifically – and recalling the dereferencing function deref and HTTP lookup function get from Section 4 – we can define the authoritative function which gives the set of terms for which a graph at a given Web location (source) speaks authoritatively:

$$auth : \mathbf{S} \to 2^{\mathbf{C}}$$
$$s \mapsto \{b \in \mathbf{B} \mid b \in t \in \mathsf{get}(u)\}$$
$$\cup \{u \in \mathbf{U} \mid \mathsf{deref}(u) = s\}$$

where a Web document is authoritative for the blank nodes it contains and the URIs which dereference to it; e.g., the FOAF vocabulary is authoritative for terms in its namespace. Note that no document is authoritative for literals.

Now we wish to perform reasoning over terms as mandated in the respective authoritative document. For example, we want to perform inferencing over data instantiating FOAF classes and properties as mandated by the FOAF vocabulary, and not let third-party vocabularies (not recursively referenced by FOAF) affect said inferencing. To negate the effects of non-authoritative axioms on reasoning over Web data, we apply restrictions to the $\mathcal{T}$-split application of rules in $\mathcal{R}^{\mathcal{T}\mathcal{G}}$ (rules with non-empty

$\mathcal{A}nte^{\mathcal{T}}$ and $\mathcal{A}nte^{\mathcal{G}}$), whereby the document serving the T-Box data bound by $\mathcal{A}nte^{\mathcal{T}}$ must be authoritative for at least one term bound by a variable which appears in both $\mathcal{A}nte^{\mathcal{T}}$ and $\mathcal{A}nte^{\mathcal{G}}$: that is to say, the document serving the terminological data must speak authoritatively for at least one term in the assertional data being reasoned over. [46]

Example 11 Take the OWL 2 RL/RDF rule **cax-sco**:

```
?c1 rdfs:subClassOf ?c2 . ?x a ?c1 . ⇒ ?x a ?c2 .
```

where we use `a` as a shortcut for `rdf:type`. Here, ?c1 is the only variable that appears in both $\mathcal{A}nte^{\mathcal{T}}$ and $\mathcal{A}nte^{\mathcal{G}}$. Take an A-Box triple

```
ex:me a foaf:Person .
```

Here, ?c1 is bound by `foaf:Person`, and `deref(foaf:Person) = foaf:`, the FOAF spec. Now, any document serving a binding for

```
foaf:Person rdfs:subClassOf ?c2 .
```

must be authoritative for the term `foaf:Person`: the triple must come from the FOAF spec. Note that ?c2 need not be authoritatively bound; e.g., FOAF can *extend* any classes they like. ◇

We do not consider authority for rules with empty $\mathcal{A}nte^{\mathcal{T}}$ or $\mathcal{A}nte^{\mathcal{G}}$. Also, we consider reasoned T-Box triples as non-authoritative, thus *effectively* excluding these triples from the T-Box: in Table B.4, we give an informal indication as to how this affects completeness, showing how the inferences mandated by the inferred T-Box axioms could instead be supported by recursive application of rules in $\mathcal{R}^{\mathcal{TG}}$ – we claim that we would miss some `owl:Thing` membership inferences (which we in any case filter from the output) and some inferences based on some-values-from and all-values-from axioms. [47]

We refer the reader to [77,72] for more detail on authoritative reasoning, including analysis of the explosion of inferences encountered without the notion of authority. Note that the previous two examples from documents in Footnotes 39 & 40 are ignored by the authoritative reasoning. Since authoritativeness is on a T-Box level, we can apply the above ad-

ditional restriction to our templating function when binding the terminological patterns of the rules to derive a set of *authoritative template rules*.

8.1.5. *Local Evaluation*

In [78], we evaluated the various template rule optimisations presented in Section 8.1.3 for authoritative reasoning over the same raw dataset crawled for the purposes of this paper on one machine: we found that the initial approach (no template rules) presented in Algorithm 2 took 118.4 h; we estimated that brute force application of the template rules would take 19 years; with indexing of linked rules, application of the template rules took 22.1 h; including the merge function reduced the time to 17.7 h; saturating the rules *increased* the runtime to 19.5 h. Thus, the best approach – omitting saturation – took 15% of the time of the naïve approach presented in Algorithm 2.

8.2. *Distributed Approach*

We now show how the above techniques can be applied to perform authoritative reasoning wrt. our ruleset over our distributed framework. Again, assuming that the data is distributed (preferably evenly, and possibly in an arbitrary fashion) over the slave machines, we can apply the following distributed approach:

(i) **run**: each slave machine scans its segment of the knowledge-base in parallel, extracting terminological statements; each machine also annotates the terminological statements with authoritative values, and attempts to 'reduce' the T-Box by removing irrelevant statements – e.g., non-authoritative axioms or irrelevant RDF collections;

(ii) **gather**: the master machine firstly executes all axiomatic rules locally; the master machine then gathers all terminological statements found by the slave machines in the previous step, which then:
  – indexes the terminological statements in memory;
  – runs the 'T-Box only' rules, outputting results locally;
  – creates the authoritative templated rules, merging, indexing and linking them;

(iii) **flood**: the master machine sends the authoritative template rule index to all machines;

---

[46] Currently, we only consider the case where the T-Box segment of the antecedent is matched by one document. For OWL 2 RL/RDF rules in $\mathcal{R}^{\mathcal{TG}}$, this is not so restrictive: these rules may contain multiple terminological patterns, but these always correspond to an 'atomic axiom', which the OWL abstract syntax restricts to be bound in one document and use local blank-nodes [77].

[47] Similar, more formal analysis is given in [98] for RDFS.

#machines	1	2	4	8
mins	46.7	25.2	14.5	8.7

Table 7

Time taken for reasoning of 31.3 m statements for differing numbers of machines

(iv) **run**: the slave machines perform application of the authoritative template rules over their segment of the knowledge-base (A-Box), and output inferences locally.

Thus, our notion of a separate T-Box, and restriction of our rules to those with zero or one assertional patterns allows us to flood the template rule index – which encodes the small T-Box – to all machines, and avoids the need to compute potentially expensive A-Box joins across machines; that is to say, given the 'T-Box' as global knowledge, the slave machines can perform reasoning over the large A-Box in an embarrassingly parallel fashion.

As before, in order to evaluate the benefit of distributing the reasoning process over multiple machines, in Table 7 we present the time taken for reasoning over 31.3 m distributed across 1, 2, 4 and 8 machines. The demonstrated scale is near-linear, with the common aggregation of T-Box information causing the non-linear factor. In total, 28.9 m inferences are produced (92% increase), with 207 k T-Box triples (0.66%) creating 118 k template rules, which are merged to 70 k. We will see more detailed evaluation in the following section.

8.3. *Full-Scale Evaluation*

Continuing the thread of the paper, we applied the above reasoning approach over the consolidated data (1.113b statements) generated in Section 6. Note that we presented similar evaluation in [78], but we adapt the evaluation slightly for the purposes of SWSE: (i) we output quadruples from the reasoning process, where the context encodes a URI which refers to the rule *directly* responsible for the inference; [48] (ii) we extract the T-Box information from the raw data, but we apply reasoning over the consolidated data: we want to ensure that `owl:sameAs` statements do not affect terminological knowledge (e.g., again see Footnote 24) – such caution is necessary, but has little effect on the performance evaluation presented.

The entire process took 235.3 min.

Extraction of the T-Box took 66 min, with an average idle time of 17.3 min (26.2%); one machine took 18 minutes longer than the next slowest, but extracted 27.7% of the total T-Box data (more than twice the average): this was due to one document [49] which contained 360 k triples and from which that slave machine extracted 180 k T-Box statements. In total, the T-Box consisted of 1.06 m statements after remote reduction and removal of non-authoritative data (0.1% of total data – judging from 0.66% for 31 m dataset, it seems that the ratio of T-Box data shrinks as the Web crawl size increases, with the "long tail" containing mainly A-Box data).

Aggregation of the T-Box on the master machine – including application of T-Box only rules, loading the T-Box into memory, and performing authoritative analysis – took 11.2 min (4.8% of total reasoning time). Reasoning over the T-Box on the master machine produced 2.64 m statements. In total, 301 k templated rules were created: Table 8 gives the breakdown of templated rules for each original rule, where notably 70.3% of rules are produced through `cax-sco` which supports `rdfs:subClassOf`. The total number of templated rules was subsequently reduced to 216 k (reduced to 71.8%) by merging. The rule index contained 1.15 m labelled links between dependant rules.

Parallel application of the A-Box rules – and materialisation of the inferred data – took roughly ∼157.6 min, with an average idle time of ∼3.6 min (∼2.3%). Reasoning over the A-Box produced 1.558 g raw inferred quads (140% increase in data), which was filtered down – removing non-RDF generalised statements and tautogolical statements – to 1.138 g output inferences; through subsequent post-processing in the indexing phase described in the next section, we found 941 m unique and novel (not previously asserted) inferred *triples* (∼82.7% of raw inferred quad count – 84.6% increase from original data).

Overall, 95% of total reasoning time is spent in embarrasingly parallel execution; however, on average 9.3% of this time was spent idle by the slave machines due to the one slow machine extracting the T-Box. In total, 11.7 min was spent on the master machine, mostly aggregating the T-Box.

---

[48] Currently, we do not properly support 'inference tracking' and merely use rule-encoding contexts as a placeholder.

[49] `http://www.ebusiness-unibw.org/ontologies/eclass/5.1.4/eclass_514en.owl`

rule	templated rules generated
prp-dom	13,875
prp-rng	13,469
prp-symp	99
prp-spo1	8,600
prp-eqp1	100
prp-eqp2	85
prp-inv1	703
prp-inv2	694
cls-int2	313
cls-uni	10,252
cls-svf2	2
cls-hv1	13
cls-hv2	11
cax-sco	211,519
cax-eqc1	22,895
cax-eqc2	18,544
total	301,075

Table 8
Total templated rules generated for each original rule.

8.4. *Related Work*

In this section we presented the extension of our earlier presented work on SAOR [77] towards larger coverage of OWL 2 RL/RDF and parallel distribution of inference. Similarly, other works have been presented that tackle large-scale reasoning through parallelisation: Urbani et. al. [123] presented a MapReduce approach to RDFS reasoning in a cluster of commodity hardware similar to ourselves., identifying that RDFS rules have, at most, one assertional pattern in the antecedent, discussing how this enables efficient MapReduce support. Published at the same venue, Weaver and Hendler [126] also leverage a separation of terminological data to enable distribution of RDFS reasoning. Although the above works have demonstrated scale in the order of hundreds of millions and a billion triples respectively, their experiments were focussed on scalability issues and not on counter-acting poor data quality on the Web. Weaver et al. [126] focus on evaluation over synthetic LUBM data; Urbani et al. [123] apply RDFS over ∼865 m Linked Data triples, but produce 30 g inferences which is against our requirement of reduced output – they do not consider authoritative reasoning or source of data, although they note in their performance-centric paper that an algorithm similar to that in SAOR could be added.

A number of systems have tackled the distributed computation of A-Box joins. The Marvin [105] system uses a "divide-conquer-swap" technique for performing joins in a distributed setting, avoiding hash-based data partitioning to avoid problems with data-skew inherent in RDF [87]. Following on from [123], Urbani et al. introduced the WebPie system [122], applying incomplete but comprehensive pD* to 100 g LUBM triples, discussing rule-specific optimisations for performing pD* "A-Box join rules" over MapReduce. Although these works are certainly a large step in the right direction, we feel that applying such rules over 1 g triples of arbitrary Linked Data is still an open research question given our previous experiences documented in [77]: for example, applying full and quadratic materialisation of transitive inferences over the A-Box may become infeasible (if not now, then almost certainly in the future).

With respect to template rules, DLEJena [93] uses the Pellet DL reasoner for T-Box level reasoning, and uses the results to template rules for the Jena rule engine; they only demonstrate methods on synthetic datasets up to a scale of ∼1 m triples. We take a somewhat different approach, discussing template rules in the context of the partial indexing technique, giving a lightweight bottom-up approach to optimisations.

A viable alternative approach to Web reasoning employed by Sindice [33] – the relation to which is discussed in depth in [77] – is to consider a small "per-document" closure which quarantines reasoning to a given document and the related documents it either implicitly or explicitly imports. Although such an approach misses inferences made through the merge of documents – for example transitivity across sources – so does ours given our current limitation of not computing A-Box joins.

Falcons employ a similar approach to our authoritative analysis to do reasoning over class hierarchies, but only include custom support of `rdfs:subClassOf` and `owl:equivalentClass`, as opposed to our general framework for authoritative reasoning over arbitrary $\mathcal{T}$-split rules [26].

8.5. *Future Directions and Open Research Questions*

In order to make reasoning over arbitrary Linked Data feasible – both in terms of scale and usefulness of the inferred data – we currently renounce a lot of inferences theoretically warranted by the OWL semantics. We would thus like to extend our approach to cover a more complete fragment of OWL 2 RL/RDF, while still meeting the requirements outlined. This would include, for example,

a cost-benefit analysis of rules which require A-Box joins for reasoning over Web data. Similarly, since we perform partial-materialisation – and indeed since full OWL 2 RL/RDF materialisation over Linked Data will probably not be feasible – we would like to investigate some backward-chaining (runtime) approaches which complement a partial-materialisation strategy. Naturally, such extensions would push the boundaries for scalability and performance even further than our current, cautious approach.

Relatedly, we do not currently consider the combination of ranking into the reasoning process, where ranking is currently applied before (and independently of) reasoning. In more exploratory works [72,16], we have extended our approach to include some notion of ranking, incorporating the ranks of triples and their contexts (using a variation of the algorithm in Section 7) into inference, and investigating the applicability of ranking as a quantification of the trustworthiness of inferences. We use these ranks to repair detected *inconsistencies*: contradictions present in the corpus. In particular, we found ∼301 k inconsistencies after reasoning, although ∼294 k of these were given by invalid datatypes, with ∼7 k members of disjoint classes. Along similar lines, inclusion of ranking could be used to facilitate top-k materialisation: for example, only materialising triples relating to popularly instantiated classes and properties. Integration of these methods into the SWSE pipeline is the subject of future work.

9. Indexing

Having now reached the end of the discussion on the data acquisition, analysis and enhancement components, we look at creating an index necessary to allow users perform top-k keyword lookups and focus lookups (see Section 2.1) over our ameliorated Linked Data crawl, which has been consolidated and includes the results of the reasoning process. Note that in previous work, we demonstrated a distributed system for allowing SPARQL querying over billions of triples [62]; however, we deem SPARQL out-of-scope for this work, focussing instead on a lightweight, bespoke index optimised for the requirements of the user interface.

To allow for speedy access to the RDF data we employ a set of indices: we employ an inverted index for keyword lookups based on RDF literals (text),

and a sparse index for lookups of structured data. Inverted indices are standard for keyword searches in information retrieval. We employ a sparse index because it represents a good trade-off between lookup performance, scalability and simplicity [62]. Following our previous techniques aiming at application over static datasets, our index structure does not support updates and is instead read-optimised [62]; in principle, we could employ any sufficiently optimised implementation of an index structure that offers prefix lookup capabilities on keys.

9.1. Inverted Index

The inverted index is required to formulate the direct response to a user keyword query, including information for result-snippets to be rendered by the UI (again, see Figure 1). Our inverted index for keyword search is based on the Lucene [64] [50] engine, and is constructed in the following way during a scan of the data:

– for each entity in the RDF graph, construct a Lucene document with the union of all string literals related by some property to the RDF subject;
– to each entity, add fields containing the identifiers (URI(s) or blank node given by the subject and/or `owl:sameAs` values), labels (`rdfs:label`, `dc:title`, etc.), descriptions (`rdfs:comment`, `dc:description`, etc.), classes (objects of `rdf:type` triples), and possibly other metadata such as image URIs if required to create keyword result snippets;
– in addition, globally computed ranks are added for each identifier.

For lookups, we specify a set of keyword terms for which matching identifiers should be returned, and in addition the desired slice of the result set (e.g., `result 1 to 10`). Following standard information retrieval techniques, Lucene combines the globally computed ranks with query-dependent *TF*IDF* (query-relevance) ranks and selects the slice of results to be returned. We additionally associate entity labels with a fixed "boost" score, giving label-term matches higher relevance, here assuming that many keyword-searches will be for entity labels (e.g., `galway`, `dan brickley`, etc.). For this, we use Lucene's off-the-shelf similarity engine [64] which can be sketched as follows.

The additional non-textual metadata stored in Lucene allows for result snippets to be directly created from the Lucene results, without requiring access to the structured index: from the contents of the additional fields we generate an RDF graph and return the results to higher layers for generating the results page.

9.2. Structured Index

The structured index is implemented to give all information relating to a given entity (e.g., focus view; again see Figure 2). The structured index is implemented using "sparse indices" [62], where a blocked and sorted ISAM file contains the RDF quads and lookups are supported by a small in-memory index which holds the first entry of each block: binary search is performed on the in-memory index to locate the on-disk blocks which can potentially contribute to the answer, where subsequently those blocks are fetched, parsed and answers filtered and returned. Currently, we only require lookups on the subject position of quads, and thus only require one index sorted according to the natural order (s, p, o, c).

There are two tuning parameters for such an index. The first is block size, which determines i) the size of the chunks of data fetched from disk and ii) indirectly, the size of the in-memory portion of the index. The second parameter for tuning is compression: minimising the amount of data transferred from disk to memory should speed up lookups, provided that the time saved by smaller data transfers outweighs the time required for uncompressing data. We will now look at evaluation of these parameters, as well as performance of the inverted index.

9.3. Index Evaluation

In this section, we focus specifically on *local* lookup performance – i.e., the baseline performance – for our inverted and structured indexes. We will detail indexing performance in the next section, and evaluate distributed query-processing in Section 11.3.

In order to evaluate at large scale, we build a local index over the entire consolidated and reasoned dataset consisting of 2.044 g unique quads on one machine. [51]

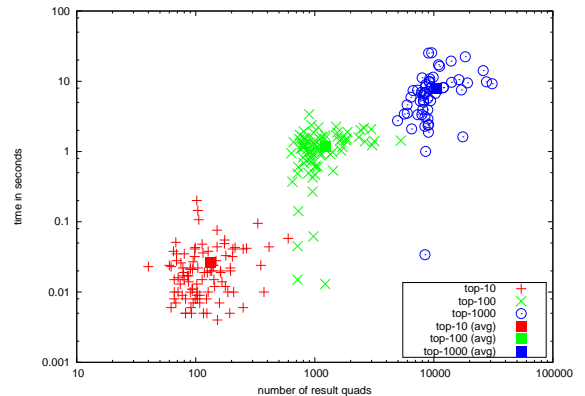[51] This is created by merge-sorting the results of the dis-

Fig. 11. Result size vs. lookup and snippet generation time on log/log scale for top-k results ($k = \{ 10, 100, 1000\}$) of 100 popular keyword searches over an inverted-index built from 2.044 g quads

To evaluate the inverted index, we request keyword-result snippets for the top 100 keyword searches users posed to the online SWSE system: Figure 11 plots the time elapsed versus result size on a log/log scale. Note that in a realistic scenario, we would be posing top-10 queries to the index, but herein also demonstrate top-100 and top-1000 results in order to stress-test the system – we exclude keyword queries that did not meet the quota for a given top-k experiment, where we only include 96 queries which return 10 results, 80 queries which return 100 results, and 57 queries which return 1,000 results. Keyword-result snippets are returned in the form of quads, with an average of ∼12 quads returned per result. For the top-10 queries, the average time taken to generate and stream the results snippets is 26 ms, with the slowest response taking 201 ms. The top-100 requests take on average 1.1 s, with top-1000 results taking on average 7.8 s. On average, Lucene returns identifiers and ranks for hits in ∼16 ms per-query in each of the three setups – the balance of the time is spent retrieving the content of each hit to generate the result snippet.

For the structured index, we tested lookups in various index configurations: from experiments we determined 8k blocks (pre-compression) as a competitive block size [56]. We now compare our index using different compression techniques against MySQL in version 5.0.51a. We created graphs consisting of 5 m to 125 m RDF quads using the random graph model proposed by Erdos-Reny [43], and randomly created lookup keys which were evaluated against the index.

---

tributed indexing detailed in the next section.

We additionally tested a number of configurations for compression: `no compression`, `gzip`, `zlib`, and a combination of these with a simple flyweight encoding (labeled `rle`) where repeating RDF constants inside a block are encoded with a unique integer after the first occurrence. Figure 12 shows the results; the figure includes results for the relational database to up to 8 m statements – larger data would trigger `OutOfMemory` exceptions when performing lookups. Even for data sizes where MySQL returns results, our index organisation generally outperforms the aforementioned relational database setup.
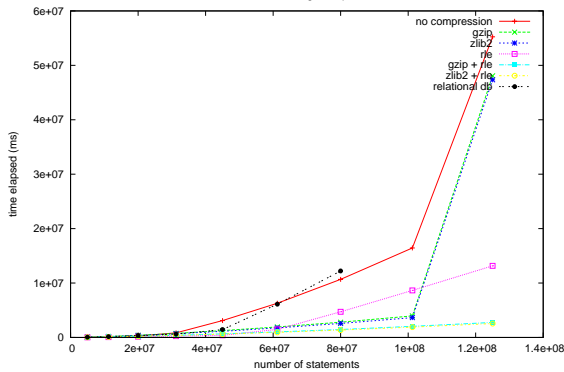

Fig. 12. Lookup performance for 5 m to 125 m statements for various configurations of the index.

We subsequently performed scale-up experiments over the local 2.044 g quad index, where we perform lookups for each result returned by the top-10 keyword-lookup for the 100 most popular keyword searches posed to SWSE – we retrieve and stream all quads for which the hit in question is the subject. The results are illustrated in Figure 13, where on average each lookup takes 7.6 ms. The observant reader will notice one pathological result near the top left which takes 3.3 s, [52] without which the average becomes 4.3 ms – roughly equivalent to a disk-seek given that our compressed index would exist in a given locality on the hard-disk, and perhaps factoring in low-level caching (we include no application-level caching). It's worth noting that the largest result – `dbpedia:Italy` – contained 78.3 k quads from 526 sources, taking 264 ms to process.

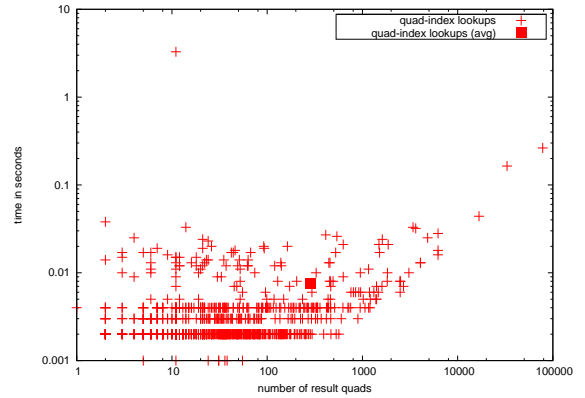[52] Note that this result was not repeatable, seemingly a slow disk seek.

Fig. 13. Result size vs. lookup and result access time on log/log scale for 972 lookups on our structured index built from 2.044 g quads

9.4. Distributed Approach

Again, the core operations required are keyword-search using an inverted-index, and lookups on subject-keys using a structured-index. In order to build the inverted-index documents, we require all information about a given subject on one machine, and so we require a data-placement strategy which gathers common subjects on one machine for our distributed index. We thus use a modulo-hashing function on the subjects of quads to distribute the raw data: not only does this gather common subjects on one machine, but also allows for obvious optimisations in our query processing, discussed in Section 11.

Thus, with respect to building the index in a distributed manner, we identify the following operations:

(i) **scatter**: the master machine splits and sends the (relatively small) results of the T-Box (schema-level) reasoning, hitherto resident only on one machine;

(ii) **co-ordinate**: the slave machines hash triples from the consolidated data and the reasoned data according to the subject position – the data fragments are then sent *directly* to the appropriate peer and incoming data fragments are received from all peers;

(iii) **run**: the slave machines perform a merge-sort of the gathered data and produce a sorted quad index as described;

(iv) **run**: the slave machines then build the inverted-keyword index over the sorted data.

Note that it is during the sorting and index-build steps that the slave machines detect and re-

#machines	1	2	4	8
mins	64.7	35.0	18.6	9.7

Table 9
Time taken for indexing of 31.3 m input statements and 16.7 m reasoned statements for differing numbers of machines

move duplicate reasoned quads: removing reasoned quads containing triples that have already been asserted, or removing all but one reasoned quad for a given triple – in other words, we consider reasoning-generated contexts as expendable.

Table 9 presents the results of applying the indexing procedure on 1, 2, 4, and 8 machines. Again, we observe a largely linear trend with respect to the number of machines: indeed, here our **co-ordinate** distributed function proves its worth, by avoiding the bottleneck of **gathering**/**scattering** on the master machine.

9.5. *Full-Scale Evaluation*

In the final step of our pre-runtime evaluation, we must build the index over the 2.252 g raw consolidated and reasoned statements. The entire process took 534.7 min (8.92 h).

The master machine took 2.3 min to split and sort the reasoned T-Box data, and <1 second to scatter the result. The co-ordinate function – hashing and splitting, sorting and scattering the consolidated and reasoned data on each slave machine – took 363.1 min with an average idle time of 10.1 min (2.8%). On each machine, less than a minute was spent tranferring data, where most time is spent parsing, hashing, splitting and sorting the data. In total, 2.044 g quads were indexed, with a mean of 255.5 m quads per machine, and an average absolute deviation of 157 k (∼0.06% of the mean) representing almost perfectly even distribution given by hashing on subject.

Building the quad indexes in parallel on the slave machine – including merge-sorting the gathered batches and writing the GZipped RLE-encoded index file and creating the sparse-index – took 70.6 min with an average idle time of 74 s (1.7%). Building the Lucene inverted-keyword index took 97 min with an average idle time of 36.5 min (37.2%): one machine took 36.9 min longer than the next slowest machine (we cannot quite discern why – the generated Lucene index was the same on-disk size as the rest of the machines – and the reason seems to be internal to Lucene).

The structured blocked-compressed index occu-

pied 1.8GB on-disk; the Lucene index on each machine was 10.8GB.

In total, 530.7 min was spent in parallel execution (99.3%); however, on average 9% of this time was spent idle by slave machines. A total of 4 min was spent on the master machine.

9.6. *Related Work*

A veritable plethora of RDF stores have been proposed in the literature, most aiming at providing SPARQL functionality, and each bringing with it its own set of priorities for performance, and its own strengths and weaknesses. A subset of these systems rely on underlying relation databases for storage, including 4store [55], Bigdata® [53], Hexastore [128], Jena SDB [54], Mulgara [55], Sesame [19], Virtuoso [44], etc.; the rest rely on so called "native" RDF storage schemes, including HPRD [90], Jena TDB [56], RDF3X [101], SIREn [34], Voldemort [57], etc.

We note that many SPARQL engines include inverted indices – usually Lucene-based – to offer keyword search over RDF data. The authors of [96] describe fulltext-search benchmarking of existing RDF stores – in particular Jena, Sesame2, Virtuoso, and YARS2 – testing queries of varying degrees of complexity involving fulltext search. They showed that for many types of queries, the performance of YARS2 was often not as competitive as other stores, and correctly verified that certain types of queries (e.g., keyword matches for literals of a given property) are not supported by our system. With respect to performance, we have only ever implemented naïve full-SPARQL query-optimisation techniques in YARS2, and have instead focussed on creating scalable read-optimised indexes, demonstrating batch-processing of joins in a distributed environment and focussing on efficiently supporting simple lookups which potentially return large result sets. For example, we choose not to use OIDs (internal integer representations of constants): although OIDs are a proven avenue for optimised query-processing involving large amounts of intermediate results (e.g., cf. [101]), we wish to avoid the expensive translation from internal OIDs to poten-

tially many external constants, instead preserving the ability to stream results directly. In general, we do not currently require support for complex structured queries, and question the utility of more complex full-text functionality to lay users.

9.7. *Future Directions and Open Research Questions*

The future work of the indexing section is inextricably linked to that of the future direction of the query processing and user interface components. At the moment, our index supports simple lookups for entities matching a given keyword, data required to build a keyword snippet, and the quads for which that subject appears.

Given a relatively static query model, a custom-built structured index can be tailored to offer optimised service to the user interface, as opposed to, e.g., a generic SPARQL engine. The main directions for future work in indexing would be to identify an intersection of queries for which optimised indexes can be built in a scalable manner, and queries which offer greater potential to the UI.

Further investigation of compression techniques and other low-level optimisations may further increase the base performance of our system – however, we feel that the combination of RLE encoding and GZIP compression currently demonstrates satisfactory performance.

10. Offline Processing Summary

Having discussed distributed data acquisition, enhancing, analysis and indexing components, we have now reached the end of the off-line index generation process. In this short section, we briefly provide the overall picture of the total task time of these components.

In Table 10, we provide such a summary. In particular, we note that 76.6% of total offline processing is spent crawling, 1.5% consolidating, 3.1% ranking, 5.7% reasoning and 13% indexing. Excluding crawling, 94.5% of time is spent executing tasks in parallel by the slaves, where in total, 52.8 min (5.5%) is required for aggregation and co-ordination on the master machine – this time is spent idle by the slaves and cannot be reduced by the addition of more machines. Total time including crawling was 68.3 h, and excluding crawling 16 h. Thus, at this scale and

with this setup, an index could easily be crawled and built from scratch on a weekly period.

Task	Time(m)	S	%S	M	%M	%T_0	%T_1
crawl	3150	3149	100	1	∼0	—	**76.6**
extract `owl:sameAs`	12.5	12.5	100	0	0	**1.3**	**0.3**
load `owl:sameAs`	8.4	0	0	8.4	100	**0.9**	**0.2**
rewrite data	42.3	42.3	100	0	0	**4.4**	**1**
consolidate	63.3	54.8	86.6	8.5	14.4	**6.6**	**1.5**
extract PLD graph	27.9	27.9	100	0	0	**2.9**	**0.7**
extract ID ranks	67.6	67.6	100	0	0	**7**	**1.6**
aggregate ID ranks	27.7	0	0	27.7	100	**2.9**	**0.7**
rank	126.1	97.5	77.3	28.6	22.7	**13.1**	**3.1**
extract T-Box	66	66	100	0	0	**6.9**	**1.6**
load/reason T-Box	11.2	0	0	11.2	100	**1.2**	**0.3**
reason A-Box	157.6	157.6	100	0	0	**16.4**	**3.8**
reason	235.3	223.6	95	11.7	5	**24.5**	**5.7**
scatter r. T-Box	2.3	0	0	2.3	100	**0.2**	**0.1**
co-ordinate data	363.1	363.1	100	0	0	**37.8**	**8.8**
index quads	70.6	70.6	100	0	0	**7.4**	**1.7**
index keywords	97	97	100	0	0	**10.1**	**2.4**
index	534.7	530.7	99.3	4	0.7	**55.7**	**13**
T_0: total w/o crawl	**959.7**	**906.6**	**94.5**	**52.8**	**5.5**	**100**	**23.4**
T_1: total w/ crawl	**4110**	**4056**	**98.7**	**53.8**	**1.3**	—	**100**

Table 10
Breakdown of time taken in individual off-line components and tasks, with task time in minutes, time spent in parallel execution by the slaves (**S**), percentage of time in parallel execution (%**S**), time spent executing on master machine (**M**), percentage of time in local execution (%**M**), percent of time wrt. total time excluding crawling (%T_0), and percent of time wrt. total time including crawling (%T_1).

11. Query Processing

With the distributed index built and prepared on the slave machines, we now require a query-processor to accept user queries, request and orchestrate lookups over the slave machines, and aggregate, process and stream the final results. In this section, we assume that the master-machine hosts the query-processor: however, we look at different configurations in Section 12. Herein, we aim to characterise the query-processing steps, and give performance for sequential lookups over the distributed index, and for the various information-retrieval tasks required by the user-interface. In particular, we describe the two indices needed for processing user keyword-queries and user focus-queries respectively (see Section 2.1).

11.1. *Distributed Keyword-Query Processing*

For a top-k keyword query, the co-ordinating machine requests k result identifiers and ranks from each of the slave machines. The co-ordinating machine then computes the aggregated top-k hits and requests the snippet result data for each of the hits

from the originating machines, and streams data to the initiating agent. For the purposes of pagination, given a query for page n the originating machine requests the top $n * k$ result identifiers and associated ranks (which, according to Section 9.3 is answerable by Lucene in constant time for varying result sizes), and then determines, requests and streams the relevant result snippets.

11.2. *Distributed Focus-Query Processing*

Creating the raw data for the focus view of a given entity is somewhat complicated by the requirements of the UI. The focus view mainly renders the information encoded by quads for which the identifier of the entity appears in the subject position; however, to provide a more legible rendering, the UI requires human-readable labels for each predicate and object, as well as ranks for prioritising elements in the rendered view (see predicate/object labels in Figure 2). Thus, to provide the raw data required for the focus view of a given entity, the master machine accepts the relevant identifier, performs a hash-function on the identifier, and directly requests data from the respective slave machine (which itself performs a lookup on the structured index). Subsequently, the master machine generates a unique set of predicates and objects appearing in the result set; this set is then split by hash, with each subset sent in parallel to the target slave machines. The slave machines perform lookups for the respective label and global rank, streaming results to the co-ordinating machine, which in turn streams the final results to the initiating agent.

As such, collating the raw data for the focus view is more expensive than a simple lookup on one targeted machine – although helped by the hash-placement strategy, potentially many lookups may be required. We mitigate the expense using some application-level LRU caching, where the co-ordinating machine caches not only keyword snippet results and focus view results, but also the labels and ranks for predicates and objects: in particular, this would save repetitive lookups on commonly encountered properties and classes.

11.3. *Full-scale Evaluation*

In order to test the performance of distributed query evaluation, we built versions of the 2.044 g statement index on 1, 2, 4 and 8 slave machines, with
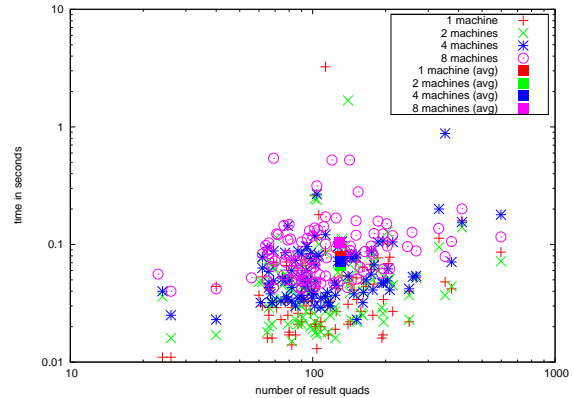

Fig. 14. Result size vs. lookup and snippet generation time on log/log scale for top-10 results of 100 popular keyword searches over an inverted-index built from 2.044 g quads on 1, 2, 4 and 8 slave machines

a remote master machine co-ordinating the query processing.

Figure 14 gives the performance of sequential keyword-lookup and snippet generation for top-10 results for each of the 100 most popular SWSE keyword queries. Please note that due to differences in the *TF* measure for each setup, the *TF*IDF*-based Lucene score can vary slightly and sometimes produce different top-10 results for differing numbers of machines. Besides two outliers (3.3 s and 1.7 s on 1 and 2 machines respectively), all responses are sub-second with average times of 79 ms, 66 ms, 72 ms and 104 ms respectively on 1, 2, 4 and 8 machines. It seems that the keyword-lookups are generally so cheap that distribution is not required, and can even increase average response time given extra network latency. The average number of quads generated for each result listing is ∼130.

In Figure 15, we give the performance of sequential focus-view lookups for 972 entities returned as results for the previous 100 top-10 keyword queries. Generally, we see that on 1 machine, the index begins to struggle [58] – otherwise, the performance of the different setups is roughly comparable. The slowest lookup on each setup was also the largest: `dbpedia:Italy` consisted of 154.7 k quads, requiring 76.4 k additional predicate/object lookups for labels and ranks, roughly doubling the result size – this focus-lookup took 33 s, 9.3 s, 9.2 s and 9 s on 1, 2, 4 and 8 machines respectively. The average focus-view lookup was serviced in 715 ms, 50 ms, 54 ms, and 63 ms on 1, 2, 4 and 8 machines respectively,

---

[58] In particular, we observed ∼17GB of virtual memory being consumed: the limits of the machine are being reached.
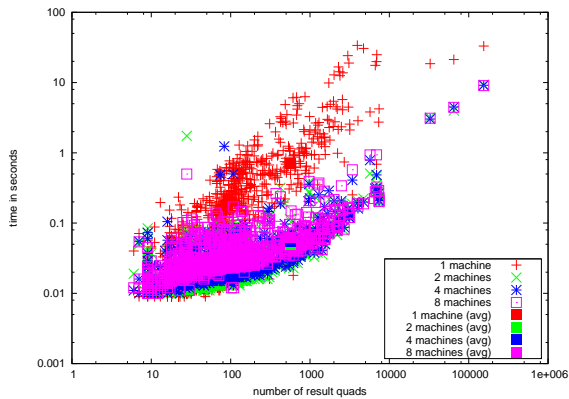
Fig. 15. Result size retrieval time on log/log scale for 972 focus-view lookups over an index built from 2.044 g quads on 1, 2, 4 and 8 slave machines

with an average results size of ∼565 quads (additional labels and ranks roughly double result-size), and an average ∼188 atomic lookups required for label/rank information. We note that moving from 1 to 2 machines offers a huge performance boost, but when further doubling machines, distribution does not significantly affect performance under low loads (sequential lookups) – we will evaluate higher loads in Section 12.

11.4. Related Work

Besides query-processing components for systems referenced in Section 9.6, other types of query-processing have been defined in the literature which do not rely on data warehousing approaches.

The system presented in [63] leverages Linked Data principles to perform live lookups on Linked Data sources, rendering and displaying resulting data; however, such an approach suffers from low recall and inability to independently service keyword queries. In [59] we have described an approach which uses a lightweight hashing-based index structure – viz. a Q-Tree – for mapping structured queries to Linked Data sources which could possibly provide pertinent information; these sources are then retrieved and query-processing performed. Such approaches suffer from poorer recall than data-warehousing approaches, but enable the provision of up-to-date results to users, which is particularly expedient for query processing over highly dynamic sources. We could investigate inclusion of a live-lookup component in SWSE for a subset of queries which we identify to be best answered by means of live lookup; however, further research in this area is

required to identify such queries and to investigate the performance of such a system.

Preliminary work on the DARQ [111] system provides federated query processing over a set of autonomous independent SPARQL endpoints. Such an approach may allow for increased query recall; however, the performance of such a system is still an open question; also, keyword search is still not a standard SPARQL operation and thus federating keyword queries would probably require manual wrappers for different SPARQL endpoints.

11.5. Future Directions and Open Research Question

With respect to current query-processing capabilities, our underlying index structures have proven scalable. However, the focus-view currently requires on average hundreds – but possibly tens or hundreds of thousands – of lookups for labels and ranks. Given that individual result sizes are likely to continue to grow, we will need to incorporate one of the following optimisations: (i) we can build a specialised join index which pre-computes and stores focus-view results, requiring one atomic lookup for the entire focus-view result at the cost of longer indexing time, and (judging from our evaluation) a doubling of structured-index size; and/or (ii) we can generate a top-k focus-view result, paginating the view of a single entity and only retrieving incremental segments of the view – possibly asynchronously.

Extending the query processing to handle more complex queries is a topic of importance when considering extension and improvement of the current spartan UI. In order to fully realise the potential benefits of querying over structured data, we need to be able to perform optimised query processing. For querying data, there is a trade-off between the scalability of the approach and the expressivity of the query language used.

In the general case, joins are expensive operations, and when attempting to perform arbitrary joins on very large datasets, either the system consumes a large amount of resources per query or becomes slow. Some systems (such as [81]) solve the scalability issue by partitioning the data sets into smaller units and have the user select a sub-dataset before further browsing or querying; however, such a solution impinges on the data-integration properties of RDF which provides the raison d'être of a system such as SWSE. Another solution is to pre-compute joins, al-

42

lowing for direct lookup of results emulating the current approach; however, materialising joins can lead to quadratic growth in index sizes. Investigation of partial join materialisation – perhaps based on the expense of a join operation, materialised size of join, runtime caching, etc. – may enable sufficiently optimised query processing.

Another open research question here is how to optimise for top-k querying in queries involving joins; joins at large-scale can potentially lead to the access of large-volumes of intermediary data, used to compute a final small results size; thus, the question is how top-k query processing can be used to immediately retrieve the best results for joins, allowing – e.g. – path queries in the UI (joins on objects and subject) such that large intermediate data volumes need not be accessed, and rather than the approach of joining several attribute restrictions (e.g. facets) as done in the threshold algorithm [45].

12. User and Application Programming Interface

Having discussed distributed data acquisition, enhancing, analysis, indexing and query-processing components, we have now come full circle: in the following section we briefly give the final performance evaluation of our user interface – as described at the outset in Section 2.1 – over our large corpus of enhanced and integrated data, in particular looking more closely at concurrency issues.

12.1. *User Interface Configurations*

Our user-interface uses XSLT to convert the raw data returned by the query-processor into result pages, offering a declarative means of specifying user-interface rendering and ultimately providing greater flexibility for tweaking presentation.

For the distributed setup, we identify two possible configurations for the user-interface, each of which has significant implications for load-balancing:

(i) Figure 16 shows the *master UI configuration*, where the user-interface is hosted alongside the query-processor on the master machine; in our current setup, the query-processing generates little load on the machine, and thus if necessary, the UI result-page generation can absorb the lions-share of the master machine's resources. (This configuration follows naturally from the previous section.)

(ii) Figure 17 shows the *slave UI configuration*, where a query processor and UI instance is hosted on each slave machine and where we assume that some load-balancing function directs users to the individual machines; under heavier loads, this setup avoids the potential bottleneck of hosting the UI on one machine.

We evaluate the above two setups in the following section.

12.2. *Full-scale Evaluation*

In order to evaluate the two competing UI setups, we emulate heavy concurrent access to the UI. We create a query-mix incorporating the top-10 requests for the 100 most popular keyword searches, and 972 focus lookups for each result returned. [59] We create HTTP URIs which encode a GET request for the respective keyword/focus result page: for multiple UIs, we randomly assign queries to the different UIs, offering a cheap form of load-balancing. We then create varying numbers of threads (1, 4, 16, 64, 256, 1,024) to retrieve the content returned by the URIs, emulating different load characteristics for different levels of concurrent access: the requests are made on a clean machine within the same network.

The average result size for the keyword result pages was 17kB and for the focus view responses was 76kB. Figure 18 gives the average response times for both master and slave UI setups, whilst Figure 19 gives the respective maximum response times. We can see that at low user-loads, the master setup performs best, whereas at higher loads, the slave setup performs best, with an average response time of ~30% the master setup for the highest load. Using least squares, we estimated the linear-slope of average time to be 0.031 (s/threads) for the slave machine setup, and 0.106 for the master machine setup (with a standard-error of 0.002 and 0.005 respectively). It is also worth noting that the response times are more variant on the slave setup for low-loads, with maximum response times much greater than for the master setup – more mature load-

---

[59] Note that we analysed one week of SWSE logs in late July 2010, and found a ratio of 25:1 for focus to keyword lookups – we believe that software agents may be probing SWSE, which would explain the high ratio, and some observed repetitive keyword queries. As a side note, we observe certain keyword queries – such as `why did dinosaurs disappear suddenly?` – which give a humbling indication as to how far away we are from fulfilling certain users' expectations for semantic search.
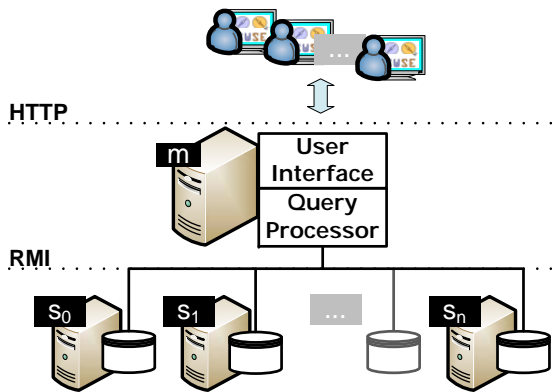
Fig. 16. Master UI configuration, with query processor and user-interface on one machine

Fig. 17. Slave UI configuration, with individual query processors and user-interfaces on each slave machine

balancing based on CPU usage monitoring may perhaps help here. In general, it would seem that the combined UI and query-processing computation is significant wrt. index lookups, and can become a bottleneck at higher loads.

Out of interest, we also ran the same experiment for 1,024 threads, but turning off the predicate/object label and rank lookups which we deemed in previous experiments to be expensive; in this mode, the UI resorts to displaying lexicographically ordered predicate/object values and labels generated from URI suffixes where available. For the master UI setup, only a 2% saving on average response time was observed, whereas an 11% saving was observed for the slave UI setup – in the former setup, it seems that the UI is generating the bottleneck and extra index-lookups make little difference to performance.

In summary, for the slave UI setup, aside from accommodating 1,024 very patient users – at which level of load all requests were successfully responded to, but with the slowest queries taking upto 1.5 min – it seems that we can currently accommodate a maximum of ~30 concurrent users whilst continuing to offer *average* sub-second response times.

12.3. *Related Work*

There has been considerable work on rendering and displaying RDF data; such systems include: BrowseRDF [104], Explorator [30], gFacet [68], Haystack [83], Longwell [60], Piggybank [80], (Power)Magpie [52], Marbles [61], RKBEx-

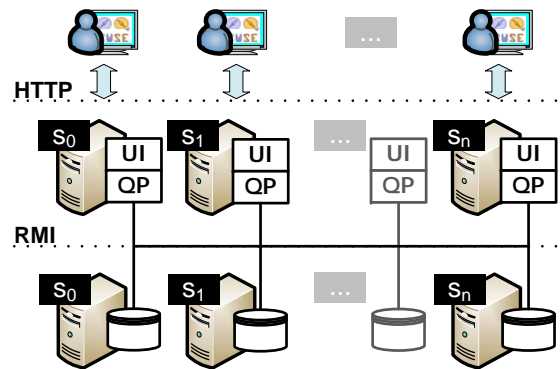plorer [62], Tabulator [10], Zitgist [63], as well as user interfaces for previously mentioned engines such as Falcons, Sig.ma, Sindice, Swoogle, Watson, etc.

Fresnel [108] has defined an interesting approach to overcome the difficultly of displaying RDF in a domain-agnostic way by providing a vocabulary for describing how RDF should be rendered, thus allowing for the declarative provision of schema specific views over data; some user-interfaces have been proposed to exploit Fresnel, including LENA [86]: however, Fresnel has not seen widespread adoption on the Web thus far.

12.4. *Future Directions and Open Research Questions*

Firstly, we must review the performance of the UI with respect to generating results pages – we had not previously considered this issue, but under high-loads, UI result-generation seems to be a significant factor in deciding response times.

With respect to functionality, we currently do not fully exploit the potential offered by richly structured data. Firstly, such data could power a large variety of visualisations: for example, to render SIMILE's timeline view [64] or a Google map view [65]. Countless other visualisations are possible: for a history and examples of visualisations, cf. [49]. Research into rendering and visualising large graph-structured datasets – particularly user evaluation

---

[60] http://simile.mit.edu/wiki/Longwell
[61] http://marbles.sourceforge.net/

[62] http://www.rkbexplorer.com/explorer/
[63] http://dataviewer.zitgist.com/
[64] http://www.simile-widgets.org/timeline/
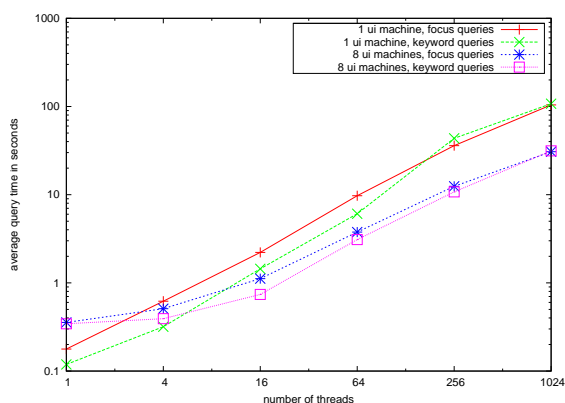[65] http://maps.google.com/

Fig. 18. Average response time for keyword and focus queries with master UI and slave UI setup for a varying number of threads emulating concurrent access
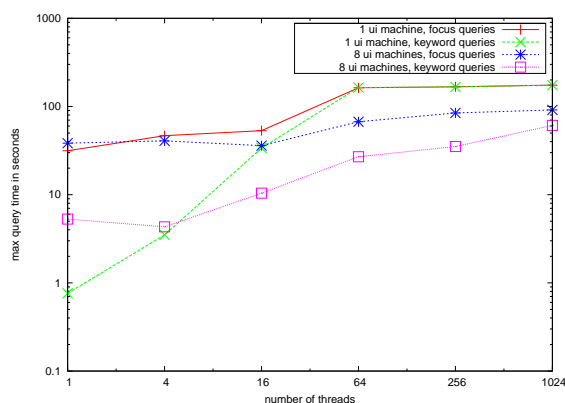

Fig. 19. Maximum response time for keyword and focus queries with master UI and slave UI setup for a varying number of threads emulating concurrent access

thereof – could lead to novel user interfaces which better suit and exploit such information.

Secondly, offering only keyword search and entity browsing removes the possibility of servicing more expressive queries which offer users more direct answers: however, designing a system for domain-agnostic users to formulate such queries in an intuitive manner – and one which is guided by the underlying data to avoid empty results where possible – has proven non-trivial. We have made first experiments with more expressive user interfaces for interacting with data through the VisiNav system [66] [57], which supports faceted browsing [129], path traversal [4] and data visualisations on top of the keyword search and focus operations supported by SWSE. Within VisiNav, we encourage users to incrementally create expressive queries while browsing, as opposed to having a formal query formulation step – users are offered navigation choices which lead to non-empty results. However, for such extra functionality – specifically the cost of querying associated with arbitrary join paths – VisiNav must make scalability trade-offs: VisiNav can currently handle in the realm of tens of millions of statements.

Efforts to provide guided construction of structured queries (e.g., cf. [97]) may be useful to so-called 'power-users'; however, such methods again rely on some knowledge of the schema of the pertinent data and query. Other efforts to match keyword searches to structured queries (e.g., cf. [119,91,27]) could bring together the ease of use of Web search engines and the precise answers of structured data querying; however, again such formulations still re-

quire some knowledge of the schema(ta) of the data, and which types of entities link to which by what type of link.

For the moment, we focus on providing basic functionality as should be familiar to many Web users. Previous incarnations of SWSE offered more complex user-interaction models, allowing, e.g., filtering of results based on type, traversing inlinks for an entity, traversing links from a collection of entities, etc. From informal feedback received, we realised that features such as inlink traversal (and the notion of directionality) were deemed confusing by certain users – or, at least by our implementation thereof. [67] We are thus more cautious about implementing additional features in the user interface, aiming for minimalistic display and interaction. One possible solution is to offer different versions of the user-interface; e.g., a default system offering simple keyword-search for casual users, and an optional system offering more complex functionality for power users. In such regards, user-evaluation (currently out of scope) would be of utmost importance in making such design-choices.

We also wish to investigate the feasibility of offering programmatic interfaces through SWSE. [68] The main requirements for such APIs are perfor-

---

[66] http://visinav.deri.org/

[67] In any case, our reasoning engine supports the `owl:inverseOf` construct which solves the problem of directionality, and we would hope that most (object) properties define a corresponding inverse-property.

[68] Please note that practical limitations with respect to the availability and administration of physical machines has restricted our ability to provide such interfaces with high reliability; indeed, we used to offer timeout SPARQL queries over ∼1.5 bn statements through YARS2, but for the meantime, we can no longer support such a service.

mance and reliability: the API has to return results fast enough to enable interactive applications, and has to have high uptime to encourage adoption by external services. Full SPARQL is likely too powerful (and hence too expensive to evaluate) to provide stable, complete *and* fast responses for. One possible workaround is to provide timeout queries, which return as many answers as can be serviced in a fixed time period; another possible solution is to offer top-k query processing, or a well-supported subset of SPARQL (e.g., DESCRIBE queries and conjunctive queries with a limited number of joins, or containing highly-selective patterns) such that could serve as a foundation for visualisations and other applications leveraging the integrated Web data in SWSE.

13. High-Level Future Directions

With respect to high-level future directions for SWSE, we foresee the following goals:
– to scale further, where we would see ∼10bn triples as the next feasible goal to aim for on the given hardware;
– to investigate modifications and extensions to the existing components – as discussed throughout the paper – to better realise their effectiveness under their presented requirements;
– to look into methods for evaluating the precision and recall of our consolidation and reasoning methods;
– to create a cyclical indexing framework whereby the system is constantly building the new index while the old is being queried against;
– to investigate incremental update methods, re-using knowledge (data, statistics, etc.) acquired from previous index builds to accelerate incremental builds;
– to further enhance the user-interface and provide more complex features in a manner unobtrusive to our minimalistic aesthetics;
– to investigate and conduct user evaluation as a litmus test for the real-world utility of our research.

With respect to scaling further, current known limits relate to those components relying on large in-memory indices (which comprises the global knowledge required by all machines); viz.: reasoning and consolidation. However, as discussed for the case of reasoning, we could simply store the T-Box in an on-disk structure with heavy caching, and as discussed for consolidation, we can revert to batch processing techniques which do not rely on an in-memory equal-

ity index (see, e.g., [72]). Improving user-interface response times at larger scale and high loads would again be an open question.

With respect to cyclical indexing, we could separate interests by having one set of machines preparing the next index, and one set of machines offering query processing, switching between them as appropriate. This would allow us to maintain update-to-date information, with freshness dependant on the index build interval. In a similar vein – and although all of the methods presented are tailored for the application over static datasets – we could investigate methods for increment updates which avoids repetitive work for each such index build.

Finally, it would of course be appropriate to invest more time in the end-product of our system: improving our user interface and performing user evaluation to verify the benefit of our methods and the usability of our system. Our main focus is on the underlying research required for the realisation of the Semantic Web Search Engine, but such research would gain practical prioritisation, inspiration and motivation from user feedback.

14. Conclusion

In this paper, we have presented the results of research carried out as part of the SWSE project over the past six years. In particular, we have adapted the architecture of large-scale Web search engines to the case of structured data. We have presented lightweight algorithms which demonstrate the data-integration possibilities for Linked Data, and shown how such algorithms can be made scalable using batch processing techniques such as scans and sorts, and how they can be deployed over a distributed architecture. We have also discussed and shown the importance of taking the source of information into account when handling arbitrary RDF Web data, showing how Linked Data principles can be leveraged for such purposes, particularly in our ranking and reasoning algorithms. Throughout, we have presented related work and possible future directions: based on the experiences collected, we have identified open research questions which we believe should be solved in order to – directly or indirectly – get closer to the vision of search over the Web of Data discussed in the introduction.

Research on how to integrate and interact with large amounts of data from a very diverse set of independent sources is fairly recent, as many charac-

teristics of the research questions in the field became visible after the deployment of significant amounts of data by a significant body of data publishers. The traditional application development cycle for data-intensive applications is to model the data schema and build the application on top – data modeling and application development are tightly coupled. That process is separated on the Semantic Web: data publishers just model and publish data, often with no particular application in mind. That approach leads to a chicken-egg problem: people do not have an incentive to publish data because there are no applications that would make specific use of the data, and developers do not create useful applications because of the lack of quality data. Indeed, the quality of data, that a system such as SWSE operates over, is perhaps as much of a factor in the system's utility as the design of the system itself.

Recently there has been significant success with Linked Data where an active community publishes datasets in a broad range of topics, and maintains and interlinks these datasets. Again, efforts such as DBpedia have lead to a much richer Web of Data than the one present when we began working on SWSE. However, data heterogeneity still poses problems – not so much for the underlying components of SWSE – but for the user-facing components and the users themselves: allowing domain-oblivious users to create flexible structured queries in a convenient and intuitive manner is still an open question. Indeed, the Web of Data still cannot compete with the vast coverage of the Web of Documents, and perhaps never will [109].

That said, making Web data available for querying and navigation has significant scientific and commercial potential. Firstly, the Web becomes subject to scientific analysis [12]: understanding the implicit connections and structure of the Web of Data can help to reveal new understandings of collaboration patterns and the processes by which networks form and evolve. Secondly, aggregating and enhancing scientific data published on the Web can help scientists to more easily perform data-intensive research, in particular allowing for the arbitrary re-purposing of published datasets which can subsequently be used in ways unforeseen by the original publisher; indeed, the ability to navigate and search effectively through a store of knowledge integrated from multiple sources can broaden the pool of information and ideas available to a scientific community. Thirdly, making the Web of Data available for interactive querying, browsing, and navigation has applications in areas such as e-commerce and e-health, allowing data-analysts in such fields to pose complex structured queries over a dataset aggregated from multitudinous relevant sources.

Commercial success – and bringing a system such as the Semantic Web Search Engine into the mainstream – is perhaps a longer term goal, relying on the increased growth in RDF data becoming available: data which is of mainstream interest, and has a broad coverage of topics. In any case, there are still many open research questions to tackle in the years to come.

References

[1]   H. Alani, C. Brewster, N. Shadbolt, Ranking ontologies with AKTiveRank, in: 5th International Semantic Web Conference, 2006.

[2]   H. Alani, S. Dasmahapatra, K. O'Hara, N. Shadbolt, Identifying communities of practice through ontology network analysis, IEEE Intelligent Systems 18 (2) (2003) 18–25.

[3]   K. Anyanwu, A. Maduko, A. Sheth, SemRank: ranking complex relationship search results on the semantic web, in: 14th International Conference on World Wide Web, 2005.

[4]   N. Athanasis, V. Christophides, D. Kotzinos, Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL), in: 3rd International Semantic Web Conference, 2004.

[5]   S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. G. Ives, DBpedia: A Nucleus for a Web of Open Data, in: ISWC/ASWC, 2007.

[6]   A. Balmin, V. Hristidis, Y. Papakonstantinou, Objectrank: authority-based keyword search in databases, in: Proceedings of the 13th International Conference on Very Large Data Bases, 2004.

[7]   S. Batsakis, E. G. M. Petrakis, E. Milios, Improving the performance of focused web crawlers, Data Knowl. Eng. 68 (10) (2009) 1001–1013.

[8]   S. Bechhofer, R. Volz, Patching Syntax in OWL Ontologies, in: International Semantic Web Conference (ISWC 2004), vol. 3298 of Lecture Notes in Computer Science, Springer, 2004.

[9]   T. Berners-Lee, Linked Data, Design issues for the World Wide Web, World Wide Web Consortium, `http://www.w3.org/DesignIssues/LinkedData.html` (2006).

[10] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, D. Sheets, Tabulator: Exploring and analyzing linked data on the semantic web, in: In Proceedings of the 3rd International Semantic Web User Interaction Workshop, 2006.

[11] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, http://tools.ietf.org/html/rfc3986 (January 2005).

[12] T. Berners-Lee, W. Hall, J. Hendler, N. Shadbolt, D. J. Weitzner, Creating a Science of the Web, Science 313 (11).

[13] C. Bizer, R. Cyganiak, D2R Server - Publishing Relational Databases on the Web as SPARQL Endpoints, in: ISWC, 2006, (poster).

[14] C. Bizer, T. Heath, T. Berners-Lee, Linked Data - The Story So Far, Int. J. Semantic Web Inf. Syst. 5 (3) (2009) 1–22.

[15] P. Boldi, B. Codenotti, M. Santini, S. Vigna, S. Vigna, UbiCrawler: a scalable fully distributed web crawler, Software: Practice and Experience 34 (2002) 2004.

[16] P. A. Bonatti, A. Hogan, A. Polleres, L. Sauro, Robust and Scalable Linked Data Reasoning Incorporating Provenance and Trust Annotations, Journal of Web Semantics (In Press).

[17] P. Bouquet, H. Stoermer, M. Mancioppi, D. Giacomuzzi, OkkaM: Towards a Solution to the "Identity Crisis" on the Semantic Web, in: Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop, vol. 201 of CEUR Workshop Proceedings, 2006.

[18] S. Brin, L. Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, Computer Networks 30 (1-7) (1998) 107–117.

[19] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, in: 2nd International Semantic Web Conference, Springer, 2002.

[20] D. Cai, X. He, J. Wen, W. Ma, Block-level link analysis, in: 27th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004.

[21] J. Caverlee, L. Liu, QA-Pagelet: Data Preparation Techniques for Large-Scale Data Analysis of the Deep Web, IEEE Trans. Knowl. Data Eng. 17 (9) (2005) 1247–1262.

[22] S. Chakrabarti, M. van den Berg, B. Dom, Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery, Computer Networks 31 (11-16) (1999) 1623–1640.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber, Bigtable: A Distributed Storage System for Structured Data, in: OSDI, 2006.

[24] K. C.-C. Chang, B. He, Z. Zhang, Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web, in: CIDR, 2005.

[25] Z. Chen, D. V. Kalashnikov, S. Mehrotra, Exploiting relationships for object consolidation, in: IQIS '05: Proceedings of the 2nd international workshop on Information quality in information systems, ACM Press, New York, NY, USA, 2005.

[26] G. Cheng, W. Ge, H. Wu, Y. Qu, Searching Semantic Web Objects Based on Class Hierarchies, in: Proceedings of Linked Data on the Web Workshop, 2008.

[27] G. Cheng, Y. Qu, Searching Linked Objects with Falcons: Approach, Implementation and Evaluation., Int. J. Semantic Web Inf. Syst. 5 (3) (2009) 49–70.

[28] T. Cheng, K. C.-C. Chang, Entity Search Engine: Towards Agile Best-Effort Information Integration over the Web, in: CIDR, 2007.

[29] M. d'Aquin, M. Sabou, E. Motta, S. Angeletou, L. Gridinoc, V. Lopez, F. Zablith, What Can be Done with the Semantic Web? An Overview Watson-based Applications, in: SWAP, 2008.

[30] S. F. C. de Araújo, D. Schwabe, Explorator: a tool for exploring RDF data through direct manipulation, in: Linked Data on the Web WWW2009 Workshop (LDOW2009), 2009.

[31] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: OSDI, 2004.

[32] S. Decker, M. Erdmann, D. Fensel, R. Studer, Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information, in: DS-8: IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics, Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 1998.

[33] R. Delbru, A. Polleres, G. Tummarello, S. Decker, Context Dependent Reasoning for Semantic Documents in Sindice, in: Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008), Karlsruhe, Germany, 2008.
URL http://www.polleres.net/publications/delb-etal-2008.pdf

[34] R. Delbru, N. Toupikov, M. Catasta, G. Tummarello, A Node Indexing Scheme for Web Entity Retrieval, in: Proceedings of the Extended Semantic Web Conference (ESWC 2010), 2010.

[35] R. Delbru, N. Toupikov, M. Catasta, G. Tummarello, S. Decker, Hierarchical Link Analysis for Ranking Web Data, in: Proceedings of the Extended Semantic Web Conference (ESWC 2010), 2010.

[36] H. Dietze, M. Schroeder, Semplore: A Scalable IR Approach to Search the Web of Data, BMC Bioinformatics 10.

[37] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori, Focused Crawling Using Context Graphs, in: VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[38] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. C. Doshi, J. Sachs, Swoogle: A Search and Metadata Engine for the Semantic Web, in: 13th ACM Conference on Information and Knowledge Management, ACM Press, 2004.

[39] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, P. Kolari, Finding and ranking knowledge on the semantic web, in: 4th International Semantic Web Conference, 2005.

[40] H. Dong, F. K. Hussain, E. Chang, State of the Art in Semantic Focused Crawlers, in: ICCSA '09: Proceedings of the International Conference on Computational Science and Its Applications, Springer-Verlag, Berlin, Heidelberg, 2009.

[41] M. Ehrig, A. Maedche, Ontology-focused crawling of Web documents, in: SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, ACM, New York, NY, USA, 2003.

[42] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios, Duplicate Record Detection: A Survey, IEEE Transactions on Knowledge and Data Engineering 19 (1) (2007) 1–16.

[43] P. Erdös, A. Rényi, On random graphs, I, Publicationes Mathematicae (Debrecen) 6 (1959) 290–297.

[44] O. Erling, I. Mikhailov, RDF Support in the Virtuoso DBMS, in: CSSW, 2007.

[45] R. Fagin, Combining fuzzy information from multiple systems (extended abstract), in: PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM, 1996.

[46] D. Fensel, F. van Harmelen, Unifying Reasoning and Search to Web Scale, IEEE Internet Computing 11 (2) (2007) 96, 94–95.

[47] R. Fielding, J. Gettys, J. Mogul, H. F. Nielsen, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, ftp://ftp.isi.edu/in-notes/rfc2616.txt (June 1999).

[48] T. Franz, A. Schultz, S. Sizov, S. Staab, TripleRank: Ranking Semantic Web Data By Tensor Decomposition, in: 8th International Semantic Web Conference (ISWC2009), 2009.

[49] M. Friendly, A Brief History of Data Visualization, in: C. Chen, W. Härdle, A. Unwin (eds.), Handbook of Computational Statistics: Data Visualization, vol. III, Springer-Verlag, Heidelberg, 2006.

[50] H. Glaser, I. Millard, A. Jaffri, RKBExplorer.com: A knowledge driven infrastructure for linked data providers, in: ESWC Demo, Lecture Notes in Computer Science, Springer, 2008.

[51] B. C. Grau, B. Motik, Z. Wu, A. Fokoue, C. Lutz, OWL 2 Web Ontology Language: Profiles, W3C Working Draft, http://www.w3.org/TR/owl2-profiles/ (Apr. 2008).

[52] L. Gridinoc, M. Sabou, M. d'Aquin, M. Dzbor, E. Motta, Semantic Browsing with PowerMagpie, in: ESWC, 2008.

[53] R. V. Guha, R. McCool, R. Fikes, Contexts for the Semantic Web, in: 3rd International Semantic Web Conference, Hiroshima, 2004.

[54] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, H. S. Thompson, When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data, in: International Semantic Web Conference (1), 2010.

[55] S. Harris, N. Lamb, N. Shadbolt, 4store: The Design and Implementation of a Clustered RDF Store, in: 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009), 2009.

[56] A. Harth, Exploring Linked Data at Web Scale, Ph.D. thesis, Digital Enterprise Research Institute, National University of Ireland, Galway (2010).

[57] A. Harth, Visinav: A system for visual search and navigation on web data, J. Web Sem. 8 (4) (2010) 348–354.

[58] A. Harth, S. Decker, Optimized Index Structures for Querying RDF from the Web, in: 3rd Latin American Web Congress, IEEE Press, 2005.

[59] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, J. Umbrich, Data summaries for on-demand queries over linked data, in: WWW, 2010.

[60] A. Harth, S. Kinsella, Topdis: Tensor-based ranking for data search and navigation, Tech. rep., DERI (6 2009).

[61] A. Harth, S. Kinsella, S. Decker, Using Naming Authority to Rank Data and Ontologies for Web Search, in: 8th International Semantic Web Conference (ISWC 2009), 2009.

[62] A. Harth, J. Umbrich, A. Hogan, S. Decker, YARS2: A Federated Repository for Querying Graph Structured Data from the Web, in: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, 2007.

[63] O. Hartig, C. Bizer, J. C. Freytag, Executing SPARQL Queries over the Web of Linked Data, in: International Semantic Web Conference, 2009.

[64] E. Hatcher, O. Gospodnetic, Lucene in Action, Manning Publications, 2004.

[65] P. Hayes, RDF Semantics, W3C Recommendation, http://www.w3.org/TR/rdf-mt/ (Feb. 2004).

[66] B. He, M. Patel, Z. Zhang, K. C.-C. Chang, Accessing the Deep Web, Commun. ACM 50 (5) (2007) 94–101.

[67] J. Heflin, J. Hendler, S. Luke, SHOE: A Knowledge Representation Language for Internet Applications, Tech. Rep. CS-TR-4078, Dept. of Computer Science, University of Maryland (1999).

[68] P. Heim, J. Ziegler, S. Lohmann, gFacet: A Browser for the Web of Data, in: Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08), CEUR-WS, 2008.

[69] A. Heydon, M. Najork, Mercator: A Scalable, Extensible Web Crawler, World Wide Web 2 (1999) 219–229.

[70] J. Hirai, S. Raghavan, H. Garcia-Molina, A. Paepcke, WebBase: a repository of Web pages", Computer Networks 33 (1–6) (2000) 277–293.

[71] P. Hitzler, F. van Harmelen, A Reasonable Semantic Web, Semantic Web – Interoperability, Usability, Applicability 1.

[72] A. Hogan, Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora, Ph.D. thesis, Digital Enterprise Research Institute, National University of Ireland, Galway, available from http://aidanhogan.com/docs/thesis/ (2011).

[73] A. Hogan, S. Decker, On the Ostensibly Silent 'W' in OWL 2 RL, in: Third International Conference on Web Reasoning and Rule Systems, (RR2009), 2009.

[74] A. Hogan, A. Harth, S. Decker, ReConRank: A Scalable Ranking Method for Semantic Web Data with Context, in: 2nd Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006), 2006.

[75] A. Hogan, A. Harth, S. Decker, Performing Object Consolidation on the Semantic Web Data Graph, in: 1st I3 Workshop: Identity, Identifiers, Identification Workshop, 2007.

[76] A. Hogan, A. Harth, A. Passant, S. Decker, A. Polleres, Weaving the Pedantic Web, in: Linked Data on the Web WWW2010 Workshop (LDOW2010), 2010.

[77] A. Hogan, A. Harth, A. Polleres, Scalable Authoritative OWL Reasoning for the Web, Int. J. Semantic Web Inf. Syst. 5 (2).

[78] A. Hogan, J. Z. Pan, A. Polleres, S. Decker, SAOR: Template Rule Optimisations for Distributed Reasoning over 1 Billion Linked Data Triples, in: International Semantic Web Conference, 2010.

[79] A. Hogan, A. Polleres, J. Umbrich, A. Zimmermann, Some entities are more equal than others: statistical methods to consolidate Linked Data, in: 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS2010), 2010.

[80] D. Huynh, S. Mazzocchi, D. R. Karger, Piggy Bank: Experience the Semantic Web inside your web browser, J. Web Sem. 5 (1) (2007) 16–27.

[81] D. F. Huynh, D. Karger, Parallax and Companion: Set-based Browsing for the Data Web, available online (2008-12-15) http://davidhuynh.net/media/papers/2009/www2009-parallax.pdf.

[82] X.-M. Jiang, G.-R. Xue, W.-G. Song, H.-J. Zeng, Z. Chen, W.-Y. Ma, Exploiting PageRank at Different Block Level , in: 5th International Conference on Web Information Systems, 2004.

[83] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, V. Sinha, Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data, in: CIDR, 2005.

[84] A. Kiryakov, D. Ognyanoff, R. Velkov, Z. Tashev, I. Peikov, LDSR: a Reason-able View to the Web of Linked Data, in: Semantic Web Challenge (ISWC2009), 2009.

[85] J. M. Kleinberg, Authoritative Sources in a Hyperlinked Environment, Journal of the ACM 46 (5) (1999) 604–632.

[86] J. Koch, T. Franz, LENA – Browsing RDF Data More Complex Than Foaf, in: International Semantic Web Conference (Posters & Demos), 2008.

[87] S. Kotoulas, E. Oren, F. van Harmelen, Mind the data skew: distributed inferencing by speeddating in elastic regions, in: WWW, 2010.

[88] H.-T. Lee, D. Leonard, X. Wang, D. Loguinov, IRLbot: Scaling to 6 billion pages and beyond, ACM Trans. Web 3 (3) (2009) 1–34.

[89] Y. Lei, V. Uren, E. Motta, Semsearch: A search engine for the semantic web, in: 14th International Conference on Knowledge Engineering and Knowledge Management, 2006.

[90] B. Liu, B. Hu, HPRD: A High Performance RDF Database, in: NPC, 2007.

[91] V. Lopez, V. S. Uren, E. Motta, M. Pasin, AquaLog: An ontology-driven question answering system for organizational semantic intranets, J. Web Sem. 5 (2) (2007) 72–105.

[92] F. Manola, E. Miller, B. McBride, RDF Primer, W3C Recommendation, http://www.w3.org/TR/rdf-primer/ (Feb. 2004).

[93] G. Meditskos, N. Bassiliades, DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet, J. Web Sem. 8 (1) (2010) 89–94.

[94] S. Melnik, S. Raghavan, B. Yang, H. Garcia-Molina, Building a Distributed Full-Text Index for the Web, in: 10th International World Wide Web Conference, Hong Kong, 2001.

[95] M. Michalowski, S. Thakkar, C. A. Knoblock, Exploiting secondary sources for automatic object consolidation, in: Proceeding of 2003 KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation, 2003.

[96] E. Minack, W. Siberski, W. Nejdl, Benchmarking Fulltext Search Performance of RDF Stores, in: ESWC, 2009.

[97] K. Möller, O. Ambrus, L. Josan, S. Handschuh, A Visual Interface for Building SPARQL Queries in Konduit, in: International Semantic Web Conference (Posters & Demos), 2008.

[98] S. Muñoz, J. Pérez, C. Gutiérrez, Minimal Deductive Systems for RDF, in: ESWC, 2007.

[99] M. Najork, J. L. Wiener, Breadth-First Search Crawling Yields High-Quality Pages, in: In Proc. 10th International World Wide Web Conference, 2001.

[100] M. Najork, H. Zaragoza, M. Taylor, HITS on the Web: How does it Compare?, in: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 2007.

[101] T. Neumann, G. Weikum, The RDF-3X engine for scalable management of RDF data, VLDB J. 19 (1) (2010) 91–113.

[102] H. B. Newcombe, J. M. Kennedy, S. J. Axford, A. P. James, Automatic Linkage of Vital Records: Computers can be used to extract "follow-up" statistics of families from files of routine records, Science 130 (1959) 954–959.

[103] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, G. Tummarello, Sindice.com: A document-oriented lookup index for open linked data, Int. J. Metadata Semant. Ontologies 3 (1) (2008) 37–52.

[104] E. Oren, R. Delbru, S. Decker, Extending Faceted Navigation for RDF Data, in: International Semantic Web Conference, 2006.

[105] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, F. van Harmelen, Marvin: Distributed reasoning over large-scale Semantic Web data, J. Web Sem. 7 (4) (2009) 305–316.

[106] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, Tech. rep., Stanford Digital Library Technologies Project (1998).

[107] G. Pant, P. Srinivasan, Learning to crawl: Comparing classification schemes, ACM Trans. Inf. Syst. 23 (4) (2005) 430–462.

[108] E. Pietriga, C. Bizer, D. R. Karger, R. Lee, Fresnel: A Browser-Independent Presentation Vocabulary for RDF, in: International Semantic Web Conference, 2006.

[109] A. Polleres, A. Hogan, A. Harth, S. Decker, Can we ever catch up with the Web?, Semantic Web – Interoperability, Usability, Applicability 1.

[110] E. Prud'hommeaux, A. S. (eds.), SPARQL Query Language for RDF, W3C Recommendation, `http://www.w3.org/TR/rdf-sparql-query/` (Jan. 2008).

[111] B. Quilitz, U. Leser, Querying Distributed RDF Data Sources with SPARQL, in: ESWC, 2008.

[112] S. Raghavan, H. Garcia-Molina, Crawling the Hidden Web, in: VLDB, 2001.

[113] D. Reynolds, OWL 2 RL in RIF, W3C Working Group Note, `http://www.w3.org/TR/rif-owl-rl/` (Jun. 2010).

[114] M. Sabou, C. Baldassarre, L. Gridinoc, S. Angeletou, E. Motta, M. d'Aquin, M. Dzbor, WATSON: A Gateway for the Semantic Web, in: ESWC 2007 poster session, 2007-06.

[115] M. K. Smith, C. Welty, D. L. McGuinness, OWL Web Ontology Language Guide, W3C Recommendation, `http://www.w3.org/TR/owl-guide/` (Feb. 2004).

[116] M. Stonebraker, The Case for Shared Nothing, IEEE Database Eng. Bull. 9 (1) (1986) 4–9.

[117] H. J. ter Horst, Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary, Journal of Web Semantics 3 (2005) 79–115.

[118] M. Thelwall, D. Stuart, Web crawling ethics revisited: Cost, privacy, and denial of service, Journal of the American Society for Information Science and Technology 57 (2006) 1771–1779.

[119] T. Tran, H. Wang, S. Rudolph, P. Cimiano, Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data, in: ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, 2009.

[120] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, S. Decker, Sig.ma: Live views on the Web of Data, in: Semantic Web Challenge, 2009.

[121] J. Umbrich, A. Harth, A. Hogan, S. Decker, Four heuristics to guide structured content crawling, in: Proceedings of the 2008 Eighth International Conference on Web Engineering-Volume 00, IEEE Computer Society, 2008.

[122] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, H. E. Bal, OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples, in: ESWC (1), 2010.

[123] J. Urbani, S. Kotoulas, E. Oren, F. van Harmelen, Scalable Distributed Reasoning Using MapReduce, in: International Semantic Web Conference (ISWC 2009), vol. 5823, Springer, Washington DC, USA, 2009.

[124] J. Volz, C. Bizer, M. Gaedke, G. Kobilarov, Discovering and Maintaining Links on the Web of Data, in: International Semantic Web Conference, 2009.

[125] T. D. Wang, B. Parsia, J. A. Hendler, A Survey of the Web Ontology Landscape, in: International Semantic Web Conference, 2006.

[126] J. Weaver, J. A. Hendler, Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples, in: International Semantic Web Conference (ISWC2009), 2009.

[127] W. Wei, P. M. Barnaghi, A. Bargiela, Search with Meanings: An Overview of Semantic Search Systems, Int. J. Communications of SIWN 3 (2008) 76–82.

[128] C. Weiss, P. Karras, A. Bernstein, Hexastore: Sextuple Indexing for Semantic Web Data Management, PVLDB 1 (1) (2008) 1008–1019.

[129] K.-P. Yee, K. Swearingen, K. Li, M. Hearst, Faceted metadata for image search and browsing, in: SIGCHI Conference on Human factors in Computing Systems, 2003.

Appendix A. Selected Dataset Statistics

Herein, we give some selected statistics about the 1.118 g Linked Data corpus crawled in Section 5. The resulting evaluation corpus is sourced from 3.985 m documents and contains 1.118 g quads, of which 1.106 g are unique (98.9%) and 947 m are unique triples (84.7% of raw quads).

To characterise our corpus, we first look at a breakdown of data providers. We extracted the PLDs from the source documents and summated occurrences: Table A.1 shows the top 25 PLDs with respect to the number of triples they provide in our corpus, as well as their document count and average number of triples per document. We see that a large portion of the data is sourced from social networking sites – such as `hi5.com` and `livejournal.com` – that host FOAF exports for millions of users. Notably, the `hi5.com` domain provides 595 m (53.2%) of all quadruples in the data: although the number of documents crawled from this domain was comparable with other high yield domains, the high ratio of triples per document meant that in terms of quadruples, `hi5.com` provides the majority of data. Other sources in the top-5 include the `opiumfield.com` domain which offers LastFM exports, and `linkedlifedata.com` and `bio2rdf.org` which publishes data from the life-science domain.

Continuing, we encountered 199.4 m unique subjects (entities), of which 165.3 m (82.9%) are 'identified' by a blank-node and 34.1 m (17.1%) are identified by a URI. Figure A.1 shows the distribution of the number of edges attached to each entity (appearances in the subject position of a quad), where the largest entity has 252 k edges, and the plurality is three edges with 154 m entities. [69]

With respect to objects, we found that 668 m are URIs (60.4%), 273.5 m are literals (24.7%), and 164.5 m (14.9%) are blank-nodes.

Next, we look at usage of properties and classes in the data: for properties, we analysed the frequency of occurrence of terms in the predicate position, and for classes, we analysed the occurrences of terms in the object position of `rdf:type` quads.

---

[69] *Plurality*: having more than all alternatives, without necessarily constituting a majority.

We found 23,155 unique predicates: Figure A.2 gives the distribution of predicate occurrences (property usage), where the plurality is 4,659 predicates appearing only once; Table A.2 gives the listing of the top 25 predicates, where unsurprisingly, `rdf:type` heads the list, and also where `foaf` properties feature prominently. We found 104,596 unique values for `rdf:type`: Figure A.3 gives the distribution of `rdf:type`-value occurrences (class usage), where the plurality is again 29,856 classes appearing only once; Table A.3 gives the listing of the top 10 classes, where again `foaf` – and in particular `foaf:Person` – dominates the list.

In order to get an insight into the most instantiated vocabularies, we extracted the 'namespace' from predicates and URI-values for `rdf:type` – we simply strip the URI upto the last hash or slash. Table A.4 gives the top-25 occurring namespaces for a cumulative count, where `foaf`, `rdfs`, and `rdf` dominate; in contrast, Table A.5 gives the top 25 namespaces for unique URIs appearing as predicate or value of `rdf:type`, where in particular DBpedia, Yago and Freebase related namespaces offer a diverse set of instantiated terms; note that (i) the ontology mentioned in Footnote 49 does not appear as its terms are not instantiated, (ii) the terms need not be defined in that namespace (or may be misspelt versions of defined terms) [76], and (iii) we found 460 quads with predicates of the form `rdf:_n`.

PLD	quads	documents	quads/document
hi5.com	595,086,038	255,742	2,327
livejournal.com	77,711,072	56,043	1,387
opiumfield.com	66,092,693	272,189	243
linkedlifedata.com	54,903,837	253,392	217
bio2rdf.org	50,659,976	227,307	223
rdfize.com	38,107,882	161,931	235
appspot.com	28,734,556	49,871	576
identi.ca	22,873,875	65,235	351
freebase.com	18,567,723	181,612	102
rdfabout.com	16,539,018	135,288	122
ontologycentral.com	15,981,580	1,080	14,798
opera.com	14,045,764	82,843	170
dbpedia.org	13,126,425	144,850	91
qdos.com	11,234,120	14,360	782
l3s.de	8,341,294	163,240	51
dbtropes.org	7,366,775	34,013	217
uniprot.org	7,298,798	11,677	625
dbtune.org	6,208,238	181,016	34
vox.com	5,327,924	44,388	120
bbc.co.uk	4,287,872	262,021	16
geonames.org	4,001,272	213,061	19
ontologyportal.org	3,483,480	2	1,741,740
ordnancesurvey.co.uk	2,880,492	43,676	66
loc.gov	2,537,456	166,652	15
fu-berlin.de	2,454,839	135,539	18

Table A.1

Top twenty-five PLDs and (i) the number of quads they provide, (ii) the number of documents they provide, (iii) the average quads per document.
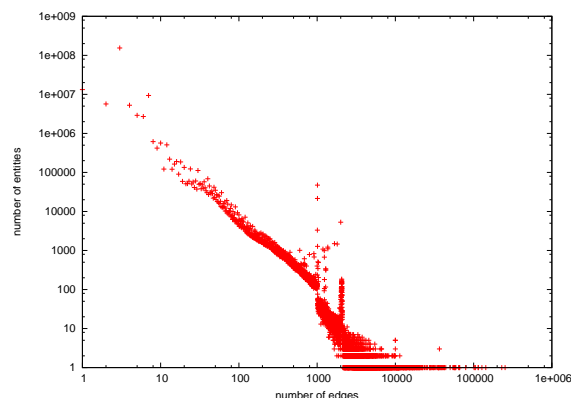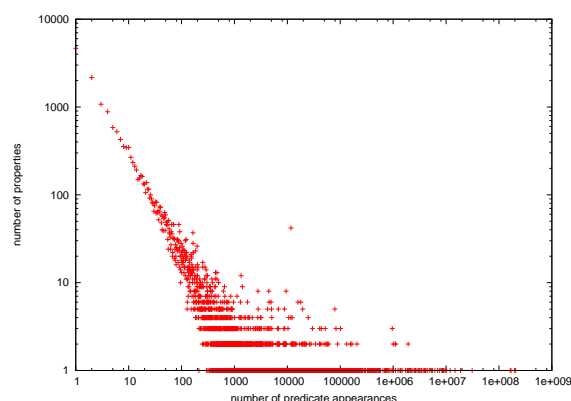

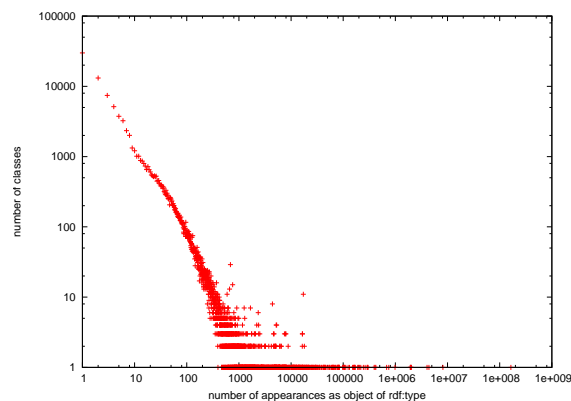Fig. A.1. Entity size distribution

Fig. A.2. Property usage distribution

Fig. A.3. Class usage distribution

predicate	count
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	206,799,100
http://www.w3.org/2000/01/rdf-schema#seeAlso	199,957,728
http://xmlns.com/foaf/0.1/knows	168,512,114
http://xmlns.com/foaf/0.1/nick	163,318,560
http://bio2rdf.org/bio2rdf_resource:linkedToFrom	31,100,922
http://linkedlifedata.com/resource/entrezgene/pubmed	18,776,328
http://www.w3.org/2000/01/rdf-schema#label	14,736,014
http://www.w3.org/2002/07/owl#sameAs	11,928,308
http://xmlns.com/foaf/0.1/name	10,192,187
http://xmlns.com/foaf/0.1/weblog	10,061,003
http://xmlns.com/foaf/0.1/homepage	9522912
http://linkedlifedata.com/resource/pubmed/chemical	8,910,937
http://xmlns.com/foaf/0.1/member_name	8,780,863
http://xmlns.com/foaf/0.1/tagLine	8,780,817
http://xmlns.com/foaf/0.1/depiction	8,475,063
http://xmlns.com/foaf/0.1/image	8,383,510
http://xmlns.com/foaf/0.1/maker	7,457,837
http://linkedlifedata.com/resource/pubmed/journal	6,917,754
http://xmlns.com/foaf/0.1/topic	6,163,769
http://linkedlifedata.com/resource/pubmed/keyword	5,560,144
http://purl.org/dc/elements/1.1/title	5,346,271
http://xmlns.com/foaf/0.1/page	4,923,026
http://bio2rdf.org/ns/bio2rdf:linkedToFrom	4,510,169
http://www.w3.org/2004/02/skos/core#subject	4,158,905
http://www.w3.org/2004/02/skos/core#prefLabel	4,140,048

Table A.2

Top twenty-five predicates encountered (properties).

namespace	count
http://xmlns.com/foaf/0.1/	615,110,022
http://www.w3.org/2000/01/rdf-schema#	219,205,911
http://www.w3.org/1999/02/22-rdf-syntax-ns#	213,652,227
http://bio2rdf.org/	43,182,736
http://linkedlifedata.com/resource/pubmed/	27,944,794
http://linkedlifedata.com/resource/entrezgene/	22,228,436
http://www.w3.org/2004/02/skos/core#	19,870,999
http://rdf.freebase.com/ns/	17,500,405
http://www.w3.org/2002/07/owl#	13,140,895
http://rdf.opiumfield.com/lastfm/spec#	11,594,699
http://purl.org/ontology/mo/	11,322,417
http://purl.org/dc/elements/1.1/	9,238,140
http://ontologycentral.com/2009/01/eurostat/ns#	9,175,574
http://purl.org/dc/terms/	6,400,202
http://bio2rdf.org/ns/	5,839,771
http://rdfs.org/sioc/ns#	5,411,725
http://www.rdfabout.com/rdf/schema/vote/	4,057,450
http://www.geonames.org/ontology#	3,985,276
http://skipforward.net/skipforward/.../skipinions/	3,466,560
http://dbpedia.org/ontology/	3,299,442
http://purl.uniprot.org/core/	2,964,084
http://ontologycentral.com/.../table_of_contents#	2,630,198
http://linkedlifedata.com/resource/lifeskim/	2,603,123
http://pervasive.semanticweb.org/ont/2004/06/time#	2,519,543
http://dbpedia.org/property/	2,371,396

Table A.4

Top twenty-five namespaces for *cumulative* count of URIs found in predicate position (property), or as the value of `rdf:type` (class).

class	count
http://xmlns.com/foaf/0.1/Person	163,699,161
http://xmlns.com/foaf/0.1/Agent	8,165,989
http://www.w3.org/2004/02/skos/core#Concept	4,402,201
http://purl.org/ontology/mo/MusicArtist	4,050,837
http://xmlns.com/foaf/0.1/PersonalProfileDocument	2,029,533
http://xmlns.com/foaf/0.1/OnlineAccount	1,985,390
http://xmlns.com/foaf/0.1/Image	1,951,773
http://rdf.opiumfield.com/lastfm/spec#Neighbour	1,920,992
http://www.geonames.org/ontology#Feature	983,800
http://xmlns.com/foaf/0.1/Document	745,393
http://www.w3.org/2002/07/owl#Thing	679,520
http://ontologycentral.com/...of_contents#cphi_m	421,193
http://purl.org/goodrelations/v1#ProductOrServiceModel	407,327
http://purl.org/ontology/mo/Performance	392,416
http://rdf.freebase.com/ns/film.performance	300,608
http://rdf.freebase.com/ns/tv.tv_guest_role	290,246
http://rdf.freebase.com/ns/common.webpage	288,537
http://purl.org/dc/dcmitype/Text	262,517
http://ontologycentral.com/...of_contents#irt_h_euryld_d	243,074
http://www.w3.org/2002/07/owl#Class	217,334
http://www.ontologyportal.org/WordNet.owl#WordSense	206,941
http://www.rdfabout.com/.../usbill/LegislativeAction	193,202
http://rdf.freebase.com/ns/common.topic	190,434
http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement	169,376
http://www.kanzaki.com/ns/music#Venue	166,374

Table A.3

Top twenty-five values for `rdf:type` encountered (classes).

namespace	count
http://www.mpii.de/yago/resource/	41,483
http://dbpedia.org/class/yago/	33,499
http://dbtropes.org/resource/Main/	16,401
http://rdf.freebase.com/ns/	14,884
http://dbpedia.org/property/	7,176
http://www.ontologydesignpatterns.org.it/.../own16.owl#	1,149
http://semanticweb.org/id/	1,024
http://sw.opencyc.org/2008/06/10/concept/	937
http://ontologycentral.com/.../table_of_contents#	927
http://dbpedia.org/ontology/	843
http://www.ontologyportal.org/SUMO.owl#	581
http://www.w3.org/1999/02/22-rdf-syntax-ns#	489
http://bio2rdf.org/	366
http://xmlns.com/wordnet/1.6/	303
http://data.linkedmdb.org/resource/movie/	251
http://purl.uniprot.org/core/	207
http://www.aifb.kit.edu/id/	196
http://www4.wiwiss.fu-berlin.de/factbook/ns#	191
http://xmlns.com/foaf/0.1/	150
http://rdf.geospecies.org/ont/geospecies#	143
http://bio2rdf.org/ns/	133
http://skipforward.net/skipforward/resource/ludopinions/	132
http://www.openlinksw.com/schemas/oplweb#	130
http://ontologycentral.com/2009/01/eurostat/ns#	127
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/	119

Table A.5

Top twenty-five namespaces for *unique* URIs found in predicate position (property), or as the value of `rdf:type` (class).

Appendix B. Rule Tables

Herein, we list the rule tables referred to in Section 8, including rules with no antecedent ($\mathcal{R}^{\emptyset}$ – Table B.1), rules with only terminological patterns ($\mathcal{R}^{\mathcal{T}\emptyset}$ – Table B.2), and rules with terminological pattern(s) and one assertional pattern ($\mathcal{R}^{\mathcal{T}\mathcal{G}^1}$ – Table B.3). Also, in Table B.4, we give an indication as to how recursive application of rules in $\mathcal{R}^{\mathcal{T}\mathcal{G}^1}$ can be complete, even if the inferences from rules in $\mathcal{R}^{\mathcal{T}\emptyset}$ are omitted from the T-Box.

$\mathcal{R}^{\emptyset}$: no antecedent		
OWL2RL	**Consequent**	**Notes**
prp-ap	p a :AnnotationProperty .	For each built-in annotation property
cls-thing	:Thing a :Class .	-
cls-nothing	:Nothing a :Class .	-
dt-type1	dt a rdfs:Datatype .	For each built-in datatype
dt-type2	l a dt .	For all l in the value space of datatype dt
dt-eq	l_1 :sameAs ?l_2 .	For all l_1 and l_2 with the same data value
dt-diff	l_1 :differentFrom ?l_2 .	For all l_1 and l_2 with different data values

Table B.1
Rules with no antecedent

$\mathcal{R}^{\mathcal{T}\emptyset}$: only terminological patterns in antecedent		
OWL2RL	**Antecedent** *terminological*	**Consequent**
cls-00	c :oneOf $(x_1...x_n)$.	$x_1...x_n$ a c .
scm-cls	c a :Class .	c rdfs:subClassOf c ; rdfs:subClassOf :Thing ; :equivalentClass c . :Nothing rdfs:subClassOf c .
scm-sco	c_1 rdfs:subClassOf c_2 . c_2 rdfs:subClassOf c_3 .	c_1 rdfs:subClassOf c_3 .
scm-eqc1	c_1 :equivalentClass c_2 .	c_1 rdfs:subClassOf c_2 . c_2 rdfs:subClassOf c_1 .
scm-eqc2	c_1 rdfs:subClassOf c_2 . c_2 rdfs:subClassOf c_1 .	c_1 :equivalentClass c_2 .
scm-op	p a :ObjectProperty .	p rdfs:subPropertyOf p . p :equivalentProperty p .
scm-dp	p a :DatatypeProperty .	p rdfs:subPropertyOf p . p :equivalentProperty p .
scm-spo	p_1 rdfs:subPropertyOf p_2 . p_2 rdfs:subPropertyOf p_3 .	p_1 rdfs:subPropertyOf p_3 .
scm-eqp1	p_1 :equivalentProperty p_2 .	p_1 rdfs:subPropertyOf p_2 . p_2 rdfs:subPropertyOf p_1 .
scm-eqp2	p_1 rdfs:subPropertyOf p_2 . p_2 rdfs:subPropertyOf p_1 .	p_1 :equivalentProperty p_2 .
scm-dom1	p rdfs:domain c_1 . c_1 rdfs:subClassOf c_2 .	p rdfs:domain c_2 .
scm-dom2	p_2 rdfs:domain c . p_1 rdfs:subPropertyOf p_2 .	p_1 rdfs:domain c .
scm-rng1	p rdfs:range c_1 . c_1 rdfs:subClassOf c_2 .	p rdfs:range c_2 .
scm-rng2	p_2 rdfs:range c . p_1 rdfs:subPropertyOf p_2 .	p_1 rdfs:range c .
scm-hv	c_1 :hasValue i ; :onProperty p_1 . c_2 :hasValue i ; :onProperty p_2 . p_1 rdfs:subPropertyOf p_2 .	c_1 rdfs:subClassOf c_2 .
scm-svf1	c_1 :someValuesFrom y_1 ; :onProperty p . c_2 :someValuesFrom y_2 ; :onProperty p . y_1 rdfs:subClassOf y_2 .	c_1 rdfs:subClassOf c_2 .
scm-svf2	c_1 :someValuesFrom y ; :onProperty p_1 . c_2 :someValuesFrom y ; :onProperty p_2 . p_1 rdfs:subPropertyOf p_2 .	c_1 rdfs:subClassOf c_2 .
scm-avf1	c_1 :allValuesFrom y_1 ; :onProperty p . c_2 :allValuesFrom y_2 ; :onProperty p . y_1 rdfs:subClassOf y_2 .	c_1 rdfs:subClassOf c_2 .
scm-avf2	c_1 :allValuesFrom y ; :onProperty p_1 . c_2 :allValuesFrom y ; :onProperty p_2 . p_1 rdfs:subPropertyOf p_2 .	c_1 rdfs:subClassOf c_2 .
scm-int	c :intersectionOf $(c_1...c_n)$.	c rdfs:subClassOf $c_1...c_n$.
scm-uni	c :unionOf $(c_1...c_n)$.	$c_1...c_n$ rdfs:subClassOf c .

Table B.2
Rules containing only terminological antecedent patterns

$\mathcal{R}^{\mathcal{TG}^1}$: at least one terminological *and* exactly one assertional pattern in antecedent			
OWL2RL	**Antecedent**		**Consequent**
	terminological	*assertional*	
prp-dom	p rdfs:domain c .	x p y .	x a c .
prp-rng	p rdfs:range c .	x p y .	y a c .
prp-symp	p a :SymmetricProperty .	x p y .	y p x .
prp-spo1	p_1 rdfs:subPropertyOf p_2 .	x p_1 y .	x p_2 y .
prp-eqp1	p_1 :equivalentProperty p_2 .	x p_1 y .	x p_2 y .
prp-eqp2	p_1 :equivalentProperty p_2 .	x p_2 y .	x p_1 y .
prp-inv1	p_1 :inverseOf p_2 .	x p_1 y .	y p_2 x .
prp-inv2	p_1 :inverseOf p_2 .	x p_2 y .	y p_1 x .
cls-int2	c :intersectionOf $(c_1$ ... $c_n)$.	x a c .	x a $c_1...c_n$.
cls-uni	c :unionOf $(c_1...c_i...c_n)$.	x a c_i	x a c .
cls-svf2	x :someValuesFrom :Thing ; :onProperty p .	u p v .	u a x .
cls-hv1	x :hasValue y ; :onProperty p .	u a x .	u p y .
cls-hv2	x :hasValue y ; :onProperty p .	u p y .	u a x .
cax-sco	c_1 rdfs:subClassOf c_2 .	x a c_1 .	x a c_2 .
cax-eqc1	c_1 :equivalentClass c_2 .	x a c_1 .	x a c_2 .
cax-eqc2	c_1 :equivalentClass c_2 .	x a c_2 .	x a c_1 .

Table B.3

Rules with at least one terminological and exactly one assertional pattern in the antecedent

$\mathcal{R}^{\mathcal{T}\emptyset}$ coverage in $\mathcal{R}^{\mathcal{TG}^1}$	
$\mathcal{R}^{\mathcal{T}\emptyset}$	covered by $\mathcal{R}^{\mathcal{TG}^1}$ rules
scm-cls	*incomplete* for `owl:Thing` membership inferences [a]
scm-sco	**cax-sco**
scm-eqc1	**cax-eqc1, cax-eqc2**
scm-eqc2	**cax-sco**
scm-op	*no unique inferences*
scm-dp	*no unique inferences*
scm-spo	**prp-spo1**
scm-eqp1	**prp-eqp1, prp-eqp2**
scm-eqp2	**prp-spo1**
scm-dom1	**prp-dom, cax-sco**
scm-dom2	**prp-dom, prp-spo1**
scm-rng1	**prp-rng, cax-sco**
scm-rng2	**prp-rng, prp-spo1**
scm-hv	**prp-rng, prp-spo1**
scm-svf1	*incomplete:* <u>**cls-svf1**</u>, **cax-sco**
scm-svf2	*incomplete:* <u>**cls-svf1**</u>, **prp-spo1**
scm-avf1	*incomplete:* <u>**cls-avf**</u>, **cax-sco**
scm-avf2	*incomplete:* <u>**cls-avf**</u>, **prp-spo1**
scm-int	**cls-int2**
scm-uni	**cls-uni**

Table B.4

Coverage of rules in $\mathcal{R}^{\mathcal{T}\emptyset}$ by rules in $\mathcal{R}^{\mathcal{TG}^1}$: underlined rules are not supported, and thus we would encounter incompleteness by not including the inferences of the respective $\mathcal{R}^{\mathcal{T}\emptyset}$ rule in the T-Box (would not affect a full OWL 2 RL/RDF reasoner which includes the underlined rules).

[a] In our scenario, are not concerned – we consider such triples as tautological and they cannot lead to further inferences in our authoritative reasoning with the given rules.