Mind the Data Skew: Distributed Inferencing by Speeddating in Elastic Regions

Spyros Kotoulas
kot@few.vu.nl

Eyal Oren
eyal@cs.vu.nl

Frank van Harmelen
Frank.van.Harmelen@cs.vu.nl

Department of Computer Science
VU University Amsterdam

ABSTRACT

Semantic Web data exhibits very skewed frequency distributions among terms. Efficient large-scale distributed reasoning methods should maintain load-balance in the face of such highly skewed distribution of input data. We show that term-based partitioning, used by most distributed reasoning approaches, has limited scalability due to load-balancing problems.

We address this problem with a method for data distribution based on clustering in elastic regions. Instead of assigning data to fixed peers, data flows semi-randomly in the network. Data items "speed-date" while being temporarily collocated in the same peer. We introduce a bias in the routing to allow semantically clustered neighborhoods to emerge. Our approach is self-organising, efficient and does not require any central coordination.

We have implemented this method on the MaRVIN platform and have performed experiments on large real-world datasets, using a cluster of up to 64 nodes. We compute the RDFS closure over different datasets and show that our clustering algorithm drastically reduces computation time, calculating the RDFS closure of 200 million triples in 7.2 minutes.

Categories and Subject Descriptors

I.2.3 [**Deduction and Theorem Proving**]: Inference Engines; E.1 [**Data Structures**]: Distributed data structures; C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Peer-to-peer, Distributed, Reasoning, Self-organisation

1. INTRODUCTION

This paper deals with distributed reasoning over large-scale RDF datasets. Several approaches have been proposed for distributed reasoning, based for example on graph partitioning [12], peer-to-peer networks such as distributed hashtables (DHTs) [2, 3] or task-farming frameworks such as MapReduce [18].

These approaches distribute the reasoning load over several compute nodes ("peers") by partitioning the data across those peers; the partitioning is typically based on fixed mappings from triple terms to peers (term-based partitioning), sending all triples with some term to the same peer. Term-based partitioning seems appropriate since the RDFS/OWL-Horst rules [8, 17] require joins on at least one common term: locating all triples that share a term on the same peer allows the peer to perform the join.

However, as we will show in Section 3, the frequency distribution of terms in RDF data is highly skewed: some terms occur much more often than others. As we discuss in Section 4, this skewed term distribution prevents scalability for any algorithm that uses term-based partitioning, since some "unlucky" peers will receive all triples corresponding to a popular term.

In this paper, we propose a solution based on *speeddating in elastic regions*, that is robust against such highly skewed data-distributions. In short, data is not deterministically routed to one fixed peer, but instead is attracted to a region of peers; the regions for each term stretch dynamically based on the population density: if more triples arrive, more peers are recruited and the region is stretched; once triples arrive at a region, they "speeddate" with other triples to infer conclusions. This algorithm is explained in Section 5.

The key points of our solution are:

- no (central) coordination is needed to construct and maintain the regions: the regions emerge dynamically.
- the stretching of regions is dynamic, without requiring upfront data analysis to determine and assign partitions;
- the probability of triples meeting each other remains high, even though triples are not assigned to one specific peer;

After discussing related work in Section 2, we will analyse some realistic RDF datasets in Section 3 and show that they are indeed highly skewed. In Section 4 we analyse the problems this causes for deterministic term-based data-partitioning, including both a theoretical scalability upperbound, and an analysis of how this upperbound works out on some realistic RDF datasets. In Section 5 we present our solution to these problems: speeddating in elastic regions, and we present simulated results on the behaviour of this approach. In Section 6 we run experiments using realistic datasets. We compare our speeddating approach to random exchanges, which are relatively inefficient but do not suffer

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

from skewed data distributions, and to deterministic term-based partitioning, which is efficient but cannot scale in the face of skewed distributions. We evaluate performance by calculating the RDFS closure of large real-world datasets (of up to 200M triples) on a cluster of up to 64 compute nodes. These experiments show that the speeddating algorithm results in competitive performance while maintaining near perfect load-balancing. Furthermore, our approach scales well with increasing numbers of compute nodes. To our knowledge, these results are the largest reported deployment of P2P reasoning, in terms of data size.

2. RELATED WORK

Several techniques have been proposed for RDFS reasoning using distributed hashtables [2, 3, 5, 10, 1]. These approaches distribute the data across peers using term-based partitioning, sending all triples with some term to the same peer. These approaches seem to neglect the highly skewed data distribution in real-world RDF datasets. We will discuss this issue in detail in Section 4.

MacCartney *et al* [12] show that graph-partitioning techniques improve reasoning over first-order logic knowledge bases, but do not apply this in a distributed or large-scale context. Some *et al* [16] present a technique for parallel OWL inferencing through partitioning, inspired by similar techniques for Datalog [6]. They experiment with both data partitioning (each peer gets a different chunk of data and applies all rules) and rule partitioning (each peer gets all data but applies only some rules), using different partitioning techniques. Experimental results show good speedup (both sub- and super-linear, depending on the dataset) but on relatively small datasets (1M triples); runtime is not reported, nor scalability over larger datasets. In contrast, our approach is scalable and does not require the expensive up-front partitioning of the data: our partitions emerge dynamically, making our approach usable with node churn and changing datasets.

Weaver and Hendler [19] present straightforward parallel RDFS reasoning on a supercomputer. Their approach replicates all schema triples to all processing nodes and distributes instance triples randomly. Each node calculates the closure using a conventional reasoner and the results are merged. In contrast to our approach, it does not compute the complete RDFS closure, since triples extending the RDFS schema are ignored. They evaluate their system using the LUBM benchmark and report calculating the closure of 172M triples on 64 nodes in 262 seconds if I/O time is not included and 466 seconds if I/O time is included. This amounts for a throughput of 656Ktps (thousand triples per second) with I/O time excluded and 369Ktps with I/O time included.

Urbani *et al* [18] show a parallel implementation for RDFS reasoning using the MapReduce programming paradigm. For the DBpedia dataset (150M triples) on 64 nodes, they report a runtime of 156 seconds. This amounts for a throughput of 961Ktps. In this system, terms are rewritten using a dictionary encoding. The time required for this was not taken into consideration for calculating the throughput. As an indication, dictionary encoding 850M triples on 32 nodes took one hour.

As we will report in Section 6.3.4, the throughput of our system is 450Ktps. Our system outperforms all RDF stores mentioned in [1]. Nevertheless, a direct comparison is not meaningful since these stores do more than materialization; namely, they index triples to resolve queries.

System	Size	Throughput
Our approach	195M	451Ktps
Weaver et al.	172M	369Ktps
Urbani et al.	150M	961Ktps

Table 1: Comparison of triple throughput on 64 nodes

A more direct comparison can be done with the systems by Urbani *et al* [18] and Weaver and Hendler [19], since they both do RDF materialisation. Table 1 shows the triple throughput of each system for their best performing dataset and comparable input size. In terms of triple throughput, our system is in league with the competition. In addition, our approach has some critical advantages:

- The approaches by Weaver *et al.* and Urbani *et al.* are optimized for RDFS reasoning. Specifically, the work by Urbani *et al.* circumvents load-balancing problems by exploiting properties of large RDF datasets (e.g. the schema is much smaller than instance data). Our approach is generic and can be applied to any monotonic logic.
- Our approach does not have a central coordination component, which could become a bottleneck and a single point of failure.
- In our approach, the input is constantly evaluated. Thus, adding data while the system is running is possible.

In this paper, we focus on P2P data partitioning and routing strategies to maximise the number of triple joins ("meetings") on each peer, while maintaining load-balance in terms of storage. This is independent from the particular reasoners that are deployed on the nodes.

3. TERM DISTRIBUTIONS IN RDF DATA ARE HIGHLY SKEWED

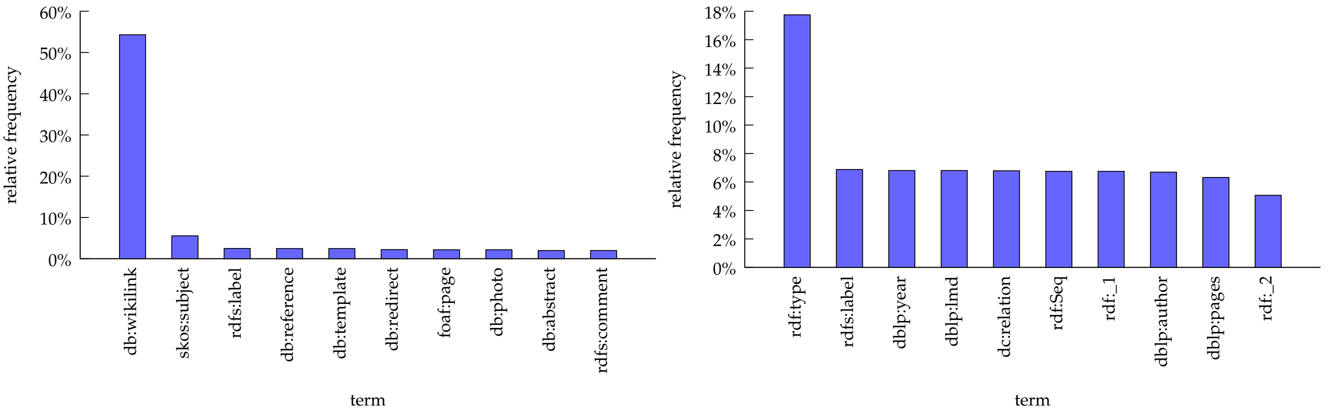
It is well-known that the frequency distribution of terms in RDF data is highly skewed: some terms occur much more often than others. In typical datasets, term occurrence and other related frequencies seem to follow a power-law [13, 4].

To confirm this common knowledge, we have counted term frequency in three large datasets used in the 2008 Billion Triple challenge[2], namely the DBpedia, Geonames, and SwetoDBLP datasets. We have also aggregated these three datasets for later experiments; finally, we also counted term frequency in all Billion Triple datasets combined, consisting of 865M triples in total. Table 2 and Figure 1 give an overview of these datasets and the fraction of triples that contain the most-popular term.

Most popular terms are URIs. In the DBpedia dataset, the most common term is `dbpedia:wikilink`, appearing in 55% of the triples; in Geonames it is `geo:lat`, appearing

[1] `http://esw.w3.org/topic/LargeTripleStores`

[2] `http://www.cs.vu.nl/~pmika/swc/btc.html`



(a) (b)

Figure 1: Top-10 terms in DBpedia and SwetoDBLP

dataset	size	$f_{topterm}$	top term
SwetoDBLP	14.9M	17%	`rdf:type`
Geonames	69.8M	9%	`geo:lat`
DBpedia	110.2M	55%	`dbpedia:wikilink`
combined	194.9M	31%	`dbpedia:wikilink`
all	864.8M	14%	`"0"^^xsd:integer`

Table 2: Datasets, with size (in nr. of triples) and fraction of most popular term

in 9% of triples, in SwetoDBLP it is `rdf:type`, appearing in 17% of the triples. In all datasets combined (including data from Web crawls, US census, Freebase, etc.) the most frequent term was the literal “0”, which occurred in 14% of the dataset (123M triples); next was `rdf:value` in 7% (63M triples), `dbpedia:wikilink` in 7%, `rdf:type` in 5%, and `dc:title` in 4% of the triples.

As we will explain in Section 4.3, these numbers are bad news indeed for any deterministic term-based partitioning strategy. The single peer responsible for the most popular term would by itself be responsible for anywhere between 9–55% of the entire dataset, irrespective of how many other peers are available for the other terms. We will also show that load-balancing techniques for DHTs do not help either.

4. LIMITATIONS OF TERM-BASED PARTITIONING

A distributed reasoner scales by distributing the computation. To distribute the computation, the data must be partitioned over the peers. However, to produce a logical conclusion, triples involved in the premisses must be located at the same peer.

Most approaches for data partitioning described in the literature apply term-based partitioning, grouping triples based on (a combination of) their terms. Term-based partitioning is analogous to the database indices maintained by typical RDF stores, which group triples on terms and combinations of terms (compound keys). Given the skewed distributions of data (shown in section 3), these groups will be of very uneven size. In a centralised setting, term-based partitions are not a problem since all groups share the same storage space. However, as we will see, in a distributed setting, term-based partitioning leads to load-balancing problems.

Most distributed reasoning approaches also use term-based partitioning, assigning all data in a partition to some specific peer. This peer is either chosen in a centralised manner (parallel computing) or decentralised (DHT). Most approaches for distributed RDFS/OWL reasoning use term-based partitioning on top of DHTs. We focus on the partitioning strategy, irrespective of the underlying network overlay. We will show that term-based partitioning leads to severe storage load-balancing problems and does not scale for Semantic Web data.

4.1 Reasoning with term-based partitioning

Distributed hashtables can be used to partition triples in a distributed manner, using the terms (subject, predicate or object) of the triples as keys for the DHT. In this way, all triples with a given term will be located in the same peer, and that peer can draw all conclusions that can be derived from those triples [5, 10, 1].

When using term-based partitioning, RDFS reasoning is straightforward. As shown in Table 3, all RDFS rules have either one or two antecedents. For rules with only one antecedent no join is needed, and each peer that hosts that triple can infer the consequent. For rules with two antecedents, the antecedents always share one or more terms; the triples will therefore be co-located on the peer responsible for that shared term.

The basic algorithm for forward-chaining RDFS reasoning with term-based partitioning is thus: (a) each triple is inserted three times on the DHT, using each of its terms as a key, (b) peers apply all RDFS rules locally, on their data, (c) inferred triples are also inserted (three times) in the DHT; these steps are repeated until a fixpoint is reached and no new triples are inferred. A similar algorithm can be used for OWL reasoning, as demonstrated by Fang *et al* [5].

4.2 Where does it go wrong?

In principle, forward-chaining based on term-based partitioning works. However, as explained, triple terms follow a highly skewed distribution. In fact, we can expect the most popular term to appear in around 10–20% of all triples (see

1:	s p o (if o is a literal)		⇒ _:n rdf:type rdfs:Literal
2:	p rdfs:domain x	& s p o	⇒ s rdf:type x
3:	p rdfs:range x	& s p o	⇒ o rdf:type x
4a:	s p o		⇒ s rdf:type rdfs:Resource
4b:	s p o		⇒ o rdf:type rdfs:Resource
5:	p rdfs:subPropertyOf q	& q rdfs:subPropertyOf r	⇒ p rdfs:subPropertyOf r
6:	p rdf:type rdf:Property		⇒ p rdfs:subPropertyOf p
7:	s p o	& p rdfs:subPropertyOf q	⇒ s q o
8:	s rdf:type rdfs:Class		⇒ s rdfs:subClassOf rdfs:Resource
9:	s rdf:type x	& x rdfs:subClassOf y	⇒ s rdf:type y
10:	s rdf:type rdfs:Class		⇒ s rdfs:subClassOf s
11:	x rdfs:subClassOf y	& y rdfs:subClassof z	⇒ x rdfs:subClassOf z
12:	p rdf:type rdfs:ContainerMembershipProperty		⇒ p rdfs:subPropertyOf rdfs:member
13:	o rdf:type rdfs:Datatype		⇒ o rdfs:subClassOf rdfs:Literal

Table 3: RDFS rules [8]

Table 2). Using term-based partitioning, a single peer will be responsible to store all triples with this term. This will create a significant data load imbalance that will inhibit the scalability of the system. In general, the peers responsible for popular terms will become "hotspots": they will have to store a large portion of the complete dataset and their data will be requested very frequently.

In the ideal situation, a single peer will have to store only all triples with one term (typically though, each peer will store triples for many keys). Even then, the maximum capacity of the system will be limited by the maximum capacity of a single peer, namely the peer storing the most popular term. This observation holds for any system that partitions data using fixed keys:

Proposition 1 (Limited scalability). *Assume a distributed system storing key/value pairs. Data must be distributed over the peers such that all items with the same key reside on the same peer. Let each peer have some fixed capacity for storing data. Let $f_{topterm}$ denote the frequency of the most popular key in the system. Any distributed system that deterministically partitions data on keys will be able to handle at most $1/f_{topterm}$ more data than the corresponding centralised approach, regardless of the number of peers.*

Proof. *Let us assume that each peer can handle a maximum of c data. Hence, the centralised system (i.e. a system with one peer) will be able to handle at most c data. We will assume an ideal distributed system with complete knowledge of the key distribution. Such a system will assign the complete capacity of one peer to the most frequent key. It is not possible to assign several peers because that would break the premise that all items with the same key should be located at the same peer. Let us call the total amount of data in the distributed system N. By definition, we have that the frequency of the most common term would be the number of its occurrences in the dataset divided by the size of the dataset ($f_{topterm} = N(topterm)/N$). Since all items with the most popular key will need to be stored at the same peer, we also have that $N(topterm) \leq c$. Thus $f_{topterm} \cdot N \leq c$, thus $N \leq c \cdot 1/f_{topterm}$.*

This "hotspot" problem was identified by Cai [2], but only in the context of query answering. Their solution (ignore popular keys and ignore queries over those keys) does not work for the reasoning scenario, since popular terms are used for reasoning (e.g. consider RDFS rule 2 or 3 and `db:wikilink` as `p`).

dataset	triples	$f_{topterm}$	max. triples
SwetoDBLP	14.9M	17%	7.1M
Geonames	69.8M	9%	13.3M
DBpedia	110.2M	55%	2.1M
combined	194.9M	31%	3.9M
all	864.8M	14%	8.6M

Table 4: Maximum triple capacity for systems doing deterministic term partitioning

4.3 Effect on distributed reasoning

We will now show the consequences of this theoretical analysis on realistic RDF datasets. The compute nodes used in our experiments (see Section 6) have a reasoner capacity of 100.000 triples. Furthermore, they are able to store up to 1.100.000 triples in in-memory buffers. Any additional triples have to be swapped to disk in a random manner (for performance, we do not maintain a disk index). Thus, we will assume a capacity $c = 1.200.000$ triples for the deterministic system. According to Proposition 1, the maximum size of the system is inversely proportional to the relative frequency of the most popular term. Table 4 shows the resulting maximum attainable size for a system with deterministic term-based partitioning, based on the observed term distributions. It is obvious that such systems cannot handle web-scale datasets such as the ones mentioned in Table 4. Note that these numbers even ignore the derived triples.

Since we can expect the most-frequent term to occur in 10–20% of the triples, we can conclude from Proposition 1 that regardless of the number of peers, a distributed system with term-based partitioning cannot cope with more than 5–10x the triples of a centralised approach. The effect of this limitation are:

- adding more peers will not alleviate this problem, since the triples with the most popular term are still routed to the same peer.
- standard techniques for load-balancing in DHTs [11] will not improve the situation: typically, they either adjust the key range that one peer handles (for which we have already assumed a perfect mechanism) or make multiple peers responsible for a key. The latter will

also not help, since then the keys will not be located in the same peer, which is required for performing the subsequent inference steps.

- the situation seems almost a Catch-22 [9]: the triples cannot be at the same peer, because they do not fit together; if we distribute them over many peers, they fit, but then they do not meet; if we make them meet in some peer, they will again not fit.
- subdividing the keys into multiple partitions, and distributing those partitions across multiple peers, will not help, since URIs are atomic identifiers: the joins take place at the level of URIs and we have no mechanism to further partition these URIs. Partitioning on compound keys (*sp*, *po*, ...) instead of single keys would lead to the same problem.

Ideally, we would like to keep the optimal clustering of term-based partitioning (triples with shared terms go to one peer, where they meet), while alleviating the load caused by popular terms by clustering these over as many peers as needed. Our solution, presented next, takes exactly this approach.

5. SOLUTION: SPEEDDATING IN ELASTIC REGIONS

As described above, any distributed system relying on term-based partitioning suffers from limited scalability because a single peer must store all items with the most-popular key. Nevertheless, in order to make inferences, triples with the same key should be collocated.

In this section, we present our "speeddate" approach which introduces a temporal dimension to triple collocation: we require that *triples with the same key are collocated at some point in time*. Instead of deterministically routing data to a fixed peer, randomised data exchanges are used to eventually collocate data with the same key. Furthermore, we introduce a routing policy that clusters data during these random exchanges to increase the probability that data with the same key are collocated.

5.1 Speeddating through constant data exchanges

In our method, autonomous peers constantly re-partition the data by randomly exchanging data. Through these data exchanges, triples that produce inference will be eventually collocated. In previous work, we have shown that, for a system performing random exchanges, this collocation happens in finite time [14]. Random exchanges are load-balanced but inefficient; we increase efficiency by clustering data, as explained in the next subsection.

The "speeddate" approach leads to the memory hierarchy shown in Figure 2: together, all peers can store billions of triples, and exchange these over the network. Reasoning is then performed in-memory on each peer; for this, peers have a limited capacity of some hundreds of thousands of triples. The total number of triples that can be kept in-memory is an order of magnitude higher (because less indices are needed than for reasoning). Assuming that the total memory capacity of all peers is less than the space required to store all triples, some data needs to be paged to disk. This data is not indexed, since this would incur significant overhead.

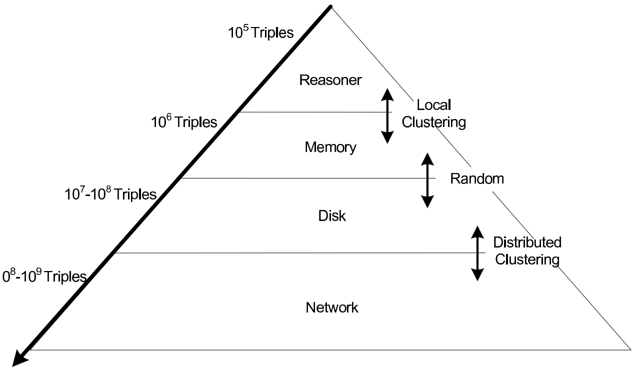


Figure 2: Memory hierarchy in the "speeddate" approach

5.2 Clustering in elastic regions

Despite the guarantee of eventual completeness, completely random partitioning is inefficient and not scalable: as the number of peers, triples and keys increases, the chance of relevant triples being collocated over time decreases asymptotically.

We introduce two methods to improve the chance that triples with the same key are collocated, shown in see Figure 2. The first performs load-balanced distributed clustering across the network. The second performs local clustering by selecting the data that will be loaded to the reasoner from memory.

5.2.1 Distributed clustering

In this section, we will describe a fully distributed and load-balanced partitioning method in which triples with a shared term are likely to end up in the same peer (Algorithm 1).

The main idea is that peers do not exchange random data, but exchange data which, to their knowledge, is most relevant for the receiver. The peer knowledge is very lightweight, namely a numeric identifier for each peer, assigned randomly from a 32-bit space. Similarly, terms get a key in the same space, using some hash-function on their string representation. In practice, this means that terms will be "closer" to the identifiers of some peers. Peers then use the numeric ordering over term keys and peer identifiers to decide which triples to exchange.

For each triple in the system, we create three replicas. Each of these replicas is assigned a key using a hash-function on *s*, *p* or *o*, as in term-based partitioning. A request for triples includes the identifier of the requesting peer. The requested peer responds with those triples whose keys are closest to the identifier of the requesting peer. After sending, the responding peer removes the triples from his own storage. Exchanges are repeated ad infinitum.

Using this ordering of peers and terms, and the described response strategy, triples mentioning particular terms will cluster around particular peers (as shown below). To further increase clustering, peers also prefer requesting data from adjacent peers (in the sense of adjacent peer identifiers).

Figure 3(c) shows the data distribution on a simulated run of our algorithm, compared to random exchanges (3a) and deterministic term-based partitioning (3b). The horizontal axis represents the peer identifiers while the vertical axis the

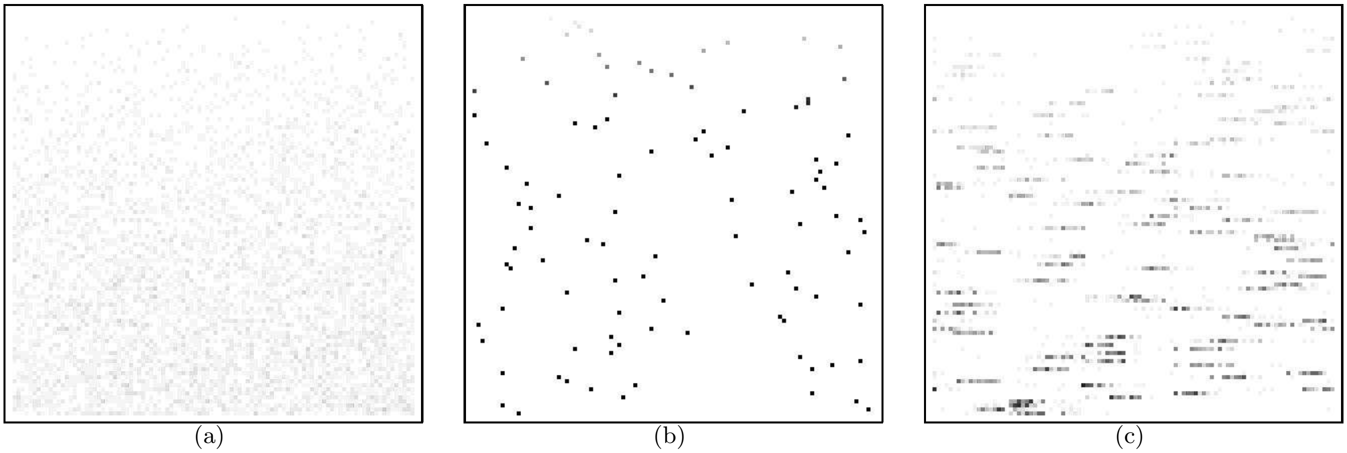


Figure 3: Simulated data distribution with (a) random exchanges, (b) term-based partitioning and (c) our distributed clustering algorithm

Algorithm 1 Distributed clustering

```
Let I be the set of triples in local node, p the set of IDs
of neighboring peers and S the local peer's ID.
procedure MAIN
    while $true$ do
        $p \leftarrow select\_gaussian\_random(P, S)$
        $e \leftarrow arg(min_{e \in I}\{|key(e) - p|\})$   ▷ select the triple
                                                            ▷ with key closest to
                                                            ▷ selected peer's key
        $send(e, p)$
        $I \leftarrow I - e$
    end while
end procedure
procedure RESPOND_TO(p)
    $e \leftarrow arg(min_{e \in I}\{|key(e) - p|\})$
    $send(e, p)$
    $I \leftarrow I - e$
end procedure
```

data keys. The intensity of the dots represents the number of triples with the given key. The keys in the bottom of the graph are more popular than the keys at the top.

In the random exchange approach (left), we can see that keys are uniformly distributed across peers, since dots are more or less of equal intensity. In term-based partitioning approach (middle), we can see that keys are mapped to a specific peer. Although ideal for clustering, as triples with the same key are located on the same peer, this partitioning is detrimental to load balancing, since peers responsible for popular items will have to store excessive amounts of data. For our distributed clustering approach (right), we observe the following:

- *regions emerge*: triples with the same key form horizontal lines, which means that they are stored by adjacent peers. The intensity of the dots is much higher than that of the random approach, but less than that of the term-based partitioning. Triples will float around the clustered regions, while "speeddating" with other triples with the same key.
- *regions are elastic*: the size of the lines depends on the frequency of the key: the lines in the top part of the figure (which refer to infrequent keys) are shorter than those in the bottom. More peers "help" to maintain the most frequent keys, which is beneficial for load balancing. This elastic effect (larger regions for more popular terms) is not obtained by some a priori or central allocation strategy, but is an emergent property of the routing strategy.

The advantages of our approach are the following:

- *no upfront data analysis*: our approach does not require any a priori data analysis. It is not required to know the frequency distribution of keys. Furthermore, it can support heterogeneous networks where not all peers have the same processing and data storage capacity; peers do not even need to be known in advance.
- *no central organisation*: the network self-organises to respond to shifts in data distribution and peers leaving or joining.
- *peer load is balanced*: as we will show in the results, our approach is as well load-balanced as the random approach.
- *more clustered than random*: as we will show in the results, our approach clusters data much better than the random approach, allowing triples to meet (and thus to infer new triples) with very high frequency.

5.2.2 Local clustering

We expect that loading triples with the same terms/keys in the reasoner should lead to more derivations. We therefore perform local term-based partitioning: we index the in-memory triples based on *s*, *p* and *o*; when loading triples into the reasoner, we prefer triples that share terms. Remember that although all triples that share a certain term are likely to end up at the same peer due to our clustering algorithm discussed above, the converse does not hold: not all triples at a given peer will share terms, because each peer will be responsible for many different terms (after all, the number of terms will be far greater than the number of peers). Triples stored on disk are not indexed, because of the associated performance cost.

6. EXPERIMENTS

We have implemented our method on the MaRVIN platform[14, 15]. Although the techniques presented in this paper are peer-to-peer in nature (i.e. all nodes have symmetric functionality and there is no central control), we have conducted our experiments on a distributed compute cluster. This means that our system is peer-to-peer, but has not been evaluated in a wide area network. Thus, our evaluation mainly dealt with computational cost and issues such as bandwidth, peer failures and network latency have not been considered. Our implementation is open-source[3].

Since our approach deals with routing of triples, and not with the inference process that happens on the peers once the triples have been collocated, our approach is independent of any particular reasoner or logic (as long as it is monotonic). Thus, we will abstract from the actual reasoning in order to make our evaluation independent of the logic used. Based on the observation that all RDFS and most OWL-Horst rules either fire on a single triple or two triples sharing one or two terms, we will count these encounters. These encounters are essentially join operations over the triples.

For our evaluation, we will name *unary join* a join of two triples sharing a term and *binary join* a join of two triples sharing two terms. To illustrate our case, we note that the RDFS subclass transitivity rule (rule 11 in Table 3), requires a binary join: the two triples share two terms: the predicate `subclassOf` and the variable y. A unary join refers to rules such as rule 7, in which the two triples only share the variable x. Despite the name "unary", even the unary joins are indeed *joins*: they require that two triples have one term in common, and a join must be performed across these two triples.

To demonstrate the scalability of our system in absolute terms, we will calculate the closure of some of the Billion Triple datasets, listed in Table 2. Our routing strategy is independent of the reasoner used on each node. For the purposes of our experiments, we have implemented a simple and fast RDFS reasoner that produces the closure of the input, but similar results could have been achieved with an off-the-shelf reasoner such as Sesame, OWLIM or Jena. The system was stopped manually when no new triples were derived.

6.1 Experimental platform

Our experiments were run on the Distributed ASCI Supercomputer 3 (DAS-3). We have used up to 64 nodes with two dual-core processors and 4GB of RAM each interconnected through 10Gbps Myrinet.

Initially, data was placed in a shared file-system and randomly partitioned. Each node read a random partition.

We should note that although the network could sustain very high transfer rates (some 250MB/s), the bottleneck in our system, if no reasoning is employed on the peers, is the CPU throughput for marshalling and unmarshalling data. When peers apply local reasoning on their data, the reasoner is typically the scalability bottleneck.

6.2 Evaluation criteria

We will evaluate our approach according to the following criteria:

- *unary and binary joins*: as described above, we use the number of unary and binary joins performed per second as the criterion on how fast can our system match rules based on these joins. We believe that this is a good measure for comparison between different clustering methods because it applies to any logic with such rules.
- *triples produced*: to demonstrate the performance of our system in absolute terms, we use the number of triples produced. Note that this is not as general as the previous measure, since it is dependent on the actual reasoner deployed on the nodes, but it allows comparison with other systems.
- *data transferred*: the amount of data processed per peer is an indication of load-balancing. A skewed load distribution would mean that some peers are underutilised.
- *number of nodes*: we will show the ability of our system to perform better given additional compute nodes using the notions of speedup and speedup efficiency. The ideal 100% efficiency (n-fold increase of nodes leading to an n-fold increase in speed), is typically not achieved due to coordination overhead.
- *ruleset*: we can distinguish two rule categories in Table 3: the ones that have one antecedent (rules 1, 4a, 4b, 6, 7, 8, 10, 12 and 13) and rules that have two antecedents (rules 2, 3, 5, 9 and 11). The former are easy to implement in a distributed setting: create arbitrary partitions, and apply each rule to each triple in each partition. The latter rules are more challenging, since they require a join over two triples sharing one or two terms. When performing RDFS reasoning, we focus on the "hard" rules: those requiring a join over multiple triples. Although we have validated our results using the "easy" rules as well we will report only reasoning results with the "hard" ones.

6.3 Results

In this section, we will present the results of our empirical evaluation.

6.3.1 Data clustering

To evaluate the performance of our clustering algorithm, we have performed experiments on the SwetoDBLP dataset on 32 nodes. Instead of a reasoner, we have used a join counter for the unary and binary joins of the data loaded.

Figure 4 shows the number of joins performed per peer per second. Our first observation is that, regardless of the clustering method, the system can perform a very large number of joins per second, reaching an aggregate of 8.1 billion joins per second over 32 nodes (both unary and binary joins combined). Note that the same join may be counted twice, since there is no efficient way to store which joins have already occurred.

We compare four clustering techniques: *random* uses random exchanges and loads random data, *local* clusters data locally (i.e. selects data that have some term in common

[3] `http://svn.larkc.eu/public/marvin`

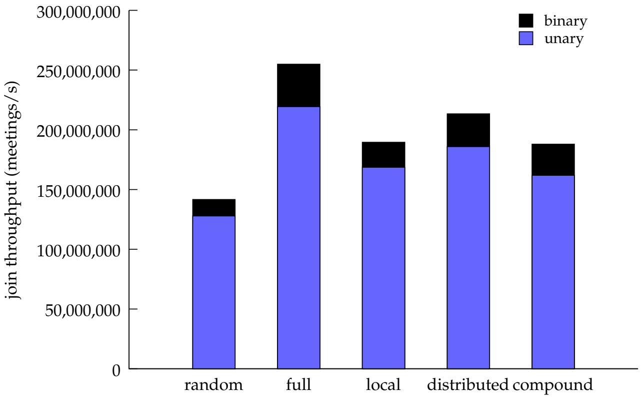


Figure 4: Join throughput (meetings/s) with various clustering methods

to load from storage), *distributed* uses the speeddate algorithm described in section 5.2.1 for exchanges between peers while loading random data from storage and *full* combines the previous two approaches.

Our results show that either local or distributed clustering benefits the join throughput of our system. The combination of local and distributed clustering yields an additional benefit, showing an improvement of 71% for unary joins and 159% for binary joins, compared to the random approach.

We should also note that these results do not show the full benefit of the clustering method. The distributed clustering algorithm creates a replica for each term in the triple, and routes them in the network. When a triple is loaded in a peer, only one of its terms was actually used for clustering: the one that was used as a key. On the other hand, when we count the joins, all terms are taken into account. This means that each triple replica is clustered according to only one of its terms. Any joins happening with any of the other two terms will be by "chance", similar to the random approach. In practice, the result for the distributed clustering refers to distributed clustering for one third of the terms, and random exchanges for two thirds of the terms.

6.3.2 Compound keys

We have also tested a variant of *full* in which not only the terms of each triple are used as keys for speeddating, but also combinations of terms. Namely, besides using *s, p* and *o* as keys, we also use *sp, po* and *so.* Such compound keys are common in the literature on indexing RDF data [7, 20].

This method produces twice the number of replicas for each triple, but should benefit the binary joins, since triples sharing two terms would have an additional chance to be collocated on the same peer, since they share an additional key.

Our results indicate that this is not the case: the overhead of maintaining additional triple replicas outweighs the benefits of the additional keys for binary joins. Compared to the standard approach, we get a decrease in join throughput of 26% for unary joins and 27% for binary joins.

6.3.3 Load Balancing

Figure 5 shows that the performance improvement of our strategy over randomised allocation of triples to peers does not go at the cost of load-balancing. Even in the face of our highly skewed datasets, the load across nodes in the speeddating approach is almost as balanced as the perfectly load-balanced uniform random partitioning.

nodes	runtime (s)	speedup	efficiency
1	2276	1.00	1.00
2	1213	1.88	0.94
4	598	3.81	0.95
8	330	6.90	0.86
16	197	11.55	0.72
32	179	12.72	0.40
64	88	25.86	0.40

(a)

nodes	runtime (s)	speedup	efficiency
1	15518	1.00	1.00
2	10432	1.49	0.74
4	5541	2.80	0.70
8	2822	5.50	0.69
16	1480	10.49	0.66
32	767	20.23	0.63
64	433	35.84	0.56

(b)

Figure 6: Closure time with increasing nr. of peers, on (a) SwetoDBLP and (b) combined datasets

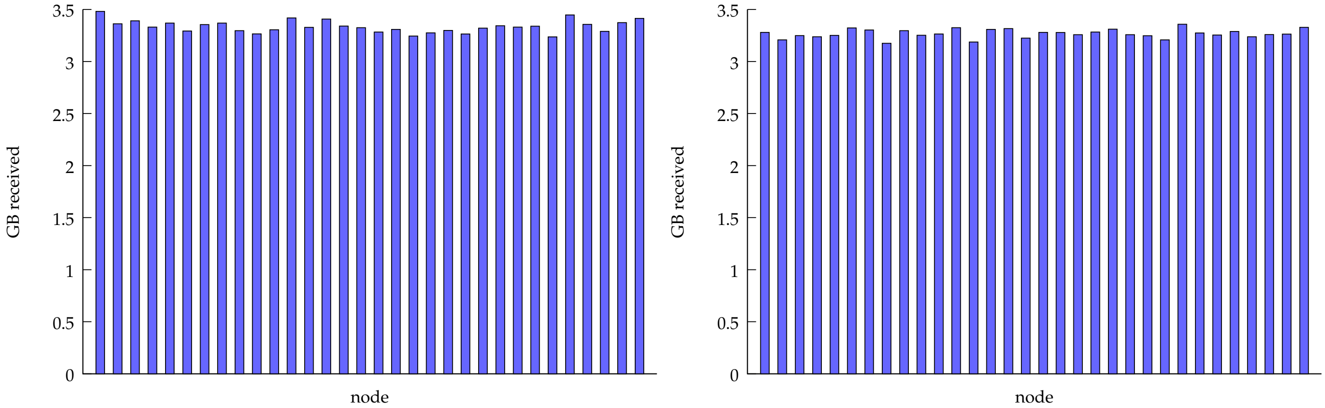
6.3.4 Scalability over peers and data

We also demonstrate the performance of our system on RDFS reasoning. Figure 6 shows the time needed to compute the complete RDFS closure of the SwetoDBLP and combined datasets (SwetoDBLP, DBpedia and Geonames), on increasing number of nodes. Also shown in the tables is the speedup (how much faster is the system with added nodes) and the resulting speedup efficiency. The latter is defined as the speedup divided by the number of nodes.

We can see that the time needed to compute the RDFS closure of the SwetoDBLP dataset on 64 nodes is 88 seconds. The input data contains some 15M triples, the closure (we consider only the "hard" rules) contains some 21.5M triples. Similarly, the time needed to compute the closure for the combined datasets (SwetoDBLP, DBpedia and Geonames) on 64 nodes is 7.2 minutes. The input data contains some 195M triples, the "hard" closure contains some 220M triples. This amounts to an aggregate throughput of 450.000 triples/sec.

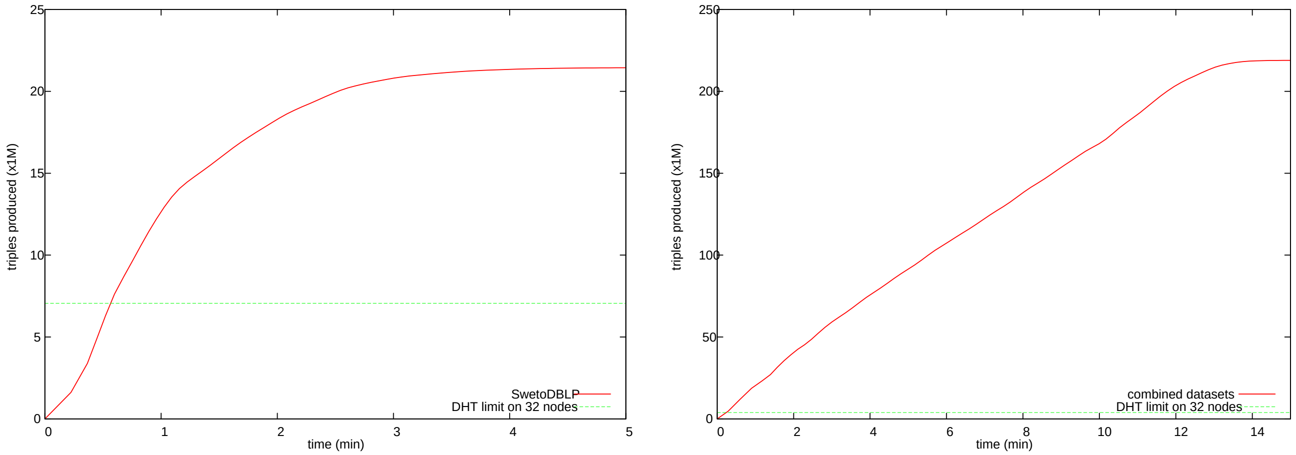
We can see that the system scales gracefully with the number of nodes. For the SwetoDBLP dataset, using additional nodes becomes increasingly inefficient, since the time to process all data is overshadowed by the time to set up the environment. We can see a similar situation for the *combined* dataset, albeit with a less steep curve, since the input is much larger.

Figure 7 shows the triple production curve for 32 nodes for the SwetoDBLP and the combined datasets. For comparison, the limited scalability of term-based partitioning approaches is also shown. For SwetoDBLP, given that the most-popular term in this dataset occurs in 17% of the triples, and assuming a maximum capacity per peer of 1.2M triples, a system doing term-based partitioning can only handle 7M triples, which is not even enough to store the input data, let alone the closure. Similar (but worse) num-



(a) (b)

Figure 5: Load-balance across peers in (a) speeddate and (b) random



(a) (b)

Figure 7: Time to compute closure for (a) SwetoDBLP and (b) combined datasets (on 32 nodes)

bers apply for the combined datasets, highlighting the low scalability of term-based partitioning approaches.

6.3.5 *Completeness*

All clustering techniques produce complete results for the RDFS rules with multiple antecedents. We have cross-checked our reasoning results with those presented in [18]. Furthermore, we have verified the correctness of our results for the SwetoDBLP dataset by loading it in on single machine, using the reasoner we have developed.

In all of this, remember that we are ignoring single-antecedent rules in our experiments (rules 1, 4a, 4b, 6, 7, 8, 10, 12 and 13). To output the complete closure, the best approach would be to first make all "difficult" derivations, and then, in a single pass, apply all the "simple" rules. Note that mainstream reasoners also typically ignore the trivial rules.

7. CONCLUSION

In this paper we have studied the problem of data skew in Semantic Web datasets, and how it affects distributed reasoners. We established the presence of substantial data skew in some realistic Semantic Web data. We showed theoretically that distributed reasoners would suffer from substantial load balancing problems if they were to use a deterministic term partitioning strategy (as most current systems do). We gave a theoretical bound on the performance ceiling of such systems, and illustrated this bound on realistic datasets.

To alleviate these load-balancing problems, we proposed a novel P2P technique that relies on constant data exchanges and clustering, while maintaining efficiency. Instead of deterministically routing triples with a shared term to a single peer (which causes load-balancing problems for skewed data), our approach attracts these terms to a region around a single peer, with the size of these regions "elastically" adjusting their size depending on the density of the triples in the region. The triples then move around in the self-adjusting region in order to meet other triples for inferencing (the "speeddating"). Our approach is fully self-organising, needs no a priori data analysis and no upfront parameter adjustments, leading to emerging elastic neighbourhoods and self-adjusting load-balancing.

We evaluated our technique on large scale datasets from the Web. Our evaluation is based on the number of data-joins that any reasoning system would have to perform, making our results independent of the choice of any particular reasoner. Our results show that our clustering technique approach clearly outperforms an approach based on random exchanges, and clearly outscales techniques based on term-partitioning. We computed the RDFS closure of 200M triples in 7.2 minutes on a 64-node cluster, with an aggregate throughput of 450.000 triples/sec.

Calculating the RDFS closure is an important building-block towards scalable reasoning and querying. In this respect, our method can be used to perform fast, parallel forward reasoning on the input of a conventional triple store such as OWLIM or 4store[4]. Alternatively, it can be used for methods that extract subsets of a materialised dataset, like the one presented in [21]. Finally, our method can be extended to querying, since matching the antecedents of a rules is equivalent to querying. Yet, there is a crucial difference: for reasoning, the main performance goal is to match as many rules as possible in a given amount of time. For querying, the response time also plays a very important role. Investigating the applicability of our method to querying is future work.

8. ACKNOWLEDGEMENTS

This work was supported by the EU IST project LarKC (FP7-215535).

9. REFERENCES

[1] D. Battré, A. Höing, F. Heine, and O. Kao. On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores. In *Proceedings of the VLDB Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2006.

[2] M. Cai and M. Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proceedings of the International World-Wide Web Conference*, pages 650–657, 2004.

[3] M. Cai, M. Frank, B. Yan, and R. Macgregor. A subscribable peer-to-peer rdf repository for distributed metadata management. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(2):109–130, December 2004.

[4] L. Ding and T. Finin. Characterizing the Semantic Web on the web. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 242–257, 2006.

[5] Q. Fang, Y. Zhao, G. Yang, and W. Zheng. Scalable distributed ontology reasoning using DHT-based partitioning. In *Proceedings of the Asian Semantic Web Conference (ASWC)*, 2008.

[6] S. Ganguly, A. Silberschatz, and S. Tsur. A framework for the parallel processing of datalog queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 143–152, 1990.

[7] A. Harth and S. Decker. Optimized index structures for querying RDF from the web. In *Proceedings of the Latin-American Web Congress (LA-Web)*, pages 71–80, 2005.

[8] P. Hayes, editor. *RDF Semantics*. W3C Recommendation, Feb. 2004.

[9] J. Heller. *Catch-22*. Simon and Schuster, 1961.

[10] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS reasoning and query answering on top of DHTs. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2008.

[11] D. R. Karger and M. Ruhl. Simple efficient load-balancing algorithms for peer-to-peer systems. *Theoretical Comput. Sci.*, 39(6):787–804, 2006.

[12] B. MacCartney, S. A. McIlraith, E. Amir, and T. Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[13] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[14] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. van Harmelen. Marvin: A platform for large-scale analysis of Semantic Web data. In *Proceedings of the International Web Science conference*, Mar. 2009.

[15] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. Ten Teije, and F. van Harmelen. Marvin: distributed reasoning over large-scale semantic web data. *Journal of Web Semantics*, 2009.

[16] R. Soma and V. Prasanna. Parallel inferencing for OWL knowledge bases. In *International Conference on Parallel Processing*, pages 75–82, 2008.

[17] 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(2–3):79–115, 2005.

[18] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using mapreduce. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2009.

[19] J. Weaver and J. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *8th International Semantic Web Conference (ISWC2009)*, October 2009.

[20] C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple indexing for Semantic Web data management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1008–1019, 2008.

[21] G. T. Williams, J. Weaver, M. Atre, and J. A. Hendler. Scalable reduction of large datasets to interesting subsets. In *8th International Semantic Web Conference (Billion Triples Challenge)*, 2009.

[4] `http://www.4store.org/`