An architecture for peer-to-peer reasoning

George Anadiotis, Spyros Kotoulas, and Ronny Siebes

Department of Artificial Intelligence, Vrije Universiteit Amsterdam, The Netherlands
{gan,kot,ronny}@few.vu.nl

Abstract. Similar to the current Web, the key to realizing the Semantic Web is scale. Arguably, to achieve this, we need a good balance between participation cost and perceived benefit. The major obstacles lie in coping with large numbers of ontologies, authors and physical hosts, inconsistent or inaccurate statements and the large volume of instance data. Our focus is on scalability through distribution. Most current approaches split ontologies into triples and distribute them among peers participating in a structured peer-to-peer overlay. Identifying a series of drawbacks with this, we propose an alternative model where each peer maintains control of its ontologies.

1 Introduction

The success of the Web is attributed to its scalability and its low entry cost. One would expect at least the same requirements for the Semantic Web. Unfortunately, state-of-the-art technology permits for neither, as current methods and systems make assumptions that limit its usability, especially with regard to scale.

In [9], a series of assumptions in logical reasoning are identified, which are also largely present in infrastructures developed for the Semantic Web, namely *Small set of axioms*, e.g. limited number of concepts/relationships, *Small number of facts*, e.g. limited number of instance data, *Trustworthiness, correctness, completeness and consistency of facts and axioms*, implying some sort of central control or management and *Static domains*, e.g. infrequent updates or fixed semantics.

With aspirations toward a truly usable and global Semantic Web, research has turned into a number of directions such as approximation, trust infrastructures, database technologies and distribution. The focus of this paper will be on distribution.

In this domain, peer-to-peer (p2p) systems are often seen as a vehicle for the democratization of distributed computing. Rather than relying on a possibly large set of professionally run commercial servers, they consist of community-volunteered hosts that collaborate on equal terms to achieve a common goal. Some of their perceived advantages are *low cost*, through the distribution of computation and self-organization, *no single point of failure*, due to their symmetric functionality and redundancy, *no single point of administration or control*,

making censorship or preferential disclosure of information impossible and, under some conditions, *scalability*, due to the fact that the network can grow on demand.

We can tap into the vast resources offered by p2p systems to develop scalable infrastructures for the Semantic Web. A plethora of approaches has already been suggested [5, 2, 20, 16, 14, 4, 19, 10, 17, 12], mainly focusing how to efficiently distribute large numbers of triples among peers in the network. We argue against this approach, claiming that although it solves scalability issues concerning the number of facts in the system, it fails to address the rest of the issues mentioned above and, in some cases, it actually makes additional non-realistic assumptions.

We propose an alternative approach, using ontologies instead of triples as the standard level of data granularity, thus moving complexity from the p2p overlay to peer interactions. This allows for efficient and secure maintenance of information provenance and control of the publishers over access and availability of information. We also hope that this model will eventually facilitate the development of methods to attest results calculated in a distributed manner and improve performance over current systems, since it can exploit concept locality in ontologies.

We are aspiring to combine the scalability of structured p2p overlays with the perceived advantages of our model. To this end, we are sketching an architecture that uses a global index maintained by a Distributed Hash Table(DHT) to find the correct peers that interact to resolve queries. Furthermore, some technologies that would be useful in this architecture are suggested.

The rest of the paper is structured as follows: In section 2.2 we are presenting the most important systems for RDF storage and reasoning. We argue that there is a need for a shift of paradigm in section 3. Section 4 is a description of our approach, for which we are giving some performance indicators in section 5. We are concluding and outlining future work in section 6.

2 Relevant literature

2.1 Distributed hash tables

DHTs are a well researched flavour of structured p2p systems [15] . Nodes function autonomously and collectively form a complete and efficient system without any central coordination. In DHT overlays, each object is associated with a key, chosen from a large space. This space is partitioned in zones, and each peer is responsible for the keys and corresponding objects in a zone. Peers need to maintain connections only to a limited number of other peers and the overlay has the ability to self-organize, with respect to peer connections and object distribution, to handle network churn. In principle, all DHT-based systems provide the following functionality: *store(key, object)* storing an object identified by its key, and *search(key)* which returns the object (when it exists) from the peer responsible for the key. Current systems need approximately $O(log(N))$ messages to search or store and each peer needs to keep from $O(1)$ to $O(log(N))$ pointers to other peers, where N is the number of peers in the network.

2.2 Scalable RDF storage

DHT-based Research into scalable RDF storage lies closest to the focus of this paper. Considerable research has been conducted in the area with most approaches sharing the following fundamental design choices:

- RDF queries are broken down into subqueries, namely triples with one or more variable values, for instance `<?,ns:lives_in,cities:amsterdam>`.
- Query results are sets of bindings for variables.
- No single node can be assumed to have the answers to all subqueries, so the problem then consists of decomposing the original query and routing the 'right' subqueries to the 'right' node, and then composing partial results to obtain the answer to the original query.

The first to propose the use of DHTs to implement a distributed RDF store was RDFPeers [5]. The basic functionality for storing RDF triples involves hashing the triple's subject, predicate and object and storing it in the three peers that are responsible for each of the resulting keys. Queries are answered by hashing the (at least one) constant part of the query triple pattern and routing the query to the node responsible for storing that constant. RDFPeers has the ability to resolve atomic, disjunctive and conjunctive multi-predicate RDF queries at a minimum cost of $log(N)$ (for atomic queries), however it has poor load balancing capabilities, completely lacks reasoning support and assumes a shared RDF schema.

GridVine [2] constitutes a logical layer of services offered on top of the P-Grid [1] DHT. It exposes higher level functionalities built on top of P-Grid: Insert(RDF schema), Insert(RDF triple), Insert(Schema translation) and Search-For(query). RDF triples are inserted into GridVine by using the same method introduced by RDFPeers and it can also answer the same set of queries, but has the additional advantage of supporting translations between RDF schemata.

PAGE [20] is a proposal for a distributed RDF repository implementation that combines the index structure of YARS [11] with a DHT. YARS uses 6 different indexes and stores RDF quads (triples augmented with context information). PAGE works by using the same indexes (hence replicating triples 6 times) and achieves more efficient query processing, but also lacks reasoning support and load balancing.

RDFCube [16] builds on RDFPeers by adding a second overlay that indexes triples based on an 'existence bit' and then performs a logical AND operation on this existence bit before actually retrieving the triples when evaluating a query. This results in more lookups and higher maintenance cost for the extra overlay, but reduces the required amount of data that has to be transferred on the network.

[14] proposes two different algorithms for evaluating conjunctive multi-predicate queries. The first one, QC, uses the indexing scheme of RDFPeers to index triples, with a small modification: if there are more than one constant parts for a subquery, then preference is given to indexing on the subject, then the object and then the predicate, as it is expected that this will also be their ranking according

to discrete values. Subqueries are also sorted according to expected selectivity before execution. Then, a 'query chain' is formed that consists of the nodes responsible for each subquery. The second algorithm, SBV, uses additional triple indexing and dynamic query chain formation exploiting local variable bindings for subqueries.

In BabelPeers [4], nodes are also organized in a DHT overlay, and inserted triples are hashed on their subject, predicate and object, and stored by the node responsible for the resulting key. BabelPeers nodes however host different RDF repositories, making a distinction between local and incoming knowledge and applying RDFS reasoning rules. Nodes are additionally organized in a tree overlay structure in order to deal with overly popular values.

Non-DHT based [19] is based on the notion of path queries to build an index only on paths and subpaths, but not on individual elements for a datasource. Every RDF model is seen as a graph, where nodes correspond to resources and arcs to properties linking these resources. The result of a query to such a model is a set of subgraphs corresponding to a path expression. Since the information that makes up a path might be distributed across different datasources, the index structure to use should also contain information about subpaths without losing the advantage of indexing complete paths, and the most suitable way to represent this index structure is a hierarchy, where the source index of the indexed path is the root element. In terms of space, the complexity of the index is $O(s * n2)$, where s is the number of sources and n is the length of the schema path. The trade-off is that query answering without index support at the instance level is much more computationally intensive, so different techniques (partly similar to the ones used in [14], in terms of query chain formation and subquery ordering) are applied on the basis of an initial naive query-processing algorithm in order to perform join ordering and overall optimization, under the assumption that nodes do not have local join capabilities.

Bibster [10] follows an unstructured semantic-based p2p architecture: each peer knows about its expertise and finds out about the expertise of neighboring peers through active advertising. Thus peers form *expertise clusters*. When a peer receives a query, it tries to answer it, or forwards it to other peers whom it judges likely to be able to answer the query, based on similarity functions between the subject of the query and the previously-advertised expertise topics, using the schema hierarchy and text-similarity methods.

Edutella [17] is a p2p architecture designed for distributed search of educational content based on meta-data. The meta-data is stored in RDF format in distributed repositories that form a super-peer-based p2p network, arranged in a hypercube topology. While it allows the use of multiple schemas, neither mappings nor RDF semantics are supported. Additionally it uses a broadcast-based approach which would not scale gracefully.

In SomeWhere[3] and DRAGO[18], peers are organized in a topology determined by sets of mappings between the local ontologies of peers. These mappings have to be manually created by users (i.e. peers enter the system by mapping their local ontologies to ontologies of participants in the network). In turn, they

are used to rewrite queries as they are forwarded among peers. Although their work on distributed reasoning is relevant and applicable to our approach, these systems assume manually created peer topologies and do not address the problem of finding the correct peers.

Federated RDF repositories [12] aim at offering unified access among different RDF repositories by integrating them according to the federated repositories approach. Semantic Federations are collections of heterogeneous distributed RDF repositories that can be accessed as a unique local Semantic Repository. This approach however is based on static definition of participating repositories and uses flooding to distribute queries among repositories; it therefore lacks the ability to scale and to dynamically update federation membership.

3 Motivation

For the remainder of this paper, we will focus on systems using a DHT infrastructure, since, so far, they are the only scalable solutions that do not rely on fixed schemata. Efficient as they may be in storing instance data and ontologies, these approaches do not address scalability in reasoning, are not dealing with provenance of information and do not support user/peer control over their own data. Hence, we argue that they are not appropriate infrastructures for the Semantic Web and are more similar to distributed databases, useful and important in their own regard. In the following paragraphs, we highlight some of their shortcomings, also in respect to the set of criteria mentioned in the introduction.

3.1 Reasoning

Partly due to their computational complexity, current reasoning techniques do not scale beyond a relatively small set of axioms. Focusing on approaches that distribute the reasoning process and, in particular, some of the systems presented in section 2.2, we can identify performance problems in both storing and retrieving triples:

Storing All approaches that support reasoning store the transitive closure of triples. Assume a music hierarchy where a class "Music" has hundreds of subclasses like "Rock", "Pop" etc. Storing the statement `<Joe,likes, 70's Rock>` implies storing a triple for each superclass of rock (e.g. `<Joe,likes, Classic Rock>`, `<Joe,likes, Rock>`, `<Joe,likes, Music>` etc), which may count in the dozens. Similarly, assuming that we use an approach like [2], to store these tuples in a DHT, we will need at least twice the number of messages as the number of tuples to be stored. To make matters worse, updating the ontology can be very expensive. Adding the statement `<Music,subclass_of,Art>` means that for all statements with `Music`, we need to insert an additional triple. The number of these triples increases by $O(N)$ with the number of axioms in the system, i.e. we have overall storage and message complexity of $O(M \times N)$ where M is the number of facts and N is the number of axioms in the system.

Querying Let us assume a query to find all subclasses of `Music` which are not a subclass of `Rock`. Resolving this implies retrieving all triples `<?,subclass_of,Rock>` and proceeding recursively down the hierarchy. Then all subclasses of `Music` have to be retrieved and the intersection of the two sets has to be calculated. To resolve this query, the entire hierarchy has to be retrieved. Although in terms of data traffic, this may sometimes be acceptable, the number of messages required is prohibitively high: resolving this query means sending at minimum a number of DHT messages roughly equal to the number of concepts in the hierarchy.

The aforementioned examples clearly indicate the shortcomings in the current approaches for triple generation in large-scale systems.

3.2 Control over ontologies

All DHT-based stores presented in section 2.2 share the following design assumption: *All ontologies and instance data are made public and are maintained in a distributed manner*. This is done by using the triple notation and distributing these triples among the hosts in the network, according to some indexing scheme. This means that hosts effectively have no control over the location and administration of their ontologies and instance data. We can identify the following weaknesses in this design:

Provenance of information The issue of information reliability that pertains the Web is also valid and even more exacerbated for the Semantic Web, since in this case information is meant to be processed and acted upon via automated reasoning techniques. Existing techniques[6] dealing with this issue are limiting in that they do not enforce identity verification, but assume a trusted environment. On the other hand, the only way to guarantee data integrity in such a distributed and dynamic environment would be the use of electronic signatures; i.e. each peer signs the triples it inserts in the system using its electronic key, which is certified by some certification authority. This however would impose a disproportionate overhead, since storing an electronic signature for each triple would require more space than the triple itself.

Publishers are not in control of their ontologies Ontologies and instance data are becoming important assets for businesses and organizations as they are expensive to develop, may contain business intelligence etc. Thus, it is very unlikely that publishers would want to relinquish their control to a set of community-volunteered computers. This would be as preposterous as suggesting to large companies to use one of the existing p2p file sharing systems to distribute their software.

Ontologies and instance data are made public Even in the case where relinquishing control would be acceptable, there would be many cases where access control would be required. Again important issues arise on how should this access control be implemented by a number of untrusted and unreliable peers.

Having identified a set of limitations that could inhibit the development of the Semantic Web on current infrastructures, we will propose an alternative paradigm that could provide solutions to some of these problems and lay the foundation for future research.

4 Our approach

The main innovation of our approach is shifting the level of granularity for peer data from triples to entire ontologies. We propose a model where peers retain control of their ontologies and reasoning is done in a p2p manner. Query resolution is done in an incremental and distributed manner.

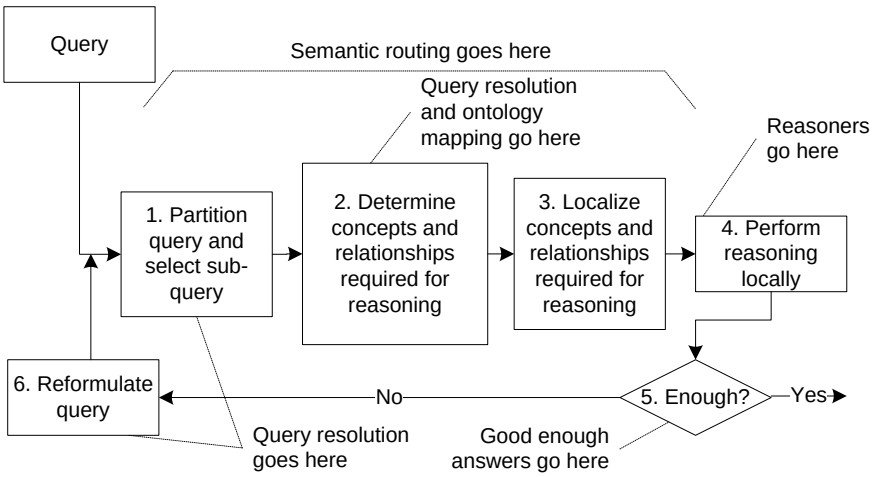

Fig. 1. Querying in our proposed model.

All peers have reasoning capabilities and are able to decide when they have had enough answers and query processing should be finished. Furthermore, queries can be decomposed into triple patterns(e.g. `<?, type_of, mtv:MUSIC>`). Figure 1 summarizes our proposed model. We will illustrate the explanation of each step using a simple example, the resolution of the query

```
SELECT X WHERE X type_of mtv:music
```

using RDFS reasoning rules (i.e. this query should return all X that are the predicate of a "type_of" relationship with object being `mtv:music` or any of its subclasses.

1. **Partition query and select sub-query** Initially, the part of the query to be resolved first needs to be determined. Our example query can be written in a triple pattern format as `<?, type_of, mtv:music>`. Obviously, there is no point in splitting this query further.

2. **Determine concepts and relationships required for reasoning** Out of the triple pattern <?, type_of, mtv:music>, we need to select a starting point for routing our query. There are the following two choices: type_of and mtv:music. Intuitively, the best choice would be mtv:music, since it is more selective and we can use semantic routing techniques to determine that in a distributed manner. Furthermore, note that instead of mtv:music, we may have a literal and not a concept. In this case, we will need to anchor it to a concept. This is where ontology anchoring and ontology mapping techniques come in handy.
3. **Localize concepts and relationships required for reasoning** For this step, either the triples that match the pattern have to be retrieved or the query should be forwarded to the peer(s) that store the ontology(-ies) with these triples. In our example, as in most cases, it is wiser to forward the query, since it is much smaller in size (just a single pattern with no results so far, in this case).
4. **Perform reasoning locally** Now reasoning can be performed locally and the first results can be returned.
5. **Determine if answers are adequate** The next choice is whether the retrieved results were enough for the user or application. If enough results were found, query resolution stops, otherwise, the query is reformulated to be further processed.
6. **Reformulate query** To retrieve additional results and according to RDFS semantics, instances that have a type which is a subclass of mtv:music should be returned[1]. Therefore, we can reformulate the query as follows:

   ```
   SELECT X WHERE X type_of Y and Y subclass_of mtv:music
   ```

   Alternatively, the local peer may start a new search for

   ```
   SELECT Y WHERE Y subclass_of mtv:music
   ```

   and continue processing once it gets back the results. Assuming that the peer followed the first option, query resolution would resume to step 1.

1'. Now, the query will be `SELECT X WHERE X type_of Y and Y subclass_of mtv:music`. The choice now lies between pattern <?, type_of, ?'> and <?, subclass_of, mtv:music>. The latter is preferred, since it has more bounded variables.
2'. mtv:music will be preferred over subclass_of, since it is more selective.
3'. The local peer is already knowledgeable about mtv:music (see 3.), so chances are, no forwarding is needed.
4'. The Y that are subclasses of music are found.
5'. Answers are still not adequate.
6'. Query is reformulated as `SELECT X WHERE X type_of Z and Z subclass_of Y`.

1". <?, subclass_of, Y> will be selected, since Y is bound.
2". Y will be selected, since it is more selective than subclass_of.
3". Query will be forwarded to peers with some of the possible Y, if they are not located on the local peer.
4". Additional results will be returned.
5". Assuming that there are now enough answers, querying is finished. Otherwise, we can continue with step 6.

.

[1] Note that this is not the only RDFS rule that applies in this case; for instance, we could look for subclasses of the relationship

4.1 Architecture

We propose an architecture abiding to the above model (fig. 2). Ontology descriptions or part of ontologies (i.e. concepts or relationships) are stored in a distributed public index and querying takes place in a p2p manner. The public index is maintained by a DHT consisting of a set of volunteer peers with adequate computational resources and fast, stable Internet connections. This index is used to *resolve URIs to locations*, i.e. locating the peers containing the ontologies and instance data for each relationship, concept or instance and to *Anchor terms to concepts in ontologies*, in case we want to anchor literals to ontology concepts or relationships (e.g. anchor "lives" to `namespace1:lives`).

Each peer stores a number of ontologies. Although ontologies may be moved across peers and replicated, this is not necessary. i.e. peers may choose to retain complete control over their ontologies or replicate them for performance. For instance, the RDFS ontology is used in the inference process and is public. So, it should be replicated to practically all peers for performance reasons. On the other hand, some peers may decide that they do not want their ontologies fully disclosed, and therefore store them only locally and answer queries on them. For such cases, the approach described in [13] comes to direct use.

In the simplest form of the system, all URI lookups are done through the DHT. Indexing is equally straightforward: Peers store on the DHT mappings from the URIs of the resources they want to answer to their address.

4.2 Optimizations

A series of simple optimizations are suggested to improve the efficiency of the system.

Triple caches To avoid redundant network messages, peers may cache received triples. This would drastically improve performance but it would also imply some sort of soft-state mechanism to manage updates or deletions.

Ontology caches/replicas Sometimes, a peer may need data from an ontology so often, that it would make sense to keep a copy of the entire ontology, and perhaps share it with other peers. Note however, that this would only be possible for public ontologies.

A semantic topology Apart from maintaining the global index, peers can be organized in a semantic topology, determined by the overlap of their resource descriptions. To this end, they would maintain a set of pointers to "interesting" peers, along with the resource descriptions they contain. This would substitute expensive DHT messages with direct network messages and would improve performance on the expense of some additional storage space per peer, which is generally considered of minor importance. Updating these pointers is straightforward. When a new ontology is inserted with a triple `<X,r,Y>`. For each triple, a pointer will be stored to the peer with the relevant concepts/relationships. For example, for `<wwf:seal, rdfs:subclass_of, mom:monk_seal>`, we will make a lookup for `wwf:seal` and `mom:monk_seal` and retrieve the peers which have triples

with these concepts. We will store a pointer to the publishing peer to each one of those peers. Thus, future queries that involve these concepts will be forwarded without having to consult the DHT.

5 Performance indicators

In this section we try to evaluate how our system would work by analyzing some properties of ontologies currently available on the web. For example, by analyzing the re-use of concepts (i.e. inter-linkage) between ontologies we can predict the consequences on how scalable our approach is since our approach performs better where there is not much re-use.

Swoogle [8] is a search and meta-data engine for the Semantic Web. Besides the core search functionality, it also provides detailed statistics about the more than 10.000 ontologies it stores, where Swoogle considers a *Semantic Web document* (SWD) to be a document represented as an RDF graph and a *term* refers to a rdfs:Resource node in a SWD.

5.1 Namespace usage

Namespaces used in an ontology are pointers to other ontologies and therefore an indication of re-use. Their statistics[2] show that there are 4576 namespaces used by 329987 SWDs. The purple line in figure 3a[3]shows the distribution of namespaces. As can be seen, the popularity follows some power-law disribution, meaning that only a few namespaces are very popular (like the `rdf` namespace) and most are rarely used. This confirms our hypothesis that ontologies are not strongly connected which means that in most cases the possible answers can be found locally on a single peer hosting the ontology/-ies of interest.

5.2 Local reasoning.

Swoogle provides statistics on the distribution of SWDs per website. There are 132206 websites indexed that are hosting 337182 SWDs, meaning an average of three SWDs per website. However figure 3a shows that the distribution follows Zipf's law except in the tail, meaning that most hosts[4] will only have one or two ontologies. If we combine this with the statistics on the number of terms per SWD (distribution shown in figure 3b) we see that in most cases local reasoning only needs to be done over a relatively small number ontologies. Namely, the figure shows that the number of class and property definitions in most cases is smaller than 10, one order of magnitude smaller than the number of populations. Most SWDs do not define classes or properties at all, but just populate instances,

[2] http://swoogle.umbc.edu/2005/modules.php?name=Swoogle_Statisticsfile=usage_namespace
[3] http://swoogle.umbc.edu/2005/modules/Swoogle_Statistics/images/figure5-2004-09.png
[4] note that we consider the number of hosts to be equal to the number of websites

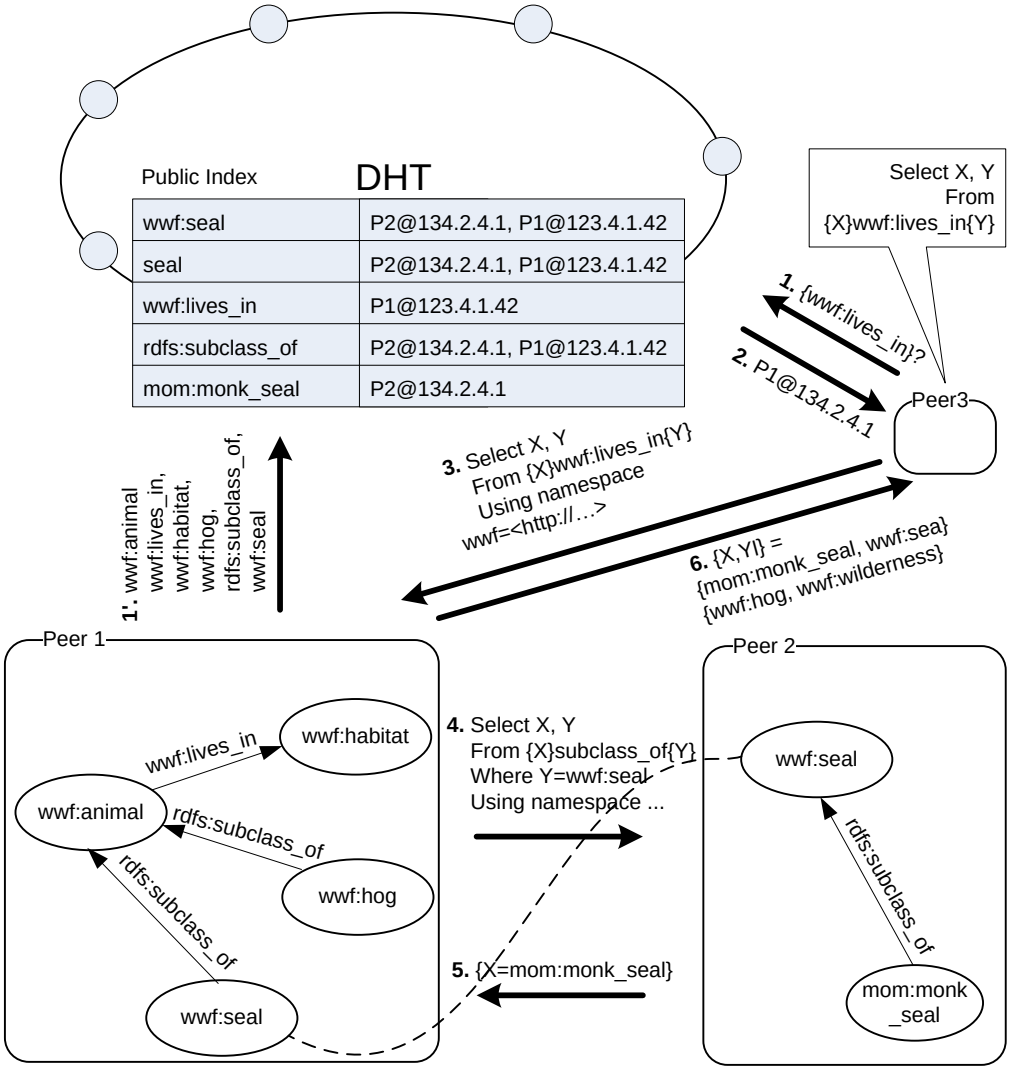

Fig. 2. The proposed architecture. **1'** Peers index (parts of) their ontologies by sending a flat list to the distributed index. **1-6** Querying consists of (1) Lookup on the index for a peer containing concepts or relationships that are part of the query, (2) Index returns the address(es) of the matching peer(s), (3) Query is forwarded to the selected peer(s), (4) Peer 1 creates a new sub-query according to RDFS reasoning rules and forwards it to Peer 2 using the semantic topology, (5) Peer 2 returns the results of the subquery to Peer 1, (6) Peer 1 aggregates the results and returns them to the querying peer.

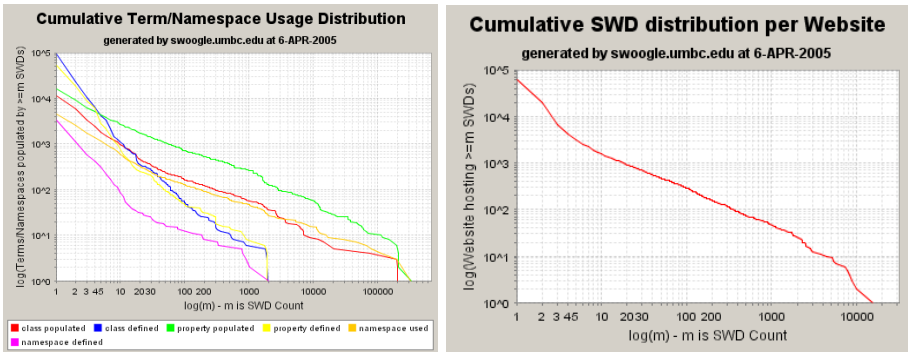

Fig. 3. a) Cumulative Term/Namespace Usages Distribution and **b)**Cumulative SWD

meaning that, in most cases, only local reasoning on instance checking needs to be done.

The number of SWDs per suffix[5] , not shown in this paper, shows that most ontologies are written only in RDF and only a few also in OWL, DAML or RDFS, meaning that not much extra reasoning is currently needed apart from simple RDF triple matching. For now, this is a counter argument to our approach in favor to distributed triple storage mechanisms, in terms of lack of need of complex local reasoning in the latter approach. However the arguments of desired local control and provenance still favor our approach. Besides this, [7] states that the increased use of two OWL equality assertions: owl:sameAs (279,648 assertions in 17,425 SWDs) and owl:equivalentClass (69,681 assertions in 4,341 SWDs) may be an indication of increased ontology alignment, and therefore increased use of richer languages.

6 Conclusions and future work

In this paper, we have presented a new method for distributed ontology storage and querying which has ontologies as the normal level of granularity for data distribution. Examining the ontologies currently on the Internet indicates that local reasoning is, most of the times, sufficient for query resolution. In this case, our approach outperforms ones that rely on triple distribution on top of DHT, since only local reasoning is required.

Future work lies in more diligent evaluation of our approach, doing simulation and emulation experiments. Furthermore, we have not examined the scenario where peers do not have the capacity to store their own ontologies/instance data. In this case, the latter would have to be split and distributed among several peers. It would be very interesting to investigate methods to accomplish that, for example using past queries to determine which concepts/instances/relationships are used together, or splitting the ontology graph so as to keep overlap between the resulting graphs to a minimum.

[5] http://swoogle.umbc.edu/2005/modules.php?name=Swoogle_Statisticsfile=swd_suffix

7 Acknowledgements

This work was supported by the EU OpenKnowledge Project (IST-2005-027253).

References

1. Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.
2. Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. Gridvine: Building internet-scale semantic overlay networks. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2004.
3. Philippe Adjiman, Philippe Chatalic, Francois Goasdou, Marie-Christine Rousset, and Laurent Simon. Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic Web . *Journal of Artificial Intelligence Research*, 25:269–314, 2006.
4. D. Battré, A. Höing, F. Heine, and O. Kao. On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT based RDF stores. In *Fourth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P) in scope of 32st International Conference on Very Large Data Bases*, 2006.
5. Min Cai and Martin Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. 13th International World Wide Web Conference*, New York City, NY, USA, May 2004.
6. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.
7. Li Ding and Tim Finin. Characterizing the Semantic Web on the Web. In *Proceedings of the 5th International Semantic Web Conference*, November 2006.
8. Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM Press.
9. Dieter Fensel and Frank Van Harmelen. Unifying reasoning and search to web scale. *IEEE Internet Computing*, 11(2):94–96, March/April 2007.
10. Peter Haase, Jeen Broekstra, Marc Ehrig, Maarten Menken, Peter Mika, Michal Plechawski, Pawel Pyszlak, Björn Schnizler, Ronny Siebes, Steffen Staab, and Christoph Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004*, NOV 2004.
11. A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In *LA-WEB '05: Proceedings of the Third Latin American Web Congress*, page 71, Washington, DC, USA, 2005. IEEE Computer Society.
12. Javier Jaen. MoMo: A Grid Infrastructure for Hybrid Museums, PhD Thesis. Polytechnic University of Valencia, 2006.
13. Spyros Kotoulas and Ronny Siebes. Scalable discovery of private resources. In *IEEE SECOVAL at SECURECOMM '07*, Nice, France, 2007.

14. E. Liarou, S. Idreos, and M. Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web - ISWC 2006*, volume 4273 of *LNCS*, pages 399–413. Springer, 2006.
15. Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2004.
16. A. Matono, S., Mirza, and I. Kojima. RDFCube: A P2P-based Three-dimensional Index for Structural Joins on Distributed Triple Stores. In *Databases, Information Systems, and Peer-to-Peer Computing*, Trondheim, Norway, 2006. Springer.
17. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: A P2P networking infrastructure based on rdf. In *Proceedings to the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.
18. Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2005.
19. H. Stuckenschmidt, R. Vdovjak, J. Broekstra, and G-J Houben. Towards distributed processing of RDF path queries. *Int. J. Web Engineering and Techonology*, 2(2/3):207–230, 2005.
20. E. Della Valle, A. Turati, and A. Ghioni. PAGE: A Distributed Infrastructure for Fostering RDF-Based Interoperability. In *DAIS*, pages 347–353, 2006.