OpenKnowledge

FP6-027253

Empirical tests of semantic routing

George Anadiotis, Spyros Kotoulas, and Ronny Siebes

{gan,kot,ronny}@few.vu.nl,
Vrije Universiteit Amsterdam,
De Boelelaan 1081HV,
Amsterdam,
The Netherlands

Report Version: final
Report Preparation Date: 14-12-2007
Classification: deliverable 7.3
Contract Start Date: 1.1.2006 Duration: 36 months
Project Co-ordinator: University of Edinburgh (David Robertson)
Partners: IIIA(CSIC) Barcelona
Free University of Amsterdam
University of Edinburgh
KMI, Open University
University of Southampton
University of Trento

Preface

This deliverable concerns empirical tests regarding semantic routing. We have had two papers published on the subject, which form deliverable 7.3. The paper entitled "Rarity-based routing in structured Peer-to-Peer overlays", published at the IEEE WETICE, presents a novel routing technique for structured Peer-to-Peer overlays, its application on the OpenKnowledge system and an empirical large-scale evaluation. Although not strictly required, we have also included the paper entitled "An architecture for Peer-to-Peer reasoning". This is a first step towards knowledge sharing through Peer-to-Peer interactions, combined with the scalability of structured overlay networks.

Rarity-based routing in structured peer-to-peer overlays

Spyros Kotoulas and Ronny Siebes
Vrije Universiteit Amsterdam
Department of Computer Science
De Boelelaan 1081, 1081HV, Amsterdam
The Netherlands
{kot,ronny}@few.vu.nl

Abstract

The OpenKnowledge project aims at knowledge sharing through open and flexible peer interactions. Within this project, we are developing a system that supports searching, developing and sharing of interactions/workflows consisting of roles implemented by software that can be shared and executed by peers. Its main requirements are openness, scalability, decentralization and robustness. Part of this system is a discovery service, which will be the focus of this paper. This service aspires to fulfill the above requirements featuring a Peer-to-Peer architecture and Distributed Hash Tables (DHTs) to achieve robustness through redundancy and scalability through decentralization. Resources are discovered using a set of attribute-value pairs. A straightforward DHT-based approach that creates a distributed inverted index suffers from a linear increase of messages and replicas with the number of attributes. We try to reduce this number by proposing an efficient multi-attribute routing algorithm. We emulate and test our implementation on the DAS-2 distributed supercomputer.

1 Introduction

Peer-to-Peer is a promising technology addressing some of the major challenges in modern distributed systems since it provides scalability through distribution of the deployment cost and all peers have the same functionality, providing robustness by redundancy.

The EU-funded OpenKnowledge project[1] has as one of its goals to build a P2P system, which we call the OK-system, for sharing knowledge, not only in the form of data but also in the way the data is processed [2]. Using this system, people can publish workflow descriptions (also called *Interaction Models or IMs*), and peers can subscribe themselves to play one or more of the roles in them. The role-code (i.e. software) that a peer needs to have to play such a role can also be shared and downloaded via the OK-system. Peers can also subscribe themselves to be coordinators of IMs, being responsible for their correct execution. All components of the OK-system will be implemented in a way that everything runs distributed without any central control. In this paper we focus on the component responsible for finding:

- *Interaction Models* Interaction models define the way services interact, expressed in a formal language like LCC[11] or BPEL[3]. They are described by a set of attribute-value pairs, they have small size and their search is facilitated by multi-attribute search. In table 1 an example descriptor of an IM is given. The unique identifier of the IM is `Auction5443FF`, and it is described by a set of attributes pertaining to its use and characteristics. These attributes (e.g. `Type` or `Role`) are fixed for OpenKnowledge, although we will see later that our discovery service does not depend on such a fixed schema.

- *Service descriptions* Services are described by a (potentially large) set of attribute-value pairs. They are small in size, and expected to be transferred over the network often. Consequently, we need efficient mechanisms for multi-attribute search, incorporating methods from information retrieval. In table 1 we show an example descriptor of a service description. For example, a user that has found the example interaction model can search for {'IM=Auction5443FF','Role=Buyer'} to find compatible service implementations for the role she wants to play, which is possible because a service description has at least one pointer to a role it implements for a given IM.

- *Service implementations* Service implementations are pieces of software statically bound to a service descrip-

[1] http://www.openk.org

[2] for additional information about OpenKnowledge the reader is referred to: http://www.cisa.informatics.ed.ac.uk/OK/deliverables.html

[3] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

tion. They are typically of average size and consist of a compressed file.

- *Coordinators* Finally, we need functionality to find coordinators. As can be seen in table 1, a coordinator should specify the languages it is able to interpret. Intelligent as opposed to random selection of coordinators for an interaction can have many advantages (e.g. if we select the same coordinator for multiple invocations of the same interaction, we can add interaction participants at runtime).

To the best of our knowledge, there exist no scalable, efficient and fully-distributed implementations for multi-attribute indexing and search. The JXTA project claims to have implemented such a system, but unfortunately, after conducting extensive tests, we discovered that its implementation could not scale to more than a handful of rendezvous peers[7]. The focus of this paper is on scalable, open, efficient and robust publishing and discovery of these resources through a community-supported Peer-to-Peer system.

DHT implementations [10, 12, 1, 9, 2] are currently seen as an important building block for Peer-to-Peer systems for storing content in a completely decentralized way [5]. 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 [9, 10, 12, 1]. Although they seem to deal very well with key lookups, automatic load balancing and robustness, their application domain is constricted by the absence of efficient methods to search for richly described content.

The discovery service of OpenKnowledge is implemented as a layer on top of DHTs to provide efficient discovery, multi-attribute search and distributed storage through use of the semantics of data being stored. In this paper, we are providing the fundaments for such a system by presenting a novel popularity-based algorithm for multi-attribute search over richly-described content.

In section 2 we outline the motivation our research. Our approach is described in section 3. We give an evaluation of the system in 4 and we conclude our work in section 6.

1. `Auction5443FF {Type=IM, Descr.Term=Auction, Descr.Term=Buy, Descr.Term=Sell, Descr.Term=Dutch Auction, Role=Buyer, Role=Seller}`
2. `Buyer4325 {Type=Service description, IM=Auction5443FF, Role=Buyer, Descr.Term=Buy}`
3. `CoordinatorFF32 {Parser=LCC, Parser=BPEL, Certification=Verisign}`

Table 1. Examples of resource descriptors.

2 Multi-attribute search by joining single-attribute searches

It is relatively easy to design a discovery system over a DHT that maintains a distributed inverted index over all attribute-value pairs. Namely, to **insert** a new resource into the system, one could hash each attribute-value pair and store it together with a pointer to the resource in the DHT overlay. Furthermore, to **retrieve** a resource, for each attribute-value pair of a query, a lookup on the hash of this pair is performed and a local join is performed over the results.

We have adumbrated the design of a simple DHT-based discovery system with perfect recall. Be that as it may, to the best of our knowledge there exists no efficient implementation of such a system. Where does it all go wrong?

- *Distributed join is costly* To perform a distributed join, the initiating peer has to gather all index entries for all the attribute-value pairs in the query. The cost of this can be prohibitively high, especially for queries with many pairs. The problem is further aggravated by their distribution of frequency. In document retrieval systems, terms usually follow a zipf distribution[4] (also see figure 1). Therefore, a query with at least one of these common attribute-value pairs will be too expensive to calculate, since it would imply retrieving all the indexes containing that attribute-value.

  Note that it would not be possible to limit the number of entries sent for common terms, since there is no way to know in advance which of these entries are popular. To illustrate our case, consider a query for ”Type=Service description”,”IM=Auction5443FF”; ”Type=Service description” would appear in hundreds of thousands of descriptions while ”IM=Auction5443FF” would appear in only a few.

  We can circumvent this problem by storing the entire description to the peers that correspond to the hash of each attribute-value pair. This comes at the cost of additional storage and bandwidth costs for inserting descriptions but it makes query answering a local operation and querying now costs only 1 message in the

DHT. In the next two paragraphs, we see why this is still inadequate.

- *Load balancing* As previously mentioned, the term frequency distribution often follows a Zipf pattern. This gives rise to severe load balancing problems, considering that a peer responsible for a very common attribute/value would have to store a large number of descriptions and process a large fraction of the total queries. To partly alleviate this problem, we can bound the number of descriptions that a peer can store and send queries to the peers for each of the attribute-value pair. Even so, this would solve the problem only for the querier, since the peers which would have to store popular content would be unresponsive for all keys that are mapped to them. For instance, imagine that the very popular attribute-value "Type=Service description" and rare attribute-value "Descr. Term=Dutch Auction" are both mapped to the same peer. The first will cause the peer to be overloaded and unresponsive. Despite the fact that this may not be a serious problem for the popular term, since descriptions with popular terms are common by definition, it will be problematic for the rare term.

- *Long descriptions* In the previous two paragraphs, we have assumed that descriptions are replicated to all the peers responsible for each of their attribute-value pairs. What if these descriptions are large? It is not unrealistic to assume that they contain hundreds of terms. In this approach, the number of messages and replicas increases linearly with the number of attribute-value pairs in the description, which makes the approach non-scalable.

3 Our approach

As a solution to the problem of managing large descriptions and multi- attribute search, this work is focused on popularity-based approaches. The key idea is that popular content is easily available on the network due to a high degree of replication. Therefore, we do not need to spend much effort on indexing it, in contrast to rare items.

In [15], the authors suggest that for queries for common items, flooding queries is sufficient, while for rare items, DHTs perform best. Research in the context of the PIER project [4] and in [8] also suggests a hybrid flooding/DHT mechanism (albeit with no efficient way to determine which items are rare). Indeed, for commons terms we are not interested in getting *all* results, if there are millions of them; a hundred would be enough. On the other hand, for rare terms, we are interested in *all* results. Nevertheless, most popularity-based approaches assume prior knowledge of which items are popular which is unrealistic. Our approach is to use statistical information, which is automatically calculated in a distributed way, to determine, on-the-fly, which terms are rare and which queries refer to them, and adapt the routing process accordingly.

[4] http://pier.cs.berkeley.edu

Algorithm 1 Rarity-based walk

Require: A description d with attributes/values $(t_1 \cdots t_n)$ and identifier id . Let parameter PR denote the originating peer set and D the description set of this peer
Ensure: d is stored.

1: $t := t_m \in (t_1 \cdots t_n) | \forall t, |Dt| > |Dt_m| \; and \; P_t \notin Pr$
2: **if** $(t = \emptyset)$ **then**
3: **return**
4: **else**
5: $Pr := Pr \cup this$
6: send(d,P_{t_m},Pr)
7: **end if**

Require: A query q for terms $(t_1 \cdots t_n)$, originating peer set Pr, the description set of this peer D.
Ensure: q is forwarded.

1: **if** (enough results found) **then**
2: **return**
3: **else**
4: $t := t_m \in (t_1 \cdots t_n) | \forall t, |Dt| > |Dt_m| \; and \; P_t \notin Pr$
5: **if** $(t = \emptyset)$ **then**
6: **return**
7: **else**
8: $Pr := Pr \cup this$
9: send(q,P_{t_m},Pr)
10: **end if**
11: **end if**

An interesting and relevant approach is Mercury[3], supporting efficient multi-attribute and range search using a small to medium-size schema. It relies on hubs, consisting of a collection of peers responsible for storing indexes with a specific attribute, functioning as a sort of sub-overlays. Descriptions are routed to *all* hubs, which means that the number of messages and replicas is proportional to the number of attributes in a description. Furthermore, each peer in the network needs to know at least one peer in each hub, meaning that the number of peer references that need to be kept by each peer is at least proportional to the number of attributes in the system. Needless to say, this approach cannot scale beyond a schema with a dozen of attributes.

In this paper, we focus on index placement and do not investigate caching techniques, or the use of shortcuts to peers that gave good results in the past[14]. The methods proposed in this paper are orthogonal to them and it may be expected that both can benefit from each-other.

The novelty of our approach lies in exploiting the structure in DHTs to extract statistical information useful for routing, with the goal of alleviating the problems with joining single-attribute searches and current popularity-based approaches. We will describe an algorithm that uses statistical information from the local storages of peers to place descriptors more efficiently. The intuition behind our popularity-based approach is that rare attribute-value

pairs are preferred for replication, since: (1) For common attributes-values, it is likely that we will find answers anyway, since more matching descriptors will exist in the system. (2) Rare attributes-values yield a higher information value. (3) Peers responsible for common descriptions are likely to be overwhelmed by descriptions.

To illustrate our case, imagine the following resource description: In the simple approach described in 2, the description 'Buyer4325' would be replicated to the peers responsible for 'Type=Service description', 'IM=Auction5443FF', 'Role=Buyer' and 'Descr.Term=Buy'. It is reasonable to expect that a query for {'Type=Service description'} would be easily satisfied by many peers. Therefore, it would be a waste of resources and a network hot-spot to replicate the description to the peer responsible for this attribute/value pair (i.e. to the peer where the string 'Type=Service description' maps to). On the other hand, 'IM=Auction5443FF' is rare, and the description should be replicated to the peer responsible for it, since it would be difficult to find another peer that has this rare attribute-value. But how do we determine whether an attribute-value is rare? Unlike previous approaches, we do not assume external or centralized sources of statistical information, but rely on the properties of the distribution of terms into descriptions and the DHT. So, going back to our example, assume that initially(by random choice) the peer responsible for 'IM=Auction5443FF' is selected for replication. It is very likely that this peer will already have descriptions with 'Type=Service description' since (a) 'Type=Service description' is a very common attribute-value and (b) 'IM=Auction5443FF' and 'Type=Service description' are semantically correlated. Therefore, in our approach, it should decide not to replicate it to the peer responsible for 'Type=Service description' since subsequent queries including 'IM=Auction5443FF' would be answered by the peer responsible for 'IM=Auction5443FF', while queries including for 'Type=Service description' can be easily answered by many other peers (or at least by the peer responsible for 'Type=Service description').

Our algorithm is described in natural language in the following paragraphs and formally in algorithm 1.

Inserting descriptions Insertion messages consist of the description and the attribute/value pairs of the description that have already been used (this set is initially empty). All attribute/value pairs with frequency over a given threshold parameter Dt_m, over the descriptions of the local peer are marked as used. The attribute/value pair that has the lowest frequency and has not been used is selected (i.e the attribute/value pair with the smallest number of occurrences in the descriptions stored locally in the peer). If such an attribute-value pair exists, it is marked as used and the message is forwarded to the peer responsible for that attribute/value pair in the DHT.

Querying For each attribute-value in the description, the hash-value is calculated and the query is routed to the peer in the DHT to which that value corresponds. However, if enough answers are found on the peers en-route, the message is not routed further toward the destination peer (according to the DHT routing algorithm) and the query process for that attribute-value pair stops. This is meant to protect peers to which popular attribute-value pairs map to.

Compared to an algorithm that replicates according to attribute-value pairs chosen at random, our approach has negligible additional computational costs, as both determining which are the rarest terms in a description and maintaining a list of term frequencies is very fast.

Furthermore, it is interesting to note its anytime behavior. In the beginning, when peers have no overview of which terms are rare, they will replicate descriptions to *all* peers, since the threshold for replication will not be reached. As the number of descriptions in the system grows, so will the local knowledge in each peer about which attributes-values are popular, since peers can approximate the popularity of an attribute-value by counting the occurrences in their own data. No additional mechanisms to decide whether there is enough information are required.

4 Evaluation

In this section, we evaluate our approach against an approach that replicates according to attribute-value pairs chosen at random.

4.1 Dataset

Since, to the best of our knowledge, there exists no large dataset for resource descriptions for service workflows/interaction models, we decided to use a dataset created for general-purpose information retrieval, developed for [13]. We believe that this is a realistic assumption because, in a discovery setting attribute-value pairs in a description are semantically correlated. Future research should give additional insight on attribute-value distribution. For now, we assume that it is similar which the distribution of terms in documents. Our dataset was created by crawling a large number of real user queries from SearchSpy[5] and applying a natural language processing method on the results retrieved for these queries using Google[6], to get relevant descriptions. The input to our system was derived from the following:

- **Corpus** We have used a corpus of 260.000 documents, resulting in the same number of descriptions. Each description consists of a set of terms.
- **Descriptions** From these descriptions, we have selected a random set of 100.000. On average, each document contained approx. 104 terms (the distribution is shown in fig.1). The distribution of terms, as expected,

[5] http://www.infospace.com/info.xcite/searchspy
[6] http://www.google.com

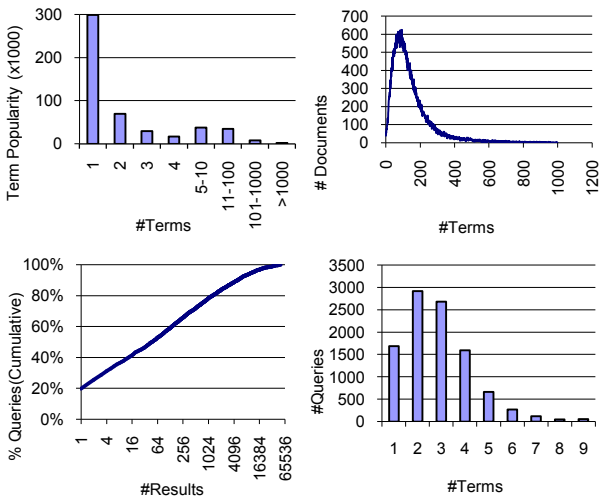

Figure 1. **Top left**: Term popularity. For example, around 290.000 terms appear only once in the dataset, and around 60.000 appear 2 times. **Top right**: Distribution of the number of terms per document. **Bottom left**: Number of results per query (cumulative). We can see that for approx. 50% of the queries, we have less than 50 results and for 30% of the queries, we have less than 4 results. **Bottom right**: Number of terms per query.

follows a zipf-like distribution(fig.1). It is interesting that more than half of all terms appear only 1 time, while 1 term appears in more than half of the descriptions (58204 times).

- **Queries** To generate queries, we have used the following method: (a) Randomly pick the number of terms $|t|$ for each query (according to the distribution in fig.1(bottom right)). (b) Pick at random a description out of the corpus. (c) Pick $|t|$ terms (randomly using a uniform distribution) from the chosen description and use them as the query terms. For most queries, there are fewer than 50 answers

4.2 Criteria

In this paper, we will evaluate our system in terms of *description recall*. To gain additional insight, we will always take into consideration the number of answers in the system, but limit the number of answers we are interested in to (e.g. maximum 50 answers, which is a parameter that can be changed for each query). The reason behind this, is that, in a discovery setting, there is no point in trying to retrieve *all* answers. Instead, we are interested in getting *enough* answers to satisfy the user. Nevertheless, this is not a limitation of our algorithm itself, it is a choice to make our evaluation more realistic. Thus, for our experiments, recall is defined as follows:

$$D_{Recall} = \frac{|D_{returned} \bigcap D_{relevant}|}{\min(|D_{relevant}|, 50)}$$

4.3 Design, implementation and experimentation

Design Our system is based on a three-layer architecture, the bottom layer consisting of a DHT implementation. The second layer consists of an object store and a distributed index supporting multi-attribute search and relies on the algorithms described in 3. Finally, the third layer is application specific, in our case the OpenKnowledge service and peer discovery.

Implementation We have implemented our system using Java 1.5. For the bottom layer, we have used the FreePastry DHT implementation, version 2.0b[7]. The second layer is an implementation of the algorithm in section 3 and the subset replication approach. The application on top is the discovery service of the OpenKnowledge system[6], as described in the introduction of this paper.

Testing and experimentation We have used the DAS-2 distributed supercomputer[8] to test and evaluate our system. One node on the DAS-2 acted as a bootstrap, being used as an access point to the system for the rest. We have used Globus[9] to start 500 instances of our system, which contacted the bootstrap node, and self-organized according to the Pastry protocol[12]. This process took less than 5 minutes. Next, nodes published in parallel 200 descriptions each (100.000 in total). Finally, each node made 100 queries and collected the results.

We have compared our approach to one that replicates descriptions according to a subset of its attributes/values, chosen randomly. For our evaluation, and to have a reference point, we have chosen replicate according to a maximum of 10 attributes/values, thus maintaining 10 index replicas. An approach that would replicate according to all terms would offer perfect recall. Nevertheless, it would require an average of 104 DHT messages and index replicas for each description, which is not scalable. Subset replication does not offer perfect recall, but it does reduce the number of replicas for each description by a factor of 10. Fortunately, as we will see, it does not lead to a proportionate reduction in recall.

For our rarity-based walk algorithm we have adjusted Dt_m to get the same number of messages and replicas as the subset replication approach. Moreover, for querying, we have used the same policy for both approaches. Therefore, the network, computational and storage costs for the rarity-based walk and the subset replication are very similar.

5 Results and discussion

Figure 2 shows the results for the two approaches. Each approach required the same number of query messages,

[7] http://www.freepastry.org
[8] http://www.cs.vu.nl/das2/
[9] http://www.globus.org/

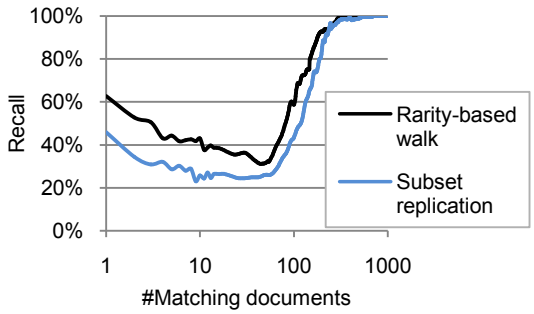

Figure 2. Description recall as a function of the number of matching descriptions per query.

an average of 2,5 DHT messages. The first impression is that, for both approaches and for queries with more than 300 matching descriptions, we get almost perfect recall using ten times less replicas and messages compared to an approach that would replicate according to all attributes/values. Our rarity-based walk yields a recall in excess of 60% for queries with only a single matching description, which are the most difficult to answer, an increase of approx. 35%, compared to the subset replication approach. In total, even for a relatively small overlay of 500 peers, we gain a substantial increase in recall using the same number of messages. It is expected that as the network size grows, the recall of the subset replication will deteriorate faster than that of our rarity-based walk approach, since the load balancing problems of the former will be augmented. At the moment of writing this paper we are expecting that in the following weeks, the DAS-3 supercomputer[10] will be made available, providing substantially greater computational power for experimentation which will enable us to verify the aforementioned claim.

6 Conclusions

In this paper we outline the functionality and design of a scalable peer-to-peer discovery service. We provide an implementation that incorporates a novel algorithm that reduces the scalability problems of multi-attribute indexing and search in DHT networks by automatically calculating attribute/value popularity and using it to reduce the degree of replication. Our implementation was tested by emulating 500 instances of our system on the DAS-2 supercomputer. The results indicate that an algorithm that takes attribute/value popularity into consideration for routing outperforms an algorithm that does not. Future work lies in testing how this gain in performance changes as the network size increases and measuring the robustness of our system toward a high peer churn rate.

[10] http://www.cs.vu.nl/das3/

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[2] M. S. Artigas, P. G. López, J. P. Ahulló, and A. F. Gómez-Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *Peer-to-Peer Computing*, pages 49–56. IEEE Computer Society, 2005.

[3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In R. Yavatkar, E. W. Zegura, and J. Rexford, editors, *SIGCOMM*, pages 353–366. ACM, 2004.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM'99 conference*, pages 126–134, New York, USA, March 1999.

[5] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *IPTPS '03*, February 2003.

[6] A. P. de Pinninck, D. Dupplaw, S. Kotoulas, and R. Siebes. The openknowledge kernel. Technical report, Openknowledge consortium, 2006.

[7] S. Kotoulas. Extracting and incorporating keyword semantics in DHTs, MSc thesis. Vrije Universiteit Amsterdam, 2006.

[8] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid p2p search infrastructure. In *In Proc. IPTPS*, 2004.

[9] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world, 2003.

[10] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[11] D. Robertson. Multi-agent coordination as distributed logic programming. In B. Demoen and V. Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 416–430. Springer, 2004.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[13] R. Siebes. pnear - combining content clustering and distributed hash tables. In *Proceedings of the IEEE'05 Workshop on Peer-to-Peer Knowledge Management.*, San Diego, CA, USA, July 2005.

[14] G. Skobeltsyn and K. Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *P2PIR '06: Proceedings of the international workshop on Information retrieval in peer-to-peer networks*, pages 33–40, New York, NY, USA, 2006. ACM Press.

[15] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of the IEEE INFOCOM conference*, San Fransisco, CA, USA, march 2003.

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 [8], 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 in 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 [4, 2, 18, 15, 13, 3, 17, 9, 16, 11], 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 [14] . 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 [4]. 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 SearchFor(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 [18] is a proposal for a distributed RDF repository implementation that combines the index structure of YARS [10] 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 [15] 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.

[13] 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 [3], 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 [17] 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 [13], 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 [9] 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 for *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 [16] 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 mapping nor RDF semantics are supported. Additionally it uses a broadcast-based approach which would not scale gracefully.

Federated RDF repositories [11] 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[5] 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.

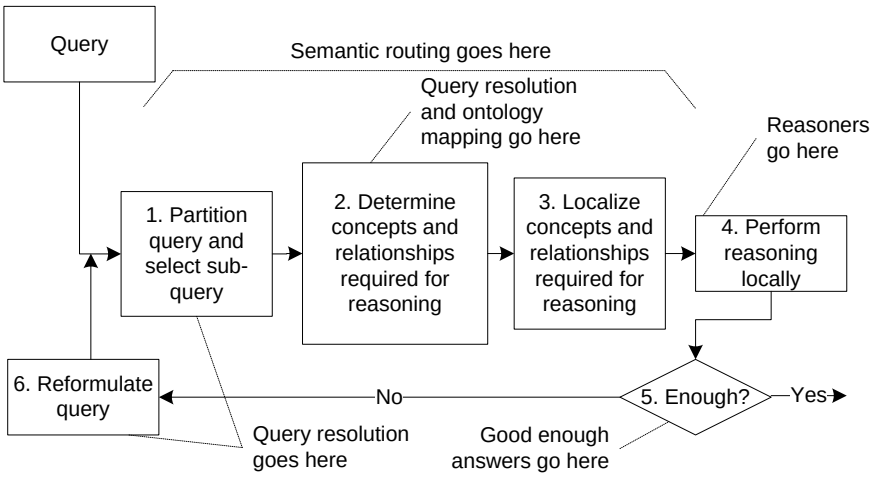

Fig. 1. Querying in our proposed model.

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.
2". `Y` will be selected.
3". Query will be forwarded to peers with some of the possible `Y`.
4". Additional results will be returned
5". Assuming that there are now enough answers, querying is finished. Otherwise, we can continue with step 6.

.

4.1 Architecture

We propose an architecture abiding to the above model. 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,

[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

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 [12] 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.

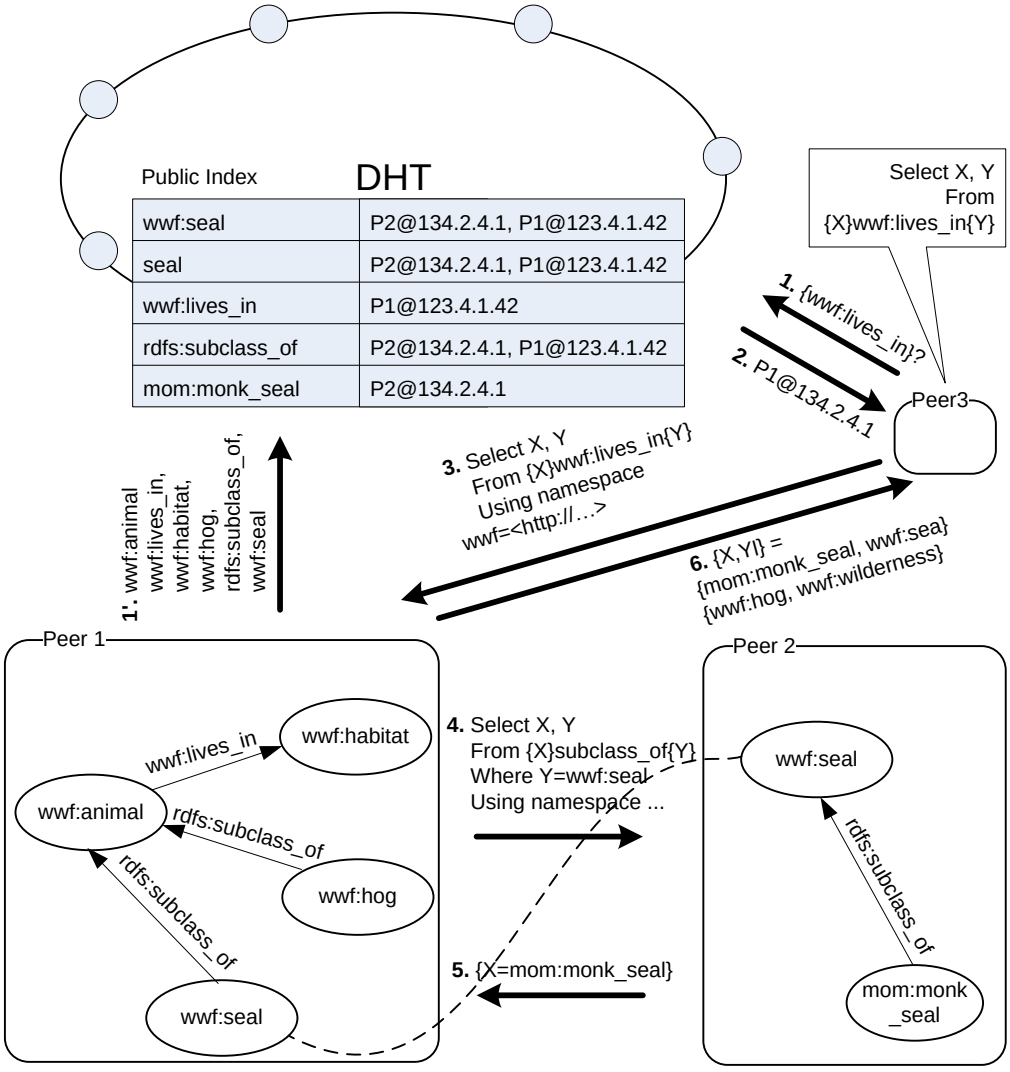

Fig. 2. The proposed architecture. **1'** Peers index (parts of) their ontologies by sending a flat list to the distributed index. **1-6** Querying consists of (1) Lookup on the index for a peer containing concepts or relationships that are part of the query, (2) Index returns the address(es) of the 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.

Swoogle [7] 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.

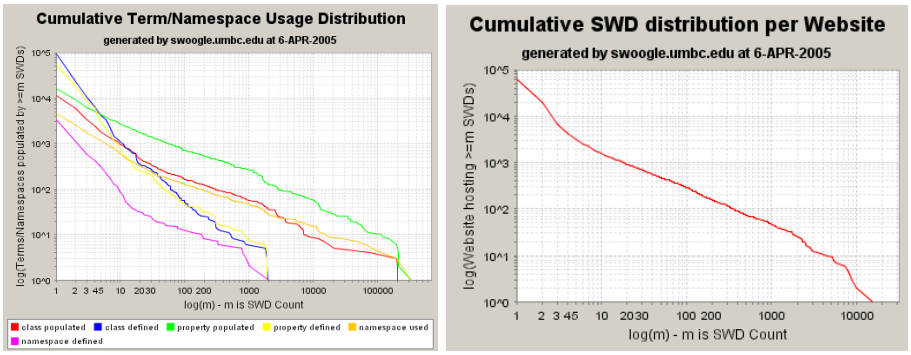

Fig. 3. a) Cumulative Term/Namespace Usages Distribution and **b)**Cumulative SWD

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 ontologies. Namely, the figure shows that the number of class and property definitions in most cases is smaller than 10 is one order of magnitude smaller than the number of

[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

populations. Most SWDs do not define classes or properties at all, but just populate instances, 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 needed currently than simple rdf triple matching. For now, this is a counter argument to our approach in favor to distributed triple storage mechanisms in terms of the need of lack of complex local reasoning in the latter approach. However the arguments of desired local control and provenance still favor our approach. Besides this, [6] 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 clearly outperforms ones that rely on triple distribution on top of DHT.

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.

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. 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.
4. 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.

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

5. 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.
6. Li Ding and Tim Finin. Characterizing the Semantic Web on the Web. In *Proceedings of the 5th International Semantic Web Conference*, November 2006.
7. 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.
8. Dieter Fensel and Frank Van Harmelen. Unifying reasoning and search to web scale. *IEEE Internet Computing*, 11(2):94–96, March/April 2007.
9. 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.
10. 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.
11. Javier Jaen. MoMo: A Grid Infrastructure for Hybrid Museums, PhD Thesis. Polytechnic University of Valencia, 2006.
12. Spyros Kotoulas and Ronny Siebes. Scalable discovery of private resources. In *IEEE SECOVAL at SECURECOMM '07*, Nice, France, 2007.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. E. Della Valle, A. Turati, and A. Ghioni. PAGE: A Distributed Infrastructure for Fostering RDF-Based Interoperability. In *DAIS*, pages 347–353, 2006.