Towards Taxonomy-based Routing in P2P Networks

Alexander Löser

Berlin University of Technology, 10587 Berlin, Germany
aloeser@cs.tu-berlin.de

Abstract. In Peer-to-Peer networks limited system resources prohibit broadcasting a query to all possible peers. Distributed Hash Table(DHT)-based approaches scale well, but do not consider ontological structures, such as existing classifications of peers in taxonomies. We investigate adaptive taxonomy-based routing overlays, restricting a search only to those peers offering possible results and present approaches for their automatic computation. We propose taxonomy-based models and discuss how models are stored, queried and managed in a DHT-based catalog allowing an efficient taxonomy-based routing of queries in a peer-to-peer network.

1 Introduction

The development of smart, scalable approaches for the discovery and location of data sources in distributed heterogeneous information systems is an important problem in many scientific and commercial domains. For a given query, limited system resources prohibit forwarding a query to all possible data sources. Due to the user's lack of global knowledge about suitable data sources and the unpredictable dynamics of available data sources, the user is not interested in looking for appropriate sources on its own. In fact,

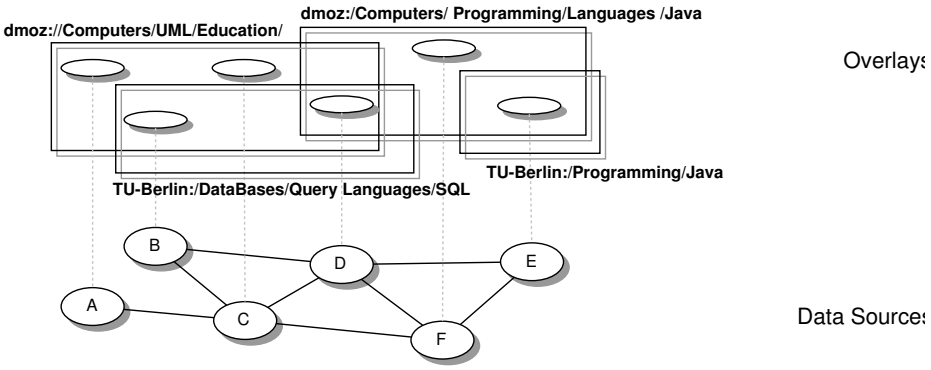

Fig. 1. Taxonomy-Based Overlays

after submitting a query from a user to the system, the query should be routed only to those data sources able to answer the query and available at query time. For instance, in

the e-learning domain during the last years a large number of digital e-learning repositories has been build. In the Open Directory[1] 400 Online Courses are registered, the Global Network Academy[2] has listed about 30.000 free open source online courses. Furthermore in local university curriculums courses and materials are classified in a taxonomical structure. Consider a learner is interested in materials about *Java Programming*. The naive way would be to send a query to all data sources. Obviously this would not scale for a large number of data sources and learners.

In our approach data sources are classified by a taxonomy. In Figure 1 data sources D, E and F are classified as *dmoz:/Computers/Programming/Languages/Java*. Furthermore data sources can belong to more than one classification, e.g. data source D belongs to *dmoz:/Computers/UML/Education/*, *TU-Berlin:/DataBases/Query Languages/SQL* and *dmoz:/Computers/Programming/Languages/Java*. A query is processed by sending it only to suitable classified data sources, e.g. a query for *Java Programming* is only sent to data sources classified as *google:/../Languages/Java*. In Figure 1 the query is only sent to D, E and F.

Data source classifications are stored in a fully distributed catalog. A query will be first matched against a catalog of descriptions of all available e-learning data sources and then routed to a small set of selected data sources providing the materials. Furthermore as new data sources become available, they automatically become part of the catalog. We assume that the catalog must handle several thousand published models of data sources and millions of queries. Data sources as well as queries are posted on an arbitrary node of the catalog. The result of a query are IDs of relevant data sources. We assume a completely distributed catalog of data sources stored on a set of interconnected nodes in a symmetric network, where each node stores a part of the catalog. Motivated by the recent advantages of scalable location and routing protocols based on distributed hash tables (DHT), such as CAN, CHORD, PASTRY, TAPESTRY and P-GRID, we investigate efficient storage and query strategies for models of data sources in a fully distributed catalog based on a DHT-based infrastructure. We use CHORD, since it is simple, robust, provides a simulation frame work and has been proved and evaluated in [19].

In this paper we study how to implement a distributed catalog for taxonomy based routing on top of the CHORD protocol. The main contributions of this paper are:

- We model data sources and queries using XML syntax.
- We store models and taxonomies in a fully distributed system by using the new concept of key-to-query relations in distributed hash table.
- Traditional DHT networks only allow exact matchings. By storing hierarchical relations in a DHT, we provide a data structure for the execution of of hierarchical queries as well as conjunctive queries for two or more taxonomy paths while allowing our catalog to scale for a very large number of data source models and queries.
- We propose storage load balancing concepts for storing a large number of peer models
- We provide first evaluation results of our approach

[1] $http://directory.google.com$

[2] http://www.gnacademy.org

2 System Model

In this section we introduce models of data sources and policies for querying models. Later we propose indexing and lookup strategies for models and taxonomies in a distributed hash table.

2.1 Super-Peer based Architecture

Recently a new wave of peer-to-peer systems is advancing an architecture of centralized topology embedded in decentralized systems; such a topology forms a super-peer network. Super-peer networks occupy the middle-ground between centralized and entirely symmetric peer-to-peer networks. They introduce hierarchy into the network in the form of super-peer nodes, peers which have extra capabilities and duties in the network (see [7]). A super-peer is a node that acts as a centralized server to a subset of clients, e.g. information provider and information consumer. Information provider submit its model to the super-peer node, while information consumer submit queries to their super-peer node and receive results from it, as in a hybrid system. However, super-peers are also connected to each other as peers in a pure system are (see also figure 2), routing messages over this overlay network, and submitting and answering queries on behalf of their clients and themselves. Examples of super-peer networks are JXTA[8], Edutella[13] or Morpheus. Because a super-peer network combines elements of both

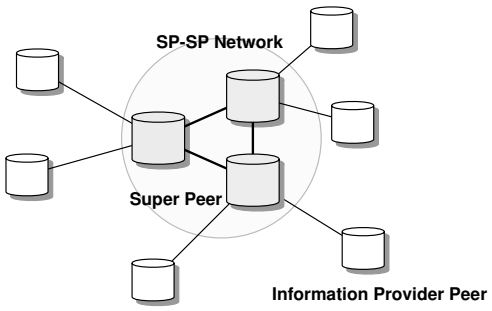

Fig. 2. Super-Peer Network

pure and hybrid systems, it has the potential to combine the efficiency of a centralized search with the autonomy, load balancing[22], robustness to attacks and at least semantic interoperability [1] provided by distributed search.

2.2 Models and Queries

In our approach each peer registers its model at a super-peer. We assume that each peer is classified in a classification hierarchy. E-learning documents are very often classified in a curriculum, which forms a taxonomy. If such as classification is not the case e-learning documents are mostly accompanied by keywords, characterizing their semantic

content. We can utilize tools, such as *Google/DMOZ*, *Autonomy.com*, *Semio.com* or [10] that can classify the documents, given their keywords, into hierarchical structured categories.

Each peer stores documents of different categories. Typically the number of documents is considerable larger than the number of categories, each peer has to manage. The model of each peer consists of a summary of all categories belonging to documents stored at this peer. Thus each peer is classified by paths in one or more taxonomies and publishes a model based on semi-structured XML data with the taxonomies and paths. To illustrate the behavior of the system, Figure 3 shows a simple example of a

```
<model>
  <PID>E</PID>
  <classification>
    <ns>http://dmoz.org</ns>
    <path>
        <Computers>
            <Programming>
                <Languages>
                    <Java/>
                </Languages>
            </Programming>
        </Computers>
    </path>
  </classification>
  <classification>
    <ns>http://cis.cs.tu-berlin.de</ns>
    <path>
        <Programming>
                <Java/>
        </Programming>
    </path>
  </classification>
</model>
```

Fig. 3. Peer Model using two different Taxonomies

peer with the PID=”E” providing documents on *Java Programming* classified in the ’global’ Google/DMOZ hierarchy and classified in the ’local’ taxonomy of a university. To simplify our example we use simple letters for peer IDs instead of IP address are unique IDs, such as in JXTA. Lookups for peers are posted against a super-peer.

```
q1=/model/classification/[[ns/’http://dmoz.org’][path/Computers/Programming/Languages/Java]]
q2=/model/classification/[[ns/’http://dmoz.org’][path/Computers/Programming//]]
q3=/model/classification/[[ns/’http://dmoz.org’][path/Computers/Programming/Languages/Java]]
[[http://cis.cs.tu-berlin.de][path/Programming/Java]]
```

Fig. 4. Sample Model Queries

To lookup peer models we use a subset of the XPATH language. Since documents are represented as a tree of nodes, XPATH allows to specify and select parts of a tree. An

XPATH expression contains one or more location steps separated by slashes. Predicates are generally specified between brackets. An XML document matches an XPATH expression when the evaluation of the expression yields a non-null object. E.g. the query *q1* in Figure 4 tests if a model is classified with the *DMOZ* classification and if the model is classified with the path */Computers/Programming/Languages/Java*. Furthermore the ancestor/descendant operator // matches all children. E.g., in query q2 all peer models classified in DMOZ as */Computers/Programming//* and its children, e.g. */Computers/Programming/Languages/Java*, are returned. Query q3 is a conjunctive query, testing the existence of two paths in two different taxonomies.

2.3 Distributed Hash Tables (DHT)

Distributed Hash tables have very desirable characteristics. Their goal is to locate efficiently data items in a very large and dynamic distributed system. As an example DHT we will use the CHORD [19] protocol. The Chord protocol supports just one lookup operation: It maps a given key to a node. Depending on the application this node is responsible for associating the key with the corresponding data item (object). Chord uses hashing to map both keys and node identifiers (such as IP address and port) onto the identifier ring (Figure 1). Each key is assigned to its successor node, which is the nearest node travelling the ring clockwise. Nodes and keys may be added or removed at any time, while Chord maintains efficient lookups using just O(log N) state on each node in the system.

Following [20], current DHT designs provide the technology to create large fully distributed self organizing networks, are resistent against distributed denial of service attacks, require only low costs and provide fresh data. Since keys in a DHT are hashed, storing and searching complex data types in a DHT, such as taxonomies, remains still an open problem. Throughout the paper we will present a new approach to solve this problem.

2.4 Indexing Peer Models and Taxonomies in a DHT

Models are indexed in a catalog based on a Distributed Hash Table. The Catalog is distributed among the SP-SP network. Consider the model with PID=E in figure 3 consisting of two taxonomy paths.

$[EA66]http://dmoz.org/Computers/Programming/Languages/Java$
$[AB55]http://cis.cs.tu-berlin.de/Programming/Java$

For each path a hash value is computed, in CHORD using an SHA-1 algorithm. For each hash value the corresponding object value is PID="E". To represent the tree structure of the taxonomy path further for each taxonomy path tree structured sub sequences are computed and hashed, e.g. for the first and second taxonomy path:

$[\$EA66]http://dmoz.org/Computers/Programming/Languages/Java$
$[\$BB32]http://dmoz.org/Computers/Programming/Languages/$
$[\$6520]http://dmoz.org/Computers/Programming$

$[\$1276] http://dmoz.org/Computers/$
$[\$F076] http://dmoz.org/$

$[\$AB55] http://cis.cs.tu-berlin.de/Programming/Java$
$[\$08BB] http://cis.cs.tu-berlin.de/Programming$
$[\$34FF] http://cis.cs.tu-berlin.de$

For representing the successor relations between keys we use a key to query relationship and represent this within the hash table with the operator $SUCC$. E.g. to express http://dmoz.org/ has the successor http://dmoz.org/Computers/ we use the notation $\$F076\ SUCC\ http://dmoz.org/Computers/$. Table 1 shows keys and object values for taxonomy paths of model PID=E.

Hash Key	**Object Value**
$F076	SUCC ($http://dmoz.org/Computers/$)
$1276	SUCC ($http://dmoz.org/Computers/Programming$)
$6520	SUCC ($http://dmoz.org/Computers/Programming/Languages$)
$BB32	SUCC ($http://dmoz.org/Computers/Programming/Languages/Java$)
$EA66	PID=(E)
$34FF	SUCC ($http://cis.cs.tu-berlin.de/Programming/$)
$08BB	SUCC ($http://cis.cs.tu-berlin.de/Programming/Java$)
$AB55	PID=(E)

Table 1. Hash Keys and inverted list of all models in Figure 1

Often two or more models are classified using the same taxonomy path. E.g. in Figure 1 model with the IDs D, E, F are classified as *http://dmoz.org/../Java*. In the DHT this is reflected by building an inverted index of the corresponding PIDs at the corresponding hash key. The inverted index is ordered. E.g. using the following hash keys for the taxonomy path in Figure 1 Table 2 shows the hash table for all peers of Figure 1. For a better understanding the path is shown in italics, while for simplification the successor relationships are omitted. The complete hash table will be stored as a distributed hash

Hash Key	**Object Value**	*Path Value for the Hash Key, not stored in the real table*
$EA66	PID=(D,E,F)	$http://dmoz.org/Computers/Programming/Languages/Java$
$AB55	PID=(E)	$http://cis.cs.tu-berlin.de/Programming/Java$
$CC33	PID=(B,D)	$http://cis.cs.tu-berlin.de/DataBases/QueryLanguages/SQL$
$D435	PID=(A,B,C,D)	$http://dmoz.org/Computers/UML/Education/$

Table 2. Hash Keys and inverted indices for all models of Figure1

table, in our approach in the CHORD system. Figure 5 shows a CHORD ring with four nodes. Each node SP1..SP4 represents a super-peer. Keys are stored clockwise at the

closest node with the next higher hash value. E.g. the key $AB55 is stored at node SP2 since its node id is $C0BB and the closest node id clockwise. Each node stores the keys of taxonomies and as objects successor relationships as well as the inverted index of PIDs from peer models. For simplifications only already presented successor relationships in table 1 are shown.

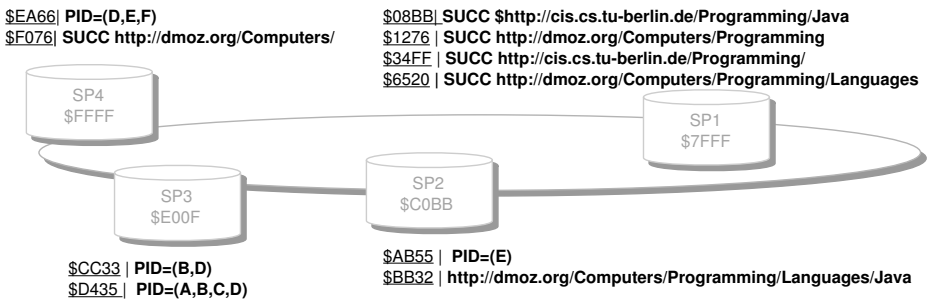

Fig. 5. Storing Models in a CHORD Ring

2.5 Lookup Models in a DHT

We propose tree lookup operations for models in a DHT: Exact lookups with one lookup predicate, BFS-based lookups, where children of a path are also considered using a breath first search (BFS) and conjunctive lookups with two or more predicates. We will now explain the lookups more detailed:

Exact Lookups This is the simplest form of a lookup. An example is query *q1* in Figure 4, where all models are searched matching the following path of the DMOZ taxonomy: $http://dmoz.org/Computers/Programming/Languages/Java$. The taxonomy path of the query is hashed to $EA66 and then a lookup on the Chord ring is executed. The result of the lookup is a set of PIDs storing models with this classification path, e.g the peers with the PID: D,E,F.

BFS-based Lookups Although DHT's only allow an exact lookup, we can provide a 'narrowed' search for the classification path. Since in the DHT the complete classification tree structure is stored, a *Breath First Search* can be easily implemented. Query *q2* in Figure 4 asks for PID's classified as *http://dmoz.org/Computers/Programming/* or as children of the taxonomy path, thus more specialized on *Computer Programming* This is expressed by the operator *//*. To answer this query first the hash value for the path is computed ($6520). Now the object for this hash value is looked up in the index. Following the index, as presented in table 1 or in Figure 5 at this hash value no PID is stored but a key-to-query reference to a successor of this path. A Breath First Search is now executed following this key-to-query references. Finally the result of this query are the PIDs (D,E,F)with a BFS dimension, the distance from the query node origin, of two dimensions. Figure 6 shows the result of *q2* within the context of a more complex tree.

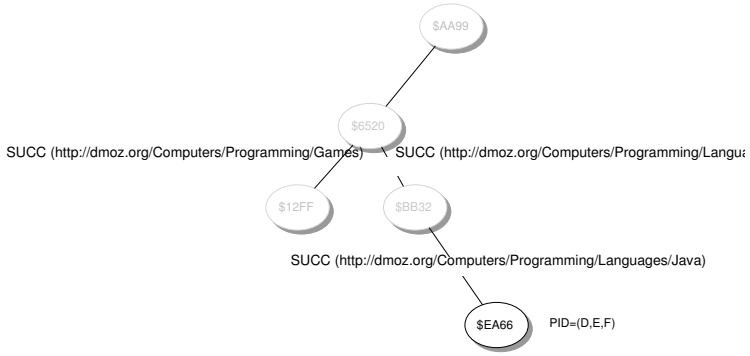

Fig. 6. Breath First Search in a DHT

Conjunctive Lookup In most cases a query consists of two or more taxonomy paths a model has to match. An example for such a query is *q3* in Figure 4, both taxonomy paths

$[\$EA66]\ http://dmoz.org/Computers/Programming/Languages/Java$
AND
$[\$AB55]\ http://cis.cs.tu-berlin.de/Programming/Java$

must match a model of a peer. To solve this problem we have to find the intersection I of the object values sets S_j of the corresponding hash keys

$$S_{\$EA66} = (D, E, F),\ S_{\$AB55} = (E);\ I_{S_{\$AB55} \cap S_{\$EA66}} = (D, E, F) \cap (E) = (E),$$

thus only the peer with the PID=(E) matches both taxonomy paths. An efficient approach known from traditional data base technology is to use the *sort-merge* algorithm. Furthermore, pre-fetching of the set order should be used, since it is cheaper to send the smaller of the two sets to the super-peer holding the larger set; the latter super-peer would perform the intersection and return PIDs. We assume, that this naive approach will be sufficient for our taxonomy-based overlay networks, because PIDs are clustered within an existing taxonomy, thus since sets in our scenario remain considerable small. E.g., in the Open Directory 'large sets' for a path consists of some hundred URLs of relevant web sites only.
The primary challenge in computing the intersection of two sets of peer IDs in a peer to peer environment is limiting the amount of bandwidth used. So far we haven't implemented a specific strategy for computing remote intersections. Although we studied existing research and will give a brief overview of possible strategies: [11] give a good overview over technologies for implementing such an operator for inverted lists in a DHT-based system. Considering their expertise, our approach could be enhanced by compressing the sets using bloom filters or gap compression.We believe, we can reduce the amount of transferred data by the factor 40x.

Reynolds and Vahdat suggest streaming results to users using incremental intersection [17]. Assuming users are usually satisfied with only a partial set of matching results, this will allow savings in communication as users are likely to terminate their queries early. Incremental intersection is most effective when the intersection is big relative to the sets so that a significant number of matching results can be generated without needing to transfer an entire posting set. However, for this approach the results must be ranked. Such a ranking of peers could be for instance by considering the number of documents matching a taxonomy path. Peers with more documents for a path are ranked higher. Evaluating and proving this theses is out of scope of the paper and remains very interesting open work.

3 Storage Load Balancing Strategies

Using a DHT abstraction, distributes objects randomly among peer nodes in a way that results in some nodes having O log(n) times as many objects as the average node. Further imbalance may result due to nonuniform distribution of objects in the identifier space and a high degree of heterogeneity in object loads and node capacities. To reduce this imbalance and to avoid overloading a single super-peer we allow multiple super-peers to share the load of a popular taxonomy path. As a single key can be mapped by the DHT to only one peer we need to introduce a level of indirection. The super-peer that is responsible for a taxonomy path does not store the data under the taxonomy path itself. It only has pointers to other super-peers that are responsible for storing the data. When more than one peer is responsible for storing the data then each responsible peer only stores a part of the data in a partition. A super-peer periodically determines whether its current load is above its target load. In this case the peer requests a new partition for the overloaded key. A new partition is immediately requested when the load jumps above an emergency threshold. New nodes for partitions are acquired from directories that store load information about nodes that have not reached their target load. These directories are also stored in the DHT. Nodes with light load periodically advertise their current target load to one of the directories.

Inserting data in the DHT requires two steps (see Figure 7). First the insert location

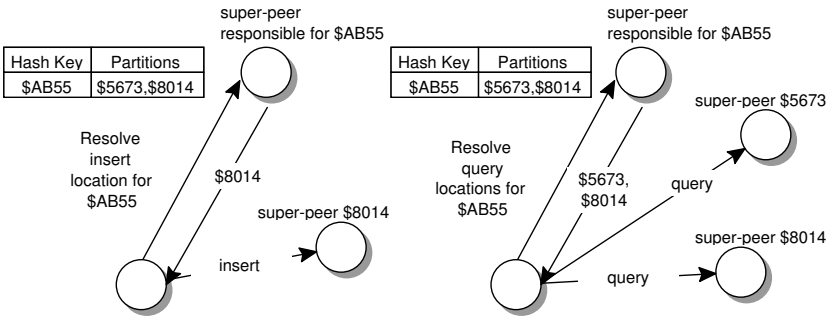

Fig. 7. Partition based Storage Load Balancing

needs to be resolved. A request to get an insert location is sent to the peer that would normally store the data. This peer returns one of the peers that are responsible for storing data under the requested key. Every time an insert request is issued a different peer is returned. Thus the data is uniformly distributed among the partitions. The data is then sent to the received insert location. Queries also take two steps. The query locations are resolved in the same way but this time all partitions are returned. In order to query all data the query must be send to all returned partitions.

4 Implementation and Evaluation

The main benefits of our load balancing approach over systems like [16] that move keys between virtual nodes is that we don't have the overheads caused by virtual nodes. Further, virtual nodes storing popular keys would receive a high number of object insertions, which would be too high for a single node to handle.

But this comes at an increased query cost when there is more than one partition. To prove this theory and compare the costs of our approach in terms of used bandwidth in contrast to virtual server we implemented both approaches using the *Narses* simulation framework [12]. We simulated a catalog based on 50 super-peers and 15000 peers. Each peer node joins and leaves the catalog within 3600 sec. The generated profile of each peer node includes on average ten taxonomy paths and has an average size of 800 bytes. The taxonomy paths in the peer profiles are Zipf distributed, regarding the fact that most of the peers share popular music but only a few share rare music. We simulated the required bandwidth of the peer-to-super network and the super-peer to super-peer network, using four different load balancing approaches: without load balancing (-LBM, -VS), virtual server(+VS), partition based load balancing (+LBM) and finally the combination of partition based load balancing and virtual server (+LBM,+VS). Figure 8 shows the results of our simulation. Our load balancing approach performs better

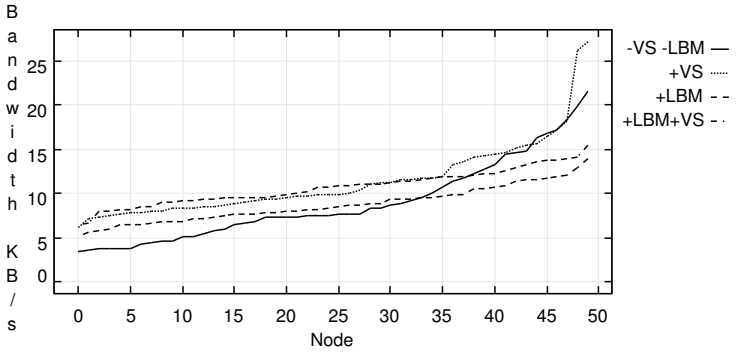

Fig. 8. Bandwidth usage for storing and removing peer profiles

than virtual server and the simulation without any load balancing. The combination of

our approach with virtual server produces an extra overhead of 10% in contrast to a simulation using partition based load balancing only. This result are valid for a small super-peer network, such as simulated in our experiment. However we assume that for larger super-peer networks the combination of our approach and virtual server will work best. Please note, in our approach we are only able to reduce the number of taxonomy paths a super-peer is responsible for. Virtual server allow to distribute the load over

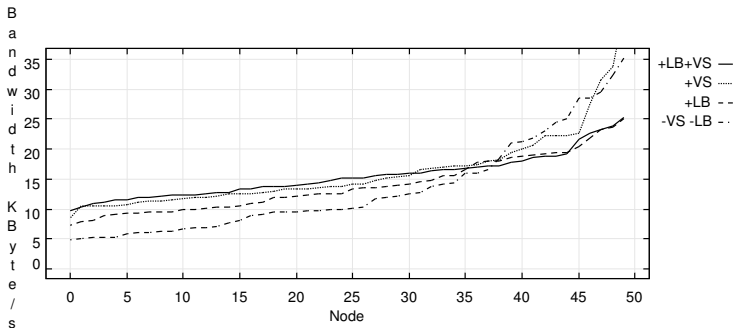

Fig. 9. Overall Bandwidth usage of the System

several super-peers using a many to many scheme. To prove this thesis we will conduct further experiments and mathematical calculations.

To analyze the overall bandwidth of the approach including queries we conducted a number of experiments using the load balancing strategies from above. Each peer issues each 240 sec an exact query for a taxonomy path. Our simulation in Figure 9 has shown that the average required bandwidth for serving queries and joining and leaving peers of each super-peer is 25 KByte/sec. For running a small network with only 15.000 peers less than DSL bandwidth is required. However, our storage load balancing approach is not completely able to reduce the query load balancing costs. Figure 10 shows the costs using our storage load balancing approach only for joining leaving peer nodes (J/L) and for issuing queries and joining and leaving peer nodes (J/L +Query). Since queries are Zipf distributed as well, we suggest caching taxonomy paths at intermediate nodes that lie on the path taken by search query and refer to this as *Path Caching with Expiration (PCX)* because cached taxonomy path entries typically have expiration times after which they are considered stale and require a new search. [18] proposes *Controlled Update Propagation* a more sophisticated approach to maintain caches of meta data in a peer-to-peer network where each (super-peer) node maintains a directory of the cached copies of its meta data, thus no expiration time is needed. PCX and CUP are desirable because they distributes query load for popular taxonomies across multiple nodes, reduces latency, and alleviates hot spots.

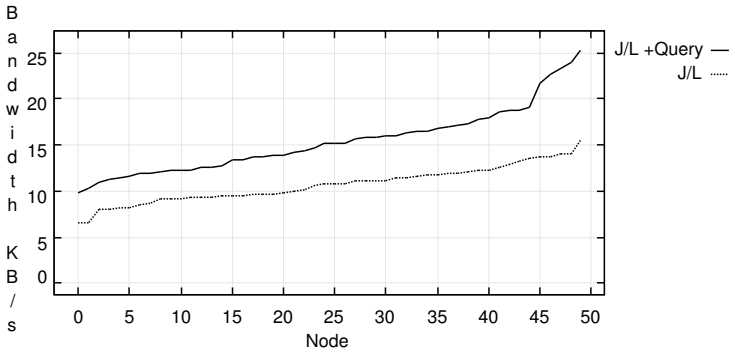

Fig. 10. Overall Bandwidth Usage vrs. Bandwidth Usage for Join/Leave Operations only

5 Related Work

Our approach incorporates mainly work from distributed databases, DHT-based resource discovery approaches and (semantic) overlay networks. In the database community catalogs to select relevant data sources for a given query have been investigated recently. However, most of the approaches assume a central broker architecture [15,5], which does not scale well or an asymmetric broker topology [9,4], which is costly and hard to maintain.

Edutella [14] proposes routing and clustering strategies in peer-to-peer networks for educational materials based on routing indices. Information provider and consumer connect to a super-peer network and propose its query capabilities as well as queries. Queries are routed along the routing indices only to information provider peers matching the query capabilities. The use of super-peer reduces the frequency of joining and leaving peers significantly and concentrates routing and query processing only to super-peers. The approach works for small-to-medium sized network. However, cost for maintaining the routing indices and routing costs grow linear with the number of peers connected to the system.

To our best knowledge, distributed catalogs for data source selection based on a symmetric topology have been investigated only by Galanis et.al [6]. The approach stores attributes of query schemas in a DHT. The idea is interesting, however a search is limited to exact matches of schema attributes. Content-specific attributes, such as taxonomy classifications, as well as 'fuzzy' searches are not considered.

Crespo et.al [3] inspired us with the idea of Semantic Overlay Networks (SON) and proposed to classify peers and queries by taxonomies. Music sharing peers classify its MP3 files using existing taxonomies and publish only the most popular taxonomies of each peer in a catalog, queries are also classified against taxonomies. Query routing is two folded: first queries are matched against peer descriptions and later routed to a small set of peers forming a suitable SON. Our work bases on this idea, however since the authors gave no further details about efficient querying and storing taxonomies in a

distributed environment, we extended their vision with our approach of storing the peer descriptions in DHT-based catalog and enabling hierarchical, as well as conjunctive queries.

More closely to our approach come DHT based publish/subscribe systems. INS/ Twine[2], a resource discovery system, extracts prefix subsequences of attributes and values, called 'strands' from resource descriptions and indexes the hash values for each of these strands in a peer-based resolver. In contrast to our approach the system has the disadvantage that it assumes all resources are equally requested and stores descriptions of resources redundantly in the network and resource and device information are stored redundantly on all peer resolvers that correspond to the numeric keys.

pSearch [21] is a system that organizes contents around their semantic in a P2P network. This makes it possible to achieve an accuracy comparable to state-of-the-art centralized IR systems while visiting only a small number of nodes. The authors of pSearch propose the use of hierarchical clustering, rolling-index, and content-directed search to reduce the dimensionality of the search space and to resolve the dimensionality mismatch between semantic space and CAN and employ content-aware node bootstrapping to balance the load.

6 Summary and Further Work

We presented a completely new approach for enabling efficient semantic query routing in P2P networks. Queries are no longer flooded throughout the whole system but are routed only to peers providing the right content to answers a query. Our basic concept is a distributed catalog, storing semantic annotations about the available content of peers in the network. Our approach is capable of storing a large number of peer profiles in a space efficient way. Due to the use of key-to-query mappings, inverted indices and tree structured sub sequences we are able to store large taxonomies in a DHT. Our data structure allows to resolve exact, narrowed and broadened queries. Further we presented strategies for realizing conjunctive queries in our system. The catalog is built on top of a distributed hash table based infrastructure using a partition based storage load balancing approach. Our simulation has shown, that our approach scales to ten thousands of peers using a small set of super-peers connected to each other with less than DSL bandwidth.

However, this paper offered only a high-level description of our approach. Much work remains, for example dynamic storage load balancing strategies allowing super-peers to join and leave the catalog with a high frequency while the catalog remains robust. The use of existing query load balancing strategies, such as CUP or PCX should be evaluated, to allow the system to scale to a high number of queries. Further, using our existing literature survey we will evaluate conjunctive lookup queries based on incremental intersections of ranked sets. To reduce costs of a tree based search we will investigate the use of an iterative deepening depth first search strategy, limiting results only to top N peer nodes. Last, we allow only hierarchical relations to be stored in the DHT. Although not needed in our scenario our data structure allows to store much more complex constructs than rather simple hierarchies. Therefore our further work will include the formulation of a road map towards storing and querying such structures.

Acknowledgements I thank Wolfgang Nejdl, Martin Wolpers and Wolf Siberski from the EDUTELLA project for comments on various aspects of our work and I thank Ralf-Detlef Kutsche and Agnes Voisard from the Fraunhofer Institute ISST Berlin for many helpful discussions.

References

1. K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. The chatty web: Emergent semantics through gossiping. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.
2. M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *M. Balazinska, H. Balakrishnan, and D. Karger. In Proceedings of Pervasive 2002.*, 2002.
3. A. Crespo and H. G. Molina. Semantic overlay networks, November 2002. Stanford University, Technical Report.
4. R. Dolin, D. Agrawal, L. Dillon, and A. E. Abbadi. Pharos: A scalable distributed architecture for locating heterogeneous information sources. Technical Report TRCS96-05, 16, 1996.
5. J. C. French, A. L. Powell, C. L. Viles, T. Emmitt, and K. J. Prey. Evaluating database selection techniques: A testbed and experiment. In *Research and Development in Information Retrieval*, pages 121–129, 1998.
6. L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating data sources in large distributed systems. In *Proceedings of the 29 VLDB*, Berlin, Germany, 2003.
7. H. Garcia-Molina and B. Yang. Efficient search in peer-to-peer networks. In *Proceedings of ICDCS*, 2002.
8. L. Gong. Project JXTA: A technology overview. Technical report, SUN Microsystems, Apr. 2001. http://www.jxta.org/project/www/docs/TechOverview.pdf.
9. L. Gravano and H. García-Molina. Generalizing GlOSS to vector-space databases and broker hierarchies. In *International Conference on Very Large Databases, VLDB*, pages 78–89, 1995.
10. T. Joachims. Text categorization with support vector machines: learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.
11. J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems. ITPTS*, 2003.
12. M.Baker and T. Giuli. Narses: A scalable flow-based network simulator. Technical report, Stanford University, 2002.
13. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P Networking Infrastructure based on RDF. In *Proceedings of the 11th International World Wide Web Conference*, Hawaii, USA, May 2002. http://edutella.jxta.org/reports/edutella-whitepaper.pdf.
14. W. Nejdl, M. Wolpers, W. Siberski, A. Löser, I. Bruckhorst, M. Schlosser, and C. Schmitz. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.
15. M. Oezsu and P. Valduriez. *Principles of distributed database systems*. Prentice Hall, 2nd edition edition, 1999.

16. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Proc. IPTPS*, 2003.
17. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *ACM MIDDLEWARE Conference*, 2003.
18. M. Roussopoulos and M. Baker. Cup: Controlled update propagation in peer to peer networks. In *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.
19. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. pages 149–160.
20. C. Tang and S. Dwarkadas. Peer-to-peer information retrieval in distributed hashtable systems. In *USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.
21. C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.
22. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proccedings of the ICDE*, March 2003.