Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

TWOLAYERED ARCHITECTURE FOR PEERTOPEER CONCEPT SEARCH

Dharanipragada Janakiram,
Fausto Giunchiglia, Harisankar Haridas and
Uladzimir Kharkevich

March 2010

Technical Report # DISI-10-022

Two-Layered Architecture for Peer-to-Peer Concept Search

Dharanipragada Janakiram
Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India
djram@cse.iitm.ac.in

Fausto Giunchiglia
Department of Information Engineering and Computer Science University of Trento, Italy
fausto@disi.unitn.it

Harisankar Haridas
Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India
hhari@cse.iitm.ac.in

Uladzimir Kharkevich
Department of Information Engineering and Computer Science University of Trento, Italy
kharkevi@disi.unitn.it

ABSTRACT

Peer-to-peer search is an alternative to address the scalability concerns regarding centralized search, due to its inherent scalability in terms of computational resources. Concept search is an information retrieval approach which is based on retrieval models and data structures of syntactic search, but which searches for concepts rather than words thus addressing the ambiguity problems of syntactic search approach. In this paper, we combine peer-to-peer search with concept search and compare two different approaches to peer-to-peer concept-based search. In a single layer peer-to-peer concept search, a single universal knowledge is used to map terms to concepts. In a two layered peer-to-peer concept search, the universal knowledge is used in identifying the appropriate community given query terms and the community's background knowledge is used to map query terms to appropriate concepts within selected community. Since the search is done only in relevant communities, there will be improvement in bandwidth utilization for search. Since two layered peer-to-peer concept search uses the community's background knowledge to map terms to concepts, it improves quality and efficiency of search. We perform experiments to compare the two layered peer-to-peer concept search with single layered peer-to-peer concept search and present our results in this paper. We also show in this paper how the two layers of knowledge help in achieving scalable and interoperable semantics.

1. INTRODUCTION

Conventional search engines implement search for documents by using *syntactic search*, i.e., words or multi-word phrases are used as atomic elements in document and query representations. The search procedure, in syntactic search, is essentially based on the *syntactic matching* of document and query representations. But semantics need to be considered to improve the search quality. Our earlier work, *Concept Search* [8] (*CSearch* in short) extends syntactic search with semantics. The main idea is to keep the same machinery which has made syntactic search so successful, but to modify it so that, whenever possible, syntactic search is substituted with semantic search, thus improving the system quality. As a special case, when no semantic information is available, *CSearch* reduces to syntactic search, i.e., the results produced by *CSearch* and syntactic search are the same. But, semantics need to be introduced in a scalable and interoperable manner.

WWW2010, April 26-30, 2010, Raleigh, North Carolina.

Nowadays, the major search engines are based on a centralized architecture. The size, dynamics, and distributed nature of the Web make the search problem extremely hard, i.e., a very powerful server farm is required to have complete and up-to-date knowledge about the whole network to index it. Peer-to-peer architecture has been shown to be a scalable alternative over a centralized or a super-peer architecture in various applications like file sharing, voice chat etc. Robustness, inherent scalability, availability and lack of central authority are major advantages of the P2P architecture over the centralized architecture. In case of peer-to-peer search, each peer in the P2P network organizes only a small portion of the documents in the network, while being able to access the information stored in the whole network. Efforts are going on to improve the scalability of peer-to-peer information retrieval.

In this work, we propose a two layered architecture for peer-to-peer search, which introduces semantics to search in a two-layered manner. The architecture aims to achieve scalable and interoperable semantics for performing search. At the same time, it also attempts to improve the scalability of peer-to-peer search in terms of network efficiency.

In the rest of the paper, we first give detailed motivation and overview of the two-layered architecture in Section 2. The section provides high level description of the components and provides references to appropriate sections for details. We then discuss details of community formation and search within community in Section 3. We discuss the inter-community (global) search in Section 4. We show experimental results based on the current version of the prototype in Section 5, survey the related work in Section 6 and conclude in Section 7.

2. TWO-LAYERED ARCHITECTURE

The motivations for a two layered architecture for peer-to-peer concept search are explained first. Then, the details of the architecture are explained.

2.1 Need for a two layered architecture

The following are the motivating factors for a two layered architecture for peer-to-peer concept search

2.1.1 fractal nature of semantics

As discussed in [4], most of the ontologies will be established by small communities working together on specific domains, where more agreement on semantics exists. A global shared ontology should be present for providing interoperability across communities. Hence, to achieve scalable and interoperable semantics globally, community specific background knowledge bases maintained by different communities by extending parts of the universally accepted knowledge, based on their specific interests are required. Hence, a two layered architecture is required which facilitates formation of communities with different background knowledge bases in the lower layer, while maintaining the globally accepted universal knowledge in the top layer.

2.1.2 word sense disambiguation problem

The terms in query and documents need to be mapped into concepts for performing concept based search. The set of possible candidate concepts for a word can be obtained from a knowledge base. This set can be further narrowed down based on the context of the word through word sense disambiguation (WSD). But WSD is a hard task given limited context in search queries and gives low performance thereby affecting search quality. But the performance of WSD can be significantly improved if WSD is performed in a specific branch of knowledge as explained in [5]. Within a specific domain knowledge, the set of possible candidate concepts for a word will be smaller than that in a universal knowledge base. Thus, by grouping peers into domain-based communities in the lower layer and performing WSD for search based on the background knowledge of a community, the quality of peer-to-peer concept search can be improved. Universal knowledge in the global layer ensures interoperability across communities.

2.1.3 scoped search and parallelization

The two layered architecture for search enables to identify and search only relevant communities in the network for a document, rather than searching the entire network. Thus, it improves the network utilization incurred during search. Since, the search in multiple relevant communities can be performed in parallel, the search speed is also improved.

2.2 Two layered architecture for peer-to-peer concept search

The two layer architecture decomposes various functionalities required for performing peer-to-peer search into two layers viz, global layer and community layer. This enables to classify and implement various search related operations at appropriate levels of granularity. The peers are organized into communities based on their interest. Each community has its own background knowledge maintained within, based on the specific interests of the peers in the community. Document indexing and search within the community is performed based on the background knowledge of the community. To enable interoperability of knowledge/semantics and search visibility across communities, the global layer is maintained connecting all the communities. The global layer maintains an index of the communities for enabling search

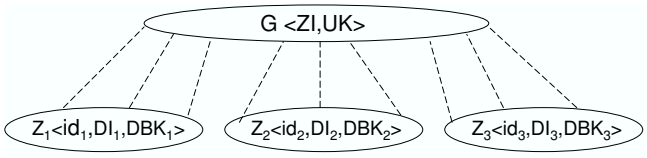

Figure 1: Two layered architecture for P2P CSearch

across communities. It also maintains a universal knowledge base, accepted globally to ensure interoperability among the different knowledge bases maintained by the communities. The two layered architecture for peer-to-peer concept search is represented in figure 1. Formally,

The set of peers, P is organized as a set of communities Z $\{Z_1, Z_2, Z_3, ..., Z_z\}$. Each community is defined by $< id_i, P_i, DI_i, DBK_i >$ where:

id_i is a unique identifier for the community.

P_i is the set of peers forming the community, $P_i \in 2^P$,

DBK_i consists of the community's distributed background knowledge based on a set of concepts BC_i. DBK_i is collaboratively managed by the peers in the community, P_i. The details of DBK are explained in section 3.3.1,

DI_i is the document index of the documents shared by peers in P_i based on the set of concepts BC_i,

There is a global layer G defined by $< ZI, UK >$ where:

UK is the universal knowledge accepted globally based on a set of concepts, UC.

ZI is an index of the communities based on the concepts in UC.

Each DBK_i is bootstrapped from the UK to ensure interoperability and extended further by the community.

Search can be performed in two modes:

Search within community: The query Q expressed in natural language is used for search within community Z_i. The search within community is performed based on the DBK_i of the community returning results R_{Qi} to the querying peer, ie $ZS(Q, Z_i) \rightarrow R_{Qi}$. Hence, word sense disambiguation for search and semantics enabled search are performed at a community-level granularity. Implementation details of search within community are explained in section 3.3.

Search for communities: The query Q, expressed in natural language, is used to search for communities in the top layer. A distributed search scheme based on UK, GS returns a list of communities Z_Q, relevant to the query Q, i.e., $GS(Q, G) \rightarrow Z_Q$. Implementation details of the search for communities are explained in section 4.1.

Using the above modes for search supported by the architecture, the peers can search in the following manners:

- **Scheme 1**: The peer can select one of the communities of which it is a member to perform the search.
- **Scheme 2**: The peer can initially search for the list of relevant communities for a query. Then, interesting communities can be chosen and searched in parallel.
- **Scheme 3**: The peer can search without choosing any community. In this case, the relevant communities for the query are automatically identified and search is performed on the most promising communities.

Overlay structure. The overlay structure used to implement the peer-to-peer two layered architecture is derived from our earlier work on Vishwa [15] and Virat [20]. Vishwa

is a reconfigurable peer-to-peer middleware for grid computations while Virat defines a node-capability aware replica management scheme for peer-to-peer grids.

The nodes in the peer-to-peer network are organized into communities. A global layer ensures connectivity across communities. A dynamically selected node, called the bootstrap node is used to bootstrap nodes into the community. On contacting the bootstrap node, a new node is assigned a unique id by prefixing the community id to a quasi random identifier generated using SHA-1. The node then becomes part of two overlays: Pastry [17] within the community (intra-community bootstrap) and a Chord ring [21] across communities (inter-community bootstrap). The details of overlay structure is available in [15], but the important features of the overlay are briefly explained below.

Every joining node gets mostly different set of nodes from its neighboring communities as part of its independent Chord ring. The different Chord rings together, form the global layer overlay. Pastry ensures efficiency in intra-community routing while Chord across communities ensures correctness in routing even if very few routing table entries are correct. Lookups with a guarantee of O(logN) hops are supported. The total routing table size per node is also O(logN) which is same as that of a single large DHT.

When a peer initially joins in the system, it participates in the overlay of the community to which it belongs. When a peer needs to participate in more communities, it establishes a link with one peer each from the desired communities (sends a subscription message). The other peers act as proxies for the peer in the other communities, i.e., indexing, search, modification of background knowledge etc. in the other communities are performed by the peer through the proxy peers. Note that this is equivalent to the concept of client and super-peer in superpeer networks. Henceforth, a peer in a community could mean either a peer participating in the community overlay, or a proxy peer in the community overlay acting on behalf of another peer.

The storing (*put*) and retrieval (*get*) operations of values indexed with keys in the overlay are as explained below:

put and *get* operations in the community layer and global layer are seamlessly supported by the id generation scheme. Consider two bit sequences A and B. Let $[A,B]$ denote a n-bit identifier with an m-bit prefix having the first m-bits of A and the remaining part having the first $(n\text{-}m)$ bits of B.

The node identifier of peer P within a community having a community id *cid* will be of the form [hash(*cid*),hash($P_{IP:PORT}$) where $P_{IP:PORT}$ is the IP address : port combination of the peer application. An identifier for key k within a community with community id *cid* will be of the form [hash(*cid*),hash(k)] where hash() is the SHA-1 hash function. An identifier for key k in the global layer will be of the form [hash(k),hash(k)].

The *put* and *get* primitives are defined as:

- *put*(b,*key*,*value*) will store *value* in the node responsible for identifier [hash(b),hash(*key*)].
- *get*(b,*key*) → *value* will return the *value* for the identifier [hash(b),hash(*key*)].

Based on the above two primitives the following operations can be performed by a peer within a community C with community id *cid*:

- **putC(**k,*object*) - stores *object* within the community C with key k. Internally, a *put*(*cid*,k,*object*) is performed.
- **getC(**k) → *object* - returns the object associated with key k from the community C. Internally, a *get*(*cid*,k) is performed.
- **putG(**k,*value*) - stores *object* in the global layer with key k. Internally, a *put*(k,k,*object*) is performed.
- **getG(**k) → *object* - returns the object associated with key k from the global layer. Internally, a *get*(k,k) is performed.

Thus the search application can use the above methods and seamlessly perform operations in the two layers.

3. SEARCH WITHIN A COMMUNITY

In our approach, communities are created in the same way as public communities are formed in case of flickr, orkut etc. The peer deciding to start a community provides the name and description of the community. A unique community identifier is generated for the community based on the community name. A new community overlay is formed and the peer initially acts as the bootstrap peer within the community. The community overlay gets connected to the global layer through inter-community bootstrap. The bootstrap peer then obtains part of the universal knowledge relevant to the community from the global layer, which forms the background knowledge of the community. The creator peer of the community will be the bootstrap peer for the community initially. Later, other peers can become the bootstrap peer based on node identifier (bootstrap peer can be the peer responsible for a specific id within the community).

A new peer can join the community by contacting any other peer in the community. The peer joining a community performs a bootstrap in the community overlay by contacting the bootstrap node of the community. The bootstrapping procedure also ensures the participation of the peer in the global overlay. The members of the community can collaboratively edit the background knowledge of the community using various primitives described in detail in Section 3.3.1. Moreover, the members can search for the documents which are indexed within the community based on the community-specific background knowledge.

3.1 DHT Based Syntactic Search

A straightforward way to implement syntactic search in a single community is to use the DHT to distribute peers' inverted indexes in the P2P network [16]. Peers locally compute posting lists $P(t)$ for every term t and store them in the network by using the DHT 'put' operation. The *key* in this case is a term t while the *value* is a posting list $P(t)$ associated with t. In DHT, each peer is responsible for a few terms and for every such term t, the peer merges all the posting lists $P(t)$ for t from all the peers in the network. In order to find the set of documents which contains a term t we just need to contact the peer responsible for t and retrieve the corresponding posting list. The DHT 'get' operation does exactly this. In the basic scheme, in order to search for more than one term, we, first, need to retrieve posting lists for every single term, and then to intersect all these posting lists.

The above approach has several problems (see e.g. [9, 23]). Let us consider some of these problems.

Storage. For a large document collection, the number and the size of posting lists can be also large. Therefore,

the storage needed to store the posting lists can potentially be bigger than the storage peers can (or want to) allocate.

Traffic. Posting lists need to be transferred when peers join or leave the network. Searching with multiple terms requires intersection of posting lists, which also need to be transferred. In [9], it is shown that the efficiency of DHT can be even worse than the efficiency of a simple flooding algorithm.

Load balancing. Popularity of terms, i.e., the number of occurrences of the terms, can vary enormously among different terms. It can result in an extremely imbalanced load e.g., some peers will store and transfer much more data than others.

Several approaches were proposed in order to address the above described problems and to improve performance of information retrieval in structured P2P networks. Some of the optimization techniques (e.g., Bloom Filters), which can improve the performance of posting list intersection, are summarized in [9]. Caching of results for queries with multiple terms is discussed e.g. in [19]. In [19] only those queries which are frequent enough are cached and simple flooding is used for rare queries. In [23], only important (or top) terms are used for indexing of each document. Some techniques to balance the load across the peers are also presented in [23]. Normally, users are interested only in a few (k) high quality answers. An example of the approach for retrieving *top k* results, which does not require transmitting of entire posting lists, is discussed in [26]. In [11], indexing is performed by terms and term sets appearing in a limited number of documents. Different filtering techniques are used in [11], in order to make the vocabulary to grow linearly with respect to the document collection size.

Ambiguity. Another important problem with the above approaches is that all of them implement syntactic search, i.e., a word or a multi-word phrase is used as an atomic element in document and query representations and a syntactic matching of words is used for matching document and query terms. Therefore, the problems of syntactic search, i.e., problems of (i) polysemy, (ii) synonymy, (iii) complex concepts, and (iv) related concepts [8], can also affect the quality of the results produced by these approaches.

3.2 Concept Search

In order to address the problems of syntactic search, in *CSearch* [8], syntactic search is extended with semantics, i.e., words are substituted with (complex) concepts and syntactic matching of words is extended to semantic matching of (complex) concepts. Below, we briefly describe how the words are converted into the complex concepts and also how the semantic matching $SMatch$ is implemented in *CSearch*. We refer the interested reader to [8] for a complete account.

In *CSearch*, complex concepts (C) are computed by analyzing meaning of the words and natural language phrases (e.g., noun phrases). Single words are converted into atomic concepts uniquely identified as *lemma-sn*, where *lemma* is the lemma of the word, and *sn* is the sense number in the background knowledge BK (e.g., WordNet). For instance, the word *dog* used in the sense of a domestic dog, which is the first sense in the BK, is converted into the atomic concept *dog*-1. Note that if no relevant concepts were found for a word then the word itself is used as a concept. If there are many possible candidate concepts for a word and the meaning of the word cannot be reliably disambiguated, then we keep all the possible concepts of the word. Noun phrases are translated into the logical conjunction of all the atomic concepts corresponding to the words, e.g., the noun phrase *A little dog* is translated into the concept *little*-4 $\sqcap$ *dog*-1.

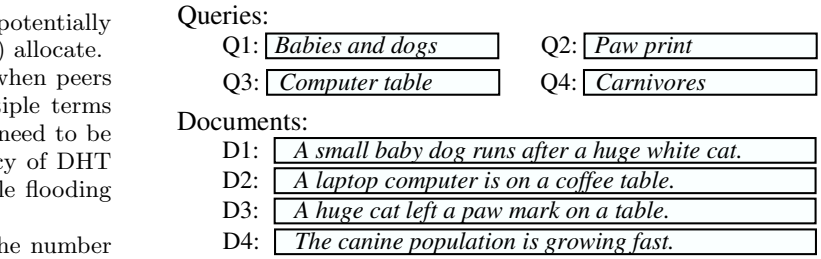

Figure 2: Queries and a document collection

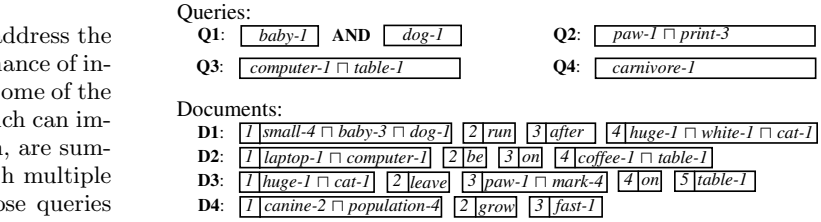

Figure 3: Document and Query Representations

After computing complex concepts, documents are represented as enumerated sequences of these complex concepts $\sqcap A^d$, i.e. conjunctions ($\sqcap$) of atomic concepts (A^d). For example, in Figure 3, we show the sequences of $\sqcap A^d$ extracted from documents in Figure 2. The examples are taken from [8]. Rectangles in Figure 3 represent complex concepts $\sqcap A^d$. The number in the square at the left side of a rectangle represents the *position* of the rectangle.

In *CSearch*, we can search for documents describing complex concepts which are semantically related to complex concepts in the user query. Formally a query answer $QA(C^q, \mathcal{T})$ for a single complex concept C^q is defined as follows:

$$QA(C^q, \mathcal{T}) = \{d \mid \exists C^d \in d, \text{ s.t. } \mathcal{T} \models C^d \sqsubseteq C^q\} \quad (1)$$

where C^q is a complex query concept extracted from the query q, C^d is a complex document concept extracted from the document d, and $\mathcal{T}$ is a terminological knowledge base (i.e., the BK) which is used in order to check if C^d is more specific then C^q. $\mathcal{T}$ can be thought of as an acyclic graph, where links represent subsumption relations in the form $A_i \sqsubseteq A_j$, with A_i and A_j atomic concepts.

Query answer $QA(C^q, \mathcal{T})$, defined in Equation 1, is computed by using a positional inverted index, where the inverted index dictionary consists of atomic concepts from $\mathcal{T}$ (e.g., concepts *baby*-3 and *dog*-1). The posting list $P(A) = [\langle d, freq, [position]\rangle]$ for an atomic concept A stores the positions of complex concepts which contain A, where $\langle d, freq, [position]\rangle$ is a posting consisting of a document d associated with A, the frequency $freq$ of A in d, and a list [*position*] of positions of A in d. For example, *P(dog-1)*=$\langle D1, 1, [1]\rangle$.

Note that in this paper, we used a modified version of *CSearch*, where instead of indexing document concepts by more general atomic concepts, we expanded atomic concepts in the complex query concepts by more specific atomic concepts. This modification allows *CSearch* to adapt to

changes in the background knowledge without the need for re-indexing of the documents. The drawback of this approach is that the search time grows with the number of expanded concepts. To keep the time reasonable, we constrain the number of more specific concepts which can be used in concept expansion process.

In *CSearch*, query concepts C^q can be combined into more complex queries q, e.g., by using the boolean operator AND. Query answer $QA(q, \mathcal{T})$ in this case is computed as follows:

$$QA(q_i \text{ AND } q_j, \mathcal{T}) = QA(q_i, \mathcal{T}) \cap QA(q_j, \mathcal{T}) \quad (2)$$

For instance, the query answer for query *baby*-1 AND *dog*-1 (in Figure 3) is computed as follows: $QA($*baby*-1 AND *dog*-1, $\mathcal{T}) = QA($*baby*-1, $\mathcal{T}) \cap QA($*dog*-1, $\mathcal{T}) = \emptyset \cap \{D1, D3\} = \emptyset$

The main limitation of *CSearch* is that it is a centralized system, i.e., the BK and the inverted index are stored in a single place. As any other centralized system, *CSearch* cannot scale without the need for powerful servers.

3.3 DHT Based Concept Search

This section defines the search within community operation ($ZS(Q, Z_i) \rightarrow R_{Qi}$) mentioned in section 2.

In order to perform semantic search within the community overlay (DHT), we propose to extend the centralized version of *CSearch* to *P2P CSearch*[1]. First, we extend the reasoning with respect to a single background knowledge $\mathcal{T}$ to the reasoning with respect to the background knowledge $\mathcal{T}_{P2P}$ which is distributed among all the peers in the community. Second, we extend the centralized inverted index (II) to distributed inverted index build on top of DHT. The idea is schematically represented in the equation below.

$$\boxed{CSearch \xrightarrow{Knowledge(\mathcal{T} \rightarrow \mathcal{T}_{P2P}), Index(II \rightarrow DHT)} P2P\ CSearch}$$

In the following, we show how, the distributed background knowledge $\mathcal{T}_{P2P}$ can be implemented on top of DHT and also, how DHT can be used, in *P2P CSearch*, to perform efficient distributed semantic indexing and retrieval.

3.3.1 Distributed Background Knowledge

To access the background knowledge $\mathcal{T}$, stored on a single peer, *CSearch* needs at least the following three methods:

getConcepts(W) returns the set of all the possible meanings (atomic concepts A) for the word W. For example, *getConcepts(canine)* $\rightarrow$ {*canine-1 ('conical tooth'), canine-2 ('mammal with long muzzles')*}.

getChildren(A) returns the set of all the more specific atomic concepts which are directly connected to the given atomic concept A in $\mathcal{T}$. For example, *getChildren (carnivore-1)* $\rightarrow$ {*canine-2, feline-1*}.

getParents(A) returns the set of all the more general atomic concepts which are directly connected to A in $\mathcal{T}$. For example, *getParents(dog-1)*$\rightarrow${*canine-2*}.

In order to provide access to background knowledge $\mathcal{T}_{P2P}$ distributed over all the peers in the P2P network, we create distributed background knowledge *DBK*. In *DBK*, each atomic concept, A is identified by a unique concept ID (A_{ID}) which is composed of peer ID (P_{ID}), where peer is the creator of the atomic concept, and local concept ID in the Knowledge Base of the peer. Every atomic concept A is represented as a 3-tuple: $A = \langle A_{ID}, POS, GLOSS \rangle$, where A_{ID} is the concept ID; POS is the part of speech; and $GLOSS$ is the natural language description of A. In the rest of the paper, for the sake of presentation, instead of complete representation $\langle A_{ID}, POS, GLOSS \rangle$ we use just *lemma-sn*.

DBK is created on top of the community DHT. Atomic concepts are indexed by words using the *putC* operation, e.g., *putC(canine,* {*canine-1, canine-2*}*)*. Moreover, every atomic concept is also indexed by related atomic concepts together with the corresponding relations. We use a modification of the *putC* operation, *putC(A, B, Rel)*, which stores atomic concept B with relation *Rel* on the peer responsible for (a hash of) atomic concept A, e.g., *putC(canine-2, dog-1, '$\sqsubseteq$'), putC(canine-2, carnivore-1, '$\sqsupseteq$')*.

After the *DBK* has been created, *getConcepts(W)* can be implemented by using the *getC* operation, i.e., getConcepts(W) = getC(W). Both methods *getChildren(A)* and *getParents(A)* are implemented by using a modified *getC* operation *getC(A, Rel)*, i.e., *getChildren(A)* = *getC(W, '$\sqsubseteq$')* and *getParents(A)* = *getC(W, '$\sqsupseteq$')*. The operation *getC(A, Rel)* finds a peer responsible for atomic concept A and retrieves only those concepts B which are in relation *Rel* with A.

Let us now see how *DBK* can be bootstrapped. In the beginning, we have only one single peer (creator) in the community and *DBK* is equivalent to the background knowledge $\mathcal{T}$ of this peer. A new peer joining the community bootstrap its own background knowledge from *DBK* through the following three steps. First, the peer computes a set of words which are used in the local document collection. Second, the peer downloads from *DBK*, the set of all the atomic concepts which are associated with these words by using 'getConcepts' method. Finally, the peer downloads all the more general atomic concepts by recursively calling 'getParents' method. After bootstrapping, the user of each peer can extend the knowledge stored in *DBK* in the domain of her expertise according to her needs.

The potential problem with the above approach is that it can require a lot of messages to collect and keep up to date all the more general concepts for every concept on each peer. In order to address this problem, we propose to cache the content of *DBK* on a special peer which we call the *caching peer*. We fix an ID in the community DHT, and the *caching peer* is dynamically selected based on the range of IDs the peer is responsible for (i.e., the ID should belong to the ID space of the peer). The request for a part of the DBK first goes to the *caching peer* (whose ip address can be cached) and then if the *caching peer* is busy or unavailable the request is processed by using the *DBK*.

3.3.2 Indexing and Retrieval

The query answer defined in Equation 1, can be extended to the case of distributed search in community by taking into account of the fact that the document collection $D = D_{P2P}$ is equivalent to the union of all the documents from all the peers in the community (where each document d is uniquely identified by an ID) and also that the background knowledge $\mathcal{T}_{P2P}$ is distributed among all the peers in the community.

$$QA(C^q, \mathcal{T}_{P2P}) = \{d \in D_{P2P} \mid \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q\} \quad (3)$$

Let us consider a subset $QA(C^q, \mathcal{T}_{P2P}, A)$ of the query answer $QA(C^q, \mathcal{T}_{P2P})$. $QA(C^q, \mathcal{T}_{P2P}, A)$ consists of documents d which contain at least one complex concept C^d which is more specific than the complex query concept C^q and con-

[1]A preliminary version of this idea was proposed in [7]

tains atomic concept A.

$$QA(C^q,\mathcal{T}_{P2P},A)=\{d \in D_{P2P} \mid \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q \text{ and } \exists A^d \in C^d, \text{s.t. } A^d = A\} \quad (4)$$

If we denote the set of all atomic concepts A^d, which are equivalent to or more specific than concept A by $C(A)$, i.e.,

$$C(A) = \{A^d \mid \mathcal{T}_{P2P} \models A^d \sqsubseteq A\} \quad (5)$$

then, it can be shown that, given Equation 4, the query answer $QA(C^q,\mathcal{T}_{P2P})$ can be computed as follows

$$QA(C^q,\mathcal{T}_{P2P}) = \bigcup_{A \in C(A^*)} QA(C^q,\mathcal{T}_{P2P},A) \quad (6)$$

where A^* is an arbitrarily chosen atomic concept $A^q \in C^q$.

Given Equation 6, the query answer can be computed by using the algorithm described below. The algorithm takes complex query concept C^q as input and computes a query answer QA in six macro steps:

Step 1 Initialize the query answer: $QA = \emptyset$. Initialize auxiliary sets $\mathbf{C}_{ms} = \emptyset$ and $\mathbf{C}'_{ms} = \emptyset$.

Step 2 Select one atomic concept A from complex query concept C^q and add it to $\mathbf{C}_{ms}$.

Step 3 For every $A \in \mathbf{C}_{ms}$, repeat steps 4 and 5.

Step 4 Compute $QA(C^q, \mathcal{T}_{P2P},A)$ and add it to QA.

Step 5 Compute the set of all more specific atomic concepts B which are directly connected to the given atomic concept A in $\mathcal{T}_{P2P}$ and add them to $\mathbf{C}'_{ms}$.

Step 6 If $\mathbf{C}'_{ms} \neq \emptyset$, then assign $\mathbf{C}_{ms} = \mathbf{C}'_{ms}$, $\mathbf{C}'_{ms} = \emptyset$ and repeat step 3.

Note, that on step 2, atomic concept, A can be selected arbitrarily. In order to minimize the number of iterations, we choose A with the smallest number of more specific atomic concepts. The smaller the number, the fewer times we need to compute $QA(C^q, \mathcal{T}_{P2P},A)$ on step 3.

In the following, first, we show how documents are indexed in *P2P CSearch*, and then we show how the algorithm described above can be implemented.

In *P2P CSearch*, complex concepts are computed in the same way as in *CSearch* (see Section 3.2). The only difference is that now if an atomic concept is not found in the local background knowledge $\mathcal{T}$, then $\mathcal{T}_{P2P}$ is queried instead. *P2P CSearch* also uses the same document representation as *CSearch* (see Figure 3).

After document representations are computed, the indexing of documents is performed as follows. Every peer computes the set of atomic concepts, A which appear in the representations of the peer's documents. For every A, the peer computes the set of documents d which contain A. For every pair $\langle A, d\rangle$, the peer computes the set $S(d, A)$ of all the complex document concepts C^d in d, which contain A.

$$S(d, A) = \{C^d \in d \mid A \in C^d\} \quad (7)$$

For example, if d is document $D1$ in Figure 3 and A is equivalent to *dog-1*, then $S(d, A) = \{$*small-4*$\sqcap$*baby-3*$\sqcap$*dog-1*$\}$. For every A, the peer sends document summaries corresponding to A, i.e., pairs $\langle d, S(d, A)\rangle$, to the peer p_A responsible for A in community DHT. The peer p_A indexes these summaries using the local *CSearch*. Overall, every peer in the network is responsible for some words and atomic concepts. Peers maintain the following information of the words and concepts which they are responsible for:

Word senses	
canine	*canine-1*, *canine-2*
More specific concepts	
canine-2	*dog-1*, *wolf-1*
More general concepts	
canine-2	*carnivore-1*
CSearch index	
canine-2	$\langle D4, 1, [1]\rangle$
population-4	$\langle D4, 1, [1]\rangle$

Figure 4: Peer's information

1. For every word, the peer stores a set of atomic concepts (word senses) for this word.
2. For every atomic concept, the peers stores a set of immediate more specific and more general concepts.
3. Document summaries $\langle d, S(d, A)\rangle$ for all the atomic concepts A (for which the peer is responsible) are stored on the peer and indexed in the local *CSearch*.

An example of the information, which can be stored on the peer responsible for a single word *canine* and for a single atomic concept *canine-2*, is shown in Figure 4.

Now, let us see how different steps of the query-answer-algorithm for computing the query answer are implemented in *P2P CSearch*:

Step 1 A peer p_I initiates the query process for complex query concept C^q.

Step 2 p_I selects A in C^q with the smallest number of more specific atomic concepts. C^q is propagated to the peer p_A responsible for A. On peer p_A, QA is initialized to empty set and A is added to $\mathbf{C}_{ms}$.

Step 3 p_A selects an atomic concept B from $\mathbf{C}_{ms}$ and repeats steps 4 and 5.

Step 4 p_A submits C^q to the peer p_B responsible for B. p_B receives the query concept C^q and locally (by using *CSearch*) computes the set $QA(C^q,\mathcal{T}_{P2P},B)$. The results are sent back to p_A. Note that if B=A, then the query propagation is not needed. On receiving new results $QA(C^q,\mathcal{T}_{P2P},B)$, p_A merges them with QA.

Step 5 Moreover, p_B also computes the set of atomic concepts which are more specific than B by querying locally stored (direct) more specific concepts (e.g., see 'More specific concepts' in Figure 4). The results are also propagated to peer p_A where they are added to set $\mathbf{C}'_{ms}$. If B=A, then the set of more specific concepts are computed on p_A itself.

Step 6 If $\mathbf{C}'_{ms} \neq \emptyset$, then p_A assign $\mathbf{C}_{ms} = \mathbf{C}'_{ms}$, $\mathbf{C}'_{ms} = \emptyset$ and repeat step 3. Otherwise the results are sent to the initiator peer p_I.

Note, that, in order to optimize query propagation, peer p_A can pre-compute addresses of peers p_B which are responsible for more specific concepts, and use DHT to locate such peers only when pre-computed information is outdated.

An example showing how the query answer $QA(C^q,\mathcal{T}_{P2P},A)$ is computed is given in Figure 5. Peers, represented as small circles, are organized in the community DHT, represented as a circle. A query consisting of a single query concept

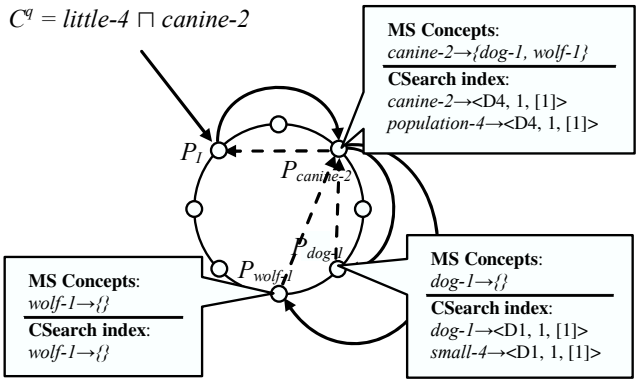

Figure 5: Query Answering

C^q = *little*-4 ⊓ *canine*-2 is submitted to peer P_I. Let us assume that atomic concept, *canine*-2 has smaller number of more specific atomic concepts than concept *little*-4. In this case, C^q is propagated to a peer $P_{canine\text{-}2}$, i.e., the peer responsible for atomic concept *canine*-2. The query propagation is shown as a firm line in Figure 5. $P_{canine\text{-}2}$ searches in a local *CSearch* index with C^q. No results are found in this case. $P_{canine\text{-}2}$ collects all the atomic concepts which are more specific than *canine*-2, i.e., atomic concepts *dog*-1 and $wolf$-1. Query concept C^q is propagated to peers $P_{dog\text{-}1}$ and $P_{wolf\text{-}1}$. $P_{wolf\text{-}1}$ finds no results while $P_{dog\text{-}1}$ finds document D_1. D_1 is an answer because it contains concept *small*-4⊓*baby*-3⊓*dog*-1 which is more specific than *little*-4⊓*canine*-2. D_1 is sent to P_A. The propagation of the results is shown as a dash line in Figure 5. Both peers $P_{dog\text{-}1}$ and $P_{wolf\text{-}1}$ have no more specific concepts than *dog*-1 and $wolf$-1, therefore C^q is not propagated to any other peers. P_A sends the final result, i.e. D_1, to peer P_I.

Note that the deeper we go in propagating the query, the less precise can be the answer. For instance, the user searching for *canine*-2 might be more interested in documents about concept *canine*-2 than in documents about concept *dog*-1, and she may not be interested at all in documents about very specific types of dogs (e.g., $affenpinscher$-1). In *P2P CSearch*, we allow a user to specify the maximum allowed distance in terms of numbers of links between atomic concepts in $\mathcal{T}_{P2P}$. Notice that this distance is similar to a standard time-to-live (TTL) [16]. Moreover, we allow the user to specify the maximum number of more specific concepts which can be used per each atomic concept in C^q.

In order to compute the query answer for a more complex query, e.g., query *baby*-1 AND *dog*-1 (in Figure 3), the intersection of the posting lists needs to be computed (see Equation 2). In this case, one concept is chosen from each of the complex concepts in the query (see step 2 in the algorithm). The lengths of posting lists and responsible peers for each of these concepts are obtained from the DHT using *getC* operations in parallel. The order in which these peers need to be traversed (*peer order*) is encoded in the query message (increasing order of length of posting lists). The query is then sent to the responsible peers in the corresponding order. Within each peer, the search is performed as mentioned in the P2P CSearch query-answer-algorithm described before. The resulting postings are intersected with results obtained from the previous peer in the *peer order*. The intermediate results obtained at each stage is sent to the next peer in the *peer order*. The final results are transferred to the peer who issued the query, by the last node in the *peer order*. Since our approach is not replacing syntactic search but extending it with semantics, for an efficient implementation of the intersection, we can reuse the optimization techniques developed in P2P syntactic search (see e.g. Section 3.1).

3.3.3 Ranking in a Single Community

The relevance of documents is computed by adopting the cosine similarity from the vector space model. Atomic concepts in a query are weighted by *tf-idf* weight measure. The only difference is that now frequency $tf'(A^q, d)$ of atomic concept A^q in document d is computed by taking into account *word-concept* and *concept-concept* similarities:

$$tf'(A^q,d) = \sum_{A^d \sqsubseteq A^q} \frac{1}{10^{d(A^q,A^d)}} \cdot \frac{f(A^q,w^q)}{maxF(w^q)} \cdot \frac{f(A^d,w^d)}{maxF(w^d)} \cdot tf(A^d,d) \quad (8)$$

where w^q (w^d) is a word in the query (in the document) from which the atomic concept A^q (A^d) is extracted, $d(A^q, A^d)$ is the distance between concepts A^q and A^d in the concept hierarchy from $\mathcal{T}_{WN}$, $f(A, w)$ is the value e.g. provided by WordNet which shows how frequently the specified word w is used to represent the meaning A (incremented by one in order to avoid zero values), $maxF(w)$ is the maximum $f(A, w)$ for word w, and $tf(A^d, d)$ is a frequency of A^d in d. Informally, we assign higher ranks for the following: (i) concepts A^d which are closer in the meaning to the concept A^q, (ii) concepts A^q which are more frequent for word w^q, and (iii) concepts A^d which are more frequent for word w^d.

4. GLOBAL SEARCH

In this section we explain how the global search in the network (i.e. Scheme 3 in Section 2.2) can be performed by using the search within communities explained in Section 3.

4.1 Search for Communities

Search for communities involves returning a list of communities, such that, there is a high chance that the documents expected by a peer issuing a query will be found inside these communities. Various sophisticated resource selection techniques used in meta-search and federated peer-to-peer search can be reused for this purpose [10]. The current implementation uses a simple popularity-based method to determine the potential communities relevant to a query.

In order to allow for an efficient search for communities, the communities are indexed and retrieved on the global layer similarly to how documents are indexed and retrieved inside each of the communities. The first difference is that UK is used instead of BK. The concepts from UK (and also words, if the corresponding concepts are not found in UK) are used for indexing the community in the global layer. In each community, when the length of the posting list for an atomic concept A with id AID exceeds a specified threshold, the peer responsible for the concept, sends an "index community with concept A_{UK}" message to the overlay through $putG(A_{UK}, AID)$, where A_{UK} is the most specific subsumer in UK for concept A. The document frequency $df(A, c)$ and the proportion $w(A, c)$ of documents in the community c which have concept A are also stored. The second difference is that when we compute the score of the community

($score(c)$), the term frequency $tf'(A,d)$ (in Equation 8) is replaced by $w(A,c)$. The third difference is that the $getC$ operations are replaced by $getG$ operations.

4.2 Search for Documents

When the relevant communities are discovered and ranked by their scores, a subset of the most relevant communities is automatically selected. Only those communities c_i are selected, such that, the score of the community $score(c_i)$ is bigger than than a predefined threshold t, i.e., $score(c_i) > t * max_i(score(c_i))$. The query is then propagated to all the selected communities. In every community, the query is propagated to the peer which is responsible for id AID. Search inside each community is performed as described in Section 3.3.2.

The top-k results from all the relevant community are then propagated back to the requesting peer and are merged into a single ranked result list. Scores, $score(d,c_i)$ for the documents d from each community are then normalized by the community score $score(c_i)$: $score(d) = score(c_i) * score(d,c_i)$.

5. EVALUATION

In order to evaluate our approach, we developed real implementations of the following three prototypes: P2P Syntactic Search (see Section 3.1) which is based on Lucene[2], P2P Concept Search[3] (see Section 3.3) built on top of CSearch [8], and two-layered P2P CSearch (see Section 4). Experiments with the real implementation could not be performed on thousands of nodes due to physical limitations. Hence, a custom simulator was developed by reusing the real implementation for both 1-layer and 2-layer approaches. For validation of the simulator, the real implementations of P2P Syntactic Search and P2P Concept Search were tested on a cluster of 47 heterogeneous nodes. The same queries were performed on the simulator and the results were found exactly same as that in a real network setting. The validation of the two layered approach in real network was not performed and is the subject of future work.

5.1 P2P Syntactic Search vs. P2P CSearch

In this section, we compare the performance of P2P Syntactic Search and P2P Concept Search (with WordNet used as a knowledge base).

As a data-set for this experiment, we used the TREC adhoc document collection[4] (disks 4 and 5 minus the Congressional Record documents) and the query set from TREC8 (topics 401-450). The title for each topic was used as a query. The data-set consists of 523,822 documents and 50 queries. In the evaluation we used the standard IR measures: the mean average precision (MAP) and precision at K (P@K), where K was set to 5, 10, and 15. The results of the evaluation are shown in Figure 6. The experiments show that, on TREC ad-hoc data set the results achieved by *P2P CSearch* are better than those achieved by P2P syntactic search. Note that the results achieved by distributed approaches are comparable with the results of the centralized versions of Lucene and CSearch (see [8]), i.e., quality is not lost much due to distribution.

TREC8 (401-450)		
	P2P Syn. Search	P2P CSearch
MAP	0.1648	0.1884(+14.3%)
P@5	0.4040	0.4440(+9.9%)
P@10	0.3860	0.4200(+8.8%)
P@15	0.3733	0.3907(+4.7%)

Figure 6: Evaluation results: Syntactic vs. Semantic

The average number of postings (document id and additional information related e.g. to score of the document) transferred per query (network overhead) was 49710.74 for syntactic search and 94696.44 for concept search. Thus the network overhead of concept search is 1.9 times that of syntactic search. But, the average length of intermediate posting lists transferred for concept search is only 37.48% as that of syntactic search even though the cumulative size is bigger. Thus by incorporating the optimizations proposed in section 3.3.2 (i.e. pre-compute addresses of peers responsible for more specific and getting respective postings in parallel), the response time for semantic search could be reduced compared to that of syntactic search. But the current basic prototype, doesn't include sophisticated optimizations and hence the search time comparisons are not made.

The number of postings transferred per query was taken as the measurement of network bandwidth consumption as they form the majority of network traffic for search (DHT lookup cost is comparatively negligible). Also, different optimizations like Bloom filters can be used to improve the transfer bandwidth for both syntactic and concept search.

5.2 P2P CSearch vs. Two-Layered P2P CSearch

In this section, we compare the performance of 1-layer and 2-layer P2P Concept Search approaches.

The data-set for this experiment was automatically generated by using data from *WordNet Domains* [3], *YAGO* ontology [22], and *Open Directory Project*[5] also known as *DMoz*. The DMoz was chosen for generating document collections because it is a collaborative effort of more than 80,000 editors who independently work in different domains and, therefore, the resulting document collections approximate the documents in the real communities well.

First, we created 18 background knowledge bases BK by using the following top-level WordNet Domains: *agriculture*, *architecture*, *biology*, *chemistry*, *computer science*, *earth*, *engineering*, *food*, *history*, *law*, *literature*, *mathematics*, *medicine*, *music*, *physics*, *religion*, *sport*, *transport*. Universal knowledge was created by using concepts and relations from WordNet which do not correspond to any domain (i.e., *factotum* in WordNet Domains). Knowledge from UK was added to every BK and then we enriched concepts in each BK with the more specific concepts (and corresponding relations) from YAGO ontology. Thus UK represents globally accepted knowledge while BK is specialized based on community interest.

Second, for every domain we selected a corresponding subtree in DMoz (e.g. *Top/Science/Agriculture* for *agriculture* domain and *Top/Society/History* for *history* domain). Doc-

[2] `http://lucene.apache.org/java/docs/index.html`

[3] To validate that there are no optimizations which can affect the fair comparison between syntactic and semantic approaches we compared the results of syntactic approach with the results of semantic approach when the background knowledge is empty. No significant difference was found.

[4] `http://trec.nist.gov/data/test_coll.html`

[5] `http://www.dmoz.org/`

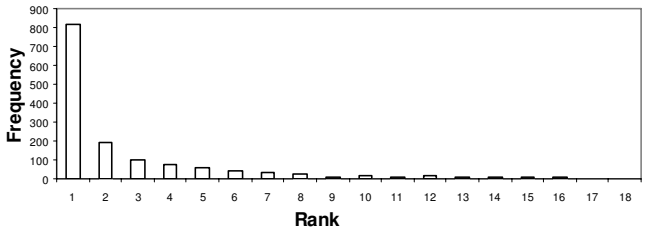

Figure 7: Ranking of communities

WordNet Domains + YAGO + DMoz				
	1-layer	2-layers (t=0.9)	2-layers (t=0.5)	2-layers (opt.)
MAP	0.1653	0.1376(-5.3%)	0.1761(+21.2%)	0.2175(+49.7%)
P@5	0.2778	0.2778(-0.0%)	0.3222(+16.0%)	0.3556(+28.0%)
P@10	0.2333	0.2167(-7.1%)	0.2556(+9.6%)	0.3111(+33.3%)
P@15	0.2037	0.1741(-14.5%)	0.2148(+5.4%)	0.2741(+34.6%)

Figure 8: Evaluation results: 1 layer vs. 2 layers

uments classified to categories in the corresponding domain subtree are used as the document collection for the domain. The concatenation of DMoz description, title, and the first page was used as the document. The document collection size varies from 1387 documents in *architecture* domain to 71661 documents in *computer science* domain. The total number of documents in the data-set is 318175.

Third, queries were generated by using labels of categories in the domain subtree, namely, queries were created by concatenation of categories's and its parent's labels adding "," in between. Queries which contain punctuation, special symbols, or boolean operators (e.g., '+', and '*'), queries which contain the words shorter than 3 letters, and queries created from nodes which contained less than 10 documents were eliminated. Two query sets were created. First query set contains 1432 queries (10 or less queries were randomly selected from each domain) and is used to evaluate the community selection process. Second query set contains 18 queries (1 query from each domain) and is used to compare 1-layer P2P CSearch versus 2-layer P2P CSearch.

Finally, in order to generate a set of relevance judgments for the second query set, we used a mapping from categories for which queries are created to the documents classified to these categories by DMoz editors. For every category, we collect all the documents classified in the subtree of this category. All such documents are considered to be relevant to the query.

In Figure 7, we show the results of the community selection algorithm. We plot the number of times (y-axis) the correct (the one with the most relevant documents) community was ranked at position n (x-axis). As we can see from Figure 7, in 57% of cases the correct community was ranked first. In 78% of cases it was ranked within first three. Note that this was obtained using our simple community selection algorithm and could be improved as mentioned in Section 4.1.

In Figure 8, we show the results of the evaluation of single and two layered approaches. For the two-layer approach we performed two experiments where a set of communities was automatically selected with thresholds $t = 0.9$ and $t = 0.5$ (see Scheme 3 in Section 2.2). Also we performed an experiment where only the correct communities were selected, i.e. we evaluated the performance in the optimal case, when the user manually select the set of communities to be queried (see Schemes 1 and 2 in Section 2.2) From Figure 8, we can see that if the threshold is too high (e.g. 0.9), then we have a high risk of retrieving the results which are worse than in the case of one layer approach. It is mainly because the correct communities can be missed by the community selection algorithm. If the threshold is relatively low (e.g. 0.5) it is likely that the correct community will be among the selected ones and the quality of the results is improved.

In the above experiments, for one-layered approach, the network cost was 85562.33 postings per query. For scheme 3, the network cost was 46404.50 postings per query for $t = 0.5$ and 32496.11 postings per query for $t = 0.9$. Thus, with $t = 0.5$, the network cost is **54.23%** as that of one-layered approach while it reduces to **37.98%** when $t = 0.9$. Thus, with our simple community search algorithm itself, the network cost was drastically reduced. With scheme 2, *ie* when the relevant community was manually selected, the network cost was 5811.33 postings per query which is only **6.79%** as that of single layered approach. This shows that with user intervention in community selection, the search quality as well as network cost reduction can be dramatically improved. Also, there is a lot of scope for better community search algorithms to improve the performance of Scheme 3 further.

6. RELATED WORK

A number of variants of DHT inverted index based peer-to-peer search approaches have been proposed in the literature (for an overview see [16]). Examples of how a full text retrieval can be efficiently implemented on top of structured P2P networks are described in [9, 23, 26, 19, 2, 11]. Bloom filters can be used to reduce the intersection bandwidth, but extra network overhead is involved due to the presence of false positives [25].

All of these approaches are based on syntactic matching of words and, therefore, the quality of results produced by these approaches can be negatively affected by the problems related to the ambiguity of natural language. *P2P CSearch* is based on semantic matching of concepts which allows it to deal with ambiguity of natural language. Note that, since our approach extends syntactic search and does not replace it, the optimization techniques which are used in P2P syntactic search can be easily adapted to *P2P CSearch*.

Some P2P search approaches use matching techniques which are based on the knowledge about term relatedness (and not only syntactic similarity of terms). For instance, statistical knowledge about term co-occurrence is used in [24]. Knowledge about synonyms and related terms is used in [12]. Differently from these approaches, *P2P CSearch* is based on semantic matching of complex concepts.Knowledge about concepts and concept relatedness is distributed among all the peers in the network. Note that, in principle, the knowledge about concept relatedness can be stored in DHT based RDF triple store (see e.g. [1]). But in Concept Search we use only the single type of relation (i.e., subsumption) and therefore the ad-hoc approach for storing distributed background knowledge (as discussed in Section 3.3.1) will require less resources, e.g., we don't need to index triples by predicates as in [1].

Semantic overlay networks [6] follow the scheme of classifying nodes and queries based on semantics. The nodes are clustered together by forming links among each other based on the classification. The queries are classified and are directed to the corresponding cluster. This approach increases the scalability of the search compared to a pure un-

structured approach. But, a globally accepted shared classification scheme is required which may not be feasible in a general case.

In general, formation of peer groups to improve the efficiency of peer-to-peer search has been exploited in various approaches. The node grouping could be based on term distributions[13], group hierarchy[18] or clustering[14]. [14] uses distributed clustering to form peer clusters and uses the term overlap in clusters to reduce publishing costs. Unlike these approaches, in our architecture, search and indexing in each node group(community) is done based on background knowledge of the community, made interoperable through a universal knowledge base.

Federated peer-to-peer search[10] addresses the problems of resource representation, ranking and selection, result merging, heterogeneity, dynamicity and uncooperativeness for searching federated text-based digital libraries. Our approach can be visualized as a community search counterpart of federated digital library search where each community is analogous to a digital library. Though the knowledge, community overlay level issues etc. differ, functions like resource selection, resource ranking and result merging in federated peer-to-peer search can be studied and reused to improve our approach.

7. CONCLUSIONS

We have proposed and implemented a two-layered architecture for peer-to-peer concept-based search on peer communities. Our experiments show that peer-to-peer concept-based search achieves better quality compared to traditional peer-to-peer keyword-based search. The proposed two-layered architecture improves the search quality further and reduces network bandwidth consumption compared to a single layered approach. We expect the approach to be a stepping stone in achieving scalable and pragmatic global-scale semantic search. However, there is still more issues to be explored in improving the quality and efficiency using the two-layered architecture. This include implementing various optimizations proposed in the paper, introducing novel search selection mechanisms and experimenting in a large scale deployment across universities which is the subject of future work.

8. REFERENCES

[1] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. Van Pelt. Gridvine: Building internet-scale semantic overlay networks. In *ISWC'04*, 2004.

[2] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. P2P content search: Give the web back to the people. In *IPTPS*, 2006.

[3] L. Bentivogli, P. Forner, B. Magnini, and E. Pianta. Revising wordnet domains hierarchy: Semantics, coverage, and balancing. In *COLING*, 2004.

[4] T. Berners-Lee and L. Kagal. The fractal nature of the semantic web. *AI Magazine*, 29(3):29, 2008.

[5] P. Buitelaar, B. Magnini, C. Strapparava, and P. Vossen. Domain-Specific WSD. *Word Sense Disambiguation: Algorithms and Applications*, 2006.

[6] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. *LNCS*, 3601:1, 2005.

[7] F. Giunchiglia, U. Kharkevich, and S. R. Noori. P2P concept search: Some preliminary results. In *Semantic Search 2009 Workshop*, 2009.

[8] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search. In *ESWC*, 2009.

[9] Jinyang Li, Boon Thau, Loo Joseph, M. Hellerstein, and M. Frans Kaashoek. On the feasibility of peer-to-peer web indexing and search. In *IPTPS*, 2003.

[10] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *ECIR*, 2005.

[11] T. Luu, G. Skobeltsyn, F. Klemm, M. Puh, I. Podnar Žarko, M. Rajman, and K. Aberer. AlvisP2P: scalable peer-to-peer text retrieval in a structured p2p network. In *Proc. of the VLDB Endow.*, 2008.

[12] Wenhui Ma, Wenbin Fang, Gang Wang, and Jing Liu. Concept index for document retrieval with peer-to-peer network. In *Proc. of the SNPD*, 2007.

[13] Linh Thai Nguyen, Wai Gen Yee, and Ophir Frieder. Adaptive distributed indexing for structured peer-to-peer networks. In *CIKM*, 2008.

[14] O. Papapetrou, W. Siberski, W. T Balke, and W. Nejdl. DHTs over peer clusters for distributed information retrieval. In *AINA*, 2007.

[15] M. V. Reddy, A. V. Srinivas, T. Gopinath, and D. Janakiram. Vishwa: A reconfigurable P2P middleware for grid computations. In *ICPP*, 2006.

[16] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50:3485–3521, 2006.

[17] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware*, 2001.

[18] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *IPTPS*, 2004.

[19] Gleb Skobeltsyn and Karl Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *Proc. of the P2PIR*, 2006.

[20] Vijay Srinivas and D Janakiram. Node capability aware replica management for peer-to-peer grids. *IEEE Transactions on SMC Part A: Systems and Humans*, 39:807–818, July 2009.

[21] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[22] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, 2007.

[23] C. Tang and S. Dwarkadas. Hybrid global-local indexing for effcient peer-to-peer information retrieval. In *NSDI*, 2004.

[24] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. *SIGCOMM*, 2003.

[25] Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *IEEE INFOCOM*, pages 2658–2669. IEEE Press, 2006.

[26] Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *P2P*, 2005.