eXO: Decentralized Autonomous Scalable Social Networking

Andreas Loupasakis
Computer Engineering & Informatics Dept.
University of Patras, Greece
loupasak@ceid.upatras.gr

Nikos Ntarmos
Computer Science Dept.
University of Ioannina, Greece
ntarmos@cs.uoi.gr

Peter Triantafillou
Computer Engineering & Informatics Dept.
University of Patras, Greece
peter@ceid.upatras.gr

ABSTRACT

Social networks have been receiving increasingly greater attention. As they stand now, users are required to upload their content to make it available to others. Typically, this involves releasing ownership and control. As a result, battles have begun regarding the ownership, exploitation, and control of content between users and social network owners. Further, given the rates of growth witnessed recently, questions about the scalability of the social network services are being raised. Therefore, the following questions naturally emerge: Is it possible to architect, design, and implement decentralized social networking services that ensure scalability and efficiency, while, respecting users' control of their content? Can this be achieved while content can be available to others for viewing and commenting, and permit key social networking activities like tagging? This paper presents *eXO*, a completely decentralized, scalable system that offers fundamental social networking services. We describe the architecture of *eXO* and its key components which encompass (i) techniques for content indexing, (ii) novel algorithms for ranked retrieval, (iii) appropriate similarity definitions that consist of a 'content' and a 'social' part, (iv) novel algorithms for efficient distributed content retrieval, (v) novel techniques that efficiently and scalably facilitate tagging and exploit tags to enrich query results, and (vi) novel scalable methods which permit the creation of personal networks, which allow for more "intimate" sharing and associations between users. We report on our evaluation which showcases its efficiency and scalability characteristics. *eXO* source code will be freely available to all, to use and test.

Categories and Subject Descriptors

H.3.4 [**Systems & Software**]: Distributed Systems, Query Processing

General Terms

Algorithms, Design, Performance

5th Biennial Conference on Innovative Data Systems Research (CIDR'11) January 9–12, 2011, Asilomar, California, USA

Keywords

Social networks, peer-to-peer, tagging, similarity, top-k

1. INTRODUCTION

Social networking services and entrepreneurship have been growing steadily and rapidly. User-generated content (mainly multimedia, such as images, videos, etc) is being produced at high rates. Interestingly, people show a great desire to share their content, view content shared by others, tag (comment on) it, search for content of interest and/or other users with specific profiles, and make 'friends' (become members of specialized groups sharing common interests). This is not all new. Since Napster appeared, a large number of content sharing applications have been implemented, providing a variety of options and actions to the user. Nowadays, users have moved on, creating large scale social network applications, such as Facebook, MySpace, YouTube, Flickr, etc. These applications run on centralized sites facilitating different kinds of social network sharing. The heart of these applications are users' interactions: content sharing and viewing experiences are being enriched, taking into account the profiles of those contributing the content and of those who have tagged it.

As they now stand, social network services force users to upload their content to a specific site in order to make it available to others. Typically, this involves releasing ownership/control of uploaded content. After a few years of tremendous growth in popularity and use, already some issues have been raised and battles have begun regarding the ownership and exploitation of content between users and social network site owners. Further, given the rates of growth witnessed recently, the scalability of the social network services becomes increasingly doubtful. Therefore, the following questions naturally emerge: Is it possible to architect, design, and implement decentralized social networking services that ensure scalability and efficiency? In addition, can users be allowed to retain full control of their content and efficiently make it available to others for viewing, sharing, and commenting? And, all this, while permitting other key social networking activities like tagging, so to enrich the users' querying experiences?

Our interest in future social network services is to ensure two key characteristics: highly decentralized social network functionality and with full user control when sharing. First, we wish to "go distributed". Already our community has produced a great wealth of knowledge for highly decentralized, efficient, and scalable content sharing – see for exam-

ple P2P network architectures (such as those built on top of DHTs). We wish to build upon these efforts and answer the questions whether the scalability of the underlying network can be leveraged to ensure scalable social network sharing and accessing and whether this can be done efficiently. Second, so far content providers may lose control over their data. We believe it is important for users to maintain control over their data: for instance, they should be able to control who accesses their data, when and how their content is replicated, if at all, and to which sites, etc.

These goals bear a number of important implications for system design which, in turn, create new research challenges. First, to respect autonomy, content must not be stored remotely. Further, to facilitate sharing, this content must be indexed appropriately so that users can search for it and locate it. This index must be distributed for scalability and efficiency reasons. Thus, no central organization/site exists, which is responsible for the content and related metadata, such as indices. Moreover, great care must be exercised with respect to how and what will be inserted into this distributed index, in order to avoid scalability and efficiency barriers, as has been shown in the realm of P2P search engines [14]. Further, defining appropriate similarity functions must account for high expense when computing relevant statistical metadata in distributed settings (despite valiant efforts [2]).

Second, autonomy implies for users the ability to name content on their own, resulting in a duplicity of names for the same content. Thus, any particular *search query* may be matched to a number of different sources. Our system must on the one hand efficiently maintain such metadata on data sources, and on the other, provide efficient algorithms for retrieving this content.

Third, the semantics of search queries are different. A query can request one such item, or all of them, or (most likely) a specific number of items, e.g., running top-k queries [10]. The ranking of query results must of course take into account the characteristics of content items and how close they are to a given query (a la traditional IR environments). However, in addition, ranking must take into account other social-network characteristics, such as the profile of the querying user, the profiles of the users who uploaded content items, and their similarity. Thus, new similarity functions are needed to facilitate such query-result ranking.

Fourth, running distributed top-k queries is a notoriously difficult problem. Despite recent advances (for instance [6, 15]), distributed top-k execution can introduce a rather large cost for the overall system resources (such as number of messages, network bandwidth, and local per-peer processing costs) and for the user (large query execution times). As we shall see, this cost depends on the similarity function. It is therefore imperative to produce similarity functions that can both exploit social network information and facilitate efficient distributed top-k query execution.

Finally, dealing with and exploiting user tags in a decentralized environment is not simple. On the one hand, it is important for the system to exploit these tags in order to identify relevant content [8, 13]. However, on the other, given that tagging is a very popular activity, this must be done in a way that does not introduce great overheads and scalability problems. Reconciling these goals is a key undertaking for any decentralized social network system.

Diaspora[1], an open-sourced personal web server implementing decentralized social networking services, is a first commercial crack at this; the $200k in donations received by the Diaspora authors and their recent (albeit early alpha-stage) source code release, testify to the plausibility of and widespread interest in such systems. Diaspora only implements the most basic of social networking primitives (that is, friends and friend groups), lacking any support for more advanced operations, such as (even simple) system-wide content/user queries – let alone such features as distributed top-k operations, tag clouds, personal social networks, etc.

With this work we present *eXO*, a novel decentralized system offering fundamental social networking services, consisting of:

- mechanisms for indexing user-generated content and related metadata definitions;
- appropriate similarity functions that combine social networking features with traditional query-to-content relevance;
- discovery and retrieval of related content via a combination of DHT-based indexing and of unstructured, query-relevant, social networks,
- efficient top-k algorithms for search-for-content and search-for-user queries;
- efficient algorithms for retrieving content items exploiting the architecture of the overlay network on the one hand, and the characteristics of content in social network sharing environments on the other;
- scalable social tagging mechanisms and methods which can exploit social tags to improve the quality of query results;
- methods for building social networks; and
- an implementation of the system and a performance evaluation substantiating the claims for scalability and efficiency.

To our knowledge, this is the first effort addressing comprehensively the design and architecture of decentralized social network services. The rest of this paper is organized as follows. First we present a brief overview of DHT basics. Section 2 discusses the data and query models, and section 3 the system architecture and key design decisions. Section 4 discusses query processing chores, presenting our approach for indexing, similarity definitions, ranking of query results, and optimized algorithms for retrieving content over the DHT. Section 5 discusses how we facilitate tagging and how we exploit it for search and retrieval. Section 6 shows how to build unstructured (personal) social networks that can be used to improve search results quality. Section 7 discusses our implementation and experimental results. Section 8 discusses related work and finally sections 9 and 10 conclude this work.

2. THE DATA AND QUERY MODEL

eXO is built upon a network, consisting of a possibly large number of nodes. Each node runs a routing protocol for a structured overlay DHT network (such as Pastry [9], Chord [21], Tapestry [24], CAN [18], etc.) Shareable content and nodes acquire IDs from the same ID space, using a public hash function, such as SHA-1, matching each of them to a specific position in the ID space. Structured overlay networks require nodes to maintain routing tables of size

[1] `http://joindiaspora.com/`

logarithmic to the size of the network (i.e., number of participating nodes). Further, they ensure that a message will be delivered to its destination in a logarithmic (again, relative to the size of the network) number of hops. For instance, in Pastry[9], the protocols guarantee $O(\log_{2^b} N)$ hops for the routing process, where N is the number of nodes in the network and typically $b = 4$. In Pastry, this routing process will also end up pointing to the node with the arithmetically closest ID to the destination ID (the input ID of routing function).

In *eXO* a user is associated with a unique network ID (*UID*), which is computed using a hash function (e.g. SHA-1) on a user-specified string, e.g. an e-mail address. Each user is connected to a specific node of the network; thus, a UID plays indirectly the role of a node identifier as well. Last, we use the term *content* to refer to every type of data that can be indexed, retrieved, and stored in the system (in *eXO* we are mostly interested in images, audio, and video content and secondarily in text).

Content & User Profiles. For indexing purposes, each content item is represented by a set of terms (keywords) that describe it. We call this set of terms the *content profile*. Similarly, each user is described by a set of terms defined by the users themselves, denoted the *user profile*. Each term is associated with an ID (term ID – *TID*). TIDs are also computed using a hash function on the term string. For example, a content profile term could be the file name of some multimedia file or words taken from the id3 tag of an mp3 file, while a user profile term may include information about user interests, *expertise* in some area, etc. A checksum of the actual content or user ID is computed to distinguish among the many different objects with the same profile. Content items may be replicated from the source to adjacent nodes in the ID space of the overlay in order to improve content availability. Content is indexed to appropriate network nodes, so it can be discovered. Note that when a content item is first shared, its owner decides on the content's profile - that is, the set of terms with which to index it. In any case, the owner of a content item holds complete power over what tags are attached to her profile/content items; however, subsequent tagging by other users (to be discussed shortly) may sway her selection of index tags and lead to a richer and more accurate indexing. Moreover, user profile terms play a central role during the ranking of query answers: for example, items uploaded by users who are either experts in a related area or whose profile matches a specific input user profile, are ranked higher than content items uploaded by other users.

Tags. *Tags* are terms contributed by users to describe a specific content item or user. Tags are very important in order to help achieve higher query result quality by exploiting the community's wisdom. As above, a tag term is associated with a TID. In *eXO* tags are stored at the *taggee*'s side associated with the tagged content and/or user profile. In addition, inverted lists for each tag term are maintained at the *tagger*'s node, associating each such term with a set of thus tagged resources. This information can be exploited appropriately to enhance the searching process as we will explain later. We refer to *social tags* as the tags whose tagger is a user other than the owner of the tagged item, as opposed to *self (owners') tags*.

Friends. Friendship is a major aspect in any social network service environment. A user can make friends and share more private data with them. To support this functionality two users can become *friends* by a typical handshake process, in which the first user requests a friendship with another user and the latter can accept or reject this request. The system stores *friend lists* (that is, list of friend UIDs) on the related nodes, updated every time a friend is added or deleted. The friend list structure is used to locate friends in the network, to allow access control of the data and to improve the lookup ranking process. The latter can be achieved by ranking the friends' content higher.

Queries. Queries in our model refer to content or user profiles indexed in the network. A query is a set of keywords and the query processing engine is meant to return the top-k most relevant items to the query, using a data structured called the "Catalogue" (to be discussed shortly). A query is a set of keywords which are used to obtain a number of data sources for the most similar to the keywords catalogue entries, supported through a similarity matching and a top-k ranking process. To be more formal, let $U = \{u_1, ..., u_m\}$, $C = \{c_1, ..., c_n\}$, $T = \{t_1, ..., t_o\}$ be the set of all user profiles, the set of all content profiles, and the set of all terms, respectively. For example, $u_i = \{t_i, ..., t_a\} \in U$ is a single user profile, and $c_i = \{t_j, ..., t_b\} \in C$ is a single content profile. A query may contain a content part and a user part and is given as $Q = Q_c + Q_u$, where $Q_c = \{t_1, ..., t_{q_1}\}$ is the content part and $Q_u = \{t_1, ..., t_{q_2}\}$ is the user profile part. In brief, queries may contain both or either of a content-related part and a user-related part, where the content (user) part may be missing when the querying party is specifically interested in finding user (respectively, content) profiles.

If we search for users, then a user query is formed. In this case $Q_u \neq \emptyset$, consisting of the set of terms which are used to route to the catalogues. Q_c is generally ignored in this case. User queries are used to list the k most similar user catalogue entries to the querying terms. The returned user catalogue entries contain user profiles and are also indices of the users' source nodes, so they can be used to navigate towards the user nodes in order to browse or tag their content.

On the other hand, content queries are used to find catalogue entries of shared content objects. It this case, $Q_c \neq \emptyset$ is used to route the process to the appropriate catalogue nodes. If, in addition, $Q_u \neq \emptyset$ then Q_u is used to boost the scoring of content contributed by similar users in the catalogues. The content profiles are encapsulated in content catalogue entries together with the content owner's source node; thus, in essence these catalogue entries are overlay "links" to access the shared content.

3. EXO ARCHITECTURE AND DESIGN

Here we discuss the basic components of the system and its design rationale.

Autonomy and privacy. The system is designed so that users can exercise full control over their content and their node's resources. In *eXO*, content shared by a user is kept only on the user's node (source node). For content availability reasons (e.g. for when the user leaves the network due to either a failure or a graceful disconnection) and at the user's discretion, the system can further maintain a number

CATALOGUE FOR "ACROPOLIS" @ node *hash(Acropolis)*		
UID	CONTENT PROFILE	USER PROFILE
A1FE458B...	{term1, ..., Acropolis, ..., termm}	{term2 , 'peter'. term r}
....	....	....

Figure 1: Example catalogue data

of content replicas. These replicas are stored only on nodes adjacent to the user node in the ID space, and their maintenance is performed automatically by the substrate network (e.g Pastry[9]).

The user can mark her content as *public* or *private* to define which users can access it. Public content items are indexed in the network and can be seen by anyone in the overlay network, while private content is not indexed and is thus non-visible to the public. Instead, it is made visible and can be accessed only by friends. Similarly, there exists both a *public* and a *private* user profile in the network; the public user profile is also indexed and may be replicated, while the private part is stored only on the source node. Note that owners can reject access requests made by other users, even for public content. The basis for this rejection can be any criteria the owner chooses to use (e.g., on the id of the querying user).

Node roles. The software running on each node takes on a number of tasks. First, it acts as a front end, serving user requests and dispatching them to the appropriate peer nodes in the system (*request resolver role*). Second, it provides a *network storage* interface for the content and profile replicas. Last, nodes may store indexing data structures for remote shareable content and user profiles in the system, in which case they are denoted *Catalogue Nodes*.

Catalogues. Each term ID is assigned to such a node (using the DHT's mapping primitive); that node then stores, for every TID it is responsible for, a data structure coined the *Catalogue*, mapping the TID to a set of related UIDs, along with (a portion of) the user profile, content profiles, and optionally other data (e.g. friend lists). Thus, every TID in the network has its own Catalogue data structure, keeping track of user nodes storing shareable content or user profiles containing the TID keyword.

A Catalogue has two parts. First, the User part is used to store the user profile and the UID of each user whose public profile contains the relevant term; the user catalogue is used to answer user-centered requests, such as "find the top-k similar users". Second, the Content part (see figure 1) holds the content profile terms and the user profile of the content owner, mapped to the content owner UID. This data structure is used to answer content related queries such as "find the top-k similar content items". As an enhancement for similarity matching, we also store the corresponding owner's user profile in each content catalogue entry. This allows to boost our scoring mechanism by using the comparison of the owners' profiles with the querying user profile.

Handling Tags. Social tags are stored only on the tagger and taggee nodes associated with the annotated object. We chose not to index tags in the network for that would introduce great overhead costs [11]. Indexing tags, on the other hand, would provide extra scoring capabilities, e.g. similarity boosted by tags. To reconcile this trade-off, we allow the benefits of social tagging for query resolution as follows: we store locally social tags (avoiding their distribution and related maintenance overheads) and we use a low cost mechanism based on query reformulation to enrich our search results with tag data (to be described shortly).

The public network and Personal Social Networks. The above rationale leads to a two-pronged approach. The *public network* consists of the DHT and the content and user profiles which are stored at the DHT nodes and have been made DHT-indexable and DHT-accessible. On this public network, users can issue queries specifying user profiles of interest and thus identifying other interesting users. Accessing those users' nodes, a querying user can also identify and retrieve tags with which the interesting user profiles have been annotated. Then, a querying user can expand her query and thus identify more interesting users. Once identified, the querying user can add them to her friends' lists, establishing and enriching her personal social network.

Thus, *eXO* consists of a public, structured network and a series of private/personal, unstructured networks. Owners have complete autonomy of what content to share, who to befriend, what tags to accept, what further functionalities to perform (e.g., transitively expanding friendship relationships) and exercise it at the per-user level, at the per-content item level, etc.

4. QUERY PROCESSING

This section describes the indexing and query execution processes, similarity definitions, the mechanism for ranking query results, and content retrieval optimizations.

Indexing and Catalogues. Indexing is performed as follows. First, the object's (user or content) profile is fetched. For each term of the profile, the TID is computed and is used as input to the overlay routing function. An indexing message is created containing the UID of the source node, the object's profile, plus, for content objects, the owner's user profile. The message is routed towards the destination (which is the overlay node responsible for TID). This destination node plays the role of the catalogue node for this term. Depending on the type of object (user or content profile), an appropriate catalogue entry is formed from the message data and stored in the catalogue. The same steps are taken *in parallel* for every term in the object's profile. For content objects or users which have very large profiles, indexing of all the terms would be very network-hungry. For this reason the indexing process may choose not to use all of the terms in the profile. Here, we must note that real social data, as shown in section 5 by our crawling efforts, usually contain user or content profiles of acceptable size.

Similarity and Local Ranking. Search query messages are delivered to every catalogue node corresponding to the query terms' TIDs. There, a *local* similarity matching process takes place to compare the catalogue entries with the query terms and then a *local* ranking of the catalogue entries is performed. This similarity function has a user and a content part; the former is matched against terms in content profiles while the latter against terms in user profiles. This allows us to search for content with specific properties, users

with specific properties, or a combination of these two (i.e. content with specific properties shared by users with specific properties). To compare sets of terms of user profiles and/or content profiles we use the vector space model. Specifically, we consider a multidimensional space, where every discrete term determines just one dimension. Vectors of term weights are used to map profiles to this multidimensional space, producing feature vectors. Vectors are compared by using the cosine similarity equation, with similar profiles having high cosine values.

A significant part of this scoring computation is the term weight formula. Firstly, let U, T, C be the set of all user profiles, the set of all terms and all content profiles, respectively. $u \in U$ is a single user profile, $c \in C$ is a single content profile and $t \in T$ is a single term. Obviously, $T = (\cup_{i=1}^{|U|} u_i) \bigcup (\cup_{j=1}^{|C|} c_j)$. In our approach weight w_{t_i} of term t_i, $0 < i \leq |T|$ is computed by the next general approach reflecting the relations among content, user profile, and term:

$$w_{t_i} = a_c w_{t_i}^c + a_u w_{t_i}^u$$

where $w_{t_i} \in [0, 1]$, $w_{t_i}^c$ ($w_{t_i}^u$) denotes the relevant weight of the content (respectively user) profiles for t_i, and a_c and a_u are coefficients which indicate the significance of each weight, with $a_c + a_u = 1$.

We adopt a simplified boolean weight formula to compute the weights; that is, $w_{t_i}^c, w_{t_i}^u \in \{0, 1\}$, where a value of 1 signifies that the term t_i exists in the user/content profile. Let $E = [w_{t_1} w_{t_2} ... w_{t_m}] = E_c + E_u$ be the vector of the catalogue entry's weights for the corresponding content profile $c_e = t_1, t_2, ..., t_{m_c}$ and user profile $u_e = t_1, t_2, ..., t_{m_u}$, where m denotes the number of unique terms in the catalogue entry ($|u_e \cup c_e|$ of this entry), E_c is the vector of content profile weights and E_u is the vector of user profile weights. Remember that content catalogue entries contain both a content profile part and a user profile part, while user catalogue entries contain only a user profile part (i.e., for user catalogue entries, it holds that $E_c = \emptyset$).

Consider the query vector $Q = [w_{t_1} w_{t_2} ... w_{t_l}] = a_c Q_c + a_u Q_u$ for the corresponding user profile $u_q = t_{1_u}, t_{2_u}, ..., t_{l_u}$ and content profile $c_q = t_{1_c}, t_{2_c}, ..., t_{l_c}$, where l is the number of unique query terms, and Q_c, Q_u are the vectors corresponding to E_c, E_u, respectively. The weights of the query vector are computed by the same formulas as the catalogue entries. Using the cosine similarity as scoring function we get:

$$\text{Sim}[Q, E] = \frac{Q \otimes E}{\|Q\| \|E\|} \tag{1}$$

The $Q \otimes E$ is the inner product of the two vectors and $\|Q\| \|E\|$ is the product of their norms. The denominator is used for normalization purposes, in order to use unitary vectors.

Using the previous weights and according to the query model, we discern two cases.

1. Queries on user profiles. In this case only the user profile part contributes to the score computation, thus $w_{t_i} = w_{t_i}^u, a_u = 1, a_c = 0$, and:

$$\text{Sim}[Q, E] = \frac{Q_u \otimes E_u}{\|Q_u\| \|E_u\|} = \frac{|u_q \bigcap u_e|}{\|Q_u\| \|E_u\|} \tag{2}$$

2. Queries on content profiles. In this case we allow the similarity between the sharing and the querying users' profiles to be taken into account, thus $w_{t_i} = \frac{1}{2} w_{t_i}^c + \frac{1}{2} w_{t_i}^u, a_c = 1/2, a_u = 1/2$, and:

$$\text{Sim}[Q, E] = \frac{Q_c \otimes E_c + Q_u \otimes E_u}{\|Q\| \|E\|} = \frac{|u_q \bigcap u_e| + |c_q \bigcap c_e|}{2 \cdot \|Q\| \|E\|} \tag{3}$$

A catalogue node receiving a search request, uses the above formulas to score each catalogue entry, creating a sorted score table. Note that the above can straightforwardly be enriched with more elaborate social metadata, such as friends and measures of friendship [19, 20] at least for "direct" (non-transitive) friends.

Global Ranking. After the similarity processes have ended in each of the catalogue nodes and a sorted list of catalogue entries has been created, the appropriate results must be returned to the querying node. The search mechanism considers a similarity threshold on the obtained results. This implies a top-k choice of local catalogue entries. As we will explain and prove in the next section, our scoring process has each catalogue node compute and return a score that is a *global score* for the object represented by the entry for a query. So it is enough to get only the first k results from each local sorted list of catalogue entries and send them to the query origin node in order to get the global top-k list. This phase is called *global ranking*. Every catalogue node returns the top-k catalogue entries from the local sorted list along with their corresponding score values to the query source node. The latter then merges all incoming lists and a new list of catalogue entries is formed, sorted by score, from which the global top-k result list emerges.

Note that whenever an item is found in more than one catalogue node lists, its score is *not* aggregated (summed). This is so, because each catalogue entry already reflects a global relevance score. Recalling our similarity definition, we see that at each catalogue entry, global information is taken into account to produce the local score; that is, local scoring accounts for all terms that are contained in the query and all terms/tags that are contained in the content profile and the querying user's and content owner's profiles. Hence, the local score is also a global score.

eXO Top-k Query Analysis. The top-k execution process outlined above is engineered to be very simple and lightweight. It requires only a single phase of communication between the querying node and the catalogue nodes for the query terms. Key role to this plays the way the similarity is defined: on the one hand we want it to take into account both user-user and query-content similarities, as it is necessary in a social network. On the other, each local score of an item to a query is, by construction, also a global score of that item for this query.

Despite valiant efforts, top-k algorithms in decentralized environments can be complex and very expensive in two respects: communication messaging and latency and (disk) IO latency. For instance, KLEE and TPUT [15, 6] require at least two communication phases between the querying node and the nodes holding the index lists and may have to go very deep into the index lists before yielding the results. The key difference in our approach is that we store in each catalogue the whole user and content profiles' info, and this provides the flexibility to compute global score values in every catalogue node. Further, each node only needs to

send the top k entries, yielding a very small bandwidth cost. More formally, we can prove the following.

CLAIM 1. *Consider an object, O, indexed using term set T_O. Given a query Q with query term set T_Q with $|T_Q \cap T_O| = n \geq 2$, the similarity score (for the catalogue entry in each of the n nodes responsible for these TIDs) to the query Q, is exactly the same.*

PROOF. Sketch: Remember that we store the profile of the owner along with every content profile and that each catalogue entry is uniquely defined by its owner's UID (along with a content checksum for content entries). Also remember that the content part of the query is matched against the content part of catalogue entries, while the user part is matched against the accompanying user part. Thus, each of these n nodes would be matching T_Q against the same T_O, hence for any given catalogue entry, the scoring function yields the same score across all n nodes. □

CLAIM 2. *The global list with the top-k most relevant entries stored in the catalogue nodes can be computed in one communication phase, by returning to the querying node the local top-k entries of each visited catalogue node.*

PROOF. Sketch: Assume that an entry which has not been returned to the querying node, belongs to the global top-k. By the definition of our similarity, it follows that this entry must belong in at least one of the local top-k lists returned to the querying node. Then each one of the local top-k lists must have included the specific entry in its reply to the querying node. From 1 we showed that in local lists the scores are global. Hence, it trivially follows that this entry does not belong to the global top k results. □

We employ a boolean weight to compute the cosine similarity. This choice was preferred over a more fine-grained tf-idf weight for two reasons. First, in a social network the majority of content which is being shared by the users is multimedia, whose metadata is usually a short set of descriptive terms without big differences in term frequencies and their body is just binary data. Content profiles created from these content items are low dimensional. This is in contrast to text documents which have analyzable bodies and heavier sets of terms describing their data.

Moreover, to compute a tf-idf weight, global statistics, such as the size of users and the size of content objects in the network, should be computed. In fully decentralized environments this is very complex and expensive and best be avoided. On the other hand, boolean weights as more coarse-grained, do not need global network information. The boolean weight suffers from not taking into account the term frequencies in profiles. This limits the range of score values, so can cause many ties in the scoring process. But the maintenance gain is very more important and we will present later alternative techniques to refine search results.

Content Retrieval Algorithms. After receiving the top-k results, the content retrieval algorithm uses the UID of each catalogue entry to route to the content owners nodes and initiate downloading. As aforementioned, it is unavoidable that different content will be associated with the same score with respect to a query. Think of different pictures of the Acropolis at night, tagged with the terms *Acropolis, Athens, night*. A query with these terms does not really care which of the different photos it receives and all photos with these self-tags will have identical scores to the query. *eXO* exploits this observation to expedite content retrievals, by implementing a new DHT primitive coined *retrieve-m-from-n*.

Specifically, assume a user has requested the top-10 results and *eXO* responded with, say, the top-50 results sharing the top-10 scores. The basic idea of the optimized retrieval algorithms is to, at the first overlay hop, hand over to the next forwarding node a list of the top-50 IDs of content items and their scores. Each node thereafter, looks locally to see to which of the 50 nodes it can route faster. For instance, in Pastry each node can inspect its routing table to see if one of 50 destinations is stored there and hence choose that destination to obtain the desired object. In essence, this optimization provides the flexibility to route to the closest 10 objects among the 50 possibilities.

5. SOCIAL TAG CLOUDS IN EXO

Remember that "social tags" (i.e., terms submitted by other users in order to characterize content or users) are stored on the same node as the original content. More specifically, the set of social tags on some content item is coined the *content tag cloud*. Social tags can also be associated with a specific user, and not just with content items, yielding a *user tag cloud*. The tags constituting a tag cloud can be ranked according to some measure and sorted according to this rank. At this stage, we view this as an orthogonal issue and we leave it to future work. However, for simplicity, within user tag clouds, we sort tags by the number of users having used the specific tag for the same item.

Tagging Process. Social tagging involves two user nodes: the user doing the tagging (*tagger*) and the user owning the tagged object (*taggee*). A tagger issues a tagging request with one or more tags and *eXO* transfers this message to the taggee's node. There, an association of the object and the set of tags is stored, together with the origin user profile of each tag. The association is formed as a content or user tag cloud, indicating the significance of each tag. In addition, on the tagger's side, an inverted list for each tag term is maintained. After a successful tagging operation, the inverted list is updated with the profile of the tagged object. Thus, a bidirectional relation is created between a tagger's and taggee's nodes.

Query Enrichment with Social Tagging. The ultimate goal is to improve query-result quality. Recall the *eXO* approach to indexing content and user profiles: only an owner's tags (a small number) are used to index each object. This is a pragmatic concern in order to avoid overburdening the catalogue nodes with continuous updates to index data, etc. However, it also introduces the potential of missing important items when responding to queries. Consider, for instance, an item with large multi-dimensional semantic content, which requires a fairly large number of terms in order to be adequately characterized, or, an item whose important 'dimensions' change with time. Since only a fraction of these terms may actually be used to index the item within the catalogue nodes, a query specified with these terms will not return the item. For example, an important article involving Obama's high school friends would typically not be

indexed with the keyword "Obama" and therefore a query with keywords "Obama", "youth" will not return the article.

During content (or user-profile) retrieval, *eXO* provides the opportunity to not only download the object, but also to retrieve the social tags (tag cloud) as we visit the content owner's node. Hence, the tag cloud for each retrieved object may be piggybacked onto the response. This way a new set of terms (i.e. those in the tag cloud), submitted by other users and reflecting other opinions, can be discovered by the querying user. Then, a new search query can be formulated, being enriched by tag cloud terms, with the user selecting which tag cloud terms to use based on the terms' relative rank. If the tag cloud is too big, a threshold is used, choosing the most dominant tags of the cloud before piggybacking the cloud onto the response.

Catalogue Updates. Note that the ranking of tags within a cloud is performed at the owner's node, reflecting the owner's perception of the importance of each cloud term for his object. When ranks of these terms exceed some user-perceived importance threshold, our design permits the owner to update the "owner tags"; that is, utilize high-ranked social tags, and index the object using these tags as well. This mechanism is employed in order to cope with time dynamics, especially for object's whose important "dimensions" change with time. That is, an owner can decide, based on the tagging activity on an object, that new terms can be used to index the object and make it visible and appropriate from now on.

6. PERSONAL SOCIAL NETWORKS

Up to this point, we have elaborated how the DHT-based catalogue served as the basis for discovering interesting items when responding to a query and for retrieving these items. Social networks, however, are ad hoc in their essence, and basing all functionality upon the DHT may seem too stringent and artificial.

Building Personal Social Networks. To this extent, *eXO* harnesses the extra information supplied by the user tagging activity to further advance the state of the art.

As mentioned earlier, a bidirectional relation between the tagger's and taggee's nodes is created with every tag; this also holds for tags on user profiles. These couplings exist in the form of cross-referenced UIDs stored locally on the tagging/tagged nodes and not as separate network connections or routing table entries. Further, note that users can search for users similar to their profiles, or for experts in specific domains, using relevant terms, and so on; then, through the tag clouds returned by such queries, the users can further discover new relevant tags and new related users, resulting in the emergence of and association with specific social groups (be it of social friends, professional acquaintances, experts in domains of interest, etc).

These virtual unstructured networks can then materialize through "connection requests": the local node sends a befriending request to the remote nodes; if such a request is accepted by the remote user, the two communicating nodes keep each other's UID in separate "acquaintance" lists. *eXO* users may then opt to share part of their information with only members of these social groups, or direct queries towards specific groups that are expected to return more and/or higher quality results.

Querying Personal Social Networks. Putting it all together, queries in *eXO* can be of three types: (i) queries directed to the DHT catalogue nodes only; (ii) queries disseminated through the personal social networks; and (iii) queries directed in parallel to both of the above mechanisms.

In the first case (see section 4), queries are directed to the DHT catalogue nodes using which the top-k items for the query are identified, based on catalogue information (owner tags). Additionally, querying users can retrieve any tag clouds associated with the items and reformulate the query using important social tags. This process can be repeated until the user is satisfied with the results.

In the second case, queries are directed to the querying user's appropriate social network(s). If no appropriate social network exists, the user can establish it first as outlined above and then issue the query. The search query visits each of the users that are neighbors in the social net. At each neighbor, the local tagger's data structure is searched locally for entries which, like catalogue entries, contain user/content profiles and UIDs, and they are returned to the querying user. It is likely that these inverted lists are small, although a similarity processing like the one we mentioned in the previous subsections, can be applied to return the most relevant to the query results. Thus, these results can constitute a complementary list of search results, which have been obtained by *eXO*'s similarity query engine, retrieval algorithms, and tagging metadata. Querying users can decide how deep into the query-relevant social network they should delve before they stop.

Finally, in the last case queries are directed in parallel to both the DHT catalogue nodes and to the relevant unstructured social network. Thus, a new query can proceed in parallel to the catalogue nodes and to the nodes making up the related user-group.

Link prediction. The system, currently, does not perform link prediction per se. Instead, it is user-centric. Each user can utilize the public network's infrastructure to identify other social links of interest by issuing specific user-profile queries. Retrieving these profiles and their tags, they can reissue more elaborate queries and thus establish/enrich their PSNs. Having done this, more elaborate algorithms can be employed to perform link prediction, for example by utilizing *gateway* nodes, belonging to more than one PSN, through which to route befriending requests. Alternatively, as PSN nodes share elaborate, private profiles of their friends, they can detect common friends, which are not themselves direct friends, and advise them to link-up. This is a straightforward activity facilitated and easily supported by *eXO*.

7. EXPERIMENTATION

We implemented *eXO* over FreePastry[2]. We ran our experiments on an 8-core, 30GB RAM system. Due to space reasons, we report on only type 1 queries.

Methodology. We used two real datasets from Flickr and Facebook. The Flickr dataset was formed from the Tagora Project[3], containing associations among users, resources (e.g.

[2] `http://freepastry.org/FreePastry/`
[3] `http://www.tagora-project.eu/data/`

photos), and tags. With this we created content profiles for shared items. We used a slice of around 49,832 content items corresponding to 1000 users and to 264,379 terms. Second, we systematically crawled Facebook and formed a set of user profiles from public user data. Public profiles contain a sample of users' friends (at most 8 in our data) and a collection of users' interests and tastes, given as hyperlinks and phrases. We extracted this information (36,261 user profiles) and placed it in XML files which fed our tests. Before each experiment starts, every network node was initialized and allowed to exchange necessary messages to construct its routing table. Then, the indexing phase begun. The indexing concerned both user and content data. When the indexing process was completed, the system was ready to execute the test scenarios. We then composed queries using real user and content profiles. The first kind of queries were formed by randomly selecting a set of terms from the indexed Flickr content items, whereas the second one by choosing one by one the terms from random user profiles indexed in the network. We varied the number of query terms, the value of k in each query, and the network size.

Performance Metrics. We measured the average number of messages (hops) on the network for each search or index request in relation to the network size. Moreover, the average number of bytes (bandwidth) for each query result list was computed. Bytes are counted for all the result list data returned from the catalogue nodes to the query source node. Last, we measured the node load, computed as the number of times each node is visited during query processing. The load was compared to the document frequency distribution of each indexed term to appropriately explain the results. "Document" frequency in our approach refers to the number of content or user profiles that contain a specific term. Please note that randomly selecting query terms from the set of all terms does *not* result in uniform-random loads, since different terms have substantially different document frequencies. Also note that the load balancing issues are largely orthogonal to this work: any off-the-shelf approach can be utilized to deal with extremely popular terms, for example, that could overburden an index node responsible for this term. Note that such issues are inherent into any design, be it DHT-based or not, since nodes with popular content will be hit more frequently than others.

Results. Figure 2(a), depicts the load distribution among nodes for the Flickr dataset, showing a balance in load, except for some relatively mild peaks. The peaks are justified if we take a look at the Flickr dataset's "document" frequency distribution graph in figure 2(b). The "document" frequency, for the Flickr dataset contains a number of terms which are dominant, explaining the peaks. Please note that, in any case, any load distribution strategy can be employed in cases where load balancing issues emerge. Figure 3(a) presents the load distribution in network nodes for the Facebook dataset, while 3(b) show the relevant "document" frequency. We observe a fairly balanced distribution of node hits. Also, the percentage of the total hits that any node receives, is very low. Thus, we expect no bottlenecks at catalogue nodes, despite the non-uniform distribution of the terms' "document" frequency.

The next experiment involves the indexing of 8,000 user profiles (one per node), and 160,000 queries, for various

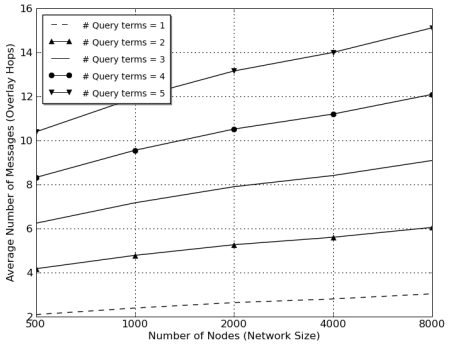


Figure 4: Number of messages versus network size

Table 1: Average Bytes Per Query Result

Top-K	# Query Terms	Bytes
Flickr Dataset		
10	2	2.2k
20	2	4.3k
50	2	10.8k
100	2	21.8k
Facebook Dataset		
10	1	3.7k
10	2	7.5k
10	3	11.2k
10	4	15.5k
10	5	19.1k

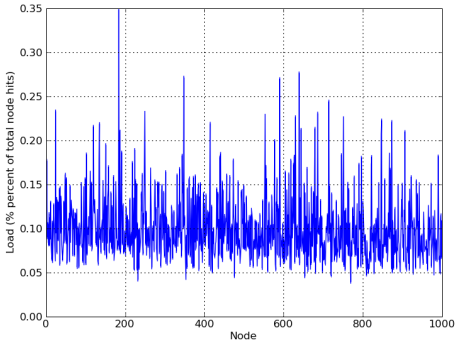
network sizes. We computed the average number of overlay messages to the destination catalogue nodes for various query sizes. Figure 4 shows results for queries with one to five different keywords. The results confirm the logarithmic relation between network size and number of messages for each search request. As mentioned earlier, a search request initiates a number of parallel routing requests, one request for every keyword in the query. So we expect to have multiples of the first curve in our graph depending on the number of query keywords, which we indeed observe.

Finally, table 1 presents the number of bytes communicated between the source and the catalogue nodes. For this experiment, we issued 20,000 queries per test and computed the average payload for those queries which returned exactly the maximum total number of result lists (a multiple of K). The results show the average size of communicated data per query to be very low.

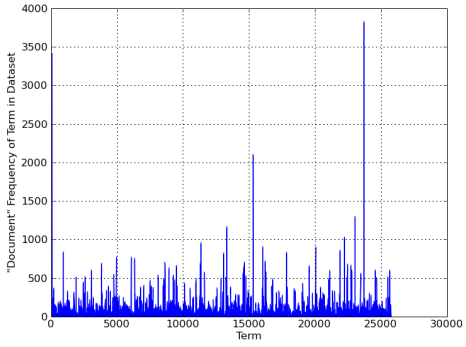
Overall, the results confirm that: (i) scalability is ensured with respect to a growing network, given (a) the relation between the number of messages and the network size, (b) the fairly well-balanced load among the network nodes, and (c) the small size of bandwidth consumption for all queries; (ii) the parallel nature of the search-query process ensures that the user-perceived latency will be low, as the results are computed in just one communication phase.

8. RELATED WORK

The backbone of existing approaches for top-k query processing is the classic paper of Fagin, et al.[10], which has been extended to distributed environments in works such as

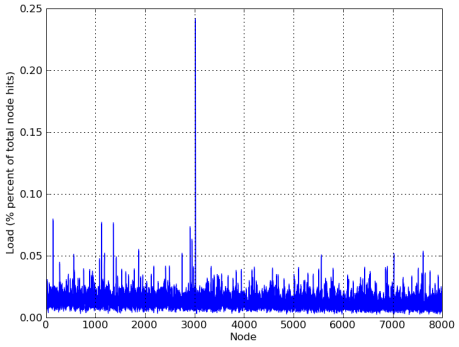


(a) Load Distribution

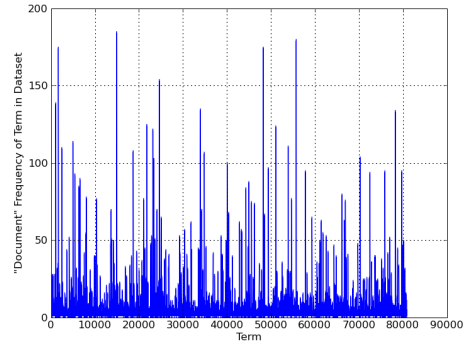


(b) Term "Document" Frequency Distribution

Figure 2: Flickr Dataset



(a) Load Distribution



(b) Term "Document" Frequency Distribution

Figure 3: Facebook dataset

KLEE[15] and TPUT[6]. However, all these approaches are quite expensive. KLEE and TPUT contribute an overhead of at least two phases of communication among the query source and destination nodes. In comparison, our approach requires only a single phase of communication.

Social web search has also been studied in [16], where web searching results are appropriately enhanced by the social links' data. Top-k processing in collaborative tagging sites has been studied in [23] where user networks are created (e.g. based on the number of same tags they used for tagged items, or on the overlap of their tagged items) and uses generalized versions of Fagin's algorithms to exploit social tags. An item is deemed relevant to a query issued by a user u based on how many users in u's network have tagged this item with a query term. [5] presents top-k algorithms for selecting the most relevant tags given a document. *eXO* provides mechanisms to support the building and utilization of such networks.

In [1] a gossip protocol, is employed so to associate users with similar tagging behavior and to develop enough state locally for top-k queries to execute swiftly. This is in contrast to our *eXO* design where a DHT is used for this purpose. Note that using a DHT and our top-k approach will permit faster retrievals and associations with other users, but lacks the flexibility afforded by gossiping over a completely unstructured network.

In [19] top-k processing employs similarity functions taking into account features such as friends and (friends of friends), and fine-grained relations of tags to documents and profiles and tag expansions are defined. In *eXO* we also provide similarity definitions that consist of query- and user-specific parts and can be enhanced by social tags. Although not allowing for transitive friendships yet, this can be implemented too by visiting direct friends and issuing queries for their friends etc. However, this will be much costlier in a decentralized environment than the centralized environment in [19]. Finally, the top-k algorithm in [19] considers additional information that is not easily available in decentralized settings, such as all the documents tagged by any user, all the tags used by any user, etc.

Related work in P2P searching includes the Minerva and Galanx [3, 22] DHT platforms. Like *eXO* Minerva maintains indices on the DHT. However, these indices index different information than our catalogues, query processing involves a multiphase top-k algorithm to return the appropriate results, no special content retrieval algorithms exist, and there is no support for social tagging, social query expansion, building personal networks, and relevant scoring functions. High costs also occur in Tagster[12]: based on a vector space model, it adds extra overhead for the computation and maintenance of global statistics, such as the document frequencies. Global statistics are costly to com-

pute and maintain despite efforts such as [2].

With the exception of [1], none of the above addresses the issues of decentralized scalable and efficient social networking. The gossiping protocols presented in [1] could be combined with *eXO* to incorporate the benefits of gossiping-style discovery and search. This is left for future work.

9. LIMITATIONS AND OPEN ISSUES

This work was a first crack at designing and implementing fully decentralized and widely distributed social networking services, bringing the power and burden of content ownership and sharing to the end users at the edge of the network. Our system provides efficient decentralized support for the cornerstones of social networks: friend lists, user/content tagging, and distributed processing of keyword/profile-aware queries. However, several issues remain open for future work.

Privacy and Availability. First and foremost, our work does not deal with content availability for privately/intimately shared content. One could devise and advocate some secret-sharing technique to allow private content to be replicated across the network so as to boost its availability without compromising its privacy. Any candidate scheme should be flexible enough to allow for frequent membership changes, while imposing a low network overhead; a formidable task in its own. Replication to "friends" (who already have access to private content) would be another attractive route; this solution would in turn call for a technique to transparently and remotely revoke access rights and replicated content from "x-friends". Moreover, even public data replication is an open issue in its own, if one further desires to perform it so as to enhance query performance in addition to data availability.

Meta-services. Second, social network users are accustomed to a specific "workflow" and meta-services, such as news feeds and games. The former could easily be implemented via an elementary publish/subscribe system of sorts; it would suffice to send notifications of new events to all nodes in the friend list of a given node. More elaborate schemes may also be devised (e.g., using dissemination trees or application-level multicast techniques). Similar solutions could also be applied to distributed games, albeit the requirements with regard to the latency of notification propagation will be much more stringent. All this may stress the network overlay, calling for new (possibly hierarchical) structures, such as those outlined in [17], or even completely novel overlays, specifically tailored for the needs of decentralized social networking.

Gossiping and PSNs. Third applying gossip- or broadcast-based protocols to the design of our personal social networks may yield some very interesting results. Directed broadcast has already been studied in DHTs[7]; further examination and application of such solutions in our environment remains an open issue. Similarly, gossiping algorithms may be employed to traverse the social graph (viewed as the union of all PSNs) and identify new social links, as is advocated in [4]. This is an interesting issue. One must decide on which metrics and features to use in order to establish further social ties between users. For example, identifying common neighbors, aggregating similarity weights across several traversed PSN links, deciding on the appropriate summary structures to use to represent such features (which are propagated through PSNs) are open, exciting issues.

Anonymity, Autonomy, and Privacy. These are conflicting goals. Specifically, owners may wish to know the identity of requesting users, before they grant a request to access local content. Striking a balance between the goals of autonomy and anonymity is an open issue. However, note that the decentralized nature of *eXO* does not create as serious of an anonymity problem as that of centralized systems. In the worst case, querying users may have to reveal their identities to the users whose content they wish to retrieve. However, there is no globally available record and log which records a users' accesses, behaviors, and social ties. Viewed differently, to uncover private data (e.g., the user's PSNs, the content items accessed by her, etc.) a large distributed set of meta-data must be accessed, which renders the task more difficult. Even more importantly, there is no single, central organization which maintains this information and can use it serendipitously or maliciously.

Distributed collaborative filtering. *eXO* can be leveraged to perform collaborative filtering. For instance, to recommend (perhaps using continuous background queries) new items of interest to users, based on what other users/friends are doing. In this direction an infrastructure that can perform the following tasks would be highly desirable:

- Monitor events of interest (such as when a user expresses an interest on an item, or highly tags/evaluates an item, etc).
- Collect such behavior information efficiently from the public network and PSNs.
- Detect when new recommendations can be effectively made to others (minimum support levels for similarity and number of event occurrences, etc).
- Make recommendations to those appropriate friends, based on friends profiles.
- Figuring out what is new to some friends and what they already know about.

Doing all this in a decentralized manner is a challenging issue.

Last, there are several information retrieval related issues that our work has not touched. For example, selection of the best tags to use for indexing or of the most promising tags in a tag cloud to use for query enrichment, stopword-like filtering of highly popular tags, and distributed novelty-based computations on the set of events delivered to recipients of news feeds, are just a few of them.

10. CONCLUSION

We presented *eXO*, a novel decentralized social network. *eXO* consists of (i) mechanisms for indexing content and related metadata, (ii) appropriate similarity functions with social networking and traditional query-to-content relevance, (iii) efficient top-k algorithms for search-for-user and search-for-content queries, (iv) efficient overlay algorithms for retrieving content, exploiting the architecture of the overlay network on the one hand, and the characteristics of content in social network sharing environments on the other, (v) scalable social tagging mechanisms and methods which

exploit social tags to improve the quality of query results, (vi) methods for building social networks, (vii) discovery and retrieval of related content via a combination of DHT-based indexing and of unstructured, query-relevant, social networks and (viii) an implementation of the system and a performance evaluation substantiating the claims for scalability and efficiency. With this work we have endeavored to showcase the appropriateness and feasibility of fully decentralized social networking services. The widespread adoption of Diaspora has testified that the users are interested in and probably ready for such a move. We therefore put forth this paradigm as our route of choice.

11. REFERENCES

[1] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, pages 87–98, 2010.

[2] M. Bender, S. Michel, P. Triantafillou, and G. Weikum. Global document frequency estimation in peer-to-peer web search. In *Proc. Intl. Workshop on the Web and Databases (WebDB)*, 2006.

[3] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative p2p search. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2005.

[4] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The Gossple anonymous social network. In *Proc. IFIP/ACM IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, 2010.

[5] A. Budura, S. Michel, P. Cudré-Mauroux, and K. Aberer. To tag or not to tag – harvesting adjacent metadata in large-scale tagging systems. In *Proc. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*, 2008.

[6] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proc. ACM Symposium on Principles on Distributed Computing (PODC)*, pages 206–215, 2004.

[7] M. Castro, M. Costa, and A. Rowstron. Should we build gnutella on a structured overlay? In *Proc. HotNets II*, 2003.

[8] P. A. Dmitriev, N. Eiron, M. Fontoura, and E. Shekita. Using annotations in enterprise search. In *Proc. Intl. Conf. on World Wide Web (WWW)*, pages 811–817, New York, NY, USA, 2006. ACM.

[9] P. Druschel and A. Rowstron. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, 2001.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

[11] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 2006.

[12] O. Görlitz, S. Sizov, and S. Staab. Tagster - tagging-based distributed content sharing. In *Proc. ESWC*, 2008.

[13] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search? In *Proc. Intl. Conf. on Web Search and Web Data Mining (WSDM)*, pages 195–206, New York, NY, USA, 2008. ACM.

[14] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proc. Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[15] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. In *Proc. Intl. Conf. on Very Large Databases (VLDB)*, pages 637–648, 2005.

[16] A. Mislove, K. P. Gummadi, and P. Druschel. Exploiting social networks for internet search. In *Proc. HotNets*, 2006.

[17] N. Ntarmos and P. Triantafillou. AESOP: Altruism-endowed self-organizing peers. In *Proc. Intl. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2004.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM Symposium on Communications Architectures and Protocols (SIGCOMM)*, 2001.

[19] R. Schenkel, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. X. Parreira, and G. Weikum. Efficient top-k querying over social-tagging networks. In *Proc. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 523–530, New York, NY, USA, 2008. ACM.

[20] R. Schenkel, T. Crecelius, M. Kacimi, T. Neumann, J. Xavier Parreira, M. Spaniol, and G. Weikum. Social wisdom for search and recommendation. *IEEE Data Engineering Bulletin*, 31(2):40–49, June 2008.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM Symposium on Communications Architectures and Protocols (SIGCOMM)*, pages 149–160, 2001.

[22] Y. Wang, L. Galanis, and D. J. de Witt. GALANX: An efficient peer-to-peer search engine system. `http://www.cs.wisc.edu/~yuanwang`.

[23] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *PVLDB*, 1(1):710–721, 2008.

[24] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U. of California at Berkeley, CSD, 2001.