Toward Personalized Peer-to-Peer Top-K Processing

Xiao Bai, Marin Bertier, Rachid Guerraoui, Anne-Marie Kermarrec

Toward Personalized Peer-to-Peer Top-K Processing

Xiao Bai Marin Bertier
INSA de Rennes, France

Rachid Guerraoui
EPFL, Switzerland

Anne-Marie Kermarrec
INRIA Rennes, France

ABSTRACT

We present the first personalized peer-to-peer top-k search protocol for a collaborative tagging system. Each peer maintains relevant personalized information about its tagging behavior as well as that of its social neighbors, and uses those to locally process its queries. Extensive experiments based on a real-world dataset crawled from del.icio.us shows that very little storage at each peer suffices to get almost the same results as a hypothetical centralized solution with infinite storage.

1. INTRODUCTION

A central task in information retrieval consists in processing queries in order to obtain the top-k items with the highest scores under a monotonic function. This is particularly appealing in collaborative tagging systems, also called *folksonomies*, such as *Flickr*, *del.icio.us* and *CiteULike*, which have become highly popular for publishing and searching contents. Yet this can turn into a nightmare because of the unstructured nature of tagging: there is usually no fixed ontology and users typically choose their tags in a free and possibly ambiguous manner.

One way to introduce some structure in such a scheme is to capture affinities between users with common tagging behaviors and then leverage these in the top-k processing. More relevant results for a given user could be achieved should the search be directed in a restricted network of users exhibiting similar tagging bahaviors. For example, a computer scientist and a Keanu Reeves fan might probably not be interested in the same results when Googling 'matrix'. User affinities can disambiguate these situations and alleviate the need for reformulating the same query through several steps with more refined keywords.

Very elegant centralized approaches have recently been proposed to capture the personalization through social scoring models that compute the user-centric correlation among different tags in order to improve information retrieval quality [1, 2]. In Amer-Yahia et al. [3], the score of an item only depends on how users sharing similar preferences have tagged it. The reference retrieval solution consists in maintaining one inverted list per (tag, user), but this is extremely space consuming. Alternative solutions to save storage space do exist but their processing time is increased.

We argue that scalability calls for fully decentralized solutions to process top-k queries. In fact, decentralized solutions have indeed been proposed. In Michel et al. [4], precomputed inverted lists are distributed across peers and partial information is transmitted in the network progressively to approximate the top-k results. In Cuenca-acuna et al. [5], a gossip scheme is used to implement distributed content search and ranking. None of these decentralized solutions is however 'personalized'.

This paper presents, to our knowledge, the first decentralized and personalized *top-k* processing scheme. In our scheme, each user maintains its own inverted list by periodically discovering its network.[1] Our approach alleviates the storage space problem while enabling highly efficient local query processing. We use a gossip-based network management protocol to identify users' personal networks in a peer-to-peer way; once these personal networks are established, users locally process their queries using a classical top-k algorithm, namely NRA (No Random Access). Interestingly, only a subset of (well selected) users suffices to provide the most relevant items while preserving their relative order for a given query. We explore various strategies to prevent overloading individual users by limiting the size of their personal networks. This results in little degradation in the quality of the top-k results.

We have implemented our decentralized and personalized top-k processing scheme in *PeerSim* [6] with a dataset crawled from del.icio.us. Interestingly, if all qualified users are kept as neighbors, after 50 cycles of gossips, we retrieve at least 8 relevant items out of 10 with a negligible storage overhead with respect to the idealized centralized solution of [3] (called Exact). Thanks to our optimization strategy which simply keeps the closest neighbors, more than 35% storage space is further economized for each user.

The rest of this paper is organized as follows: Section 2 establishes the social model of our system and provides a brief review of relevant top-k processing and network-aware search techniques. Section 3 describes our peer-to-peer per-

[1]Note that this is different from a social network in the traditional sense, as neighbors in our network are those with similar tagging behavior and might be and remain unknown.

sonalized top-k processing scheme and some optimization techniques. Section 4 presents our experimental setup and results. Section 5 concludes the paper.

2. SOCIAL NETWORK MODEL AND BACKGROUND

Social Network Model.

In a social network model, collaborative tagging sites are typically presented as information space $U \times I \times T$, where U denotes the set of users. Each user has a profile that expresses his endorsement of visited items by tagging them. The profile is described in the form of $i\{t_1, ..., t_n\}$, meaning that an item i is tagged by tags $t_1, ..., t_n$. I is the set of items that appear in the system and T is the set of all related tags. $Tagged(u, i, t)$ captures the fact that user u tags the item i with the tag t.

We model the social network as a directed graph where a node corresponds to a user and an edge presents the relationship between two users. We use $Link(u, v)$ to indicate the existence of a directed edge from user u to user v. For a user $u \in U$, $Network(u)$ is the set of u's neighbors, i.e., $Network(u) = \{v | Link(u, v)\}$.

There are many possibilities for establishing $Link(u, v)$ according to user preferences. We do not assume any particular semantics; as such we follow the criterion in [3] that $Link(u, v)$ exists if and only if v tags a sufficient number of items with the same tag as u, i.e.,

$$|\{i | \exists t, Tagged(u, i, t) \wedge Tagged(v, i, t)\}| > threshold,$$

where $threshold$ is a predefined number. The number of commonly tagged items is also used to measure the strength of $Link(u, v)$, denoted by $Strength_{Link(u,v)}$.

Query And Scoring Model.

We consider a query $Q = \{u, t_1, ..., t_n\}$, issued by a user u with a set of tags $t_1, ..., t_n$. Returned items for a query should be ranked according to its overall score. Our score is user-specific and network-aware. The score of an item i for user u and tag t_j is defined as the number of users in u's network who tags i with t_j, i.e., $Score_{t_j}(u, i) = | Network(u) \cap \{v \mid \exists t_j, Tagged(v, i, t_j)\} |$. The overall score of an item *i* for user u is the sum of all the scores related to the query, i.e., $Score(u, i) = \sum_{t_j \in Q} Score_{t_j}(u, i)$. We use the same scoring functions as in [3] for ease of comparison. (Note that alternative functions can be used in our social network model.)

Top-k Processing.

Given a query, a top-k processing algorithm (e.g., [7]) aims at retrieving the k most relevant items. The goal is typically to minimize the time it takes to come up with these items as well as the amount of storage needed to perform the actual computation.

Information is organized in per tag inverted lists. Each entry of the inverted list contains the identifier of an item and its score for that tag. Inverted lists are sorted in a descending order of scores. Below we review the well-known NRA algorithm, as this is considered particularly effective: it underlies our work as well as that of [3].

For a query of n tags, NRA scans the n inverted lists in parallel. In order to do so, NRA maintains a heap of candidate items. Each candidate has a score lower-bound and a score upper-bound, which are the overall scores it can attain based on the information available at the moment. The score lower-bound takes the most pessimistic assumption that if an item has not been seen in some lists, then it does not exist in them. Alternatively the score upper-bound takes the most optimistic assumption that its scores in those lists equal to the scores of last seen items in them. Once an item is seen, it is either added to or updated in the heap. The score upper-bounds of items already in the heap are also updated. Candidate items are sorted based on their score lower-bounds. For those with equal lower-bounds, the one with a larger upper-bound is ranked ahead. The processing stops when none of the items out of the top-k items has an upper-bound larger than the lower-bound of the kth item.

Network-Aware Search.

Amer-Yahia et al. [3] proposed several strategies to achieve efficient network-aware top-k processing. Again, for a network-aware search, the score of an item only depends on the querier's network. The most straightforward strategy, called *Exact*, is to build one inverted list for each (tag, user) pair. Query Q generated by a user u is processed on the (t_j, u) lists, where $t_j \in Q$, using a traditional top-k processing algorithm such as NRA. However, storing these lists is prohibitive space-wise for a single server. Therefore, they explore another strategy, *Global Upper-Bound*, that maintains user-independent inverted lists whose entries only contain the max scores over all users. The exact score of an item is computed at query time. With Global Upper-Bound, considerable storage space is saved but much more time is required for query processing. To strike a balance between the two extremes, users are then clustered and only score upper-bounds over all clusters are maintained in per (tag, cluster) pair inverted list.

3. PEER-TO-PEER SOLUTION

In our decentralized scheme, each user maintains a set of neighbors that form its personal network. A query is processed with the information available in the querier's network to get personalized top-k results. If its network fails to provide any satisfactory results, a default search mechanism will be activated. Here we concentrate on the personalized processing. In our setting, a key problem is how to build a personal network for each user in a peer-to-peer way as quickly as possible so as to guarantee the quality of top-k results.

3.1 Personal Network Construction

In our scheme, the personal network is discovered and maintained through a two-layer gossip protocol. The bottom-layer gossip protocol, typically known as a random peer sampling protocol (RPS) [8], is in charge of keeping the overlay connected. Basically it provides each peer with c random peers. On top of the peer sampling protocol, a top-layer protocol is in charge of tracking the similarity between users' profiles and discovering new neighbors. Once the top protocol has stabilized, it is still possible to discover new related peers through the peer sampling protocol.

We assume that each peer executes the same protocol in the same manner every T time units, referred to as a *cycle*. A gossip protocol, bottom and top, consists of two threads: an *active* thread initiating communication with other peers, and a *passive* thread waiting for incoming messages. A gossip protocol is fully characterized by three functions: *(i)* the peer selection (choice of gossip target); *(ii)* the data exchange (which data are exchanged in a gossip interaction) and, *(iii)* the data processing (which data are kept after the interaction). In this paper, the data exchanged are peers (IP addresses and profiles of peers).

At the beginning of each cycle, the peer selection selects a neighbor with the oldest *TimeStamp* as a gossip destination. Once a peer is picked, its TimeStamp is set to zero and other neighbors' TimeStamps increase by one. The variable TimeStamp ensures that all the neighbors have a comparable chance of participating in gossiping. The data exchange function selects *gossip-size* neighbors from the views of both layers and sends the profiles of selected peers to the gossip destinaton. When the gossip destination receives this information, its *passive* thread is activated. It also selects gossip-size neighbors and sends their profiles to the peer from who the message came. The data processing function selects the *network-size* closest peers according to some predefined metric. Two users are added to each other's view if they tag a minimum number of items in common with at least one same tag. Note that the top layer provides each peer with its *personal network*.

3.2 Inverted Lists and Query Processing

In the process of gossiping, the profiles of a user's neighbors are stored by the user. To enable efficient top-k processing, inverted lists for each (*tag*, *user*) pair are also computed and stored by the user itself. When a user generates a query, it first checks whether the inverted lists for the tags in the query already exist in its cache and whether they should be updated to reflect new neighbors. Once all the related lists are up-to-date, the user processes its query locally with NRA to get the top-k results.

The inverted lists are constructed lazily so that an inverted list is computed only when it is necessary for the query processing. There is no need to pre-compute the inverted lists for the tags that may never be queried in the Exact centralized case. Note that the computation of inverted lists are not more expensive than Exact when changes occur in the network. In contrast, gossip-based protocols enable to capture such dynamics.

3.3 Personal Network Optimization

We adapt the criteria in [3] for choosing neighbors in our setting. (Note that any metric that captures the affinity of users will work as well with our gossip-based personal network construction protocol). Since the common-item-based neighbor choosing criteria may cause a scalability problem when the network grows and users add more contents in their profiles, we propose several optimization strategies that reduce the size of user's network without degrading the top-k quality.

Instead of maintaining all the users that meet the criteria to be a neighbor, only n of them are kept in a user's personal network. The effect of n will be evaluated in the next section. These strategies can also be used independently of the criteria in [3] to form personal networks.

Random n random neighbors are chosen from the candidate users. The intuition is that a random sample is usually representative of the population from which it is drawn.

Biased Random Like Random, only n users are randomly selected as neighbors. However the probability that a user is kept as neighbor is proportional to the strength of the link between the two users. So users with stronger links have better chances of being chosen as neighbor.

Nearest As users are more likely to enjoy what is preferred by the users having similar preferences, and the similarity of user preferences is measured by the strength of the link between them, this strategy has confidence in users possessing strongest links with the gossip initiator and chooses them as neighbors.

Nearest With Enhanced Link Strength (Nearest-ELS) Similar to Nearest, users having similar preferences are also chosen here. In collaborative tagging sites, similarity of user preferences depends on the tagging behaviors exhibited in their profiles. Normally, the overlap of used tags in users' profiles implies their common interests on topics, while an overlap of tagged items reveals specific objects they prefer. As a tag can be used for several items and a same item can receive different tags from different users, it is more accurate to compare tagging behaviors by the number of (item, tag) pairs in common rather than the number of items tagged by both users with same tags. The intuition is that the more common tags are used for an item, the more similar the way on which the users understand and judge the world is. For this reason, we re-define the strength of $Link(u, v)$ as $\mid \{(i, t_j) \mid Tagged(u, i, t_j) \wedge Tagged(v, i, t_j)\} \mid$.
For each candidate neighbor v of user u, we re-compute $Strength_{Link(u,v)}$ and keep the n users having strongest links with u as neighbors.

4. EVALUATION

4.1 Experimental Setup

Data Set and Query Generation.

We have implemented our decentralized and personalized top-k processing scheme in *PeerSim*, an open source simulator for peer-to-peer protocols. All experiments presented here are executed on a cluster of servers including 10 Dell PowerEdge 1855 machines equipped with Bi-pro Intel Xeon processors with 3.40GHz CPU and 4GB memory.

The dataset used in our evaluation was crawled from del.icio.us. It contains 13,521 distinct users who participate in 31,833,700 tagging actions in the form of $Tagged(u, i, t)$. 4,741,631 distinct items and 620,340 distinct tags are concerned by these actions. We randomly picked 10,000 users from the dataset and built their profiles with the items and tags used by at least 10 distinct users. Note that this does not affect the top-k results and processing time because only the items ranked at the tail of the inverted lists are dropped. After removing uncommon tags, the interference of the noisy and often meaningless tags is also eliminated. The remaining dataset contained a total of 101,144 items, 31,899 tags and 9,536,635 tagging actions.

In our experiments, each user processed exactly one query. We randomly picked an item from a user's profile and composed the query with the tags used by the user to annotate this item. This is motivated by the reasonable observation that the tags in the query reflect the user's understandings of this item and their combination within the same query is meaningful.

Evaluation Metrics.

We were mainly interested in the implicit semantic relations exhibited by users' tagging behaviors. As in [3], there is a link between two users if they tag two items in common with at least one same tag. The personal relationships are formed as the network converges. If all users have their complete networks, the same top-k results would be obtained for a given query. We assume that there is neither arrival nor departure of user for ease of comparison with Exact and Global Upper-Bound in [3], which are considered ideal in terms of processing time and storage space respectively. Important questions related to our decentralized setting are then how fast the personal network of each user can be established and what is the influence on the top-k results.

A user begins building its personal network by first discovering the IP address of any user currently in the system with some bootstrap mechanisms. At any following time, each user has a number of neighbors thanks to gossiping. The ratio of this number to the user's network-size in a centralized setting implies how close its current personal network is to the target. We measure the speed of convergence with the average ratio of all users, i.e.,

$$Speed = \frac{1}{|U|} \sum_{u \in U} \frac{|Current\ Network(u)|}{|Network(u)\ in\ centralized\ setting|}.$$

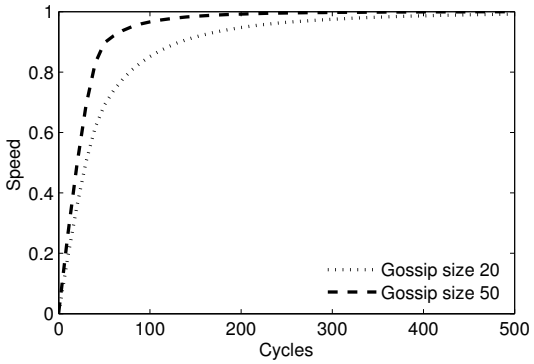


Figure 1: Convergence speed

$Speed$ attains 1 when all users find their personal networks in the centralized implementation.

Our goal is to show that an efficient network-aware top-k processing can be achieved in a peer-to-peer way, so we take the top-k results in [3] as a reference. Note that all algorithms proposed in [3] provide the same top-k items and only differs from each other in processing time and storage space. We then try to obtain as similar top-k items as possible for the same query with less time and space consumption in our setting.

We use the accepted metric, *recall*, in information retrieval to evaluate the quality of top-k results. The recall R_k is the proportion of the total number of relevant items that are retrieved in the top *k*:

$$R_k = \frac{Number\ of\ Retrieved\ Relevant\ Items}{Total\ Number\ of\ Relevant\ Items}$$

Recall quantifies the coverage of the result set and varies between 0 and 1.

The space overhead for each user is estimated by the number of entries in the inverted lists and the number of entries in the profiles of its neighbors. The query execution time for a query is quantified by the number of sequential accesses of related inverted lists. Both metrics are highly dependent on the query and the user, so we are more interested in the relative improvement compared to the centralized implementation of Exact and Global Upper-Bound for a given user and query.

4.2 Convergence of Personal Networks

We start with some observations on how the network converges and how fast our algorithm enables users to find their own networks, which are considered as the basis of our personalized top-k processing. The gossip-size is set to 20 and 50 respectively for both layers in two different experimental settings. We can see from Figure 1 that exchanging more information provides higher convergence speed. After a small number of cycles, users can get almost its whole personal network in a centralized setting.

We run top-10 processing in a centralized implementation of Exact and take the 10 returned items for each query as relevant items and compare our top-10 results with them.

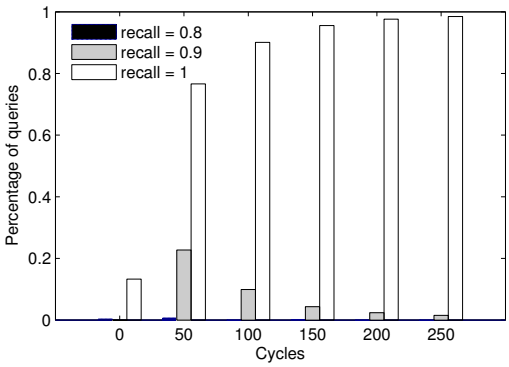


Figure 2: Recall evolution (gossip-size=50)

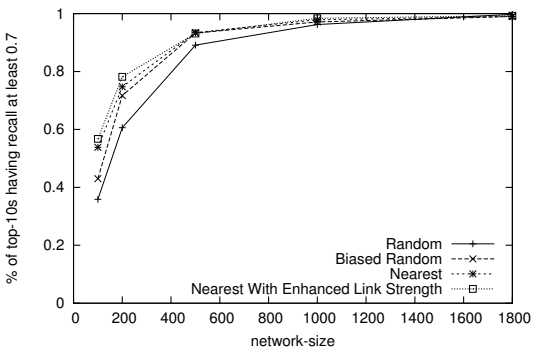


Figure 3: R_{10} for the four strategies with varying network-sizes

Table 1: Network-size in different systems

Number of users	Average network-size	Max network-size
1000	180	750
5000	897	4165
10000	1801	8350

The following results are from the setting of gossip-size set to 50. Figure 2 plots the evolution of R_{10} in the process of network converging.

At cycle 50, more than 77% of queries get exactly the same results as in the centralized implementation and the rest of the queries retrieve at least 8 relevant items. R_{10} continues improving as time passes and about 98.5% of queries obtain all their relevant items at cycle 250.

4.3 Comparison of Optimization Strategies

Constructing personal networks in a peer-to-peer way can provide almost the same personalized top-k results as the centralized implementation can do. However, when the number of users and the items tagged by each user increase, it is possible that users' local disks become fully occupied, because too many profiles of neighbors and inverted lists need to be stored. With our own dataset, average network-size and maximum network-size over all users are listed in Table 1. On average, each user should maintain about 18% of other users as neighbors to construct inverted lists for top-k processing, which is infeasible in large-scale networks. It is therefore important to choose neighbors in a more intelligent way.

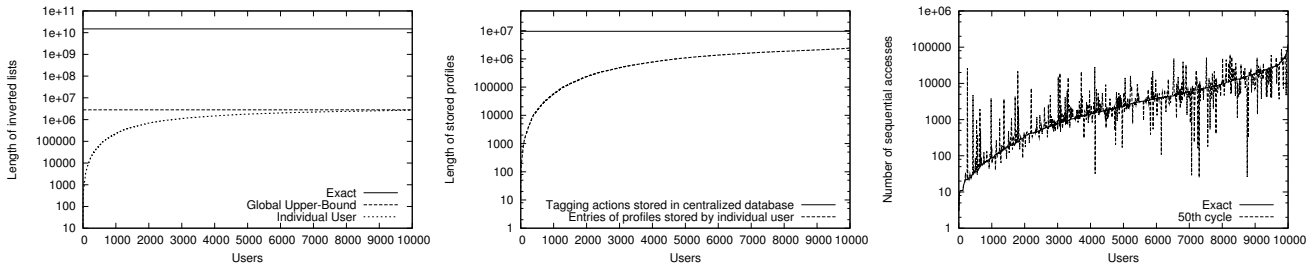
Figure 3 compares the performance of the four strategies we proposed in Section 3.3 The horizontal axis corresponds to the maximum number of neighbors a user can have. We consider Recall with no less than 0.7 as satisfactory top-10 results. The vertical axis shows how many queries receive such results. In this figure, only the queries sent by users having corresponding number of neighbors are taken into account, because others will get the same results as in the centralized implementation using the users' whole personal networks. It is seen that the strategy Nearest With Enhanced Link Strength outperforms the others in terms of top-10 quality. When the network-size is relatively small, the difference among these strategies is more pronounced, while the difference decreases when fewer limits are set on network-size. This difference also conveys the fact that users with similar tagging behaviors are more representative and contribute more to the final top-k results.

4.4 Space Overhead And Response Time

The maximum space overhead for each user is attained when its complete personal network is established. Figure 4 compares individual user's space overhead with that of Exact and Global Upper-Bounds. As mentioned earlier, the individual user's space overhead depends on the total length of its inverted lists Figure 4(a) and entries in all its neighbors' profiles Figure 4(b). Users are ranked in ascending order of their space overhead. As expected, no user needs to store as much information as a centralized database even for Global Upper-Bounds. Space is no longer a severe problem with our decentralized storage.

Figure 4(c) illustrates the response time at cycle 50. Shorter inverted lists do not necessarily mean less execution time because the lack of information may decrease the scores of certain items and as a result, more entries in the lists may need to be checked to get the final top-10. However, the penalty in time consumption is not too expensive. On average, only 3% more time is required to process these queries.

With the optimization strategies to limit network-size, fewer profiles and shorter inverted lists are stored by each user. If we use the Nearest-ELS strategy to choose neighbors and fix the network-size to 500, there is no difference for about 28% of the users whose network-size is 500 or smaller. For the others, on average storage space for 34.5% of the entries in inverted lists and 54.2% of the entries in neighbors' profiles are saved. As the inverted lists are changed, the number of sequential accesses to get the top-10 items is no longer the same as before. About 28% of queries need the same time to retrieve the results while 35.9% consume less time and 36.1% require more time. On average, there is no great difference as before in terms of processing time.



(a) Length of inverted lists (b) Length of stored user profiles (c) Number of sequential accesses per query

Figure 4: Space overhead and processing time at cycle 50 (log scale)

4.5 Scalability

As shown in Table 1, the personal network-size grows linearly as the number of users increases using the criteria in the centralized implementation to choose neighbors. With our optimization, we find that the necessary number of neighbors to obtain top-10 results of same quality remains stable even when the number of users continues increasing (see Figure 5). Clearly, the larger the network, the smaller the ratio of necessary network-size over number of users. Our optimization scales well, and similar phenomena can also be found in other optimization strategies. This suggests that there is no need to maintain a large network to obtain good personalized top-k results. Well selected neighbors conserve the relative order of relevant items to a query even though their overall scores decrease.

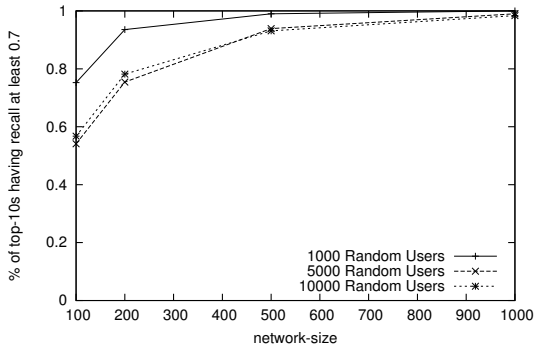


Figure 5: Recall of Nearest-ELS in systems of different size

5. CONCLUSION

We presented in this paper the first peer-to-peer approach to personalized top-k processing. We described the design and implementation of our approach and investigated its performance. The experimental results are encouraging and we believe that decentralization is the right way to provide personalized top-k processing in a scalable manner.

We are exploring several complementary research directions, including system dynamics, in terms of churn as well as frequent changes in tagging behaviors. We are also considering various techniques to improve privacy as well as reducing redundant information within a cluster.

6. REFERENCES

[1] R. Schenkel, T. Crecelius, M. Kacimi, T. Neumann, S. Michel, J.X. Parreira, and G. Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR '08*.

[2] S. Xu, S. Bao, B. Fei, Z. Su, and Y. Yu. Exploring folksonomy for personalized search. In *SIGIR '08*.

[3] S. Amer-Yahia, M. Benedikt, Laks V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. In *VLDB'08*.

[4] S. Michel, P. Triantafillou, and G. Weikum. Klee: a framework for distributed top-k query algorithms. In *VLDB '05*.

[5] F.M. Cuenca-acuna, C. Peery, R.P. Martin, and T.D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC'03*.

[6] M. Jelasity, A. Montresor, G.P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[7] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31:2002, 2002.

[8] M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.