Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks

Wolfgang Nejdl, Wolf Siberski, Uwe Thaden[1] and Wolf-Tilo Balke[2]

[1] L3S and University of Hannover, Hannover
{nejdl,siberski,thaden}@l3s.de
[2] EECS, University of California, Berkeley
balke@eecs.berkeley.edu

Abstract. Increasing the number of peers in a peer-to-peer network usually increases the number of answers to a given query as well. While having more answers is nice in principle, users are not interested in arbitrarily large and unordered answer sets, but rather in a small set of "best" answers. Inspired by the success of ranking algorithms in Web search engine and top-k query evaluation algorithms in databases, we propose a decentralized top-k query evaluation algorithm for peer-to-peer networks which makes use of local rankings, rank merging and optimized routing based on peer ranks, and minimizes both answer set size and network traffic among peers. As our algorithm is based on dynamically collected query statistics only, no continuous index update processes are necessary, allowing it to scale easily to large numbers of peers.

Keywords top-k retrieval, peer-to-peer query processing, ranking

1 Introduction

Meaningful querying for information, whether on the Web or in information systems and databases, often retrieves answers together with an indication of how well the results match the query. Various kinds of metadata available through the Semantic Web offer additional semantic information which may be integrated into the retrieval process. However, this generally comes at the price of large result set sizes that are often unmanageable for the individual user. Since users are usually only interested in a few *most relevant* answers to their query, the goal in top-k retrieval techniques is to return manageable result sets consisting of these most relevant answers.

This paper is concerned with querying peer-to-peer networks, which in recent years turned out to be a good basis for distributed information storage and exchange infrastructures. We will focus on *schema-based* peer-to-peer networks like Edutella [1], which is a peer-to-peer infrastructure for storing and retrieving RDF metadata in a distributed environment.

Let us consider a sample scenario, where the need for restricting the number of answers to a given query becomes apparent. Suppose that a student, John, prepares for an exam in logic programming and looks for appropriate exercises. We can assume that the educational ontology he uses allows him to specify his request for exercises, and his topic ontology ACM-CCS[3] lets him specify 'Logic Programming' as topic. As this is a fairly common area, we'll expect him to get a large list of matches more or less matching the query (e.g. some exercises specifically on logic programming, some exercises that contain logic programming questions among others and a number of exercises discussing just a few aspects of the topic). With current peer-to-peer infrastructures (including Edutella) there is no way for John to retrieve a list where all exercises are ordered by relevance, and only the best ones are presented in the first step, to be expanded whenever needed. John might also try to be more restrictive in subsequent queries until a sufficiently small result set is retrieved, but without knowing exactly which data are available this is a quite tedious process.

The situation gets even worse if we allow approximate keyword search. Suppose John looks specifically for Prolog programming exercises, but the corresponding entry is not part of the used topic ontology. A good solution would be to allow a combined schema-based and keyword search. John could then add the keyword 'Prolog' to his query. Again, keyword searches which return unordered lists of matching documents are not particularly useful. Similar to internet search engines, the success of any keyword search depends on its ability to identify a limited set of the most relevant documents.

Ranking scores each resource that matches a query using a certain set of criteria and then returns it as part of a ranked list. Additionally, we need to restrict the number of results, to make it easier for the user to use the results, and to minimize traffic in a peer-to-peer environment. John then gets a manageable number of results which includes those answers that are the most relevant for his query. This approach is referred to as top-k retrieval in database query processing, where only the k best matching resources are returned to the user.

In this paper we present a distributed top-k algorithm for peer-to-peer infrastructures. Our algorithm retrieves the k most relevant results in a peer-to-peer network without having to rely on any centralized or global knowledge and without need for a complete distributed index. Furthermore, our top-k algorithm does not only deliver more relevant results, it also allows us to optimize query distribution and routing. Based on statistics gathered at super-peers, queries are distributed only to the most promising peers. Compared to current approaches like PlanetP [2], because we do not need to maintain a complete distributed index, we avoid continuous updates whenever peers join or leave the network. Instead we show how information can be gathered dynamically based on the queries posed to the network, enabling advanced routing and top-k answers based on previous query statistics.

2 Peer-to-Peer Infrastructures and Ranking for Top-k Routing

2.1 Schema-Based Peer-to-Peer Networks

Schema based querying. Let us first start with the necessary background on peer-to-peer infrastructures for our algorithm. In previous papers, we have described the RDF-based peer-to-peer infrastructure Edutella [1, 3] which is an example of a more advanced approach to peer-to-peer networks called schema-based peer-to-peer networks. Schema-

[3] ACM Computing Classification System, http://www.acm.org/class/1998/

based peer-to-peer networks have a number of advantages compared to simpler peer-to-peer networks such as Napster or Gnutella. Instead of prescribing one global schema to describe content, they support arbitrary metadata schemas and ontologies, which is crucial for the Semantic Web, and thus have been investigated heavily during the last years [4–7]. These systems allow complex and extensible descriptions of resources, and provide more complex query capabilities than simple keyword-based search. Edutella uses RDF and RDFS for resource and schema description. The distributed nature of RDF is a perfect match to the distributed nature of peer-to-peer networks, and the flexibility and extensibility of RDFS allows us to combine arbitrary schema elements for resource description as well as for query formulation.

Semantic Routing. If we have additional semantic information about the data available in the network, we can use this information to optimize query routing. [8] uses this approach for an unstructured peer-to-peer network and evaluates different kinds of routing indices used to forward requests in the right direction. Other peer-to-peer approaches which support routing based on semantic characteristics are P-Grid [9, 4] and Piazza [5].

Edutella uses a super-peer topology, where super-peers form the network backbone and take care of routing queries through the network [3]. Only a small percentage of nodes are super-peers, but these are assumed to be highly available nodes with high computing capacity. In the Edutella network they are arranged in the HyperCuP topology [10]. The HyperCuP algorithm is capable of organizing peers into a recursive graph structure from the family of Cayley graphs, out of which the hypercube is the most well-known topology. This topology allows for $\log_2 N$ path length and $\log_2 N$ number of neighbors, where N is the total number of nodes in the network (i.e. the number of super-peers in our case). The algorithm works as follows: All edges are tagged with their dimension in the hypercube. A node invoking a request sends the message to all its neighbors, tagging it with the edge label on which the message was sent. Nodes receiving the message forward it only via edges tagged with higher edge labels (see [10] for details).

The super-peers employ routing indices which explicitly take schema information into account. Queries are analyzed regarding the schema elements used, and only peers which actively use these are considered when distributing the query.

2.2 Ranking Results

The second ingredient for our algorithm is ranking. Ranking allows us to reduce the overall number of answers in the result set and also to return close matches to avoid empty result sets in case no exact matches are found. Answers returned are ordered using a score value computed for each resource. We will use this score to limit the number of answers and return only the k best matching results. Let us describe two particularly useful ranking-methods, which we will then use in our algorithm presented in 3.2.

Topic-distances in Taxonomies. In the context of the Semantic Web, the use of ontologies and taxonomies to classify resources is very common. Given such a taxonomy, resources are associated with topics/concepts in the hierarchy using appropriate metadata annotations, and we can use methods as discussed in [11] to measure how similar and thus how relevant a resource is with respect to a query.

TFxIDF. TFxIDF stands for Term Frequency and Inverse Document Frequency and is a content-based ranking method. It calculates the relevance of a document, based on how often a search term appears in a document (term frequency TF), and how often the term exists in the whole document collection (inverse document frequency IDF). The more search terms are found in a document, the more important the document is, taking into account how often the search term is found in the collection, i.e. weighting rare terms in documents higher. A detailed introduction can be found in [12]. For a term t_i from a set of keywords and a document d_j from a document collection T_r TFxIDF is defined as

$$TFxIDF(t_i,\ d_j) = \underbrace{n(t_i,\ d_j)}_{TF} \cdot \underbrace{\log \frac{|T_r|}{n(t_i)}}_{IDF} \tag{1}$$

where $n(t_i,\ d_j)$ denotes the number of occurrences of the term t_i in the document d_j and $n(t_i)$ is the number of documents that contain the term t_i.

3 Top-k Query Answering and Routing

3.1 Challenges

Top-k ranking in peer-to-peer networks has to address four challenges:

Mismatch in scoring techniques and input data used by the different peers can have a strong impact on getting the correct overall top-scored objects. Since we want to minimize network traffic, but nevertheless integrate the top-scored objects from all different peers (and super-peers) within each super-peer, each super-peer has to decide how to score answers to a given query. In this paper we will assume that every peer throughout the network uses the same methods to score documents with respect to a query, though input data to compute these scores may be different.

Using only distributed knowledge and thus different input data to score answers complicates top-k retrieval, because many scoring measures that take global characteristics into account simply cannot be evaluated correctly with limited local knowledge. For our TFxIDF measure we can calculate term frequency locally for each document, but inverted document frequency depends on all documents in the peer network and thus can only be determined globally. Joining and leaving peers influences the calculation further. When we calculate IDF based on the local knowledge of each peer only, monotonicity of local rankings will not be preserved at the super-peer (who has more complete input to calculate IDF) and thus the overall top score list at the super-peer may miss relevant results. In the following we therefore will distinguish between two different kinds of measures: those that can be evaluated locally, i.e. that rely on the characteristics/content of the resources in a peer's local collection only (possibly in connection with some network-wide constants), and those that incorporate collection wide information, that depends on the resources in the (global) collection of all peers.

Minimizing network data transfer means that we should strive to only exchange the minimal amount of information necessary between peers and super-peers. A good example for limiting network data transfer is semantic-based routing. Queries and results are only routed through those (super-)peers that are associated with resources relevant to the query. This is also relevant for merging result sets in super-peers, where we want to minimize incoming data yet still produce a complete top-k answer list.

No continuous index updates. In peer-to-peer networks peers join and leave the network at unpredictable intervals. Top-k retrieval and routing has to take this into account without requiring continuous index updates limiting scalability of the algorithm. As a tradeoff we are willing to accept a certain degree of non-optimality in our top-k results as long as changes in the network are reflected "fast enough" in our results. Obviously, volatility of the peers cannot be arbitrarily high, as no kind of statistics would be meaningful then.

Taking this considerations into account, we will describe a top-k answering and routing algorithm, which is based on local rankings at each peer, aggregated during routing of answers for a given query at the super-peers involved in the query answering process. Each peer computes local rankings for a given query, results are merged and ranked again at the super-peers and routed back to the query originator. On this way back, each involved super-peer again merges results from local peers and from neighboring super-peers and forwards only the best results, until the aggregated top k results reach the peer that issued the corresponding query. While results are routed through the super-peers, we maintain statistics which peers / super-peers returned the best results. This information is subsequently used to directly route queries that were answered before mainly to those peers able to provide top answers. Additionally, a small percentage of queries will additionally be forwarded randomly to enable lazy update of these indices to adapt to changes in the peer-to-peer network. The more volatile the network is, the higher the percentage routed to random peers has to be in order to adapt to changing data allocations.

The following sections describe the different parts of the algorithm, starting with local ranking, then covering the merging of the results at the super-peers, and then finally showing how we can improve routing based on query / answer statistics.

3.2 Ranking at single peers

When a super-peer receives a query, it asks its peers to rank their resources locally. The results are top-k ranked variable bindings from each peer, which are sent back to the super-peer, along with score information for merging at the super-peer. To simplify the presentation we describe the algorithm taking the view of a user who wants to retrieve

URIs of the best matching RDF-resources. Thus, we will only refer to the resource instead of complete variable bindings.

In the context of this paper, we restrict ourselves to queries which allow conjunctive triple matching with constraints. Current Semantic Web query languages like RDQL [13] allow only 'hard' constraints, i.e. only resources for which all constraints are fulfilled match the query. On the other hand, for SQL additional language constructs for specification of 'weak' constraints, weighting of constraints and specification of the max number of results have already been defined in previous work [14].

Here we do not propose a language extension, but use a formal notation for our queries instead. A query Q is a tuple $Q = (Atoms_Q,\ k)$ where $Atoms_Q = \{(q, w_q)\}$ define the constraints of the query together with a weight w_q which is used to specify how important the different atoms are for the query-result-rankings. Constraints have the form $q = (prop_q, op_q, c_q)$ where prop is an RDF property , op is an operator and c is a constant RDF literal or resource.

Using this notation our scenario query from section 1 (find all 'lomedu:Exercise's with a subject-classification in the ACM CCS most similar to 'acmccs:LogicProgramming' and where the content of the property 'dc:description' is most relevant with respect to the keyword 'prolog') looks as follows:

$Q = (\{(q_1,\ 0),\ (q_2,\ 0.7),\ (q_3,\ 0.3)\}\ ,\ 12)$ with
$q_1 = (rdf : type,\ =,\ lomedu\!: Exercise)$
$q_2 = (dc : subject,\ \approx,\ acmccs\!: LogicProgramming)$
$q_3 = (dc : description,\ \sqsupseteq,\ 'prolog')$.

The operators that can be used in queries are distinguished into 'hard' and 'soft' operators: $OP_{hard} = \{=,\ <,\ >\}$ and $OP_{soft} = \{\approx,\ \sqsupseteq\}$: $OP = OP_{hard} \cup OP_{weak}$. In our example the hard operator $=$ specifies that the results must be of the type lomedu:Exercise. Our soft operators are a similarity operator ('$\approx$') and a keyword search operator('$\sqsupseteq$') ('$\approx$' measures similarities between topics in a taxonomy and '$\sqsupseteq$' triggers a measurement based on keyword search in text).

In general our approach can build upon any kind of similarity measures. In this paper we focus on two methods as examples: a measurement for distances in taxonomies and TFxIDF for plain text searches. In contrast to the taxonomy distance calculation, TFxIDF also needs non-local information.

Li et al. have developed and compared several measures for topic similarity in taxonomies [11]. We chose the following measure which yielded the best results in their study. The similarity of two topics in a taxonomy is defined as

$$\mathrm{sim}_p(r,\ prop,\ c) = \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & : \text{ if } \mathrm{value}(r,\ prop) \neq\ c \\ 1 & : \text{ otherwise} \end{cases} \tag{2}$$

where l is the shortest path between the topics in the taxonomy tree and h is the depth level of the direct common subsumer. α and β are parameters to optimize the similarity measurement (best setting is usually $\alpha = 0.2$ and $\beta = 0.6$). We might sometimes get different results on different peers due to different property values for the same resource. This can be caused either by annotation errors or if the property value is user-dependent, e.g. user ratings.

Centralized IR system use a global index containing term and inverse document frequencies. In a distributed context this approach is not feasible. PlanetP solves this problem by maintaining a replicated network-wide index which is updated using gossiping within the network [2]. In contrast, our approach does not build a complete index in advance at all, but rather retrieves the necessary non-local information as soon as it is needed for the evaluation of a query. For TFxIDF, this is the inverse document frequency, and idf denotes the non-local value. In section 3.3 we describe how this value is calculated and distributed to the peers. Corresponding to equation 1 we define tfidf for a peer p and a resource r as

$$\mathrm{tfidf}_p(r,\ prop,\ c) = \mathrm{tf}_p(r,\ prop,\ c) \cdot \mathrm{idf}(prop,\ c) \tag{3}$$

where $\mathrm{tf}_p(r,\ prop,\ c)$ is the term frequency (= number of occurrences) of constant c in property $prop$ of resource r and $\mathrm{idf}(prop, c)$ is the network-wide inverse occurrence frequency of c in property $prop$.

We use the index '$_p$' to specify that a function or set is related to the context of peer p. In this section we restrict ourselves to simple peers; in 3.3 we will also take super-peers into account.

Depending on the type of property and constant used in the atom the appropriate measure is selected.

$$\mathrm{score}_p(q,\ r) = \begin{cases} \mathrm{tfidf}_p(r,\ prop_q,\ c_q) & : \quad op_q = \sqsupseteq \\ \mathrm{sim}_p(r,\ prop_q,\ c_q) & : \quad op_q = \approx \wedge (c_q \text{ is a topic}) \end{cases} \tag{4}$$

Using the weights from the atoms we can calculate the overall score of a resource with respect to a complex query.

$$\mathrm{score}_p(Q,\ r) = \begin{cases} 0 & : \exists q\ \mathrm{score}(q,\ r) = 0 \wedge op_q \in OP_{hard} \\ \sum\limits_{q \in Atoms_Q} w_q \cdot \mathrm{score}_p(q,\ r) & : \text{otherwise} \end{cases} \tag{5}$$

3.3 Query Distribution and Result Merging

In order to use the peer scores from single peers to determine the top k resources within the network, peers do not only return the scores, but will need to provide some additional information that will be used at the super-peer to do a ranking over the top k results from the peers.

Let us first extend the single-peer measures discussed in the last section to take super-peers into account. P denotes the set of all peers and SP the set of all super-peers. For the set of all peers connected to a super-peer p we use the notation P_p and the notation $SP_p(Q)$ for the set of super-peers a query has to be forwarded to from super-peer p[4].

[4] We could introduce this set formally as well, but give only an informal sketch here, because the HyperCuP topology is not our main focus in this paper. In HyperCuP, each super-peer can be represented as a vector with binary entries, which describe the location regarding the dimension in the graph. A super-peer will only forward queries to other super-peers that are one dimension higher. To know which super-peers will send results for a query to the current super-peer we use a difference-function on the vectors. This gives us a list of vectors representing the super-peers that will return results.

Score aggregation. Aggregation of taxonomy scores is straightforward, because this measure is independent of the originating peer's context. We use the maximum score of a resource as consolidated score[5]:

$$\mathrm{sim}_p(r,\ prop,\ c) = \begin{cases} \mathrm{Max}_{o \in P_p \cup SP_p(Q)}(\mathrm{sim}_o(r,\ prop,\ c)) : p \in SP \\ \text{same as for simple peer} \quad : p \in P \end{cases} \tag{6}$$

Aggregating the TFxIDFs from the peers means we have to compute an overall TFx-IDF measurement for all results delivered from the peers. We calculate the aggregation of the occurrences of a constant in a text property (=TF) together with the number of all resources at the peers related to the number of all matching resources at the peers (=IDF). To cope with different TF values for one resource in different peers we will always use the maximum value. The extended TFxIDF functions are defined as

$$\begin{aligned} \mathrm{df}(prop,\ c) &= \sum_{o \in P} \mathrm{df}_o(prop,\ c) \\ R(prop) &= R_p(prop) = \bigcup_{o \in P} R_o(prop) \\ \mathrm{idf}(prop,\ c) &= \log \frac{|R(prop)|}{\mathrm{df}(prop,\ c)} \end{aligned} \tag{7}$$

where $R_o(prop)$ is the set of all resources of peer o having the property $prop$ and $\mathrm{df}_o(prop,\ c) = |\{r \mid r \in R_o \wedge \mathrm{tf}_o(r,\ prop,\ c) > 0\}|$ which is the same weighting as used in equation 1.

As mentioned in the previous section, we do not build an index over idf values in advance, but calculate the necessary values on demand. Already calculated values are cached in an index, so repeated evaluations of the same query can reuse them. If a query is not yet in this index, we have to distribute it twice. In the first round the peers deliver only their local document count and document frequency. This information is merged according to equation 7, and added to the super-peer index. In the second round, the necessary idf values are added to the query message, so that the aggregated information becomes available to all peers and super-peers. The other super-peers also add these values to their indices. The peers can calculate resource scores now, and return them as well as their current local document count and document frequency. That way an idf value is updated during each evaluation of a query which contains the corresponding query atom. No additional messages are necessary to maintain the index.

Investigating distributed information retrieval techniques over document collections in the Web, Viles and French [15] propose to use statistical averages that are updated only once in a while. They show that the effectiveness of retrieval is usually only slightly affected by this simplification, because changes in the collections usually tend to compensate each other on average and major changes generally need some time to develop. Thus relying on slightly outdated information based on a previous query evaluation will only lead to deterioration in extremely volatile P2P networks[6].

[5] In this and all following equations o and p always represent peers, r and s resources.

[6] To improve the precision further, we can re-evaluate the query if the difference of an idf value stored in the index and the value updated from the responses exceeds a certain threshold.

Query driven update of indices is possible because queries are not posed randomly but usually follow a Zipf distribution, where few queries make up the majority of requests. Zipf distributions are ubiquitous in content networks, the Internet and other collections. The distribution was first discovered by the Harvard linguistic professor G. K. Zipf, and has become one of the most empirically validated laws in the domain of linguistic quantities. If we count the number of times each word appears in a text (called frequency) and assign each word a rank based on its frequency (i. e. rank=1 is the word that appears the most), we can see that the product frequency · rank for each word is roughly equal to a constant. In a more general context, this corresponds to the observation that the frequency of occurrence of an event as a function of the rank is a power-law function.

Recent research has discovered this distribution as a typical distribution of information on the Internet [16–19]. In P2P networks typical consumers are interested in only subsets of all available content and content categories [20]. Documents are also distributed following Zipf's law, i.e. many consumers are interested in resources which are held by a few providers.

This makes it safe to assume that also the occurrence frequency of query atoms rsp. of the property/constant pairs they contain will also follow a Zipf distribution., and therefore, for most queries posed to the network, the necessary idf values will already be in the index, and we only seldom have to distribute a query twice.

$$\mathrm{tf}_p(r,\ prop,\ c) = \begin{cases} \mathrm{Max}_{o \in P_p \cup SP_p(Q)}(\mathrm{tf}_o(r,\ prop,\ c)) : p \in SP \\ \text{same as for simple peer} \qquad : p \in P \end{cases} \tag{8}$$

These definitions as well as the following are recursive; the calculation terminates at the super-peers which do not need to forward the query to other super-peers.

Merging Having defined the scores for peers and super-peers we can now use equations 4 and 5 to calculate an aggregated query score for each resource at the super-peer.

Simply collecting all top k resources from each peer and merging these sets would cause more network traffic than necessary. A simple top-2 examples illustrates this: In order to get the overall top scored resource in the super-peer we only need the maximum-scored resource of all its local peers and super-peers along the spanning tree starting at the query originator. Having chosen the maximum resource from any of the peers, we can select the top-2 of this set and exclude all peers which didn't get their top-1 resource placed in the selection. All the other peers still offer their top-scored resources. So for determining the overall second best resource for our merged top 2 result set, we only additionally need to ask the two best peers that already delivered our top-scored resources. For higher numbers of query results this process can be repeated inductively until all top k resources are delivered, as we show below.

Since the best objects are determined one by one, the merging super-peer can immediately deliver each result resource to the super-peer directly up the super-peer backbone, enabling it in turn to also return its merged results at the earliest point in time. This successive query result delivery behavior not only optimizes bandwidth use, but also helps to improve the psychologically felt response time for the user by offering correct result objects for consideration already at an early stage.

To formalize this approach let us first define some sets in each peer for bookkeeping:

For a resource $r \in M$ the set $BetterThan(Q, r, M)$ contains all resources $s \in M$ with a better score than r with respect to query Q:

$$BetterThan_p(Q, r, M) = \{s \in M \mid score_p(Q, s) > score_p(Q, r)\} \tag{9}$$

$Top_p(i, Q, M)$ is the subset of the i best scoring resources in a set M:

$$Top_p(i, Q, M) = \{r \in M \mid \quad |BetterThan_p(Q, r, M)| < i\} \tag{10}$$

For a peer p we define the resource(s)[7] with the i-th most score as

$$\begin{aligned} ResAt_p(1, Q) &= Top_p(1, Q, R_p) \\ ResAt_p(i, Q) &= Top_p(i, Q, R_p) \setminus Top_p(i-1, Q, R_p) \end{aligned} \tag{11}$$

where R_p is the set of all resources at peer p. For super-peers, we define $ResAt$ at the end of this section (16).

We define the following sets for a super-peer sp doing the merge: $ConsideredPeers_{sp}(Q, i)$ is the set of all peers, which have to be asked (are considered) for resources in the i-th iteration to guarantee a correct result set. $TopResCandidates_{sp}(Q, i)$ is an intermediate set of resources which could be in the top k set for the i-th iteration, i.e. the current best scored resources from all contributing peers, where peers that already contributed to the merged result delivered their respective next best resources. $TopRes_{sp}(Q, i)$ is a set of cardinality k of top resources after i iterations. Since we always get a guaranteed overall i-th best resource by choosing the maximum resource from $TopResCandidates_{sp}(Q, i)$ we can also guarantee that the top i resources of $TopRes_{sp}(Q, i)$ are already correct, while our current $(i + 1)...k$-th resources may still be replaced with better ones (as shown at the end of this section).

In the first iteration we have to consider all connected peers. In the i-th iteration we can consider only those peers from which at least one resource made it into $TopRes$ during the $i-1$-th iteration, because all other peers still offer their current best resources:

$$\begin{aligned} ConsideredPeers_{sp}(Q, 1) &= P_{sp} \cup SP_{sp}(Q) \\ ConsideredPeers_{sp}(Q, i) &= \{p \mid TopRes_{sp}(Q, i-1) \cap ResAt_p(i-1, Q) \neq \emptyset\} \end{aligned} \tag{12}$$

For a super-peer $AllResAt$ is the union of all $ResAt$-sets of connected peers

$$AllResAt_{sp}(i, Q) = \bigcup_{o \in ConsideredPeers(Q,i)} ResAt_o(i, Q) \tag{13}$$

To determine the $TopRes(Q, i)$, we first merge $TopRes(Q, i - 1)$ with the new resources delivered from the considered peers ($AllResAt_{sp}(i, Q)$), and then select the top i resources from this union.

$$\begin{aligned} TopResCandidates_{sp}(Q, 1) &= AllResAt_{sp}(1, Q) \\ TopResCandidates_{sp}(Q, i) &= AllResAt_{sp}(i, Q) \cup TopRes_{sp}(Q, i-1) \end{aligned} \tag{14}$$

$$TopRes_{sp}(Q, i) = Top_{sp}(k, Q, TopResCandidates_{sp}(Q, i)) \tag{15}$$

[7] This can be more than one resource, if there are several resources with the same score.

Example. The following top-4 example shows how our algorithm works. We assume that super-peer sp is connected to peers p_1, p_2 and p_3. We denote resources with numbers and assume that the resource number is equal to its score for query Q.
$R_{p_1} = \{r_{11}, r_{12}, r_{13}\}$ with $score(Q, r_{11}) = 0.9$, $score(Q, r_{12}) = 0.8$, $score(Q, r_{13}) = 0.1$
$R_{p_2} = \{r_{21}, r_{22}, r_{23}\}$ with $score(Q, r_{21}) = 0.7$, $score(Q, r_{22}) = 0.3$, $score(Q, r_{23}) = 0.1$
$R_{p_3} = \{r_{31}, r_{22}, r_{33}\}$ with $score(Q, r_{31}) = 0.6$, $score(Q, r_{32}) = 0.5$, $score(Q, r_{33}) = 0.4$

Then we get:
$ConsideredPeers_{sp}(Q, 1) = \{p_1, p_2, p_3\}$, $AllResAt_{sp}(Q, 1) = \{r_{11}, r_{21}, r_{31}\}$,
$TopRes_{sp}(Q, 1) = \{r_{11}, r_{21}, r_{31}\}$
$ConsideredPeers_{sp}(Q, 2) = \{p_1, p_2, p_3\}$, $AllResAt_{sp}(Q, 2) = \{r_{12}, r_{22}, r_{32}\}$,
$TopRes_{sp}(Q, 2) = \{r_{11}, r_{12}, r_{21}, r_{31}\}$
$ConsideredPeers_{sp}(Q, 3) = \{p_1\}$, $AllResAt_{sp}(Q, 3) = \{r_{13}\}$,
$TopRes_{sp}(Q, 3) = \{r_{11}, r_{12}, r_{21}, r_{31}\}$
$ConsideredPeers_{sp}(Q, 4) = \emptyset$

We can stop here, because no more resources have to be considered.

Our final result is $TopRes_{sp}(Q, k)$, but in step i we can already identify (and forward) the i-th most scored resource (set). The $ResAt$ definition extended for the super-peer case is:

$$\begin{aligned} ResAt_{sp}(1, Q) = & \quad TopRes_{sp}(Q, 1) \\ ResAt_{sp}(i, Q) = & \; TopRes_{sp}(Q, i) \setminus TopRes_{sp}(Q, i-1) \end{aligned} \tag{16}$$

We can stop the iteration as soon as $ConsideredPeers_{sp}(Q, i) = \emptyset$, because in this case $TopRes_{sp}(Q, i) = TopRes_{sp}(Q, i-1)) :$
$$\begin{aligned} & TopRes_{sp}(Q, i) \\ & = Top_{sp}(k, Q, TopResCandidates_{sp}(Q, i)) \\ & = Top_{sp}(k, Q, AllResAt_{sp}(i, Q) \cup TopRes_{sp}(Q, i-1)) \\ & = Top_{sp}(k, Q, \bigcup_{o \in ConsideredPeers(Q,i)} ResAt_o(i, Q) \cup TopRes_{sp}(Q, i-1)) \\ & = Top_{sp}(k, Q, \bigcup_{o \in \emptyset} ResAt_o(i, Q) \cup TopRes_{sp}(Q, i-1)) \\ & = Top_{sp}(k, Q, \emptyset \cup TopRes_{sp}(Q, i-1)) \\ & = Top_{sp}(k, Q, TopRes_{sp}(Q, i-1)) \\ & = TopRes_{sp}(Q, i-1)) \end{aligned}$$

Now we only have to show that $TopRes_{sp}(Q, i)$ contains the i resources with the highest score. We do this by induction. Be r_i the resource with network-wide rank i, $r_{p,i}$ the resource with rank i at peer p. For case 1 the assertion is obviously correct: $\exists p\ r_1 = r_{p,1} \Rightarrow r_1 \in AllresAt_{sp}(1, Q) \Rightarrow r_1 \in TopRes_{sp}(Q, 1)$.

Assume that $r_{i-1} \in TopRes(Q, i-1)$. If $r_i \in TopRes_{sp}(Q, i-1)$, then obviously $r_i \in TopRes_{sp}(Q, i)$. In the case $r_{i-1} \notin TopRes_{sp}(Q, i-1)$ we have to show that $r_i \in AllResAt_{sp}(i, Q)$ which we can do by showing that $\exists p\ r_i = r_{p,i}$. We know $\exists p, j\ r_i = r_{p,j}$. Both $i > j$ and $i < j$ can't be true: If $\exists j\ j > i \wedge r_i = r_{p,j}$ $\Rightarrow score_p(r_{p,i}) > score_p(r_i) \Rightarrow |BetterThan_p(Q, r_i, R_p)| \geq i$ which contradicts our assumption that r_i is at rank i.

On the other hand, if $\exists j\ j < i \wedge r_i = r_{p,j} \Rightarrow \exists j\ j > i \wedge r_i \in AllResAt_{sp}(j, Q)$. Also, $\forall l : 1 \leq l \leq j-1\ r_{p,l} \in TopRes(Q, l) \Rightarrow \forall l : 1 \leq l \leq j-1$ $p \in ConsideredPeers_{sp}(Q, l) \Rightarrow r_i \in TopRes_{sp}(Q, j) \Rightarrow r_i \in TopRes_{sp}(Q, i-1)$ which contradicts our initial assumption. Therefore, $r_i = r_{p,i} \Rightarrow r_i \in ResAt_p(Q, i)$ $\Rightarrow r_i \in AllResAt_{sp}(Q, i) \Rightarrow r_i \in TopRes_{sp}(Q, i)$. $\Box$

So the iteration terminates after at most k steps, delivering the top k resources.

3.4 Routing Optimization

We can use the query answering information collected at the super-peers to optimize routing. Let us define the set of peers at super-peer p, which have contributed to the top k resources, as

$$TopPeer_p(Q) = \{o \in P_p \cup SP_p \mid \exists r \in R_o \wedge r \in TopRes_p(Q)\} \tag{17}$$

Routing can be optimized significantly if each super-peer sends a query Q to the (super-)peers in $TopPeer_p(Q)$ only. To achieve this goal, the super-peer has to store this set for each query it evaluates. The routing index created in this way is a map of $(Q, TopPeer_p(Q))$ pairs.

Every time a super-peer receives a query, it checks whether the query is already in its index or if it is a new query. If the query matches a previous query then the super-peer will use the associated $TopPeer(Q)$ set to determine the direction in which the query should be forwarded to retrieve the top k. To find the matching index entry for a new query, we use the query containment algorithm introduced in [21]. Note that for the application in top k retrieval the number k of objects that have to be returned is an integral part of the query. It is easy to see that if a local routing index contains a query Q, every query Q' also matches Q, if Q' contains exactly the same predicates as Q, but the value k' of requested results in Q' is smaller than the value k in Q.

As we already remarked before, in a peer-to-peer network peers may join and leave at any time and the content at peers may change. Since we do not have (and do not want to have) any kind of notification of those changes, the routing of the queries must adapt to such changes automatically. To achieve this, super-peers send the query to other arbitrary (super-)peers as well, with a specific probability, depending on the volatility of the network, to capture the network dynamics. This also updates the idf values stored in the indices for these queries.

For a peer-to-peer network there are three typical distributions that we consider. Under an *arbitrary distribution* resources at one peer are not related. For this case, our hypothesis is that the more arbitrarily the resources are distributed among peers, the less optimization will be achieved by our routing indices. In contrast we have a *clustered distribution* when the resources provided by each peer are highly related. In this case our index/routing algorithm will result in much reduced message forwarding.

4 Related Work

In the context of peer-to-peer networks, only very few authors have explored retrieval algorithms taking rankings into account. The idea of PeerSearch [22] is comparable to our approach, since they are also creating an aggregation of the top k results from the peers. In contrast to our ranking-algorithm they do not use any broadcast-topology, but use CAN [23] in combination with the vector space model (VSM) and latent semantic indexing (LSI) to create an index which is stored in CAN using the vector representations as coordinates.

PlanetP [2] concentrates on peer-to-peer communities in unstructured peer-to-peer networks with sizes up to ten thousand peers. They introduce two data structures for searching and ranking which create a replicated global index, using gossiping. Each peer maintains an inverted index of its document and spreads the term-to-peer index. Based on this replicated index a TFxIDF-ranking algorithm is implemented.

Aberer and Wu provide a good theoretical background in [24] where they present a ranking algebra as a formal framework for ranking computation. They show that not only one global ranking should be taken into account, but several rankings must be seen in different contexts. Their ranking algebra allows aggregating the local rankings into global rankings.

[25] discusses the combination of distributed information retrieval and unstructured peer-to-peer networks. The basic idea is to filter results while they are routed back to the query originator. Each peer maintains a list of neighbors. These lists change over time to keep the better neighbors (i.e. the ones giving better results, which is defined as the peers which can offer more results).

In the context of databases Agrawal et al. [26] propose several approaches to rank database query results, under the assumption that there is only one table, and only conjunctive queries are used. Instead of returning only complete matches, a similarity value (similarity between query and table row) is calculated. If a condition is matched by the row data, the frequency factor (analogous to inverse document frequency) is added to the similarity value. They present similarity-measures for numerical data and range conditions.

Bruno et al. [27] use a similar formalisation like we do, but discuss the characteristics of retrieval in web databases, when several web accessible databases are used to create a top-k list, where each atom maybe answered from a different database.

5 Conclusion

Large peer-to-peer networks with semantically meaningful cooperative query answering capabilities may lead to vast result sets, which makes it necessary to investigate how to rank answers and return only the best ones in such a distributed environment. Inspired by the success of ranking algorithms in Web search engines and top-k retrieval algorithms in databases, this paper provides a solution to this problem by presenting a decentralized top-k retrieval and routing algorithm for peer-to-peer networks. The algorithm makes use of local rankings, rank merging and optimized routing based on peer ranks, and does not have to maintain a central index or locally replicated copies of such an index. The algorithm guarantees a manageable result set size and at the same time minimizes network traffic among peers. While we do assume a super-peer network, no specific network topology is required. In contrast to other approaches, our top-k indices need only to be updated during query answering, and therefore do not involve further update traffic in addition to query forwarding. We also showed two examples of how to integrate different ranking methods into our algorithm.

We are currently simulating our algorithm in the Edutella/HyperCuP environment, using different assumptions about data and query distribution, to further quantify the effects and advantages of our approach for different distributions.

6 Acknowledgements

We especially want to thank Kevin Chang from the DAIS group at Urbana-Champaign for giving valuable input to an earlier draft of this report as well as for providing several very useful references on top-k query answering and information retrieval.

References

1. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: EDUTELLA: a P2P networking infrastructure based on RDF. In: Proceedings of the Eleventh International World Wide Web Conference (WWW2002), Hawaii, USA (2002)
2. Cuenca-Acuna, F.M., Peery, C., Martin, R.P., Nguyen, T.D.: PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities. In: Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press (2003)
3. Nejdl, W., Wolpers, M., Siberski, W., Löser, A., Bruckhorst, I., Schlosser, M., Schmitz, C.: Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In: Proceedings of the Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary (2003)
4. Aberer, K., Cudré-Mauroux, P., Hauswirth, M.: The chatty web: Emergent semantics through gossiping. In: Proceedings of the Twelfth International World Wide Web Conference, New York, USA, ACM Press (2003) 197–206
5. Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I.: Piazza: Data management infrastructure for semantic web applications. In: Proceedings of the Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary (2003)
6. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In: Proceedings of the Fifth International Workshop on the Web and Databases, Madison, Wisconsin (2002)
7. Nejdl, W., Siberski, W., Sintek, M.: Design issues and challenges for RDF- and schema-based peer-to-peer systems. SIGMOD Records (2003)
8. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Proceedings International Conference on Distributed Computing Systems. (2002)
9. Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. In: In Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS), Trento, Italy (2001)
10. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: HyperCuP—Hypercubes, ontologies and efficient search on P2P networks. In: International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy (2002)

11. Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Transactions on Knwoledge and Data Engineering **15** (2003)
12. Witten, I., Moffat, A., Bell, T.: Managing Gigabytes. Morgan Kaufman, Heidelberg (1999)
13. Hewlett Packard Research Labs: RDQL - RDF data query language (2004) http://www.hpl.hp.com/semweb/rdql.html.
14. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. In: Proceedings of the 29th International Conference on Very Large Databases, Berlin, Germany (2003) 754–765
15. Viles, C.L., French, J.C.: On the update of term weights in dynamic information retrieval systems. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM '95), Baltimore, MD, USA, ACM (1995) 167–174
16. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: ACM SIGCOMM Computer Communication Review, Vol 29(4). (1999)
17. Chen, Q.: The origin of power laws in internet topologies revisited. In: 21st Annual Joint Conference of the IEEE Computer and Communications Societies. (2002)
18. Medina, A., Matta, I., Byers, J.: On the origin of power laws in internet topologies. ACM SIGCOMM Computer Communication Review **30(2)** (2000)
19. Adamic, L.A., Huberman, B.A.: Zipf's law and the internet. Glottometrics **3** (2002)
20. Crespo, A., Molina, H.G.: Semantic overlay networks for P2P systems. Technical report, Stanford University (2003)
21. Chirita, P.A., Idreos, S., Koubarakis, M., Nejdl, W.: Publish/subscribe for RDF-based P2P networks. In: In Proceedings of the 1st European Semantic Web Symposium. (2004)
22. Tang, C., Xu, Z., Mahalingam, M.: Peersearch: Efficient information retrieval in peer-to-peer networks. Technical Report HPL-2002-198, Hewlett-Packard Labs (2002)
23. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications. (2001)
24. Aberer, K., Wu, J.: A framework for decentralized ranking in web information retrieval. In: Proceedings of the fifth Asia Pacific Web Conference (APWeb2003). (2003)
25. Yu, B., Liu, J., Ong, C.S.: Scalable P2P information retrieval via hierarchical result merging. Technical report, Dep. of CS, University at Urbana-Champaign (2003)
26. Agrawal, S., Chaudhuri, S., Das, G., Gionis, A.: Automated ranking of database query results. In: Proceedings of the Second Conference on Innovative Data Systems Research. (2003)
27. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, IEEE Computer Society (2002)