Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks

Wolf-Tilo Balke[1] Wolfgang Nejdl[2]
[1]*University of California, Berkeley, USA*
balke@eecs.berkeley.edu

Wolf Siberski[2] Uwe Thaden[2]
[2]*L3S, University of Hannover, Germany*
{nejdl, siberski, thaden}@l3s.de

Abstract

Query processing in traditional information management systems has moved from an exact match model to more flexible paradigms allowing cooperative retrieval by aggregating the database objects' degree of match for each different query predicate and returning the best matching objects only. In peer-to-peer systems such strategies are even more important, given the potentially large number of peers, which may contribute to the results. Yet current peer-to-peer research has barely started to investigate such approaches. In this paper we will discuss the benefits of best match/top-k queries in the context of distributed peer-to-peer information infrastructures and show how to extend the limited query processing in current peer-to-peer networks by allowing the distributed processing of top-k queries, while maintaining a minimum of data traffic. Relying on a super-peer backbone organized in the HyperCuP topology we will show how to use local indexes for optimizing the necessary query routing and how to process intermediate results in inner network nodes at the earliest possible point in time cutting down the necessary data traffic within the network. 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, as well as dynamic additions/deletions of peers. We will show our approach to always deliver correct result sets and to be optimal in terms of necessary object accesses and data traffic. Finally, we present simulation results for both static and dynamic network environments.

1. Introduction

With information needs emerging beyond a simple exact match paradigm, databases and information systems have since long catered for extended retrieval paradigms like top-k retrieval or skyline queries. Query languages like SQL over relational databases have been extended to facilitate rank- and/or score-based retrieval algorithms assigning a degree of match with respect to all query predicates to each database object and then aggregating the rank/score values to get only the set of best matching answers. Moreover, the new retrieval paradigms allow for the direct incorporation of user preferences into queries for a more cooperative retrieval behavior [9]. Whereas too specific query predicates under the exact match paradigm would far too often lead to empty result sets, the notion of best matches and relative importance of predicates can thoroughly satisfy a user's information needs independent of the respective database instance. Especially top-k queries [12] delivering a well defined set of k best answers according to a user-provided, probably weighted compensation function, have shown their broad applicability in various areas like Web search engines, mobile database applications, or content-based retrieval in multimedia collections or digital libraries.

Recently the provisioning of information has, however, entered a new stage of flexibility beyond centralized database servers that process queries. Peer-to-peer networks offer flexible access to large collections of information, data objects or documents for exchange. Emerging from relatively simple architectures to exchange e.g. audio files peer-to-peer networks have been developed to complex ad-hoc data management systems that can already be employed to connect communities in different areas like e-learning or collaborative working (cf. [15], [14], [1]). Together with more sophisticated applications also the basic processing of queries has evolved. Relying on backbone structures of so-called super-peers, queries are not necessarily flooded through the entire network anymore, but can be purposefully routed to relevant parts of the peer-to-peer network or even single peers using centralized, distributed or even synchronized local indexes [15], [17]. Recent studies show that these approaches can drastically reduce the data traffic within the network. Up to now, however, the use of these indexes was restricted to exact match queries, each index pointing to (super-)peers that are known to provide a certain piece of information.

In this paper we will extend traditional query processing in peer-to-peer networks by allowing the distributed processing of top-k queries with a minimum of object data traffic. Relying on a super-peer backbone organized in the HyperCuP topology, we will show how to use sophisticated local indexes for optimizing the necessary query routing and how to process intermediate query results in inner network nodes at the earliest possible point in time. Furthermore, we propose dynamic query driven index updates, which avoid continuous index update traffic in a P2P environment, while working well especially for Zipf-like distributions of queries, i.e. after an initial build-up phase over 90% of queries are index hits.

The rest of this paper is organized as follows: section 2 will provide an overview on related work in top-k query processing in traditional information systems and current query processing in peer-to-peer networks. Section 3 will present our distributed top-k retrieval algorithm and give an insight in how it works. A discussion of the algorithm's correctness and the optimality of data traffic will be given in section 4, followed by a detailed practical evaluation in section 5. We will conclude with a short summary and an outlook on further interesting open research problems.

2. Top-k Retrieval Model and Related Work

Since the top-k paradigm has been first introduced into the area of database systems a large number of different algorithms have been proposed, see e.g. [12], [13]. Algorithms for top-k retrieval in databases generally try to minimize the number of database objects that have to be accessed before being able to return a correct result set of the k best matching objects. Improving the naïve algorithm that simply aggregates scores for all database objects, an optimal algorithm was given for multimedia retrieval [13], databases [12] and Web searches [8]. To be more exact, this basic algorithm was proven to be optimal in minimizing the necessary object accesses for top-k querying [12]. Basically all algorithms distinguish between different query predicates evaluating different characteristics of database objects, which often even have to be retrieved from various subsystems or Web sources. Each subsystem assesses numerical score values (usually normalized to [0, 1]) to each object in the collection. The physical implementation of object accesses always strongly depends on the application area and usually differs from system to system. Then the integration of the individual subsystems' score lists is performed on a central server. As a rule minimizing the number of necessary object accesses and thus also the overall query runtimes is paramount to build practical systems (with real-time constraints) like discussed in [4].

In the context of peer-to-peer networks, only very few authors have explored retrieval algorithms taking rankings into account. PlanetP [11] 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 algorithms. Each peer maintains an inverted index of its documents and spreads the term-to-peer index. Based on this replicated index a simple TFxIDF-ranking algorithm can be implemented.

Aberer and Wu provide a good theoretical background in [2] where they present a ranking algebra as a formal framework for ranking computation. They show that not only one global ranking should be taking into account, but several rankings must be seen in different contexts. Their ranking algebra allows aggregating a number of local rankings into global rankings.

Another important aspect is to take different peer capabilities into account, as peers often vary widely in bandwidth and computing power. Exploiting these different capabilities - as discussed in [20] - can lead to a more efficient network architecture, where a small subset of peers, called super-peers, takes over specific responsibilities, like e.g. query routing. Only a small percentage of nodes are super-peers, but these are assumed to be highly available nodes with relatively high computing capacity. Peers join the network directly connecting to one of the super-peers. Super-peer based P2P infrastructures usually exploit a two phase routing architecture, which routes queries first in the super-peer backbone, and then distributes them to the peers connected to the super-peers.

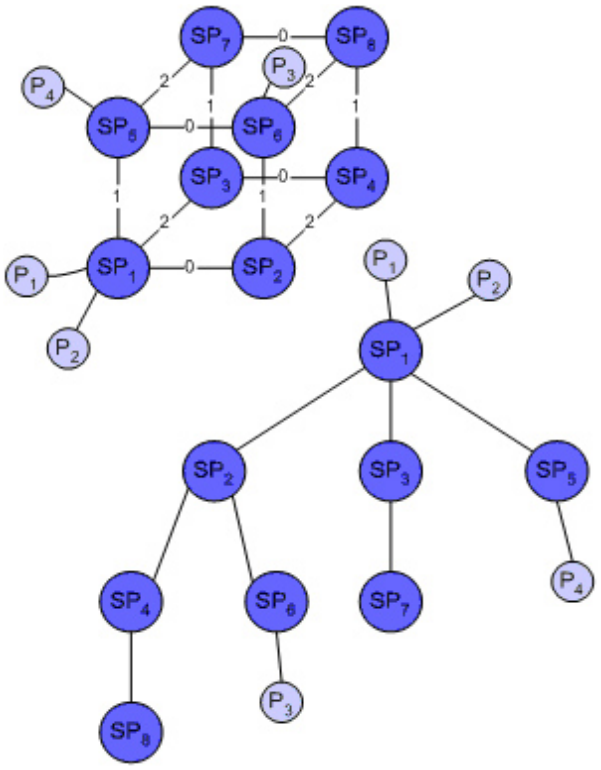

Fig. 1: Simple HyperCube and an implicit spanning tree

One important decision is how to arrange the super-peers in order to optimize the routing of queries in the network. In our Edutella P2P-network [16] they are organized in the HyperCuP topology [18]. The HyperCuP algorithm is capable of organizing super-peers into a recursive graph structure from the family of Cayley graphs, out of which the hypercube is the most well-known topology. Having N super-peers in a network, the HyperCuP ensures a maximal path length of $\log_2 N$ thus allow-

ing for optimal broadcasting as well as implicit spanning trees from each node of the network. The basic query routing algorithm in HyperCuP topologies works as follows: All edges are tagged with their dimension in the hypercube. A node invoking a request sends a message to all its neighbours, tagging it with the edge label on which the message was sent. Nodes receiving a message forward it only via edges tagged with lower edge labels (see [18] for details). Using this topology we will have an optimal number of hops independently from the node that poses the query. Figure 1 shows the implicit spanning tree for a message originating from peer SP1 (since Hypercubes are symmetric, any super-peer can be the root of a spanning tree). Please note that the Hypercube is not limited to a special dimensionality and that super-peers can join and leave the network with very little overhead, allowing for a flexible approximation of a genuine hypercube structure.

Additional routing indexes maintained by these super-peers can restrict broadcasts to relevant super-peers directions [17] and can be used to enable optimized processing of more complex queries in a peer-to-peer network [5]. However, additional ranking and top-k optimization is necessary in such an environment and will be discussed in detail in the current paper.

3. A Distributed Top-k Retrieval Algorithm for Peer-to-Peer Networks

In this section we will present our algorithm for basic top-k querying capabilities in P2P networks with minimum transfer of object data. According to the distributed nature of the retrieval and the P2P network we will divide the distributed retrieval algorithm into three parts that are respectively executed by

- the super-peer initially receiving the query,
- the super-peers in the HyperCuP backbone,
- and the local peers at each super-peer.

Since a dissemination of global knowledge should be avoided due to the overhead of data transmission, a basic concept of our algorithm is to locally evaluate as many parts of the query as possible. This means only the super-peer receiving the query (i.e. the root node of our implicit HyperCuP spanning tree) needs full information to control the execution of the queries in order to guarantee a correct top-k result set with a minimum transmission of data. This super-peer will hand on the query to the relevant super-peers along the backbone of adjacent super-peers, which in turn will forward the query to their relevant adjacent super-peers and connected local peers, without having to have full information about how the query answering is progressing. The local peers will just execute the query over their local object collections or databases and retrieve some best matching objects. We will present all relevant steps in detail in the following.

For the scope of this paper we will rely on a set of super-peers managing a number of local peers and interconnected by a backbone using the HyperCuP topology. We will also assume all peers to be cooperative and provide normalized scores that can be compared to distinguish the quality between different objects (see section 4.1 for a detailed discussion). Each super-peer SP manages an index I_{SP} in that information about which of its local peers and adjacent super-peers contributed results for answering recently posed queries like shown in [5]. These indexes can be maintained efficiently even in rather volatile P2P networks to hold "current enough" information about object distributions. All index entries are time-stamped and expire after a certain time, which is set depending on the volatility of the network. Thus the individual index entries can be kept "up-to-date enough" to allow for improved query processing even in volatile networks.

Let us now present the algorithm to answer a top-k query Q posed by peer P to super-peer SP. The algorithm is entirely controlled by super-peer SP, which – whenever necessary – poses requests to connected peers and super-peers for localized information gathering. Since all super-peers are organized in a HyperCuP topology and SP is the root node of an implicit spanning tree containing all super-peers, we will use the notion of *adjacent* super-peers of SP, i.e. those super-peers that can directly receive messages from SP, but are more distant from the root node, i.e. whose HyperCuP edge label is smaller. Moreover, let us assume that the query is answered using a snapshot of the current P2P network at query time, i.e. the connections stay constant for the time of the query answering process. We will deal with disconnections during query processing later by introducing time-outs for peers that do not answer a request within a tolerable time span.

Algorithm for Peer-to-Peer Top-k Retrieval

0. Assign a unique transaction identifier T depending on the query *Q*, querying peer *P* and super-peer *SP*. Initialize a counter $i := 0$.
1. Choose the participating peers and super-peers for answering the query *Q*: If query *Q* is contained in index I_{SP}, and this index entry is not expired, assign sets of contributing local peers P_T and adjacent super-peers SP_T as given by I_{SP}, else set P_T as the set of all locally connected peers and SP_T as the set of all adjacent super-peers.
2. Initialize a datastructure $TopRes_T$ as a $|P_T| + |SP_T|$ - dimensional array of oid and score pairs. Assign each (super-)peer in P_T and SP_T to a specific field in $TopRes_T$. Initialize three sets $BestPeers_T := \emptyset$, $Delivered_T := \emptyset$, and $RequestResults_T := \emptyset$.
3. Send an open_transaction(T, *Q*, *SP*) request to each (super-)peer in P_T and SP_T and add the respective(super-)peer to set $RequestResults_T$

4. Send an get_next(T, *Q*, *SP*) request to each (super-) peer in $RequestResults_T$ and remove the respective peer from $RequestResults_T$.
5. For each incoming message that a (super-)peer cannot deliver more result objects, send a close_transaction(T, *Q*, *SP*) request to the (super-)peer, remove the (super-)peer from P_T or SP_T and delete its assigned field in $TopRes_T$.
6. For each incoming oid/score pair from (super-)peers with respect to transaction T do
   6.1. If oid $\notin Delivered_T$, store the oid/score pair in the respective field in $TopRes_T$ assigned to the delivering (super-)peer, else discard the pair and add the delivering (super-)peer to $RequestResults_T$.
7. If there are still missing entries in any field in $TopRes_T$, proceed with step 4.
8. Select all distinct objects with current maximum score from $TopRes_T$. While $i \leq k$ and there are still objects, do:
   8.1. Pick any object *o* having maximum score and deliver its oid and score to peer *P* as the *i*-th best object.
   8.2. Add object *o*'s oid to the set $Delivered_T$ and increase $i := i+1$.
   8.3. Remove object *o*'s oid and score from all occurrences in $TopRes_T$ and add the corresponding (super-)peers of the respective fields to $BestPeers_T$ and $RequestResults_T$.
9. If $i > k$, send a close_transaction(T, *Q*, *SP*) request to all (super-) peers in P_T and SP_T and update I_{SP} for query *Q* using the (super-)peers in $BestPeers_T$. Discard all temporary results and datastructures and terminate the algorithm.
10. If set $RequestResults_T$ is empty, abort query *Q* at peer *P* with the message that only *i* results are available and discard all temporary results and datastructures, else proceed with step 4.

Please note that although generally speaking the number k of objects to be returned is an integral part of the query *Q*, the sets P_T and SP_T in step 1.1 can also be assigned, if index I_{SP} does not contain query Q, but query Q' with the same query predicates, but a larger number of objects to return than k. The resulting sets P_T and SP_T will in that case not be optimal for query *Q*, but usually still result in much better performance than simply flooding the query through the network.

Another interesting side effect is the successive output of result objects in step 8.1 such that the user can already investigate some first overall best result objects before all k overall best result objects have been found. Though the total retrieval time stays the same as in the case where all objects are returned after all top k objects have been determined, the psychological response time is reduced for the users. Moreover, once a user is satisfied by the object(s) from the result set retrieved so far, she can terminate the query at an early stage before the full result set has been retrieved and thus improve bandwidth usage.

Let us now consider the functions that are called in our distributed algorithm requesting locally connected peers and adjacent super-peers to join into a query processing task and to deliver their best matching objects. These function calls are open_transaction(T, *Q*, *SP*), get_next(T, *Q*, *SP*), and close_transaction(T, *Q*, *SP*). Since their basic functionality does not differ for peers and super-peers, though their local execution has to be slightly different, we will assume that their individual implementations are simply overwritten to suit the respective (super-)peers. For the requesting super-peer *SP* their purpose, interface, and expected results do not differ between peers and super-peers. We will start with the implementations of the functions in local peers and then turn to the functions in the super-peers handling the local aggregation tasks.

Since local peers may differ in their information management and querying techniques, their implementation may essentially differ between peers. Some peers may rely on a database management system to store and query data, while other may use a variety of custom made applications to manage their data. We will assume a component in each peer that wraps the results and messages according to the super-peers' needs, and leave the actual local top-k querying and scoring to the individual peers. For example peers relying on a local DBMS may use any algorithm like [13], [4] or [5], whereas other peers may rely on filtering techniques for their collections. In the following we will only assume that if asked to, every peer joins a transaction, is able to evaluate a top-k query locally, iterate over the respective result set, and deliver its objects using a specific oid (e.g. an URI, etc.) together with a distinctive score value normalized to the interval [0, 1]. When running out of deliverable objects a peer notifies its super-peer with a suitable message. The requests sent to a local peer are:

Basic Functions at Each Local Peer:

open_transaction(T, Q, SP):
If a peer receives this request it will prepare to answer the top-k query *Q* over its local data (e.g. open a result set and position a cursor) and assign all further requests with the identifier T to the respective result set.

get_next(T, Q, SP)
If a peer receives this request it will iterate over the result set assigned to T (e.g. move the cursor or get the next best object from a progressive retrieval algorithm) and send the respective result object's oid and score to super-peer *SP*. If there are no more results that can be delivered, if will send a respective message to super-peer *SP*.

close_transaction(T, Q, SP)
If a peer receives this request it aborts the query assigned to T (e.g. close an open result set) and may discard the related temporary results.

The functionality in super-peers is a bit more difficult than for the local peers, but quite similar to respective steps in the algorithm at the querying super-peer. Assume that a super-peer SP sends a request to an adjacent super-peer SP'. To open the transaction for a specific query this super-peer SP' will also have to choose local peers and his adjacent super-peers to participate in the query from its local routing index. If requested, it will have to aggregate information, determine the current best object locally, and hand it on to the super-peer that requested it. Eventually the super-peer will have to close the transaction:

Functions at each super-peer SP':
open_transaction(T, Q, SP):
1. Choose the participating peers and super-peers for answering the query *Q*: If a query *Q* posed by SP is already contained in index $I_{SP'}$, and this index entry is not expired, assign sets of contributing local peers P_T and adjacent super-peers SP_T as given by $I_{SP'}$, else set P_T as the set of all locally connected peers and SP_T as the set of all adjacent super-peers.
2. Initialize a datastructure $TopRes_T$ as a $|P_T| + |SP_T|$ -dimensional array of oid and score pairs. Assign each (super-)peer in P_T and SP_T to a specific field in $TopRes_T$. Initialize three sets $BestPeers_T := \emptyset$, $Delivered_T := \emptyset$, and $RequestResults_T := \emptyset$.
3. Send an open_transaction(T, *Q*, *SP*) request to each (super-)peer in P_T and SP_T and add the respective(super-)peer to $RequestResults_T$

get_next(T, Q, SP)
1. If set $RequestResults_T$ is empty, send a message to super-peer SP that no more objects can be delivered, else do
   1.1. Send an get_next(T, *Q*, *SP'*) request to each (super-) peer in $RequestResults_T$ and remove the respective peer from $RequestResults_T$.
   1.2. For each incoming message that a (super-)peer cannot deliver more result objects, send a close_transaction(T, *Q*, *SP*) request to the (super-)peer, remove the (super-)peer from P_T or SP_T and delete its assigned field in $TopRes_T$.
   1.3. For each incoming oid/score pair from (super-) peers with respect to transaction T do
      1.3.1. If oid $\notin Delivered_T$, store the oid/score pair in the respective field in $TopRes_T$ assigned to the delivering (super-)peer, else discard the pair and add the delivering (super-)peer to $RequestResults_T$.
   1.4. If there are still missing entries in any field in $TopRes_T$, proceed with step 1.2.
   1.5. If there are objects in $TopRes_T$, select an object with current maximum score from $TopRes_T$. and deliver its oid and score to super-peer SP. Add the object's oid to the set $Delivered_T$ and remove the object's oid and score from all occurrences in $TopRes_T$ and add the corresponding (super-) peers of the respective fields to $BestPeers_T$ and $RequestResults_T$.

close_transaction(T, Q, SP)
1. Send a close_transaction(T, *Q*, *SP*) request to all (super-) peers in P_T and SP_T and update $I_{SP'}$ for query *Q* and querying peer SP using the (super-)peers in $BestPeers_T$.
2. Discard all current sets and datastructures

Please note that for using the index $I_{SP'}$ in processing the open_transaction(T, *Q*, *SP*) request, it is of essential importance that the query *Q* has been posed by super-peer SP before, as the child nodes (and thus adjacent super-peers that can deliver relevant results) depend on the respective edge weight of the implicit super-peer spanning tree in the HyperCuP topology. Thus all the knowledge of which adjacent nodes may provide interesting information for the top-k case, is dependent on the topology for each query instance. Due to the characteristics of the HyperCuP topology it is irrelevant, whether SP is the querying peer running the top-k search, or just passing the query on. It is only important that the query was routed via the edge between SP and SP'. As stated before, index entries for identical queries delivering more top elements as in the current request, can be used instead of querying all peers and adjacent super-peers.

The index I_{SP} always holds information for routing optimization in sense of avoiding irrelevant destinations. For each query it contains the peers, which have recently contributed to a top-k result set. While results are delivered by the super-peers, for each query transaction T we maintain statistics in a set $BestPeers_T$, which peers/super-peers returned the best results. On query completion this information is used to update all local routing indexes. To adapt to changes in the peer-to-peer network, we let all index entries expire after a specified time span. The more volatile the network is the shorter the expiration period has to be in order to adapt to changing data allocations. Query driven update of indexes is possible, because queries are not posed randomly, but usually follow a Zipf distribution, where few queries make up the majority of all requests. Zipf distributions are ubiquitous in content networks, the Internet and other collections and have become one of the most empirically validated laws in the domain of linguistic quantities and networks (in the form of power law distributions) [3].

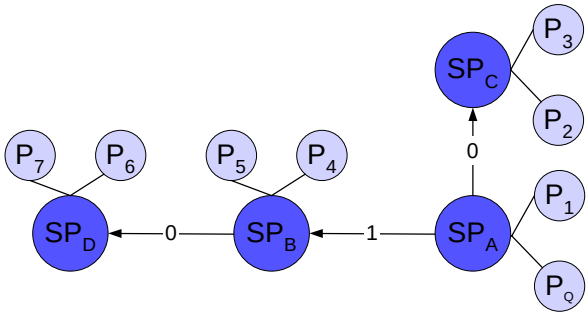

Fig. 2: Querying a super-peer-based P2P-network

To show in more detail how the algorithm works, let us consider an example for the simple scenario given in Figure 1. Suppose we have a backbone of four super-peers each accommodating two local peers. A top-k query Q by peer P_Q will define SP_A as root of the super-peer backbone spanning tree (cf. labeled edges in Fig. 2) and start the query processing in SP_A for transaction T. Assume that the query is a top 2 query that has recently been posed. SP_A can check its local index to find out that only P_1 and SP_B have contributed to the result (step 1.1), i.e. P_T and SP_T are single element sets containing P_1 and SP_B respectively. Assuming that our P2P network is not too volatile, we won't present index entry expiration in this example and initialize a two-dimensional array containing the current top elements of SP_B and P_1 (step 2) Note that in contrast to flooding query frameworks SP_C will not be bothered at all with query answering, since SP_A already knows that it most probably cannot deliver suitable results. SP_A then will open transactions in P_1 and SP_B (step 3) and add both to a set $RequestResults_T$.

Receiving the request P_1 will perform its local query, whereas SP_B will look up its index and may find that the query was recently posed by super-peer SP_A and the top 2 results came from SP_D and P_4. Subsequently SP_B will open transaction T in P_4 and SP_D (which may then in turn open the transaction in say P_7) and initialize all datastructures needed. Now SP_A will request the top objects from P1 and SP_B (step 4). Assume that P1 offers an object o_1 with score 0.8. Receiving the request SP_B will in turn request the top results from P_4 and SP_C, which in turn will request the top object from P_7. SP_B has to wait until it has both results from P_4 and SP_C (to be on the safe side there will be a timeout after which SP_B will assume that any (super-) peer, which has not yet delivered, has dropped out and thus will not be considered in the query any further). Assume P_7 delivers an object o_2 with score 0.7, which is handed on by SP_C to SP_B. SP_B also receives the top object o_3 with score 0.9 by P_4. SP_B now has to pass on the maximum object o_3 to SP_A, which in turn chooses the maximum object o_3 (step 8) and passes it on to P_Q as the overall top object. Since we need the top 2 objects and the best object of P_1 is still valid, SP_A will request the next object only from SP_B who in turn will only request the next object from P_4 (say o_4 with score 0.7) and deliver it to SP_A (steps 4-7). Now SP_A can again choose the top object o_1 by P_1, deliver it to P_Q (step 8) and close the transaction (step 9). Since the top objects came from SP_B and P_1 there is no need to update SP_A's index. SP_B will in turn close the transaction in P_4 and SP_C, however remove SP_C from its local index, because it did not contribute to the result of query Q.

4 Correctness and Optimality Results

The discussion of correctness of retrieval results in P2P networks is a difficult matter because of their volatile nature. In traditional database retrieval complex transaction protocols have been designed to assure that no phantom objects occur in the retrieval process (e.g. when objects are updated) and it is easy to decide, if a retrieval algorithm has left out relevant results or retrieved false results (cf. precision/recall in IR). In exact match P2P retrieval each peer returns all its results matching the query (flooding of queries) or the objects are retrieved after a matching entry (and thus the address of a peer offering a result object) in some centralized or distributed index structure has been found. However, it cannot be guaranteed that

- a peer is always online for the necessary time to transfer these result objects,
- a peer for a matching index entry is still available,
- or that no new peer offering relevant content has occurred after a part of a distributed index has been evaluated.

Thus the correctness of retrieval results in P2P retrieval is always considered on a 'static snapshot' of the P2P network. If no peers drop out or new peers are registered between the atomic evaluation of the query and the delivery of the result set, the query has been answered correctly. Such a model becomes of course less adequate the more volatile a P2P network gets.

The problem how to define correct retrieval gets even worse, if top-k queries have to be answered. Besides the difficulties with the volatility of the P2P network, also the heterogeneity of the peers plays an important part, since each peer only knows its local objects and different peers may also feature different scoring or retrieval strategies. To compare between objects is thus only possible, if the cooperative behavior of all peers is assumed and a normalization of the score values (e.g. to the interval [0, 1]) is requested of each single peer. Nevertheless different peers may still score the same object differently. Since the only solution of retrieving all objects and scoring them centrally with a single method (even the naïve approach of retrieving only the top k objects of every peer and then rescoring results does not solve this problem, because different scoring methods usually do not maintain relative

monotonicity) is clearly impractical and generally not desirable in the P2P context anyway, we will always use the maximum score value any queried peer has assigned to an object as the relevant score for this object.

Since [16] shows that querying with partial indexes along a super-peer backbone (pointing always to the top peers, which have recently contributed to the retrieval result of the same query) usually results in a good enough response behavior, our algorithm will combat the problems of volatility and heterogeneity relying on the experiences with such indexes. We will therefore investigate correctness regarding correct top results with respect to the relevant part of a minimum spanning tree induced by an index entry maintained by the super-peer receiving the query. Moreover, we assume that the additional querying of arbitrary (super-)peers in step 1.2 will be sufficient to react to changes of content allocation within the network. We will directly use the normalized scorings of local peers for the result set and take only the maximum score for each distinct object into account.

Lemma 1 (Correctness of the Top-k Result Set): Given that only cooperative peers are part of the peer-to-peer network and retrieval scores between peers can be compared with respect to the objects' relevance to answering the top-k query, i.e.
a) individually assigned scores are reliable and no peer maliciously assigns high scores to irrelevant objects
b) for any two score assignments for different objects from different peers, it can be correctly decided, whether one of the objects is better than the other(s),
the distributed top-k algorithm always retrieves the correct set of k overall best objects with respect to the index of the querying super-peer, if the maximum assigned score is used as the relevant object's score in the case of some peers assessing the score for a database object differently.

Proof:
Since - as discussed above - the correctness of a result set can only be decided at a certain point in time over a fixed set of database objects/documents, we consider the retrieval situation for our algorithm with up-to-date indexes in the super-peers and assume that neither new documents are added to the collection, nor are documents deleted.

For the proof we use an inductive argument based on the aggregation of the local rankings. Since our indexes are up-to-date we know that some overall top scored object o_{top} is in one of the local collections of at least one peer in some index. Without loss of generality let us assume there would only be a single object o_{top}. For the sake of contradiction now assume that our querying super-peer would deliver a dominated object o_x with $score(o_x) < score(o_{top})$ as overall top-scored object. Since all super-peers indexes are up-to-date and we know from step 1 that all relevant peers have been queried and their best objects have been requested (step 4), there must be a path through the P2P network from our querying super-peer to at least one peer holding o_{top}.

This local peer has to return o_{top} as best object to its connected super-peer due to the monotonic iteration through scorings in each local peer (i.e. at any point in time every local peer on request returns the highest scored object it can offer and which has not yet been output). Otherwise, as we always use the overall maximum scoring as relevant score for each object, there would exist an object having a strictly better score than o_{top}. Moreover, since there is a path along the super-peer backbone and each super-peer has to wait until all results have arrived (step 4-7) before determining the local maximum score within its local collection (step 8) all super-peers along the path from the querying super-peer to the local peer hosting o_{top} also have to select o_{top} as best object and pass it on, until it eventually arrives at the querying super-peer. Thus by delivering o_x the super-peer would have chosen a dominated object though knowing o_{top}, which is a contradiction to the maximum search in step 8.1 of our algorithm. Inductively and considering that already delivered objects offered by any local peer are replaced by the next best objects (i.e. with monotonically decreasing score values) this peer can offer (step 6.1), we get the guaranteed correctness of the best k retrieval results. ■

Having proven that a correct result set with respect to an up-to-date index is retrieved by our algorithm, we also have to consider the overall costs of transmitting the necessary information about object data. The next lemma shows that in order to retrieve the result set, only a minimum amount of object data needs to be transmitted.

Lemma 2 (Optimality of transferred object data): Given the assumptions and conditions of lemma 1, the distributed top-k retrieval algorithm needs to transfer only a minimum amount of object data to find the correct result set.

Proof:
To show the optimality of the transmitted object data we focus on the score constraints that are maintained by the local peers and super-peers along the backbone. To decide for the correct maximum in each single super-peer we have to request the best objects from each local peer in its index and the adjacent super-peers towards the leaves of the minimum spanning tree. Once the maximum is chosen, the best available objects are still correct for all but one local peer or adjacent super-peer. For the next maximum choice only the next best object has to be requested from this and only this (super-)peer (unless the same maximum object was offered by two or more peers in which case step 8.3 will also request next best objects from these other peers).

As the querying super-peer at the root of the super-peer backbone knows what object was returned as part of

the retrieval result, which (super-)peer's best objects do still hold and how many objects are still needed, the iterative requests of next best objects have to be issued by this super-peer as in step 4. Therefore only (super-) peers have to refresh their offer (and thus transmit more object data), whose current best offers have already been part of the final result set. This transmission is necessary, since after delivery of a peer's best object, the next best object of this peer can still have a higher score than the best objects of the other peers at the same super-peer. Omitting the transmission when starting a new search for the maximum scored object in a super-peer could violate the correctness of the result set and hence the amount of object information transmitted is optimal. ■

The last lemma is useful for handling disappearing peers in volatile networks. The typical situation in top-k queries is that a user poses a query on a rather small amount of objects/documents to retrieve. Typical values for k can be assumed to range around 10 in most practical applications. The result set delivered can then either consist of k pointers to the peers and objects/documents together with the actual score value or the actual objects/documents themselves together with the respective scores. Given that the network can be volatile (especially since big parts of the objects under consideration have to be handed on along the super-peer backbones and each super-peer has to wait until it has collected all necessary objects), it can happen that a peer vanishes after one of its objects has been delivered as part of the result set. If the original document has been delivered in the result set, this does not pose a problem. In contrast, if only a pointer has been delivered, the retrieval of the result document fails and there will be less than k objects. So immediately delivering the objects/documents often is a sensible approach that - given the result of lemma 2 - does not result in unnecessary waste of valuable network bandwidth.

In some situations, applications might favor optimizing the number of messages instead of bandwidth, e.g. if the bandwidth is large, but the RTT long. In this case our algorithm can be straightforwardly adapted by letting *get_next()* return a set of up to *k* results instead of just one result per message. The optimal value for this batch transfer could even be chosen depending on the actual connection characteristics between two nodes.

5 Simulation and Results

We implemented the algorithm in a super-peer network simulator, whose first version we described in [5], and which we extended for the algorithms and simulations described in the current paper. We will in brief describe the simulator design in 5.1, the simulation scenarios in 5.2, and the simulation results in 5.3.

We wanted to test the following hypotheses:

1. After the routing index is built up, we will not touch more than $n_{SP} + k$ peer nodes, where n_{SP} is the number of super-peers.
2. On average, the difference between the top-k document set delivered with our algorithm and the document set which would be retrieved by a central computation of the union of all peers documents is small (i.e. the index-based routing also in volatile P2P networks only causes slight changes).

5.1 Simulator design

Query- and Resource Description. The main functionality of our simulator is to experiment with query routing in super-peer based P2P networks based on index information. Query messages consist of a list of keywords used to formulate the request as a simple conjunction. For the generation of such queries a configurable distribution is taken into account. We can set the average number of keywords used in a query (and deviation), and the distribution used to chose terms (e.g. Zipf or uniform). When a peer is created, we randomly assign documents from an existing collection to it. When a query is received by a peer, an appropriate response set is generated. For our network it makes no difference whether the queries originate at local peers or directly at super-peers. The generated queries are distributed evenly to the super-peers' input queues.

Super-Peer based Topology. As stated above the simulation framework assumes a super-peer topology. All simple peers have exactly one connection to a super-peer. The super-peers form their own peer-to-peer network (it would also be possible to simulate a conventional P2P network by instantiating the super-peer backbone only). The super-peer network topology and protocol is pluggable. For the experiments described in this paper we used the HyperCuP topology. In contrast to other simulations our approach doesn't rely on a TCP/IP network simulation, but models connections between peers on a higher level. Connections are assigned a certain bandwidth (specified by messages per second) and delay (in msec). Both properties are modeled as normal distributions with configurable deviation. As we assume that SP/SP connections typically have a higher capacity than SP/P connections, the parameters can be set separately for these connection categories. Super-peers are assumed to be highly available, so we don't model their up- and downtime, but simulate using a static backbone. This makes it very simple to create different super-peer topologies because it is not necessary to implement a full connection/disconnection protocol. Instead, a topology class creates all super-peers and the connections between them on simulation startup. Of course, the implementation for the real network has to consider joining and leaving super-peers, but, as super-peer joins or failures will be

rare, their influence on the network performance will not be significant.

Connections (network characteristics). All connections are considered bi-directional. Each peer (including super-peers) has an incoming message queue per connection, a single processing queue and an outgoing message queue per connection. Messages between the peers are interpreted as discrete events.

Implementation. Our simulator is based on the discrete simulation framework SSF (Scalable Simulation Framework [10]). The SSF provides an interface for discrete-event simulations supporting object-oriented models to utilize and extend the framework. Thus the potential for direct reuse of model code is maximized, while the dependencies on a particular simulator kernel implementation are minimized. The framework's primary design goal was to support high performance simulations and to make models efficient. The SSF provides several classes that we used to map the P2P-behavior to the mode. Entities are the central class in SSF, which can have processes for event-processing. Events change the status of the system and are used for communication between entities.

Processes are used to handle events during the simulation. An entity can have one or more processes. In- and out-channels are the communication channel between the entities. An entity can have several in- and out-channels, which are always connected in a 1:1 fashion. This is a perfect match for P2P simulation. Peers are modeled as entities and the message they exchange are events. We provide classes which implement the general behavior of a super-peer based network (e.g. Peer, Super-peer, Message, and Topology). For a specific setting these classes are extended; for the simulation described here we provided specializations for peers, super-peers, and query and response messages. Additional details about the simulator are given in [5].

5.2 Data and Simulation Setup

We evaluated the query processing for top-1 and top-10 queries, both in a static and in a dynamic network setting.

Top-10. The top-10 case is the typical search engine use case. In most cases, users expect to find an appropriate document among the best 10 hits. Therefore it is a reasonable value for the number of requested hits. If no suitable documents are found, users either refine their query (covered by the scenario) or ask for more results (not included in the simulation, but of course possible within our approach).

Top-1. This situation often occurs, when applications or services execute a query to use the results in an internal process. A typical example can be found in the context of Web Services. To execute a predefined process consisting of several Web Services, the process execution engine has to find best matches for each service specification used in the process specification. However, only the best match is relevant, because in automated process execution only this service will be called as part of a workflow. These searches apply to both, necessary service discovery operations, e.g. [5], and the actual selection of a semantically best matching service like in [6].

Static scenario. To evaluate how efficient the index works and how fast it builds up, we started with a static scenario where all peers are set up before the simulation starts. This scenario has practical applications e.g. for a network of service registries where registry nodes join and leave very rarely.

Dynamic scenario. Obviously, we were also interested how good the algorithm works in a volatile network, the typical P2P case. To simulate volatility, we gave each peer a lifetime (with normal distribution). After its lifetime, the peer leaves the network, and soon afterwards a new peer with new documents joins. The lifetime was adjusted so that during a simulation run about 20% of the peers left the network and were replaced by new peers. In this scenario we need to update the indexes. As described in section 3, each index entry gets an expiration time on creation, based on a specified expiration period. This "best before" date will guarantee that our indexes do not age too much and provide out-dated information.

For our experiments we varied the network size between 100 and 2000 peers (connected via 2 to 16 super-peers), each holding 50 documents on average (with standard deviation 10). We used LA Times articles from the TREC-CD 5. The documents were distributed randomly among the peers. This is obviously the worst case for our approach: when documents are clustered on peers, the number of addressed peers can be essentially reduced. However, appropriate clustering algorithms are out of the scope of this paper, and the quality of our algorithms can be shown even in this worst case scenario (which basically affects only the number of super-peers contained in the indexes negatively).

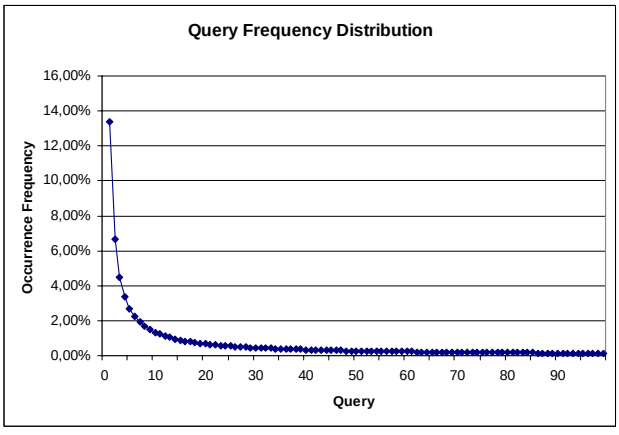

Fig. 3: Query Distribution

We posed keyword queries with query term count average 2.0 and standard deviation 1.0. For these queries, we selected terms from the documents randomly. We assume a Zipf-shaped query frequency distribution with skew-factor 1.0 (see Fig. 3). The queries were pre-generated for occurrence frequencies > 1%. Below this threshold queries were generated randomly with uniform distribution. The originating peer was selected randomly.

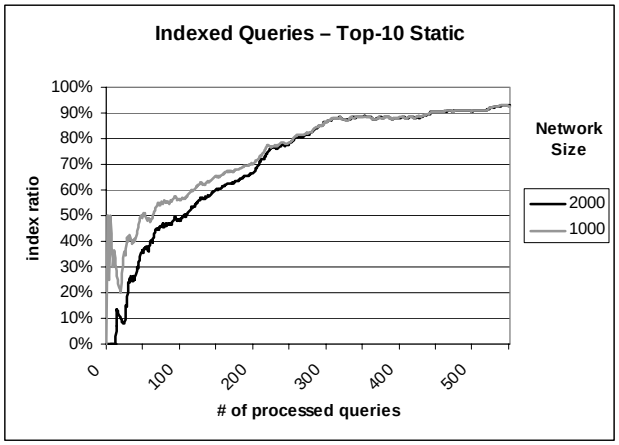

Fig. 4: Index Development (first 550 queries)

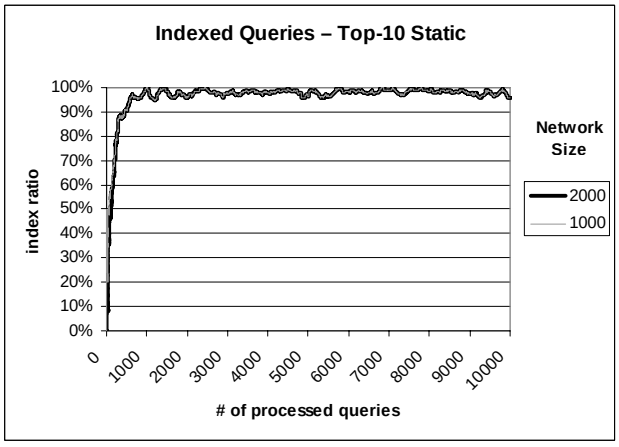

Fig. 5: Index Development (10000 queries)

5.3 Simulation Results

Static scenario. Fig. 4 shows how the index develops as more and more queries are processed by the network. The simulation starts with an empty index. As described in section 3, each query is added to the index on closing of the transaction. Thus the number of already indexed queries increases continuously. If a new query is sent, the algorithm checks if it is already in the index. The following figures show how many of the queries were already in the index (as floating average with an interval of 200). The index coverage increases quickly. After 550 queries more than 90% of all queries are found in the index. After about 550 queries the index ratio stays rather constant, with some random variations, as shown in Fig. 5.

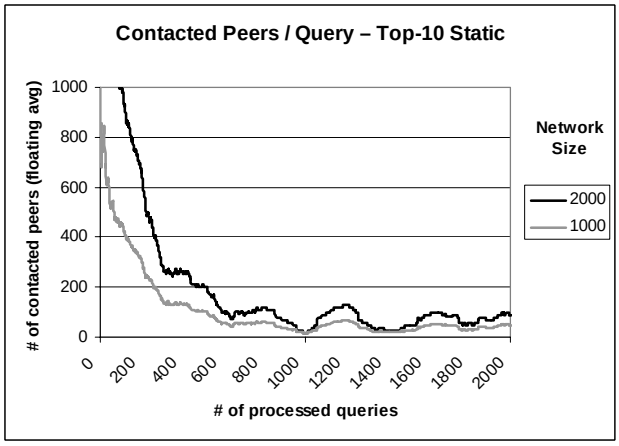

Fig. 6: Contacted peers per query

The rapid build up of the index corresponds to a significant reduction of contacted peers (see Fig. 6). After 1000 queries the average number of contacted peers has become relatively constant (from 10 for 100 peers to 49 for 2000 peers). Therefore we show only the first 2000 queries here. The top-1 case shows the same trend. As can be expected, the average of contacted peers is a little lower (after 2000 queries it ranges from 5 for 100 peers to 41 for 2000 peers). However, our simulation results show a smaller difference than could be expected. The number of messages sent to evaluate a query is proportional to the number of contacted peers. We illustrate this in Fig. 7 for the top-1 static case.

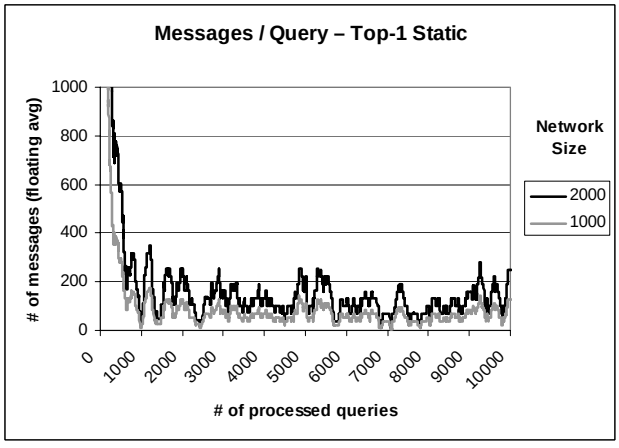

Fig. 7: Messages per query (top-1, static)

Dynamic scenario. Fig. 8 shows the average number of contacted peers for the top-1 dynamic case. While it is slightly higher than the top-1 static results, the general index performance is not significantly affected by the introduction of an expiration period for the index entries

forcing periodic broadcasts for all indexed terms to get a new snapshot of the changed P2P network in the index. On average the number of contacted peers in the dynamic case will increase by a few percent, e.g. in the top-1 case the difference between static and dynamic is only 6%.

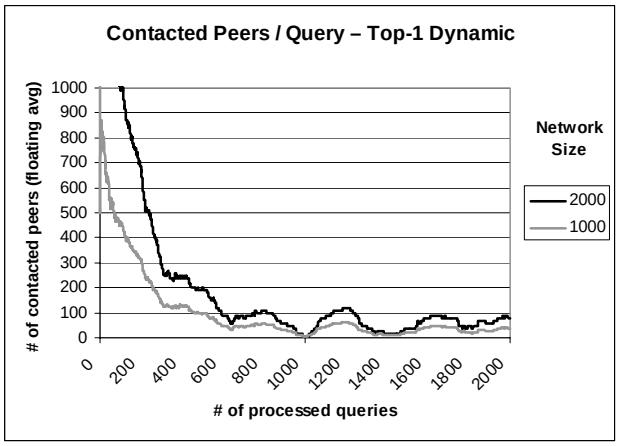

Fig. 8: Contacted peers per query (top-1, dynamic)

The curves exhibit a slight waveform. Besides random fluctuations this is caused by the way the expiration works. Frequently posed queries are indexed shortly after simulation start. They expire all in the same time frame, causing the first broadcast wave. Over time, these waves get less significant due to the probabilistic query frequency distribution. In reality, such waves wouldn't occur because node number and query frequency would grow over time instead of starting off with a fixed amount, thus also causing a less steep increase of index content.

In Table 1 we show for how many queries the computed result differs from the 'perfect' result (which is obtained by evaluating the same query against the joint document collection of all currently existing peers). Though network changes can temporarily invalidate index entries, this is corrected as soon as affected entries expire. Our experiments show the ratio of incorrect results to remain sufficiently low, rising only slightly with network sizes. Please note that we counted *all* differences between our and the perfect result as being 'incorrect' retrievals. However, such incorrect results may still be quite good matches, just not the best possible as given by centralized approaches using expensive broadcasts.

Network Size	Results		
	All	Incorrect	Ratio
100	5000	91	1.82%
500	5000	109	2.18%
1000	5000	151	3.02%
2000	5000	174	3.48%

Table 1: Ratio of incorrect hits

Network Size	Contacted Peers	
	Estimated (n_{SP} + k)	Measured
100	2 + 10 = 12 = 12%	10 = 9.7%
500	4 + 10 = 14 = 2.8%	19 = 3.8%
1000	8 + 10 = 18 = 1.8%	29 = 2.9%
2000	16+10 = 26 = 1.3%	49 = 2.5%

Table 2: Estimated vs. measured contacted peers

Lessons learned. Coming back to our two initial hypotheses our experiments allow to state the following confirming results:

1. Table 2 presents the expected and actual ratio of contacted peers for the top-10 case. The last column contains the average contacted peers of queries 2001 to 10000. The first 2000 queries are regarded as initial phase until the simulation stabilizes and are thus left out. It shows that we do not fully reach the estimated numbers of contacted peers. The reason is because, though a large percentage of queries gets indexed very quickly, there constantly remains a small amount of queries that still have to be broadcasted. The impact of this phenomenon depends on the network size. It is caused by our query generation strategy which (following the assumption of a Zipf distribution) creates random queries for occurrence frequencies below 1% (see 5.1). Still the ratio of contacted peers/network size is very low, and the number of connected peers increases sub-linearly with increasing network size.
2. The assumption that our approach works nearly as well in a volatile network as it does in a static network can also be confirmed, as we have shown by the results in our dynamic scenarios. Though in our practical tests we assumed quite a high volatility (20% of the network dropping out and being replaced), in all cases the number of contacted peers increased only by a few percent over the static case, and remained relatively stable during the simulation. Our indexing scheme is thus applicable for most practical environments.

6. Summary and Outlook

In this paper we have in detail discussed the benefits of best match/top-k queries for distributed peer-to-peer information infrastructures. So far this important kind of queries had mainly been investigated for traditional centralized information systems and web search environments, We have described an innovative routing index-based retrieval algorithm, which implements distributed processing of top-k queries in general peer-to-peer networks while maintaining a minimum of object data traffic. In order to achieve scalability also for large peer-to-peer networks, our algorithm is based on dynamically collected query statistics only, and does not rely on ex-

plicit index updates every time new peers and data are dynamically added to or drop out of the P2P network. Moreover, we have proven that our algorithm always delivers correct result sets and is optimal in terms of necessary object accesses and data traffic. Finally, we have presented several simulations of our approach, which show the quality of our algorithm regarding index construction and updates in static and dynamic environments.

Interesting additional problems to explore in future research are the local processing of retrieval tasks involving collection-wide information and the exploitation of semantic structures like clusters of peers providing semantically similar objects or documents. The clustering of such peers with semantically similar content (as for example given in distributed digital library scenarios) can be expected to further improve the overall performance of our retrieval algorithm and enable flexible and efficient wide area document sharing applications without the maintenance of central indexing servers. Another interesting area will be the generalization of our optimality results from the minimum number of object accesses to a minimum network traffic, where administrative messages can be saved, if a set of objects can be fetched by each *get_next()* operator. Heuristically managing the resulting trade-off between saved messages by pre-fetched relevant result objects, and unnecessary transmissions of objects irrelevant for the result set, can probably optimize the overall network traffic.

7. Acknowledgements

Part of this work was generously funded within the Emmy Noether program of the German Research Foundation (DFG). We also want to thank Tobias Buchloh for his significant contribution to the simulation development.

8. References

[1] K. Aberer, P. Cudré-Mauroux, M. Hauswirth: The Chatty Web: Emergent Semantics Through Gossiping. *Intern. World Wide Web Conf.,* Budapest, Hungary, 2003.

[2] K. Aberer, J. Wu: A Framework for Decentralized Ranking in Web Information Retrieval. *Asian-Pacific Web Conf. on Web Technologies and Applications*, Xian, China, 2003

[3] Adamic, L.A., Huberman, B.A.: Zipf's law and the internet. *Glottometrics 3*, 2002.

[4] W.-T. Balke, U. Güntzer, W. Kießling: On Real-time Top k Querying for Mobile Services. *Intern. Conf. on Cooperative Information Systems*, Irvine, USA, 2002.

[5] Wolf-Tilo Balke, Matthias Wagner. Cooperative Discovery for User-centered Web Service Provisioning. *Intern. Conf. on Web Services,* Las Vegas, USA, 2003.

[6] Wolf-Tilo Balke, Matthias Wagner. Towards Personalized Selection of Web Services. *Intern. World Wide Web Conf.,* Budapest, Hungary, 2003.

[7] I. Brunkhorst, H Dhraief, A. Kemper, W. Nejdl, C. Wiesner: Distributed Queries and Query Optimization in Schema-Based P2P-Systems. *Intern. Workshop on Databases, Information Systems, and Peer-to-Peer Computing,* Berlin, Germany, 2003.

[8] N. Bruno, L. Gravano, A. Marian: Evaluating Top-k Queries over Web-Accessible Databases. *Intern. Conf. on Data Engineering*, San Jose, USA, 2002.

[9] J. Chomicki: Querying with intrinsic preferences. *Intern. Conf. on Advances in Database Technology*, Prague, Czech Republic, 2002.

[10] J. Cowie, H. Liu, J. Liu, D. Nicol, A. Ogielski.: Towards realistic million-node internet simulations. *Intern. Conference on Parallel and Distributed Processing Techniques and Applications,* Las Vegas, USA, 1999

[11] F. Cuenca-Acuna, C. Peery, R. Martin, T. Nguyen: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. *Intern. Symp. on High-Performance Distributed Computing,* Seattle, USA, 2003.

[12] R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware. *ACM Symp. on Principles of Database Systems*, Santa Barbara, USA, 2001.

[13] U. Güntzer, W.-T. Balke, W. Kießling: Optimizing Multi-Feature Queries for Image Databases. *Intern. Conf. on Very Large Databases,* Cairo, Egypt, 2000.

[14] A. Halevy, Z. Ives, P. Mork, I. Tatarinov: Piazza: Data Management Infrastructure for Semantic Web Applications. *Intern. World Wide Web Conf.*, Budapest, Hungary, 2003.

[15] W. Nejdl, W. Siberski, M. Sintek: Design Issues and Challenges for RDF- and Schema-based Peer-to-peer Systems. *SIGMOD Records,* 32(3), ACM, 2003.

[16] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, T. Risch: EDUTELLA: a P2P Networking Infrastructure based on RDF. *Intern. World Wide Web Conf.*, Honululu, Hawaii, USA, 2002.

[17] W. Nejdl, M. Wolpers, W. Siberski, A. Löser, I. Bruckhorst, M. Schlosser, C. Schmitz: Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. *Intern. World Wide Web Conf.*, Budapest, Hungary, 2003.

[18] M. Schlosser, M. Sintek, S. Decker, W. Nejdl. HyperCuP – Hypercubes, Ontologies and Efficient Search on P2P Networks. *Intern. Workshop on Agents and P2P Computing*, Bologna, Italy, 2002.

[19] W. Siberski, U. Thaden: A Simulation Framework for Schema-based Query Routing in P2P Networks. I*ntern. Workshop on Peer-to-Peer Computing & Databases*, Heraklion, Crete, Greece, 2004.

[20] B. Yang, H. Garcia-Molina: Designing a super-peer network. *Intern. Conf. on Data Engineering*, Bangalore, India, 2003.