

# Search for the Best but Expect the Worst - Distributed Top- $k$ Queries over Decreasing Aggregated Scores

Sebastian Michel  
Max-Planck-Institut Informatik  
Saarbruecken, Germany  
smichel@mpi-inf.mpg.de

Thomas Neumann  
Max-Planck-Institut Informatik  
Saarbruecken, Germany  
neumann@mpi-inf.mpg.de

## ABSTRACT

We consider distributed top- $k$  queries in wide-area networks where the index lists for the attribute values (or text terms) of a query are distributed across a number of data peers. In contrast to existing work, we exclusively consider distributed top- $k$  queries over decreasing aggregated values. State-of-the-art distributed top- $k$  algorithms usually depend on threshold propagation to reduce expensive data access across the network, but fail to compute tight thresholds if the aggregation function is decreasing. Decreasing aggregation functions, however, occur naturally, for example when considering conjunctive queries. Our proposed algorithms allow for efficient execution of these kind of queries, using a combination of threshold propagation and semijoin techniques. We demonstrate these techniques for the problem of top- $k$  peer selection in a Peer-To-Peer Web search engine. Our experimental results on real-world data shows the superiority of our approach over pure thresholding.

## 1. INTRODUCTION

Distributed top- $k$  query processing has recently become an essential functionality in a large number of emerging application classes like Internet traffic monitoring and Peer-to-Peer Web search. This paper addresses efficient algorithms for distributed top- $k$  queries in wide-area networks where the index lists for the attribute values (or text terms) of a query are distributed across a number of data peers. In contrast to existing work, we exclusively consider distributed top- $k$  queries over decreasing aggregated values.

As a real-world application of these special kind of aggregation functions, we consider top- $k$  peer selection in P2P search engines.

Consider peers in a wide-area-network maintaining lists of per-peer for particular keys (aka. attributes) that describe the suitability of a particular peer w.r.t. a particular term. The mapping of a term to the corresponding lists is predetermined by an underlying *distributed hash table* (DHT), such

as Chord [31], P-Grid [1], or Pastry [29]. This DHT-based directory builds, roughly speaking, the core of our P2P Web search engine, coined Minerva [5]. The entries in these so called *PeerLists* consist not only of simple scores but of more complex data synopses, that describe the suitability of peers w.r.t. to terms. The goal is to select  $k$  most promising peers for a given (multi-term) query.

Figure 1 shows an example of two *PeerLists* describing published peer-descriptions for term 1 and term 2. The single entries inside the PeerLists are the sharing-granule of our system. Speaking in top- $k$  query processing terminology, the per-term PeerLists represent the per-term index-lists. As shown in Figure 1, the entries inside a PeerList consist of contact-information like IP and port, a time-to-live tag, quality of service information like the document frequency (w.r.t. a term), and a hash-sketch [15] or any other compact data-synopses suitable to describe a set of documents.

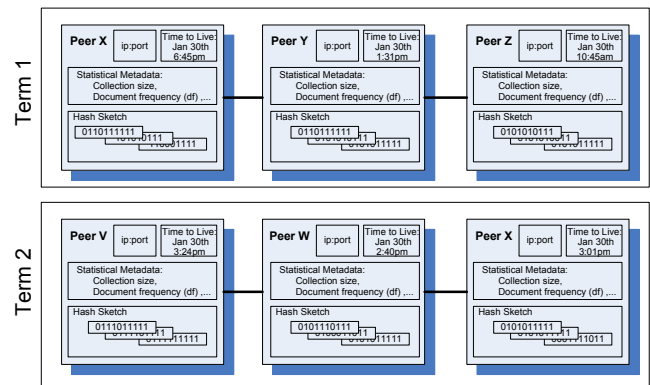


Figure 1: PeerList Example

Traditional peer selection strategies [10, 17, 16] assume that the peers inside a per-term *PeerList* can be ordered by some score (e.g., by the number of documents for a particular term) and apply an aggregation function (usually a sum) to determine the score for the whole query. However, selecting peers based on the sum of per-term scores is a crude and usually flawed approximation. Consider for instance the query “native”, “American”, and “music”. Identify the best peers for each of the keywords separately, and finally combine them (e.g., by some form of aggregating the summary scores) may lead to mediocre query results as the best peers for the entire query may not be among the top can-

didates for the individual keywords. For a given query with  $m$  terms  $t_1, t_2, \dots, t_m$  we try to identify  $k$  peers that have the highest number of documents that contain *all* of the query terms. We are interested in the top- $k$  peers w.r.t. the quantity  $|D_{t_1} \cap D_{t_2} \cap \dots \cap D_{t_m}|$  where  $D_{t_i}$  denotes the set of documents that contains term  $t_i$ . Admittedly, this “scoring model” does not consider the quality but only the quantity to assess the suitability of a peer. However, combining *df*-based measures with other IR-techniques to increase the expected result quality has been well addressed in the literature. Any further investigation is out of the scope of this paper.

The limitation to *per-key peer summaries* seems unavoidable, for statistics on all keyword pairs would incur a quadratic explosion and a challenging issue of distributed parameter estimation over a very-high-dimensional and extremely sparsely populated feature space, leading to a breach with the goal of scalability.

To model the document set  $D_t$  for term  $t$  we let peers include in their posts compact data synopses [4, 24], e.g. hash-sketches [15] or min-wise independent permutations [7, 9]. Using these synopses, we are able to estimate the value of  $|D_{t_1} \cap D_{t_2} \cap \dots \cap D_{t_m}|$  for each peer. These data-synopses can then, in addition, be used for estimating the *global document frequency* [4] or to select peers in an overlap-aware way [24].

Note, that there is actually no static ranking inside the per-term lists since a synopsis by itself cannot describe the suitability for the complete query. However, sorting the lists by the number of documents (i.e. document frequency (*df*)) is sufficient for pruning purposes. The *df* can be either obtained from the hash-sketches or explicitly included in the published information.

## 1.1 Problem Statement

We consider a query  $Q$  given by  $m$  attributes  $\alpha_1, \alpha_2, \dots, \alpha_m$ . Like in traditional top- $k$  aggregation queries we are interested in the  $k$  items (e.g., documents) that have the highest aggregated score. The index list for each attribute  $\alpha_i$  is stored on peer  $P_i$ . An index list  $L_i$  for attribute  $\alpha_i$  consists of (id, score)-pairs where *id* is the globally unique identifier of the item and *score* is the value of the item w.r.t. the attribute  $\alpha_i$ . As in prior work, we consider the index-lists to be sorted by score descending. Scores are real-valued, non-negative numbers. We consider aggregation functions with the following property:

$$\forall_{1 \leq i \leq m} : \text{aggr}(S_1, \dots, S_n) \leq S_i$$

Typical examples for this kind of aggregation function are min, or set intersection.

The top-1 query (the generalization to top- $k$  is straight forward but lengthy) can then be formulated as

$$\arg \max_{id} \text{aggr}(L_1[id].\text{score}, \dots, L_m[id].\text{score})$$

Note that this implies that conceptually each list contains scores for all items. For items which are not contained in a list we assume an implicit score of 0 for this lists (which results in an aggregated score of 0 due to the decreasing aggregation).

As a consequence, we cannot decide if an item has a score  $> 0$  unless we have seen it in all lists. This breaks the existing distributed top- $k$  algorithms, as they cannot decide

early about the items which can be removed. We will discuss this in Section 3.

## 1.2 Contribution

In this paper we make the following contributions.

1. We consider the special case of distributed top- $k$  queries over decreasing aggregated values and show why existing state-of-the-art approaches fail in this scenario.
2. We present algorithms that meet these special requirements.
3. As a showcase for the practical importance of the presented algorithms, we show that this problem is of fundamental importance for the peer selection in P2P search engines and present a comprehensive performance study showing the superiority of our algorithms.

To our knowledge, this is the first work that addresses these issues.

This paper is organized as follows. Section 2 discussed related work. Section 3 illustrates the basic idea of our algorithms, which are explained in Section 4. Section 5 explains why considering set-intersections is desirable for a meaningful peer selection. Section 6 reports on the performance evaluation. Section 7 presents an outlook to interesting problems and their solutions. Section 8 concludes the paper.

## 2. RELATED WORK

Top- $k$  query processing has received much attention in a variety of settings such as similarity search on multimedia data [26, 13, 18], ranked retrieval on text and semi-structured documents in digital libraries and on the Web [22, 32, 3, 28], and network and stream monitoring [2, 21, 11]. Within this rich body of work, the *TA* (threshold algorithm) family for monotonic score aggregation [14, 18, 26] has proven to be an extremely efficient and highly versatile method. It comes in variants with sequential scans of index lists only (NRA), or with a flexible combination of sorted and random accesses (CA).

The first distributed TA-style algorithm for top- $k$  queries over Internet data sources has been proposed by Bruno et al. in [8]. This hybrid algorithm allows both sorted and random access to input lists but tries to avoid random accesses depending on the access costs and limitations of the data sources. Scheduling strategies for random accesses to expensive data sources were also addressed also [12] for a setting with centralized sorted accesses. Zhang and Suel [34] consider distributed variants of TA with sorted accesses only, continuously sending parts of the lists between the network nodes until the top- $k$  answers have been found, where the number of communication steps is limited only by the size of the shortest list. Cao and Wang [11] propose TPUT, an algorithm that calculates the exact top- $k$  answers in exactly three phases:

- Phase 1: The query initiator  $P_{init}$  retrieves the top  $k$  entries from each list and aggregates the scores. *min-k* denotes the score of the document currently at rank  $k$ .
- Phase 2:  $P_{init}$  sends the *min-k/m* threshold to the involved peers which send back all (itemId, score)-pairs with *score*  $>$  *min-k/m*.

- Phase 3:  $P_{init}$  retrieves all missing scores for the candidate items.

Our own work on distributed top- $k$  algorithms [25] makes a strong case for approximate top- $k$  algorithms. It is similar to [11] but uses in addition compression and filtering techniques.

### 3. IDEA

The main goal of distributed top- $k$  processing is pruning items as early as possible, as network transfer is expensive. A related problem is detecting when the correct top- $k$  entries have been discovered and the algorithm can stop (related, as it implies that all other items have been pruned). The standard technique to do this is to maintain *bestscore* and *worstscore* bounds, containing the best and worst score an item can still get during continued processing. These bounds allow to determine which items dominate other items and therefore allow for early pruning. In the centralized case this pruning can be done directly, while for the distributed case the bounds are propagated as score thresholds to the involved peers. This transformation into a range-query is preferable since it greatly reduces the number of network round-trips.

For decreasing aggregation functions, the bestscore is (at most) the minimum of all partial scores seen so far, while the worstscore is zero until the item has been fully evaluated. As a consequence, no items can be pruned before  $k$  items have been fully evaluated, which can take some time when scanning lists sequentially. Existing algorithms like TPUT [11] assume that they can derive a suitable threshold from the first  $k$  entries from each list, which is not the case here.

So the goal is to get fully evaluated items early. A simple idea is to look up missing partial scores by explicitly issuing random lookups at the relevant peers. These *random accesses* provide fully evaluated items, which can then be used for pruning. Unfortunately random accesses are relatively expensive due to the latency involved, and may not even help: To be useful for pruning, it is not enough to get  $k$  fully evaluated items, but  $k$  fully evaluated items with an aggregated score  $> 0$ . Just issuing random accesses may therefore not help for pruning, besides being expensive.

We therefore propose *three key concepts*:

- The random accesses have to be piggy-backed on sequential accesses. During sequential reads, the query coordinator can issue interesting items that will then be included in the sequential access answer, reducing latency costs.
- Range queries are preferable over direct sequential or random access to the data, again due to latency costs.
- Finally, when all candidates from a list have been seen, all top- $k$  candidates have been seen (due to the decreasing aggregation), which allows for semijoin like pruning of lists.

We discuss these concepts together with appropriate algorithms in the next section.

### 4. ALGORITHMS

We now discuss three algorithms with an increasing level of complexity. We start with a more classical top- $k$  algorithm (adjusted to batched accesses due to the distributed

nature of the query) and improve it to handle decreasing aggregation functions efficiently.

#### Algorithm 1: Batched SA

The first algorithm retrieves all items using sequential accesses (SA). It requests batches of size  $k$  from every peer (sorted by decreasing local score), aggregates them and updates the worst score/best score bounds for the candidates seen so far. It stops querying a peer if the local score at that peer drops below the current *min-k* (pruning all candidates that have not been seen on this peer, as they must have score below *min-k*), and terminates if no candidates can make it above *min-k* any more. Note that there is a fundamental difference to "normal" top- $k$  processing: The worst score of each candidate is zero until the candidate has been fully evaluated (as is can get arbitrarily bad from yet unseen list entries). In particular, *min-k* is zero until the algorithm has fully evaluated  $k$  entries. This can require scanning deeply into the lists, and no pruning is possible before.

#### Algorithm 2: Batched SA/RA

The second algorithm reduces this problem by integrating random accesses (RA) into the sequential access. When requesting a new batch from a peer, it piggy-backs the item ids of all candidates not yet seen on this peer into the message. The peer looks up the local scores of these items and includes them in the answer batch. Thus, the algorithm get fully evaluated items much quicker, which allows for raising the *min-k* threshold, pruning candidates earlier.

#### Algorithm 3: Batched SA/RA with Range Queries

Both of the previous algorithms require multiple round-trips over the network to fetch the result batches. This causes latency and is relatively slow. It has been shown in prior work [11, 25] that in a distributed setting range queries are usually faster than such fine-grained data accesses. Unfortunately, an algorithm like TPUT [11] is not directly applicable here, as the *min-k* threshold after the first  $k$  items will most likely be zero, resulting in a range query over the whole data set.

Instead, we start with the second algorithm and use a range query when possible: We run the second algorithm until we have  $k$  fully evaluated candidates with a non-zero score (and thus a non-zero *min-k*), and then switch to a range query over all items with a local score  $\geq \text{min-k}$ . This gives us all relevant candidates and avoids the costly round trips. To improve query performance, we send asynchronous messages to the peers since *min-k* increases while aggregating results, thus shrinking the range over time. Note that in contrast to TPUT we do not require a final random access phase: All candidates that are not fully evaluated after the range query are known to have a score below *min-k*.

We can prune even more efficiently by using the conjunctive nature of the query for a semijoin technique [20]: When we have read any list down to *min-k*, we know that we have seen all candidates (as all not seen on the list must get a score below *min-k*). Thus, when a range scan finishes, we construct the set of remaining candidates and send it as a Bloom filter [6] asynchronously to the remaining lists. When the Bloom filter arrives, the peers can skip over all items in the remaining range not matched by the Bloom filter. Note that the filter only contains the (typically few) remaining top- $k$  candidate and is therefore much more accurate than just building a filter over the whole list (in our experiments,

many of the lists contain the same items).

## 5. ESTIMATING INTERSECTIONS

A major motivation for decreasing aggregation functions is set intersection. Therefore we give a short introduction into the estimation and usage of set intersection here.

As already mentioned, we assume that peers include in their published information (cf. Figure 1) Hash Sketches describing the content of their local collections, i.e., they “insert” in the Hash Sketch the identifiers of their local documents that contain a particular term. Thus, these content descriptions are on a per-term basis.

Hash sketches were first proposed by Flajolet and Martin in [15], to probabilistically estimate the cardinality of a multiset. Recall that the overall goal is to select peers based on their contribution to the whole query, i.e., the number of documents they can deliver.

Fortunately, Hash Sketches have a nice property: Given a Hash Sketch for a set  $A$  and a Hash Sketch for a set  $B$  we can simply create the Hash Sketch that describes the set  $A \cup B$  by combining the bitmaps from the single Hash Sketches via a bitwise-OR operation. Thus, since we have a way to obtain the Hash Sketch for  $A \cup B$ , we can estimate the cardinality of the union, i.e.,  $|A \cup B|$ . Then, we can calculate  $|A \cap B|$  using  $|A \cap B| = |A| + |B| - |A \cup B|$ . This can be generalized to more than two sets, using the inclusion-exclusion principle and the sieve formula by Poincaré and Sylvester.

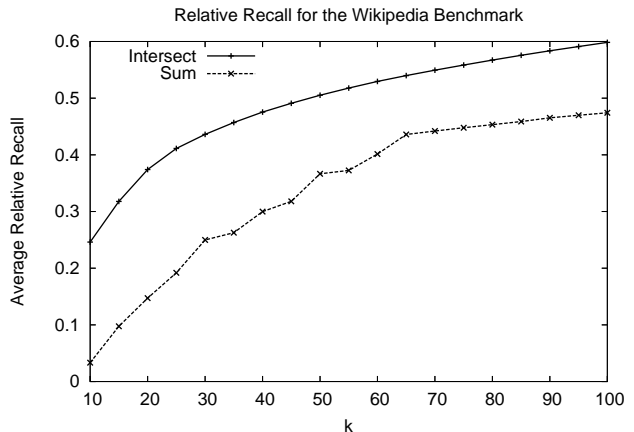
Once we know the Hash Sketch of peer  $P$  for terms  $t_1$  and  $t_2$ , we can estimate the number of documents in  $P$ ’s collection that contain both  $t_1$  and  $t_2$ .

Recall that the overall objective is to select the most promising peers for a query with respect to the number of documents the peer can deliver for the query, i.e., number of documents that contain both terms. In prior work [23] we have considered the usage of multi-key statistics, i.e., let peers publish statistics about documents that contain particular key-sets. These key-sets of interest have been learned through query-log analysis. As also shown in [23], extending the single-key statistics with Hash Sketches can lead to drastic performance gains as well.

Figure 2 shows the average relative recall for different numbers of asked peers (i.e.,  $k$ ) for the Wikipedia benchmark which we will also use later when comparing the efficiency of our algorithms. The relative recall is the fraction of obtained documents that are also among the top documents when executing the queries on a hypothetically combined collection (i.e., over all peers). Obviously, the relative recall increases when involving more and more peers in the query execution. As shown in Figure 2, the “intersection” based scoring-model achieves a much higher relative recall than the simple sum based model. This holds throughout all  $k$  values, but is in particular impressive for small values of  $k$ . For instance, when considering the top-10 peers, the intersection based technique achieves a relative recall of 24.6% whereas the sum based techniques achieves 3.3%.

## 6. EVALUATION

### 6.1 Setup



**Figure 2: Comparison of the relative recall performance of two different peer-scoring functions: “intersect” measures the suitability of a peer w.r.t. a query by the number of documents that contain all query terms, whereas “sum” considers only the sum of the single document frequencies.**

### Cost Model

For the sake of reproducibility we have opted for modeling the network component. As we consider distributed query processing in wide-area-networks, we assume a network latency of  $50ms$  and a network bandwidth of  $800kb/s$  [33, 30]. Each index-list entry is assumed to be  $1024$  bytes, i.e., the size of a reasonably sized hashsketch plus an IP address to locate the peer which has published the description.

### Measures of Interest

**Number of transferred items:** We report on the number of transferred index-list entries during query execution, i.e., the overall number of transferred items that does not consider parallelism.

**Query response time:** This is the time the query initiator has to wait until the top- $k$  query execution is finished. In opposite to the number of transferred items, the query response time does consider parallelism during query execution.

### Dataset and Queries

We use the publicly available benchmark as proposed in [27]. It is based on Wikipedia articles and the assignment of documents to peers is well specified by a graph-based clustering strategy. The clustering is an approximation to a crawler behavior and produces topic specific clusters. The documents in each topic are then clustered into smaller related chunks, using the same clustering algorithm on each topic cluster. These *chunks* are assigned to peers using a sliding window technique: Each peer is assigned a certain number of chunks and the next peer shifts the window a certain number of steps, creating any desired degree of overlap. For this paper we have created 1000 peers (collections). As for the query workload, we use queries taken from Google’s Zeitgeist query collection as proposed in [27] but consider only those queries with at least two terms. Considering also

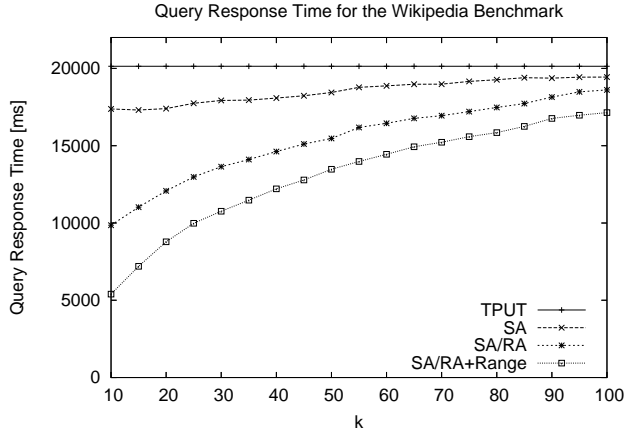


Figure 3: Average query response time for different values of  $k$

the single-term queries would render a meaningful performance comparison difficult since top- $k$  aggregation queries over only one index-list are trivial.

### Algorithms under Consideration

As for the performance comparison, we consider a slightly modified version of TPUT [11] as a baseline. TPUT has to be adjusted to decreasing aggregation functions, as the threshold computation changes. We compare it to the algorithms from Section 4, which represent increasing levels of refinement.

All algorithms are exact, i.e., calculate the exact top- $k$  result. Therefore, there is no need to report on recall.

### 6.2 Experimental Results

Figure 3 shows the average query response time for different values of  $k$ . It is easy to see that TPUT performs worst for all values of  $k$  (x-axis). However, as  $k$  becomes larger, the performance differences between the different algorithms becomes smaller and smaller. This is due to the fact that for large  $k$  nearly the whole lists have to be read, simply because  $k$  is a significant fraction of the total list length (the average list length is 881). SA/RA performs much better than SA as it gets fully evaluated candidates much quicker and can thus prune and stop earlier. The SA/RA algorithm with range queries is superior to the other competitors. The large gain in query response time compared to the small gain in number of retrieved items (cf. Figure 4) stems from the fact that the range queries create less latency than the batched accesses used by SA and SA/RA.

Figure 4 reports on the average number of transferred items for different values of  $k$  (x-axis). Here, again, TPUT is clearly outperformed by the other competitors. SA/RA with range queries is the clear winner and SA performs worse than SA/RA. Note that the candidate filtering during the range queries is essential for the low number of transferred items (and thus the overall good performance) of the SA/RA with range queries algorithm. Without it, the range query sends considerable more candidates over the net than SA/RA, canceling the latency gains.

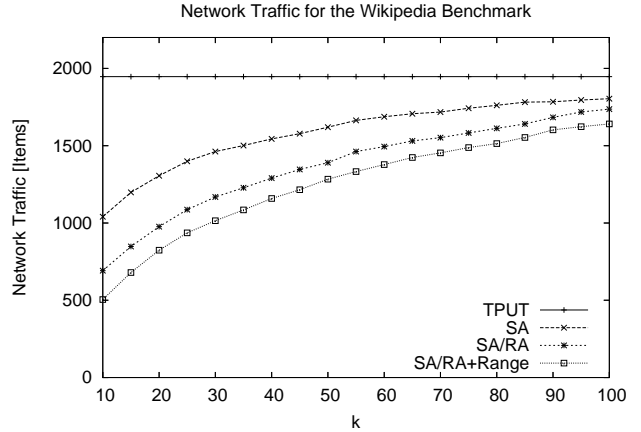


Figure 4: Average number of retrieved items for different values of  $k$ .

## 7. OUTLOOK

The problem of an efficient top- $k$  peer selection algorithm becomes even harder when we consider an overlap aware peer selection. In recent work on query routing [19, 24] it has been shown that it is beneficial to consider the mutual peer overlap as a selection criteria, i.e., the task is not only to select the top- $k$  peers according to some quality measure, but also to consider the “novelty” of a peer. For instance, in a non-overlap aware result, the peer at rank 4 might be completely useless since its document collection is a subset the document collection provided by the peer at rank 3. Thus, the usefulness of a peer cannot be expressed as a static score (or hashsketch) but depends on the peers selected so far (at a higher rank), i.e., on the current state of the algorithm.

The novelty of a peer, given its Hash Sketch, can be calculated if there is a reference Hash Sketch. The reference Hash Sketch would be continuously updated during query processing and sent to the peers that maintain the index-lists. These peers have to re-rank their index-list-entries according to the new reference. Choosing an appropriate batch size for the communication is a nice optimization problem that we will address in future work.

One problem we did not consider so far is the appropriate choice of  $k$ .  $k$  determines the number of peers to ask. A large  $k$  might waste network resources, whereas a small  $k$  could lead to mediocre results as there are too few peers involved in the query. Fortunately, as we already deal with Hash Sketches that describe the peers’ contributions, we can predict the relative contribution of the peers selected so far with the expected global number of results. This allows for a prediction of a suitable  $k$  w.r.t. a desired amount of recall that is much easier to define than  $k$ .

Note that the problem of scores that depend on the current state of the algorithms is not a specific problem in our P2P peer selection scenario.

Another interesting application scenario are traditional top- $k$  algorithms like Fagin’s algorithm without random accesses (NRA). The index lists are usually sorted by worst-score (partial-score, lower-bound) descending, but sorting by bestscore (upper-bound) would be beneficial since the most promising candidates would be delivered first. The best-

score, however, is a measure that cannot be pre-computed as it depends on the current state of the algorithm.

We will address these issues in future work.

## 8. CONCLUSION

In this paper we have addressed the problem of conjunctive top- $k$  aggregation queries in distributed systems where the aggregated values are decreasing. We have introduced three algorithms and conducted a performance evaluation on real-world data showing the superiority of our approaches w.r.t. state-of-the-art algorithms. In addition, and as an application scenario, we have addressed the issue of efficiently selecting the top- $k$  peers in a distributed Web search engine where the per-term, per-peer descriptions are distributed on a per-term basis using a distributed hash table.

## 9. REFERENCES

- [1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS*, pages 179–194, 2001.
- [2] B. Babcock and C. Olston. Distributed top- $k$  monitoring. In *SIGMOD*, 2003.
- [3] M. Bawa, R. Jr, S. Rajagopalan, and E. Shekita. Make it fresh, make it quick – searching a network of personal webservers. In *WWW*, 2003.
- [4] M. Bender, S. Michel, P. Triantafillou, and G. Weikum. Global document frequency estimation in peer-to-peer web search. In D. Zhou, editor, *WebDB 2006*, pages 69–74, Chicago, USA, 2006.
- [5] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative p2p search. In *VLDB*, pages 1263–1266, 2005.
- [6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [7] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3), 2000.
- [8] N. Bruno, L. Gravano, and A. Marian. Evaluating top- $k$  queries over web-accessible databases. In *ICDE*, 2002.
- [9] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Trans. Netw.*, 12(5):767–780, 2004.
- [10] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR*, pages 21–28, 1995.
- [11] P. Cao and Z. Wang. Efficient top- $k$  query calculation in distributed networks. In *PODC*, 2004.
- [12] K. C.-C. Chang and S. won Hwang. Minimal probing: supporting expensive predicates for top- $k$  queries. In *SIGMOD*, 2002.
- [13] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1), 1999.
- [14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.
- [15] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
- [16] N. Fuhr. A decision-theoretic approach to database selection in networked ir. *ACM Trans. Inf. Syst.*, 17(3):229–249, 1999.
- [17] L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: Text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.
- [18] U. Guntzer, W.-T. Balke, and W. Kiesling. Optimizing multi-feature queries for image databases. In *The VLDB Journal*, 2000.
- [19] T. Hernandez and S. Kambhampati. Improving text collection selection with coverage and overlap statistics. poster at WWW 2005.
- [20] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.
- [21] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB*, 2004.
- [22] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *VLDB*, 2003.
- [23] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, and C. Zimmer. Discovering and exploiting keyword and attribute-value co-occurrences to improve p2p routing indices. In *CIKM*, pages 172–181, 2006.
- [24] S. Michel, M. Bender, P. Triantafillou, and G. Weikum. IQN routing: Integrating quality and novelty in p2p querying and ranking. In *EDBT*, 2006.
- [25] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top- $k$  query algorithms. In *VLDB*, 2005.
- [26] S. Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *ICDE*, 1999.
- [27] T. Neumann, M. Bender, S. Michel, and G. Weikum. A reproducible benchmark for p2p retrieval. In *ExpDB*, pages 1–8, 2006.
- [28] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society of Information Science*, 47(10), 1996.
- [29] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
- [30] D. Salomoni and S. Luitz. High performance throughput tuning/measurement. [http://www.slac.stanford.edu/grp/scs/net/talk/High\\_perf\\_ppdg\\_jul2000.ppt](http://www.slac.stanford.edu/grp/scs/net/talk/High_perf_ppdg_jul2000.ppt). 2000.
- [31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*. ACM Press, 2001.
- [32] M. Theobald, G. Weikum, and R. Schenkel. Top- $k$  query evaluation with probabilistic guarantees. In *VLDB*, 2004.
- [33] A. Tirumala et al. iperf: Testing the limits of your network. <http://dast.nlanr.net/projects/iperf/>. 2003.
- [34] J. Zhang and T. Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Peer-to-Peer Computing*, 2005.