Processing Top-N Queries in P2P-based Web Integration Systems with Probabilistic Guarantees

Katja Hose Marcel Karnstedt Kai-Uwe Sattler Daniel Zinn
Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

ABSTRACT

Efficient query processing in P2P-based Web integration systems poses a variety of challenges resulting from the strict decentralization and limited knowledge. As a special problem in this context we consider the evaluation of top-N queries on structured data. Due to the characteristics of large-scaled P2P systems it is nearly impossible to guarantee complete and exact query answers without exhaustive search, which usually ends in flooding the network. In this paper, we address this problem by presenting an approach relying on histogram-based routing filters. These allow for reducing the number of queried peers as well as for giving probabilistic guarantees concerning the goodness of the answer.

1. INTRODUCTION

Schema-based Peer-to-Peer (P2P) systems, also called Peer Data Management Systems (PDMS) are a natural extension of federated database systems which are studied since the early eighties. PDMS add features of the P2P paradigm (namely autonomous peers with equal rights and opportunities, self-organization as well as avoiding global knowledge) to the virtual data integration approach resulting in the following characteristics: each peer can provide its own database with its own schema, can answer queries, and is linked to a small number of neighbors via mappings representing schema correspondences.

Though, PDMS are surely not a replacement for enterprise data integration solutions, they are a promising approach for loosely-coupled scenarios at Internet-scale, such as Web integration systems. However, the expected advantages like robustness, scalability and self-organization come not for free: In a large-scale, highly dynamic P2P system it is nearly impossible to guarantee a complete and exact query answer. The reasons for this are among others possibly incomplete or incorrect mappings, data heterogeneities, incomplete information about data placement and distribution and the impracticality of an exhaustive flooding. Therefore, best effort query techniques such as similarity selection and join, nearest neighbor search and particularly top-N operators are most appropriate. By "best effort" we mean, that we do not aim for exact results or guarantees but instead try to find the best possible solution wrt. the available local knowledge. However, even if we relax exactness or completeness requirements we still need estimations or predictions about the error rate. For a top-N query this means for example that we can give a probabilistic guarantee that x percent of the retrieved objects are among the top N objects which we would get if we had asked all peers in the system.

Eighth International Workshop on the Web and Databases (WebDB 2005)
June 16-17, 2005, Baltimore, Maryland

Consider a scenario from the scientific domain where a virtual astronomical observatory is built by integrating individual peers (observatories) offering data about sky observations. We assume XML as the underlying data format and XQuery as the common query language. A typical query in this scenario would ask for astronomical objects that match a condition to a certain degree. For instance a researcher could query for objects next to a certain point in space defined by a set of coordinates or objects with an average brightness "similar" to a predefined one.Thus, we can describe the pattern of such queries as follows:

```
for $i in fn:doc("doc.xml")/path
order by rank($i/path) limit N return ...
```

where *rank* is a ranking function defining an order on the elements and **limit** restricts the result set to the first N elements returned by **order by**. The ranking function can be defined simply by exploiting the order of the attribute values or even using the euclidian distance, e.g., for coordinate values. An example from the above mentioned scenario is the following query asking for the 10 stars closest to a given sky position:

```
for $s in fn:doc("sky.xml")//objects
order by distance($s/rascension,
                  $s/declination, 160, 20)
limit 10 return ...
```

In order to allow an efficient evaluation such queries should be treated using a special top-N operator $\hat{\sigma}^N_{rank}(E)$ where E denotes a set of elements. This operator returns the N highest ranked elements from E. The implementation of this operator as well as strategies for processing in a PDMS are the subjects of our work and are described in the following sections.

2. RELATED WORK

Several approaches and efficient strategies for top-N query processing have been developed concerning classical RDBMS. As introduced in [5, 3] one promising approach is to use existing data structures (i.e. indexes) and statistics (i.e. histograms) for mapping a top-N query into a traditional selection query. The aim is to determine a tight n-dimensional rectangle that contains all the top-N tuples for a given query but as few additional tuples as possible thereby dealing with histogram bucket boundaries instead of single tuple values. The algorithm is based on finding a minimal score that is sent along with the query as selection predicate guaranteeing at least k elements that are ranked higher. Another approach that tries to quantify the risk of restarting a query probabilistically is presented in [8], but the authors only deal with the problem of determining an optimal selection cutoff value.

TPUT [4] is an algorithm that was designed for efficiently calculating top-N aggregate queries in distributed networks. This

algorithm represents an enhanced version of the *Threshold Algorithm (TA)* that was independently developed by multiple groups [9, 11]. Further variants of TA have been developed for multimedia repositories [6], distributed preference queries over web-accessible databases [12], and ranking query results of structured databases [1]. Another work [13] that is based on TA introduces a family of approximate top-N algorithms that probabilistically try to predict when it is safe to drop candidate items and to prune the index scans that are an integral part of TA. All these algorithms, however, need several round-trips in order to retrieve the final result whereas our approach tries to ask each peer only once.

The algorithms presented in [2] try to improve top-N query processing by dynamically collecting query statistics that allow for better routing when the query is issued the next time. The first time, however, all peers have to participate in processing the query while several round-trips are required in order to retrieve the final result. This often leads to situations where peer have to wait for each other.

3. EVALUATING TOP-N QUERIES

3.1 Classification

Evaluating top-N queries provides four different main approaches each describing one main principle and thus one general class of more detailed processing techniques. Without loss of generality we assume a tree view on all peers established at the query initiating peer, the so called peer or query tree.

1. Naive top-N: collect N (if possible) data elements from each available peer, sort and evaluate at initiator.
2. Partial top-N: same as naive strategy, but combine and prune results at peers passed on the way back to the initiator.
3. Globally optimized Top-N: minimize the set of queried peers before actually querying them – based on global information, e.g., using a global index structure.
4. Locally optimized Top-N: also minimize the set of queried peers, but decide at each involved peer again which peers to discard, based on locally available information.

Class 1 and 2 algorithms promise poor efficiency. The problem with strategies of class 3 is their need for global information. Global knowledge may not be achievable, because peers do not provide information about all their data. Furthermore, maintenance and building tasks are expensive and top-N queries over the attribute(s) not indexed require flooding the network or an index rebuild (e.g., re-hash). In this work we focus on class 4 approaches, based on knowledge gained from feedback of processed queries. We alternatively call this class locally pruned top-N, because at each peer again we prune the set of possible query paths using only locally available information.

3.2 Histograms and Routing Filters

In order to be able to prune query paths locally we need information about the data distribution at each peer. Approaches based on local index structures have been shown to be suitable ([7]) and we adopted them to our needs of schema and instance level information in previous works. The developed data structures do not simply index on pre-selected attributes, rather they collect information about all attributes occurring with high frequencies, thus we call them *routing filters*. Because histograms are successfully used in a wide variety of optimization and estimation techniques we decided to integrate them into the filters in order to approximate data distribution on instance level.

At each peer for each established connection to a neighbor $neigh$ one separate filter is maintained, which includes schema and instance level information for all peers reachable by querying $neigh$ (that is, all peers "behind" $neigh$). Routing filters are built and maintained using a query feedback approach, thus they are empty when a peer joins and grow as time passes by according to the received query results. If they are not limited to a certain horizon around the owning peer (e.g., by using a hop count [7]) and if we assume that no peer leaves the network, they will finally converge to global knowledge wrt. the query workload.

Histograms approximate data distributions by partitioning a sort parameter into intervals (buckets). An approximated source parameter value (we focus on frequencies) is stored for each bucket. A wide variety of histograms has been developed. We use *V-Optimal histograms*, which partition the sort parameter in such a manner that the variance of source parameter values, i.e., frequencies, within each bucket is minimized [10]. The value domain is approximated by a continuous distribution in the bucket range. Note that our algorithms do not depend on the optimality of V-Optimal histograms. The only thing they rely on is a characteristical value describing the error of the assumed frequency distribution in each bucket. V-Optimal histograms lead to good approximations.

As routing filters and types of histogram are not focus of this work, we omit details. In Figure 1 we picture routing filters exemplarily. In that figure simple histograms on attribute 'x' for two neighbors of the filter owning peer are depicted. In the next section we will show how to exploit the routing filters' information for optimizing top-N query processing and how to provide probabilistic guarantees for the result.

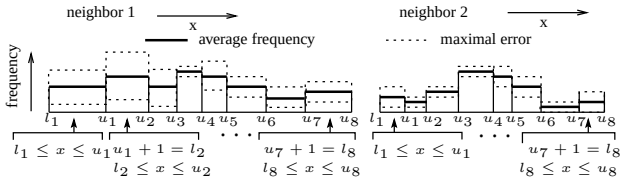

Figure 1: Histograms of routing filters on 'x' for two neighbors

3.3 Histograms and Probabilities

Routing filters combine information about schema and instance level. What is important for optimizing the evaluation of top-N queries is the approximation on instance level provided by the histograms, therefore in the following we focus on these histograms. Moreover, due to the lack of space, we concentrate on the evaluation of only nearest neighbor queries as a prominent representative of top-N queries. The proposed algorithm draws conclusions about the frequency distribution based on the average frequency and the maximum absolute error per bucket. In this subsection we will at first introduce a notation before describing the main algorithm. For simplicity we only consider discrete sort parameters and one-dimensional histograms. Continuous values can either be regarded as special case where values have to be discretized before they can be processed alike or similar algorithms can be developed that are optimized for dealing with such sort parameters. The proposed methodology as described in the following can also be used with multi-dimensional histograms. Evaluating top-N queries based on ranking functions defined over more than one attribute, we have to revert to multi-dimensional histograms. The introduced calculations and approximations hold in analogy.

Notation & Basics Given n data values, represented by function $R : \{1..N\} \rightarrow \mathbb{N}$, we can determine the frequency $F(k)$ for any sort parameter value k with $F(k) = |\{e|R(e) = k\}|$. F's domain is now being partitioned into buckets that can have different sizes.

Several statistical information is assigned to these buckets in order to approximate the original function F. Let bucket B_i be defined as $B_i = \{l_i, \ldots, u_i\}$, where l_i specifies the lower and u_i the upper boundary. For each bucket the average frequency h and the maximum absolute error e are defined as follows:

$$h_i := \frac{\sum_{k=l_i}^{u_i} F(k)}{u_i - l_i + 1} \quad , \quad e_i := \max_{k=l_i..u_i} \{|F(k) - h_i|\}$$

For the rest of this paper we will use the terms T and B_i for symbolizing intervals and buckets as well as for referring to the corresponding set of elements. Considering an arbitrary interval of interest T, $T \subseteq B_i$ and $card(T) := \sum_{t \in T} F(t)$ the following boundary can be identified for the minimum cardinality:

$$card_{min} = \max\{|T| \cdot \lceil h_i - e_i \rceil, |B_i| \cdot h_i - (|B_i \setminus T| \cdot \lfloor h_i + e_i \rfloor)\} \quad (1)$$

A boundary for the maximum cardinality can be defined in analogy. Considering subsets T that are not necessarily limited to one bucket, similar equations can be defined. For any bucket B_j that is partly included in T, $card(T \cap B_j)$ minimum and maximum values can be determined. For any bucket B_j that is completely included in T, we can exactly determine the cardinality via $card(B_j) = |B_j| \cdot h_j$. By summing up these values $card(T)$ can be restricted by minimum and maximum values but usually not determined exactly. However, assuming a probability distribution for $F(i)$ in "cut" buckets, a probability distribution for $card(T)$ can be calculated. Based on this distribution our top-N algorithm that is defined in the following provides probabilistic guarantees.

Specification for Top-N Query Algorithm Our main algorithm takes the following input parameters: the queried value X, the top-number N, a value M_{in} ("missed"), two probabilities P_{in}^C (C for "correct") and P^{OP} (OP for "one pass"), and a distance $dist$ (not provided to, but determined by the initiating peer). It always returns an $(N+2)$ tuple $Res := (data^N, M, P)$[1], which complies with the following statements: With a probability of minimum $P, P \geq P_{in}^C$, the algorithm missed at most M data items – that is the globally correct top-N result based on all data in the network contains at most M data items that are not returned by our algorithm. With a minimum probability of P^{OP}, M is equal to or less than M_{in}. Thus, P^{OP} represents a parameter adjusting the probability of having to re-initiate a query. We handle P^{OP} as an internal system parameter, in our first tests we set $P^{OP} = 0.9$. When re-initiating, P^{OP} must be shifted accordingly. Our algorithm tries to minimize costs by minimizing the number of peers involved for processing the query – with respect to the (probabilistic) boundaries mentioned above.

Problem statement and the algorithm's specification are related as follows: A user initiates a query and provides a required quality guarantee that x percent of the retrieved objects are among the real top-N objects ("real" symbolizes the result we would get if all peers are queried). We reflect x to the value $M_{in} := N - x \cdot N$. Adjusting P_{in}^C provides an additional possibility to affect the accuracy of the retrieved result and therefore the performance of query processing. The algorithm guarantees by a probability of P_{in}^C that the returned result includes at least x percentage of the globally correct result. If not specified, the default value is $P_{in}^C = 1$. P_{in}^C defines the maximum acceptable uncertainness when evaluating the histograms and deciding which peers to discard.

Synopsis for Top-N Query Algorithm In general, all peers execute the following steps:

1. determine the set of neighbors that promises the lowest execution costs while adhering to the provided quality limits
2. distribute the quality limits among the peers to query
3. forward the query with adopted quality limits to the determined subset of peers
4. wait for answers, combine local data and results, set resulting guarantees, return a combined answer to the querying peer

Figure 2 summarizes these steps in an algorithmic form. The interesting details are how to determine the subset of peers to query and how to distribute the required restrictions among these neighbors.

Input: N: number, X: queried value, M_{in}: number, P_{in}^C: probability, P^{OP}: probability, $dist$: number

```
1  if initiating peer then dist = calc-dist(N, X, M_in, P^OP); fi
2  A, dist, P_L^C, M_L = calc-neighs(N, X, M_in, P_in^C, dist);
3  P_ch^C, M_ch = distribute-missed(P_in^C, M_in, P_L^C, M_L);
4  foreach peer a ∈ A do                          /* in parallel! */
5      R_ch, M_ch^a, P_ch^{C,a} = a →process-query(N, X, M_ch, P_ch^C, P^OP, dist);
6  done
7  R_L, M_L, P_L^C = combine(R_L, M_L, P_L^C, R_ch, M_ch^a, P_ch^{C,a});
8  if initiating peer then M_L = adjust-missed(R_L, M_L, dist); fi
9  send-answer(R_L, M_L, P_L^C);
```

Figure 2: Procedure *process-query*

At first, the query initiating peer determines an initial distance $dist$. In order to do so, the locally available histograms are regarded as a view on the global data distribution. Thus, the global range of the queried attribute is expected to equal the maximal range combined over all neighbor histograms and the local data. $dist$ reflects the smallest range around the queried point X in which at least $N - M_{in}$ elements are expected by a probability of P^{OP}. This interval is determined using a sub-algorithm called *calc-dist* and is provided to all subsequently queried peers and used by them for evaluating possible subsets of peers. This is applicable because we assume the initiating peer having gathered global knowledge in the queried attribute's histogram wrt. the established peer tree, and therefore simplifies the following explanations. Different assumptions, e.g., exploiting limited knowledge, result in having to calculate $dist$ repeatedly at each peer and come along with some minor algorithmic modifications. Because each queried peer always returning N elements (if available, all the same if these values lie in interval $[X - dist, X + dist]$ or not), the algorithm works with each $dist$, even if it is 0 or randomly chosen. As we will see, the problem is that we will hardly discard any peers when choosing it too large – in general choosing it too low will result in missing more top elements and having to decrease the returned guarantee value accordingly in order to match the algorithm's specification. Whether probability P_{in}^C holds or not when pruning query paths is only tested in interval $dist$, thus we should choose it appropriately.

In the next paragraphs we will briefly describe each of the subroutines called in Figure 2.

Procedure *calc-dist* An interval I is iteratively and symmetrically widened starting with $I = [X]$. In an iteration step $d = 0, 1, \ldots, \max\{\max_i\{u_i\} - X, X - \min_i\{l_i\}\}$, it is tested whether the probability that at least $N - M_{in}$ data items lie within the interval $I := [X - d, X + d]$ is greater than or equal to P^{OP}. If so, d is returned. Note that the algorithm always ends.

Procedure *calc-neighs* The difficult part is to find all possible subsets of neighbors at each involved peer that fulfill the algorithm's specification in respect to M_{in} and P_{in}^C. For each considered subset A' of all peers A the probability p that the peers in $A \setminus A'$ provide at most M_{in} values inside interval I is determined. If $p \geq P_{in}^C$ holds we may decide to query only the peers from A'. If the final result is completely included in I, discarding these peers does not violate the probabilistic limits. This can only be checked at the initiating peer after the final result is received. If any element

[1]Given that the domain of R includes at least N values

of the result is not included in $[X-dist-1, X+dist+1]$ we have to adjust the returned M_L in order to adhere to the specification of our algorithm, which is done calling *adjust-missed*.

Procedure *distribute-missed* The algorithm allows for missing up to M_{in} relevant data items in $dist$, thus we have to guarantee:

$$P\left(\begin{array}{l}\text{globally discarded peers provide less than or}\\ \text{equal to } M_{in} \text{ values in } [X-dist, X+dist]\end{array}\right) \geq P_{in}^C \quad (2)$$

Enforcing equation 2 and handling final result elements from outside $dist$ accordingly our algorithm works correct as specified.

As the algorithm allows each participating peer to discard some of its neighbors, we have to "distribute" the allowed overall error. If we prune a query path we discard a single set of peers A_i. Assume all globally discarded sets are $A := \{A_i | \forall i \neq j : A_i \cap A_j = \emptyset; i,j = 1,\ldots,v\}$, where v is the number of totally pruned query paths, and M_D is the set of all possible corresponding distributions of M, $M_D := \{\{M_i | M_i \geq 0 \wedge \sum_i M_i = M; i = 1,\ldots,v\}\}$. Let $P_M(A_i)$, respectively $P_{\leq M}(A_i)$, denote the probability that all peers from A_i have exactly, respectively at most, M values in $[X-dist, X+dist]$. Assuming that the missed values are distributed independently from the peers the following equation holds:

$$P_M(\bigcup_i A_i) = \sum_{M_d \in M_D} \prod_{M_i \in M_d} P_{M_i}(A_i) \quad (3)$$

With respect to a certain distribution M'_d of M_{in}, equation 4 can be used to estimate p when distributing data items and probabilities over several peers:

$$p = P_{\leq M}(\bigcup_i A_i) \geq \prod_{M_i \in M'_d} P_{\leq M_i}(A_i) \geq P_{in}^C \quad (4)$$

The right unequal sign exactly is what the algorithm guarantees. We omit the simple proof for the left unequal sign – listing the included factors and summands the principle of proof gets evident. Thus, by ensuring that the product of $P_{\leq M_i}(A_i)$ forall i is greater than or equal to P_{in}^C and that the sum of all M_i is not greater than M_{in}, we can guarantee that equation 2 holds and the result complies with our algorithm's specification. Another interesting point is which distributions of M_{in} to consider. In our basic version we distribute M_{in} uniformly over all queried peers, more sophisticated approaches could weight this distribution according to the amount of elements expected from each peer. Based on these considerations, our algorithm determines possible M_{in} and P_{in}^C parameters for forwarding the query to neighbored peers.

While we are currently passing one value for M_{in} and P_{in}^C to neighbored peers, an improved algorithm can pass a whole probability distribution of possible M_{in} values. Then it would be possible to use the exact equation 3 rather than the approximation 4.

Procedure *adjust-missed* As we have mentioned above we have to adjust the missed value that is returned if any element e of the final result does not lie inside the interval $[X-dist-1, X+dist+1]$. In case we discarded at least one peer, we could have missed one "better" (closer to X) element e' in $[X-dist-1, X+dist+1]$ for each element e. This is due to the fact that all tests whether to discard a peer or not are only applied to the interval $[X-dist, X+dist]$. Therefore, we adjust M by:
$M = \max\{M_L, |\text{result items outside } [X-dist-1, X+dist+1]|\}$.

Calculating Probabilities Finally, we give a brief look at how to calculate $P_{\leq M}(A_i)$, $P_{\geq M}(A_i)$ respectively. The performance and correctness of the whole processing strategy strongly depends on these calculations, as we will show in Section 4. Following from

$$P_{\leq M}(A_i) = 1 - P_{\geq M+1}(A_i)$$

we only need to implement one of these functions. Let I denote the interval of interest. First of all, we can calculate the exact values for all buckets that are completely included in I. After that, we could calculate minima and maxima of all subsets of buckets that are 'cut' via I. The overall minimum/maximum is the sum of all these minima/maxima plus the exact values determined before. The crucial question is how to determine a probability distribution for all values in between one arbitrary interval $[min, max]$. This may be handled by assuming for instance uniformly distributed or normally distributed frequencies. When assuming normally distributed values we can revert to the variance in a bucket in order to approximate $P_{\leq M}(A_i)$. For simplicity, in our first implementation we simply store the variance for each bucket. Naturally, this increases the space needed by the histograms which could be used to improve their accuracy – in order to bypass this disadvantage we could as well approximate the variance using the stored maximal error (and vise versa), but we will in general be limited to only use a rule of thumb for this purpose. Corresponding investigations are part of our ongoing work.

$P_{\leq M}(A_i)$ can now be calculated using the individual distributions for $F(k)$ for all $k \in I$ on all peers $\in A_i$ as separate random variables, which requires computing the convolution between them and therefore mostly depends on solving complex integrals. [13] gives interesting approximations and equations of how to compute the convolution for uniform, Poisson and arbitrary distributions. Due to the lack of space we for our part assume normal distributions, and thus can easily calculate an approximated probability distribution for $card(I)$. The sum of n $N(\mu_j, \sigma_j^2)$, normally distributed random variables, $N(\mu, \sigma^2) := N(\sum_n \mu_j, \sum_n \sigma_j^2)$, is normally distributed, too. We only have to truncate the resulting probability distribution in between $[min, max]$. If $F_{N(\mu,\sigma^2)}$ denotes the cumulative distribution function for a $N(\mu, \sigma^2)$ normally distributed random variable, $P_{\leq M}(A_i)$ can be approximated as:

$$P_{\leq M}(A_i) = \begin{cases} 0 & , min > M \\ 1 & , max \leq M \\ \frac{F_{N(\mu,\sigma^2)}(M) - F_{N(\mu,\sigma^2}(min)}{F_{N(\mu,\sigma^2)}(max) - F_{N(\mu,\sigma^2}(min))} & , \text{else} \end{cases} \quad (5)$$

3.4 Pruning Query Paths

The main intuition behind our method is to prune query paths based on probabilistically guaranteed assumptions in order to optimize processing costs. During the introduction of our algorithms we indicated when and where any query path is pruned, but we did not mention how costs are integrated. This is done by sorting all subsets of directly connected peers according to their costs, starting with the cheapest one (which only includes the local peer). Running through this list, the algorithm stops investigating further subsets when the first set matching the probabilistic conditions is found.

Our prototype algorithm implements a greedy philosophy: each peer tries to discard as many neighbors as possible. This early pruning results in discarding larger subtrees than pruning in later phases of processing. In the current stage of development our implementation only considers the number of queried peers as cost factor. Intuitively, these cost calculations should be extended to take into consideration, e.g., the number of messages, bandwidth, data volume, hops or whatever measure is suitable for expressing costs. This also includes lowering the amount of elements returned by each peer below N, but we have to keep in mind that this goes along with algorithmic modifications and in our current implementation ensures correctness (even when unexpected peer failures occur). A more sophisticated approach could try to deduce trade-offs between the costs for one subset and the achievable percentage of the global top-N result. In this case the system, or the user in an interactive way, may decide which trade-off to choose, rather

than restrict the processing to minimal costs. This strategy reflects Quality-of-Service approaches but is not the focus of this paper.

Example Figure 3 shows how our algorithm processes a query issued at peer 1 with the following parameters: queried value $X = 5$, number of queried top items $N = 5$, the number of allowed missed values $M = 2$ (for illustration M symbolizes M_{in}). We assume a considered distance $dist = 2$ and probability $P_{in}^C = 1$.

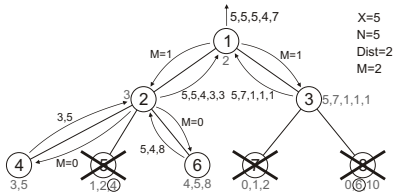

Figure 3: Example for applying algorithm

At first peer 1 forwards the query to its neighbors 2 and 3 distributing M equally among them, both peers are acting likewise but do not forward the query to peers 5, 7 and 8. 5 and 8 are pruned because peers 2 and 3 are allowed to miss one data item each. Peer 7 can be pruned because it has no data item in the interval of interest defined by X and $dist$. After having merged, ranked and pruned the received neighbor and local results, peers 2 and 3 send their results to peer 1 which acts likewise and provides the final result.

4. EVALUATION

In this section we will analyze and interpret the results that we gained running some experiments. We use a simple simulation environment that allows for using routing filters based on histograms as index structures. A bundle of extended tests will follow in future work. For each bucket we store the average frequency, the variance, and the maximum error between average and value frequencies. For simplicity but without loss of generality we assume global bucket boundaries for all histograms. Note that this is no requirement for our algorithm. Using this simple environment we are able to evaluate our algorithm by comparing the processed results and the number of queried peers. The number of resulting messages and generated data volume is not analyzed in the following due to a strong correlation with the number of queried peers.

4.1 Experiments

Our experiments will show the quality of the guarantees given by our algorithm and the performance benefit of pruning. We distinguish between (i) probabilistic guarantee and (ii) specified correctness in terms of the ratio $1 - \frac{M_{in}}{N}$ running from 0 ($M_{in} = N$) to 1 ($M_{in} = 0$), i.e., with a probabilistic guarantee of p the result misses only M_{in} result items in comparison to the global top-N result. What we will evaluate in particular includes:

- The quality of our probabilistic guarantees using histograms.
- The benefit of pruning using histograms.
- The impacts of frequency distribution and network structure.
- The influence of user parameters (P_{in}^C, N, M_{in}) on query results and guaranteed correctness (defined as $1 - \frac{M_{out}}{N}$).

Not in the focus of this work but part of our future work is evaluating the impact of dynamics and filter maintenance strategies.

Impact of value frequency distribution within each bucket In our first experiments we investigated the impact of the approximated distribution within each bucket, i.e., what happens if the actual frequency distribution does not comply with a normal distribution. The results shown in figures 4 (left) and 5 (left) represent the average of several test runs with $N = 100$ and $P_{in}^C = 0.6$ where each run queries another target value X of the same attribute. The P2P network forms a tree of 100 peers with each peer (except leaf nodes) having 4 children. Our astronomical test data is distributed arbitrarily among all peers. Both figures show that P_{out}^C gets higher with increasing specified correctness. This is due to the fact that with higher specified correctness the algorithm has to ask more peers. Doing so the probability of missing any data items decreases which is represented by increasing P_{out}^C.

Figure 4 (left) shows the results of queries where only those buckets are queried whose value frequency distributions more or less comply with a normal distribution. As we have expected, our algorithm works fine in such situations since this is exactly what it assumes for each bucket. The guaranteed correctness that the algorithm gives for its result is always lower than the *actual correctness* that we define as the ratio $\frac{|R_G \cap R_P|}{|R_P|}$, where R_G represents the global and R_P the query result. This means that the algorithm always gives right guarantees. Both lines are situated above the straight line representing the specified correctness. This means that the result is always even better than demanded by the user.

Figure 5 (left) shows what happens when querying buckets whose actual frequency distributions do not at all comply with the assumed normal distribution. Both guaranteed and actual correctness often have lower values than the specified correctness. This obviously leads to incorrect query results in terms of having missed more values of the global top-N result than specified by the user and guaranteed by the algorithm. Since our algorithm is based on assuming a normal distribution this is not astonishing. Combining this basic approach with other distributions should lead to better results which we will investigate in future work.

Influence of N In further tests we analyzed the influence of the number of queried values N on the result quality. All tests were run on the same network structure and with the same parameters as our first ones. Instead of varying the specified correctness we set the specified M_{in} value to $\frac{N}{2}$ in order to have an equal correctness ratio for all test runs. As revealed by figure 4 (right) we expected the quality to decrease with higher N, our approach works well with small N. For larger N, however, the guaranteed correctness is higher than the actual correctness. This is due to the fact that with increasing N our approach has to combine more and more random variables. Therefore, small estimation errors for each variable result in a rather large overall error depending on the size of N. Since the reason for issuing top-N queries is retrieving only a few result items out of many, this can be regarded as a minor restriction of our approach. In order to increase the performance of our algorithm for larger N we could increase the number of buckets per histogram. Due to limited space we do not investigate this aspect of the optimal number of buckets any further.

Influence of P_{in}^C Specifying the value of P_{in}^C the user tries to minimize execution costs by taking the risk of retrieving a result that misses more than M_{in} data items of the global result. Thus, the higher P_{in}^C (i.e., the smaller the risk) the higher should be the number of queried peers, since asking more peers minimizes the risk of missing any result items. Exactly this is shown by the tests whose results are presented in figure 4 (middle), where the only differences to the preceding tests are varying P_{in}^C and setting N to 100 and M_{in} to 25, i.e., setting the specified correctness to 0.75.

Influence of network structure In order to show that our approach does not only work with one special network structure, we created other network trees. The results for a tree with an arbitrary number of neighbors between 0 and 6 is presented in figure 5 (middle). Because of all parameters being set to the same values as in our first tests and the test data being distributed arbitrarily among all peers, the differences in figures 4 (left) and 5 (middle) must be due to the network structure. Since the curve progression in both figures does not differ in any remarkable manner, we can

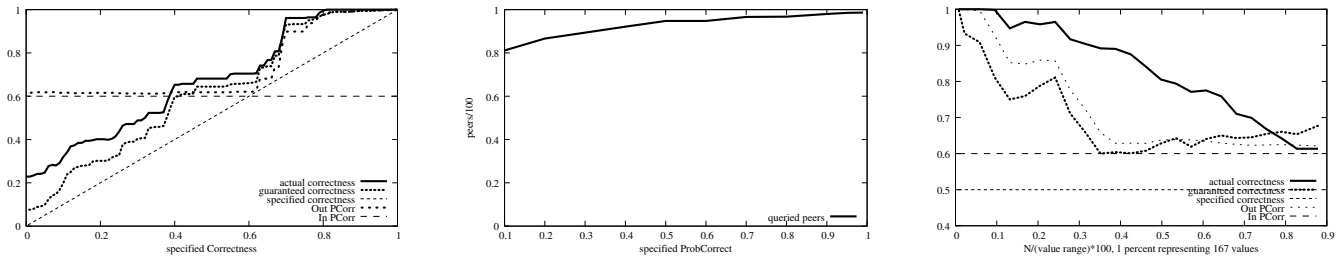

Figure 4: (left): Querying buckets with overall normal value frequency distribution, (middle): Influence of P_{in}^C on the number of queried peers, (right): Influence of N on guaranteed correctness

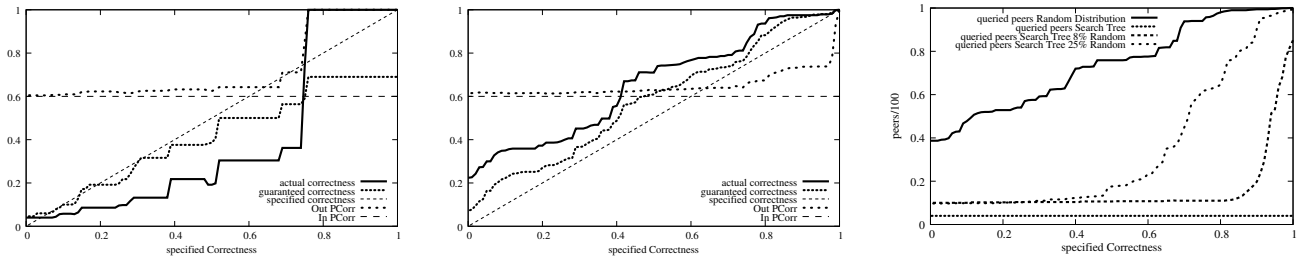

Figure 5: (left): Querying buckets with non-normal value frequency distribution, (middle): Influence of peer tree structure on guaranteed correctness, (right): Influence of P_{in}^C and data distribution on the number of queried peers

reason that the network structure does not influence quality or performance of our approach. Future work will consider more general network structures including cycles and dynamics.

Influence of data distribution on the number of queried peers There are two extreme cases concerning data distribution: (i) every peer has relevant data and (ii) we have a search tree where we can answer queries by asking a minimum number of peers. Figure 5 (right) shows the number of peers with varying specified correctness. The upper line originates from the same test run as depicted in figure 4 (left) where all peers had relevant data, so that with a specified correctness of 1 and $P_{in}^C = 1$ all peers had to be asked. The curves of all other test scenarios should lie in between the two corresponding curves. Figure 5 (right) proves this by showing the curves of two scenarios based on the search tree mentioned above but with a few per cent of randomly distributed data.

5. CONCLUSION AND OUTLOOK

Querying structured data in large-scale Web integration systems is often feasible only by applying best effort techniques such as top-N queries. In this paper, we have presented an approach for processing such queries in a schema-based P2P integration system. We have shown that we can reduce the number of asked peers by relaxing exactness requirements but are still able to guarantee a required percentage of the complete result by a certain probability.

However, the approach presented in this paper represents only a first step. There are two major directions for our ongoing work. First, so far we have considered only one-dimensional histograms and simple rank functions. Though, in principle the approach can be easily extended to multidimensional histograms there are still some open issues. We encountered additional open aspects of the proposed method, i.e., assumed frequency distribution and its impacts, which we plan to investigate more detailedly. Second, we have assumed that each participating peer already owns histograms for all of its neighbors and the queried attributes. In P2P environments we expect limited knowledge and high dynamics. This could be handled by modifying our algorithm and combining it with an incremental processing strategy and gossiping approaches for approximating global knowledge. In parallel, we will optimize the query feedback strategy and apply extended cost calculations.

6. REFERENCES

[1] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[2] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *ICDE'05*, 2005.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS, Vol. 27, No. 2*, 2002.

[4] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *PODC '04*, pages 206–215, 2004.

[5] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *VLDB'99*, pages 397–410, 1999.

[6] S. Chaudhuri, L. Gravano, and A. Marian. Optimizing queries over multimedia repositories. *IEEE TKDE*, 16(8):992–1009, 2004.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *22nd Int. Conf. on Distributed Computing Systems*, pages 23–32, July 2002.

[8] D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top n queries. In *VLDB'99*, pages 411–422, 1999.

[9] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing multi-feature queries for image databases. In *VLDB 2000*, pages 419–428, 2000.

[10] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *ACM SIGMOD'95*, pages 233–244. ACM Press, 1995.

[11] A. Lotem, M. Naor, and R. Fagin. Optimal aggregation algorithms for middleware. In *PODS'01*, Mar. 03 2001.

[12] A. Marian, N. Bruno, and L. Gravano. Evaluating top-queries over web-accessible databases. *ACM TODS'04*, 29(2):319–362, 2004.

[13] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB 2004*, pages 648–659, 2004.