Processing Rank-Aware Queries in P2P Systems

Katja Hose, Marcel Karnstedt, Anke Koch, Kai-Uwe Sattler, and Daniel Zinn

Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract. Efficient query processing in P2P systems poses a variety of challenges. As a special problem in this context we consider the evaluation of rank-aware queries, namely top-N and skyline, on structured data. The optimization of query processing in a distributed manner at each peer requires locally available statistics. In this paper, we address this problem by presenting approaches relying on the R-tree and histogram-based index structures. We show how this allows for optimizing rank-aware queries even over multiple attributes and thus significantly enhances the efficiency of query processing.

1 Introduction

Schema-based Peer-to-Peer (P2P) systems, also called Peer Data Management Systems (PDMS), have recently attracted attention as 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. However, the expected advantages of PDMS like robustness, scalability and self-organization do not come 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 seem to be more 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 w.r.t. the available local knowledge. Examples of such query operators are among others similarity operations, nearest neighbor search, top-N as well as skyline operators.

However, even if we relax exactness or completeness requirements we still need estimations about the error rate. In case of top-N queries this means that we give a probabilistic guarantee that x percent of the retrieved objects are among the top N objects that we would get if we asked all the peers in the system.

Assuming an astronomical application scenario with XML data from sky observations and XQuery as the common query language, a typical query would

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 171–178, 2007.

ask for astronomical objects that match a condition to a certain degree. For instance, a researcher could query the 10 stars closest to a given sky position as shown in the following top-N query:

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

Here, `distance` is used as a ranking function and `limit` restricts the result set to the first N elements returned by `order by`.

Though one can easily combine multiple ranking functions it is often difficult to define the weights for the individual attribute rankings in order to determine the global rank. For this purpose, a more feasible operator is the skyline operator [1] returning the set of those points that are not dominated by any other point[1]. For example, a query asking for the brightest stars close to a given sky position could be formulated as follows:

```
for $s in fn:doc("sky.xml")//objects
skyline of distance($s/rascension,
                $s/declination, 160, 20), max($s/brightness)
return ...
```

Of course, the skyline operator can also be combined with the **limit** clause in order to restrict the size of the result set.

In order to process such queries in P2P systems in an efficient way, appropriate strategies are needed that reduce the number of queried peers as well as the size of the transferred (intermediate) result sets. The contribution of this paper is twofold: *(i)* we present a novel routing filter capturing multidimensional data summaries and *(ii)* we discuss strategies for processing top-N and skyline queries in P2P systems by exploiting these routing filters.

2 Multidimensional Routing Indexes Based on the QTree

Before presenting techniques for realizing the operators motivated in the previous section we will describe the principles of query processing in P2P systems. As sketched in the example in Section 1 we assume peers to export their data in XML and to be able to process queries based on XPath. We further assume the existence of correspondence links between pairs of peers representing schema mappings that we can use for query translation.

A first but naive strategy would be flooding the network, i.e., asking all the peers that are available in the P2P system. Of course, this works but results in high execution costs. These costs can be reduced by minimizing the number of asked peers. In P2P systems this is usually done by applying routing indexes [2,3] for routing the query to only those peers that are most likely to contribute to the final result. For this purpose, we use the concept of routing filters, as presented in [4]. These routing filters cover both schema and instance level and are based on one-dimensional histograms for numerical data.

[1] A point *dominates* another point if it is as good as or better in all dimensions and better in at least one dimension.

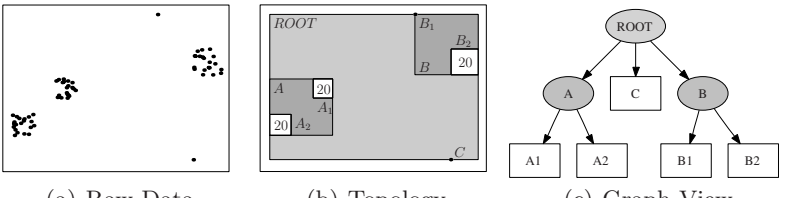
2.1 Routing Indexes

In general, routing indexes represent summarized information about the data a peer can provide. Thus, situations occur where we cannot exactly determine whether the indexed peer actually provides query relevant data or not. Consequently, in addition to reducing message volume and the number of round-trips (our algorithms are restricted to only one round-trip), further cost reduction can be achieved by not forwarding the query to such 'questionable' peers. This results in taking the risk of 'missing' some 'good' data items. The risk that the strategy takes can be quantified and output to the user as a guarantee for the result. In [5] we have presented a strategy that provides probabilistic guarantees for one-dimensional top-N queries. Due to limited space we will not discuss this approach more detailedly.

Having routing filters for one-dimensional queries based on one-dimensional histograms, using histograms for the multidimensional case seems to be the natural consequence. But as also discussed in [6] one-dimensional histograms are insufficient to adequately capture the necessary information about data distributions in the multidimensional case. However, as the main motivation for histograms is to anticipate the number of data items for selection and join queries, it is likely that one bucket covers a large area that contains only few data items and many buckets are used for approximating an area containing a large number of data items. Though this is a good approach for anticipating selection queries, it is not very clever for processing search queries like top-N and skyline. In that case it makes a big difference, if and especially where single data items are located in a bucket. Thus, we have developed a data structure (*QTree*) as a symbiosis of histograms and R-trees. It fulfills the following demands: *(i)* providing information about attribute correlation, *(ii)* being resource-adaptive and resource-efficient in terms of required disk space, *(iii)* being efficient in construction, maintenance and modification in terms of CPU cycles.

2.2 QTree-Based Routing Indexes

Each node in a QTree corresponds to a multidimensional rectangular bounding box. Just like in R-trees, a child's bounding box is completely enclosed in the box of its parent. Leaf nodes are represented by 'buckets' containing statistical



(a) Raw Data (b) Topology (c) Graph View

Fig. 1. QTree

information about those data points that are contained in the bucket's bounding box. The smallest buckets consist of only one point. In the following, we will consider buckets that only provide the number of data points as statistical information, though it is also possible to store mean value or other measures like standard deviation in a bucket. Each QTree has two parameters: (i) f_{max} maximum fanout, (ii) b_{max} maximum number of buckets. b_{max} limits the total number of a tree's buckets and thus its size and memory requirements. Fig. 1 illustrates a two-dimensional QTree with the following parameters: $f_{max} = 3$, $b_{max} = 5$. Fig. 1(a) shows the original data, Fig. 1(b) the QTree's bounding boxes, and Fig. 1(c) the QTree with its buckets and inner nodes.

The QTree represents the basis of the routing filters (*QRoutingFilters*) that we will use in the next section to process multidimensional top-N and skyline queries. A QRoutingFilter describes the data of all neighboring peers in just one index structure. The root node of a QRoutingFilter has one child for each neighbor of the filter owning peer. The data of each child is represented by a QTree and subsumes not only the data of the neighbor itself but also the data that is accessible via this neighbor within a specified hop count distance. The main benefit of maintaining the data altogether consists in the fact that the number of buckets for each neighbor can be chosen (even altered) dynamically.

3 Processing Multidimensional Top-N Queries

This section presents an algorithm based on QRoutingFilters that allows for efficiently processing top-N queries in P2P systems. Considering a peer's local data as well as the buckets of its QRoutingFilter we can efficiently determine the subset of neighbors that provide relevant data.

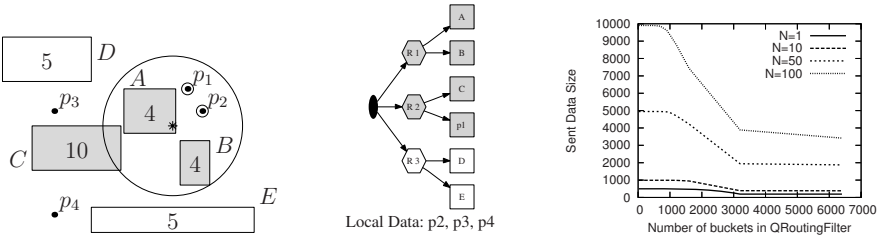
W.l.o.g. let us assume that the score value (that is assigned to a data item by the ranking function) has to be minimized. Let $s_{max}(B)$ and $s_{min}(B)$ denote the maximum and minimum scores that any point in bucket B might have. Furthermore, let $count(B)$ denote the number of data points in B and $\mathbf{B_{all}}$ the set of all buckets. The basic principles of the top-N algorithm are:

- determine a set $\mathbf{B_{suff}} \subseteq \mathbf{B_{all}}$ so that the worst score s is minimized and the following equation holds:
$$\sum_{B_i \in \mathbf{B_{suff}}} count(B_i) \geq N \, , \; s := \max_{B_i \in \mathbf{B_{suff}}} s_{max}(B_i)$$
- based on the worst score s determine all buckets $\mathbf{B_{add}} \subseteq \mathbf{B_{all}} \setminus \mathbf{B_{suff}}$ that might contain data items that have a better score than s:
$$\mathbf{B_{add}} := \{B_j \in \mathbf{B_{all}} \setminus \mathbf{B_{suff}} \,|\, s_{min}(B_j) < s\}$$
- $\mathbf{B_{top-N}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}}$

Based on $\mathbf{B_{top-N}}$, the top-N algorithm is defined as follows:

1. Calculate the top-N result considering all local data and all the buckets of the routing filter

2. Forward a top-K query to all neighboring peers p owning buckets in $\mathbf{B_{top-N}}$, where $K := \min\left\{N, \sum_{p \text{ owns } B_i} count(B_i)\right\}$
3. Receive the answers of those neighbors and combine their results to a preliminary top-N result that is either sent to the query's sender or displayed to the user



(a) Algorithm (b) QRoutingFilter (c) Total Message Volume

Fig. 2. Top-N Algorithm using QRoutingFilters

Fig. 2(a) illustrates an example of our top-N algorithm: Assume we are looking for the top 10 elements near the asterisk. Fig. 2(b) shows the corresponding QRoutingFilter. p_2, p_3, and p_4 are local data items, hence they are not indexed by the filter. In the example, buckets A and B (each containing 4 data items), p_1, and p_2 would be sufficient to provide the top 10 elements: $\mathbf{B_{suff}} = \{A, p_1, p_2, B\}$. The worst-case score for any point in $\mathbf{B_{suff}}$ is visualized as a circle around the asterisk. $\mathbf{B_{add}}$ contains bucket C since this is the only bucket that might provide better data items than $\mathbf{B_{suff}}$. Considering $\mathbf{B_{top-N}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}}$ only neighbors 1 and 2 have to be queried whereas no query has to be forwarded to peer 3.

Experimental Evaluation. For evaluation we analyzed the influence of the size that each routing filter is granted on the number of asked peers as well as on the total network traffic. Based on the attributes *rahour* (right ascension of the observation) and *dedeg* (declination of the observation) of our astronomical test data we calculated 3-dimensional x,y,z-coordinates using a fixed distance of 1000 as radius. Thus, all objects are located on the surface of a sphere. Together with the *vmag* (stellar magnitude in the V system) attribute we built 4-dimensional QRoutingFilters for each peer. The maximum number of buckets in our experiments varies whereas f_{max} is set to 10. All our top-N queries randomly choose a point on the sphere and minimize the sum of the Manhattan distances to the chosen point and to the maximum of *vmag*.

We also varied N in our tests. The results of our tests are shown in Fig. 2(c). The total number of data points sent through the network is shown as a function of b_{max}. The more elements are asked the more data has to be sent. Larger routing filters significantly reduce the network traffic. These results perfectly match our expectations. However, the total number of involved peers could only be reduced significantly when allowing a high number of buckets.

4 Skyline Queries

The main idea of our strategy for processing skylines based on QRoutingFilters is to generalize the "dominates" relation $\succ$ in a way that it can be applied not only on data items but also on arbitrary buckets A and B:

$$A \succ B :\Leftrightarrow a \succ b \ \forall a \in A, \forall b \in B$$

Single data items are interpreted as buckets with no extension. Based on this relation it is possible to calculate a skyline over the local data of a peer, enriched by all buckets of the QRoutingFilter. For this purpose $\succ$ over buckets can be determined using the idea of a worst and a best data item. If items $a_{worst} \in A$ and $b_{best} \in B$ can be constructed such that

$$\forall a' \in A : a' \succ a_{worst} \text{ and } \forall b' \in B : b_{best} \succ b'$$

then

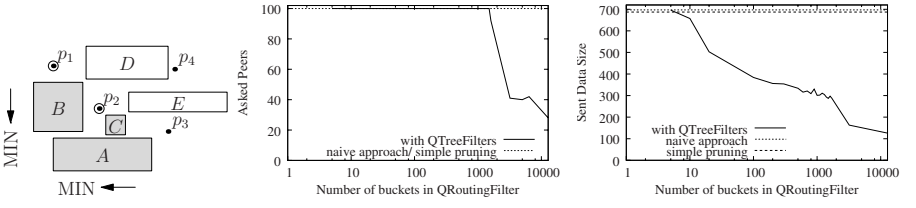
$$A \succ B \Leftrightarrow a_{worst} \succ b_{best}$$

The following fact directly leads to an algorithm for processing skyline queries in P2P systems: All buckets containing data items that are elements of the resulting skyline are elements of the skyline over buckets. The reason is that if an arbitrary bucket B contains a point b that would be part of the overall skyline, there cannot exist any other bucket A that dominates B. Assuming that such an A exists leads to a contradiction: All possible elements in A had to dominate all possible elements in B - thus, also b had to be dominated by all elements in A. Furthermore, as A is not empty, there exists at least one element $a \in A$. So we have found an element a for that $a \succ b$ holds. This is a contradiction to: b is part of the resulting skyline. The skyline algorithm for each peer can be defined accordingly:

1. Calculate the skyline over the local data and all buckets of the routing filter
2. Forward the query to those peers corresponding to the buckets of that skyline
3. Combine the peers' answers to a preliminary skyline that is sent back or displayed to the user

In order to further reduce the data volume that is shipped back during query execution, the data of some buckets is sent along with the query: those, that are likely to dominate a huge amount of the receiver's data. In order to choose these "most selective" buckets, one has to determine how many data items each bucket dominates. Calculating this exactly would be quite inefficient, so we only count for each bucket A how many elements are thrown out of the result skyline because of A. We use a nested loop algorithm for local skyline processing. Before each loop all buckets are sorted decreasingly according to the number of buckets they have already thrown out of the skyline. Thereby, those buckets that have already thrown out a huge amount of data are tested first. Consequently, "selective" buckets become even more "selective". The data of those buckets that superseded the most data items of the corresponding peer is forwarded to the selected neighbor peers. This data consists of a bucket's worst point since this is all information the receiver has to know about a bucket.

Fig. 3(a) shows an example of a skyline over buckets, the corresponding routing filter is shown in Fig. 2(b) with p_2, p_3, and p_4 being local data items. Assuming that both dimensions are to be minimized, buckets A, B, C as well as p_1 and p_2 are members of the resulting skyline. The reason is that only the following dominations occur: $A \succ p_3$, $C \succ E$, $C \succ p_4$, $p_2 \succ p_4$, $B \succ D$, and $B \succ p_4$. Notice that $A \not\succ p_2$ because a point in the bottom left corner of A would dominate p_2. Furthermore, $A \not\succ C$ since it might be possible that A only has data items that are located left of C. As a result the query has to be forwarded to only neighbors 1 and 2.



(a) Algorithm (b) Asked Peers (c) Total Message Volume

Fig. 3. Skyline Algorithm using QRoutingFilters

Experimental Evaluation. In our skyline tests we analyzed the total number of points that was sent through the network in order to answer a query. Like in the top-N experiments we varied the number of buckets for the QRoutingFilters. We queried the skyline over our astronomical test data using the Manhattan distance as ranking function. For each query we randomly chose a point P on the sphere and a *vmag* value. Whereas the distance was to be minimized the *vmag* value was to be maximized. The resulting skylines had an average of 10 data items. The approach like stated above uses QRoutingFilters for routing the query efficiently and sends the "most selective skyline points" along with the query. We used a threshold of 1 what means that all those skyline points that dominated at least one other data point are sent along with the query. In Fig. 3 this strategy is referred to as "*with QRoutingFilters*". It is compared to two other approaches: *naive* and *simple pruning*. *Naive* means that the network is flooded and each peer sends its local skyline to the initiating peer which then processes the final result. The *simple pruning* approach is based on the same principle but the skyline is already checked for dominance by those peers that forward the answer to the initiating peer. Thus, the answer is pruned by forwarding only not dominated skyline points.

Simulation results are shown in Fig. 3(b) and Fig. 3(c). The sum of all points that were sent through the network is displayed on the axis of ordinate whereas QRoutingFilter sizes are displayed in a logarithmic scale on the abscissa. The *naive* and *simple pruning* approaches do not differ significantly because pruning the answer results reduces costs only a little. Using routing indexes, especially QRoutingFilters, on the other hand can effectively reduce the network traffic: A filter that occupies not more than 100 buckets almost halves the amount of

sent data volume compared to a strategy without filter usage. Although these are good results w.r.t. the network traffic the number of asked peers starts decreasing only for large filter sizes (1600 and more buckets per peer), see Fig. 3(b). In order to further decrease the number of asked peers we are working on probabilistic algorithms that relax correctness and completeness requirements.

5 Conclusion

In this work we followed two main goals: processing rank-aware queries while reducing the number of involved peers as well as the amount of data sent through the network. We introduced strategies for achieving both of these goals and focused on the evaluation of a novel data structure called QTree. A QTree combines histograms with the advantages of R-trees. We have shown that the utilization of this structure allows for processing multidimensional top-N and skyline queries efficiently on numerical data. Main open issues and primary content of our current and future work are:

- Approximate query answering techniques, as they promise much more efficiency than always trying to provide exact answers.
- Details of the QTree which are in the first line the construction and maintenance of this innovative data structure.
- How to support rank-aware queries based on string data? This includes finding an index structure that efficiently represents strings, is easy to maintain, and supports lookups for arbitrary strings.
- How to support arbitrary combinations of attributes? This involves combinations of numerical data and string data.

Further aspects, but not in the primary focus of our current work, are the support of limited knowledge in P2P systems and a comparison of QTrees to multidimensional histograms.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: Proceedings of ICDE 2001. (2001) 421–430
2. Crespo, A., Garcia-Molina, H.: Routing Indices for Peer-to-Peer Systems. In: Proc. Int. Conf. on Distributed Computing (ICDCS 2002), Vienna, Austria. (2002) 23–34
3. Petrakis, Y., Koloniari, G., Pitoura, E.: On Using Histograms as Routing Indexes in Peer-to-Peer Systems. In: Proc. DBISP2P 2004. (2004) 16–30
4. Karnstedt, M., Hose, K., Stehr, E.A., Sattler, K.: Adaptive Routing Filters for Robust Query Processing in Schema-Based P2P Systems. In: IDEAS 2005, Montreal. (2005) 223–228.
5. Hose, K., Karnstedt, M., Sattler, K., Zinn, D.: Processing Top-N Queries in P2P-based Web Integration Systems with Probabilistic Guarantees. In: Proc. WebDB 2005. (2005) 109–114
6. Babcock, B., Chaudhuri, S.: Towards a robust query optimizer: A principled and practical approach. In: Proceedings of SIGMOD 2005. (2005) 119–130