A Relaxed But Not Necessarily Constrained Way from the Top to the Sky

Katja Hose[1], Christian Lemke[1], Kai-Uwe Sattler[1], and Daniel Zinn[2]

[1] Dept. of Computer Science and Automation, TU Ilmenau
[2] Dept. of Computer Science, University of California, Davis

Abstract. As P2P systems are a very popular approach to connect a possibly large number of peers, efficient query processing plays an important role. Appropriate strategies have to take the characteristics of these systems into account. Due to the possibly large number of peers, extensive flooding is not possible. The application of routing indexes is a commonly used technique to avoid flooding. Promising techniques to further reduce execution costs are query operators such as top-N and skyline, constraints, and the relaxation of exactness and/or completeness. In this paper, we propose strategies that take all these aspects into account. The choice is left to the user if and to what extent he is willing to relax exactness or apply constraints. We provide a thorough evaluation that uses two types of distributed data summaries as examples for routing indexes.

1 Introduction

One of today's challenges in data integration and distributed data management is to cope with large-scale dynamic environments. A promising solution are Peer Data Management Systems (PDMS), which combine the peer-to-peer (P2P) paradigm and its characteristics such as self-organization, robustness, scalability, and the absence of global knowledge with ideas from classical federated databases. In a PDMS, each peer provides its own data with its own schema and in this way preserves the sovereignty over its data. Furthermore, each peer can answer and process queries and is linked to a small set of neighbors via mappings representing schema correspondences.

However, an inherent problem of large-scale dynamic data management systems is to guarantee complete and exact query answers. In principle, this requires querying all the peers in the system (e.g., by exhaustive flooding) or to know all peers holding relevant data (i.e., to have global knowledge). This is mostly impossible simply due to the mere size of the system. A possible improvement to avoid expensive flooding is the usage of routing indexes [2]. In their original sense they index files by means of keywords. A slightly different understanding of routing indexes came up later [9, 5]. These indexes use summarizing data structures to describe numerical attributes of data records. In accordance to [5] we call such routing indexes *Distributed Data Summaries* (*DDS*). Figure 1 illustrates the two variants (QTree-based [5, 11], histogram-based) we are using in this paper.

Because of the heterogeneity of the data as well as the autonomy and dynamicity expecting exact or complete results often makes no sense: mappings are incomplete, peers can join or leave the system at any time, data is dirty, etc. Thus, we argue that *relaxing*

R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part I, LNCS 4803, pp. 399–407, 2007.

exactness/completeness expectations is a key for efficient query processing in PDMS. Such relaxations can be achieved on more than just one level. First of all, *ranking query operators* such as skyline and top-N by definition do not aim at providing a complete and detailed answer to a query. They provide an overview over the data in the system. Skyline queries [1] are the logical consequence of top-N queries, where the user is not only allowed to define one single ranking function but an arbitrary number of them. These user-defined ranking functions may differ substantially from one query to the next, so that any kind of preprocessed results would not help. Sometimes, however, the user is not interested in the whole data space but only in a subspace that he can specify with *constraints*. This leads to constrained skylines [3] and constrained top-N queries that only consider records in a subspace of the whole data space. Still, there is another option to reduce execution costs: relaxing the completeness/exactness requirements by allowing *fuzziness*. A representative of a fuzzy area may represent several result records clustered in one region.

In this paper, we build upon previous work [11, 5, 6, 4] that discusses the use of DDS and relaxation for processing skyline queries. We present a strategy that adopts and enhances these techniques such that not only skyline but also top-N queries benefit from the application of a fuzzy parameter. We additionally introduce constraints and examine the benefits that we gain from two different types of DDS (QTree-based, histogram-based). The remainder of this paper is structured as follows. After having sketched related work in Section 2, Section 3 presents a strategy for processing relaxed queries in PDMS. In Section 4 we extend this strategy to constraints. Section 5 shows the results of our evaluation and Section 6 concludes this paper.

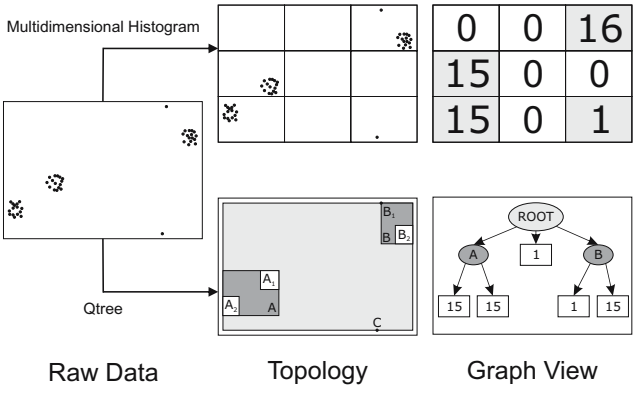


Fig. 1. Base Structures for DDS Illustrated at the Example of a Two-Dimensional Data Set. Left: Original Data, Center: Topology View with Regions/Buckets Representing the Data, Right: Graph View (QTree: Inner Nodes Depicted as Ellipses, Statistic Nodes as Rectangles; Histogram: Regions with the Number of Represented Records Corresponding to Statistic Nodes in the QTree)

2 Related Work

In contrast to the skyline operator, which came up in database research only a few years ago [1], the top-N operator had already played an important role in RDBMS before.

Later on, when distributed systems gained more importance, the Threshold Algorithm (TA) [8] and algorithms based on it have been developed. However, all approaches based on TA have some severe drawbacks: first, the vertical distribution of data and the specialized network structure, where the processing node has direct access to all other nodes. Second, TA requires sorted lists of all objects that we cannot assume to exist in P2P environments.

Already before [1] introduced the skyline operator into database research, the problem had been known as the maximum vector problem [7] before. However, most of the works published have been designed for centralized systems and only a few approaches consider computing skylines in distributed environments. Some exploit the TA principle of sorted lists for processing skylines. The same reasons as stated above make them hardly applicable to PDMS. Another recent work that considers skyline processing in distributed environments is DSL [10]. Although this approach works well for structured overlays, we focus on a more general solution that is able to process skylines in unstructured P2P networks, where we cannot influence the data a peer holds.

3 Distributed Query Processing

In this section we present our algorithm for processing relaxed rank-aware queries in a completely decentralized fashion using the information provided by DDS. Let us first give a formal definition of relaxation:

Given a set D of data objects, a top-N or skyline query $\mathfrak{T}$, a distance function $d : D \times D \rightarrow \mathbb{R}$, and a limit $\varepsilon \in \mathbb{R}$, then any subset R of D for that

$$\forall t \in \mathfrak{T}(D)\ \exists r \in R :\ d(t,r) \leq \varepsilon \tag{1}$$

holds, is called a *relaxed top-N/skyline result*. Furthermore, if for $r,\ t \in D$: $d(r,t) \leq \varepsilon$ holds, r is called a *representing record* of t. A *representative* is the combination of such a representing record and the region that is represented. Thus, a relaxed top-N/skyline result can be defined as a set R that contains a representative for each result record $t \in \mathfrak{T}(D)$. Note that there are usually many R for a data set D that fulfill Equation 1. Furthermore, several records can be represented by one single representative.

The guarantee that is output to the user for such a result is an inherent part of any representative. It guarantees that all records that are represented by the representative are situated within the region that is part of its definition. The maximum distance between the representing record and any point in the region never exceeds ε with respect to distance function d.

Algorithm. Since the ε value limits the maximum approximation error that the algorithm is allowed to make,ε and d have to be specified by the user and added to the query definition. The algorithm for relaxed rank-aware queries can be summarized by the following steps that need to be executed at each peer that receives the query:

1. Compute the query locally – considering local data, DDS regions, constraints, and if provided data received along with the query.
2. Try to find representatives for all regions that are part of step 1's result – using distance function d and the maximum distance ε received along with the query.
3. Forward the query to all neighbors whose regions could not be represented.

4. After all queried neighbors have answered: determine the result over the union of their answer data and the local result – both may include representatives.
5. Try to minimize the number of representatives.
6. Forward the result to the query's sender - data records as well as the remaining representatives.

In the following, we at first describe how to determine representatives for regions. Next, we discuss how to compute rank-aware queries over representatives and finally we discuss how to minimize the number of representatives. Due to limited space we focus on processing top-N queries.[1] All techniques including the application of constraints presented in Section 4 can be used for processing skyline queries as well.

Determining Representatives. Finding representatives for the data that is relevant to the query and provided by neighboring peers reduces execution costs. In the best case all regions that are part of the query result over regions (resulting from step 1) can be represented by known data records such that the query does not have to be forwarded at all. More precisely, choosing representatives means: Given a region B, a distance function d, and a distance ε we have to find a set of records D_{loc} that represents B with respect to the given query specification such that for any point $p \in B$ there exists a local data record $l \in D_{loc}$ for that $d(p,l) \leq \varepsilon$ holds. The problem we encounter is to find a minimal set of representatives that correctly represents a region. As a simple solution to this problem our implementation represents regions only if it is possible to represent them with one single representative, i.e., a region is represented if it is completely enclosed in the region defined by d, ε, and the representing data record.

Top-N Computation over DDS Regions. Without loss of generality let us assume that the score value, which is assigned to a data record 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 region B might have. Furthermore, let $count(B)$ denote the number of data records contained in B. Finally, let $\mathbf{B_{all}}$ be the union of the set of all regions provided by the DDS (only statistic nodes of the QTree) and all local data records – treated as regions with no extensions and a statistics value of 1.

Then, a peer has to determine a set $\mathbf{B_{suff}} \subseteq \mathbf{B_{all}}$ such that the worst score s is minimized and the following equation holds:

$$\sum_{B_i \in \mathbf{B_{suff}}} count(B_i) \geq N \ , \ s := \max_{B_i \in \mathbf{B_{suff}}} s_{max}(B_i) \tag{2}$$

Based on the worst score s the peer determines all regions $\mathbf{B_{add}} \subseteq \mathbf{B_{all}} \setminus \mathbf{B_{suff}}$ that might contain data records that have a better score than s:

$$\mathbf{B_{add}} := \{B_j \in \mathbf{B_{all}} \setminus \mathbf{B_{suff}} | \, s_{min}(B_j) < s\} \tag{3}$$

Finally, the peer determines the set of relevant regions $\mathbf{B_{topN}}$ as:

$$\mathbf{B_{topN}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}} \tag{4}$$

[1] The full version of this paper is available at *http://mordor.prakinf.tu-ilmenau.de/papers/dbis/2007/CoopIS07full.pdf*

The additional information that is forwarded along with the query is p_{worst}. It is the coordinates of the worst record that might be contained in the result set defined by $\mathbf{B_{suff}}$. A peer that receives p_{worst} along with the query only considers regions and local data records that are ranked better than p_{worst}.

Top-N Computation over Representatives. Let us consider each representative R to be a pair (r, B) where B is the represented region and r denotes the record that represents B. Remember that the basic algorithm for processing top-N queries needs to determine a best and a worst score for each region – $s_{\min}$ and $s_{\max}$. We can do the same for each representative by considering the boundaries of the region that it represents. Thus, we can use the same algorithm.

Minimizing the Number of Representatives. Finally, we need to minimize the number of representatives that remained in the result. For this purpose, we split up the representatives into two lists: $\mathcal{R}$ for the data records that represent the regions and $\mathcal{B}$ for the regions that are represented. Given these two lists we try to find a minimal subset of $\mathcal{R}$ that still represents all regions $\mathcal{B}$:

After having split up the representatives, $\mathcal{R}$ is sorted in descending order by the number of regions the entries could represent. We start with an empty set of chosen representatives. For each region $B \in \mathcal{B}$ we try to find an already chosen pair of (r', B') where r' can represent the merged region $B \cup B'$ without violating the approximation constraints defined by d and ε. If such a pair is found we merge the two regions and obtain a larger region that is represented by r'. If there is no such pair that has already been chosen we choose an element from $\mathcal{R}$ that could represent B. Since we have sorted $\mathcal{R}$, those $r \in \mathcal{R}$ that could represent the most regions are considered first and therefore favored. The algorithm ends after all regions of $\mathcal{B}$ are represented.

4 Introducing Constraints

Let $\mathbb{A}$ be the set of all attributes, then we define the set of user-defined constraints C as:

$$C := \{(a, n_l, n_u) | a \in \mathbb{A}, n_l, n_u \in \mathbb{R}\} \tag{5}$$

where n_l and n_u define the interval $[n_l, n_u]$ that constrains attribute a. Of course, we also support single-sided constraints by using negative and positive infinity as default for non-defined values.

In order to adapt the algorithm of Section 3 to work with constraints we have to preprocess the input data set $\mathbf{B_{all}}$ for local query processing, we obtain $\mathbf{B_{con}}$:

$$\mathbf{B_{con}} := \left\{ B_i \in B_{all} | \underset{c \in C}{\forall} overlaps(B_i, c) \right\} \tag{6}$$

This is the set of all regions and local data records that at least partially overlap with the data space defined by the constraints. In other words, all those records and regions are discarded that at least contradict one of the constraints.

5 Evaluation

To evaluate the algorithms presented in this paper we used the two DDS variants illustrated in Figure 1: one based on multidimensional equi-width histograms (HDDS) and

one on the QTree (QDDS). We used three different setups, each is based on the same cycle-free topology of 100 peers with each peer having at most 4 neighbors. We also ran tests with other network sizes and found the same tendencies. Thus, in the following we only present our results for the network of 100 peers, where each peer provides 50 four-dimensional data records (all values restricted to $[0, 1000]$). Figure 2 shows the two-dimensional projection of the data sets. For the first setup "*Random Data, Random Distribution*" the attribute values of each data record and for all dimensions are chosen randomly from the interval $[0, 1000]$. In the second setup "*Clustered Data, Clustered Distribution*" the data of each peer is organized in a cluster. Each cluster has a diameter of 20 and is assigned randomly to a peer. For the third setup "*Anti-correlated Data, Random Distribution*" data records are chosen randomly on the line defined by the points $p_1(1000, 0, 0, 1000)$ and $p_2(0, 1000, 1000, 0)$ and offset by $[-10, 10]$.

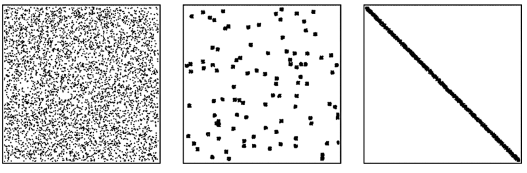
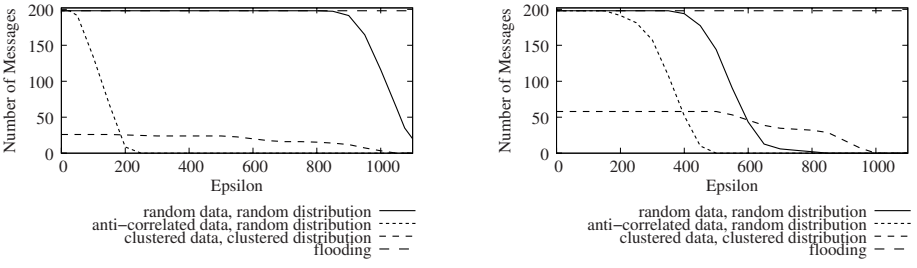


Fig. 2. Data Sets: Random (left), Clustered (middle), and Anti-Correlated (right)

In all tests the DDS are defined on all 4 attribute dimensions. QDDS use a maximum number of statistic nodes of 50 or 100 and 4 as the maximum fanout of inner nodes. HDDS use 5 or 10 buckets per dimension (i.e., 625 or 10000 for each neighbor). In all our tests we varied ε from 0 to 1100, applied the Euclidean distance as distance function, and evaluated the same test query with the same peer as initiator. In order to make use of our multidimensional index structures the top-N test query is defined on 2 attributes by the following ranking function: $attribute1 + attribute2$.



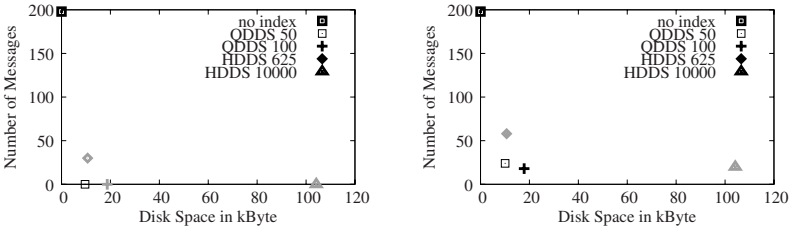
(a) Number of Messages, QDDS 50 (b) Number of Messages, HDDS 625

Fig. 3. Distributed Processing of Relaxed Top-N Queries

Results. With respect to relaxation of top-N queries let us first discuss Figure 3(a) in that we used QDDS with 50 statistic nodes. As the general reduction in the number of messages for all setups indicates, in accordance to our intention the application of

fuzziness reduces query execution costs: the higher ε the more cost reduction. Apart from this general tendency we can also infer the cost reduction that originates from the mere use of DDS as an ε of 0 means that relaxation is not used. In the case of clustered data in a clustered distribution the number of messages necessary to answer a query is reduced to less than 15%. The reason for this effect is that this is the best case scenario for DDS since clusters can be represented easily with low approximation error.

Figure 3(b) shows the corresponding results for HDDS. In these experiments the data of *each* neighbor was described by a histogram with 625 buckets. Remember that the QDDS were only allowed 50 statistic nodes for *all* neighbors altogether. In comparison to Figure 3(a) we see that for all setups the general tendency of message reduction with increasing ε is the same for both DDS types. In situations with little relaxation QDDS are clearly the best choice for almost all setups. But there are some situations where HDDS are the better choice: "random data, random distribution" in conjunction with higher relaxation. The reason is that in this setup there are no clusters that could easily be described by QDDS regions and this is their strength. Due to the higher number of buckets HDDS approximate the data more accurately which in turn enables a more efficient pruning and thus leads to the reduction in the number of messages. However, we still consider this a fair comparison between QDDS (50 statistic nodes) and HDDS (625 statistic nodes) because although the respective number of buckets (statistic nodes) is considerably different, their memory consumption is almost the same. As Figure 4 shows, increasing the number of statistic nodes for QDDS counteracts this problem.

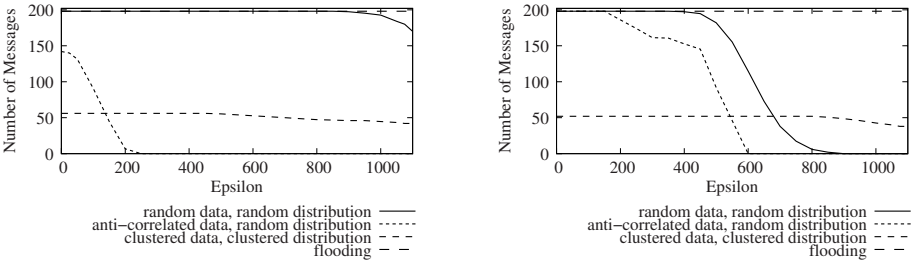


(a) Anticorrelated Data, Random Dist. (b) Clustered Data, Clustered Dist.

Fig. 4. Cost Benefit Analysis using a Skyline for Top-N Query Processing, $\varepsilon = 400$

Figure 4 shows the dependency of execution costs (i.e., the number of messages) on the disk space that is required to manage DDS. It shows some examples for QDDS and HDDS in two of our three setups. But what is the best choice? We have two dimensions that we want to minimize: disk space and execution costs. Thus, a skyline might help to discover the "good" choices: the black points in Figure 4 represent the skylines using an ε of 400. In both skylines QDDS dominate HDDS.

Finally, let us answer the question what happens when we additionally apply constraints. The constraints we applied restricted each queried attribute to $[500, 1000]$. One might expect that reducing the query space to a quarter might reduce execution costs as well since there is less relevant data in the network. Our results presented in Figures 5(a) and 5(b) teach us otherwise. In comparison to the full space queries, execution costs increase or stay more or less the same for 2 of 3 test scenarios and for both DDS.



(a) Number of Messages, QDDS 50 (b) Number of Messages, HDDS 625

Fig. 5. Distributed Processing of Constrained and Relaxed Top-N Queries

Only in the anti-correlated data setup with use of QDDS costs are reduced. The reason is that in the other 2 setups the result set of top-N records changes for our queries (in fact, it might be very different from the full space result) and new regions and records that have not been relevant before suddenly become interesting and have to be evaluated. This is different for the anti-correlated setup. In that case the algorithm can safely discard half of the data space without having to investigate new regions. This results in the cost reduction that we have found astonishing in the first place. As QDDS have already a rather good performance for the full space, we see this effect more clearly in Figure 5(a) than in Figure 5(b). Of course, if we reduced the data space to a very small portion of the original one, we would find a cost reduction for all setups.

6 Conclusion

In this paper, we have discussed efficient processing of rank-aware query operators in distributed environments. One of the key concepts is the use of Distributed Data Summaries (DDS) as DDS enable an efficient query routing to only those peers that are most likely to contribute to the final result. Apart from the basic strategy we have proposed the use of fuzziness such that the result does not only consist of data records but also contains representatives. Our evaluation results show that this in conjunction with DDS is an effective possibility to reduce query execution costs. Another concept that we introduced are constraints. As the evaluation shows this is only a severely limited possibility to reduce costs. The application of such constraints in general leads to a task that is not easier than the original one. However, relaxation is not restricted to distributed environments. Since the benefit was very good especially for the anti-correlated data set, future work will consider combining this technique with centralized algorithms for that especially anti-correlated data means the worst case scenario.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: ICDE 2001, pp. 421–432 (2001)
2. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: ICDCS 2002, pp. 23–32 (2002)

3. Dellis, E., Vlachou, A., Vladimirskiy, I., Seeger, B., Theodoridis, Y.: Constrained Subspace Skyline Computation. In: CIKM 2006, pp. 415–424 (2006)
4. Hose, K., Karnstedt, M., Koch, A., Sattler, K., Zinn, D.: Processing Rank-Aware Queries in P2P Systems. In: DBISP2P 2005, pp. 238–249 (2005)
5. Hose, K., Klan, D., Sattler, K.: Distributed Data Summaries for Approximate Query Processing in PDMS. In: IDEAS 2006 (2006)
6. Hose, K., Lemke, C., Sattler, K.: Processing Relaxed Skylines in PDMS Using Distributed Data Summaries. In: CIKM 2006, pp. 425–434 (2006)
7. Kung, H.T., Luccio, F., Preparata, F.P.: On Finding the Maxima of a Set of Vectors. Journal of the ACM 22(4), 469–476 (1975)
8. Lotem, A., Naor, M., Fagin, R.: Optimal Aggregation Algorithms for Middleware. In: PODS 2001 (2001)
9. Petrakis, Y., Koloniari, G., Pitoura, E.: On Using Histograms as Routing Indexes in Peer-to-Peer Systems. In: Ng, W.S., Ooi, B.-C., Ouksel, A.M., Sartori, C. (eds.) DBISP2P 2004. LNCS, vol. 3367, pp. 16–30. Springer, Heidelberg (2005)
10. Wu, P., Zhan, C., Feng, Y., Zhao, B., Agrawal, D., Abbadi, A.E.: Parallelizing Skyline Queries for Scalable Distribution. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Boehm, K., Kemper, A., Grust, T., Boehm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 112–130. Springer, Heidelberg (2006)
11. Zinn, D.: Skyline Queries in P2P Systems. Master's thesis, TU Ilmenau (2005)