Rev. Téc. Ing. Univ. Zulia. Vol. 32, Nº 2, 170 - 179, 2009

Efficient evaluation of Top-k Skyline queries

Marlene Goncalves and María-Esther Vidal

Departamento de Computación, Universidad Simón Bolívar. Sartenejas-Baruta, Venezuela. Código Postal 1080. Fax: 0212-9063243. Telf. 0212-9063269. {mgoncalves,mvidal}@usb.ve

Abstract

Emerging technologies have made available very large data repositories, which may be unreliable for a given preference criteria. In order to be able to process these repositories, users may need to discard useless information based on some preference conditions. Different preference-based query languages have been defined to support the bases for discriminating poor quality data and to express user's preference criteria. In this paper, we consider the preference-based query language, "Top-k Skyline", which combines the order-based and score-based paradigms. Thus, "Top-k Skyline" is able to identify the top-k objects w.r.t. a score function *f* among the ordering induced by a multicriteria function *m*. Several algorithms have been proposed to implement these two paradigms independently; however, the problem of efficiently evaluating "Top-k Skyline" queries remains open. In this work, we propose evaluation strategies for "Top-k Skyline" queries and we report initial experimental results that show the properties of our proposed solutions.

Key words: Preference-based queries, Skyline, Top-k.

Evaluación eficiente de consultas Top-k Skyline

Resumen

Tecnologías emergentes permiten el acceso a grandes repositorios de datos, mucho de los cuales pueden publicar datos pocos confiables. Los usuarios que acceden independientemente a estos repositorios deben ser capaces de identificar y descartar información que no sea de utilidad, basándose en condiciones de preferencias. Diferentes lenguajes de consultas basadas en preferencias han sido definidos, los cuales permiten discriminar datos de poca calidad y expresar criterios de preferencias. En este trabajo, se presenta el lenguaje "Top-k Skyline", que combina los paradigmas basado en orden y *score*, y es capaz de identificar los mejores *k* objetos de acuerdo a una función de *score f* entre el orden inducido por una función multicriterio *m*. Distintos algoritmos han sido propuestos para implementar dichos paradigmas independientemente; sin embargo, el problema de evaluación eficiente de consultas "Top-k Skyline" permanece abierto. En este trabajo, se proponen estrategias de evaluación para consultas "Top-k Skyline" y se presentan resultados experimentales iniciales que muestran las propiedades de las soluciones propuestas.

Palabras clave: Consultas basadas en Preferencias, Skyline, Top-k.

Introduction

Currently, Google has indexed between ten and thirteen billion of Web pages[1]. Some of these pages may publish irrelevant data, and users that access these data need to discard useless information based on their preferences or decision criteria. To express these criteria for a certain collection of data, different preference-based query languages have been defined. These languages

1 http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

can be grouped into three paradigms: order-based where the objective is to identify all the objects that are non-dominated by any other objects [1-7]; score-based that linearly orders the input data and identifies the top-k objects from this ordering [8-12]; and a hybrid paradigm that retrieves the top-k objects among the ordering induced by a multicriteria function [2, 13-17]. In these paradigms, the input dataset can be distributed among several data sources.

The main problem of order-based query engines is to compute the first stratum or Skyline, i.e., all optimal or non-dominated objects in terms of a multicriteria function, which induces a partially ordered set or strata. On the other hand, score-based query engines rank the top-k objects in terms of a positive linear score function that induces a total order of the objects. Finally, hybrid languages combine score and multicriteria functions, and the problem is to identify the top-k objects among a partially ordered set or strata.

Existing hybrid query engines only compute the top-k objects of the first stratum or Skyline [2, 13-17]. Although the answers produced by these engines are sound, they may be incomplete when k is greater than the Skyline cardinality. In this paper, we propose the hybrid language Top-k Skyline, and evaluation algorithms that overcome limitations of existing hybrid query engines by reducing the number of non-necessary accesses and probes under certain properties of the input dataset.

Formally, given a Top-k Skyline query, a multicriteria function m, a score function f and a dataset of objects O which may be distributed among several data sources, we are interested in defining algorithms to efficiently identify the top-k objects from strata R of O, using the functions f and m, respectively. Strata R is a sequence of subsets $<R_1,..., R_n>$, where each R_i is a stratum, $R_i \in O$, and objects in R_i are better than objects in R_{i+1} for each criterion in m. Thus, R is a partition of O according to m. Objects in R_i are non-dominated, i.e., none of these objects is better than the others for all criteria in m. Therefore, our objective is to identify the Top-k Skyline while non-necessary probes of functions m and f are reduced.

To illustrate how a hybrid query engine works, consider a research company that has two vacancies and received applications from seven candidates. Candidates are described by an identifier, degrees, publications, years of professional experience, and grade point averages. Suppose that the following relational table represents candidate's information: *Candidate(Id, Degree, Publications, Experience, GPA)*. Additionally, Table 1 illustrates information of seven candidates.

According to the company policy, all criteria are equally important and relevant; hence, either a weight or a score function cannot be assigned. A candidate can be chosen if there is any other candidate with higher degree, publications, and experience. To nominate a candidate, one must identify candidates that are not dominated by any other candidate in terms of these criteria. Thus, tuples in table *Candidate* must be selected in terms of *Degree*, *Publications*, and *Experience*. For example, the candidate "d" dominates candidates "e", "f", and "g" because he has higher degree, more publications and experience. Following this idea, the nominate candidates in Table 2 are identified.

Table 1
Candidates for two vacancies of a research company

Id	Degree	Publications	Experience	GPA
a	Post Doctorate	9	2	3.75
b	Post Doctorate	10	1	4.00
c	PhD	12	2	3.75
d	MsC	13	4	3.60
e	Engineer	6	3	3.25
f	Engineer	5	2	3.20
g	Engineer	4	2	3.15

Table 2
Nominate Candidates for two vacancies of a research company

Id	Degree	Publications	Experience	GPA
a	Post Doctorate	9	2	3.75
b	Post Doctorate	10	1	4.00
c	PhD	12	2	3.75
d	MsC	13	4	3.60

Since the company only has two vacancies, it must apply another criterion to select the two new staff members and discard the others. Staff members will be selected among nominates in terms of the top 2 values of one or more score functions.

First, considering the maximum GAP as one score function, three candidates are the new nominates: "b", "a", and "c". Then, taking into account the degree, the tie between "a" and "c" is broken, and the selected members are: "b" and "a". From a procedural point of view, to select the staff members, preference-based queries need to be posted against the table *Candidate*.

There are several query languages to specify preference criteria. Skyline and Top-k are two user preference languages that could be used to identify some of the staff members. However, none of them will provide the complete set, and post-processing may be needed. On one hand, Skyline offers a set of operators to build a set of points that are non-dominated by any other point in the dataset. Thus, by using Skyline, one could just obtain the nominated candidates and a score function needs to be applied afterwards to identify the new staff members. On the other hand, Top-k allows referees to implement a score function and may filter only some of the winners in terms of the combined function. In order to choose staff members, Top-k computes the score for each tuple without checking dominance relationship between tuples in the dataset. Nevertheless, it is not possible to define such function, because all criteria are equally important and relevant. Therefore, to solve the problem of selecting the staff members, the top k elements among the objects in a partially ordered set need to be computed, and a hybrid approach that combines the benefits of Skyline and Top-k is required.

Time complexity for answering preference-based queries is high and it depends on the size of the input and the number of probes performed. In general, the problem of identifying the Skyline or first stratum is $O(n^2)$; this is because all the input instances need to be compared against themselves to probe the multicriteria function m[2]. Additionally, the time complexity of selecting the top-k objects is $O(nlogn)$ because in the worst case, the whole input set needs to be ordered[3]. Finally, since a Top-k Skyline query engine requires to stratify the input data until the top-k objects are computed, more than one stratum may be needed; and in consequence, the time complexity is mainly impacted by the cost of building the strata.

To reduce the processing time, Top-k Skyline query engines must implement efficient mechanisms that minimize the score and multicriteria function probes. In this paper, we propose two evaluation techniques to solve Top-k Skyline queries, and present our initial results.

The paper is comprised of five sections. In Section 2, we describe related approaches. In Section 3, we define two algorithms for computing the Top-k Skyline. In Section 4, we report our

2 A study of complexity Skyline problem is presented in .
3 First the score function is probed for each instance, then data is ordered and ?nally, the top-k instances are returned.

experimental results. Finally, in Section 5, the concluding remarks and future work are pointed out.

2. Related Work and Background

In [3, 4] algorithms identify the Skyline by scanning the whole dataset. On the other hand, progressive (or online) algorithms for computing Skyline have been introduced [17, 18]. A progressive algorithm returns the first results without having to read the entire input and produces more results during execution time. Although these strategies could be used to implement our approach, they may be inefficient because they may perform a number of non-necessary probes or require index structures which are not accessible in Web data sources.

In order to process Skyline queries against Web data sources, efficient algorithms have been designed considering sequential and random accesses. Each data source contains object identifiers and their scores. A sequential access retrieves an object from a sorted data source while a random access returns the score from a given object identifier.

The Basic Distributed Skyline (BDS) defined by Balke et al. [1] is one of the algorithms to solve this kind of Skyline queries. BDS obtains a Skyline superset in a first phase and in a second phase; it discards the dominated ones of this superset. A second algorithm known as Basic Multi-Objective Retrieval (BMOR) is presented by Balke and Güntzer [2]; in contrast to BDS, it compares all the seen objects once it finds that some seen object dominates a virtual object that is updated constantly. Both algorithms avoid to scan the whole dataset, and to minimize the number of probes.

To understand BDS and BMOR, consider the relational table *Candidate*. Publications and grade point averages are collected from two different data sources as can be seen in Table 3. Sources are sorted by Publications or Grade Point Averages, respectively. Also, suppose that a department is interested in candidates with maximum number of publications and grade point averages.

To solve this query, BDS performs sorted access in a round robin fashion on the data set in order to compute a Skyline superset as is shown in Table 4. Once BDS completely sees the object "b" which dominates any unseen object, it stops, and the Skyline superset is comprised of the objects "a", "b", "c", and "d". BDS can stop at this

Table 3
Datasets exported by sources S_1 and S_2

S_1		S_2	
Id	Publications	Id	GPA
d	13	b	4.00
c	12	a	3.75
b	10	c	3.75
a	9	d	3.60
e	6	e	3.25
f	5	2	3.20
g	4	2	3.15

Table 4
Data accessed by BDS and BMOR

BDS			BMOR			
Id	Publications/GPA	Source	Id	Publications/GPA	Source	Virtual Object
d	13	S_1	d	13	S_1	(13,)
b	4.00	S_2	b	4.00	S_2	(13,4)
c	12	S_1	c	12	S_1	(12,4)
a	3.75	S_2	a	3.75	S_2	(12,3.75)
b	10	S_1	b	10	S_1	(10,3.75)

point, because data is ordered in the sources and any unseen object will be worse in each attribute than "b". Then, BDS performs random access to retrieve the unseen scores from "a", "c", and "d"; using all of these values, it discards dominated objects from the Skyline superset. Finally, it outputs the Skyline which is composed by "a", "b", "c", and "d".

Similarly, BMOR scans the same objects than BDS but constructs a virtual object. A virtual object contains the worst values seen in each sequential access. For each seen object, the algorithm performs random access for retrieving unseen values and compares pair-wise the seen objects versus the updated virtual object. Table 4 presents the list of virtual objects produced, where each pair represents the worst values seen for the sources S_1 and S_2. At this point, the seen object "c" dominates the last virtual object *(10,3.75)* and therefore, a Skyline superset was obtained because the object "c" dominates any unseen object. The algorithm discards dominated objects and produces the same result as BDS.

Both algorithms perform a minimal number of probes for computing the first stratum [1, 2]. Since, it maybe required to scan more than the first stratum to solve Top-k Skyline queries, these two approaches need to be extended to produce all the necessary strata. In this paper, we extend these two algorithms to efficiently implement the Top-k Skyline.

3. Top-k Skyline Algorithms

We propose two algorithms that assume certain order in the values of the attributes to identify the point where to stop probing the score and multicriteria functions. The first algorithm is built upon a final object, where scores in the multicriteria function m have been completely seen before the stopping point; while the second considers a virtual object that contains the minimum seen values in each of the attributes contained in function m.

Our algorithms assume that the values of each attribute A_i are stored in an ordered list Si. All these lists are scanned in a round robin fashion and only one object is recovered during each access. The virtual object is built with the worst seen values in each list Si. If an object in the stratum dominates the virtual object, then it is not necessary to continue looking for non-dominated objects. This is because the virtual object will dominate any unseen object, since any unseen object has worse value in each attribute than the virtual object. Thus, we have various virtual objects until finding the last necessary stratum. Similarly, the final object is built accessing in a round robin fashion each list Si, and because the final object will dominate any unseen object, it is not required to look for more non-dominated objects.

3.1. Basic distributed Top-k Skyline (BDTKS)

We have extended the BDS Algorithm introduced in [1] in order to construct a set of Top-k Skyline objects. Our algorithm, showed in Figure 1, first builds all the necessary strata NS, and then, it probes the value of the function f for these objects. Elements are considered respecting the order induced by m, i.e., the top-k tuples correspond to the best k objects in the topological sort of $NS^{(4)}$.

The Basic Distributed Top-K Skyline (BDTKS) algorithm in Figure 1 iteratively constructs necessary stratum until the Top-K Skyline tuples are identified (step 2 **SEARCH TOP-K SKYLINE**). To build each necessary stratum, the algorithm presented in Figure 1 follows three phases labeled as: **SEARCH**, **SCAN,** and **LOAD STRATUM**. During the **SEARCH**, the algorithm finds the first object, called final object, i.e., the object where the multicriteria attribute scores have been seen in each list S_i. In **SCAN** or second phase (steps from 2b to 2c), it verifies if final object's scores are not duplicate; then, it executes a sequential access for each list S_i, and stops when a different score is found in each list. The last phase (steps from 2d to 2g) or **LOAD STRATUM** randomly accesses each partially seen object, and stores non-dominated objects in a current

4 Topological sorting is done according to the combined score function f.

```
INPUT: O: Data set; k: integer; m: multicriteria function; f: combined score
function;
OUTPUT: Top-K Skyline objects;

//INITIALIZE
1. P1 ← ∅; K1,...,Kn ← ∅; i ← n; s ← 1; cont ← 0;
// SEARCH TOP-K SKYLINE
2. WHILE (cont < k and ∃o ∈ O)
      // SEARCH
      a) WHILE (∃o ∈ O and all the scores si(o) are not known)
            i) Select an object o by sorted access from any list i in round robin
               fashion and add o to Ki;
      //SCAN
      b) i ← 1;
      c) WHILE (i ≤ n)
            //VERIFY
            i) WHILE (the next score in list Si is equal to si(o))
                  A. Select an object o by sorted access from list i and add o to
                     Ki;
            ii) i ← i + 1;
      // LOAD STRATUM
      d) i ← 1;
      e) WHILE (i ≤ n)
            i) Do all necessary random access for the objects in Ki;
            ii) Compare all seen objects pairwise in Ki;
            iii) Add non-dominated objects to Ps;
            iv) Remove non-dominated objects from Ki;
            v) Create Ps+1; i ← i + 1;
      f) Calculate and order all non-dominated objects by the combined function f
         in stratum Ps;
      g) cont ← cont + size(Ps); s ← s + 1;
// EXIT
3) return Top-k Skyline objects;
```

Figure 1. Basic distributed Top-K Skyline.

stratum and dominated objects in a temporal stratum. Objects in the temporal stratum are not discarded and are passed to the next iteration of the loop. All the objects that constitute a stratum output in an iteration of the algorithm are removed from the input.

Steps **SEARCH**, **SCAN,** and **LOAD STRATUM** correspond to the steps performed by the Basic Algorithm which has been proven to be completed (Theorem 1 [1]), i.e., these steps compute the Skyline of a given dataset. Our algorithm Basic Distributed Top-K Skyline iterates over these steps until at least the top-k tuples are identified. In each iteration, the Basic Distributed Top-K Skyline algorithm receives as input a dataset where the Skyline identified in the previous iteration has been removed; thus, the Basic Distributed Top-K Skyline computes all the strata required to produce the top-k tuples, i.e., the algorithm is complete. On the other hand, the Basic Distributed Top-K Skyline stops iterating immediately after computing the minimal strata whose cardinality is greater or equal to "k" and non-necessary strata are not computed; in consequence, the Basic Distributed Top-K Skyline is optimal w.r.t. the number of necessary strata and it minimizes the number of probes.

3.2. Basic multi-objective retrieval for Top-k Skyline (BMORTKS)

The Basic Multi-Objective Retrieval for Top-K Skyline (BMORTKS) algorithm in Figure 2 is also an extension of a previously defined algorithm; in this case, we extended the basic multi-objective retrieval algorithm proposed by Balke and Güntzer's [2]. The BMORTKS algorithm iteratively constructs necessary strata until the top-k objects are identified.

Each necessary stratum is built in two steps: **SEARCH** and **LOAD STRATUM**. In **SEARCH**, a list S_i is visited in a round robin fash-

```
INPUT: O: Data set; k: integer; m: multicriteria function; f: combined score
function;
OUTPUT: Top-K Skyline objects;

//INITIALIZE
1. P1 ← ∅; cStrata ← ∅; i ← 1; cont ← 0;
// SEARCH TOP-K SKYLINE
2. WHILE (cont < k and ∃o ∈ O)
      // SEARCH
      a) Select an object o by sorted access from any list in round robin
fashion;
      b) Add o to cStrata;
      c) For new objects perform random accesses on the other lists and
calculate
         their object's combined score function f;
      d) Update a virtual database object p characterized by the minimum score
         values that have occurred in each list;
      // LOAD STRATUM
      e) WHILE (some object p has already been seen for which holds w does not
               dominate p)
            i) Compare all seen objects pairwise in cStrata;
            ii) Add non-dominated objects of cStrata to the stratum Pi;
            iii) Create temporal stratum TemporalStratum with dominated objects of
                 cStrata;
            iv) cStrata ← TemporalStratum;
            v) cont ← cont + size(Pi); i ← i + 1;
// EXIT
3) return Top-k Skyline objects;
```

Figure 2. Basic multi-objective retrieval for Top-K Skyline.

ion and an object *o* is sequentially accessed from this list; the rest of the non-seen scores for *o* are randomly accessed from the other lists. Then, the algorithm probes the combined score function for each seen object (step2c). In each iteration, a virtual object is updated with the minimum score values that have occurred in each list (step 2d) and when a seen object dominates the virtual object (step 2e), the non-dominated objects are added to current strata and the dominated objects are included into a temporal stratum, which will be used in the next iteration. The algorithm stops when just *k* tuples have been produced.

Similar to the Basic Distributed Top-K Skyline, the Basic Multi-Objective Retrieval for Top-K Skyline is correct based Theorem 1 presented in Balke and Güntzer [2] that establishes that the Multi-Objective Retrieval Algorithm is correct. In addition, the algorithm computes a minimal number of strata because it stops iterating right after computing the minimal strata whose cardinality is greater or equal to "k"; in consequence, we can say that the Basic Multi-Objective Retrieval for Top-K Skyline algorithm is complete and optimal w.r.t. the number of strata.

4. Experimental Study

Our experimental study was performed on Oracle 9i. The study consisted of experiments running over relational tables with 10,000 tuples. Each table contains an identifier, and a column of type real, that represents the score. Each tuple in the natural join of the six tables corresponds to an object. Values of the real columns varied from 0.0 to 1.0. A column may have duplicated values. The attribute values were generated for the following two data distributions: Mixed and Correlated, where:

- *Mixed*: Attributes are independent of each other. Data are divided into two groups of three columns: one group was generated using a uniform distribution and the other, using a Gaussian distribution.
- *Correlated*: Data are divided in two groups of the three real columns. In each group, the attribute values for a base column were generated following a uniform distribution.

The two algorithms were implemented in Java 1.5 and the experiments were executed on a SunFire V440 machine equipped with 2 proces-

sors Sparcv9 of 1.281 MHZ, 16 GB of memory and 4 disks Ultra320 SCSI of 73 GB running on SunOS 5.10 (Solaris 10).

We randomly generated ten queries characterized by the following properties: (a) there is only one table in the FROM clause; (b) the attributes in the multicriteria function and the score function were chosen randomly among the attributes of the table, following a uniform distribution; (c) directives for each attribute of the multicriteria function were selected randomly considering only maximizing and minimizing criteria; (d) the number of attributes of the multicriteria function was two, four and six; (e) the score function was selected randomly among the minimum, average, and geometric average; (f) the number of score function arguments was chosen randomly following a uniform distribution; and (g) k corresponds to 1% of data size.

Figure 3 reports the average of multicriteria function probes (MP)($\times 10^4$ probes) and average time ($\times 10^6$ miliseconds) required by each algorithm. Average time is the average of the ten queries run against the tables with data generated according to the mixed distribution. In general, we can observe that:

1. The number of multicriteria function probes is lower for the BDTKS algorithm. This is because this algorithm only compares all seen objects until it finds the final object. On the other hand, the BMORTKS algorithm has to compare all the objects with the virtual object and then, once this condition is satisfied, the algorithm has to compare all these objects with themselves to determine which of them are non-dominated. Thus, probing with respect to the virtual object increases the number of comparisons of the BMORTKS algorithm.
2. The BDTKS algorithm requires more time. This might be because the position of the final object is higher than the virtual object and therefore, the BDTKS algorithm requires more sequential accesses.

Figure 4 reports the results of the experiments for correlated data. We can summarize that:

1. The number of multicriteria function probes is higher for the BDTKS algorithm. The correlation seems to affect the BDTKS algorithm. This might be because the number of sequential accesses required to find the final object is greater than the same measure in the BMORTKS algorithm, and therefore, the number of multicriteria function probes increases.
2. Similar to non-correlated data, the BDTKS algorithm requires more time.

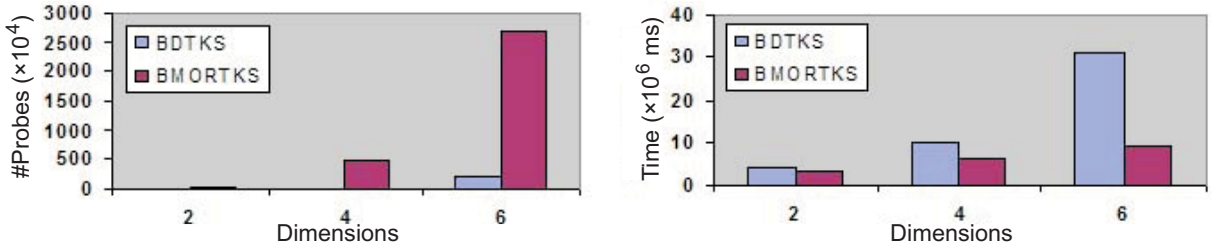

Figure 3. Non-correlated dataset.

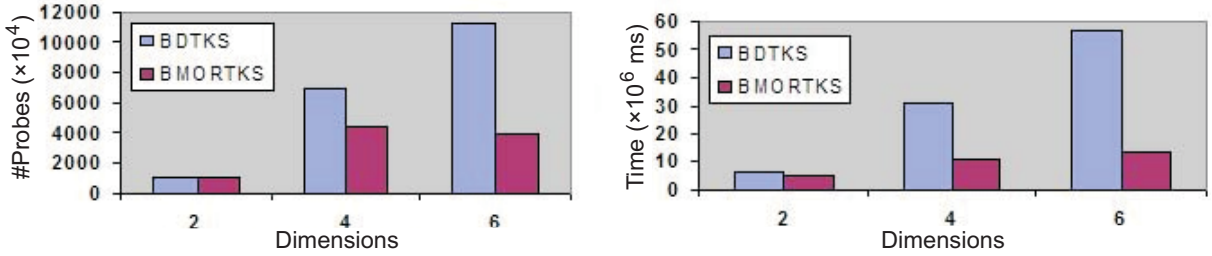

Figure 4. Correlated dataset.

5. Conclusions and FutureWork

In this work, Top-k Skyline techniques have been proposed: the BDTKS and BMORTKS algorithms. The first is based on the computation of a final object whose scores have been completely seen. The second computes a virtual object which is comprised of the minimum scores seen so far. Both algorithms are complete and construct fewer strata than a naive solution. Additionally, the number of probes performed is reduced, since only the necessary strata are considered. Initial experimental results show that BDTKS performs less multicriteria function probes and spends more evaluation time when data is non-correlated. Finally, we could observe that data correlation may affect the behavior of both algorithms, and BMORTKS has better performance in terms of multicriteria function probes and evaluation time.

However, our algorithms are not able to stop before the last necessary stratum will be built completely; in consequence, these algorithms are not optimal w.r.t. the number of probes. In the future, we will define new techniques that overcome this limitation.

On the other hand, we have not considered that there may be several execution plans for a given Top-k Skyline query. Considering preference criteria during the query optimization might help to identify better execution plans. We plan to integrate Top-k Skyline techniques into a relational engine to select good execution plans that take into account preference criteria.

References

1. Balke, W-T., Güntzer, U., and Zheng, J. "Efficient Distributed Skylining for Web Information Systems". Proceedings of the International Conference on Extending Database Technology (EDBT), Greece (2004), 256-273.
2. Balke,W-T. and Güntzer, U. "Multi-objective Query Processing for Database Systems". Proceedings of the International Conference on Very Large Databases (VLDB), Canada (2004), 936-947.
3. Börzönyi, S., Kossman, D., and Stocker, K. "The Skyline operator". Proceedings of the International Conference on Data Engineering (ICDE), Germany (2001), 421-430.
4. Godfrey, P., Shipley, R. and Gryz, J. "Maximal Vector Computation in Large Data Sets". Proceedings of the Conference on Very Large Data Bases (VLDB), Norway (2005), 229-240.
5. Lee, K., Zheng, B., Li, H. and Lee, W.-C. "Approaching the skyline in Z order". Proceedings of the International Conference on Very Large Databases (VLDB), Austria (2007), 279-290.
6. Vlachou, A., Doulkeridis, C. and Kotidis, Y. "Angle-based space partitioning for efficient parallel skyline computation". Proceedings of the ACM International Conference on Management of Data (SIGMOD), Canada (2008), 227-238.
7. Bast, H., Majumdar, D., Schenkel, R., Theobald, M. and Weikum, G. "IO-Top-k: index-access optimized top-k query processing". Proceedings of the International Conference on Very Large Databases (VLDB), Korea (2006), 475-486.
8. Carey, M., Kossman, D. "On saying "Enough already!" in SQL". Proceedings of the ACM SIGMOD Conference on Management of Data, Vol. 26, No.2 (1997), 219-230.
9. Deshpande, P., Deepak, P., and Kummamuru, K. "Efficient Online Top-K Retrieval with Arbitrary Similarity Measures". Proceedings of the International Conference on Extending Database Technology (EDBT), France (2008), 356-367.
10. Chang, K. And Hwang, S-W. "Optimizing access cost for top-k queries over Web sources: A unified cost-based approach". Technical Report UIUCDS-R-2003-2324, University of Illinois at Urbana-Champaign (2003).
11. Fagin, R. "Combining fuzzy information from multiple systems". Journal of Computer and System Sciences (JCSS), Vol.58, No.1 (1996), 216-226.
12. Vlachou, A., Doulkeridis, C. Noorvaeg, K. Vazirgiannis, M. "On Efficient Top-k Query Processing in Highly Distributed Environments". Proceedings of the ACM International Conference on Management of Data (SIGMOD), Canada (2008), 753-764.

13. Goncalves, M and Vidal, M.E. "Preferred Skyline: A hybrid approach between SQLf and Skyline". Proceedings of Database and Expert Applications (DEXA), Denmark (2005), 375-384.

14. Goncalves, M and Vidal, M.E. "Top-k Skyline: A Unified Approach". Proceedings of OTM (On the Move) 2005 PhD Symposium, Cyprus (2005), 790-799.

15. Lee, G. Y. J. and Hwang, S. "Telescope: Zooming to interesting skylines". Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA), Thailand (2007), 539-550.

16. Lin, X., Yuan, Y., Zhang, Q. and Zhang, Y. "Selecting Stars: the k most representative skyline operator". Proceedings of the International Conference on Data Engineering (ICDE), Turkey (2007), 86-95.

17. Lo, E., Yip, K., Lin, K-I. and Cheung, D. "Progressive Skylining over Web-Accessible Database". Journal of Data and Knowledge Engineering, Vol. 57, No. 2 (2006), 122-147.

18. Papadias, D., Tao, Y., Fu, G. and Seeger, B. "Progressive Skyline computation in database systems". ACM Transactions Database Systems, Vol. 30, No. 1 (2005), 41-82.

Recibido el 7 de Marzo de 2008
En forma revisada el 9 de Marzo de 2009