Tri-Layer Resolution: A Coalesced Approach for Indexing Top-k Queries

Ritika Singhal
Computer Science and Engineering
New Horizon College of Engineering
Bangalore, INDIA

N. Deepika
Sr. Asst. Professor
Computer Science and Engineering
New Horizon College of Engineering
Bangalore, INDIA

***Abstract* - In the world of changing scenarios and uncertainty, users sometimes specify values for certain attributes in a query, without asking for the exact match in return. The result is termed as a "ranked query" or a "top-k" query. In this paper, we go for advancement in indexing such ranked queries on the basis of Partitions and Layered techniques. We employ horizontal partitioning first, thereby reducing size of datasets. This partitioning technique is well-versed in Oracle and even in Hive. Then the dual-resolution layer that consists of coarse-level and fine-level layers, is applied. So we propose a combinatorial technique to make the processing of top-k queries faster and more efficient.**

Keywords – top-k query, partitioning, skyline, convex skyline

I. INTRODUCTION

Approximation and Generalization are the keywords for the Ranked queries. The user specifies a ranking function or a scoring function. This function's value is calculated on every data row for a specified set of attributes and the output is that each data row has a rank associated with it. (Something similar to applying a hash function) Then the top-k rank holders are returned as the desired output.

Example 1.1 Consider a user interested in finding a Hotel room in a city where combined cost of the room and the cost of travelling to the famous tourist attraction is minimum. The user is only interested in five such Hotels.

So here, the scoring function is:Min(room price, travel price). This can be depicted graphically as shown in Fig. 1. In the figure, w specifies the scoring function given by the user. The dots represent some hotels. The scoring function is calculated as an average of both the coordinates.

The naïve way to answer this is to perform a join of rows with cheapest room rates from Hotel database and rows of least distance to tourist point from Tourism Database. But this is indeed costly. In general terms, to process a top-k query, a naive method would calculate the score of each tuple according to the score function, andthen, finds the top k tuples by sorting the tuples based on their scores. This method, however, is not appropriate for a querywith a relatively small value of k over large databases because it incurs a significant overhead by reading even those tuples that cannot possibly be the results.

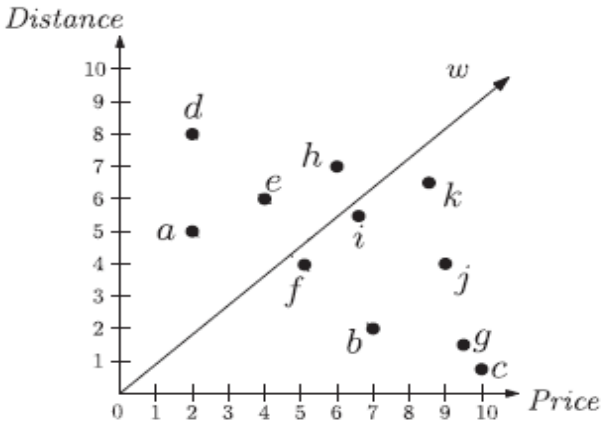

Fig. 1. Sample DataSet

There have been a number of methods proposed to efficiently answer top-k queries by accessing only a subset of the database instead of unnecessarily accessing the entire one.The techniques formulated till now for such efficient processing of top-k queries can be categorized into 3 approaches:

1. *Layer-based approach:*

This approach constructs a global index by dividing the records into consecutive layers. These layers are divided on the basis of all attributes. There exists one-to-one mapping between a tuple and a record in certain layer. This mapping is the key of traversal. Any top-k query can be answered by fetching records from at most k-layers. ONION[2] and AppRI use this approach.

ONIONbuilds the index by making layers with the vertices of the convex hulls over the set of tuples represented as point objects in the multi-dimensional space. That is, it makes the first layer with the convex hull vertices over the entire set of tuples, and then, makes the second layer with the convex hull vertices over the set of remaining tuples, and so on. As a result, an outer layer geometrically encloses inner layers.

AppRI constructs the database in much finer layers than ONION does by pre-computingthe domination relation for all the pairs of tuples in the database.

2. *List-based approach:*

This approach uses sorted lists for each attribute. The result for the top-k query is fetched by aggregating the

values from the lists searched according to the attributes given in the query. The search goes through the list in Round-Robin fashion. The lists are independent of each other. Also the efficiency of this approach decreases as the number of attributes mentioned in the queryincreases. The Threshold Algorithm (TA) uses this approach.TA and its variants are useful for a distributed database environment where exploiting the attribute correlation is difficult due to vertical partitioning. However, the performance of these methods are penalized due to lack of exploiting the attribute correlation

3. View-based approach:

This approach uses lots of space. The answers to a class of queries on every subset of attributes is pre-computed and stored. Whenever the query demands the top-k answers, the pre-computed results are scanned and the query is answered. If there are no exact pre-computed matches, then the nearest matching answers are returned. As a result this approach has no guarantee for efficiency. PREFER[6] and LPTA uses this approach.

Out of all the above approaches, the layered approach is the most efficient one. In this paper, we first focus on reducing the size of concerned dataset by partitioning. Then we go for an optimized layer based approach which guarantees that the first k layers include top-k answers regardless of the scoring function. This approach focuses on the relationship between tuples in adjacent layers at a finer granularity.

So the Tri-Layer approach has following three layers: (1) Partitioning the dataset on the basis of some partitioning columns or keys. (2) Coarse-level layers build on skylines which exploit dominance relationships. (3) Fine-level layers build on convex skylines which exploit relaxed dominance relationships.

II. PRELIMINARIES

We define some notations and definitions before we formally define the problem.

Let R be a relation containing tuples t_i with cardinality n. The attributes A_i are defined over domains dom(A_i) and degree of R is d. A top-k query also consists of a linear combination scoring function

$$F(t) = \sum_{i=1}^{d} w_i t_i$$

Where w_i is the user-specific weight function.

Also, $\sum_{i=1}^{d} w_i = 1$. Thus the scoring function F preserves the monotonicity.

Definition 1(Top-k Query): Given a scoring function F and the retrieval size k, a top-k query returns an ordered set of tuples such that $F(t_1) \leq \cdots \leq F(t_k)$.

Defintion 2(Dominant tuple): A tuple t dominates another tuple t' if t is better than t' in all the attributes.

Definition 3(Skyline)[1]: A tuple t is a skyline if any other tuple t' does not dominate t on all attributes A_i. A skyline is a set of skyline tuples, denoted as SKY(R).

Definition 4(Convex Skyline)[1]: A tuple t is a convex skyline if it has the minimum score for any linear combination function F. A convex skyline is a set of convex skyline tuples, denoted by CSKY(R).

Definition 5(Convex Hull)[8]: The convex hull of a set X of points is the smallest convex set that contains X. An object is convex or belongs to a convex set if for every pair of points within the object; every point on the straight line segment that joins the pair of points is also within the object.

TABLE I. Description of Notations

Symbols	Semantics
R	The target relation
n	Cardinality of R
d	The number of attributes in R or the degree of R
A_i	The ith attribute of R
t_i	A tuple in R
w[i]	The weight vector over the i^{th} attribute
L_i or L^i	The i^{th} layer

III. COALESCING LAYERS

In this section, we study how partitioning can be done and then we move on to the two-level layers[1], which can work on each partition. The result is an efficient retrieval of top-k queries.

A. Partitioning

Partitioning allows tables and indexes to be subdivided into smaller pieces, enabling these database objects to be manages and accessed at a finer level of granularity. Each partition has its own name and some storage characteristics. Each row in a partitioned table is unambiguously assigned to a single partition. The partitioning key is comprised of one or more columns that determine the partition where each row will be stored.

Partitioning can be done via three techniques:

1) *Range*: in this the records are classified into partitions based on some ranges of values of the partitioning key. This is most common form of partitioning.
2) *Hash*: It maps data to partitions based on the hashing algorithm applied to the partitioning key. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size.
3) *List*: it enables the user to explicitly control the mapping of records into partitions by specifying a list of discrete values for the partitioning key in the description of each partition.

Partitioning is well supported with Oracle. If we are dealing with Big Data, then Hive also lets the user to create partitions.

The main advantage of partitioning in top-k query retrieval is that the dataset to be worked upon is reduced in size. Hence the access to the required records is faster. Further, each partition can be added, modified or dropped according to the partitioning key as per the trend of user queries.

Considering the same example, partitioning can be shown graphically as in Fig. 2.

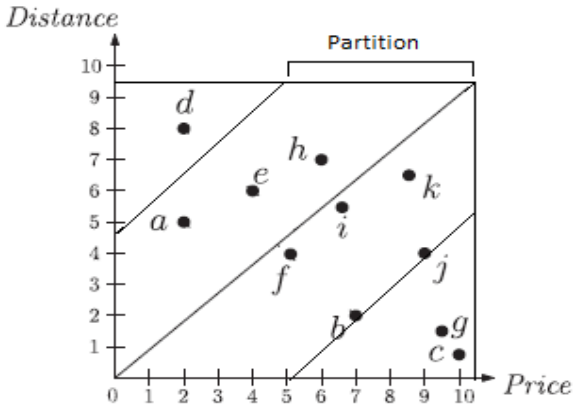

Fig. 2. Partitioned DataSet

In Fig. 2. the dataset is divided in four different partitions. These partitions contain different tuples distributed based on some partitioning key. The selection of the partitioning key and the partitioning technique is dependent on the database administraor.

This partitioning forms the first layer in top-k query traversal. The basic notion is based on the heuristics that the user queries will point to some common partitioning keys. Partitioning the dataset into layers, thus, involves some manual intervention also.

B. Two-level Layer

The dual-resolution layer is then built on every partition in concern. The dual-resolution layer has the following two phases:

1) *Coarse-level layer construction*: in this the skyline layers are built sequentially.The first layer L_1 is the Skyline SKY(R).Considering our example, one particular partition can have 3 layers as shown.

L1 = {k}
L2={i,j}
L3={f,b}

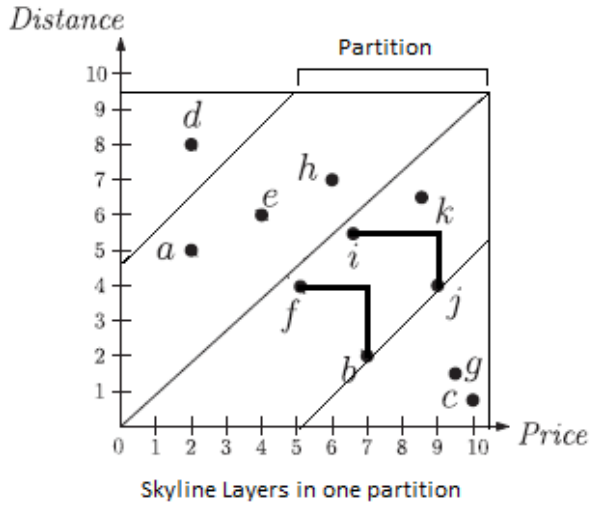

Fig. 3. Skyline Layers in one partition
Fig. 4.

2) *Fine-level layer construction*: We split the coarse-level layer into consecutive convex layers at a finer granularity. Let L_{ij} be the j^{th} fine-level layer L_{i1} in L_i coarse-level layer. The first fine-level layer L_{i1} in L_i is the convex skyline CSKY(L_i).

When these layers are built up, each tuple in R is associated with a certain layer. Then, given a scoring function F and retrieval size k, the top-k answers are identified by traversing the layers through relationships between tuples. In this process, some tuples are selectively accessed as the candidate tuples of the top-k answers. The relationships between the tuples in these layers hold an important place. To define these relationships, first, the dominance relationship for coarse-level layers can be used in the dominant graph, called DG[3].

*Dominance between coarse layers:*For any scoring function F, if t $\in L_i$ dominates t'$\in L_{i+1}$, the score F(t) of t is always smaller than the score F(t') of t'. This relationship is also known as ∀-dominance.

Second, a relaxed dominance relationship between adjacent fine-level layers can be used for further selective access. Specifically, the coarse-level layer is divided into multiple fine-level layers. In this case, because no dominance relationship holds for adjacent fine-level layers, we adopt the relaxed dominance relationship for the fine-level layers.

We introduce an ∃-dominance set (EDS). We consider a hyperplane H spanning a set of d tuples in d-dimensional space. Given a hyperplane H and a tuple t', it is said that the tuple set on hyperplane H is an EDS of tuple t' if H is closer to the origin than tuple t'. It is said that every tuple t $\in$EDS ∃-dominates t'.

∃-dominance between fine layers:For any scoring function F, if t $\in L_{ij}$∃-dominates t'$\in L_{i(j+1)}$, the score F(t) of t is smaller than the score F(t') of t.

Basically, the ∃-dominance sets for L_{ij}have to encompass all tuples in$L_{i(j+1)}$, *i.e.*, every tuple in$L_{i(j+1)}$is connected by the ∃-dominance relationship for any tuple in L_{ij}.

Algorithm 1 BuildDualResolutionLayer($\mathcal{R}$)

Input: $\mathcal{R}$: a target relation on $\mathcal{A}$
Output: $\mathcal{L}$: a dual-resolution layer

```
1: L ← {} // Initialize a dual-resolution layer.
2: i ← 1 // Initialize the identifer of the coarse-level layer.
3: while R ≠ {} do
4:    L^i ← {} // Initialize the i-th coarse-level layer.
5:    j ← 1 // Initialize the identifier of the fine-level layer.
6:    S ← SKY(R) // Compute a skyline for R.
7:    while S ≠ {} do
8:       // Compute the j-th fine-level layer in L^i.
9:       L^ij ← CSKY(S) // Compute a convex skyline for S.
10:      if j > 1 then
11:         Update the ∃-dominance between L^i(j−1) and L^ij
12:      end if
13:      L^i ← L^i ∪ L^ij // Insert L^ij into L^i.
14:      S ← S − L^ij // Update current skyline S.
15:      j ← j + 1 // Update the identifier of the fine-level layer.
16:   end while
17:   if i > 1 then
18:      Update the ∀-dominance between L^i−1 and L^i
19:   end if
20:   L ← L ∪ L^i // Insert L^i into L.
21:   R ← R − L^i // Update target relation R.
22:   i ← i + 1 // Update the identifier of the coarse-level layer.
23: end while
24: return L
```

Considering our example, the dual-resolution is constructed as shown in Fig. 4.

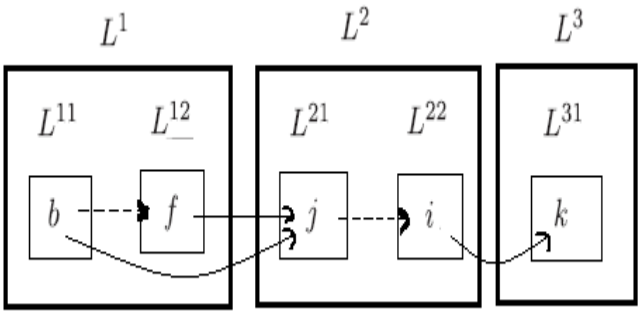

Fig. 5. Dual Resolution Layer Construction over one partition

The entire data in the concerned partition is divided into three coarse-level layers(shown by solid boxes. The relationships between these layers are shown by the solid arrows ie. ∀-dominance relationships. Each coarse-level layer is then divided into multiple fine-level layers (light smaller boxes). Then ∃-dominance relationships between fine-level layers are shown by dotted arrows.

Top-k query processing over the layer-based index can be viewed as a *graph traversal problem*. The tuples are then accessed on the basis of a filtering condition. Each tuple is now accessed on the basis of a status. This status is decided as follows.

Definition 6 (∀-dominance-free). A tuple t'∈$L_{i+1}$is ∀-dominance-free, if (1) t_ has no connected edge from t ∈$L_i$or (2) every tuple t ∈L_i connected to t'is in the top-(k − 1)answers.

Definition 7 (∃-dominance-free). A tuple t'∈$L_{i(j+1)}$ is ∃-dominance-free, if (1) t'has no connected edge from L_{ij} or(2) any tuple t ∈L_{ij} connected to t'is in the top-(k − 1)answers.

*Filtering condition:*A tuple t has to be only accessed if t is both ∀-dominance-free and ∃-dominance-free.

In this way, the retrieval of top-k queries is optimized and made more efficient with a three-layer resolution.

IV. CONCLUSION

This paper has studied the problem of designing a layer based index for supporting scalable top-*k* query computation.In particular, we focused on the optimization goal of making selective access to each layer in a specific partition. To pursue this optimization goal, we first reduced our dataset by choosing only one or more concerned partitions. Then we designed a two-level layer, in which each coarse-level layer was further divided into multiple fine-level sub layers at a finer granularity.

ACKNOWLEDGMENT

This paper is written for the purpose of paper submission in IJERT. The authors thank the reviewers for their feedback and to all the people involved in this study.

REFERENCES

[1] J.Lee, H.Cho,S.Lee and S-won Hwang, "Towards ScalableIndexing for Top-k Queries" in IEEE Transactions on Knowledge and data engineering, Vol 26, No. 12, Dec 2014.

[2] Y.-C. Chang *et al.*, "The onion technique: Indexing for linearoptimization queries," in *Proc. ACM SIGMOD*, Dallas, TX, USA,pp. 391–402, 2000.

[3] L. Zou and L. Chen, "Dominant graph: An efficient indexingstructure to answer top-k queries," in *Proc. IEEE 24th ICDE*, 2008,pp. 536–545.

[4] J. Heo, J. Cho, and K. Whang, "The hybrid-layer index: Asynergic approach to answering top-k queries in arbitrary subspaces,"in *Proc. IEEE 26th ICDE*, Long Beach, CA, USA, 2010,pp. 445–448.

[5] L. Zou and L. Chen, "Pareto-based dominant graph: An efficientindexing structure to answer top-k queries," *IEEE Trans. Knowl.Data Eng.*, vol. 23, no. 5, pp. 727–741, May 2011.

[6] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "PREFER: Asystem for the efficient execution of multi-parametric rankedqueries," in *Proc. 2001 ACM SIGMOD*, pp. 259–270.

[7] J.-S. Heo, K.-Y. Whang, M.-S. Kim, Y.-R. Kim, and I.-Y. Song, "Thepartitioned-layer index: Answering monotone top-k queries usingthe convex skyline and partitioning-merging technique," *Inf. Sci.*,vol. 179, no. 19, pp. 3286–3308, 2009.

[8] http://en.wikipedia.org/wiki/Convex_hull

Ritika Singhal is an M.Tech. student with New Horizon College of Engineering, Bangalore, India. She received her B.Tech. degree in Information Technology at Kurukshetra University, Kurukshetra, India. Her current research interests include database systems and query languages.

N. Deepika is a Senior Assistant Professor with Department of Computer Science and Engineering at New Horizon College of Engineering, Bangalore, India. She received the M.Tech. degree in computer science from JNTU, Hyderabad, India. Her current research areas include data mining, web mining and computer networks.