International Conference on Methods and Models in Computer Science, 2009

Top-k Query Processing with Multidimensional Range Search

Shiva Rudrani[1], Saleena N[1]

[1]*Department of Computer Science and Engineering*
National Institute of Technology Calicut,Calicut, India.
e-mail: rudrani.shiva@gmail.com saleena@nitc.ac.in

Abstract— An *m*-dimensional top-*k* query (with *m* search conditions) is primarily processed by scanning the corresponding *m* index lists in descending score orders in an interleaved manner (and by making judicious random accesses to look up index entries of specific data items). In this paper a new algorithm is proposed that makes use of a data structure that facilitates multidimensional range search. An *m*-dimensional top-*k* query can be processed by searching for the data items that satisfies a range of score over each dimension. At every step of the algorithm a new set of ranges (one for each dimension) is specified such that more accurate tuples are added in the candidate top-*k* set. The process continues till we get the actual top-*k* data items. The incremented range set is specified with the help of the statistics of the distribution of data items in *m*-dimensional space. Thus, efficiency of the algorithm very much depends on the proper study and analysis of the distribution of the data items in the *m*-dimensional space and the data structure used.

***Keywords*-Top-k Query Processing; Information Retrieval; Multidimensional Range Search; Database Indexing; TA; FA.**

I. INTRODUCTION

Top-k query processing is a key building block for data discovery and ranking, and has been intensively studied in the context of information retrieval, multimedia similarity search, text and data integration, business analytics, preference queries over product catalogs, Internet-based recommendation sources, distributed aggregation of network logs and sensor data, and many other important application areas. Top-k queries operate on index lists for a query's elementary conditions and aggregate scores for result candidates. One of the best implementation methods in this setting is the family of threshold algorithms(TA), which aim to terminate the index scans as early as possible based on lower and upper bounds for the final scores of result candidates. This procedure performs sequential disk accesses for sorted index scans, but also has the option of performing random accesses to resolve score uncertainty. This entails scheduling for the two kinds of accesses:

1. The prioritization of different index lists in the sequential accesses, and
2. The decision on when to perform random accesses and for which candidates.

TA also incurs a lot of useless accesses to the lists. Here the data items can be any search item. For example, in context of database, it may be tuples from a table or in context of internet search, it can be web documents. Since the background for this paper is the Threshold Algorithm (TA), in this section we briefly describe TA.

In the TA or FA model, we assume that each database consists of a finite set of objects. We shall typically take N to represent the number of objects. Associated with each object R, are m fields x1, x2, ..., xm where $x_i \in [0,1]$ for each i. We may refer to xi as the ith field of R. The database can be thought of as consisting of a single relation, where one column corresponds to the object id, and the other columns correspond to m attributes of the object. Alternatively, the way we shall think of a database in TA model as consisting of m sorted lists L1, ..., Lm, each of length N (there is one entry in each list for each of the N objects). We may refer to Li as list i. Each entry of Li is of the form (R, xi), where xi is the ith field of R. Each list Li is sorted in descending order by the xi value. We are taking into account only access costs, and ignoring internal computation costs. Thus, in practice it might well be expensive to compute the field values, but we ignore this issue here, and take the field values as being given.

TA considers two modes of access to data. The first mode of access is sorted (or sequential) access. Here the middleware system obtains the grade of an object in one of the sorted lists by proceeding through the list sequentially from the top. Thus, if object R has the lth highest grade in the ith list, then l sorted accesses to the ith list are required to see this grade under sorted access. The second mode of access is random access. Here, the middleware system requests the grade of object R in the ith list, and obtains it in one random access. If there are s sorted accesses and r random accesses, then the sorted access cost is scS the random access cost is rcR, and the middleware cost is scS + rcR (the sum of the sorted access cost and the random access cost), for some positive constants cS and cR.

A. *The Threshold Algorithm (TA) [1]*

Do sorted access in parallel to each of the m sorted lists Li. As an object R is seen under sorted access in some list, do random accesses to the other lists to find the grade xi of object R in every list Li. Then compute the grade t(R) = t(x1, x2, ... , xm) of object R, where t is monotone aggregation function. If this grade is one of the k highest we have seen then remember object R and its grade t(R) (ties are broken arbitrarily, so that only k objects and their grades need to be remembered at any time).

For each list Li, let xi be the grade of the last object seen under sorted access. Define the threshold value τ to be t(x1, x2, ..., xm). As soon as at least k objects have been seen whose grade is at least equal to τ then halt.

Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(R, t(R)) \mid R \in Y\}$.

B. *The idea behind the proposed Algorithm*

We already have m lists L1, ..., Lm corresponding to m attributes/keywords. These lists consist of local scores of objects corresponding to assigned attributes. Assigned scores within a list form a range. Thus, we have m ranges of scores. Now consider an m-dimensional space. Without loss of generality each of the m lists can be assigned one of the dimensions and it is always possible to do this.

Now, if we fix a range of scores for each dimension and do a multidimensional range search (using some suitable data structure) then we can get as result a set of tuples that comes under specified ranges for all the lists. Thus, if ranges are specified such that it will give good candidate tuples for top-k, then the final top-k tuples can be obtained out of the candidate tuples. In correspondence to the TA, the set Y is the set of candidate tuples. Using this idea we can get the top-k results. The algorithm is discussed in detail in section V. This method can be useful over TA in many aspects, among which, important ones are:-

- The overhead of scheduling sorted access (SA) and random access (RA) is removed.
- The algorithm will stop scanning much early (provided input to the algorithm is based on the statistics).
- Running Time of the proposed algorithm is better than TA and based on the efficiency of the data structure used for doing multidimensional range search.

II. PROBLEM DEFINITION

The problem that is addressed in this paper, is to overcome the overhead of scheduling index-access steps in TA-style top-k query processing, early stopping of scanning and improving the time complexity. "How to leverage the statistics to find the cut-off ranges?" is also studied, so that top-k results can be obtained in least possible number of multidimensional range searches.

III. MOTIVATION

Implementation methods based on threshold algorithms (TA) perform sequential disk accesses for sorted index scans, but also have the option of performing random accesses to resolve score uncertainty. This entails scheduling for the two kinds of accesses:

the prioritization of different index lists in the sequential accesses, and

the decision on when to perform random accesses and for which candidates.

The proposed algorithm, completely discard the use of such scheduling. TA also incurs a lot of useless accesses to the lists.

When scanning multiple index lists (over attributes from one or more relations or document collections), top-k query processing faces an optimization problem: combining each pair of indexes is essentially an equi-join (via equality of the tuples or document ids in matching index entries), and we thus need to solve a join ordering problem. As top-k queries are eventually interested only in the highest score results, the problem is not just standard join ordering but has additional complexity. The proposed Algorithm does not have to consider such problems and uses an entirely different approach to match tuples or documents under given attribute/keywords.

IV. COMPUTATIONAL MODEL FOR THE PROPOSED ALGORITHM

For the explanation of the algorithm, the case of a database system (as is used by the TA model, discussed above) is taken, but without loss of generality, the algorithm can be used for any relevant systems. The input to the TA is a set of sorted lists. For our discussion we can visualize the proposed algorithm as scanning over sorted lists parallely from top to bottom. We also visualize these lists put adjacent to each other so that each row of each list is at same level of the corresponding rows of the other lists or in other words, these lists, when put adjacent to each other forms a matrix. Now each row of this matrix will be referred by the index variable "j". These lists are used to determine new (incremented) ranges during the running of the algorithm. Note that actual implementation of the proposed algorithm may not even require these lists at all. What we need, is to index the database with a data structure that supports multidimensional data (e.g. kd-tree[4], UB-tree[5] etc). These lists are helpful in finding the new lower bounds, so that we can obtain a new set of ranges for next search. If we can guess the new set of ranges (for example, by analyzing the distribution of data items in m-dimensional space), then we may not need these lists.

To make the algorithm clear and simple, a baseline algorithm is discussed first. A more concrete algorithm is presented in the implementation section of this paper. We name the proposed algorithm as 'MRS-Top-k' which can be expanded as 'Multidimensional Range Search Top-k'. Following is the Baseline algorithm.

A. Baseline MRS-Top-k Algorithm

Each list Li is sorted in descending order by the xi value.

1. Determine the ranges <L1[j].Score , L1[0].Score>, <L2[j].Score , L2[0].Score> , ... , <Lm[j].Score , Lm[0].Score> for each list where Li[j] being the jth entry of ith list. That is, if the lists are concatenated next to each other, it will form a matrix and we have to find the entries in jth row for each column. The value of j can be calculated based on statistics of previous search or analysis of distribution of data.
2. Do a multidimensional range search with the specified ranges on their respective axes.
3. If we get less than k results then we repeat the search with the ranges <L1[j'].Score , L1[0].Score>, <L2[j'].Score , L2[0].Score> , ... , <Lm[j'].Score , Lm[0].Score> for each list, where j' > j and j' - j = c, where c is a positive integer.
4. Do j = j'.
5. Repeat steps 2 and 4 till we get k or more results.
6. After getting k or more than k results, we sort the retrieved records in descending order of their aggregate value and check whether the first k records among the sorted records have the top k aggregate score among all the records. This checking is based on a condition which is explained in the concrete algorithm in section V. If we get the final top k results then we exit else we increment the ranges as in step 3 and 4.
7. Do a multidimensional range search with the new ranges and sort the retrieved records according to the aggregate score.
8. Check whether the top k records among the sorted records have the top k aggregate score among all the records. If it is then we exit else we increment the ranges and repeat from step 7.

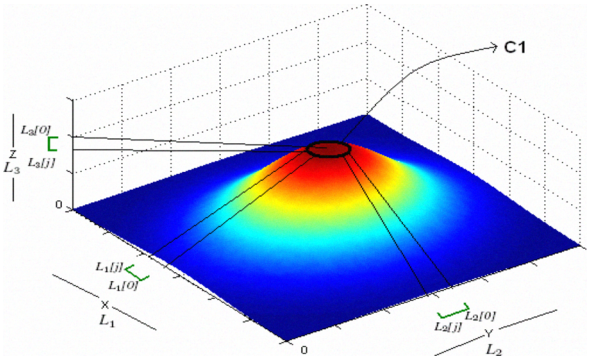

Fig. 1: 3D Normal distribution with high density at top.

B. Leveraging statistics to find the cut-off ranges

Let us consider the 3D distribution of scores under the attributes A1, A2 and A3 as depicted by Fig. 1. If the density of the distribution on the top of the curve is high, the ranges <L1[j].Score , L1[0].Score>, <L2[j].Score , L2[0].Score> and <L3[j].Score , L3[0].Score> on lists L1, L2 and L3 respectively, are considerably small and vice-versa. In Fig. 1, the points above the circle C1 may be considered if the density is high at top.

Similarly if the density is uniform, the points above the circle C2 (see Fig. 2) may be considered, as they cover comparatively larger range on each dimension.

Finally, if we are not certain about the distribution, we can refer to Fig. 3.

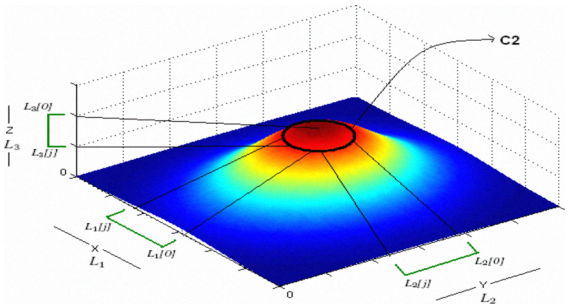

Fig. 2: 3D Normal distribution with uniform density.

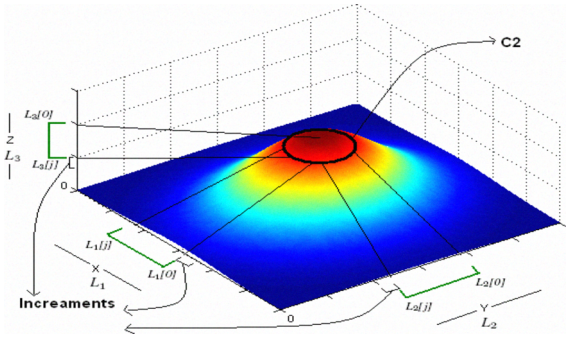

Fig. 3: 3D Normal distribution with unknown density.

V. IMPLEMENTATION

We first start with discussing, what are the inputs required by the proposed MRS-Top-*k* algorithm. The algorithm asks user to input the value of *k* in top-*k*. It also requires a set of initial ranges and an *increment* value (a positive integer, basically indicating the number of rows by which to move down the matrix). As discussed in computational model that each row of the matrix will be referred by the index variable '*j*', so for setting up the initial ranges, we can identify a row i.e. give a value to '*j*' so that for each column, the initial range will be from the value at that row to value at the top most row. With these initial ranges, the algorithm does the first range search. For the next search, the new range set is calculated with the help of increment value.

For the proposed algorithm, a data structure suitable for multidimensional range search is required. There are

many available data structure like kd-tree, hB-tree, UB-tree etc. Clearly, the chosen data structure is also implemented and populated with scores, as discussed in the previous section. As mentioned before, for terminating the running of the algorithm, some condition is required. The checking is done on the following variables.

CurMinAgg = Minimum Aggregate of retrieved records in the current top-k search.

CurMaxAgg = Maximum Aggregate of retrieved records in the current top-k search.

PrevMinAgg = Minimum Aggregate of retrieved records in the previous top-k search.

PrevMaxAgg= Maximum Aggregate of retrieved records in the previous top-k search.

That is if more than k or equal to k records are found then sort them according to aggregate score in descending order. Let the sorted List of Retrieved Records be L, such that L[1].Agg is the Maximum Aggregate score of the top k records. Similarly L[k].Agg is the Minimum Aggregate score of the top k records.

Thus,

CurMinAgg = L[k].Agg and CurMaxAgg = L[1].Agg.

Therefore a more concrete algorithm can be as follows.

MRS- Top-k Algorithm

Input: value of k, Initial Ranges (value of j i.e. row number), Increment

1. Range Search with Initial Ranges
   <L1[j].Score,L1[0].Score>,
   <L2[j].Score,L2[0].Score> , ... , <Lm[j].Score , Lm[0].Score> for each list.
2. If (number of retrieved records < k)
   Increment the ranges i.e. j = j + Increment.
3. Range Search with new range set.
4. Repeat step 2 and 3 till (number of retrieved records) ≥ k
5. If (number of retrieved records ≥ k)
   Sort according to aggregate score
6. Set,
   CurMinAgg = L[k].Agg
   CurMaxAgg = L[1].Agg
7. j = j + Increment, range search with new ranges.
8. Set,
   PrevMinAgg = CurMinAgg
   PrevMaxAgg = CurMaxAgg
   Calculate the CurMinAgg and CurMaxAgg as in Step6.
9. If ((PrevMinAgg == CurMinAgg) and
   (PrevMaxAgg == CurMaxAgg) and
   $(\forall t \ ((\sum_{i \neq t}(L_i[0]) + \text{Lt[j]}) < \text{CurMinAgg}))$

   Where, i, t = 1, 2… m. m is the number of lists.

   Then, stop and return top-k records.

   Else, repeat from step 7.

VI. Result & Future work

The most efficient algorithm proposed so far for answering top-*k* queries over sorted lists is the Threshold Algorithm (TA). However, TA may still incur a lot of useless accesses to the lists. As mentioned in the section IV, we actually do not need sorted lists for running the algorithm. The lists were introduced to determine new set of ranges for next search, in a systematic way. Instead, algorithm can be provided with new set of ranges in an interactive way. With the help of lists we can easily visualize the running of the algorithm. Also, when we do a multidimensional range search (as explained in section VI) with a suitable data structure, we try to optimize the search by minimizing the search area. While traversing the tree from top to bottom our intension is to be as close to the given ranges as possible. Finally, we get the data items that exactly satisfy the ranges. Thus, we find that the need for sorted access and random access is discarded in the proposed algorithm. Also, the proposed algorithm does not do scheduling of random accesses and sequential accesses, thus this overhead is also relieved.

Since the proposed algorithms involve multidimensional range search, it becomes very essential to correctly judge the data structure according to the application. The complexity of the algorithm depends upon the complexity of the range search performed on the selected data structure. In most cases this complexity depends on the height of the tree. If we consider a height balanced tree then height of the tree is approximately O(log*N*), where *N* is the number of records in the database. kd-tree is not that efficient when $k > 10$. UB-tree is more efficient for the multidimensional range search. The number of times the range search will be done depends upon the value of *k* in top-*k*. Whereas, TA in its worst case takes the running time of Fagine's Algorithm i.e. $O(N^{(m-1)/m} K^{1/m})$.

As a future work, we can develop an statistic analyzer which should analyze the distribution of the data items in *m*-dimensional space and give more accurate input to the algorithm(i.e. the initial range and increment) so that algorithm will stop much early and should perform minimum number of range searches. Usually such distributions follows some curve for example say normal curve or follow some pattern, thus it is always possible to develop such an analyzer. If we can do so, this algorithm should give excellent performance with large databases.

Also, if someone can come up with a more strong terminating condition for the algorithm then that will be an edge. Because we can easily find that the algorithm's terminating condition is very weak, this is the root cause that the efficiency of the algorithm depends on the statistics of the distribution of the data tuples in *m*-

dimensional space. Once the above goals are achieved, we can test the performance of the proposed algorithm in real applications.

REFERENCES

[1] R. Fagin, A. Lotem and M. Naor., "Optimal aggregation algorithms for middleware", PODS Conf, 2001.

[2] U. Guntzer, W. Kiessling and W.-T. Balke., "Towards efficient multi-feature queries in heterogeneous environments.", IEEE Int. Conf. on Information Technology, Coding and Computing (ITCC), 2001.

[3] S. Nepal and M.V. Ramakrishna., "Query processing issues in image (multimedia) databases.", ICDE Conf., 1999.

[4] Andrew W. Moore., "An introductory tutorial on kd-trees, Extract from Andrew Moore's PhD Thesis: Efficient Memory-based Learning for Robot Control PhD. Thesis. ", Technical Report No. 209, Computer Laboratory, University of Cambridge., 1991, pp. 10-29.

[5] Rudolf Bayer,, "The Universal B-Tree for multidimensional Indexing.", TUM-I9637., November 1996.

[6] S. Michel, P. Triantafillou and G. Weikum. , "KLEE: A Framework for Distributed Top-k Query Algorithms.", VLDB Conf., 2005.

[7] Reza Akbarinia, Esther Pacitti, Patrick Valduriez. "Best Position Algorithms for Top-k Queries", VLDB'07 Vienna, Austria, September 23-28, 2007.

[8] Thomas H. Corman, Charles E. Leiserson, Ronald L. Rivest, "Introduction to Algorithm", MIT Press, Cambridge, MA, USA, 1990