ISSN 2047-3338

Fast Nearest Neighbor Search Using Efficient Spatial Index

Eldhose Paul[1] and Ierin Babu[2]

[1,2]Computer Science and Engineering, ASIET, Kalady, India

Abstract— **Spatial queries, such as nearest neighbor retrieval and range search, include only conditions on geometric properties. A spatial database handles multidimensional objects and offers quick access to those objects based on different range criteria. Many applications demand a novel form of queries to discover the objects that considering both a spatial predicate, and a predicate on their related texts on now-a-days. For example, considering all the hotels, a user would request for the hotel that is the nearest among those whose menus contain the specified keywords all at the same time. Here deals with the new method called Efficient Spatial Index that extends the conventional inverted index to handle with multidimensional data, and arises with algorithms that can response nearest neighbor queries with keywords in real time.**

Index Terms— **Spatial Inverted List, Spatial Queries, IR^2-Tree and Multidimensional Data**

I. INTRODUCTION

INFORMATION retrieval is the activity of discovering information from a pool of information resources. Searches can be based on metadata or on full-text indexing. An information retrieval starts when a user enters a query into the system. User queries are matched against the database information. User queries are a broad term that actually denote to non-spatial queries [1], [2] and spatial query. The former refer to the names, phone numbers, email addresses of people whereas the latter refers to spatial elements such as points and regions. A spatial database handle multidimensional objects and provides quick access to those objects based on different selection criteria. IR^2-tree is used in the existing system for providing best solution for finding nearest neighbor.

This method has few deficiencies. So here deals with new method called efficient inverted index to improve the space and query efficiency. Inspired by this, develop a new access method called the efficient spatial index that extends the conventional inverted index to manage with multidimensional data, and comes with algorithms that can answer nearest neighbor queries with keywords in real time. The proposed technique is better than that of IR^2-tree in query response time significantly, often by a factor of orders of magnitude.

II. RELATED WORK

A multidimensional or spatial index, in contrast to a B+ tree [1] , utilizes some kind of spatial relationship to organize data entries, with each key value seen as a point (or region, for region data) in a k-dimensional space, where k is the number of fields in the search key for the index. In a B+ tree index [1], the two-dimensional space of (age, salary) values is linearized. In contrast, a spatial index stores data entries based on their proximity in the underlying two-dimensional space. A space-filling curve [1] executes a linear ordering on the domain. The curve used represents the Z-ordering curve for domains with two-bit representations of attribute values. Consider the point with X = 01 and Y = 11. The point has Z-value 0111, obtained by interleaving the bits of the X and Y values; we take the first X bit (0), then the first Y bit (1), then the second X bit (1), and finally the second Y bit (1). In decimal representation, the Z-value 0111 is equal to 7, and the point X = 01 and Y = 11 has the Z-value 7. Grid files [1], [4] rely upon a grid directory to identify the data page containing a desired point. When searching for a point, first find the corresponding entry in the grid directory. The grid directory entry identifies the page on which the desired point is stored, if the point is in the database.

The key idea of the R-Tree [5], [7] is to collect near objects and denote them with their minimum bounding rectangle in the next higher level of the tree; the "R" in R-tree stands for rectangle. Since all objects lie within this bounding rectangle, a query that does not cross the bounding rectangle also cannot cross any of the contained objects. At the leaf level, each rectangle defines a single object; at higher levels the combination of an increasing number of objects. This can also be seen as an increasingly coarse approximation of the data set. An R+ tree [6] is a method for looking up data using a location, often (x,y) coordinates, and often for locations on the surface of the earth. An R+ tree is a tree data structure, a modified form of the R tree, used for indexing spatial information. R+ trees are a concession between R-trees and kd-trees: they escape overlapping of internal nodes by introducing an object into multiple leaves if necessary. Coverage is the total area to cover all connected rectangles. Overlap is the total area which is contained in two or more nodes. R* trees [7] is another modified form of R-trees used for indexing spatial information. R*-trees have a little higher implementation cost than standard R-trees, as the data may need to be reinserted; but the resulting tree will usually have an improved query performance. IR^2-Tree [10] is combination of two concepts: R-tree, a standard spatial index [8], and signature file [9], a better method for keyword-based document retrieval. By doing so they develop a structure called the IR^2-tree [10], which has the powers of both R-trees and signature files. Like R-trees, the IR^2- tree preserves objects spatial proximity, which is the key to solving spatial queries efficiently. As with many new solutions, the IR^2-tree also has a few disadvantages that affect its efficiency. The most important one of all is that the number of false hits can be really large when the object of the final result is far away from the query point, or the result is simply empty.

III. METHODOLOGY

A) Efficient Spatial Index

The essential compressed version of I-index with embedded coordinates is, Efficient Spatial Index. Using, Efficient Spatial Index we can do query processing by two ways, either by merging or by together with R* tress in a distance browsing method. The defect of conventional I-index is eliminated by merging because, Efficient Spatial Index use small amount of space.

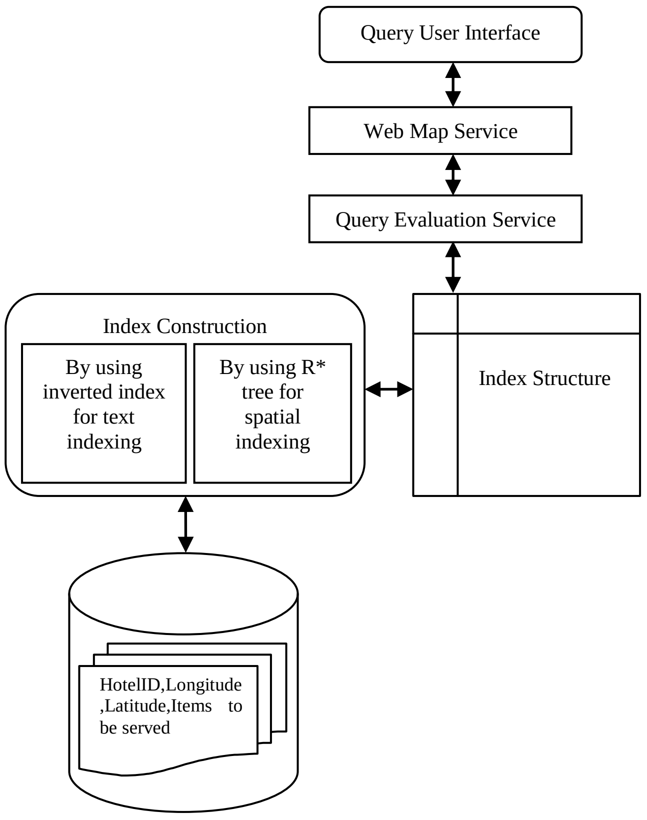

Query User Interface
Web Map Service
Query Evaluation Service
Index Construction
By using inverted index for text indexing
By using R* tree for spatial indexing
Index Structure
HotelID,Longitude ,Latitude,Items to be served

Fig. 1: Efficient Spatial Index Architecture

ESI Algorithm

Algorithm: ESI (q, BO, Kw)

```
finallist←NULL
L ← BO.result ( );
for i = 1 to L.length( )
p=L[i];
BCP ← compute BC( p );
if ( q not belongs to BCP ) then
return false; {the first NN fails}
else
if ( Kw belongs to p ) then
finallist.add ( p);
else
return false;
end if
end if
end for
for i = 1 to finallist.length( )
 for j = 1 to finallist.length( )
if( i not equal to j)
if(distance(finallist[i],q)=distance(finallist[ j]  ,q))
row[i].append(finallist[ j ]);
end if
end if
end for
end for
return true
```

Example: Consider the query top-2 hotels from point containing the keywords {“c”, “d”} on the data. The trace of ESI algorithm is the following on the basis Table 1 .Steps are explained below:

TABLE 1. INVERTED INDEX

Word	Inverted list
a	P1,P4
b	P1,P2,P7
c	P2,P3,P5,P6,P8
d	P2,P3,P6,P7,P8
e	P4,P5,P6,P7

1 P2, P3, P5, P6, P8 are returned by the inverted index for keyword “c”.
2 P2, P3, P6, P8, P7 are returned by the inverted index for keyword “d”.
3 P2, P3, P6, P8 are the result after the intersection.

1. Objects P6, P2, P3, P8 are accessed to get their coordinates.
2. Add P6 to list L= {(P6, 12)}
3. Add P2, P3, P8 to list L= {(P8, 18), (P2, 15), (P3, 52), (P6, 12)}.

4. Apply compression scheme that is convert {(P2,15,1) , (P3,52,1) , (P6,12,0), (P2,18,2)} into {(P6,12,0) , (P2,15,1) , (P8,18,2), (P3,52,6)}.And apply gap keeping and sorted with help of pseudo-id {(P6,12,0),(P2,15,1),(P8,18,2),(P3,52,6)} into {(P6,12,0),(P2,3,1),(P8,3,1),(P3,34,4) }.
5. Then the four point becomes {(0,12),(1,3),(1,3),(4,29)} then form R-tree.
6. The top-2 result is{P6,(P2,P8)} .

IV. EXPERIMENTAL RESULTS

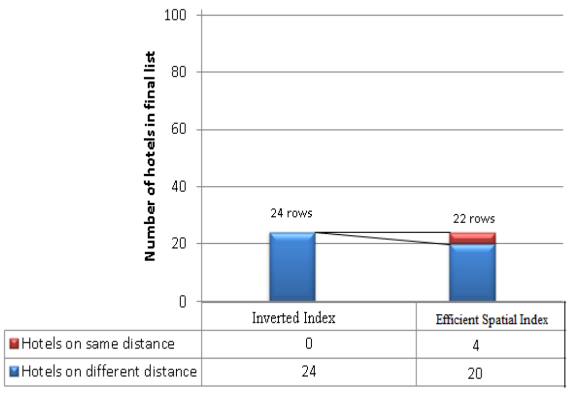
R* Tree after compression scheme, it is finished explaining how to build the leaf nodes of an R*tree on an inverted list. As each leaf is a block, all the leaves can be stored in a blocked Efficient Spatial Index. Building the non-leaf levels is trivial, because they are invisible to the merging-based query algorithms, and hence, do not need to preserve any common ordering. It is noteworthy that the non-leaf levels add only a small amount to the overall space overhead because, in an R* tree, the number of non-leaf nodes is by far lower than that of leaf nodes.

Theoretical analysis: Theoretical analysis of various spatial indices, its advantages and disadvantages. Next from a theoretical viewpoint that the compression scheme has a low space complexity as shown below. As the handling of each inverted list is the same, it suffices to focus on only one of them, denoted as L. Let us assume that the entire data set has p>=1 points and l of them appear in L. To make analysis general, take the dimensionality d into account. Also, recall that each coordinate ranges from 0 to m, where m is a large integer. Naively, each pseudo-id can be represented with log p bits, and each coordinate with log m bits.

- Therefore, without any compression, inverted index can represent the whole L in O (l (log p + d log m)) bits.
- The compression scheme stores L with
  O (l (log (p/l) +log (m^d/l))) bits.

Proof: Our compression scheme essentially applies gap keeping to two sets of integers. The first set includes all the pseudo-ids of the points in L, and the second includes their Z-values. Every pseudo-id ranges from 0 to p- 1, while each Z-value from 0 to m^d-1. Hence, from 1, the space needed to store all r pseudo-ids is l (log (p/l)), and the space needed to store r Z-values is l (log (m^d/l)).

Experimental Evaluation based on space complexity: Efficient Spatial Index (ESI) conserve the spatial area of data points, and comes with an R* tree build on every inverted list at small space overhead. It contains the set of points and the points are related to the set of keywords and the keywords are related to derive the set of documents. Here check for the hotels that contains items {sandwich, burger, pizza} and evaluate in two methods: Inverted Index, Efficient Spatial Index Fig. 2.



	Inverted Index	Efficient Spatial Index
🟥 Hotels on same distance	0	4
🟦 Hotels on different distance	24	20

Fig. 2: Evaluation based on Space complexity

From the evaluation two, Efficient Spatial Index conserves the spatial area of data points, and small space overhead. It contains the set of points and the points are related to the set of keywords and the keywords are related to derive the set of documents.

Experimental Evaluation based on execution time: Another evaluation based on item count and time taken to execute. The time taken by SI Index [11], Efficient Spatial Index is 0.006, 0.087 sec for one item. The time taken by SI Index, Efficient Spatial Index is 0.112, 0.197 sec respectively for two items. The time taken by SI Index, Efficient Spatial Index is 0.345, 0.683sec for three items. Efficient Spatial Index use more time to execute can see from the Fig. 3, because it uses the compression scheme.

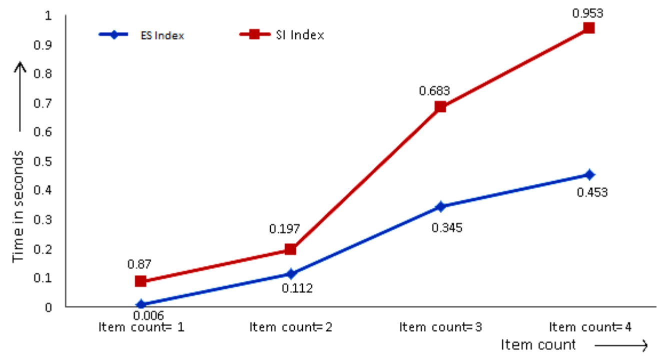


Fig. 3: Evaluation based on Execution time

From the evaluation two, the drawback is when keyword size has only a single word; the performance based on time is slight difference with compare to keyword size with two words. But keyword size increasing the more time it consumes.

V. CONCLUSION

The Efficient Spatial Index is using both capacity of the R* Tree and the processing of signature files. Related with the earlier work the existing systems are not efficient to provide the real time answers. In Efficient Spatial Index, the proposed concept of the list merging and distance alignment are used to help for searching, and the compression scheme is used to provide the effectiveness of the quick search and make the final list with low space complexity.

REFERENCES

[1] Raghu Ramakrishnan, Johannes Gehrke, “Database Management Systems”, Chapter 26 ,pp. 777-795, McGraw Hill, Third Edition, 2004.

[2] X. Cao, L. Chen, G. Cong, C.S. Jensen, Q. Qu, A. Skovsgaard, D.Wu, and M.L. Yiu, “Spatial Keyword Querying,” Proc. 31st Int’l Conf. Conceptual Modeling (ER), pp. 16-29, 2012.

[3] M Ester, HP Kriegel, J Sander “Spatial Data Mining,” A Database Approach, Springer 1997.

[4] J. Nievergelt, H. Hinterberger, K. C. Sevcik: “The grid file: An adaptable, symmetric multikey file structure,” ACM Trans. on Database Sys. 9, 1, 38-71 (1984).

[5] A Guttman “R-trees a dynamic mdex structure for spatial searching,” Proc ACM SIGMOD Int Conf on Management of Data, 47-57, 1984

[6] Timos K. Sellis, Christos Faloutsos “ The R+-Tree- A Dynamic Index for Multi-Dimensional Objects ,” In ACM Trans,1987

[7] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 322-331, 1990.

[8] C. Faloutsos and S. Christodoulakis, “Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation,” ACM Trans. Information Systems, vol. 2, no. 4, pp. 267-288, 1984.

[9] I.D. Felipe, V. Hristidis, and N. Rishe, “Keyword Search on Spatial Databases,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 656-665, 2008.

[10] J. Zobel, A. Moffat, K. Ramamohanarao “Inverted Files Versus Signature Files for Text Indexing,” In ACM Trans. Database Syst. 23(4): 453-490 (1998)

[11] Yufei Tao and Cheng Sheng ,” Fast Nearest Neighbor Search with Keywords”, IEEE, April 2014.