ISSN: 2321-7782 (Online)
Volume 1, Issue 5, October 2013
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

Efficient Database Indexing Using P+ Tree and R* Tree indexing

Nikhil Dasharath Karande
Scholar of Singhania University, Rajasthann
Assistant Professor, Department of Computer Science and Engineering
Bharati Vidyapeeth's College of Engineering
Kolhapur – India

Abstract: Data warehouse system is becoming more and more important for decision-makers. Most of the queries against a large data warehouse are complex and iterative. The ability to answer these queries efficiently is a critical issue in the data warehouse environment. If the right index structures are built on columns, the performance of queries, especially ad hoc queries will be greatly enhanced. The objective of this paper is to propose an indexing techniques being used in both academic research and industrial applications. In addition, we identify the factors that need to be considered when one wants to build a proper index on base data. Indexing is the key to achieve this objective without adding additional hardware. This paper focuses the R*tree and P+ tree indexing techniques for efficient database accessing.

Keywords: Indexing; R* tree indexing; P+ tree indexing.

I. INTRODUCTION

A data warehouse (DW) is a large repository of information accessed through an Online Analytical Processing (OLAP) application. This application provides users with tools to iteratively query the DW in order to make better and faster decisions. The information stored in a DW is clean, static, integrated, and time varying, and is obtained through many different sources. Such sources might include Online Transaction Processing (OLTP) or previous legacy operational systems over a long period of time. Requests for information from a DW are usually complex and iterative queries. Complex queries could take several hours or days to process because the queries have to process through a large amount of data. A majority of requests for information from a data warehouse involve dynamic ad hoc queries. The P+ tree appears to be the most robust method to use in a Data Mining context where one deals with large dataset sizes and high dimensionality and more that one nearest neighbours are required in the answer. P+-tree supports both window queries and KNN queries under different workloads efficiently. This index structure first divides the data space into clusters that are essentially hyper-rectangles. Then it transforms each subspace into a hypercube and applies the pyramid technique in each subspace. The number of subspaces is always an integral power of 2. The order of Division is defined as the times we divide the space and it is an important parameter of the P+ tree. To facilitate the building process of the P+ tree and query processing, there is an auxiliary structure called the space-tree, which is built during the space division process. The space-tree is similar to the k-d tree, but it stores the transformation information instead of data points in the leaf nodes. For the construction phase, a P+-tree is basically a B+-tree where the data records with their keys are stored in the leaf nodes.

R*-trees are a variant of R-trees used for indexing spatial information. R*-trees support point and spatial data at the same time with a slightly higher cost than other R-trees It was proposed by Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger in 1990. R*-Tree built by repeated insertion. There is little overlap in this tree, resulting in good query performance. Red and blue MBRs are index pages, green MBRs are leaf nodes. Requests for information from a DW are usually complex and iterative queries. Such complex queries could take several hours or days to process because the queries have to

process through a large amount of data. A majority of requests for information from a data warehouse involve dynamic ad hoc queries. Users can ask any question at any time for any reason against the base table in a data warehouse. The ability to answer these queries quickly is a critical issue in the data warehouse environment. Among the various solutions such as summary tables, indexes, parallel machines, etc. to speed up query processing, Indexing is the best key to overcome this problem.

In this research work, among the various data mining techniques we use the efficient mining techniques. So, we choose the suitable technique for proposed indexing techniques P+ tree and R*tree. We applied P+ tree and R*tree indexing techniques separately on data warehouse. The core work of this project is implementing the P+ and R* indexing techniques for the efficient retrieval of the data is carried out. We perform Comparative study after applying different queries separately on P+ tree and R*tree techniques.

II. INDEXING TECHNIQUES

A. P+ Tree

The basic idea of the P+-tree is to divide the space into subspaces and then apply the Pyramid technique in each subspace. To realize this, we first divide the space into clusters which are essentially hyper rectangles. We then transform each subspace into a hypercube so that we can apply the Pyramid technique on it. At the same time, the transformation makes the top of the pyramids located at the cluster center. Assuming that real queries follow the same distribution as data, most of the queries would be located around the top of the pyramids, that is, the "good position". Even if some queries may be located at the corner or edge of the cluster and therefore causes a large region to be accessed, the data points accessed are not prohibitively large because most of the data points are gathered at the cluster center. In addition, the region accessed by a query is significantly reduced by space division. Thus, the P+-tree can alleviate the inefficiencies of the Pyramid technique. We note that although we cluster the space into subspaces, our scheme also works for uniform data since uniform data is a special case of clustered data. While uniform data does not benefit from the transformation, dividing the space into subspaces is still an effective mechanism for performance improvement. To facilitate building the P+-tree and query processing, we need an auxiliary structure called the *space-tree*, which is built during the space division process. The leaf nodes of the space-tree store information about the transformation. We will first introduce the data transformation, so that readers know what information is stored. Then, we present the space division process. At last, we show how the P+-tree is constructed.

B. R Tree*

R-trees support point and range queries, and some forms of spatial joins. Another interesting query, supported to some extent by R-trees, is the k nearest neighbours (k-NN query. R-tree can be thought of as an extension of B-trees in a multi-dimensional space. It corresponds to a hierarchy of nested n-dimensional MBBs. R-tree performance is usually measured with respect to the retrieval cost (in terms of DAC) of queries. Variants of R-trees differ in the way they perform the split algorithm. The well-known R-tree variants include R_-trees and R+-trees. We can find a more detailed description as well as depiction of other R-tree variants. It is not usually efficient to insert a large amount of data into an R-tree using the standard insert operation. The split algorithm is rather an expensive operation; therefore, the insertion of many items may take quite a long time. Moreover, this algorithm is executed many times during the insertion. The query performance is greatly influenced by utilization of the R-tree. A common utilization rate of an R-tree created with a standard insert algorithm is around 55%. On the other hand, the utilization rate of the R-tree created with the bulk-loading method rises up to 95%.

Several bulk-loading methods have been developed. All bulk-loading methods first order input items. Method utilizes one-dimensional space filling curve criterion for such ordering. This method is very simple and allows ordering input items very fast. The result R-tree preserves suitable features for the most common data.

The R*-tree is a data partitioning structure that indexes MBRs (minimum bounding rectangles). The minimization of both coverage and overlap of the MBRs influences the performance of R* tree. When overlap occurs on data query or insertion, more

than one branch of the tree needs to be expanded and traversed (due to storage redundancy). When the coverage is minimized this has the effect of improving pruning performance, so whole pages are excluded from search more often.

The R*-tree attempts to reduce both, with a combination of a revised node split algorithm and the concept of forced reinsertion when nodes overflow. This is based on the observation that R-tree structures are highly sensitive to the order in which their entries are inserted, so an insertion-built (rather than bulk-loaded) structure is likely to be sub-optimal. So the deletion and reinsertion of some entries allows them to "find" a place in the tree that may be more appropriate than their original location. When a node overflows, a portion of its entries are removed from the node and reinserted into the tree. This produces better-clustered groups of entries in nodes, with the effect that node coverage is reduced. Furthermore, actual node splits are often postponed, causing average node occupancy to become higher. Re-insertion can be seen as a method of incremental tree optimization triggered on node overflow.

R*-Tree built by repeated insertion (in ELKI). There is little overlap in this tree, resulting in good query performance. Red and blue MBRs are index pages, green MBRs are leaf nodes. Minimization of both coverage and overlap is crucial to the performance of R-trees. Overlap means that, on data query or insertion, more than one branch of the tree needs to be expanded (due to the way data is being split in regions which may overlap). A minimized coverage improves pruning performance, allowing excluding whole pages from search more often, in particular for negative range queries.

C. Performance

1) Improved split heuristic produces pages that are more rectangular and thus better for many applications.

2) Reinsertion method optimizes the existing tree, but increases complexity.

3) Efficiently supports point and spatial data at the same time.

D. Algorithm and Complexity

1) The R*-tree uses the same algorithm as the regular R-tree for query and delete operations.

2) When inserting, the R*-tree uses a combined strategy. For leaf nodes, overlap is minimized, while for inner nodes, enlargement and area are minimized.

3) When splitting, the R*-tree uses a topological split that chooses a split axis based on perimeter, then minimizes overlap.

4) In addition to an improved split strategy, the R*-tree also tries to avoid splits by reinserting objects and sub trees into the tree, inspired by the concept of balancing a B-tree.

III. SYSTEM ARCHITECTURE

The Fig. 1 shows the system architecture, which depicts the layout of the process included to efficiently searching the record of work done. There are two subparts of system process are as follows:

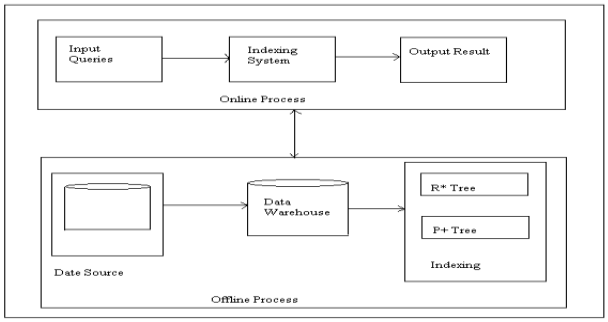

Fig. 1 System Architecture

A. Offline Process

Data warehouse is created on the available data source. Data in data warehouse gives as input to the implemented efficient indexing modules separately.

B. Online Process

Required queries applied to generated indexing system during offline process and output of these queries is our final output which is used for comparative study.

IV. EXPERIMENTAL RESULTS

The system implemented was tested for the P+ tree or R* tree indexing technique and the results obtained were satisfactory under the known constraints. The user needs to select any attribute for which want to assigned indexing, then write a query to select particular record. If the record is found then message will display as a record found, with the time require to search the record. The following snapshot shows the details of obtained experimental results.

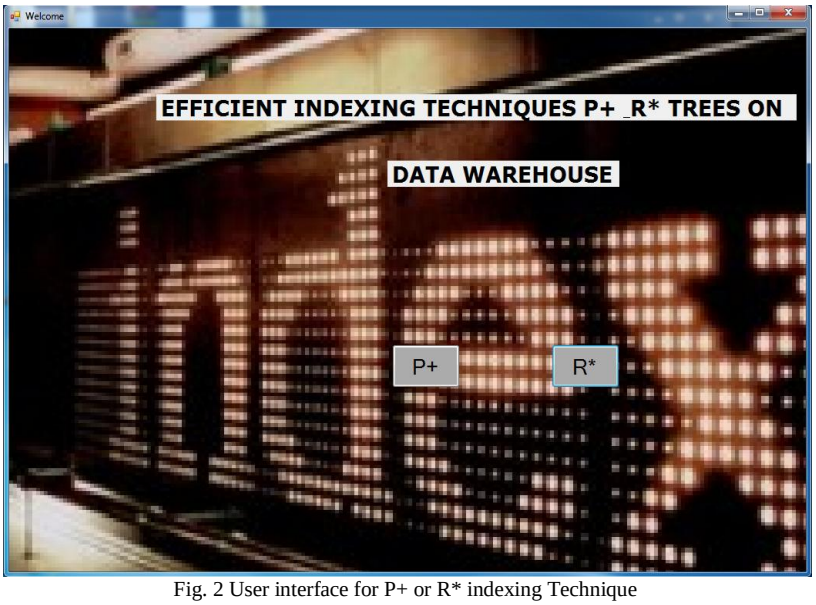

Fig. 2 User interface for P+ or R* indexing Technique

The Fig. 2 shows the user interface where user here can interact to choose the P+ tree or R* tree indexing technique.

The Fig. 3 shows the details in which user can interact to applying indexing by choosing the appropriate table attribute and write a query to search the record.

For example: indexing attribute “city” and search a record city “paris”.

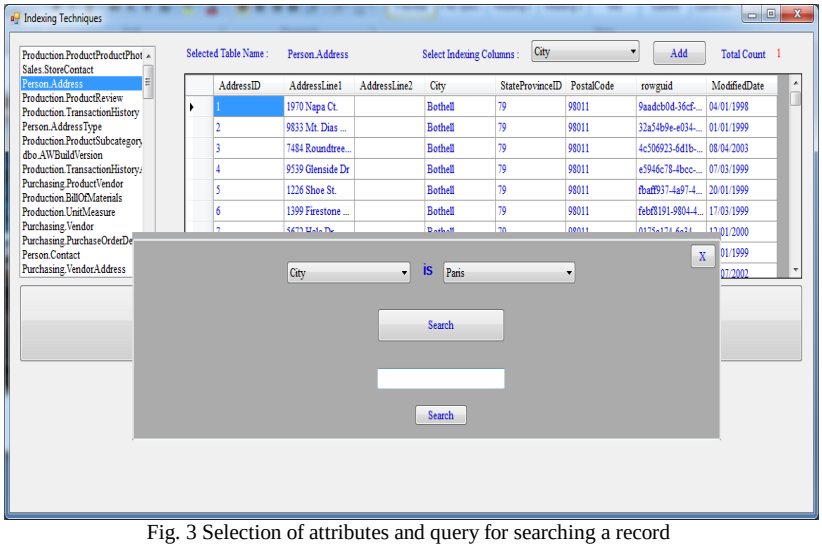

Fig. 3 Selection of attributes and query for searching a record

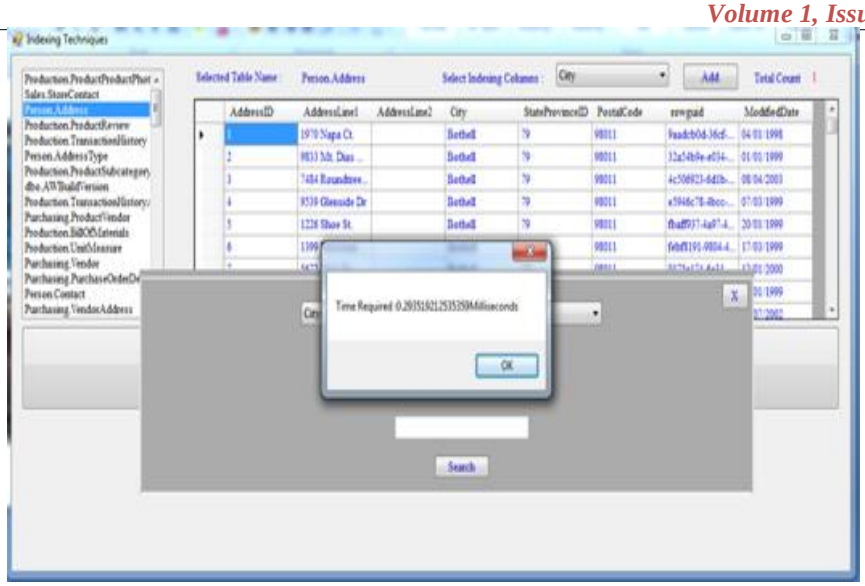

Fig. 4 Reslut with the time require to seacrh a record

The Fig. 4 shows the datils of the search record as message " paris was found", with the system time required to search the record.

V. CONCLUSION

The ability to extract data to answer complex, iterative, and ad hoc queries quickly is a critical issue for data warehouse applications. A proper indexing technique is crucial to avoid I/O intensive table scans against large data warehouse tables. The challenge is to find an appropriate index type that would improve the queries' performance. The experimental results, where the major strength of the P+-tree is that it works well for various workloads. Extensive experiments demonstrate that the P+-tree has considerable speedup over existing indexing methods for both window queries and KNN queries. R*-trees support point and spatial data at the same time with a slightly higher cost than other R-trees. The next step of research is to optimize and obtain the different indexing methods for more efficiently searching of queries.

References

1. Indexing Techniques for Data Warehouses' Queries by Sirirut Vanichayobon Le Gruenwald.
2. Efficient Indexing Methods In The Data Mining Context by Nikolaos Kouiroukidis, Department of Applied Informatics, University of Macedonia.
3. V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently", In Proc. of the ACM SIGMOD Conf. on Management of Data, Jun. 1996.
4. P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes",SIGMOD,1997
5. Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. In Proc. SIGMOD.
6. Transaction Processing Performance Council (TPC), "TPC Benchmark D, Decision Support", Standard Specification Revision 2.0.1, December 5, 1998.

AUTHOR(S) PROFILE

Mr. N. D. Karande, (Scholar of Singhania University, Rajasthan), received the B.E. degree in Computer Science and Engineering from Bharati Vidyapeeth College of Engineering, Kolhapur, India in 2006. He received the M.Tech degree in Computer Science and Technology from Shivaji University, Kolhapur, India in 2010. From 2008 to till date, he is working as Assistant Professor at Bharati Vidyapeeth College of Engineering, Kolhapur, India. He has published various papers in the area of Database Indexing, Security and Natural Language Processing.

