Multi-Dimensional Indexes in DBMSs

Thomas Mercieca, Faculty of ICT, University of Malta, Msida, Malta
Joseph G. Vella, Faculty of ICT, University of Malta, Msida, Malta

ABSTRACT

Multi-dimensional data is present across multimedia, data mining and other data-driven applications. The R-Tree is a popular index structure that DBMSs are implementing as core for efficient retrieval of such data. The gap between the best and worst-case performance is very wide in an R-tree. Thus, building quality R-trees quickly is desirable. Variations differ in how node overflow are approached during the building process. This article studies the R-Tree technique that the open-source PostgreSQL DBMS uses. Focus is on a specific parameter controlling node overflows as an optimisation target, and improved configurations are proposed. This parameter is hard-wired into the DBMS, and therefore, an implementation is presented to allow this parameter to become accessible through an SQL construct. The access method designer can resort to configuring this parameter when trying to meet specific storage or time-related performance targets. With this study, the reader can gain an insight into the effects of changing the parameter by considering the spatial indexes on well-known workloads.

KEYWORDS
Access Methods, Indexing, Node Overflows, Node Splitting, Parameter Optimisation, PostgreSQL, R-Tree, Spatial, Storage Utilisation

INTRODUCTION

Emerging applications such as those within multimedia, geographical and medical domains operate in high-dimensional space. These applications make use of abstract data types that go beyond the standard, one-dimensional types that traditional DBMSs offer.

While traditional data types can utilize the prevalent B-Tree (Comer, 1979) for efficient data retrieval because of the natural ordering within these types, such a relationship is not evident for data types whose dimension is more than one (Gaede & Günther, 1998). Emerging applications with the latter type of data underperform with the traditional relational model (Manolopoulos, Nanopoulos, Papadopoulos & Theodoridis, 2010). Eventually, several multi-dimensional access methods were proposed such as Quad Trees (Finkel & Bentley, 1974) and K-D-B Trees (Robinson, 1981).

The R-Tree (Guttman, 1984) is one such multi-dimensional indexing technique that is being widely adopted for its suitability to higher dimensions and to secondary memory implementation. The R-Tree has found itself in several DBMSs in response to their requirements of having an efficient access method for multi-dimensional data types with many efforts having been directed towards the building of this structure for a well-performing index to meet the demands of queries.

The authors observe PostgreSQL’s current R-Tree implementation assumes a specific parameter affecting the storage utilisation of the index structure. The authors also observe that PostgreSQL and PostGIS (Obé & Hsu, 2015), a PostgreSQL Geographic Information System (GIS) extension, differ on the value of this configuration. This study demonstrates the benefits that arise out of fine-tuning this parameter, while introducing an improved method of interacting with this configuration. As a result, a data modeller is able to accommodate a wider variety of multi-dimensional datasets and performance targets at index creation time.

DOI: 10.4018/JCIT.2019070103

Copyright © 2019, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
LITERATURE REVIEW

The R-Tree

Guttman’s R-Tree (1984) is a height-balanced tree structure characterized by grouping space into subspaces. Spaces are represented with a Minimum Bounding Rectangle (MBR) structure across dimensions. Each MBR is represented by a node or a page on a disk-based system. An MBR contains several entries of other MBRs, except at the leaf-level, in which the MBR tightly covers the object it is representing and points towards the data record of this object. The maximum value of an R-tree’s height can be given by \( h_{\text{max}} = \lceil \log_m N \rceil - 1 \) where \( m \) is the minimum number of entries allowed per node, and \( N \) is the number of data elements being indexed. The maximum number of nodes possible in a specific tree (Manolopoulos et al., 2010) is derived by summing the maximum possible number of nodes at each level of the tree:

\[
\sum_{i=1}^{h_{\text{max}}} \left[ N / m^i \right] = \left[ N / m \right] + \left[ N / m^2 \right] + \cdots + 1
\]

Searching an R-Tree consists of a traversal from the topmost level of the tree, checking which MBRs overlap with the input search item. Once the leaf level is reached, all elements on the node are examined to identify which element matches the input item.

Because MBRs can overlap, it is inevitable that an element is not found on the first path explored and multiple paths would have to be subsequently investigated. If the entries in an instance of a node are represented by \( E_1, \ldots, E_p \), then the total overlap of an entry in a node, as interpreted by Beckmann, Kriegel, Schneider and Seeger (1990), can be expressed using Equation (2). This overlap is attributed to the large coverage and underused space of each MBR illustrated in Figure 1.

Minimum Bounding Rectangles (MBRs)

\( A, B \) and \( C \) minimally cover the indicated elements. In \( i \) (as well as from \( d \) to \( k \)), each element takes the form of an MBR representation of an object with a pointer to its data tuple. Retrieving \( k \) requires a traversal down \( A \) and \( C \) before \( k \) is finally found – an example of a multipath problem. Figure has been adapted from Introduction to PostGIS (Boundless, 2017) which is licensed under the Creative Non Commercial-Commons Attribution-ShareAlike 3.0 United States License. A copy of this license can be viewed at https://creativecommons.org/licenses/by-nc-sa/3.0/legalcode.

\[
\text{overlap}(E_k) = \sum_{i=1\cap k}^{p} \text{area}(E_k.MBR \cap E_i.MBR), \ 1 \leq k \leq p
\]

The insertion operation builds the R-Tree structure. An insertion identifies a leaf node in which to allocate the new element through finding the least MBR enlargement required. In the case that the enlargement is the same as insertions in other locations, this conflict is resolved by choosing the MBR of the smallest area. Such insertions may overflow a node, in which case a node split is performed. When splitting a node having \( M \) entries, the \( M+1 \) entries are distributed over two nodes.

The difficulty behind node splitting lies in obtaining a good quality distribution. A decision between possible distributions has to be taken in reasonable time and in some way that future searches on that structure attenuate the problem of traversing multiple paths for a search. Guttman (1984), in the original R-Tree proposal, suggests minimizing the area of the MBR of each resulting node and discusses three splitting heuristics based on their computational time complexity: