Efficient Content-based Multimedia Retrieval Using Novel Indexing Structure in PostgreSQL

Fausto C. Fleites and Shu-Ching Chen
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{*fflei001,chens*}*@cs.fiu.edu*

***Abstract*—This demo paper presents a system based on PostgreSQL and the AH-Tree that supports Content-Based Image Retrieval (CBIR) through similarity queries. The AH-Tree is a balanced, tree-based index structure that utilizes high-level semantic information to address the well-known problems of semantic gap and user perception subjectivity. The proposed system implements the AH-Tree inside PostgreSQL's kernel by internally modifying PostgreSQL's GiST access mechanism and thus provides a DBMS with a viable and efficient content-based multimedia retrieval functionality.**

***Keywords*-databases, multimedia indexing, content-based retrieval;**

I. Introduction

The importance of efficiently indexing multimedia data has been exalted in recent years by the explosive growth of social networks and mobile devices. The problem of CBIR has been actively studied by the research community, but no solution has been able to provide both an efficient access method for large multimedia datasets and an effective multimedia content-based retrieval mechanism. For text-based information, traditional database management systems (DBMSs) have provided tools for storage and retrieval via index methods, e.g., B^+-tree, but have been unable to do the same for multimedia retrieval due to the well-known problems of (a) the semantic gap between low-level features and high-level concepts and (b) the subjectivity in the users' perception.

This demo paper presents a system based on PostgreSQL [1] that utilizes the AH-Tree [2] as index method to efficiently support CBIR. The AH-Tree allows multimedia retrieval through similarity queries (i.e., range and nearest neighbor queries) utilizing high-level semantic information during the retrieval process to address the semantic gap and user perception subjectivity problems. The high-level semantic information used by the AH-Tree during the retrieval queries is obtained from the Markov Model Mediator (MMM) mechanism [3] as affinity relationships, which provide a way to model the users perspective toward the semantic relationships between the multimedia data. To efficiently implement the AH-Tree and eliminate the I/O overhead it incurs when populating the affinity information through the tree structure for each query, the presented system only utilizes the affinity information at the leafs of the tree, making possible its implementation in PostgreSQL while still providing the same level of functionality. Consequently, the presented system combines the benefits provided by traditional DBMSs with those of a meaningful and efficient content-based retrieval mechanism.

II. System Architecture

The proposed system consists of a PostgreSQL DBMS whose kernel-level indexing mechanism has been extended to support the AH-Tree. The incorporation of the AH-Tree was achieved via modifying PostgreSQL's internal GiST mechanism [4], which serves as a template framework for implementing balanced, tree-based index structures such as the B-tree. Figure 1 depicts the implementation of the AH-Tree in PostgreSQL. Given a user query, the query processor in the DBMS kernel parses the query, finds the optimal execution plan, and executes the plan by interacting with the access method, i.e., the AH-Tree.

The GiST-based implementation of an index method requires several user-implemented index support functions, which are utilized by GiST in its insert, search, and delete core functions. Figure 1 shows the support functions in a blue font. For example, for an insert query, the insert algorithm descends the tree by selecting at each level the node with minimum insertion penalty ($Penalty$ function), and when it reaches a leaf node, the insertion of the new key is attempted. If there is space in the selected leaf, the key is inserted; otherwise, the node is split ($PickSplit$ function), and changes are propagated upward in the tree using the $Union$ support function. For a search query, the $Consistent$ function is utilized to determine the tree nodes that have to be visited. GiST's delete algorithm searches the key to be deleted, removes the key from its leaf node, and if the deletion causes an underflow, changes are propagated upward in the tree. Furthermore, during the insert, search, and delete processes, the access method needs to interact with the storage manager to access persistent data and guarantee concurrent executions via locking mechanisms.

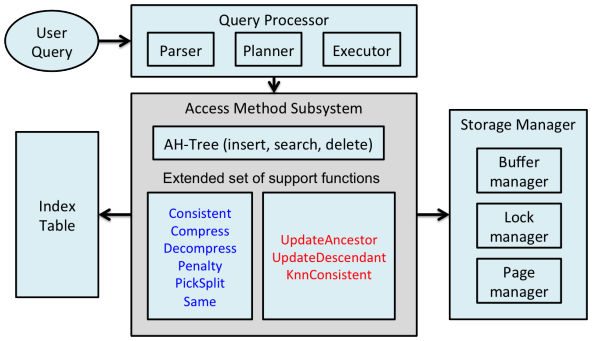

Figure 1. AH-Tree's implementation in PostgreSQL

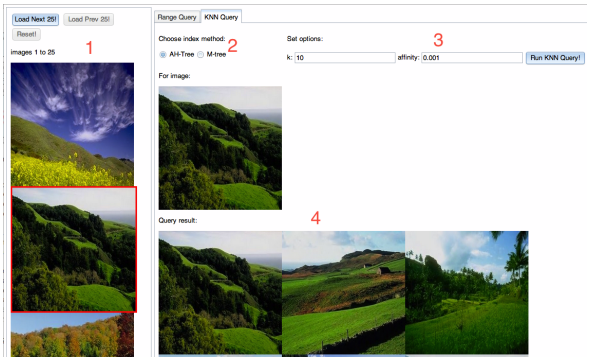

Figure 2. Web interface

Figure 3. Zoomed view of the web interface

One key feature of the AH-Tree is that it uses the triangular inequality property to avoid making unnecessary distance computations. This feature is partially implemented by having tree entries store the distances to their parents, which are updated during node splits. However, the $Union$ function, which is used to create parent keys when a node is split, does not support such updates as it does not distinguish between parent and child keys. For this reason, the proposed system modifies PostgreSQL's core GiST insert, search, and delete functions to use an extended set of support functions that replaces the $Union$ function by the functions $UpdateAncestor$ and $UpdateDescendant$ and adds the $KnnConsistent$ function for nearest neighbor queries. The updated set of support functions is shown in a read font in Figure 1. The functions $UpdateAncestor$ and $UpdateDescendant$ allow the update of the parent entry based on its child entries and a child entry based on its parent entry, respectively. These two functions are invoked during a node split.

III. Demonstration

The presented system will be demonstrated via a web application that allows users to browse an image database and submit similarity queries. The main goal of the demonstration is to allow users compare the results of the AH-Tree and the M-tree (a popular index method) [5] and visually evaluate the effect of high-level semantic relationships on the relevance of the results. For the purpose of the demonstration, an image dataset consisting of 10,000 images is utilized, where images are represented using 12 color features extracted from the HSV color space. The affinity information was derived from the MMM model by users with no knowledge of the presented system.

Labeled for description purposes, figures 2 and 3 depict the interface presented by the demo application. Shown in figure 2, the interface has four important visual regions. On the left region (labeled as 1), users can browse the image dataset and select a desired query image; on the top-center region (labeled as 2), users can select either the M-tree or AH-Tree as the index mechanism; on the top-right region (labeled as 3), users can input the query parameters and submit range or nearest neighbor queries; and on the center region (labeled as 4), users can see the query results. The first image shown in region 4 is the query image. Figure 3 provides a zoomed view of the regions labeled 2 and 3. The presented web application permits users to easily interact with the proposed system and evaluate the results from both relevance and efficiency perspectives.

Acknowledgment

This research was supported in part by the U.S. Department of Homeland Security under grant Award Number 2010-ST-062-000039, the U.S. Department of Homeland Securitys VACCINE Center under Award Number 2009-ST-061-CI0001, and NSF HRD-0833093.

References

[1] The PostgreSQL Global Development Group, "Postgresql 8.4.3 documentation," http://www.postgresql.org/files/documentation/pdf/8.4/postgresql-8.4.3-A4.pdf, May 2010.

[2] K. Chatterjee and S.-C. Chen, "Affinity hybrid tree: An indexing technique for content-based image retrieval in multimedia databases," in *Proceedings of ISM 2006*, Dec. 2006, pp. 47–54.

[3] M.-L. Shyu, S.-C. Chen, M. Chen, C. Zhang, , and C.-M. Shu, "MMM: A stochastic mechanism for image database queries," in *Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering*, Dec. 2003, pp. 188–195.

[4] J. M. Hellerstein and J. F. Naughton, "Generalized search trees for database systems," in *Proceedings of VLDB 1995*, 1995, pp. 562–573.

[5] P. Ciaccia, M. Patella, and P.l Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proceedings of VLDB 1997*, 1997, pp. 47–54.