Journal of Visual Languages and Computing 25 (2014) 800–807

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Short Paper

Efficiency of hybrid index structures—Theoretical analysis and a practical application ☆

Richard Göbel, Carsten Kropf*, Sven Müller

Institute of Information Systems, Hof University, Alfons-Goppel-Platz 1, D-95028 Hof, Germany

ARTICLE INFO

Article history:
Received 22 August 2014
Received in revised form
19 September 2014
Accepted 20 September 2014
Available online 28 September 2014

Keywords:
Hybrid index structures
Theoretical analysis
Experimental validation

ABSTRACT

Hybrid index structures support access to heterogeneous data types in multiple columns. Several experiments confirm the improved efficiency of these hybrid access structures. Yet, very little is known about the worst case time and space complexity of them. This paper aims to close this gap by introducing a theoretical framework supporting the analysis of hybrid index structures. This framework then is used to derive the constraints for an access structure which is both time and space efficient. An access structure based on a B+-Tree augmented with bit lists representing sets of terms from texts is the outcome of the analysis which is then validated experimentally together with a hybrid R-Tree variant to show a logarithmic search time complexity.

1. Introduction

Modern database systems often manage data of multimedia types. Texts, images or video data are stored inside those database systems. Some approaches with specialised database systems which allow storing and retrieving those data fast exist. Relational database management systems are still the most used technique, especially as data stores in enterprises, although NoSQL databases are also present. Mixing up different storage systems does not help in retrieving the data fast, because of having to search multiple systems and generating a finally intersected result set at the end. This implies, on one hand, a large overhead of temporarily allocated memory and, on the other hand, a large overhead of time as the distinct search results must be combined to a final result set.

Most existing hybrid access structures focus on the efficient storage and retrieval of data composed by textual and geographical data. In this paper, we focus on a probably more common scenario of data consisting of texts and conventional relational (single-valued) data sets. For this purpose the access structure is based on a conventional B+-Tree augmented by bit lists for indicating the presence of terms below a node. Besides this structure, also an R-Tree based one is evaluated.

Although several of these hybrid approaches with the ability to index data of this mixed type are present, there is no evidence about the temporal and spatial worst case complexity. The major contributions of this work are a theoretical basis to analyse hybrid access structures, an in-depth analysis of index structures leading to the theoretical construction of a hybrid index structure and an analysis of asymptotic time and space complexity (see Section 3).

Finally, a practical construction and evaluation of the previously analysed hybrid index structure with focus towards the theoretical analysis (see Section 4) is carried out. Based on a lack of space, related work is only discussed shortly.

2. Related work

New hybrid indexing strategies, enhancements, variations and compositions of existing concepts, like the B-Tree [1] or the R-Tree [2] have been proposed to address

☆ This paper has been recommended for acceptance by Shi Kho Chang.

* Corresponding author.
E-mail addresses: richard.goebel@iisys.de (R. Göbel), carsten.kropf@iisys.de (C. Kropf), sven.mueller@iisys.de (S. Müller).

http://dx.doi.org/10.1016/j.jvlc.2014.09.004
1045-926X/

performance issues on heterogeneous data. Approaches are present treating terms differently according to the frequency like [3]. Also a couple of different hybrid indexing methods or methods for management of data in hybrid data spaces like [4] exists. Approaches like [5] (KR*-Tree), [6] ((M)IR^2-Tree) or [7] (bR*-Tree) investigate, among others, the use of hybrid index structures combining textual and spatial retrieval utilising the R-Tree [2] or its variants (e.g. R*-Tree [8]) which augment the R-Tree with certain secondary structures (bitlists or inverted lists) to enable set annotations at R*-Tree elements. Approaches like [9,10] or [11] represent hybrid index structures for textual and spatial types which differ the treatment of textual entries based on the relative or absolute term frequency.

3. Analysis of access structures

This section analyses the worst case time and space complexity of hybrid access structures. For this purpose we will introduce a formal notation as a general basis for analysing non-trivial access structures.

The general idea to formalise an index structure is similar to the work of Hellerstein et al. (e.g. [12]). The differences in our approach are motivated by the fact that this paper deduces upper bounds for search time complexity instead of lower bounds as in [12].

We will also show that a hybrid tree providing information about both single-valued and multi-valued columns in the upper nodes of the primary tree structure ensures a time complexity of $\mathcal{O}(\log(n)\cdot m)$ and a space complexity of $\mathcal{O}(n)$. The upper nodes of the tree only have to be sorted according to the single-valued column.

3.1. Basic definitions

For analysing the access structure we will consider a database table with "normalised" columns containing single values and "non-normalised" columns with multiple values. Although most of this analysis is more generic we will assume that a multi-valued column contains a set of terms. We will denote the set of entries for such a table by a capital E and individual entries by e:

$$E=\{e_1,\ldots,e_n\} \tag{1}$$

For reasons of simplicity we assume a single set of values V for all columns. A set of k projection functions p_i retrieves the values of the k individual columns:

$$p_i: E\to 2^V \quad \text{with } i=1,\ldots,k \tag{2}$$

Projection functions may also be applied to sets of entries:

$$p_i(\{e_1,\ldots,e_j\})=p_i(e_1)\cup\cdots\cup p_i(e_j) \tag{3}$$

Single-valued (normalised) columns contain not more than one value per entry:

$$\forall e\in E:\ |p_i(e)|\le 1 \tag{4}$$

With q_i we denote the intersections between the sets of values in the related column:

$$q_i(\{e_1,\ldots,e_j\})=p_i(e_1)\cap\cdots\cap p_i(e_j) \tag{5}$$

The key idea of many access structures $\mathcal{A}$ is the assignment of entries to groups which are not necessarily disjoint:

$$\mathcal{A}=\{N_1,\ldots,N_p\} \quad \text{with } N_1,\ldots,N_p\subseteq E \tag{6}$$

With this approach not all of these groups need to be searched. For this purpose, each group N usually corresponds to a value v which occurs in all entries of N at the related column i:

$$\forall N\in\mathcal{A}\,\exists v\in V: v\in q_i(N) \tag{7}$$

It is also important that each group N provides every entry e which contains the corresponding value $v\in q_i(N)$ in the related column:

$$\forall v\in V,\quad e\in E,\quad N\in\mathcal{A}: v\in p_i(e)\wedge v\in q_i(N)\Rightarrow e\in N \tag{8}$$

An index structure with this definition is usually called an inverted index. This definition, however, is sufficiently generic to represent the group of entries referenced from the (leaf) nodes of a tree structure (e.g. a B-Tree) for a normalised column as well ($\forall e\in E:|p_i(e)|\le 1$).

With condition (7) and condition (8) the groups of the access structure for a single-valued column are disjoint.

3.2. Complexity of queries addressing single columns

With the definitions from the previous section we are ready to introduce complexity measures for time and space required for processing queries. A simple search condition for a column i is a set of alternative values $C_i\subseteq V$. With this approach we cannot only model conditions specifying a single value for a column but also other types of conditions like search ranges. All entries $e\in E$ which contain at least one value of this set ($p_i(e)\cap C_i\neq\emptyset$) are part of the result set.

Complex search conditions consist of multiple search conditions which may not only refer to different columns but also to the same column. An example is a set of words which all need to be included in a text column. For lack of space we do not provide a formal definition of complex search conditions in this paper.

We need to visit a group of an access structure if at least one entry in the group satisfies the search condition. Accordingly we define the function *visit* returning exactly these groups:

$$visit(C_i,\mathcal{A})=\{N|N\in\mathcal{A}\wedge(p_i(N)\cap C_i)\neq\emptyset\} \tag{9}$$

The result set for a search condition is the union of all groups of the access structure which need to be visited:

Lemma 1. *Let $C_i=\{v_1,\ldots,v_p\}$ be a simple search condition and $\mathcal{A}$ be an access structure. Then the following function "result" provides all entries which satisfy C_i:*

$$result(C_i,\mathcal{A})=\bigcup_{N\in visit(C_i,\mathcal{A})} N$$

This lemma follows directly from conditions (7) and (8).