A COMPARATIVE STUDY OF CONCURRENCY CONTROL METHODS IN B—TREES

Alexandros Biliris
Computer Science Department
Boston University
Boston, MA 02215

ABSTRACT

In this paper, we described a number concurrency control methods for B-trees. Among them, a protocol which allows a number of independent search, insertion, and deletion processes, acting concurrently on a B-tree, to operate even when multiple insertions or deletions are pending. A number of properties have been also proposed to compare such protocols. The set of properties includes the number of lock types being used, degree of data sharing, number of processes that are permitted to access a node simultaneously, how fast a reader can reach the leaf node, number of nodes locked by the three processes during their operation, number of nodes being accessed sequentially, and the number an updater passes through the tree. Based on these properties, seven protocols are compared and discussed.

1. INTRODUCTION

In many computing systems, there is a need for sharing data and resources among processes. The problem of synchronizing such processes that compete with one another for shared objects is called, in the database literature, the concurrency control problem. Given that the system has not failed, the concurrency control mechanism must ensure that consistency of objects is preserved and that all processes will complete their operations in finite time, [KOHL81].

An important part of any database is its index mechanism which speeds up the retrieval process by directing the searcher to a small portion of the database containing the desired item. Hashing and its variants provide a mean for data accessing. On the other hand, multilevel (tree) structures in which each index at some level points to another index in the level below until the actual data have been reached, have became extremely popular. While no single scheme can be best for all applications a particular multilevel structure, called B-tree [BAYE72], has became the most widely used technique for storing large files of information, especially on external storage devices. Knuth [KNUT73] provides a survey of the basic techniques and Comer [COME79] discusses and analyzes the B-trees and its variants.

Although, simultaneous accesses to the same database component may be rare, there is a high probability that indexes will be repeatedly accessed, and therefore a great deal of the time spent during the database access is attributed to searching through indexes. Serializing access to the indexes of a database may create an undesirable bottleneck and degrade the entire system. Since B-trees are the most widely used access aids, for both primary and secondary indexing, maximizing concurrency on them is the most contributing factor in the overall degree of concurrency.

The rest of the paper is organized as follows. Section 2 briefly presents some representative algorithms for concurrency control on B-trees. Section 3 describes the mU protocol and in section 4 twelve properties for comparison of similar algorithms are proposed. Finally, section 5 summarizes this work.

1.1. DEFINITIONS

B-tree Definition: A B-tree of order m is a balanced tree which has the following properties.

- Every node, except for the root has between m and 2m children.
- The root has between two and 2m children.
A nonleaf node consists of $s$ pointers to its children ($P_0, P_1, ..., P_{s-1}$) and $s-1$ keys ($K_1, K_2, ..., K_{s-1}$) arranged in such a way that for every key $K$ in the subtree pointed to by $P_i$, the following relationships hold:

\[
\begin{align*}
&K < K_i, i=0 \\
&K_i \leq K < K_{i+1}, 0 < i < s-1 \\
&K_{i} \leq K, i=s-1
\end{align*}
\]

A leaf node with $s$ children contains $s$ keys.

We assume that each node is organized sequentially, and that the set of all keys appears in the leaves. In case that $P_i$ belongs to a leaf node it may point to records keeping actual data associated with the key $K_i$.

Data associated with each key are of no interest in the following discussion and are omitted. An example of a three level B-tree of order two is shown in Figure 1.

Fig 1. A B-tree structure.

The operations to be performed, concurrently, on a B-tree structure will be of three kinds: search for a key, insert a (key, pointer) pair, and delete a pair from the tree. The processes that perform these operations are called read process (RP), insertion process (IP) and deletion process (DP), respectively. IPs and DPs are collectively called updaters.

An updater goes through the following two phases. Searching: it reads the tree starting from the root to determine the leaf node for a particular key. Restructuring: Once the leaf node is found, it adds (or removes) the key and restructures the tree if necessary. It is exactly this restructuring phase that creates most of the problems in a concurrent environment.

Most of the solutions to the problem of supporting concurrent operations in B-trees make use of the following observations. There exists a node which is the root of a subtree above which no change in data and structure due to an update can propagate. This node is called a safe node [BAYE77b]. A node consisting of less than $2m$ children is called insertion-safe (i-safe), because a new key can be added without forcing a split. A node with more than $m$ children is called deletion-safe (d-safe), because a key can be deleted without going below the $m$-children minimum. The portion of the access path from the deepest safe node to a leaf is called the scope of the Updater. We also refer to the child of the Updater’s deepest safe node on its scope, as the highest unsafe node.

The protocols compared in this paper are defined using the Conditional Compatibility and Convertibility Graph (CCCG). [BILI85b].

CCCG Definition: The CCCG is a weighted directed graph in which vertices represent lock types and edges with their associated weight specify the relation among these locks on a node, as follows:

if $a$ and $b$ are vertices (locks) then

...