An Aysmptotically optimal multiversion B-tree

Bruno Becker, Stephan Gshwind, Thomas Ohler, Bernhard Seeger and Peter Widmayer

Abhishek Srivastava
Student ID: 861307778

March 9, 2017
CS 236, Winter 2017

The problem:

In this paper the author is trying to provide an efficient method of handling range-queries for a transaction time databases which is updating the last version but able to query over all the previous versions as well.

The contribution:

The authors present *Multiversion B-tree* method which is directed acyclic graph of B-tree nodes produced by doing incremental changes to the original B-tree. Nodes are partitioned for different versions so that different B-tree root nodes have different internal version. MVBT is used for storing the temporal databases which fits with partial persistence in an efficient manner and also able to apply access methods efficiently on the range queries.

The method:

The general idea behind the mechanism is to store all the snapshots of the tree which is basically storing all the versions of B-tree evolved over time. All the access methods such as Inserts, Updates and Deletes are applied on the latest version of the tree and after applying the changes its current version number is increased. Queries over MV B-tree first identifies the version of tree which it requires for its result.

General method of doing this is to transform single version structure to single version external access structure which provides high utilization of disk blocks and reduces the cost in time and space requirement by a constant factor. Such increase in performance is although asymptotically optimal.

Leaf nodes and Inner nodes vary in some ways. Leaf nodes store key, insert version, delete version and information about the node but inner nodes store router info instead of key info in their nodes. Delete version is represented by * if the node is not yet deleted in current state. When a node is deleted its end version is updated but it can suffer from "block overflow" or "Weak version underflow". Structural modification is needed in MVBT and it is done by copying the blocks and remove all those entries which are deleted leaving only present version entries. If the block predominantly contains present version entries split will occur after few more insertions.

The main MVBT algorithm is as follow. Leaf node is found with new key and block is inserted. Block overflow is handled during insertion depending on version split, strong version underflow and strong version overflow. For deletion find the node and update its value, check for weak version underflow if true merge with the sibling node.

Comments:

The paper presents way of handling range queries efficiently by maintaining partial persistent databases using multiversion B-tree. Authors evaluated MVBT to be asymptotically optimal.

However, some drawbacks can be found in the proposed model:

- Using very old and similar way to write once read many mechanism which can be updated.
- It can also be expanded to bi-temporal databases and Overlapping B+ tree.