Building a Bw Tree Takes More Than Just Buzz Words

wu

May 10, 2024

1 Abstract

Bw-tree is a lock-free index that provides high throughput for transactional database workloads in SQL Server's Hekaton engine. The Bw-Tree avoids locks by appending delta record to tree nodes and using an indirection layer that allows it to atomically update physical pointers using compare-and-swap (CaS).

This paper has two contributions:

1. First, it is the missing guide for how to build an **in-memory** lock-free Bw-Tree.

2. our evaluation shows that despite our improvements, the Bw-Tree still does not perform as well as other concurrent data structures that use locks.

Background HackerNews's discussion.

2 Introduction

The high-level idea of the Bw-Tree is that it avoids locks by using an indirection layer that maps logical identifiers to physical pointers for the tree's internal components. Threads then apply concurrent updates to a tree node by appending delta records to that node's modification log. Subsequent operations on that node must replay these deltas to obtain its current state.

The indirection layer and delta records provide two benefits.

1. it avoids coherence traffic of locks by decomposing every global state change into atomic steps.

2. it incurs fewer cache invalidations on a multi-core CPU because threads append delta records to make changes to the index instead of overwriting existing nodes.

3 Bw-tree Essentials

Assume that the Bw-Tree is deployed inside of a database management system with a thread pool, and has worker threads accessing the index to process queries. If non-cooperative garbage collection is used,pn the DBMS also launches one or more background threads periodically to perform garbage collection on the index.

The most prominent difference between the Bw-Tree and other B+Tree-based indexes is that the Bw-Tree avoids directly editing tree nodes because it causes cache line invalidation. Instead, it stores modifications to a node in a delta record (e.g., insert, update, delete), and maintains a chain of such records for every node in the tree. This per-node structure, called a **Delta Chain**, allows the Bw-Tree to perform atomic updates via CaS. The **Mapping Table** serves as an indirection layer that maps logical node IDs to phisical pointers, making atomic updates of several references to a tree node possible.

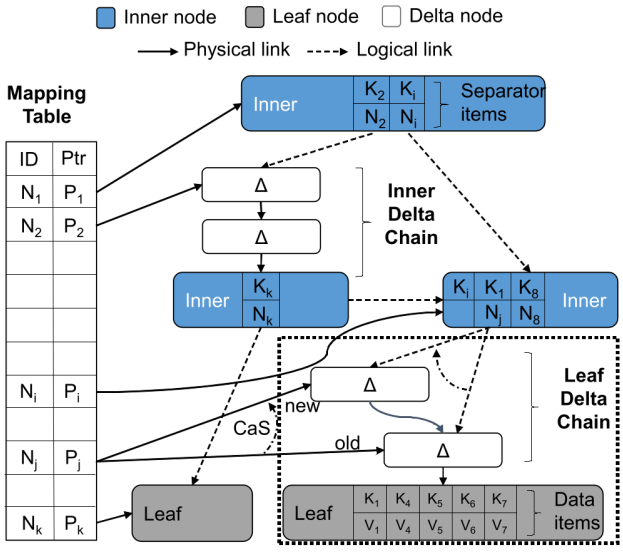

Figure 1: Architecture Overview – An instance of a Bw-Tree with its internal logical links, Mapping Table links, and an ongoing CaS operation on the leaf Delta Chain.

As in Fig. 3, every node in the Bw-tree has a unique logical node ID (64-bit). Instead of using pointers, nodes refer to other nodes using these IDs (**logical links**). When a thread needs the physical location of a node, it consults the Mapping Table to translate a node ID to its memory address.

3.1 Base Nodes and Delta Chains

A **logical node** in the Bw-Tree has two components:

1. a base node:
   (a) **inner base node** holds a sorted (key, node ID) array
   (b) **leaf base node** holds a sorted (key, value) array
2. a Delta Chain.

Initially, the Bw-Tree consists of two nodes, an empty leaf base node, and an inner base node that contains one separator item referring to the empty leaf node. Base nodes are immutable.

As shown in Fig. 3.1, a Delta Chain is a singly linked list that contains a chronologically-ordered history of the modifications made to the base node. The entries in the Delta Chain are connected using physical pointers, with the tail pointing to the base node. The entries in the Delta Chain are connected using physical pointers, with the tail pointing to the base node. Both the base node and its delta records contain additional meta-data that represent the state of the logical node at that point in time. That is, when a worker thread updates a logical node, they compute the latest attributes of the logical node and store them in the delta record.

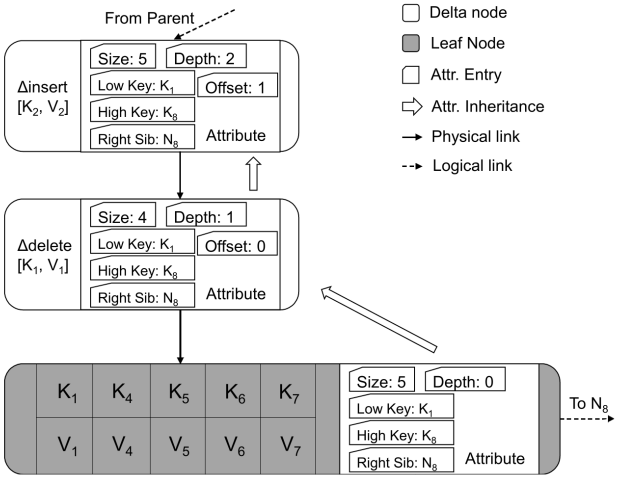

Figure 2: Delta Records Overview – A more detailed illustration of a logical leaf node from Fig. 1 with its base node and two delta nodes.

Table 1: Node Attributes - The list of the attributes that are stored in the logical node's elementes

Attribute	Description
low-key	The smallest key stored at the logical node. In a node split, the low-key of the right sibling is set to the split key. Otherwise it is inherited from the element's predecessor.
high-key	The smallest key of a logical node's right sibling. split records use the split key for this attribute. merge records use the high-key of the right branch. Otherwise, it is inherited from the element's predecessor.
right-sibling	The ID of the logical node's right sibling
size	The number of items in the logical node. It is incremented for insert records and decremented for delete records.
depth	The number of records in the logical node's Delta Chain
offset	The location of the inserted or deleted item in the base node if they were applied to the base node. Only valid for insert and delete records

3.2 Mapping Table

Bw-Tree allows a thread to update all references to a node in a single CaS instruction that is available on all modern CPUs. If a thread's CaS fails then it aborts its operation and restarts. This restart is transparent to the higher-level DBMS components. Threads always restart an operation by traversing again from the tree's root. The nodes that a thread will revisit after a restart will likely be in the CPU cache anyway.

3.3 Consolidation and Garbage Collection

Worker threads will periodically consolidate a logical node's delta records into a new base node. Consolidation is triggered when Delta Chain's length exceeds some threshold. Microsoft reported that a length of eight was a good setting.

At the beginning of consolidation, the thread copies the logical node's base node contents to its private memory and then applies the Delta Chain. It then updates the node's logical link in the Mapping Table with the new base node. After consolidation, the index reclaims the old base node and Delta Chain memory after all other threads in the system are finished accessing them.

The original Bw-Tree uses a centralized epoch-based garbage collection scheme to determine when it is safe to reclaim memory

3.4 Structural Modification

As with a B+Tree, a Bw-Tree's logical node is subject to overflow or underflow. These cases require **splitting** a logical node with too many items into two separate nodes or **merging** together underfull nodes into a new node. Bw-Tree's **structural modification** (SMO) protocols for handling node splits and merges without using locks. The main idea is to use special delta records to represent internal structural modifications.

The SMO operation is divided into two phases:

1. **logical phase**: appends special deltas to notify other threads of an ongoing SMO

   some thread t appends a δ**insert**, δ**merge** or δ**remove**

2. **physical phase**: performs the SMO