Lecture #08: OLTP Indexes – Part 1

15-721 Advanced Database Systems (Spring 2018)
http://15721.courses.cs.cmu.edu/spring2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 T-Trees

Order-preserving index Based on AVL Trees that is designed specifically for in-memory databases. Instead of storing keys in nodes, store pointers to their original values. Threads perform breath-first search ordering of keys.

Proposed in 1986 from database researchers at University of Wisconsin–Madison [1]. Used in **TimesTen** and other early in-memory DBMSs developed 1990s. T-Trees are not used in any new in-memory DBMS because they are not cache friendly.

Advantages:

- Uses less memory because it does not store keys inside of each node.
- Inner nodes contain key/value pairs (like B-Tree).

Disadvantages:

- Difficult to re-balance.
- Difficult to implement safe concurrent access.
- Have to chase pointers when scanning range or performing binary search inside of a node.

2 Skip List

Multiple levels of linked lists with extra pointers that **skip** over intermediate nodes. Proposed by researchers at the University of Maryland–College Park in the 1990s [4]. The index maintains keys in sorted order without requiring global re-balancing.

A collection of lists at different levels:

- Lowest level is a sorted, singly linked-list of all keys.
- 2nd level links every other key.
- 3rd level links every fourth key.
- In general, a level has half the keys of one below it.

The key idea of a skip list is that it is a **probabilistic data structure**. To insert a new key, flip a coin to decide how many levels to add the new key into. Provides approximate O(log n) search times.

Advantages:

- Uses less memory than a B+Tree (only if you do not include reverse pointers).
- Insertions and deletions do not require re-balancing.

Disadvantages:

- Lots of random memory access (i.e., not cache friendly) because threads have to follow pointers when moving from one node to the next.
- Does not support reverse range scans (i.e., backwards) because the linked-lists only point in one direction. Require extra effort to do this.

Concurrent Skip List

It is possible to implement a concurrent skip list using only CaS instructions [3]. The data structure only support links in one direction because CaS can only swap one location in memory (i.e., one pointer) atomically. If the DBMS invokes operation on the index, it can never "fail". A transaction can only abort due to higher-level conflicts.

Insert:

- For insert, CaS should happen **bottom up**.
- If a CaS fails, then the index will retry until it succeeds.

Delete:

- First **logically** remove a key from the index by setting a flag to tell threads to ignore.
- Then **physically** remove the key once we know that no other thread is holding the reference.
- Deletion should start from the top down to bottom.

Optimizations

The skip list as described in most textbooks is inefficient [5].

Potential Optimizations:

- Reducing `RAND()` invocations.
- Reduce the number of random memory look-ups by packing multiple keys into a single node:
  - **Insert Key:** Find the node where it should go and look for a free slot. Perform CaS to store new key. If no slot is available, insert new node.
  - **Search Key:** Perform linear search on keys in each node.
- Reverse iteration with a stack.
  - Perform a regular range query, but add the keys into a stack.
  - In the end, pop they keys from the stack, and they will pop in reverse order.
- Reusing nodes with memory pools.

3 Bw-Tree

Latch-free index designed by Microsoft Research for the Hekaton project [2].

Since CaS only updates a single address at a time, this limits the design of a data structure. Threads never need to set latches or block.

Key Idea #1 – Deltas:

- No updates in place. This reduces cache invalidation.
- Each update to a page produces a new delta.
- Delta physically points to base page.
- Install delta dress in physical address slot of mapping table using CaS.

Key Idea #2 – Mapping Table:

- Maps (logical) page IDs to their physical address locations in memory.

- Nodes only store page IDs. Threads check the mapping table to find out where in memory they need to go to when traversing the tree.
- Allows for CaS of physical locations of pages.

Operations

Update:

- Each update to a new page produces a new delta.
- Delta physically points to base page.
- Install delta address in physical address slot of mapping table using CaS.
- Threads may try to install updates to same state of the page. Winner thread succeeds, any loser thread must retry their operation.

Search:

- Traverse tree like regular B+tree.
- If mapping table points to delta chain, stop at first occurrence of search key.
- Otherwise, perform binary search on base page.

Delta Records

- **Record Update Deltas:** Insert/update/deletes
- **Structure Modification Deltas:** Split/Merge information
- **Consolidation:**
  - Consolidate updates by creating new page with deltas applied.
  - CaS-ing the mapping table address ensures no deltas are missed.

Garbage Collection

The Bw-Tree uses an epoch-based garbage collection scheme. Also called RCU in Linux.

- All operations are tagged with an **epoch**.
- Each epoch tracks the threads that are part of it and the objects that can be reclaimed.
- Thread joins an epoch prior to each operation and post objects that can be reclaimed for the current epoch (not necessarily the one it joined).
- Garbage for an epoch is reclaimed only when all threads have exited the epoch.

References

[1] T. J. Lehman and M. J. Carey. A study of index structures for main memory database management systems. In *VLDB '86: Proceedings of the 12th international conference on Very large data bases*, pages 294–303, August 1986. URL http://www.vldb.org/conf/1986/P294.PDF.

[2] J. J. Levandoski, D. B. Lomet, and S. Sengupta. The bw-tree: A b-tree for new hardware platforms. In *ICDE '13 Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, pages 302–313, April 2013. doi: https://dl.acm.org/citation.cfm?id=2510649.2511251.

[3] W. Pugh. Concurrent maintenance of skip lists. Technical report, 1990. URL https://dl.acm.org/citation.cfm?id=93717.

[4] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6): 668–676, June 1990. URL https://dl.acm.org/citation.cfm?id=78977.

[5] Ticki. Skip lists: Done right. Sept 2016. URL http://15721.courses.cs.cmu.edu/spring2018/papers/08-oltpindexes1/skiplists-done-right2016.pdf.