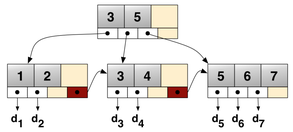


FM Tree: A Durable Flash Memory Search Tree

James N. Clay III and Kevin A. Wortman { jaclay and kwortman }@fullerton.edu

California State University Fullerton™

Problem

The lifespan of flash memories has been of serious concern since their inception; flash memory degrades proportionally to the number of times it is erased [1]. However newer multi-level flash memory has properties that may ease this problem.

This is an interesting problem, for:

1. Writes may only increase flash cell values.
2. Erasures happen in entire blocks, not individual cells.
3. Erasures are costly in terms of device wear.
4. Writes and erasures are slow, reads and increments are fast.

Contributions

We have created a durable variant of a commonly used database and operating system data structure.

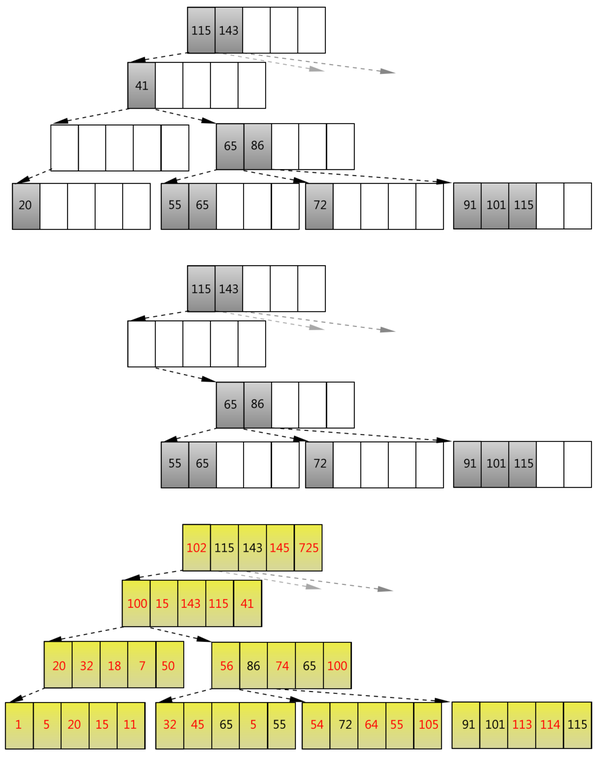
The method is based on [2]. Our main contributions are specific changes to B-tree operations:

1. Perform erasures lazily.
2. Relax key and value storage requirements.
3. Allow unused to nodes to remain in the tree.
4. Retain logarithmic space and time bounds.

We also give theoretical results indicating that our FM tree always has fewer erasures than a B-tree.

Method

Deletion of Key 20



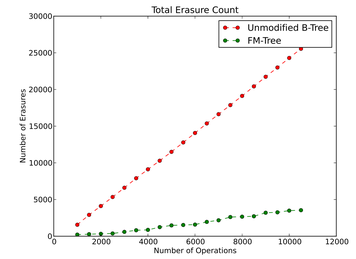
The standard B-Tree required keys to be sorted to enable fast tree traversal. As every B-Tree node must contain an ordered list of the keys for traversal, inserts into and deletes from the tree amplify the number of writes necessary to store the key values. Inserting a new key value at a specified place in the list requires shifting all of the other node values to the right; without this restriction we can perform economical insertions that minimally increase the net state of the block they are stored in.

To prevent erasures during element deletions, we add an active/inactive flag to each key-pointer record in our nodes. Individual key-pointer entries can be deleted by simply toggling that flag, rather than erasing an entire block. When inserting a key-pointer record, the tree can attempt to reuse an inactive key-pointer record.

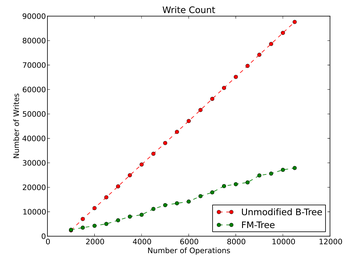
Forcing a specific number of cells to remain vacant within each node increases the number of options where a key can be subsequently inserted.

To attune the tree to periodic rebuilding we include the concept of *ghost nodes*. An internal node may become a ghost node if enough deletions occur that all the node contains is a forwarding pointer to its last child. This node is part of the space that the B^- tree requires [2].

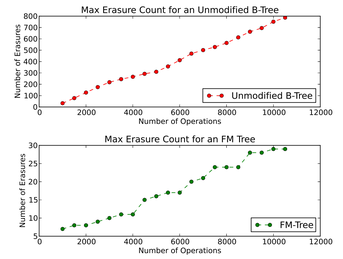
Results



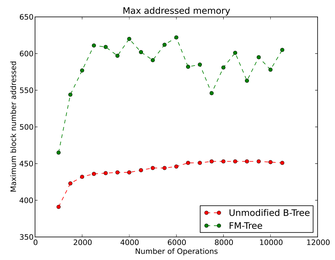
Total Erasures



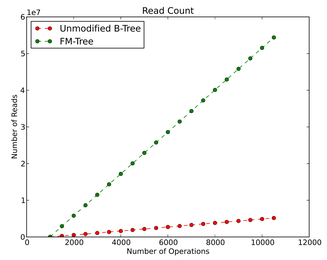
Write Count



Max Erasure Count



Max Addressed Memory



Read Count

We present a variety of experiments performed on a Python implementation of the FM Tree. We emulate the flash memory, FM Tree, and B-tree to run a variety of benchmarks. Every experimental trial consists of randomly generated data sets that are inserted into both trees. Each tree is inserted with a baseline of 1000 elements. Following these initial insertions, 10000 randomly chosen insertions and deletions are performed. We repeat this process a total of 4 times independently and determine the average for each data point. We then calculate the FM Tree performance by comparing the erasures, reads and writes between it and the B-tree. This process indicates that the FM Tree performs 27 times to 72.2 times fewer erasures. While the total read count was higher, the total writes and erasures performed were far lower. As these are the most expensive operations in terms of time, realistically the FM Tree would also be far faster than the B-tree.

Future Work

We believe these durability gains can be applied to other fundamental data structures. We are currently researching applying the same bounds to hash table data structures, however there are many other data structures which may benefit from these gains as well. Image and compressed audio data structures should also benefit from our approach. Future work may also look into the areas of finding better rebuild constants and tighter bounds on erasure count.

References

[1] Anxiao Jiang, Robert Mateescu, Moshe Schwartz, and Jehoshua Bruck. Rank modulation for flash memories. *IEEE Trans. Inf. Theor.*, 55:2659–2673, June 2009.

[2] Siddhartha Sen and Robert Tarjan. Deletion without rebalancing in multiway search trees. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 832–841. Springer Berlin / Heidelberg, 2009.

Further references available in paper or presentation.