High Concurreny B-Trees for Insert Heavy Workloads

An in-depth comparison

Isaac Bowen*
University of Michigan College of Engineering
MSE: Computer Science and Engineering
irbowen@umich.edu

G.K.M. Tobin†
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

ABSTRACT

Code can be found here Fill this in once we're done

You should try to write the best research paper that you can using the results of your project. You have read many good papers throughout this class, so by this point you should have a good idea of what makes a good research paper! Basically, your report needs to clearly present the following:

1. the problem statement
2. The motivation, why its important
3. The literature review (the previous work in this area
4. Main idea and approach
5. Implementatino techniques
6. Experimental setup
7. Results

Keywords

B-Trees, Reader Writer Locks, B-Link, Lock-Free

1. INTRODUCTION

2. PREVIOUS WORK

B-trees have long been the primary access data structure for databases, file systems, and various other systems because of their logarthimc `insert()` and `get()`. They also have a host of other properties that make them also for large systems. A short list:

1. Maintain everything inorder. This means that merge operations can be done without sort, and SMJ, and often used join in database systems, is very efficent
2. Block access. The strucutre can store as much data is it can on a single page, and doing locking on a single page. This made B-Tree's very popular early on, when database's could not fit in memory, and had to be dumped onto disk.

3. PLANS

1. Terminology: B+ Tree vs B*Trees
2.

4. ARCHITECTURE

What all did we have to build to make this work and test out our ideas?

- Testing Framework: We want to test the system at different levels of
  - Input Size
  - Number of THreads
  - Read Write Percent Workload
  - Fannout on inner nodes
  - Data slots on leaf nodes

  To testing all of these things, I will be creating a dope ass main.ccp that can read input from test ases
- Sequential Tree: We had to build a simple, single-thread B-tree first. We built on this for our other version, and will also use it as the baseline for testing.
- ReaderWriter Tree: This tree uses different kinds of locks, shared and exclusive, and also does checking to see if the child node can split, to see if this node is "safe" from splitting
- LockingFree Tree: I think I'm about to start this
- BLink Tree: This structure is different in that it insert into the leaf node, and acquires locks on the way back up

*A note about this author
†The secretary disavows any knowledge of this author's actions.

UofM:CoE:EECS 2015, Ann Arbor MI

ACM ISBN 123-4567-24-567/08/06.
DOI: 10.475/123_4

5. CITATIONS

Some papers that we read to give ourselves background on the topics:

- "Concurrent B-trees with Lock-free Techniques"[1] gives us the inspiration for the lock free B-tree. They, however, didn't not have access to the large memory, built in C++14 atomics, and they were not focusing on writes - they were trying to build a general purpose lock free btree for NUMA computers.
- "A survey of B-tree locking techniques"[2], much like the title says, presents an over of many of the different approaches that have been taken so far. However, it reallys skims over, almost dismissing, lock free techniques.
- "Efficient Locking for Concurrent Operations on B-Trees" [3] This one gave us the idea of the `can_split()` function, and gave background on reader writer locks and other, etc
- "Concurrent Cache-Oblivious B-Trees" [4] I need to read this one again
- "A Concurrent Blink-Tree Algorithm Using a Cooperative Locking Protocl" [5] Ryan is making this verion

6. EXPERIMENTAL RESULTS

We will need some tables. I have some data from the sequential tree

7. CONCLUSIONS

Conclusions

8. ACKNOWLEDGMENTS

Acknowledgments

9. REFERENCES

[1] Peter Graham Afroza Sultana, Helen Cameron. Concurrent b-trees with lock-free techniques. *DUNNO*, 0(0), 2010.

[2] Goetz Graefe. A survey of b-tree locking techniques. *ACM Transactions on Database Systems*, 35(2), 2010.

[3] Philip L. Lehman and S. Bing Yao. Efficient locking for concurrent operations on b-trees. *ACM Transactions on Database Systems*, 6(4):650–670, December 1987.

[4] Seth Gilbert Bradely C. Kuszmaul Michael A. Bender, Jermey T. Fineman. Concurrent cache-oblivious b-trees. 2005.

[5] Joonseon Ahn Sung-Chae Lim and Myoung Ho Kim. A concurrent blink-tree algorithm using a cooperative locking protocol. 2003.