**The Effects of Rotating Skiplist on LevelDB**

Our Github Repository containing source code, tests, LevelDB builds: <https://github.com/jsngalloway/leveldb_rotating_skiplist>

**Topic**

The aim of this project is focused around LevelDB (<https://github.com/google/leveldb>), an open-source, in-memory database designed and used by Google. Our project is an attempt to replace the underlying skiplist data structure it employs as an index with a rotating skiplist, as outlined in the paper “A Skip List For Multicore”, by Dick, Fekete, and Gramoli.

**Background**

LevelDB is a fast key-value storage that provides an ordered mapping of string keys to string pairs. Basic operations include Put(key, value), Get(key), Delete(key).

Rotating Skiplist is a type of skiplist that claims to be the fastest concurrent skiplist to date. It aims to improve throughput performance by rotating the head of the skiplist for minimizing contention, using wheels instead of towers in the internal structure, and no locks. According to the paper, Rotating skiplist outperforms all other novel logarithmic skiplists/trees (Fraser’s skiplist, Crain’s no hotspot skiplist, Optimistic skiplist, Transaction-based skiplist, Speculation-Friendly BST, Citrus Tree, Non-blocking BST) across 0%, 10%, 30%, and 90% update transactions in terms of throughput.

**Motivation**

The rotating skiplist we aim to implement boasts fast speeds.

**Major Challenges**

The first major challenge we encountered in this project was getting an instance of LevelDB to build. Initially, we worked with a windows port of LevelDB with limited documentation. To remedy this, we {insert information about docker image containing cmake and whatnot}. This allowed us to easily build, maintain, and test our LevelDB implementation.

Our first test intended to measure the performance of LevelDB’s existing skiplist against Synchrobench’s implementation of the Rotating skiplist. This proved to be difficult, however, as synchrobench’s results were not originally comparable to our custom LevelDB tests.

After producing a replicable, maintainable version of LevelDB, we wrote tests measuring LevelDB’s performance in accordance with the tests described in “A Skiplist for Multicore”. We tested monotonically increasing keys/values, across single and multithreaded instances, using different proportions of updates. We were interested in comparing the speed of these operation

LevelDB build on windows

Understanding leveldb’s structure

* Arena
* Skiplist implementation
* Locking mechanisms

Understanding synchrobench

* Performing tests on data structures
* Isolating rotating skiplist
* Understanding results

Porting Rotating skiplist to leveldb

* File structure
* Locking
* Includes
* Deletes

Testing Results

**Implementation Details**

In an ideal world, this implementation would be different.

**Results**

**References**

I. Dick, A. Fekete, V. Gramoli. A Skip List for Multicore. Concurrency and Computation: Practice and Experience. 2016.

V. Gramoli. More than You Ever Wanted to Know about Synchronization. PPoPP 2015

K. Fraser. Practical lock freedom. PhD thesis, Cambridge University, 2003.

T. Crain, V. Gramoli and M. Raynal. No hot spot non-blocking skip list. In ICDCS, 2013.

M. Herlihy, Y. Lev, V. Luchangco and N. Shavit. A Simple Optimistic Skiplist Algorithm. In SIROCCO, p.124-138, 2007.

T. Crain, V. Gramoli and M. Raynal. A speculation-friendly search tree. In PPoPP, p.161–170, 2012.

M. Arbel and H. Attiya. Concurrent updates with RCU: Search tree as an example. In PODC, 2014.

P. Felber, V. Gramoli and R. Guerraoui. Elastic Transactions. In DISC 2009.