CS 798-003 Paper Summary
A Practical Concurrent Binary Search Tree

Jun-Qing Lim

jq3lim@uwaterloo.ca

1 Brief Overview

Ordered maps are fundamental building blocks for many parallel programs and in-memory ordered maps are typically implemented as skip lists or self-balanced binary search tree. Concurrent skip lists are complicated and as such the paper proposed a concurrent relaxed balanced AVL tree that is fast, scalable and able to handle contention. The algorithm adopted technique of optimistic concurrency control from software transactional memory (STM) to reduce overheads and unnecessary retries.

The main optimization and performance improvement comes from leveraging the idea of controlling the size of critical regions and algorithm-specific logic in tolerating high contention. This algorithm is then evaluated against other concurrent ordered maps implementation with over a range of contention levels and operation mixes and found an improved performance in both single-threaded and multi-threaded environment.

2 Positive Points

- Good categorization and explanation of each methods design and its implementation in a consistent manner.
- Wide coverage of performance testing on different ordered maps with various workload of the function calls, key ranges, contending threads, etc., acting as fair performance measurement benchmarks on its comparison.
- Well-explained on the related background work that gives convincing reasoning behind the motivation of this proposed algorithm and the need for it.

3 Negative Points

- Lacks discussion on future work and how the ideas proposed could be expanded forward.
- The design of the algorithm contains many sophiscated elements: versioning, linking, readers blocking, etc. thus it may be challenging to implement in practice.

4 Insightful Discussions

- The paper adopted copy-on-write to support atomic clone operation and snapshot isolation during iteration. Are there any other data structures that adopted similar design for fast cloning?
- What is the memory utilization of the algorithm? How much does it scale against increasing number of threads?