CS 798-003 Paper Summary

Fast Concurrent Lock-Free Binary Search Trees

Jun-Qing Lim

jq3lim@uwaterloo.ca

1 Brief Overview

Concurrent access of data structures of multiple processes creates high contention where a process may be stalled for some time. Lock-free algorithms provide alternatives with read-modify-write instructions on a hardware level. Several lock-free concurrent binary search tree algorithms have been proposed but all of them has contention issues which drain its performance. The authors proposed a new lock-free algorithm for concurrent manipulation of binary search tree in an asynchronous shared memory system. It uses compare-and-swap (CAS) and bit-test-and-set (BTS) atomic instructions for data manipulation.

The primary optimization of the proposed algorithm comes from that it operates at the edge-level (marking edges than nodes) with flagging and tagging to allow easier coordination between conflicting operations. The algorithm is then experimented to have better performance than all other lock-free algorithms due to reasons like smaller contention window.

2 Positive Points

- Deep explanations on the leaf-based implementation alongside with its correctness proof.
- Novel approach on marking edges (that I have seen so far) instead of nodes, quite a great optimization here to be done in a single CAS.
- Great experiment with broad coverage (of reads, writes and both) and discussion on why the algorithm exhibited fast performance.
- I like how the authors discussed related work on concurrent binary search trees and how it combined multiple ideas in building this algorithm.

3 Negative Points

- The paper's algorithm might be difficult to be implemented in practice due to the sheer size of steps involved.
- The paper lacks coverage on the downsides of the proposed algorithm such as the trees built are not balanced where the height of a tree can be $O(N)$ where N is the number of nodes.

4 Insightful Discussions

- Performance comparison of this lock-free algorithm with existing non-locking algorithms?
- Any possible memory issues (i.e. high utilization) from this algorithm?