CS 798-003 Paper Summary

Lock-Free Linked Lists and Skip Lists

Jun-Qing Lim

jq3lim@uwaterloo.ca

1 Brief Overview

Mutual exclusion locks is one popular way to share data structures, but has performance issue and it is argued that lock-free shared data structures has better performance and reliability. In existing lock-free implementation, when a concurrent operation fails, the entire operation is restarted, causing performance bottleneck in "redundant work". This paper improves the performance by "recovering" from the failed operation (instead of "restarting") using backlist pointers to traverse back to a previous start point and flag bits to eliminate long-chain traversals of backlist pointers. The paper also proposed an additional implementation of skip list dictionary data structure that uses proposed linked list in maintaining each level of the skip list with performance improvement to $O(\log n)$.

2 Positive Points

- High performance algorithm that leverages the idea of reducing repeated work of traversal to optimize the steps taken.
- Deep discussion on the amortized analysis (calculation) of the linked list, giving good and accurate representation on how much improvement the algorithm has made.
- Nice analysis on correctness of the algorithm with consistent structures of definitions, invariants and proofs of linearization.

3 Negative Points

- The algorithm requires to maintain massives of backlist pointers and flags in each node, which may incur high memory usage.
- It lacks discussions or empirical evidences that compares its performance with other lock-free algorithms, but rather its ideas are merely on a "theoretical level". This makes it hard to assess its relative strengths to other algorithms.
- Lacks discussion on future work and applicability of this algorithm.

4 Insightful Discussions

- The paper has adopted a tradeoff between memory in return for higher performance, are there any other disadvantages of the proposed method?
- How well does this algorithm scale against different percentages of reads and writes?
- How does this algorithm perform in scenarios of high contention vs low contention?