Lock-free Binary Search Tree and Skip List
Project Milestone

Name: Chenhao Li; Andrew ID: chenhao3

April 7, 2022

1 Work Completed

According to my schedule, I have finished implementing the baseline concurrent dictionary with STL data structures protected by a readers-writer lock and the lock-free binary search tree without memory reclamation based on [Fra04]. I also got more work done than expected. I have finished part of the correctness test suites and performance test suites. They can already yield some satisfactory results, and will be extended in the following schedule.

I am carefully examining the lock-free skip list implementation in [Fra04]. I expect it would be less difficult to implement than the lock-free binary search tree.

2 Goals and Deliverables

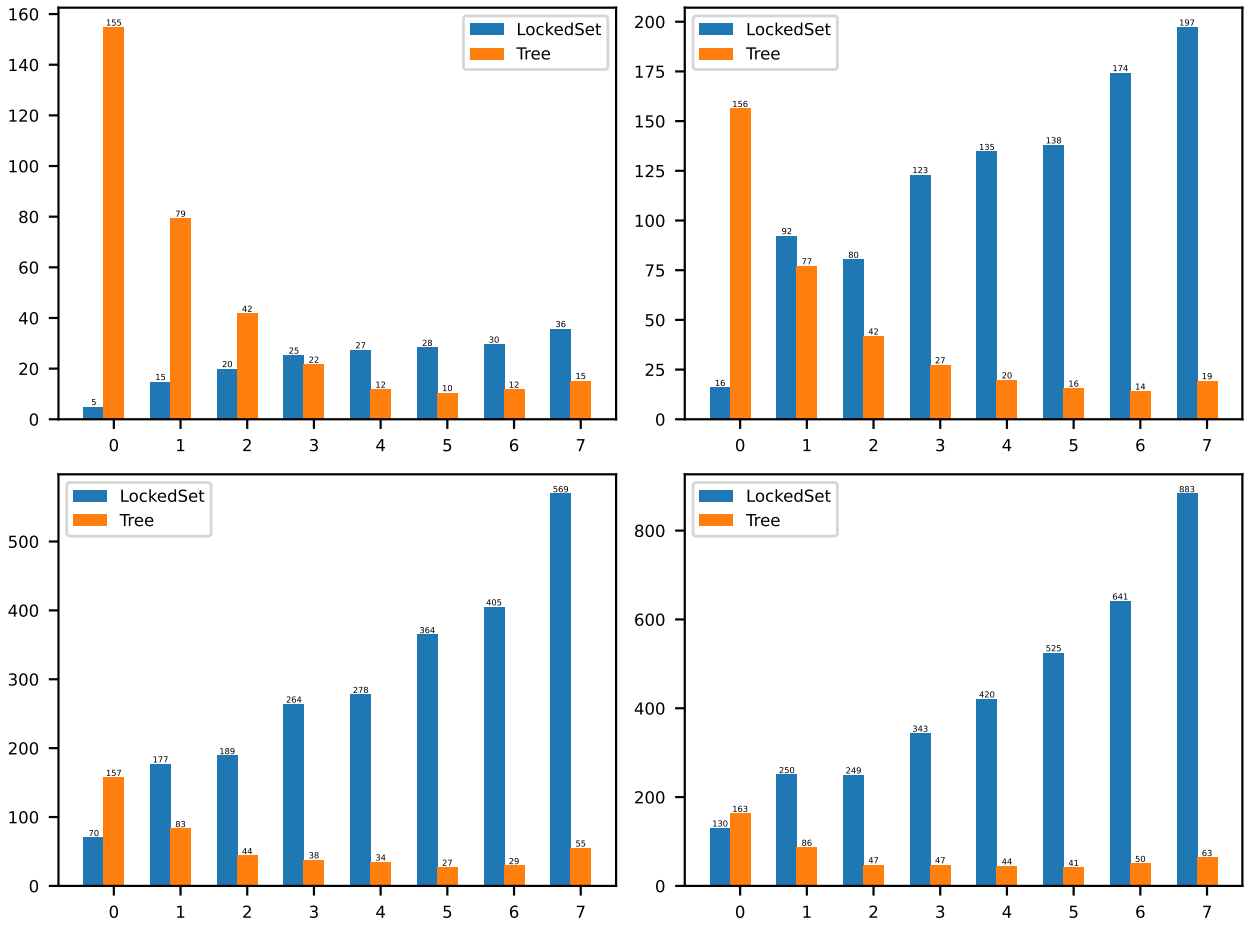
I still believe that I will be able to produce all my planned deliverables. They remain the same as those in the proposal. The project schedule also does not need to be changed. The highest priority work now is implementing the lock-free skip list.

- Implement a baseline concurrent dictionary with STL data structures protected by a readers-writer lock.
- Implement a lock-free binary search tree, with `search`, `insert` and `remove` operations. According to my experience and literature reviews, the complexities of these three operations are likely to increase in order.
- Implement a lock-free skip list, with `search`, `insert` and `remove` operations.
- Implement a garbage collection module to enable the previous two data structures to reclaim memory.
- Measure the performance of the previous three data structures under different workload patterns and configurations. The garbage collection module can also help measure memory consumption. These measurements are likely to produce plots suitable for my poster session. Hopefully, I will be able to come up with some observations and insights in the trade-off in these two data structures through experiments.

3 Preliminary Results

My performance test suites can already yield some plots. This plot demonstrates the runtime comparison of the baseline concurrent dictionary (`LockedSet`) and the lock-free binary search tree (`Tree`) under 250000 random dictionary operations. The y-axis is the runtime in milliseconds. The x-axis is the logarithm of the number of threads ($\log_2(T)$).

The tests were performed on PSC Bridges-2 Regular Memory machines. In the four sub-figures, the operation ratios of modifying the dictionary (`insert` plus `remove`) are 0.01, 0.1, 0.5, 0.9 respectively. It can be seen from them that `LockedSet` has certain performance advantages when there are very few write operations. But as soon as the ratio of writes increases slightly, it immediately exhibits very severe conflicts. On the other hand, `Tree` can benefit from multithreading well, but more write operations will also limit its acceleration.



4 Issues of Concern

While the highest priority work now is implementing the lock-free skip list, I expect it will not be much difficult. The garbage collection module is my biggest concern now. I read [Mic04], but I think hazard pointers may be difficult to implement because it involves a lot of non-trivial changes to existing code. I would prefer an implementation that is simpler but may require some sort of global locking.

References

[Fra04] Keir Fraser. Practical lock-freedom. Technical report, University of Cambridge, Computer Laboratory, 2004.

[Mic04] M.M. Michael. Hazard pointers: safe memory reclamation for lock-free objects. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):491–504, 2004.