15-418 Project Milestone Report

Aidan Smith

December 4, 2023

1 Schedule

- Dec 4 - Dec 6: Finish lock-free hash table implementation
- Dec 7 - Dec 9: Debug and optimize my implementation
- Dec 10 - Dec 11: Devise and perform benchmarks
- Dec 12 - Dec 14: Write my report and make graphics for presentation

2 Summary

During the week of thanksgiving I didn't have much time, so the bulk of my research was done after returning from break. After reading many papers and blogs about concurrent hash tables, I feel that I have a good understanding of existing techniques including and outside of Java's `ConcurrentHashMap` that I initially intended to base my implementation off of. As I just eluded to, now having learnt more about other approaches, I spend a couple days playing around with a `ConcurrentHashMap` approach and comparing it with various open-addressed approaches. I will note that I did lose a couple days hacking on the build system for my project, as I wanted to include unittesting and thought it would be a good proof of concept for importing the libraries I will using during benchmarking, however, getting this to compile on different (OSX and Ubuntu) proved more challenging and time consuming than I expected.

At this time I have decided to focus most of my efforts into an open-addressing approach, due to my proposal's initial focus on latency sensitivity. This is one of the main strengths of open-addressing as all entries can be stored in contiguous memory. Although at this point, I haven't done as much coding as I initially anticipated, I am not concerned with finishing on time, as the time I spent researching has already been paying for itself. Prototyping different approaches has really helped my understanding, making the final implementation much easier.

3 Goals and Deliverables

My goals for this project have essentially remained the same since the proposal. I have made good progress towards the main goal which is to have a working hash table to show for. The research I did gave me a good understanding of existing techniques and helped me begin the coding phase without making any major errors.

The main “nice to haves” are optimization related. Experimenting with different locking techniques under certain workloads, as well as, making optimizations for more primitive key-value types would be things I will explore if I have time after completing and benchmarking a correct solution.

For the poster session I intent to show graphs comparing my implementations performance against other concurrent hash table implementations out there.

3.1 Plan to Achieve

- Implement a correct lock-free concurrent hash map using CAS primitives
- Compare and benchmark my implementation with existing library implementations including Boost, Folly, and oneTBB

3.2 Nice to Haves

- Relax lock-free requirement, and explore mixing lock-free and locking techniques for performance benefits
- Specialized implementation for 8-byte key-value pairs that can take advantage of hardware atomics

4 Concerns

The main concern that I foresee to do with setting up other libraries for benchmarking. I spent quite some time just getting my code to compile on my machine and the Andrew machines using the `catch2` unittesting framework, so I imagine there will be some struggles linking the other libraries and stubbing them out. I have allocated extra time for this to compensate.

Other concerns are difficult to debug concurrency bugs, however, I feel that I am fairly equipped to handle this, and have allocated 3 days for "nice to haves" that can be repurposed to debugging time if need be.