15418 Final Project: Milestone Report

srisaidk, akanagal

Due Dec. 3rd 2023

- **One to two paragraphs, summarize the work that you have completed so far. (This should be easy if you have been maintaining this information on your project page.)**

  As we near this midpoint of our final project, we recount the work that has been completed so far. We are fairly on track according to the schedule outlined on our project website. We were able to find a base implementation of the B+ Tree that covers all the necessary bases, while also being simple and easy to read in the language we opted to do the project in (GoLang).

  The traditional locking mechanism that is implemented in B+ Trees is called latch-crabbing. We sought to implement this from scratch, to not only familiarize ourselves with the codebase we are working in, but also to get a feel for the "locked" version of the data structure, to give us more insight into how to design/where to start on constructing our lock-free version. We were able to complete this latch mechanism on the Insert operation, with the Find and Delete operations in the works (to be done by tomorrow night).

- **Describe how you are doing with respect to the goals and deliverables stated in your proposal. Do you still believe you will be able to produce all your deliverables? If not, why? What about the "nice to haves"? In your milestone writeup we want a new list of goals that you plan to hit for the poster session.**

  We are moving at a fairly on par pace to what we detailed in our proposal. A lot of the work we intended to complete up until this point was background/research, to set us up for success in our actual implementation. We've found several great resources, the following of which are just a couple highlights:

  - Paper on Efficient Lock-free Binary Search Trees: https://arxiv.org/abs/1404.3272 . This paper details an algorithm for concurrent internal BSTs, which is informative of what we are trying to do, but still quite disparate.

- How-to guide on Lock-free structures: https://www.baeldung.com/lock-free-programming This resource gave a high-level introduction, similar to what was presented in lecture but a bit more broken-down.
- Paper of Lock-free Monte Carlo Search Trees: Lock-free MCTS paper This again, gave us an in-depth look into an adjacent data structure, giving us the necessary inspiration to design our own implementation for a different data structure.

Aside from this, we had planned to complete implementation of latch-crabbing in the original structure, to give us an appropriate baseline to run our tests against for performance later on in the project. While we are a tad behind in this, largely due to configuring ourselves in a new repo/a codebase that is not ours, we are making good headway and plan to have this entirely completed by tomorrow (12/4). As we mentioned, the Insert operation is completely done, with Find and Delete left to be debugged.

We do in fact believe we will be able to produce the major deliverables, in that we will deliver a lock-free B+ Tree structure, and compare its performance to that of the standard latch-crabbing implementation of the structure. We have already achieved our first goal, of making our code thread-safe with a global mutex. We are currently nearing completion on our second goal, of examining fine-grained synchronization to reduce contention for this single lock. Our "nice to haves" include multiple versions of our lock-free implementation. From what we have seen in terms of such designs, it is seeming as though the construction of a fairly simple design may take longer than we expected. There is some buffer built into our schedule on the project website, so there is some time to build these multiple versions, each performing a different, smaller optimization, but the main goal is to first complete the lock-free implementation to correctness.

New Goals:

Our goals are largely unchanged from the original proposal. We want to properly build a lock-free B+ tree.

- Build latch crabbing B+ tree.
- Build lock-free B+ tree
- Test the scalability of our B+ tree on the PSC machines with 128 cores.

- **What do you plan to show at the poster session? Will it be a demo? Will it be a graph?**

  At the poster session, we will likely present some 5-7 graphs depicting various performance comparisons between the locked and lock-free implementation of this DS across varying numbers of cores. An additional 2-4 graphs may also be presented, if we are able to construct multiple versions of our implementation, with these graphs comparing the specific

optimized feature across implementations, and even to the original locked code, serving as a control of sorts.

We would like to be able to also have some sort of B+ tree visualizer, to visually showcase the correctness of our lock-free implementation, but this is more for the viewing pleasure of the audience, and does not convey that much useful/interesting information, so this is a secondary addition.

- **List the issues that concern you the most. Are there any remaining unknowns (things you simply don't know how to solve, or resource you don't know how to get) or is it just a matter of coding and doing the work? If you do not wish to put this information on a public web site you are welcome to email the staff directly.**

  The issues that concern us the most, is that we may run into difficulty designing smaller sub-components of the nuances within B+ trees. It is a fairly complex data structure, and with finals season upon us, I think the fear is that we may run into an issue that we have not given ourselves enough time to deal with. We constructed our schedule to allow for this, giving us buffer space here and there so that we can tackle these issues as they arise. For the most part, we do not anticipate anything major, it is simply a matter putting pen to paper and then getting the code done.