15418-final-project

TITLE:

An implementation of lock-free B+ Trees by Dhruti Kuchibhotla (*srisaidk*) and Adhvik Kanagala (*akanagal*)

URL:

https://github.com/dhrutik/15418-final-project/edit/main/README.md

SUMMARY:

We are going to construct a lock-free implementation of the canonical data structure upon which most databases are built, the B+ Tree. We are going to compare this algorithm's efficiency with the more traditional B+ Tree implementation that makes use of locks, specifically latch crabbing.

BACKGROUND:

The main application in the real world of B+ Trees is in the realm of database systems. They are often used for efficient storage and retrieval of data. B+ trees are widely employed in database indexing, and making them lock-free can help us curb some of the latency in accessing the structure, bettering both concurrency and parallelism of access in a multi-threaded environment.

For some background on the data structure itself, B+ trees are balanced tree data structures where each node contains a certain number of keys and pointers. The keys are sorted, allowing for efficient search, insertion, and deletion operations. The tree's structure ensures that the data is stored in a way that optimizes range queries and sequential access. The locking procedure that is most often used it commonly known as "latch crabbing." This is technique used in concurrent B+ tree implementations to reduce contention and improve parallelism over write-heavy operations, such as node splits or merges. Traditional, or naive lock implementations can lead to contention issues, especially in high-concurrency scenarios. The latch crabbing technique allows for the minimal subsection of the tree to be contained in the critical section, to decrease the amount of contention, and latency lost as a result of it. Specifically, instead of holding one single, large lock for the entire duration of the operation, latch crabbing involves temporarily releasing locks as the operation progresses. The concurrent thread moves through the tree, releasing locks on nodes that are no longer deemed "critical" to the current operation. This allows other threads to access those nodes concurrently. To illustrate a small example, if the left-most leaf node is to be split, this change may not propagate all the way to the right-most leaf, so we can deem the right-most leaf to be non-critical, and release the latch on it, so other threads can potentially perform operations on this remaining portion.

THE CHALLENGE:

This is a challenging problem because B+ trees are not inherently parallel objects. Although each subtree might be independent, B+ trees are self-balancing. This means that some operations on a subtree are not guaranteed to be isolated to that subtree, because it may cause a split or merge. In the worst case, a split or merge could have

cascading effects all the way from a leaf of the tree all the way up to the root. We would need to figure out ways to minimize dependencies on the tree.

The main dependencies are between different operations on the tree. As mentioned, it is not possible to predict exactly how each operation on the tree will affect its overall structure. The memory access characteristics will be fairly random, because each node in the tree is allocated at a different address on the heap.

The issue is that the workload consists of many operations on the same set of data. Thus, the workload cannot simply be distributed to multiple processors without running into race conditions.

RESOURCES:

Because the most interesting/relevant component of this project is not the base implementaion of B+ Trees, but rather tha conversion to a lock-free implementation, we will start with an existing implementation of the tree data structure. As of now, we intend to work in GoLang, and pull our initial code from this repository: https://github.com/collinglass/bptree. However, as we explore further, if we find that Go is a poorly-structured language to reach our goals as intended, we may pivot to a C++ base implementation (at which point we will update this README).

As of now, the main resource that we will use to guide our project is this paper, https://dl.acm.org/doi/10.14778/3402707.3402719 (PALM: parallel architecture-friendly latch-free modifications to B+ trees on many-core processors).

Jason Sewall, Jatin Chhugani, Changkyu Kim, Nadathur Satish, and Pradeep Dubey. 2011. PALM: parallel architecture-friendly latch-free modifications to B+ trees on many-core processors. Proc. VLDB Endow. 4, 11 (August 2011), 795–806. https://doi.org/10.14778/3402707.3402719

We will likely benchmark this code on the PSC machines, in order to investigate how well the data structure parallelizes.

GOALS AND DELIVERABLES:

Overall, our goals center around benchmarking various implementations of the B+ tree to compare how different optimizations can improve performance and parallelizability.

- Our first, simple goal will be to make the starter version of our code thread-safe using a global mutex on the data structure.
- Our second goal will be to examine how fine-grained synchronization can be used to further reduce lock usage. We can instead have each operation on the tree lock the specific nodes that it needs to access (latch crabbing), and relinquish control over the rest of the tree.
- The third goal will be to properly implement a lock-free B+ tree. We don't know exactly what the simplest algorithm will look like, but we will strive for simplicity and understandability.
- After we have built a lock-free B+ tree, we will focus on building multiple versions, each with one or more performance optimizations designed to reduce dependencies between cores.

If we are able to build the lock-free tree, our stretch goal will be to implement a version of the tree that is able to achieve at least a 10x performance boost over the

original global-lock tree. If not this, we hope to achieve a scalable performance speedup for our tree that is nearly linear in the number of threads.

Even if we run into trouble, the primary goal will be to implement a lock-free tree. We hope that any reasonable implementation will scale, but even if it is slow we hope it will demonstrate our understanding of parallel algorithms.

Our demo will likely look like running a series of operations on each version of the B+ tree to demonstrate speedup differences between each version. We will definitely have speedup graphs to display how well each implementation is able to scale over multiple cores. We hope to show that there is a clear visible difference in throughput between each of the major versions of our B+ tree. We also hope to show that we were able to clearly improve our performance between different versions of the lock-free tree.

PLATFORM CHOICE:

Go is a convenient language for our needs because it has a rich standard library and is easier to work with. We feel that we can have higher development velocity when working in Go, which allows for quicker iteration and debugging. Go also offers more synchronization primitives out the box, chiefly channels. Channels allow for more elegant synchronization than mutex (which is also offered) and enable more explicit communication between threads when necessary.

SCHEDULE:

Week of Nov. 13 (Note: Project proposal due Nov. 15)

- Find potential resources to read regarding background of project + guiding principles of our project
- Complete project proposal by Nov. 15

Week of Nov. 20 (Thanksgiving)

- Ensure initial implementation (from resource repo) works as intended
- Read up on lock-free implementations, specifically with respect to tree-like data structures. Examine "simpler" data structures, such as AVL trees, standard self-balancing BSTs. Implement basic lock-free versions of these/find resources that implement lock-free versions to see what is generally done in the realm of tree data structures, and to give us a sense of how to begin our implementation on this more complex tree structure.
- Have a basic implementation on B+ trees written (but not necessarily working/debugged)

Week of Nov. 27 (Note: Project milestone due Dec. 3)

- Do necessary debugging to ensure lock-free B+ tree implementation works correctly
- Complete Project milestone report
- Begin rough outline of the more formal final report

Week of Dec. 4

- Run tests comparing our implementation to the reference latch-based implementation across various metrics
- Construct tables/graphs depicting the results of these tests

- Write introduction/background/methods/resources portion of the final report

Week of Dec. 11 (Note: Final project report due Dec. 14, Poster Presentation due Dec. 15)

- Finish writing the analysis portion of the final report
- Assemble poster with our figures
- Present!!