15418 Final Project Milestone Report
B^+ Forest - A comprehensive comparison of Parallel B^+ Trees

Yutian Chen, Yumeng Liu
{yutianch,yumengli}@andrew.cmu.edu
Carnegie Mellon University

December 5, 2023

1 Project Summary

The B+ Forest project focuses on the implementation and analysis of different B+ tree structures, an essential data structure in database and filesystem management. The versions include sequential, coarse-grain lock, fine-grain lock, lock-free (PALM Tree), and a distributed version utilizing OpenMPI. This variety caters to different computational environments, ranging from single-threaded to highly concurrent systems to distributed systems.

The sequential version forms the foundation, while the coarse and fine-grain lock versions address parallelism with varying locking mechanisms. The lock-free version, based on the PALM algorithm, offers a solution for high-concurrency scenarios without traditional locks. Finally, the distributed version leverages OpenMPI for cross-node parallelism in a high-performance computing context.

This project not only aims to implement these diverse structures but also to conduct extensive benchmarking. This evaluation will compare their performance under various conditions, providing insights into the suitability and efficiency of each version in different scenarios.

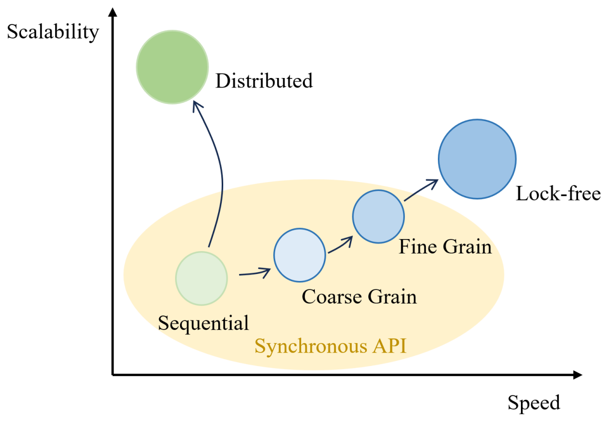


Figure 1. 5 Different Types of B+ Trees studied by the B+ Forest Project and their qualitative speed / scalability

2 Current Progress

Currently we have successfully implemented all variants of B+ trees in the scope of this project and test them thoroughly for the correctness. All of our B+ Trees support the `ITree` interface with three operations, namely `insert`, `remove`, and `get`.

```
template <typename T>
class ITree {
    public:
        void insert(T key);
        void remove(T key);
        std::optional<T> get(T key);
};
```

- Sequential B+ Tree: The foundational sequential version of the B+ tree has been successfully implemented. This version is crucial as it serves as the baseline for all subsequent parallel and distributed versions. Its completion marks a significant milestone in understanding the basic mechanics of B+ trees.
- Coarse-Grained B+ Tree: The development of a coarse-grain B+ tree with a global mutex lock represents a leap into parallel computing. This version is a primary step in exploring how traditional locking mechanisms can be employed in a parallel context, providing valuable insights into the challenges and opportunities of scaling up from sequential to parallel processing.
- Fine-Grained B+ Tree: The fine-grained lock version demonstrates a more sophisticated approach to parallelism, addressing the limitations of coarse-grained locking by reducing the scope of lock contention. This version is significant for its potential to offer better performance in multi-threaded environments where high concurrency is expected.
- Lock-Free B+ Tree (PALM Tree): The implementation of a lock-free B+ tree based on the PALM algorithm is a notable achievement. It stands out for its advanced approach to concurrency, avoiding common issues such as bottlenecks and deadlocks that are prevalent in lock-based systems. This version is particularly important for its applicability in high-concurrency scenarios, providing a solution that could significantly outperform traditional lock-based approaches.
- Distributed B+ Tree: This version represents the pinnacle of the project's current phase, showcasing advanced parallel computing techniques. By enabling efficient parallel computing across multiple nodes, this version tackles the complexities of distributed systems. The implementation of this model offers a scalable solution for large-scale data management in distributed environments.

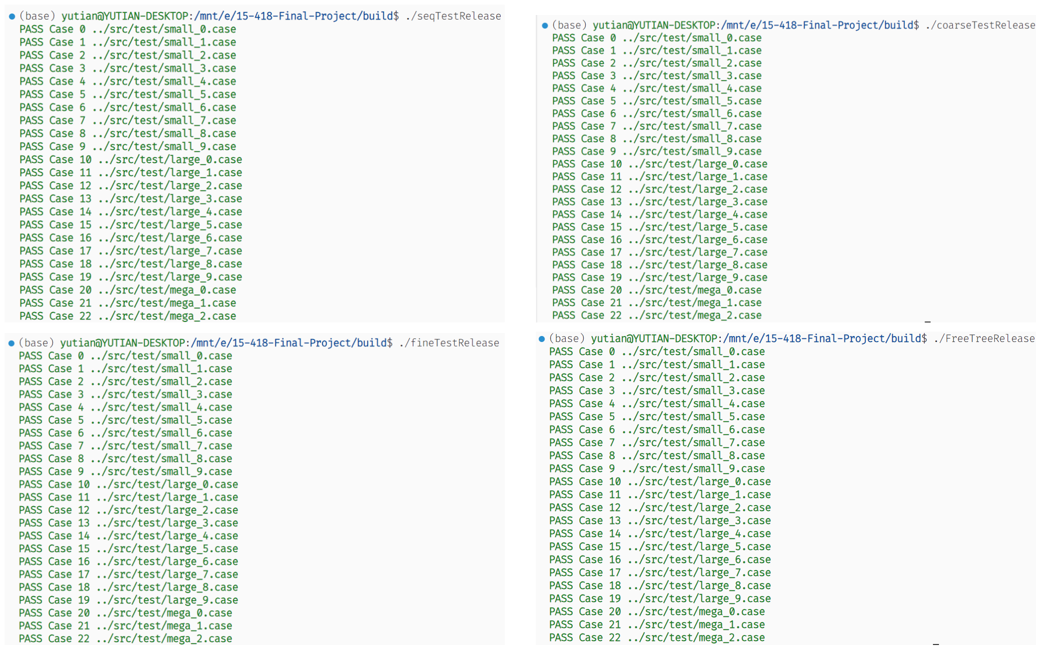
The progress made in lays the groundwork for the upcoming challenges in distributed computing with OpenMPI. Each step forward in this project contributes to a deeper understanding of parallel and distributed data structures, which is pivotal in the advancement of high-performance computing applications.

3 Goals and Deliverable

At the poster session, we plan to use graphs and diagrams to show the algorithm for sequential B+ Tree, fine-grained B+ Tree, lock-Free B+ Tree, and distributed B+ Tree. We could also do a Sequential B+ Tree visualization to help demonstrate.

4 Preliminary Results

Currently, all of our 5 versions of B+ Tree passed correctness tests, but we have not written the test engine for distributed B+ Tree for large test cases, hence we only show four graphs below. But our distributed B+ Tree is implemented and the code is in the github repo.



```
● (base) yutian@YUTIAN-DESKTOP:/mnt/e/15-418-Final-Project/build$ ./seqTestRelease
  PASS Case 0 ../src/test/small_0.case
  PASS Case 1 ../src/test/small_1.case
  PASS Case 2 ../src/test/small_2.case
  PASS Case 3 ../src/test/small_3.case
  PASS Case 4 ../src/test/small_4.case
  PASS Case 5 ../src/test/small_5.case
  PASS Case 6 ../src/test/small_6.case
  PASS Case 7 ../src/test/small_7.case
  PASS Case 8 ../src/test/small_8.case
  PASS Case 9 ../src/test/small_9.case
  PASS Case 10 ../src/test/large_0.case
  PASS Case 11 ../src/test/large_1.case
  PASS Case 12 ../src/test/large_2.case
  PASS Case 13 ../src/test/large_3.case
  PASS Case 14 ../src/test/large_4.case
  PASS Case 15 ../src/test/large_5.case
  PASS Case 16 ../src/test/large_6.case
  PASS Case 17 ../src/test/large_7.case
  PASS Case 18 ../src/test/large_8.case
  PASS Case 19 ../src/test/large_9.case
  PASS Case 20 ../src/test/mega_0.case
  PASS Case 21 ../src/test/mega_1.case
  PASS Case 22 ../src/test/mega_2.case
```

```
● (base) yutian@YUTIAN-DESKTOP:/mnt/e/15-418-Final-Project/build$ ./coarseTestRelease
  PASS Case 0 ../src/test/small_0.case
  PASS Case 1 ../src/test/small_1.case
  PASS Case 2 ../src/test/small_2.case
  PASS Case 3 ../src/test/small_3.case
  PASS Case 4 ../src/test/small_4.case
  PASS Case 5 ../src/test/small_5.case
  PASS Case 6 ../src/test/small_6.case
  PASS Case 7 ../src/test/small_7.case
  PASS Case 8 ../src/test/small_8.case
  PASS Case 9 ../src/test/small_9.case
  PASS Case 10 ../src/test/large_0.case
  PASS Case 11 ../src/test/large_1.case
  PASS Case 12 ../src/test/large_2.case
  PASS Case 13 ../src/test/large_3.case
  PASS Case 14 ../src/test/large_4.case
  PASS Case 15 ../src/test/large_5.case
  PASS Case 16 ../src/test/large_6.case
  PASS Case 17 ../src/test/large_7.case
  PASS Case 18 ../src/test/large_8.case
  PASS Case 19 ../src/test/large_9.case
  PASS Case 20 ../src/test/mega_0.case
  PASS Case 21 ../src/test/mega_1.case
  PASS Case 22 ../src/test/mega_2.case
```

```
● (base) yutian@YUTIAN-DESKTOP:/mnt/e/15-418-Final-Project/build$ ./fineTestRelease
  PASS Case 0 ../src/test/small_0.case
  PASS Case 1 ../src/test/small_1.case
  PASS Case 2 ../src/test/small_2.case
  PASS Case 3 ../src/test/small_3.case
  PASS Case 4 ../src/test/small_4.case
  PASS Case 5 ../src/test/small_5.case
  PASS Case 6 ../src/test/small_6.case
  PASS Case 7 ../src/test/small_7.case
  PASS Case 8 ../src/test/small_8.case
  PASS Case 9 ../src/test/small_9.case
  PASS Case 10 ../src/test/large_0.case
  PASS Case 11 ../src/test/large_1.case
  PASS Case 12 ../src/test/large_2.case
  PASS Case 13 ../src/test/large_3.case
  PASS Case 14 ../src/test/large_4.case
  PASS Case 15 ../src/test/large_5.case
  PASS Case 16 ../src/test/large_6.case
  PASS Case 17 ../src/test/large_7.case
  PASS Case 18 ../src/test/large_8.case
  PASS Case 19 ../src/test/large_9.case
  PASS Case 20 ../src/test/mega_0.case
  PASS Case 21 ../src/test/mega_1.case
  PASS Case 22 ../src/test/mega_2.case
```

```
● (base) yutian@YUTIAN-DESKTOP:/mnt/e/15-418-Final-Project/build$ ./FreeTreeRelease
  PASS Case 0 ../src/test/small_0.case
  PASS Case 1 ../src/test/small_1.case
  PASS Case 2 ../src/test/small_2.case
  PASS Case 3 ../src/test/small_3.case
  PASS Case 4 ../src/test/small_4.case
  PASS Case 5 ../src/test/small_5.case
  PASS Case 6 ../src/test/small_6.case
  PASS Case 7 ../src/test/small_7.case
  PASS Case 8 ../src/test/small_8.case
  PASS Case 9 ../src/test/small_9.case
  PASS Case 10 ../src/test/large_0.case
  PASS Case 11 ../src/test/large_1.case
  PASS Case 12 ../src/test/large_2.case
  PASS Case 13 ../src/test/large_3.case
  PASS Case 14 ../src/test/large_4.case
  PASS Case 15 ../src/test/large_5.case
  PASS Case 16 ../src/test/large_6.case
  PASS Case 17 ../src/test/large_7.case
  PASS Case 18 ../src/test/large_8.case
  PASS Case 19 ../src/test/large_9.case
  PASS Case 20 ../src/test/mega_0.case
  PASS Case 21 ../src/test/mega_1.case
  PASS Case 22 ../src/test/mega_2.case
```

Figure 2. Correctness Test for Sequential (upper left), Coarse grain (upper right), Fine grain (lower left), and Lock-free trees (lower right)

5 Concerns and Future Works

We faced multiple issues when we were implementing fine-grained B+ Tree, including the delete operator in our helper functions for the operation `remove`. But after digging into how C++ garbage collect works under the hood and some edge cases of concurrent B+ Tree, we were able to solve these problems.

Here are the remaining unknowns:

- For Lock-free B+ Tree, currently we are using lock-free queue from Boost, but we find it not as efficient, so we are going to come up with and write our own data structure that speedup the communication and computation.

Some Future works to do:

- Reduce the usage of `std::optional<T>` as this can cause significant overhead in computation-dense application as the benchmarking. [1]
- Benchmark all versions of B+ Tree. Specifically for Lock-free B+ Tree, need to experiment with different batch sizes and the number of workers to fit different demand (test cases)

[1] `https://stackoverflow.com/questions/23523184/overhead-of-stdoptionalt`