

Project Proposal

Pranshu Kumar¹

University of Waterloo, Waterloo, Canada, p9kumar@uwaterloo.ca

1 Introduction

Modern database systems [6] use main memory architectures, where the data is stored and processed in RAM, unlike traditional architectures where the data is stored in secondary storage on disks. With all the data in RAM, the data hierarchy becomes simplified and the delays to access data go down by potentially many orders of magnitudes. With these changes in hardware, there have to be corresponding changes in the algorithms to take advantage of the speedups. Previously, concurrent data structures would make use of locks and latches [8, 2, 3, 5], and they didn't incur a performance penalty since the system had to wait for the slow disk reads anyway. With main-memory systems, the slow reads are gone and the bottlenecks introduced by contention for locks are the main cause for slowdowns.

Lock-free algorithms offer alternate approaches [7] to concurrent operations without relying on locks and mutual exclusion. These algorithms instead of rely on atomic primitives like fetch-and-add (FAA) and compare-and-swap (CAS) [4] to ensure correctness when many threads work on the same data. This project will explore the fundamental of lock-free algorithms, including primitives like CAS (and its persistent memory variants), with the goal of understanding BzTree[1], a state of the art lock free index structure. The goal of this project will be to develop a thorough understanding of how BzTree works and achieves its efficiency and correctness properties. This will involve examining BzTree's design, correctness guarantees, progress guarantees, and concurrency mechanisms. If time permits, we would also like to benchmark an implementation of BzTree, assuming it can be converted to work in RAM without too much difficulty since we do not have access to a server with large persistent main memory.

2 Proposed Methodology

In this section we elaborate on the specifics of BzTree that we are interested in, and the questions we want to answer. The central question will be surrounding the design of the algorithm, and how CAS is used for each of the operations in the data structure. Below is a rudimentary list of related questions:

1. BzTree works in non-volatile memory and uses variants of CAS designed for persistent memory [9]. How does persistent memory CAS differ from normal volatile memory CAS?

2. The predecessor of BzTree, the Bw-Tree used a transaction manager to achieve correctness and concurrency. Is the design of BzTree simpler and easier to analyze compared to Bw-Tree?
3. The design of the algorithm and how CAS operations ensure correctness and linearizability.
4. BzTree is a latch-free algorithm, not a lock-free one. What are the limitations preventing it from being lock-free?
5. What progress guarantees does BzTree have?
6. How does BzTree recover in the event of a crash or hardware failure?
7. How is memory reclamation handled?
8. If we get time to benchmark BzTree, how does the performance scale for various workloads (search heavy, update heavy, insert heavy etc.) as the number of threads and contention increases.

3 Expected results

From our undertaking of this project, we expect to develop a comprehensive overview of BzTree and form a solid foundation for the exploration of concurrent data structures used in modern main memory database systems. With this foundation, it should be easier to analyze other algorithms and potentially develop our own. We should also be able to identify performance limitations in hardware and software used in databases.

References

1. Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. Bztrees: A high-performance latch-free range index for non-volatile memory. *Proceedings of the VLDB Endowment*, 11(5):553–565, 2018.
2. Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141, 1970.
3. Rudolf Bayer and Mario Schkolnick. Concurrency of operations on b-trees. *Acta informatica*, 9(1):1–21, 1977.
4. Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.
5. Philip L Lehman and S Bing Yao. Efficient locking for concurrent operations on b-trees. *ACM Transactions on Database Systems (TODS)*, 6(4):650–670, 1981.
6. Tobin J Lehman and Michael J Carey. Query processing in main memory database management systems. In *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*, pages 239–250, 1986.
7. Justin J Levandoski, David B Lomet, and Sudipta Sengupta. The bw-tree: A b-tree for new hardware platforms. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 302–313. IEEE, 2013.
8. John Turek, Dennis Shasha, and Sundeep Prakash. Locking without blocking: making lock based concurrent data structure algorithms nonblocking. In *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 212–222, 1992.

9. Tianzheng Wang, Justin Levandoski, and Per-Ake Larson. Easy lock-free indexing in non-volatile memory. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 461–472. IEEE, 2018.