

The State of the ART: An Index Structure for Main Memory Databases

JONAS FRITSCH, Technische Universität München

With recent trends in hardware, main memory capacities have grown to an extent where most traditional DBMS can fit entirely into main memory. These changes introduced a new shift of the performance bottleneck from disk-based I/O to Main-Memory access.

While previous Index-Structure like the B-Tree were optimized for minimizing disk access, the adaptive radix tree (ART) is a trie-based index structure designed explicitly for in-memory usage. It utilizes newer architecture features like SIMD and caching effectively and compresses its structure dynamically, both horizontal and vertical. With these measures, ART achieves a performance that beats other state-of-the-art order-preserving index structures in both insertion and single-lookup time.

1 INTRODUCTION

The architecture of DBMS has constantly been evolving due to advances in hardware. Over the last few decades, main memory capacities increased from several megabytes up to thousands of gigabytes, such that nowadays, databases can fit entirely into main memory. This change significantly impacted the general architecture of DBMS, which resulted in performance improvements by several factors [1], [7].

The design of index structures used to query a set of data more efficiently was heavily influenced by the main performance bottleneck of disk I/O in traditional disk-based DBMS. Original index structures like the B-Tree designed to minimize disk accesses perform poorly in an in-memory environment.

The T-Tree [3] was one of the first index structures proposed for main memory DBMS. However, over the last 35 years, the hardware landscape changed dramatically, causing T-Trees and all other index structures not explicitly designed with caching effects in mind to be rendered inefficient. Further focus on developing cache-sensitive index structures resulted in many different search tree variants.

Cache-sensitive search trees (CSS-Trees) [4], while utilizing cache lines efficiently, introduce a significant overhead for updates as the tree is compactly stored in an array. Similarly, the more recent k-ary search tree [6] and the Fast Architecture Sensitive Tree (FAST) [2] both utilize Single Instruction Multiple Data (SIMD) instructions for data-level parallelism (DLP) to increase performance. However, as static data structures, they do not support incremental updates. A way to circumvent this limitation is to use a delta mechanism where another data structure stores differences and is periodically merged into the static structure. This comes at an additional performance cost. The cache-conscious B⁺-Tree (CSB⁺) [5] introduced as a variant of B⁺-Trees improves cache utilization by reducing the need to store all different child pointers in each node.

Hash-Tables have been a popular indexing choice for main memory databases as they provide optimal $O(1)$ - as opposed to $O(\log n)$ for search trees - single-lookup and update time on average. Many different hashing schemes and hash functions have been developed over time, but Hash-Tables generally do not support any range-based queries due to the nature of hash functions. Additionally,

Hash-Tables can require complete re-hashing with $O(n)$ complexity upon reaching its load balance.

2 ADAPTIVE RADIX TREE

Lorem ipsum.

3 CONSTRUCTING BINARY-COMPARABLE KEYS

Lorem ipsum.

4 EVALUATION

Lorem ipsum.

5 RELATED WORK

Lorem ipsum.

6 CONCLUSION AND FUTURE WORK

Lorem ipsum.

REFERENCES

- [1] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. 2008. OLTP through the Looking Glass, and What We Found There. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada) (SIGMOD '08). Association for Computing Machinery, New York, NY, USA, 981–992. <https://doi.org/10.1145/1376616.1376713>
- [2] Changkyu Kim, Jatin Chhugani, Nadathur Satish, Eric Sedlar, Anthony D. Nguyen, Tim Kaldewey, Victor W. Lee, Scott A. Brandt, and Pradeep Dubey. 2010. FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA) (SIGMOD '10). Association for Computing Machinery, New York, NY, USA, 339–350. <https://doi.org/10.1145/1807167.1807206>
- [3] Tobin J Lehman and Michael J Carey. 1985. *A study of index structures for main memory database management systems*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
- [4] Jun Rao and Kenneth A Ross. 1998. Cache conscious indexing for decision-support in main memory. (1998). <https://doi.org/10.7916/D8T441ZB>
- [5] Jun Rao and Kenneth A. Ross. 2000. Making B+- Trees Cache Conscious in Main Memory. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Dallas, Texas, USA) (SIGMOD '00). Association for Computing Machinery, New York, NY, USA, 475–486. <https://doi.org/10.1145/342009.335449>
- [6] Benjamin Schlegel, Rainer Gemulla, and Wolfgang Lehner. 2009. K-Ary Search on Modern Processors. In *Proceedings of the Fifth International Workshop on Data Management on New Hardware* (Providence, Rhode Island) (DaMoN '09). Association for Computing Machinery, New York, NY, USA, 52–60. <https://doi.org/10.1145/1565694.1565705>
- [7] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. 2015. In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1920–1948. <https://doi.org/10.1109/TKDE.2015.2427795>