University of Zurich[UZH]

Department of Informatics

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

Prof. Dr. Michael Böhlen
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

Zürich, 13. August 2018

MSc Basismodul
Topic: Implementing the Adaptive Radix Tree (ART)

The advent of in-memory database systems has fundamentally changed the requirements for index structures. Block transfers from disks are no longer the deciding factor in index performance, while efficient CPU cache utilization has become an important performance characteristic. These new requirements make standard B+ tree indexes inappropriate for in-memory settings. While hash indexes are an efficient method for point queries in in-memory systems, they do not support range queries.

The Adaptive Radix Tree (ART) [1] promises to fill this gap, offering efficient support for updates, point and range queries for in-memory databases. A standard radix tree splits the digital representation of a search-key into pieces (i.e., bit sequences) of fixed span s. These pieces are hierarchically indexed. A node in a radix tree has a fanout of 2^s pointers, and a s bit piece of the key is used as offset into a node to either find a pointer to the next inner node of the tree or a pointer to a value in case of a leaf node. The static parameter s affects the size and performance of the tree. If s is small, the tree is compact but deep, which slows down queries. Likewise, if s is large, the tree requires more space, but offers better query performance.

Like a standard radix tree, ART uses nodes that logically can store up to 2^s pointers, where typically $s = 8$. ART, however, adaptively changes the actual capacity of a node depending on the number of non-null pointers that are stored in a node. ART defines a set of four node types (`Node4`, `Node16`, `Node48`, and `Node256`) that are highly optimized (e.g., through SIMD instructions). Adapting a node's capacity promises a lower storage consumption, while retaining good query performance. Standard radix trees often create chains of nodes (each with only one child) in the tree when long keys are inserted in the tree. Likewise, these long chains are pruned if keys are deleted. ART implements two optimizations, path compression and lazy expansion, that eliminate these long chains in order to save space.

The goal of this project is to study, implement, and empirically evaluate ART.

Tasks

1. Study and understand the Adaptive Radix Tree [1].
2. Implement a prototype of ART in C++. The following functionality must be supported: insertion, deletion, and point and range queries.
3. Evaluate your implementation of ART experimentally.
   - Evaluate the runtime for insertion, deletion, and point and range queries.
   - Report the number of times ART applies lazy expansion and path compression.
   - Use the DELL dataset provided to you.
4. Summarize your findings in a short report.

Optional Tasks

1. Compare ART with C++'s built in `std::map` (red-black tree based) and a standard textbook radix tree.
2. Implement range queries
3. Extend your implementation of point and range queries to use SIMD instructions.

References

[1] V. Leis, A. Kemper, and T. Neumann. The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 38–49, 2013.

Supervisor: Kevin Wellenzohn (wellenzohn@ifi.uzh.ch)

Start date: 02.07.2018

End date: 02.10.2018

Oral exam date: 25.09.2018, 3pm

University of Zurich
Department of Informatics

Prof. Dr. Michael Böhlen
Professor