Sphinx: A Succinct Perfect Hash Index for x86

Sajad Faghfoor Maghrebi
smaghrebi@cs.toronto.edu

Niv Dayan
nivdayan@cs.toronto.edu

Computer Science
University of Toronto

Motivation

Modern log-structured key-value stores rely on in-memory indexes to map keys to their storage locations. Fingerprint-based indexes reduce memory usage by storing a small hash-digest of the keys, but as they shrink further, their shorter fingerprints sharply increase the false-positive rate, triggering excess I/Os [1]. Dynamic perfect hash tables eliminate false positives, but their compact bit-stream is hard to encode and decode, which results in higher latency [2]. This creates a tension between memory efficiency and low latency on commodity hardware, leading prior work to adopt FPGAs and ASICs to mitigate these limitations at the cost of more expensive hardware [2].

Overview

In this presentation, we introduce ***Sphinx*** **[3]**, a dynamic, succinct perfect hash table engineered for x86. Sphinx distinguishes all present keys without storing them by cleverly maintaining an index of approximately 4 bits per entry and sustains low latency, almost as fast as a single cache miss per query.

Sphinx achieves memory efficiency and fast processing through the following carefully designed techniques: (i) Sphinx consists of 256-bit blocks. The layout of each block is structured in such a way that it can be parsed using modern CPU bit manipulation instructions[1], eliminating the need for inefficient bit-by-bit traversal. (ii) A small, cache-resident Decoder is used to handle common patterns among the slots encountered during traversal through table lookups. (iii) Instead of relying on pointer chains, which tend to increase cache misses, Sphinx uses near-blocks for cases where a block cannot handle all of its assigned entries. (iv) Expansion is handled smoothly and incrementally by grouping blocks together and expanding them one at a time, rather than resizing the entire structure at once.

Sphinx uses rank and select optimized for modern CPUs to traverse the blocks. The *rank(b, i)* command counts the number of set bits (1s) from the start of Bitmap b up to and excluding Index i. The *select(b, i)* command finds the offset of the i^{th} set bit in Bitmap b.

Sphinx then creates the block structure to be rank/select friendly. In other words, within a block, the number of set bits to pass before reaching the target slot is carefully designed to be predictable. Using rank and select, we can skip over irrelevant slots and jump directly to the one of interest in constant time. The selected slot is then decoded by the cache-resident Decoder in about 98% of cases, minimizing bit-level parsing and keeping the hot path entirely in L1 cache.

Results

We compare Sphinx to the best alternatives and show that it leads to a 2x reduction in query latency, update latency, and memory footprint.

References

[1] B. Chandramouli, G. Prasaad, D. Kossmann, J. J. Levandoski, J. Hunter, and M. F. Barnett. Faster: A concurrent key-value store with in-place updates. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 275–290, 2018.

[2] N. Dayan, M. Twitto, Y. Rochman, U. Beitler, I. B. Zion, E. Bortnikov, S. Dashevsky, O. Frishman, E. Ginzburg, I. Maly, A. P. Meir, M. Mokryn, I. Naiss, and N. Rabinovich. The end of moore's law and the rise of the data processor. *Proc. VLDB Endow.*, 14(12):2932–2944, jul 2021.

[3] S. F. Maghrebi and N. Dayan. Sphinx: A succinct perfect hash index for x86. *Proc. VLDB Endow.*, 18(11):4424–4437, Sept. 2025.

[1]Advanced Bit Manipulation Instructions (BMI1 and BMI2), such as PEXT, POPCNT, TZCNT, etc.