Space-Efficient Dynamic Filter Expansion

Hyuhng Min Kim

Department of Computer Science
University of Toronto
hmkim@cs.toronto.edu

1 Introduction

A filter is a space-efficient probabilistic data structure that encodes a set of keys to support approximate membership queries. It ensures that no existing key is ever reported as missing (no false negatives), while tolerating a small probability of false positives to achieve substantial space savings. Because of their compact nature, filters are typically kept in fast memory such as DRAM or SRAM, even when the underlying dataset resides on slower storage or remote servers. This allows systems to efficiently eliminate non-existent keys before performing costly disk or network lookups, thereby reducing both latency and I/O overhead.

Many real-world applications manage ever-growing datasets with unpredictable final sizes. Traditional filters like Bloom filters [2] are static and must be pre-sized to meet a desired false-positive rate. Rebuilding the filter as data grows is impractical, as it requires a full dataset scan. To overcome this, dynamic filters have been proposed that expand incrementally without reconstruction. The simplest approach links multiple filters in sequence, directing new insertions to the latest one but forcing queries to check all, which raises lookup latency and false-positive probability.

Modern dynamic filters, such as quotient filters [1], store fingerprints compactly in hash tables. When expansion is required, they double the table size and utilize one fingerprint bit to redistribute entries without rehashing keys. The latest design, Aleph Filter [3], offers constant-time operations and maintains a stable false-positive rate under growth. Yet this flexibility comes at a steep memory cost, diminishing the core space advantage of filter-based designs.

2 Challenges

Unlike traditional hash tables, which can resize by any factor through key rehashing, filters cannot access original keys and thus cannot rehash. They typically double in size by dedicating one fingerprint bit to slot addressing, resulting in at least 50% space overhead right after expansion. This inefficiency is hard to avoid since a bit is the smallest storage unit, making finer-grained scaling challenging.

Furthermore, during expansion, the system allocates a new, larger filter while retaining the original one in memory to facilitate gradual entry migration. Although this additional space consumption is temporary and subsides once migration completes, it nevertheless requires extra memory to be reserved during the process. Such transient overhead can lead to memory pressure, causing other resources to be swapped out and ultimately degrading overall system performance.

3 Our Work

We propose a dynamic filter that addresses both inefficiency and temporary space overhead in prior designs. It stores fingerprints in a compact hash table and expands by less than a factor of two using a novel technique called Stretching. Stretching progressively increases entry spacing until the next power of two, when one fingerprint bit is reassigned to remap entries. Further, an in-place expansion mechanism with a sub-linear indirection layer removes transient memory costs and keeps lookups cache-efficient.

References

[1] Michael A Bender, Martin Farach-Colton, Rob Johnson, Bradley C Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. In *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, 2011.

[2] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Niv Dayan, Ioana-Oriana Bercea, and Rasmus Pagh. Aleph filter: To infinity in constant time. *Proceedings of the VLDB Endowment*, 17(11):3644–3656, 2024.