The Cuckoo Filter: It's Better Than Bloom

Bin Fan (student author), David G. Andersen, *Michael Kaminsky
{binfan,dga}@cs.cmu.edu, michael.e.kaminsky@intel.com
*Carnegie Mellon University, *Intel Labs*

Approximate set-membership tests, exemplified by Bloom filters [1], have numerous applications in networking and distributed systems. A Bloom filter is a compact data structure to quickly answer if a given item is in a set with some small false positive probability ε. Due to its simplicity and high space efficiency, Bloom filters become widely used in network traffic measurement, packet routing, distributed caching, network intrusion detection, distributed joins in databases and so on.

Limitations of conventional Bloom filters One major limitation of Bloom filters is that the existing items cannot be removed without rebuilding the entire filter. Several proposals have extended classic Bloom filters to support deletion, but with significant space overhead: *counting Bloom filters* [3] are 4× larger and the recent *d-left counting Bloom filters* (dl-CBFs) [2], which adopt a hash table-based approach, are still about 2× larger than a space-optimized Bloom filter.

Cuckoo filter overview This work shows that supporting deletion for approximate set-membership tests does not require higher space overhead than conventional Boom filters. We propose the *cuckoo filter*, a practical data structure that can replace both counting and traditional Bloom filters with three major advantages: (1) it supports adding and removing items dynamically; (2) it achieves higher lookup performance; and (3) it requires less space than a space-optimized Bloom filter when the target false positive rate ε is less than 3%. A cuckoo filter is a compact variant of a cuckoo hash table [4] that stores *fingerprints* (i.e., a short hash) for each item inserted, instead of the entire item. Cuckoo hash tables can have more than 95% occupancy, which translates into high space efficiency when used for set membership.

Challenge and our solution Cuckoo hashing associates each item with multiple possible locations in the hash table by different hash functions. This flexibility in where to store an item improves the table's occupancy, but also raises several challenges, the most important of which we discuss here.

When inserting new items, cuckoo hashing often refines its previous location assignment by relocating the existing fingerprints to their alternative locations. A straightforward but *space-inefficient* solution is to store each item (perhaps external to the table) in addition to its fingerprint; then an item's alternate location can be calculated by fetching and rehashing the original item. To avoid storing all the items, we use *partial-key cuckoo hashing* to find each item's alternate location using only its fingerprint, and thus we can add new items dynamically to the cuckoo filters without storing all inserted items somewhere.

Cuckoo filters are easy to implement—our implementation consists of only 500 lines of C++ code. The following table compares the space consumption achieved by the space optimized (counting) Bloom filters and our cuckoo filters with false positive rate $\varepsilon = 1\%$ and 0.01%. Micro-benchmark results also show that, cuckoo filters provide faster lookup speed than the space-optimized Bloom filters, especially for workloads with a large fraction of positive queries.

	bits per item		deletion
	$\varepsilon = 1\%$	$\varepsilon = 0.01\%$	support
Bloom filter	9.6	19.1	no
counting Bloom filter	38.3	76.5	yes
cuckoo filter	9.1	16.1	yes

Summary Cuckoo filters provide the flexibility to add and remove items dynamically while achieving higher lookup performance and using less space than conventional Bloom filters, for applications that require low false positive rates ($< 3\%$). We believe cuckoo filters could become preferable in serving approximate set-membership queries for a broad variety of network and distributed applications.

References

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

[2] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *14th Annual European Symposium on Algorithms, LNCS 4168*, pages 684–695, 2006.

[3] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. ACM SIGCOMM*, pages 254–265, Vancouver, British Columbia, Canada, Sept. 1998.

[4] R. Pagh and F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51 (2):122–144, May 2004.