Cheap and Large CAMs for High Performance Data-Intensive Networked Systems

Presented By: Abhinav Dutta
Abstract

- Authors build cheap and large CAMs, or CLAMs, using a combination of DRAM and flash memory.
- Using DRAM to maintain hash tables is quite expensive, while on-disk approaches are too slow.
- In contrast, CLAMs cost nearly the same as using existing on-disk approaches but offer orders of magnitude better performance.
- Their design leverages an efficient flash-oriented data-structure called BufferHash that significantly lowers the amortized cost of random hash insertions and updates on flash.
- BufferHash also supports flexible CLAM eviction policies.
- Prototype CLAMs using SSDs from two different vendors. They can offer average insert and lookup latencies of 0.006ms and 0.06ms (for a 40% lookup success rate), respectively.
Buffer Hash

- a novel data structure
- Key idea behind BufferHash is that instead of performing individual random insertions directly on flash, DRAM can be used to buffer multiple insertions and writes to flash can happen in a batch.
- This shares the cost of a flash I/O operation across multiple hash table operations, resulting in a better amortized cost per operation.
- Idea of batching operations for hash tables is novel
Figure 1: A Super Table
**Buffer.** This is an in-memory hash table where all newly inserted hash values are stored, built using existing fast algorithms. A buffer can hold a fixed maximum number of items, when it reaches its capacity, the entire buffer is flushed to flash.

**Incarnation table.** This is an in-flash table that contains old and flushed incarnations of the in-memory buffer. The table contains \( k \) incarnations, where \( k \) denotes the ratio of the size of the incarnation table and the buffer. The incarnations are placed in a circular manner i.e. the oldest incarnation exists at the tail end and the latest incarnation at the head end.

**Bloom filters.** Whenever a hash is flushed to the flash memory the filter is also assigned to it. To search for a particular hash key, test the Bloom filters for all incarnations; if any Bloom filter matches, then the corresponding incarnation is retrieved from flash and looked up for the desired key.

*A Bloom filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set.*

*The hash table which is flushed into the Flash is called the Incarnation.*
(1) “A key idea behind BufferHash is that instead of performing individual random insertions directly on flash, DRAM can be used to buffer multiple insertions and writes to flash can happen in a batch.” Very briefly explain the difference between the ways of FAWN and BufferHash in which they locate a KV pair written on the flash?

Buffer hash has an index containing the location of the KV pair (tested using Bloom Filter) so we are not required to look up all the hash tables but the FAWN looks up in the whole Flash memory in a sequential manner making the memory overhead high.
BufferHash consists of multiple super tables. Each super table has three main components: a buffer, an incarnation table, and a set of Bloom filters.” Use Figure 1 to describe BufferHash’s data structure.

These components are split into two level hierarchy:

Components in the higher level are maintained in DRAM, while those in the lower level are maintained in flash.

The DRAM consists of the Buffer and the Bloom filters and the Flash memory consists of the incarnation tables.

- The Buffer is an in-memory hash table where all newly inserted hash values are stored. The buffer takes all the newly inserted values into the hash table in DRAM and then flushes the contents to the Flash once the buffer memory is full.
- A Bloom Filter is assigned to each of the buffer when it is initialized after flushing contents to the flash. The Bloom filters are indexed to provide for the lookup operations.
- The hash table which is flushed into the Flash is called the Incarnation. The incarnations are placed in a circular manner i.e. the oldest incarnation exists at the tail end and the latest incarnation at the head end.
Buffer is the collection of KV pairs in the Hash Table present on the DRAM where all the new hashes are added.
Incarnations are the buffers which are moved into the Flash memory when the buffer gets filled up to the buffer capacity.
(4) “Since the incarnation table contains a sequence of incarnations, the value for a given hash key may reside in any of the incarnations depending on its insertion time.” Please explain why Bloom filters are needed.

Since the Hash Tables are updated in batches, KV pair lookups will be very expensive on flash. But by using a bloom filter for each batch makes lookups faster and less expensive. To search for a particular hash key, we first test the Bloom filters for all incarnations; if any Bloom filter matches, then the corresponding incarnation is retrieved from flash and looked up for the desired key.
(5) “A super table supports all standard hash table operations” Describe the steps involved in insert, lookup, update/delete operations.

**Insert:** To insert a (key, value) pair, the value is inserted in the hash table in the buffer. If the buffer does not have space to accommodate the key, the buffer is flushed and written as a new incarnation in the incarnation table.

**Lookup:** A key is first looked up in the buffer. If found, the corresponding value is returned. Otherwise, Bloom filters are used to check for in-flash lookups. In-flash incarnations are examined in the order of their age until the key is found.

**Update/Delete:** Flash does not support small updates/deletions efficiently; hence, bufferhash support them in a lazy manner. The updates are done when the hash tables are flushed into the Flash memory.
(6) Does BufferHash support range search?

No, BufferHash does not support range search, as seen in FAWN and SkimpyStash hashing does not support range search.