Wayne State University

ECE 7650: Scalable and Secure Internet Services and Architecture

SkimpyStash: RAM Space Skimpy Key-Value Store on Flash-based Storage

Seminar QA Report

Amith Nagaraja

Student ID: Fs7045

Access ID: 004470192
Introduction to SkimpyStash: A RAM space key-value store on flash storage

SkimpyStash is a RAM space skimpy key-value store on flash storage which provides high throughput and low latency server applications. It uses hash table directory in RAM. It indexes key-value pairs stored in log-structured manner on flash. The distinguishing feature of SkimpyStash is the design goal of extremely low RAM footprint at about 1 (± 0.5) byte per key-value pair, which is more aggressive than earlier designs. To break the barrier of a flash pointer (say, 4 bytes) worth of RAM overhead per key, it “moves" most of the pointers that locate each key-value pair from RAM to flash itself. This is realized by:

(i) Resolving hash table collisions using linear chaining, where multiple keys that resolve (collide) to the same hash table bucket are chained in a linked list.

(ii) Storing the linked lists on flash itself with a pointer in each hash table bucket in RAM pointing to the beginning record of the chain on flash, hence incurring multiple flash reads per lookup. Two further techniques are used to improve performance

(iii) Two-choice based load balancing to reduce wide variation in bucket sizes (hence, chain lengths and associated lookup times), and a bloom filter in each hash table directory slot in RAM to disambiguate the choice during lookup

(iv) Compaction procedure to pack bucket chain records contiguously onto flash pages so as to reduce flash reads during lookup

The evaluations on commodity server platforms with real-world data centre applications show that SkimpyStash provides throughputs from few 10,000s to upwards of 100,000 get-set operations/sec.
Question and Answers

1. Our base design uses less than 1 byte in RAM per key-value pair and our enhanced design takes slightly more than 1 byte per key-value pair."In FAWN, even a pointer to a KV pair needs a 4-byte pointer. How can possibly SkimpyStash achieve such a low memory cost for metadata?

- Skimpy Stash Moves most of pointers that locate key-value pair to flash itself instead of RAM
- It uses linear chaining to avoid collision(hash table)
- Pack bucket chain records contiguously onto flash pages to reduce flash reads

2. "SkimpyStash uses a hash table directory in RAM to index key-value pairs stored in a log-structure on flash." Why are key-value pairs on the flash organized as a log?

- Log structured data organization on flash provides high write throughput
- As all the updates to the data and the metadata are written in a sequential order in the log.

3. “The average bucket size is the critical design parameter that serves as a powerful knob for making a continuum of tradeoffs between low RAM usage and low lookup latencies.” Please explain this statement.

- Smaller the bucket size, linked list will be larger in size – hence more lookup latency
- Larger the bucket size, smaller would be the linked list size hence smaller the lookup latency – More RAM utilization
- Hence average bucket size is the critical design parameter
4. “The client [write] call returns only after the write buffer is flushed to flash.” Why cannot such a call be acknowledged earlier?

- It can’t acknowledge earlier as memory is volatile in RAM buffer
- Volatile memory is computer storage that maintains data only when device is powered on.

5. **Basic functions: Store, Lookup, Delete** Explain how these basic functions are executed?

- **Store**:
  - Write to RAM Write Buffer → Certain no of keys fills flash page → Write to flash → Insert to RAM HT directory
- **Lookup**:
  - GET → RAM Write Buffer → IF MISS → Lookup HT directory in RAM → Search key-value pair records on flash in respective bucket
- **Delete**:
  - Key to be Deleted → Insert NULL value for the key → NULL entry and earlier inserted values of the key on flash will be garbage collected
  - As garbage records in flash log exceeds threshold, garbage collection operation is initialized (reclaim flash storage)
  - Starts to scan KV pairs from head of the log
  - Discards garbage KV pairs

6. “The chain of records on flash pointed to by each slot comprises the bucket of records corresponding to this slot in the HT directory. This is illustrated in Figure 3.”. Please use the figure to describe SkimpyStash’s data structure. Also explain how lookup, insert, and delete operations are executed.
• Resolve collision using linear chaining
• Store Linked list on flash
• Pointer in each HT bucket In RAM pointing to Beginning of record of chain on flash
• Each KV pair consists of Key, Value fields and a pointer to the next field
• The following describes how Lookup, Insert, Delete operation works
  o Lookup:
    ▪ Key uses HASH function H to obtain HT directory bucket that this key belongs to.
    ▪ Uses pointer stored in that slot to follow chain of records on flash to search key
    ▪ Upon key match, it returns the value
  o Insert:
    ▪ Hash function H to obtain HT directory bucket that key belongs to
    ▪ Record is created to the inserted KV pair
    ▪ Each time new record is inserted to beginning of chain.
  o Delete:
    ▪ Its same as insert/update operation with null value for the key
    ▪ Garbage collection used
7. “Because we store the chain of key-value pairs in each bucket on flash, we incur multiple flash reads upon lookup of a key in the store.” Please explain how this issue can be alleviated.

- This can be alleviated by:
  - Periodically compacting chain on flash in bucket placing valid keys in chain contiguously on one or more flash pages

8. “..two-choice based load balancing strategy is used to reduce variations in the number of keys assigned to each bucket”. Explain how this is achieved.

- Each key would be hashed to 2 candidates HT directory buckets using 2 hash function h1, h2
- Inserting the key-values to the one that has fewer elements
9. "...when the last record in a bucket chain is encountered in the log during garbage collection, all valid records in that chain are compacted and relocated to the tail of the log.". Please explain how garbage is collected.

- Certain amount (configurable) of garbage is accumulated in the log
- Pages on flash from the head of log is recycled
- Valid entries from head of log are written back to end of log while invalid entries are skipped
- This effectively leads to the design decision of garbage collecting entire bucket chains on flash at a time.