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

ChenYu Fan
2018/04/03
Motivation

- Many designs store the full key-value pairs on flash memory and use a small amount of metadata per key-value pair in RAM to support faster insert and lookup operations.
- The amount of available RAM space limits the total number of key-value pairs that could be indexed on flash memory.
- The distinguishing feature of SkimpyStash is the design goal of extremely low RAM footprint at about 1 byte per key-value pair.
“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 it be possible for SkimpyStash to achieve such a low memory cost for metadata?
Hash Table with Chain

- One pointer to many KV pairs.
- Resolve hash table collisions using linear chaining.
- Ex. pointer uses 4 bytes in RAM space. If 4 key-value pairs in a chain, each key-value uses 4 / 4 = 1 byte RAM space.
Question 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 file system

- Flash
  - A major drawback of the flash memory is that it does not allow in-place updates (overwrite).
  - Page write operations in a flash memory must be preceded by an erase operation and within a block, pages need be to written sequentially.

- Use log-structured file system with append operation involves a flash page worth of data, typically 2KB or 4KB
Question 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.
How long is the chain
Question 4

“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.
Lookup And Insert
Delete
Question 5

“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. [Hint: please refer to Section “Compaction to Reduce Flash Reads during”]
Compaction

Flas
Question 6

“..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.
Power of Two Choice

- Each key would be hashed to two candidate HT directory buckets, using two hash functions $h_1$ and $h_2$, and actually inserted into the one that has currently fewer elements.
Question 7

"... 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.
Garbage Collect

- Scan from the head of log.
- Encounter a key-value pair.
  - Valid: copy it to the tail of log.
  - Invalid: skip it (recycle space)