SILT: A Memory-Efficient, High-Performance Key-Value Store

NOTE: Your slides/presentation need to cover the assigned sections and questions in a clear and well-organized manner. You are allowed to borrow contents from other resources, such as online slides, as long as you acknowledge them. For a slide that covers a given question, please print the question on the slide. However, you don’t have to answer the question using a long paragraph of text on the slide. Instead, use bullet points, graph, animation, or oral explanation to answer the question. In your Q&A report, use text to more thoroughly answer the questions.

You only need to cover Section 1 -- Sections 3.3

(1) "Figure 1: The memory overhead and lookup performance of SILT and the recent key-value stores. For both axes, smaller is better." Explain the positions of FAWN-DS, Skimpystash, BufferHash, and SILT on the graph.

(2) Two design goals of SILT are low read amplification and low write amplification. Use any KV store we have studied as an example to show how these amplifications are produced.

(3) Describe SILT’s structure using Figure 2 (Architecture of SILT). Compared with LevelDB, SILT has only three levels. What’s concern with a multi-level KV store when it has too few levels?

(4) Use Figure 3 (Design of LogStore: an in-memory cuckoo hash table (index and filter) to describe how a PUT request and a GET request is served in a LogStore. In particular, explain how the tag is used in a LogStore.

(5) Use Figure 4 to explain how a LogStore is converted into a HashStore?

(6) “Once a LogStore fills up (e.g., the insertion algorithm terminates without finding any vacant slot after a maximum number of displacements in the hash table), SILT freezes the LogStore and converts it into a more memory-efficient data structure.” Compared to LogStore, what’s the advantage of HashStore? Why doesn’t SILT create LogStore at the beginning (without first creating LogStore)?

(7) “When fixed-length key-value entries are sorted by key on flash, a trie for the shortest unique prefixes of the keys serves as an index for these sorted data.” While a SortedStore is fully sorted, could you comment on the cost of merging a HashStore with a SortedStore? Compare this cost to the major compaction cost for LevelDB?

(8) “Figure 5 shows an example of using a trie to index sorted data.” Please use Figures 5 and 6 to explain how the index of a SortedStore is produced.