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

NOTE: Your slides/presentation only need to cover background information necessary to answer the given 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. The presentation should be mostly question-focused and proceed mostly in a Q&A format. Please include the questions in your slides. Don’t write detailed answers in the slides and read them to the class. Instead, use bullet points, graphs, or animations to explain your answers to the class.

In your Q&A report, use text to more thoroughly answer the questions. Include a short paragraph at the beginning of the report to summarize the paper.

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. 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?

3. “SILT uses a memory-efficient, high-performance hash table based upon cuckoo hashing.” Explain what the cuckoo hashing is and why it is used.

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 for cuckoo hashing.

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 HashStore 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?