Implementing an LSM-Based Key-Value Store

Background

Log-structured merge-trees (LSM-trees) [1, 2, 3] are one of the most commonly used data structures for persistent storage of key-value entries. LSM-based storages are in use in several modern key-value stores including RocksDB at Meta, LevelDB and BigTable at Google, Cassandra and HBase at Apache, and so on. LSM-trees store data in the disk as immutable logs (also known as sorted sequence tables (SSTs)), which are maintained in hierarchical levels of increasing capacity. To bound the number of logs that a lookup has to probe, LSM-trees merge logs of similar sizes. The two possible merging strategies are (i) leveling (optimized for lookups) and (ii) tiering (optimized for updates).

Objective

The objective of the project is to implement an LSM-tree using only a single process thread. Review the LSM-tree literature to understand the principles and operations supported in an LSM-tree. Implement a vanilla LSM-tree for both merging strategies – leveling and tiering.

Language of Implementation. It is strongly suggested that you use C/C++ to do your implementation if you are familiar with the languages. If you are unfamiliar with C/C++, but want to learn it while working on the project, you can find some under **Useful Resources** in the projects page. If neither of the above is suitable for you, you may implement the project in Java.

Workflow. If you are implementing the project in C/C++, click on this link. If you plan to implement in Java, click here. The general workflow for the project is as follows.

1. Once you clone the repository and navigate into it, you will find a basic implementation of TemplateDB, which includes the `DB`, `BloomFilter`, and `Operations` components. These will serve as the foundation for building an LSM-based key-value datastore. Your task is to build an LSM-based key-value datastore on top of this TemplateDB implementation. Before starting, ensure you thoroughly review the `README` file. You are free to modify certain components to fit your specific implementation approach.

2. For leveling, make sure your data at each level is stored and stored in multiple files to facilitate partial compactions. Similarly, for tiering, each tier should comprise of multiple files. Alternatively, you can implement compaction by maintaining a single sorted run (or one SST file) at each level for leveling, and during compaction, merge all data from level i with level $i + 1$. For tiering, store each tier as a single sorted run, and whenever level i accumulates T runs, sort-merge them and write a single run to the next level.
(note: Although we do not expect you to implement partial compaction in the leveled LSM-tree, you are welcome to do so.)

3. The creation of Bloom filters may vary depending on your implementation. If all data is kept in a single file at each level, only one Bloom filter per level may be enough. However, if the data is stored in multiple files, a Bloom filter will be required for each file to support partial compactions.

4. Fence pointers (or zone maps) are expected to be maintained at the granularity of pages.

5. Your LSM-tree implementation should support insert, update, and deletion of key along with point and range queries.
   (Note: The output of range queries must be sorted. You are not allowed to store intermediate results from all levels in a large vector and then sort them together. Instead, you need to implement a merge process for several iterators across different runs to avoid unnecessary space amplification.)

6. Make sure to report all necessary performance numbers after executing a workload.

Do NOT upload your code to public repositories, such as GitHub and Bitbucket.

References

[1] Chen Luo, Michael J. Carey. LSM-based storage techniques: a survey. VLDB J. 29(1): 393-418 (2020). DOI: https://doi.org/10.48550/arXiv.1812.07527.

[2] Niv Dayan, Manos Athanassoulis, Stratos Idreos. Monkey: Optimal Navigable Key-Value Store. SIGMOD Conference 2017: 79-94. DOI: https://doi.org/10.1145/3035918.3064054.

[3] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil. The Log-Structured Merge-Tree (LSM-Tree). Acta Inf. 33(4): 351-385 (1996).
DOI: https://doi.org/10.1007/s002360050048.