Related Work

Related studies of LSM-tree optimization include read amplification optimization and write amplification. We will introduce the two types of studies respectively in this section.

Read amplification optimization: Bloom Filter is a common optimization method. By using Bloom filters in point queries, layers that do not contain data can be skipped, greatly improving the performance of point queries. Wu et al. [1] set up Bloom filters in groups, used the hash value of KV data keywords as prefixes, and put data with the same prefixes into the same group to improve the performance of point queries on LSM trees, but this storage structure partitions the keywords with hash values and cannot support range queries. On the other hand, Amur et al. [2] exploit the balancing ability of B+ trees and efficient indexing ability to improve read performance and reduce read amplification. However, hash-based grouping likewise leads to poor range query performance, while the presence of indexes also makes the structure maintenance overhead high. Zhong et al. [3] propose an optimization method REMIX for range queries, where the authors construct indexes based on the invariance of the SSTable before merging and using the ordered views generated during the query, which effectively improves the query performance. However, the process of data writing involves the maintenance of indexes, which incurs additional disk read overhead.

Write amplification optimization: For the write amplification problem caused by merging, Yao et al. [4] propose the lightweight merging structure LWC-tree, which simplifies the traditional reading, sorting, and rewriting process and improves the merging efficiency, but Bloom filters and sparse indexes take up more memory space. Alternatively, the KV separation technique represented by WiscKey [5] proposes to store keywords and values separately. The merging process only needs to perform merging and rewriting of the keyword file, which greatly reduces the amount of data read and written during the merging process and improves the read and write performance. However, it requires storing multiple versions of the value data and thus has a high rubbish collection overhead. Also, Huang et al. [6] propose to isolate key-value items that bring a disruptive effect on the LSM-tree by using a multi-level log structure. It avoids the write amplification caused by updates. KV separation technique can effectively reduce the read and write overhead of the merging process, but there is the problem of difficulty in recovering the data copies of the value files, and the sequential reads on the keyword files will most likely change to random reads on the value files, which This affects the ability of range queries.

[1]. Wu X, Xu Y, Shao Z, and Jiang S. LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items. In: Proceedings of the 2015 USENIX Annual Technical Conference. 2015, 71-82

[2]. Amur H, Andersen D, Kaminsky M, and Schwan K. Design of a Write-Optimized Data Store. Georgia Tech Digital Repository, 2013

[3]. Zhong W, Chen C, Wu X, and Jiang S. REMIX: efficient range query for LSM-trees. In Proceedings of the 19th USENIX Conference on File and Storage Technologies. 2021, 51–64

[4]. Yao T, Wan J, Huang P, He X, Wu F, and Xie C. Building Efficient Key-Value Stores via a Lightweight Compaction Tree. ACM Transactions on Storage, 2017, 13(4): 1–28

[5]. Lu L, Pillai T, Gopalakrishnan H, and Arpaci-Dusseau A, Remzi H. Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-Conscious Storage. ACM Trans. Storage, 2017, 13(1), 1–28

[6]. Huang K, Jia Z, Shen Z, Shao Z, and Chen F. Less is More: De-amplifying I/Os for Key-value Stores with a Log-assisted LSM-tree. In: Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE). 2021, 612-623