Student ID: 1505022

Modern key-value stores use LSM-based storage, where unlike traditional index structures, these do not perform in-place updates. Rather, LSM tree first buffers all writes in main memory, and subsequently flushes the buffer as sorted run to disk whenever it fills up, and organizing the disk-runs into a number of levels of increasing sizes. LSM-tree later sort-merges these runs. This design has lots of benefits including superior write performance, high space utilization, tunability, and simplification of concurrency control and recovery. Two well known optimizations used by most LSM-trees include Bloom filters (a probabilistic data-structure designed to improve point lookup costs) and Partitioning. Recent LSM-trees typically organize their memory components using skip-list or B+ tree and organize their disk components using B+ trees or sorted-string tables (SSTables).

In this paper, the authors survey the recent research efforts on improving LSM-trees. The six major issues discussed by the authors are as follows:

Write Amplification, where it impacts both the write performance of LSM-tree and also the lifespan of SSDs due to frequent disk writes. Some possible improvements provided in the paper include Tiering, which has lower write amplification than levelling. Authors describe four structures which are variants of partitioned tiering with vertical grouping such as: WriteBuffer(WB) which uses hash-partitioning to achieve workload balance , light-weight compaction tree (LWC-tree) where if a SSTable group contains too many entries, it will shrink the key range after it is merged, PebblesDB (built using Fragmented Log-Structured Merge Tree, uses the idea of guards) and dCompaction which uses the idea of virtual SSTabls and virtual merges to reduce merge frequency.