Key-Value Stores Today Are Suboptimal for Dynamic Workloads

NoSQL key-value stores are widely popular today. LSM-Tree and B-Tree data structures commonly back key-value stores. An optimal data structure design is determined by the specific workload distribution. Key-value stores optimized for one fixed workload are suboptimal for dynamic modern applications.

The Solution: Transitions

Repeatedly remove the k blocks of entries with lowest keys from the LSM-Tree and append them to the end of the B-Tree leaf level. Convert the lowest level of the LSM-Tree into a B-Tree, avoiding disk IO. Then repeatedly insert batches of k blocks of entries with the lowest keys into the B-Tree.

Cost Model to Choose the Optimal LSM-Tree to B-Tree Transition

We compare the IO costs of the two transition approaches described above.

Sort-Merge Cost = \( \sum_{i=1}^{\text{num levels}} \frac{\text{bytes in lowest level}}{\text{page size}} \cdot \left( \frac{1 + \text{write cost}}{\text{read cost}} \right) \)

Batch-Insert Cost = \( \frac{\text{bytes in lowest level}}{\text{page size}} + \sum_{i=1}^{\text{num levels}} \frac{\text{bytes in th level}}{\text{page size}} \cdot \frac{\text{write cost}}{\text{read cost}} \)

Denoting \( \phi \) as the ratio of IO write to read cost, \( d \) as the entry size in bytes, and \( p \) as the number of entries per page, we find an elegant condition for when we ought to prefer the batch-insert algorithm over the sort-merge algorithm.

Gradual Transitions Enable Low Overhead

When transitioning from an LSM-Tree to a B-Tree, the transition cost can be amortized over an arbitrary number of steps. We maintain a hybrid key-value store to handle queries while the transition is in progress.

Transitioning Outperforms B-Trees and LSM Trees

Our implementation of this hybrid data structure proves that transitioning databases can provide superior query performance on dynamic workloads than classic LSM or B-Trees can.