Exploring Data Deduplication in LSM Tree-based Key-Value Stores

Safdar Jamil\textsuperscript{1}, Awais Khan\textsuperscript{2}, Youngjae Kim\textsuperscript{1}
\textsuperscript{1}Sogang University, Seoul, South Korea, \textsuperscript{2}Oak Ridge National Lab, TN, USA

1 Introduction

LSM-tree is used as a write-optimized index structure to handle bursty inserts in key-value (KV) stores [1]. It organizes KV pairs in hierarchical levels of increasing size. Compaction between these levels in LSM-tree generates large I/Os resulting in high write and space amplification. Wiseckey [9] employed KV separation to minimize the write amplification (WA) in LSM-tree. However, it suffers from space amplification (SA) and depends highly on garbage collection, which interferes with the foreground I/Os. On the other hand, deduplication (dedup) can eliminate duplicate data, reducing the size of the primary storage system [3–8, 10–12].

2 Problems with MEMDEDUP

The naive approach is to incorporate inline dedup at the memtable (MT) of the LSM-tree, MEMDEDUP. Figure 1 shows the design and operational flow (in red) of the MEMDEDUP. The dedup layer performs the dedup operations, such as chunking, fingerprinting, duplicate detection and metadata update. MEMDEDUP intercepts the put operation and performs dedup operations before inserting the KV pairs to the MT. MEMDEDUP performs fixed-size chunking on the value of KV pair and maintains dedup metadata based on it. The metadata includes Chunk Information Table (CIT), which manages the dedup meta-data, such as the fingerprint of the value (FP(V)), list of parent keys (PK[]) referring to the value, reference count (RC) and value offset. Every Get/Compaction I/O touches CIT to access the KV pairs. MEMDEDUP has two major challenges: First, there is a high performance overhead due to fingerprinting of the values and frequent dedup metadata updates/traversals. Every access to KV pair needs to touch the CIT. For instance, the compaction process, which is based on keys, has to first traverse the CIT (linearly due to PK[]) and then update it based on the modifications. Second, the in-place update support in MT leads to inconsistency issues.

3 DELTAKV: Proposed Design

DELTAKV delays the dedup to be performed at the FLUSH operation, when IMTs are flushed to the SSTs, which is performed by the background threads. The advantages of DELTAKV over MEMDEDUP are two folds: (i) removes dedup from the critical I/O path and perform dedup in the background where it does not interfere with foreground I/Os, and (ii) by delaying dedup, it eliminates the consistency issue of the in-place update operations as MT serves the incoming requests and becomes immutable once it meets the threshold size. DELTAKV performs dedup on the immutable data which is not updated directly hence mitigate the consistency issues. For further optimization, dedup metadata is divided into two different data structures, value information table (VIT) and B+tree, as shown in Figure 1. The operation flow (in blue) of DELTAKV is shown in Figure 1. At flush operation, firstly, each KV pair goes through the dedup layer where it performs the dedup operation by referring to the VIT (comprised of the FP(V) and RC). If a unique value is encountered, it is stored in the SSTs and the dedup metadata is updated. For unique value, a new entry is created in the VIT while the PK and offset are stored in the B+-tree. However, when a duplicate value is detected in VIT, the RC of that value is incremented while in the B+-tree, a new entry of PK and the offset are updated. Meanwhile, Get I/O traverses the B+-tree, whereas the compaction process accesses/updates both data structures in parallel.

4 Preliminary Evaluation

We implement MEMDEDUP and preliminary version of DELTAKV atop Facebook’s RocksDB [1] and evaluate on Intel Xeon CPU with Samsung 970 EVO SSD with YCSB benchmark [2]. DELTAKV updates the VIT for KV pairs without duplicate detection and does not include B+-tree in this evaluation. We run two different workloads of YCSB with three different sizes based on the KV pairs as shown in Figure 2. Evaluation shows that DELTAKV maintains the performance as it does not perform dedup in the critical I/O path of the LSM-tree. However, a slight performance drop with bigger workload is observed in DELTAKV which is attributed to fingerprint computation.
References


