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

Safdar Jamil[1], Awais Khan[2], Youngjae Kim[1]
[1]*Sogang University, Seoul, South Korea,* [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. Wisckey [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 IOs. On the other hand, deduplication (dedup) can eliminate duplicate data, reducing the size of the primary storage system [3–8, 10–12]. In this work, we present DELTAKV which incorporates the dedup at the *flush* operation, when Immutable Memtables (IMTs) are flushed to Sorted String Tables (SSTs). Dedup-based LSM-tree will reduce the overall amount of data, hence, resulting in less I/Os during compaction operations. We argue that incorporating dedup in LSM-tree design mitigates the WA and SA problems without significant performance loss.

## 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 perform dedup operations before inserting the KV pairs to the MT. MEMDEDUP performs fixed-size chunking on the value of KV pair and maintain dedup metadata based on it. The metadata includes Chunk Information Table (CIT), which manages the dedup metadata, 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. It happens when an existing KV pair with RC > 1 updates its value in MT, making other keys inconsistent, i.e., keys in the PK[] list of the CIT points to the old missing value which is replaced due to in-place update. It is possible to handle such inconsistency via out-of-place update in MT but it complicates KV pair reconstruction using CIT in MEMDEDUP.

## 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 IO path and perform dedup in the background where it does not interfere with foreground IOs, 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. Meantime, Get IO traverses the B$^+$-tree, whereas the compaction process accesses/updates both data structures in parallel.

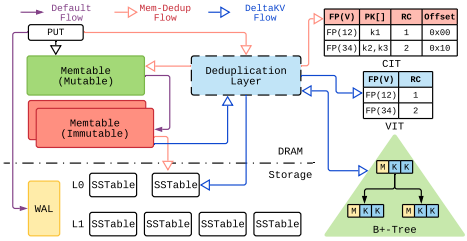


Figure 1: Design overview of MEMDEDUP and DeltaKV

## 4 Preliminary Evaluation

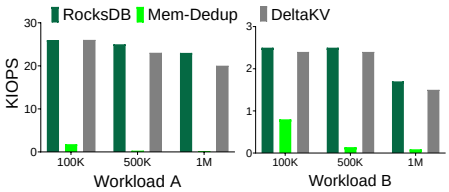


Figure 2: Performance of MEMDEDUP, DELTAKV and RocksDB. Workload A (100%:W) and Workload B (50%:W & 50%:R)

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.

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP), Korea government (MSIT) (Development of low-latency storage module for I/O intensive edge data processing) under Grant 2020-0-00104. This work was also supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).

## References


[1] Rocksdb. http://rocksdb.org. Accessed: 2022-01-16.

[2] Yahoo cloud serving benchmark. https://github.com/brianfrankcooper/YCSB/. Accessed: 2022-01-21.

[3] DedupSearch: Two-Phase deduplication aware keyword search. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA, February 2022. USENIX Association.

[4] DeepSketch: A new machine Learning-Based reference search technique for Post-Deduplication delta compression. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA, February 2022. USENIX Association.

[5] DUPEFS: Leaking data over the network with filesystem deduplication side channels. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA, February 2022. USENIX Association.

[6] Zhichao Cao, Shiyong Liu, Fenggang Wu, Guohua Wang, Bingzhe Li, and David H.C. Du. Sliding Look-Back window assisted data chunk rewriting for improving deduplication restore performance. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 129–142, Boston, MA, February 2019. USENIX Association.

[7] P. Hamandawana, A. Khan, J. Kim, and T. Chung. Accelerating ml/dl applications with hierarchical caching on deduplication storage clusters. *IEEE Transactions on Big Data*, (01):1–1, aug 5555.

[8] Awais Khan, Chang-Gyu Lee, Prince Hamandawana, Sungyong Park, and Youngjae Kim. A robust fault-tolerant and scalable cluster-wide deduplication for shared-nothing storage systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 87–93, 2018.

[9] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Hariharan Gopalakrishnan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Wisckey: Separating keys from values in ssd-conscious storage. *ACM Trans. Storage*, 13(1), March 2017.

[10] João Paulo and José Pereira. A survey and classification of storage deduplication systems. *ACM Comput. Surv.*, 47(1), June 2014.

[11] Yucheng Zhang, Wen Xia, Dan Feng, Hong Jiang, Yu Hua, and Qiang Wang. Finesse: Fine-Grained feature locality based fast resemblance detection for Post-Deduplication delta compression. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 121–128, Boston, MA, February 2019. USENIX Association.

[12] Xiangyu Zou, Jingsong Yuan, Philip Shilane, Wen Xia, Haijun Zhang, and Xuan Wang. The dilemma between deduplication and locality: Can both be achieved? In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 171–185. USENIX Association, February 2021.