Acheron: Persisting Tombstones in LSM Engines

Zichen Zhu
Boston University
zczhu@bu.edu

Subhadeep Sarkar
Boston University
ssarkar1@bu.edu

Manos Athanassoulis
Boston University
mathan@bu.edu

ABSTRACT

Modern NoSQL storage engines frequently employ Log-structured merge (LSM) trees as their core data structures because they offer high ingestion rate and low latency for query processing. Client writes are captured in memory first and are gradually merged on disk in a level-wise manner. While this *out-of-place* paradigm sustains fast ingestion rate, it implements delete operations via inserting *tombstones* which *logically invalidate older entries*. Thus obsolete data cannot be removed instantly and may be retained in LSM trees for arbitrarily long time. Therefore, out-of-place deletion in LSM trees may on the one hand, violate data privacy regulations (e.g., *the right to be forgotten* in EU's GDPR, *right to delete* in California's CCPA and CPRA), and on the other, it hurts performance.

In this paper, we develop `Acheron`, which demonstrates the performance implications of out-of-place deletes and how our method achieves timely persistent deletes. We integrate both prior state-of-the-art compaction policies and our recently presented method, FADE, into `Acheron` and visualize the life cycle of tombstones in LSM trees. Using `Acheron` visualization, the users can observe that the state of the art does not provide guarantees on when obsolete entries can be physically removed, and also observe that FADE can achieve timely persistent deletes without full tree compaction. The users can further customize the workload, LSM tuning knobs, and disk parameters to investigate the impact of different factors on tombstones and the performance. This demonstration provides key insights into the impact of tombstones for LSM-interested researchers and practitioners.

1 INTRODUCTION

LSM-based Key-Value Stores. Data-intensive applications (e.g., Internet-of-things, edge computing, 5G communications, and autonomous vehicles) generate a huge amount of data at unprecedented rates. Several modern database systems rely on LSM-tree-based key-value stores to sustain efficient ingestion for OLTP workloads (e.g., BigTable, MyRocks, and CockroachDB). In LSM trees, incoming entries are batched in an in-memory write buffer, and once the buffer is full, it is flushed to storage as an immutable sorted run. When the number of accumulated similar-sized runs exceeds a pre-defined threshold, they are merge-sorted to form a larger immutable run (this process is also termed *compaction* [3]).

SIGMOD '23, June 18-23,2023, Seattle, WA, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

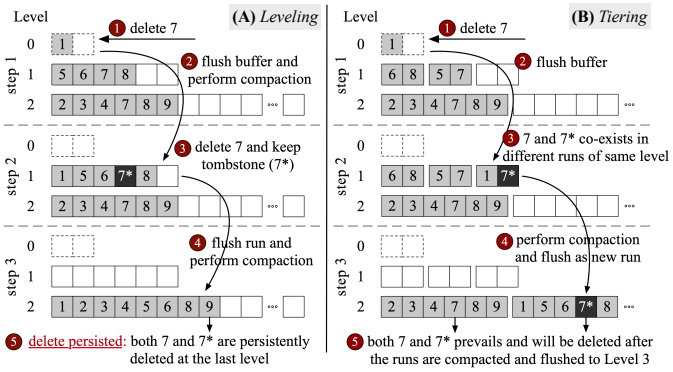

Figure 1: In an LSM-tree, for every tombstone, there can be (A) one matching entry per level for leveling or (B) one matching entry per tier per level (T per level) for tiering, where $T = 3$ in this example.

The immutability of sorted runs does not allow for in-place updates/deletes, rather all updates lead to sequential writes during flushing and compaction.

Deletes in LSM Trees. However, the immutability of sorted runs comes at the cost of *out-of-place* deletes. Every delete operation is implemented by inserting a *tombstone*, as shown in Figure 1. Within the memory buffer, a tombstone eagerly deletes any older matching entries, and it is maintained to invalidate further matching entries in the tree. When the memory buffer is full and it is being flushed as a file on disk, its tombstones are tagged with a time stamp. As more data is ingested, this file will be involved in compactions, during which older entries with the same key will be *physically* discarded.

Problems. When a tombstone is inserted, older entries may exist in deeper levels, and thus, we can only safely remove the tombstone when it reaches the last level of the LSM tree through iterative compactions. Tombstones and obsolete entries may co-exist for an arbitrarily long time until all tombstones are compacted to the last level, which has the following implications:

(a) Tombstones and obsolete entries increase *space amplification*.
(b) Before obsolete entries are physically discarded, they are likely to be involved in other compactions, which may lead to to increased *write amplification*.
(c) Invalid entries may pollute the indexes and filters [1], and thus hurt range and point query performance.
(d) Out-of-place deletion does not guarantee on older records are physically removed, which may *violate data privacy regulations* [2] such as *the right to be forgotten* in EU's GDPR, *right to delete* in California's CCPA and CPRA.

Our approach. The approach that we demonstrate [2] proposes a new compaction policy that prioritizes files containing older tombstones. Compared to a full tree compaction, this approach retains LSM tree's benefit of an amortized merging cost with no latency spikes and has less write amplification and shorter write stalls. Specifically, compactions are more eagerly triggered based on the maximum age of tombstones for each level, so that tombstones can reach the last level within a user-defined threshold.

In this paper, we develop Acheron[1], which demonstrates the performance implications of tombstones, and how our method achieves timely persistent deletes. To our knowledge, there are no previous systems or tools that reveal tombstones' impact on new data privacy regulations and system performance in LSM trees. LSM-interested researchers and practitioners can benefit from Acheron to gain more useful insights into this impact.

Demonstration. Conference participants can interact with Acheron to compare and analyze the life cycle of tombstones in LSM trees under different scenarios. From the visual interface, participants can see (i) when tombstones move from shallower levels to deeper levels and are physically deleted in the last level (ii) how our method purges obsolete data compared to other compaction policies, and (iii) how tombstones affect the system performance. As a remark, Acheron also offers the opportunity for participants to vary the workload composition and LSM tunings (e.g., delete percentage, size ratio, storage read/write speed) to explore more scenarios. The demo is available at https://disc-projects.bu.edu/acheron/research.html, including a short video on how to use it[2].

2 ENABLING TIMELY PERSISTENT DELETES

To enable timely persistent deletes, in prior work, we introduced FADE, a new family of delete-aware compaction policies. FADE (short for *FAst DEletes*) piggybacks the task of timely delete persistence to the LSM-tree's compaction routine while retaining the LSM tree's benefit of amortized merging cost and predictable performance. Acheron taps into the key design elements of FADE and presents an interactive framework highlighting how one can navigate the performance-privacy trade-off given a threshold time for delete persistence and a target performance. In this section, we describe the technical details of FADE and present the performance metrics we use in Acheron.

2.1 FADE

Delete Persistence Threshold (DPT). FADE ensures that all the tombstones are persisted within a user-specified delete persistence threshold (DPT). Delete Persistence Threshold (DPT) is formally defined as, the worst-case time required, following the insertion of a tombstone, the tree is void of any entry (including tombstones) with a matching (older) key to that of the inserted tombstone. Typically, DPT is specified as part of the service level agreement (SLA) concerning the data retention. With DPT and other basic LSM tuning-knobs as inputs, the complete algorithm for FADE is presented in Algorithm 1. Compared to the state-of-the-art LSM compaction policy, FADE augments it in the following two aspects.

[1] *Acheron* is the Greek mythological river used to transport souls to the underworld (persist tombstones).
[2] Video: https://disc-projects.bu.edu/acheron/files/Demo.mp4

Algorithm 1: FADE

Input : delete persistence threshold (DPT); levels in tree (L_{old}); size ratio (T); size of memory buffer (M)

```
FADE():
begin
    L_new = getCurrentTreeLevel()
    d_0 = 0
    if L_new > L_old then
        for i ∈ [1 : L_new] do
            d_i = d_{i-1} + DPT·(T − 1)/(T^L − 1) · T^{i−1}
    for i ∈ [1 : L_new] do
        csize(i) = 0, ttl_i = 0, cap_i = M · T^i
        for j ∈ [1 : getFileCountInLevel(i)] do
            csize(i) += size(j)
            if d_i < age_j then
                ttl_i++
        score[i] = csize(i)/cap_i + ttl_i
    compact_level = getLevelToCompact(score[])
    compact_file = getFileToCompact(compact_level)
    initiate compaction with compact_file
getLevelToCompact(score[])
begin
    c_level = score[0]
    for i ∈ [1 : L_new] do
        if score[i] > score[i − 1] then
            c_level = i
    return c_level
getFileToCompact(compact_level)
begin
    files = getFilesInLevel(compact_level)
    for i ∈ [1 : files.size() − 1] do
        if d_{compact_level} ≥ files[i].age then
            return files[i]
    sort files by overlapping ratio in an ascending way
    return files[0]
```

Compaction Trigger. In FADE, a compaction is triggered not only when a level is saturated, but also when there exists at least one file of which the oldest tombstone expires. To support the additional trigger, FADE maintains the age of oldest tombstones per file and assigns every file a *time-to-live* (TTL). If TTL is fixed as DPT for all the files, all the expired tombstones/files in shallow levels can result in cascading compactions and thus write amplification. To avoid cascading compactions, FADE assigns a smaller TTL, d_i, for every file in level i such that $\sum_{i=1}^{L-1} d_i = \text{DPT}$, where L is the number of levels in the current LSM tree. The allocation strategy for d_i proposed by FADE follows a geometric sequence with increasing ratio T (the same as the size ratio), because the exponential assignment is coherent with the capacity in each level and thus leads to fewer concurrent compactions.

File Picking Policy. State-of-the-art LSM-engines pick the file with the smallest overlap to reduce the write amplification (while there are many other picking policies [3], this is the most common one and thus selected to compare with Lethe in Acheron). However, in FADE, files with expired TTL have to be prioritized over files with least overlapping rate to enforce the timely physical deletion. In other words, compactions triggered by TTL expiration only pick expired files to compact while compactions that are only triggered by saturation still picks the file with minimum overlapping ratio. If

there is a tie in terms of expired TTLs, minimum overlapping ratio policy will take in action again.

2.2 Demonstration Metrics

Lethe ensures that all the tombstones are persisted by the time their lifetime reaches DPT by triggering more compactions, which naturally introduces a trade-off between tombstones and compactions. In other words, more compactions lead to fewer tombstones, and vice versa. To better capture this trade-off, Acheron benchmarks the following metrics during the emulation.

Tombstone-related Metrics. After every flush and every compaction, Acheron records *the number of deletes*, *the number of existing tombstones*, *the number of expired tombstones* (with respect to the specified DPT), and *the maximum age of existing tombstones*.

Compaction Metrics. Compaction metrics include *the number of compactions*, *the average compaction (input) size*, *the average compaction latency*, and *the worst-case compaction latency*. These metrics are updated every time compaction finishes.

Performance Metrics. Many performance metrics can be also affected by extra tombstones and compactions. For example, fewer tombstones indicate less disk space, smaller Bloom Filters (BFs), and indexes required for the LSM tree, and thus Acheron also records *storage space* and *memory footprint* for auxiliary structures. On the other hand, the average compaction latency can significantly affect *the average ingest cost* and *write amplification*, which are also tracked in Acheron.

3 THE ACHERON DEMONSTRATION

Overview. The overall structure of Acheron UI is summarized in Figure 2, which consists of an input panel, a progress-bar control panel, an emulation panel, and a performance panel. Users interact with Acheron through the following operation flow: (i) specify the configuration in the input panel, (ii) click "Play" button in the control panel (after which Acheron starts emulating the ingestion process), (iii) watch the animation of how tombstones propagate through iterative compactions in the emulation panel, (iv) compare the metrics that are presented in the performance panel.

Input and Control Panels. Users can customize the workload by specifying *the number of total ingests*, *the key size*, *the entry size*, and *the percentage of deletes* **(1)**. Instead of generating the real workload, Acheron emulates it through scaling down the number of ingests. When estimating the performance metrics, Acheron takes into account the scaling factor to approximate them using smaller dataset. In addition to the workload specification, Acheron allows users to configure the main memory parameters (i.e., *the memory size for MemTable*, *the bits per key for BFs*) **(2)**, *the size ratio* **(3))**, disk parameters (i.e., *read latency per I/O*, *write latency per I/O*) **(4)**, and *persistence threshold (DPT)* **(5)**. After setting the input, users can control the emulation progress via buttons "Play", "Pause", "Finish", and even dragging the progress bar in the control panel.

Emulation and Performance Panels. The emulation panel shows iterative compactions in three LSM trees with different compaction policies – i.e., FADE **(7)**, MinOverlappingRatio **(8)**, and Round-Robin **(9)**. To better depict the difference in compaction policies, Acheron shows the compaction progress in detail, as shown in Figure 3. In the visualization of LSM trees, each rectangle represents a file (where the length is proportional to the file size). Further, the length of the gray/white striped part in a rectangle represents the proportion of tombstones in this file. The darkness of the striped part indicates the maximum tombstone age and when the oldest tombstone expires, the color turns completely dark. The darkness of the blue part indicates the overlapping ratio of this file. The structure of LSM-tree is updated every flush and every compaction. For illustration, every flush is forced to be synced across different policies but the actual time could differ. When watching the animated compaction process, users can also observe that the performance metrics and plots (the x-axis represents the flushed data in total) are updated on the fly **(10)**. If users want to investigate the impact of different DPT in FADE, they can also switch to the "DPT Analysis in FADE" to compare Lethe with three different DPTs **(11)**.

4 DEMONSTRATION SCENARIOS

The participants can fully interact with Acheron to understand the life cycle of tombstones, how the life cycle differs across different compaction policies, and also the impact of DPT along with twelve system metrics. We use the following three example scenarios to demonstrate possible interactions between users and Acheron.

Scenario 1: Visualize the Life Cycle of Tombstones Through Iterative Compactions. We consider the default setting: $10M$ 128-byte entries with 25% deletes are ingested into an LSM tree with $16MB$ buffer size, 10 bits per key for BFs, 50-second DPT, and the size ratio as 2. After users click the "Play" button, they can see the whole tree construction progress, which includes how tombstones moves from shallower levels to deeper levels, how tombstones age (the color gradually changes from white to gray), and persist (no gray strips/tombstones in the last level).

Scenario 2: Compare Compaction Policies for Deletes. When comparing different compaction policies, users can clearly see some tombstones expire (gray strips are replaced with pure black) under MinOverlappingRatio and Round-Robin policy. In contrast, pure black tombstones never appear in FADE since DPT is applied as a hard constraint for tombstones' TTL. Besides, users can also observe the extra compaction cost from Lethe through those compaction metrics since Lethe achieves timely persistence by prioritizing expired tombstones.

Scenario 3: Examine the Trade-Off Between Delete Timeliness and Write Amplification. Users can also switch to "DPT Analysis" mode to explore how different DPT affects performance. The color of tombstones with lower DPT is darker since the maximum obsolete age ratio becomes small. Meanwhile, users can observe from the performance panel that extra compactions are triggered with lower DPT, and thus the amortized ingest cost of lower DPT is also higher. On the other hand, large DPT leads to high space amplification as more tombstones can retain in LSM trees.

5 CONCLUSION

In this paper, we introduce Acheron, which visualizes the life cycle of LSM trees under different settings. Acheron can help users to understand why tombstones can stay for an arbitrarily long time under state-of-the-art compaction policy and how FADE achieves timely persistent deletes without full tree compaction. Acheron

Figure 2: The Acheron UI allows the participants to visualize the life cycle of tombstones under different compaction policies.

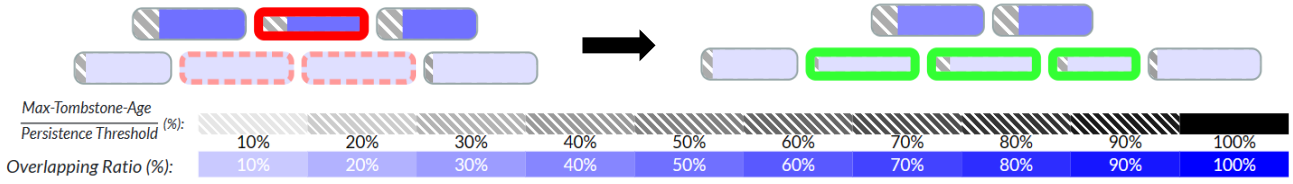

Figure 3: Acheron UI and visualizations. Each oblong represents a file. During a compaction, the file with red solid border indicates that this file is being picked to compact to the next level and the files with red dashed border means that they overlap with the selected file. After the compaction finishes, there will be newly generated files, marked by green solid border.

also allows users to explore the impact of DPT on tombstones and performance.

REFERENCES

[1] Gui Huang, Xuntao Cheng, Jianying Wang, Yujie Wang, Dengcheng He, Tieying Zhang, Feifei Li, Sheng Wang, Wei Cao, and Qiang Li. 2019. X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 651–665. https://doi.org/10.1145/3299869.3314041

[2] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. 2020. Lethe: A Tunable Delete-Aware LSM Engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 893–908. https://doi.org/10.1145/3318464.3389757

[3] Subhadeep Sarkar, Dimitris Staratzis, Zichen Zhu, and Manos Athanassoulis. 2021. Constructing and Analyzing the LSM Compaction Design Space. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2216–2229. http://vldb.org/pvldb/vol14/p2216-sarkar.pdf