Compactionary: A Dictionary for LSM Compactions

Subhadeep Sarkar
Boston University, USA
ssarkar1@bu.edu

Kaijie Chen
Boston University, USA
kaijiec@bu.edu

Zichen Zhu
Boston University, USA
zczhu@bu.edu

Manos Athanassoulis
Boston University, USA
mathan@bu.edu

ABSTRACT

Log-structured merge (LSM) trees are widely used as the storage layer of modern NoSQL data stores, as they offer efficient ingestion performance. To enable competitive read performance and reduce space amplification, LSM-trees re-organize data layout on disk iteratively, through *compactions*. Compactions are at the heart of every LSM-based storage engine, fundamentally influencing their performance. However, the process of compaction in LSM-engines is often treated as a black-box that is rarely exposed as a tuning knob. In this paper, we demonstrate `Compactionary`, a dictionary for LSM compactions, that helps to visualize the implications of compactions on performance for different workloads and LSM tunings.

`Compactionary` breaks down the LSM compaction black-box, expressing compactions as an ensemble of four first-order design choices: (i) *when* to compact, (ii) *how to organize* the data after compaction, (iii) *how much* data to compact, and (iv) *which* data to compact. We configure `Compactionary` to demonstrate the operational flow of several state-of-the-art LSM compaction strategies and how each strategy affects performance. The participants can (i) customize the workload, (ii) configure the LSM tuning, and (iii) switch between advanced compaction options, to understand individually the impact of the different factors on performance. Further, to engage the interested participants, we extend the demonstration by allowing the participants *to create custom hybrid compaction strategies*, as well as *to configure the settings separately for each strategy* in an individual analysis phase. The demo is available at https://disc-projects.bu.edu/compactionary/#interactiveDemo.

CCS CONCEPTS

• **Information systems** → **Data layout**; **Data access methods**; **Data structures**; **Key-value stores**.

KEYWORDS

Compaction, LSM-tree, NoSQL, Storage engine, Data layout

ACM Reference Format:
Subhadeep Sarkar, Kaijie Chen, Zichen Zhu, and Manos Athanassoulis. 2022. `Compactionary`: A Dictionary for LSM Compactions. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3514221.3520169

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3520169

1 INTRODUCTION

Background. Log-structured merge (LSM) trees are widely used as the storage layer of several modern NoSQL, relational, and array-based data stores [2, 3]. LSM-based data stores offer high throughput for ingestion by batching writes in memory and good utilization of disk space by using immutable file structures to store data on disk [1]. To facilitate efficient query processing, LSM-trees periodically re-organize the data layout on disk, through the process of *compactions* [4, 5]. Compactions affect the performance of LSM-engines along several axes, including write throughput, point and range query performance, and space and write amplification [6].

Problem and Motivation. Despite compactions being critical to LSM performance, the process of choosing an appropriate compaction strategy requires a human in the loop. In practice, decisions on *how to (re-)organize data on disk*, and thereby, *which compaction strategies to use* are often subject to the expertise of the engineers or the database administrators. This is largely due to two reasons. (A) The process of compaction in LSM-trees is often treated as a black-box and is rarely exposed as a tunable knob. While the LSM compaction design space is vast, the lack of a formal template for compactions leads to heavy reliance on individual expertise, and leaves a large part of the design space unexplored. (B) There is a lack of analytical and experimental data on how compactions influence the performance of LSM engines subject to different workloads and LSM tunings. Relying on human expertise to hand-pick the appropriate compaction strategies for each application does not scale, especially for large-scale system deployments.

Approach. The research that led to this demo paper, pursues the fundamental theory of decomposing LSM compaction strategies into first-order primitives, and thereafter, analyzing the performance implications of the compaction primitives subject to different workload characteristics and LSM tunings [6]. Specifically, our prior work formally constructs the design space of LSM compactions and provides key insights on how compactions are central to the overall performance of LSM engines.

Toward this, we introduce `Compactionary`, a demonstration framework that showcases side-by-side (A) the operational flow of different state-of-the-art compaction strategies and (B) how the different compaction strategies affect performance. The participants can (i) change the workload composition (such as, entry size, ingestion count, and filter memory size), (ii) configure the LSM tuning (such as, size ratio of a tree, buffer size, and page size), and (iii) switch between advanced compaction options (such as, enable/disable background compactions and threshold for background compactions) to understand *the physical interpretation of the several degrees of freedom embedded with the LSM compaction logic*. Further, an extended demonstration panel allows the participants to individually configure the settings for different compaction strategies enabling a low-level analysis of the strategies.

Lastly, Compactionary also allows the interested participants to create their own custom compaction strategies with hybrid data layout and compaction granularity, and explore parts of the uncharted design space of LSM compactions. The demo is available at https://disc-projects.bu.edu/compactionary/#interactiveDemo.

Goal. The goal of this demonstration is to visualize LSM compactions to understand when and how LSM engines perform compactions, and the impact of compactions along 14 performance metrics. Toward this, we break down the compaction black-box into four first-order design primitives and discuss how any state-of-the-art and even new compaction strategies can be expressed as an ensemble of the primitives. Compactionary provides the participants with the necessary infrastructure to compare different LSM compaction strategies, understand the impact of each compaction primitive, and through this process, build useful insights on how to choose appropriate compaction strategies based on workloads, LSM tunings, and the target performance. To our knowledge, this is the first demonstration that reveals the internals of the LSM compaction black-box, and provides useful insights to the LSM-interested researches and practitioners.

2 LSM COMPACTIONS

Compactionary taps into the internals of state-of-the-art LSM compaction strategies and identifies the fundamental design choices made under the compaction black-box. Toward this, we present the four first-order design primitives for LSM compactions and present how combinations of the primitives help construct different compaction strategies, and thereby, the LSM compaction design space [6]. Compactionary exposes these primitives to the participants as configurable knobs and expresses through closed-form modeling how they affect the LSM performance.

2.1 Compaction Primitives

We define an LSM compaction strategy as *an ensemble of four design primitives representing the fundamental decisions about the physical data layout on disk and the data (re-)organization policies.*

1) **Compaction trigger** outlines the *events that can initiate compactions*. The most common compaction trigger is based on the *degree of saturation* of a level, which is defined as the ratio of the data size (in bytes) in a level to the theoretical capacity of that level (in bytes). Once the degree of saturation goes beyond a pre-defined threshold, one or more immutable files from Level i are marked for compaction. Other compaction triggers include the *staleness of a file*, the *tombstone-based time-to-live*, and *space* and *read amplification*.
2) **Data layout** determines the *number of sorted runs per disk level*. The data layout is commonly classified as *leveling* and *tiering*. With leveling, once a compaction is triggered in a level, the file(s) marked for compaction are merged with the overlapping file(s) from the next level. For tiering, each level may contain more than one sorted run with overlapping key domains. Once a compaction is triggered, all sorted runs in a level are merged together and the result is written to the next level as a new sorted run. A generalization of this idea allows each level to separately decide between leveling and tiering, generating a continuum of hybrid data layouts.
3) **Compaction granularity** determines the *amount of data moved and merged during a single compaction job*. One way to compact data is by merging and moving all data from a level to the next level, which is commonly referred to as *full compaction*. Alternatively, many production-scale leveled LSM-based engines employ *partial compaction*, where instead of moving a whole level, a smaller chunk of data participates in every compaction. The compaction granularity can be a single file or multiple files, depending on the system design and the workload. Tiered LSM designs typically compact data at the granularity of *sorted runs*, compacting only runs from the same level.
4) **Data movement policy** *determines which file(s) are to be chosen for compaction* in LSMs with partial compactions. A naïve way to choose file(s) is by using a round-robin policy. To optimize for lookups, many production data stores select the *coldest* file(s) in a level for compaction. Another common optimization goal is to minimize write amplification, where files with the least overlap with the target level are chosen for compaction. To reduce space amplification, some storage engines choose files with the highest number of tombstones and/or updates.

Together, these primitives determine *when* and *how* to re-organize the data layout on disk. Table 1 outlines the various options for each compaction primitive as either adopted in production-scale LSM engines or proposed in state-of-the-art LSM literature. The proposed primitives can capture any state-of-the-art LSM compaction strategy and can also synthesize new unexplored compaction strategies.

2.2 Compaction as an Ensemble of Primitives

Every compaction strategy takes one or more values for each of the four primitives. The trigger, granularity, and data movement policy are multi-valued primitives, whereas data layout is single-valued. For example, a **leveled** LSM-tree (*data layout*) may perform compaction at the *granularity* of a **file**, with compactions being *triggered* if a **level reaches its capacity**. Once triggered, the *data movement policy* chooses the **file with the least overlap with the parent level** for compaction. Similarly, a tiered LSM may initiate compactions when the number of sorted runs in a level reaches a threshold, and compact all runs in the level to the next level.

Two compaction strategies are considered different from each other if they differ in at least one of the four primitives. Compaction strategies differing only on one primitive, may have vastly different performance. Plugging in some typical values for the cardinality of the primitives, we estimate the cardinality of the compaction universe as $>10^4$, a vast yet largely unexplored design space. In Section 3, we outline the different compaction strategies demonstrated by Compactionary, as well as, how Compactionary allows the participants to create new custom compaction strategies.

2.3 Performance Modeling & Metrics

The various design choices of the four compaction primitives affect the overall performance of an LSM engine. Below, we present a brief account of the performance metrics which we focus on in the demonstration. In the web UI, hovering the mouse pointer of a metric name reveals its definition along with the formulae used to model it. Throughput for ingestion and lookups are modeled based

Primitives	Physical interpretation	Options
Trigger	**When** to re-organize the data layout?	Level saturation; #Sorted runs; File staleness; Space amplification
Data Layout	**How** to organize the data on device?	Leveling; Tiering; 1-Leveling; L-Leveling; Hybrid
Granularity	**When** to re-organize the data layout?	Level; Sorted run; Single sorted file; Multiple sorted files
Data movement policy	**Which** block of data to be moved?	Round-robin; Least overlapping (LO) parent; LO grandparent; Oldest; Coldest

Table 1: The compaction primitives, their physical interpretation, and the different values that they can take.

on the device bandwidth input and assuming complete utilization of the device bandwidth.

Meta-Metrics. Meta-metrics are parameters that are artifacts of the compaction strategies employed, which in turn, influence performance along several axes. Meta-metrics include the *number of levels* and *sorted runs* in a tree, *number of compactions* performed, and the *average amount of data moved* due to compactions.

Ingestion Metrics. Ingestion metrics include the ingestion performance in terms of *write amplification* and the *average worst-case ingestion throughput* and the compaction performance in terms of *average* and the *worst-case compaction latency*.

Read Performance. We model the read performance separately for *point lookups on existing* and *non-existing keys*, *short range lookups* (spanning at most two files per sorted run), *long range lookups* (with variable selectivity).

Storage Footprint. Finally, we measure the storage footprint in terms of *space amplification*, the total space occupied by the *data on disk*, and the *size of in-memory data structures*.

3 THE COMPACTIONARY UI

Overview. Figure 1 shows the structure of the `Compactionary` UI. The input panel allows the user to specify (i) the workload, (ii) the allocated main memory and (iii) disk parameters, and (iv) the data layout settings. Based on the input, the visualization panel of `Compactionary` demonstrates through vivid animations the step-wise re-organization of the disk data layouts via compactions. During the illustration process, the metrics in the next performance panel are updated on the fly, and they are also used to construct the performance plots in a secondary panel.

Description. The **input panel** allows the participants to construct the size of ingestion workload by specifying the entry size and the number of entries to be ingested **(1)**. The participants can determine the allocation of the main memory between the memory buffer and Bloom filters **(2)** and the page size and file size on disk **(3)**. They can also input the size ratio of the LSM-tree and the default data layout **(4)**. Finally, under *Advanced settings*, the users can also specify the selectivity for long range lookups, read/write bandwidth support for a device, and even the speed of animation **(5)**. After setting the input parameters, the participants can start, pause, or quick finish the emulation using the **control panel (6)**.

The first column in the **emulation panel** shows the formation of an LSM-based data store that adopts *full compaction* which performs compaction at the granularity of levels **(7)**. The second column demonstrates a *partial compaction* strategy where files are picked at random or based on a round-robin policy **(8)**. A *hybrid compaction* strategy with lazy leveling, i.e., leveling in the last level and tiering in the others, is demonstrated in the third column **(9)**. The final column allows the participants to create their own custom compaction strategies with tiering in the shallower i ($i \leq L$) levels and leveling in the rest **(10)**. To visualize compactions, a light-bulb is placed against each level of the tree. Every time a compaction is triggered in a level, the corresponding light-bulb can be observed to glow against that level. Further, for LSM variants that store data across multiple files within every sorted run, consecutive files are differentiated by two different shades of blue.

The **performance panel** enables comparative analysis for 14 performance metrics side-by-side for up to four compaction strategies at a time **(11)**. The participants can also visualize the key performance metrics in form of analytical time-series plots by clicking on the *Generate plots* button, which allows them to derive useful insights from the experimental outputs **(12)**.

Functionality. `Compactionary` allows the participants to compare the behavior of several compaction strategies side-by-side, while providing them the necessary infrastructure to change the workload and/or LSM tuning. In addition, `Compactionary` provides a customizable framework to the participants where they can synthesize new compaction strategies, as required and compare them against state-of-the-art LSM compaction routines.

The *Individual Analysis* panel allows the participants to have different settings for the compaction strategies, and yet, compare them alongside **(13)**. This is particularly useful for understanding the implications of workload and the underlying hardware on performance. We conclude with a summary of our key observations on how compactions are central to the performance of LSM engines and how we can extract (near) optimal performance by choosing the appropriate compaction strategy.

4 DEMONSTRATION SCENARIOS

We demonstrate `Compactionary` in two scenarios with opportunities for extended interactions with the interested participants.

Scenario 1: Exploring LSM Compactions. We use the default input parameters to ingest 10M 128B entries (data size ~1.2GB) to an LSM-tree with size ratio 4 and a buffer size of 16MB (holding up to 2^{17} entries). Bloom filters are allocated 10 bits-per-entry, and the page size on disk is set to 4KB. By default, the vanilla LSM design is set to leveling and the customized compaction strategy is set to 1-leveling, with tiering in the first level and leveling in the others.

The participant then initiates the demonstration using the control panel, and the `Compactionary` UI illustrates through animation the step-wise construction of LSM-trees using four different compaction strategies. Each step in the animation shows the structure of the LSM-trees after every buffer flush. During the animation, the participants can observe the construction of LSM trees with four different compaction strategies side-by-side. Hovering over a level in the tree, reveals the number of entries and the number of files

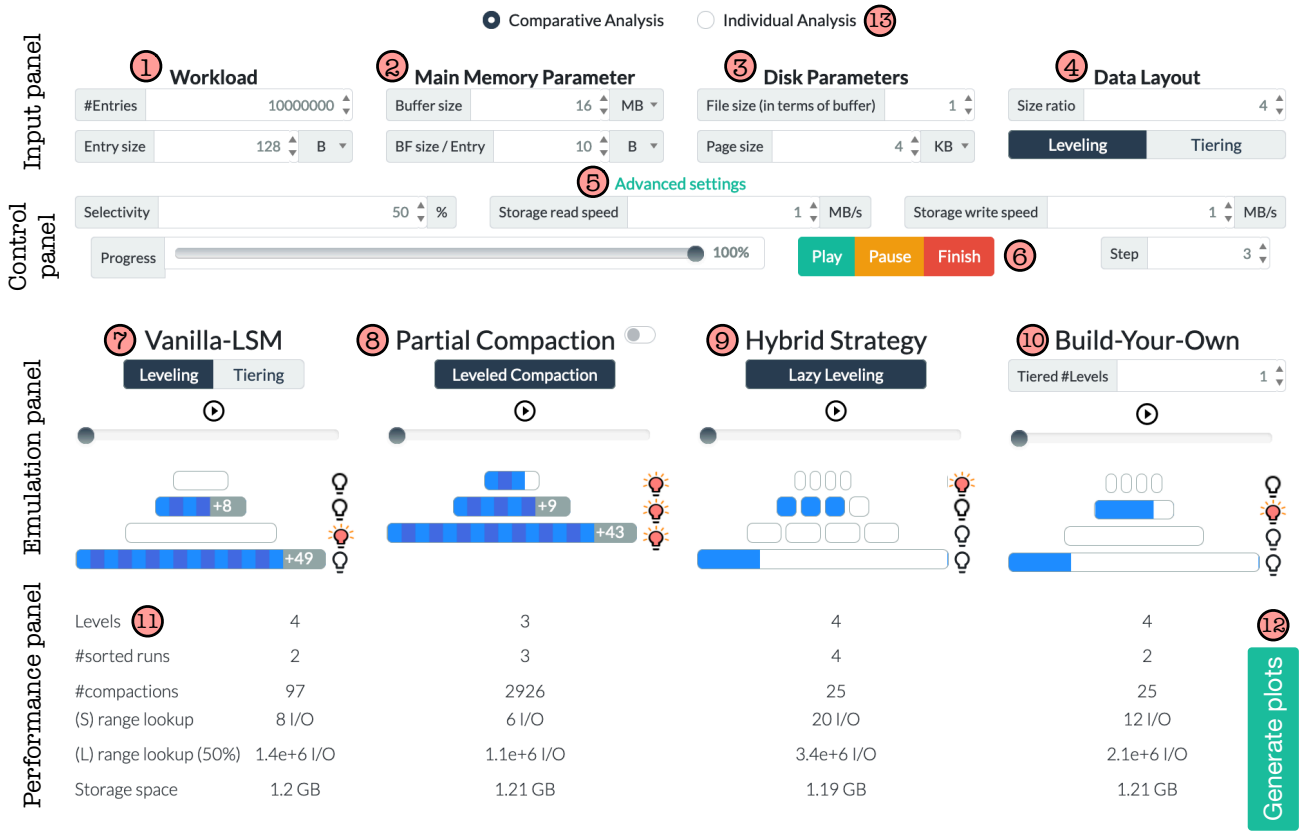

Figure 1: The `Compactionary` UI allows the participants to visualize and compare several compaction strategies simultaneously.

the level has along with its capacity. We point out that at the end of the emulation, the vanilla LSM design with full compaction has more levels compared to the other compaction strategies, due to compacting data at a larger granularity. The partial compaction routine, on the other hand, has a tree with the fewest number of levels, but performs compactions the most frequently.

Scenario 2: Exploring Custom Compactions. Next, we allow the participant to create their own hybrid compaction strategy. The participants can specify with the first $i \leq L$ levels implemented as tiering, while the remaining as leveling with full compaction. We point out how the new design compares against the other three compaction strategies, and how the input value for i affects the ingestion and lookup performance.

Bonus Scenario: Ad-Hoc Exploration. For the interested LSM-aware participants, `Compactionary` also has advanced options that enables a richer in-depth analysis. The *Individual Analysis* panel allows the participants to visualize the workflow of multiple compaction strategies, each running a different workload under different settings and LSM tunings. The goal of this analysis is to allow the participants to gain insights about how new hybrid designs affect the performance of compactions in LSM-trees. The *Partial Compaction* panel is also set up with a switch that enables running compaction jobs in the background. Participants can enable this option and set a threshold for level saturation to understand the implications of background compactions in LSM engines.

5 CONCLUSION

In this demonstration, we visualize different compaction strategies adopted in state-of-the-art LSM engines to understand the effects of compactions on performance. We introduce `Compactionary`, a dictionary of LSM compaction strategies, that allows the participants to analyze multiple compaction strategies and create new compaction strategies to explore the design space of LSM compactions.

ACKNOWLEDGEMENTS

We are thankful to Guanting Chen for his contributions in the early stages of the project. This work was partially funded by NSF under Grant No. IIS-1850202 and a Facebook Faculty Research Award.

REFERENCES

[1] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum. Optimizing Space Amplification in RocksDB. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.
[2] C. Luo and M. J. Carey. LSM-based Storage Techniques: A Survey. *The VLDB Journal*, 29(1):393–418, 2020.
[3] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.
[4] S. Sarkar, K. Chen, Z. Zhu, and M. Athanassoulis. Compactionary: A Dictionary for LSM Compactions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2022.
[5] S. Sarkar, T. I. Papon, D. Staratzis, and M. Athanassoulis. Lethe: A Tunable Delete-Aware LSM Engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 893–908, 2020.
[6] S. Sarkar, D. Staratzis, Z. Zhu, and M. Athanassoulis. Constructing and Analyzing the LSM Compaction Design Space. *Proceedings of the VLDB Endowment*, 14(11):2216–2229, 2021.