2024 IEEE International Conference on High Performance Computing and Communications (HPCC) | 979-8-3315-4046-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCC64274.2024.00035

A Window-Driven Compaction Mechanism in LSM-tree-based Key-Value Stores through Near-Data Processing

Hui Sun, Rui Jin, and Jiaming Huang
Sch. of Computer Sci. & Tech.
Anhui University, Hefei, China
sunhui@ahu.edu.cn,
{e21201042,e22201052}@stu.ahu.edu.cn

Yinliang Yue
Zhongguancun Laboratory
Beijing, China
yylhust@qq.com

Xiao Qin
Dept. of Computer Sci. & Software Engg.
Auburn University, Auburn, USA
xqin@auburn.edu

***Abstract*—LSM-tree-based key-value stores or KV stores, with their advantage of sequential writes, are widely deployed as backend storage engines. LSM-trees achieve updates by merging data through compaction operations. During compaction, however, background tasks – read, write, and filter large amounts of data – compete against foreground requests for the system's computational and I/O resources. Although numerous studies have sought to enhance system performance using high-speed storage along with computational devices, bottlenecks in CPU computation and I/Os still persist. Near-data processing (NDP) emerges as an effective solution to mitigate the bottleneck issues: the computational units of an NDP device not only expand the system's computational resources but also significantly reduce data movement by internally performing computations. Prior collaborative efforts in the NDP device have adopted a time-aware dynamic scheduling mode. Nonetheless, there still exists a noticeable disparity in processing time between a host and its device. To address this gap, we propose WinDB – a KV store that utilizes a window-driven task allocation approach. WinDB advances task allocation in fine-grained key ranges, ensuring that the data volume of compacted SSTables matches the computing capabilities of both host and device. Apart from device-level parallelism, WinDB embraces thread-level parallelism to enhance the system's overall performance. The experimental results unfold that WinDB bolsters the throughput by up to 4x that of RocksDB, with a significant reduction in average latency. WinDB is capable of curtailing resource contention, which in turn leads to shortened front-end response time.**

***Index Terms*—LSM-tree, Compaction, Near-data processing, Collaborative, Key range**

I. Introduction

Emerging e-commerce [1], large-scale language models [2], graph computation applications [3], and smart IoT applications [4], [5] yield large-scale unstructured data, which causes significant overhead on storage systems [6]. Therefore, the data structure for these unstructured data management plays a pivotal role in the quality of application services. Notably, LSM-tree-based key-value stores – KV stores – have been widely adopted as a back-end storage engine for processing unstructured data thanks to KV stores' high performance and scalability [7].

More often than not, popular KV stores – including RocksDB [8], Cassandra [9] – utilize the log-structured merge tree (LSM-tree) [10] as a primary data structure. These KV stores employ a buffer to append data through sequential writes, committing the data to persistent storage as a sorted-string table (SSTable). In LSM-tree, each level has a capacity threshold to minimize I/O overhead and improve data access. These thresholds increase with each level at a fixed ratio. When the data volume at a level exceeds its threshold, some data is migrated to the lower level. SSTables across levels may have overlapping key ranges, requiring them to be loaded into memory, merged, and rewritten into new SSTables to maintain data order. This process is known as compaction.

The compaction process entails numerous reads and writes, which consume time and I/O bandwidth between the host and its storage devices. Consequently, foreground user requests are inevitably delayed [11]. KV pairs processed during compaction deplete CPU resources, becoming a performance bottleneck [7], [12] under write-intensive workloads.

A growing number of studies are proposed to mitigate the impact of compaction on system performance. For instance, the LSM-tree structure was redesigned to revamp the performance of KV stores. The KV separation approach [13], [14] reduces write amplification (WA) caused by large-sized values. A new data structure was introduced to allow overlapping key ranges to exist at a level in an LSM-tree [15], thereby averting rewrites at the

979-8-3315-4046-3/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCC64274.2024.00035

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

same level. A level-driven method and sequential model [16]–[18] were incorporated to improve compaction throughput.

On the flip side of the coin, a number of merge operations place a strain on computational resources – a performance bottleneck in case of insufficient CPU resources. In prior studies [12], [19]–[21], offloading the compaction tasks to heterogeneous computational devices can lower CPU load. Nevertheless, the task offloading strategy needs consider the heterogeneity resources and the distribution of compaction. High-performance storage devices are widely applied in KV stores. A tiered storage architecture – one that embraces high-speed devices like non-volatile memory (NVM) [22], [23] – is suitable for enterprises operations owing to the budget constraints [24]. Near-data processing (NDP), integrating computation units in the storage device, provides new solutions for KV stores. Executing tasks on an NDP device reduces data movement – and lessens the host-side CPU burden. A handful of prior efforts [25]–[27] confirm that NDP devices have a full potential in enhancing compaction performance.

In this study, we design an approach to offload compaction tasks to an NDP device, leveraging the combined computing power of the host and its NDP device. We introduce WinDB, a key-value store that employs a window-driven task allocation strategy on an NDP-enabled platform. In WinDB, the task allocation estimates the data volume based on the computing capabilities of both the host and NDP device, ensuring that tasks are appropriately matched to the device's capabilities and minimizing the processing time gap between the host and device. Notably, the computing capability is defined as the ratio of compacted data volume to the compaction time in this paper. WinDB also incrementally adjusts the window over a fine-grained key range to identify the SSTables involved in a compaction task. Offloaded tasks are further divided into subtasks, which are executed in parallel on the device using multiple threads. To ensure data consistency, WinDB manages metadata produced by both the host and NDP device, with the NDP device acting solely as a metadata consumer.

Our contributions are summarized as follows.

- We propose a NDP-enabled KV store named WinDB, and use a window-driven strategy to allocate compaction tasks between the host and NDP device. This strategy predicts the data volume based on the computational capabilities of host and NDP device, expanding the compacted key range before task offloading.
- We implement multi-level parallelism frameworks, including both device- and thread-level parallelism, to speed up compaction execution by dividing tasks into subtasks and processing them in parallel.
- WinDB employs a cross-layer approach to manage metadata. We fully utilize the storage-computation integration offered by the NDP device. The host and NDP device retrieve metadata from both memory and external storage, minimizing the overhead in metadata transfer and parsing.
- We implement WinDB with the window-driven allocation strategy based on RocksDB. We conduct extensive experiments on a NDP platform. The results unveil that WinDB improves the throughput by up to 4× that of RocksDB – and the average latency is reduced.

The rest of the paper is organized as follows: Section II presents the background and related work on KV stores. Section III introduces the architecture of WinDB and the details of each module. Section IV provides the hardware and the software configuration in the experiment. In Section V, we conduct extensive experiments to demonstrate the performance of WinDB. Finally, we conclude our work in Section VI.

II. Background and related work

A. Background

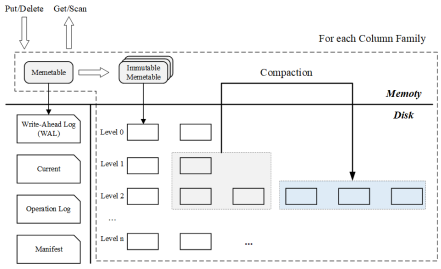


Fig. 1. An exemplified compaction procedure in RocksDB.

1) RocksDB: In RocksDB, a memtable in memory is generated to buffer users' write data. Data are initially committed to a log to mitigate the crash risks. When the memtable reaches a threshold, it will be converted into an immutable memtable. During the write process, RocksDB upgrades the write throughput via concurrent write operations. Data is stratified across multiple levels (L_0, L_1, ..., L_n), each one of which includes SSTables. Except for level L_0, key ranges are strictly ordered at other levels. Nonetheless, key-range overlaps may occur across different levels, which sparks compaction operations. The compaction procedure – consuming computational and storage resources – adversely affects response time and throughput.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

We take the leveled compaction in RocksDB as an example. Data is moved to the subsequent level when the data volume at a level surpasses the predefined threshold. This process elects SSTables at level L_k (k>=0) – and identifies overlapping SSTables at level L_{k+1}. Fig. 1 illustrates that the keys and values are filtered and written to a new SSTable at level L_2. After these operations are completed, the system deletes outdated SSTables. The process is conducted at each level down to the lowest, and the multiple rewrites of KV pairs lead to high write amplification.

RocksDB offers leveled and tiered compaction strategies. The latter – addressing key-range overlapping at the same level, can decrease the number of compaction operations while reducing write amplification. This method, however, is bounded to increases both space and read amplification. A reasonable practice is dividing a compaction task into numerous small tasks in scenarios with large-sized tasks. Individual threads manage these subtasks to accelerate the compaction via parallel computing.

2) Near-Data Processing: In data-intensive applications [28], large-scale data are processed to extract features – and build computational models. However, data migration for data processing incurs high costs. The data is often discarded after processing, leading to low performance and wasted resources. Traditional storage systems have become insufficient for the requirements of data-intensive applications. Consequently, researchers have proposed the near-data processing (NDP) architecture [29], which performs tasks near where data is stored using the computing units in memory-level [30] and storage-level [25] devices. We advocate for the storage-level NDP model because it can decrease data movement costs between the host and device and reduce host-side CPU burden. However, the resources in an NDP device are different from those on the host, and simple compaction task offloading policy can impair the performance of the KV stores. Therefore, we in this paper aim to consider the heterogeneous resources on both the host and NDP device.

B. Related Work

1) Optimizations of compaction in LSM-tree: Although KV stores manage unstructured data through the compaction process, this is achieved by sacrificing partial performance. Consequently, some studies focus on reducing the adverse impact of compaction on system performance.

First, some studies optimize the LSM-tree structure. Inspired by the skip list, PebblesDB [15] introduces a FLSM data structure, which lessens the constraints on non-overlapping key ranges at a level. WiscKey [13] separates the storage of keys and values, which diminishes the rewrites of values during compaction, but increases the overhead of garbage collection. HashKV [14] – an approach based on hash partition – addresses the issue of garbage collection in WiscKey. FGKV [31] proposes a fine-granularity compaction mechanism based on the LSpM-tree structure, reducing rewrites of irrelevant data. The second approach is to optimize the compaction strategy. LDC [16] and ALDC [17] reduce the granularity of compaction by using a low-level driver approach.

2) Computing architectures for LSM-tree: Unlike storage devices, the GPU, DPU, FPGA, ASIC, and ARM provide additional computational resources to process compaction tasks in KV stores. LUDA [32] utilizes the GPU resources to pack/unpack SSTables to reduce the host-side CPU load. Meanwhile, gLSM [21] applied a pair of CPU and GPU to concurrently execute compaction tasks, adopting the key-value separation approach to minimize data transfers between the CPU and GPU. Zhang *et al.* [12] customizes the FPGA resources to accelerate the offloaded compaction task. dComp [19] utilizes the accelerators within the data processing unit (DPU) to enhance SSTables compression and decompression, while selectively offloads compaction tasks to the DPU processor based on levels in KV stores to mitigate CPU contention.

3) Near-Data Processing for Compaction in KV stores: There have been studies applying NDP in KV stores. Co-KV [25] introduces a novel collaboration framework between the host and NDP device for compaction tasks. TStore [26] optimizes the task scheduling strategy in a dynamic task offloading model. MStore [33] encompasses multiple NDP devices to establish a KV store that balances computing and storage. Multiple NDP devices are applied to conduct compaction tasks.

III. WinDB Design

In this section, we present the overview system of WinDB and provide details of components in WinDB.

A. Overview System

WinDB is composed of a host-side subsystem derived from RocksDB and an NDP-device subsystem. The host-side subsystem manages users' I/O requests and the overall system; meanwhile, the NDP-device subsystem is an NDP-enabled storage device, extending computational resources to collaboratively perform compaction tasks with the host-side subsystem.

Fig. 2 presents an overview system of WinDB. The host-side subsystem includes three components. The H-Compaction module conducts compaction tasks on

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

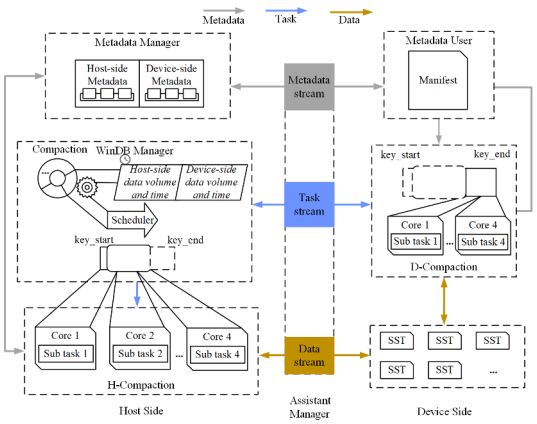


Fig. 2. Overview system of WinDB.

the host. The WinDB manager receives task completion information (such as metadata) from the host and the device. The assistant manager is integrated with the data and information management, charged with encapsulating and parsing data for the packet. The device-side subsystem has modules corresponding to those on the host.

B. Window-driven Approach for Compaction Task Allocation

Using the number of SSTables in the compaction task allocation may cause large-sized SSTables to be allocated on one subsystem, leading to many compaction tasks on the side. Therefore, we propose a window-driven task allocation approach according to data volume within the key range and the computing capabilities of the host and NDP device.

We obtain the execution time and data volume on the host and NDP device during compaction to gauge the computing capability. Then, we can derive the task processing ratio, which can be recorded as a metric to forecast the data volume in the subsequent task allocation. Notably, the initial task allocation is predicated on an 1:1 data volume distribution ratio for the host and NDP device.

The host- and device-side data volume (D_H and D_D) and execution time (T_H and T_D) can be derived as

$$V_H = \frac{D_H}{T_H} \ and \ V_D = \frac{D_D}{T_D}. \tag{1}$$

Given the data volume D_{all} for the compaction, V_H and V_D denote the computing capabilities of the host and device, respectively. We can predict the processing time of a compaction task, which is denoted as

$$T_{predict} = \frac{D_{all}}{V_H + V_D}. \tag{2}$$

According to the $T_{predict}$ and V_H, we can obtain the involved data volume on both sides, which are denoted as $D_{H-predict}$ and $D_{D-predict}$. For the host side, we have

$$D_{H-predict} = V_H \times T_{predict}. \tag{3}$$

According to Equ. (2), we have

$$D_{H-predict} = D_{all} \times \frac{V_H}{V_H + V_D}. \tag{4}$$

For the NDP device, we have

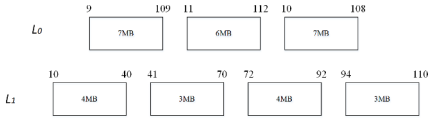
$$D_{D-predict} = D_{all} \times \frac{V_D}{V_H + V_D}. \tag{5}$$

After determining the data volume, we divide the compaction task according to key range distribution. Specifically, the key range is incrementally expanded, like a sliding window, until the cumulative data volume encompassed by the window (*i.e.,* key range) approximates the predicted value. Nonetheless, the division of the key range should not be in a finer granularity. As mentioned above, according to an SSTable's metadata, we can obtain its key range.

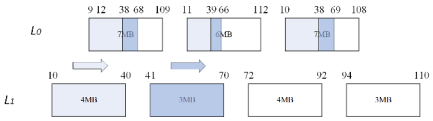
The key range selection needs to be in a reasonable length. An excessive length can lead to a difference between the divided data volume and the predicted data volume. In LSM-tree, we find that the threshold of data volume at each level escalates in a proportional relationship. The data volume at a lower level in an identical key range also increases proportionately. This occurrence is pronounced at level L_0, where the key range of an SSTable might encompass the entire key range of L_1. It is supposed that the tasks are divided based on the key range of SSTables at level L_0. When an SSTable is added to the task on the host, the whole task may be offloaded to one side while the other side does not execute the task, which cannot utilize full parallelism. Therefore, partitioning based on the key range in the lower-level SSTables of the compaction task is comparatively reasonable.

In WinDB, the task set is assembled progressively. First, we add the first SSTable at level L_{k+1} to the task on one side. Then, we compare the total data volume within the key range to the predicted value. This process continues until the difference between the sum of data volume and the predicted value is within an acceptable range. The smallest key of the next SSTable at level L_{k+1} becomes the boundary key. For instance, as shown in Fig. 3, we investigate SSTables at levels L_0 and L_1 in a compaction task. The total data volume is 34 MB, see Fig. 3(a). According to the results from the last task, the predicted data volume on the host is 14 MB. Initially, we select the key range of 9-41, see light-colored in Fig. 3(b). Within this key range, the data volume for the three SSTables is only 9 MB, which cannot meet the predicted 14 MB. Thus, we

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.



(a) Empty window in the compaction task.



(b) Initializing the key range of the window to 9-41 at level L_1.

Fig. 3. The expansion process of window-driven approach for task allocation. In Fig. 3(b), at level L_0, the data volume of SSTables in the key ranges that overlap with 9-41 are 2 MB, 1 MB, and 2 MB.Expanding the key range of the window to 9-72 at level L_1. At level L_0, the data volume of SSTables in the key ranges that overlap with 9-72 are 3 MB, 2 MB, and 3 MB.

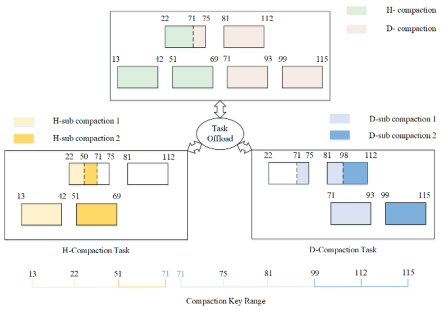


Fig. 4. In the host and NDP device, subtasks are further divided after both-side task division is completed.

expand the key range to 9-72, see dark-colored in Fig. 3(b), where the data volume of the divided task is 15 MB, approaching the predicted value of 14 MB. Consequently, key 72 is used as the boundary for the tasks, with 9-72 is the key range of the host side and 72-110 for the NDP device.

C. Methodology for Multi-level Parallelism

When compaction tasks are allocated to the host and device sides, we can consider these tasks as a single task on one side, which can be further divided into multiple subtasks. This method is defined as multi-level parallelism.

The host and NDP device divide subtasks based on the distribution of key ranges and the data volume. After establishing the relevant SSTables and boundary, the host creates a series of sequential and non-overlapping key ranges, see Fig. 4, which utilizes the smallest and largest keys from each SSTable at levels L_k and L_{k+1}. It is critical to ensure that the division of subtasks maintains the contiguity and completeness of the segments. To mitigate disparities in execution time across subtasks, each subtask must encompass an appropriate data volume. We forecast data volume to determine the number of subtasks and identify the initial key for each segment. The device divides subtasks in a similar manner to the host.

In Fig. 4, the host-side key range is 13-71. The iteration must ensure that the host's keys remains below 71. In contrast, the key for iteration in the NDP must be greater than or equal 71. We consider the smallest key from each file to be a candidate key for the key range of the subtask, including the end key. Consequently, the candidate keys for the host subtask include keys 13, 22, 51, and 71. We can calculate the total data volume spanning keys 13 to 71. Through balancing data volume among subtasks, the total number of subtasks is set to two. Each subtask is established through a step-by-step methodology. The initial subtask is incrementally delineated within the key range of 13-51, while the second subtask is built from key 51 to 71.

Each sub-task has independent task information, which ensures that there is no interference with each other. Furthermore, the information of the subtasks serves as credentials for various threads to perform the tasks in parallel. There is no thread switching overhead during critical resource access. We need to delete obsolete files after all subtasks complete.

D. Cross-layer Metadata Management

Existing NDP-enabled collaboration frameworks [25], [26] transfer task information from the host to the NDP device, including metadata. When the number of compacted SSTables is small, the impact of this method is negligible. However, as the number of SSTables increases, the overhead in data transfer and parsing becomes pronounced. Consequently, we design the cross-layer metadata management method.

The system records snapshots on the disk. These snapshots include the latest metadata. Upon receiving task information, the NDP device can independently access the metadata from the snapshot. The host merely indicates specific information about the task. The snapshot may have multiple metadata versions, and the NDP device autonomously selects the most latest version. The NDP device's direct retrieval of metadata from local storage is considerably swifter than transferring from the host.

After completing each task, the NDP device generates metadata of the new SSTables. The host still controls metadata updates to manage metadata. The host, after receiving metadata feedback from the NDP, integrates this information and synchronizes updates in the memory and Manifest file.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

IV. Hardware and Software Settings

We implement WinDB based on RocksDB and conduct experiments to validate its performance. RocksDB is used as the baseline in this experiment. We compare WinDB with Co-Rocks and T-Rocks, which incorporate the collaborative compaction schemes of Co-KV [25] and TStore [26] in RocksDB. We build a test platform to emulate the NDP-enabled KV store architecture, where the data transfer between the host and device is facilitated via Ethernet, see Fig. 5. This tested platform is easily extended to a real-world NDP platform.

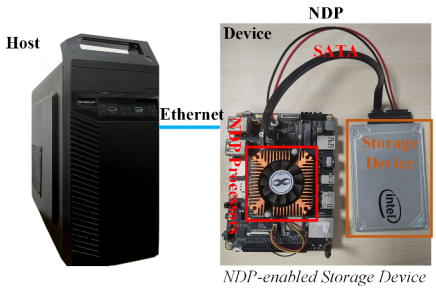


Fig. 5. An NDP-enabled experimental platform.

The host subsystem is a server with 8 Intel Core i3-10100 CPU cores and 16-GB DRAM. The NDP device operates on an ARM-based development board with 2 Cortex A72 cores, 4 Cortex A53 cores, 2-GB DRAM, and a 256 GB Intel SSD. The host and NDP device connect via a 1,000 Mbps network cable. In addition, we run Ubuntu 16.04 on the host, and we use RocksDB version 6.10.2. The NDP device is configured with a lightweight embedded OS. For RocksDB, we set the buffer and SSTable sizes to 4 MB, with a 4 KB block, and keep the level settings consistent. We limit the number of sub-compactions on the host to four, which is also applied to the device-side subsystem in WinDB. Other parameters remain at these default settings.

TABLE I
Characteristics of Workloads

Workload 1 (DB_Bench)			
Type	Feature	Data volume (10GB)	Value size (KB)
Load	100% Writes	1×, 2×, 3×, 4×	1×
Workload 2 (YCSB-C)			
Type	Feature	Data volume (GB)	Value size (KB)
Load	100% Writes	20	1×
A	50% Reads, 50% Updates		
B	95% Reads, 5% Updates		
C	100% Reads		
D	95% Reads, 5% Inserts		
E	5% Inserts, 95% Range Queries		
F	50% Reads, 50% Read-Modify-Writes		

Table. I lists the details of DB_Bench and YCSB-C used in the experiment. The DB_Bench workloads are used to evaluate random-write performance. YCSB-C is applied to gauge the system's performance under mixed read-write workloads.

V. Performance Evaluation

A. DB_Bench Benchmarks

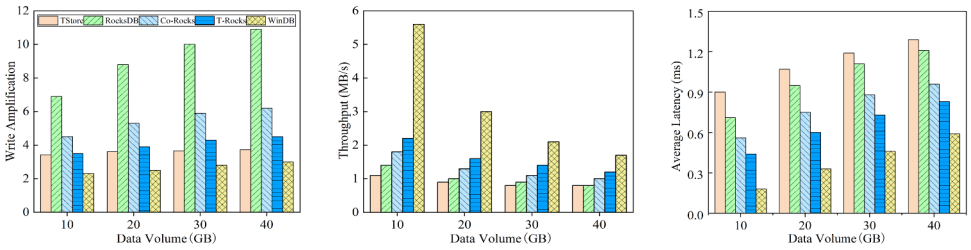
WinDB is designed to improve the compaction performance, and we evaluate its write performance in terms of write amplification, throughput, and average latency under workload DB_Bench.

1) Write Amplification (WA): A higher WA signifies a larger write data volume, which degrades the system performance. WA of LevelDB and RocksDB are caused by the write-ahead log and the compaction on the host. RocksDB exhibits higher WA than LevelDB due to a high-frequent compaction operations. WA escalates with increasing data volume because the number of compaction operations is proportional to the data volume. As shown in Fig. 6(a), under the workload with a 10-GB dataset, WA of Co-Rocks and T-Rocks is reduced by 35% and 49%, respectively, compared with RocksDB. This is because all compaction tasks in RocksDB are performed on the host. In NDP-enabled KV stores, the collaborative approach offloads a portion of a compaction task to the NDP device, reducing WA. Notably, WinDB demonstrates a 67% lower WA than RocksDB, and the similar results also exhibit in workloads with 20- and 30-GB datasets. Under a 40-GB workload, WA of Co-Rocks, T-Rocks, and WinDB decreases by 43%, 59%, and 72% compared with RocksDB, respectively; meanwhile, WinDB shows a 33% reduction compared with T-Rocks. These results unveil that the allocation strategy in WinDB is flexible and excellent. WA of WinDB does not present significant changes.

2) Throughput and Latency: In Fig. 6(c), the average latency of TStore ranges from 0.9 ms to 1.3 ms. It is noticeable that WinDB achieves a downward trend in average latency compared with TStore, indicating a lower response time to foreground requests. Compared with RocksDB, WinDB reduces the latency by 75% under workload with a 10-GB dataset. Meanwhile, Co-Rocks and T-Rocks reduce the latency by 21% and 38% compared with RocksDB, respectively. The maximum latency of WinDB is still below 0.6 ms under 40-GB workload, which achieves a 51% reduction than RocksDB. Additionally, WinDB shows a maximum latency of 59% lower than T-Rocks.

In Fig. 7(b), the device-side bandwidths of KV stores present apparent differences. On the device side, the highest bandwidth of TStore is only 4.4

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

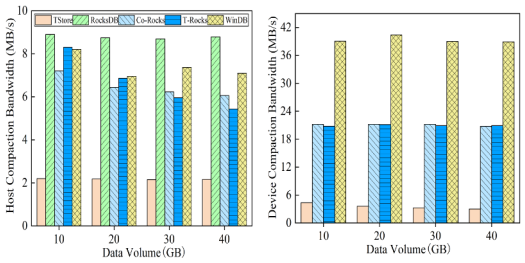


(a) Write amplification (WA). (b) Throughput. (c) Average latency.

Fig. 6. Write performance under DB_Bench with various data volumes.

MB/s. In contrast, Co-Rocks improves the bandwidth by up to 5.9× compared with TStore. WinDB leverages device-side multi-thread parallelism to fully utilize the computational capacity and shorten execution time. Therefore, the bandwidth of WinDB is maximally 1.91× that of T-Rocks under a 20-GB workload. T-Rocks and Co-Rocks have similar device-side bandwidth.

Except RocksDB, other KV stores employ collaborative compaction processing strategy to accelerate compaction performance, thereby improving the throughput of KV stores. The parallel write capability and multi-thread of RocksDB enhance the efficacy of the collaborative in NDP-enabled KV stores. WinDB improves performance by optimizing allocation strategy through an decision-making approach that integrates the computational capability of two-side subsystems with the data volume of tasks.



(a) Host side (b) Device side

Fig. 7. Host- and device-side compaction bandwidth of WinDB under DB_Bench with various data volumes

3) Compaction Bandwidth: The compaction bandwidth represents the performance of compaction processing in KV stores. In RocksDB, all compaction tasks are conducted on the host, whereas other KV stores provide the bandwidth on both the host and NDP device. The host-side compaction bandwidth is lower than that of the NDP device. This is because the throughput between the host and device is lower than that inside the NDP device, even though the host-side merging operations during compaction are rapid. In Fig. 7(a), the single-threaded TStore does not utilize the full computational capabilities of multi-core in the host; thus, the average bandwidth on the host side is around 2 MB/s.

In contrast, RocksDB uses multi-thread parallelism to expand the host-side bandwidth to over 8 MB/s, surpassing other KV stores. This is because the host handles all the tasks, causing a large-scale data volume. The collaboration scheme in NDP-enabled KV stores enhances the processing efficiency rather than the host-side bandwidth. The host-side bandwidth of WinDB is improved by 17% and 30% compared with Co-Rocks and T-Rocks under a 40-GB workload, respectively.

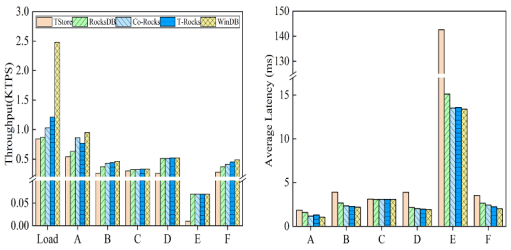
In Fig. 7(b), the device-side bandwidths of KV stores present an apparent difference. The peak device-side bandwidth of TStore is only 4.4 MB/s. In contrast, Co-Rocks achieves a 3.9× to 5.9× improvement in terms of bandwidth over TStore. Co-Rocks employs autonomous metadata read, similar to WinDB. T-Rocks and Co-Rocks have similar device-side bandwidth. WinDB leverages device-side multi-thread parallelism to unlock the computational capacity and shorten processing time. Consequently, the bandwidth of WinDB is maximally 1.91× that of T-Rocks under a 20-GB workload.

B. YCSB-C Benchmarks

YCSB-C offers real-world workloads, and we employ YCSB-C to evaluate the compaction performance of KV stores, *e.g.,* TStore, RocksDB, Co-Rocks, and T-Rocks, see Fig. 8. The data volume of the workloads is uniformly set to 20 GB for both the Load and Run stages. We configure record sizes of 1 KB and 4 KB and employ six types of workloads in the experiment, see Table I.

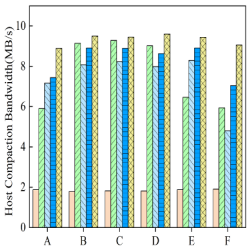
1) Load Stage: The Load stage is featured with a write-intensive workload that regularly triggers compaction. The access patterns in YCSB-C are different from that in DB_Bench; however, the advantage of WinDB in terms of compaction optimization still exists, see Fig. 8(a). The throughput of WinDB is improved by 1.9× compared with RocksDB. Co-Rocks,

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

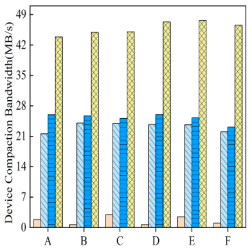


(a) Throughput.

(b) Average latency.



(c) Host-side compaction bandwidth.



(d) Device-side compaction bandwidth.

Fig. 8. Write performance under YCSB-C with 1-KB record size.

incorporating a static collaborative mode, narrows the performance gap with WinDB. Even though T-Rocks is implemented with the time-aware dynamic task scheduling, the allocation approach of WinDB further reduces the time gap between the host and NDP device in the collaborative mode. Consequently, the throughput of WinDB achieves 2.41× and 2.05× that of Co-Rocks and T-Rocks, respectively.

In Run stage, we can obtain two primary findings. First, the collaborative mode improves the performance of WinDB under write-intensive workloads. Workloads A and F have the highest write operation ratios (*i.e.,* 50%). Consequently, the throughput of WinDB exceeds that of Co-Rocks and T-Rocks by 10% and 23%, respectively under workload A. Under workload F, the throughput of WinDB achieves a 20% and 11% improvement over Co-Rocks and T-Rocks, respectively. However, when the proportion of write operations in workloads becomes small, the throughput of WinDB exhibits slight differences from that of other KV stores. Second, under read-intensive workloads, WinDB has a similar throughput to other KV stores.

In WinDB, we focus on optimization for the compaction process; therefore, the read performance remains unaffected. In contrast, TStore exhibits lower read performance than WinDB because the excessive number of SSTables leads to more I/O operations. This results in frequent switching between read/write and lookup states, increasing lookup operations.

2) Latency: Latency is a critical metric for KV stores, and latency affects users' response time as KV stores are extensively deployed in real-time applications. Fig. 8(b) shows the average latency of WinDB under workload A. Under write-intensive workloads, WinDB decreases the average latency by around 9% and 19% than Co-Rocks and T-Rocks under workload A, respectively. WinDB has a similar trend to workloads A and F.

3) Compaction Bandwidth: Figs. 8(c) and 8(d) present the compaction bandwidth of WinDB in the Load and Run stages under workloads. WinDB achieves the highest bandwidth on both the host and NDP device. However, the host-side bandwidth has slight improvement owing to the limitation of I/O transmission. Consequently, the host-side bandwidth of WinDB is improved by 89% and 29% than that of Co-Rocks and T-Rocks under workload F, respectively. Nonetheless, WinDB considerably enhances the device-side bandwidth by leveraging multi-thread parallelism. The average bandwidth of the device-side WinDB is higher than that of Co-Rocks and T-Rocks by 98% and 83%, respectively. The scheduling mechanism of WinDB can allocate tasks efficiently according to the computing capabilities of the host and NDP device.

VI. Conclusion

KV stores employ the compaction procedure for data reorganization, which consumes significant CPU and I/Os resources and impacts write performance. The host-side subsystem offloads compaction tasks to the NDP device to alleviate resource contention. Given the devices' heterogeneous resources, adopting a suitable task scheduling mode is vital to minimize processing time. We designed WinDB, a KV store using a window-driven task allocation approach. WinDB allocates tasks based on data volume in fine-granularity – key ranges, reducing execution waiting time. WinDB uses multi-level parallelization on the device and thread levels to enhance parallelism. Experimental results show that WinDB's task allocation optimizes compaction performance, increasing the throughput by up to 4× compared to RocksDB. .

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grants 61702004. Xiao Qin's work is supported by the U.S. National Science Foundation (Grants IIS-1618669 and OAC-1642133), the National Aeronautics and Space Administration (Grant 80NSSC20M0044), the National Highway Traffic Safety Administration (Grant 451861-19158), and Wright Media, LLC (Grant Number 240250).

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li, "X-engine: An optimized storage engine for large-scale e-commerce transaction processing," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 651–665.

[2] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H _2 o: Heavy-hitter oracle for efficient generative inference of large language models," *arXiv preprint arXiv:2306.14048*, 2023.

[3] Y. Matsunobu, S. Dong, and H. Lee, "Myrocks: Lsm-tree database storage engine serving facebook's social graph," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3217–3230, 2020.

[4] S. He, K. Shi, C. Liu *et al.*, "Collaborative sensing in internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1435–1474, 2022.

[5] W. Gong, H. Liu, K. Liu *et al.*, "Exploiting channel diversity for rate adaptation in backscatter communication networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[6] S. Pei, J. Yang, and Q. Yang, "Registor: A platform for unstructured data processing inside ssd storage," *ACM Transactions on Storage (TOS)*, vol. 15, no. 1, pp. 1–24, 2019.

[7] S. Dong, A. Kryczka, Y. Jin, and M. Stumm, "Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications," *ACM Transactions on Storage (TOS)*, vol. 17, no. 4, pp. 1–32, 2021.

[8] Facebook, "Rocksdb,a persistent key-value store for fast storage enviroments," 2016, http://rocksdb.org/.

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS operating systems review*, vol. 44, no. 2, pp. 35–40, 2010.

[10] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (lsm-tree)," *Acta Informatica*, vol. 33, pp. 351–385, 1996.

[11] O. Balmau, F. Dinu, W. Zwaenepoel, K. Gupta, R. Chandhiramoorthi, and D. Didona, "Silk: Preventing latency spikes in log-structured merge key-value stores," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 753–766.

[12] T. Zhang, J. Wang, X. Cheng, H. Xu, N. Yu, G. Huang, T. Zhang, D. He, F. Li, W. Cao *et al.*, "Fpga-accelerated compactions for lsm-based key-value store," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 225–237.

[13] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Wisckey: Separating keys from values in ssd-conscious storage," *ACM Transactions on Storage (TOS)*, vol. 13, no. 1, pp. 1–28, 2017.

[14] Y. Li, H. H. Chan, P. P. Lee, and Y. Xu, "Enabling efficient updates in kv storage via hashing: Design and performance evaluation," *ACM Transactions on Storage (TOS)*, vol. 15, no. 3, pp. 1–29, 2019.

[15] P. Raju, R. Kadekodi, V. Chidambaram, and I. Abraham, "Pebblesdb: Building key-value stores using fragmented log-structured merge trees," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 497–514.

[16] Y. Chai, Y. Chai, X. Wang, H. Wei, N. Bao, and Y. Liang, "Ldc: a lower-level driven compaction method to optimize ssd-oriented key-value stores," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 722–733.

[17] Y. Chai, Y. Chai, X. Wang, H. Wei, and Y. Wang, "Adaptive lower-level driven compaction to optimize lsm-tree key-value stores," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 2595–2609, 2020.

[18] Z. Zhang, Y. Yue, B. He, J. Xiong, M. Chen, L. Zhang, and N. Sun, "Pipelined compaction for the lsm-tree," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 777–786.

[19] C. Ding, J. Zhou, J. Wan, Y. Xiong, S. Li, S. Chen, H. Liu, L. Tang, L. Zhan, K. Lu *et al.*, "Dcomp: Efficient offload of lsm-tree compaction with data processing units," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 233–243.

[20] Z. Cao, H. Dong, Y. Wei, S. Liu, and D. H. Du, "Is-hbase: An in-storage computing optimized hbase with i/o offloading and self-adaptive caching in compute-storage disaggregated infrastructure," *ACM Transactions on Storage (TOS)*, vol. 18, no. 2, pp. 1–42, 2022.

[21] H. Sun, J. Xu, X. Jiang, G. Chen, Y. Yue, and X. Qin, "glsm: Using gpgpu to accelerate compactions in lsm-tree-based key-value stores," *ACM Trans. Storage*, vol. 20, no. 1, jan 2024.

[22] T. Yao, Y. Zhang, J. Wan, Q. Cui, L. Tang, H. Jiang, C. Xie, and X. He, "Matrixkv: reducing write stalls and write amplification in lsm-tree based kv stores with a matrix container in nvm," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, 2020, pp. 17–31.

[23] S. Kannan, N. Bhat, A. Gavrilovska, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Redesigning lsms for nonvolatile memory with novelsm," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 993–1005.

[24] H. Chen, C. Ruan, C. Li, X. Ma, and Y. Xu, "Spandb: A fast, cost-effective lsm-tree based kv store on hybrid storage," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 17–32.

[25] H. Sun, W. Liu, J. Huang, and W. Shi, "Collaborative compaction optimization system using near-data processing for lsm-tree-based key-value stores," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 29–43, 2019.

[26] H. Sun, W. Liu, J. Huang, S. Fu, Z. Qiao, and W. Shi, "Near-data processing-enabled and time-aware compaction optimization for lsm-tree-based key-value stores," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–11.

[27] T. Vincon, A. Bernhardt, I. Petrov, L. Weber, and A. Koch, "nkv: near-data processing with kv-stores on native computational storage," in *Proceedings of the 16th International Workshop on Data Management on New Hardware*, 2020, pp. 1–11.

[28] A. Margara, G. Cugola, N. Felicioni, and S. Cilloni, "A model and survey of distributed data-intensive systems," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–69, 2023.

[29] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle, "Active disks for large-scale data processing," *Computer*, vol. 34, no. 6, pp. 68–74, 2001.

[30] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 113–124.

[31] H. Sun, G. Chen, Y. Yue, and X. Qin, "Improving lsm-tree based key-value stores with fine-grained compaction mechanism," *IEEE Transactions on Cloud Computing*, vol. 11, no. 4, pp. 3778–3796, 2023.

[32] P. Xu, J. Wan, P. Huang, X. Yang, C. Tang, F. Wu, and C. Xie, "Luda: Boost lsm key value store compactions with gpus," *arXiv preprint arXiv:2004.03054*, 2020.

[33] H. Sun, Q. Wang, Y. L. Yue, Y. Zhao, and S. Fu, "A storage computing architecture with multiple ndp devices

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.

for accelerating compaction performance in lsm-tree based kv stores," *Journal of Systems Architecture*, vol. 130, p. 102681, 2022.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 23:21:38 UTC from IEEE Xplore. Restrictions apply.