This work was previously submitted to an SCIE journal (not an MDPI journal) and underwent a six-month review process over three rounds. Eventually, we were informed that resubmission was no longer possible, and the manuscript was formally rejected. Nevertheless, we revised the manuscript based on the reviewers' final comments. Given the valuable and constructive feedback received during the review process, we believe it is worthwhile to include the prior reviewer comments along with our corresponding revisions for reference.

Please note that in this version, we have made structural adjustments to the Related Work section, primarily involving changes to paragraph order. To facilitate readability for the reviewers, we have also updated the section numbers referenced in our responses accordingly.

Round 1: Reviewer Comments and Our Responses

Reviewer#1, Concern # 1: The design and implementation of TB-Collect is described in the paper, but some of the design decisions are not well grounded, and it is recommended that the authors add descriptions of some of the design choices to improve the reliability of the study.

Author response: Thank you for your feedback. We have supplemented the article with the rationale behind several key technical decisions and highlighted their advantages compared to existing similar strategies. Below is a detailed explanation of the revisions we have made.

Author action:

1. **Why adopt the tile-block separation method:**

We divided the tuple version into two parts: a tuple_header, stored in DRAM, and a tuple_content, stored in NVM, with a one-to-one correspondence between them. In Section 4.1, we added an explanation to clarify the rationale and advantages of this design. Below is a summary:

The tuple_headers in the tile are used for transaction processing, including the tuple's start and end timestamps and version chain pointers. After a system crash recovery, the transaction IDs are reset to their initial values, and only one snapshot version remains. Therefore, the content in the tuple_headers does not need to be persisted and can be stored in DRAM. On the other hand, the tuple_contents in the block record the version data, which must be persisted to ensure database durability. As a result, they are stored in NVM.

The advantages of this separation between tile and block storage are as follows:

1) It significantly reduces the NVM write amplification caused by small writes to the tuple_headers[1].

2) It supports subsequent version chain consolidation strategies, making it much more efficient compared to similar strategies (see Section 4.4).

[1] G. Liu, L. Chen, and S. Chen. Zen: a high-throughput log-free oltp engine for non-volatile main memory. Proceedings of the VLDB Endowment, 14(5):835–848, 2021.

2. **Advantages of the Epoch Mechanism**

We have added a discussion on the advantages of the Epoch mechanism in Section 4.2.
The Epoch mechanism significantly reduces contention on the last Commit ID. Traditional methods require threads to frequently access and update the last Commit ID, leading to performance bottlenecks in high-concurrency scenarios [2-4]. The Epoch mechanism, however, manages time through a global timer, eliminating the need for frequent synchronization of transaction states. Threads only need to read the current epoch at the start of a transaction, and subsequent operations use the local maintained time information, avoiding contention between threads. Additionally, the logical timestamp ordering of the Epoch mechanism simplifies the conditions for garbage collection and improves the execution efficiency of concurrent transactions, especially in multi-core systems, where hardware advantages can be fully utilized.

[2] Diaconu, Cristian, Craig Freedman, Erik Ismert, Per-Ake Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, and Mike Zwilling. "Hekaton: SQL server's memory-optimized OLTP engine." In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1243-1254. 2013.

Harvard

[3] Bernstein, Philip A., Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. Vol. 370. Reading: Addison-wesley, 1987.

[4] Lomet, David, Alan Fekete, Rui Wang, and Peter Ward. "Multi-version concurrency via timestamp range conflict management." In 2012 IEEE 28th International Conference on Data Engineering, pp. 714-725. IEEE, 2012.

3. Chain consolidation strategy in TB-Collect

We have added a section in Section 4.4 to explain the differences and advantages between the chain consolidation design in TB-Collect and the design in OneshotGC. The key points are as follows:

Owing to the separation of tile and block in TB-Collect, the version chain information, which occupies less space, can be fully stored in DRAM, while the tuple content in the block can be garbage collected without breaking the version chain. Compared to OneshotGC, our method has significant advantages. OneshotGC constructs a large hash index for each partition to query versions with broken version chains, which includes data from all tables. In contrast, TB-Collect does not require such a heavy operation; it only needs to directly reclaim the tuple content in NVM, and after long transactions end, it will reclaim the tuple header in DRAM. Additionally, due to the small size of blocks in TB-Collect, it avoids the issue in OneshotGC, where large regions (typically dozens of GB of partitions) must wait for long-running transactions to finish before garbage collection can proceed. This leads to more efficient garbage collection in TB-Collect.

Below is our supplemental explanation regarding the key design decisions in the manuscript. If you find any other areas that require further clarification, please feel free to let us know in subsequent revisions.

Reviewer#1, Concern # 2: The paper compares the experimental performance of TB-Collect

with that of the existing parties, but the discussion of the core design differences between the two is not in-depth enough, and it is suggested that the authors add more comparative analyses.

Author response:

Thank you for your feedback. In our previous experiments, we used TPC-C to evaluate the throughput, scalability, and latency of systems employing different GC strategies. Additionally, we utilized YCSB to assess the impact of the Zipf write skew factor, the number of operations per transaction, and the proportion of write operations on the throughput of each system. The purpose of these experiments was to evaluate the garbage collection efficiency of TB-Collect and other strategies under various scenarios.

As you pointed out, our discussion on the core differences in garbage collection design was insufficient. Therefore, we have designed a new set of experiments to further illustrate the design differences and advantages of TB-Collect compared to other strategies.

Author action:

We use YCSB with a workload of 50% reads and 50% updates, setting the Zipf parameter to 0 to ensure uniform data access. The initial dataset consists of 5 million tuples. Since the workload comprises only reads and updates, the number of tuples remains constant, but the number of versions increases over time. We observe the average version chain length and the average number of accesses per tuple across different systems over a given period. The average version chain length reflects the efficiency of GC in reclaiming space and influences the ease of accessing historical versions. A longer version chain indicates lower GC space reclamation efficiency and reduced read performance. The average number of accesses per tuple is calculated as the total number of accesses across all versions divided by the total number of tuples. This total access count includes regular read and write operations as well as accesses during GC. To ensure consistent read and write access patterns, we control the system throughput at a fixed value of 350,000 transactions per second (TPS) in the experiment. Thus, the average number of accesses per tuple determines GC efficiency: the more frequently versions are accessed, the lower the GC efficiency and the greater its impact on overall system performance [5].

The experimental results are shown in Fig.14. With 40 concurrent threads, the x-axis represents runtime (0 to 30 minutes), and the y-axis represents either the average version chain length or the average number of accesses per version. The average version chain lengths in Steam and N2DB are shorter than those in other systems, but their average access counts are significantly higher-up to 1.70× those of other systems. This is because both strategies reclaim old versions during transaction execution, achieving higher timeliness at the cost of frequent version chain traversal. OneshotGC exhibits the lowest average access count among all systems but has the longest average version chain length. This is due to its partition-based reclamation approach, where GC frequency is very low, requiring all versions in a partition to expire before reclamation starts, leading to excessively long version chains. In contrast, TB-Collect achieves a balanced performance, with an average version chain length only 51% of that in OneshotGC, and an average access count only 62% of that in Steam. This is attributed to TB-Collect's block-based reclamation approach, which operates at a finer granularity than partition-based methods while avoiding frequent version chain traversals during GC.

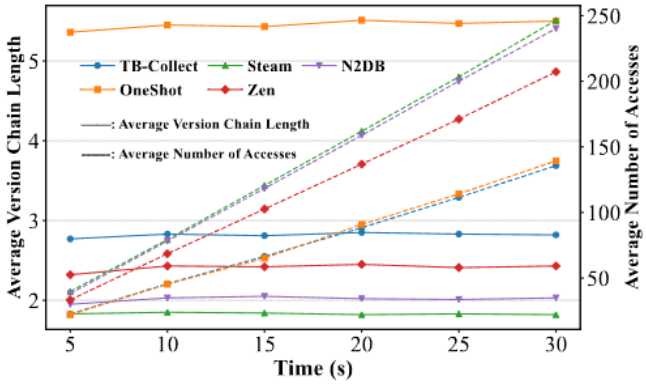


Fig. 14: Average Version Chain Length and Accesses Over 30 Minutes

[5] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. One-shot garbage collection for in-memory oltp through temporality-aware version storage. Proceedings of the ACM on Management of Data, 1(1):1–25, 2023.

Reviewer#1, Concern # 3: The paper illustrates the superiority of TB-Collect in many respects, but does not provide an in-depth analysis of its possible limitations, and it is recommended that the relevant discussion be added.

Author response:

Thank you for your feedback. We have now added a discussion section before the conclusion chapter, which includes an explanation of TB-Collect's applicability, limitations, and potential future work.

Author action:

TB-Collect exhibits highly efficient garbage collection (GC) capabilities, as it eliminates the need to traverse version chains during obsolete version reclamation and avoids constructing hash indexes for old versions during version chain consolidation. Currently, TB-Collect and other GC strategies adopt on-demand reclamation mechanisms. For example, TB-Collect triggers reclamation when the reclamation queue is full, OneShotGC initiates it when all versions in a partition become obsolete, and Steam triggers reclamation of obsolete versions before transaction commits. However, these strategies are not well-suited to handling high-concurrency workloads. Under such conditions, frequent GC triggers can consume excessive system resources. Conversely, reducing the GC frequency may lead to increased NVM usage and longer version chains, which can negatively impact normal data access. In future work, we plan to develop an adaptive mechanism that constructs a mathematical model based on statistical data, enabling TB-Collect to dynamically adjust the GC frequency according to the length of version chains.

Reviewer#2

Comments:

This paper proposes a high-performance GC method specifically designed for NVM OLTP

engines. Overall, it demonstrates a certain degree of innovation, presents comprehensive experiments, and is well-structured in its writing. The work holds notable practical value and is recommended for publication.

Round 2: Reviewer Comments and Our Responses

Reviewer#1, Concern # 1: The introduction section should provide more details on the current state of the art and challenges of the application of non-volatile memory (NVM) in OLTP (Online Transaction Processing) engines, especially the importance of the rubbish collection (GC) issue. This helps the reader to better understand the background and motivation of the research.

Author response: Thank you for your feedback. We have added a description of the current research status of NVM OLTP Engines and the importance of garbage collection issues in Section 1. The content is as follows:

Author action:

Currently, NVM OLTP engines can process millions of transactions per second while ensuring data consistency after system crash [r1]. This is due to NVM's near-DRAM read and write speeds and its non-volatility [r2]. Researches on NVM OLTP engines primarily focus on NVM indexing and recovery mechanisms [r3-r8]. There are no GC methods specifically designed for NVM OLTP engines. Instead, existing GC approaches in NVM OLTP engines are based on three prior methods. Due to the hardware characteristics and performance differences between NVM and DRAM, directly applying GC methods from DRAM OLTP engines can significantly degrade the performance of NVM OLTP engines. This issue can be analyzed from the following three aspects:

First, in DRAM OLTP engines, when a system crash occurs, all data stored in DRAM is lost due to its volatility, including obsolete versions. However, in NVM OLTP engines, the non-volatility of NVM means that obsolete versions are not cleared during a system crash but are instead retained. This necessitates the design of crash-specific GC strategies tailored to NVM OLTP engines. Second, GC methods like timely version chain pruning, background scanning collect obsolete versions by traversing the version chain, leveraging the byte-addressability of DRAM. However, this approach has been proven to be highly inefficient as the numerous random accesses to DRAM significantly impact system performance [r9]. Although NVM also has byte-addressability, its read and write speeds are lower than those of DRAM [r10, r11]. Consequently, using byte-addressability to traverse version chains for collecting obsolete versions amplifies the inefficiency in NVM OLTP engines. Third, to address the issue of frequent random accesses to DRAM during GC collection, the GC method partition clearing collects obsolete versions at the granularity of partitions. However, it requires partitions to be configured as large DRAM regions, resulting in increasingly long version chains within a partition, which directly affects the access efficiency of old tuple versions. Additionally, this method requires building a hash index for tuple versions within partitions to resolve version chain consolidation issues. For NVM, which has lower read and write speeds compared to DRAM, these two problems are further exacerbated.

[r1] G. Liu, L. Chen, and S. Chen. Zen: a high-throughput log-free oltp engine for non-volatile main memory. Proceedings of the VLDB Endowment, 14(5):835–848, 2021.
[r2] Intel optane dc persistent memory architecture and technology. https://www.intel.com/content/www/us/en/architecture-and-technology/ optane-dc-persistent-memory.html.
[r3] S. Ma, K. Chen, S. Chen, et al. Roart: range-query optimized persistent art. In 19th USENIX Conference on File and Storage Technologies (FAST 21), pages 1–16, 2021.
[r4] B. Zhang, S. Zheng, Z. Qi, et al. Nbtree: a lock-free pm-friendly persistent b+-tree for eadr-enabled pm systems. Proceedings of the VLDB Endowment, 15(6):1187–1200, 2022.
[r5] S. Chen and Q. Jin. Persistent b+-trees in non-volatile main memory. Proceedings of the VLDB Endowment, 8(7):786–797, 2015.
[r6] T. Wang and R. Johnson. Scalable logging through emerging non-volatile memory. Proceedings of the VLDB Endowment, 7(10):865–876, 2014.
[r7] S. Shin, S. K. Tirukkovalluri, J. Tuck, et al. Proteus: A flexible and fast software supported hardware logging approach for nvm. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pages 178–190, 2017.
[r8] M. Zhang and Y. Hua. Silo: Speculative hardware logging for atomic durability in persistent memory. In 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pages 651–663, 2023.
[r9] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. One-shot garbage collection for in-memory oltp through temporality-aware version storage. Proceedings of the ACM on Management of Data, 1(1):1–25, 2023.
[r10] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better i/o through byte-addressable, persistent memory. In ACM SIGOPS Operating Systems Review, 2009.
[r11] Bruce Jacob, David Wang, and Spencer Ng. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann, Burlington, MA, 2010.

Reviewer#1, Concern # 2: The manuscript should provide a more in-depth explanation of the design principles and implementation details of TB-Collect, please add.

Author response: Thank you for your feedback. We have added and modified several descriptions of the design principles and implementation details for TB-Collect.

Author action:

1) In Section 4.3, we have added more implementation details about the transaction execution process, the collection of obsolete blocks, and the reclamation of obsolete blocks. The main content is as follows:

Transaction Execution & Obsolete Blocks Collection. Fig. 5 shows the obsolete block marking process during the execution of transaction Txn 200, as well as the reclamation process of obsolete blocks after the transaction is committed. This transaction consists of a simple update statement. First, when the transaction begins (step 1), it reads the tuple version from tile 1 and sets the end_ts in the tuple_header of the tuple version to the current txn_id, marking the time when the tuple version becomes an old version. Note that at this point, the old version is not yet obsolete, as the transaction may fail to commit and roll back. Next, the tuple

version is marked as 1 in the update_map, and the max_txnid of the tile/block containing the tuple version is updated to the current txn_id (step 2). The max_txnid indicates the most recent modification time of the block. This process is NVM cache-aligned and requires only one NVM write. If the number of 1 in the update_map exceeds θ, meaning that old versions in the tile/block account for more than θ, the block becomes eligible for reclamation. However, the tile/block 1 cannot be immediately added to the local_gc_queue because the transaction is still executing, and the tile/block 1 is not yet obsolete. To address this, we designed a buffer queue called the gc_pending_queue, which is maintained and exclusively owned by each thread. The logical address of the tile/block is added to the gc_pending_queue. Meanwhile, the gc_pending_queue is scanned, and blocks that have become obsolete, i.e., those with block.max_txnid less than local_epoch_min, are dequeued and moved to the local_gc_queue (step 4). The local_gc_queue is also threadbound to the corresponding transaction. A block in the local_gc_queue indicates that the number of obsolete tuple versions within it exceeds θ and is ready for cleanup and reclamation.

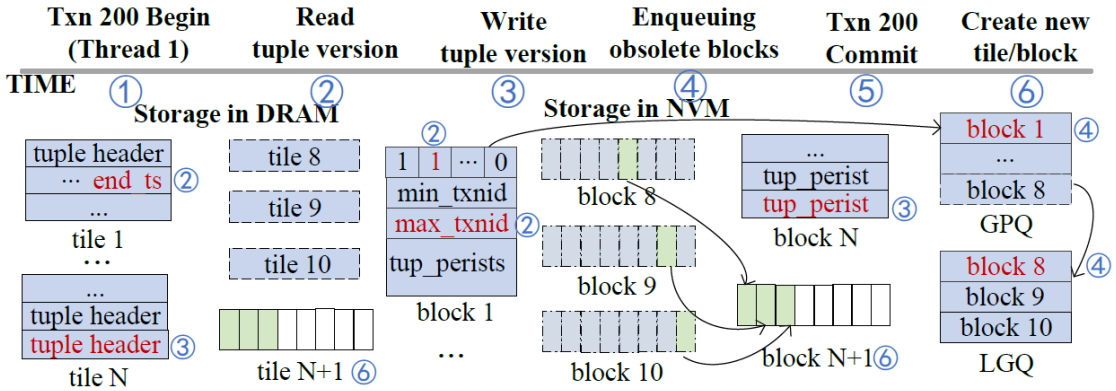


Fig. 5: GC Process of TB-Collect

Obsolete Blocks Cleanup and Reclamation. In this phase, after Txn 200 is committed, the transaction thread checks its maintained local_gc_queue. If the local_gc_queue is full, the thread enters GC mode. The address of the local_gc_queue is assigned to thread.local_gc_queue_frozen, and a new queue is created with its address assigned to local_gc_queue. Subsequently, for each block in local_gc_queue_frozen, the latest versions are quickly retrieved based on the offsets in the update_map (a value of 0 in the update_map indicates that the corresponding tuple version has never been modified) and are inserted into the new (block N+1) (step 6). Finally, all DRAM and NVM space corresponding to the tiles/blocks in local_gc_queue_frozen are released.

2) In Section 4.4, we have added the technical principles of TB-Collect in handling long transactions and cleaning obsolete blocks. The main content is as follows:

The process of collecting obsolete tiles and blocks, as well as the technical principles for handling long transactions, is as follows. For a block K where the number of obsolete tuple versions reaches θ, it is handled based on three scenarios: First, if the ID of the smallest active transaction Txn Min is greater than the max_txnid of block K, then the block can be reclaimed. Second, if there exists an active transaction Txn L whose ID is smaller than the min_txnid of block K, and the IDs of all other active transactions are outside the range of min_txnid to

max_txnid, then it indicates that Txn L is a long transaction. Moreover, none of the active transactions will access the data in block K. In this case, a part of the corresponding DRAM tile of the block can be retained to preserve the integrity of the version chain, and the block itself can be directly reclaimed. When the ID of Txn Min becomes smaller than the max_txnid of block K, the block can be reclaimed later. Third, if there exists an active transaction whose txnid falls within the range of min_txnid to max_txnid of block K, it indicates that the data in block K may still be accessed by active transactions, and thus block K will not be reclaimed.

3) In Section 4.4, we added specific details regarding the release of DRAM and NVM space at the end of the GC process. The main content is as follows:

Finally, each tile/block's logical address in local_gc_queue_frozen is traversed to locate the metadata in DRAM associated with the tile, including update_map and tile_header. These metadata are directly removed from DRAM, and any related data in the cache are invalidated to ensure that subsequent accesses do not reference the already-released DRAM space. Meanwhile, the block's logical address is used to locate the corresponding physical address range (start_addr and size). This range is marked as available and added to the free block list in the NVM space manager.

Reviewer#1, Concern # 3: For the experimental design section should be described in detail, and it is also recommended to add a full data justification of the experimental results

Author response: Thank you for your feedback. We have added specific descriptions of certain experimental designs to ensure that all experimental results are averages from multiple experiments, excluding the highest and lowest values. We have also emphasized this point in Section 5.1.

Author action:

1) We have added the experimental design details for YCSB and provided specific descriptions for the four variables. The main content is as follows:

In this evaluation, we use the YCSB workload to evaluate the impact of the garbage collection module on different systems. To assess the performance of various garbage collection methods in several common scenarios more deeply, based on previous studies [r1, r2] and the analysis in this paper, we have set up four controllable variables: the proportion of write operations in transactions, the write skew ZipF [r3], the number of concurrent threads, and the transaction size. First, as the proportion of write operations in a transaction increases, garbage collection is more likely to be triggered. By controlling the write proportion, we can evaluate the impact of the garbage collection module on the system under high load. Second, the higher the ZipF skew, the more concentrated the range of read and write operations will be. By controlling ZipF, we can evaluate the efficiency of garbage collection in reclaiming obsolete tuple versions for both hotspots and non-hotspots. Third, as the number of concurrent threads increases, the read and write pressure on the system also increases. By controlling the number of concurrent threads, we can evaluate the performance trend of each system under different levels of concurrency when garbage collection is enabled. Fourth, as

the transaction size increases, the total number of operations executed per unit of time also increases, causing versions to expire faster, thereby increasing the frequency of garbage collection. By controlling the transaction size, we can indirectly evaluate the impact of garbage collection on the system. In the following experiments, we adopt the method of controlling variables, where only one dimension is varied while the others are kept constant to ensure the accuracy of the results.

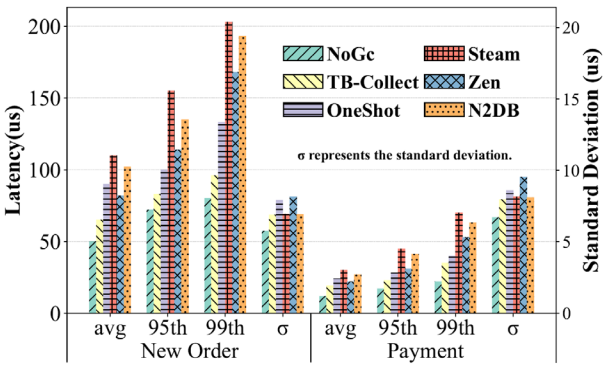
[r1] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. One-shot garbage collection for in-memory oltp through temporality-aware version storage. Proceedings of the ACM on Management of Data, 1(1):1–25, 2023.

[r2] Jan Böttcher, Viktor Leis, Thomas Neumann, and Alfons Kemper. Scalable garbage collection for in-memory mvcc systems. Proceedings of the VLDB Endowment, 13(2):128–141, 2019.

[r3] Yue Yang and Jianwen Zhu. Write skew and zipf distribution: Evidence and implications. In ACM Transactions on Storage (TOS), volume 12, pages 1–19. ACM, 2016.

2) We have added the standard deviation results for the TPCC latency experiment and revised the description of the experimental results. The main content is as follows:

Latency. As shown in Fig. 7 (c), NoGC exhibits the best performance in system latency. TB-CollectC shows the second-best performance in terms of response time, with average latencies in the NewOrder and payment transactions approximately 1.3x and 1.6x those of NoGC, respectively. However, its response time is highly variable, with standard deviations of 6.85us and 7.92us, because it does not require scanning the version chain during garbage collection (GC), and the append-only write method is also NVM-friendly. Zen performs well in average latency but poorly in 99th percentile latency, with the maximum difference between the average and 99th percentiles reaching 2.4x. Zen also has the worst latency stability among all systems, with response time standard deviations of 8.13us and 9.48us, due to the frequent scanning of the version chain during GC, which increases transaction latency. Moreover, GC is completed by background threads. OneShotGC shows a small difference between the average and 99th percentiles, as it releases memory globally during GC. However, due to the high overhead of building hashmap indexes for the version chain, its average latency is higher than TB-Collect, with the average latency of the NewOrder transaction being about 1.26x that of TBCollect. Steam and N2DB exhibit the worst response times in all systems, but their response times are very stable, with standard deviations similar to TB-Collect. Both Steam and N2DB need to frequently scan the version chain during GC; however, most of their GC work is done by the transaction execution thread at the time of transaction commit. As a result, both systems have long response times but very small standard deviations.

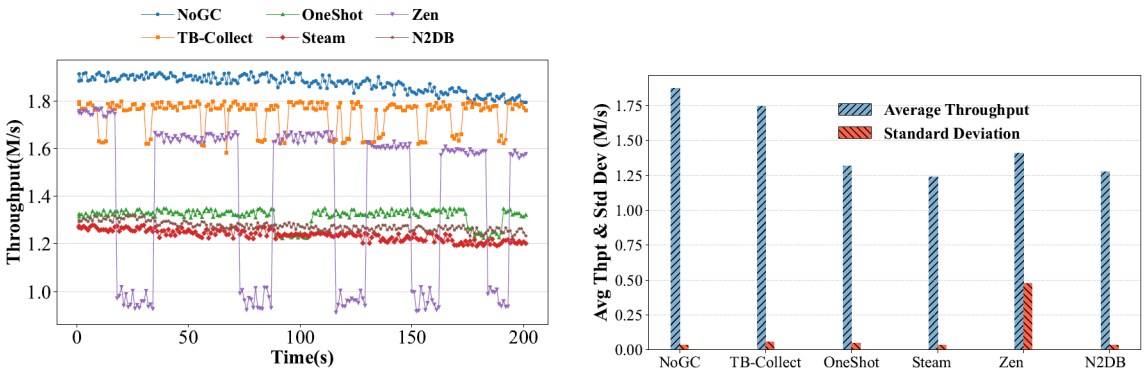


(c) Latency on TPCC

Fig. 7: TPCC Performance

3) We have added the average throughput and standard deviation for the system stability experiment running for 200 seconds. The main content is as follows:

The Impact of Garbage Collection. We evaluate the impact of GC on the system while it is running. We control the system to run TPCC and record the throughput per second. As shown in Fig. 13, with 40 concurrent threads and an initial warehouse count of 200, the x-axis represents time from 0 to 200 seconds, and the y-axis represents throughput. Before 150 seconds, the NoGC system, with the GC module turned off, exhibits the best stability and throughput. However, after 150 seconds, NoGC experiences a decrease in both performance and stability, while other systems with GC continue to maintain their previous performance. Over the entire process, NoGC has a throughput standard deviation of 0.035 M/s, which is worse than Steam and N2DB. This is because, in NoGC, the number of old versions increases, leading to a decrease in the efficiency of traversing long version chains. Steam has the worst throughput among all systems, with an average throughput of only 70.1\% of TB-Collect. However, its stability is the best of all systems, with a throughput standard deviation of 0.032 M/s. This is because Steam does not have an additional GC thread; instead, the local thread performs GC checks and recovery during commit. N2DB's design is similar to Steam's, but with some optimizations in NVM writes, resulting in better performance than Steam. TB-Collect has the second-highest throughput, only trailing NoGC. It performs the best among all systems with GC enabled. Despite having less stability than Steam, N2DB, and OneShotGC, with a throughput standard deviation of 0.058 M/s, its average throughput is much higher, about 1.23x to 1.40x that of the other systems. Zen is the most unstable, as its GC is performed by a background thread. It is worth noting that over time, the throughput of Steam, N2DB, and Zen shows an overall decreasing trend. This is because they do not fully release the space of obsolete tuple versions but instead reuse this space. As the reused space accumulates, the competition for allocation increases, leading to a decline in throughput.



(a) Throughput over 200 Seconds

(b) Throughput & Standard Deviation

Fig. 13: System Stability Over Time with Various GC Approaches

---

Reviewer#1, Concern # 4: The manuscript should explore in more depth how TB-Collect leverages NVM features (e.g., byte addressing, persistence, etc.) for more efficient rubbish collection. This would help the reader understand the technical innovation points of the approach.

Author response: Thank you for your insightful question. We would like to share our thoughts on this matter:

As is well known, the performance bottleneck of DRAM OLTP Engines lies in the I/O bandwidth of persistent storage devices. The initial approach was to replace the persistent devices with faster read/write speeds to improve system performance, from HDDs to SSDs and then to NVM. Since NVM has read/write speeds close to DRAM, as well as persistence and byte-addressability, the traditional DRAM OLTP Engine's log + checkpoint approach appears redundant. As a result, many studies began designing storage engines specifically for NVM, avoiding the traditional log + checkpoint approach for fast data recovery. Gradually, this has led to the current research ecosystem for NVM OLTP Engines. The garbage collection module, as an essential component in OLTP Engines, has not become more efficient due to the characteristics of NVM. On the contrary, it is precisely due to the performance limitations of NVM that the GC used in DRAM OLTP Engines becomes highly inefficient when applied to NVM OLTP Engines.

Next, we will mainly elaborate on the characteristics of NVM, focusing on its persistence, byte-addressability, and high-speed read/write capabilities.

First, in DRAM OLTP engines, when a system crash occurs, all data stored in DRAM is lost due to its volatility, including obsolete versions. However, in NVM OLTP engines, the non-volatility of NVM means that obsolete versions are not cleared during a system crash but are instead retained. This necessitates the design of crash-specific GC strategies tailored to NVM OLTP engines. Second, GC methods like timely version chain pruning, background scanning collect obsolete versions by traversing the version chain, leveraging the byte-addressability of DRAM. However, this approach has been proven to be highly inefficient as the numerous random accesses to DRAM significantly impact system performance [r1]. Although NVM also has byte-addressability, its read and write speeds are lower than those of DRAM [r2, r3]. Consequently, using byte-addressability to traverse version chains for collecting obsolete versions amplifies

the inefficiency in NVM OLTP engines. Third, to address the issue of frequent random accesses to DRAM during GC collection, the GC method partition clearing collects obsolete versions at the granularity of partitions. However, it requires partitions to be configured as large DRAM regions, resulting in increasingly long version chains within a partition, which directly affects the access efficiency of old tuple versions. Additionally, this method requires building a hash index for tuple versions within partitions to resolve version chain consolidation issues. For NVM, which has lower read and write speeds compared to DRAM, these two problems are further exacerbated.

In summary, the garbage collection (GC) methods designed for DRAM are not suitable for NVM OLTP Engines due to the hardware differences between NVM and DRAM. The hardware characteristics of NVM themselves are the root cause of the challenges in GC design for NVM OLTP Engines. Therefore, our design approach is to minimize the impact of GC on the throughput of NVM OLTP Engines. Given the performance gap between NVM and DRAM, we aim to design more efficient GC mechanisms to compensate for the hardware limitations of NVM.

[r1] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. One-shot garbage collection for in-memory oltp through temporality-aware version storage. Proceedings of the ACM on Management of Data, 1(1):1–25, 2023.

[r2] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better i/o through byte-addressable, persistent memory. In ACM SIGOPS Operating Systems Review, 2009.

[r3] Bruce Jacob, David Wang, and Spencer Ng. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann, Burlington, MA, 2010.

Thank you once again for your valuable suggestions on this manuscript. Today is the third day of the Chinese New Year. We would like to extend our best wishes to you for a happy and prosperous New Year, good health, and much success in the year ahead!

Reviewer#2:

Comments:

Thank you for the author's careful response. I recommend accepting this article.

Round 3: Reviewer Comments and Our Responses

Reviewer#1, Concern # 1: The authors are advised to further explore the comparative advantages of TB-Collect over other existing garbage collection methods, especially in terms of performance under different workloads and hardware configurations.

Author response: Thank you for your feedback. In order to test the garbage collection performance under different hardware configurations, we use Quartz [r1] to simulate the bandwidth of different NVMs. The specific approach is detailed in the response to Concern #2. Additionally, this paper lacks testing in complex scenarios, so we add the performance of YCSB

with mixed read/write operations, as explained in the response to Concern #3.
Currently, the standard benchmark sets for testing concurrent transaction performance of NVM OLTP engines are mainly YCSB and TPCC [r2, r3]. There are scenarios that we have already tested. In YCSB, we control dimensions related to garbage collection, such as concurrency scale, write ratio, transaction size, and write skewness, to deeply test the impact of garbage collection on the system in different scenarios. In TPCC, we observe latency, scalability, throughput, as well as the increase in storage space and changes in version chain length over time to evaluate the efficiency of garbage collection.

Author action:

We add a series of experiments, with detailed explanations in the responses to Concern #2 and #3.

[r1] Haris Volos, Guilherme Magalhaes, Ludmila Cherkasova, and Jun Li. Quartz: A lightweight performance emulator for persistent memory software. In Proceedings of the 16th Annual Middleware Conference, pages 37–49, 2015.
[r2] G. Liu, L. Chen, and S. Chen. Zen: a high-throughput log-free oltp engine for non-volatile main memory. Proceedings of the VLDB Endowment, 14(5):835–848, 2021.
[r3] Z. Ji, K. Chen, L. Wang, et al. Falcon: Fast oltp engine for persistent cache and non-volatile memory. In Proceedings of the 29th Symposium on Operating Systems Principles, pages 531–544, 2023.

Reviewer#1, Concern # 2: The paper lacks validation through actual case studies and experimental data. The authors are encouraged to validate the effectiveness of the model through real-world NVM OLTP application scenarios, such as analyzing the actual performance of garbage collection in a specific database system. Additionally, designing simulation experiments to test the model's performance under different environmental conditions would enhance the credibility of the research.

Author response: Thank you for your feedback. In order to test the performance of a real database system after applying garbage collection approaches, we implement an NVM-based Storage Engine in MySQL 8.4.2 [r2], and then implement the GC approaches mentioned in this paper. Additionally, to test the execution efficiency of garbage collection approaches on different NVM hardware devices, we use Quartz [r5] to simulate several NVM configurations. We then test the throughput, latency, scalability, and other performance metrics of each system with GC enabled.

After testing, we conclude that TB-Collect achieves the best GC efficiency under complex transaction processing workloads and is adaptable to real-world database system applications. As the NVM bandwidth and cache line size gradually approach and equal that of DRAM, the performance advantage of TB-Collect over other GC approaches diminishes, but it still maintains a certain level of superiority.

It is important to note that, unlike the lightweight transaction testing platform DBx1000, MySQL is a real database system, which requires additional overhead for network transmission, query parsing, and query optimization. As a result, the throughput of MySQL is much lower than that of DBx1000. Therefore, in the "**Efficiency of Space Reclamation**" evaluation, we are able to run longer tests on MySQL without causing overflow of DRAM and NVM.

Author action:

We have added the above experiments in Section 5.6, with the specific content as follows:

Garbage Collection in MySQL with Different NVM Configurations

In this evaluation, we will evaluate the performance of a real database system after applying garbage collection approaches. MySQL is a well-known open-source database system that has been widely adopted in commercial applications. Its Customer Storage Engine feature allows users to easily implement custom storage engines [r1]. Therefore, we implemented an NVM-based Storage Engine in MySQL 8.4.2 [r2], disabled MySQL's binlog [r3], and removed external locks [r4]. On this basis, we implement the GC approaches TB-Collect, TVCP, BS, and PC, naming them My_TB-Collect, My_TVCP, My_BS, and My_PC, respectively.

My_NoGC: MySQL with NVM append-only storage, without garbage collection, used as a baseline.

My_TB-Collect: A version of MySQL that implements the garbage collection approach described in this paper.

My_TVCP: A version of MySQL that implements the timely version chain pruning approach with append-only NVM storage.

My_BS: A version of MySQL that implements the background scanning approach with append-only NVM storage.

My_PC: A version of MySQL that implements the partition clearing approach with delta NVM storage.

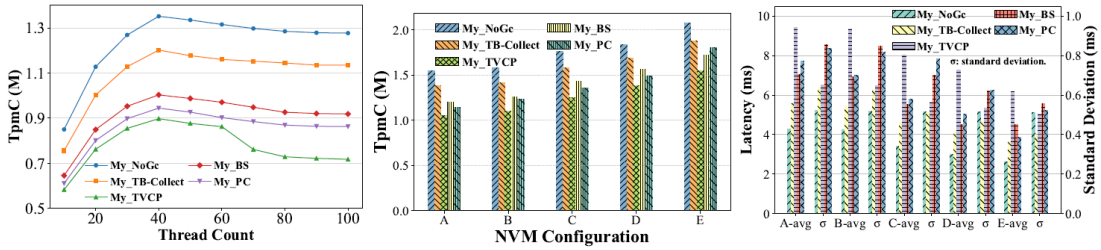
NVM Simulation Configurations. In order to test the execution efficiency of GC approaches on different NVM hardware devices, we simulate several NVM configurations using Quartz [r5]. Quartz is an NVM hardware simulation software that uses DRAM to simulate NVM read-write bandwidth, cache line size, latency, and other parameters. We set up the following five NVM hardware configurations, as shown in **Table II**. The configurations range from low to high bandwidth, ultimately reaching the performance of DRAM. The mainstream commercial NVM currently available (and used in this paper) is Intel Optane DC Persistent Memory [r7], which has 50% read and 30% write performance of DRAM, along with a cache line size of 256B. With the continuous advancements in semiconductor technology, NVM hardware will continue to evolve. The purpose of our testing is to verify whether the TB-Collect is suitable for the ever-evolving NVM devices. It is worth noting that in Quartz, as the NVM bandwidth is increased, the latency also changes accordingly. Therefore, our configuration does not include the NVM latency parameter.

TABLE 2: Five NVM Hardware Configurations Simulated by Quartz

Config	Read/Write Bandwidth (% of DRAM)	Cache Line
A	70%, 50%	256B
B	70%, 50%	128B
C	90%, 70%	128B
D	90%, 90%	128B
E	100%, 100%	64B

Experimental Environment. We use an Intel Xeon Gold 5218R CPU and 64GB DDR4 DRAM device as the request node for the TPCC load, while the transaction processing node is

equipped with the device described in Section 5.1. The two experimental devices are connected to a single switch and equipped with gigabit network cards. We use the TPCC benchmark, which simulates real-world e-commerce transactions, to evaluate the impact of GC on system performance and the recovery rate. The testing tool is BenchmarkSQL 5.0 [r6]. In addition, we use TpmC (Transactions per Minute C), a unit commonly used in industrial testing, as the system throughput metric, which represents the number of New Orders the system can process per minute while running TPCC. TpmC is primarily used to measure the performance of database systems under high load, particularly in commercial OLTP scenarios.



(a) Scalability on TPCC (b) Throughput on TPCC (c) Latency on TPCC

Fig. 15: TPCC Performance in MySQL with GC Implementation under Different NVM Hardware Configurations

Scalability. In this experiment, we test the throughput of each system as the thread count varies using real NVM hardware (described in Section 5.1). As shown in Fig.15 (a), it can be observed that with the increase in the number of concurrent threads, My_TB-Collect consistently delivers the best throughput among all systems with GC enabled. Furthermore, at 40 threads, it reaches 88% of the My_NoGC throughput. When the thread count exceeds the number of logical CPU cores, the throughput of all systems decreases and eventually stabilizes. During the entire process, My_TB-Collect maintains a performance advantage, achieving throughput approximately 1.23x-1.58x higher than other systems.

Throughput. In this experiment, we control the number of threads to 40 and evaluate the system throughput with different NVM configurations. As shown in Fig.15 (b), we can see that when the NVM cache line size decreases from 256B to 128B, the changes in My_TB-Collect, My_TVCP, and My_BS are not significant, with only a 3%-5% increase. My_PC shows the largest improvement, about 10%, because as the NVM cache line size decreases, the number of write amplification operations also decreases. My_TB-Collect has the fewest write amplification operations among all systems, so the performance improvement is not very noticeable. On the other hand, My_PC, due to using in-place version storage, has the largest write amplification operations, resulting in the largest performance improvement. With the continuous increase in NVM read/write bandwidth, the performance of all systems also improves, and My_TB-Collect still performs the best among the systems. When the NVM read/write bandwidth is fully aligned with DRAM and the granularity is 64B, which means NVM can match the performance of DRAM, we observe significant performance improvements in My_TVCP, My_BS, and My_PC, all of the GC approaches in these systems are designed for DRAM. Among them, My_PC shows the greatest improvement, about 95% of My_TB-Collect's performance. Nevertheless, My_TB-Collect still maintains a certain performance advantage. The reason is that My_TVCP and My_BS require traversing version chains during GC, which leads to performance degradation, while TB-Collect does not. Moreover, compared to My_PC, My_TB-Collect has a faster GC frequency, and its version chain length remains low, resulting in

higher access efficiency for old versions compared to My_PC.

Latency. Fig.15 (c) shows average latency of the systems under different NVM configuration hardware scenarios. In this experiment, we evaluate the response time of the New-Order transaction, which has the highest proportion in TPCC, while controlling the number of threads to 40. With the increase in NVM hardware bandwidth, the response time of all systems decreases significantly. My_TB-Collect still maintains the lowest average latency. In terms of latency stability, except for My_TVCP, the other three systems show a significant improvement. This is because the TVCP approach performs version collection during each transaction execution. In contrast, the other three systems perform triggered collection after reaching a threshold, which requires executing a larger number of collection operations at once. As the bandwidth increases, the latency decreases more when executing large-scale collection tasks, resulting in improved stability.

Efficiency of Space Reclamation. In this experiment, we control the number of threads to 40 and test the system's NVM and DRAM space increase after running for 120 minutes under different NVM configurations. To ensure fairness, we control each system's throughput at 800K transactions per minute, with an initial warehouse size of 200. As shown in Fig.16, the increase in NVM and DRAM space does not change with the increase in NVM bandwidth. This is because the space efficiency of GC depends on how the approach handles expired versions and the storage layout, which is independent of NVM bandwidth. Similar to the results in Fig.12 **(in Section 5.5)**, TB-Collect exhibits the best reclamation efficiency.

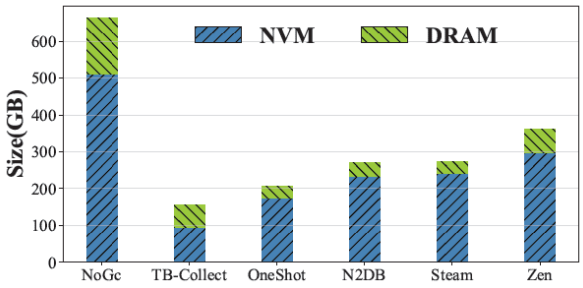


Fig. 12: Space Size Increase Over 10 Minutes

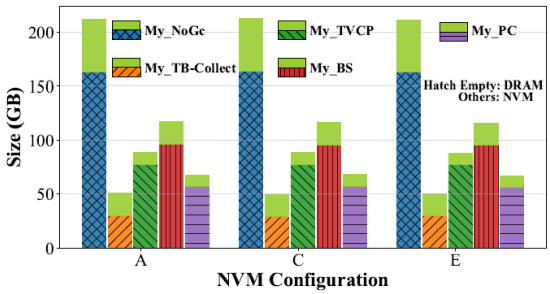


Fig. 16: Space Size Increase Over 120 Minutes with Different NVM Hardware Configurations

[r1] Paul DuBois. MySQL. Addison-Wesley, 2013.
[r2] Mysql 8.4 reference manual. https://dev.mysql.com/doc/refman/8.4/en/.
[r3] The binary log. https://dev.mysql.com/doc/refman/8.4/en/binary-log.html.
[r4] External locking. https://dev.mysql.com/doc/refman/8.4/en/external-locking.html.
[r5] Haris Volos, Guilherme Magalhaes, Ludmila Cherkasova, and Jun Li. Quartz: A lightweight

performance emulator for persistent memory software. In Proceedings of the 16th Annual Middleware Conference, pages 37–49, 2015
[r6] Benchmarksql. https://github.com/pingcap/benchmarksql.
[r7] Intel optane dc persistent memory architecture and technology. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

Reviewer#1, Concern # 3: Although the theoretical derivation is detailed, the discussion on the limitations and applicability of these methods is insufficient. For example, the paper does not fully explore whether these methods remain effective under complex workloads, such as mixed read and write operations.

Author response: Thank you for your feedback. In Section 5.3, for the read/write ratio dimension in transactions, testing only 50% read, 50% write, and 100% write is not sufficient. As you mentioned, we lack testing for more complex mixed read/write workloads. Therefore, we design a set of experiments to demonstrate that TB-Collect remains effective under the aforementioned scenarios. We control the number of concurrent threads to 40, the ZipF parameter to 0, and the transaction size to 20, to eliminate interference from these variables. At the same time, we control the total read/write ratio to 100%, starting with 10% write and 90% read, and then reduce the read percentage by 10% while increasing the write percentage by 10%, until reaching 100% write. We then obtain the throughput ratios of each GC approach compared to NoGC.

The experimental results show that, as the write ratio increases, although the throughput of each system gradually approaches, TB-Collect still maintains a certain advantage.

Additionally, in the previous experiments, we use TPCC for testing. The TPCC workload consists of five relatively complex transactions that simulate a complex order transaction scenario [r1]. These include:

Transaction 1 (45%): Approximately 45 operations, read/write ratio: 22/23.
Transaction 2 (43%): 9 operations, read/write ratio: 5/4.
Transaction 3 (4%): 4 operations, read/write ratio: 4/0.
Transaction 4 (4%): Approximately 50 operations, read/write ratio: 20/30.
Transaction 5 (4%): 1 operation, read/write ratio: 1/0.

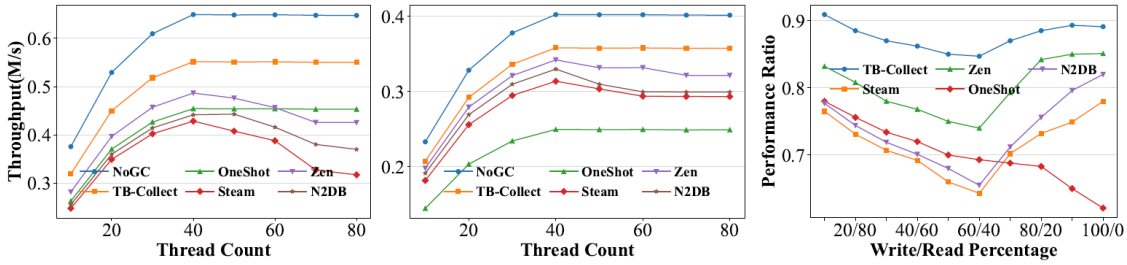
When running TPCC, the system using TB-Collect achieves the best throughput, lowest latency, and optimal scalability.

[r1] Tpc benchmark c. http://www.tpc.org/tpcc/.

Author action:

1) We add the following content to the second half of the paragraph in Section 5.3

Then, to further explore the performance of these GC approaches under mixed read-write workloads, we control the concurrent threads to 40, ZipF parameter θ to 0, and transaction size to 20. We vary the write/read percentage and obtain the throughput ratios of each GC system compared to NoGC, as illustrated in Fig.8 (c).



(a) 50% Writes (b) 100% Writes (c) Varying Read-Write Percentages

Fig. 8: YCSB Performance with Thread Count and Read-Write Percentages

2) We add the following content to the fifth paragraph of Section 5.3.

Third, to further validate the results in Fig.8 (b) and explore the performance of these GC strategies under mixed read-write workloads, we design a set of experiments. While keeping other parameters unchanged, we control the total read-write percentage to 100%, starting with 10% writes and 90% reads, and reduce the read ratio by 10% while increasing the write ratio by 10% in each step, until reaching 100% writes. As shown in Fig.8 (c), we obtain the throughput ratios of each GC strategy compared to NoGC. The experimental results show that as the write percentage increases in the workload, TB-Collect's advantage over other systems becomes more apparent. When the transaction consists of 60% writes and 40% reads, TB-Collect's advantage peaks, with throughput 10.2% to 20.3% higher than other systems. When the write percentage exceeds 80%, TB-Collect still maintains a performance advantage over other systems, but the performance of most systems, except for OneShot, gradually converges. This is because, in scenarios with a high write load, the bottleneck of each system gradually shifts to NVM writes, which limits the full potential of TB-Collect's garbage collection advantages. On the other hand, OneShot's performance degradation is more significant due to its use of Delta storage, which is detrimental to NVM writes.

Reviewer#2:

Comments:

The revised manuscript meets the publication standards of the journal. I wish the authors continued success and valuable contributions in their future research endeavors.