IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 29, 2021, accepted April 11, 2021, date of publication May 6, 2021, date of current version May 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3077898

Write-Optimized and Consistent Skiplists for Non-Volatile Memory

RENZHI XIAO, DAN FENG, (Senior Member, IEEE), YUCHONG HU, (Member, IEEE), FANG WANG, (Member, IEEE), XUELIANG WEI, XIAOMIN ZOU, AND MENGYA LEI

Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System, Ministry of Education of China, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Corresponding author: Dan Feng (dfeng@hust.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003305, in part by the National Natural Science Foundation of China under Grant 61772222, in part by the Shenzhen Knowledge Innovation Program under Grant JCYJ20170307172447622, and in part by the Key Laboratory of Data Storage System, Ministry of Education.

ABSTRACT Skiplist as an in-memory index performs pretty well on rapid insertions because there are no rotations or reallocations for rebalancing. The emerging non-volatile memory (NVM) technologies have spurred a deep interest in designing efficient NVM-based skiplist. Because the data written to NVM may be partially updated or reordered by the memory controller, write operations in NVM-based persistent skiplist may suffer data inconsistency in the face of system failures. Moreover, a traditional Redo-Logging-based consistent Skiplist (RLS for short) guarantees data consistency but introduces double NVM writes, which can significantly degrade the lifetime of NVM. In this paper, we propose two write-optimized and consistent skiplists for NVM, called Atomic Skiplist (AS for short) and Atomic and Selective Consistency Skiplist (ASCS for short). AS exploits log-free failure-atomic writes for both the 0th and internal levels of skiplist to avoid double NVM writes of redo-logging. ASCS leverages selective consistency and log-free failure-atomic writes to further reduce NVM writes from index pointers. Compared with RLS, experimental results show that AS and ASCS reduce the number of cache-line flushes by 67.5% and 75%, decrease the insertion latency by 32.3% - 40.9% and 36.2% - 54.2%, degrade the deletion latency by 38.7% - 47.1% and 44.9% - 57.8%, as well as increase insertion throughput by 49.1% and 65.0% and deletion throughput by 65.1% and 80.5%, respectively.

INDEX TERMS Non-volatile memory, consistency, write efficiency, skiplist.

I. INTRODUCTION

Non-volatile memory (NVM) technologies, such as phase-change memory (PCM) [1], spin-transfer torque magnetic RAM (STT-RAM) [2], resistive random access memory (ReRAM) [3], and Intel Optane DC Persistent Memory (DCPMM) [4], combine the non-volatility property of HDD with byte-addressability and similar latency to DRAM to offer some real competitive edges over DRAM. Skiplist is a widely used in-memory index structure in key-value stores [5], [6]. Therefore, an NVM-based persistent skiplist will keep data persistent in the face of system power failure.

Unfortunately, when the system fails, the data written to NVM may be partially updated or reordering, resulting in data inconsistency in NVM. Therefore, NVM-based data structures must ensure data consistency in the face of system failures [7]. The failure-atomic write unit of NVM is 8 bytes which is much smaller than that of block-based storage [8]. For write units larger than 8 bytes, traditional consistency methods such as logging are used to ensure data consistency [9]. NVM has limited endurance (e.g.,10^7-10^8 writes [10]), but logging requires double writes, which will harm the lifetime of NVM.

Skiplist does not require updating many nodes for re-balance as B+Tree does and is easy to implement [11], but at the expense of many pointer updates in the insert and delete operations [12]. We find that only guaranteeing the consistency of the 0th level of skiplist (i.e., the essential list) can ensure the consistency of the entire NVM-based persistent skiplist. Moreover, traditional double-write techniques such as write-ahead-logging combined with frequent pointer updates in skiplist increase consistency overhead and reduce

The associate editor coordinating the review of this manuscript and approving it for publication was Ashish Mahajan.

the write performance of NVM-based persistent skiplist. Therefore, NVM-based persistent skiplist needs to tackle data inconsistency and write problems.

In this paper, we propose two write-optimized and consistent skiplists for NVM, Atomic Skiplist and Atomic and Selective Consistency Skiplist. Atomic Skiplist (AS for short) uses a log-free failure-atomic write through ordered 8-byte atomic write with memory fence (MFENCE) and cache-line flushes (CLFLUSH) instructions. Atomic and Selective Consistency Skiplist (ASCS for short) employs a selective consistency (SC) mechanism over log-free failure-atomic write (Atomic) to further reduce consistency overhead by only guaranteeing the consistency of the essential list of skiplist. We have implemented AS and ASCS and evaluated them using four micro-benchmarks and two macro-benchmarks generated by YCSB. Experimental results show that our proposed AS and ASCS outperform a traditional Redo-Logging NVM-based consistent Skiplist (RLS for short) by 67.5% and 75% in the number of CLFLUSH, 32.3% - 40.9% and 36.2% - 54.2% in the insertion latency, 38.7% - 47.1% and 44.9% - 57.8% in the deletion latency, 49.1% and 65.0% in the insertion throughput, as well as 65.1% and 80.5% in the deletion throughput, respectively.

The rest of the paper is organized as follows: We introduce the skiplist index for NVM as background and our motivation for the designs of new skiplists in Section II. We describe the design of AS and ASCS in Section III. Section IV evaluates the performance of AS and ASCS. Section V is the related work, and Section VI concludes.

II. BACKGROUND AND MOTIVATION

A. NON-VOLATILE MEMORY

Unlike traditional memory technologies such as SRAM and DRAM that suffer from limited scalability and high power consumption, emerging non-volatile memory technologies, e.g., Intel Optane DC Persistent Memory [4], ReRAM [3], PCM [13] and STT-RAM [2] can provide far larger memory capacity than DRAM and near-zero leakage power. Moreover, NVMs can provide both the persistent property of HDDs and nearly comparable read/write latency to and byte-addressability properties of DRAM. Therefore, NVMs have been considered promising candidates of next-generation main memory.

However, NVMs are prevented from wide use as main memory by their common limitations. First, NVMs generally have limited write endurance, e.g., 10^7-10^8 writes [10]. Second, write latency is much higher than reading latency (i.e., 3-8X) [14], and writes also consume higher energy than reads. Thus, reducing the amount of data to be written into the NVM can increase the lifetime of NVM and reduce total system latency. Many prior studies have devoted to improving the lifetime of NVM by reducing wear unevenness or reducing writes [15]. Furthermore, and significantly, when we use NVMs as main memory, the volatility-persistence boundary moves from the interface between DRAM and HDD to the interface between the volatile cache and persistent NVM [16]. Data consistency, i.e., the correctness of data after a system failure, must be guaranteed in NVM-based systems [9]. Compared to traditional block storage devices, NVM generally has a much smaller failure-atomic unit of 8 bytes [8], [17], [18]. For update units larger than 8 bytes, the order of memory writes must be carefully ensured [19]. As a result, NVM-friendly data structures have to address these limitations of NVMs, including tree-based structures, hashing-based structures, and skiplist-based structures, and so on.

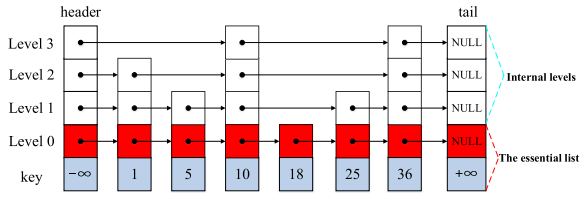


FIGURE 1. Example of a 4-level skiplist.

B. SKIPLIST

As they are simpler and more stable than their balanced tree counterparts to build from bottom to up [11], skiplist-based data structures are widely used in main memory key-value (KV) stores (HBase, MemSQL, LevelDB). As shown in Figure 1, each level in a 4-level skiplist is a linked list. A node with a unique key in the skiplist represents the node, and its ***internal forward pointers*** point to its successor nodes. A level count in each node represents the number of internal forward pointers of this node [11]. For example, the level count of node 25 is 2. There are many pointers in the skiplist point to this node 25 called ***external forward pointers***. Level 0 is an essential ordered linked list that contains all real KV data and nodes in the skiplist called ***the essential list***. Other inner levels in the skiplist except level 0 are called ***internal levels***. A level number in each skiplist represents the number of levels in the skiplist's essential list and internal levels. All pointers include internal forward pointers and external forward pointers, except the pointers in the essential list, are called ***index pointers***.

C. CHALLENGES

One challenge in designing an NVM-based persistent skiplist is hard to ensure crash consistency. As shown in Section II(A) and II(B), write operations in skiplist may cause data inconsistency because NVM writes could partially be written or reordered by the memory controller in the face of system failure.

1) DATA INCONSISTENCY IN FACE OF SYSTEM FAILURE

Inserting a KV node to an NVM-based persistent skiplist may cause data inconsistency. There are four inconsistent crash cases when inserting a 2-level node 25 shown in Figure 2. Invalid-key node means real KV data is partially written

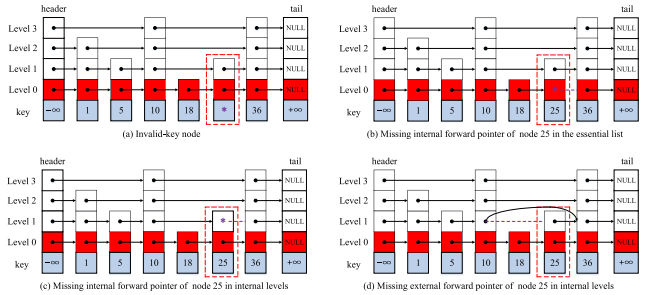


FIGURE 2. Four crash inconsistent cases of a skiplist when inserting key 25.

shown in Figure 2(a). Missing the internal forward pointer of node 25 in the essential list could cause previous existing node 36 cannot be found in Figure 2(b). Figure 2(c) shows that an uninitialized internal forward pointer of node 25 in internal levels could point to an illegal address. These three cases bring wrong results, so we must repair them after a system failure. Figure 2(d) shows that missing external forward pointer in internal levels only violates the semantics of the skiplist and affects the search efficiency.

Therefore, memory write orders must be carefully ensured [19] for consistency's sake. To guarantee consistency between non-volatile memory and volatile CPU caches for an NVM-based persistent skiplist, we must ensure the ordering of memory writes in some efficient ways.

2) WRITE PROBLEMS IN NVM-BASED CONSISTENT SKIPLIST

Current processors typically provide ordered memory write operations such as the memory fence instruction (MFENCE) and the cache line flush instruction (CLFLUSH) to ensure data consistency [18], [20]–[23]. CLFLUSH is used to flush a dirty cache line back to memory. An MFENCE instruction guarantees the previous store instruction before this MFENCE persisted earlier than the store instruction after this MFENCE. Unfortunately, these CLFLUSH and MFENCE instructions are quite expensive to implement and execute [20], [21], [24], because they incur significant execution overhead that is proportional to the number of NVM writes [20], [24]. For granular updates larger than 8 bytes, the traditional approach ensures data consistency through double-write mechanisms such as redo or undo logging. The logging mechanism first stores the new data (redo logging) or old data (undo logging) into a log area in NVM and then updates this data in place. However, the logging mechanism introduces additional NVM writes, adversely affecting NVM lifetime and system performance [9], [18], [20]. Therefore, a traditional logging-based skiplist guarantees data consistency in the face of system failure but further destroy NVM lifetime.

D. MOTIVATION

Logging-based persistent skiplist guarantees the data consistency but introduces double NVM writes, which can degrade the lifetime of NVM. To obtain data consistency of NVM-based persistent skiplist and decrease NVM writes, we propose two write-optimized and consistent skiplists for NVM. One exploits log-free failure-atomic writes for both the essential list and internal levels of skiplist to avoid double logging writes. The other employs selective consistency (SC) combined with log-free failure-atomic writes to reduce data consistency overhead and the number of NVM writes. The selective consistency mechanism ensures the consistency of the 0th level (essential list) of the skiplist and rebuilding internal levels of the skiplist with small-scale reconstruction.

III. THE DESIGN OF AS AND ASCS

In this section, we describe the design of the Atomic Skiplist and Atomic and Selective Consistency Skiplist, which are write-optimized and consistent NVM-based persistent skiplists.

A. DESIGN GOALS

Based on the above observations, our design goals are:

- **Reduce NVM writes and cache line flush numbers through log-free failure-atomic write.** Log-free failure-atomic write (Atomic) is an 8-byte atomic write ordered with CLFLUSH and memory fence (MFENCE) to avoid double writes of the redo/undo logging. We implement write operations (insert and delete) of NVM-based persistent skiplist with log-free failure-atomic writes to obtain an NVM-based consistent skiplist called Atomic Skiplist (AS for short).
- **Reduce consistency overhead by a selective consistency mechanism over log-free failure-atomic writes.** Selective consistency (SC) ensures the consistency of the 0th level of the NVM-based persistent skiplist with log-free failure-atomic writes and not guarantees that of the internal levels of skiplist with no MFENCE and CLFLUSH instructions. We implement the write operations with Atomic and SC can further reduce NVM writes and CLFLUSH numbers to obtain an NVM-based low-consistency persistent skiplist called Atomic and Selective Consistency Skiplist (ASCS for short).

B. THE OVERVIEW OF ATOMIC SKIPLIST

Atomic Skiplist (AS for short) uses log-free failure-atomic writes for both the 0th level (i.e., essential list) and internal levels of NVM-based persistent skiplist. As shown in Figure 3, for a k-level skiplist (here k is 4), AS guarantees the consistency of the 0th level using log-free failure-atomic writes (Atomic). AS also ensures the consistency of the 1st to (k-1)-th levels by Atomic.

1) OPERATIONS

In this section, we introduce the write operations in AS stemming from insertion and deletion. For simplicity, we use the clflush_with_mfence function to ensure the log-free failure-atomic write of the data in NVM, which is executed by

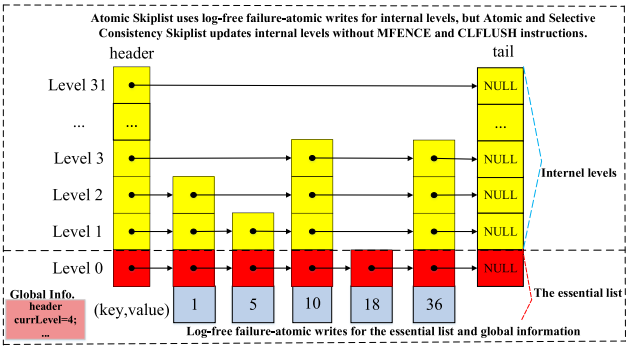


FIGURE 3. The layout of AS and ASCS.

a CLFLUSH instruction then followed by an MFENCE instruction.

Insertion: For inserting a new key-value pair, for example, (25, value), AS first find the node vector update[i] of the positions to be inserted in every level from *currLevel-1* to 0, as shown in Figure 4(a). We use a random number generator to generate a 2-level node 25. We first persist the key-value pair,i.e., (25, value) in NVM with the clflush_with_mfence function shown in Figure 4(b); Second, we copy the 0th level's the forward pointer of the node 18 (P18.1) to the 0th level's the forward pointer of the node 25 (P25.1) and persist P25.1 to NVM with log-free failure-atomic write shown in Figure 4(c). Then we copy the address of node 25 to P18.1 and persist P18.1 with the clflush_with_mfence function shown in Figure 4(d). AS updates other internal levels of persistent skiplist from bottom to top with log-free failure-atomic writes like level 0 shown in Figure 4(e).

Deletion: For example, for deleting a key-value pair, key 25, AS first searches the node to be deleted according to the key. If node 25 exists, AS first finds the two external forward pointers point to node 25 from up to bottom in Figure 5(a). AS updates the internal levels of persistent skiplist from top to bottom with CLFLUSH and MFENCE instructions shown in Figure 5(b), then updates the essential list with CLFLUSH and MFENCE instructions (log-free failure-atomic writes) demonstrated in Figure 5(c). After persisting the essential list, we then free the node 25 shown in Figure 5(d).

Recovery: AS uses log-free failure-atomic writes to ensure consistency of the essential list and internal levels of NVM-based persistent skiplist. Hence, its recovery process is easy to implement from either normal shutdown or system failures.

There is a regular shutdown flag in AS, a valid flag (e.g., 1) stands for normal shutdown, and a false flag stands for system failure. Because AS is all stored in NVM, we can quickly recover it without any more operations when the system is normal shutdown. However, the recovery of AS is more complicated when system failures. Since the insertion and deletion operations of AS may cause the data inconsistent problem, we take the insertion operation as an example to demonstrate the recovery process of AS. As shown in Figure 4(b), we first persist the key-value pair with CLFLUSH and MFENCE instructions. If a system failure occurs in this process, AS may partially update the key-value entry to NVM. It doesn't affect the consistency of AS because AS cannot find the partially updated key-value entry. As shown in Figure 4(c) and Figure 4(d), then we strictly control the ordering of forwarding pointer updates of the essential list in AS to guarantee the 0th-level consistency of AS. When a system failure occurs between Figure 4(c) and Figure 4(d), the new key-value pair also cannot found in the essential list of AS, so it was not to affect the data consistency of AS. If a system failure occurs after Figure 4(d) but before Figure 4(e), we can replay the process of updating other internal levels of AS after the consistency of the essential list is guaranteed. Since AS ensures the consistency of internal levels like the essential list with CLFLUSH and MFENCE instructions shown in Figure 4(e), AS can quickly recover the consistency of internal levels. In this way, AS can recover to a consistent state when system failures.

C. THE OVERVIEW OF ATOMIC AND SELECTIVE CONSISTENCY SKIPLIST

Atomic and Selective Consistency Skiplist (ASCS for short) uses log-free failure-atomic writes only for the essential list but updates the internal levels without MFENCE and CLFLUSH instructions. ASCS uses selective consistency (SC) scheme to divide the NVM-based skiplist into two groups for different consistency requirements, i.e., the 0th-level of skiplist (the essential list) is consistent, and the internal levels of skiplist are persistent. Still, ASCS can reconstruct it by the consistent 0th-level of skiplist in the face of system failures. As shown in Figure 3, for a k-level skiplist (here k is 4), ASCS guarantees the consistency of the essential list using log-free failure-atomic writes (Atomic) and rebuilds internal levels through the consistency of the essential list of skiplist.

1) OPERATIONS

Insertion: The insertion of ASCS is very similar to that of AS when inserting a 2-level node 25 shown in Figure 4. Since ASCS only needs to ensure the skiplist's essential list's consistency, the only difference is that ASCS adjusts other internal levels of the skiplist like level 0 but without the clflush_with_mfence function.

Deletion: The deletion of ASCS is very similar to that of AS when deleting a 2-level node 25 shown in Figure 5. Compared with Atomic Skiplist, the only difference is that ASCS updates the internal levels of persistent skiplist from top to bottom without CLFLUSH and MFENCE instructions.

Recovery: Since the essential list of ASCS is consistent, rebuild-from-consistent is sufficient to recover an ASCS from either normal shutdown or system failures. When inserting a 2-level node 25, the recovery process of ASCS is very similar to that of AS. ASCS changes other internal levels for

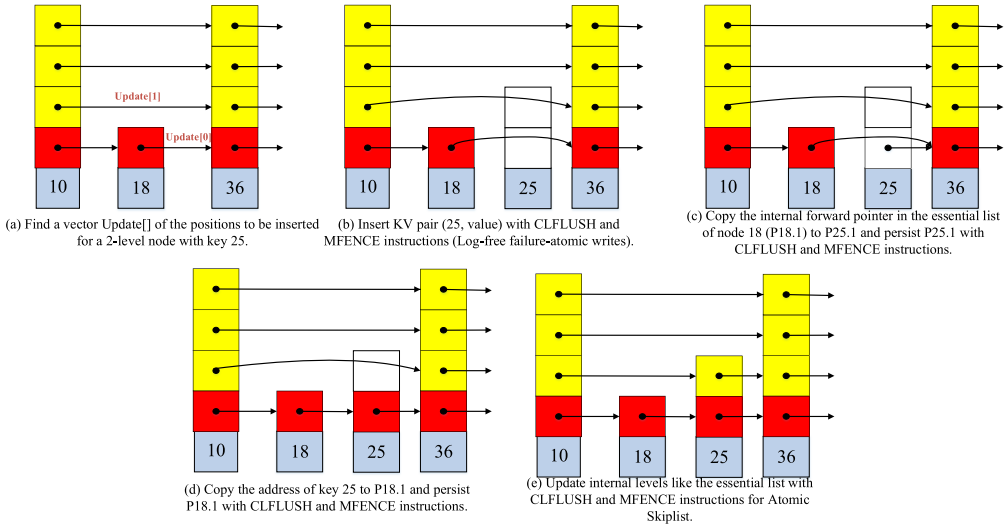


FIGURE 4. The insertion algorithm of Atomic Skiplist when inserting a 2-level node 25.

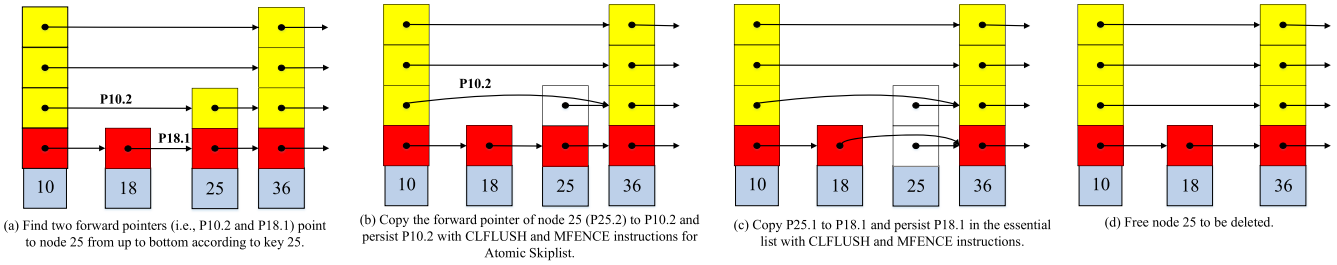


FIGURE 5. The deletion algorithm of Atomic Skiplist when deleting a 2-level node 25.

no consistent requirements, so ASCS needs more time than AS to recover the internal levels from the consistency of the essential list of skiplist.

The insertion, query, and deletion of a particular key-value node require an average time complexity of O(log(N)), where N is the number of nodes in the skiplist. AS can significantly reduce NVM writes to avoid the double writes of write-ahead logging. Compared with AS, ASCS can further reduce NVM writes by SC, so ASCS can further extend the lifetime of NVM.

IV. EVALUATION

This section evaluates the number of cache line flush per insert and micro-benchmark latency and macro benchmark throughput for our proposed write-optimized and consistent skiplists.

A. EXPERIMENTAL SETUP

We use the HP Labs Quartz simulator [25] to simulate NVM by adding extra delay in the CLFLUSH operation. Table 1 shows our server configurations.

For comparison, we implemented a Redo-Logging-based scheme for consistent and persistent Skiplist (RLS for short),

TABLE 1. Server configurations.

CPU	Intel Xeon E5-2630 V3, 2.4 GHz
CPU cores	8
Processor cache	32KB/256KB/20MB, L1/L2/L3 cache
NVM	100 GB
Operating system	Ubuntu 16.04, kernel version 4.15.0

and a non-consistency version of persistent skiplist for the PM to get better performance purpose called No Barrier and No Flush Skiplist (NBNFS for short). Our consistent skiplists for persistent memory include Atomic Skiplist (AS for short) and Atomic and Selective Consistency Skiplist (ASCS for short). We use four micro-benchmarks such as Random Number, Document Word, Fingerprint, and Sequence Number to test the number of CLFLUSH count per key-value pair insert operation and the latency of insert and delete operation. We also use two macro-benchmarks such as INSERT and DELETE datasets to test the throughput of different skiplists.

RandomNum: The dataset has 2^{24} keys and each key has 16 bytes. This dataset is generated by a random function and widely used in previous works [18], [26]. We use random keys in this dataset as the keys of the skiplist's entries.

Docword: The dataset has five text collections, and we use the biggest collection, PubMed abstracts, for evaluation. PubMed abstracts include 8.2 million documents and a total of 730 million words [27]. We use document IDs and word IDs in conjunction as the keys of the skiplist's entries.

Fingerprint: The dataset extracted from MacOS included daily snapshots of the Mac OS X server collected by the Stony Brook University File System and Storage Lab [28]. We use the 16-byte MacOS's MD5 fingerprints of data chunks as keys of entries in the skiplist.

SequenceNum: The dataset contains 2^{24} keys, and each key is a sequential number from 0 to 2^{24} - 1. We set this dataset to observe the possible difference between the Skiplist_Log and our consistent skiplists between RandomNum and SequenceNum.

INSERT: The dataset contains 24 million keys, and each key is Zipfian generated from YCSB workload-A in the load phase.

DELETE: The dataset is the same as the INSERT dataset. We only change all insert operations of the INSERT dataset to delete operations for the DELETE dataset.

The NVM read/write latency is set to 200/600 nanosecond. All experimental results are the average of 5 independent runs.

B. THE NUMBER OF CLFLUSH PER INSERT

We study the number of cache line flush (CLFLUSH) per insert to test the write efficiency of different skiplists. The number of CLFLUSH per insert is higher means that the write efficiency per insert is lower. In these experiments, we first insert items into skiplists until the load number reaches the predefined value (e.g., 2/4/8 million items). Then we use the total number of CLFLUSH divided by the predefined value to calculate the average number of CLFLUSH of requesting an item. Figure 6 shows the average number of CLFLUSH of inserting an item. As shown in the figures, RLS has the highest number of CLFLUSH because RLS uses a redo-logging mechanism to guarantee data consistency which takes two NVM writes (one for logging first, another for write in-place). Compared with RLS, our consistent skiplists AS and ASCS significantly degrade the average number of CLFLUSH of inserting an item. Compared with RLS, AS reduces the number of CLFLUSH per insert by 67.5% because AS uses log-free failure-atomic write method (Atomic) to avoid double-write to NVM and only write the NVM with ordered MFENCE and CLFLUSH instructions. Compared with RLS and AS, ASCS reduces the number of CLFLUSH per insert by 75% and 33.3% respectively, because ASCS adopts Selective Consistency (SC) based on Atomic to further reduce the number of CLFLUSH in the cost of increasing recovery overhead. Therefore, the write-optimized efficiency of AS and ASCS have higher than that of RLS.

C. INSERT LATENCY

We compare the average insertion latency of different skiplists, as shown in Figure 7. In all experiments, we first insert items into skiplists until the load number reaches the predefined value (e.g., 2/4/8 million items). After that, we insert 10,000 items into the skiplist, then delete 10,000 items from the skiplist. Eventually, we calculate the average insertion latency of requesting an item. For 4 million items, compared with RLS, AS and ASCS reduce the average insertion latency of requesting an item by 32.3% - 40.9% and 36.2% - 54.2% in four different workloads, respectively. There are similar insertion rules for 2 million items and 8 million items. Compared with non-consistent persistent skiplist NBNFS, AS and ASCS obtain the data consistency in the cost of increasing some insertion latency. Atomic Skiplist (AS for short) has lower insertion latency than RLS because AS uses a log-free failure-atomic write mechanism to avoid double write overhead of the redo-logging method. Compared with AS, Atomic and Selective Consistency Skiplist (ASCS for short) uses selective consistency based on the failure-atomic write mechanism to further degrade the average insertion latency by up to 22.5%, because ASCS can reduce the number of CLFLUSH at the cost of sacrificing the performance of rebuilding internal levels. To insert a 2-level node 25 in figure 4 as an example, Table 2 shows the differences between AS and ASCS in the reconstructed internal levels.

TABLE 2. The differences between AS and ASCS in the reconstructed internal levels.

Skiplist type	Recovery scale in internal levels
AS	Only restore internal and external forward pointers of node 25.
ASCS	Rebuild all pointers in internal levels of the ASCS.

D. DELETE LATENCY

As shown in Figure 8, we compare the average deletion latency of requesting an item for different skiplists. Similar to insertion operation, we delete 10,000 entries in different skiplists then calculate the average latency of deleting an item. For 4 million items, compared with RLS, AS and ASCS decrease the average deletion latency by 38.7% - 47.1% and 44.9% - 57.8% in four different workloads, respectively. There are similar deletion rules for 2 million items and 8 million items. Redo-Logging consistent Skiplist (RLS for short) has the highest deletion latency because it mainly has double write overhead caused by the redo-logging mechanism; AS has a lower deletion latency than RLS because it uses Atomic to avoid double write to NVM. ASCS employs Atomic and Selective Consistency (SC) to further reduce the average deletion latency by up to 20.4% compared with AS because ASCS reduces the consistency overhead of deletion operation at the cost of the additional overhead of rebuilding internal levels. Compared with NBNFS, AS and ASCS obtain data consistency at the expense of additional deletion latencies.

E. EFFECT OF NVM WRITE LATENCY

Various NVM has different write latencies. To test the effect of diverse NVM write latencies on skiplists, we set all

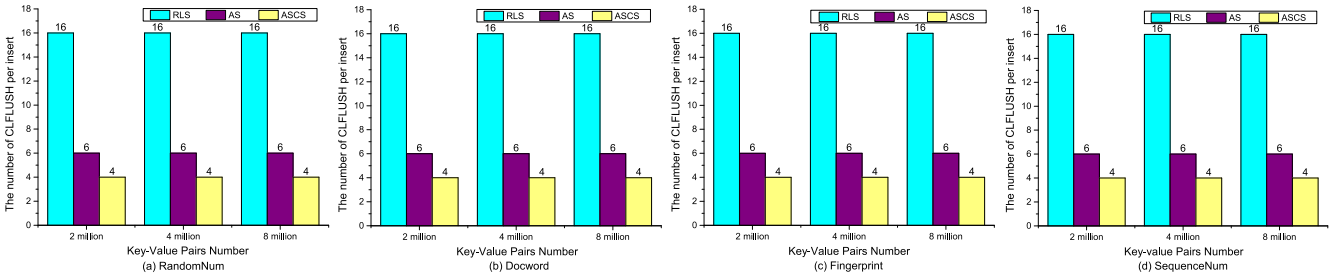


FIGURE 6. The average number of CLFLUSH when inserting a key-value entry. Each result is an average of five different runs.

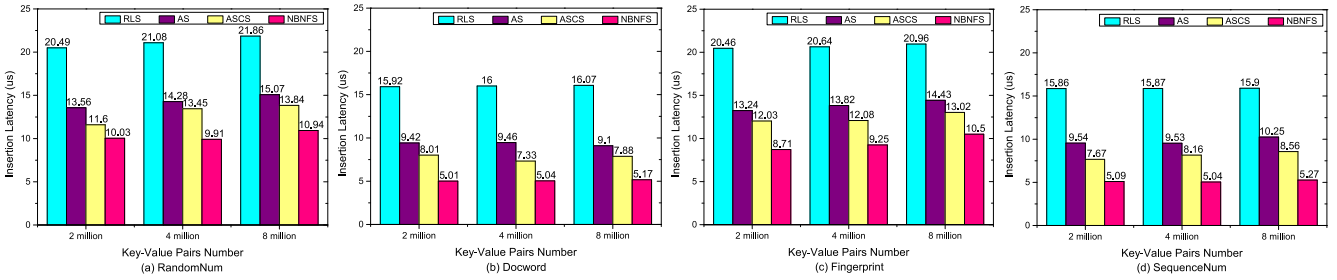


FIGURE 7. Results of the average insertion latency when inserting a key-value entry.

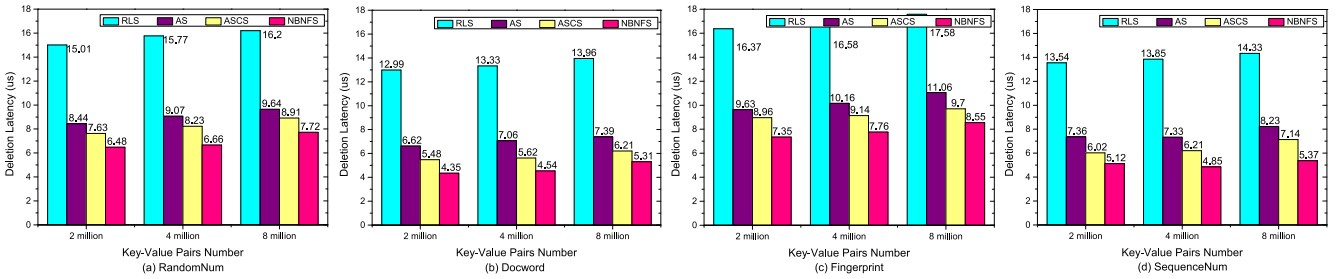


FIGURE 8. Results of the average deletion latency when deleting a key-value entry.

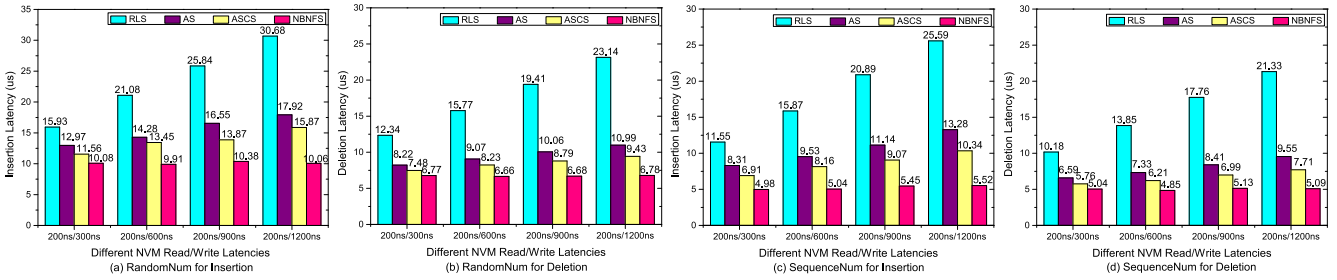


FIGURE 9. The average insertion and deletion latency of a request with different NVM write latencies (Load number is 4 million items).

NVM read latency to 200 nanoseconds, and set its write latency to 300, 600, 900, and 1200 nanoseconds, respectively. We insert or delete 10,000 items to skiplists when they have loaded 4 million items and calculate the latency of a request. We choose two micro-benchmark (e.g., RandomNum and SequenceNum) to observe how the random and sequential characteristics of the key affect different skiplists, especially RLS. Figure 9 shows the average insertion and deletion latency for four different skiplists with various NVM write latencies. The increase in NVM write latency has led to an increase in the insertion and deletion latency of skiplists, but the write optimization of skiplists we proposed is better than RLS. When the NVM write latency is 300ns, compared with RLS, AS and ASCS reduce the average insertion latency by 18.6% and 27.4% in Figure 9(a) and 28.1% and 40.2% in Figure 9(c), respectively; and when the NVM write latency

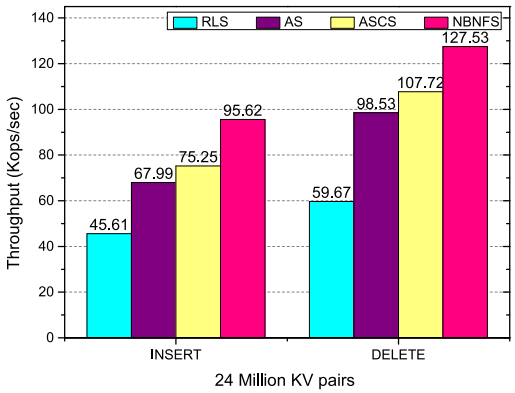


FIGURE 10. Results of the average throughput when requesting 24 million items.

is 1200ns, compared with RLS, AS and ASCS decrease the average insertion latency by 41.6% and 48.3% in Figure 9(a) and 48.1% and 59.6% in Figure 9(c), respectively. The deletion latency of skiplists has a similar rule as the insertion latency of skiplists shown in Figure 9(b) and Figure 9(d). The insertion and deletion latency in the sequential feature of keys (SequenceNum) lower than that in the random characteristic of keys (RandomNum) for all skiplists.

F. PERFORMANCE OF YCSB WORKLOADS

As shown in Figure 10, we compare the average insertion and deletion throughput of different skiplists. We generate two YCSB workloads, e.g., the INSERT dataset and the DELETE dataset. The generative method of these two datasets can be seen in the experimental setup section (e.g., IV(A)). To test the insertion and deletion throughput of skiplists, we record the time of inserting 24 million key-value items into skiplists, then record the time of deleting these 24 million items from skiplists. Finally, the insertion and deletion throughputs of different skiplists are calculated by dividing 24 million by the corresponding insertion and deletion time. Figure 10 shows the insertion and deletion throughput of skiplists in YCSB 24 million key-value (KV) items. Compared with RLS, AS and ASCS increase the average insertion throughput by 49.1% and 65.0%, and improve the average deletion throughput by 65.1% and 80.5% respectively. Because AS uses log-free failure-atomic write to avoid the double write to NVM of RLS, and ASCS takes the SC mechanism on the top of log-free failure-atomic write to further reduce the number of rights to NVM. Therefore, ASCS outperforms AS by 10.7% and 9.3% in insertion and deletion throughputs, respectively.

V. RELATED WORK

A. NVM-BASED TREE STRUCTURES

Most previous studies focus on adapting persistent B-Tree or B+Tree or radix tree to NVM [17], [20], [21], [24], [29], [30]. NVM-based persistent B+Tree and skiplist have different characteristics, so they face diverse challenges. NVM-based B-Tree or B+Tree has to shift half entries of a node on average when inserting a new KV entry, while skiplist has better locality and no need to move existing items. To guarantee data consistency, shifting each entry requires multiple CLFLUSH and MFENCE instructions. Moreover, B-Tree or B+Tree may suffer split or merge operation for re-balance, which causes more NVM writes and reduces insertion or deletion performance. The wB+Tree [21], and NV-Tree [20] employ unsorted leaf nodes to avoid moving entries, so they significantly reduce the number of NVM writes. NV-Tree [20], FPTree [30] and ROART [31] leverage selective consistency or persistence to reduce data consistency overhead. This method inspires our work. FAST_FAIR [17] uses Failure-Atomic ShifT (FAST) and Failure-Atomic In-place Rebalance (FAIR) for node split/merge operation to avoid expensive duplicate copy operations, which inspire our log-free failure-atomic writes used for NVM-based persistent skiplist.

B. NVM-BASED SKIPLIST

NoveLSM [32], SLM-DB [33] and MatrixKV [34] place persistent skiplist of log-structure merge tree (LSM-Tree)in NVM to reduce serialization and compaction overhead. NV-Skiplist [35] is an NVM-DRAM hybrid consistent skiplist whose internal levels are stored in DRAM and primary essential list stored in NVM. However, NV-Skiplist requires more time to recover to a consistent state when system failures due to the volatility of the internal index of a large-scale skiplist. Inspired by TSU [36], we improve the analysis of the inconsistency cases of skiplist when system failures. In this paper, we propose Atomic Skiplist (AS for short) and Atomic and Selective Consistency Skiplist (ASCS for short), which are orthogonal to previous works, and strives to explore the consistency of skiplist stored only in NVM without DRAM participant. Chen *et al.* [37] propose a persistent skiplist for the Intel Optane DC Persistent Memory.

VI. CONCLUSION

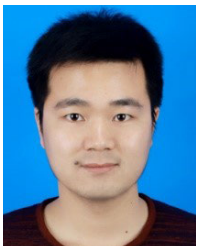
In this paper, we present two NVM-based write-optimized and consistent skiplists called AS and ASCS, for systems with all skiplists stored in only a persistent NVM environment. AS employs log-free failure-atomic writes to significantly reduce NVM writes of traditional Redo-Logging-based Skiplist (RLS for short). AS and ASCS both guarantee data consistency when system failures. Compared with RLS, experimental results demonstrate that AS and ASCS reduce the number of cache-line flushes (CLFLUSH), decrease the insertion and deletion latencies, and increase the insertion and deletion throughputs. We also evaluate the write efficiencies of AS and ASCS in different NVM write latencies for various NVM technologies.

REFERENCES

[1] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[2] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Piscataway, NJ, USA, Apr. 2013, pp. 256–267.

[3] M. Mao, Y. Cao, S. Yu, and C. Chakrabarti, "Optimizing latency, energy, and reliability of 1T1R ReRAM through appropriate voltage settings," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD)*, Piscataway, NJ, USA, Oct. 2015, pp. 359–366.
[4] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the Intel Optane DC persistent memory module," 2019, *arXiv:1903.05714*. [Online]. Available: http://arxiv.org/abs/1903.05714
[5] L. George, *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. Newton, MA, USA: O'Reilly Media, 2011.
[6] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.
[7] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 265–276, Oct. 2014.
[8] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, New York, NY, USA, 2014, p. 15.
[9] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 91–104, Mar. 2011.
[10] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2009, pp. 14–23.
[11] W. Pugh, "Skip lists: A probabilistic alternative to balanced trees," *Commun. ACM*, vol. 33, no. 6, pp. 668–676, Jun. 1990.
[12] M. A. Bender, M. Farach-Colton, R. Johnson, S. Mauras, T. Mayer, C. A. Phillips, and H. Xu, "Write-optimized skip lists," in *Proc. 36th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst.*, New York, NY, USA, May 2017, pp. 69–78.
[13] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 24–33, Jun. 2009.
[14] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA, Feb. 2013, pp. 282–293.
[15] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Dec. 2009, pp. 14–23.
[16] Y. Lu, J. Shu, and L. Sun, "Blurred persistence in transactional persistent memory," in *Proc. 31st Symp. Mass Storage Syst. Technol. (MSST)*, Piscataway, NJ, USA, May 2015, pp. 1–13.
[17] D. Hwang, W.-H. Kim, Y. Won, and B. Nam, "Endurable transient inconsistency in byte-addressable persistent b+-tree," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, Berkeley, CA, USA, 2018, pp. 187–200.
[18] X. Zhang, D. Feng, Y. Hua, J. Chen, and M. Fu, "A write-efficient and consistent hashing scheme for non-volatile memory," in *Proc. 47th Int. Conf. Parallel Process.*, New York, NY, USA, Aug. 2018, p. 87.
[19] F. Xia, D. Jiang, J. Xiong, and N. Sun, "HiKV: A hybrid index key-value store for DRAM-NVM memory systems," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Berkeley, CA, USA, 2017, pp. 349–362.
[20] J. Yang, Q. Wei, C. Chen, C. Wang, K. L. Yong, and B. He, "NV-Tree: Reducing consistency cost for NVM-based single level systems," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, Berkeley, CA, USA, 2015, pp. 167–181.
[21] S. Chen and Q. Jin, "Persistent b+-trees in non-volatile main memory," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 786–797, Feb. 2015.
[22] P. Zuo, Y. Hua, and J. Wu, "Write-optimized and high-performance hashing index scheme for persistent memory," in *Proc. USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Berkeley, CA, USA, 2018, pp. 461–476.
[23] M. Nam, H. Cha, Y.-R. Choi, S. H. Noh, and B. Nam, "Write-optimized dynamic hashing for persistent memory," in *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*, Berkeley, CA, USA, 2019, pp. 31–44.
[24] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell, "Consistent and durable data structures for non-volatile byte-addressable memory," in *Proc. FAST*, Berkeley, CA, USA, vol. 11, 2011, pp. 61–75.
[25] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li, "Quartz: A lightweight performance emulator for persistent memory software," in *Proc. 16th Annu. Middleware Conf.*, R. Lea, S. Gopalakrishnan, E. Tilevich, A. L. Murphy, and M. Blackstock, Eds. Vancouver, BC, Canada: ACM/IFIP, 2015, pp. 37–49.
[26] P. Zuo and Y. Hua, "A write-friendly hashing scheme for non-volatile memory systems," in *Proc. MSST*, Piscataway, NJ, USA, Dec. 2017, pp. 1–10.
[27] D. Newman. (2008). *Bags of Words Data Set*. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Bag+of+Words
[28] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Berkeley, CA, USA, 2012, pp. 261–272.
[29] S. Chen, P. B. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *Proc. CIDR*, 2011, p. 5.
[30] I. Oukid, J. Lasperas, A. Nica, T. Willhalm, and W. Lehner, "FPTree: A hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory," in *Proc. Int. Conf. Manage. Data*, New York, NY, USA, Jun. 2016, pp. 371–386.
[31] S. Ma, K. Chen, S. Chen, M. Liu, J. Zhu, H. Kang, and Y. Wu, "ROART: Range-query optimized persistent ART," in *Proc. 19th USENIX Conf. File Storage Technol. (FAST)*, Berkeley, CA, USA, 2021, pp. 1–16.
[32] S. Kannan, N. Bhat, A. Gavrilovska, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Redesigning LSMs for nonvolatile memory with NoveLSM," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, pp. 993–1005, 2018.
[33] O. Kaiyrakhmet, S. Lee, B. Nam, S. H. Noh, and Y.-R. Choi, "SLM-DB: Single-level key-value store with persistent memory," in *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*, 2019, pp. 191–205.
[34] 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 matrix container in NVM," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 17–31.
[35] Q. Chen and H. Yeom, "Design of skiplist based key-value store on non-volatile memory," in *Proc. IEEE 3rd Int. Workshops Found. Appl. Self Syst. (FASW)*, Sep. 2018, pp. 44–50.
[36] S. Wang and Q. Cao, "TSU: A two-stage update approach for persistent skiplist," in *Proc. Conf. Adv. Comput. Archit.* Kunming, China: Springer, 2020, pp. 163–177.
[37] C. Chen, J. Yang, M. Lu, T. Wang, Z. Zheng, Y. Chen, W. Dai, B. He, W.-F. Wong, G. Wu, Y. Zhao, and A. Rudoff, "Optimizing in-memory database engine for AI-powered on-line decision augmentation using persistent memory," *Proc. VLDB Endowment*, vol. 14, no. 5, pp. 799–812, Jan. 2021.



RENZHI XIAO received the B.E. degree in software engineering from the Jiangxi University of Science and Technology, Nanchang, China, in 2013. He is currently pursuing the Ph.D. degree with the Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST), Wuhan, China. His current research interests include in-memory key-value store, non-volatile memory, and NVM-based data structures.

DAN FENG (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively. She is currently a Professor and the Dean of the School of Computer Science and Technology, HUST. She has over 100 publications in major journals and international conferences, including the IEEE TC, the IEEE TPDS, ACM-TOS, FAST, USENIX ATC, EuroSys, ICDCS, HPDC, SC, ICS, IPDPS, DAC, and DATE. Her research interests include computer architecture, non-volatile memory technology, distributed and parallel file systems, and massive storage systems. She is a member of the Association for Computing Machinery and the Chair of the Information Storage Technology Committee, Chinese Computer Academy. She served on the program committees of multiple international conferences, including SC, in 2011 and 2013, and MSST, in 2012 and 2015.

YUCHONG HU (Member, IEEE) received the B.Eng. degree in computer science and technology from the Special Class for the Gifted Young (SCGY), University of Science and Technology of China (USTC), in 2005, and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), in 2010. He has published 12 papers as the first/corresponding author in conferences FAST (twice), INFOCOM, SoCC, and journals TOS, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON NETWORKING, *TIT*, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He also published more than 40 articles in major journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, ATC, *DSN*, MSST, IWQoS, SRDS, ICPP, ICPADS, ISPA, *ISIT*, and ICC. His research mainly focuses on designing and implementing intelligent reliability mechanisms, based on fault-tolerance, such as network coding or erasure coding, to improve reliability, performance, and security for storage systems, which include cloud storage, big-data storage, deduplicated backup, heterogeneous/hierarchical storage, and in-memory NoSQL database.

FANG WANG (Member, IEEE) received the B.E. and master's degrees in computer science and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994, 1997, and 2001, respectively. She is currently a Professor of computer science and engineering with HUST. She has more than 80 publications in major journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *ACM TACO*, *SC*, MSST, DATE, HiPC, ICDCS, HPDC, ICCD, ICDE, and ICPP. Her research interests include distribute file systems, parallel I/O storage systems, and graph processing systems.

XUELIANG WEI received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 2015, where he is currently pursuing the Ph.D. degree. His current research interests include non-volatile memories and computer architecture.

XIAOMIN ZOU received the B.E. degree in digital media technology from Nanchang University, China, in 2017. She is currently pursuing the Ph.D. degree in computer architecture with the Huazhong University of Science and Technology (HUST), China. Her research interests include in-memory key-value store, non-volatile memory, and NVM-based data structures.

MENGYA LEI received the B.E. degree in electronic information engineering technology from the Wuhan University of Science and Technology (WUST), China, in 2017. She is currently pursuing the Ph.D. degree in computer architecture with the Huazhong University of Science and Technology (HUST), China. Her research interests include non-volatile memory (NVM) and NVM-based operating systems.