Efficient Sanitization Design for LSM-based Key-Value Store over 3D NAND Flash

Liang-Chi Chen[1,6], Shu-Qi Yu[1,7], Chien-Chung Ho[1,2,5], Wei-Chen Wang[4,8], Yung-Chun Li[4,9] and Kun-Chi Chiang[3,4,10]
[1] Department of Computer Science and Information Engineering, National Chung Cheng University
[2] Department of Computer Science and Information Engineering, National Cheng Kung University
[3] Department of Computer Science and Information Engineering, National Tsing Hua University
[4] Macronix Emerging System Lab, Macronix International Co., Ltd.
{[5]ccho, [6]u07410027,[7]u07410081}@cs.ccu.edu.tw, {[8]raymondwang, [9]monixslee}@mxic.com.tw,[10]sailfish0814@gmail.com

***Abstract*—Key-value stores based on Log-Structured Merge-Trees (LSM trees) are widely adopted in large-scale environments to meet the characteristics of various modern data-intensive applications. However, the LSM tree-based key-value can encounter serious data security issues when there arises the need for data sanitization. The data sanitization requires to meet the needs of completely removing the sensitive data from the storage devices physically. Since deleting data in LSM Tree is by inserting a delete mark to the specified key, it leaves several out-of-date values to the specified key. This situation can deteriorate when the LSM tree-based key-value stores are deployed on the 3D NAND flash-based storage devices (i.e., SSDs). This work explores the efficient sanitization design for LSM-based key-value store over 3D NAND flash memories to address the above issues. To simultaneously maintain the performance of LSM tree and improve the sanitization efficiency, we focus on integrating the processes of key-value pair updating and the execution of sanitization by exploiting our proposed influence-conscious programming method. In addition, hot-cold separation and switching pool wear leveling policies are proposed to improve space utilization and lifetime. The capability of the proposed design is evaluated by a series of experiments, for which we have very encouraging results.**

***Keywords*—**

I. Introduction

In recent years, key-value stores based on Log-Structured Merge-Trees (LSM trees) [17] are widely adopted in large-scale environments to meet the characteristics of various modern data-intensive applications, such as LevelDB, RocksDB, and HBase [7]–[9]. The success of LSM tree-based key-value store credits to its excellent write performance and the efficient operation designs, e.g., insertions, point lookups, and range queries. Although the design of LSM trees significantly improves efficiency of storage devices, it also leads to new challenges on data security. As data security has risen to be one of the most critical concerns of computer professionals, data sanitization techniques have been proposed to meet the needs of completely removing the sensitive data from the storage devices physically. However, deleting data in LSM Tree is by inserting a delete mark to the specified key, and it thus leaves several out-of-date values to the specified key. Those out-of-date data are still obtainable on the storage device before they are deleted by compaction operations, and thus leave the risk of being stolen and referenced illegally. The situation could deteriorate further as LSM tree-based key-value stores are deployed on the 3D NAND flash-based devices, i.e., solid-state drives (SSDs)[1]. Although lots of existing researches have addressed the sanitization designs for NAND flash-based devices, they do not take properties of LSM trees over 3D NAND flash-based devices into consideration. As a result, existing approaches cannot simultaneously maintain the performance of LSM tree management and guarantee the sanitization efficiency. Such observations motivate this work to explore an efficient sanitization design for LSM-based key-value stores over 3D NAND flash. We aim at enabling the sanitization feature for 3D NAND flash-based LSM tree designs while the performance and sanitization efficiency can be maintained.

As more and more users gradually store and access their data from the remote locations, it leads to urgent needs on exploring stricter data management designs. To meet legal compliance requirements of protecting data from unauthorized uses and sanitizing (or secure deleting) data, the concept of sanitization is proposed to meet the needs of completely removing the sensitive data form the storage devices physically. Since flash memory is lack of *in-place-update* feature, it is not trivial to realize sanitization on flash-based storage systems. Thus, to enable the sanitization features over NAND flash-based devices, lots of excellent works had been proposed in past decades. Those works can be roughly categorized into three types, i.e., *erasure-based*, *encryption-based*, and *overwriting-based* methods, according to their core ideas. The erasure-based methods [4], [18] sanitize the target data immediately by invoking the garbage collection (GC) processes and executing erase operations on the corresponding physical flash blocks. It however results in the live-page-copying overhead caused by moving out valid data before erasing the block that contains to-be-sanitized data. The encryption-based methods sanitize the target data by destroy small-size encryption keys since the data can be recovered with the ruined keys [2], [13]. While encryption-based methods are adopted to resolve the performance issue, it leads to the additional overhead on the key maintenance and thus brings the new challenges. The overwriting-based methods [6], [12], [21] aim at improving sanitization efficiency by reusing page programming to sanitize the previously-written data immediately. The main idea is to reprogram the same page and break the original V_t distribution of flash cells. These approaches will introduce additional program disturbance to any adjacent pages and thus worsen the reliability problem which cannot be ignored especially for 3D flash memories with more disturbance directions. In addition, some researchers proposed to securely and efficiently sanitize data by mixing different types of techniques [5], [14], [15], [20]. These mixed-type sanitization methods strive to improve the sanitization efficiency by either combining encryption-based and erasure-based methods or integrating erasure-based and overwriting-based methods.

This research is motivated by the raised security issues when the LSM tree-based key-value stores are deployed on the 3D NAND flash-based storage devices (i.e., SSDs). The objective of this work is to enable an efficient sanitization

[1]To simplify the discussion, this paper focuses on the 3D NAND flash memory. Hereafter, we refer to NAND flash memory and flash memory interchangeably when there is no ambiguity

feature for LSM-based key-value store over 3D NAND flash memories. To simultaneously maintain the performance of LSM tree and improve the sanitization efficiency, we propose a novel influence-conscious programming method to integrate the processes of key-value pair updating and the execution of sanitization. In addition, with the support of our proposed influence-conscious programming method, a hot-cold separation data allocation method and a switching pool wear leveling policy are proposed to improve space utilization and lifetime. We further evaluated the feasibility and capability of the proposed design through a series of experiments, for which we have very encouraging results.

II. Background and Research Motivation

An LSM-tree is an efficient indexing structure designed for executing the high rate of insertion and deletion operations over HDD-based key-value store applications. To effectively unleash the sequential performance of HDDs, the conventional LSM-tree defers and batches data writes into large chunks to eliminate the performance overhead caused by random writes [17]. To fully utilize the parallelism of SSDs, a separating structure of LSM-tree, named *WiscKey*, is proposed to educe the write by eliminating the needs of value compaction, as shown in Figure 1. *WiscKey* proposed to separate the value from the key-value entry [16]. While the value is inserted into a value log, the key with value offset pair is inserted into the existing LSM-tree. When it needs to insert a new key-value pair into an LSM-tree, the inserted corresponding value is appended to an on-SSD sequential value log, so as to enable recovery in case of a crash. After that, the key and value address pair is added to the in-memory MemTable, which is sorted by keys; Once in-memory MemTable reaches its size limit, it will be switched to immutable MemTable and written to on-SSD $Level\ 0$, as an SSTable. When any level $Level\ k$ reach its size limit, a "compaction" procedure with applying the merge sort will be triggered to merge SSTable files in between the two adjacent levels ($Level\ k$ and $Level\ k+1$). Simlar to the process of inserting a key-value pair, deleting a data in LSM Tree is procedded by inserting a delete mark to the specified key. As a result, the mechnism of insert, deletion, and compaction of LSM-tree inherently cause many out-of-date values to the specified key over the storage devices, as shown in Figure 1. This situation can deteriorate when the LSM tree-based key-value stores are deployed on the 3D NAND flash-based storage devices, i.e., SSDs. The out-of-date values can lead to the security and performance issues on SSDs.

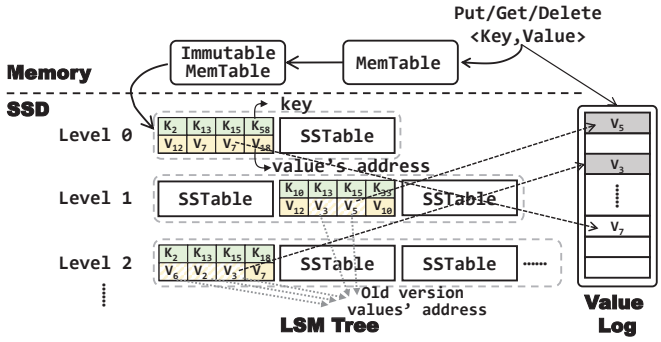


Fig. 1. Existing LSM-tree design can leave several out-of-date values to the specified key over SSDs.

As the security becomes one of the most important concerns in enterprise storage applications, it is neccearry to explore an efficient and effective sanitization design for LSM-based key-value store over 3D flash-based SSDs. With taking the inherently high rate of insertion/deletion on LSM tree and SSD's characteristics into consideration, the conventional rewriting-based and erasure-based sanitization methods are not suitable for LSM-based key-value store over 3D flash-based SSDs. The additional live-page-copying or migration overhead is generated to avoid the disturbance issue, and these management processes would compelete I/O bandwidth with the normal management and execution of LSM tree. Therefore, the *instantaneous sanitization*, which exploits the disturbance effect to sanitize the adjacent pages [21], shows its great potential to enable the sanitize feature for LSM-based key-value store over 3D flash-based SSDs. Different from past works which adopt ISPP to program cells' V_t distribution into the narrow and non-overlapped shape, the instantaneous sanitization deliberately create a certain number of disturbance-induced errors on a flash page in the very beginning stage while it is programmed. The main idea is to speed up the accumulation of disturbance-induced errors on a flash page. With the purposely generated overlapping, the page data can be instantaneously sanitized with being exactly once disturbed when any of its adjacent pages are programed. Santizing the out-of-date However, the instantaneous sanitization is designed based on 3D SLC flash memories and cannot properly function when it is deployed on 3D MLC flash memories. As illustrated in Figure 2, if the idea is directly adopted to deliberately create a certain number of disturbance-induced errors on a 3D MLC flash page in the very beginning stage while it is programmed, the 3D MLC flash page will suffer from the serious retention error issue when time expires. Different from the wide available V_t window of 3D SLC flash memory, the available V_t window of 3D MLC flash memory is much narrower and easy to accumulate a large number of error bits when cells' V_t distribution is left-shifted.

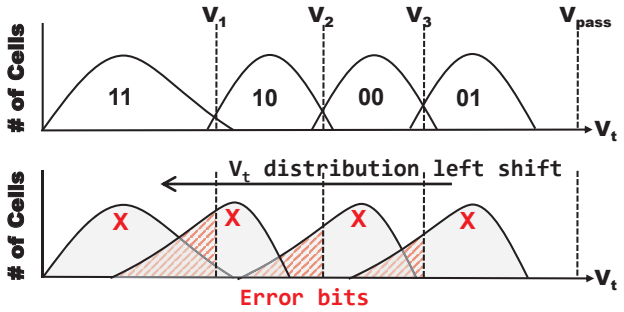


Fig. 2. Directly adopting the instantaneous sanitization [21] could lead to the serious retention error issue when time expires.

This work is inspired by the challenges of exploring an efficient and effective sanitization design for LSM-based key-value store over 3D MLC NAND flash memories. To achieve the highest degree of data security/sanitization and realize an efficient sanitizable LSM-tree design, we aim at proposing a novel *influence-conscious programming* (reffered to as ICP) method which simultaneously considers the property of instantaneous sanitization method [21] and V_t distribution characteristics of 3D MLC flash memories. To realize the targeted efficient sanitizable LSM-tree design, The technical problems then fall on (1) how to design the programming method, i.e., the proposed ICP method, for key-value pair insertion by jointly considering the disturbance property of 3D MLC flash memories and management of LSM trees, (2) how to property control the data allocation to create the sanitization-friendly data pattern for efficiently utilizing the proposed ICP method and decreasing the additional overhead, e.g., capacity utilization, and (3) how to address the wear unbalancing issue while the proposed design is adopted. In the following section, we shall propose to consider the properties of LSM tree and 3D MLC flassh memory (e.g., MLC, TLC, and QLC) jointly. Particularly, an efficient sanitizable LSM-tree design, which consists of an *influence-conscious programming* method, a hot-

cold separating data allocation method, and a swtich-pool wear leveling policy, will be presented.

III. Efficient Sanitizable LSM-tree Design

A. Design Overview

In this section, an *efficient sanitizable LSM-tree design* will be proposed to realize data security for LSM-tree-based key-value stores over 3D MLC NAND flash memory. Specifically, we will smartly leverage the programming disturbance for achieving highly efficient data sanitization design for LSM trees. It is worth noting that, the proposed design in this work aims at achieving the highest degree of data security for LSM-tree-based key-value stores, i.e., the old version of each value in LSM-tree shall be sanitized immediately when the new version of value is updated. With this objective, we are able to eliminate the conventional management overheads on tracking the old version(s) of data.

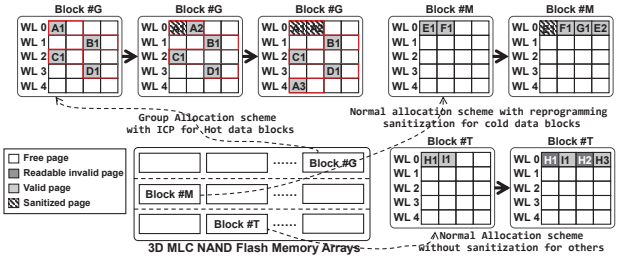


Fig. 3. The structure and working flow of proposed efficient sanitizable LSM-tree design.

We first identify that it is more important to sanitize values instead of keys in LSM-trees, since the keys are only used for indexing and the values (in the value log file) are the genuine concern for data security. Thus, we propose to store keys and values of LSM-tree in separated regions (i.e., value region and key-structure region) of 3D MLC NAND flash, and will only sanitize values when any data update comes. Figure 3 demonstrates that the value (resp. key-structure) region stores the values (resp. keys and structure) in our proposed efficient sanitizable LSM-tree design. The advantages of proposed key-value separation are two-folds. First, there is no need to have extra keys for garbage collection (GC)[2] as in the conventional LSM trees [16]. Second, we do not need GC threads either.

However, even with the proposed key-value separation, the problem of how to efficiently sanitize values in LSM trees still remains to be solved. In the following sections, we will present three methods comprised in our proposed efficient sanitizable LSM-tree design. Firstly, in order to achieve efficient data sanitization for the values of LSM tree, we propose an *influence-conscious programming method* from the perspective of device level in Section III-B. In particular, we propose to efficiently sanitize values of LSM tree by exploiting the programming disturbance in 3D MLC NAND flash memories. In Section III-C, from the perspective of system level, we propose a *hot-cold separating data allocation method* to enhance not only sanitization/system performance but also space utilization over 3D MLC NAND flash. Eventually, a *switching-pool wear leveling method* is proposed to address the wear-out issue in our proposed design in Section III-D. Note that, our proposed design can also be adopted on other multi-level-cell (e.g., TLC or QLC) NAND flash memories with limited mechanism modification.

[2]Here, the garbage collection denotes the collection procedure in LSM trees, rather than that in NAND flash memories.

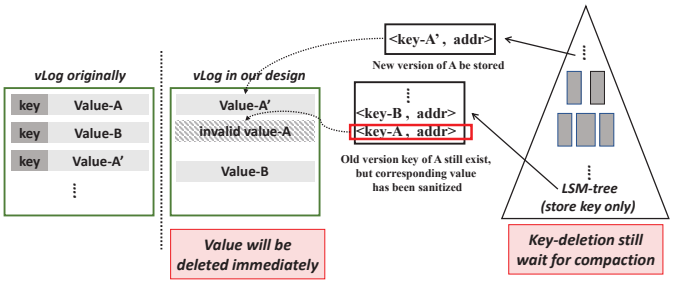
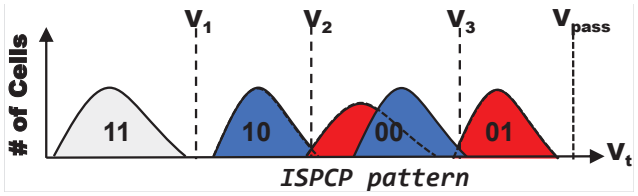


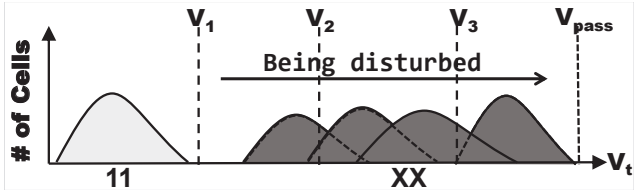
Fig. 4. Sanitzable LSM-tree

B. Influence-Conscious Programming (ICP) Method

As we mentioned in Section II, there is an urgent need for efficient data sanitization over high-density 3D NAND flash, such as MLC, TLC and QLC. In order to address this issue, we propose an influence-conscious programming (ICP) method. Our goal is to leverage the programming disturbance in 3D NAND flash for achieving high-performance data sanitization. However, compared to 3D SLC NAND flash, 3D MLC NAND flash suffers from the narrow V_t window. As a result, data retention error would become a huge problem in 3D MLC NAND flash. To deal with the problem, we propose to first identify the "slow" and "fast" cells in NAND pages[3]. Once we successfully identify the fast/slow cells, we are able to construct an ideal V_t distribution that can not only achieve efficient sanitization but also mitigate the data retention problem over 3D MLC NAND. Figure 5(a) and Figure 5(b) illustrate the V_t distributions with adopting our proposed ICP method and after being sanitized/disturbed, respectively. The reason why such V_t distribution is an appropriate pattern for enabling efficient sanitization over 3D MLC NAND is that fast cells are programmed to the right in each V_t window, and slow cells are programmed to the left in each V_t window. By this mean, we could decrease the probability of retention error since fast cells are difficult to be erred beyond the capability of ECC, and also maintain the effectiveness of efficient sanitization since slow cells are still easy to be disturbed.



(a) After being programmed.



(b) After being sanitized/disturbed.

Fig. 5. V_t distribution of our proposed influence-conscious programming (ICP) method.

We then propose six steps to achieve the ideal V_t distribution, as shown in Figure 6. Note that, instead of the conventional ISPP approach [11], we adopt the one-shot programming

[3]Due to the imperfection in the process of 3D NAND manufacture, the tendency degree of a programmed/disturbed flash cell could be different [3]. In this paper, the cells that more easy (resp. difficult) to be programmed/disturbed are referred to as fast (resp. slow) cells.

approach to program data (and separate hot/cold cells) in our proposed ICP method [10], so that the write latency can be enhanced. In the first step, we separate the fast flash cells from the initial state to the right of state '10' with relatively low programming voltage, as shown in Figure 6. Secondly, we further separate the fastest flash cells from the state '10' to the right of state '00'. To this end, we have identified the fast cells and program them into the desired states. In the following steps, we propose to adopt the similar approach on slow cell, and thus we can derive the slow and slowest cells and program them to the appropriate states (i.e., states '01' and '00') with low programming voltage, as shown in Figure 6. Eventually, we successfully conduct the desired V_t distribution which could enable efficient and reliable data sanitization for LSM trees over 3D MLC NAND flash.

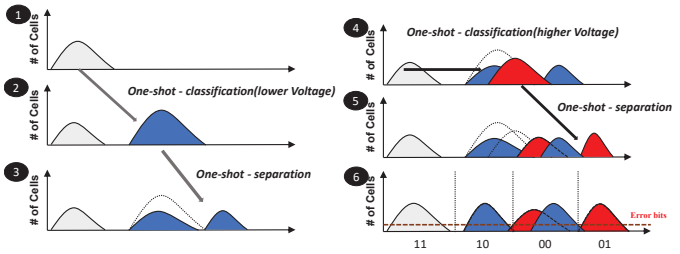
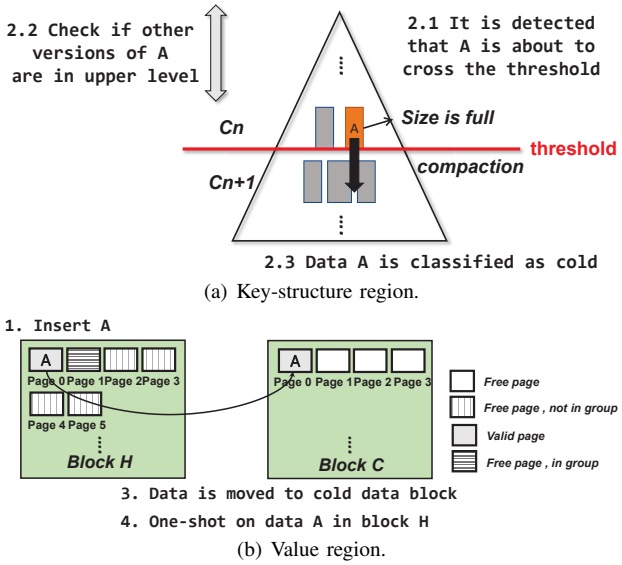


Fig. 6. Steps of ICP.

C. Hot-Cold Separating Data Allocation Method

With having our proposed efficient influence-conscious programming method from the device-level perspective, we further propose a hot-cold separating data allocation method from the system-level perspective. The objective of this method is to appropriately allocate data in the value region with high sanitization efficiency and low management overhead. First, we adopt the proposed ICP method to program only hot data. For the cold data, they are programmed and sanitized by the general programming method (i.e., ISPP) and deletion method (i.e., garbage collection), respectively. The rationale behind this design is that hot data dominates most of the update and cold data rarely updates in LSM trees. In other words, hot data which usually generate multiple versions of data is the crucial security concern for key-value storage, so adopting the ICP method over hot data could make our system more secure with high sanitization efficiency. On the contrary, cold data commonly have few version, and thus adopting the conventional ISPP and garbage collection methods to program and sanitize them can obtain the decent balance between performance and management overhead. However, due to the different characteristics of hot/cold data, how to identify hot/cold data and also manage them properly but differently then becomes an important issue. As a result, we propose to identify hot/cold data by wisely leveraging a typical built-in function in LSM trees, i.e., compaction. Originally, compaction is used to improve read performance and space utilization by continuously recycling old data and merging multiple layers into one. In our proposed method, we first assume every keys (and also their corresponding values) in an LSM tree as hot data initially. Besides, we set a hot-cold threshold δ. When the level of a datum (key) in an LSM tree becomes deeper than the threshold δ, it will be categorized from hot to cold, as shown in Figure 7(a). Contrarily, if a datum is originally cold and then being updated, the datum will be categorized from cold to hot.

Our proposed hot-cold separating data allocation method can be better explained with the example in Figure 7. Figure 7 shows an example flow of hot-cold separating data allocation method in both key-structure and value regions. Assume that



(a) Key-structure region.

(b) Value region.

Fig. 7. An example flow of hot-cold separating data allocation method.

the hot-cold threshold δ is between the layer C_n and C_{n+1}, and the key of data A is allocated in the layer C_n and categorized as hot data. As shown in Figure 7(a), at the beginning, if a compaction process is triggered, and key A is compacted from layer C_n to layer C_{n+1}. We then check if any other version of key A exists in the upper layers. After checking, we classify data A from hot to cold if there is no other version in the upper layers. Once key A is identified as cold data, we trigger the data migration to re-allocate value A from the hot block (i.e., Block H) to the cold block (i.e., Block C), as shown in Figure 7(b). Besides, from guaranteeing the highest degree of data security, value A in Block H will also be sanitized by the one-shot sanitization [10].

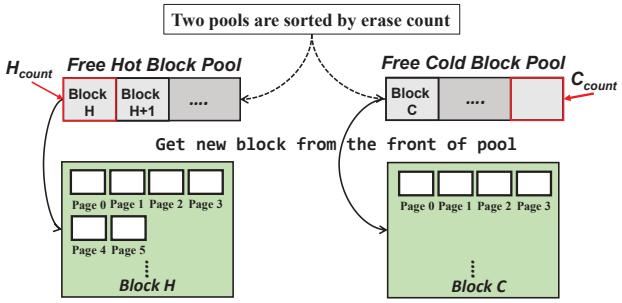


Fig. 8. An example of the switching-pool wear leveling.

D. Switching-Pool Wear Leveling

As we divide the flash blocks into hot blocks and cold blocks for storing hot/cold data separately, it would incur the wear-out problem since the program/erase times of hot and cold blocks can be largely different. A switching-pool wear leveling is proposed to resolve the wear-out problem in our proposed design. We first design two pools, i.e., a hot pool and a cold pool; the hot pool (resp. cold pool) stores all the *sorted* program/erase count of hot blocks (resp. cold blocks) in descending order. Besides, H_{count} (resp. C_{count}) denotes the erase count of the hottest (resp. coldest) block in the hot (resp. cold) pool, and a threshold Th denotes the threshold value of wear leveling. In the background idle period, we periodically check whether the value difference between H_{count} and C_{count}. If the value difference is larger than Th, which means the hot/cold difference degree of the NAND flash

is too large and may arise the wear-out problem, we will trigger the switching-pool wear leveling process to the two blocks (i.e., the hottest and coldest blocks). In particular, we will migrate the block with H_{count} to the cold pool, and migrate the block with C_{count} to the hot pool. By this mean, the "overheated" block can be cooled down and the "overcold" block can be heated up, and thus the wear-out problem can also be resolved.

Algorithm 1: Switching-Pool Wear Leveling

```
Input: Th: Threshold Value of Wear Leveling
Input: H_count: Erase Count of the Coldest Block in Hot Pool
Input: C_count: Erase Count of the Hottest Block in Cold Pool
// If the difference between the erasure times of the
   two pools is too large
if H_count − C_count > Th then
    switch_pool();
end
```

IV. Experiment

A. Experiment Setup

In this section, we conducted a series of experiments to evaluate the feasibility and efficiency of our proposed efficient sanitizable LSM-tree design. Our proposed approach and the compared approaches are implemented and evaluated in MQSim [19]. A 3D MLC NAND flash memory is adopted as the experimental device, and its parameters are as shown in Table I. Noting that the SPWL threshold T is 30%, which denotes that the switching process in the switching-pool wear leveling method will be triggered when the difference between the erase count of the hottest and coldest blocks is larger than the endurance limitation of 3D NAND. In the experiments, we implement five different approaches for various kinds of comparison, i.e., *Erase, Scrubbing, ICP, ICP w/ hot-cold, ICP w/ hot-cold and SPWL*. *Erase* denotes adopting the conventional built-in erase function in 3D NAND to sanitize data whenever the data update request comes. *Scrubbing* denotes adopting the scrubbing operation [6] to sanitize data. *ICP* denotes adopting our proposed influence-conscious programming to program and sanitize data. *ICP w/ hot-cold* and *ICP w/ hot-cold and SPWL* denote programming and sanitizing data by adopting our proposed influence-conscious programming with only hot-cold separating data allocation method and with both hot-cold separating data allocation and switching-pool wear leveling methods, respectively. Note that, to simplify the discussion, a page-level address mapping (i.e., FTL [1]) is adopted on all the approaches in the experiments.

TABLE I. 3D MLC NAND PARAMETERS.

Chip size	4GB	Block size	1MB
Page size	4KB	Read latency	75μs
Write latency	750μs	Erase latency	3.8ms
Scrubbing latency	750μs	SPWL threshold T	30%

For our experiments, the objective is to evaluate the system performance, space utilization and lifetime of the proposed efficient sanitizable LSM-tree design. Thus, we first measure the average sanitization performance and overall performance to verify the effectiveness and efficiency of our proposed design. Moreover, the space utilization is also evaluated. Last but not least, we evaluate the endurance of our proposed design. To simulate various applications that may have different proportions of hot/cold data in the real world, we generate four workloads with different types of hot/cold proportion (i.e., 10%:90%, 20%:80%, 30%:70% and 50%:50%) as our data benchmarks.

B. Experiment Results

Figure 9 shows the sanitization performance of the four investigated sanitzation methods under four different workloads. The x-axis denotes the four sanitization methods, and the y-axis denotes the sanitization performance, in terms of the average sanitization time. It is first observed that the sanitization performance of erase is extremely poor under every condition, since the erase operation usually causes abundant of live-page copying. On the contrary, compared to the scrubbing method, our proposed ICP method can improve the sanitization performance by $8.3\times$, $8.6\times$, $8.6\times$ and $8.2\times$ under the hot-cold ratio of 1:9, 2:8, 3:7 and 5:5, respectively. It is because our proposed ICP method sanitize data by exploiting programming disturbance, and would not incur any additional programming process or live-page copying.

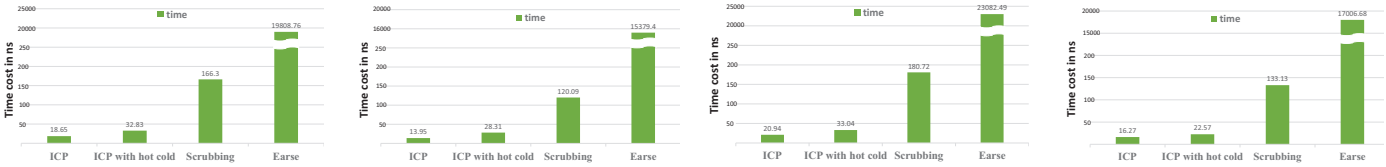
Figure 10 demonstrates the overall performance of the four investigated sanitzation methods under four different workloads. The x-axis denotes the four sanitization methods, and the y-axis denotes the overall performance, i.e., the average read/write/sanitization time. Overall, the trend of overall performance is similar to that of sanitization performance. That is, the overall performance of erase operation is still the worst, and our proposed ICP method is able to outperform the scrubbing method by $3.13\times$, $3.22\times$, $3.07\times$ and $3.08\times$ under the hot-cold ratio of 1:9, 2:8, 3:7 and 5:5, respectively. It is worth noting that our proposed ICP method with adopting the hot-cold separating data allocation method may slightly decrease the sanitization and overall performance, but it would be advantageous for its high space utilization.

Figure 11 shows the capability of our proposed switching pool wear leveling policy. The x-axis denotes the four sanitization methods, and the y-axis denotes the lifetime which is normalized to that of scrubbing-based approach. It is found that our proposed switching pool wear leveling policy can significantly improve the lifetime of 3D MLC NAND flash devices, and its performance is very closed to that of scrubbing-based approach. Compared to ICP method without applying our proposed switching pool wear leveling policy, the lifetime can be further improved by $2.6\times$, $1.7\times$, $1.56\times$, and $1.25\times$ under the hot-cold ratio of 1:9, 2:8, 3:7 and 5:5, respectively.

Finally, we illustrate the space utilization analysis when applying the proposed hot-cold separating data allocation method in Figure 12. The x-axis denotes different hot/cold data ratio, and the y-axis denotes the wasted space utilization over 3D MLC NAND flash memory. The wasted space utilization is defined as the ratio of unavailable space to all the space, where the unavailable space includes the all the valid page, all the invalid pages, and those allocated pages that cannot be used due to the ICP striction. It is found that the proposed ICP with hot-cold separating data allocation method can significantly reduce the wasted space by 43.5%, 34.5%, 42.5%, and 24.3% respectively.

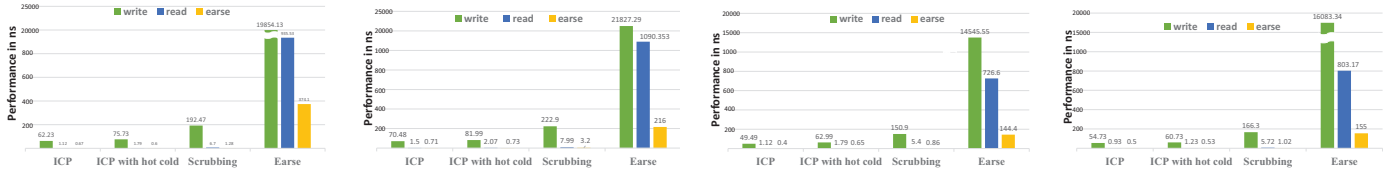
V. Conclusion

To achieve data security for 3D MLC NAND flash memory devices, this work proposes new program strategy - ICP which has a high sanitization performance and reliable retention. Because of characteristic of separating k-v LSM-tree, we can easily apply ICP only on value region to meet data security requirement on LSM-tree based DB. In this work, we also provide Hot-Cold Separation and Switching Pool Wear Leveling to improve memory space utilization and endurance. To sum up, this work has a high reliability and feasibility, beacause we just have to apply ICP skillfully on those specific blocks and use the allocation and data management scheme indiscriminately. The result proves that this work can fulfill high performance and high privacy at the same time. It just takes only about 20% of scrubbing cost time and even much lower(0.17%) than erase-based sanitization cost time. This work reaches an achievement about providing low overhead sanitization on MLC for sparating k-v LSM-tree stroage.



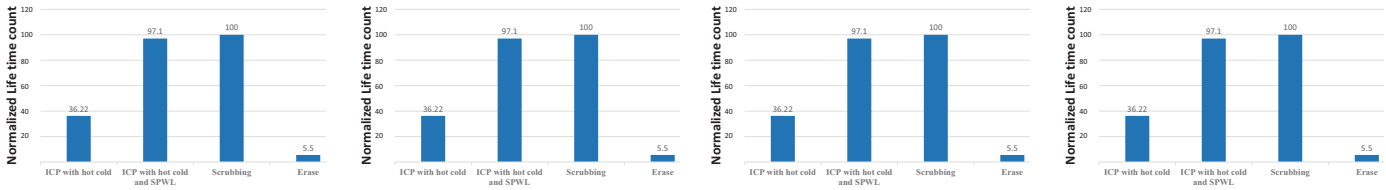
(a) 1:9 (b) 2:8 (c) 3:7 (d) 5:5

Fig. 9. Average sanitization performance under different workloads.



(a) 1:9 (b) 2:8 (c) 3:7 (d) 5:5

Fig. 10. Overall performance under different workloads.



(a) 1:9 (b) 2:8 (c) 3:7 (d) 5:5

Fig. 11. Endurance results.

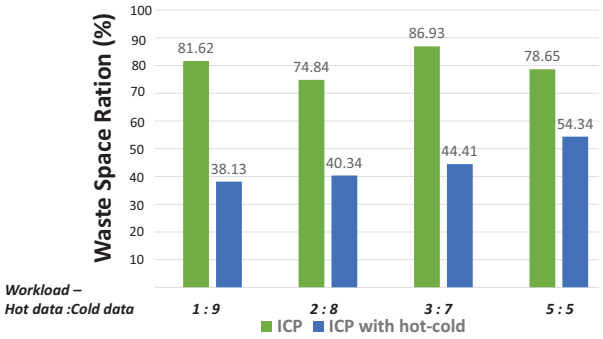


Fig. 12. Space utilization analysis.

REFERENCES

[1] A. Ban. Flash File System. US Patent 5,404,485. In *M-Systems*, April 1995.

[2] A. M. Braga and A. H. G. Colito. Adding secure deletion to an encrypted file system on android smartphones. In *Proc. SECURWARE*, pages 106–110, 2014.

[3] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu. Data retention in mlc nand flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 551–563, 2015.

[4] S. Diesburg, C. Meyers, M. Stanovich, M. Mitchell, J. Marshall, J. Gould, A.-I. A. Wang, and G. Kuenning. Trueerase: Per-file secure deletion for the storage data path. In *Proceedings of the 28th annual computer security applications conference*, pages 439–448, 2012.

[5] C. L. et al. Erasucrypto: A light-weight secure data deletion scheme for solid state drives. In *Proceedings on Privacy Enhancing Technologies*, 2017.

[6] M. W. et al. Reliably erasing data from flash-based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11. USENIX Association.

[7] Facebook. Rocksdb, 2015.

[8] S. Ghemawat and J. Dean. Leveldb, 2011.

[9] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Analysis of {HDFS} under hbase: a facebook messages case study. In *12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*, pages 199–212, 2014.

[10] C.-C. Ho, Y.-C. Li, Y.-H. Chang, and Y.-M. Chang. Achieving defect-free multilevel 3d flash memories with one-shot program design. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[11] C.-C. Ho, Y.-C. Li, P.-H. Lin, W.-C. Wang, and Y.-H. Chang. A stride-away programming scheme to resolve crash recoverability and data readability issues of multi-level-cell flash memory. In *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 67–72, 2018.

[12] S. Jia, L. Xia, B. Chen, and P. Liu. Nfps: Adding undetectable secure deletion to flash translation layer. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, pages 305–315, 2016.

[13] S. Jia, L. Xia, B. Chen, and P. Liu. Deftl: Implementing plausibly deniable encryption in flash translation layer. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2217–2229, 2017.

[14] H. A. Khouzani, C. Liu, and C. Yang. Architecting data placement in ssds for efficient secure deletion implementation. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2018.

[15] B. Li and D. H. Du. Tasecure: Temperature-aware secure deletion scheme for solid state drives. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pages 275–278, 2019.

[16] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Wisckey: Separating keys from values in ssd-conscious storage. *ACM Transactions on Storage (TOS)*, 13(1):1–28, 2017.

[17] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[18] S. Swanson and M. Wei. Safe: Fast, verifiable sanitization for ssds. *San Diego, CA: University of California-San Diego*, 2010.

[19] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu. Mqsim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 49–66, 2018.

[20] M. Wang, J. Xiong, R. Ma, Q. Li, and B. Jin. A novel data secure deletion scheme for mobile devices. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, 2018.

[21] W.-C. Wang, P.-H. Lin, Y.-C. Li, C.-C. Ho, Y.-M. Chang, and Y.-H. Chang. Toward instantaneous sanitization through disturbance-induced errors and recycling programming over 3d flash memory. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.