LaVA: An Effective Layer Variation Aware Bad Block Management for 3D CT NAND Flash

Shuhan Bai
Huazhong University of Science and Technology
City University of Hong Kong

You Zhou, Fei Wu*, Changsheng Xie
Huazhong University of Science and Technology

Tei-Wei Kuo
National Taiwan University
Mohamed bin Zayed University of Artificial Intelligence

Chun Jason Xue
Mohamed bin Zayed University of Artificial Intelligence

***Abstract*—3D NAND flash with charge trap (CT) technology has been developed by stacking multiple layers vertically to boost storage capacity while ensuring reliability and scalability. One of its critical characteristics is the large endurance variation among and inside blocks and layers. With this feature, traditional bad block management (BBM), which determines block lifetime by the page with worst endurance, results in underutilization of solid state drive (SSD) usage. In this paper, a layer variation aware and fault-tolerant bad block management, named LaVA, is proposed to prolong the lifetime of 3D NAND flash storage. The relevant layer, instead of the entire flash block, is discarded at a finer granularity when a page failure is encountered. Experimental results based on real-world workloads show that LaVA can significantly extend the endurance of 3D CT NAND flash (30.6%-62.2%) with a small performance degradation (less than 10% increase of tail I/O response time), compared to the conventional technique.**

***Index Terms*—bad block management, 3D charge trap NAND flash, endurance, reliability**

I. Introduction

Planar NAND flash is experiencing scaling limits, hindering storage capacity growth with low cost due to an increased raw bit error rate (RBER) as feature sizes shrink and more information bits squeeze into a single cell [12]. As an alternative, through stacking memory cells vertically (in Z direction with multiple layers), 3D structures can boost density, lower price-per-bit, and guarantee reliability since adequate feature sizes [9]. Nowadays, charge trap (CT) technology and triple-level cell (TLC) become mainstream, offering better scalability, higher storage capacity, and considerable reliability [16], [26]. However, the endurance of 3D NAND flash memory is constrained.

Three types of errors, read/program/erase errors, occur with flash memory. Repeated program/erase (P/E) cycles increase RBER as the solid state drive (SSD) wears out, leading to uncorrectable errors and lifetime loss [20]. Bad block management (BBM) detects and retires blocks whose reliability cannot be guaranteed to maximize SSD usage while ensuring reliability and performance. To extend SSD's endurance, [14], [15] propose machine learning models to predict the optimal time for retiring a block, tackling the lifetime challenge due to 3D flash memory's process variations. [25] suggests managing bad blocks with cluster similarity to also consider SSD's reliability. Nevertheless, all of these research works focus on block or cluster granularity, and the "big block" problem of 3D flash memory exacerbates the efficiency of bad block management.

One critical characteristic of 3D CT NAND flash is its large endurance variation [24]. Through careful chip characterization, we find different layers show significant endurance variances (called cross-layer variation) while flash pages in the same layer have similar endurance (called intra-layer concentration). With these features, the block-granular naïve bad block management, which determines lifetime by the unit with worst endurance, contributes to storage capacity underutilization. Based on this observation, we propose an effective layer variation aware and fault-tolerant bad block management named LaVA. It discards only the relevant layer at fine granularity rather than the entire block when a page failure happens, providing significant improvements in storage capacity utilization and endurance extension for a wide range of application scenarios.

LaVA faces two challenges. The first is usage generality and flexibility. To break granularity limitations of both FTL- and retire-level implementations, lazy marking to merely update the layer status bitmap and keep other tables invariant whenever a retirement, whose size is decided by the introduced layer failure threshold, happens is presented. The second is the impact on I/O performance under LaVA. During a retirement, valid data within bad layers needs to be migrated, which causes read/write (r/w) bursts that interfere with user I/O. To this end, we propose delayed migration executed when the device is idle.

In summary, this paper makes the following contributions:

- We characterize reliability features of 3D CT NAND flash and summarize critical observations into intra-layer concentration and cross-layer variation.
- We develop an efficient layer-aware and fine-grained flash management technique, LaVA, to optimize storage capacity utilization and prolong flash lifetime. Additionally, cooperation with a layer failure threshold provides flexibility and balance between endurance and performance.
- We simulate LaVA with real-world workloads. Results show that LaVA can significantly improve SSD lifetime (30.6%-62.2%) with a slight performance degradation, compared to the traditional bad block management.

*Corresponding author: wufei@mail.hust.edu.cn

979-8-3503-4859-0/DATE24/© 2024 EDAA

II. Background and Motivation

A. 3D Charge Trap TLC NAND Flash

Mainstream flash manufacturers have turned to the charge trap technology [10] in their 3D NAND flash products. A CT cell adopts an insulating charge-trapping layer typically made of silicon nitride, providing better scalability and less coupling effects [16]. When tens of millions of cells are arranged together in a 3D structure, a NAND flash array can be formed. The storage medium comprises an array of word lines (WLs) continuously connected from top to bottom along the channel side. A WL is the basic unit of program operations, and each WL consists of three pages corresponding to the 3-bit data stored in a TLC cell. WLs on the same X-Y plane make up a layer, while those on the same X-Z plane belong to a string.

B. Traditional Bad Block Management

NAND flash blocks no longer reliable for storing and retrieving data are referred to as bad blocks. Flash devices can ship from factories with bad blocks that originated in the manufacturing process. More importantly, NAND flash memory suffers from limited endurance. As repeated program/erase operations gradually wear out cells' tunnel oxide, bit errors occur due to retention, fast de-trapping, and read/program disturb, arising data corruption and bad blocks. Mitigating or tolerating bit errors on these blocks often requires a much higher cost than benefits of using them. Therefore, flash-based storage systems employ bad block management to identify and retire bad blocks so that they will no longer jointly threaten data safety. Once any page fails, traditional BBM [21] marks the corresponding block as bad and considers all of its pages as failed, even those that could still be used. Then, all valid data within the block is migrated in background (i.e., when SSD controller is not busy).

C. Endurance Extension Challenges and Opportunities

To gain a better understanding of adopting 3D CT NAND flash characteristics in flash management techniques, specifically in the aspect of reliability enhancement, we conducted a motivational study to identify challenges and opportunities that have not yet been explored. The test data were obtained from our ARM- and FPGA-based experimental platform [24].

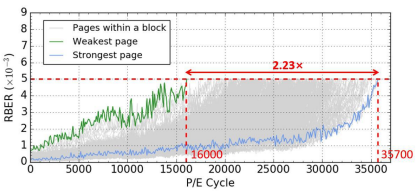


Fig. 1. Page lifetime distribution within a block.

Fig. 1 shows the page lifetime distribution within a block. Considering the efficiency and overhead issues, the capability of the error correction code (ECC) is limited. We use RBER, the fraction of bits that contain incorrect data before applying ECC, to represent ECC's error correction capability, and assume the maximum RBER that ECC can correct is 5×10^{-3} [2] (denoted by the horizontal red dotted line in Fig. 1). The RBER is calculated as follows:

$$RBER = \frac{Number\ of\ error\ bits\ per\ page}{Total\ number\ of\ bits\ per\ page} \quad (1)$$

Flash pages whose RBER exceeds 5×10^{-3} are regarded as bad pages. The lifetime of a page is denoted as the P/E cycle when it becomes bad. Page lifetime within a block has a notable difference. When reaching ECC's maximum error correction capability, the lifetime of the strongest page is 35700 P/E cycles, which is 2.23× of the weakest one (16000 P/E cycles).

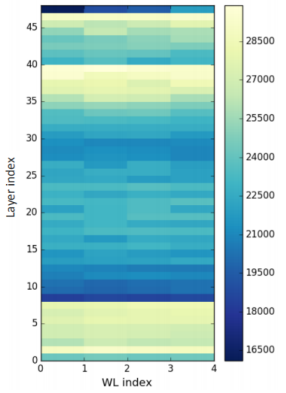


Fig. 2. Lifetime distributions of WLs within a block.

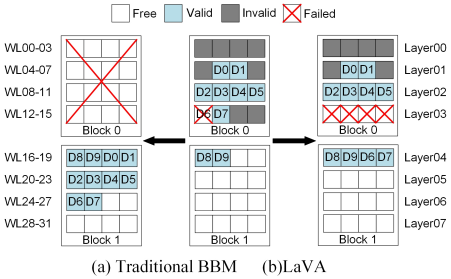
In the tested flash chip, one block contains 48 layers, each layer has 4 WLs, and each contains 3 pages. These three pages share one flash cell, therefore, if the number of errors within one of the pages goes beyond the capability of ECC, the entire WL in which the page is located is unhealthy, and the internal data needs to be migrated. The shortest page lifetime within a WL is viewed as its lifetime. Fig. 2 shows lifetime distribution of different WLs in a block, where a darker color indicates a shorter lifetime. Two features of WL failure occurrence exist. The first is *intra-layer concentration*. When a failed WL appears in one layer, other WLs in the same layer are likely to fail shortly. The second is *cross-layer variation*, i.e., large endurance variations exhibited among different layers, which is significant (up to 1.7x). The nature of intra-layer concentration is caused by the cross-layer variation in 3D CT NAND flash memory, which is related to the manufacturing process and layered voltage applied. Moreover, such phenomena may become more prominent as 3D structure stacks more layers.

Aforementioned results indicate that using traditional BBM in 3D structures cannot maximize flash lifetime and storage utilization. Large endurance variations among and inside blocks and layers cause storage capacity underutilization both spatially and chronologically. A deeper understanding and employment of medium characteristics is necessary for the efficiency of flash management techniques, leading to LaVA's development.

III. Design

To account for intra-layer concentration and cross-layer variation, we propose a layer variation aware bad block management, LaVA, to enhance storage utilization and extend flash endurance. We first provide LaVA's overview before delving into detailed implementation in read, write and erase operations.

A. LaVA Overall Design



(a) Traditional BBM (b)LaVA

Fig. 3. Traditional BBM and LaVA.

One page is regarded as failed if its RBER exceeds the maximum error correction capability. Instead of coarse-grained retirement, LaVA merely considers pages in the layer containing

excessive bit errors as failed and marks that layer as bad. Additionally, only valid data in that layer are migrated. Fig. 3 displays an example of LaVA's overall flow and compares it to the traditional BBM. For simplicity, only Block 0 and 1 are depicted, with 4 layers (not the actual 48) drawn in each block for demonstration. Each layer has 4 WLs represented by small squares of different colors indicating their states (free, valid, invalid, or failed). As shown, when a page in WL12 fails, instead of viewing the entire Block 0 as a bad block and migrating all of the valid data therein like traditional BBM do; LaVA only reallocates the valid data in WL12 and WL13 after marking Layer03 where WL12 is located to the bad state.

We maintain a bitmap to record each layer's bad or healthy state. As for BiCS2 TLC from Toshiba, a representative 3D CT NAND flash product we choose to conduct experiments, its chip has a capacity of 2.48Tb (310GB) and contains 31552 blocks. Considering every block has 48 layers, the bitmap size is approximately 185 (=(31552×48×1bit)/(8×1024)) KB, indicating a negligible space overhead. Due to the independent layer status bitmap, LaVA techniques can be applied to any device that uses the flash translation layer (FTL) managed with various granularity, such as page-level [6], [8], [13], block-level [3], [4], and hybrid-level mapping [11], [23].

LaVA tackles challenges associated with incorporating reliability characteristics of 3D CT NAND flash into bad block management through three specific aspects, which are elaborated subsequently. We take the simple and classical page-level mapping algorithm as an example for straightforward clarification to illustrate LaVA. Three tables are required during the FTL operations: 1) an address translation mapping table that directs a logical page to its corresponding physical page (L2P), 2) a physical page status table, and 3) a layer status bitmap.

B. LaVA in Read Operations

The process of read operations for LaVA is described in Algorithm 1. First, data is read from the corresponding physical page, which is retrieved according to the mapping table (Line 1-3). ECC is then performed on this data to determine the page's RBER (Line 4). If the RBER exceeds ECC's maximum error correction capability, the layer including the current page is marked as a bad layer (Line 5-7).

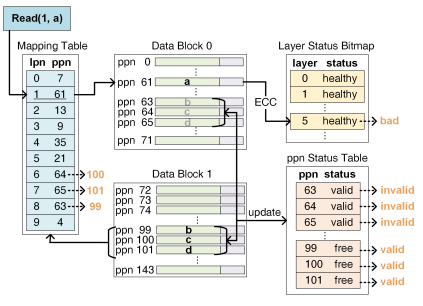


Fig. 4. Read operations for LaVA.

Fig. 4 shows the implementation of read operations using LaVA at FTL level. As a `read()` system call issued to retrieve data a from the logical page number (lpn) 1, the mapping table is accessed first. Following FTL algorithm, data is read from the mapped physical page number (ppn) 61, and ECC is then executed to calculate the page's RBER. Once the RBER goes beyond ECC's maximum capability, the corresponding layer's status is changed from healthy to bad in the bitmap, and RAID (Redundant Arrays of Independent Disks) is invoked for data correctness. Note that the status of pages remains. We reserve conventional design, which leverages two information bits to represent three page states (valid, invalid, and free), to break the constraints about FTL's implementation granularity [22]. The failed WL depicted in Fig. 3 indicates the bad layer merely. Such a ***lazy marking*** approach also alleviates the overhead for updating management data since it operates at layer level with fewer records than pages. Furthermore, lazy marking keeps page status management unaltered, leading to slight costs only for bitmap consistency with multiple processes/threads.

Algorithm 1 LaVA with layer failure threshold introduced in Read Operations.

Input:
The logical page number for the read operation, lpn_rd.
Output:
Read the content from the corresponding physical page.
1: Check the mapping table with lpn_rd as an index.
2: Get the mapped physical page number ppn_rd.
3: Read the content from ppn_rd.
4: Execute ECC and calculate its RBER.
5: **if** RBER $>$ maximum error correction capability **then**
6: Calculate its layer number.
7: Mark its layer status as bad in the layer status bitmap.
/* layer failure threshold introduced */
8: Count the number of bad layers in the block that ppn_rd belongs to according to the layer status bitmap.
9: Calculate the bad layer ratio of the number of bad layers to the total number of layers in the block.
10: **if** bad layer ratio $>$ layer failure threshold **then**
11: Mark the entire block as bad in the layer status bitmap.

The migration of valid data will occur when storage devices are idle. A task queue is created to assist. During the current operation, valid data in the bad layer is temporarily stored in DRAM buffer, and a task is inserted into the queue, indicating a ***delayed migration*** of data to flash will be executed when the SSD is not busy. Thanks to the fast DRAM access, this delay gains performance benefits by preventing time-consuming underlying writes from present operation. Additionally, storing in the temporal buffer avoids errors of retention and read/program disturbance, focusing on both reliability and performance. Due to space limitations, Fig. 4 only displays the movement of a single WL's valid data in the bad layer. During the SSD's free period, the valid data residing in the bad layer (ppn $63-65$) is transferred to free pages in a healthy layer (ppn $99-100$). According to the inverse mapping table, the corresponding items in the mapping table are changed to the newest ppn, and the states of pages involved in data relocation are updated.

Marking a bad block after all its layers fail can maximize space utilization and endurance extension. However, with less space available due to layer failures, garbage collection (GC) happens more frequently and performance suffers. Meanwhile, fragmentation becomes increasingly problematic. There exists a tradeoff between the lifetime and performance in deciding when to retire an entire block due to gradual layer failures. We set a threshold named ***layer failure threshold***. If the ratio of failed layers to all layers in a block exceeds this threshold, the block is marked as bad and will not be used anymore (Line 8-11). Note that read operations do not need to check the layer status bitmap

since delayed migration enables subsequent requests to retrieve valid data from either the bad layer or the newly allocated healthy block. Hence, read operations under LaVA are similar to the conventional ones as they only require mapping table lookups. Relying on the common SSD controller's inherent strategy is sufficient for multiple processes or threads.

C. LaVA in Write Operations

Performing write operations with LaVA is similar to the conventional solution, except LaVA needs to determine the status of the belonging layer of retrieved or selected physical page in the bitmap (Line 4 and 15 in Algorithm 2). Once data writing is complete, updates are made to both the L2P mapping table (Line 18) and *ppn* status table (Line 8, 12 and 17) based on various data layouts.

Algorithm 2 LaVA in Write Operations.

Input:
The logical page number for the write operation, lpn_wr;
Written content.
Output:
Write the content to the corresponding physical page.
1: Check the mapping table with lpn_wr as an index.
2: **if** lpn_wr exists in the mapping table **then**
3: Get the mapped physical page number ppn_wr_1.
4: Calculate its layer number and check its layer status.
5: **if** the layer status is healthy **then**
6: **if** the status of ppn_wr_1 is free **then**
7: Write the content to ppn_wr_1.
8: Valid ppn_wr_1 in the physical page status table.
9: **else**
10: Go to Line 12.
11: **else**
12: Invalid ppn_wr_1 in the physical page status table.
13: Go to Line 15.
14: **else**
15: Find a free page ppn_wr_2 with healthy layer status.
16: Write the content to ppn_wr_2.
17: Valid ppn_wr_2 in the physical page status table.
18: Update the mapping table.

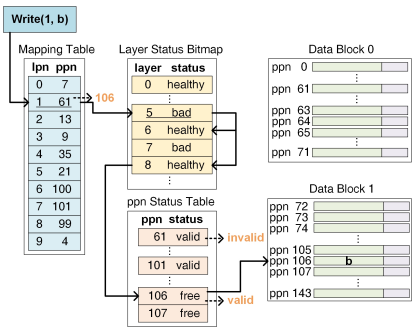


Fig. 5. Write operations for LaVA.

Fig. 5 demonstrates how LaVA is used to perform write operations at FTL level. The file system writes data b to lpn 1 when execute the command `write(1, b)`. The layer status bitmap is then checked to determine the corresponding status with the calculated layer number. If a bad layer is encountered, LaVA searches through the bitmap to find the closest free page in a healthy layer. In this case, *ppn* 106 is selected and b is written into it. Finally, the mapping table and *ppn* status table entries related to *ppn* 106/61 are updated. Given delayed migration, updating the state of *ppn* 61 as invalid is necessary to avoid redundant data movement. As mentioned, leveraging the lazy marking approach simplifies the write operation process with LaVA, as looking up a free page in the layer status bitmap costs less than traversing through the page-level *ppn* status table.

When a program operation fails, the affected layer is marked as bad and migration is delayed, which is the same as read operations encounter errors beyond ECC's capability. The only difference is that the write operation requires additional bitmap checks. To ensure correctness under multiple processes/threads, we maintain a request queue for all requests sent to SSD, so that only one request can access the bitmap at any time.

D. LaVA in Erase Operations

When the SSD nearly reaches full capacity, the garbage collection is invoked. The SSD controller picks the flash block with most invalid data as the target and relocates internal valid data to another healthy block. Then, the erase operation is executed on the victim block to make more pages free. In LaVA, no revisions are made to original erase operations considering fragmentation issues and delayed migration. Thus, no additional overhead even for multiple processes/threads. Due to the fact that the data status in the bad layer that has already been migrated delayed will be updated to invalid, there is no redundancy in data movement during garbage collection. Even if erase operations occur before delayed migration, the transfer of valid data within the entire block, including bad layers, will further alleviate fragmentation issues. Once the transfer is complete, the original data is marked as invalid, preventing subsequent delayed migrations from moving duplicate data. To avoid time-consuming erase verification failures, sub-block erase designs [5] can be used to skip bad-marked layers.

IV. Evaluation

A. Experimental Setup

Conducting experiments on a hardware SSD prototype can be costly and time-consuming due to the difficulty in obtaining a 3D SSD with open-source firmware. As such, we evaluate LaVA against traditional BBM on SSDsim, a prevalent and hardware-verified SSD simulator that can accommodate various hardware configurations [7]. We modify SSDsim to simulate a 9GB NAND flash storage with 3D structures, following specifications based on the motivational study's testbed (see Table I). Temporal simulation is adopted without actual data reads/writes involved. All management data is stored in memory, including the layer status bitmap which has an extra storage overhead of about 48KB. To ensure the consistency of metadata, we keep backups to simulate real hardware's power failure protection. The simulator employs page-level DFTL [6], greedy garbage collection and static wear leveling. We maintain lists of each page's maximum P/E cycles under different ECC capabilities collected through experiments on real platforms to simulate ECC operations, with a space overhead of 2.25MB. Additionally, traditional BBM and LaVA are implemented with an exclusive queue to postpone migration until the device is idle based on SSDsim's feature of event-driven. We revise read/write processes to include bitmap lookups and status checking/setting, which are performed before the chronological-simulated underlying flash operations.

TABLE I
PARAMETERS OF SIMULATED NAND FLASH MEMORY.

Parameter	Value	Parameter	Value
User-visible/Physical Capacity	7.5GB/9GB	Block Size	48 layers (9MB)
Channel Number	2 channels	Layer Size	12 pages (192KB)
Channel Size	2 chips	Page Size	16384 bytes (16KB)
Chip Size	1 die	Over-provisioning	20%
Die Size	1 plane	Read/Program Time	75μs/1100μs
Plane Size	256 blocks	Erase Time	10ms

TABLE II
WORKLOADS PARAMETERS.

Workload	Total Requests	Write Request Ratio	Avg. Req. Size (KB)	Avg. Req. Interval (ms)
exchange [17]	600,000	67.99%	16.1	1.18
hm0 [18]	3,993,317	64.50%	8.9	15.15
usr0 [19]	2,236,836	59.61%	22.8	27.03
ts0 [19]	1,801,487	82.40%	9.3	38.80
vps [1]	6,451,273	51.04%	11.2	27.71

We select five block-level trace, exchange, hm0, usr0, ts0, and vps, collected from various servers and systems, with their information outlined in Table II. Diversity in read/write ratio, request size and intensity make them fit various usage scenarios well and they are also widely studied and evaluated in previous works [1], [19]. Limited write requests within each trace are insufficient to wear out SSD. By feeding and replaying trace files repeatedly, statistics can be obtained when SSD reaches its end-of-life. As more layers and blocks fail, the available over-provisioning space shrinks and performance would significantly decrease despite the I/O requests can still be served. In our experiments, we regard the SSD lifetime to end when the over-provisioning space falls below 4% of the user-visible capacity, due to intolerable performance degradation.

B. Lifetime Extension Evaluation

To quantitatively evaluate the efficacy of LaVA lifetime extension, we need to redefine SSD lifetime. Typically, page endurance is measured by its P/E cycles. Therefore, SSD lifetime can be intuitively represented by the maximum data amount of flash writes or user data writes until the ECC can no longer guarantee reliability. The lifetime is calculated in

$$lifetime \iff D_{TBBM} = \sum_{i=1}^{n} C_{P/E}(i,x) \cdot D_{block} \quad (2)$$

and

$$lifetime \iff D_{LaVA} = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{P/E}(j,x) \cdot D_{layer} = \sum_{i=1}^{n} \sum_{j=1}^{m} [C_{P/E}(i,x) + \Delta C_{P/E}(j,x)] \cdot D_{layer} \quad (3)$$

where D_{TBBM} and D_{LaVA} represent the maximum data amount of flash writes or user data writes for the traditional BBM and LavA. D_{block} and D_{layer} refer to the amount of data stored in a block (n blocks in total) or layer (m layer per block). $C_{P/E}(k,x)$ indicates P/E cycles of the weakest page within the k^{th} block or layer under maximum error correction capability of x, and Δ means its increment among layers.

1) Maximum volume of flash writes: We analyze the lifetime extension of LaVA for various error correction capabilities of ECCs (Fig. 6). Dotted line represents maximum error correction capability of 5×10^{-3} for ECC that can actually be achieved, where the right side means a larger number of errors can be corrected, indicating a stronger error correction capability, and vice versa. LaVA gains pronounced lifetime extension for various capabilities of ECC. It is noticeable that the increase with weak capability is much more substantial than that with strong capability. This can be attributed to weaker ECCs tend to mark flash pages as bad at an early stage (i.e. a smaller $C_{P/E}(k,x)$), leading to low space utilization under traditional BBM and providing more optimization space for LaVA. Conversely, stronger error correction capability results in failed pages appearing later and larger $C_{P/E}(k,x)$ being accumulated. This larger base makes the increment (Δ) less significant, causing a gentle decrease in LaVA's endurance extension.

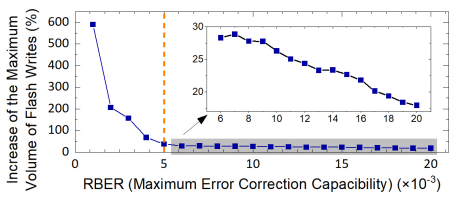


Fig. 6. The increase of the maximum volume of flash writes (LaVA vs. Traditional BBM, layer failure threshold = 1).

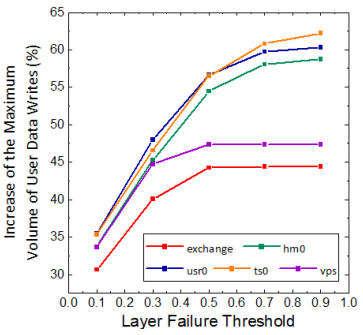
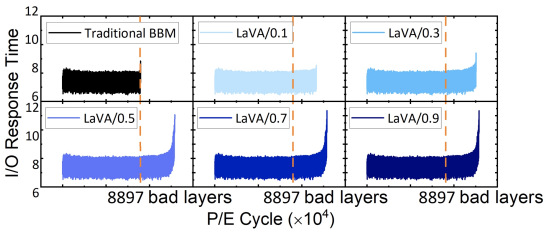


Fig. 7. The increase of the maximum volume of user data writes (LaVA vs. Traditional BBM).

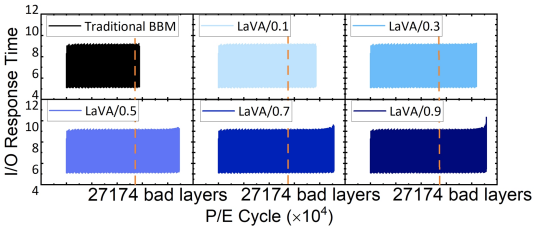
2) Maximum volume of user data writes: SSDs suffer from write amplification (WA) due to garbage collection, several more bytes are written to the flash at device level for every byte written by an application, significantly increasing data traffic written to flash by orders of magnitude compared to data requested by applications. In Fig. 7, we comprehensively evaluate LaVA's lifetime extension in terms of user data writes under various workloads and layer failure thresholds. Considering the larger optimization space under the current reachable maximum error correction capability of ECC, experiments are carried out under the RBER constraint of 5×10^{-3}. As the layer failure threshold grows, the impact of LaVA on lifetime extension gradually increases (ranging from 30.6% to 62.2%, depending on the traces). In the case of the largest threshold (0.9), there is an average increase of 54.6%, while for the smallest threshold (0.1), the average increment is only about 33.8%. One reason for this trend is that a higher threshold allows for more storage and write operations before blocks wear out by providing more space. Overall, compared to traditional BBM, using LaVA greatly boost SSD endurance, and larger layer failure thresholds result in greater improvements.

C. I/O Performance Evaluation

We now assess how LaVA affects I/O performance. Fig. 8 presents comparisons of response time (read/write latencies) between LaVA (with various layer failure thresholds) and traditional BBM across different workloads through the SSD lifetime (only two traces are shown due to space constraints). The dotted lines indicate the time points when bad layers started to appear for each scenario. Both techniques exhibit a similar trend in the variation of I/O response time. The comparable response time during early runtime suggests negligible overhead of LaVA on SSD performance in read/write operations. However, there is an increase in response time towards the end of SSD lifetime (the right side of dotted lines) under LaVA, especially as the threshold grows. This is due to less available



(a) exchange



(b) hm0

Fig. 8. I/O response time (log scale on y-axis).

over-provisioning space when SSDs wear out, causing frequent garbage collection and performance degradation.

We use the 99^{th} and 95^{th} percentile I/O response times as the metrics for our analysis. Exchange (Fig. 8 (a)) is much more I/O-intensive than other traces; therefore, more significant GC overheads add to its response time, resulting in a larger decline in performance. Regarding other workloads (refer to Fig. 8 (b)), LaVA outperforms traditional BBM by increasing SSD lifetime by an average of 57.2%, while only raising the 99^{th}/95^{th} percentile I/O response time by an average of 7.8%/3.9% under the largest layer failure threshold (0.9). To be more specific, compared to the traditional bad block management:

- for the threshold of 0.1, LaVA improves SSD lifetime by 30.6% and increases the 99^{th}/95^{th} percentile I/O response time by 8.0%/2.6% (less than 10%);
- for the threshold of 0.3, LaVA improves the SSD lifetime by 40.1% and increases the 99^{th}/95^{th} percentile I/O response time by 17.9% / 5.6% (less than 20%);
- for larger thresholds, LaVA significantly increases the 99^{th}/95^{th} percentile I/O response time (more than 5×).

In summary, LaVA considerably prolongs the lifetime of SSD (30.6%-62.2%) compared to the traditional BBM with only a slight drop in performance (less than 10% increase in tail I/O response time). This approach achieves a good balance between endurance extension and performance degradation. For those who pursue an even longer lifetime, a large layer failure threshold is preferable. However, if performance is a concern, a moderate layer failure threshold is recommended.

V. Conclusion

This paper proposes a layer variation aware and fault-tolerant bad block management, LaVA, to effectively extend the endurance of 3D CT NAND flash storage. LaVA is inspired by the nature of cross-layer variations and intra-layer concentration of errors among pages and blocks to maximize the lifetime potency of each page. A fine-grained technique is introduced as opposed to the traditional coarse-grained bad block management. When a page failure occurs, only pages inside the relevant layer of the block are discarded, rather than all pages within the block. Such a layer-level strategy achieves good trade-offs between lifetime improvement and performance maintenance while providing users with greater flexibility. Experimental results based on real-world workloads demonstrate that LaVA dramatically enhances SSD lifetime (30.6%-62.2%) with minimal performance degradation (less than a 10% increase in tail I/O response time) compared to the traditional bad block management.

Acknowledgment

This work was supported by National Natural Science Foundation of China under Grants No. 62372197, No. U2001203, No. U22A2071, No. 62102155, by Natural Science Foundation of Shandong Province under Grant No. ZR2020LZH014, and by Research Grants Council of the Hong Kong Special Administrative Region, China under Grant No. CityU 11217020.

References

[1] D Arteaga et al. Client-side flash caching for cloud systems. *SYSTOR'14*.
[2] Yu Cai et al. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, 2017.
[3] Siddharth Choudhuri and Tony Givargis. Performance improvement of block based nand flash translation layer. In *CODES+ISSS '07*.
[4] Tae-Sun Chung, Stein Park, Myung-Jin Jung, and Bumsoo Kim. Staff: state transition applied fast flash translation layer. In *ARCS 2004*.
[5] Hongbin Gong, Zhirong Shen, and Jiwu Shu. Accelerating sub-block erase in 3d nand flash memory. In *ICCD 2021*.
[6] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices*, 2009.
[7] Y Hu et al. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *ICS '11*.
[8] Hyukjoong Kim and Dongkun Shin. Clustered page-level mapping for flash memory-based storage devices. *IEEE TCE*, 2015.
[9] Jiyoung Kim, Augustin J Hong, et al. A stacked memory device on logic 3d technology for ultra-high-density data storage. *Nanotechnology*, 2011.
[10] C-H Lee, Jungdal Choi, et al. Multi-level nand flash memory with 63 nm-node tanos (si-oxide-sin-al2o3-tan) cell structure. In *2006 Symposium on VLSI Technology, 2006. Digest of Technical Papers.* IEEE.
[11] Hyun-Seob Lee, Hyun-Sik Yun, et al. Hftl: hybrid flash translation layer based on hot data identification for flash memory. *IEEE TCE*, 2009.
[12] Yan Li and Khandker N. Quader. Nand flash memory: Challenges and opportunities. *Computer*, 46(8):23–29, 2013.
[13] Dongzhe Ma, Jianhua Feng, et al. Lazyftl: A page-level flash translation layer optimized for nand flash memory. In *ACM SIGMOD 2011*.
[14] Ruixiang Ma et al. Blockhammer: Improving flash reliability by exploiting process variation aware proactive failure prediction. *TCAD 2020*.
[15] Ruixiang Ma, Fei Wu, Zhonghai Lu, et al. Rber-aware lifetime prediction scheme for 3d-tlc nand flash memory. *IEEE Access*, 2019.
[16] Rino Micheloni et al. *3D Flash memories*. Springer, 2016.
[17] Microsoft Enterprise Traces. *http://iotta.snia.org/traces/130*.
[18] MSR Cambridge Traces. *http://iotta.snia.org/traces/388*.
[19] Dushyanth Narayanan et al. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)*, 2008.
[20] Tianyu Ren, Qiao Li, Min Ye, and Chun Jason Xue. Read disturb and reliability: The complete story for 3d ct nand flash. In *NVMSA 2023*.
[21] Bianca Schroeder, Arif Merchant, and Raghav Lagisetty. Reliability of nand-based ssds: What field studies tell us. *Proceedings of the IEEE*.
[22] Debao Wei, Libao Deng, et al. Peva: A page endurance variance aware strategy for the lifetime extension of nand flash. *IEEE TVLSIS*, 2015.
[23] Chin-Hsien Wu and Tei-Wei Kuo. An adaptive two-level management for the flash translation layer in embedded systems. In *ICCAD '06*.
[24] Fei Wu, Yue Zhu, Qin Xiong, et al. Characterizing 3d charge trap nand flash: Observations, analyses and applications. In *ICCD 2018*.
[25] Jui-Nan Yen, Yao-Ching Hsieh, Cheng-Yu Chen, Tseng-Yi Chen, et al. Efficient bad block management with cluster similarity. In *HPCA 2022*.
[26] Jung H Yoon, Ranjana Godse, Gary Tressler, et al. 3d-nand scaling and 3d-scm—implications to enterprise storage. *Flash Memory Summit*, 2017.