Wear Leveling in SSDs Considered Harmful

Ziyang Jiao
Syracuse University
zjiao04@syr.edu

Janki Bhimani
Florida International University
jbhimani@fiu.edu

Bryan S. Kim
Syracuse University
bkim01@syr.edu

ABSTRACT

We argue that wear leveling in SSDs does more harm than good under modern settings where the endurance limit is in the hundreds. To support this claim, we evaluate existing wear leveling techniques and show that they exhibit anomalous behaviors and produce a high write amplification. These findings are consistent with a recent large-scale field study on the operational characteristics of SSDs. We discuss the option of forgoing wear leveling and instead adopting capacity variance in SSDs, and show that the capacity variance extends the lifetime of the SSD by up to 2.94×.

CCS CONCEPTS

• **Information systems** → **Flash memory**; **Information lifecycle management**.

KEYWORDS

SSD, wear leveling, endurance, lifetime, write amplification, capacity variance

ACM Reference Format:

Ziyang Jiao, Janki Bhimani, and Bryan S. Kim. 2022. Wear Leveling in SSDs Considered Harmful. In *14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '22), June 27–28, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538643.3539750

1 INTRODUCTION

Wear leveling (WL) in solid-state drives (SSDs) seeks to equalize the amount of wear so that no cells prematurely fail prior to the end of the SSD's lifetime [3, 6–8, 10]. While there are different approaches to implementing WL (from static [3, 6, 10] to dynamic [7, 8]), the underlying goal is to use younger blocks with fewer erases more than the older blocks. Static wear leveling techniques [3, 6, 10], in particular, proactively relocate data within an SSD, thereby incurring additional write amplification for the sake of equalizing the number of erases. Dynamic wear leveling [7, 8], on the other hand, combines WL with other SSD-internal tasks such as garbage collection, reducing the efficiency of victim block selection. In other words, WL techniques incur additional wear-out to increase the overall lifetime of the SSD.

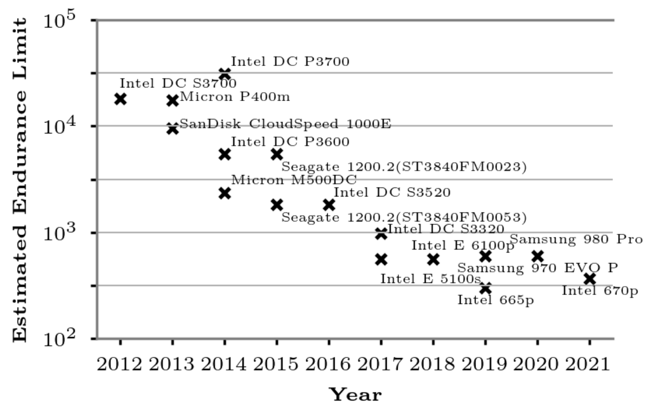


Figure 1: The estimated endurance limit of various SSDs in the past years. We estimate the endurance limit by dividing the SSD's TBW (terabytes written: the total amount of writes the SSD manufacturer guarantees) by the logical capacity. This estimation is consistent with recent works [21, 22].

Ideally, the wear leveling algorithm would minimize its overhead while maximizing its effectiveness. However, a recent large-scale field study on millions of SSDs reveals that the WL techniques in modern SSDs present limited effectiveness and are far from perfect [23]. This study shows that some WL algorithms are unable to achieve their intended goal as some of the blocks wear out 6× faster than the average. Furthermore, some SSDs exhibit a median write amplification factor (WAF) of around 100, although the cause of this cannot be definitive. With the endurance limit of flash memory steadily decreasing, as shown in Figure 1, it will become increasingly challenging to design an effective (equal wear) yet efficient (low write amplification) wear leveler.

To understand the underlying reasons for the ineffectiveness of WL in SSDs, we evaluate three representative WL techniques [3, 6, 7] that have been compared against a wide variety of other WLs [5, 10, 15, 24, 28, 30, 33]. Our experiments find that WL algorithms produce a counter-productive

HotStorage '22, June 27–28, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9399-7/22/06…$15.00
https://doi.org/10.1145/3538643.3539750

Table 1: Representative wear leveling algorithms.

Name	Type	Parameters	Principle	Comparisons
DP [3]	Static	Fixed, a predefined threshold (TH)	Hot-cold swapping	HC [15], 2L [33], EP [30], OBP [10]
PWL [6]	Static	Adaptive, a initial threshold (THR_{init})	Cold-data migration	BET [5] and Rejuvenator [24]
DAGC [7]	Dynamic	Adaptive, no external parameters	Adjust GC victim	DTGC [28]

result where the erase counts diverge, increasing the spread rather than reducing it. This happens when the WL attempts to move data that it incorrectly perceives to be cold into old blocks. In addition, we observe that WL-induced WAF can reach as high as 11.49 where WL's attempt to achieve a tight distribution of erase count comes at the cost of a high WAF.

Instead of designing a new wear leveling algorithm that patches these issues, we fundamentally ask if wear leveling is worth the trouble. Wear leveling exists to maintain the fixed capacity abstraction, when in reality, the underlying media for SSDs fail partially [26]. Instead, we explore and quantify the benefits of capacity variance in an SSD that gracefully reduces its capacity as flash memories become bad [18]. Our experimental results show that capacity variance allows up to 84% more writes to the SSD with wear leveling, and up to 2.94× more writes *without* WL.

2 WEAR LEVELING: BOON OR BANE?

Motivated to reproduce the results from a recent large-scale study [23], we examine the behavior of WL algorithms under a synthetic microbenchmark. We evaluate three representative wear leveling (WL) algorithms, **Dual-Pool (DP)** [3], **Progressive Wear Leveling (PWL)** [6], **Dynamic Adjustment Garbage Collection (DAGC)** [7], and Table 1 summarizes their characteristics.

2.1 Experimental Setup

We extend FTLSim [9][1] for our experiments. This prior work validates the analytical model for SSD performance and thus focuses on accurately modeling SSD-internal statistics, rather than SSD-external performance such as latency and throughput. This allows us to simulate the entire lifetime of an SSD (a few hundreds of TiB written) within a few months and also observe its internal activities. Table 2 summarizes the SSD configuration and policies for our experiments.

To understand the behavior of WL techniques, we synthetically generate workload to control the I/O pattern better. All I/Os are small random writes, but the distribution is controlled by two parameters r and h ($0 < r < 1$ and $0 < h < 1$): r fraction of writes go to the h fraction of the footprint (hot addresses) [29]. We use r/h to indicate that the r fraction of writes occur on the h fraction of the workload footprint. Unless otherwise noted, we generate the I/O workload for the entire logical address space. Prior to each experiment, we pre-condition the SSD with one sequential full-drive write, followed by three random full-drive writes (256 GiB sequential + 768 GiB random write) to put the drive into a steady-state [31].

2.2 Performance of Wear Leveling

We investigate the performance of wear leveling in the following three aspects: (1) write amplification, (2) effectiveness in equalizing the erase count, and (3) behaviors under different access footprints.

Write amplification. We measure the WL-induced write amplification (WA) by using a synthetic workload of $r/h = 0.9/0.1$ for up to 100 full-drive writes (25 TiB). The WL parameter values we experiment with are similar to those used in the prior work [3, 6, 7]. Figure 2 shows the write amplification, and we make the following four observations. First, the overall write amplification can be as high as 11.49, in which 5.4 is caused by WL. This overhead is as much as the WA caused by garbage collection. This means that for each 256 GiB user data written, wear leveling alone will create an additional 1.35 TiB of data writes internally. Second, the WA is sensitive to the WL threshold parameter, TH. Changing the TH from 10 to 5 for the DP algorithm will amplify the amount of data written to 1.6×. Third, PWL produces a significantly high WA of 11.49 once the SSD ages beyond 80 full-drive writes. PWL is an adaptive WL algorithm, and it becomes overly aggressive at a later stage while being dormant during the early stage. Lastly, WA steadily increases over time as the SSD ages, indicating that SSD aging will accelerate as more data are written.

Table 2: SSD configuration and policies. Only the parameters relevant to understanding the wear leveling behavior are shown.

Parameter	Value	Parameter	Value
Page size	4 KiB	Physical capacity	284 GiB
Pages per block	256	Logical capacity	256 GiB
Block size	1 MiB	Over-provisioning	11%
Block allocation	FIFO	Garbage collection	Greedy

[1] Our extension is available at https://github.com/ZiyangJiao/FTLSim-WL.

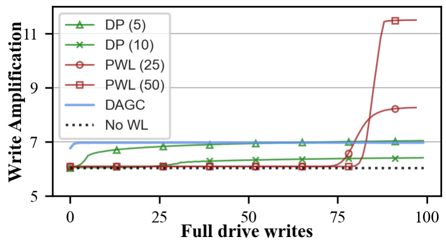
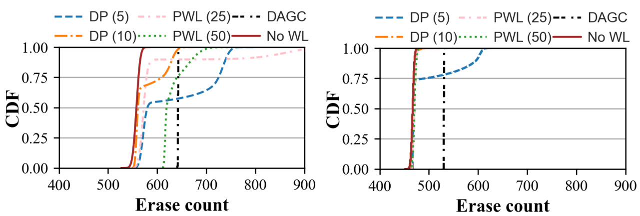


Figure 2: The write amplification caused by wear leveling under a $r/h = 0.9/0.1$ synthetic workload. The parenthetical values in the legends are the WL threshold parameters. $PWL(50)$ that aggressively performs wear leveling at the late stage causes its WAF to be as high as 11.49.



(a) $r/h = 0.9/0.1$ **with large footprint.**

(b) $r/h = 0.5/0.5$ **with large footprint.**

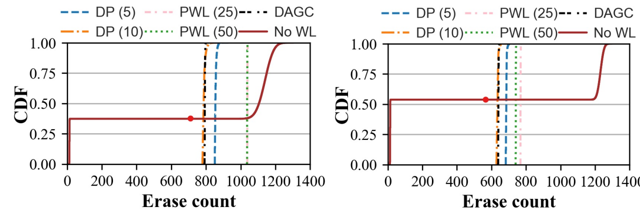
Figure 3: The distribution of erase count when full logical address space is used after writing 25TiB. WL shows the performance anomaly under $r/h = 0.9/0.1$ workload (Figure 3a). On the other hand, the benefit from wear leveling is negligible compared to not running at all under $r/h = 0.5/0.5$ (Figure 3b).

Wear leveling effectiveness. We measure the distribution of erase count under a synthetic workload as shown in Figure 3. We perform 100 full-drive writes (25 TiB) using a workload with $r/h = 0.9/0.1$ (Figure 3a), and with $r/h = 0.5/0.5$ (Figure 3b).

With $r/h = 0.9/0.1$, as shown in Figure 3a, all configurations of DP and PWL show a worse distribution of erase counts than not running WL ($NoWL$). DP and PWL show a concave dip in the CDF curve, indicating a bimodal distribution of erase counts. NoWL, on the other hand, shows a nearly vertical line, meaning that the erase counts are more tightly distributed. We consider this to be a *performance anomaly* of wear leveling because it behaves the opposite of what is expected. We examine the bimodal distribution of $DP(5)$ and find that the blocks associated with the cold pool are older than those in the hot pool. The DP algorithm's underlying assumption is that blocks containing hot data are older than blocks with cold data, and it compares the erase count of the oldest block in the hot pool and the youngest block in the cold pool. If the youngest block in the cold pool happens to be older than the oldest block in the hot pool, however, it will still trigger the swap between the two blocks, causing this inversion. DAGC also achieves good evenness but amplifies data writes by 18% compared to $NoWL$.

On the other hand, with a uniformly random workload (Figure 3b), there is a negligible difference among DP, PWL, and NoWL. This is because, with a uniform workload, all blocks are used equally, and there is little room for wear leveling. We do observe, however, that $DP(5)$ still exhibits a performance anomaly though at a smaller degree than under $r/h = 0.9/0.1$ (*cf.* Figure 3a) As for DAGC, the overall efficiency for garbage collection is reduced as its victim selection considers both valid ratio and erase count, incurring 15% more data writes than NoWL.

These experiments show that WL algorithms are a double-edged sword. As shown in Figure 3a, it can make the distribution of wear worse than not running WL at all. On the other hand, it can achieve good wear leveling but at a high cost of accelerated overall wear state.



(a) $r/h = 0.9/0.1$ **with small footprint.**

(b) $r/h = 0.5/0.5$ **with small footprint.**

Figure 4: The distribution of erase count when only 5% of the logical address space is used. The red dot indicates the average erase count for NoWL.

Small access footprint. Here we explore the performance of wear leveling when the accesses are restricted to a small address space (5% of total) using two synthetic workloads, $r/h = 0.9/0.1$ and $r/h = 0.5/0.5$, as shown in Figure 4.

Overall, we observe that most WL techniques are effective in equalizing the erase count, as shown by the near-vertical CDF curve in both Figure 4a and Figure 4b. $NoWL$, on the other hand, shows a bimodal distribution between used blocks and unused blocks in both workloads. We also observe that when the workload is skewed (Figure 4a), the WL techniques achieve this evenness by amplifying the amount of data writes, as shown by the rightward shift in the CDF curves. For a uniform workload, on the other hand (Figure 4b), the overall write amplification from wear leveling is much lower as data are equally likely to be invalidated.

Unlike the results from Figure 3a and Figure 3b where the entire logical address space is written, WL is effective only when a small fraction of the address space is used, restricting its overall usefulness.

2.3 Summary of Findings

Table 4 summarizes the effectiveness of WL from our experiments using synthetic workloads. Only when the access pattern is uniform and footprint is small, WL is beneficial; otherwise, it is detrimental or has negligible effect.

Instead of proposing a new wear leveling algorithm that solves both the write amplification overhead and performance anomaly, we question the circumstances that require wear leveling and examine its necessity in the next section.

Table 4: Qualitative effectiveness of wear leveling.

	Skewed access	Uniform access
Large footprint	Anomalous (Figure 3a)	Negligible (Figure 3b)
Small footprint	Write amplified (Figure 4a)	Effective (Figure 4b)

3 CASE STUDY ON CAPACITY VARIANCE

If the interface were to allow a reduction in the SSD's exported capacity, WL becomes unnecessary as it does not need to ensure that all blocks wear out evenly. The idea of capacity variance is not new [18]: the Zoned Namespace (ZNS) specification allows zones to be taken offline [34], effectively shrinking the SSD's capacity. In this section, we study such a case of capacity reduction and the overall lifetime of the SSD with and without WL.

We implement a capacity-variant SSD on the extended FTLSim [9] from § 2 and use the SSD configuration in Table 2 for our evaluation. However, we set the endurance limit to 500 erases, a typical level for QLC [21, 22], and once a block reaches this, it will be mapped out and no longer used in the SSD, effectively reducing the SSD's physical capacity. For the fixed capacity SSD, the SSD is considered to reach its end of life once the physical capacity becomes smaller than its logical capacity: the SSD is considered to have failed once this happens. On the other hand, the capacity-variant SSD gracefully reduce its capacity below the initial logical space, to a user defined threshold (if set) or as low as the access footprint for the workload. For a capacity-variant SSD, if the logical capacity can no longer be reduced without losing user data, the SSD is considered to have failed.

For the workload, we use nine real-world block I/O traces that were collected from running YCSB [36], a virtual desktop infrastructure (VDI) [19], and Microsoft production servers (WBS, DTRS, DAP-PS, LM-TBE, MSN-CFS, MSN-BEFS, RAD-BE) [17]. In particular, the Microsoft production traces are outdated, but we use it to include a wider variety of workloads. The traces are modified into a 256GiB range (the logical capacity of the SSD), and all the requests are aligned to 4KiB boundaries. Similar to the synthetic workload evaluation, the SSD is pre-conditioned with one sequential full-drive write and three random full-drive writes on the entire logical space. The traces run in a loop indefinitely, continuously generating I/O until the SSD becomes unusable. Table 3 summarizes the trace workload characteristics.

We evaluate the following eight designs.

Fix_NoWL runs no WL on a fixed capacity SSD.
Fix_DP runs $DP(5)$ on a fixed capacity SSD.
Fix_PWL runs $PWL(50)$ on a fixed capacity SSD.
Fix_DAGC runs $DAGC$ on a fixed capacity SSD.
Var_NoWL runs no WL on a capacity-variant SSD.
Var_DP runs $DP(5)$ on a capacity-variant SSD.
Var_PWL runs $PWL(50)$ on a capacity-variant SSD.
Var_DAGC runs $DAGC$ on a capacity-variant SSD.

Figure 5 shows the amount of data written to the SSD before failure for the nine I/O traces. The y-axis is in terms of the number of drive writes. For example, for 100 drive writes, 25TiB of data have been written. Overall, we observe that with fixed capacity SSDs, running WL is better than not running WL, but only by a small margin: Fix_DP extends the lifetime by only 13% on average compared to Fix_NoWL, and with workloads such as `VDI` and `DTRS`, Fix_DP and Fix_DAGC perform worse than Fix_NoWL. However, with

Table 3: Trace workload characteristics. `YCSB-A` is from running YCSB [36], `VDI` is from a virtual desktop infrastructure [19], and the remaining 7 (from `WBS` to `RAD-BE`) are from Microsoft production servers [17].

Workload	Description	Footprint (GiB)	Avg. write size (KiB)	Hotness (r/h)	Sequentiality
YCSB-A	User session recording	89.99	50.48	64.69/35.31	0.49
VDI	Virtual desktop infrastructure	255.99	17.99	64.45/35.55	0.14
WBS	Windows build server	56.05	27.82	60.34/39.66	0.02
DTRS	Developer tools release	150.63	31.85	54.20/45.80	0.12
DAP-PS	Advertisement payload	36.06	97.20	55.02/44.98	0.16
LM-TBE	Map service backend	239.49	61.90	60.29/39.71	0.94
MSN-CFS	Storage metadata	5.58	12.92	69.28/30.72	0.25
MSN-BEFS	Storage backend file	31.42	11.62	70.18/29.82	0.03
RAD-BE	Remote access backend	14.73	13.02	65.51/34.49	0.33

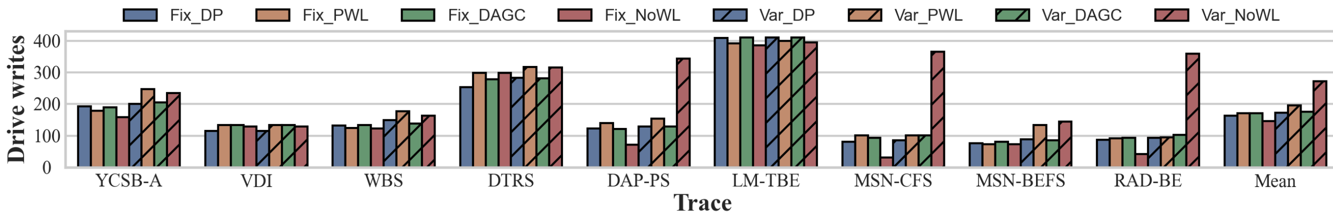


Figure 5: Evaluation of the presence and absence of wear leveling in both a fixed capacity and a capacity-variant SSD. Capacity variance extends the lifetime by 86% on average, and as high as 2.94× in the case of `RAD-BE`.

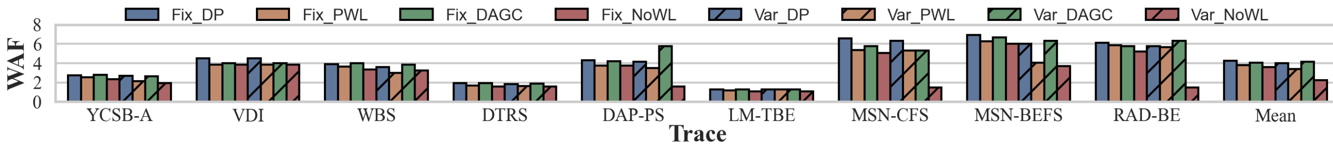


Figure 6: Write amplification caused by WL and GC. While a large sequential workload such as `LM-TBE` only has a low write amplification overhead of 1.26, most other workloads exhibit high wear leveling overhead for DP and DAGC, reaching as high as 6.9. Without wear leveling, the average write amplification is only 2.89.

capacity variance, not running WL is better than running WL by a large margin: Var_NoWL extends the lifetime by 86% on average, and as much as 2.94× for `RAD-BE`. We explain this result by the measurement of write amplification caused by wear leveling, shown in Figure 6.

Workloads with a relatively small footprint. We observe that capacity variance is most effective on workloads such as `DAP-PS`, `MSN-CFS`, and `RAD-BE`. These workloads are characterized by a small access footprint where gracefully reducing the capacity achieves more lifetime extension than WL. Specifically, capacity variance without wear leveling allows 2.94× more data to be written to the SSD for `RAD-BE`.

`MSN-BEFS` also has a small footprint, but we observe a comparatively lower lifetime extension of 0.91×. In fact, the lifetime of Fix_NoWL isn't too far off from that of Fix_DP, only 5% less. The reason for this is due to garbage collection: This workload contains a lot of small random writes, causing garbage collection to be active, dwarfing the WL-induced WAF. Because of this, `MSN-BEFS` only allows 145 full-drive writes (36.27 TiB) even for the capacity-variant SSD.

Workloads with a relatively large footprint. `LM-TBE` and `VDI` are two workloads with the largest footprint, and the benefit of capacity variance is diminished in such workloads. However, we find that capacity variance still achieves the similar lifetime extension compared to the best case via WL under this scenario: for `VDI`, Fix_PWL extends the lifetime by only 3.1% compared to Var_NoWL, and for `LM-TBE`, Fix_DP extends it by only 3.6% compared to Var_NoWL. A large footprint means that there is little to gain from reducing the capacity as data are still in use. For `LM-TBE`, the large sequential write with relatively high uniformity causes the write amplification for WL to be small, as low as 1.18. This allows wear leveling to squeeze more writes out of the SSD.

`DTRS` is one of the rare occasions where not running WL is better in a fixed capacity SSD. Fix_NoWL allows 18% more writes compared to Fix_DP, and 7% more compared to Fix_DAGC. This is due to the high write amplification of wear leveling. Although the write access pattern of `DTRS` is fairly uniform, we suspect that a wear leveling anomaly occurred, causing a subset of blocks to age rapidly. Introducing capacity variance extends the lifetime for all three cases, however, with Var_NoWL extending the lifetime by 24% compared to Fix_DP. Var_PWL outperforms Var_NoWL, but the difference is only 3%.

4 DISCUSSION AND RELATED WORK

Wear leveling is a mature and well-understood topic in both academia and industry, but *getting it right* has proven to be difficult as shown by the recent large-scale field study [23]. This study on millions of modern SSDs shows that some blocks wear out 6× faster than the average, revealing the ineffectiveness of wear leveling algorithms. We discuss related work on wear leveling and file system support necessary for capacity variance.

Wear leveling and write amplification. There exists a large body of work on garbage collection and its associated write amplification (WA) for SSDs, from analytical approaches [9, 12, 27, 37] to experimental results [4, 16, 38]. However, there is surprisingly limited work that measures the WA caused by wear leveling (WL), and they often rely on a *back-of-the-envelope calculation* for estimating the overhead and lifetime [39]. Even those that perform a more rigorous study evaluate the efficacy of WL by measuring the amount of writes the SSD can endure [6, 14, 20, 35] or the distribution of erase count [1, 3, 13]; only the Dual-Pool algorithm [3] present the overhead of WL.

File system support. Using a capacity-variant SSD would need support from the file system. Thankfully, the current system design can make this transition less painful for the following reasons. First, the `TRIM` command, widely supported by interface standards such as `NVMe` allows the file system to explicitly declare that the data (at the specified addresses) are no longer in use. This allows the SSD to discard the data safely and would help determine if the exported capacity can be gracefully reduced. Second, modern file systems can safely compact their content so that the data in use are contiguous in the logical address space. Log-structure file systems such as `F2FS` support this more readily, but file system defragmentation can also achieve the same effect in in-place update file systems such as `ext4`. Lastly, zoned namespace (ZNS), a new abstraction for storage devices that gained significant interest in the research community [2, 11, 32], already supports shrinking the device capacity by taking zones offline [34]. The capacity variance potentially incurs overhead for the file system to relocate data from one logical space to another. Naïvely, the file system would relocate not only the data at the high address space, but also update any metadata for the block allocation and inode. A more advanced command such as SHARE [25] can be used to reduce the relocation overhead.

5 CONCLUSION

From a system design standpoint, it is easier to build the storage stack with a fixed capacity abstraction. However, this abstraction requires the implementation of a wear leveler in SSDs that is surprisingly both ineffective and inefficient. Furthermore, with increasing flash memory block size and decreasing endurance limit for flash, we expect the wear leveling problem to exacerbate in the near future. We believe it is necessary to re-think the benefits and costs of the wear leveler in SSDs and the block interface abstraction.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and suggestions that help us to improve the quality of this paper. Peter Desnoyers provided the original version of FTLSim used in this work. This research was supported, in part, by the National Science Foundation awards CNS-2008324 and CNS-2008453.

REFERENCES

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark S. Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX Annual Technical Conference (ATC)*. 57–70.

[2] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *USENIX Annual Technical Conference (ATC)*. 689–703.

[3] Li-Pin Chang. 2007. On efficient wear leveling for large-scale flash-memory storage systems. In *ACM Symposium on Applied Computing (SAC)*.

[4] Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. 2004. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. Embed. Comput. Syst.* 3, 4 (2004), 837–863.

[5] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. 2010. Improving Flash Wear-Leveling by Proactively Moving Static Data. *IEEE Trans. Computers* 59, 1 (2010), 53–65.

[6] Fu-Hsin Chen, Ming-Chang Yang, Yuan-Hao Chang, and Tei-Wei Kuo. 2015. PWL: a progressive wear leveling to minimize data migration overheads for NAND flash devices. In *Design, Automation & Test in Europe Conference & Exhibition, (DATE)*.

[7] Zhe Chen and Yuelong Zhao. 2020. DA-GC: A Dynamic Adjustment Garbage Collection Method Considering Wear-leveling for SSD. In *Great Lakes Symposium on VLSI (GLSVLSI)*. 475–480.

[8] Mei-Ling Chiang, Paul CH Lee, and Ruei-Chuan Chang. 1999. Using data clustering to improve cleaning performance for flash memory. *Software: Practice and Experience* 29, 3 (1999), 267–290.

[9] Peter Desnoyers. 2012. Analytic modeling of SSD write performance. In *International Systems and Storage Conference (SYSTOR)*.

[10] Thomas Gleixner, Frank Haverkamp, and Artem Bityutskiy. 2006. UBI - Unsorted Block Images. http://linux-mtd.infradead.org/doc/ubidesign/ubidesign.pdf.

[11] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 147–162.

[12] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman A. Pletka. 2009. Write amplification analysis in flash-based solid state drives. In *Israeli Experimental Systems Conference (SYSTOR)*. 10.

[13] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. 2017. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs. In *USENIX Conference on File and Storage Technologies (FAST)*. 375–390.

[14] Xavier Jimenez, David Novo, and Paolo Ienne. 2014. Wear unleveling: improving NAND flash lifetime by balancing page endurance. In *USENIX conference on File and Storage Technologies (FAST)*, Bianca Schroeder and Eno Thereska (Eds.). 47–59.

[15] Han joon Kim and Sang goo Lee. 2002. An Effective Flash Memory Manager for Reliable Flash Memory Space Management. *IEICE Transactions on Information and Systems* 85 (2002), 950–964.

[16] Won-Kyung Kang, Dongkun Shin, and Sungjoo Yoo. 2017. Reinforcement Learning-Assisted Garbage Collection to Mitigate Long-Tail Latency in SSD. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 134:1–134:20.

[17] Swaroop Kavalanekar, Bruce L. Worthington, Qi Zhang, and Vishal Sharda. 2008. Characterization of storage workload traces from production Windows Servers. In *International Symposium on Workload Characterization (IISWC)*.

[18] Bryan S. Kim, Eunji Lee, Sungjin Lee, and Sang Lyul Min. 2019. CPR for SSDs. In *Workshop on Hot Topics in Operating Systems (HotOS)*.

[19] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. 2017. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *ACM International Systems and Storage Conference (SYSTOR)*.

[20] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. 2012. Lifetime management of flash-based SSDs using recovery-aware dynamic throttling. In *USENIX conference on File and Storage Technologies (FAST)*.

[21] Shuwen Liang, Zhi Qiao, Sihai Tang, Jacob Hochstetler, Song Fu, Weisong Shi, and Hsing-Bung Chen. 2019. An Empirical Study of Quad-Level Cell (QLC) NAND Flash SSDs for Big Data Applications. In *IEEE International Conference on Big Data (Big Data)*. 3676–3685.

[22] Chun-Yi Liu, Yunju Lee, Myoungsoo Jung, Mahmut Taylan Kandemir, and Wonil Choi. 2021. Prolonging 3D NAND SSD Lifetime via Read Latency Relaxation. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 730–742. https://doi.org/10.1145/3445814.3446733

[23] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. 2022. Operational Characteristics of SSDs in Enterprise Storage Systems: A Large-Scale Field Study. In *USENIX Conference on File and Storage Technologies (FAST)*. 165–180. https://www.usenix.org/conference/fast22/presentation/maneas

[24] Muthukumar Murugan and David Hung-Chang Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*.

[25] Gihwan Oh, Chiyoung Seo, Ravi Mayuram, Yang-Suk Kee, and Sang-Won Lee. 2016. SHARE Interface in Flash Storage for Relational and NoSQL Databases. In *International Conference on Management of Data (SIGMOD)*. 343–354.

[26] Open NAND Flash Interface. 2021. ONFI 5.0 Spec. http://www.onfi.org/specifications/.

[27] Changhyun Park, Seongjin Lee, Youjip Won, and Soohan Ahn. 2017. Practical Implication of Analytical Models for SSD Write Amplification. In *ACM/SPEC on International Conference on Performance Engineering (ICPE)*. 257–262.

[28] Yi Qin, Dan Feng, Jingning Liu, Wei Tong, and Zhiming Zhu. 2014. DT-GC: Adaptive Garbage Collection with Dynamic Thresholds for SSDs. In *2014 International Conference on Cloud Computing and Big Data*. 182–188. https://doi.org/10.1109/CCBD.2014.28

[29] Mendel Rosenblum and John K. Ousterhout. 1992. The Design and Implementation of a Log-Structured File System. In *PhD thesis, University of California at Berkeley.*

[30] Sandisk. 2003. Sandisk Flash Memory Cards Wear Leveling. http://www.sandisk.com/Assets/File/OEM/WhitePapersAndBrochures/RS-MMC/WPaperWearLevelv1.0.pdf.

[31] Esther Spanjer and Easen Ho. 2011. The Why and How of SSD Performance Benchmarking - SNIA. https://www.snia.org/sites/default/education/tutorials/2011/fall/SolidState/EstherSpanjer_The_Why_How_SSD_Performance_Benchmarking.pdf.

[32] Theano Stavrinos, Daniel S. Berger, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Don't be a blockhead: zoned namespaces make work on conventional SSDs obsolete. In *Workshop on Hot Topics in Operating Systems (HotOS)*. 144–151.

[33] STMicro. 2006. Wear Leveling in Single Level Cell NAND Flash Memories. STMicroelectronics Application Note (AN1822).

[34] Western Digital. 2020. Zoned Namespaces (ZNS) SSDs. https://zonedstorage.io/introduction/zns/.

[35] Ellis Herbert Wilson, Myoungsoo Jung, and Mahmut T. Kandemir. 2014. ZombieNAND: Resurrecting Dead NAND Flash for Improved SSD Longevity. In *IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. 229–238.

[36] Gala Yadgar, Moshe Gabel, Shehbaz Jaffer, and Bianca Schroeder. 2021. SSD-based Workload Characteristics and Their Performance Implications. *ACM Trans. Storage* 17, 1 (2021), 8:1–8:26.

[37] Yudong Yang, Vishal Misra, and Dan Rubenstein. 2015. On the Optimality of Greedy Garbage Collection for SSDs. *SIGMETRICS Perform. Evaluation Rev.* 43, 2 (2015), 63–65.

[38] Qi Zhang, Xuandong Li, Linzhang Wang, Tian Zhang, Yi Wang, and Zili Shao. 2015. Optimizing deterministic garbage collection in NAND flash storage systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 14–23.

[39] Tao Zhang, Aviad Zuck, Donald E. Porter, and Dan Tsafrir. 2017. Flash Drive Lifespan *is* a Problem. In *Workshop on Hot Topics in Operating Systems (HotOS)*. 42–49.