CPR for SSDs

Bryan S. Kim
Seoul National University

Eunji Lee
Soongsil University

Sungjin Lee
DGIST

Sang Lyul Min
Seoul National University

Abstract

Modern storage systems are built upon the assumption that the capacity of a storage device does not change. This *capacity-invariant* interface forces a flash-based storage device to trade performance for reliability when, in fact, it can maintain both if a graceful reduction in capacity were to be allowed. We argue that relaxing the fixed capacity abstraction of the storage device allows for a better capacity-performance-reliability (CPR) tradeoff. We then outline existing device-internal mechanisms for building a *capacity-variant* flash device, and describe the necessary changes in the storage stack.

CCS Concepts • **Information systems** → **Flash memory**; • **Computer systems organization** → *Secondary storage organization*; • **Software and its engineering** → *File systems management*;

ACM Reference Format:
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), May 13–15, 2019, Bertinoro, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3317550.3321437

1 Introduction

A storage device provides a permanent home for persistent data, and it strives to maintain a high level of data integrity (low error rate) ideally forever (long mean time to failure). When it does inevitably fail, the device is expected to exhibit a *fail-stop* behavior [54] so that it can be identified and replaced, with its data hopefully replicated elsewhere or reconstructed through redundancy [51]. This model of data preservation is built around traditional hard disk drives (HDDs) that break down completely when their mechanical parts crash [52].

However, storage devices such as the flash-based solid state drives (SSDs) exhibit a *fail-partial* symptom. Flash memory manufacturers allow run-time bad blocks—physical memory blocks, not logical I/O blocks—in flash memory [2], and specify a limit to how many times a block can be programmed and erased [2, 45]. However, not all flash memory blocks are created equally: they physically wear (degrade) at different rates because of process variations. Furthermore, blocks become more error-prone as they wear [45], requiring SSDs to implement a variety of error correction and prevention techniques [9]. These techniques, in turn, cause performance overheads [29], making the fail-partial device's effort to project a fully healthy drive manifest in *fail-slow* symptoms [19, 22].

While it seems reasonable that the performance should be degraded to guarantee lifetime and data integrity, performance throttling represents an extreme version of this. A number of SSDs internally implement write throttling features to meet the device's lifetime warranty [1, 6, 35]. Furthermore, the responsibility of limiting writes is delegated to the user through conditional warranty restriction under *DWPD (drive writes per day)*, *TBW (terabytes written)*, or *GB/day (gigabytes written per day)* [4], and, even more, requiring client-side workload shaping such as the following anecdote [3]:

> ... [Customer A's] workload was not able to meet the 5 years life span. The only way to avoid such was setting the limit for the write performance.

We believe that the storage system and its ecosystem built around HDDs fundamentally limit the benefits of SSDs. The frame of guaranteeing lifetime for a fixed capacity inadvertently steers SSDs towards performance throttling [1, 6, 35] that simply negates their performant characteristics. This inefficiency extends even in a RAID subsystem [51] that is designed to protect against device-level failures: organizing SSDs in a RAID fashion has been shown to be detrimental in terms of both performance and lifetime for SSDs [8, 27, 46]. The crux of the issue is that the current block interface requires an SSD, a device that is inherently fail-partial and internally over-provisioned, to behave like an HDD that fails in an *all-or-nothing* manner.

HotOS '19, May 13–15, 2019, Bertinoro, Italy

ACM ISBN 978-1-4503-6727-1/19/05…$15.00
https://doi.org/10.1145/3317550.3321437

This paper argues that a *capacity-variant* block interface will relieve some of the challenges we face in building efficient SSD-based storage systems. By allowing the device's exported capacity to gracefully reduce, the device would be able to maintain a stable level of performance and reliability, and this would further make it unnecessary for the host system to implement performance-inefficient reliability enhancements. In doing so, the device and the system as a whole achieve a better tradeoff among capacity, performance, and reliability (CPR) for SSDs. There already exists a number of SSD-internal mechanisms for this tradeoff, and we argue that a capacity-variant storage system makes it easier to maintain a level of performance and prevent data loss disasters.

We explain the device's failure characteristics that motivate our proposal in § 2, and describe a capacity-variant block interface in § 3. We then give examples of existing in-device mechanisms for the CPR tradeoffs in § 4, and sketch how to build file and storage systems on top of these devices in § 5. We conclude by discussing our research plan in § 6.

2 Storage Device Failures

The *fail-stutter* work [7] points out that HDDs, in fact, do not entirely behave in a fail-stop model [54], and the subsequent studies [16, 52, 57] further explore this observation for HDD-based file and storage systems. However, maintaining a fixed capacity interface is more natural for HDDs than it is for SSDs because of the following reasons.

- Although HDDs perform internal re-mapping [7, 56], a high number of re-mapping count correlates strongly with an impending complete failure [42].
- Any mechanical failure such as the *head crash* leads to a permanent failure and irreparable damage to the device [52].

SSDs, on the other hand, have the opposite device-internal characteristics: mapping is natural and a commonplace task due to flash memory's out-of-place update requirement, and SSDs have no mechanical moving parts, only composed of solid state components.

A number of prior work study the failure trends of commercial SSDs in large production environments [43, 47, 55], but with differences in experimental methodologies and *what* constitutes as a failure. However, based on some of their insights, we argue the following.

- **Host-side efforts are inefficient.** The amount of host-side writes does not accurately indicate SSD-internal wear [43], and the amount of data errors have no correlation to the number of reads [55]. These results reveal the inefficiency of performance throttling [3, 6], and make it difficult to manage device failures based on the number of I/O operations.
- **SSDs hide too much.** Although flash memory blocks turning into unusable bad blocks and flash memory chips becoming no longer operational occur at a significant rate, the replacement rate of SSDs is much lower than that of HDDs [55]. This mix of information indicates that SSDs internally hide the fail-partial nature of flash memory through SSD-internal tasks such as wear leveling [14, 61] to a point where they have no other option but to fail-stop.

A study on performance faults in large-scale systems suggests that hardware vendors should expose more details on the device's error handling and statistics, but also recognizes the difficulty for this realization [19].

3 Honey, I Shrunk the SSD

A traditional storage device exports a fixed capacity through its block interface, as shown in Figure 1. An SSD is internally over-provisioned, and the internal re-mapping allows partial failures in the form of bad blocks and bad chips to be hidden. This reduces the underlying physical capacity, while the logical address space seen by the file system above remains fixed. The reduction in reserved space further accelerates wear by increasing the effective write amplification [17, 24], which in turn wears down the physical capacity faster: it is a vicious cycle.

In an attempt to break this cycle, the host system can indirectly leave a portion of the SSD unused [27]. This is done by never writing to parts of the address space or explicitly unmapping data through `TRIM`. However, it is never clear for the system above how much space should be freed and what the expected reliability benefits are, leaving much of it as guesswork.

We argue that, if the exported capacity were to be elastic, the device would make a better tradeoff among capacity, performance, and reliability (CPR). The device internally has better visibility on the state of the media by tracking the data's reliability-related parameters [29, 45]. Given its requirements, the device determines the necessary CPR tradeoffs to guarantee both performance and reliability.

In meeting the above conditions, we propose a capacity-variant block interface (Figure 2) that allows the device to recommend an address space (number of sectors) to the file system above, given the file system's requirement for

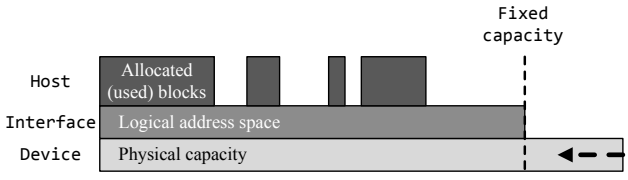
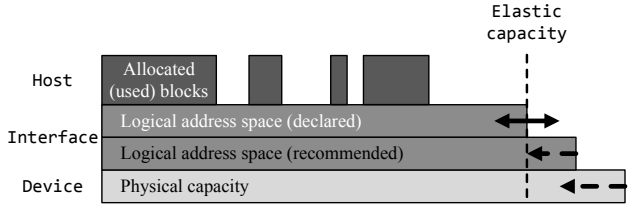
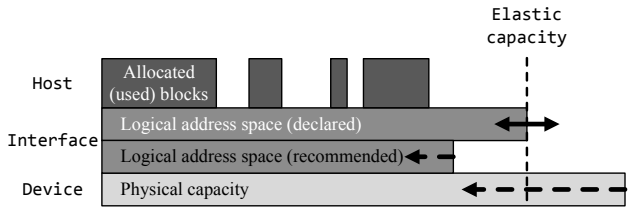


Figure 1. Fixed capacity block interface.



(a) The device meeting its guarantees.



(b) The device unable to meet its guarantees.

Figure 2. Capacity-variant block interface.

performance and reliability. The file system also declares a contiguous address space for use to the device below: this facilitates the device's decision for the CPR tradeoff. This is illustrated in Figure 2a and Figure 2b where the declared address space elastically changes depending on use, while the recommended may shrink. In Figure 2a, the recommended address space is larger than the declared, allowing the device to meet the performance and reliability requirements. Over time as the device ages, the recommended capacity may fall below the declared, and the device may not be able to guarantee its performance and reliability.

We propose necessary commands to the block interface for this extension. The file system declares its address spaces through the `Declare_Capacity` command, and the device returns the number of recommended sectors via the `Recommend_Capacity`. In addition, the file system provides performance and reliability requirements to the device through the `Set_Performance_Level` and `Set_Reliability_Level` commands.

`Declare_Capacity`
down: *device ID, number of declared sectors*
up: *status*
`Recommend_Capacity`
down: *device ID*
up: *number of recommended sectors*
`Set_Performance_Level`
down: *device ID, performance level*
up: *status*
`Set_Reliability_Level`
down: *device ID, reliability level*
up: *status*

The capacity-variant interface is general enough that it subsumes the traditional interface. If the file system were to ignore the recommended capacity throughout the lifetime of the device, the system would behave identically to the fixed capacity interface.

4 Capacity-Variant SSDs

In this section, we explore design decisions for building a capacity-variant SSD. First and foremost, a capacity-variant SSD does not perform wear leveling [14, 61]. Wear leveling aims to equalize the erase counts across memory blocks so that the SSD fails in an all-or-nothing manner. This is contrary to our goal of allowing partial failures. However, a capacity-variant SSD uses advanced internal management algorithms to predict media errors [29, 45], and to apply techniques that exhibit CPR tradeoffs to maintain both performance and reliability.

Table 1 outlines these SSD-internal techniques. They are orthogonal to each other and will complement one another if composed together. Among the eight, the first three (over-provisioning, redundancy, and SLC mode change) are the most relevant to the capacity-variant SSD, and will be discussed in detail. The remaining five techniques can also be deployed in conjunction with the first three to augment their strengths or to compensate for their weaknesses.

4.1 Over-Provisioning

SSDs internally reserve space not only to map out bad faulty blocks, but also to delay the need to reclaim space through garbage collection. A number of studies analytically show that increasing the over-provisioning factor trades capacity for performance [17, 24]. In terms of use cases, SSDs perform better when less data is stored [27], and ironically caching less data improves the overall performance of an SSD-HDD tiered storage [48].

In the context of CPR tradeoffs, we also need to consider reliability, and we believe errors in SSDs are more predictable than those in HDDs. The error rate for the data in flash memory is a function of its prior history of operations such as the number of erases, the number of reads, and the time since its last write [29, 45]. This makes it possible for an intelligent device to expect the number of errors and deploy reliability-related countermeasures. These techniques reduce the effective capacity of the device, so it becomes necessary for the capacity-variant SSD to procure more internally reserved space by recommending a reduced exported capacity.

Extending beyond a single host-single device setting, multi-tenancy presents an interesting challenge for guaranteeing performance and reliability. Under such environment where a single SSD is attached to multiple hosts, determining how to allocate its internal resources (such as internally

Table 1. An overview of SSD-internal techniques for CPR tradeoffs.

Techniques	Capacity	Performance	Reliability	Related work
Over-provisioning	↓	↑	-	... analytical studies [17, 24] ... use cases [27, 48]
Redundancy	↓	↓	↑	Dynamic striping [32, 36] Intra-block striping [49] Parity reduction [25, 38]
SLC mode change	↓	↑	↑	... for performance [34] ... for lifetime [59]
Integrity relaxation	-	↑	↓	Approximate storage [53] Read recovery level [5]
Data re-programming	-	↓	↑	In-place update [13] Data relocation [21]
Data re-read	-	↓	↑	LDPC [18, 39] Read retry [10, 12, 41]
Operational voltage tuning	-	↓ / ↑	↑ / ↓	Read pass-through voltage [11, 21] Program voltage [26, 40]
Data reduction	↑	↓	-	Compression [37, 50, 62] De-duplication [15, 20, 30]

reserved blocks) affects the performance perceived by each host [28, 31, 58]. We expect this problem to be increasingly important in the coming years, and holistically considering per-tenant CPR requirements at the device-level is necessary.

4.2 Redundancy

SSDs can internally add redundancy across multiple flash memory chips, similar to how RAID [51] protects data using multiple storage devices. SSD-internal redundancy is constructed using either the logical address [25, 38] or the physical address [32, 36, 49]. The logical address-based version is similar to that of a conventional RAID scheme: a stripe group is composed of data from consecutive logical addresses, defined by the stripe size, and the parity data is computed using the data from those addresses.

The physical address-based counterpart, on the other hand, creates a stripe group based on where the data is written. One parity data is created for every n data writes, and they are striped across different flash memory blocks, dynamically creating a stripe group. This is feasible in SSDs because of the need for internal indirection. Furthermore, the physically addressed redundancy makes it possible to increase and decrease the stripe size dynamically depending on the CPR requirement and the reliability state of the device-internal components.

Adjusting the degree of redundancy at deploy-time is particularly attractive for the capacity-variant SSDs. Storing the redundant data consumes additional space and the need to write the parity data reduces the overall performance, but the scheme allows the SSD to tolerate uncorrectable errors within a stripe group.

4.3 SLC Mode Change

High-density 2-bit multi-level cells (MLC), 3-bit triple-level cells (TLC), and even 4-bit quadruple-level cells (QLC) can be turned into 1-bit single-level cells (SLC) for better performance [34] and for lifetime extension [59]. When switching to SLC mode, the cell's physical properties do not change, only the interpretation for the electrons stored in the cells does. Memory operations perform faster as there are fewer voltage level distributions in a cell, and because of this, reading data in SLC is less error-prone.

Because SLC mode is simply how the data is interpreted (and not something intrinsic to the memory), the SSD-internal management is responsible for deciding if the memory is to be accessed in SLC mode or not. In a likely scenario where only a subset of data is maintained under SLC mode, the SSD must track how the data is interpreted and issue corresponding flash memory operations to switch back and forth between modes.

4.4 Other Techniques

There exists a number of techniques that do not trade capacity for performance or reliability. However, they can be used to further improve the performance and/or the reliability of an SSD.

- Approaches such as integrity relaxation re-defines how a storage device behaves when its data becomes corrupted. Approximate storage [53] returns approximate data for

error-tolerant applications, while read recovery level [5] allows the device to *fail-fast* by returning a data error early instead of spending time trying to recover it.
- Data re-programming is a preventive measure that identifies stale data and re-writes them either in-place [13] or at another location [21]. This approach is analogous to data scrubbing used in HDD-based systems [56].
- If a read error occurs at the memory level, subsequent re-reads can recover the data. LDPC (low-density parity-check) [18, 39] is an error encoding scheme that uses soft information (the probability of each bit being 0 or 1) for decoding, and read retry [10, 12, 41] finds the optimal threshold voltage for reading the data correctly.
- Flash memory can be accessed quickly (and lose reliability) or slowly (and ensure data is correctly written and read) by tuning its operational voltages. This can be done for both programs [26, 40] and reads [11, 21].
- Compression [37, 50, 62] and de-duplication [15, 20, 30] techniques can be performed inside the SSD to reduce the amount of data actually written in the memory. Although they incur computational overheads, the reduced amount of data written effectively increases the over-provisioning factor, which benefits the overall performance.

4.5 Discussion

SSDs internally use a number of techniques in order to guarantee a level of reliability, but the fixed capacity interface forces the device to sacrifice performance through the use of redundancy, data re-programming, data re-reads, and operational voltage tuning. A graceful reduction in capacity, however, allows the SSD to perform the reliability enhancement techniques with mitigated impact on the overall performance.

We expect a number of challenges in seeing a capacity-variant SSD in production and deployment. First, this interface makes it difficult to define product warranties because the performance, the error rate, and the lifetime all are a function of the capacity. Second, the file system and the device may act in a selfish manner, negating the benefits of the interface. The file system can declare the entire address space in use, making the device comply with the fixed capacity interface, or the device can un-gracefully reduce its recommended capacity. We believe these challenges are simply due to the schism between systems and devices, and both sides must work together towards making storage systems performant, reliable, and manageable.

Lastly, an interesting prospect is the self-healing property of flash memory [44, 60]. It has been shown that under high temperature, electrons unintentionally trapped in the insulator can be de-trapped, rejuvenating the cells. For practical use, we envision SSDs organized with detachable flash memory *pods* that can be hot-swapped for repair. This would, in a way, resuscitate aged SSDs (CPR for SSDs).

5 Using Capacity-Variant SSDs

We now discuss the implications in the storage stack for supporting capacity-variant SSDs. In particular, we consider file systems, RAID, and LVM (logical volume management).

5.1 File Systems

The proposed capacity-variant interface allows the file system to persist its data on top of an intelligent device that guarantees some level of performance and reliability. There are two issues to consider when using a capacity-variant device: elastically setting the appropriate declared space, and deciding what to do when the recommended space is reduced below the declared.

Although the file system can declare the entire address space for use from the start, this makes the elastic reduction difficult once the allocated space becomes fragmented. Copy-on-write file systems such as `Btrfs` support this more readily than others, but in general, shrinking a file system is a complex procedure that results in data loss if not done carefully. However, a conservative approach—where the declared address space is kept close to the amount of data—causes troubles of its own. The file system's block allocation logic will exhibit long tail latencies looking for unused blocks [23] and log-structured file systems such as `F2FS` will need to reclaim space more frequently, increasing the write amplification of the file system.

Now consider the choices when the device is old and aged. Although the device's `SMART` information warns the file system of the device's critical status, replacement is the only option in a fixed capacity interface. In the capacity-variant interface, this is also possible, but it offers another choice: reduction in capacity to maintain a level of performance and reliability. This is a preferred alternative over a possibly pre-mature device replacement, or keeping the fail-slow device.

5.2 RAID and LVM

RAID and LVM can abstract multiple devices into one, and capacity-variant SSDs present exciting prospects in building storage arrays.

Traditional RAID assumes that its storage components are homogeneous. However, this is not the case for HDDs [33], and certainly not for SSDs either. We are interested in exploring possibilities when this heterogeneity is taken even further: configuring SSDs to have different CPR traits. For example, in order to address correlated failures in SSD-based RAID arrays DiffRAID [8] skews the distribution of writes

among the devices, and thereby trades performance for reliability. Capacity-variant devices open up more possibilities in preventing correlated failures by constructing an SSD-based storage array with components with complementary CPR characteristics.

Unlike RAID that statically distributes data, LVM offers more flexibility and allows indirection. Under a scenario where a capacity-variant device reduces its recommended space to maintain reliability, LVM technology enables a seamless relocation of data from one device to another without file system dependency. Using capacity-variant devices with LVM, a gradual retirement of devices with near-zero recommended space is possible while maintaining both the overall performance and reliability.

6 Conclusion and Agenda

In this paper, we present a case for a capacity-variant block interface for a fail-partial device such as the SSD. SSDs export a block device interface by hiding the quirks of flash memory, but as the underlying memory becomes more error-prone, projecting an image of a fully healthy drive will make it more difficult in guaranteeing both performance and reliability. Prior large-scale studies have shown evidence that failure characteristics of SSDs are different from those of HDDs, and fitting a storage device that wears with use into an ecosystem built around a mechanical drive is becoming increasingly inefficient.

We argue that by allowing the SSD's exported capacity to gracefully reduce, the device can provide stable performance and reliability characteristics, and a number of SSD-internal mechanisms already exists for the CPR tradeoff. We also sketch how to build systems on top of these devices.

Going forward, we plan to investigate both device- and system-level side changes for the proposed interface.

- On the device side, we plan to quantify the exact performance and reliability improvements for each technique, and translate high-level requirements for the device into the appropriate composition of the techniques. The outcome would be a capacity recommendation.
- On the system side, we plan to develop an emulated device that gracefully reduces its capacity, and implement necessary modifications to the allocation logic for elastic online expansion and shrinking.
- RAID and LVM storage systems present an interesting opportunity for capacity-variant devices. We plan to explore the possibility of composing devices with complementary CPR traits to build a cost-effective, performant, and reliable storage array.

Acknowledgments

We thank the anonymous reviewers for their constructive and insightful comments. This work was supported in part by SK Hynix and the National Research Foundation of Korea under the Basic Science Research Program (NRF-2017R1D1A1B03031494, NRF-2018R1A5A1060031, and NRF-2017R1E1A1A01077410) and the PF Class Heterogeneous High Performance Computer Development (NRF-2016M3C4A7952587). Institute of Computer Technology at Seoul National University provided the research facilities for this study.

References

[1] 2012. SMART Storage Systems XceedIOPS2 200GB eMLC Enterprise SSD Review. http://www.thessdreview.com/. The SSD Review.

[2] 2014. ONFI 4.0. http://www.onfi.org/specifications/. Open NAND Flash Interface Specification 4.0.

[3] 2015. SSD Endurance. What Does It Mean To You? https://itpeernetwork.intel.com/ssd-endurance-what-does-it-mean-to-you/. Intel.

[4] 2017. Comparing Wear Figures on SSDs. https://thessdguy.com/comparing-wear-figures-on-ssds/. The SSD Guy.

[5] 2017. Solving Latency Challenges with NVM Express SSDs at Scale. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_SIT6_Petersen.pdf. Flash Memory Summit.

[6] 2018. Dell Solid State Drive FAQ with Servers and Storage. https://www.dell.com/support/article/sln156240/. Dell.

[7] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. 2001. Fail-Stutter Fault Tolerance. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany*. 33–38.

[8] Mahesh Balakrishnan, Asim Kadav, Vijayan Prabhakaran, and Dahlia Malkhi. 2010. Differential RAID: Rethinking RAID for SSD Reliability. In *European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems, EuroSys 2010, Paris, France, April 13-16, 2010*. 15–26.

[9] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. 2017. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proc. IEEE* 105, 9 (2017), 1666–1704.

[10] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2013. Threshold voltage distribution in MLC NAND flash memory: characterization, analysis, and modeling. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*. 1285–1290.

[11] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*. 438–449.

[12] Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. 2015. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. 551–563.

[13] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrián Cristal, Osman S. Ünsal, and Ken Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *30th International IEEE Conference on Computer Design, ICCD 2012, Montreal, QC, Canada, September 30 - Oct. 3, 2012*. 94–101.

[14] Li-Pin Chang. 2007. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*. 1126–1130.

[15] Feng Chen, Tian Luo, and Xiaodong Zhang. 2011. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*. 77–90.

[16] Timothy E. Denehy, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2002. Bridging the Information Gap in Storage Protocol Stacks. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference, June 10-15, 2002, Monterey, California, USA*. 177–190.

[17] Peter Desnoyers. 2012. Analytic modeling of SSD write performance. In *The 5th Annual International Systems and Storage Conference, SYSTOR '12, Haifa, Israel, June 4-6, 2012*. 14.

[18] Robert G. Gallager. 1962. Low-density parity-check codes. *IRE Trans. Information Theory* 8, 1 (1962), 21–28.

[19] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. 2018. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *16th USENIX Conference on File and Storage Technologies, FAST 2018, Oakland, CA, USA, February 12-15, 2018*. 1–14.

[20] Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam. 2011. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*. 91–103.

[21] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. 2016. An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory. *IEEE Trans. on CAD of Integrated Circuits and Systems* 35, 7 (2016), 1079–1091.

[22] Mingzhe Hao, Gokul Soundararajan, Deepak R. Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. 2016. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016*. 263–276.

[23] Jun He, Duy Nguyen, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2015. Reducing File System Tail Latencies with Chopper. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*. 119–133.

[24] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. 2009. Write amplification analysis in flash-based solid state drives. In *Proceedings of of SYSTOR 2009: The Israeli Experimental Systems Conference 2009, Haifa, Israel, May 4-6, 2009*. 10.

[25] Soojun Im and Dongkun Shin. 2011. Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD. *IEEE Trans. Computers* 60, 1 (2011), 80–92.

[26] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. 2014. Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling. In *Proceedings of the 12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014*. 61–74.

[27] Nikolaus Jeremic, Gero Mühl, Anselm Busse, and Jan Richling. 2011. The Pitfalls of Deploying Solid-State Drive RAIDs. In *Proceedings of of SYSTOR 2011: The 4th Annual Haifa Experimental Systems Conference, Haifa, Israel, May 30 - June 1, 2011*. 14.

[28] Bryan S. Kim. 2018. Utilitarian Performance Isolation in Shared SSDs. In *10th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2018, Boston, MA, USA, July 9-10, 2018*.

[29] Bryan S. Kim, Jongmoo Choi, and Sang Lyul Min. 2019. Design Tradeoffs for SSD Reliability. In *17th USENIX Conference on File and Storage Technologies, FAST 2019, February 25-28, 2019, Boston, MA, USA*. 281–294.

[30] Jonghwa Kim, Choonghyun Lee, Sang Yup Lee, Ikjoon Son, Jongmoo Choi, Sungroh Yoon, Hu-ung Lee, Sooyong Kang, Youjip Won, and Jaehyuk Cha. 2012. Deduplication in SSDs: Model and quantitative analysis. In *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20, 2012, Asilomar Conference Grounds, Pacific Grove, CA, USA*. 1–12.

[31] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*. 183–189.

[32] Jaeho Kim, Eunjae Lee, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2016. Chip-Level RAID with Flexible Stripe Size and Parity Placement for Enhanced SSD Reliability. *IEEE Trans. Computers* 65, 4 (2016), 1116–1130.

[33] Elie Krevat, Joseph Tucek, and Gregory R. Ganger. 2011. Disks Are Like Snowflakes: No Two Are Alike. In *13th Workshop on Hot Topics in Operating Systems, HotOS XIII, Napa, California, USA, May 9-11, 2011*.

[34] Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim. 2009. FlexFS: A Flexible Flash File System for MLC NAND Flash Memory. In *2009 USENIX Annual Technical Conference, San Diego, CA, USA, June 14-19, 2009*.

[35] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. 2012. Lifetime management of flash-based SSDs using recovery-aware dynamic throttling. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*. 26.

[36] Sehwan Lee, Bitna Lee, Kern Koh, and Hyokyung Bahn. 2011. A lifespan-aware reliability scheme for RAID-based flash storage. In *2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*. 374–379.

[37] Sungjin Lee, Jihoon Park, Kermin Fleming, Arvind, and Jihong Kim. 2011. Improving performance and lifetime of solid-state drives using hardware-accelerated compression. *IEEE Trans. Consumer Electronics* 57, 4 (2011), 1732–1739.

[38] Yangsup Lee, Sanghyuk Jung, and Yong Ho Song. 2009. FRA: a flash-aware redundancy array of flash storage devices. In *7th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2009, Grenoble, France, October 11-16, 2009*. 163–172.

[39] Shu Lin and Daniel J Costello. 2004. *Error Control Coding*. Prentice-Hall, Inc.

[40] Ren-Shuo Liu, Chia-Lin Yang, and Wei Wu. 2012. Optimizing NAND flash-based SSDs via retention relaxation. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*. 11.

[41] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2018. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. 504–517.

[42] Ao Ma, Fred Douglis, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor W. Hsu. 2015. RAIDShield: Characterizing, Monitoring, and Proactively Protecting Against Disk Failures. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*. 241–256.

[43] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. 2015. A Large-Scale Study of Flash Memory Failures in the Field. In *2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Portland, OR, USA, June 15-19, 2015*. 177–190.

[44] Neal Mielke, Hanmant P. Belgal, Albert Fazio, Qingru Meng, and Nick Righos. 2006. Recovery Effects in the Distributed Cycling of Flash Memories. In *44th IEEE International Reliability Physics Symposium, IRPS'06 San Jose, CA, USA, March 26-30, 2006*.

[45] Neal R. Mielke, Robert E. Frickey, Ivan Kalastirsky, Minyan Quan, Dmitry Ustinov, and Venkatesh J. Vasudevan. 2017. Reliability of Solid-State Drives Based on NAND Flash Memory. *Proc. IEEE* 105, 9 (2017), 1725–1750.

[46] Sangwhan Moon and A. L. Narasimha Reddy. 2016. Does RAID Improve Lifetime of SSD Arrays? *TOS* 12, 3 (2016), 11:1–11:29.

[47] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine M. Khessib, and Kushagra Vaid. 2016. SSD Failures in Datacenters: What? When? and Why?. In *9th ACM International on Systems and Storage Conference, SYSTOR 2016, Haifa, Israel, June 6-8, 2016*. 7:1–7:11.

[48] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2012. Caching less for better performance: balancing cache size and update cost of flash memory cache in hybrid storage systems. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*. 25.

[49] Heejin Park, Jaeho Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2015. Incremental redundancy to reduce data retention errors in flash-based SSDs. In *IEEE 31st Symposium on Mass Storage Systems and Technologies, MSST 2015, Santa Clara, CA, USA, May 30 - June 5, 2015*. 1–13.

[50] Youngjo Park and Jin-Soo Kim. 2011. zFTL: power-efficient data compression support for NAND flash-based consumer electronics devices. *IEEE Trans. Consumer Electronics* 57, 3 (2011), 1148–1156.

[51] David A. Patterson, Garth A. Gibson, and Randy H. Katz. 1988. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 1-3, 1988*. 109–116.

[52] Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2005. IRON file systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*. 206–220.

[53] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. 2013. Approximate storage in solid-state memories. In *The 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Davis, CA, USA, December 7-11, 2013*. 25–36.

[54] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (1990), 299–319.

[55] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. 2016. Flash Reliability in Production: The Expected and the Unexpected. In *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016*. 67–80.

[56] Thomas J. E. Schwarz, Qin Xin, Ethan L. Miller, Darrell D. E. Long, Andy Hospodor, and Spencer W. Ng. 2004. Disk Scrubbing in Large Archival Storage Systems. In *12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4-8 October 2004, Vollendam, The Netherlands*. 409–418.

[57] Muthian Sivathanu, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2004. Improving Storage System Availability with D-GRAID. In *Proceedings of the FAST '04 Conference on File and Storage Technologies, March 31 - April 2, 2004, Grand Hyatt Hotel, San Francisco, California, USA*. 15–30.

[58] Xiang Song, Jian Yang, and Haibo Chen. 2014. Architecting Flash-based Solid-State Drive for High-performance I/O Virtualization. *Computer Architecture Letters* 13, 2 (2014), 61–64.

[59] Ellis Herbert Wilson, Myoungsoo Jung, and Mahmut T. Kandemir. 2014. ZombieNAND: Resurrecting Dead NAND Flash for Improved SSD Longevity. In *IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, MASCOTS 2014, Paris, France, September 9-11, 2014*. 229–238.

[60] Qi Wu, Guiqiang Dong, and Tong Zhang. 2011. Exploiting Heat-Accelerated Flash Memory Wear-Out Recovery to Enable Self-Healing SSDs. In *3rd USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage'11, Portland, OR, USA, June 14, 2011*.

[61] Ming-Chang Yang, Yuan-Hao Chang, Tei-Wei Kuo, and Fu-Hsin Chen. 2016. Reducing Data Migration Overheads of Flash Wear Leveling in a Progressive Way. *IEEE Trans. VLSI Syst.* 24, 5 (2016), 1808–1820.

[62] Aviad Zuck, Sivan Toledo, Dmitry Sotnikov, and Danny Harnik. 2014. Compression and SSDs: Where and How?. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads, INFLOW '14, Broomfield, CO, USA, October 5, 2014*.