공학박사학위논문

초고용량 솔리드 스테이드 드라이브를 위한 신뢰성 향상 및 성능 최적화 기술

Optimizing the Reliability and Performance of Ultra-large SSDs

2021년 8월

서울대학교 대학원
컴퓨터공학부
홍두원

초고용량 솔리드 스테이드 드라이브를 위한 신뢰성 향상 및 성능 최적화 기술

Optimizing the Reliability and Performance of Ultra-large SSDs

지도교수 김 지 홍

이 논문을 공학박사 학위논문으로 제출함
2021년 6월

서울대학교 대학원
컴퓨터공학부
홍두원

홍두원의 공학박사 학위논문을 인준함
2021년 7월

위 원 장 _____ 유 승 주

부위원장 _____ 김 지 홍

위 원 _____ 김 진 수

위 원 _____ 이 재 욱

위 원 _____ 이 성 진

Abstract

The development of ultra-large NAND flash storage devices (SSDs) is recently made possible by NAND flash memory semiconductor process scaling and multi-leveling techniques, and NAND package technology, which enables continuous increasing of storage capacity by mounting many NAND flash memory dies in an SSD. As the capacity of an SSD increases, the total cost of ownership of the storage system can be reduced very effectively, however due to limitations of ultra-large SSDs in reliability and performance, there exists some obstacles for ultra-large SSDs to be widely adopted. In order to take advantage of an ultra-large SSD, it is necessary to develop new techniques to improve these reliability and performance issues.

In this dissertation, we propose various optimization techniques to solve the reliability and performance issues of ultra-large SSDs. In order to overcome the optimization limitations of the existing approaches, our techniques were designed based on various characteristic evaluation results of NAND flash devices and field failure characteristics analysis results of real SSDs.

We first propose a low-stress erase technique for the purpose of reducing the characteristic deviation between wordlines (WLs) in a NAND flash block. By reducing the erase stress on weak WLs, it effectively slows down NAND degradation and improves NAND endurance. From the NAND evaluation results, the conditions that can

most effectively guard the weak WLs are defined as the gerase mode. In addition, considering the user workload characteristics, we propose a technique to dynamically select the optimal gerase mode that can maximize the lifetime of the SSD.

Secondly, we propose an integrated approach that maximizes the efficiency of copyback operations to improve performance while not compromising data reliability. Based on characterization using real 3D TLC flash chips, we propose a novel per-block error propagation model under consecutive copyback operations. Our model significantly increases the number of successive copybacks by exploiting the aging characteristics of NAND blocks. Furthermore, we devise a resource-efficient error management scheme that can handle successive copybacks where pages move around multiple blocks with different reliability. By utilizing proposed copyback operation for internal data movement, SSD performance can be effectively improved without any reliability issues.

Finally, we propose a new recovery scheme, called reparo, for a RAID storage system with ultra-large SSDs. Unlike the existing RAID recovery schemes, reparo repairs a failed SSD at the NAND die granularity without replacing it with a new SSD, thus avoiding most of the inter-SSD data copies during a RAID recovery step. When a NAND die of an SSD fails, reparo exploits a multi-core processor of the SSD controller to identify failed LBAs from the failed NAND die and to recover data from the failed LBAs. Furthermore, reparo ensures no negative post-recovery impact on the performance and lifetime of

the repaired SSD.

In order to evaluate the effectiveness of the proposed techniques, we implemented them in a storage device prototype, an open NAND flash storage device development environment, and a real SSD environment. And their usefulness was verified using various benchmarks and I/O traces collected the from real-world applications. The experiment results show that the reliability and performance of the ultra-large SSD can be effectively improved through the proposed techniques.

Keywords: NAND Flash Memory, Flash Translation Layer, NAND Flash-Based Storage Systems, Embedded Systems, Performance Optimization, Lifetime Optimization

Student Number: 2017-36900

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Ultra-large (UL) SSDs (e.g., 32-TB SSDs in a 2.5-inch form factor [3]) are becoming popular these days in enterprise storage markets because of their advantages in reducing the total cost of ownership. As the capacity of a single SSD increases, fewer SSDs are needed to build a storage system. A smaller number of SSDs directly reduce various operating costs of storage systems, such as rack space, power, cooling, and storage-area networking costs.

The development of UL SSDs is made possible for two main reasons. The first is the continuous increase in the capacity of each NAND flash die, and the second is the development of NAND flash packaging technology and SSD controller design technology that can increase the number of NAND flash die in an SSD. Due to these technologies, the capacity of an SSD is continuously increasing, and it has the advantage of developing a cost-effective storage system. However, on the one hand, these capacity-enhancing technologies have introduced new drawbacks to the wider adoption of UL SSDs in practice.

New problems related to the capacity increase of UL-SSDs can be divided into three main categories. First, there is an issue in life-

time of 3D NAND used in UL-SSDs. As the 3D NAND continues to increase in capacity, the number of wordlines (WLs) in a block also increases, and the variation in characteristics between WLs increases. This characteristic deviation is a weakness that shortens the lifetime of UL SSDs as the lifetime of a block is limited by the weakest WL in the block. Another issue of UL SSDs is performance degradation. As the number of flash channels and ways of UL SSDs increases, the problem of performance bottlenecks increases when performing an internal data migration. As the number of UL SSDs in the storage system decreases to provide the same storage capacity, the performance required for a single UL SSD increases. However, if the internal bandwidth bottleneck becomes severe during the internal data migration, the host I/O processing performance will be degraded, resulting a performance bottleneck of an SSD. Finally, the increased number of NAND dies in an UL SSD increases the RAID recovery overhead in the storage system due to die failure and weakens the reliability of the storage system. Since the probability of NAND die failure is proportional to the number of dies in an SSD, and the recovery time of RAID is proportional to the capacity of the SSD, UL SSDs not only increase the probability of RAID rebuild process but also increase the probability of causing reliability problems such as secondary failure [2,4]) or read failure during the recovery process.

1.2 Dissertation Goals

Because the reliability and performance issues of an UL SSD are due to various reasons while increasing the capacity of the SSD, it is hard to improve them all with a single solution. Therefore, in order to satisfy various requirements for UL SSDs at the same time, multiple techniques that properly address each problem need to be developed and effectively integrated. Through this, the proposed technique to improve a specific problem should not cause problems in other aspects.

In this dissertation, we propose system-level approaches that improve the reliability and performance of UL SSDs, which overcomes the limitations of the existing techniques. In particular, our primary goal is to find new optimization hints which have not been exploited by the existing techniques, from the low-level NAND characterization to I/O characteristic of input workloads and SSD field failure characteristics. Then, we develop system-level optimization techniques to take advantage of these hints and improve the characteristics of UL SSDs in various ways.

First, we present an effective stress-relief solution for 3D NAND flash memory. Since erase operations are the main source of NAND flash wear-out [5], our solution is targeted to reduce the stress on weak WLs when performing erase operations. Because the large characteristic deviation between pages plays a negative role in limiting the lifetime of the block, by reducing the erase stress for the weak WLs, the deviation between pages in a block is reduced and the endurance of

the block is improved. From the evaluation using 3D NAND flash, we identify a relative erase stress reduction, and build a stress model that can maximize NAND endurance. Furthermore, in order to minimize the side effect of reducing the physical space due to the application of erase stress relief, we present a technique for selecting an optimal relief mode considering the workload characteristics.

Second, we present efficient internal data migration (e.g. GC or wear leveling) using a copyback operation. Since internal data copy operations directly interfere with I/O requests from user applications, how to efficiently handle internal data migrations is a key challenge for designing a high-performance SSD. Even though NAND copyback operation can effectively optimize the data migration overhead, the copyback is rarely used in modern SSDs due to reliability problem. In order to utilize copyback without reliability problem, we identify the error accumulation level of copyback according to the characteristics of each block and define the maximum number of consecutive copybacks. In addition, we present a resource-efficient error management scheme to fully utilize the copyback operation to increase performance without any side effect.

Third, we present a novel RAID recovery scheme that UL SSD failures are handled at the SSD level first before taking place a data recovery process at the RAID level. In contrast to small-sized SSDs whose capacity is few GB, it is feasible to repair failed NAND dies in UL SSDs. Since UL SSDs are composed of many NAND dies (e.g., 512 dies for a 32-TB UL SSD), failures of a few NAND dies do not badly

affect the reliability of an entire SSD and can be normally operated. Moreover, by leveraging data redundancy in RAID, UL SSDs are able to recover data of failed dies, while providing promised capacity to end-users. This self-recovery at a UL SSD level makes it possible to avoid the costly RAID reconstruction process as well as the hardware replacement. According to SSD field study [2], most SSDs fail not because their flash cells were worn over their endurance limit but because they experience unexpected component failures. Being the most dominant component of a UL SSD, NAND dies significantly contribute to such sudden failures in the UL SSD. Therefore, there is a strong incentive to devise a die-level SSD recovery scheme for UL SSDs.

1.3 Contributions

In this dissertation, we introduce three optimization techniques to improve the reliability and/or performance for Ultra-large capacity SSDs. The contributions of our work can be summarized as follows:

- We propose a new block erasing scheme called GuardedErase (or gErase) that can prolong the life of blocks by delaying weak WLs from reaching its maximum endurance level as an effective stress relief solution for 3D NAND flash memory. In order to extend the lifetime of weak WLs, GuardedErase employs two erase modes, normal erase mode and low-stress erase mode, at an individual WL level. When a WL is erased by the low-stress erase mode,

the WL experiences reduced wear stress from a block erase operation, thus effectively increasing its maximum number of P/E cycles. Using the low-stress erase mode, we performed an extensive characterization study for understanding the reliability impact of the low-stress erase mode and built a NAND endurance model for gErase-enabled NAND blocks based on the evaluation results. Based on our NAND endurance model, we have implemented the gErase-aware FTL, called longFTL, which dynamically changes the number of WLs erased by the low-stress erase mode while I/O performance is not affected. We evaluated the effectiveness of longFTL using various I/O traces. The evaluation results show that longFTL can improve the SSD lifetime by 21% over an existing gErase-unaware FTL with less than 3.1% decrease in the overall I/O performance.

- We propose an integrated approach that maximizes the efficiency of copyback operations to optimize the performance but does not sacrifice data reliability. Although our approach is based on the same motivation as FastGC [6], we improve the existing technique in two major aspects. First, we propose a novel perblock error propagation model under consecutive copyback operations. Our model aggressively exploits the aging characteristics of NAND flash memory in deciding the copyback threshold of a NAND block. (We call the maximum number of consecutive copyback operations allowed for a NAND block as the copyback thresh-

old of the NAND block.) By exploiting per-block differences during the run time, our model significantly increases the copyback threshold of most NAND blocks over FastGC. Second, we devise an efficient error management scheme that can handle successive copyback operations where pages move around multiple blocks with different reliability. When a page is migrated through blocks with different copyback thresholds, our scheme accurately maintains the remaining copyback balance of the page regardless of different copyback thresholds of migrated blocks. In managing the remaining copyback balance of a page, our scheme employs a perblock scheme instead of a more direct per-page scheme as used in FastGC. Unlike the common perception, the perblock management scheme, which can significantly reduce the memory and flash requirement over the per-page management scheme, improves both the performance and lifetime of SSDs.

- We propose a novel RAID recovery scheme, called reparo[1], which repairs UL SSDs from a die failure through efficient on-line die rebuild techniques. To the best of our knowledge, reparo is the first technique that repairs a failed SSD at the die level. In order to minimize the time to recovery from a die failure, reparo minimizes both a die failure detection time and a rebuild time. Whenever a bad block is detected, reparo checks its neighboring blocks to detect a die failure early. Once a failed die is detected,

[1] Reparo is a charm used to repair a broken object from Harry Potter.

multiple flash cores work in parallel to recover data in the failed die. Since a repaired SSD continues to be used, it is important for reparo to provide high performance and longer lifetime after the recovery. To this end, when rebuilding a failed die, reparo modifies a logical address-to-die mapping scheme in the way that minimizes space utilization imbalance among flash cores. This prevents performance degradation and lifetime drops which are caused by per-core space and workload variations. In order to verify the proposed technique, we implement reparo schemes on Samsung PM1643 SSD [3], and it showed that that the RAID recovery time is effectively reduced while minimizing the performance degradation and lifetime reduction after die failure recovery.

1.4 Dissertation Structure

This dissertation is composed of six chapters including the introduction and conclusions which are at the first and the last, respectively. The four intermediate chapters are organized as follows:

Chapter 2 briefly explains the background of our work, including basics of NAND flash memory and the overall architecture of SSDs. We also summaries the existing techniques to improve reliability or performance of an SSD which are highly related to our proposed techniques.

Chapter 3 presents an effective erase stress relief mechanism, called GuardedErase scheme, which reduces the characteristic variation

between WLs. We explain key reliability characteristics of 3D NAND flash blocks and show that low-stress mechanisms should be focused on erase operations. From the evaluation using the state-of-the-art 3D NAND flash, we figure out the effectiveness of low-stress erase by reducing erase voltage on specific WLs and describe the Guarded-Erase scheme. In addition, we propose a new system level technique, longFTL, which chooses optimal relief mode based on user workload characteristics. Evaluation results show that proposed scheme can improve the SSD lifetime without decreasing the overall I/O performance.

Chapter 4 presents effective data migration optimization scheme to increase the SSD performance using copyback operation. We propose an integrated approach that maximizes the efficiency of copyback operations while not compromising data reliability. From the error propagation characteristics of 3D NAND flash, we propose a novel per-block error propagation model under consecutive copyback operations. Furthermore, we devise a resource-efficient error management scheme that can handle successive copybacks where pages move around multiple blocks with different reliability.

Chapter 5 presents a new recovery scheme, called reparo, for a RAID storage system with ultra-large SSDs. Proposed reparo scheme repairs a failed SSD at the NAND die granularity without replacing it with a new SSD, thus avoiding most of the inter-SSD data copies during a RAID recovery step. By exploiting a multi-core processor of the SSD controller in identifying failed LBAs from the failed NAND

die and recovering data from the failed LBAs, the die failure recovery overhead is minimized.

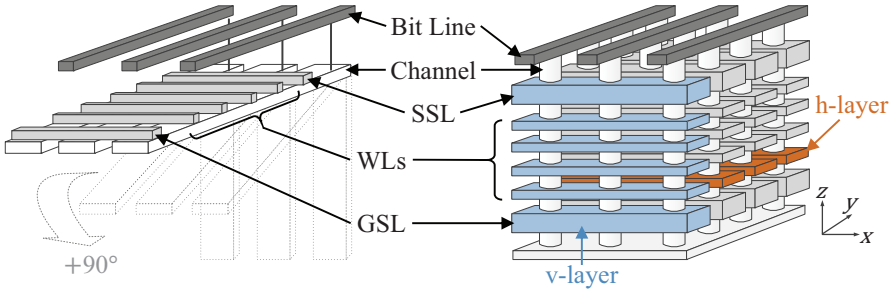
Chapter 2

Background

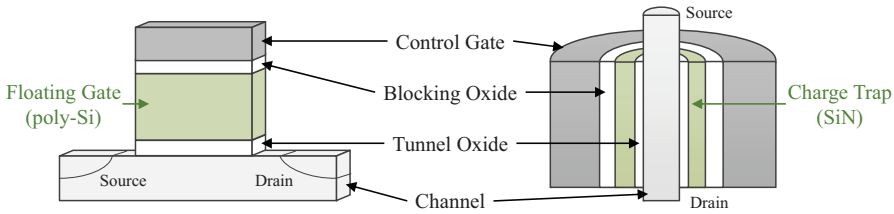
2.1 Overview of 3D NAND Flash Memory

3D NAND flash memory [7] enables continuous growth in the flash capacity by vertically stacking the memory cell to overcome various technical challenges in scaling 2D NAND flash memory. For example, 2D flash memory technologies had encountered the fundamental limits to scaling below the 10-nm process technology [8] because of the low device reliability (due to severe cell-to-cell interference) and high manufacturing complexity (e.g., high-resolution patterning). However, since 3D flash memory can integrate more memory cells by exploiting the vertical dimension even with less-resolution patterning technology, the flash capacity can be successfully increased by 50% annually while avoiding the reliability degradation [9].

3D NAND flash memory [7] enables continuous growth in the flash capacity by vertically stacking the memory cell to overcome various technical challenges in scaling 2D NAND flash memory. For example, 2D flash memory technologies had encountered the fundamental limits to scaling below the 10-nm process technology [8] because of the low device reliability (due to severe cell-to-cell interference) and high



(a) NAND organization.



(b) Cell structures.

Figure 1: Illustrations of differences between 2D NAND and 3D NAND [1].

manufacturing complexity (e.g., high-resolution patterning). However, since 3D flash memory can integrate more memory cells by exploiting the vertical dimension even with less-resolution patterning technology, the flash capacity can be successfully increased by 50% annually while avoiding the reliability degradation [9].

Compared to the conventional 2D NAND flash memory, there are two significant innovations in 3D NAND flash memory: *architectural innovation* and *material innovation*. Figure 1(a) shows the organiza-

tional difference in a NAND block[1] between 2D and 3D NAND flash memory. In this example, the 2D NAND flash memory has a matrix structure in which five WLs and three bitlines (BLs) intersect at 90 degrees. On the contrary, the 3D NAND flash memory has a *cube-like* structure. The 3D NAND block consists of four vertical layers (v-layers) in the y axis where each v-layer has four vertically stacked WLs that are separated by select-line (SSL) transistors. As shown in Figure 1(a), when the 2D NAND block is rotated by 90 ° in a counter-clockwise direction using the x axis as an axis of rotation (i.e., if the WLs are set vertically), it corresponds to a single v-layer. Similarly, the 3D NAND block can be described to have four horizontal layers (h-layers) which are stacked along the z axis, and each horizontal layer consists of four WLs. By increasing the number of v-layer of 3D NAND flash memory (i.e., stacking more h-layers along the z axis), the total number of WLs in a NAND block is effectively increased. This advantage of scalability enables 3D NAND flash memory to continuously increase its capacity by breaking through the manufacturing limit (e.g., lithography and patterning).

Another key innovation in 3D flash memory is a change in the type of NAND cells where charge is stored. Most 3D NAND devices (e.g., TCAT [10], p-BICs [11] and SMArT [12]) adopt cylindrical charge trap (CT)-type cell structures, also known as *gate-all-around* (GAA) [13]. This CT-type cell uses a non-conductive layer of silicon

[1]NAND flash memory consists of multiple blocks. Each block has multiple WLs (e.g., 128 - 256 WLs) and each WL consists of a group of flash cells (e.g., 8K - 16K cells).

nitride (SiN) that traps electrical charges to store bit information, while 2D NAND devices use floating gate cell structures which store bits in a conductor (e.g., poly-Si). As shown in Figure 1(b), each layer constituting CT-type cell (tunnel oxide, SiN, blocking oxide, and control gate) wraps a vertical channel made of poly-silicon, thus forming a three dimensional structure. These differences between 2D and 3D NAND flash memory require new optimization techniques for efficient flash-based storage systems.

2.2 Reliability Management in NAND Flash Memory

In order to reliably store data in flash cells, various reliability requirements should be satisfied. For example, NAND blocks should be used up to their limited maximum P/E cycles only. This is because the NAND cell characteristics in the NAND block deteriorate as the P/E cycle increases. The main cause of wear-out of a NAND block is electrical stress on the tunnel oxide during the execution of the program and erase operations. As program/erase operations are repeatedly performed on the block, the amount of charges trapped in the tunnel oxide layer increases. The trapped charges adversely affect the data retention characteristic of NAND cells in the block and make it difficult for the threshold voltage level (i.e., state) of the erased NAND cells to be located within their intended voltage interval [14]. Therefore, as the P/E cycle increases, the BER characteristics of data

stored in the NAND block deteriorate. In order to prevent the number of error bits from surpassing the correction capacity of an error correction code (ECC) engine, NAND manufacturers set the maximum number of P/E cycles allowed for a block, which is often called as NAND endurance. If the block continues to be used over the maximum number of P/E cycles, the BER of the block will exceed the correction capability of the ECC engine, which results in data loss (i.e. a read error). In addition to errors due to NAND wear-out, various errors (such as program errors and erase errors) can occur due to process defects during a NAND flash manufacturing procedure [15].

Although NAND operations can fail for different reasons, failed operations are managed in the block granularity within an FTL. For example, if a read operation to a page in a block B fails, the FTL identifies the block B as a bad block and replaces B with a reserved block. After the BBM module of the FTL moves all the valid data from the bad block B to the reserved block, the bad block B is no longer used.

2.3 UL SSD architecture

Since a UL SSD needs to support high performance for its huge storage space, a high performance multi-core processor is used for its SSD controller. Figure 2 shows an organizational overview of a typical UL SSD architecture. The SSD architecture consists of an $(N+1)$-core multi-processor, DRAM/SRAM memory, a host interface logic, and a

large number of NAND dies (which are grouped into different channels). For example, in Samsung enterprise SSD with 32-TB capacity, a quad-core ARM based processor is used to manage 512 512-Gib dies, which are organized into 16 channels. In order to exploit the parallelism supported by the multi-core processor without a high management complexity, each core is often dedicated to handling a specific set of tasks. As shown in Figure 2, the master core is responsible for interfacing with the host system while the flash cores, $core_0, ..., core_{N-1}$, are assigned to flash management tasks. When the host system sends I/O requests to the UL SSD, the master core distributes the I/O requests across flash cores. To make flash management simpler, each flash core is dedicated to specific NAND dies. For example, when N flash cores are used, $\frac{1}{N}$ of the NAND dies are equally assigned to each flash core. Given a logical block address (LBA), a simple address stripping method is used to decide a target flash core, $core_{target}$ (i.e.,

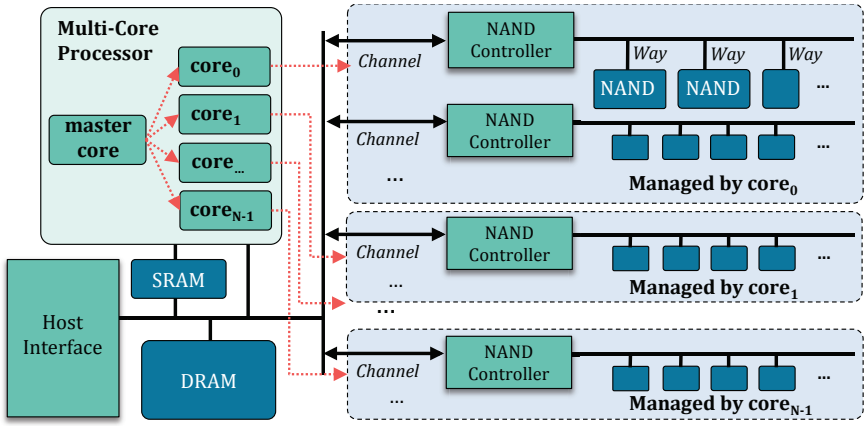


Figure 2: An overall organization of an SSD architecture.

$core_{target} = LBA mod N$). Since all the LBAs assigned to the same flash core have the same last $\lfloor log_2^n \rfloor$ bits, each flash core ignores the common last bits internally for resource optimizations (e.g., the L2P mapping table reduction). The main advantage of the static stripping method, which works well for most I/O workloads, is that it is straightforward to manage because each flash core works *in an independent fashion.*

2.4 Related Work

2.4.1 NAND endurance optimization by utilizing page characteristics difference

There have been several studies to improve the NAND endurance that has been reduced due to the NAND process scaling or multi-leveling technologies [5, 16, 17]. Jeong *et al.* conducted a study to improve the NAND endurance by adjusting the erase voltage and erase time [5]. It is similar to our study in that it improves lifetime through the reduction of stress caused by erase operation. However, their research aims to perform a stress reduction on the entire block, which is not related to a reduction in the variance of page characteristics within the block. Jimenez *et al.* focused on the variation between pages within a block and conducted a study to improve NAND endurance through a program stress relief technique [16]. However, its effect is limited in that the main cause of NAND wear-out is erase operation, not program operation. Our approach differs from their study in that we suggested a stress relief effect through a more effec-

tive erasing operation, and also achieved an optimal relief level at the system perspective, taking into account the effects of WAF increasing as a side effect of relief. Debao *et al.* proposed a bad page management (BPM) technique that improves the lifetime by continuously using the remaining pages instead of performing BBM of the entire block when a read error occurs on the specific page [17]. Considering the page variation within block, BPM is a valid approach in that the rest of the pages can still be utilized. However, there is a limitation in that the performance degradation and lifetime reduction were not considered as a side effect of increased utilization when applying BPM.

2.4.2 Performance optimizations using copyback operation

There have been several studies to improve the performance of flash-based storage systems with the copyback operation. However, many existing techniques [18–20] are not applicable for modern NAND flash memory because they assumed an ideal SLC NAND flash memory where no error propagation occurs from successive copyback operations. Other studies such as Jang *et al.* [21] considered the error propagation problem in their techniques. However, their solutions was to bring data out to the ECC module to check the validity of data, thus minimizing the potential benefit of using copyback. In recent study of Wu *et al.* [6], they proposed a technique that can use copyback without error propagation based on NAND characterization for the first time. However, there is a lot of room for improvement because their

method is a naive approach and there is overhead for error propagation management. Our technique differs from the existing technique in that it maximizes the potential benefits of copyback by taking both block characteristics and host workload characteristics into account, and has full control over error propagation issues with minimal overhead.

2.4.3 Optimizations for RAID Rebuild

Fast RAID recovery techniques have been extensively investigated in enterprise storage systems [22–27]. For example, several groups have focused on devising efficient data layout methods that can reduce the impact of a RAID rebuild process on normal I/O requests from a host. The parity declustering layout was proposed by Muntz and Lui [22] to shorten rebuild time and improve user response by minimizing the number of disks required for reconstructing a failed disk so that the rest of disks can continue to handle host requests. It was implemented and evaluated in an accurate simulator environment by Holland and Gibson [23], and improved/extended by further researches [24, 25].

Wan *et al.* [26] proposed a skewed sub-array organization in a RAID structure, which splits large disks into small logical disks to form sub-arrays but are configured to be skewed among physical disks. This enables a RAID rebuild process to be performed on multiple physical disks in parallel without access conflicts.

Although these schemes can reduce the total rebuild time by intel-

ligently overlapping rebuild operations with host request processing, they do not reduce the total amount of data that need to be read for a RAID reconstruction task. Rebuild Assist [28] takes a different approach to expedite the RAID rebuild process. When an SSD fails, Rebuild Assist distinguishes the failed LBAs from the readable LBAs in the failed SSD. For the latter, Rebuild assist simply copies their data from the failed SSD to a replacement SSD without rebuilding them using a RAID scheme, thus reducing data reads from the rest of RAID storage. Reparo, which is based on a subset of new commands proposed by Rebuild Assist, is fundamentally different from Rebuild Assist in that reparo does not build a replacement SSD but repairs failed dies.

2.4.4 Reliability improvement using internal RAID

Reparo is similar to RAIN (Redundant Array of Independent NAND) techniques [29–35] in that they can recover a failed die inside a failed SSD. However, the existing RAIN techniques work at the individual SSD level rather than the RAID storage level, making them very difficult to use efficiently in a RAID storage system. For example, when a storage system is consist of RAIN-enabled SSDs, if an SSD fails (although such an SSD failure is much less likely because of an internal RAID configuration in a RAIN-enabled SSD configuration), its RAID recovery procedure will be as slow as that of a RAID storage system with normal SSDs. Since RAID should be supported in an individual SSD, the existing RAIN techniques incur a

significant resource overhead (e.g., OP space reduction) as well as a flash lifetime degradation [36]. Therefore, the performance/lifetime of RAIN-enabled SSDs is poorer than SSDs without RAIN support. Furthermore, when a RAIN-enabled SSD is recovered after a die failure using a RAIN scheme, the OP space of the RAIN-enabled SSD will be further reduced, thus quickly degrading the performance/lifetime of the SSD. Since we are interested in continuing a normal operation of a RAID storage system after a failed die is recovered without replacing a failed SSD, we did not consider the RAIN techniques as a viable alternative solution for repairing failed dies. On the other hand, reparo, which was proposed for a RAID recovery purpose, imposes little overhead on an individual SSD level while it can minimize the impact of the die-level recovery on the performance and lifetime of the repaired SSD.

Chapter 3

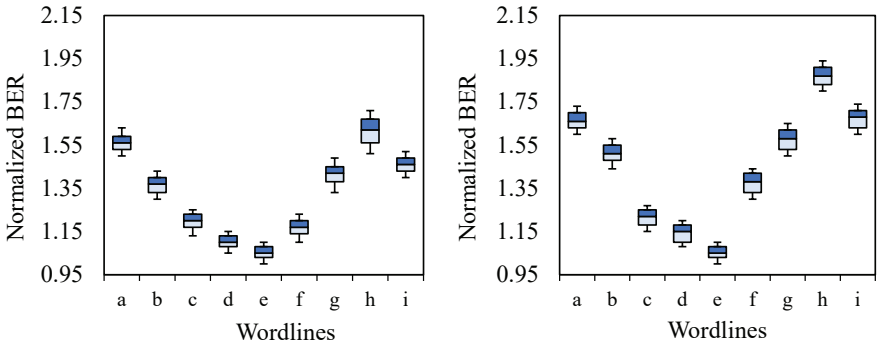
GuardedErase: Extending SSD Lifetimes by Protecting Weak Wordlines

3.1 Reliability Characterization of a 3D NAND Flash Block

In this section, we explain two key observations on 3D NAND flash characteristics which motivated the proposed GuardedErase scheme.

3.1.1 Large Reliability Variations Among WLs

Ideally, we would expect all flash cells in a block (or chip) to have identical characteristics. However, in practice, significant electrical/physical characteristic variations between flash cells are unavoidable due to many unexpected process effects during NAND flash manufacturing. Even in 2D NAND flash memory, the reliability has been known to vary depending on the physical location of WLs within a flash block, but this variation is much severe in 3D NAND flash memory. To quantify the reliability variations between WLs, we examined the inter-WL BER (Bit Error Rate) variations using 160 state-of-the-



(a) Initial P/E cycles.

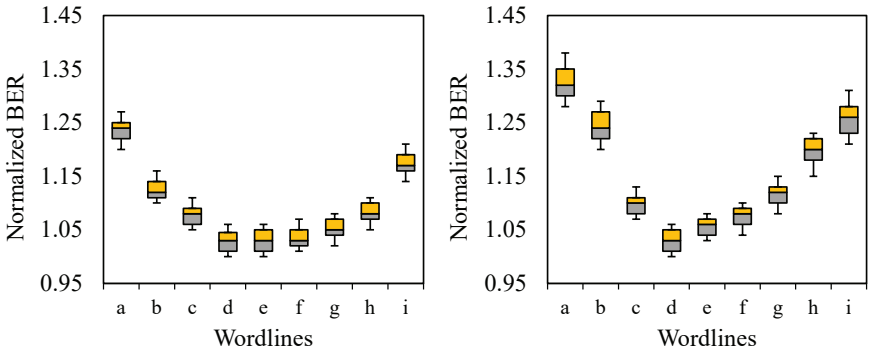
(b) 3K P/E cycles.

Figure 3: BER variations of 3D NAND flash.

art (48-layer) 3D TLC flash chips[1]. To minimize the potential distortions in the evaluation results, we evenly selected 120 test blocks from each chip at different physical block locations, and tested all the WLs in each selected block. We tested more than 10^6 WLs (a total of 3,686,400 WLs) to obtain statistically significant experimental results.

Figure 3(a) shows significant inter-WL BER variations exist within a tested block even when the blocks experience no program/erase cycles. All BER values were normalized over that of the most reliable WL. The BER of the worst WL (near the top layer) is about 60% higher than that of the best WL (near the middle layer). When flash blocks wear-out (i.e., they experience a large number of program/erase cycles), the BER variations between WLs get larger, so that the BER difference between the best and worst WL exceeds two times, as shown in Figure 3(b). We also examined the inter-WL reliability variations of

[1]Our flash chips are fabricated by 3D VNAND CT (Charge trap) technology which is known as *SMArT* [37] or *TCAT* [38].



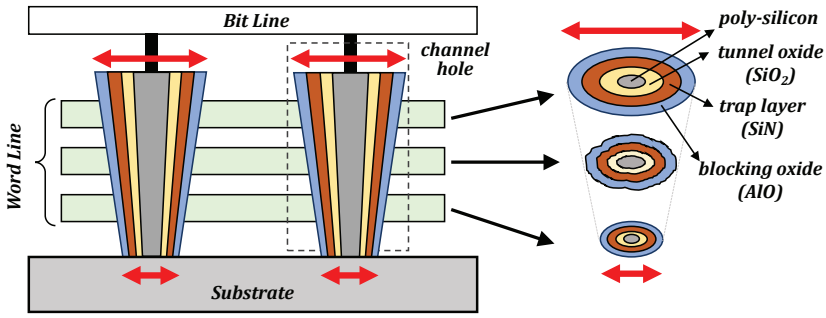
(a) Initial P/E cycles.

(b) 3K P/E cycles.

Figure 4: BER variations of 2D NAND flash.

1x-nm 2D TLC NAND flash memory. Unlike 3D NAND flash memory, BER increases toward edge WLs (e.g., WL0 or WL64), and the BER difference between WLs does not change much even if flash blocks wear out (shown in Figures 4(a), 4(b)).

The root cause of large reliability variations is related to a unique manufacturing process to form the vertical architecture of 3D NAND flash memory. Figure 5(a) shows a detailed organization of a vertical layer in 3D NAND flash memory using a cross-sectional view along the y-z plane and a top-down view (of three cross sections along the x-y plane). The stacked cells are vertically connected through cylindrical channel holes, called as *pillar*. The channel holes are formed at the early stage of 3D NAND flash manufacturing by an etching process [10]. After forming channel holes, each layer constituting a 3D flash cell (e.g., tunnel oxide, SiN layer, and blocking oxide in Figure 1(b)) is deposited around a channel hole in order. So, the shape



(a) A cross-sectional view. (b) A top-down view.

Figure 5: Inter-WL variations in 3D NAND flash memory.

of the channel holes is the key factor to determine the structure of 3D flash cells.

Ideally, we expect that each channel hole has the same geometrical structure regardless of its physical location to achieve the homogeneous reliability characteristics among flash cells. However, due to a high *aspect ratio*[2], the cylindrical channel hole cannot avoid suffering from severe structural variations depending on its vertical (z-directional) location. For example, as shown in Figure 5(b), the diameter of the channel holes varies significantly over the height of an h-layer. During etching process, the deeper the depth of the channel hole, the fewer etchant ions it reaches and the less chance of reaction for etching. Therefore, the channel hole diameter in the topmost h-layer is wider than that in the bottom h-layer (i.e., the flash cell in the topmost h-layer has larger cell size than that in the bottom h-layer.).

[2]The aspect ratio of a channel hole can be defined as the ratio of the width to height. In 64-layer 3D NAND flash memory, the aspect ratio of channel holes can be more than 60:1.

Furthermore, the shape of channel holes in some h-layers is an ellipse or a rugged shape unlike the circle in upper h-layers mainly due to the variance of etchant fluid dynamics over vertical positions.

Different channel hole diameters as well as their shapes can cause large variations in the characteristics of flash cells. Therefore, even when the same program/erase voltage is applied to flash cells, they experience different electrical stress depending on the diameter or shape of a channel hole, resulting in different reliability characteristics along vertical locations.

3.1.2 Erase Stress on Flash Reliability

Before we design a WL-level stress mitigation technique for 3D NAND flash memory, in order to quantitatively understand the main source of flash stress, we performed a comprehensive characterization study using real 160 3D TLC flash chips with 48 horizontal layers where each layer consists of 4 WLs. Since both a program operation and an erase operation incur a significant amount of wear-out stress on flash cells, the main goal of our study was to measure the relative impact of these operations on the reliability characteristics of 3D flash cells. In order to minimize potential distortions in the evaluation results, we evenly selected 120 test blocks from each chip at different physical block locations and tested all the WLs in each selected block. We tested a total of 3,686,400 WLs (11,059,200 pages) to obtain statistically significant experimental results. Following a standard evaluation metric commonly used in NAND flash reliability studies,

Table 1: A summary of three operation sequences.

Sequence type	Operations order
`Modified Base Sequence`	P → dummy P → E → dummy E
`Erase-only Sequence`	dummy P → E → dummy E
`Program-only Sequence`	dummy E → P → dummy P

P : program, E : erase

we measured changes in RBER (Raw Bit-Error Rate) after each measurement scenario.

To quantify the impact of the program and erase operations on the flash endurance stress, we designed three measurement scenarios. In each scenario, one of three operation sequences is repeated until the lifetime limit of the block. Table 1 summarizes three operation sequences with their member operations.

Note that all three sequences include dummy operations so that unexpected factors do not affect the accuracy of measurement results. [3] As expected, the endurance impact of an erase operation was significantly larger than that of a program operation in 3D NAND flash memory. The endurance stress of erase operations was responsible for almost 80% of the total stress of flash cells in 3D NAND flash memory. The relative stress impact of erase operations was almost equal

[3] As explained in Section 2.2, the flash cell's wear-out is mainly caused by trapped charges in the tunnel oxide layer during program and erase operations. If the flash cell is sufficiently erased, there is little charge remaining that can be transferred from the SiN layer to the substrate. Therefore, for example, repeating only erase operations in `Erase-only Sequence` cannot accurately measure the endurance effect of the erase operation. As shown in Table 1, dummy program operations are performed before erase operations. Dummy erase operations were added to the sequence because they were included in `Modified Base Sequence` so that the effect of erase operation can be effectively measured by comparing BER values of two sequences.

regardless of WLs. Our measurement study clearly indicates that a low-stress mechanism should be focused on erase operations, not program operations.

3.2 GuardedErase: Design Overview and its Endurance Model

3.2.1 Basic Idea

Since an erase operation is a major source of the flash wear-out, it is important to reduce the erase stress on flash cells when they are erased if the flash cells can be used for more P/E cycles. In order to reduce the erase stress when necessary, GuardedErase supports the low-stress erase mode as well as the normal erase mode. In order to extend the lifetime of a flash block, we apply the low-stress erase mode to weak WLs that may reach their endurance limit in a near future.

Consider an example 3D flash block with 10 WLs of Figure 6. Reflecting strong process variability among WLs, there are significant variations on BER values for the same number of P/E cycles. For example, when the normal block erase operation is used, WL 0, which is the weakest WL, reaches its maximum BER value (i.e., the BER threshold value of the block) after 10 P/E cycles. On the other hand, after 10 P/E cycles, WL 5, which is the strongest WL, barely reached 50% of its maximum BER value. However, since WL 0 reached its maximum BER value, the block is no more usable, thus becoming a bad block. If we employ a fine-grained BPM policy, the block can

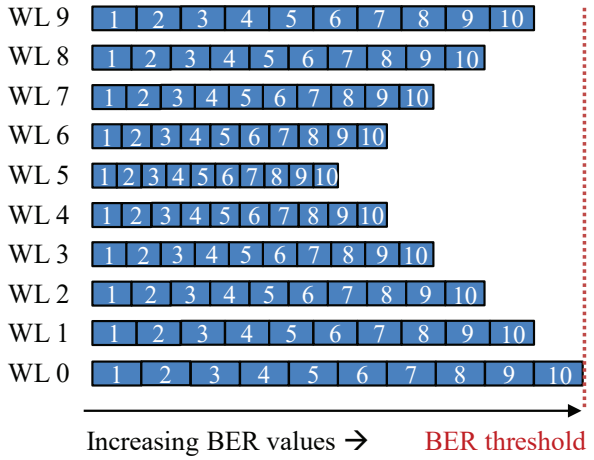


Figure 6: Per-WL BER changes under the normal block erase.

continue to be used with nine WLs after 10 P/E cycles. However, since the effective block capacity is reduced by 10% (which, in turn, may introduce a severe WAF increase), the BPM policy is rather limited in increasing the total amount of written data to the block while incurring no I/O performance degradation. In the example block in Figure 6, even if an SSD can tolerate up to 20% of the average block capacity reduction (thanks to its OP space), only 9 more WLs can be written to the block before WL 1 becomes unusable after one more block erasure.

Figure 7 illustrates how the lifetime of the example block can be extended using the proposed GuardedErase scheme. When BER values of weak WLs (such as WL 0, WL 1 and WL 9) are higher than those of other WLs, weak WLs are erased using the low-stress erase mode whose stress on flash cells is assumed to be 1/3 of that of the normal

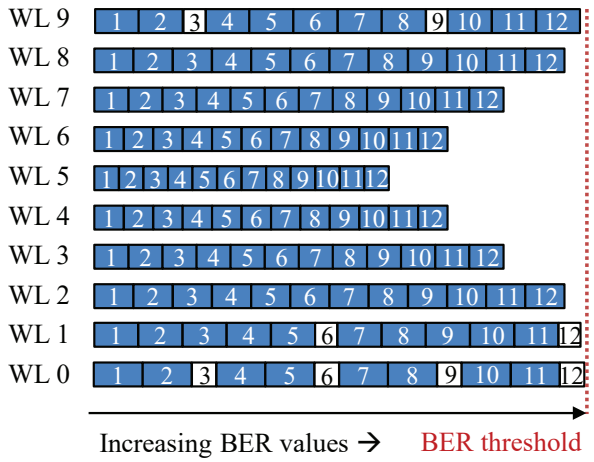


Figure 7: Per-WL BER changes under the gErase block erase.

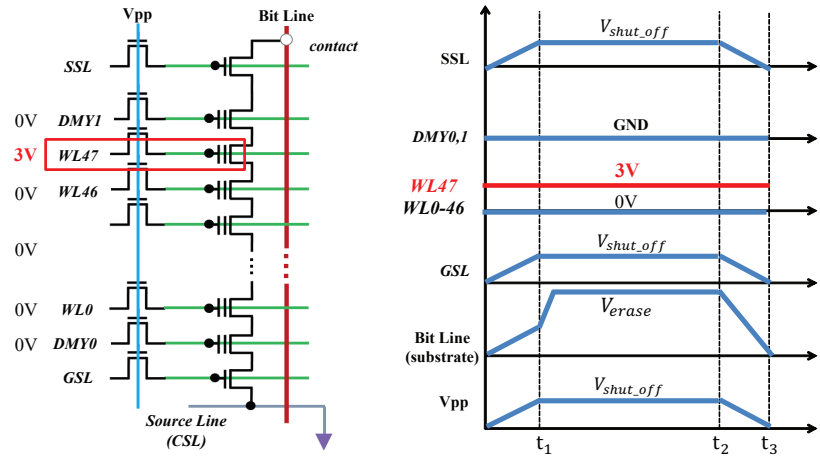
erase mode. For example, after the 1st erase cycle, the BER value of WL 0 is more than double that of WL 5. In order to protect WL 0, the low-stress erase mode, which is indicated by a white box in Figure 7, is used for WL 0 in the 2nd erase cycle. WL 0 is erased six more times using the low-stress erase mode under similar conditions. Figure 7 shows that the lifetime of WL 1 and WL 9 is extended to the 14-th erase cycle with five applications of the low-stress erase mode while that of WL 2 and WL 8 is extended to the 14-th erase cycle with two applications of the low-stress erase mode. When weak WLs are erased using the low-stress erase mode, they cannot store data for the following program cycle. For example, the effective total capacity of the block after the 2nd erase cycle is reduced to 7 WLs from the original 10 WLs. Similarly, the effective total capacity of the block after the even-numbered P/E cycles (e.g., the 4-th erase cycle and

the 6-th erase cycle) is 7 WLs only because three of five WLs, WL 0, WL 1, WL 2, WL 8 and WL 9, cannot be properly programmed. However, since the maximum number of P/E cycles for the block is increased from 10 to 14, the total amount of written data to the block effectively increases by 19% to 119 WLs.

Note that the GuardedErase scheme outperforms the BPM scheme in the total number of WLs written to the block by writing 10 more WLs. Furthermore, unlike the BPM scheme where the effective block capacity is monotonically non-increasing over P/E cycles, the GuardedErase scheme can dynamically control the effective block capacity up to the maximum block capacity depending on which erase mode is used. If the low-stress erase mode can be adaptively applied by exploiting the future write demand and intensity characteristics, the GuardedErase scheme can efficiently extend the SSD lifetime without an I/O performance degradation. In order to achieve the full potential of the GuardedErase scheme, therefore, we need to design an efficient mechanism for supporting the low-stress erase mode and devise an intelligent management policy of applying the low-stress erase mode to proper WLs at the right time.

3.2.2 Per-WL Low-Stress Erase Mode Implementation

There are two implementation options for the low-stress erase mode. One is to reduce the erase time (i.e., erase time scaling [39]) and



(a) Differential voltage driving to control gates.

(b) Voltage changes over time for a block erase.

Figure 8: An implementation of the low-stress erase mode.

the other is to reduce the erase voltage (i.e., erase voltage scaling [39]). However, since it is not easy to control the erase time at the WL granularity, we employed a scheme that lowers the erase voltage. In order to reduce the erase voltage for a specific WL, we exploited a test-mode command [40] for 3D NAND flash memory that allows to vary the driving voltage setting at the WL granularity. Although we cannot directly apply a lower erase voltage to a specific WL, this test-mode command can be used to effectively lower the erase voltage for the WL.

Figure 8(a) illustrates how to reduce the erase stress on WL 47 using the proposed method. Since the voltage difference between the control gate voltage and the voltage applied to the substrate acts as the effective erase voltage, when a higher voltage is applied to the

control gate of a WL, its erase voltage is effectively reduced, thus reducing the erase stress on the WL. In Figure 8(a), 3V, not the usual 0V, is applied to the control gate of WL 47, thus reducing the effective erase voltage of WL 47 by 3V. Figure 8(b) shows how various voltages are driven when the low-stress erase mode is used for WL 47 while the rest of WLs are erased using the normal erase mode. When the nominal erase voltage V_{erase} is applied to the bit line (or substrate) from time t_1 to t_2, the NAND flash cells of WL 47 experience a smaller electrical potential difference by 3V over the flash cells of the other WLs. Considering that V_{erase} is approximately $17V$ and the erase stress is exponentially proportional to an electrical potential difference, a significant amount of erase stress is reduced to the NAND flash cells of WL 47 during an erase operation[4].

Stress Mitigation Effect

In order to understand the endurance impact of the low-stress erase mode on WLs, we performed a comprehensive process characterization study using state-of-the-art 3D TLC NAND flash chips. For endurance comparisons, we formed two block groups whose member blocks were evenly selected from different physical locations using 30 NAND flash chips. Two block groups were erased using different erase modes, one group using the normal erase mode only and the other group using the normal erase mode and the low-stress erase mode

[4]When the low-stress erase mode is applied to a WL, we increase the verify voltage setting for the WL so that the erase operation can take the same number of erase loops as in the normal erase mode.

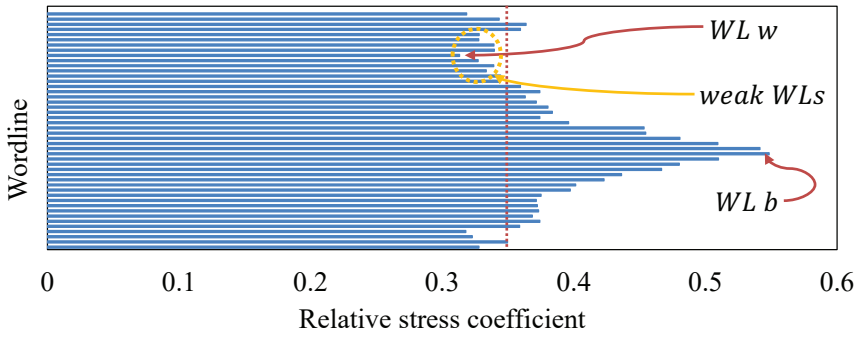


Figure 9: Per-WL relative stress coefficients.

half and half, respectively. For each WL of a tested block, we collected the maximum number of P/E cycles that keeps the BER value of the WL below the BER threshold value.

Based on the measured maximum P/E cycles per WL, we computed the relative stress coefficient S_k of WL k that indicates how much the erase stress is mitigated when the low-stress erase mode is used. S_k is a per-WL quantity that indicates how much WL k wears out when it is erased using the low-stress erase mode over the normal erase mode. For example, if the maximum number of P/E cycles of WL p was 10K when the normal erase mode was used, but it was increased to 20K when the low-stress erase mode is used, the relative stress coefficient S_p is 0.5. Figure 9 summarizes our measured relative stress coefficients for different WLs in a block. Although there are significant differences in S_k values depending on WLs, we observed that the relative stress coefficients of weak WLs (for which most of the low-stress erase mode are applied) are quite similar: all the co-

efficients belong to an interval [0.31, 0.34] with the mean of 0.33. Although the low-stress erase mode may be applied to a few strong WLs with larger relative stress coefficients in GuardedErase, when we evaluate the stress mitigation effect of the low-stress erase mode, we assume that all the WLs have the same relative stress coefficient, 0.35, for a simple analysis[5].

3.2.3 Per-Block Erase Modes

In order for an FTL to effectively exploit the endurance-capacity trade-off of the low-stress erase mode, we support nine block erase modes, gE(1), ..., gE(9). Table 2 summarizes the proposed nine block erase modes with varying numbers of protected WLs and different erase relief ratios[6]. The higher n in gE(n), the more WLs are erased using the low-stress erase mode with a higher erase relief ratio. Therefore, when a block is erased with a higher gErase mode, the maximum number of P/E cycles of the block is increased. For example, when a block is erased with gE(9) only, the maximum number of P/E cycles of the block is increased by 45% whereas when the block is erased with gE(1), it is increased by 19% only. On the other hand, as shown in Table 2, the higher the gErase mode, the more WLs become unusable

[5] In GuardedErase, a fixed number N of WLs are selected for applying the low-stress erase mode. Although N is fixed, selected N WLs change over time because N WLs with the worst BER values are protected when a block is erased. Since the relative stress coefficient of selected WLs is mostly less than 0.35 with few stronger WLs, our assumption of 0.35 is rather conservative in understanding real stress mitigation levels.

[6] The erase relief ratio of gE(n) is defined as a ratio of applying the gE(n) block erase mode over the normal block erase operation.

Table 2: A summary of nine gErase modes.

	gErase mode(n) == gE(n)								
	gE(1)	gE(2)	gE(3)	gE(4)	gE(5)	gE(6)	gE(7)	gE(8)	gE(9)
No. of protected WLs	8	12	16	20	24	24	24	28	32
Erase relief ratio	25%	33%	38%	40%	42%	50%	50%	50%	50%
Block capacity reduction	1.04%	2.08%	3.13%	4.17%	5.21%	6.25%	7.29%	8.33%	9.38%
Norm. Max P/E cycles	1.19	1.26	1.30	1.33	1.37%	1.39	1.41	1.43	1.45

for the next program cycle, thus increasing WAF values. As described in Section 3.3, it is an important task for an FTL to choose a proper gErase mode when it erases a block.

In Table 2, the percentage of block capacity reduction in gE(n) is given by (1.04 × n)%. Since our NAND block has 192 WLs, when a block is erased by gE(n), $2n$ WLs of the block become unusable for the next program cycle[7]. When a target percentage of block capacity reduction, say 1.04% in gE(1), is given, there can be multiple options to achieve the target block capacity reduction ratio using the low-stress erase mode. Let n_{WL} and f denote the number of protected WLs and the erase relief ratio. A combination $(k \times n_{WL}, f/k)$ over any k achieves the same block capacity reduction ratio as when $(n_{WL} \times f/100)$ WLs are always protected. For example, (4 WLs, 50%), (8 WLs, 25%) and (12 WLs, 16.7%) can achieve the same 1.04% reduction percentage in a 192-WL blocks. From multiple candidate combinations, we prefer ones with a lower f because such a combination can achieve higher I/O efficiency by allowing more flexible applications of gErase modes by an FTL. For example, we prefer (8 WLs, 25%) and (12 WLs, 16.7%) over (4 WLs, 50%). On the other hand, if f is too low (i.e.,

[7] $(2/192) \times \mathrm{n} \simeq 0.0104 \times \mathrm{n}$.

n_{WL} is too large), weak WLs may not be fully protected, thus limiting their endurance improvements. For example, (12 WLs, 16.7%) cannot maximally extend the lifetime of the weakest WL, WL 0, because f should be larger than 18.3% if WL 0 could fully achieve its maximum expected endurance. Therefore, as shown in Table 2, (8 WLs, 25%) was selected for gE(1).

When gErase modes are supported, a NAND endurance model based on the number of P/E cycles should account for the low-stress erase mode as well as the normal erase mode. Since we know the relative P/E stress when performing the normal erase mode and low-stress erase mode for each WL, the expected number of maximum P/E cycles for a specific WL can be easily calculated when the number of low-stress erase mode and the number of normal erase mode are known. For example, consider a WL with the maximum number of P/E cycles of 10K when the normal erase mode is used only. When the low-stress erase mode for the WL is used 20% of erase operations and the relative stress coefficient of the WL is 0.35, the maximum number of P/E cycles of the WL is computed to be 11.49K (i.e., $\frac{10K}{1\times0.8+0.35\times0.2}$)

The (n_{WL}, f) combination used for each gErase modes of Table 2 is derived using the above NAND endurance model using a two-step process. In the first step, we compute the per-WL low-stress erase mode application ratio for each WL that can maximize the endurance of the NAND block under the allowed block capacity reduction. Table 3 illustrates how we derive low-stress erase ratios for different WLs

Table 3: An illustrative example of deriving optimal low-stress erase ratios for gE(1).

WL Group	Endurance	P/E Stress (Coefficient : 0.35)		Low-stress erase ratio	Average P/E stress	Expected endurance
		Normal mode	Low-stress mode			
0	10K	1.00	0.35	18.3%	0.88	11.35
1	11K	0.91	0.32	4.7%	0.88	11.35
2	12K	0.83	0.29	0.0%	0.83	12.00
3	12.8K	0.78	0.27	0.0%	0.78	12.80
…						
44	12.3K	0.81	0.28	0.0%	0.81	12.30
45	11.5K	0.87	0.30	0.0%	0.87	11.50
46	10.7K	0.93	0.33	8.8%	0.88	11.35
47	10K	1.00	0.35	18.3%	0.88	11.35

for gE(1) with 1.04% reduction in the block capacity. In order to simplify an analysis, we grouped four WLs with similar characteristics as a WL group (WG). For example, WG 0 includes WL 0 to WL 3. In Table 3, P/E stress values were normalized over the worst WL, WL 0, of WG 0. For example, WLs in WG 2 have the P/E stress of 0.83 under the normal erase mode, which means that WL 8 to WL 11, which are stronger WLs than WL 0 experience 83% of P/E stress over WL 0. From a WG with the lowest expected endurance, the low-stress erase mode application ratio is gradually increased, and this process is repeated until the amount of block capacity reduction by the low-stress mode reaches the target block capacity reduction ratio of a gErase mode. In Table 3, WG 0 can extend its P/E cycles up to 11.35 K when 18.26% of erase operations employ the low-stress erase mode. Furthermore, when the low-stress erase mode were used for each WL as specified in Table 3, the block capacity reduction ratio

meets 1.04%[8].

In the second step, we determine the preferred conditions among the possible candidate combinations of n_{WL} and f. As explained above, we prefer (8 WLs, 25%) and (12 WLs, 16.7%). However, since WG 0 requires 18.3% of erase operations to be done with the low-stress erase mode, (12 WLs, 16.7%) cannot be selected.

3.3 Design and Implementation of LongFTL

3.3.1 Overview

Based on the proposed gErase modes of Table 2, we implemented a gErase-enabled FTL, called longFTL, which exploits the key tradeoff relationship of gErase between the block endurance extension and the block capacity reduction. The main goal of longFTL is to significantly extend the block lifetime with a negligible performance degradation by intelligently choosing a gErase mode under varying I/O workloads. Figure 10 shows an overall organization of longFTL. LongFTL, which is based on a typical page-level mapping FTL, employs three gErase-specific modules, the weak WL detector, the WAF monitor, and the gErase mode selector. The weak WL detector dynamically identifies weak WLs in a block, the WAF monitor tracks WAF changes, and the gErase mode selector determines the optimal gErase mode for the current workload characteristics estimated by the WAF monitor.

[8] $(18.26 + 4.7 + 8.78 + 18.26)\% \times 4/192$

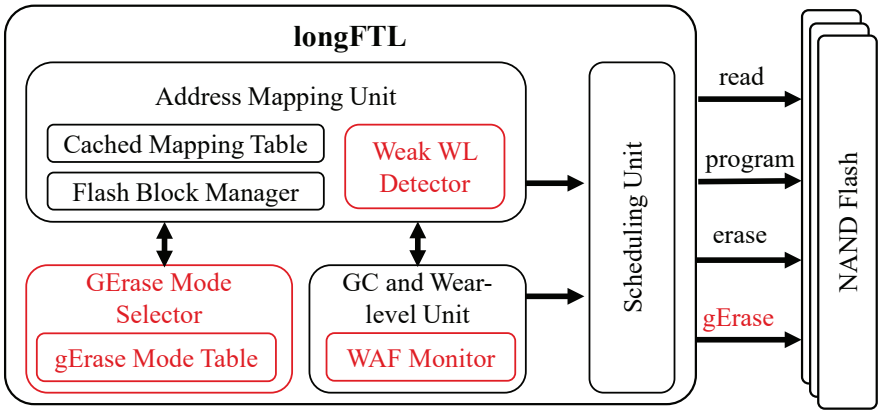


Figure 10: An organizational overview of longFTL.

3.3.2 Weak WL Detector

In order to protect weak WLs to extend the endurance of a block, it is required to identify which WLs are weak. The weak WL detector (WLD) module is responsible for maintaining WLs according to their BER values so that when N weakest WLs are requested by the gErase mode selector, the WLs with the N largest BER values can be quickly identified. The WLD module employs $(N_{ecc}^{max} - 9)$ linked lists where N_{ecc}^{max} represents the maximum number of bit errors that can be corrected by an ECC module. A linked list L_k contains all the WLs with k bit errors. Figure 11 shows an example of BER-sorted linked lists used for detecting weak WLs. As shown in the last linked list, $L_{<10}$, we do not sort strong WLs because they are not likely to be used for gErase. When 3 weakest WLs are needed, WL 148, WL 144, and WL 152 can be quickly identified. (We define the number of bit errors of a WL as the maximum number of bit errors occurred while reading a

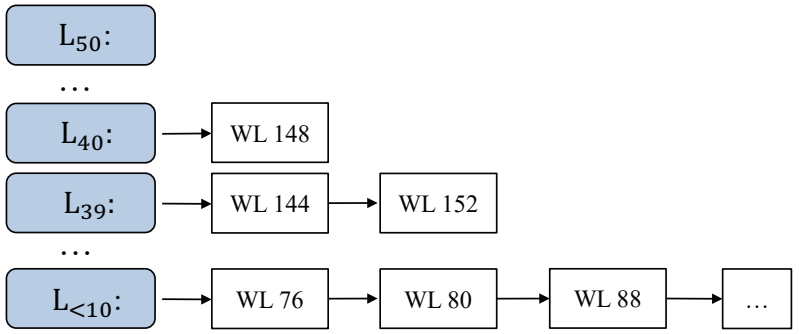


Figure 11: BER-sorted linked lists for detecting weak WLs.

page from the WL. For example, in the TLC flash memory we used for this work, the number of bit errors of a WL is set to the number of bit errors from the worst page (with the worst BER value) out of three pages encoded in the WL. In the TLC flash memory used for this study, worst BER values were observed mostly from MLC pages with a few cases from LSB/CSB page.)

In order to further reduce the time overhead of the WLD module, we do not update BER-sorted linked lists frequently. works well because the number of maximum P/E cycles is large (e.g., over 7K P/E cycles) and the BER values of blocks tend to gradually degrade thanks to wear leveling. In our current implementation, the WLD-related overhead is less than 0.1% of the total I/O time.

In addition to selecting N weakest WLs from a block, the WLD module ensures that gErase modes are evenly applied to all the blocks. If a large number of gErase modes are used for a few blocks only, the SSD lifetime may not be extended at all because the rest of blocks are erased using the normal erase mode only. We maintain a separate

linked list of free blocks by the number of gErase operations. When we select a free block, we prefer blocks with fewer gErase counts[9].

3.3.3 WAF Monitor

The WAF monitor is responsible for tracking a WAF value of an SSD. Since a block capacity is reduced when the block is erased using a gErase mode, it is important to understand if the current effective SSD capacity is adequate to properly process the current I/O workload. If the effective SSD capacity were reduced too much from aggressive gErase mode applications by the gErase mode selector, the SSD may suffer a significant I/O performance degradation. On the other hand, if gErase modes were applied too conservatively, the effectiveness of GuardedErase is quite limited.

The main function of the WAF monitor is to observe WAF fluctuations and decide if the current WAF value is *stable* enough for the gErase mode selector to make a proper mode decision. Although it is difficult to precisely define what a stable WAF value means, in the current implementation, we assume that a WAF value is stable if the WAF value has been *steady* for the last 10 observation intervals. A WAF value w is called *steady* for an observation interval $(t_s, t_e]$ if all the WAF values observed in $(t_s, t_e]$ are within [0.98w, 1.02w]. Since we are interested in knowing a long-term workload characteristics instead of fast changing short-term workload characteristics, a single

[9]Since a free block may be selected under a different criterion (such as low P/E cycles), leveling gErase counts among different blocks is supported in a combined fashion with other selection criteria.

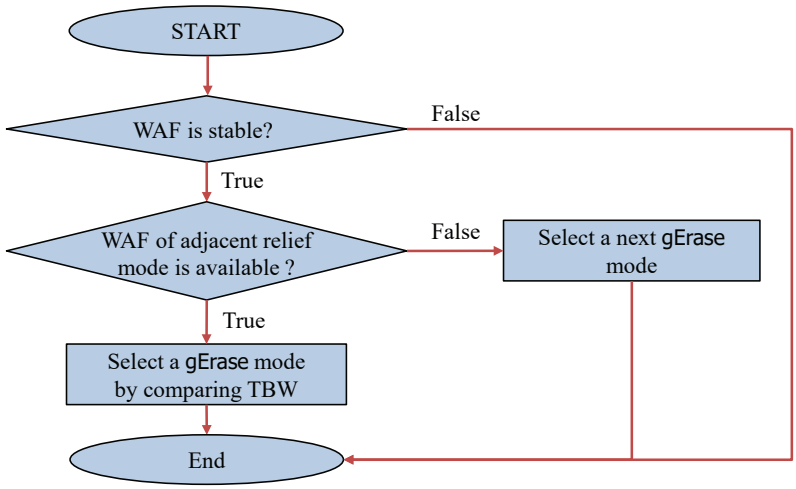


Figure 12: GErase mode selection function.

observation interval is defined as the time it takes to perform a sufficient number (e.g., 5% of total blocks) of garbage collections (GC). The WAF monitor also tracks the change in WAF values (i.e., WAF history) so that the recent WAF change trend can be considered by the gErase mode selector.

3.3.4 GErase Mode Selector

The gErase mode selector decides that which erase mode would be used for the next block erase. In selecting a right erase mode, the gErase mode selector considers two outputs from the WAF monitor: WAF stability and WAF history. Figure 12 describes the key steps of the gErase mode selector in flow chart. Function of Figure 12 is invoked each time the WAF monitor measures a WAF value (i.e., every observation interval). Under the condition that the WAF value is

stable, after calculating an expected TBW of each gErase mode, the gErase mode selector chooses a gErase mode with the larger expected TBW. The TBW comparison between gErase modes is conservatively performed only between the current gErase mode gE(c) and the adjacent modes (i.e., gE(c+1) or gE(c-1)). Therefore, a current gErase mode is only allowed to change only one step of gErase mode at a time. If the expected TBW cannot be compared because the WAF value for the adjacent gErase mode is not available (i.e., there is no WAF history for adjacent gErase mode), the next gErase mode is determined to the adjacent gErase mode and impact of WAF is evaluated later. If it is revealed that the wrong choice was made through TBW comparison, the gErase mode is changed to the previous mode.

Table 4 illustrates how the gErase mode selector works using function of Figure 12. $MAX_{P/E}$ according to each gErase mode is obtained from the gErase mode table, and the WAF value of the current gErase mode can be obtained from the WAF monitor. The gErase mode selector chooses gErase mode from gE(0) to obtain the baseline WAF value and normalize the expected TBW to 1. Since there is no WAF history about the adjacent gErase mode, the gErase mode selector changes

Table 4: An example gErase mode selection.

gErase Mode	Norm. Max P/E cycles	WAF	Norm. TBW
gE(0)	1.00	2.66	1.00
gE(1)	1.19	2.78	1.14
gE(2)	1.26	2.91	1.15
gE(3)	1.30	3.05	1.14

the mode to gE(1) and measures the WAF value, and calculates the expected TBW. Because the expected TBW of gE(1) is larger than baseline, the gErase mode selector increases the mode to gE(2) and compares the expected TBW again. This comparison is performed up to gE(3). From the obtained WAF values, gE(2) is selected as the optimal gErase mode because it can maximize the expected TBW by 15% over baseline.

According to the determined optimal gErase mode, the flash block manager decides whether to apply gErase operation at the time of allocating free blocks. Since the block relief ratio of each gErase mode was chosen as low as possible, It can be used for the purpose of performing gErase operation only when it is really necessary. The free block allocation by the flash block manager is divided into active block allocation for host request and active block allocation for GC execution [41]. In order to reduce the side effects caused by the application of erase stress-relief, gErase is selectively used for active block allocation for host request so that the recovery of the reduced physical capacity can be fast. That is, gErase was not used as much as possible when allocating an active block for GC which is used to store cold data, while gErase was actively used when allocating an active block for host request which is used for storing hot data.

3.4 Experimental Results

3.4.1 Experimental Settings

In order to evaluate the effectiveness of the proposed techniques, we implemented longFTL as an extension on a well-known FTL simulator, MQSim [42]. For our evaluation, we configured the FTL simulator to support 32/64-GB storage capacity for an efficient experiment. Our simulated storage system was configured to have four channels with one NAND flash chip per channel. Each NAND flash chip has a 2 plane configuration and each plane has 1822 blocks which are composed of 576 8-KB pages. And NAND flash interface supports up to 333 MT/s. The average page program time and the block erase time were set to 700us and 4000us, respectively. The overprovisioning ratio was set at a level of 10% considering the actual SSD, and the GC execution threshold was set to a very low value (0.002) to effectively utilize overprovisioning area.

We evaluated longFTL using MSR trace workloads [43] and additional I/O traces generated from Sysbench [44] and Filebench [45]. Among several MSR traces, the evaluation was performed on workloads with a total write amount of more than a certain level, but for traces where the total amount of write is not sufficient, traces are repeatedly replayed. The timestamp of the trace was accelerated at an appropriate level in consideration of the SSD simulation processing speed. Table 5 summarizes key I/O characteristics of these workloads. As the table shows, each workload has a different read:write ratio and

Table 5: I/O characteristics of traces used for evaluations.

	proj0	prxy1	prxy0	src10	proj2	OLTP	Fileserver	Varmail
Read:Write	6:94	64:36	5:95	46:54	86:14	70:30	40:60	40:60
Total Write (GB)	144	725	54	302	169			
WAF	1.08	1.16	1.24	2.66	3.55	1.89	2.49	2.98

different WAF characteristics. We evaluated the expected TBW and performance of the longFTL using these workloads. All evaluations were normalized based on the default page-level FTL, which does not have any stress relief scheme. In addition, the BPM technique that utilizes the page variation characteristics straightforwardly without stress relief mechanism was also comparatively evaluated.

3.4.2 Lifetime Improvement

In order to evaluate the effect of lifetime improvement, we measured the expected TBW for each workload. Figure 13 shows the normalized TBW for each workload. In the case of BPM, the block capacity reduction was evaluated for 2.1% and 5.2% conditions, respectively,

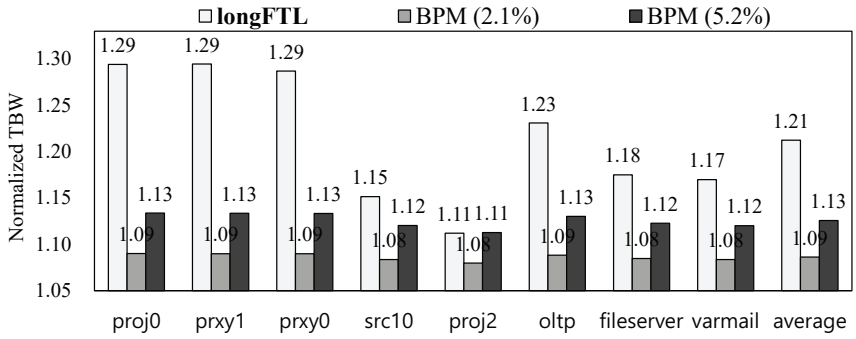


Figure 13: Lifetime improvement by GuardedErase.

in consideration of NAND specifications and performance degradation characteristics. The proposed longFTL extends the lifetime of an SSD for all workloads. Lifetime improvement varied by workload, and the overall lifetime improvement is extended by 21% on average of eight workloads. The biggest lifetime improvement is around 29% for proj0 and prxy1 workloads, but only 11% for proj2 workload. This difference is highly related to the WAF increase characteristics according to the application of gErase mode. On the other hand, lifetime improvement of BPM does not have a large variation by workloads, and the amount of lifetime improvement increases under the condition of larger block capacity reduction, but the improvement is not as great as the proposed longFTL.

Figure 14 shows the optimal gErase mode for maximizing the lifetime of an SSD in each workload and the WAF changing when operating under optimal condition. For proj0 and prxy1 workloads, which show a lot of lifetime improvement, NAND endurance could

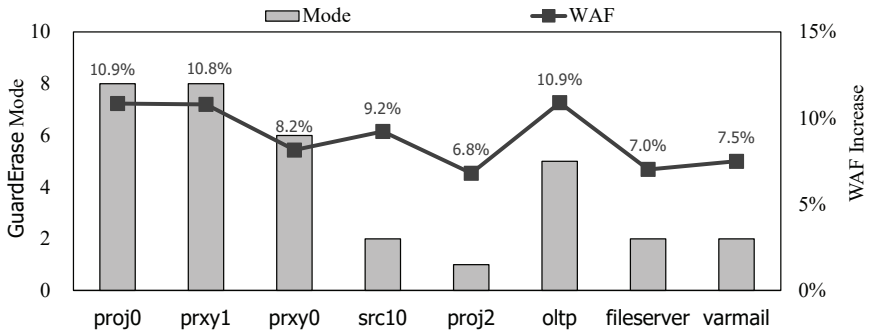


Figure 14: Optimal gErase mode and its WAF changing characteristics.

be effectively increased by using the higher gErase mode, because the WAF changing due to the block capacity reduction is not greater than 11% even when the gE(8) is used. On the other hand, in the case of proj2 workload, which shows the lowest lifetime improvement, optimal gErase mode is only gE(1), so the effect of increasing NAND endurance is not great. Even in the condition of gE(1), where the physical capacity reduction is very limited (i.e., 1.04%), the WAF changing is as high as 7%, so there is no advantage obtained by applying a higher gErase mode.

3.4.3 Performance Overhead

In order to evaluate the performance overhead of longFTL, we measured the IOPS (Input/Output Operations Per Second) in each workload. For comparison, performance evaluation was also performed for two BPM conditions. Figure 15 shows the normalized IOPS based on the baseline when operating in the optimal gErase mode for the

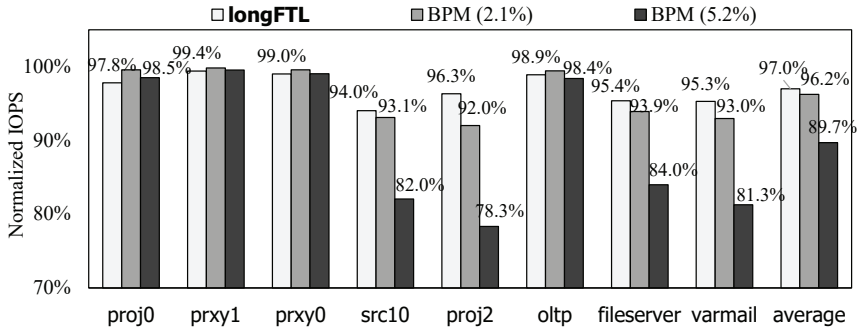


Figure 15: Performance impact of the lifetime improvement.

purpose of maximizing TBW. Overall, the performance of longFTL is degraded by 3%, and the main cause of the drop in performance is the increase in WAF due to the decrease in physical capacity by GuardedErase execution. Despite the similar level of increase in WAF, the performance degradation is larger in src10, fileserver, and varmail with high WAF values. On the other hand, in the case of proj2, although the WAF value is high, the performance degradation is not large. This is because the write ratio is relatively low, so the effect of the write performance degradation affects less on the overall IOPS. Unlike longFTL, where different levels of physical capacity reduction are applied because the optimal gErase mode is different for each workload, BPM is affected by the same physical capacity reduction regardless of workload, and overall it shows lower performance than longFTL even under physical capacity reduction of 2.1%. In the case of 5.2% capacity reduction, the performance decreases by more than 10%, which acts as a limiting point of the scheme to improve the lifetime through further reduction of physical capacity.

3.4.4 Effectiveness of Lowest Erase Relief Ratio

In order to evaluate the effectiveness of the preferred gErase mode conditions and block stress relief policy of longFTL, we compared it with the condition in which the block relief ratio is maximized. In Section 3.2.3, when deciding the gErase mode conditions, the candidate with the lowest erase relief ratio was selected among possible candidates, and GuardedErase operation was performed for the host

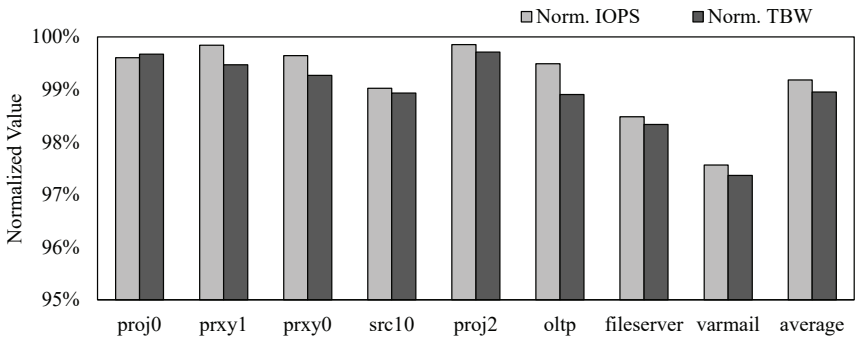


Figure 16: Comparison between preferred gErase mode condition and others.

active block only in longFTL. However, the condition for maximized erase relief ratio is that erase relief is performed during the allocation of all blocks regardless of host active block or GC active block. Figure 16 shows the expected TBW and IOPS, and the two values are normalized based on that of longFTL. The evaluation result shows the lifetime and performance degradation of about 1% on average compared to the preferred condition, and TBW and IOPS degradation occurs up to 3% in varmail workload. This shows that the approach with the lowest erase relief ratio is effective.

Chapter 4

Improving SSD Performance Using Adaptive Restricted-Copyback Operations

4.1 Motivations

4.1.1 Data Migration in Modern SSD

A typical data migration in SSDs is performed by an off-chip data copy. An SSD firmware reads data from a source page and transfers the data to a DRAM buffer through a channel bus. Before the data are sent to the DRAM buffer, errors are corrected by the ECC module of the flash memory controller (FMC). In the program phase„ the SSD firmware takes a reverse data path from the DRAM buffer to the target page. The data copy time t_{COPY} can be expressed as follows: t_{COPY} $= t_R + t_{DMA^{out}} + t_{DMA^{in}} + t_{PROG}$ where t_R, $t_{DMA^{out}}$ and $t_{DMA^{in}}$ are a data transfer time from NAND cells to a per-plane register and a DMA out/in time between the register and DRAM buffer, respectively. However, a large number of data migrations may occur at the same time in a modern SSD. A high degree of the parallelism in data migrations may significantly increase $t_{DMA^{in}}$ and $t_{DMA^{out}}$ because of contentions on the channel level as well as the serial bus to/from the

DRAM buffer. This is because the bandwidth of the DRAM is limited and the efficiency of the DRAM degrades when many masters request DRAM at the same time.

On the other hand, when a copyback operation is used, a data migration can be performed without requiring neither $t_{DMA^{out}}$ nor $t_{DMA^{in}}$. The FTL can read data from the source page to the per-plane local register and directly write back to the destination page from the per-plane local register. Since the copyback operation transfers data within a given plane, even when multiple data migrations occur at the same time, all data migrations can be completed by (t_R + t_{PROG}). Thus, it can significantly reduce the overhead of data migrations especially for modern SSD of multiple channels and multiple ways.

4.1.2 Need for Block Aging-Aware Copyback

Generally, the number of P/E cycles has been mainly used as the indicator of NAND aging. During NAND operations, the high voltage used in the erase operation damages the tunnel oxide of the NAND cells, thus increasing the Bit Error Rate (BER) observed in subsequent reads. As the number of P/E cycles increases, the tunnel oxide layer eventually reaches a state in which the cells can no longer store information reliably. Since erase operations are responsible for the wear of NAND cells, the number of P/E cycles has been regarded as a good proxy indicating the wear of NAND cells.

However, the number of P/E cycles alone cannot accurately rep-

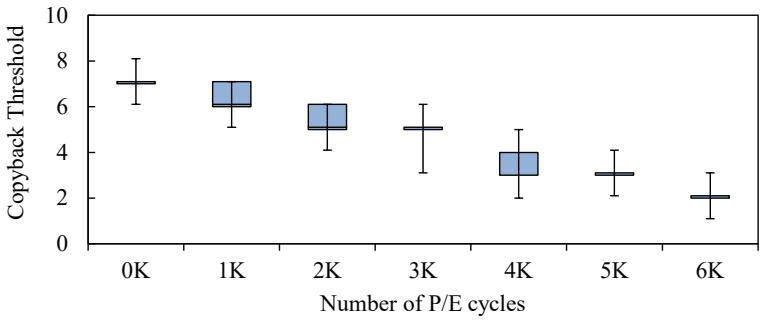


Figure 17: Copyback threshold variations on different P/E cycles.

resent the exact wear status of NAND cells. For example, when two NAND blocks experience the same number of P/E cycles, their BER could be significantly different [46]. This difference is mainly caused by process variations in the manufacturing and is accelerated by various user environments such as operating temperature. From our characterization study using 3D TLC NAND chips [47], we observed that the copyback threshold of a block cannot be accurately estimated by only using the P/E cycles as a wear indicator of NAND cells. Figure 17 shows that, even at the same P/E cycles, there is a large variation on the copyback threshold count. In FastGC, since a single copyback threshold value was used for all the blocks with the same P/E cycles, the copyback threshold was conservatively selected, thus missing many opportunities for additional copybacks on most blocks.

The rCPB scheme proposed in this thesis was mainly motivated from how to exploit these missed copybacks. From our characterization study, which will be described in Section 4.2, we observed that the copyback threshold of a NAND block can be accurately predicted

when the P/E cycles of the NAND block is augmented with the BER value measured right after a program operation. Using this extended NAND wear indicator, most of the missed copybacks in FastGC can be successfully utilized under the rCPB scheme.

4.2 RCPB: Copyback with a Limit

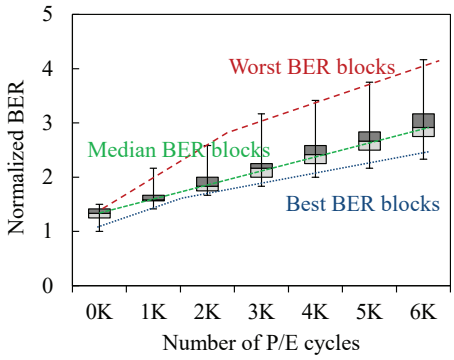
4.2.1 Error-Propagation Characteristics

In order to manage the flash reliability problem caused by successive copyback operations, it is important to understand the NAND error propagation characteristics when the same page experiences consecutive copybacks without error correction by the ECC module. We conducted a NAND reliability characterization study using 30 actual 3D TLC NAND chips [47] to better understand the error propagation characteristics under successive copybacks. In order to take into account of block-to-block variations as well as page-to-page differences, we selected 128 blocks from each chip where their physical locations were evenly distributed within the chip. For a selected block, we tested all the pages in the block. In our study, a total of 3,840 blocks and 2,211,840 pages were evaluated to obtain statistically significant experimental results.

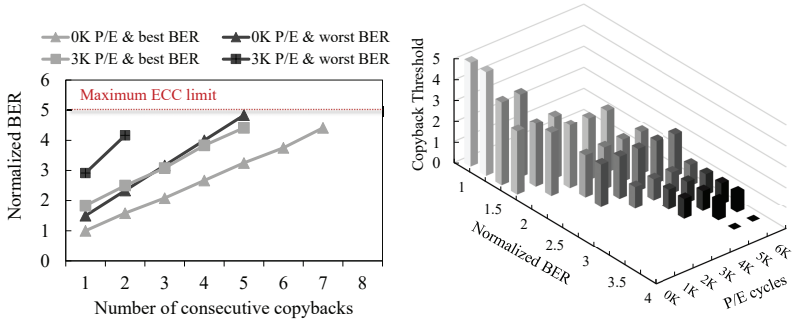
In order to derive the proposed rCPB model, we first evaluated the reliability differences within NAND blocks with the same P/E cycles. As a measurement of the block reliability, we used the BER value of a block immediately after a program operation. Since the

BER value is computed right after the block is programmed, no BER degradation is added from the retention errors or read disturb errors, so that the measured BER represents the block reliability level more accurately. Figure 18(a) shows how BERs fluctuate within blocks with the same P/E cycles. As expected, large BER variations exist among the same aged blocks by P/E cycles. For example, the BER of the worst block is 1.8-times larger than that of the best block when the number of P/E cycles is 6K. As shown in the box plot, BER values of most blocks are clustered around the average BER of a given P/E cycles. On the other hand, the worst BER values make BER distribution long-tailed ones. This contributes many missed copybacks in FastGC.

In order to characterize the effect of the block reliability level on the copyback threshold, we observed how BERs change over successive copybacks when the block reliability level changes. We divided the blocks according to the measured BER characteristics and evaluated the difference of error accumulation characteristics by copyback operations for each group. Figure 18(b) illustrates our evaluation results on the best BER blocks and the worst BER blocks under the initial (i.e., 0K) P/E cycle condition and 3K P/E cycle condition, respectively. (We used the 3-month retention requirement in this evaluation.) Under all conditions, the worse BER characteristic of the blocks, the larger the error accumulation due to the copyback operations. Considering the error accumulation level due to copyback operations for each group and the correction capability of the ECC module, it is



(a) BER variations over P/E cycles.



(b) BER variations over block characteristics.

(c) Copyback thresholds distribution.

Figure 18: Key results of the NAND characterization study.

possible to make the copyback thresholds for each condition.

Figure 18(c) shows how the copyback threshold value changes under each P/E cycle condition when considering the retention requirement with block BER characteristic. The retention requirement was assumed to be one year at 30°C. Even at the same P/E cycles, the copyback threshold varies greatly depending on BER characteristic of the NAND blocks. When P/E cycle is 3K, the best BER block can use two more copyback operations than the worst BER block. There-

fore, by distinguishing the block reliability level, the total number of copybacks can increased over the block-unaware worst-case setting of FastGC. In order to identify the copyback threshold accurately, we considered the properties of P/E cycles, retention time requirement, all possible data migration cases between source and destination pages, and the different characteristics between NAND blocks.

4.2.2 RCPB Operation Model

From our characterization study on the copyback error propagation, we constructed the copyback threshold table, $CTT(x, e, t)$, which indicates the maximum number of consecutive copyback operations that does not cause any reliability problem for x P/E-cycled blocks of BER value e under the condition of t-month retention requirement. Table 6 summarizes our proposed rCPB operation model with the different retention requirements. If 1-year retention is required at 2K P/E cycles, the copyback threshold of NAND block is determined from 2 to 4 based on the value of p. If the data migration is required more than the copyback threshold, the page must be migrated using an off-chip data copy, thus the accumulated bit errors can be corrected

Table 6: The proposed rCPB operation model.

Retention requirement	Block BER characteristic	P/E cycles					
		[0K∼0.4K]	(0.4K∼1K]	(1K∼2K]	(2K∼3K]	(3K∼4K]	(4K∼5K]
1 year	Best block	5	4	4	3	3	2
	Median block	5	4	3	3	2	1
	Worst block	3	2	2	1	1	0
3 months	Best block	6	5	4	4	3	2
	Median block	6	5	4	3	3	2
	Worst block	4	3	2	2	1	1

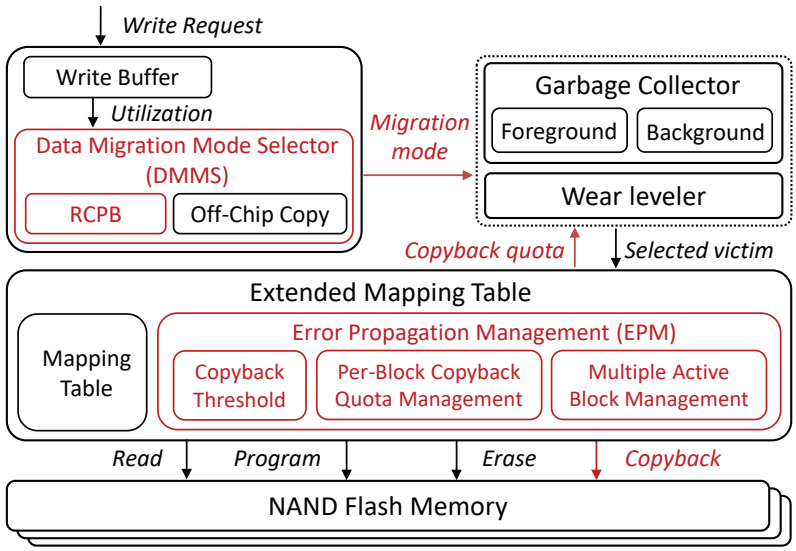


Figure 19: An organizational overview of rcFTL.

by the ECC module. As the table shows copyback threshold can be increased on 3-month retention. In this thesis, we used 1-year retention as a basic requirement in accordance with JEDEC standards.

4.3 Design and Implementation of rcFTL

Based on the proposed rCPB model presented in Section 4.2, we implemented an rCPB-enabled flash translation layer (FTL), called rcFTL, which is based on the existing page-level mapping FTL. Figure 19 shows an overall organization of rcFTL. RcFTL consists of two additional modules, the error propagation management (EPM), and the data migration mode selector (DMMS). The EPM module is in charge of checking the rCPB availability for a data migration while the DMMS module selects the most appropriate data migration mode

for a given data copy request.

4.3.1 EPM module

Quota-Based Determination of RCPB Availability

The EPM module plays a key role to ensure the data reliability while rcFTL maximally uses rCPB operations. When an rCPB operation is desired in moving a page, the EPM module checks its availability. A key challenge in determining the rCPB availability is that pages are moved across various blocks that have different copyback thresholds. If a page migration is (somehow) managed to move pages only within NAND blocks with the same threshold, determining the rCPB availability is straightforward. All we need is to keep every page's rCPB count less than the threshold of those blocks. However, such a page migration management is rather impractical because it significantly obstructs a flexible page allocation of an FTL.

In order to effectively determine the rCPB availability of page migrations across blocks with different thresholds, the EPM module employs *quota-based rCPB model* which regards the copyback threshold of a block as the quota spent upon the rCPB from the block. For example, if the copyback threshold of a block is CT, the EPM module considers that a copyback operation from the block deducts $\frac{1}{CT}$ of the maximum quota which is initially given the same amount for every block. This is based on our observation that the error-propagation in successive rCPB operations at the same block is almost linear

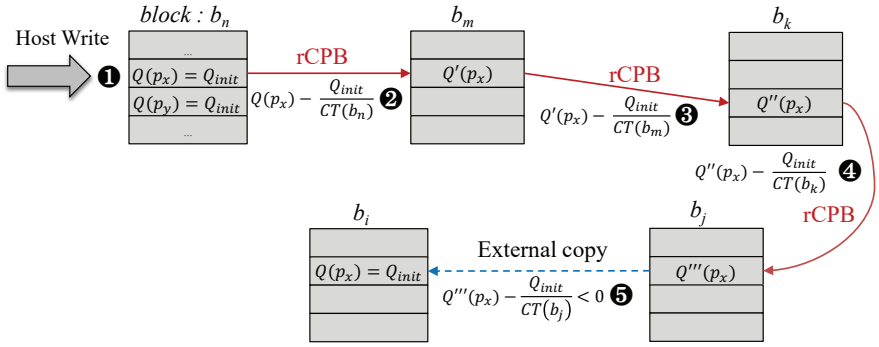


Figure 20: An illustrative example of the quota transition.

(even though the error increase per rCPB operation varies in different blocks).

Figure 20 shows how rcFTL deals with the page migrations with the quota-based rCPB model. The EPM module keeps track of the copyback quota $Q(p_i)$ for every page where p_i indicates a page whose index is i in an SSD. When a page p_x is programmed by a host write, $Q(p_x)$ is initialized with Q_{init} (❶ in Figure 20). Once rcFTL moves p_x using an rCPB operation from block b_n to b_m, the EPM module decreases $Q(p_x)$ by deducted quota, $\frac{Q_{init}}{CT(b_n)}$ (❷), where $CT(b_n)$ is the copyback threshold of b_n. We can obtain $CT(b_n)$ just by retrieving the predefined CTT with $PE(b_n)$ and $BER(b_n)$[1]. For a simple management of $Q(p_i)$, we set Q_{init} to the least-common-multiple value of all the present values in the CTT. As long as $Q(p_x) \geq 0$, p_x can be moved through successive rCPB operations (❸ and ❹). On the other hand, if the deducted quota of the source block b_j, $\frac{Q_{init}}{CT(b_j)}$, is large enough

[1]P/E cycles of block b_n, $PE(b_n)$, is maintained in typical SSDs, so rcFTL needs to additionally keep track of $BER(b_n)$ as explained in Section 4.2.

to make $Q(p_x)$ less than 0, the EPM module only allows off-chip copy so that $Q(p_x)$ is initialized with Q_{init} (❺).

Per-block Quota Management

Although the quota-based approach is effective in determining the rCPB availability, managing $Q(p_i)$'s in a per-page fashion may introduce non-trivial space overhead, considering the capacity increase of the modern SSDs. For example, suppose that Q_{init} is 30 and rcFTL manages 4-KB logical-to-physical mappings in a 16-TB SSD. In such a case, at least more than 2.5-GB[2] memory space is additionally required for the per-page quota management.

In order to avoid the overhead of per-page quota management, EPM module employs a *per-block* quota management approach. That is, the amount of copyback quota deducted by rCPB operations is managed at the block level, not at the page level. Since the number of entry for the per-block management is at least two orders of magnitude smaller than that for the per-page management, the per-block management technique significantly reduces the memory footprint for the copyback quota and minimizes the computing overhead of bookkeeping operations to a negligible level. Since all the pages in a block are assumed to have the same amount of the copyback quota in the per-block quota management, when a source page p in a victim block $b_v[Q(b_v), dQ(b_v)]$ with the copyback quota $Q(b_v)$ and the deducted quota $dQ(b_v)$ is migrated by rCPB, the page p should be moved to

[2] $(5[\frac{bit}{page}] \times 16 \times 10^{12}[\frac{byte}{SSD}] \times (4 \times 10^9[\frac{byte}{page}])^{-1} = 2.5 \times 10^9[\frac{byte}{SSD}])$

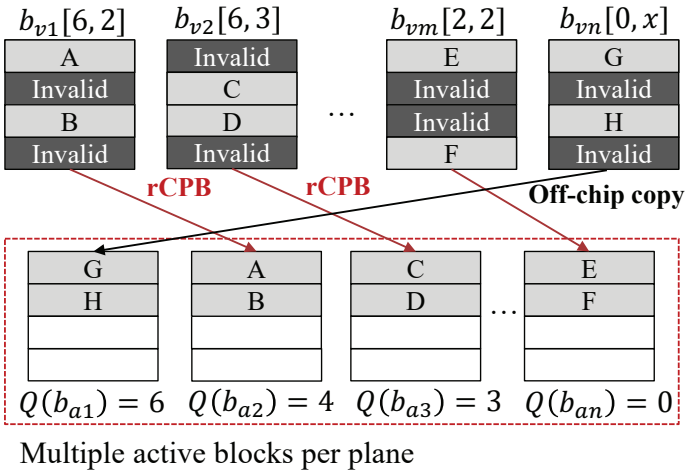


Figure 21: Data migrations in the per-block management.

a page in a block b_d where $Q(b_d) = Q(b_v) - dQ(b_v)$. In order to efficiently support this additional constraint, the EPM module manages *multiple active blocks* per plane at the same time. Figure 21 shows an example of how data migrations are performed using rCPB operations in the per-block quota management. If the Q_{init} is given by 6, the EPM module maintains five active blocks whose copyback quota is divided into 0, 2, 3, 4, and 6. When a block $b_{v2}[6,3]$ is selected as a GC victim block, its valid pages, C and D, are moved to the active block b_{a3} which has the quota of 3 by rCPB operations. On the other hand, if the block $b_{vn}[0,x]$ is selected as a GC victim block, its valid pages are moved using off-chip copies to the active block b_{a1} which has the initial quota, 6.

4.3.2 Data Migration Mode Selection

In order to take full advantages of rCPB, the DMMS module intelligently chooses when to use rCPB over a normal off-chip copy depending on the write buffer utilization ratio u. When u is low, which indicates that the current host I/O workload is not intensive, the DMMS module selects the off-chip copy mode so that more future data migrations can be supported by rCPB. On the other hand, when u is high, the DMMS module chooses the rCPB mode for higher performance. In our current implementation, the utilization threshold ratio for the mode selection was set to 50%. (That is, if u is higher than 50%, the rCPB mode is used for data migrations.) Since rcFTL employs the per-block quota management scheme and most data migration decisions are made in a block granularity, the DMMS module makes its mode selection decisions in a per-block level as well. When a data migration decision is made (e.g., by a foreground GC task), the DMMS module selects a proper mode based on the current u value. In order to filter out abrupt noise-like changes in u, the DMMS module makes its mode selection based on a t-second moving average of u. In the current implementation, t is set to an average block write time.

In rcFTL, both the GC and wear leveler operate in an rCPB-aware fashion. For urgent management tasks (such as a foreground GC task), the rCPB mode is actively used regardless of the current u ratio value. On the other hand, when background management tasks (such as a background GC task) are invoked, the DMMS module decides proper

Table 7: I/O characteristics of traces used for evaluations.

	OLTP	NTRX	Varmail	Fileserver
Read:Write	7:3	0.5:9.5	4:6	4:6
WAF	2.0	2.3	2.7	5.4

modes as explained above.

4.4 Experimental Results

4.4.1 Experimental Setup

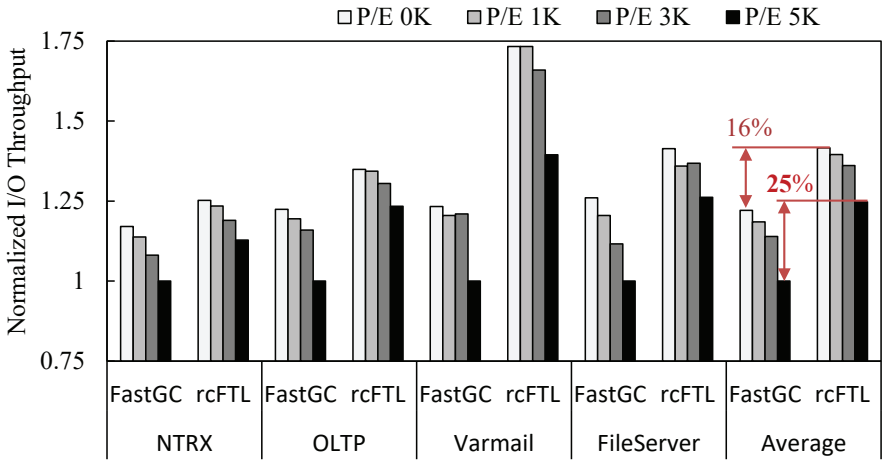
In order to evaluate the effectiveness of the proposed rcFTL technique, we implemented rcFTL as a host-level FTL on a custom flash storage system [48]. For our evaluation, we configured our flash storage system to support a 128-GB storage capacity only for efficient experimental evaluations. Our emulated storage system was configured to have eight channels with eight NAND flash chips per channel. Each NAND flash chip has 1024 blocks which are composed of 128 16-KB pages and NAND interface which supports up to 533 MT/s. The average t_{PROG} was set to 660 us [47] and the size of the write buffer was set to 10 MB. We evaluated rcFTL using four I/O traces generated from Sysbench and Filebench. As shown in Table 7, each workload has different ratios between read and write and different WAF values. Using these workloads, we evaluated the overall I/O throughput for six different P/E cycle conditions where the copyback threshold counts are distinguished and compared them with the existing techniques.

All measurements were normalized over a page-level mapping FTL which always migrates data using the off-chip copy.

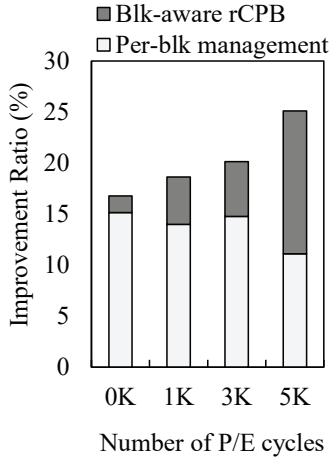
4.4.2 Evaluation Results

Figure 22(a) shows the normalized I/O performance for each workload under various P/E cycle conditions. The proposed rcFTL has better performance compared to the existing FastGC method for all workloads because it uses block-aware copyback threshold effectively and the per-block management scheme for the copyback quota improves the WAF value. The overall I/O throughput was improved by 43% on average of four workloads in initial P/E cycle over the baseline FTL. It also improved I/O throughput by 25% over the FastGC method when P/E cycles is 5K. When blocks are young, most data migrations can be supported by copybacks even in the worst blocks. As the copyback threshold increases, the degree of performance improvement due to rCPB degrades, most of the improvements in young blocks over FastGC comes from per-block management of rcFTL. On the other hand, as shown in Figure 22(b), as blocks get older, the impact of block-aware rCPB scheme grows. For example, the performance improvement by block-aware rCPB is 14% when the P/E cycle is 5K.

In order to analyze the effect of per-block management of rcFTL, we compared the WAF value of rcFTL and existing per-page management. Figure 23 shows normalized WAF value based on baseline FTL. Overall, per-block management scheme of rcFTL showed lower WAF



(a) Normalized I/O throughput.



(b) Breakdown of improvement.

Figure 22: Performance comparison between FTLs.

than per-page management for all conditions. Although the overhead of per-page management was amplified due to the small number of pages per block by limited capacity, the WAF value was increased in per-page management as the copyback threshold increases in OLTP

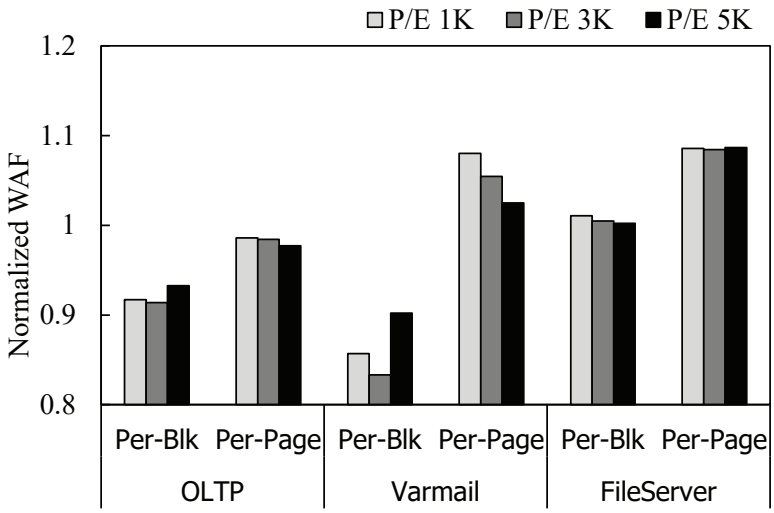


Figure 23: Effectiveness of per-block management.

and Varmail. On the other hand, the per-block management of rcFTL decreased WAF value as the copyback threshold increases. This is the result of separating data with different lifetimes into different blocks through multiple active block management of per-block scheme without consuming flash resources. However, there was no WAF reduction effect of per-block management on FileServer. This is because the workload has a strong random update characteristic that does not have any significant locality.

In order to understand how the mode selector proposed in rcFTL performs, we compared the performance of rcFTL with rcFTL– (which uses rCPB in a greedy fashion). Figure 24 shows normalized performance gain of rcFTL over rcFTL– under varying I/O intensity. In order to generate workload fluctuations, which are needed to properly evaluate the DMMS module, we generated three synthetic workloads,

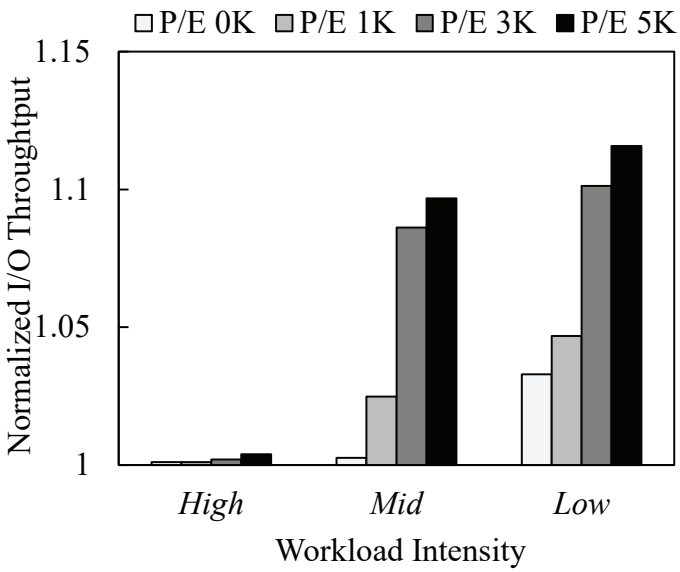


Figure 24: Effectiveness of migration mode selector.

High, *Mid* and *Low*, using Fio benchmark. In *High*, 70% of I/O requests were issued without inter-request idle times while 30% were issued with some idle times. For *Mid* and *Low*, the ratio between two requests is 50:50 and 30:70, respectively. When the I/O intensity is lower than the threshold, since the off-chip copy mode is more likely to be used in rcFTL, rCPB-eligible blocks tend to increase over rcFTL− because the copyback quota of more blocks are reset. The increased number of rCPB-eligible blocks, in turn, improves the I/O throughput when the I/O intensity is higher than the threshold. Figure 24 shows that the performance gain is higher when the workload intensity is *Low*. The performance is further improved, especially in small copyback threshold conditions, which shows that the mode selector works effectively.

Chapter 5

Reparo: A Fast RAID Recovery Scheme for Ultra-Large SSDs

5.1 SSD Failures: Causes and Characteristics

5.1.1 SSD Failure Types

SSD failures can be grouped into two categories depending on its predictability [49]. The first category is the end-of-life (EoL) failure which is caused by worn-out SSD components. On the other hand, the second category is the sudden failure, and it is highly associated with unexpected component failures which happen randomly.

Most SSD failures in the EoL failure category come from the worn-out NAND flash memory. Although the NAND flash memory is non-volatile, its reliable data retention is limited by the maximum number of program/erase (P/E) cycles which is determined by flash manufacturers. Recent 3D TLC flash memory can support up to 10K P/E cycles. When flash cells in a block are worn out beyond their reliability threshold, the block becomes a bad block because the pages in the block cannot be reliably accessed anymore [50–53]. When the number of bad blocks in an SSD reaches the pre-defined maximum

number N_{bad}^{ssd}, the SSD is considered to be failed. Since the number of bad blocks tends to increase rapidly around the end of SSD's useful life period, manufacturers set N_{bad}^{ssd} conservatively. For example, N_{bad}^{ssd} is typically set to 2.5% of the total (user-visible) blocks in an SSD [54].

The sudden failure happens abruptly at any point of time, so it often occurs much earlier than the EoL failure. The representative examples include NAND die failures and bugs in the flash translation layer (FTL). A NAND die failure occurs when an excessive number of bad blocks are found in the NAND die, but there are various reasons that cause it. One example is when the peripheral circuitry (e.g, page buffer, word line (WL) decoder and sense amplifier) of a NAND die malfunctions because of some defects. In that case, all the blocks in the NAND die become bad since they can no longer be reliably accessed. As another example, if the flash cell is not functioning properly due to a structural failure (eg, WL to WL bridge, WL to channel bridge, and WL to common source line bridge), it is identified as a bad block during the manufacturing process or the infant period [15, 55]. However, if the number of accumulated bad blocks per NAND die is below a certain threshold, it is considered normal. The NAND specification defines the maximum number of bad blocks, N_{bad}^{die}, that can occur on a NAND die within its lifetime and if a NAND die exceeds the threshold, it is considered defective [54]. This is because the NAND manufacturer does not guarantee normal operations on these NAND dies. In fact, in the SSD field study, a large number of additional bad blocks tend to be generated in a short period of time after a certain

number of bad blocks occur in most SSDs [51]. This is due to defective NAND dies with poor cell characteristics or defective peripheral circuits. Besides the excessive bad blocks, a defective NAND die can cause firmware operation failure or command timeout.

Since an SSD is tolerant for some number of bad blocks, it is important to efficiently manage bad blocks when bad blocks are detected during run time. The bad block management (BBM) module of the FTL remaps a bad block to a reserved block that comes from an over-provisioning (OP) space of the SSD. The OP space, which is a reserved space in the SSD, is used for minimizing the performance/lifetime impact of garbage collection and bad block management [56]. As the number of bad blocks increases, more blocks from the OP space are consumed to restore data of the bad blocks, which, in turn, negatively affects the performance and lifetime of the SSD.

5.1.2 SSD Failure Characteristics

SSD failure characteristics are commonly modeled using a typical bathtub curve [4] with three distinct periods: the infant period with high sudden failure rates, the useful life period with lower failure rates, and the wear-out period with high EoL failure rates. Since early failures are known to be quickly decreasing in most SSDs, most SSD reliability enhancement techniques have focused on extending the useful life period by better managing the flash wear-out speed. In particular, managing a NAND die failure was not the main focus of such techniques.

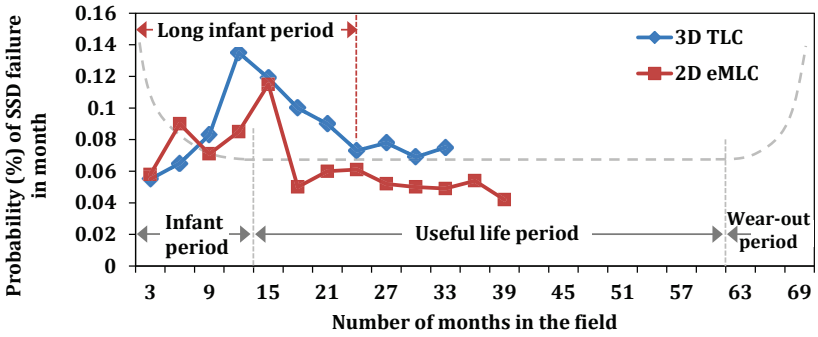


Figure 25: Distributions of SSD failures over the SSD age [2].

However, a recent field study on SSD failure characteristics in enterprise storage systems indicates that a typical bathtub model does not hold for SSD failures [2]. Redrawn from [2], Figure 25 summarizes probability distributions of SSD failures throughout the SSD lifetime for 3D-TLC SSDs and 2D-eMLC SSDs. Note that, as a reference case, a bathtub curve is also shown in a gray dotted curve. As shown in Figure 25, the SSD failure distribution is quite different from a typical bathtub model. The infant period was much longer than that of the bathtub model. Particularly, in 3D-TLC SSDs, the infant period with high failure rates lasted almost two years. Furthermore, there were no wear-out failures in most SSDs, because the number of P/E cycles performed did not exceed its limit even when SSDs have been used for several years. In fact, most SSD failures occurred within the useful life period.

The SSD failure trend reported by [2] strongly suggests that we should focus more on handling sudden SSD failures over EoL SSD

failures. There are many reasons (e.g., firmware bugs) that result in sudden SSD failures, but the SSD capacity is expected to be the most significant factor that decides a sudden failure rate of UL SSDs. Since the number of NAND die failures increases linearly as the SSD capacity, it is more likely that the impact of NAND die failures becomes significant in UL SSDs. Therefore, a new RAID recovery scheme that focuses on die failures is strongly needed.

Another interesting observation in UL SSDs is that a NAND die failure is decoupled from an SSD failure. For example, in a 512-GB SSD with 8 NAND dies, 5460 blocks become bad when one NAND die fails (assuming each die has 5460 blocks). Since 5460 bad blocks outnumber N_{bad}^{ssd}, a single die failure results in an SSD failure. On the other hand, in a 32-TB UL SSD, 5460 bad blocks is only 0.2% of the total (user-visible) blocks whose number is much less than N_{bad}^{ssd}. If a failed die can be recovered, UL SSD has a high potential to tolerate a die failure, preventing an SSD failure.

5.2 Impact of UL SSDs on RAID Reliability

In this section, we explain the key motivations behind reparo using a hypothetical RAID-5 storage system, *UL-RAID(n)*, which employs *n* 32-TB UL SSDs [3]. Since the exact failure rate of a commercial UL SSD is not available, we assume that SSDs in *UL-RAID(n)* follow the SSD failure rates of 3D-TLC SSDs shown in Figure 25. Figure 26

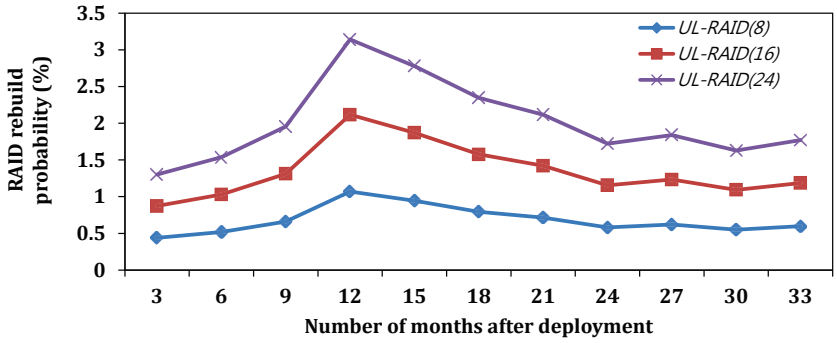


Figure 26: Probability distributions of rebuilding a RAID group over the RAID group size.

shows the probability of RAID rebuilds in *UL-RAID(n)* with varying *n*'s.[1] As the number of SSDs increases in *UL-RAID(n)*, more frequent RAID rebuilds are required. Since RAID rebuilds can interfere with normal host I/O requests, the increased number of RAID rebuilds badly affect user-perceived performance.

Frequent RAID rebuilds also cause more critical reliability issues in *UL-RAID(n)*. First, the probability of deadly double disk failures [57] increases greatly. When the second SSD failure occurs during a RAID rebuild, the failed SSD cannot be recovered thus the user data loss is inevitable. The probability P_2 of a double SSD failure can be represented as follows:

$$P_2 = \frac{MTTR_{ssd} \times (n-1)}{MTTF_{ssd}}, \tag{5.1}$$

[1] We model the number X of failed SSD in *UL-RAID(n)* as $X \sim B(n,p)$ where p is the probability of an SSD failure. The probability of RAID rebuild, therefore, is given by $\sum_{k=1}^{n} \binom{n}{k} p^k (1-p)^{n-k}$.

where $MTTR_{ssd}$ and $MTTF_{ssd}$ indicate the mean time to repair and the mean time to failure of an SSD, respectively. Since $MTTR_{ssd}$ increases linearly over the SSD capacity, P_2 increases linearly as well. For example, in *UL-RAID(16)*, P_2 increases 64 times over that in a RAID array with 16 512-GB SSDs.

Second, the probability of a read failure, P_{rf}, from a latent sector error (LSE) during a rebuild can be increased. The probability of a read failure can be expressed as follows:

$$P_{rf} = 1 - (1 - UBER)^{S_{read}}, \tag{5.2}$$

where $UBER$ (uncorrectable bit error rate) is a fixed value (e.g., 10^{-16}) by SSD manufacturers and S_{read} is the total number of reads during a RAID rebuild. In *UL-RAID(n)*, P_{rf} is significantly increased as well. For example, in *UL-RAID(16)*, P_{rf} increases 52 times over that a RAID group with 16 512-GB SSDs.

Figure 27 shows how much the probability of data loss from double failures and read failure amplifies as *n* increases in *UL-RAID(n)*. All the numbers are normalized over a RAID array with 4 512-GB SSDs. Although we estimated the data loss probability conservatively by assuming that NAND die failures contribute only 3% of the sudden SSD failures in a 512-GB SSD, the data loss probability in *UL-RAID(n)* substantially increases as *n* increases. For example, data loss is more than 1,000 times likely in *UL-RAID(24)* over the baseline RAID with 4 512-

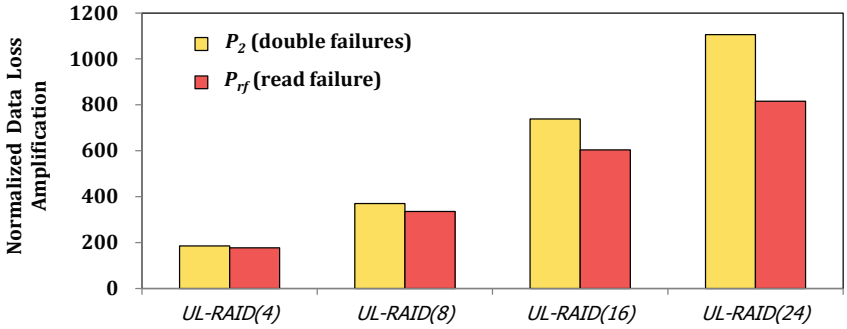


Figure 27: Normalized data loss amplification in *UL-RAID(n)*.

GB SSDs. The results in Figure 27 strongly indicate that *UL-RAID(n)* may not guarantee the same level of data reliability over when a RAID storage system was built using small-sized SSDs. Unless the reliability problem is resolved in an efficient fashion, employing *UL-RAID(n)* in real-world applications may not be practical in a near future.

5.3 RAID Recovery using Reparo

5.3.1 Overview of Reparo

The proposed reparo scheme meets two requirements of a RAID recovery scheme for UL SSDs. Unlike the existing techniques, reparo repairs a failed die, not a failed SSD, by rebuilding the failed die using a RAID logic. Figure 28 shows an organizational overview of the reparo scheme. It consists of three new modules in a UL SSD: die failure detector (*Detector*), failed LBAs identifier (*Identifier*), and data recovery handler (*Handler*). When a die failure is detected by *Detector*, reparo notifies a die failure to the host system (❶). Then

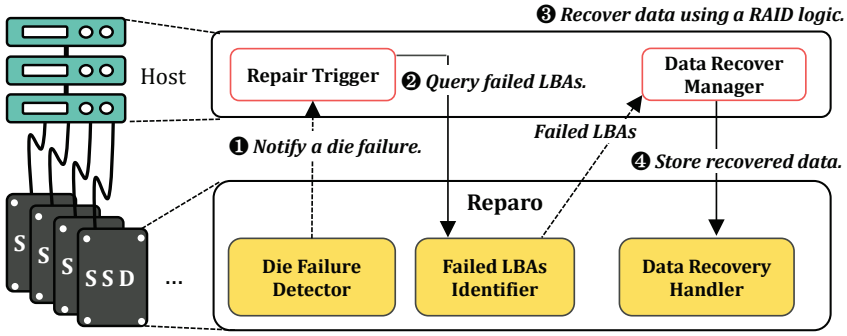


Figure 28: An organizational overview of the reparo scheme.

the host requests a list of LBAs that belong to the failed die (❷). *Identifier* finds the failed LBAs and sends them to the host, and the host recovers the data of the failed LBAs using a RAID logic (❸). When the recovered data are written back to the SSD (❹), *Handler* distributes the data to proper flash cores. When the host queries failed LBAs (❷), it limits the LBA search range (e.g., 64 MB) to control time/resource overheads. Therefore, a recovery sequence (i.e., ❷,❸, and ❹) is repeated until all the LBAs are covered.

In reparo, when a flash core $core_i$'s die fails, we call $core_i$ as the victim core and the rest of flash cores are called helper cores. When only the victim core is used for *Identifier* and *Handler*, we call it an isolated die recovery (IDR) scheme. (We denote this version of reparo by $\text{reparo}_{\text{IDR}}$.) If the helper cores, as well as the victim core, participate for *Identifier* and *Handler*, it is called a cooperative die recovery (CDR) scheme. (Similarly, we denote this version of reparo by $\text{reparo}_{\text{CDR}}$.) We present $\text{reparo}_{\text{IDR}}$ in this section, and $\text{reparo}_{\text{CDR}}$ is described in Section 5.4.

Die Failure Detector

In order to minimize the impact of a failed die on the SSD performance, reparo detects a die failure as early as possible. When a bad block B is found from a die $\mathbb{D}$, *Detector* checks blocks physically adjacent to B by reading the first page of the blocks. If the read operation to the first page fails, *Detector* proceeds to the next block. When the number of accumulated bad blocks in the die $\mathbb{D}$ exceeds a pre-defined maximum number N_{bad}^{die}, *Detector* labels the die $\mathbb{D}$ as a failed die. When a failed die is detected, reparo notifies a die failure to the host system using a well-known host-to-SSD interface (such as SMART [58, 59] or Check condition [60]). In our current *Detector* implementation, a die failure can be detected no later than dozens of milliseconds after the first bad block of a defective die is identified.

Failed LBAs Identifier

Once the host system is notified of a die failure, its recovery module asks reparo for the failed LBAs using the `get_lba_status` command that returns a list of failed LBAs from a specified LBA range $[l_{start}, \ldots, l_{end}]$[2]. If an FTL manages a separate physical-to-logical (P2L) mapping table, *Identifier* can find failed LBAs of the pages in a failed die directly from the P2L mapping table using a physical address of a page in the failed die. Unfortunately, maintaining a P2L mapping table, in addition to an logical-to-physical (L2P)

[2]This command is defined in the industry standard SCSI interface and has been extended to support Rebuild Assist [28] for a fast RAID rebuild.

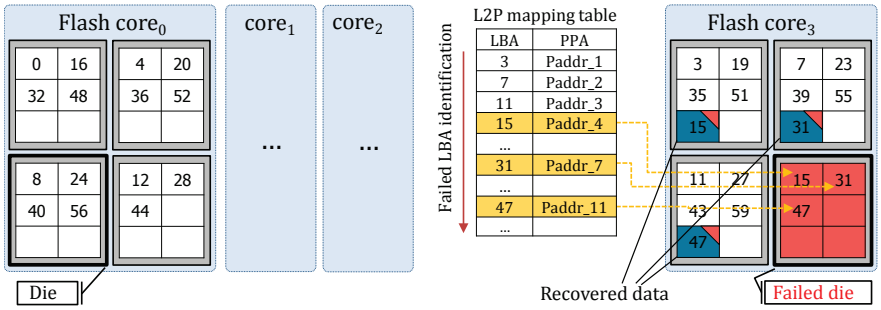


Figure 29: An illustrative example of reparo$_{IDR}$.

mapping table, is not feasible because of its large memory requirement. Typical FTLs manage only an L2P mapping table, which is essential for an FTL operation. Therefore, in order to respond to a failed LBA query, it is necessary to check the L2P mapping table for all LBAs in a query range to validate if they are stored in the failed die. For example, Figure 29 illustrates a case when a die failure occurs in one of the four dies managed by $core_3$. In reparo$_{IDR}$, $core_3$ becomes the victim core which is responsible for *Identifier* and *Handler*. When a query for failed LBAs in the LBA range of 0 to 15 is sent to the SSD, $core_3$ checks if any page in the requested LBA range is mapped to the failed die. Since LBA 15 is stored to a failed die, LBA 15 is sent to a host as a failed LBA. In order to identify all LBAs affected by a failed die, $core_3$ should check all the L2P mapping table entries of the SSD. In a UL SSD, since the number of L2P mapping entries is quite large, it is a key challenge to reduce the time overhead of *Identifier* in reparo.

Data Recovery Handler

After recovering the data of the failed LBAs through a RAID logic, a host stores the recovered data. When the victim core receives a write request, its *Handler* needs to store the recovered data to normal blocks. Since all the blocks in the failed die are bad blocks, *Handler* cannot use a conventional bad block management scheme that uses a reserved block in the same die (i.e., the failed die) for replacing a bad block. As a workaround, a reserved block of another die can be used to store the recovered data. However, this type of block remapping within the victim core complicates the LBA-to-die mapping because multiple LBAs can be mapped to the same die. Instead of the remapping approach, in $\text{reparo}_{\text{IDR}}$, the victim core's *Handler* adjusts all the FTL steps so that they can work without the failed dies. For example, the die-stripping algorithm of the victim core is modified to use one less dies than before a die failure. In Figure 29, *Handler* of $core_3$ skips the failed die so that the recovered data of LBAs 15, 31, 47 are evenly stripped and stored in the remaining dies.

Since only the victim core is used for *Handler* in $\text{reparo}_{\text{IDR}}$, the other cores work as if no die failure has occurred. For example, the master core does not need to change its static core mapping scheme while helper cores work for their allocated dies as usual. On the other hand, the victim core should work with a reduced physical space after the recovery, which can significantly impact the overall SSD performance and lifetime.

5.4 Cooperative Die Recovery

Although reparo$_{IDR}$ of the previous section is simple to implement, reparo$_{IDR}$ can be further improved by relaxing its design constraint that only the victim core is involved in the recovery process. In this section, we describe reparo$_{CDR}$, which improves reparo$_{IDR}$ in two aspects by allowing all the flash cores to be utilized in parallel during the recovery process.

5.4.1 Identifier: Parallel Search of Failed LBAs

Scanning an entire L2P mapping table to find out LBAs belonging to failed dies is a time-consuming operation. Moreover, considering a huge logical address space of UL SSDs, such a scanning operation takes a significant amount of time. For example, it takes at least half an hour for one flash core of a 32-TB Samsung PM1643 to scan the whole L2P mapping table. Since the execution time of *Identifier* can be a bottleneck in the overall recovery process, in reparo$_{CDR}$, we modify *Identifier* so that all flash cores can participate in searching the failed LBAs.

In order to parallelize *Identifier*, we modified the data organization of the L2P mapping table. Since each flash core manages its own logical space, which is separated from the other flash cores, the existing L2P table is structured so that no mapping entry can be shared among different flash cores. In reparo$_{CDR}$, when `get_lba_status` command is issued to the victim core, the L2P mapping entries in

Table 8: Changes in space utilization of IDR scheme.

	Number of dies per flash core					
	4	8	16	32	64	128
victim core (w/ die failure)	33.33%	14.29 %	6.67%	3.23 %	1.59%	0.79%
helper core	0%	0%	0%	0%	0%	0%

the requested search range are first moved to the shared memory area that all the flash cores can access. The copied mapping entries are divided into N distinct regions so that all N flash cores can work in parallel.

5.4.2 Handler: Per-Core Space Utilization Adjustment

The main side effect of the simple $\mathsf{reparo}_{\mathsf{IDR}}$ scheme is that it negatively affects the space utilization of a victim core. The space utilization U_i of a flash core $core_i$, which is defined as a capacity ratio of the logical space to the physical space allocated to $core_i$, is a key SSD metric that is directly related to the performance and lifetime of SSDs. In general, the smaller U_i's, the higher (or the longer) the performance (or lifetime) of an SSD. When each flash core was initially allocated with equal, say x dies, if one of the dies allocated to the victim core $core_v$ fails, U_v increases by $[1/(x-1) \times 100]\%$ over helper cores. For example, when 16 dies are initially allocated to each flash core, U_v increases by 6.7%. Table 8 summarizes the increase in space utilization for a single die failure according to the number of dies per flash core. An increase in U_v can reduce the same amount of the SSD

lifetime where sequential write workloads are dominant (as in modern data-intensive apps). A higher U_v also increases the WAF (Write Amplification Factor) value because the reduced OP space needs more frequent GC invocations. For example, in our evaluation, we observed that when U_v increases by 6.7%, the WAF value can be increased by 166%, which may degrade the SSD lifetime and the performance almost by 60%. In reparo$_{\mathsf{CDR}}$, we modify *Handler* so that the difference in space utilization among flash cores can be minimized.

Per-Core Logical Space Adjustment

In order to reduce the difference between U_v of a victim core $core_v$ and U_h of a helper core $core_h$, we reduce the capacity of logical space of $core_v$ while increasing that of $core_h$.[3] Assume that each flash core $core_i$ with D_i dies is allocated to the logical space LS_i where $LS_i \cap LS_j = \varnothing$ if $i \neq j$, and the capacity of LS_i is $|LS_i|$. We further assume that before a die failure, for all $0 \leq i \leq (N-1)$, 1) $|LS_i| = |LS|/N$ and 2) $D_i = D_{ssd}/N$ where D_{ssd} represents the total number of dies in an SSD[4]. In order to keep all U_i's equal after die

[3]When no OP space becomes available for a helper core in reparo$_{\mathsf{CDR}}$, a failed SSD cannot be repaired anymore. The failed SSD should be replaced by a new SSD in this case.

[4]Our technique can be generalized to a more general setting without these assumptions. However, because of a page limit, reparo$_{\mathsf{CDR}}$ is presented under these assumptions.

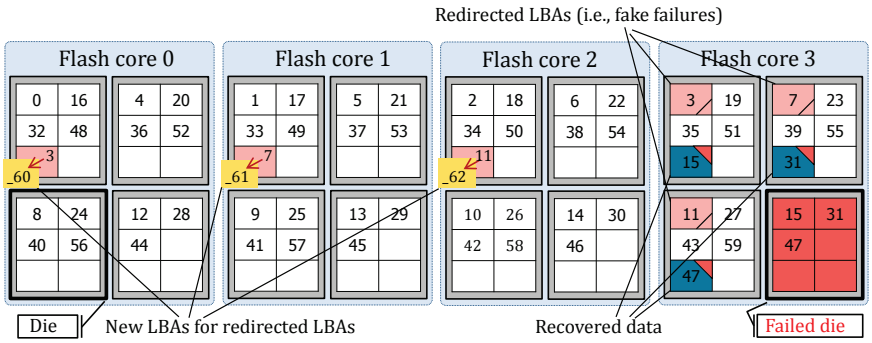


Figure 30: An illustrative example for $\text{reparo}_{\text{CDR}}$.

failure recovery, the following equation should hold:

$$\frac{|LS_h| + \alpha}{D_{ssd}/N} = \frac{|LS_v| - (N-1) \times \alpha}{D_{ssd}/N - 1} \tag{5.3}$$

where α represents the capacity of the extra logical space that should be added to each helper core. Solving Equation (5.3), α can be given as:

$$\alpha = \frac{|LS_h|}{D_{ssd} - 1}. \tag{5.4}$$

Consider an example scenario of a single die failure shown in Figure 30 where an SSD has four flash cores and each flash core has four dies ($N = 4$, $D_{ssd} = 16$ and $D_i = 4$). Assuming the capacity $|LS|$ of logical space of the SSD is 60, $|LS_i| = 15$ for all cores. Since the physical capacity of a NAND die is 6, initially, all U_i values are equal

to 0.625 (i.e., 15/24). When a die fails from $core_3$, α is computed as $|LS_h|/15$, thus increasing the logical capacity of each helper core by 1 while decreasing the logical capacity of the victim core by 3. After this adjustment, all U_i values are still the same, but the space utilization has increased by 6.67% from 0.625 to 0.67. If there were no logical space adjustment, the victim core's space utilization could increase by 33.3% to 0.83. Table 9 summarizes how space utilization changes after logical space adjustment for a single die failure under a varying number of dies per flash core. As expected, in both a victim core and a helper core, space utilization is increased by the same amount when a die fails. Furthermore, the increased amount of space utilization of the victim core is much lower compared to that of the IDR scheme because of the shared space adjustment among all cores.

Note that in the above description of the logical space adjustment technique, we assumed that all U_i values and D_i values were equal before a single die failure occurs. Since dies can fail more than once in an SSD, these assumptions generally do not hold and Eqs. (3) and (4) need to be modified. In case of multiple die failures, various failure combinations are possible. For example, multiple failures may be focused on a single flash core or they may be spread among multiple flash cores. Although treating an individual failure case using a case-specific equation will be the most accurate solution, we found that its management overhead can be substantial. Instead, we empirically evaluated the accuracy of space adjustment from Eq. (4) in multiple die failures. Our evaluation results showed that the difference between

Table 9: Changes in space utilization after the adjustment.

	Number of dies per flash core					
	4	8	16	32	64	128
victim core (w/ die failure)	6.67%	3.23%	1.59%	0.79%	0.39%	0.20%
helper core	6.67%	3.23%	1.59%	0.79%	0.39%	0.20%

the ideal adjustment solution and one from Eq. (4) was negligible. For example, in the case of four die failures, the worst case for Eq. (4) is when all four die failures occur in the same flash core. Even in this case, when each core has 64 or more dies (i.e., as in UL SSDs), α from Eq. (4) was only 0.06% apart from the ideal adjustment value.

Selective LBA Redirections

In order to implement space utilization adjustment among flash cores as described above, we need to redirect $(N-1) \times \lfloor \alpha \rfloor$ LBAs from the victim core to $(N-1)$ helper cores while each helper core receives $\lfloor \alpha \rfloor$ additional LBAs for a die failure. Since the master core is responsible for distributing a host request to a proper flash core, all the redirection decisions are made at the master core without modifying how the flash cores work. The master core forms a unit of LBA redirections by $(D_{ssd}-1)$ consecutive LBAs that were originally mapped to the victim core. From each redirection unit, the master core selects $(N-1)$ LBAs and redirects the LBAs to helper cores one by one. Therefore, $(N-1)$ LBAs out of $(D_{ssd}-1)$ LBAs of the victim core are redirected, effectively reducing its logical space by $(N-1)/(D_{ssd}-1)$. For example, in Figure 30, 15 LBAs form one

redirection unit. Out of 15 LBAs in one unit, 3 LBAs are redirected to $core_0, core_1$ and $core_2$.

In order to choose $(N-1)$ redirected LBAs from a redirection unit of $core_v$, the master core considers the first N LBAs from the redirection unit. Except for the v-th LBA, the i-th LBA is redirected to $core_i$. Formally, assume that the master core tries to select $(N-1)$ LBAs from a redirection unit $R = \{l_0, \ldots, l_{D_{ssd}-1}\}$ of $core_v$. Since each $l_i \in R$ can be expressed by $l_i = j \times N + v$ (where $(D_{ssd}-1) \times p \leq j < (D_{ssd}-1) \times (p+1)$ for $p \geq 0$), $[j \ \% \ (D_{ssd}-1)]$ indicates the redirected core if it is not v and less than N. For example, in Figure 30, $R = \{3, 7, 11, 15, 19, \ldots, 59\}$. First 3 LBAs, $3 (= 0 \times 4 + 3)$, $7 (= 1 \times 4 + 3)$, and $11 (= 2 \times 4 + 3)$ are redirected to $core_0$, $core_1$ and $core_2$, respectively.

When an LBA is redirected to a helper core, the redirected LBA is stored in the extended LBA space $LS_{redirect}$ of the helper core which is hidden from the host system. Each flash core internally maintains its $LS_{redirect}$ area so that when the master core sends a request of a redirected LBA to its $LS_{redirect}$ area, it can be properly handled. In order to distinguish $LS_{redirect}$ from the host-visible logical space, we denote an LBA in $LS_{redirect}$ with the preceding underscore such as _60. In Figure 30, for example, three LBAs, 3, 7 and 11, of the victim core are redirected to three $LS_{redirect}$ LBAs, _60, _61 and _62, respectively.

Data Migration with Fake Failures

Since the master core changes its LBA-to-core mapping algorithm after a failed die is detected, if data are already written to the redirected LBAs of the victim core, they should be moved to the newly redirected cores as well. However, since flash cores operate independently, direct data transfers between flash cores are difficult to implement. As an effective trick to this problem, we consider those redirected LBAs as failed LBAs although they do not belong to the failed die (i.e., the redirected LBAs are treated as fake failures.). In addition to the real failed LBAs, *Identifier* additionally searches fake failures and sends their LBAs to the host as well. When fake failures are recovered by the host, they are sent to the master core which then correctly redirects them to their new locations. Although this method incurs an additional RAID recovery cost, it is simple to be implemented as the existing recovery path is used. Furthermore, since the number of LBAs reported as fake failures is limited by the number of redirected LBAs (i.e., $(N-1) \times \alpha$), its overhead is not significant. For example, in Figure 30, three LBAs are reported as fake failures (because $\alpha = 1$ and $N = 4$).

5.5 Identifier Acceleration Using P2L Mapping Information

As mentioned in Section 5.3, SSDs do not generally manage P2L mapping information that can be used in identifying LBAs from a

failed die. In this section, we propose two P2L management methods and *Identifier* acceleration techniques that can further reduce the key performance bottleneck of reparo.

5.5.1 Page-level P2L Entrustment to Neighboring Die

A NAND page consists of two areas, one for storing data and the other for storing an error-correction code and various FTL metadata. The latter area, called as the spare area of the NAND page, stores key information for operating an SSD reliably. One such information is the LBA of the data stored in the NAND page. The LBA information in the spare area is used when the mapping information of an SSD is reconstructed from unexpected failures (such as sudden power-off failures [61]) or when valid pages of the GC victim block are moved to different blocks. Therefore, it is possible to find out which LBAs are stored in a specific die by checking its spare area. However, when a die failure occurs, LBA information stored in the spare area of the failed die becomes inaccessible as well, thus making it impossible to read stored LBAs of pages in the failed die.

In order for the FTL metadata on the spare area to be available even when a die fails, the data page and its FTL metadata should be stored in different dies. In general, it is quite inefficient to store a page data and its metadata on two separate pages because doing so requires two writes. However, in reparo, we propose a simple extension to a superblock-based mapping scheme [62,63] so that a data page and

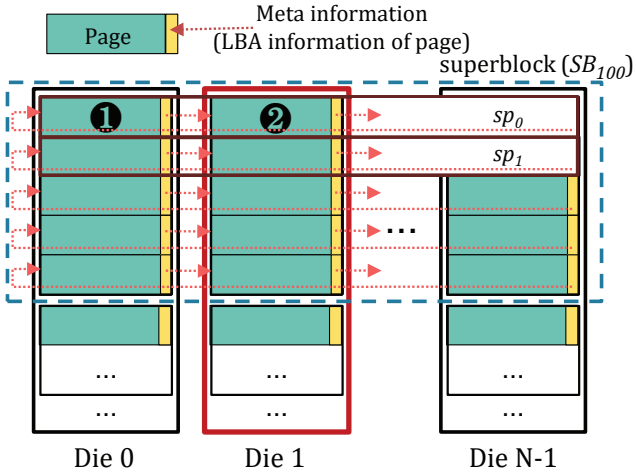


Figure 31: Illustrative examples of Page-level P2L entrustment.

its metadata can be stored without an extra write overhead on different pages in different dies. Unlike a page-level mapping scheme, the superblock-based mapping technique, which is widely used in practice, employs a superpage as a write unit. In order to support a superpage, a superblock is formed from k different blocks in k different dies. For example, Figure 31 shows that the superblock SB_{100} consists of N blocks from N different dies. A superpage of a superblock consists of k pages from k blocks (that are members of the superblock) where all k pages have the same page offset within their blocks. For example, in Figure 31, the superpage sp_0 consists of N pages that have the page offset of 0 within their blocks. In the superblock-based mapping technique, since a write to a superpage requires k writes to k pages (in different dies), we can easily separate data page and its metadata to two neighboring pages within the same superpage.

In the page-level P2L entrustment technique, we store data page and its metadata in two pages that are immediate neighbors within a given superpage. When a die fails, *Identifier* only needs to check the spare area of a neighboring die, instead of checking all the entries in the L2P mapping table. For example, as shown in Figure 31, when Die 1 fails, the failed LBAs of Die 1 can be identified from the spare area of the adjacent die, Die 0. The LBA information of the page ❷ in the superpage sp_0 can be obtained from the spare area of the page ❶ in Die 0.

5.5.2 Block-level P2L Entrustment to Neighboring Die

Although the page-level P2L entrustment technique can be efficiently implemented using the extended superblock-based mapping scheme, it incurs a significant overhead for *Identifier* because all the pages of a neighboring die should be read. In order to identify failed LBAs more efficiently, we propose a block-level P2L entrustment technique that stores all the LBAs from a neighboring block in a single reserved page of a block, thus reducing the number of page reads per block by *Identifier* from the number of pages in a block to one. We use the last page of each block for this purpose.[5] Figure 32 illustrates how the block-level P2L entrustment scheme works. For a given superblock (e.g., SB_{101}), as with the page-level P2L entrustment scheme, each

[5] In order to satisfy the sequential program constraint of the NAND flash memory, a superpage is sequentially written in a superblock. Therefore, it is logical to store the LBA list of a superblock to its last superpage.

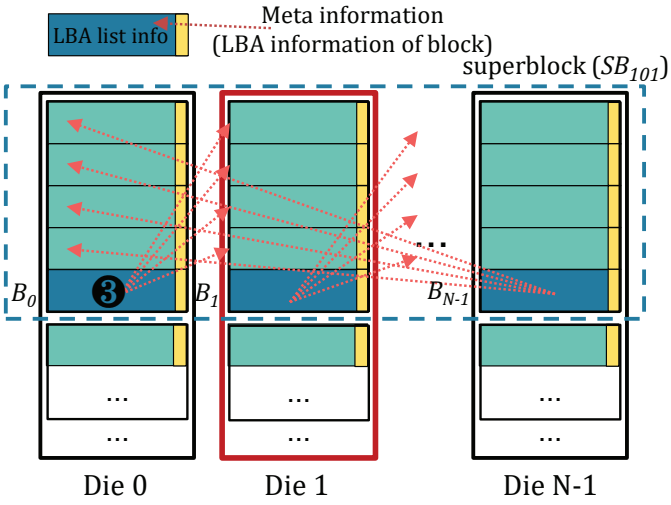


Figure 32: Illustrative examples of Block-level P2L entrustment.

block's LBA information is stored in its neighboring block. However, in the block-level P2L entrustment technique, we dedicate the last page of each block for storing P2L mapping information of the neighboring block. When Die 1 fails, the failed LBAs of the block B_1 can be identified by reading the page ❸ of the block B_0. Since all the failed LBAs in a failed block can be found with a single page read, *Identifier* can work very efficiently. We denote an extended $\text{reparo}_{\text{CDR}}$ with the block-level P2L entrustment technique by $\text{reparo}_{\text{CDR}}$*.

In the block-level P2L entrustment scheme, a single page should be able to contain all the LBA information of a block. Although recent NAND flash memory has a large number (e.g., 768) of pages in a block, a single page can easily meet this requirement. For example, consider a block with 768 pages where the page size is 16 KB. If

we support a common 4 KB-based mapping scheme, there are 3072 mapping units in a block. When each mapping unit is referenced by a 32-bit address, all the LBA lists of a block can be stored on a single page (i.e., 12 KB ¡ 16 KB). Furthermore, whenever a program operation is performed on a block, FTL needs to accumulate the list of all LBAs stored in each block in the buffer memory until all the pages in the block are programmed except for the last page. Since most FTLs limits the number of active blocks (where a requested page write is programmed) to 1 or 2 [64,65], the memory requirement for buffering the LBAs of programmed pages is reasonable compared to the SSD capacity. For example, assuming that 2 active blocks per NAND die are maintained in a 32-TB SSD with 512 NAND dies, a 12-MB buffer memory is sufficient.

5.5.3 Additional Considerations for P2L Entrustment

The proposed P2L entrustment schemes may not work when more than one die fail at the same time. For example, if two neighboring dies fail together, failed LBAs of one die cannot be identified. In this thesis, we assume that multiple die failures are possible but they do not occur at the same time. Since a die failure is a rare event and each die is physically independent of the other die, it is a reasonable assumption in practice.

When failed LBAs are identified by $\mathsf{reparo}_{\mathsf{CDR}}$*, they are not properly sorted because when they were stored to the last page of a block,

they were not sorted. However, when a host queries failed LBAs after a die failure is detected (as shown in ❷ of Figure 28), the host asks reparo of a list of failed LBAs within a specific LBA range. A naive solution would be to sort identified failed LBAs before reparo responds to the host query. However, sorting a large number of failed LBAs can be time-consuming. In the current implementation, when the failed LBAs are decided by $\text{reparo}_{\text{CDR}}$*, their L2P mapping entries are marked as failed LBAs. With a simple modification to the L2P mapping table, when the host queries for failed LBAs from a specific LBA range, $\text{reparo}_{\text{CDR}}$* can quickly identify the failed LBAs without a costly sorting step.

5.6 Experimental Results

5.6.1 Experimental Settings

In order to evaluate the effectiveness of the proposed reparo schemes, we implemented the proposed schemes in Samsung PM1643 SSD [3] which can be configured for a 4-TB SSD (with 64 dies) and a 32-TB SSD (with 512 dies). The SSD controller of a PM1643 SSD consists of four flash cores along with one master core (as described in Section 2.3). Since there are four flash cores, each flash core handles one quarter of the NAND dies in the SSD. We set the initial space utilization of each core to 0.9. We assume a storage system with 8 SSDs configured in RAID 5. In order to emulate die failures during run time, we added a special command that imitates a real die failure to

PM1643 firmware. The special command, which was implemented as a vendor-specific command of SCSI [60], makes all NAND operations fail to a selected NAND die.[6] In order to simulate a die failure, this special command was requested from a test software (e.g. DriveMaster [66]).

We compared four techniques: baseline, reparo$_{IDR}$, reparo$_{CDR}$, and reparo$_{CDR}$*. The baseline scheme, which represents a state-of-the-art RAID recovery technique, is based on Rebuild Assist [28] which was proposed for a fast RAID recovery. In baseline, when a die failure occurs, it is considered as an SSD failure, and it replaces the failed SSD with a reserved SSD by copying all the valid pages in the failed SSD to the reserved SSD. During an SSD rebuild, baseline directly copies readable pages of the failed SSD using Rebuild Assist [28]. Reparo$_{CDR}$* works in the same way as reparo$_{CDR}$ except that its *Identifier* is optimized using the block-level P2L entrustment technique.

In order to evaluate four techniques, we have used two benchmark suites: FIO benchmark [67] and Iometeter benchmark [68]. Using simple synthetic workloads (e.g., sequential read/write and random read/write) from the FIO benchmark, we compare how key steps of the recovery process work differently in the proposed schemes. In order to understand the effect of the proposed schemes in real-world settings, we used five enterprise application workloads [69] from the Iometer benchmark. Table 10 summarizes key I/O characteristics of

[6] In the current implementation, we modified the NAND flash reliability parameters (e.g., reference voltages) so that normal operations cannot be performed.

Table 10: I/O workload characteristics of five enterprise applications.

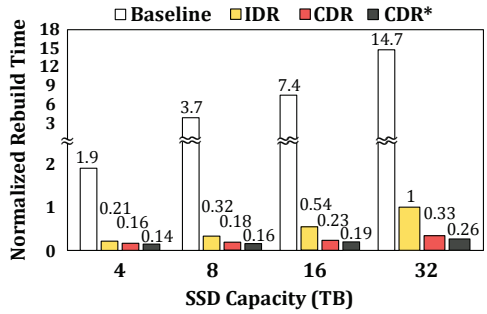
Application	Transfer Size	Read %	Write %	Random %	Sequential %
Media Streaming	64K	98	2	0	100
File Servers	8K	90	10	75	25
Database OLTP	8K	70	30	100	0
Archive	2M	55	45	95	5
Medical Imaging	1M	5	95	5	95

these workloads.

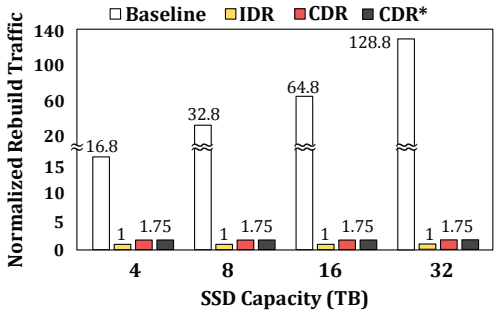
5.6.2 Experimental Results

Recovery overhead

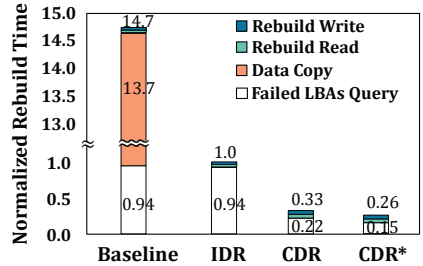
In order to compare the recovery overhead, we measured the rebuild time of each scheme in case of a die failure. Figure 33(a) compares normalized rebuild times of four techniques under varying SSD capacity. All the measurements were normalized over the result of reparo$_{\mathsf{IDR}}$ for a 32-TB SSD. Reparo$_{\mathsf{IDR}}$ completes the recovery process about 9.3 $\sim$ 14.7 times faster than baseline by minimizing data migration. Reparo$_{\mathsf{CDR}}$ is about 3.1 times faster over reparo$_{\mathsf{IDR}}$ because of its parallel *Identifier* module while reparo$_{\mathsf{CDR}}$* is about 1.27 times faster over reparo$_{\mathsf{CDR}}$ with its P2L entrustment support. Overall, reparo$_{\mathsf{CDR}}$* is 57 times faster than baseline in a 32-TB SSD, even when the full I/O bandwidth of a host system is used for the RAID recovery. If the I/O bandwidth of RAID recovery is limited (e.g., only 10% of the total I/O bandwidth for the host) to minimize the impact of the RAID recovery on the performance of host request processing, reparo$_{\mathsf{CDR}}$* is about 110 times faster than baseline which takes more than 3 days in



(a) Rebuild time comparison.



(b) Rebuild traffic comparison.



(c) A rebuild time breakdown of 32-TB SSD.

Figure 33: Comparisons of rebuild overhead.

a 32-TB SSD. Note that the rebuild time of reparo techniques very slowly increases over baseline when the SSD capacity increases. This observation illustrates the advantage of the reparo schemes that repair a single die instead of rebuilding an SSD as in baseline.

Figure 33(b) compares four techniques in terms of the total amount of data movements during the rebuild time. Compared to baseline, the reparo schemes require up to two orders of magnitude fewer data movements during the recovery. Unlike baseline, the reparo schemes generate the same amount of rebuild traffic during the recovery re-

gardless of the capacity of SSDs because they rebuild a failed die only. Although reparo$_{CDR}$ moves more data over reparo$_{IDR}$ because of logical space adjustment, the overall rebuild time of reparo$_{CDR}$ is smaller than reparo$_{IDR}$ because reparo$_{CDR}$ significantly reduces time for finding failed LBAs by parallel query processing. Figure 33(c) shows a detailed breakdown of the total rebuild time during the rebuild process in a 32-TB SSD. As shown in reparo$_{IDR}$, the time taken by *Identifier* is the dominant factor of the total recovery time. The parallel *Identifier* of reparo$_{CDR}$ reduces the *Identifier* execution time by 76% over reparo$_{IDR}$. Reparo$_{CDR}$* further reduces the *Identifier* execution time by 31% over reparo$_{CDR}$ by the P2L entrustment technique.

In order to better understand the effect of various *Identifier* optimization techniques, we compared the performance of the failed LBA identification step in detail under varying number of flash cores participating in the parallel search and different P2L entrustment techniques. Figure 34 shows the normalized performance of the failed LBAs identification step for different SSD capacities. (The identification performance represents the processing speed measured from a host after a failed LBA query was sent to an SSD. The higher the identification performance, the shorter *Identifier* takes.) All the values were normalized over the query processing speed on a 4-TB SSD when a single core was used. As the capacity of the SSD increases, the identification speed tends to increase. This tendency is related to how often failed LBAs appear. For example, on a 4-TB SSD, on average, one of the 16 LBAs in the victim core is stored on the failed die,

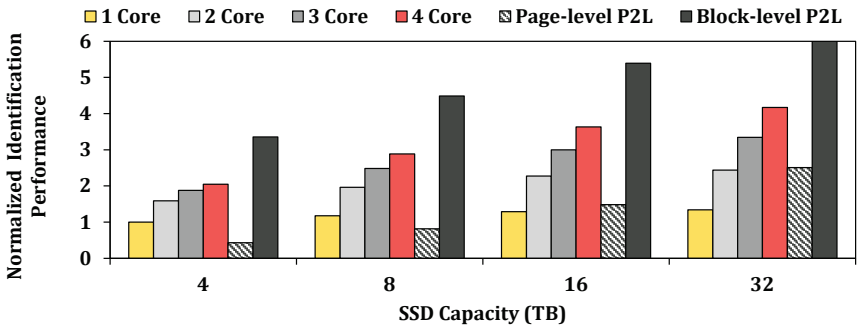


Figure 34: Comparisons of *Identifier* performance.

whereas on a 32-TB SSD, one of the 128 LBAs on the victim core is stored on the failed die. Whenever a failed LBA is identified, an internal data structure update is required to transmit the information to the host, and the mapping information related to the failed LBA needs to be updated. This is why the higher the frequency of failed LBAs, the slower the query processing speed. The high frequency of failed LBAs also gives a negative effect on the efficiency of parallel processing. Therefore, on a 4-TB SSD, the performance of the 4-core parallel search is about twice that of the single-core search. On the other hand, on a 32-TB SSD, the performance of the 4-core parallel search is 3.1 times higher than that of the single-core search.

As shown in Figure 34 the page-level P2L entrustment technique is quite slow because it needs to read a large number of pages to identify failed LBAs from neighboring pages. For example, it takes longer than the 4-core parallel search case in all four SSDs. On the other hand, the block-level P2L entrustment technique can identify failed LBAs much faster than the 4-core parallel search on all SSD capaci-

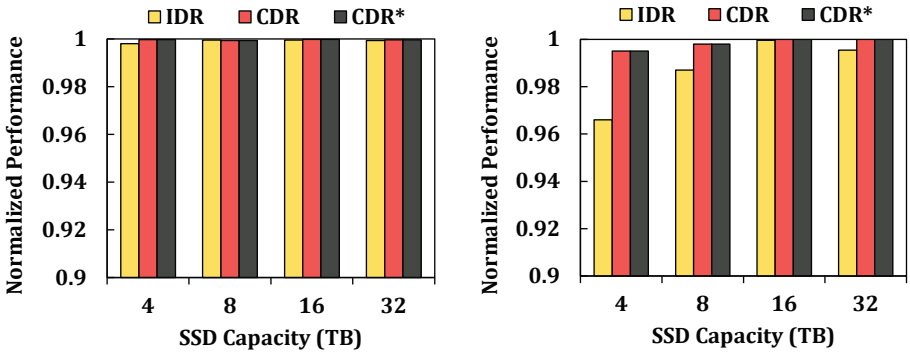
ties. For example, on a 16-TB SSD and a 32-TB SSD, the block-level P2L entrustment technique outperforms the 4-core parallel search by 48% and 44%, respectively.

Post-recovery performance/WAF impact

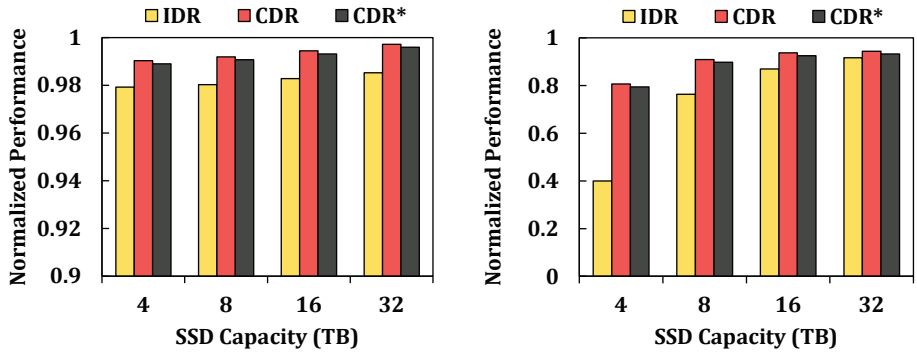
In order to understand how a reparo-repaired SSD behaves in performance and lifetime aspects, we measured IOPS and WAF values after a single die failure was repaired by the reparo schemes. All the measurements are normalized over those of a new SSD without any die failure. We used four representative synthetic workloads generated through FIO.

Figure 35 compares IOPS values between the reparo schemes under four workloads. For the sequential read and random read workloads shown in Figure 35(a) different reparo schemes show almost the same performance as the new SSD. $\text{Reparo}_{\text{IDR}}$ has a performance drop up to 3.4% for random read under low-capacity conditions when the SSD capacity is relatively low (i.e., 4 TB), but the performance degradation is marginal in the other conditions. This is because the read bandwidth of the NAND flash is sufficient to satisfy the host interface speed even if the number of dies is reduced.

On the other hand, for write workloads, there is a little more performance differentiation among the proposed schemes although their difference is not significant. As shown in Figure 35(b), in the case of sequential write workload, small performance differences are largely from the reduced NAND parallelism. For example, $\text{reparo}_{\text{IDR}}$ is 1.1%



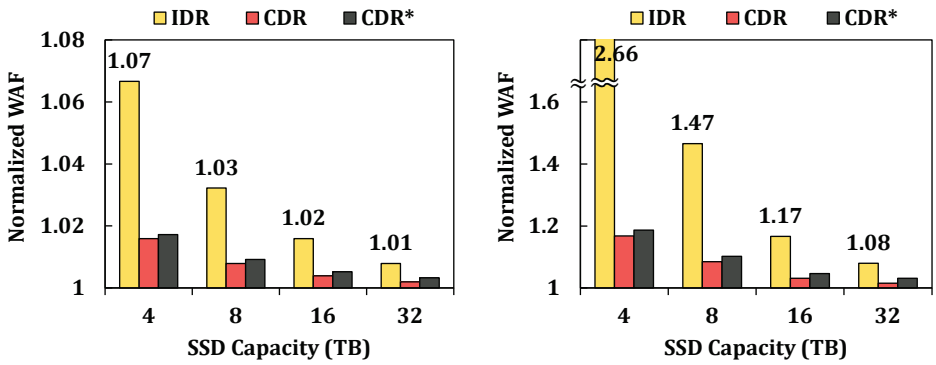
(a) Sequential read and random read performance.



(b) Sequential write and random write performance.

Figure 35: Performance comparisons after die failure recovery.

worse than $\text{reparo}_{\text{CDR}}$ in a 4-TB SSD because its NAND parallelism is affected by the isolated die recovery scheme. In the random write workload, the performance difference between $\text{reparo}_{\text{IDR}}$ and $\text{reparo}_{\text{CDR}}$ is substantial because the efficiency of GC is combined with the proposed schemes. $\text{Reparo}_{\text{CDR}}$ outperforms $\text{reparo}_{\text{IDR}}$ from 3% to 102%. This is mainly because $\text{reparo}_{\text{CDR}}$ better manages the performance-critical space utilization of flash cores over $\text{reparo}_{\text{IDR}}$. Imbalanced space utilization in $\text{reparo}_{\text{IDR}}$ directly affects the efficiency of GC, which, in



(a) Sequential write workload.

(b) Random write workload.

Figure 36: Impact of die repairs on WAF.

turn, significantly degrades the overall IOPS. $\text{Reparo}_{\text{CDR}}$* shows almost the same performance as $\text{reparo}_{\text{CDR}}$, with only a performance drop of less than 1.5% because $\text{reparo}_{\text{CDR}}$* uses the last page of each block for storing a list of LBAs, which reduces the capacity of the OP space slightly.

Figure 36 compares WAF values between the reparo schemes under two different write workloads. WAF values are normalized over the baseline case where no die failure occurs in a given SSD capacity. For the sequential write workload, as shown in Figure 36(a), WAF values do not increase significantly. Except for $\text{reparo}_{\text{IDR}}$ whose WAF value increased up to 7% at a 4-TB SSD, all the other schemes increased their WAF values by less than 2% over four different SSDs with different capacities.

On the other hand, as shown in Figure 36(b), in the random write workload, there are larger variations in WAF values among different

schemes in different SSDs. For example, the WAF value increases by 47% in $\text{reparo}_{\text{IDR}}$ for an 8-TB SSD while the WAF value increases less than 9% in $\text{reparo}_{\text{CDR}}$ for the same SSD. Large differences in Figure 36(b) come from the efficiency of GC in each scheme because each scheme affects differently on the available OP space. As shown in Figure 36(b), $\text{reparo}_{\text{CDR}}$ is the most efficient in maintaining the available OP space evenly among flash cores. $\text{Reparo}_{\text{CDR}}$*, which has a smaller effective OP space than $\text{reparo}_{\text{CDR}}$, results in an additional 1.5% increased WAF value.

Figure 37 compares the performance of each scheme after a die failure is recovered using enterprise workloads from the Iometer benchmark. All the measurements were normalized over the SSD performance of the same capacity with no die failure. As expected, there are large differences among $\text{reparo}_{\text{IDR}}$ and $\text{reparo}_{\text{CDR}}$ on a small SSD when workloads are write-intensive. For example, in Archives and Medical Imaging workloads with high write request ratios, $\text{reparo}_{\text{CDR}}$ outper-

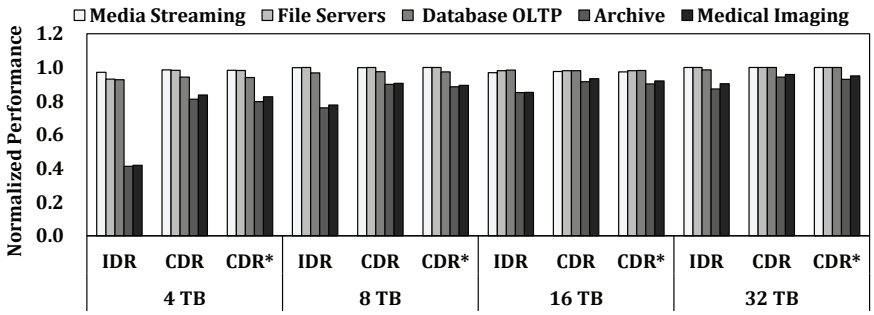


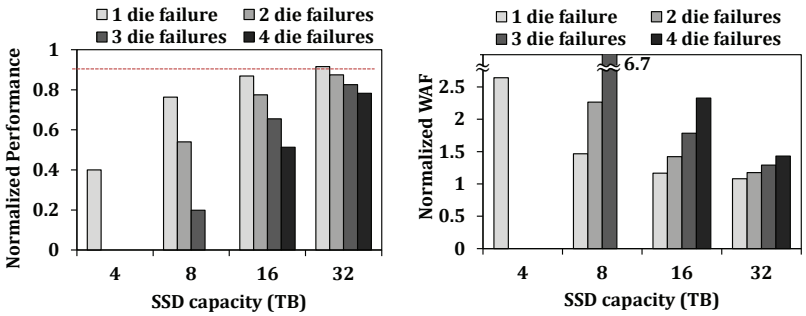
Figure 37: Performance comparisons after die failure recovery with enterprise workloads.

formed reparo$_{\mathsf{IDR}}$ by 96.4% and 99.5%, respectively on a 4-TB SSD. Even for write-intensive workloads, if the access pattern is sequential, the influence of the OP space can be low. However, when the random and sequential patterns are mixed, the influence of the OP space is large, similar to the 100% random pattern even if the random ratio is low. On the other hand, in read-intensive workloads, there is not much difference in performance for each scheme, which is in line with the evaluation results of Figure 35(a).

Note that we did not directly compare the lifetime impact of each scheme. However, since the total amount of written data to an SSD can be a useful indicator of the SSD lifetime, we can estimate the lifetime impact of each scheme using its WAF value w. Given a fixed amount W_{host} of host writes, the total amount T_{data} of written data to the SSD can be computed by $T_{data} \times w$. Using WAF values of Figure 36, for example, we can estimate that reparo$_{\mathsf{IDR}}$ can decrease the SSD lifetime over when no die fails, by 2.9% and 31.9%, respectively, for sequential workload and random workload, on an 8-TB SSD after a single die is repaired. For the same conditions, reparo$_{\mathsf{CDR}}$ shows a longer SSD lifetime, reducing the SSD lifetime by 0.8% and 7.8% only.

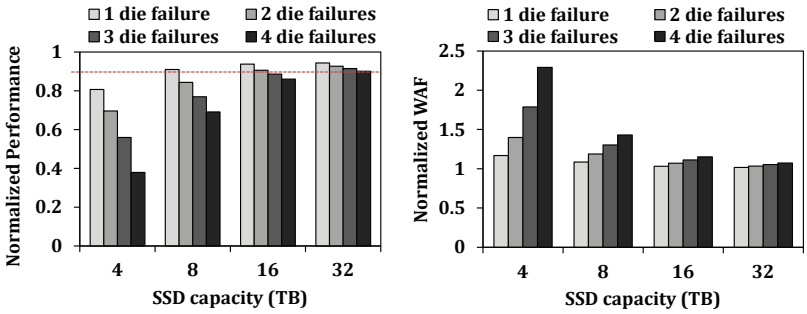
Post-recovery impact of multi-die failures

Unlike when a single die failure is repaired, reparo$_{\mathsf{IDR}}$ and reparo$_{\mathsf{CDR}}$ shows significant differences in the post-recovery performance/WAF when multiple die failures are repaired. Figures 38 and 39 show how performance and WAF changes under the random write workload as



(a) Performance changes. (b) WAF changes.

Figure 38: Impact of multi-die failures in $\mathsf{reparo}_{\mathsf{IDR}}$.

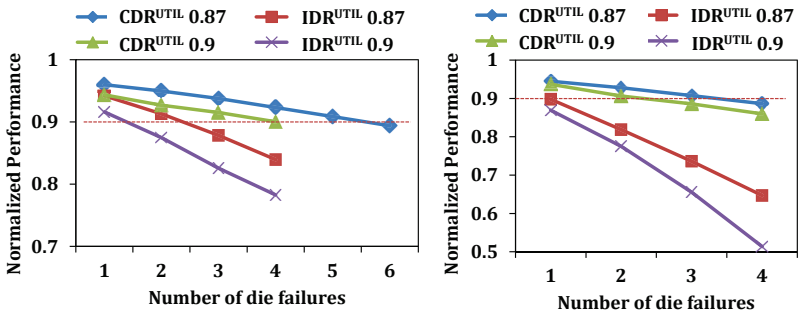


(a) Performance changes. (b) WAF changes.

Figure 39: Impact of multi-die failures in $\mathsf{reparo}_{\mathsf{CDR}}$.

the number of failed dies increases assuming that all die failures occur in the same flash core, considering the worst case. As shown in Figures 38(a) and 38(b), when an SSD is repaired by $\mathsf{reparo}_{\mathsf{IDR}}$, its performance is quickly degraded while its WAF is rapidly increased. On the other hand, as shown in Figures 39(a) and 39(b), when an SSD is repaired by $\mathsf{reparo}_{\mathsf{CDR}}$, its performance is much slowly degraded as with the WAF increase.

Slow performance degradation of $\mathsf{reparo}_{\mathsf{CDR}}$ can be an important



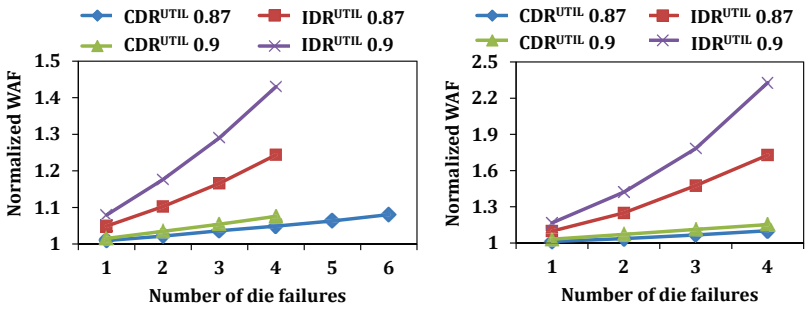
(a) 32-TB SSD. (b) 16-TB SSD.

Figure 40: Impact of the space utilization on performance.

advantage in enterprise storage systems. In such systems, in order to support the sustained RAID performance [70], each SSD has a strict requirement on the performance degradation (such as the maximum 10% performance drop) [71]. For example, when the maximum performance drop is set to 10%, $\mathsf{reparo}_{\mathsf{CDR}}$ can survive up to 4 die failures in a 32-TB SSD. On the other hand, $\mathsf{reparo}_{\mathsf{IDR}}$ can only handle a single die failure under the same condition.

Sensitivity for the space utilization

Since die failure recovery is performed by utilizing the OP space, the available size of the OP space affects the performance and WAF after die failure recovery process. If the space utilization of an SSD is low, more OP space is available and the performance impact by using reparo is reduced. To validate this observation, we measured performance and WAF while varying the space utilization from 0.9 to 0.87.



(a) 32-TB SSD. (b) 16-TB SSD.

Figure 41: Impact of the space utilization on WAF.

Figure 40 shows how performance changes for successive die failures under different space utilization ratios. When space utilization is low, the performance degradation is also slowed accordingly. For example, in a 32-TB SSD, when the space utilization ratio is 0.87, $\mathsf{reparo}_{\mathsf{CDR}}$ can repair up to five die failures with a less than 10% performance degradation over four die repairs when the space utilization ratio is 0.90. Similarly, Figure 41 shows how WAF changes for successive die failures under different space utilization ratios.

Chapter 6

Conclusions

6.1 Summary

UL SSDs are becoming popular these days in enterprise storage markets because of their advantages in reducing the total cost of ownership. However, the capacity-enhancing technologies that have made UL SSDs possible introduced new drawbacks to the wider adoption of UL SSDs in practice.. In order for UL SSDs to satisfy the high-performance and high-reliability requirements of modern computing systems as well as the large-capacity requirement, this reliability/performance problems should be properly addressed.

In this dissertation, we have proposed several system-level techniques which aim at alleviating the reliability and performance degradation originated from capacity-oriented design decisions.

First, We have presented GuardedErase scheme to reduce WL characteristic variation by reducing an erase voltage applied to weak WLs. From a 3D NAND flash characterization study, we built nine gE(n) modes that can maximize NAND endurance in a condition of different physical capacity reduction. Based on gE(n) modes, we have implemented a gErase-aware FTL, longFTL, which apply the optimal gErase mode considering user workload characteristic. Our experimen-

tal results show that longFTL can improve expected the SSD lifetime by 21% on average while insignificant degradation on the SSD performance.

Second, we have presented effective data migration optimization to increase the SSD performance by using the copyback operation. We proposed an integrated approach that maximizes the efficiency of copyback operations while not compromising data reliability. From the error propagation characteristics of 3D NAND flash, we proposed a novel per-block error propagation model under consecutive copyback operations. Furthermore, we devised a resource-efficient error management scheme that can handle successive copybacks where pages move around multiple blocks with different reliability.

Finally, we have presented a new recovery scheme, called reparo, for a RAID storage system with ultra-large SSDs. Proposed reparo scheme repairs a failed SSD at the NAND die granularity without replacing it with a new SSD, thus avoiding most of the inter-SSD data copies during a RAID recovery step. By exploiting a multi-core processor of the SSD controller in identifying failed LBAs from the failed NAND die and recovering data from the failed LBAs, the die failure recovery overhead is minimized.

6.2 Future Work

6.2.1 Optimization with Accurate WAF Prediction

The current version of longFTL can be further improved in several directions. For example, if the impact of gE(n) mode on WAF can be predicted with high precision, optimal gE(n) mode can be selected more effectively. In addition, it could also effectively respond even in unusual situations where user workload characteristics are constantly changing. Building such a predictor would be an interesting future direction if it can be possibly combined with a data-driven machine learning approach.

6.2.2 Maximizing Copyback Threshold

The current version of rcFTL can be extended in several directions. For example, if rcFTL can better estimate the data retention requirement, a higher copyback threshold could be used for the same block characteristics. As shown in Table 6, if a group of pages require less than 3-month retention time, all data migrations can be supported by rCPB up to 6 times. Furthermore, if we can manage the page characteristics difference , the copyback threshold can be further increased. In the current version, only the deviation between blocks is considered, but since the deviation of the characteristics between pages within a block is also very large, a more accurate copyback threshold can be predicted if both factors are considered. However,

since it requires more complex management techniques and resources in FTL, additional techniques are needed to effectively manage them.

6.2.3 Pre-failure Detection

The current version of reparo can be further improved in several directions. For example, if a die failure can be predicted with a high probability before they actually occur, the die failure can be handled more effectively. In addition, if data migration between flash cores can be performed inside the SSD, unnecessary data transfers could be reduced during the recovery process.

Bibliography

[1] Youngseop Shim, Myungsuk Kim, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim. Exploiting process similarity of 3d flash memory for high performance ssds. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2019.

[2] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. A study of ssd reliability in large scale enterprise storage deployments. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2020.

[3] Samsung ssd. https://www.samsung.com/semiconductor/insights/news-events/samsung-starts-producing-industrys-largest-capacity-ssd/.

[4] Bianca Schroeder and Garth Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of USENIX Conference on File and Storage Technologies*, 2007.

[5] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. Lifetime improvement of nand flash-based storage systems using dynamic program and erase scaling. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2014.

[6] Fei Wu et al. Fastgc: Accelerate garbage collection via an efficient copyback-based data migration in ssds. In *Proceedings of Design Automation Conf.*, 2018.

[7] Jungdal Choi and Kwang Soo Seol. 3d approaches for non-volatile memory. In *Proceedings of Symposium on VLSI Technology-Digest of Technical Papers*, 2011.

[8] Youngwoo Park, Jaeduk Lee, Seong Soon Cho, Gyoyoung Jin, and Eunseung Jung. Scaling and reliability of nand flash devices. In *Proceedings of the IEEE International Reliability Physics Symposium*, 2014.

[9] Chulbum Kim, Doo-Hyun Kim, Woopyo Jeong, Hyun-Jin Kim, Il Han Park, Hyun-Wook Park, JongHoon Lee, JiYoon Park, Yang-Lo Ahn, Ji Young Lee, et al. A 512-gb 3-b/cell 64-stacked wl 3-d-nand flash memory. *IEEE Journal of Solid-State Circuits*, 53(1):124–133, 2017.

[10] J. Jang, H. Kim, W. Cho, H. Cho, J. Kim, S. Shim, Y. Jang, J. Jeong, B. Son, D. Kim, K. Kim, J. Shim, J. Lim, K. Kim, S. Yi, J. Lim, D. Chung, H. Moon, S. Hwang, J. Lee, Y. Son, Y. Chung, and Y. Lee. Vertical cell array using TCAT (Terabit Cell Array Transistor) technology for ultra high density NAND flash memory. In *Proceedings of the IEEE Symposium on VLSI Technology (VLSI)*, 2009.

[11] R. Katsumata, M. Kito, Y. Fukuzumi, M. Kido, H. Tanaka, Y. Komori, M. Ishiduki, J. Matsunami, T. Fujiwara, Y. Nagata, L. Zhang, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama. Pipe-shaped BiCS flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices. In *Proceedings of the Symposium on VLSI Technology (VLSI)*, 2009.

[12] E. Choi and S. Park. Device considerations for high density and highly reliable 3D NAND flash cell in near future. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, 2012.

[13] Yoshiaki Fukuzumi, Ryota Katsumata, Masaru Kito, Masaru Kido, Mitsuru Sato, Hiroyasu Tanaka, Yuzo Nagata, Yasuyuki Matsuoka, Yoshihisa Iwata, Hideaki Aochi, et al. Optimal integration and characteristics of vertical array devices for ultra-high density, bit-cost scalable flash memory. In *Proceedings of IEEE International Electron Devices Meeting*, 2007.

[14] Yixin Luo, Saugata Ghose, Yu Cai, Erich F Haratsch, and Onur Mutlu. Heatwatch: Improving 3d nand flash memory device reliability by exploiting self-recovery and temperature awareness. In *Prpc. International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[15] "samsung v-nand technology, white paper". `https://studylib.net/doc/8282074/`

samsung-v-nand-technology, 2014.

[16] Xavier Jimenez, David Novo, and Paolo Ienne. Wear unleveling: Improving nand flash lifetime by balancing page endurance. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2014.

[17] Wei Debao, Qiao Liyan, Zhang Peng, and Peng Xiyuan. Bpm: A bad page management strategy for the lifetime extension of flash memory. In *Proceedings of International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2015.

[18] Yoon Jae Seong et al. Hydra: A block-mapped parallel flash memory solid-state disk architecture. *IEEE Trans. Computers*, 59(7):905–921, 2010.

[19] Abdul R Abdurrab et al. Dloop: A flash translation layer exploiting plane-level parallelism. In *Proceedings of Int'l Symp. Parallel and Distributed Processing*, pages 908–918, 2013.

[20] Wei Wang and Tao Xie. Pcftl: A plane-centric flash translation layer utilizing copy-back operations. *IEEE Trans. Parallel and Distributed Systems*, 26(12):3420–3432, 2015.

[21] Woo Tae Chang et al. An efficient copy-back operation scheme using dedicated flash memory controller in solid-state disks. *Int'l Journal of Electrical Energy*, 2(1):13–17, 2014.

[22] Richard R. Muntz and John C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of International Conference on Very Large Databases*, 1990.

[23] Mark Holland and Garth Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of Architectural Support for Programming Languages and Operating Systems*, 1992.

[24] GA Alverez, Walter A Burkhard, LL Stockmeyer, and Flaviu Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *Proceedings of International Symposium on Computer Architecture*, 1998.

[25] Siu-Cheung Chau and Ada Wai-Chee Fu. A gracefully degradable declustered raid architecture. *Cluster Computing*, 5(1):97–105, 2002.

[26] Jiguang Wan, Jibin Wang, Changsheng Xie, and Qing Yang. S2-raid: Parallel raid architecture for fast data recovery. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1638–1647, 2013.

[27] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. Raid+: deterministic and balanced data distribution for large disk enclosures. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2018.

[28] Seagate Technology. Reducing raid recovery downtime. https://www.seagate.com/files/staticfiles/docs/pdf/whitepaper/tp620-1-1110us-reducing-raid-recovery.pdf, 2011.

[29] Scott Shadley. SSD RAIN. https://www.micron.com/~/media/documents/products/technical-marketing-brief/brief_ssd_rain.pdf".

[30] Yangsup Lee, Sanghyuk Jung, and Yong Ho Song. Fra: a flash-aware redundancy array of flash storage devices. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*, 2009.

[31] Soojun Im and Dongkun Shin. Flash-aware raid techniques for dependable and high-performance flash memory ssd. *IEEE Transactions on Computers*, 60(1):80–92, 2011.

[32] Sehwan Lee, Bitna Lee, Kern Koh, and Hyokyung Bahn. A lifespan-aware reliability scheme for raid-based flash storage. In *Proceedings of ACM Symposium on Applied Computing*, 2011.

[33] Yi Qin, Dan Feng, Jingning Liu, Wei Tong, Yang Hu, and Zhiming Zhu. A parity scheme to enhance reliability for ssds. In *Proceedings of International Conference on Networking, Architecture, and Storage*, 2012.

[34] Heejin Park, Jaeho Kim, Jongmoo Choi, Donghee Lee, and Sam H Noh. Incremental redundancy to reduce data retention errors in flash-based ssds. In *Proceedings of International Conferece on Massive Storage Systems and Technology*, 2015.

[35] Jaeho Kim, Eunjae Lee, Jongmoo Choi, Donghee Lee, and Sam H Noh. Chip-level raid with flexible stripe size and parity placement for enhanced ssd reliability. *IEEE Transactions on Computers*, 65(4):1116–1130, 2016.

[36] Bryan S Kim, Jongmoo Choi, and Sang Lyul Min. Design tradeoffs for ssd reliability. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2019.

[37] E. Choi and S Park. Device considerations for high density and highly reliable 3d nand flash cell in near future. In *Proceedings of IEEE International Electron Devices Meeting (IEDM)*, 2012.

[38] J. Jang, H. Kim, W. Cho, H. Cho, J. Kim, S. Shim, J. Jeong, B. Son, D. Kim, J. Shim, J. Lim, K. Kim, Y. Su, J. Lim, D. Chung, H. Moon, S. Hwang, J. Lee, Y. Son, U. Chung, and W. Lee. Vertical cell array using tcat (terabit cell array transistor) technology for ultra high density nand flash memory. In *Proceedings of IEEE Symposium on VLSI Technology (VLSI)*, 2009.

[39] Jaeyong Jeong, Youngsun Song, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. Dynamic erase voltage and time scaling for

extending lifetime of nand flash-based ssds. *IEEE Transactions on Computers*, 66(4):616–630, 2016.

[40] Rino Micheloni, Luca Crippa, and Alessia Marelli. *Inside NAND flash memories*. Springer Science & Business Media, 2010.

[41] Benny Van Houdt. On the necessity of hot and cold data identification to reduce the write amplification in flash-based ssds. *Performance Evaluation*, 82:1–14, 2014.

[42] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. Mqsim: A framework for enabling realistic studies of modern multi-queue {SSD} devices. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2018.

[43] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: practical power management for enterprise storage. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2008.

[44] Sysbench. http://github.com/akopytov/sysbench.

[45] Filebench. http://filebench.sourceforge.net.

[46] Yangyang Pan et al. Error rate-based wear-leveling for nand flash memory at highly scaled technology nodes. *IEEE Trans. on Very Large Scale Integration Systems*, 21(7):1350–1354, 2013.

[47] Dongku Kang et al. 256gb 3b/cell v-nand flash memory with 48 stacked wl layers. In *Proceedings of Int'l Solid-State Circuits Conf.*, 2016.

[48] Sang-Woo Jun et al. Bluedbm: An appliance for big data analytics. In *Proceedings of Int'l Symp. Computer Architecture*, pages 1–13, 2015.

[49] Jimmy Yang and Feng-Bin Sun. A comprehensive review of hard-disk drive reliability. In *Proceedings of Annual Reliability and Maintainability Symposium*, 1999.

[50] Yu Cai, Erich F Haratsch, Onur Mutlu, and Ken Mai. Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[51] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash reliability in production: The expected and the unexpected. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2016.

[52] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, 105(9):1666–1704, 2017.

[53] Myungsuk Kim, Youngsun Song, Myoungsoo Jung, and Jihong Kim. Saro: a state-aware reliability optimization technique for

high density nand flash memory. In *Proceedings of Great Lakes Symposium on VLSI*, 2018.

[54] Micron. TN-29-59: Bad block management. https://www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn2959_bbm_in_nand_flash.pdf, 2011.

[55] Jacob Alter, Ji Xue, Alma Dimnaku, and Evgenia Smirni. Ssd failures in the field: symptoms, causes, and prediction models. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[56] Over-provisioning. https://www.seagate.com/tech-insights/ssd-over-provisioning-benefits-master-ti/.

[57] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185, 1994.

[58] Serial at attachment. https://sata-io.org/.

[59] Nvm express. https://nvmexpress.org/resources/specifications/.

[60] Scsi storage interfaces. http://www.t10.org.

[61] Mai Zheng, Joseph Tucek, Feng Qin, and Mark Lillibridge. Understanding the robustness of ssds under power fault. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2013.

[62] Ying Y Tai. High performance ftl for pcie/nvme ssds. In *Proceedings of Flash Memory Summit*, 2016.

[63] Shunzhuo Wang, Fei Wu, Chengmo Yang, Jiaona Zhou, Changsheng Xie, and Jiguang Wan. Was: Wear aware superblock management for prolonging ssd lifetime. In *Proceedings of Design Automation Conference*, 2019.

[64] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The multi-streamed solid-state drive. In *Proceedings of Workshop on Hot Topics in Storage and File Systems*, 2014.

[65] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. Fully automatic stream management for multi-streamed ssds using program contexts. In *Proceedings of USENIX Conference on File and Storage Technologies*, 2019.

[66] Ulink. DriveMaster. `https://ulinktech.com/products/drivemaster-8-enterprise-sas/"`.

[67] Jens Axboe. Fio. https://github.com/axboe/fio.

[68] Iometer. http://www.iometer.org/.

[69] Eden Kim. "enterprise applications: How to create a synthetic workload test". https://www.snia.org/sites/default/files/EdenKim_Enterprise_Applications_WorkLoad_Test_SDC_2014.pdf, 2014.

[70] Youngjae Kim, Sarp Oral, Galen M Shipman, Junghee Lee, David A Dillow, and Feiyi Wang. Harmonia: A globally coordinated garbage collector for arrays of solid-state drives. In *Proceedings of Symposium on Mass Storage Systems and Technologies*, 2011.

[71] Ulrich Hansen. "the ssd endurance race: Who's got the write stuff?". https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_TC11_Hansen.pdf, 2012.

초 록

반도체 공정의 미세화, 다치화 기술에 의해서 지속적으로 용량이 증가하고 있는 단위 낸드 플래쉬 메모리와 하나의 낸드 플래쉬 기반 스토리지 시스템 내에 수 많은 낸드 플래쉬 메모리 다이를 실장할 수 있게하는 낸드 패키지 기술로 인해 하드디스크보다 훨씬 더 큰 초고용량의 낸드 플래쉬 저장장치의 개발을 가능하게 했다. 플래쉬 저장장치의 용량이 증가할 수록 스토리지 시스템의 총 소유비용을 줄이는데 매우 효과적인 장점을 가지고 있으나, 신뢰성 및 성능의 측면에서의 한계로 인해서 초고용량 낸드 플래쉬 저장장치가 널리 사용되는데 있어서 장애물로 작용하고 있다. 초고용량 저장장치의 장점을 활용하기 위해서는 이러한 신뢰성 및 성능을 개선하기 위한 새로운 기법의 개발이 필요하다.

본 논문에서는 초고용량 낸드기반 저장장치(SSD)의 문제점인 성능 및 신뢰성을 개선하기 위한 다양한 최적화 기술을 제안한다. 기존 기법들의 최적화 한계를 극복하기 위해서, 우리의 기술은 실제 낸드 플래쉬 소자에 대한 다양한 특성 평가 결과와 SSD의 현장 불량 특성 분석결과를 기반으로 고안되었다. 이를 통해서 낸드의 플래쉬 특성과 SSD, 그리고 호스트 시스템의 동작 특성을 고려한 성능 및 신뢰성을 향상시키는 최적화 방법론을 제시한다.

첫째로, 본 논문에서는 낸드 플래쉬 불록내의 페이지들간의 특성편차를 줄이기 위해서 동적인 소거 스트레스 경감 기법을 제안한다. 제안된 기법은 낸드 블록의 내구성을 늘리기 위해서 특성이 약한 페이지들에 대해서 더 적은 소거 스트레스가 인가할 수 있도록 낸드 평가 결과로 부터 소거 스트레스 경감 모델을 구축한다. 또한 사용자 워크로드 특성을 고려하여, 소거 스트레스 경감 기법의 효과가 최대화 될 수 있는 최적의 경감 수준을 동적으로 판단할 수 있도록 한다. 이를 통해서 낸드 블록을 열화시키는 주요 원인인 소거 동작을 효율적으로 제어함으로써 저장장치의 수명을 효과적으로 향상시킨다.

둘째로, 본 논문에서는 고용량 SSD에서의 내부 데이터 이동으로 인한 성능 저하문제를 개선하기 위해서 낸드 플래쉬의 제한된 카피백(copyback) 명령을 활용하는 적응형 기법인 rCPB을 제안한다. rCPB은 Copyback 명령의 효율성

을 극대화 하면서도 데이터 신뢰성에 문제가 없도록 낸드의 블럭의 노화특성을 반영한 새로운 copyback 오류 전파 모델을 기반으로한다. 이에더해, 신뢰성이 다른 블럭간의 copyback 명령을 활용한 데이터 이동을 문제없이 관리하기 위해서 자원 효율적인 오류 관리 체계를 제안한다. 이를 통해서 신뢰성에 문제를 주지 않는 수준에서 copyback을 최대한 활용하여 내부 데이터 이동을 최적화 함으로써 SSD의 성능향상을 달성할 수 있다.

마지막으로, 본 논문에서는 초고용량 SSD에서 낸드 플래쉬의 다이(die) 불량으로 인한 레이드(redundant array of independent disks, RAID) 리빌드 오버헤드를 최소화 하기위한 새로운 RAID 복구 기법인 reparo를 제안한다. Reparo는 SSD에 대한 교체없이 SSD의 불량 die에 대해서만 복구를 수행함으로써 복구 오버헤드를 최소화한다. 불량이 발생한 die의 데이터만 선별적으로 복구함으로써 복구 과정의 리빌드 트래픽을 최소화하며, SSD 내부의 병렬구조를 활용하여 불량 die 복구 시간을 효과적으로 단축한다. 또한 die 불량으로 인한 물리적 공간감소의 부작용을 최소화 함으로써 복구 이후의 성능 저하 및 수명의 감소 문제가 없도록 한다.

본 논문에서 제안한 기법들은 저장장치 프로토타입 및 공개 낸드 플래쉬 저장장치 개발환경, 그리고 실장 SSD환경에 구현되었으며, 실제 응용 프로그램을 모사한 다양한 벤트마크 및 실제 I/O 트레이스들을 이용하여 그 유용성을 검증하였다. 실험 결과, 제안된 기법들을 통해서 초고용량 SSD의 신뢰성 및 성능을 효과적으로 개선할 수 있음을 확인하였다.

키워드: 낸드 플래시 메모리, 플래시 변환 계층, 낸드 플래시 기반 저장장치, 내장형 시스템, 성능 최적화, 수명 최적화

학번: 2017-36900

감사의 글

십여 년간 회사생활을 하는 과정에 학술연수라는 기회를 얻어 뒤는게 박사과정을 시작한지 벌써 사년 반이 지났습니다. 박사과정을 시작하면서 늦은 나이에 얻은 기회라서 잘 해낼 수 있을지 걱정이 많이 되었었는데, 많은 분들의 도움과 격려 그리고 기도로 박사학위 과정을 중도에 포기하지 않고 성공적으로 끝낼 수 있게 되었습니다. 이 지면을 빌어서 그간의 감사의 인사를 드리고자 합니다.

먼저 지도 교수님이신 김지홍 교수님께 진심으로 감사의 말씀을 드립니다. 교수님의 지도 하에 박사과정을 할 수 있었던 것은 제 인생에 큰 행운 이었습니다. 아직도 너무나 부족하지만, 교수님의 세심한 배려와 지도 덕분에 연구자의 삶에 있어 필수적인 많은 것들을 배울 수 있었습니다. 큰 가르침에 조 금이라도 답할 수 있도록 앞으로도 최선을 다해 노력하겠습니다. 더불어 부족한 저의 논문을 심사해주신 유승주 교수님, 김진수 교수님, 이재욱 교수님, 그리고 이성진 교수님께도 감사의 말을 드리고 싶습니다. 바쁘신 와중에도 귀중한 시간을 내어 논문 자격 심사부터 최종 학위 심사에 이르기까 지 아낌없이 조언해주시고 지도해 주신 것이 저의 학위 논문 완성에 큰 힘이 되었습니다.

제가 학업을 시작하고 이어갈 수 있도록 도와 주신 이동기 전무님, 이종열 부사장님, 지도임원으로 조언해 주신 조상연 전무님, 오문욱 상무님, 장실완 상무님께도 감사드립니다. 박사 학위 진학이라는 좋은 기회를 주시고 다른 어려움없이 학업에 전념할 수 있도록 전폭적으로 지원해주셔서 마음 깊이 감사드립니다. 제가 보고 배운 것들을 바탕으로 복귀하여 회사에 도움이 될 수 있도록 노력하 겠습니다. 또한 학술연수 기간동안 논문작성을 위한 실험환경에 도움을 주시고 응원해주신 많은 SSD 5반/6반 동료들에게도 감사드립니다. 특별히 박사 진학을 추천하고 졸업하지 못할까봐 걱정해 주셨던 정다운 상무님과 하건수 박사에게도 감사드립니다.

짧지않은 기간동안 학교생활에 잘 적응하고 무사히 학업을 마칠 수 있었던 것은 연구실 동료들의 도움이 컸습니다. 김태진 박사, 한상욱 박사, 박지성 박사, 그리고 같이 학술 연수를 하였던 김명석 교수님, 송영선 수석, 심영섭 수석께 많은 도움을 받았습니다. 또한 함께 연구실 생활을 하면서 도움을 준 재훈, 정석, 승욱이와 연구실 입학 동기인 명준, 유현, 그리고 연구실 후배들인 용석, 재민, 일보, 슬기, 윤아, 건희, 두솔, 재용, 보경, 준석, 상구, 인혁, 나그맛에게도 고맙다는 말을 전하고 싶습니다. 연구실 선배로서 많은 도움이 되지 못하고 떠나는 것 같아서 미안한 마음이 있습니다. 여러분들의 앞길에 항상 좋은 일들이 있기를 응원하겠습니다.

제가 학업을 끝날때 까지 따뜻한 관심과 기대를 가지고 응원해 주신 사랑하는 가족들과 지인들께도 감사드립니다. 변함없는 사랑으로 항상 아들을 지지해주신 어머니와 할머니께 감사드립니다. 또한 막내 사위를 사랑해주시고 응원해주신 장모님께도 감사드립니다. 오빠 대신 고향의 가족들을 잘 챙기는 동생 문정이와 매부에게도 감사의 마음을 전합니다. 첫째, 둘째, 셋째 처형들과 형님들의 사랑과 응원에 또한 감사드립니다.

마지막으로 매일 매일 변함없는 지지와 기도로 박사 과정을 잘 끝낼 수 있도록 내조하며 함께해준 사랑스러운 아내 희경과 착하고 예쁜 세딸 지민, 지유, 지아에게도 사랑을 듬뿍담아 감사의 마음을 전합니다. 감사합니다.

2021년 8월

홍 두 원