WORKLOAD DRIVEN DESIGNS FOR COST-EFFECTIVE NON-VOLATILE MEMORY HIERARCHIES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Timothy A. Pritchett

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Mithuna S. Thottethodi, Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Cheng-Kok Koh

School of Electrical and Computer Engineering

Dr. T.N. Vijaykumar

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of the School of Electrical and Computer Engineering

ACKNOWLEDGMENTS

I want to start with my sincere gratitude to my advisor, Prof. Mithuna S. Thottethodi, for his mentorship and unyielding support throughout my graduate school journey. His enthusiasm, knowledge, and curiosity made tackling new challenges enjoyable and his deft counsel on an array of topics, including storytelling, has greatly enhanced both my professional abilities and life in general. Furthermore, my Ph.D. and this thesis would likely have not come to fruition had it not been for his support and heartfelt guidance during the frustrating, stressful, and challenging times that accompany the pursuit of new insights.

Next, I want to thank the other members of my Ph.D. committee, Prof. Anand Raghunathan, Prof. Cheng-Kok Koh, and Prof. T.N. Vijaykumar, for their time, advice, and thought-provoking questions over the course of my graduate career.

I also want to thank Dr. Mark Johnson for his support and mentorship during my graduate teaching career.

And lastly, I want to thank my parents, Kimberley and Daniel Dillman and Thomas and Yili Pritchett, for their love, support, and teaching me to persistently pursue my goals to their full completion.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

ABSTRACT xi

1 INTRODUCTION 1

2 WRITEGUARD 6

2.1 Introduction 7

2.2 Background 8

2.2.1 Basic Architecture 8

2.2.2 Overheads of replacement policies 9

2.2.3 Alternative approaches 11

2.3 SSD Cache Write-Stream Characteristics 14

2.3.1 Computational overheads of Recency-based and Frequency-based Write Buffers 18

2.4 WriteGuard 19

2.4.1 WriteGuard Base Design 21

2.4.2 WriteGuard Sieving Filter Self-Tuning 22

2.4.3 WriteGuard Bypass-Ratio Target Self-Tuning 25

2.5 Methodology 26

2.5.1 Initial Baseline SSD Cache 26

2.5.2 Trace Analysis 27

2.5.3 Revised Experiment Baseline 27

2.5.4 Comparison with PID Controller based Threshold tuning 28

2.5.5 Design Simulations 28

2.6 Results 30

Page

2.6.1 WriteGuard Tuning 30
2.6.2 Write Stream Dilution Results 35
2.6.3 Buffer Design Run-time Cost Results 37
2.7 Related Work 39
2.8 Conclusion 41
3 OPTIMIZED STT-MRAM FOR WRITE BUFFERS 43
3.1 Introduction 44
3.2 STT-MRAM Technology Background 47
3.2.1 Adjusting STT-MRAM Cell Retention Time 48
3.2.2 Adjusting STT-MRAM Magnetic Tunnel Junction 49
3.3 Opportunity for Write Focused STT-MRAM Design 50
3.3.1 SieveStore-C Internal Access-Stream Trends 51
3.3.2 SieveStore+ Internal Screening Cache and Downstream Trends . 53
3.3.3 Write-Buffer Access-Stream Trends 58
3.4 STT-MRAM Reduced Retention-Time Trade-off 64
3.4.1 Benefits of Reduced Retention Time 64
3.4.2 Costs of Reduced Retention Time 65
3.5 Methodology 69
3.5.1 Reference Access Traces 69
3.5.2 WriteGuard Instance for Write-Buffer Internal Access-Stream . 72
3.5.3 Trace Analysis 73
3.6 Results 74
3.6.1 SieveStore-C Internal and Post-Screening Cache Data Lifespan Trends 74
3.6.2 Write-Buffer Internal Data Lifespan Trends 76
3.6.3 Impact on SSD Lifespan Improvements from Data Expiration . 81
3.6.4 STT-MRAM Cell Retention Time Trade-off Costs 84
3.7 Related Work 92

Page

3.8 Conclusions 94
4 CONCLUSION 95
REFERENCES 98

LIST OF TABLES

Table Page

2.1 Example SSD Cache Provisioning Costs based on SieveStore-C [3] 13

2.2 Summary of Reference Ensemble Trace Set 15

2.3 SieveStore-C SSD Cache Parameters 27

2.4 Summary of Key Attributes for Servers used for Design Runtime Profiling 29

3.1 SieveStore-C SSD Cache Parameters 69

3.2 Summary of Reference Ensemble Trace Set 70

3.3 WriteGuard Write-Buffer Reference Instance Parameters 73

LIST OF FIGURES

Figure Page

1.1 Reference SSD Cache System Organization 2

2.1 Reference SSD Cache System Organization (As presented in Chapter 1) . 9

2.2 Comparison of the Write-Stream Dilution Strength of Least Recently Access-based Caches and Buffers 16

2.3 Comparison of the SSD Write-Stream Dilution Strength of Least Recently/Frequently Access-based Caches and Buffers for SieveStore+ 17

2.4 Comparison of the decision making overhead profiles of LR, LF, and LF-I based Buffers 18

2.5 Visual comparison of selection by popularity sort versus selection by popularity threshold 20

2.6 WriteGuard (Static Ratio Target) Bypass Ratios Sensitivity Comparison . 31

2.7 WriteGuard (Auto-Tuned) History Factor Sensitivity Comparison 32

2.8 WriteGuard (Auto-Tuned) Evaluation Rate Sensitivity Comparison 33

2.9 WriteGuard Auto-Tuning vs. Static Bypass Ratios Comparison 34

2.10 LRU Screened Intra-Cache Write Stream Dilution Comparison 35

2.11 Underlying SSD Lifespan Improvements from Write Buffer designs 36

2.12 Comparison of the decision making run-time profiles of Write Buffer Algorithms for the SieveStore+ write-stream 38

2.13 Comparison of the allocations made by Write Buffer Algorithms for the SieveStore+ write-stream 39

3.1 Simplified Diagram of a STT-MRAM cell 47

3.2 Relative magnetic alignment for distinct STT-MRAM bit-states 48

3.3 SieveStore-C internal access-stream write-to-read ratio profile 51

3.4 SieveStore-C internal access-stream write-to-read ratio profile (ratios < 5x) 53

3.5 Write-buffer input access-stream write-to-read ratio profile 55

Figure Page

3.6 Write-buffer input access-stream write-to-read ratio profile (ratios < 5x) . 56

3.7 Comparison of pre- and post- screening cache access-streams (ratios < 5x) 56

3.8 Write-buffer internal block storage access-stream write-to-read ratio profile 59

3.9 Write-buffer internal block storage access-stream write-to-read ratio profile (zoomed to Ratios less than 5x) 60

3.10 Write-buffer internal block storage access-stream write-to-read ratio profile without eviction triggered reads 62

3.11 Write-buffer internal block storage access-stream write-to-read ratio profile without eviction triggered reads (zoomed to Ratios less than 5x) 63

3.12 Timing Diagram Illustrating Timing for Ensuring Reserve Durability . . . 66

3.13 Timing Diagram Illustrating Timing for Selective Refresh Related Activity 67

3.14 Timing Diagram Illustrating Timing for Expiration based Eviction Activity 68

3.15 Reference SSD Cache System Organization (As presented in Chapter 1) . 71

3.16 Natural Data Lifespans for logical pages within SieveStore-C and downstream of the internal screening cache 76

3.17 Natural Data Lifespans for logical and physical pages within the write buffer 78

3.18 Comparison of Natural Data Lifespans for physical pages within the write buffers of different sizes 80

3.19 Comparison of SSD Lifespan Improvements for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes 83

3.20 Comparison of extra SSD writes for write buffers using reduced STT-MRAM cell retention time with expiration eviction management scheme 85

3.21 Comparison of extra STT-MRAM data access overheads for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes (Overheads $\leq$ 400%) 87

3.22 Comparison of extra STT-MRAM data access overheads for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes (Zoomed to Overheads $\leq$ 100%) 88

Figure Page

3.23 Comparison of impact of reserving extra page lifespan by early refreshing of pages (Limited to Overheads $\leq 400\%$) 90

3.24 Comparison of impact of reserving extra page lifespan by early refreshing of pages (Zoomed to Overheads $\leq 100\%$) 91

ABSTRACT

Pritchett, Timothy A. Ph.D., Purdue University, August 2020. Workload Driven Designs for Cost-Effective Non-volatile Memory Hierarchies. Major Professor: Mithuna S. Thottethodi.

Compared to traditional hard-disk drives (HDDs), non-volatile (NV) memory technologies offer significant performance advantages on one hand, but also incur significant cost and asymmetric write-performance on the other. A common strategy to manage such cost- and performance-differentials is to use hierarchies such that a small, but intensely accessed, working set is staged in the NV storage (selective caching). However, when this working set includes write-heavy data, the low write-lifetime of NV storage necessitates significant over-provisioning to maintain required lifespans (e.g., storage lifespan must match or exceed 3 year server lifespan). One may think that employing DRAM-based write-buffers can filter writes that trickle through to the NV storage and thus alleviate the write-pressure felt at the NV storage. Unfortunately, selective caches, when used with common recency-based or frequency-based replacement, have access patterns that require large write buffers (e.g., 100MB+ relative to a 12GB cache) to filter writes adequately. Further, these large DRAM write-buffers also require backup-power to ensure the durability of disk writes. More sophisticated replacement policies that combine recency and frequency can reduce the size of the DRAM buffer (while preserving write-filtering), but are so computationally-expensive that they can limit the I/O rate, especially for simple controllers (e.g., RAID controller).

My first contribution is the design and implementation of WriteGuard– a self-tuning sieving write-buffer algorithm that filters writes as well as the highly-effective (but computationally-expensive) algorithms while requiring lightweight computation

comparable to a simple LRU-based write-buffer. While WriteGuard reduces the capacity needed for DRAM buffering (to approx. 64 MB), it does not eliminate the need for DRAM buffers (and corresponding power backup).

For my second thrust, I identify two specific application characteristics – (1) the vast majority of the write-buffer's contents is composed of write-dominant blocks, and (2) the vast majority of blocks in the write-buffer are overwritten within a period of 28 hours. I show that these characteristics help enable a high-density, optimized STT-MRAM as a replacement for DRAM, which enables durable write-buffers (thus eliminating the cost of power backup for the write-buffer). My optimized STT-MRAM-based write buffer achieves higher density by (a) trading off superfluous durability by exploiting characteristic (2), and (b) deoptimizing the read-performance of STT-MRAM by leveraging characteristic (1). Together, the techniques increase the density of STT-MRAM by 20% with low or no impact on write-buffer performance.

1. INTRODUCTION

The nature of trade-offs in storage technology continues to favor a hybrid approach as non-volatile solid-state technologies offer significantly higher bandwidth (but at significantly higher cost per bit) whereas traditional hard-disk drive (HDD) storage offers high capacity at a lower cost-per-bit. This trade-off leads to a very specific design problem in storage-tier caching: accelerated drive wear-out from the aggregation of write-intense data. Because any successful storage-tier cache will aim to hold the hot data, they are typically designed with high-performance non-volatile storage [1–3]. For example, SieveStore [3] uses a small non-volatile cache to capture a narrow (but highly popular) set of disk blocks from a large collection of backend HDD drives. Such hybrid systems minimize cost (like HDD-based storage) while achieving high performance (like NV storage).

Unfortunately, the presence of highly popular, but also frequently written, disk blocks complicates this approach. Frequently written blocks pose a dilemma because their frequent writes can affect the lifetime [1] of non-volatile storage (especially in modern MLC SSD caches), but not caching them in the NV storage causes performance degradation (given how frequently they are accessed).

I use the system proposed in [3] – SieveStore-C– as the baseline system that offers a cost-effective SSD cache. Figure 1.1 illustrates the organization as proposed in [3], updated to illustrate RAM-based caches and write-buffers that are common internal components of storage caches. The system includes a collection of servers, each with its own local filesystem and underlying buffer cache (in memory). Each such server also has a local hard disk storage (HDD in Figure 1.1). The SSD cache is a common

[1]Some argue that the lifetime problem does not matter for SSD caches. We discuss that seeming contradiction later in Chapter 2 and conclusively show that the reduction in lifetimes is a real concern.

structure that is shared across all servers. SieveStore-C uses the shared cache to hold a small (but frequently accessed) set of data.

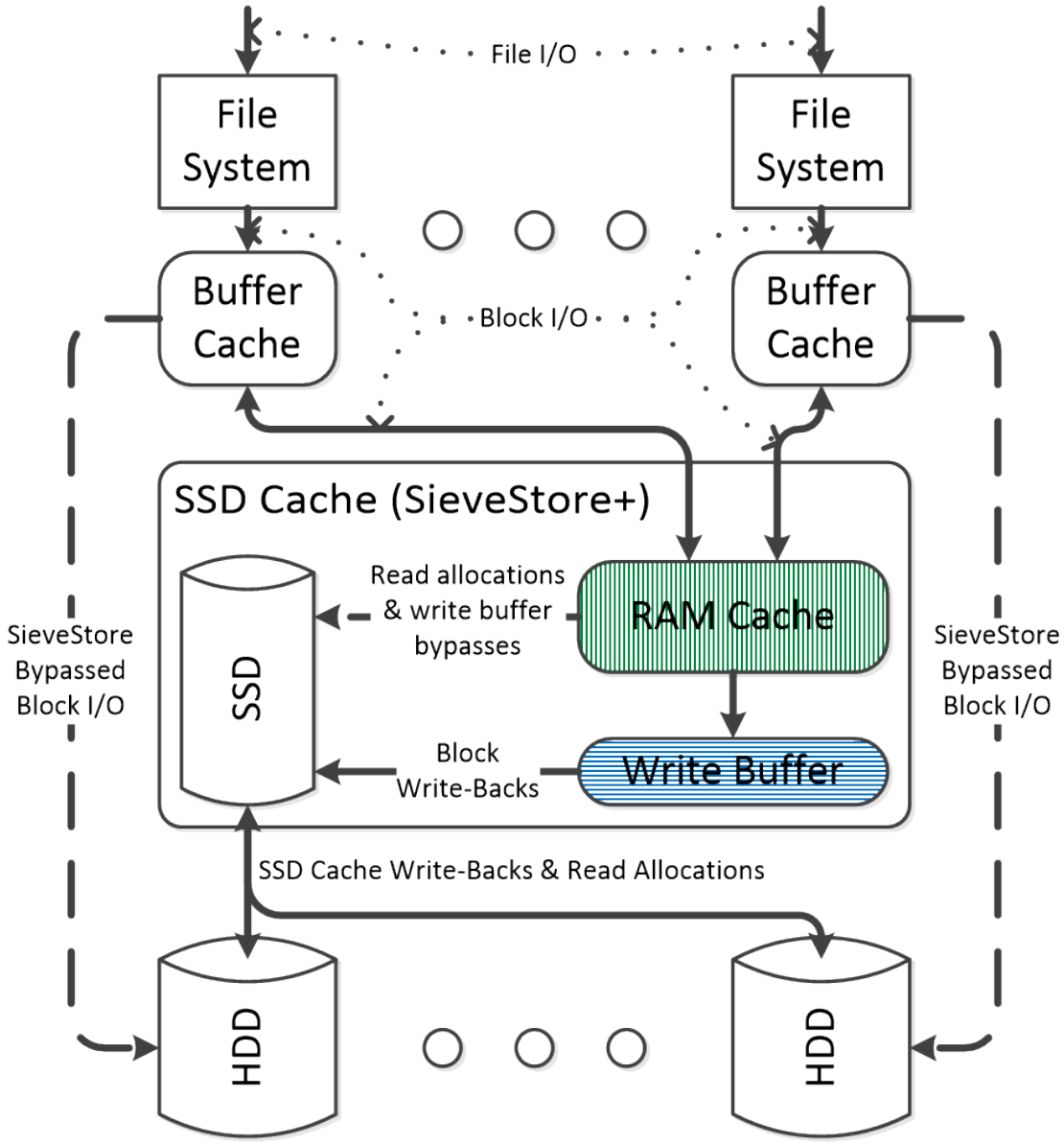


Fig. 1.1.: Reference SSD Cache System Organization

Prior techniques that allow the caching of writes and specifically write-intense data, like SieveStore-C, have largely depended upon the higher write-endurance ratings of SLC Flash to maintain SSD-Cache lifetimes on-par with typical server lifetimes (i.e., 3-5 years). However, SSD production and Flash production in general

have nearly exclusively switched to utilizing the significantly less write-tolerant MLC Flash, due their higher bit-densities, largely removing SLC Flash based SSDs from consideration in storage caches due to further increased cost premiums and limited availability. Due to the largely diminished write-endurance ratings of MLC Flash (e.g., 92× lower), sustainable caching of write-intense data would require a mix of higher-level write-buffers (typically RAM-based) to filter the writes that reach the SSD-cache and over-provisioning of the SSD-cache capacity to leverage wear-leveling approaches for extending drive lifetimes [4–6]. The hope is that the write-filtering efforts will significantly lessen the over-provisioning costs needed to restore the lifetime of SSD caches beyond that of typical server lifetime (i.e., 3-5 years). Unfortunately, RAM-based write buffers do not represent a storage-equivalent solution as RAM is volatile. Buffering file system writes in volatile memory can result in data loss in case of crash/failure of the computer in question. To achieve equivalent durability as persistent storage, the RAM-buffers must be protected by backup power sources (to enable flushing to persistent storage on a crash) with power requirements increasing with increasing buffer size.

The goal of my work is to ameliorate and/or eliminate such additional costs of write-buffering while still filtering enough writes to minimize over-provisioning costs needed to extend SSD-cache lifetimes beyond three years.

My solution has two components. The first component – WriteGuard– is a self-tuned write buffer that maximizes the write filtering for a given write-buffer capacity, (or alternately, reduces the amount of buffering needed to achieve a given level of write-filtering). In Chapter 2 I show that, while simple replacement algorithms such as *least recently used (LRU)* and *least frequently used (LFU)* can be implemented with low computation complexity, they significantly under-filter writes. On the other hand, sophisticated techniques that combine recency, frequency, and aging/inflation mechanisms (e.g., LRFU-with-inflation [7]) achieve significantly higher write filtering. However, the computational cost of these sophisticated algorithms can be so high that it becomes an IO bottleneck.

My work diagnoses the key source of complexity in the sophisticated replacement algorithms – unlike LRU and LFU which perform a constant amount of work on each access to maintain the replacement stack, the LRFU-with-inflation approach effectively has to perform as many as $log(n)$ operations to maintain the replacement stack in sorted order. In general, a replacement stack enforces the invariant that a more "valuable" block is not replaced by a less valuable block. An alternate "sieving approach" proposed in [3] uses a thresholding to prevent allocations to less popular blocks, achieving results similar to a frequency-based sort, but it requires offline trace analysis for the selection of this threshold. Based on this insight, I propose a design WriteGuard that avoids maintaining a fully sorted stack. Instead, WriteGuard eliminates the need for a sorted replacement stack with a dynamically-tuned thresholding mechanism that requires only a constant (amortized) amount of work for a given access. This approach reduces the amount of buffering needed by a factor of up-to 2x relative to recency-based and frequency-based write buffers, while requiring similar or less computational effort. I expand on this component in Chapter 2.

The second component explores an STT-MRAM-based approach for the write buffer. While STT-MRAM's non-volatility enables naturally durable write-buffer without the need for power-backup, it's higher cost relative to DRAM has limited adoption. However, factoring the full picture of durability and elimination of periodic refresh power and performance penalties alongside the lower density of current STT-MRAM has resulted in early but limited adoption in some high-performance enterprise SSD designs [8]. In Chapter 3 I show that, for storage workloads, write buffers exhibit internal access patterns that enable STT-MRAM design scaling beyond the limitations imposed in traditional workloads where the criticality of read-performance largely determines cell design. My work then illustrates the write-dominance of accesses within write-buffers enables the trading of superfluous cell retention time for increased cell density, resulting in a STT-MRAM cell density increase of 25% with low or no impact on write buffer performance. I describe this study and resulting design approach further in Chapter 3.

The remainder of this thesis is organized as follows. Chapter 2 describes the design and evaluation of my self-tuned write-buffer design – WriteGuard. Chapter 3 describes the workload characteristics of writes in storage traces and illustrates how those characteristics may be leverage to achieve a high-density (i.e., lower cost) STT-MRAM-based write-buffer design. Chapter 4 concludes this dissertation.

2. WRITEGUARD

Flash-based solid-state drives (SSDs) remain an order of magnitude more expensive per bit than traditional hard-disk drives (HDDs). Consequently, while SSDs indeed dominate the performance segment, they have not been considered an attractive proposition for bulk storage capacity where HDDs dominate. As such, previous proposals have argued that SSDs should be used as staging caches in front of a bulk-storage HDD array. In such designs, the SSDs typically handle the high-activity blocks and bypass the low-intensity block accesses to the HDD tier. While sieving caches lessen the cost of using SSDs to accelerate storage performance, the high write-intensity of the blocks held in these selective caches still necessitates the use of high-wear resistant and expensive SSDs, massive over-provisioning of SSD capacity for increased wear-leveling, or relatively large RAM based buffers that require additional system power backup.

I show that the driving factor behind the large size of RAM-based write-buffers is that the write-stream for selective caches has characteristics that lead to poor performance with both strictly recency-based and frequency-based buffer algorithms, and thus require algorithms from the more computationally expensive joint recency-frequency-based designs. I then diagnose the key sources of the added computational expense of the more effective recency-frequency-based designs - the logarithmic cost of resorting replacement stack on each access. And I propose a dynamically tuned sieving write-buffer algorithm – WriteGuard– which eliminates the need for costly meta-data resorting required by joint recency-frequency-based replacement policies. This design approach reduces the amount of buffering needed by a factor of up-to 2x relative to recency-based and frequency-based write buffers, while requiring similar or less computation effort.

2.1 Introduction

Consider the conventional approaches to using RAM-based write filtering. One could use a RAM-based write-buffer that is managed using traditional LRU or LFU replacement. (For write buffers, I specifically also consider *least recently written (LRW)* and *least frequently written (LFW)* replacement policies.) However, unlike traditional write buffers, the write buffers of SSD caches must handle a high skewed popularity distribution, which LRU/LRW, and LFU/LFW fail at. Specifically, LRU/LRW-replacement with skewed popularity leads to inadequate filtering of writes because of a large number of spills to SSD. In contrast, LFU/LFW prioritizes old hot blocks over newly trending blocks which results in unnecessary propagation of new writes to SSDs. I observed that workloads which have dynamic popularity (where the popular pages change over time) are particularly susceptible to this problem. Finally, there are variants of LFU/LFW that use 'inflation' (LFU-I/LFW-I) which manage to balance both recency and frequency. (Effectively, the frequency of a page becomes less relevant with age.) While effective in handling dynamic popularity, these approaches are computationally expensive, especially for the modest processing available in an SSD-based cache.

In this paper, I resolve the above dilemma by designing WriteGuard that combines recency and frequency to manage a RAM-based write-buffer efficiently. My work diagnoses the key source of complexity in the sophisticated replacement algorithms – unlike LRU and LFU which perform a constant amount of work on each access to maintain the replacement stack, the LRFU-with-inflation approach effectively has to perform as many as $log(n)$ operations to maintain the replacement stack in sorted order. In general, a replacement stack enforces the invariant that a more "valuable" block is not replaced by a less valuable block. An alternate "sieving approach" proposed in [3] uses a thresholding to prevent allocations to less popular blocks, achieving results similar to a frequency-based sort, but it requires offline trace analysis for the selection of this threshold. Based on this insight, I propose a design

WriteGuard that avoids maintaining a fully sorted stack. Instead, WriteGuard eliminates the need for a sorted replacement stack with a dynamically-tuned thresholding mechanism that requires only a constant (amortized) amount of work for a given access. This approach reduces the amount of buffering needed by a factor of up-to 2x relative to recency-based and frequency-based write buffers, while requiring similar or less computational effort.

Section 2.2 provides a brief discussion of my baseline system architecture and write buffer policy approaches. Section 2.3 describes the characteristics of the write stream within the base SSD cache system and the resulting benefits and costs of known write buffering approaches. In Section 2.4, I discuss the details of my proposed design and the key observation that results in it's effectiveness. I then discuss my evaluation methods in Section 2.5 followed by the corresponding results in Section 2.6. Section 2.7 discusses prior work related to my design. And I state my final conclusions in Section 2.8

2.2 Background

2.2.1 Basic Architecture

My basic architecture assumes backend HDD-based storage servers. The front-end servers access the backend storage servers. I assume that each front-end server has built-in RAM caching (which is typical for Unix-based systems with buffer caches). The SSD-cache is a tier between the HDD-storage backend and the front-end servers. The reference system organization (initially described in Chapter 1) is depicted in Figure 2.1 (for easier reference), with the additional RAM based cache discussed in Section 2.3 Claim 1 highlighted with a green vertical striped fill and the location of the write buffer targeted by the work in this paper highlighted with a blue horizontal striped fill.

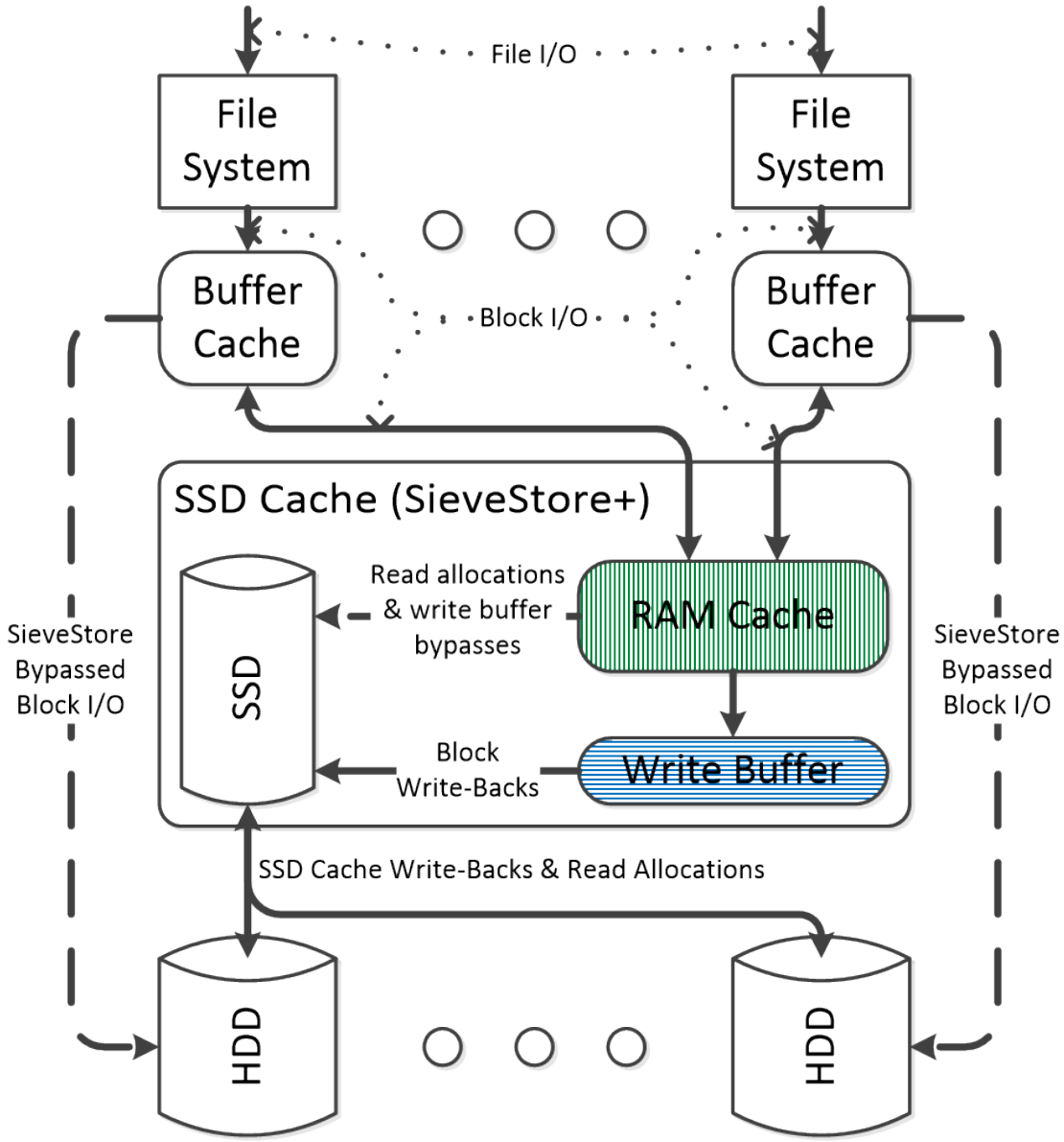


Fig. 2.1.: Reference SSD Cache System Organization (As presented in Chapter 1)

2.2.2 Overheads of replacement policies

Because cache/WB lookup and stack-maintenance occurs on every access (hit or miss), it is a common design goal to achieve fast lookup and fast stack maintenance. Consider how each of the following replacement algorithms are efficiently implemented.

LRU Caches/WBs

Software-based, fully-associative, LRU caches are typically implemented using a hash-table for fast $O(1)$ lookup. In addition, each block's metastate is also maintained in a doubly-linked list to facilitate LRU stack maintenance. Any accessed block is removed from its current position in the list and moved to the head of the list to indicate its most-recently-used (MRU) status. Because node deletion and insertion in doubly linked lists are of $O(1)$ complexity, such an implementation is efficient for LRU stack maintenance. In such an organization, the LRU block is found at the tail of the list (also in $O(1)$ time).

Unfortunately, for workloads with high popularity skew, LRU has known limitations as it prioritizes recent-but-unpopular blocks at the expense of less-recent-yet-more-popular blocks resulting in unnecessary replacements. (In the context of RAM-based write-buffers, such replacements are effectively writebacks to the lower level SSD.)

LFU Caches/WBs

Like LRU caches, LFU caches also employ hash-tables for fast lookup. The replacement stack maintenance is different; LFU caches must count the frequency of access for each block. But merely counting frequencies is inadequate; because LFU requires easy identification of the LFU block, the metastate of all blocks is maintained in an efficient priority queue (typically implemented with a heap data structure). Because the frequency-count of a block can at most increase by one upon access, this approach guarantees that a node may at most move once in the heap. Therefore the amount of maintenance work on each access is limited to $O(1)$.

Unfortunately, LFU has known issues when popularity is dynamic (i.e., when the most popular blocks change). A popular block's high access count inoculates it against replacement long after its last access which results in unnecessary replacements (and writes to lower levels).

LFU with inflation (LFU-I)

Variants of LFU address the issue of dynamic popularity by using "inflation" such as the "clock" used in [7]. The key novelty of inflation is to increment the frequency count of a block, not by 1, but by the count of the last evicted block. Effectively, this incorporates a recency bias as recent (and hence inflated) accesses contribute more to the frequency count than older (uninflated) accesses. While the replacement behavior is indeed better in that it avoids the common flaws of LRU and LFU, the stack maintenance overheads are significantly higher. Specifically, a single access can cause a large increase in the block's frequency count, resulting in more significant movement in the heap/priority-queue. Such extended data-structure manipulation can result in significantly higher compute costs as it occurs on each access.

My goal is to achieve the best of both worlds; that is to achieve the write-filtering of LFW-I while achieving fast lookup and stack maintenance (like LRU and LFU).

2.2.3 Alternative approaches

All of the above approaches use a common strategy of RAM-based buffers to filter writes to the lower-level SSD cache. In contrast, Sievestore [3] argues that the write-intensity is tolerable (i.e., storage lifetime is still at least as good as typical system lifetimes) in spite of holding write-hot blocks in SSD storage. However, the analysis is based on an estimated lifetime of 5 years for the SLC Flash-based SSD that they consider, as shown in Table 2.1. With SLC-SSDs being phased out even for enterprise-class storage, the same workloads that yielded a useful life of 5-years lead to significantly shorter lifespans for modern MLC-FLASH based SSDs, which are designed for bulk replacement where their lower write-life ratings still resulting in useful (5yr+) lifespans. Effectively, the write-intensity in selective caching is high enough to cause lifetime problems for non-volatile storage based caches using MLC-FLASH. However, recent work has shown that manufacturer warranty ratings are overly pessimistic and assume permanent cell damage from writes, while in reality

the cells naturally 'anneal' the damage overtime with per-page write-gaps of 100s to physical pages resulting 17.7x longer lifespans for SLC-Flash and 9.3x longer lifespans for MLC-Flash [9]. While these annealing effects are not strong enough to solve write-life issues (and thus over-provisioning) they do bring them down to manageable levels, which would enable effective write buffering schemes to potentially remove the need for over-provisioning with MLC-Flash based SSDs.

Table 2.1.: Example SSD Cache Provisioning Costs based on SieveStore-C [3]

Rating Class	Enterprise SLC	Enterprise MLC	Consumer MLC
Reference Drive	Intel X-25E	Samsung 983 DCT	Samsung 860 EVO
Warranty Limit (Disk Writes / GB Capacity)	31.250 TB	1.46 TB	0.600 TB
Exact-Capacity Projected Lifespan (Full Replacement)	1479.8 Years	69.14 Years	28.41 Years
Exact-Capacity Projected Lifespan (SieveStore-C)	5.66 Years	0.26 Years	0.11 Years
SieveStore-C Provisioning Factor (5-Year Minimum)	1x	47x	92x
Exact-Capacity Projected Lifespan (SieveStore-C w/ 100s gap Annealing)	99.95 Years	2.45 Years	1.01 Years
SieveStore-C w/ 100s gap Annealing Provisioning Factor (5-Year Minimum)	1x	2.04x	4.96x

2.3 SSD Cache Write-Stream Characteristics

In this section, I characterize the nature of the write stream seen at the SSD cache. I use the block IO traces from [10] (see Table 2.2), which contains all block-device requests to the storage backend below the buffer cache, as the starting point. To focus specifically on the accesses seen at the SSD cache and the writes in particular, I use a prior SSD-caching technique [3] (SieveStore-C) to model the SSD cache operation and the selective caching of the most popular blocks.

The goal of my workload characterization is to achieve the design that is most effective in filtering writes from reaching the cache's SSD. Though I offer details of my analysis that leads to such a design, I briefly summarize my key claims in advance.

1. Though each front-end server has a buffer-cache, there is value in having an additional shared RAM-based LRU-cache at the SSD-cache to capture residual reuse.

2. While the RAM-based LRU cache is useful to achieve some dilution of write intensity at the SSD cache, it is not sufficient. There is an abundance of writes that still filter through to the SSD. Furthermore, extending the size of the LRU cache is an ineffective way to capture these writes.

3. A write-buffer that uses LFU-I is the best performing design to capture the writes from the cache-filtered stream. It outperforms both LRU and LFU based designs.

I justify each of the above claims in the remainder of this section.

Justification for Claim 1:

Figure 2.2 plots the fraction of writes captured (Y-axis) by various RAM-based structures (different curves) of various sizes from 4 MiB to 256 MiB (X-axis). I primarily consider fmy types of RAM-based structures: LRU caches, LRU write-buffers, LRW write buffers, and popularity-based caches. While the first three are practical struc-

Table 2.2.: Summary of Reference Ensemble Trace Set

Key	Name	Volumes	Drives	Size (GB)
Usr	User home dirs	3	16	1367
Proj	Project dirs	5	44	2094
Prn	Print server	2	6	452
Hm	Hardware monitor	2	6	39
Rsrch	Research projects	3	24	277
Prxy	Web proxy	2	4	89
Src1	Source control	3	12	555
Src2	Source control	3	14	355
Stg	Web staging	2	6	113
Ts	Terminal server	1	2	22
Web	Web/SQL server	4	17	441
Mds	Media server	2	16	509
Wdev	Test web server	4	12	136
	Total	36	179	6449

tures, the popularity-based cache represents an oracular structure that fills the cache with the most popularly accessed blocks in a recent 2-hr time window.

The results show that a normal LRU-cache captures more than 40% of the writes at lower capacities. Indeed, it is the best performing option till about 28MiB of RAM capacity. It is this component that my design aims to capture by using a 28MiB shared LRU-cache which dilutes the write intensity that reaches the SSD-cache. I refer to the SieveStore-C augmented with a 28MiB RAM-based LRU cache as SieveStore+.

Justification for Claim 2:

While the LRU cache is useful in capturing some residual reuse, we see in Figure 2.2 that at capacities higher than 28 MiB, a popularity-based approach yields better write filtering. Blindly increasing the size of the LRU-cache is sub-optimal. Of

course, because the popularity-based approach is an oracular approach, one must still consider realistic alternatives. As I show next, some frequency-based practical designs also outperform the LRU-cache.

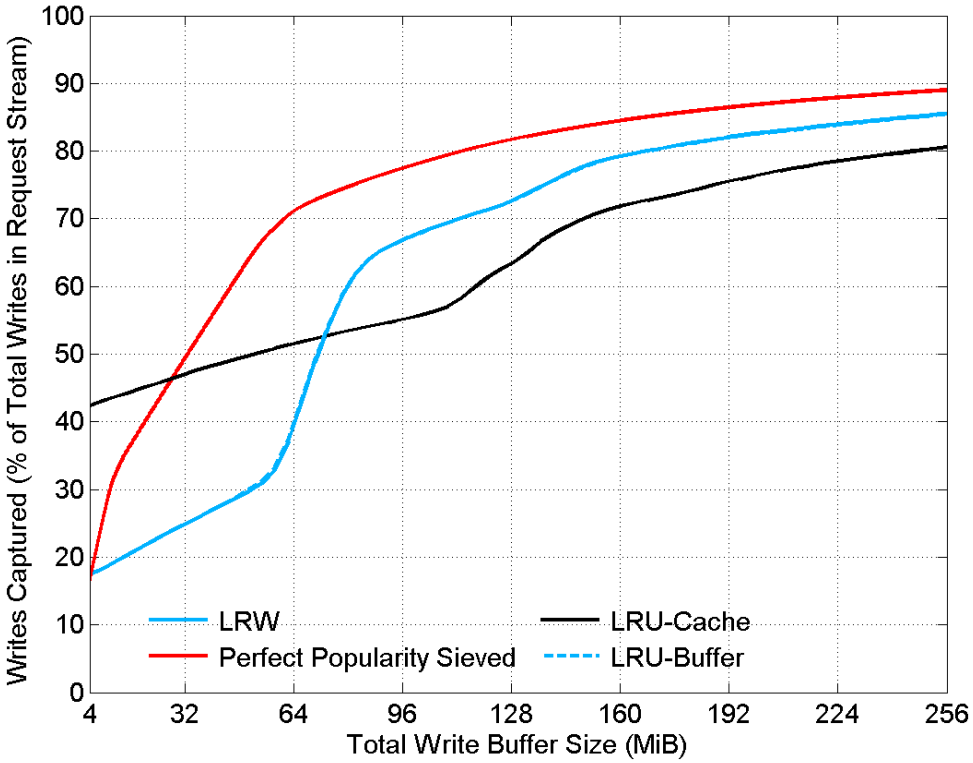


Fig. 2.2.: Comparison of the Write-Stream Dilution Strength of Least Recently Access-based Caches and Buffers

Justification for Claim 3:

Figure 2.3 characterizes the traffic *below* the RAM-based LRU-cache in terms of writes captured (Y-axis) at various RAM capacities (X-axis). Each curve corresponds to a unique configuration for the write buffers. Figure 2.3 includes two recency-based variants of the write-buffer (LRU and LRW), one frequency-based variants of the write-buffer (LFW), and one that uses a combination of frequency and recency (LFW-I). I include the oracular design with perfect sieving as well.

I can observe that in the traffic below the RAM-based LRU-cache, (a) frequency-based designs work well at lower capacities, (b) recency based designs work well at high capacities, and (c) LFW-I, which uses both recency and frequency performs well across the entire capacity range.

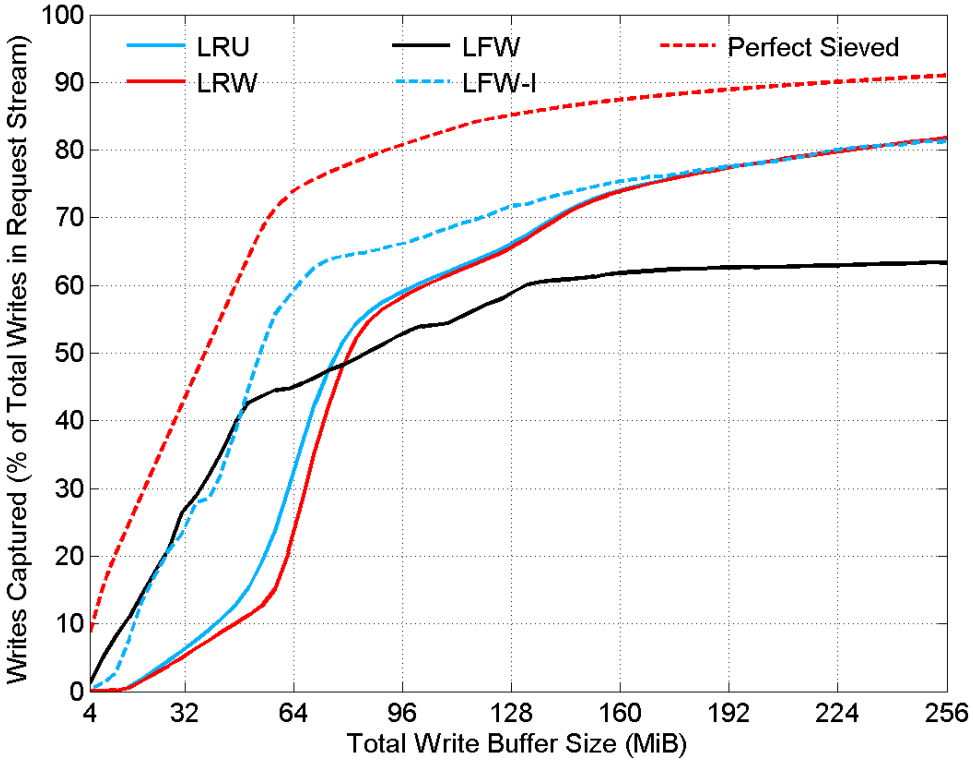


Fig. 2.3.: Comparison of the SSD Write-Stream Dilution Strength of Least Recently/Frequently Access-based Caches and Buffers for SieveStore+

While I now know the relative performance (in terms of diluting the writes seen at the SSD) of various frequency, and recency-based caching strategies for my workload traces, it is equally important to understand and quantify the overheads of of these strategies.

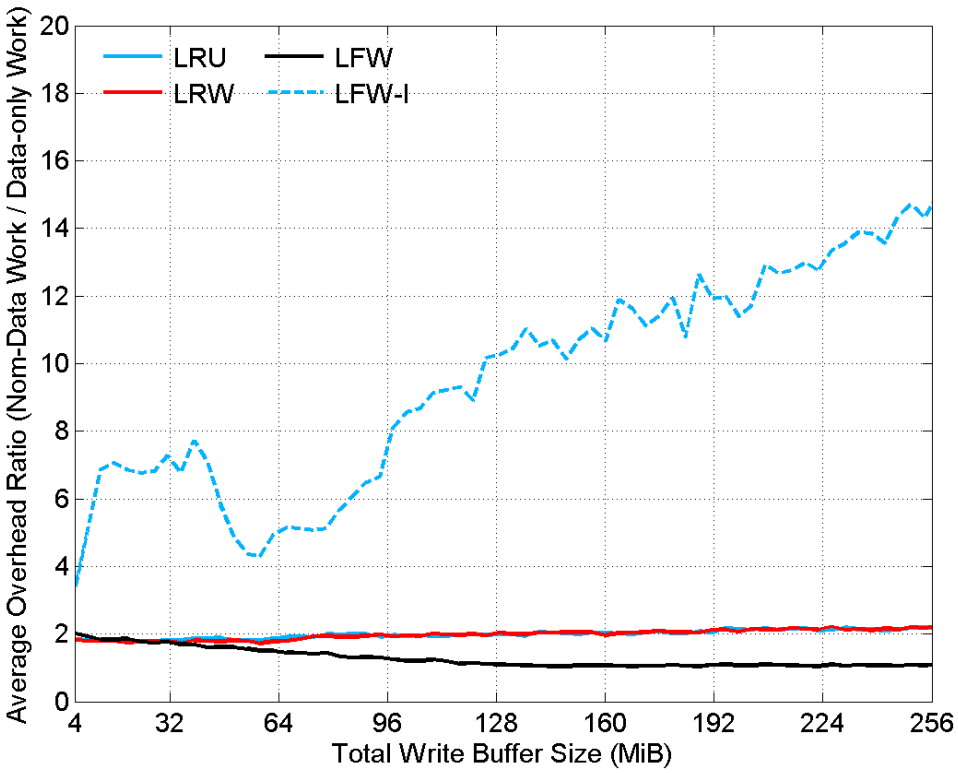


Fig. 2.4.: Comparison of the decision making overhead profiles of LR, LF, and LF-I based Buffers

2.3.1 Computational overheads of Recency-based and Frequency-based Write Buffers

Figure 2.4 shows the overhead of each technique normalized to that of the basic block access (Y-axis) at various RAM capacities (X-axis). Both recency-based designs exhibit an overhead factor of 2x regardless of buffer capacity, meaning that the time taken to make all decisions regarding a request was on average twice that of the time needed to move the needed block data in or out of the buffer. In contrast, the LFW design achieves lower overhead than the LR based designs for capacities larger than 64MiB but in an adverse way. Because I count the overhead normalized to data transfer times, LFU achieves lower relative overhead by having more evictions which would need to move both the request's block data and the evicted item's block data.

The LFW-I design has significantly worse overhead factors which get progressively worse with buffer size. LFW-I has a factor of nearly 4x at a 4MiB capacity, which is nearly twice (2x) as bad as the other designs, and soars to a nearly 14x factor at 256MiB, which is seven times (7x) worse than the LR types. Furthermore this factor of 7x at 256MiB is nearly fourteen times (14x) worse than the LFW design, despite the only difference in code or operation being the key inflation.

Summary:

I observe that from a write-dilution point-of-view, LFW-I achieves the best results. Unfortunately, LFW-I is also the most expensive from a performance overhead point of view. My design, which I present next, aims to achieve the best of both worlds, by achieving LFW-I-like write-dilution with significantly less overheads.

2.4 WriteGuard

The goal of WriteGuard is to leverage both frequency and recency data in a computationally efficient manner in order to maximize the amount of write-hits, thus write-absorption, for small write-buffers. In order to do this WriteGuard exploits the observation that sieving via thresholds (initially proposed by [3]) can achieve similar selection results as frequency-based sorting like happens in LFU-I without the cost of updating and resorting frequency meta data with every new access. Figure 2.5 briefly illustrates the difference in effort for selecting a top subset of items based on their popularity which is shown unsorted in the left most column set, followed by the middle set where the popularity data must be sorted before selecting the top 4 items, and terminated with a set were the desired items can be directly chosen from the initial unsorted set based on a previously set threshold of >60. While LFU-I does not sort from a fully unsorted list on each access, the large changes in values from the inflation effect does cause the on-access resort do involve relatively large portions of the popularity data, as indicated by the increasingly larger runtimes shown in Figure 2.4. Whereas a thresholding requires no adjustment directly to or relative to

any meta data other that directly involved with the currently accessed item. However, this approach does require a properly set threshold to be available before the accesses which can be challenging, and for the case of SieveStore-C is chosen via offline analysis of the IO activity. Therefore, dynamically determining this desired threshold value would enable a sieving approach to be a computationally more efficient and similarly easy to deploy alternative for LRFU style algorithms.

Fig. 2.5.: Visual comparison of selection by popularity sort versus selection by popularity threshold

One may think that directly adopting the previous approach would work for my context as well. Consider the way SieveStore-C implements caching that combines frequency and recency. SieveStore-C uses *sieving* which employs popularity thresholds to control allocation in the cache. Blocks with more accesses than the threshold are permitted to be allocated in the cache, whereas blocks with fewer accesses than the threshold are not. Although once blocks are in the cache, replacement is determined solely by using LRU (i.e., recency). However, SieveStore-C does have a major limitation; it requires offline analysis of the data-streams in order to choose appropriate thresholds and sizes for its block-popularity tracking structures. Too high of a threshold prevents blocks from entering the cache and wastes space in the cache (i.e., over-sieving). Too low of a threshold achieves inadequate filtering resulting in cache

pollution (i.e., under-sieving). The tuning challenge is to find the balance between using all of the space for the most popular blocks, while bypassing the rest of the accesses. Effectively, the choice of a threshold directly impacts the *bypass ratio* – the ratio of accesses that bypass the cache to the all accesses.

While WriteGuard may be thought of as a write-buffer based extension of the SieveStore-C cache design, the key novelty is that WriteGuard utilizes self-tuning behavior to achieve the best possible write filtering given a RAM-capacity. Such self-tuning is inherently a two-level problem. At the first level, the system must be able to automatically achieve a desired bypass ratio (i.e., a target set-point). At the second level, the desired bypass ratio must itself be tuned over longer timescales to maximize write-absorption because no static bypass ratio is optimal. Accordingly, WriteGuard employs a two-phase, history-based process to achieve self-tuning. The first of the two self-tuning phases directly tunes the filter threshold based a longer-term target of the ratio of bypassed writes and write-misses (i.e., the bypass ratio in the context of write buffers), and is discussed further in Section 2.4.2. The second phase tunes the longer-term target used by the first phase in order maximize the write absorption rate once the current target has been achieved and is discussed in further detail in Section 2.4.3. The resulting WriteGuard design has the LRFU properties that Section 2.3 demonstrated are required for the high write-absorption needed and maintains a runtime that is small and constant with buffer size while also not requiring off-line trace analysis for deployment.

2.4.1 WriteGuard Base Design

For the base form of my WriteGuard design (no self-tuning aspects yet) I used a simplified form of the SieveStore-C design where only the precise popularity counts are maintained. (In contrast, SieveStore-C uses a combination of precise and imprecise popularity counts for efficiency of counting over a large set of blocks). This was done both in order to narrow down the the filter threshold options that would need to

be tuned and because WriteGuard is designed to be an internal aspect of a selective SSD-base cache, which limits the amount of required meta-data to a more easily manageable amount. For example, at a 12GiB SSD cache capacity WriteGuard would only need to keep meta-data for at most 3 million 4KiB blocks and this tracking data can safely be in volatile memory and does not need power backup, as compared to the data sections of the buffer which must be kept in effectively non-volatile memory and thus will require power backup.

If SieveStore-C is slightly undersized relative to it's targeted popular set, blocks that are evicted due to this slight over-fitting will be forced to re-satisfy the filter, resulting in extra warm-up misses. WriteGuard prevents these extra warm-up penalties, that would otherwise result from slight over-fitting, by maintaining a small victim LRU queue that holds block ids for evicted blocks. The number of entries in the victim queue is sized to be only 10% of the number of slots in main WriteGuard buffer, requiring a negligibly small additional RAM that can be also be safely volatile. This victim queue is checked before attempting to filter a write-miss and if the block id for the request is found in the queue then it bypasses the filter and is allowed back into the buffer. One might consider repopulating filter information for evicted blocks as an alternative fix for this issue. However, no popularity data is kept for objects within the cache, both because SieveStore-C uses strictly LRU replacement and in order to help minimize the active precise table size. Furthermore, the popularity tracking state for these blocks, if kept, would be significantly larger than the simple block id and minimal LRU queue state for the victim queue.

2.4.2 WriteGuard Sieving Filter Self-Tuning

The primary contribution in my extension of SieveStore-C is the ability to automatically learn the appropriate filter threshold to apply during the sieving-based allocation. Before I discuss the method of auto-tuning the sieving threshold we must first have an effective metric for judging the current performance of the sieving filter.

Initially one might consider directly using the hit-rate of the buffer, however in cases of under-sieving the hit-rate can be very noisy due to the varying cache pollution effects. Another option would be to monitor for churn, but churn can happen because of a popular set being just slightly larger than the buffer size in addition to being from truly poor sieving. To navigate the murky waters of under-sieving, an intuitive but more fixed reference point is needed.

When considering the role and operation of an ideally sieved cache, the sieving effect should be such that misses are either due to intentional rejections (bypasses) or cold misses. From this I derive a meaningful metric for evaluating the effectiveness of a sieving filter's operation as the number of misses due to bypassing divided by the number of total misses, which I will refer to as the bypass-ratio. Since WriteGuard is a write-buffer it's bypass-ratio calculation only considers write-misses and not read-misses. A high bypass-ratio indicates that the majority of misses are due to bypasses and a low bypass-ratio indicates that most of the misses are for reasons other than intentional bypasses and thus the bypass-ratio directly represents the intensity of sieving currently happening. The bypass-ratio also directly relates to the intuitive notion that smaller buffers/caches would need to be more selective than larger buffers/caches and thus provides a system designer an intuitive control parameter for deploying a sieved cache or buffer. As an intermediate design point, I include one variant of WriteGuard called WriteGuard-Static (or WGS) that uses offline analysis to choose a static target bypass-ratio. This variant effectively uses only the first phase of auto-tuning to tune the thresholds to achieve the appropriate bypass-ratio.

Like SieveStore-C, I maintain popularity counts over moving windows and sub-windows of time. The algorithm described in Algorithm 1 is used to make adjustment decisions for the current filter's threshold at the end of each of the filter's sub-windows. I calculate the current sub-window's bypass-ratio from the number of bypasses and misses that have happened during the current sub-window (Line 1). I then compare my current bypass-ratio with my target bypass-ratio and decrease the threshold if the current bypass-ratio is higher than the target bypass-ratio and otherwise increase

Algorithm 1: Filter Threshold Tuning (Run at end of each filtering sub-window)

```
1  curr_bypass_ratio = window_bypasses/window_write_misses;
2  if bypass_ratio_target < curr_bypass_ratio then
3  |  delta = −1;
4  else
5  |  if prior_delta > 0 then
6  |  |  delta = prior_delta ∗ 2;
7  |  else
8  |  |  delta = 1;
9  |  end
10 end
11 filter_threshold = filter_threshold + delta;
12 prior_delta = delta;
```

the threshold (Line 2). The filter's threshold is increased even when the current bypass-ratio matches the target bypass-ratio because it is generally safer to predict that a higher sieving threshold should be used for the next window, unless I know that I are currently over-sieving. Threshold decreases are done slowly with a delta of negative one (Line 3) since over-sieving results in an under utilization of the buffer capacity but does not result in write-buffer overrun from too many blocks entering the buffer. Additionally even small decrements near hot-set boundaries can result in large increases in the number of blocks allowed through the filter, which allows for effective popular set boundary detection with small decrements. When the decision is made to increase the threshold, the prior adjustment is considered with a prior increase being accelerated (Line 6) and a prior decrease being reversed (Line 8).

Algorithm 2: Bypass-Ratio Tuning (Run at end of each tuning evaluation window)

```
Age and update the recent write hit/total history array;
curr_bypass_ratio_gap = |bypass_ratio_target − curr_bypass_ratio|;
if 0.05 > curr_bypass_ratio_gap then
    Calculate the average hit-rate for the recent history;
    hit_rate_delta = curr_avg_hit_rate − prior_avg_hit_rate;
    hit_rate_delta_pos_guard = prior_avg_hit_rate ∗ 0.05;
    hit_rate_delta_neg_guard = −(hit_rate_delta_pos_guard);
    if hit_rate_delta_neg_guard > hit_rate_delta then
        current_bypass_ratio− = prior_bypass_ratio_adjustment;
        prior_bypass_ratio_adjustment =
          −(prior_bypass_ratio_adjustment);
    else if hit_rate_delta_pos_guard < hit_rate_delta then
        current_bypass_ratio+ = prior_bypass_ratio_adjustment;
```

2.4.3 WriteGuard Bypass-Ratio Target Self-Tuning

While the target bypass-ratio (as done in WGS above) is an intuitively settable parameter that does not require a detailed offline analysis to get decent sieving performance, it does require some offline analysis to in order to find the right target ratio to optimize the sieving performance during usage of the design. Because I wish to avoid any offline analysis, the second phase of WriteGuard's self-tuning adjusts the target bypass-ratio used based on tracking the recent hit-rate history once the current target has been achieved. The need to only consider hit-rate and it's history once the current bypass-ratio target has already been met is due to the previously discussed effects of under-sieving undercutting the ability to know if poor hit-rate is due to under-sieving or over-sieving. However once the intuitively set initial bypass-ratio target has been met, I can reliably use hit-rate history to determine how to

tune my target bypass-ratio toward a better one. WriteGuard does this by executing the algorithm described in Algorithm 2 at evaluation intervals, with the time interval being defined as the filter's full window divided by a configurable evaluation factor. The hit-rate history used is an array of the number of write-hits and number of writes that occurred in prior evaluation windows, with the length of the array being controlled via a similar configurable history factor parameter (which is also expressed relative to the full window). The first step of this algorithm is to age the history array by throwing out the values from oldest evaluation window and recycling the space for the current evaluation window's values (Line 1). The the gap between the current target and the current evaluation window's bypass-ratio is calculated as the magnitude of their difference (Line 2). If this gap is less than 5%, then the current window's bypass-ratio is considered to be close enough to allow an adjustment of the current target (Line 3). If an adjustment is allowed, then the average write hit-rate is computed from the history array (Line 4), followed by the delta between the current historic average and the historic average from the prior evaluation window (Line 5). If the current hit-rate is worse than the prior one by more than 5% of the prior one's value, then the prior adjustment is reversed (Line 9). If it is better than the prior one by more than 5% of the prior one's value, then the prior adjustment is reapplied. If it is with 5% of the prior hit-rate, then the bypass-ratio and any prior adjustment information is left unchanged. The prior bypass-ratio adjustment state is initialized to a value of 0.01 (1%) during WriteGuard initialization.

2.5 Methodology

2.5.1 Initial Baseline SSD Cache

The simulation code used for the SieveStore-C runs in [3] was extended to export a trace log of all block accesses to the SieveStore-C instance's internal storage for use in both offline analysis and design profiling simulation runs. The SieveStore-C configuration used for my reference cache instance is detailed in Table 2.3. The

storage access trace from [10] was used as the input trace for the reference SieveStore-C instance and is summarized in Table 2.2.

Table 2.3.: SieveStore-C SSD Cache Parameters

Parameter	Value
Window Span	8 Hours
Number of sub-windows	4
Imprecise Filter Size	500 Million Slots
Imprecise Filter Threshold	9
Precise Filter Size	5 Million Slots
Precise Filter Threshold	2
Cache Capacity	3Mi Pages (12 GiB)

2.5.2 Trace Analysis

A custom IO trace profiling tool was used to perform all of the access stream analysis discussed during Section 2.3.

2.5.3 Revised Experiment Baseline

I extended the SieveStore-C design to have an internal 28MB LRU RAM cache after the initial findings discussed in Section 2.3. To do this, a new baseline trace was generated by extending my LRU Cache design simulator module to export it's downstream access trace to a similarly formatted log file for use as the revised baseline trace for all buffer related trace analysis and design simulations.

2.5.4 Comparison with PID Controller based Threshold tuning

Given that the threshold tuning stage of the design involves a control loop approach, I implemented a variant of WriteGuard that employs a proportional-integral-derivative (PID) controller approach for managing the direct filter threshold tuning stage for comparison. However, PID controllers need initial tuning to set constants used during the respective P, I, and D control loop steps and these values would depend on the trace patterns given those are largely what determine the result of threshold adjustments. Therefore to in order to reasonably manage the tuning of these constants for a given trace, I adapted the Ziegler-Nichols method for determining PID constants into a calibration mode that starts at the beginning of the trace and tunes the P constant (with I and D stages inactive) until oscillatory behaviors are observed, at which point the full set of P, I, and D constants are calculated and then activated for the remainder of the buffer operation. I refer to this PID based variant as WGP.

2.5.5 Design Simulations

Each design variant (e.g., recency-based, frequency-based, and hybrids) was implemented as a modular extension to the main simulation code used for the SieveStore-C simulations. This code base was then extended to track the amount of processor time used to perform all of the stack-maintenance work and the emulated block data movement for each request. This was done in order to quantify the relative impact each of these two phases have on the total time required for handling the block writes and reads to and from the buffer and cache designs. In all cases where design runtime was evaluated, five identically configured instances were run at different times and their various results were averaged to get the respective runtime result for that design and buffer capacity. Runtime sweeps were run in batched sweeps with no more than 24 design profiler instances running on the same 32-core server, such that each instance

would effectively get it's own dedicated core during execution. A summary of the key server configuration for these runtime profiling sweeps are summarized in Table 2.4.

Table 2.4.: Summary of Key Attributes for Servers used for Design Runtime Profiling

Parameter	Value
Server Processor	AMD Opteron™ 6320 (2.8GHz)
Server Processor Core Count	4
Server Processor L2 Cache Capacity	2MB
Server Processor L3 Cache Capacity	8MB
Server DRAM per Core (Average)	8GB
Design Instance Profilers per Processor Core during sweeps	1
Threads per Design Instance Profiler	1

2.6 Results

There are three main conclusions from my experiments.

1. All key parameters for WriteGuard and WriteGuard-Static can be intuitively set based on only system structure information, such as a general understanding of expected write skew (*i.e.* targeting the super-hot versus just the hot set). (Section 2.6.1)

2. I show that by using a bypass-ratio target for tuning sieving thresholds, WriteGuard and WriteGuard-Static based buffers are able to perform as well as the highly-effective and computationally expensive LFW-I based buffers, using only 64MB to absorb 60% of the writes otherwise seen by the SSD in a SieveStore+ style cache. (Section 2.6.2)

3. WriteGuard's self-tuned sieving approach results in a write buffer algorithm that has a lower runtime-overhead than even LRU or LRW buffers, and that it does not increase with buffer size. (Section 2.6.3)

2.6.1 WriteGuard Tuning

In this section I present the results from the sensitivity studies performed for WriteGuard-Static's target bypass-ratio parameter and WriteGuard's history factor and evaluation factor parameters (timing factors that control the frequency of the tuning operation, as described in Section 2.4.3).

WriteGuard-Static Bypass Ratio Sensitivity Sweep:

I profiled the WriteGuard-Static design for target bypass-ratio values ranging from 20% through 70% in 10% steps, values from 70% through 95% in 5% steps, and a value of 99%. Then the data for the six settings that were most representative of the sensitivities were plotted in Figure 2.6, with the writes captured (normalized to the total number of writes in the SieveStore+ reference trace, Y-axis) plotted against the various profiled buffer sizes in descending order (X-axis). In the smaller and hotter

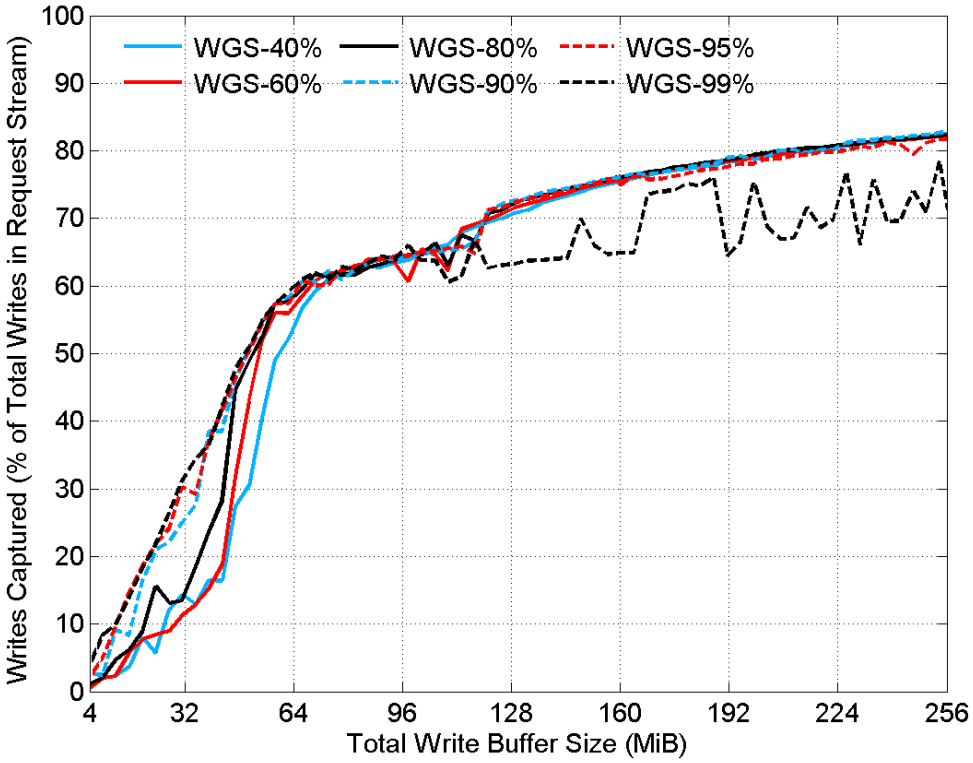


Fig. 2.6.: WriteGuard (Static Ratio Target) Bypass Ratios Sensitivity Comparison

regions higher bypass-ratio settings perform progressively better, and most settings perform similarly in the cooler regions, although the very high settings become progressively volatile in cooler regions and low settings result in more volatile performance in the warm regions (between 64MiB and <128MiB). This indicates both that it is generally better to have a high bypass-ratio setting, as long as the extremely high ranges are avoided unless intentionally targeting the super hot set of written blocks, and that only small tuning is needed for the bypass-ratio settings when not targeting the highly hot set of written blocks.

WriteGuard Auto-Tuned History Factor Sensitivity Sweep:

I profiled the WriteGuard design for history factor settings from 1x through 16x in multiples of two, with the evaluation factor fixed at a value of 4x to match the filter threshold evaluation rate from WriteGuard-Static, and the initial target bypass-ratio

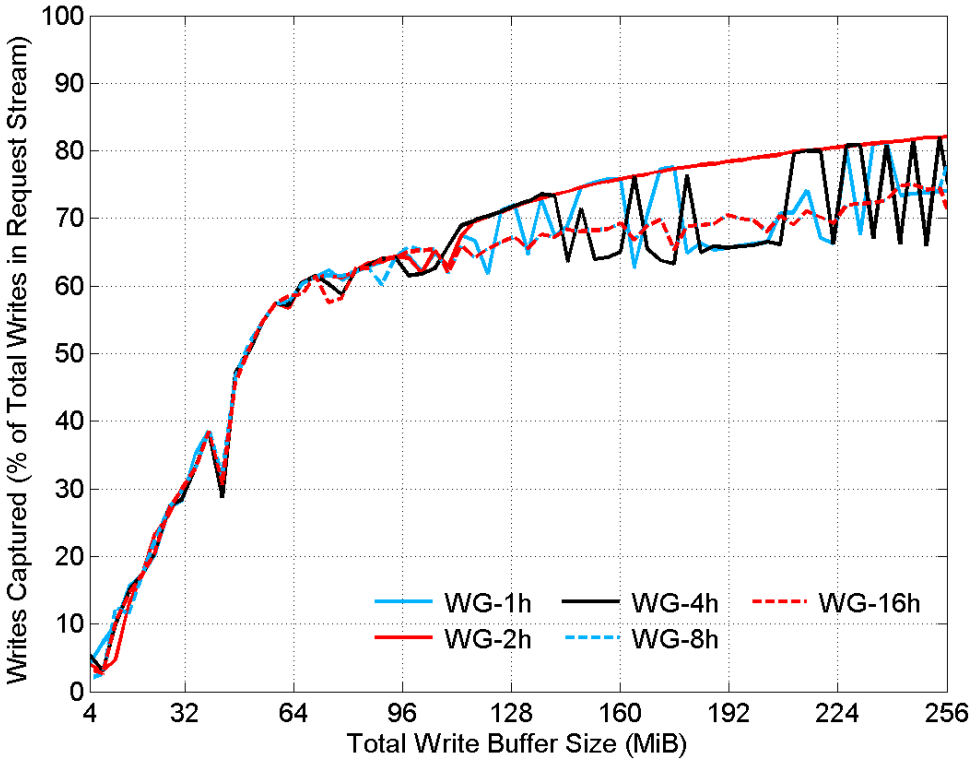


Fig. 2.7.: WriteGuard (Auto-Tuned) History Factor Sensitivity Comparison

set to 90% based on the findings in Section 2.6.1. The data is plotted in Figure 2.7, with the writes captured (normalized to the total number of writes in the SieveStore+ reference trace, Y-axis) plotted against the various profiled buffer sizes in descending order (X-axis). The various history factor settings all resulted in similar performance in the hotter regions (<64MiB) and resulted in progressively more volatile performance in the cooler regions when not at the 2x and 16x settings, with the 2x setting resulting in both cleaner and higher performance. The history factor controls how many prior full filter windows worth of stats are used for the hit-rate averaging and thus represents a strong smoothing function (after just a 2x setting) for the hit-rates compared during bypass-ratio target tuning decisions. Therefore it makes sense that the smoothing effects are effective but similar for the hotter and more dynamic regions but becomes problematic when either only the current full window is used (1x)

or too many are used (>2x), since the hit-rate changes needed for making decisions can be more easily erased in cooler regions from over-smoothing and might be hard to correctly identify if too little smoothing is done.

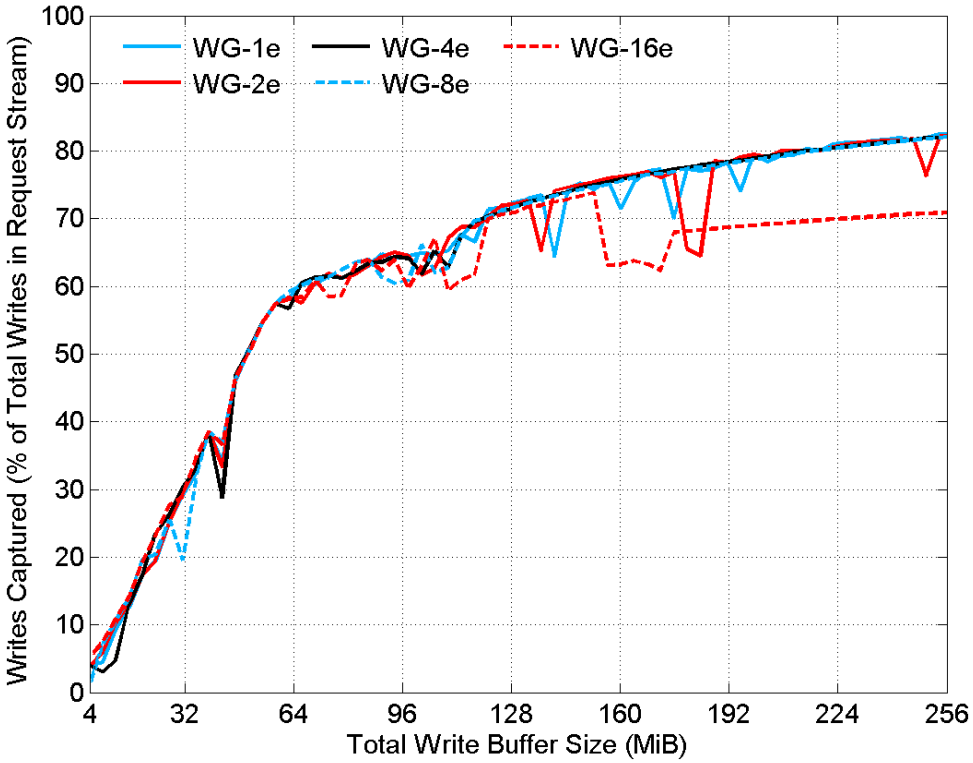


Fig. 2.8.: WriteGuard (Auto-Tuned) Evaluation Rate Sensitivity Comparison

WriteGuard Auto-Tuned Evaluation Rate Sensitivity Sweep:

I profiled the WriteGuard design for evaluation factor settings from 1x through 16x in multiples of two, with the history factor fixed at a value of 2x based on the history sensitivity findings, and the initial target bypass-ratio set to 90 based on the findings in Section 2.6.1. The data is plotted in Figure 2.8, with the writes captured (normalized to the total number of writes in the SieveStore+ reference trace, Y-axis) plotted against the various profiled buffer sizes in descending order (X-axis). The various settings resulted in close performance in the hotter regions (<64MiB). Although the 16x setting had the best performance for that region, since evaluating a

higher rate would accelerate the tuning from the initial 90% bypass-ratio toward the more optimum higher settings. In the progressively cooler regions, staying near the filter threshold decision rate, which is aligned with the underlying filter's tracking sub-windows, results in the best performance. This is due to not much tuning being needed for the bypass-ratio value, and because measurements will get noisier when making decisions at a granularity smaller than the underlying sieving filter.

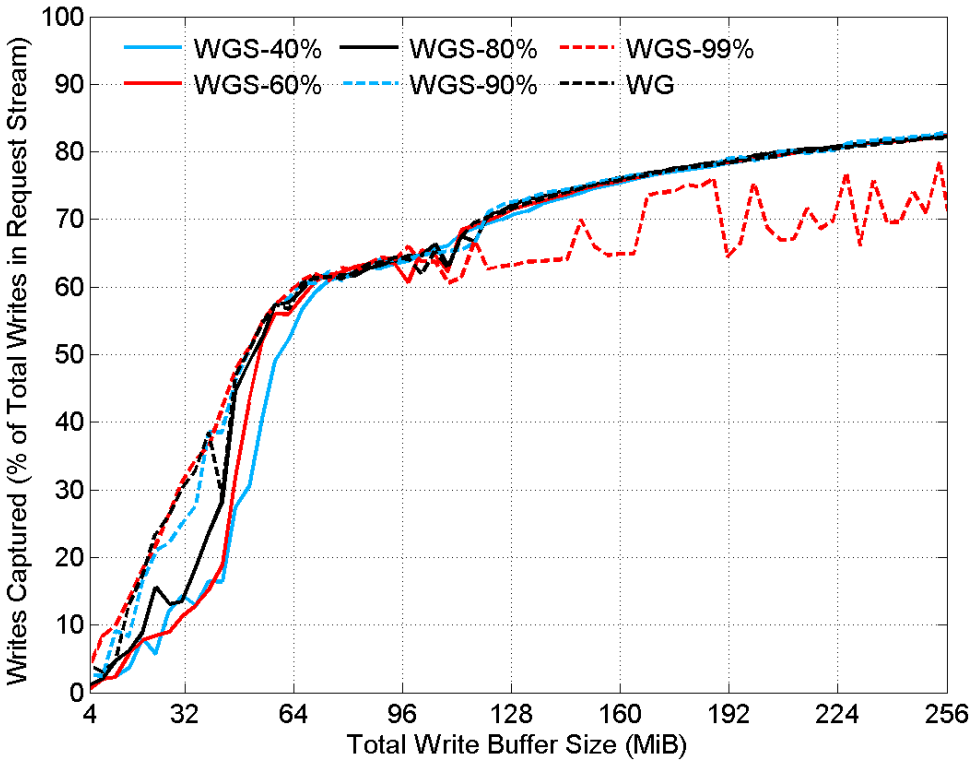


Fig. 2.9.: WriteGuard Auto-Tuning vs. Static Bypass Ratios Comparison

WriteGuard Auto-Tuning Effectiveness Comparison:

Figure 2.9 compares the performance of the tuned WriteGuard design (evaluation factor of 4x, history factor of 2x) against that of the WriteGuard-Static design for some of the target bypass-ratio settings from the prior bypass-ratio sensitivity study. In the plot, the writes captured (normalized to the total number of writes in the SieveStore+ reference trace, Y-axis) are plotted against the various profiled buffer

sizes in descending order (X-axis). From this we see that the tuned WriteGuard design is able to generally match the capture rate of the WriteGuard-Static design with the best bypass-ratio target setting for each region.

2.6.2 Write Stream Dilution Results

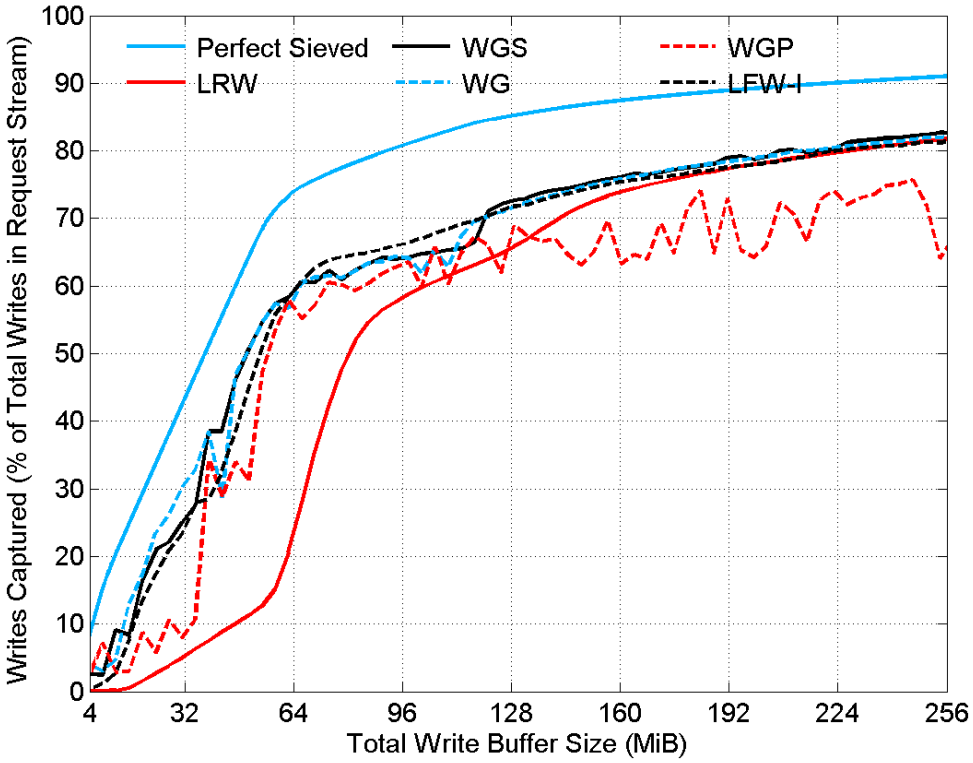


Fig. 2.10.: LRU Screened Intra-Cache Write Stream Dilution Comparison

I compare the write capture performance of the tuned WriteGuard design (4x evaluation factor, 2x history factor, 90% initial bypass-ratio target, WG), WriteGuard-Static (90% bypass-ratio target, WGS), and the PID based writeguard variant (4x evaluation factor, 2x history factor, 90% initial bypass-ratio target, WGP) with the perfectly sieved buffer, the computationally cheap LRW buffer, and the highly-effective but computationally expensive LFW-I buffer. Figure 2.10 plots the writes captured (normalized to the total number of writes in the SieveStore+ reference trace,

Y-axis) plotted against the various profiled buffer sizes in descending order (X-axis). Both WriteGuard and WriteGuard-Static designs perform better than or equal with the LFW-I design in the hotter region (<64MiB) and cooler regions (>128MiB) and only slightly worse than it for buffer capacities between 64MiB and 128MiB in size. Additionally WriteGuard nearly strictly performs better than LFW-I in the hotter region (<64MiB), and are able to capture 60% of the writes seen by the SSD while only using only a 64MiB buffer capacity. Lastly, while the PID based WG variant was able to out perform LRW in the hotter region (<64MiB) it generally under-performed relative to WriteGuard and WriteGuard-Static (and LFW-I) since PID operation depends on the system being a linear system which is not the case for thresholding given the many-to-one mapping effects that can result from threshold value changes.

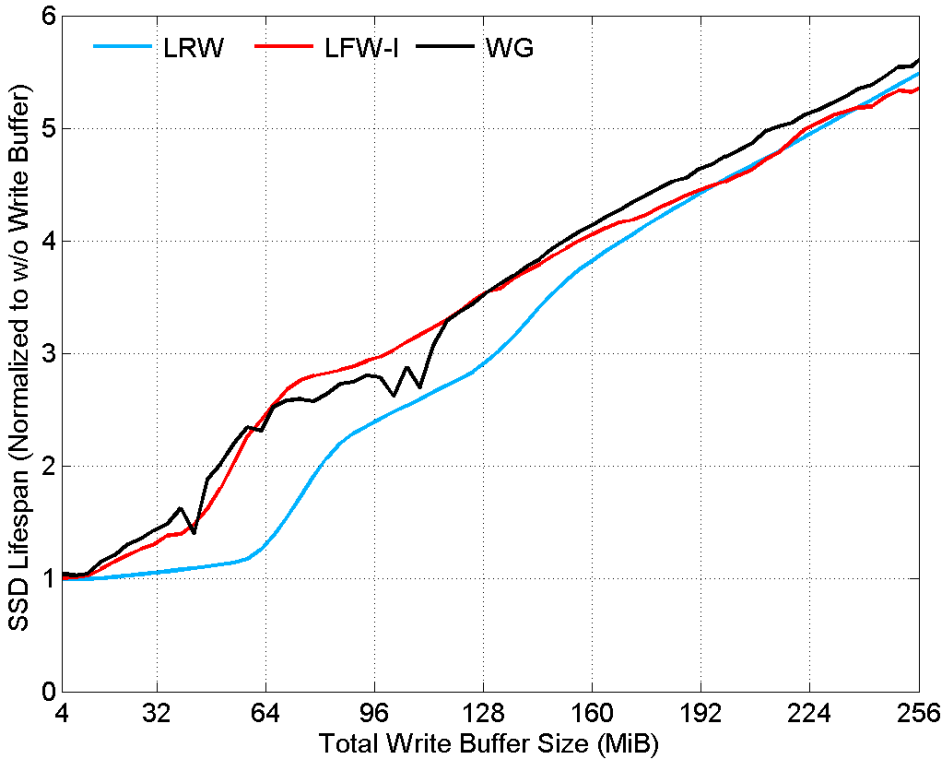


Fig. 2.11.: Underlying SSD Lifespan Improvements from Write Buffer designs

Since the ultimate goal of this work is to improve the overall lifespan of the downstream SSD I compare the ssd lifespan improvements that result from the write absorption of WriteGuard with the computationally cheap LRW buffer and the highly-effective but computationally expensive LFW-I buffer. Figure 2.11 plots the projected lifespan of the SSD with the respective upstream write buffer designs (normalized to without an upstream write buffer, Y-axis) plotted against the various profiled buffer sizes in descending order (X-axis). As expected from the prior write dilution plot, WriteGuard performs similarly well (and sometimes better) than LFW-I in the hotter regions. However, it begins to perform strictly better than both LRW and LFW-I in the cooler regions due to it's sieving nature resulting in lower churn and resulting evictions downstream as I illustrate later in Figure 2.13.

2.6.3 Buffer Design Run-time Cost Results

Next I compare the runtimes of the tuned WriteGuard and WriteGuard-Static designs with those of the LRW and LFW-I designs. Figure 2.12 plots the algorithm runtime in hours (averaged across five profiling runs) for each of the buffer designs (Y-axis) against the various profiled buffer sizes in descending order (X-axis). Both WriteGuard and WriteGuard-Static designs have runtimes matching LRW in the mingling band of curves in the 20 minutes to 30 minute range. Furthermore, on the servers used it takes approximately 20 minutes in order to just parse the whole input trace file (based on 'empty' parsing-only benchmark versions of the design profilers). This means that the runtimes of of WriteGuard, WriteGuard-Static, and LRW shown here are likely file IO bound. However, the runtime for LFW-I is both significantly longer (slower than the file IO based trace parsing) and strongly worsens as buffer size increases (ranging from 4x to 14x worse).

Since WriteGuard, WriteGuard-Static, and LRW have indistinguishable runtimes that are also constant, it useful to compare these competing algorithms in terms of another efficiency focused metric, buffer allocations. Buffer allocations directly map to

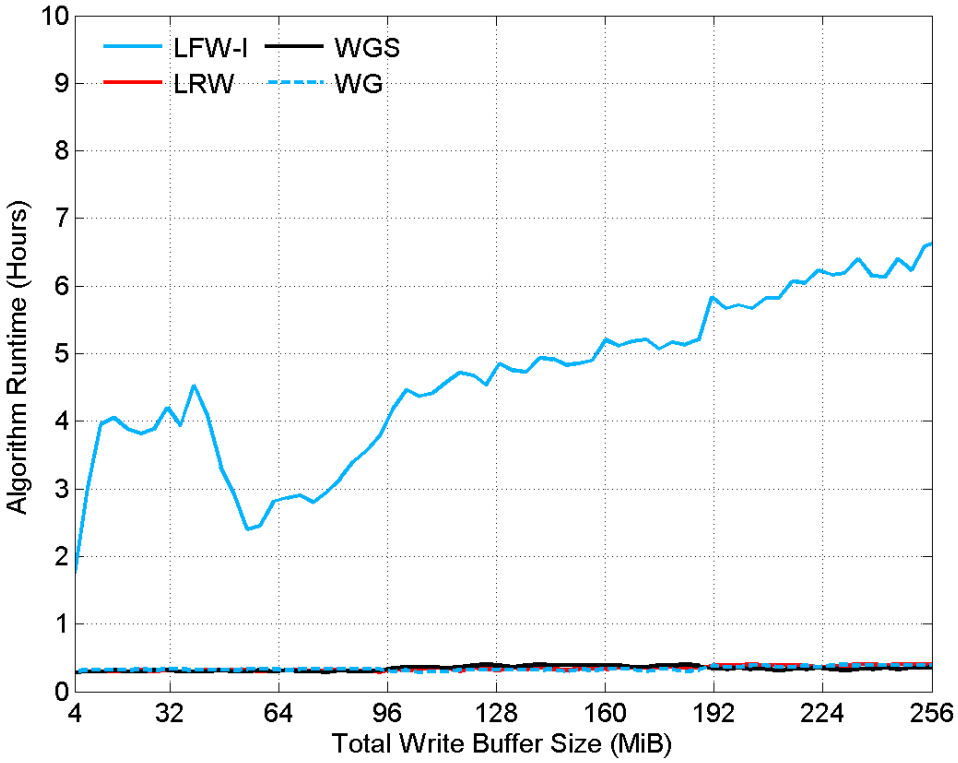


Fig. 2.12.: Comparison of the decision making run-time profiles of Write Buffer Algorithms for the SieveStore+ write-stream

overhead work, memory bandwidth usage, and power consumption that is not directly spent for write-absorption, since only write-hits correspond to filtering a write from being seen by the SSD, and thus offer a clean comparison of extra work done by the different buffer designs. Figure 2.13 plots the number of block allocations to each buffer design (in millions) against the various profiled buffer sizes in descending order (X-axis). The LRW and LFW-I designs experience massively higher allocation counts in the very hot regions (<64MiB), having as much as 10x as many allocations as the WriteGuard and WriteGuard-Static designs. The number of allocations decrease as the regions get progressively cooler, although the LRW and LFW-I still experience more than 2x as many allocations as the WriteGuard and WriteGuard-Static designs for any buffer size. The WriteGuard design experiences more allocations than the

WriteGuard-Static design in the cooler regions as a result of decisions for bypass-ratio tuning being less clear since block popularity is more similar in the long tails for cooler regions. Overall, WriteGuard and WriteGuard-Static buffers experience significantly fewer buffer allocations due to their sieving nature, resulting in a more efficient buffer design, even in the space of similar capture rates between them and LRW (buffer capacities $>$ 160MiB).

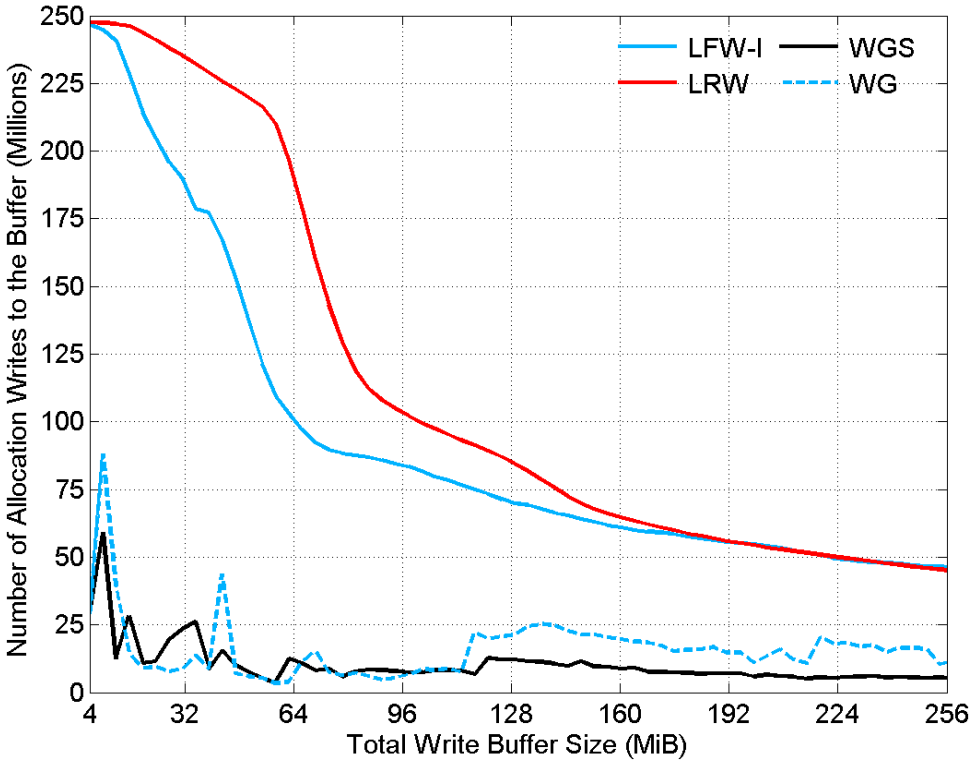


Fig. 2.13.: Comparison of the allocations made by Write Buffer Algorithms for the SieveStore+ write-stream

2.7 Related Work

A formal study of RAM buffer for flash storage to filter accesses was first proposed by [5]. In this study, authors introduced Block Padding Least Recently Used (BPLRU) buffer management scheme to improve the performance of random writes

to the flash storage. My focus is on reducing the total number of writes to flash; not to improve the performance of random writes. [11] suggests to use different replacement policy and write-back policy for the write buffer depending on the host workload in order to strike a balance between write performance and write traffic reduction. [12] proposes an address mapping and data buffering scheme named treeFTL to dynamically manage RAM buffer based on workloads. A large body of cache replacement policy studies focus on using page access recency and frequency to predict future page access pattern, including proposing variants of LRU and LFU [13–16], as well as mechanism that combines the two [17–20]. While WriteGuard also focuses on one LRFU mechanism, my goal is to do so in an efficient way.

While industry is constantly pursuing cheaper cost-per-bit NAND flash devices, as a side effect, the lifetime of such devices is also decreasing because of the increasing density and the smaller technology nodes. Most recent studies have focused on the device level to improve durability. [21, 22] propose to use Write-Once Memory (WOM) codes to improve SSD durability by more efficient page reuses. A recent study from [23] proposes techniques to optimize the lifetime and performance of a mixed high-end and low-end ssd array. [24] introduces a way to identify the weakest cells and strongest cells, and uses a wear unbalancing technique to distribute more writes to strongest cells so that the overall device lifetime is improved. Authors of [25] observed that erase voltage and erase time have a large impact on NAND endurance, and suggests to use slow erasing with a lower erase voltage to improve device endurance. While these orthogonal improvements to Flash endurance are important, WriteGuard solves the problem that exists today which is to maximize the lifetime within existing endurance limits.

Techniques to improve write performance of flash drive have also been studied. Sievestore [3] takes advantage of the skewness of disk access popularities to do selective caching to improve SSD write performance has been extensively discussed in the main body of the paper. [26, 27] suggests to apply different ECC codes to writes with different reliability guarantees to improve write latency. Other techniques, such

as using in place delta compression [28] to reduce SSD write stress have also been proposed. Exclusive caching policies for non-volatile memory based caches for flash storage has been proposed for lowering write-amplification during garbage collection by invalidating Flash storage copies of data currently resident in a non-volatile memory based cache [29].

Additionally, a recent study from Google [30] reveals that while high-end SLC drives experience lower raw bit error rates (RBERs) than MLC drives for similar program erase (PE) cycles counts, they still require replacement at similar rates to MLC drives. Their results indicate that while MLC drives are more sensitive to PE cycles than SLC drives (as previously thought) the SLCs higher PE endurance cannot be practically leveraged due to other drive errors resulting in earlier drive replacements. This increases the need for efficient and effective write-filtering like WriteGuard, since SLC drives will likely cease to be used due to their higher cost for similar replacement rates, resulting in MLC drives (with their greater PE sensitivity) remaining as the only practical option.

2.8 Conclusion

The high cost per GiB for SSDs (relative to HDDs) continues to drive their use as part of a hybrid system where they serve as a high-performance storage-tier cache for the hot data, while HDDs are used as the lower-performance bulk storage. While sieving cache designs, like SieveStore-C, greatly lessen the amount of writes to the underlying SSDs by minimizing allocation-writes, the write-hot blocks cached in these designs still pose a significant lifespan issue for the SSDs, especially since the more write-wear tolerant SLC based SSDs have been primarily phased out in favor of the higher capacity MLC SSDs. The primary means of addressing this issue are either through expensive over-sizing of SSD capacity or usage of large RAM-based write-back buffers that increase the system's backup power requirements. I show that buffer designs based on the Least-Recently-Frequent class of buffer algorithms are

needed in order to cost-effectively handle the high popularity skew present within the access-stream seen by SSDs in these caches. I then propose – WriteGuard– a self-tuning sieved write-buffer design that is able to absorb writes as effectively as the computationally expensive LFW-I design while having similar runtime costs as that of a LRW buffer.

3. OPTIMIZED STT-MRAM FOR WRITE BUFFERS

DRAM-based write buffering and caching techniques (including WriteGuard) are commonly used to help alleviate the write-pressure to SSDs, and thus over-provisioning costs, but require power backup to ensure durability of writes. Even the efficient write buffers can require sizing on the order of 1GB per 120GB capacity of the underlying SSD cache. Additionally, selective caches have access patterns that require large (e.g., 640MB-1280 MB+ relative to a 120GB cache) write buffers when used with common recency-based or frequency-based replacement. DRAM-based write buffers may need additional safeguards to ensure the durability of writes via additional power backup. Note that there may be some modest battery backup built into server/datacenter infrastructure. However, it is not a free resource that can satisfy the *additional demand* of the DRAM write-buffer backup power. As such, we account for it as an added cost.

One simple approach to avoiding the added power-backup cost is to employ Non-Volatile memory technologies such as STT-MRAM as a drop-in replacement for the write buffer's DRAM [8]. My contribution is to show that for storage write-buffers, the workload characteristics enable the use of higher-density variants of STT-MRAM. Specifically, I identify two application characteristics – (1) the vast majority of the write-buffer's contents is composed of write-dominant blocks, and (2) the vast majority of blocks in the write-buffer are overwritten within a period of 28 hours – that enable a high-density, optimized STT-MRAM as a replacement for DRAM. My optimized STT-MRAM-based write buffer achieves higher density by (a) trading off superfluous durability by exploiting characteristic (2), and (b) de-optimizing the read-performance of STT-MRAM by leveraging characteristic (1). Together, the techniques increase the density of STT-MRAM by up to 80% with low or no impact on write-buffer performance.

3.1 Introduction

DRAM-based write-buffers and caches are currently used to absorb significant portions of these problematic writes in order to minimize the over-provisioning of SSD cache capacity needed to maintain drive lifetimes (through wear-leveling) beyond that of typical servers lifespans (i.e., 3-5 years). However, these RAM buffers require power backup to ensure the durability of writes and the access patterns for these caches necessitate common recency-based or frequency-based write buffers to be large (e.g., 100MB+ relative a 12GB cache) to be effective. More effective algorithms, like the more computationally expensive class of joint recently-frequent (LRFU) policies and the computationally efficient sieving WriteGuard algorithm detailed in Chapter 2, are able to reduce the required buffer capacity (to approx. 64MB for the same 12GB cache). While these algorithms meaningfully reduce the required capacity, and thus power backup, they still require significant power backup for the DRAM buffer.

Before we begin to discuss alternative ways to ensure the durability of write buffers, it is important to first clarify the respective durability and failure model for this design space. The type of failures these buffers must be designed to endure is short-lived crashes, which typically do not result in true data loss due to data replication both within and across data-centers ensuring that other copies exist during the crash and recovery period. However, in order to maintain the desired level of replication, there are two primary choices for handling a crash event. The first is to assume that all copies on the involved node are lost which requires re-replicating all blocks held by that node onto other nodes, which involves Terabytes of data transfers and IO. The second is to simply wait for the node to restart, which involves no extra data transfers and replication levels are restored after the restart finishes. My work targets design spaces that choose option two over option one, which means that buffered blocks need to be durable enough to survive restart cycles.

Spin-Torque-Transfer Magnetic-RAM (STT-MRAM) has been gaining ground as replacement for traditional RAM technologies because of it's non-volatility, higher

read performance, and strong density scaling. Existing STT-MRAM cell designs targeted as an SRAM replacement have densities as much as 5x higher than SRAM [31]. Furthermore, STT-MRAM shows scaling potential to come close to both embedded and traditional DRAM while also achieving higher performance [32, 33], with STT-MRAM-cell density limited by design trade-offs with read performance and non-volatility [34]. Perpendicular magnetized MTJ based STT-MRAM cell designs, where the magnetic alignment within the MTJ layers is perpendicular to the plane of the wafer, have shown improvements to write energy costs and enhanced density due to their better ability to scale to smaller feature sizes and higher temperature tolerances [35]. Improvements in this design space have enabled production STT-MRAM capacities of 1Gb per chip at 22nm fabrication [36]. Although, production DRAM is currently at 16Gb per chip (before die stacking) at 10nm fabrication [37]. However, factoring the full picture of durability and elimination of periodic refresh power and performance penalties alongside the lower density of current STT-MRAM has resulted in early but limited adoption in some high-performance enterprise SSD designs [8].

Optimizing STT-MRAM-cell for use in traditional designs is complicated by multiple trade-offs between it's promising properties (read performance, bit-cell density, and non-volatile) and improving write energy and latency costs. Read performance and bit-cell density related trade-offs result from adjusting the bit-cell's access transistor or magnetic layers, with improvements for write-current, density, and latency lessening the differential between bit-states during reads or even causing destructive reads [34]. Trade-offs involving non-volatility offer a significant opportunity for improving write energy and performance without negatively impacting read performance and density, but necessitates study of data lifespans within workloads to properly leverage the cells [38,39]. Furthermore, optimization in traditional application spaces has resulted in cell designs which have MTJ stacks optimized to match the overlapping region with their underlying access transistor, resulting in no density benefits from reduced cell retention time [40]. However, this access transistor sizing (and the resulting effective MTJ size reduction limit) is only necessitated by the dependence

on read performance in traditional application domains, due to access transistor size reductions resulting in worse read performance.

In this work, I identify two specific application characteristics – (1) the vast majority of the write-buffer's contents (e.g., 95%) is composed of write-dominant blocks, and (2) the vast majority of blocks in the write-buffer (e.g., 98%) are overwritten within a period of 28 hours. I show that these characteristics help enable a high-density, optimized STT-MRAM as a replacement for DRAM, which enables durable write-buffers (thus eliminating the cost of power backup for the write-buffer). My optimized STT-MRAM-based write buffer achieves higher density by (a) trading off superfluous durability by exploiting characteristic (2), and (b) deoptimizing the read-performance of STT-MRAM by leveraging characteristic (1). Together, the techniques increase the density of STT-MRAM by 25% with low or no impact on write-buffer performance.

Section 3.2 provides a brief introduction to STT-MRAM and a discussion of cell retention time related design trade-offs. In Section 3.3 I discuss the results from analyzing the opportunity for write-buffer focused customization of STT-MRAM cell designs. In Section 3.4, I discuss the details of my proposed cell design trade-off and ways to manage it's costs. I then discuss my evaluation methods in Section 3.5 followed by the corresponding results in Section 3.6. Section 3.7 discusses related prior work. And I state my final conclusions in Section 3.8

3.2 STT-MRAM Technology Background

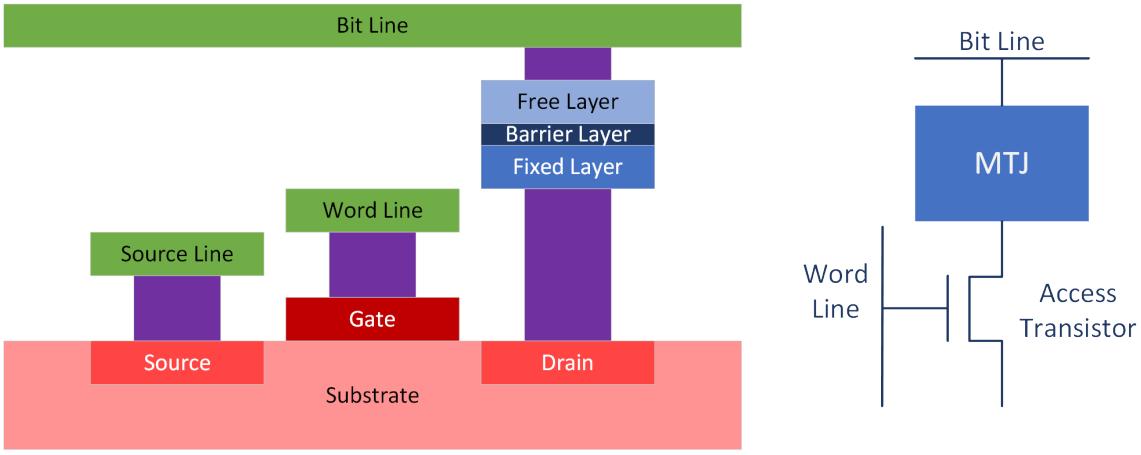


Fig. 3.1.: Simplified Diagram of a STT-MRAM cell

Figure 3.1 illustrates the general structure of a STT-MRAM bit-cell, where a switchable magnetic filter (the Magnetic Tunnel Junction or MTJ) is connected between per-bit wire and an access transistor. The MTJ is composed of a magnetic layer with a fixed magnetic alignment referred to as the fixed layer, a tunneling barrier referred to as the barrier layer, and a magnetic layer with a switchable alignment referred to as the free layer. Data stored in the bit-cell is encoded via the relative alignment of these two magnetic layers within the MTJ, with matching alignments resulting in less resistance to electric current traveling through it and opposing alignments have a higher resistance to electric current traveling through it. Earlier designs had the alignments of these magnetic layers parallel to the plane of the wafer and are now referred to as in-plane MTJ (iMTJ) and have bit states similar to those shown in Figure 3.2a. More recent designs use magnetic alignments perpendicular to plane of the wafer due to there better scaling and thermal characteristics and have bit states similar to those shown in Figure 3.2b [35]

(a) In-plane MTJ (iMTJ) (b) Perpendicular MTJ (pMTJ)

Fig. 3.2.: Relative magnetic alignment for distinct STT-MRAM bit-states

3.2.1 Adjusting STT-MRAM Cell Retention Time

The retention-time of a cell is controlled by the cell's Magnetic Tunnel Junction (MTJ) properties with respect to thermal noise resulting in changes to it's magnetic alignment, which would potentially result in a bit-flip or read error. This MTJ characteristic is referred to as the thermal barrier and denoted by Δ. The retention time for a given cell follows Equation 3.1, with C and k as fitting constants that are dependent on MTJ design but independent of the MTJ volume (MTJ volume does affect retention time via the Δ term).

$$Retention\ Time = Ce^{k\Delta} \tag{3.1}$$

Therefore the change in needed thermal barrier between two different retention times can be calculated via Equation 3.2. And the reduction in thermal barrier between the two times (in percentage) can be calculated via Equation 3.3.

$$\Delta_1 - \Delta_2 = \frac{ln(\frac{t_1}{C})}{k} - \frac{ln(\frac{t_2}{C})}{k} = \frac{ln(\frac{t_1}{C}) - ln(\frac{t_2}{C})}{k} = \frac{ln\left(\frac{\frac{t_1}{C}}{\frac{t_2}{C}}\right)}{k} = \frac{ln(\frac{t_1}{t_2})}{k} \tag{3.2}$$

$$100.0\left(\frac{\Delta_1 - \Delta_2}{\Delta_1}\right) = 100.0\left(\frac{\frac{ln(\frac{t_1}{t_2})}{k}}{\frac{ln(\frac{t_1}{C})}{k}}\right) = 100.0\left(\frac{ln(\frac{t_1}{t_2})}{ln(\frac{t_1}{C})}\right) \tag{3.3}$$

Furthermore, retention time for a cell in nanoseconds can be approximated with C set to 1ns and k set 1 [41], resulting Equation 3.4 with t measured in nanoseconds. This logarithmic relationship clearly indicates that retention time would need

to shrink by orders of magnitude for any appreciable gains in density for the cell, which is shown to be the case in Section 3.6.

$$100.0\left(\frac{\Delta_1 - \Delta_2}{\Delta_1}\right) = 100.0\left(\frac{ln(\frac{t_1}{t_2})}{ln(\frac{t_1}{1ns})}\right) \tag{3.4}$$

3.2.2 Adjusting STT-MRAM Magnetic Tunnel Junction

There two primary ways to change the MTJ (and thus its retention time) without fundamentally changing the MTJ structure (and thus it's core fabrication processes) (1) shrinking of the thickness of MTJ layer(s) and (2) shrinking of the planar area of MTJ layers and thus the footprint of the MTJ stack. This is because the thermal barrier is directly proportional to the volume (V), its in-plane anisotropy field (H_k), and it's saturation magnetization (M_s) and it is inversely proportional to absolute temperature in Kelvin (T) as captured in Equation 3.5.

$$\Delta \propto \frac{V H_k M_s}{T} \tag{3.5}$$

Furthermore, the critical write current (I_c) is proportional to the MTJ planar area (A) according to Equation 3.6, where C and *gamma* are fitting constants and J_{c0} depends on the MTJ vertical structure [42].

$$I_c(write\ time) = A\left(J_c0 + \frac{C}{(write\ time)^{\gamma}}\right) \tag{3.6}$$

Therefore both options reduce the necessary write energy and thus write power or write performance, with area changes having a directly proportional impact. Changing the layer thickness does not affect cell density because it does not result in changes to the cell area related dimensions. Whereas, changing the layer planar area has the ability to directly impact the cell area, given the matching dimensions, as long as the access transistor sizing does not set a hard upper limit for the cell area. However,

under the traditional STT-MRAM cell design focus, currently optimized cells are dimensionally limited by the size of the access transistor and have their respective MTJ stacks sized to match the overlapping area of the access transistor [40].

This constraint wherein the access transistor limits any area reduction is a consequence of the read-optimized cell sizing, which disallows any reduction in the size of the access transistor. My work shows that significant (e.g., 20%) area reduction (and hence STT-MRAM density improvements) are possible for the write-dominated workload seen at the write buffer. One may think that it is tautological to expect that writes dominate traffic at write buffers since all writes are, by definition, steered to the write buffer. However, if those written blocks are then re-read one or more times, arbitrary read-write ratios are possible. My workload characterization shows that reads to written blocks are extremely rare (e.g., less than 7% of buffered blocks have write-to-read ratios of less than 3x and effectively 100% of reads occur to blocks with write-to-read ratios that are less than 2x).

3.3 Opportunity for Write Focused STT-MRAM Design

There are three key conclusions from my opportunity search.

1. The accesses to the blocks held within selective caches are strongly clustered, with most reads and writes coming from respectively distinctly different sets of blocks. (Section 3.3.1)

2. Internal Screening caches (Internal RAM Caches) appear to primarily capture block sets that are read-dominant with more mixed access contributions and thus both filter the downstream trace to be more distinctly clustered and need a follow up internal access study for identifying amenable STT-MRAM cell type(s) (Section 3.3.2)

3. Internal write-buffers nearly exclusively handle write-dominant (and mostly write-only) blocks with the majority of the absorbed writes affecting strongly write-dominant blocks. (Section 3.3.3)

3.3.1 SieveStore-C Internal Access-Stream Trends

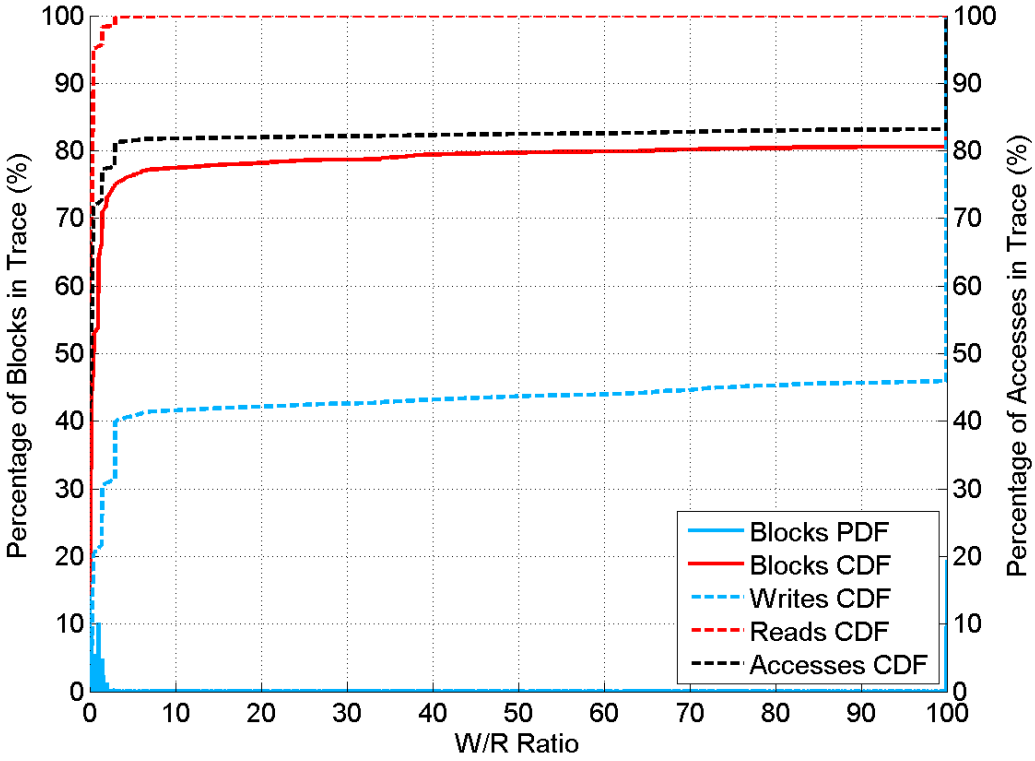


Fig. 3.3.: SieveStore-C internal access-stream write-to-read ratio profile

In this section I present the results from profiling the write-to-read ratios and respective access contributions of the blocks held within the SieveStore-C instance. The complete profiles are plotted in Figure 3.3. Block count PDF and CDF (normalized to the total number of blocks within SieveStore-C) are plotted (left Y-Axis with solid lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. Read, write, and combined access counts (normalized to the their respective totals) are plot-

ted (right Y-Axis with dashed lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. Blocks are are strongly clustered into two groups at the extremes of the write-to-read ratio bins, with approximately 20% in or near the 100x ratio bin and over 70% being in bins with ratios lower than 5x. Effectively all reads are to blocks with ratios lower than 5x, with nearly 95% being from blocks with extremely low W/R ratios. The write CDF curve sharply climbs as W/R ratios increase from extremely low values toward approximately 5x and then effectively plateau until the effectively write-only blocks (near 100x W/R ratio), with nearly 55% of writes belonging to these 'write-only' blocks. This strong read-focused and write-focused clustering make sense when one thinks about how we typically work with files in relatively focused phases, switching between creating, reading, and updating. However, a closer look is needed for the 0.0 to 5x W/R ratio region to clarify the strength of the clustering with respect to reads, since those ratios are effectively compressed into the left edge of the plot.

Figure 3.4 plots the same data from Figure 3.3 only with the X-Axis zoomed write-to-read ratio values between 0 and 5x. At this focus level, we see that the write-focusing is even stronger than previously seen, with nearly 80% of writes occurring to write-dominant blocks (W/R ratios larger than 1x) and previously observed write plateau starting around ratios of 3x. Furthermore, 95% of reads are to read-dominant blocks (W/R ratios lower than 1x), with 60% coming from effectively read-only blocks and effectively no reads from write-dominant blocks. These trends indicate a strong opportunity for leveraging a write-optimized STT-MRAM-cells for write absorption since read performance and energy characteristics would be largely irrelevant for the blocks that account for the vast majority of writes. Additionally, this indicates that read-acceleration-focused efforts could potentially have significant benefits from more strongly read-optimized (as compared to write-mitigated or balanced) STT-MRAM-cell types.

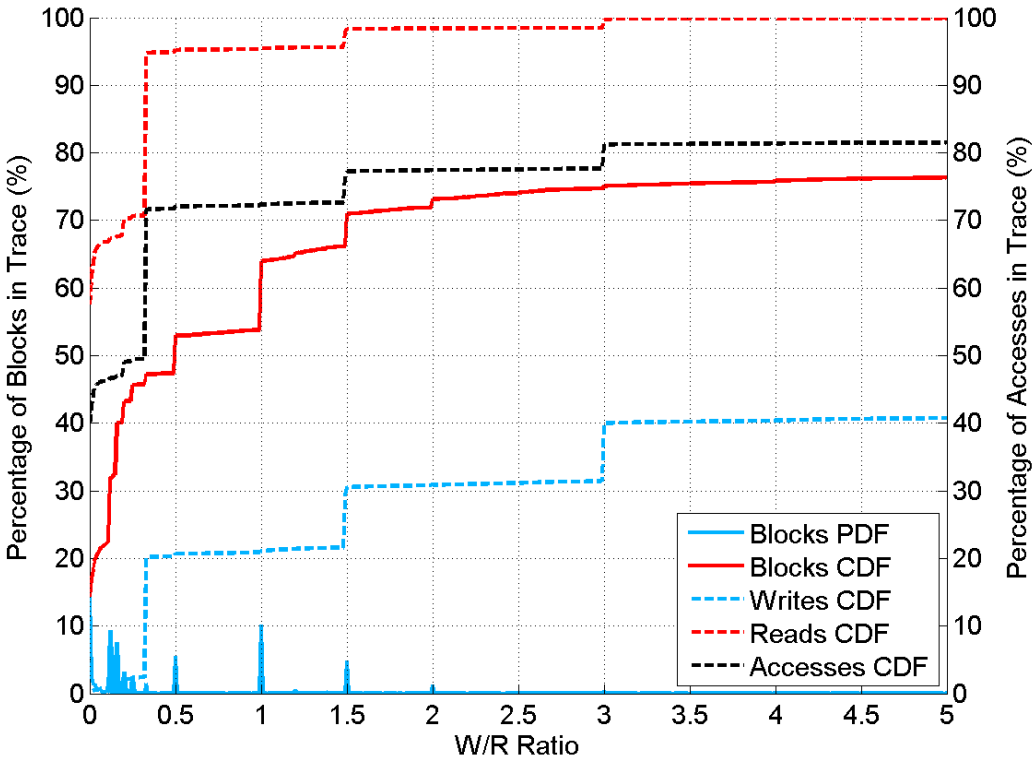


Fig. 3.4.: SieveStore-C internal access-stream write-to-read ratio profile (ratios < 5x)

3.3.2 SieveStore+ Internal Screening Cache and Downstream Trends

In order to further investigate and solidify the level of opportunity for leveraging differently optimized STT-MRAM-cell types, I analyzed the write-to-read ratio profiles for the access-stream downstream from the internal screening cache in SieveStore+. First I present the profile trends in this access-stream (Section 3.3.2), and then discuss the trend differences before and after the internal screening cache (Section 3.3.2).

SieveStore+ Write-Buffer Input Access-Stream Trends

In this section I present the results from profiling the write-to-read ratios and respective access contributions in the access-stream downstream from the screening cache in SieveStore+ (and thus the input access-stream to the SieveStore+'s write-buffer). The complete profiles are plotted in Figure 3.5 with a similarly 0 to 5x zoomed range plotted in Figure 3.6. Block count PDF and CDF (normalized to the total number of blocks within SieveStore-C) are plotted (left Y-Axis with solid lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. Read, write, and combined access counts (normalized to the their respective totals) are plotted (right Y-Axis with dashed lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. High level trends remain similar to those in SieveStore-C's access-stream, with very strong read and write clustering, nearly all reads coming from blocks with lower write-to-read ratios, nearly all writes going to write-dominated blocks, and the majority of writes going to 'write-only' blocks.

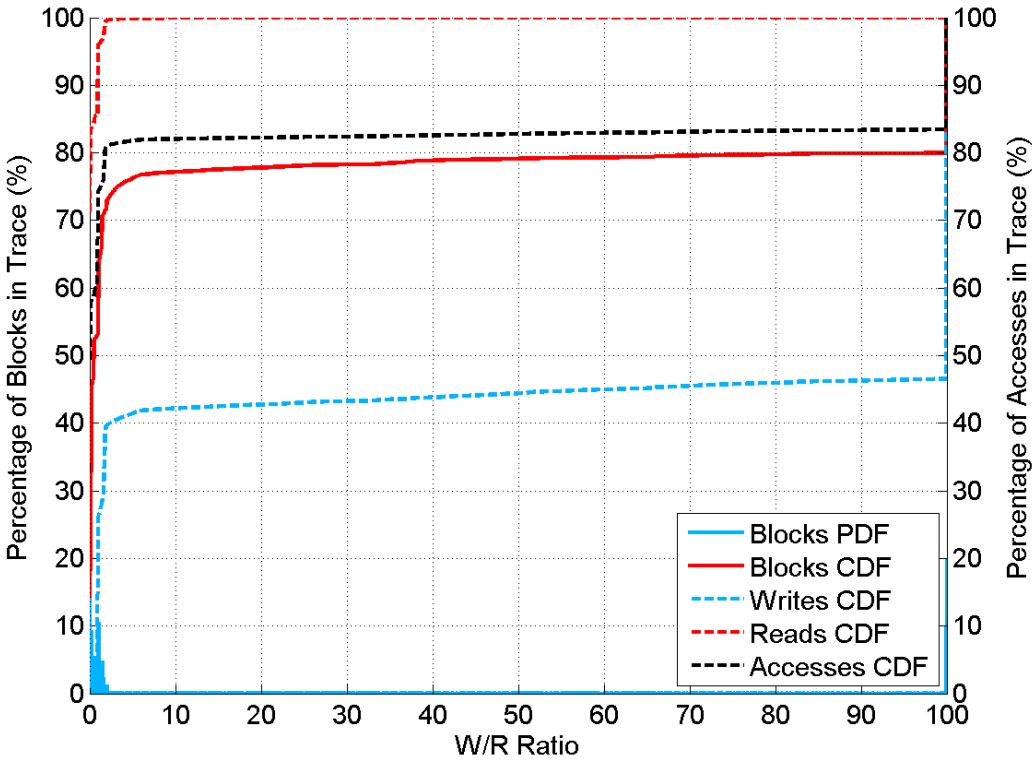


Fig. 3.5.: Write-buffer input access-stream write-to-read ratio profile

However, the plot for ratios between 0 and 5x (Figure 3.6) shows an even stronger read-focusing and write-focusing. Nearly all writes (>95%) are from blocks that have at least balanced write-to-read ratios, and consequently nearly no writes to read-dominant blocks which still account for the vast majority of reads (~ 85%). Additionally, the blocks with balanced ratios account for less than 15% of all accesses but over 25% of writes, meaning that they may still be valid candidates for write-prioritized STT-MRAM-cells in write-buffers, where write-absorption is the primary goal.

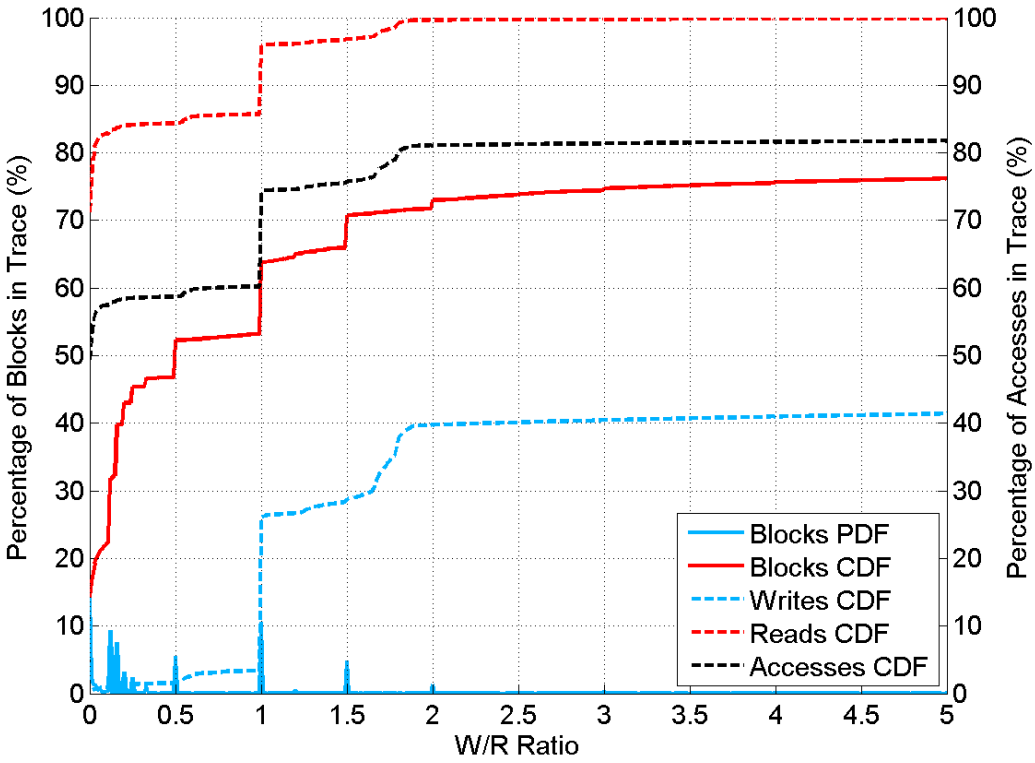
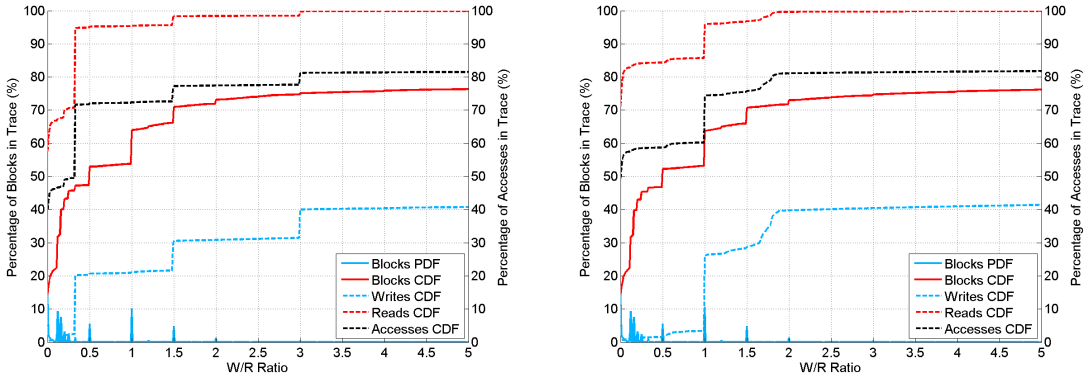


Fig. 3.6.: Write-buffer input access-stream write-to-read ratio profile (ratios < 5x)

Screening Cache Potential Opportunity Trends



(a) Screened Cache Input Stream (b) Downstream Write Buffer Input Stream

Fig. 3.7.: Comparison of pre- and post- screening cache access-streams (ratios < 5x)

In this section, I discuss the write-to-read ratio profile differences between access-streams before and after the screening cache. In the prior section it was noted that the high-level and extremely high write-to-read ratio trends are similar in both streams, so this section will focus on the region of ratios from 0x to 5x. The zoomed write-to-read ratio profile plots shown previously for the access-stream before the screening cache and the access-stream after the screening cache have been re-included side-by-side in Figure 3.7 to better facilitate this discussion.

One might think we should see a shift in the block count based PDF and thus CDF curves given the absorption of writes and reads in the screening cache. However, the screening cache in SieveStore+ has a capacity of only 28MiB (approximately 0.228% of capacity of the overall SieveStore+ cache) and so should leave the write-to-read ratios for most blocks unaffected. The screening cache captures approximately 40% of the writes (as noted in Chapter 2) and approx. 15% of the reads so we should expect to see differences in the access related profile curves from these filtered accesses. However, misses for both reads and writes will result in a downstream read or write, respectively, at a one-to-one ratio they will not trigger any access profile changes.

There are two key access related profile shifts. The first being that there are not longer very many writes (nor reads) coming from blocks with ratios of approximately 0.66 (2:3) and there is now a significant portion to blocks with balanced ratios (near ratios of 1x). This would imply that at least one of the block sets captured in the screening cache is actually more heavily read filtered and thus both would likely need more read-optimized STT-MRAM in the screening cache. The increase in the portion of reads to extremely read-dominant blocks after the screening cache is likely a result of the the total read count shrinking due to the read filtering, but may also stem from a strong write-filtering. The option of the write-optimized STT-MRAM for this 'balanced' set stems from both the focus of the write-buffer being write-absorption and that they contributed relatively less reads (around 10% of reads and 25% of writes). Secondly, the write-plateau occurs at ratios a little lower than 2x in the downstream trace and happened at ratios around 3x before the screening cache. This

implies that the screening cache has a another block set that it both more strongly write-filters and is already write-dominant and thus would likely need write-optimized STT-MRAM in the screening cache. These two key shifts indicate that the screening cache would likely need complementary sets of both read-optimized STT-MRAM-cells and write-optimized STT-MRAM-cells for optimally handling it's working set, and that it's internal trace should be studied in followup work.

3.3.3 Write-Buffer Access-Stream Trends

To solidify the level of opportunity for write-optimized STT-MRAM-cell usage in write-buffers, I analyzed the access-stream to the data buffer component inside of the WriteGuard write-buffer instance. First I present the profile trends in this access-stream (Section 3.3.3), and then discuss the impacts of reads required during the eviction of currently held blocks (Section 3.3.3).

Write-Buffer Internal Data Storage Access-Stream Trends

The complete write-to-read ratio profiles are plotted in Figure 3.8 with a zoomed view of the profiles plotted for ratios between 0 and 5x in Figure 3.9. Block count PDF and CDF (normalized to the total number of blocks within SieveStore-C) are plotted (left Y-Axis with solid lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. Read, write, and combined access counts (normalized to the their respective totals) are plotted (right Y-Axis with dashed lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin.

Similar to the previous traces, there is a clustering of read-focused and write-focused block sets. However, the writes are are more distributed throughout the range of ratios from 3x through 100x, but this range still accounts for over 55% of writes and less than 5% of reads. Writes are also generally more dominant, with

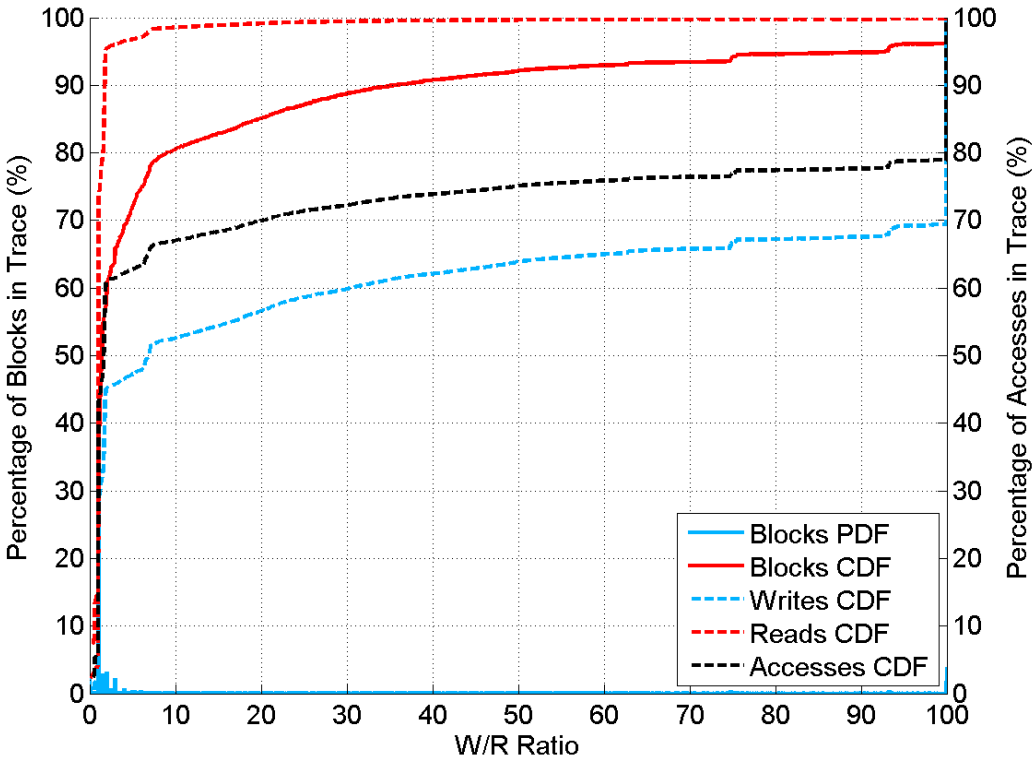


Fig. 3.8.: Write-buffer internal block storage access-stream write-to-read ratio profile

the writes to these strongly write-dominant blocks accounting for nearly 40% of all accesses, instead of 15% of the accesses like the previous traces. This makes sense for the following two reasons, (1) the only reads to the write-buffer are from read-hits and reads of blocks being evicted, since write-buffers do not allocate for read-misses and (2) reads were previously shown to be primarily mapping to blocks that had very low amounts of writes and thus would likely not be kept in the write-buffer.

When we focus on the regions for lower write-to-read ratios (Figure 3.9), we see that reads are primarily concentrated in the blocks with balanced ratios, with approximately 60% of reads mapping to these blocks and less than 15% of reads going to read-dominant blocks (ratios <1x). Given the concentration of reads and that the sum of the reads and writes to these balance blocks accounts for over 40% of

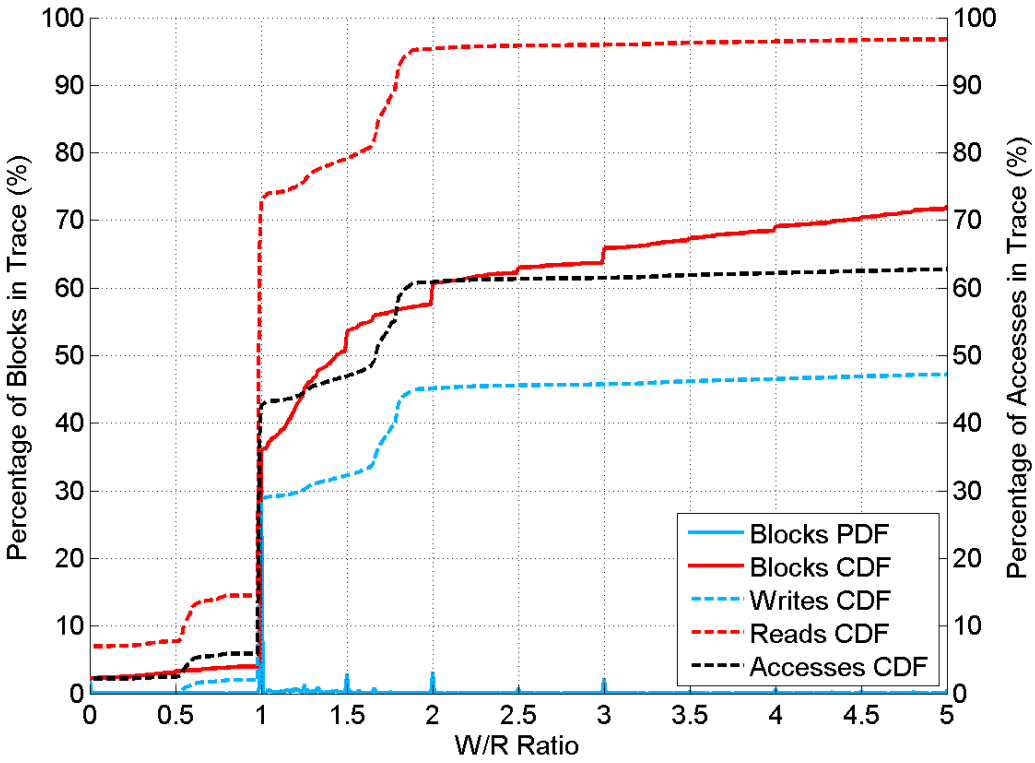


Fig. 3.9.: Write-buffer internal block storage access-stream write-to-read ratio profile (zoomed to Ratios less than 5x)

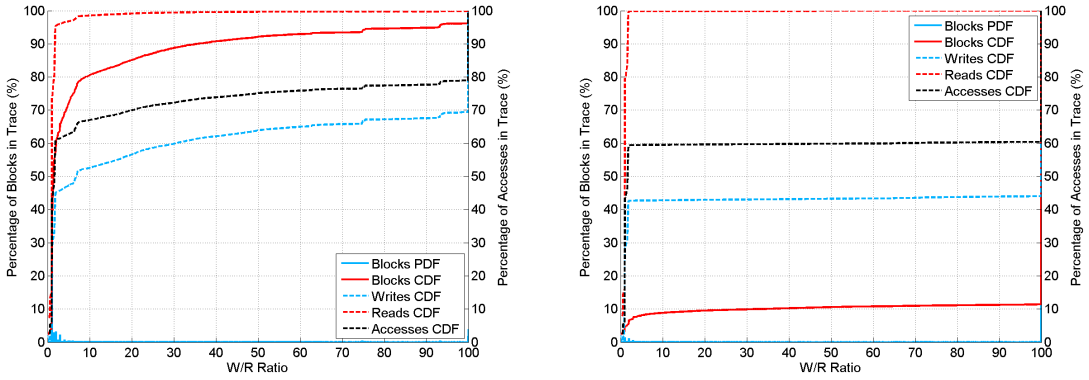
accesses, it would seem balanced STT-MRAM-cells would be needed for optimized performance. Although, the level of balance needed would depend on how bad the read-related trade-offs were, since the reads share that 40% of accesses with a significant portion (approximately 30%) of the overall writes. As in the previously analyzed traces, there are effectively no reads to strongly write-dominant blocks (ratios >2x) and these blocks account for 55% of writes, with more than half of them (30% overall) coming from the 'write-only' blocks (as was shown in Figure 3.8). Furthermore, accesses to read-dominant blocks account for no more than approximately 6% of all accesses to the internal block storage, and writes to these blocks account for no more than around 3% of writes. These trends make a strong case that write-optimized STT-MRAM-cells should be used to create optimized non-volatile write-buffers, and

that conventional read-optimized or balanced cells would be problematic as the sole (or even common) STT-MRAM-cell.

Impacts of Write-Buffer Block Eviction-Triggered Reads

Since the write-dominant region has a significantly stronger growth in the aggregate amount of writes as write-to-read ratio increases beyond 5x (approximately 25% compared to effectively no growth in previous traces), this section discusses the impacts of the reads required to evict blocks from the write-buffer. One difference between the internal storage of a write buffer and the external access streams is that when blocks are evicted from the write-buffer they must first be read from the storage, since the data is dirty and must be written back. This means that blocks that were naturally write-only would now experience a read, and thus blocks with effectively infinite (>100x) write-to-read ratios could have ratios less than 100x. In order to identify if these 'extra' reads are resulting in the greater distribution of writes in the region of write-to-read ratios that are greater than 5x, complete profiles for this stream with with eviction-triggered reads and without them are plotted side-by-side in Figure 3.10. The subplot for profiles including eviction-triggered reads (a) is the same plot as was shown previously in Figure 3.8. The subplot for profiles excluding eviction-triggered reads (b) is plotted in the same format. Block count PDF and CDF (normalized to the total number of blocks within SieveStore-C) are plotted (left Y-Axis with solid lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin. Read, write, and combined access counts (normalized to the their respective totals) are plotted (right Y-Axis with dashed lines) against write-to-read ratio bins inclusively from 0.0x to 100x (increments of 0.01), with values for ratios greater than 100x includedsi in the 100x bin.

When eviction-triggered reads are ignored, the familiar flat region for each curve (for ratios between 5x and 100x) returns, along with markedly more extreme trends



(a) With eviction-triggered reads

(b) No eviction-triggered reads

Fig. 3.10.: Write-buffer internal block storage access-stream write-to-read ratio profile without eviction triggered reads

regarding the strongly and extremely write-dominant blocks. The first of these shifts is that, more than 88% of blocks are 'write-only' when eviction-triggered reads are ignored (compared to less than 5% when they are counted). This trend is important because for these blocks reads are only occurring when the data is no longer locally needed after the read, and thus should safely allow the use of STT-MRAM-cells that were write-optimized so far that reads became destructive. The second trend is that effectively 100% of reads (without counting eviction reads) happen to blocks with W/R ratios less than or equal to 2x, which in total account for less than 7% of the blocks in held by the write buffer. This is more clearly visible in Figure 3.11, which is a zoomed version of the prior plot where the x-axis is zoomed to just display the data for blocks with write/read ratios less than or equal to 5. Additionally, the percentage of blocks that are write-dominant (write-to-read ratios $>$ 1x) is now over 95%. And lastly, more than 55% of writes and effectively 0% of reads are to strongly write-dominant blocks (ratios larger than 2x), and 72% of writes occur to write-dominant blocks. These trends indicate strong opportunity for STT-MRAM-cells designed to optimize density, even if this write-focused optimization results in significantly worsened read performance or increased risk of destructive reads.

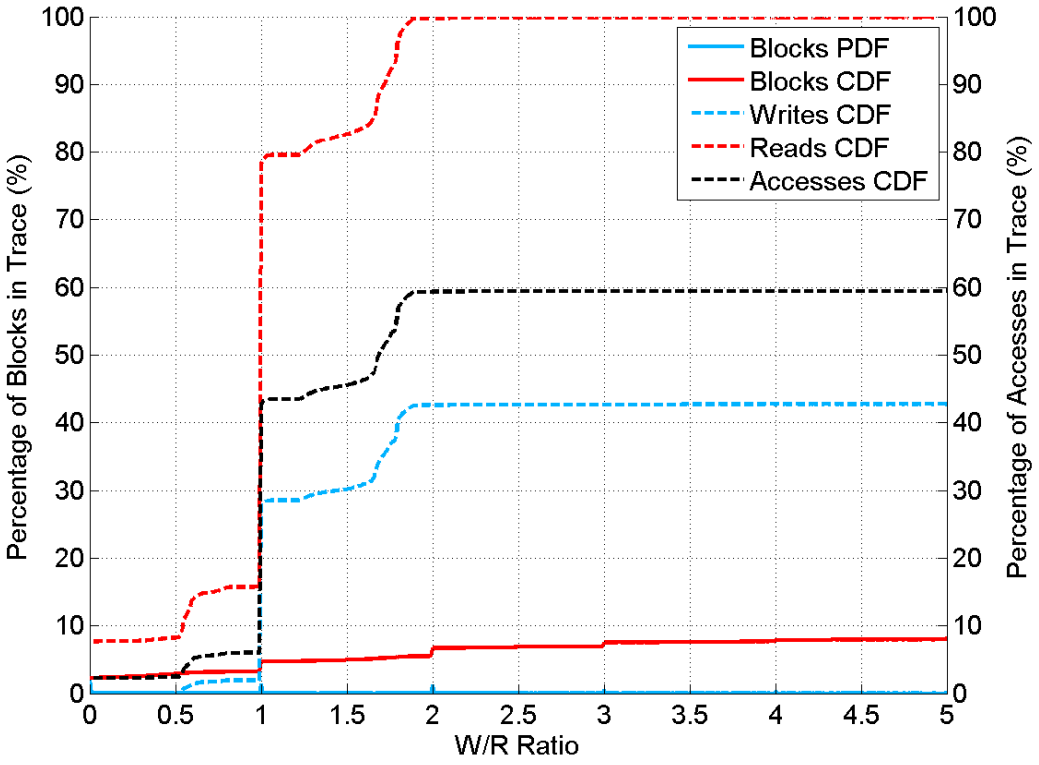


Fig. 3.11.: Write-buffer internal block storage access-stream write-to-read ratio profile without eviction triggered reads (zoomed to Ratios less than 5x)

3.4 STT-MRAM Reduced Retention-Time Trade-off

STT-MRAM cells are typically designed with a 10+ year retention-time allowing them to be handled as a general purpose long-term non-volatile storage media. However write-buffers are intermediate storage with the purpose of aggregating writes over time to locations to minimize the amount of individual writes to the underlying bulk storage, and thus would see little benefit from a 10+ year retention time due to the constant overwriting that occurs. Furthermore, Section 3.3 proved that write buffer activity is also largely insensitive to read performance given that writes make up the super majority of accesses and that nearly all reads occur to minuscule portion of the blocks held by the write buffer. However, based on this trend one could also design the write buffer to use traditional STT-MRAM for a very small portion of capacity that is selected based on write to read ratios to capture any potential upstream benefit from non-reduced read performance to those blocks. Although this added complexity in buffer design would have only minor impact to the overall SSD Cache, since the reads served from a 256MB (2% SSD capacity) write-buffer account for only 16% of the reads handled by the SSD Cache as a whole, and even the reduced read performance in a write-optimized STT-MRAM cell would remain orders of magnitude better than SSD read performance.

3.4.1 Benefits of Reduced Retention Time

Shrinking the retention time of a cell results from a few cell design changes (1) fundamental changes to the MTJ layers (2) shrinking of the thickness of MTJ layer(s) (3) shrinking of the planar area of MTJ layers(s) and thus the footprint of the MTJ stack. The first option is not of interest because it would also mean fundamental fabrication changes which if possible would be costly and result in large fabrication differences between current production and a proposed write-buffer optimized one. Both the second and third options result in reduce necessary write energies and thus write power or write performance. However only the third option allows for changes

to cell density as a result of changes in retention time, as the adjusted dimension could be chosen to align with the width of the access transistor below the MTJ stack and thus could result in density gains if the access transistor could likewise shrink. As discussed in Section 3.3, this last requirement is shown to be possible due to the write dominated access behavior and purpose of write buffers.

According to Equation 3.5 in Section 3.2.2, if only the planar area of the MTJ is adjusted, then reduction in planar area would be directly proportional to the reduction in thermal barrier. Since the width of the MTJ planar area is the dimension that aligns with the width of the underlying access transistor, if only it and the width of the access transistor are reduced in tandem then the reduction in thermal barrier would result in a directly proportional reduction in STT-MRAM cell size.

Furthermore, the critical write current is directly proportional to the MTJ planar area (as shown in Equation 3.6 in Section 3.2.2). This means that reductions the MTJ planar area also directly reduce the critical write current needed for any given write duration, which means the minimum width of the access transistor needed to support the write current for a given write time reduces directly with the MTJ planar area. This together with the write dominance of the write buffer activity mean that the access transistor can be safely scaled directly with the planar area of the MTJ and thus fully retain any density gains resulting from the planar area reduction method for reducing cell retention time. Although it should be noted that while my work relies on these previously validated scaling models [35, 40–43], independent layout and simulation of my proposed scaling may be necessary to re-validate my design to address end-to-end scalability (including wires) and process variation concerns.

3.4.2 Costs of Reduced Retention Time

When reducing the STT-MRAM cell retention time, it will likely be undesirable to limit the reduction to completely cover the longest natural lifespan of data to be held as this would likely result in very limited reduction due to only a few pages

having abnormally long data lifespans. Instead, these abnormally long lifespan pages should be handled via a expiration management scheme such that the STT-MRAM cell retention time reduction can be based on capturing a significant majority of the natural data lifespans of pages. There are two primary schemes for managing the expiration of these longer lifespan pages (1) refreshing the page before it expires and (2) evicting that page down to the underlying SSD. Furthermore, given that SSD caches may experience shorter term (few hours or less) downtime and crash induced restarts for a variety of reasons, it is necessary to have either of the above policies perform their operations early enough to ensure a desired "reserve" retention time window. This timing, which is illustrated in Figure 3.12, ensures that the data would remain durable during machine downtime or restart cycles of up to this "reserve" duration, although it will increase the respective costs associated with both approaches.

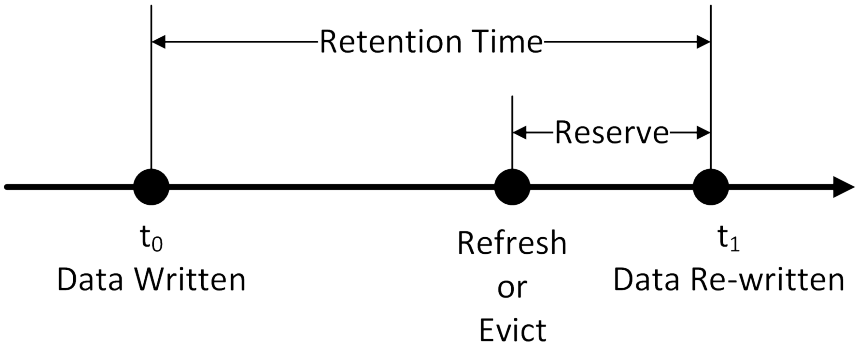


Fig. 3.12.: Timing Diagram Illustrating Timing for Ensuring Reserve Durability

Due to the nature of refresh cycles needed in this application, the extra costs for option 1 should be tolerable at worst and potentially could be minimal. Page refreshes are currently already done as periodic bulk refresh cycles at the hardware level for DRAM given it's cells have retention times in the 10s of milliseconds. Reduced retention time STT-MRAM should still be in scope of minutes to hours of retention time in order to be a suitably durable storage for write buffers and so the refreshing cycles would need happen far less often. Additionally, with the retention time chosen to handle the bulk of the natural data lifespans of write buffered pages, there should

only be a small subset of the pages that need to be refreshed. As a result selectively refreshing only the pages that need it could likely be done by the software managing the write buffer, given that it would be relatively easy and inexpensive to track the time since the prior write and refresh as needed. This tracking would simply require the addition of a timestamp added to each page's meta data that is updated on each write, followed by periodically check these stamps starting with the oldest first and refreshing the page if it is approaching expiration. For reference, WriteGuard and LRW buffers keep the metadata for all currently resident blocks in a LRW queue so there would be no additional sorting cost required to start with the least recently written page and only refreshing the subset that would need it. There would be an overhead cost in terms of extra buffer media accesses resulting from the software managed refresh, as illustrated in Figure 3.13. Although one strong benefit of this approach is that there would be zero impact to SSD lifespan increases that result from the write buffer's filtering.

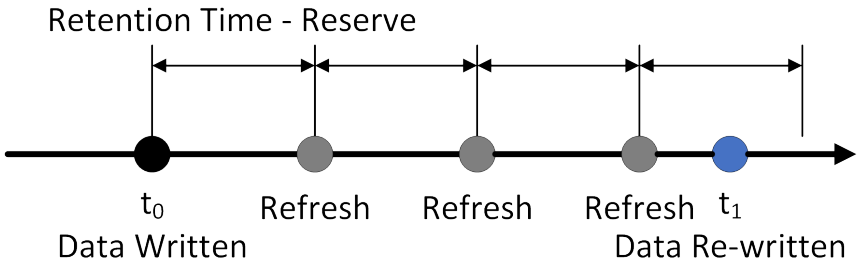


Fig. 3.13.: Timing Diagram Illustrating Timing for Selective Refresh Related Activity

The second option would have minimal extra costs to buffer runtime but could result in significant reductions in the write buffer's effective filtering, and thus lessen the SSD lifespan benefits it provides. This scheme is depicted in Figure 3.14, where each page approaching it's expiration would need to be written back downstream to the SSD to ensure no data loss. This would require the same additional metadata and similar extra metadata processing as the first scheme to identify pages that need to be written back, but would only require one additional read to the physical

page instead of potentially repeated read and write cycles required for a refresh. However, once evicted and written back to the SSD any future access to the block would be bypass misses until the block was reallocated to the buffer for selectively allocating buffers. This means that the expiration based eviction could result in more extra downstream writes than just the one corresponding to the expiration triggered eviction. In WriteGuard's case, the victim meta data buffer would help lessen this by allowing reallocation on the next write for any evicted pages with meta data still in this small buffer.

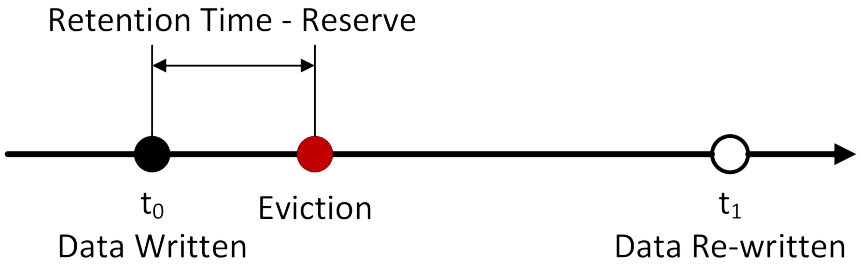


Fig. 3.14.: Timing Diagram Illustrating Timing for Expiration based Eviction Activity

3.5 Methodology

3.5.1 Reference Access Traces

Four block IO access traces, each at subsequent levels of the internal architecture of SieveStore+ staging cache from Chapter 2, were analyzed for the access-stream profile results presenting in Section 3.3. The first two traces in the sequence, the full SieveStore+ internal access-stream and the access-stream between the LRU RAM cache and WriteGuard instance, are the same ones captured during the work covered in Chapter 2. The third and fourth traces were captured during this work and are the logical access-stream and the physical access-stream to the data store component of a WriteGuard write-buffer instance and it's extraction is discussed in the following Section 3.5.2.

For ease of reference, the parameters for the SieveStore-C aspect of the SieveStore+ instance in Table 3.1, as well as a copy of the summary of the base storage ensemble trace that the analyzed traces are derived from is provided in Table 3.2. Additionally, a copy of the SieveStore+ system architecture diagram (Figure 3.15) has been included here for easier reference to the respective levels of the traces.

Table 3.1.: SieveStore-C SSD Cache Parameters

Parameter	Value
Window Span	8 Hours
Number of sub-windows	4
Imprecise Filter Size	500 Million Slots
Imprecise Filter Threshold	9
Precise Filter Size	5 Million Slots
Precise Filter Threshold	2
Cache Capacity	3Mi Pages (12 GiB)

Table 3.2.: Summary of Reference Ensemble Trace Set

Key	Name	Volumes	Drives	Size (GB)
Usr	User home dirs	3	16	1367
Proj	Project dirs	5	44	2094
Prn	Print server	2	6	452
Hm	Hardware monitor	2	6	39
Rsrch	Research projects	3	24	277
Prxy	Web proxy	2	4	89
Src1	Source control	3	12	555
Src2	Source control	3	14	355
Stg	Web staging	2	6	113
Ts	Terminal server	1	2	22
Web	Web/SQL server	4	17	441
Mds	Media server	2	16	509
Wdev	Test web server	4	12	136
	Total	36	179	6449

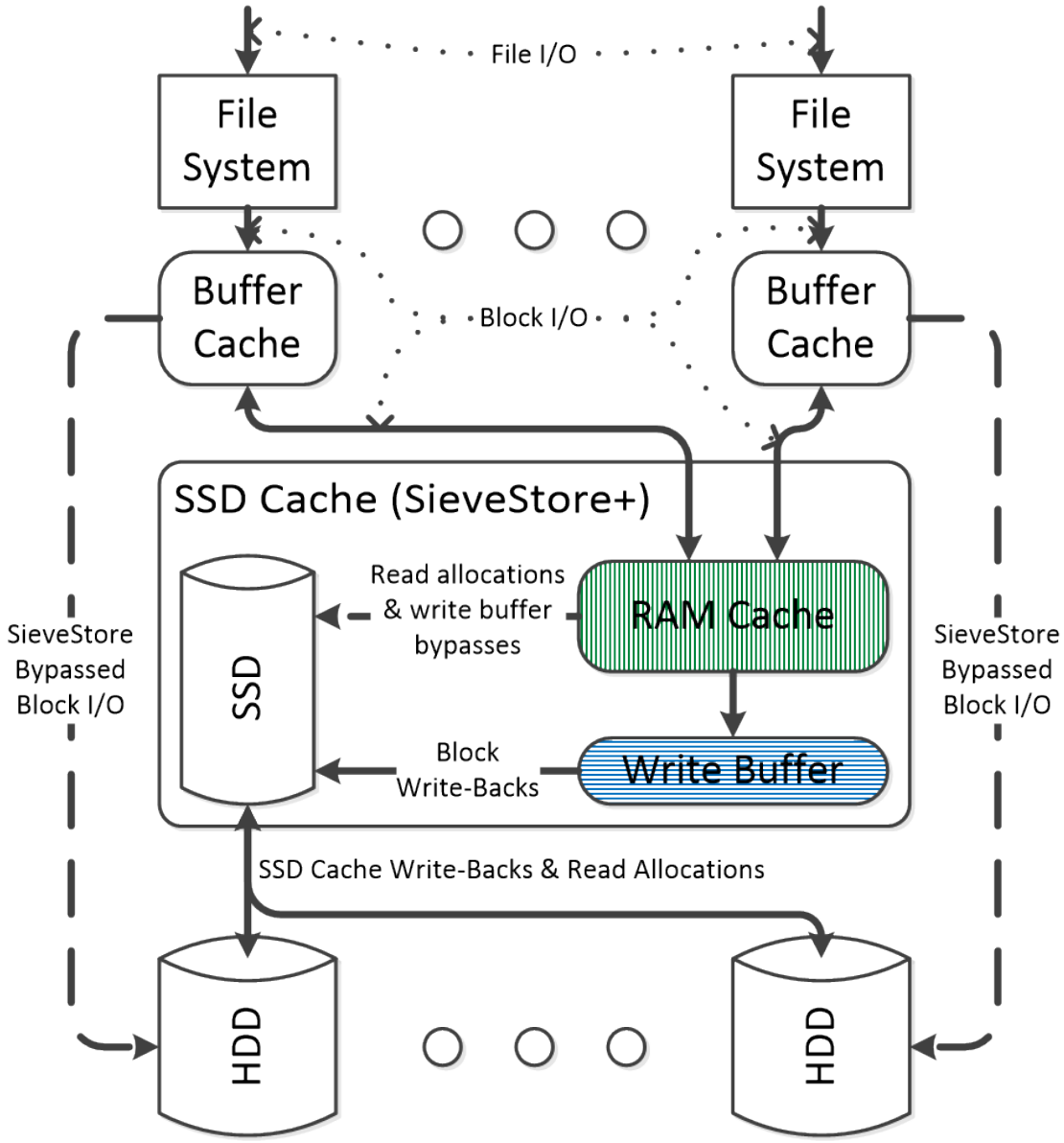


Fig. 3.15.: Reference SSD Cache System Organization (As presented in Chapter 1)

3.5.2 WriteGuard Instance for Write-Buffer Internal Access-Stream

The two access-stream traces that were captured for this work (the third trace analyzed in both Section 3.3 and Section 3.6, and the fourth trace analyzed in Section 3.6) were for the accesses to the data buffer internal to the write-buffer in the SieveStore+ system in Figure 3.15. The logical access-stream trace (the third trace) contained the accesses to the internal data store of the write buffer using the logical block addresses provided by the trace inputted to the write buffer. The physical access-stream trace (the fourth trace) contained the accesses to the internal data store of the write buffer based on the allocation to the internal pool of available blocks which represents the physical memory blocks that the write buffer would have. This trace was captured by activating the internal access logging features of the WriteGuard buffer design code, from the work in Chapter 2, for a WriteGuard instance with the parameters recorded in Table 3.3. The capacity of 256 MiB was chosen because (1) beyond this capacity all designs suffer overwhelmingly diminished returns for added capacity and (2) WriteGuard, LFWI, and LRW all have similar write-dilution effectiveness at this capacity allowing the trends analyzed for WriteGuard to be predictive for the other designs as well. The first point is important because higher capacities and higher but less-selective capture rates would result in higher read accesses and thus offer a conservative view of opportunity for write-dominance focused cell utilization, in addition to creating a more challenging cell density constraint. Regarding the second point, similar capture rates for the same capacity between LRFU and LRU algorithms would imply that similar data trends are being captured given the overlap in utilized information, resulting in the only major difference in internal block trends stemming from WriteGuard's sieved nature lessening the number of write-only blocks (which would be favorable for write-optimized STT-MRAM) due to minimizing buffer allocations. The combination of these two properties allows this single trace to serve as a conservative opportunity analysis for LRFU and LRU based buffer designs for various capacities.

Table 3.3.: WriteGuard Write-Buffer Reference Instance Parameters

Parameter	Value
Window Span	2 Hours
Number of sub-windows	4
Initial Sieving Threshold	1
Initial Bypass-Ratio Target	90%
Buffer Capacity	64 Ki Blocks (256 MiB)

3.5.3 Trace Analysis

The custom IO trace profiling tool developed and used during the work described in Chapter 2 was extended and used to perform all of the access stream analysis discussed during Section 3.3 and Section 3.6.

3.6 Results

There are three main conclusions from my analysis.

1. There is significant opportunity for cell density gains from reducing cell retention times via MTJ planar area reduction along with access transistor width reduction. (Section 3.6.1 & Section 3.6.2)

2. The expiration of pages with natural lifespans longer than the reduced cell retention time should be managed via selectively refreshing those pages instead of eviction. (Section 3.6.3)

3. Data expiration management can be done via software. (Section 3.6.4)

3.6.1 SieveStore-C Internal and Post-Screening Cache Data Lifespan Trends

In this section I discuss the results from analyzing the naturally occurring data lifespans of page writes to the SieveStore-C and those downstream from the small LRU screening cache. I analyzed the time gaps between writes to the each of the pages present within the first two block IO traces that were captured during the work in Chapter 2. The data from this is plotted in Figure 3.16, with the percentage of write-gaps shorter than or equal to a gap length (normalized to the total number of write-gaps, Y-axis) plotted against the various gap lengths (in microseconds) (X-axis). The X-Axis is plotted on base-10 logarithmic scale given that the Retention time for a cell is exponentially related to the cell's thermal barrier, necessitating orders of magnitude reduction in required retention time to result in appreciable changes to MTJ size and their corresponding cell size change. The blue curve plots the CDF of the data from the trace containing all of the block IOs cached by the SieveStore-C instance, and the red curve plots the CDF of the data from the trace containing all of the block IOs downstream from the internal screening cache. Given that these traces cover the span of 1 week the horizontal axis is limited to $10^{12}\mu s$ (11.6 days) and the curves stop short of this limit because it would not be possible to have a gap

that long. The next useful major step in gap length is $10^{11}\mu s$ (27.8 hours), which is long enough to accommodate 97.65% of all per-page write-gaps in the baseline trace and 97.15% of those in the screened trace. Going further to the left to gap lengths of $10^{10}\mu s$ (2.78 hours) is long enough to accommodate 82.72% of all per-page write-gaps in the baseline trace and 78.02% in the screened trace. After this point the percentage of write-gaps shorter than a given size decreases rapidly with only 32.73% having gaps shorter than $10^{9}\mu s$ in the baseline trace and 20.97% having gaps shorter than that length in the screened trace. Therefore, the realm of potential retention-time reduction for write-buffer targeted STT-MRAM cells should be in the $10^{11}\mu s$ range for a more conservative design point and around $10^{10}\mu s$ for a more aggressive design point. However, analysis of the traffic within the write buffer (discussed in Section 3.6.2) is still needed to solidify the value of these design points, as some of the accesses in the post screening trace will not be captured by the write buffer and buffer evictions and re-allocations could also result in changes to per-page write-gap lengths.

Additionally one can observe that nearly all gaps shorter than $10^{8}\mu s$ (1.67 minutes) and most of the gaps shorter than $10^{9}\mu s$ (16.67 minutes) that occur in the cache at large (blue curve) do not occur downstream of the screening cache. This along with only minor reductions the relative presence of longer gaps downstream of the screening cache indicates that the screening likely hold blocks with data-lifespans nearly completely in these shorter lengths, and thus could leverage even greater reductions in cell retention time than the downstream write buffer will be able to. Although power loss durability requirements would still impose limits to the useful retention time reduction such that it doesn't shrink below minutes, otherwise even minor power blip based machine restarts would result in data loss. However given it's tiny capacity, density benefits from write-focused reduced retention time STT-MRAM cells would be minor to system cost. Furthermore, as illustrated in Section 3.3.1 and Section 3.3.2 this screening cache experiences a significant higher ratios of reads to writes than the downstream write buffer, and thus would likely be better off with more traditionally

read-focused STT-MRAM, such as those proposed by [39] given it's high read activity for it's tiny capacity.

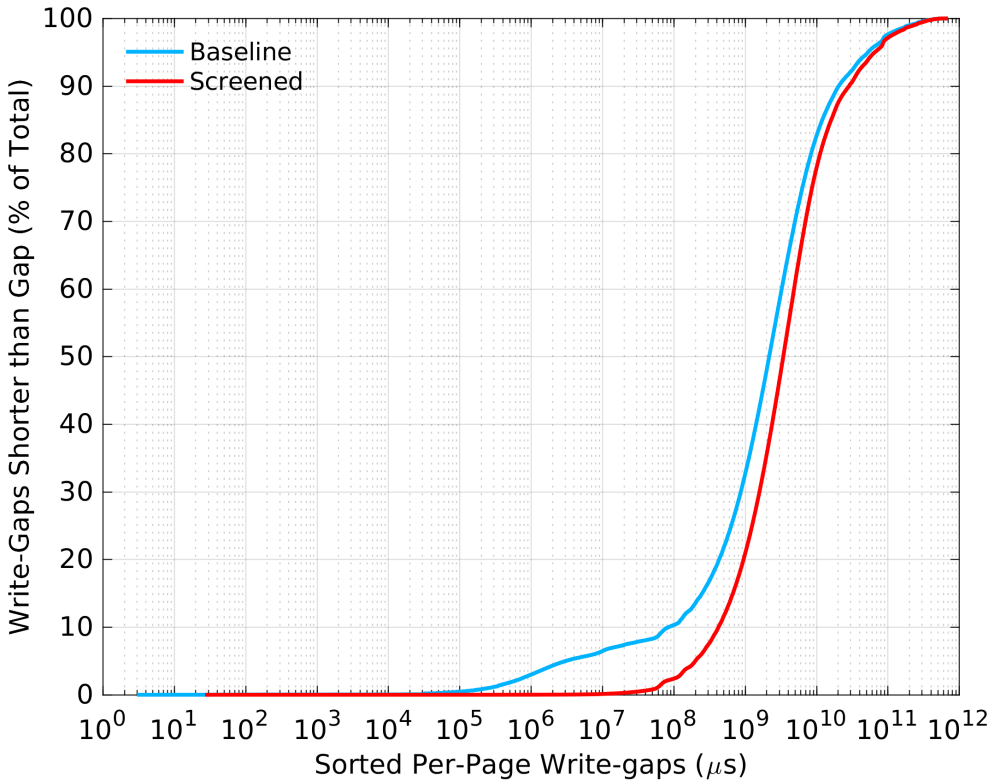


Fig. 3.16.: Natural Data Lifespans for logical pages within SieveStore-C and downstream of the internal screening cache

3.6.2 Write-Buffer Internal Data Lifespan Trends

In this section I discuss the results from analyzing the naturally occurring data lifespans of page writes to within the write buffer. I analyzed the time gaps between writes to the each of the pages present within the two traces captured for this work, with the first of the two being for the logical page activity and the second one being for the physical page activity within the write buffer. The data from this is plotted in Figure 3.17, with the percentage of write-gaps shorter than or equal to a gap length (normalized to the total number of write-gaps, Y-axis) plotted against the various

gap lengths (in microseconds) (X-axis). The X-Axis is plotted on base-10 logarithmic scale given that the Retention time for a cell is exponentially related to the cell's thermal barrier, necessitating orders of magnitude reduction in required retention time to result in appreciable changes to MTJ size and their corresponding cell size change. The blue curve plots the CDF of the data from the trace containing all of the physical page IOs within the write buffer, and the red curve plots the CDF of the data from the trace containing all of the logical page IOs within the write buffer. Given that these traces cover the span of 1 week the horizontal axis is limited to 10^{12} microseconds (11.6 days) and the curves stop short of this limit because it would not be possible to have a gap that long. Additionally, the X-axis is kept in terms of microseconds for easy of comparing with Figure 3.16 but limited to gaps greater than or equal to 10^7 microseconds (10 seconds) since shorter retention times would be useless for a durable write buffer.

One trend that stands out from this plot, is that the physical access CDF is slightly shifted to the right of the logical access CDF, meaning that slightly larger portions of the accesses tend to have longer gaps than in the logical access trace. This counter-intuitive trend results from two characteristics of the write buffer implementation (WriteGuard), (1) in selective buffers (WriteGuard) churn is minimized so there is minimal to no artificial slicing or shrinking of natural access gaps due to intermediate eviction and reallocation to the buffer and (2) physical pages get reallocated to new logical pages after the prior logical page is evicted (which would tend to be long gaps when churn is low) resulting in additional gaps that are also long. For buffer implementations with higher churn rates (such as LRU or LFW-I) there would be both more artificial slicing of natural gaps from intermediate eviction and reallocation and the reallocation of physical pages to a new logical page would happen more often resulting in additional gaps that were shorter.

The more important trend present is that there are still similarly large portions of the per-page write-gaps that fall within both the conservative ($10^{11}\mu s$) and aggressive ($10^{10}\mu s$) design points previously identified in Section 3.6.1. Regarding physical page

activity, 98% of per-page write-gaps are at most $10^{11}\mu s$ (2.78 hours) resulting in a conservative retention time reduction by more than 3,153x and 80% of per-page write-gaps are at most $10^{10}\mu s$ (2.78 hours) resulting in a more aggressive retention time reduction by more than 31,536x.

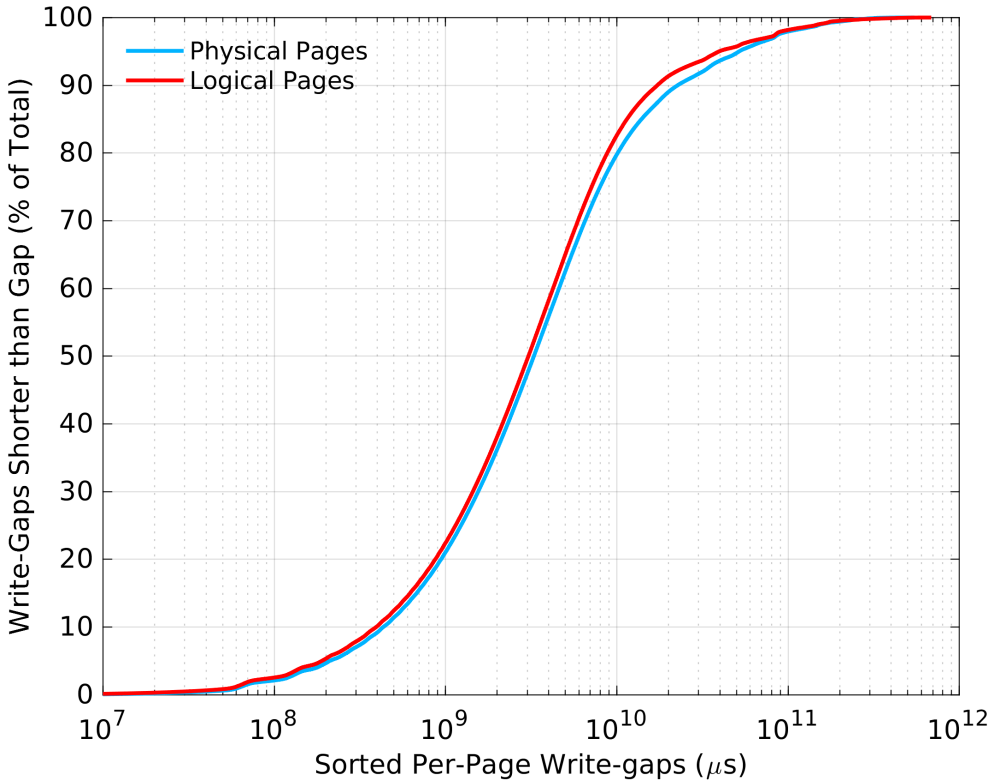


Fig. 3.17.: Natural Data Lifespans for logical and physical pages within the write buffer

Since smaller buffer capacities would naturally deal with proportionately more popular blocks, as well as higher churn rates, the natural per-page write-gaps will generally shrink some as buffer capacity decreases, resulting in more opportunity to benefit from reduced retention times. To evaluate the magnitude of this impact I also analyzed the activity to physical pages within write buffers of 128MB and 64MB capacities. The data from this is plotted in Figure 3.18, with the percentage of write-gaps shorter than or equal to a gap length (normalized to the total number of

write-gaps, Y-axis) plotted against the various gap lengths (in microseconds) (X-axis). The X-Axis is plotted on base-10 logarithmic scale given that the Retention time for a cell is exponentially related to the cell's thermal barrier, necessitating orders of magnitude reduction in required retention time to result in appreciable changes to MTJ size and their corresponding cell size change. The blue curve plots the CDF of the data from the trace containing all of the physical page IOs within the original write buffer with a capacity of 256MB (2% of cache capacity), the red curve plots the CDF of the data from the write buffer with a capacity of 128MB (1% of cache capacity), and the black curve plots the CDF of the data from the write buffer with a capacity of 64MB (0.5% of cache capacity). The X-axis is kept to have the same limits as in Figure 3.17. One clear trend is that even successive reductions in capacity by factors of 2x only have small increases in the relative portion of write-gaps that are shorter than the aggressive design point of $10^{10}\mu s$, with the 64MB capacity buffer having 90.13% compared to the 80% at the 256MB capacity point. Although this does mean that the conservative design point of $10^{11}\mu s$ becomes even safer of a design point, given that 99.68% of per-page write-gaps are shorter than it for the 128MB write buffer, and 99.94% are shorter than it for the 64MB write buffer. While it would not be practical to have different retention times for different buffer capacities due to reduced economies of scale benefits during production, this does mean that smaller buffer capacities would experience diminishing overhead costs and penalties that result from having to manage the expiration of pages with write-gaps longer than the chosen retention time for the STT-MRAM cells.

In order to better quantify the benefits of we need to apply the formulas from Section 3.2.1 and Section 3.2.2. From applying Equation 3.4, the conservative design point of $10^{11}\mu s$ (27.8 hours) results in a reduction of 20% to the required MTJ thermal barrier, and the aggressive design point of $10^{10}\mu s$ results in a reduction of 25.7%. Due to Equation 3.5, we know that reduction in STT-MRAM cell size is directly proportional to the reduction if MTJ thermal barrier when only scaling the planar area. Furthermore, from Equation 3.6 we know that the critical write current

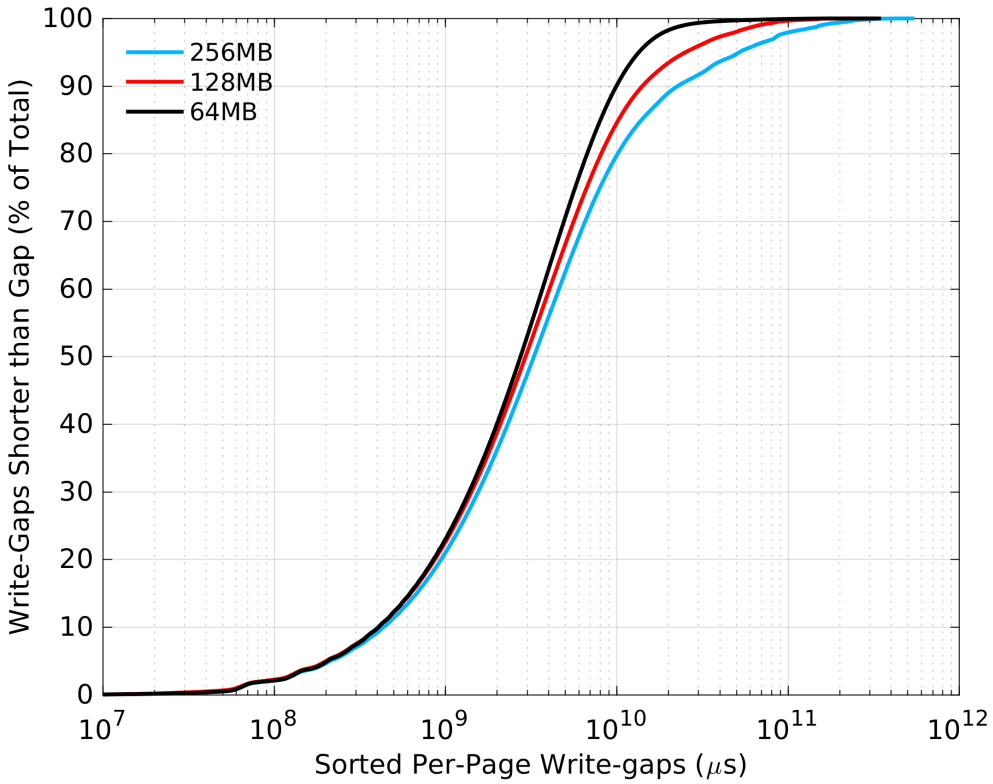


Fig. 3.18.: Comparison of Natural Data Lifespans for physical pages within the write buffers of different sizes

will also scale directly proportionally to the planar area reduction. From these we know that, if this thermal barrier reduction is achieved through MTJ planar area reduction in the dimension that matches the access transistor width, both can be safely scaled while maintaining the same write performance for the cell. Therefore, this coordinated reduction in MTJ planar area and access transistor width results in a cell size reduction of 20% for the conservative design point with a retention time of $10^{11}\mu s$ (27.8 hours) and a cell size reduction of 25.7% for the aggressive design point with a retention time of $10^{10}\mu s$ (2.78 hours).

Additionally, actual cell retention time design points do not need to precisely match the identified design points in order to retain the vast majority of the cell size reductions, due to the exponential relationship between retention time and MTJ

thermal barrier. For an aggressive design point example, the blue (256MB) curve in Figure 3.18 shows that 89.01% of per-page write-gaps to physical pages within the 256MB capacity write buffer only need a retention time of at most $2 * 10^{10} \mu s$ which would result in a thermal barrier and corresponding cell size reduction of 24%. For an ultra conservative design point example, the same curve also shows that increasing the conservative design point retention time to $2 * 10^{11} \mu s$ results in only 0.5972% of per-page write-gaps not being naturally covered by the cell's retention time and still yields a cell size reduction of 18.3%.

3.6.3 Impact on SSD Lifespan Improvements from Data Expiration

In this section I discuss the impact that the data expiration management schemes discussed in Section 3.4.2 have on the overall write buffer effectiveness and thus lifespan improvements for the downstream SSD. I analyzed the amount of write traffic going downstream from the write buffer to the SSD. This analysis does not include extra writes due to potentially extra bypass writes from the write buffer no longer having expiration-based evicted page when subsequent writes occur to that page, and thus is a best-case view of the impact that expiration-based eviction would have. The data is plotted in Figure 3.19 with the relative lifespan improvement for the SSD (normalized relative to the SSD activity without a write buffer, Y-axis) plotted against the various cell retention times in microseconds (X-axis). The x-axis is limited to values greater than $10^9 \mu s$ (16.67 minutes) given the marginal amount of natural per-page data-lifespans that would be satisfied by cell retention times smaller that that. The impact resulting from the selective refresh scheme is plotted for the three write buffer sizes previously analyzed using the dashed curves, with blue for 256MB capacity, red for 128MB capacity, and black for the 64MB capacity. The impact resulting from the expiration-based early write-back approach for these write buffer capacities is plotted using the solid curves, with blue for 256MB capacity, red for 128MB capacity, and black for the 64MB capacity. The selective refresh scheme

has no impact on the write buffer effectiveness since it, by design, results in no extra writes down stream to the SSD. The expiration-based early eviction approach experiences increasingly significant reductions to the write buffer effectiveness and resulting SSD lifespan improvements, due to the relatively large amounts of extra downstream writes that result from the write-backs (discussed in more detail in Section 3.6.4). The conservative design point experiences a drop from 5.55x improvement to 5.06x for the 256MB buffer although it experiences effectively no drop for the smaller buffer sizes, due to the smaller sizes having write-gaps that are naturally nearly 100% accommodated by the retention time. The more aggressive design point experiences a drop from 5.55x to 2.85x for the 256MB write buffer (which is a stronger impact than reducing capacity by 2x), a drop from 3.53x to 2.47x for the 128MB one (which is similar in impact to reducing capacity further by 2x), and a drop from 2.46x to 2.13x for the 64MB one. Ultimately this indicates that even a expiration-based eviction should only realistically be considered if the retention time is chosen based on a very conservative point where only a few percent of per-page data lifespans would not naturally be accommodated by the retention time. Whereas, selective refreshing of pages nearing expiration has not impact on SSD lifetime improvements regardless of capacity and cell retention-time and thus is the better approach when the chosen cell retention time does not naturally accommodate nearly all of the natural data-lifespans.

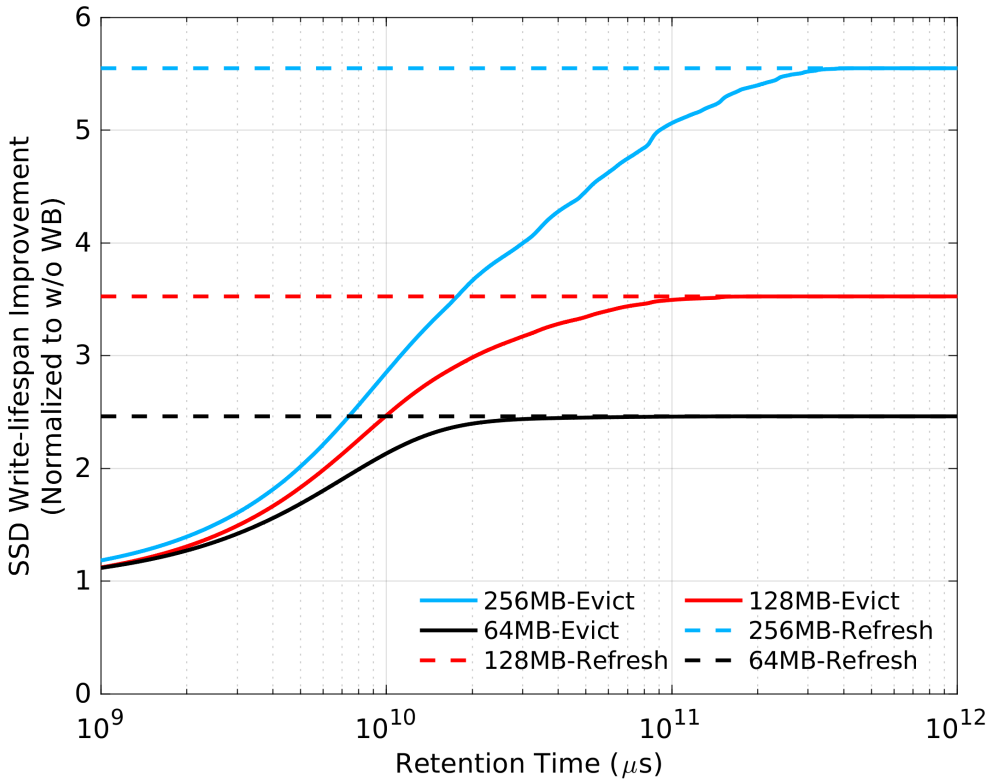


Fig. 3.19.: Comparison of SSD Lifespan Improvements for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes

3.6.4 STT-MRAM Cell Retention Time Trade-off Costs

In this section I show the results from analyzing the costs of managing the expiration buffered pages with natural data lifespans longer than the buffer's STT-MRAM cell retention time. The first cost analyzed is the additional writes sent downstream to the SSD which is the reason for the impacts to the SSD lifespan improvement discussed in Section 3.6.3. For this analysis the number of pages that would expire due to various cell retention times was calculated from the natural data lifespans extracted from the trace of activity to physical pages with the write buffer for the three capacities previously analyzed (256MB, 128MB, and 64MB). This was then compared with the amount of downstream writes if there were no retention time related downstream rights. The data is plotted in Figure 3.20, with the extra expiration-based writes (normalized as a percentage of non-expiration related writes, Y-axis) plotted against the various potential cell retention times (X-axis). The x-axis is limited to values greater than $10^9 \mu s$ (16.67 minutes) given the marginal amount of natural per-page data-lifespans that would be satisfied by cell retention times smaller that that. The blue curve plots the data for the 256MB write buffer, the red curve plots the data for the 128MB write buffer, and the black curve plots the data for the 64MB buffer. One trend that stands out is that reduction in capacity noticeably shifts the curves to the left, as should be expected given the previous plotted increasing amounts of blocks that can be accommodated by shorter retention times. However one non-obvious aspect about this trend is the magnitude of the shifts, which is far larger than the relative shifts in required retention time shown previously in Figure 3.18. This change is due to the fact that smaller write buffers have lower hit rates and thus filter out less writes downstream which further minimizes the impact of their also smaller amount of extra writes (relative to the number of extra writes for larger sizes). For example, the highly effective 256MB capacity buffer (5.55x baseline lifespan increase) experiences 9.60% extra writes from expiration-based evictions with a retention time of $10^{11} \mu s$, while the 128MB capacity buffer (3.53x baseline lifespan increase) experiences around

0.884% extra writes from expiration-based evictions, and the 64MB capacity buffer experiences 0.099% extra writes from expiration-based evictions. This also illustrated in the Figure 3.19 by the combination of lower starting points and slower reduction of lifespan improvement with shorter retention times. The key trend from this data is the sharp rise in relative amounts of extra writes from expiration-based eviction as retention time is decreased due to the large amount of write-gaps approaching or longer than $10^{10}\mu s$ relative to the small amount of writes that not originally filtered out by the write buffer. A clear indicator of this is that the 256MB capacity buffer experiences 94.46% extra writes from expiration-based evictions for a retention time of $10^{10}\mu s$ while that design point only offers a additional raw 5.7% cell reduction benefit relative compared to the 20% benefit granted with a retention time of $10^{11}\mu s$ which only results in 9.60% extra writes from expiration-based evictions.

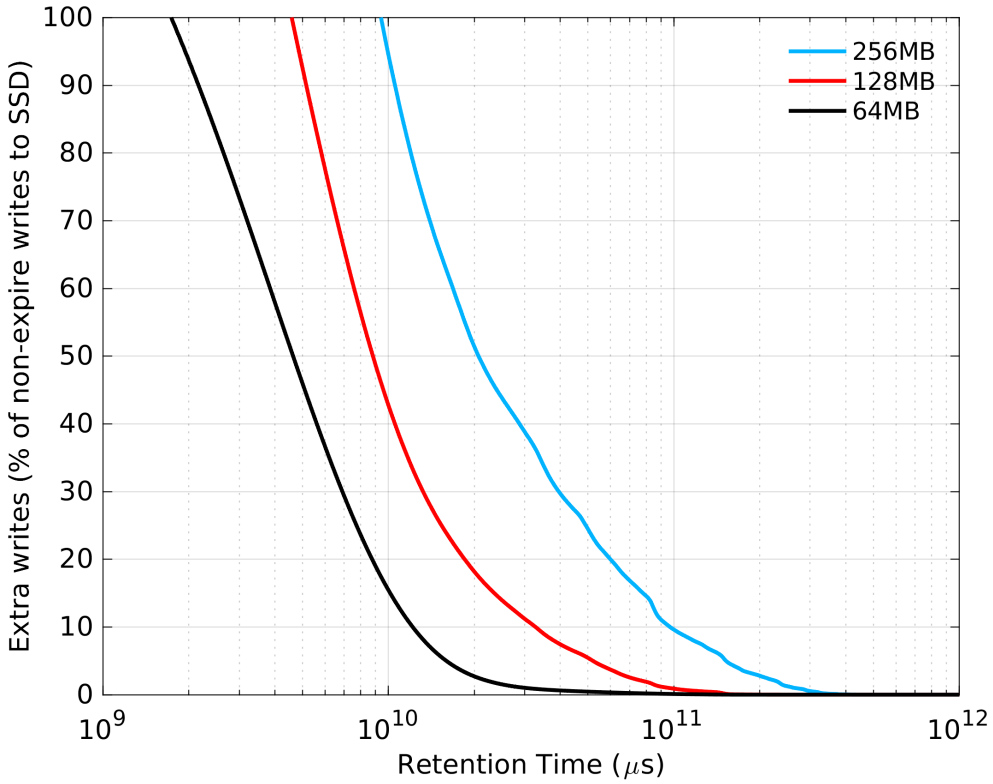


Fig. 3.20.: Comparison of extra SSD writes for write buffers using reduced STT-MRAM cell retention time with expiration eviction management scheme

The second and final cost to analyze is the cost of extra accesses to the write buffer's internal STT-MRAM media. For this analysis, the numbers of expiration-based eviction reads and the number of refresh cycles (read + write) for the various cell retention times were calculated from the natural data lifespans extracted from the trace of activity to physical pages with the write buffer for the three capacities previously analyzed (256MB, 128MB, and 64MB). This was then compared to the amount of reads and writes going to the buffer's internal data store in terms of write-hits (writes), read-hits(reads), allocations (writes), evictions (reads). This data is plotted in Figure 3.21 and Figure 3.22, with the extra data store activity (normalized as a percentage of the baseline activity, Y-axis) plotted against the various cell retention times (X-axis). The x-axis is limited to values greater than $10^9 \mu s$ (16.67 minutes) given the marginal amount of natural per-page data-lifespans that would be satisfied by cell retention times smaller that that. In these plots solid curves are used to plot the data for the selective refresh scheme for the three analyzed buffer capacities, with the blue curve for the 256MB capacity, the red curve for the 128MB capacity, and the black curve for the 64MB capacity. Additionally, one thing of note is that the jaggedness of the refresh curves is due to the computation needing to done after logarithmicly binning the per-page write-gap values prior to computation in order for it to be feasible to do. Dashed curves are used to plot the data for the expiration-based eviction scheme for the same three sizes, with the blue curve for the 256MB capacity, the red curve for the 128MB capacity, and the black curve for the 64MB capacity. Furthermore, Figure 3.21 includes extra activity percentages through 400% while Figure 3.22 zooms in to values up to 100% to allow a better view of the lower values for the conservative design point of $10^{11} \mu s$.

As was expected, the curves again shift left with reductions in capacity due to the increased percentages of page data-lifespans that would be naturally accommodated by the respective retention times and thus result in fewer refresh cycles or expiration based evictions. One key difference in the trend for the two approaches is that while the increase in extra activity is slow for the expiration-based evictions it increases very

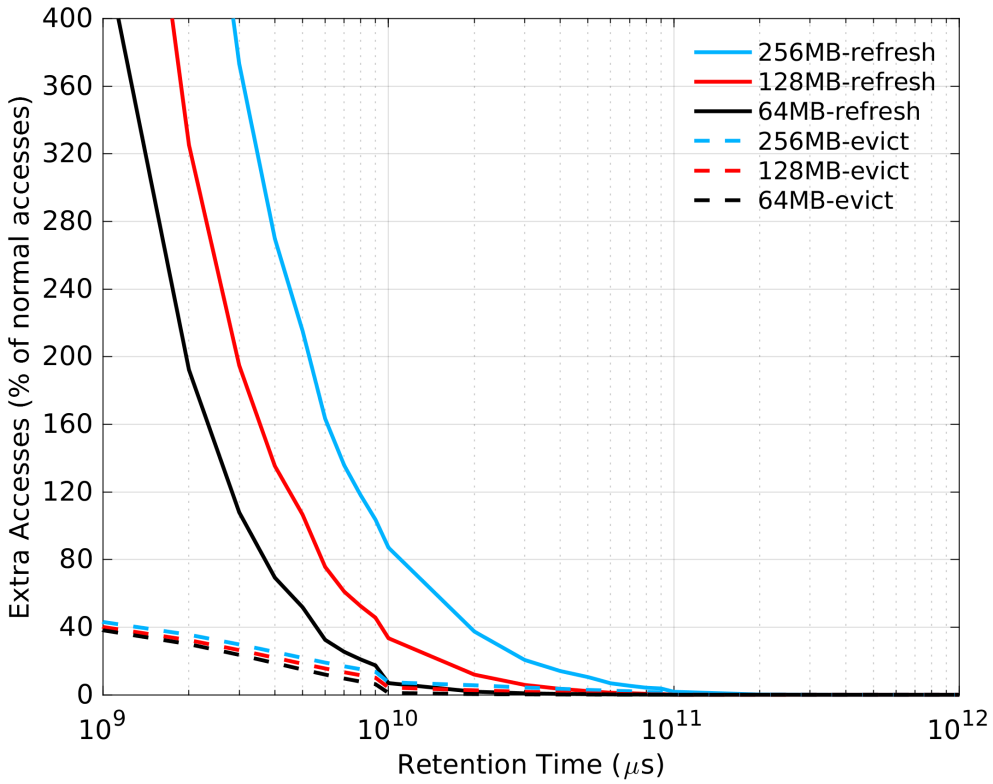


Fig. 3.21.: Comparison of extra STT-MRAM data access overheads for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes (Overheads $\leq 400\%$)

quickly for the selective refresh approach. This is because unlike the expiration-based eviction which only has 1 extra access (the read for eviction), the selective refresh requires an increasing amount of refresh cycles to prevent a page from expiring as the retention time decreases. As a result of this growing effect on needed refresh cycles, the 256MB capacity buffer experiences an refresh activity overhead of 87.05% for the aggressive design point of $10^{10}\mu s$ and a 1.77% overhead at the conservative design point of $10^{11}\mu s$. While a nearly 90% overhead seems terrible at first, data from the Chapter 2 showed that moving data in and out of the buffers data store represented only around a third of the overall runtime cost, so even a full doubling

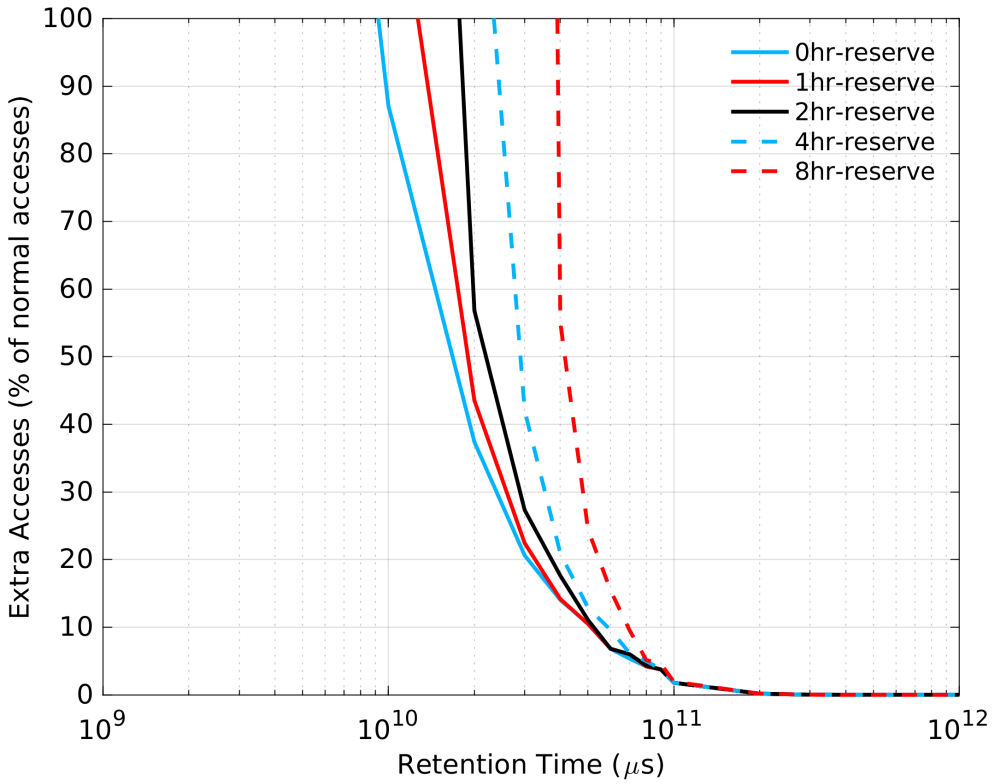


Fig. 3.22.: Comparison of extra STT-MRAM data access overheads for write buffers using reduced STT-MRAM cell retention time with either selective refresh or expiration eviction management schemes (Zoomed to Overheads $\leq 100\%$)

of data movement to or from the data store would only result in an overall runtime cost of around 33%.

Furthermore, it should be remembered that DRAM currently bulk refreshes all pages every 10 to 100 ms ($10^4\mu s$ to $10^5\mu s$), while even the aggressive design point is only refreshing a subset of pages every $10^{10}\mu s$ (10^5x slower rate than DRAM). Additionally, if the STT-MRAM chips were design to have an internal refresh command that could be triggered by the buffer there would be no extra data movement and the command overhead would be neg liable compared to the previously existing DRAM refresh performance penalties. Therefore, while the selective refresh scheme has additional overhead costs spanning between the design points and rapidly increasing

toward the aggressive design point, it is still low enough that the scheme could be managed in software by the write buffer implementation, especially if the hardware chips offered a block refresh command.

Since it is necessary to ensure a reserved duration of non-volatility for buffered data so that it will remain durable during short-term machine downtime or crash induced restart cycles, I have also analyzed the impact of refreshing pages such that certain reserve lifespans are preserved. The results of this analysis were calculated in similar fashion to the prior analysis with the only difference being that refresh interval used was reduced by reserve times of 1 hour, 4 hours, and 8 hours. This data is plotted in Figure 3.23 andFigure 3.24, with the extra data store activity (normalized as a percentage of the baseline activity, Y-axis) plotted against the various cell retention times (X-axis). The x-axis is limited to values greater than $10^9 \mu s$ (16.67 minutes) given the marginal amount of natural per-page data-lifespans that would be satisfied by cell retention times smaller that that. All curves in this plot are based on data from the 256MB capacity write-buffer instance, with the original curve with no reserved retention time included for reference. The inclusion of the extra reserved lifespan to account for machine downtime accelerated the increase in overhead in the more aggressive retention time ranges, as was expected, and had minimal impact near the conservative design point. Regarding the more aggressive design point, a one hour reserve increased the overhead from 87.05% to 129.30%. The curve for the four hour reserve time terminates at the $2 * 10^{10} \mu s$ point given the combination of the logarithmic binning and it being impossible for shorter times to provide the required reserve. And the eight hour reserve curve becomes nearly vertical at gap sizes of $4 * 10^{10} \mu s$. However, even the eight hour reserve only increase the overhead at the conservative design point of $10^{11} \mu s$ from 1.77% to 1.93%. Thus the conservative design point experiences negligible costs while providing a cell reduction of 20% and retaining durability during reasonably long machine downtime.

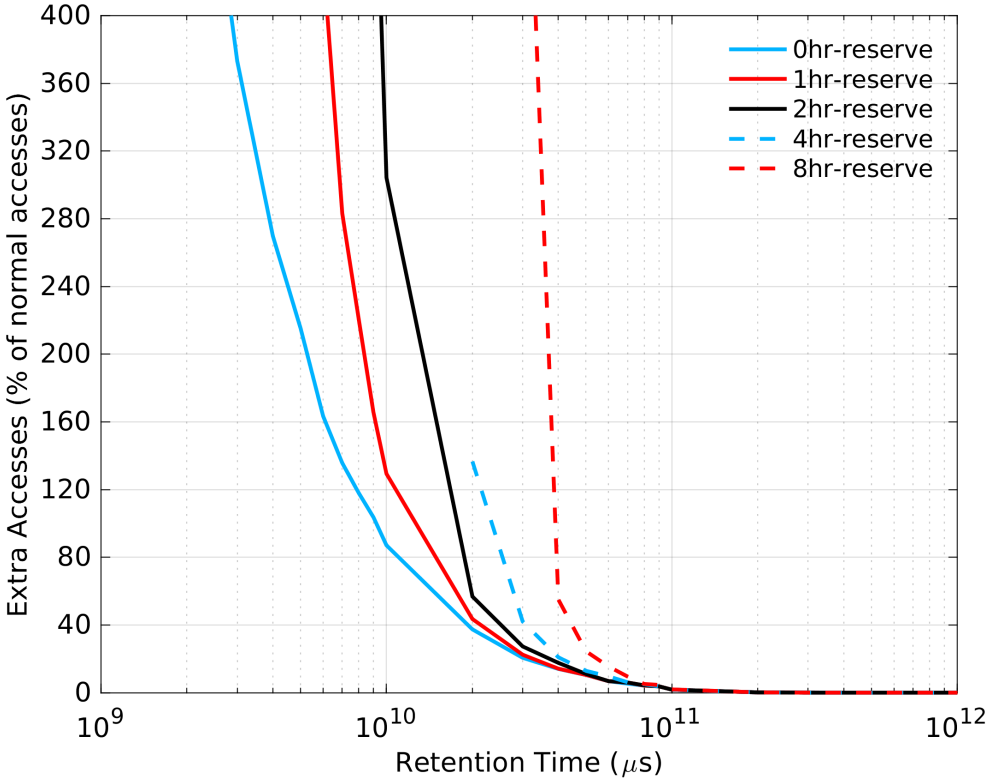


Fig. 3.23.: Comparison of impact of reserving extra page lifespan by early refreshing of pages (Limited to Overheads $\leq 400\%$)

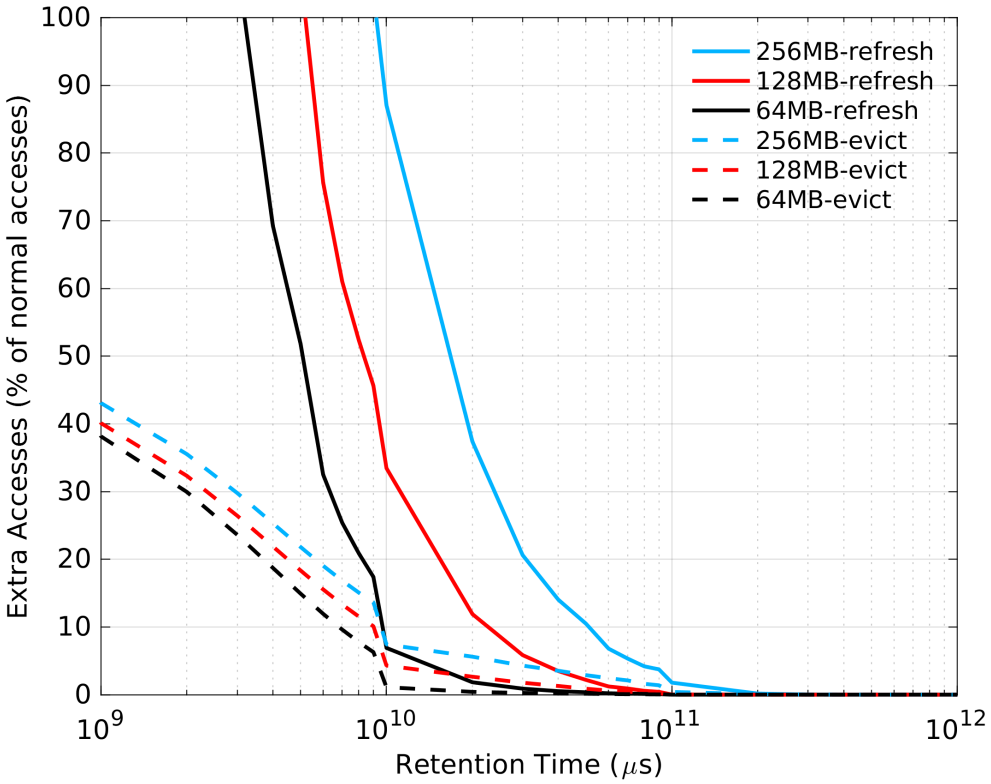


Fig. 3.24.: Comparison of impact of reserving extra page lifespan by early refreshing of pages (Zoomed to Overheads $\leq 100\%$)

3.7 Related Work

STT-MRAM has been gaining momentum as alternative to traditional memories due to STT-MRAM's non-volatility, higher read performance, and strong density scaling. As a result various prior works have tried to improve the problematic characteristics that have been limiting widespread adoption, such as write energy costs, write latency, and fabrication feature size scaling. One example is Perpendicular magnetized MTJ based STT-MRAM cell designs, where the magnetic alignment within the MTJ layers is perpendicular to the plane of the wafer, have shown improvements to write energy costs and enhanced density due to their better ability to scale to smaller feature sizes and higher temperature tolerances [35]. This design change is complementary to the non-volatility trade-off taken by my proposed write-buffer focused STT-MRAM, since the thermal barriers are also shown to remain directly proportional to their MTJ stack volume and thus the planar area that I propose to shrink. Additionally, [44] proposed STT-MRAM designs with tilted magnetic anisotropy which significantly improved write energy costs while maintaining or improving read performance, and thermal stability. This work is targeted at traditional workload focused STT-MRAM cells and involves cell design changes that would only be complementary to the non-volatility trade-off taken by my proposed write-buffer focused STT-MRAM, especially given their lower critical write currents. My work leverages volumetric scaling of MTJ stacks and reducing access transistor sizing below read-performance dictated limits that I show are artificial for write-buffer workloads. Therefore improvements to traditionally targeted STT-MRAM cell designs should in general be complementary to my proposed designs, unless they negatively alter the relationship between MTJ thermal barrier values and the MTJ planar area or would negatively impact access transistor size scaling relative to MTJ planar area scaling under write-dominated workloads.

Prior works have proposed using reduced retention time STT-MRAM cells in caches. [38] proposed designing on-chip caches using STT-MRAM with retention times

reduced to $56\mu s$ followed by DRAM style hardware refresh schemes in order to scale STT-MRAM cells from $32F^2$ to $10F^2$ as a way to reduce dynamic power of on-chip caches and have SRAM similar write times for the STT-MRAM cells. Unlike my work, [38] did not perform a study of data lifespans as their usage of reduced retention time was geared around using reduced retention time STT-MRAM as sort of high performance DRAM to replace SRAM for on-chip caches. Whereas, my work targets durable write buffering and only trades off the extraneous non-volatility that artificially limits density scaling for this high density workload. Furthermore, the designs proposed by [38] depend on baseline MTJ sizes of $32F^2$ where the MTJ is far larger than the size of the necessary access transistor as indicated by other works with optimized cell designs in the range of 2-3F^2 [40, 43]. [39] also proposed reduced retention times for on-chip STT-MRAM targeted to replace SRAM for lower level on-chip caches. Although these designs decreased the MTJ layer thickness in order to leverage the reduced retention time to lower the write power of the cells, since the optimized cells at that time were already had MTJ planar area limited by the size of the access transistor. While my work considers the same baseline optimized cell sizing limits as in this work, my work shows that this sizing limit does not hold for write buffers and thus leverages planar area reductions in MTJ and access transistor width reductions to achieve both write power and density benefits in this application space. Additionally, [39] uses a hardware refresh scheme to refresh pages just before they expire, while my work does via a software approach that is shown to be feasible in my target application space of software controlled write-buffers.

Given the asymmetric characteristics of STT-MRAM, other works [45–47] have proposed hybrid cache designs where STT-MRAM is used to hold the read-dominate data while using DRAM or SRAM for the write-dominate portions. While these designs allow for benefiting from the improved read performance and lessened dynamic power, they are not naturally durable designs and would still require power backup (although less than a full DRAM based design). Whereas my design is naturally durable for long enough to prevent data loss under typical power outage scenarios.

3.8 Conclusions

DRAM-based write buffering and caching techniques are commonly used to help alleviate write-pressure to SSDs, and thus over-provisioning costs, but require power-backup to ensure durability of buffered writes. Even the more efficient forms of these buffers require relatively large capacities on the order of 1GB for 120GB SSD capacity. One approach to avoiding the power-backup costs associate with these larger buffers is to employ Non-Volatile memory such as STT-MRAM as a drop-in replacement for the write-buffer's DRAM.

My contribution is to show that higher-density variants of STT-MRAM are usable in write buffers. More specifically, I identify two application characteristics – (1) the vast majority of the write-buffer's contents is composed of write-dominant blocks, and (2) the vast majority of blocks in the write-buffer are overwritten within a period of 28 hours – that enable a high-density, optimized STT-MRAM as a replacement for DRAM in write-buffers. My optimized STT-MRAM-based write buffer achieves higher density by (a) trading off superfluous durability by exploiting characteristic (2), and (b) deoptimizing the read-performance of STT-MRAM by leveraging characteristic (1). Together, the techniques increase the density of STT-MRAM by 20% with low or no impact on write-buffer performance.

4. CONCLUSION

The high cost per GiB for SSDs (relative to HDDs) continues to drive their use as part of a hybrid system where they serve as a high-performance storage-tier cache for the hot data, while HDDs are used as the lower-performance bulk storage. This design-point leads to a very specific design problem in storage-tier caching: accelerated drive wear-out from the aggregation of write-intense data. Frequently written blocks pose a dilemma because their frequent writes can affect the lifetime of non-volatile storage (especially in modern MLC SSD caches), but not caching them in the NV storage causes performance degradation (given how frequently they are accessed).

Prior techniques that allow the caching of writes and specifically write-intense data, like SieveStore-C, have largely depended upon the higher write-endurance ratings of SLC Flash to maintain SSD-Cache lifetimes on-par with typical server lifetimes (i.e., 3-5 years). However, SSD production and Flash production in general have nearly exclusively switched to utilizing the significantly less write-tolerant MLC Flash, due their higher bit-densities, largely removing SLC Flash based SSDs from consideration in storage caches due to further increased cost premiums and limited availability. Due to the largely diminished write-endurance ratings of MLC Flash (e.g., 92× lower), sustainable caching of write-intense data would require a mix of higher-level write-buffers (typically RAM-based) to filter the writes that reach the SSD-cache and over-provisioning of the SSD-cache capacity to leverage wear-leveling approaches for extending drive lifetimes [4–6].

Unfortunately, RAM-based write buffers do not represent a storage-equivalent solution as RAM is volatile. Buffering file system writes in volatile memory can result in data loss in case of crash/failure of the computer in question. To achieve equivalent durability as persistent storage, the RAM-buffers must be protected by backup power sources (to enable flushing to persistent storage on a crash) with power

requirements increasing with increasing buffer size. In this work I ameliorate and/or eliminate such additional costs of write-buffering while still filtering enough writes to minimize over-provisioning costs needed to extend SSD-cache lifetimes beyond three years.

I address these issues in two stages. Firstly I propose – WriteGuard– a self-tuned sieved write-buffer that maximizes the write filtering for a given write-buffer capacity, (or alternately, reduces the amount of buffering needed to achieve a given level of write-filtering). I show that, while simple replacement algorithms such as *least recently used (LRU)* and *least frequently used (LFU)* can be implemented with low computation complexity, they significantly under-filter writes. On the other hand, sophisticated techniques that combine recency, frequency, and aging/inflation mechanisms (e.g., LRFU-with-inflation [7]) achieve significantly higher write filtering. However, the computational cost of these sophisticated algorithms can be so high that it becomes an IO bottleneck. My work then diagnoses the key source of complexity in the sophisticated replacement algorithms – performing $log(n)$ operations to maintain the replacement stack in sorted order, which is used to enforce the invariant that a more "valuable" block is not replaced by a less valuable one. Based on the insight that an alternate "sieving approach" proposed in [3] uses a thresholding to prevent allocations to less popular blocks and achieves results similar to a frequency-based sort, I propose a design WriteGuard that avoids maintaining a fully sorted stack. Instead, WriteGuard eliminates the need for a sorted replacement stack with a dynamically-tuned thresholding mechanism that requires only a constant (amortized) amount of work for a given access while no longer needing offline analysis like the design in [3]. This approach reduces the amount of buffering needed by a factor of up-to 2x relative to recency-based and frequency-based write buffers, while requiring similar or less computational effort.

Next I explore using STT-MRAM to replace DRAM in write-buffers and thus eliminate the required additional power-backup. While STT-MRAM is continuing to scale in density it's higher cost relative to DRAM limits is adoption to the premium

system designs (such as the write-buffers within IBM's FlashCore modules). I show that, for storage workloads, write buffers exhibit internal access patterns that enable STT-MRAM design scaling beyond the limitations imposed in traditional workloads where the criticality of read-performance largely determines cell design. My work then illustrates the write-dominance of accesses within write-buffers enables the trading of superfluous cell retention time for increased cell density, resulting in a STT-MRAM cell size reduction of 20% with low or no impact on write buffer performance.

REFERENCES

REFERENCES

[1] T. Kgil, D. Roberts, and T. Mudge, "Improving nand flash based disk caches," in *Proc. of the 35th International Symposium on Computer Architecture*, 2008, pp. 327–338.

[2] T. Kgil and T. Mudge, "Flashcache: a nand flash memory file cache for low power web servers," in *Proc. of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, 2006, pp. 103–112.

[3] T. Pritchett and M. Thottethodi, "Sievestore: A highly-selective, ensemble-level disk cache for cost-performance," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, 2010, pp. 163–174. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815982

[4] R. H. Bruce, R. H. Bruce, E. T. Cohen, and A. J. Christie, "Unified re-map and cache-index table with dual write-counters for wear-leveling of non-volatile flash ram mass storage," Dec. 7 1999, uS Patent 6,000,006.

[5] H. Kim and S. Ahn, "Bplru: A buffer management scheme for improving random writes in flash storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 16:1–16:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364813.1364829

[6] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX 2008 Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404019

[7] L. Cherkasova, *Improving WWW proxies performance with greedy-dual-size-frequency caching policy.* Hewlett-Packard Laboratories, 1998.

[8] B. Tallis, "Ibm and everspin announce 19tb nvme ssd with mram write cache," Aug 2018. [Online]. Available: https://www.anandtech.com/print/13174/ibm-and-everspin-announce-19tb-nvme-ssd-with-mram-write-cache

[9] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan, "How I learned to stop worrying and love flash endurance," in *2nd USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2010.* USENIX Association, 2010.

[10] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. I. T. Rowstron, "Everest: Scaling down peak loads through i/o off-loading," in *OSDI*, 2008, pp. 15–28.

[11] L.-P. Chang and Y.-C. Su, "Plugging versus logging: A new approach to write buffer management for solid-state disks," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024731

[12] C. Wang and W.-F. Wong, "Treeftl: Efficient ram management for high performance of nand flash-based storage systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 374–379. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485379

[13] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache," in *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, ser. FAST '03. Berkeley, CA, USA: USENIX Association, 2003, pp. 115–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=1090694.1090708

[14] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 6, pp. 505–519, June 2004.

[15] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating content management techniques for web proxy caches," *SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 4, pp. 3–11, Mar. 2000. [Online]. Available: http://doi.acm.org/10.1145/346000.346003

[16] Z. Fan, D. H. C. Du, and D. Voigt, "H-arc: A non-volatile memory based cache policy for solid state drives," in *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, June 2014, pp. 1–11.

[17] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, "Cflru: A replacement algorithm for flash memory," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, ser. CASES '06. New York, NY, USA: ACM, 2006, pp. 234–241. [Online]. Available: http://doi.acm.org/10.1145/1176760.1176789

[18] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "Lru-wsr: integration of lru and writes sequence reordering for flash memory," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1215–1223, August 2008.

[19] Z. Fan, F. Wu, D. Park, J. Diehl, D. Voigt, and D. Du, "Hibachi: A cooperative hybrid cache with nvram and dram for storage arrays," 2017.

[20] S. Park and C. Park, "Frd: A filtering based buffer cache algorithm that considers both frequency and reuse distance," 2017.

[21] G. Yadgar, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving ssd erase costs using wom codes," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, Feb. 2015, pp. 257–271. [Online]. Available: https://www.usenix.org/conference/fast15/technical-sessions/presentation/yadgar

[22] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann, "The devil is in the details: Implementing flash page reuse with wom codes," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/margaglia

[23] S. Moon and A. L. N. Reddy, "Adaptive policies for balancing performance and lifetime of mixed ssd arrays through workload sampling," in *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*, May 2016, pp. 1–13.

[24] X. Jimenez, D. Novo, and P. Ienne, "Wear unleveling: Improving nand flash lifetime by balancing page endurance," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. Santa Clara, CA: USENIX, 2014, pp. 47–59. [Online]. Available: https://www.usenix.org/conference/fast14/technical-sessions/presentation/jimenez

[25] J. Jeong, S. S. Hahn, S. Lee, and J. Kim, "Lifetime improvement of nand flash-based storage systems using dynamic program and erase scaling," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. Santa Clara, CA: USENIX, 2014, pp. 61–74. [Online]. Available: https://www.usenix.org/conference/fast14/technical-sessions/presentation/jeong

[26] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing nand flash-based ssds via retention relaxation," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208461.2208472

[27] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen, "Duracache: A durable ssd cache using mlc nand flash," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 166:1–166:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488939

[28] X. Zhang, J. Li, H. Wang, K. Zhao, and T. Zhang, "Reducing solid-state storage device write stress through opportunistic in-place delta compression," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/zhang-xuebin

[29] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh, "Reducing write amplification of flash storage through cooperative data management with nvm," *Trans. Storage*, vol. 13, no. 2, pp. 12:1–12:13, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3060146

[30] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/schroeder

[31] S. P. Park, S. Gupta, N. Mojumder, A. Raghunathan, and K. Roy, "Future cache design using STT MRAMs for improved energy efficiency," *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*, p. 492, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2228360.2228447

[32] K. L. Wang, J. G. Alzate, and P. Khalili Amiri, "Low-power non-volatile spintronic memory: STT-RAM and beyond," *Journal of Physics D: Applied Physics*, vol. 46, no. 8, 2013.

[33] K. C. Chun, H. Zhao, J. D. Harms, T. H. Kim, J. P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, 2013.

[34] N. D. Rizzo, D. Houssameddine, J. Janesky, R. Whig, F. B. Mancoff, M. L. Schneider, M. DeHerrera, J. J. Sun, K. Nagel, S. Deshpande, H.-J. Chia, S. M. Alam, T. Andre, S. Aggarwal, and J. M. Slaughter, "A Fully Functional 64 Mb DDR3 ST-MRAM Built<newline/> on 90 nm CMOS Technology," *IEEE Transactions on Magnetics*, vol. 49, no. 7, pp. 4441–4446, 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6566109

[35] L. Thomas, G. Jan, J. Zhu, H. Liu, Y. J. Lee, S. Le, R. Y. Tong, K. Pi, Y. J. Wang, D. Shen, R. He, J. Haq, J. Teng, V. Lam, K. Huang, T. Zhong, T. Torng, and P. K. Wang, "Perpendicular spin transfer torque magnetic random access memories with high spin torque efficiency and thermal stability for embedded applications (invited)," in *Journal of Applied Physics*, 2014.

[36] E. Technologies, "Everspin achieves data center oem qualification of its 1gb stt-mram solution," Dec 2019. [Online]. Available: https://www.everspin.com/sites/default/files/pressdocs/1Gb_OEM_qualification.pdf

[37] A. Shilov, "Samsung samples 32 gb ddr4 memory chips," May 2019. [Online]. Available: https://www.anandtech.com/show/14341/samsung-samples-32-gb-ddr4-memory-chips

[38] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-volatility for Fast and Energy-efficient STT-RAM Caches," *Proceedings of 17th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 50–61, 2011.

[39] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *DAC Design Automation Conference 2012*, 2012, pp. 243–252.

[40] C. J. Lin, S. H. Kang, Y. J. Wang, K. Lee, X. Zhu, W. C. Chen, X. Li, W. N. Hsu, Y. C. Kao, M. T. Liu, W. C. Chen, Y. Lin, M. Nowak, N. Yu, and L. Tran, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2009.

[41] N. D. Rizzo, M. Deherrera, J. Janesky, B. Engel, J. Slaughter, and S. Tehrani, "Thermally activated magnetization reversal in submicron magnetic tunnel junctions for magnetoresistive random access memory," *Applied Physics Letters*, vol. 80, no. 13, pp. 2335–2337, apr 2002.

[42] A. Raychowdhury, D. Somasekhar, T. Karnik, and V. De, "Design space and scalability exploration of 1T-1STT MTJ memory arrays in the presence of variability and disturbances," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2009.

[43] A. Driskill-Smith, D. Apalkov, V. Nikitin, X. Tang, S. Watts, D. Lottis, K. Moon, A. Khvalkovskiy, R. Kawakami, X. Luo, A. Ong, E. Chen, and M. Krounbi, "Latest advances and roadmap for in-plane and perpendicular STT-RAM," *2011 3rd IEEE International Memory Workshop, IMW 2011*, 2011.

[44] N. N. Mojumder and K. Roy, "Proposal for switching current reduction using reference layer with tilted magnetic anisotropy in magnetic tunnel junctions for spin-transfer torque (STT) MRAM," *IEEE Transactions on Electron Devices*, vol. 59, no. 11, pp. 3054–3060, 2012.

[45] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, p. 34, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1555815.1555761

[46] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based energy-efficiency hybrid cache for CMPs," *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, VLSI-SoC 2011*, pp. 31–36, 2011.

[47] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an STT-RAM-based hybrid cache," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 13–24, 2014.