ZombieNAND: Resurrecting Dead NAND Flash for Improved SSD Longevity

Ellis Wilson*, Myoungsoo Jung †, Mahmut T. Kandemir*

*Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
Email: {ellis,kandemir}@cse.psu.edu

†Department of Electrical Engineering
The University of Texas at Dallas
Richardson, TX 75080
Email: jung@utdallas.edu

***Abstract*—As consumer pressure for more bits per dollar and higher density-per-solid-state disk (SSD) forces manufacturers to squeeze more than one bit per flash cell and feature sizes downwards, wear-out is again becoming an increasing concern. Specifically, while single-level cell flash at larger feature sizes used to boast over 100,000 program/erase (P/E) cycles, modern triple-level cell flash can only sustain a measly 3,000 P/E cycles before it can no longer be reliably used. However, one lesser known facet of NAND flash design is that there is no material difference between cells that store one, two, or three bits per cell – it is merely a logical interpretation of the cells contents.**

Therefore, in this work we leverage this interesting property to explore how resurrecting dead flash cells to create "Zombie-NAND" flash can improve an SSD's lifetime, and what, if any, impact on latency results in doing such. Specifically, we analyze the impact of switching a TLC or MLC cell down one bit upon death; this allows the voltage thresholds to rise and life, though at a lower capacity, to continue for that cell. Finding that traditional wear-leveling techniques actually inhibit the benefits of this scheme, we propose and explore how controlled "wear-unleveling" can work in tandem with Zombie-NAND cells to provide vastly increased life and decreased latencies for the drive. In this exploration, we perform rigorous performance measurement over a number of parameters representative of a variety of commodity and commercial SSDs.

I. Introduction

Today, NAND-flash-based Solid State Drives (SSDs) are well-known and well-used in both commodity and commercial spaces. They bring to bear latencies and bandwidths far superior to traditional Hard-Disk Drives (HDDs), which helps to lessen the otherwise vast gap between main memory latencies and HDDs. Unfortunately, while SSDs have made great strides in the recent past, they still suffer from wear-out characteristics of the underlying NAND flash material.

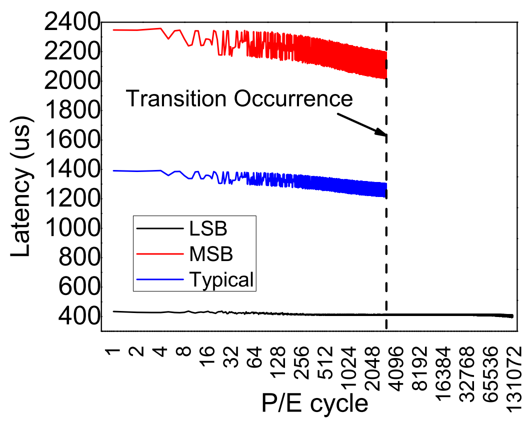
Specifically, NAND flash is only able to handle a certain number of Program and Erase (P/E) cycles before a signal can no longer be stored stably. This issue is exacerbated by the reality that, to enable increased data density at a given feature size, techniques have been developed to store two, three, and even more bits into a single NAND flash cell, leading to major reductions in the threshold at which that cell must be declared "dead." [1], [2] Furthermore, new material science has led to reductions in the very size of such cells, which also impacts their ability to reliably retain data. [2] Because of these issues, as NAND-flash-based SSDs are used and age, their performance degrades as cells become harder to program reliably, and some cells die altogether and must be worked around. There are a number of works that have attempted to cope with these wear-out problems via techniques such as improved garabage collection to reduce write-amplification (the increased impact of a single write due to garbage collection), which we elaborate on in Section V.

However, in this work, rather than proposing another technique to reduce write-amplification or avoid temporary writes ever making it to the flash, we seek an entirely "unnatural" solution to the problem: we allow the cells to die a normal death, and subsequently bring them back from the dead, which we call zombified NAND-flash-cells. More specifically, upon the death of a cell, the point at which it can no longer reliably store two or three bits, we propose an algorithm that reduces the bit-capacity of a given cell by one. This has the curious side-effect of increasing the read, write, and erase speeds of a zombified flash-cell, and, on the whole, in many cases gradually improves the performance of the drive as it ages. To fully leverage these effects, we propose a novel adaptation to existing wear-leveling and garbage collection that greatly increases the potency of the ZombieNAND technique.

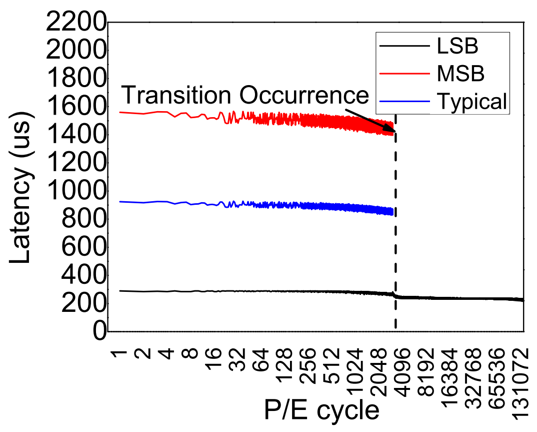
Nevertheless, there seems to be one major drawback to zombifying your NAND-flash cells: one bit of capacity is lost upon every transition to a lower bit-state. However, we argue that, since we are waiting until the cell dies to transition, the only alternative is to let the cell die and sit dormant, as it does now, which would lose you two or three bits for MLC or TLC cells, respectively. So, we prefer to think of it as *one or two bits are gained relative to complete death* – our results demonstrate this to be the case with vastly increased drive lifetimes compared to traditional SSDs.

It is critical to note and understand why we, in absolutely no case, convert some of the flash to a lower state prior to their natural death. Manufacturers aggressively compete regarding how many P/E cycles their flash promises prior to death, and therefore, are extremely unlikely to implement any strategy that jeopardizes those guarantees. Therefore, towards the goal of developing a novel lifetime-extending scheme *that can be implemented in the real-world*, we make sure never to convert a cell until it has fully worn-out at that bit-level.

Before we begin, we need to motivate that this "zombification" is indeed achievable in *real* NAND-flash. Therefore, we performed and present results from an initial evaluation on two raw MLC NAND devices from distinct manufacturers in Figure 1. Therein we plot latency to write the least-significant bit (LSB) and the most-significant bit (MSB), the two bits in an MLC cell, over their lifetime. Latency shown is to write all LSBs or MSBs in a page, the lowest granularity of writes that can be performed. We continue the process by writing LSB and then MSB for all pages in a block, and



(a) MLC NAND Flash Drive A



(b) MLC NAND Flash Drive B

Fig. 1: Proof-of-concept: Zombification process of NAND flash in two manufacturer's raw flash chips, demonstrating longer life and improved latency. Latency is shown to write all LSBs or MSBs in a page, and this process of writing all LSBs and then all MSBs in each page, for all pages in a block, and then erasing the block, are shown up until the block is declared dead. Typical latency is the average of the two.

then perform an erase on that block. This process is repeated until the block throws errors as demarcated with the vertical dotted line, at which point we transition from MLC to SLC and solely write to the LSB going forward. As can be seen, the cells continue to function in SLC-mode post-transition from approximately 3,000 Program/Erase (P/E) cycles all the way to in excess of 100,000 P/E cycles, and further, post-transition average latencies inherit the lower latencies of SLC NAND-flash. Programming the MSB is naturally slower than for the LSB because more discrete analog ranges need to be programmed carefully; the LSB only has two, whereas the MSB has four. Last, we believe latencies slightly improve over their lifetime due to trapped charges making the programming easier. Eventually this kills the cell, but in the short term it improves programming latencies.

However, manual transition of cells in this way is only a proof-of-concept – not a production-ready contribution. Therefore, to master the art of auto-zombification and management of zombie versus living NAND cells, we present the major contributions of this paper:

- **Physics-Accurate SSD Simulator:** We need to simulate much of the functionality of a flash-based SSD's flash translation layer (FTL), but, with the fidelity of a physics-based flash cell simulator because block-erase counters simply do not suffice (we do not know how much is really left post-transition). Therefore, we made extensive changes to a popularly used SSD simulator to enable it to perform physics-accurate transitions between bit-states. This is, to the best of our knowledge, the first work to enable accurate simulation of the entire lifetime of an SSD from prestine state to its death, which we leverage extensively in our exploration of the ZombieNAND technique.
- **Controlled Wear-Unleveling:** We find that traditional wear-leveling results in too great a number of the cells dying at around the same time, limiting the impact of zombification. Therefore, we develop a novel wear-leveling mechanism that is both simple and effective at achieving a controlled degree of wear-unleveling such that a configurable number of cells transition (die and become zombies) earlier than the rest, which often improves the lifetime and the performance of the drive on the whole in advanced ages.
- **Industry-Relevant Results:** Changing the bit state sacrifices capacity and therefore has potential to cause the drive to use all reserved space and die earlier than as promised by the manufacturer. To avoid this and assure our results are industry-relevant, we set an invariant that we *never* may transition a cell to a lower bit-state unless it has reached the promised number of erases by the manufacturer. Further, by prioritizing simple but effective adaptations to existing garbage-collection and wear-leveling techniques over complex and fragile algorithms, we argue ZombieNAND can be implemented with only a modicum of difficulty across a wide variety of existing FTLs.
- **Expansive Environment Evaluation:** We witness a broad spectrum of performance and lifetime impacts based on varying environment conditions such as: proportion of reserved space, workload read to write ratios, working set size relative to reserved space size, and starting at MLC versus TLC. Therefore, in our evaluation, we use our simulator to perform an expansive search over all such environment conditions from prestine state to death of the SSD to provide conclusive evidence to when zombification can bring large benefits and when it cannot.

In this work we explore our ZombieNAND technique using both synthetic workloads and real traces. We demonstrate that ZombieNAND *never* hurts the lifetime of the SSD when compared against an SSD without it for over five-hundred different synthetic experiments and over sixty trace-driven runs. For workloads and configurations where it does help, which are the majority of the results presented, for TLC drives it can extend the lifetime from 20% to over 11 times for traces, and as high as 16 times for fully random synthetic I/O. For MLC drives this lifetime extension ranges from 73% to 375%, and reaches 325% in synthetic tests. Additionally, we provide time-series analysis demonstrating that ZombieNAND never degrades average latency compared to a vanilla drive during the lifetime of the vanilla drive. While average latency does, under some workloads, degrade somewhat after the death of the vanilla drive, a bit slower is much preferable to dead, and for around half of the traces latency is instead reduced by

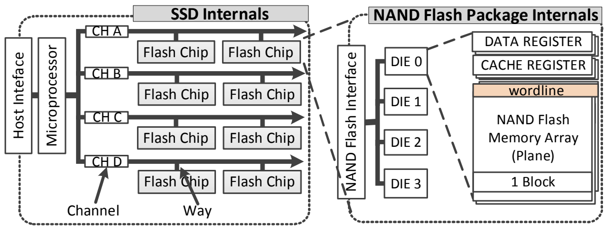


Fig. 2: Typical SSD Architecture

typically around 20% to 25%, and can reduce as much as 50% (even post-death) for TLC. MLC results demonstrate typical latency improvements from about 10% to 25% for half of the workloads, and can be as low as 40% reduced.

II. Background

To facilitate an informed vantage point for our work, we provide background starting at a high level for flash-based SSDs and drilling down. We first describe SSD architecture and some of the key parameters and mechanisms we explore in this work, and then move into the actual NAND flash itself. Having given a basic overview of the mechanisms in NAND flash, we last highlight some of the physics of NAND wear-out characteristics, which we leverage towards high-fidelity simulation of bit-switching for "undead" flash.

A. SSD Architecture

NAND-flash-based SSDs are highly parallel in nature, as shown in Figure 2. Each SSD is connected to the host typically by SATA, SAS, or PCIe interfaces, and multiple channels within connect the interface to groups of NAND packages. Within each package there exist one or more dies, and within each die there are often one or more planes, which require specialized data access modes to take full advantage of their parallelism. In total, this makes for an architecture where hundreds of individually small, low-bandwidth, flash storage areas all work in tandem to provide larger capacities, low latencies, and high bandwidth.

Managing that architecture towards optimal performance and importantly, effective wear-leveling, is a complex task that is the duty of the flash translation layer (FTL). This FTL performs duties including, but by no means limited to, address translation (between the host and the physical addresses), bad block management, wear-leveling, a variety of performance optimizations, and garbage collection. In this work the FTL plays a crucial role in allowing "undead" flash to not only clamber along after death, but also to acknowledge that these cells store a smaller capacity, and to utilize their special properties of being faster and longer-lived to extend the life and improve performance of the drive on the whole.

B. NAND Flash Overview

Moving down to analysis of the NAND flash material itself, at the level of an individual cell NAND is made up of floating-gate transistors. These transistors are capable of storing data for extended periods of time relative to DRAM and other temporary memory solutions, but have the side-effect that, in order to write over a previously written section, they must first be discharged (hereafter referred to as "erased"). For the most basic NAND flash cells, single-level cells (SLC), the floating-gate either has a high threshold voltage (V_t), which demarcates that the cell is erased and is a "1," or it has a low V_t, indicating it has been written and is "0." For increased bit counts, more voltage levels are added to indicate the increased number of bits, which divides the entire voltage range into 2^n distinct voltage levels for an n-bit flash cell.

C. The Physics of Cell Wear-Out

There are countless interesting and nuanced properties of NAND flash cells, but the critical ones for this work revolve around their physical properties relating to wearing-out. As mentioned, NAND flash suffers from wear-out due to the impact of the P/E cycle. This is specifically due to the charging process resulting in electrons getting "stuck" in the interface and oxide layers of the floating-gate transistor. As more and more electrons accumulate there, at some point V_t exceeds the margin between distinct voltage levels and therefore discerning between one level or the other becomes impossible. However, as is to be expected, this collection is not purely monotonic – after some time charge can "leak" out from the oxide, returning a cell from a dead state to a usable state again.

This process of electron trapping and leakage has been extensively studied and modelled in [3], [4], and [5], which we incorporate into this work. This level of high-fidelity wear-out simulation is required in this work because we need to know, if we switch from some higher bit-mode to a lower one, what "life" yet remains in the cells. Since switching is merely a logical interpretation, we retain the trapped voltage at a per-block granularity, which will not change on bit-switch, and reference a higher threshold relative to the current bit-mode.

Algorithm 1 Oxide Stress Model Psuedocode

```
1: procedure CALC_STRESS($cycles$)
2:     $A \leftarrow 0.08$
3:     $B \leftarrow 5.0$
4:     $Cox \leftarrow 2.15e^{-17}$
5:     $q \leftarrow 1.6e^{-19}$
6:     $\delta N_{it} \leftarrow A * cycles^{0.62}$
7:     $\delta N_{ot} \leftarrow B * cycles^{0.30}$
8:     $\delta V_{it} \leftarrow (\delta N_{it} * q)/Cox$
9:     $\delta V_{ot} \leftarrow (\delta N_{ot} * q)/Cox$
10: return $(\delta V_{it} + \delta V_{ot})$
11: end procedure
```

Pseudocode for our oxide stress model, most closely tracking the model as presented in [3], is shown in Figure 1. The input $cycles$ is simply the number of P/E cycles already performed on the block, A and B are constants derived in the mentioned work, and q is electron charge in Coulombs. As [3] failed to specify the value for Cox, which is the capacitance of the oxide, we were forced to re-calculate it by fitting to their graphs and validating against our evaluation on actual flash as shown in Figure 1. δN_{it} and δN_{ot} represent densities of traps forming in the oxide, respectively, interface traps and bulk traps. Correspondingly, δV_{it} and δV_{ot} are the interface and bulk trap voltage shifts, which add together towards the total trapped voltage in the block following the erase of δV_t.[1]

III. Wear-Unleveling for Lifetime

A. The Basics of Wear-Leveling

Modern SSDs tout all forms of complex garbage-collection and wear-leveling algorithms to assure the cells within wear-

[1]In private discussions with industry professionals we ascertained that, while this stress model roughly captures the effects of P/E cycles in modern NAND flash, recovery characteristics due to charge leakage from the oxide layers are far less consistent between manufacturers and feature sizes, and therefore difficult to predict. Therefore, to present the most conservative estimates of lifetime-enhancement, we assume zero recovery is possible.

down at roughly the same rate throughout the many planes, dies, packages, and channels within. In doing this traditional SSDs, which simply mark cells as dead when they are no longer reliable in storing the bit-count they were designed for, do an excellent job at reaching near-optimal life. Optimal lifespan for a traditional SSD can be defined as "using all of the flash medium in such a uniform manner that when a cell finally is erased and declared dead, all of the remaining cells are also one erase from death themselves." In short, if all the flash dies at nearly the same time, the most use was gotten out of the pool of flash as is possible. To fully appreciate this consider a sub-optimal alternative, where half of the flash in the SSD is on the verge of death and the half has been completely unused thus far. If that near-dead half continues to be used and dies off, the drive will report itself as wholly dead when the amount of dead flash exceeds the reserved amount (often in the range of 5% to 30% extra flash). So, in this simplified case, only half of the drive's life has been expended even though it has reported itself as entirely dead to the operating system.

Wear-leveling functionality tends to be concentrated around the garbage-collection process as that is where erases largely occur, and erases have the most negative impact on cell lifetime. In a simple garbage collection process, once it begins on a given package it does not stop cleaning until a high-watermark is reached. At a high-level, first a usage table is retrieved, which organizes all the blocks in the package into buckets based on the number of valid pages within. A page is considered valid if a logical address still points to it – upon an erase and/or rewrite of that logical address, the block is simply marked as "invalid," it is not necessarily immediately erased in physical terms. The highest priority bucket contains blocks whom have no valid pages within (and therefore no data will need to be read and written out somewhere else), and the lowest priority bucket has blocks with only one invalid page. Blocks without any invalid pages are not considered, as erasing and moving these does nothing but damage and slow the drive. By prioritizing erasure of blocks with large numbers of invalid pages, a smaller amount of data needs to be read out of these blocks, coalesced, and written back into other blocks (also known as write amplification).

This results in higher-performance garbage collection and lower write amplification, but without other protections does nothing to assure real wear-leveling is being achieved. Therefore, as the garbage collector works through the high-priority buckets, it also considers the number of times a given block has been erased in comparison to the average erase count of the entire package. If that block exceeds the average by a predefined threshold, we consider it to be overused and skip past it to another block with less erases. Again, this is a simplified garbage-collection and wear-leveling strategy – deciding which blocks to erase, which to skip past, how write-amplification should be balanced against over-worn blocks, and the like have all been studied at length in numerous other works. Moreover, with few exceptions, our proposed changes to this simplified wear-leveling mechanism as follows is sufficiently simple such that it should be orthogonally applicable to a broad base of the many garbage collectors and wear-leveling algorithms currently in the wild.

B. The Early Switching Pool

In this work a simplified wear-leveling strategy not only suffices to explore our methodology, but was found in early experiments to be overly effective. Put succinctly, when you give up the requirement that "a cell dies when it is no longer able to reliably store a predefined set of bits," and instead allow that cell to repurpose itself to a lower bit-count as we do, you no longer want your entire drive to reach a near-dead state at the same time. If such occurs, transitioning down to a lower-bit count results in only a dead drive. If an MLC drive wholly transitions to SLC, unless the reserved area is greater than 50% of the drive (unlikely for cost reasons), the drive will be forced to report itself as dead. The same property applies to TLC, although the subsequent transition to SLC means an even greater reserved area would be needed to protect against whole drive transition without the drive announcing its death.

Therefore, we propose a simple but, as we will later demonstrate, highly effective adaptation to the above wear-leveling strategy that "wear-unlevels" a predefined fraction of the drive in a controlled manner *without sacrificing any of the manufacturer's guarantees about drive lifetime* as shown in Algorithms 2 and 3. Further, we define a pool of "early switch" blocks in each and every flash die as the following:

$$\text{Early Blocks} \leq \frac{(R - W) \times B}{2^{S-2}} \tag{1}$$

Here R is the reserved percentage, W is the high-watermark percentage (when GC stops), B is the number of blocks per element, and last S is the starting bit-level of the blocks (we do make the assumption that all blocks in the SSD start at the same bit-level). All percentages refer to the entire drive size, and are expressed as integers, not decimals. Further, R is assumed to be always greater than W (otherwise you may be unable to perform GC when you run out of reserved area on a full drive). The high-water free blocks (where free means erased completely and not yet written) percentage is first removed from the pool of reserved blocks as we consider a drive dead if it exceeds this amount of dead blocks. The remaining percentage is then multiplied by the blocks per element to get our pool of blocks which can die without the element dying entirely, which we last divide by a power of two dependent upon the number of bits started with. This final term, which boils down to simply dividing by two if started as TLC, was decided upon after testing turning the entirety of that extra space into "early switch" cells. Because TLC (when compared against MLC) has a very low erase count prior to death, but a higher capacity after death and transition, getting a the early switch blocks to transition sooner is critical, and therefore concentrating accesses on a smaller early switch pool is the easiest way to achieve this.

Our wear-unleveling methodology as described in Algorithms 2 and 3 follow much of the same paths as the standard wear-leveling algorithm, with a two notable points to make: First, although unshown, whenever an erase is performed, if the block is discovered to be dead (in our simulator, indicated by having trapped charges exceeding specified thresholds), it is always transitioned down one bit (dead SLC can go no further and therefore is simply marked dead). Thresholds correspondingly raise on this action, and, provided the reserved capacity of the drive has not fallen below the high-watermark percentage, the block can live on to die another day. Second, in creating our usage table buckets, number of valid pages is still the most important property considered in sorting. Simply being an early-switcher does not allow a block to get into a higher priority bucket – write amplification due to increased valid pages would over-ride potential benefits. However, if an early-switch block and a normal block are both in the same valid-pages bucket, the early-switch block will always be chosen, and thus the early-switch property is the second most important propery to sort upon. So, although

Algorithm 2 Usage Table Construction

```
1: procedure BUILD_USAGE_TABLE($element$)
2:     $table \leftarrow$ MALLOC_TABLE
3:     for $i \leftarrow 0, params.blocks\_per\_element$ do
4:         $usage \leftarrow metadata.block\_usage[i].num\_valid$
5:         $table[usage].len$++
6:     end for
7:     $buckets \leftarrow$ MALLOC_BUCKETS($table$)
8:     for $i \leftarrow 0, params.blocks\_per\_element$ do
9:         $usage \leftarrow metadata.block\_usage[i].num\_valid$
10:        $buckets[usage][table[usage].tmp].num \leftarrow i$
11:        $buckets[usage][table[usage].tmp$++$].rem\_life \leftarrow$ CALC_LIFE_REMAINING($element, i$)
12:    end for
13:    for $i \leftarrow 0, params.pages\_per\_block - 1$ do
14:        QSORT($buckets[i], table[i].len, comp\_life\_earlyswitch$)
15:        for $j \leftarrow 0, table[i].len$ do
16:            $table[i].block[j] \leftarrow buckets[i][j].num$
17:        end for
18:    end for
19: end procedure
```

Algorithm 3 GC with Controlled Wear-Unleveling

```
1: procedure CLEAN_BLOCKS($element$)
2:     $avg\_life \leftarrow$ COMPUTE_AVG_LIFE($element$)
3:     $usage\_table \leftarrow$ BUILD_USAGE_TABLE($element$)
4:     for $i \leftarrow 0, params.pages\_per\_block - 1$ do
5:         $current\_table \leftarrow usage\_table[i]$
6:         for $j \leftarrow 0, current\_table.length$ do
7:             $blk \leftarrow current\_table.block[j]$
8:             $rem\_life \leftarrow$ CALC_LIFE_REMAINING($element, blk$)
9:             if $rem\_life < (LIFE\_THRESHOLD * avg\_life)$ then
10:                if NOT_EARLY_SWITCH($blk$) then
11:                    $continue$
12:                end if
13:            end if
14:            CLEAN_BLOCK($blk, element$)
15:            if DONE_CLEANING($element$) then
16:                $break$
17:            end if
18:        end for
19:        if DONE_CLEANING(element) then
20:            $break$
21:        end if
22:    end for
23: end procedure
```

early-switch blocks are not wear-leveled against normal blocks (and importantly, vice-versa), early-switch blocks are wear-leveled amongst themselves.

IV. Evaluation

A. Simulation Framework

In order to evaluate the ZombieNAND algorithm as just described, we needed to first build a simulator capable of returning physically-accurate results for long-running simulations. DiskSim [6], a widely known and used magnetic disk simulator, has been extended for simulation of idealized SSDs by Microsoft Research [7]. As ZombieNAND concentrates on extending the overall lifetime of the SSD, we deemed it reasonably acceptable to use an idealized SSD simulation framework rather than something higher fidelity on the short-term but much more computationally expensive. However, this extension fell quite short of our needs as it only used block-counter style of lifetime estimation. Because we are switching the logical interpretation of a cell when it reaches death in its current state, it is not, for example, acceptable to simply deduct 3,000 P/E cycles that were used when in TLC state from the subsequent MLC state. This would result in 27,000 cycles remaining in MLC, and would significantly overestimate the remaining lifetime for the drives.

Therefore, after carefully reviewing works as discussed in Section II, [3], [4], and [5], we incorporated their findings into a new, physically-aware lifetime calculation sub-system within the DiskSim SSD Extension. This was the first and major hurdle in the road towards a simulator that would provide physically-accurate results for ZombieNAND. Second, we had to incorporate the notion of different page sizes in the simulator. Because DiskSim (and the extension) were built on the premise of single size sectors and pages, we had to carefully add features to the existing functionalities to enable them to cope with potentially two or three different page sizes (in the case of MLC or TLC, respectively). Last, although DiskSim is well known and used, it is also somewhat dated for modern 64-bit machines and to our knowledge has never been used for very long-running experiments. Specifically, to properly evaluate ZombieNAND, we needed to start with a fully pristine SSD, and continue issuing operations to it until it declared itself completely dead. This process would often take as long as a week on a single machine and would ultimately have issued *dozens of billions* of operations at the SSD over its lifetime. However, we found many of the variables and data structures in place failed to scale to handle these durations and raw volumes of commands. This resulted in a number of segmentation faults and less obvious bugs until we finally adapted all of the underlying code to handle long-running simulations like ours.

B. Experimental Setup

In evaluating ZombieNAND, we note the following configuration choices as shown in Tables I and II. Five additional points are worth noting to fully understand the following experimental results: First, for the synthetic experiments we simulate an SSD with a single, reasonably small NAND flash element inside because we sought to explore a large search-space of parameters (which would be intractable at larger sizes and counts of elements). Second, in the case of the trace-driven experiments, we move to a larger, multi-element SSD sized at 1 Gigabyte, but cannot scale to modern sizes (128GB-1TB) because again, we are simulating the entire lifetime of the SSD. Even at 1GB, simulations take multiple days on modern machines, and DiskSim was not designed to scale up for HPC-style simulation (because of the large code-base, adapting it for such would be a massive undertaking). Nevertheless, we argue that our results should scale to larger sized SSDs for synthetic and trace-driven experiments so long as reserve area percentages, which we find to be the most dominant parameter, stay the same. Third, all of these experiments assume a SATA 300 interface (and corresponding block transfer times are used). Fourth, garbage collection, which has been explained to be a critical component of ZombieNAND, kicks in at a predefined minimum percentage of blocks free and runs in the background until a high-watermark percentage of blocks are available again. In all of our experiments, we use 2% and 5% for these values, respectively. Last, while we use a physics-based engine we developed based on models in prior works to determine lifetime remaining for a given flash cell,

Access Type (unit)	SLC (2KB)	MLC (4KB)	TLC (8KB)
Read (page)	0.025 ms	0.05 ms	0.15 ms
Write (page)	0.2 ms	0.5 ms	1.0 ms
Erase (block)	1.5 ms	1.5 ms	3.0 ms

TABLE I: Access latencies based on operation type (unit operation works on is shown in parentheses) and NAND bit-level. NAND bit-level headers are shown with size of a page in parentheses.

	Synthetic	Trace-Driven
Flash Chips	1	4
Blocks per Element	128	512
Planes per Element	8	8
Blocks per Plane	16	64
Pages per Block	128	128

TABLE II: Key experimental configurations of the simulated SSD for synthetic and trace-driven tests.

the parameters within have been tuned to correspond to state-of-the-art manufacturer guarantees. Specifically, we have tuned the physics engine to declare an *unswitching* SLC-, MLC-, and TLC-based block dead at roughly 75,000, 6,000, and 1,000 P/E cycles, respectively. These values were gathered from a broad survey we performed of recently-released SSDs in the various categories and manufacturer specifications associated with them, and the timings shown in Table I were drawn from specification sheets [8], [9], and [10].

C. Synthetic Results

In this set of experiments we use a synthetic access generator we built that takes two parameters, read-to-write ratio and working-set size, and test it across a number of different reserved area configurations of the SSD. This synthetic access generator simply generates a random workload across the specified working-set size, and continues issuing random accesses as fast as the SSD can handle until the drive dies. **Read-to-write ratio** simply defines the probability the randomly generated access will be a read or a write. **Working-set size** defines how large of an address space the synthetic generator can randomly choose an access to occur on. For example, since we are using a single element in these experiments, 64MB in size, in the cases of 50% working set size, addresses starting at zero and going up to 32MB of the drive can all be issued to. Last, it is perhaps most important to understand specifically what we mean by **reserved area.** In all of our synthetic and trace-driven results, reserve area denotes the percentage of the total state size *deducted* from the user-visible SSD space. So, unlike its common use, in our 1GB SSD evaluation, whether we are using a reserved area of 10% or 30%, the total flash in the SSD is still 1GB – the only thing changing is the accessible logical address space to the application. This is a critical point because we did not want to confound our results by having added more reserved area to a base amount of flash; fixing the total raw flash available keeps all of the configurations on an even playing field.

Results from *over five-hundred distinct experiments* we performed to cover variations on these three key dimensions for both TLC and MLC are shown in Figure 3. These heat-maps depict life and latency changes *relative to an unswitching execution with the exact same parameters*, which we hereafter refer to as the *baseline*. The baseline is normalized to 1.0 for both latency and lifetime plots, and in the lifetime cases, all experiments do as least as well as the baseline (a design goal). The latency results depict *latency average over the entire lifetime of the SSD*, so it is important to remember in the ZombieNAND case the lifetime is always greater than or equal to the baseline. Therefore, even though sometimes the latencies are worse than the baseline average, *these degradations only occur after the life of the baseline SSD has expired*. We lack space to show detailed, time-series latency results for these synthetic runs, but for an example of this behavior, please reference Figure 6 from the results of trace-driven evaluation.

From these synthetic results, we identify the following major three take-aways:

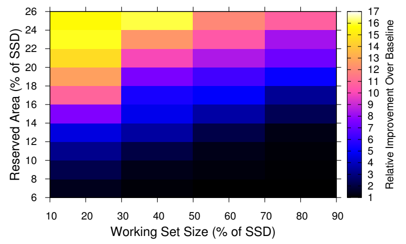
First, for TLC drives, small working-set sizes and large reserve areas result in enormous gains over the baseline in excess of an order of magnitude. At first blush these gains seem egregious, but bear in mind we are starting at TLC, which has a lifetime around 1,000 P/E cycles, and we transition to SLC, which has a lifetime 75 times greater. Obviously a full 75 times improvement is not expected as that is not how the physics of flash lifetime works, nor can the entire drive be converted to SLC and still live, but these (still large) improvements are a function of the great difference in endurance between the types. The MLC results depict a similar inter-relationship as we vary the parameters, however demonstrate lower gains possible than in the TLC case. Again, this relates to the difference in lifetimes – as MLC lasts approximately six times longer than TLC from the outset, transitioning to SLC buys us some, but not as much relative lifetime as in the case of starting at TLC. Similarly, latency differences between MLC and SLC are smaller than TLC to SLC, so these gaps (both in lifetime improvements and degradations) also lessen.

Second, we are witness to a somewhat odd latency degradation in the TLC case not in the bottom right as we would expect (highest working size, lowest reserved area) but somewhat up from that. In analyzing the results, we find that for reserved areas smaller than around 13%, there is no real change in lifetime or latency. Right around 13% lifetime begins to extend, but, because there is limited amounts of converted "fast blocks," all too often an access results in a write to a fast block *and* a normal block (because the access is larger than the smaller, fast blocks size, and no other fast blocks may be available at that time). This results in latencies somewhat larger than the normal block access until the reserve area grows enough that a decent pool of fast blocks is available at any given time in most of the invalid-page buckets.

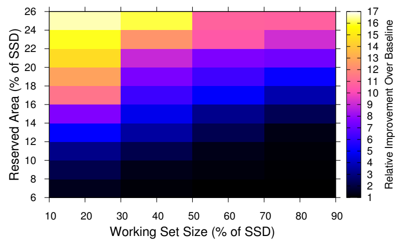
Third, and what may have been initially most apparent, we witness extremely similar *relative* behavior across low, medium, and high read-to-write ratio graphs. We emphasize relative because the absolute number of accesses is definitely not the same (since reads have extremely limited impact on cell lifetime) across equivalent configurations where only read-to-write ratio is changed.

D. Trace-Driven Results

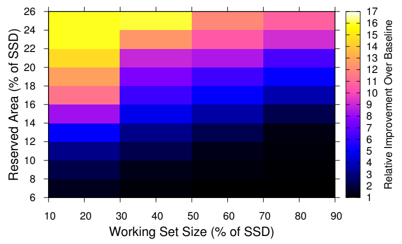
Last, we consider the efficacy of ZombieNAND in extending SSD lifetime and explore its impact on latency over a variety of real application traces. These ten traces represent I/O workloads in financial workloads, file servers, user home directories, and internet SQL servers, derived from traces available at [11] and [12]. We have quantified and present specifics about each trace, including the read-to-write ratio and random vs. sequentiality of the trace in Table III, and present address reuse of accesses in Figure 4.



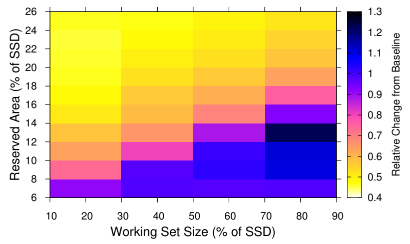
(a) Lifetime (Synthetic, TLC 20% Writes)



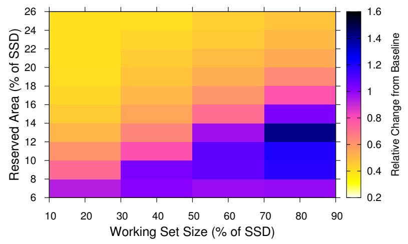
(b) Lifetime (TLC 50% Writes)



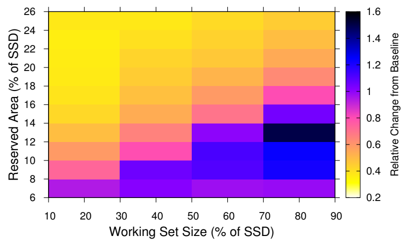
(c) Lifetime (TLC 80% Writes)



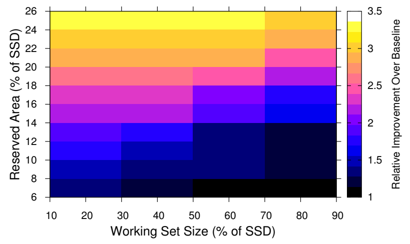
(d) Latency (TLC 20% Writes)



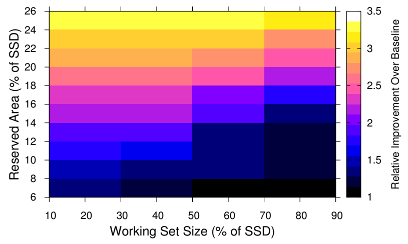
(e) Latency (TLC 50% Writes)



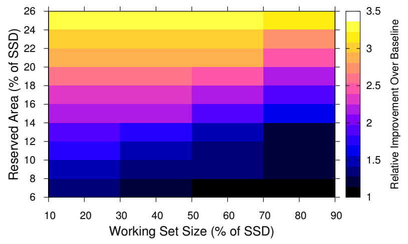
(f) Latency (TLC 80% Writes)



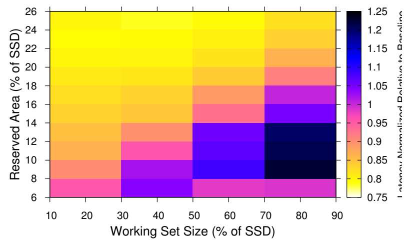
(g) Lifetime (MLC 20% Writes)



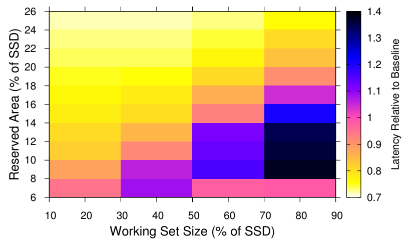
(h) Lifetime (MLC 50% Writes)



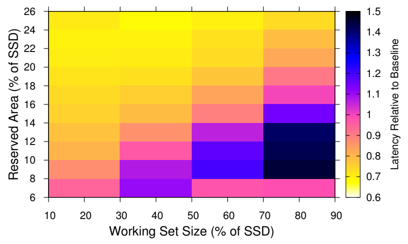
(i) Lifetime (MLC 80% Writes)



(j) Latency (MLC 20% Writes)



(k) Latency (MLC 50% Writes)



(l) Latency (MLC 80% Writes)

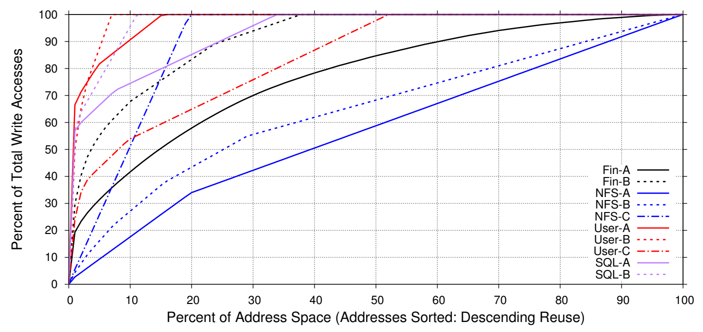
Fig. 3: Lifetime and latency results for TLC- and MLC-NAND-based SSDs under synthetically-generated random-I/O workloads. Varying amounts of reserved area, and working set size of the workload are shown within each graph, and varying amounts of writes are shown across graphs. Lighter colors indicate better lifetime or latency.

Application	Writes (in %)			Reads (in %)		
Trace	Total	Random	Sequential	Total	Random	Sequential
Fin-A	83	47	53	17	45	55
Fin-B	22	33	67	78	78	22
NFS-A	99	1	99	1	1	99
NFS-B	58	22	78	42	1	99
NFS-C	19	0	100	81	0	100
User-A	27	23	77	73	8	92
User-B	9	45	55	91	2	98
User-C	58	18	82	42	6	94
SQL-A	43	25	75	57	10	90
SQL-B	15	26	74	85	2	98

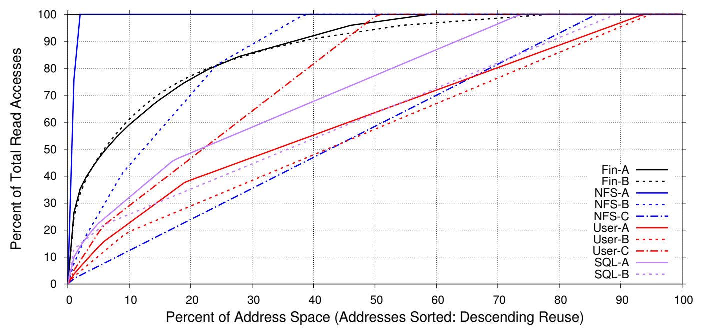
TABLE III: Application trace access composition. Percents expressed in terms of raw data moved.

In order to keep our results from becoming entangled amidst too many varying factors, we only use the first 512MB of *unique addresses* from every trace. This means some traces run very close to only issuing 512MB of accesses (very low address reuse), whereas others issue many gigabytes of accesses (high address reuse) before hitting the 512MB of addresses limit. We must do this to a) make sure the traces do not attempt to use more space than is available in our 1GB SSD, and b) to normalize the address space accessed amongst all traces to roughly 50% of the drive.

This does not in any way mean address reuse has been normalized across traces, as can be seen in Figure 4. Those graphs sort by descending reuse and provide volumes of

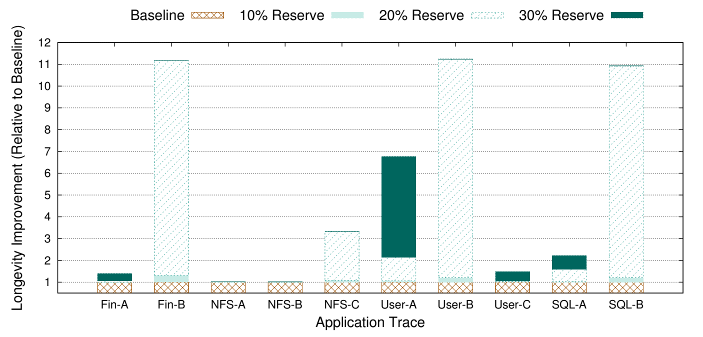


(a) Trace Address Reuse (Writes)

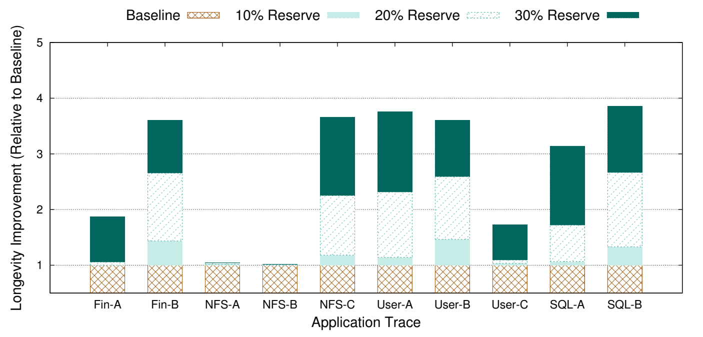


(b) Trace Address Reuse (Reads)

Fig. 4: CDF plots, which describe the fraction of total accesses commonly used addresses are accountable for. For instance, steep curves indicate a small fraction of the address space of a specific trace accounts for a large percentage of the total accesses, and low, slowly growing curves indicate fairly balanced accesses to a large portion of the address space.



(a) Trace Lifetime Comparison (TLC)



(b) Trace Lifetime Comparison (MLC)

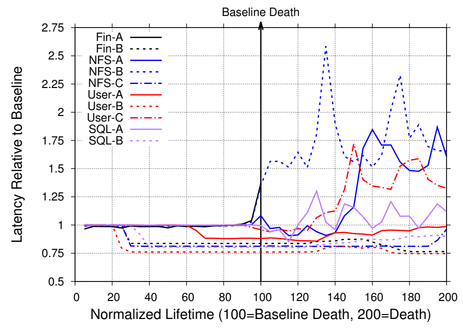
Fig. 5: Length of lifetimes shown relative to baseline (normalized to 1.0) for varying reserve areas. Our algorithm always does at least as good as the baseline, and increased reserves always do as good or better than a smaller one. As shown, some applications benefit tremendously (an order of magnitude in 3 of the 10 traces) from slight increases in reserves, but may not benefit from increased reserves beyond that, whereas others demonstrate little or no improvement over the baseline.

accesses on the y-axis, demonstrating that while some traces exhibit largely uniform accesses across a majority of the access space (e.g., NFS-C in Figure 4b), others only use a small fraction of the entire space for all of their accesses (e.g., Fin-B in Figure 4a). We provide this level of detail because address reuse is a driving factor in the efficacy of ZombieNAND, as we will explain in a moment.

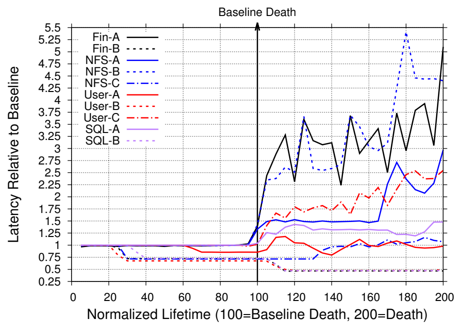
We begin by dissecting the lifetime improvements brought by employing ZombieNAND as shown in the overlaid bar graph in Figure 5. All lifetimes shown are relative to the baseline, which has been normalized to 1, and all perform as good or better than it. Further, increased reserve areas never results in degraded results, so all results are overlaid to demonstrate how much life an additional 10% of reserved area buys us. For instance, in TLC SSD executions of User-A, we can see that while 10% reserve provides extremely small improvements over the baseline, expanding to use of 20% reserve achieves a factor of 2, and moving to 30% gets us just short of a factor of 7. Curiously, this trend is not consistent amongst the traces – some benefit from 10%, but skyrocket at 20% and do not improve further at 30%, while others do not benefit from 10% but equally improve from 20% and 30%, and yet others do not benefit from any amount of reserve.

Analyzing these findings carefully next to the trace breakdown table and CDF plots as presented earlier, we identify one clear and defining driver for ZombieNAND improvements: In order for ZombieNAND to have real impact, writes must be concentrated into a reasonably small fraction of the total address space. Looking to our previous synthetic results, this trend is echoed there, where the working-set size more or less defines performance. What is starkly different from those synthetic results is that such randomized uniformity is not presented in any of these traces, and therefore increasing the reserves does not always net improvements. In fact, in our best performers, going from 20% to 30% reserve nets nothing additional. Analysis of the address reuse graph provides evidence suggesting why: Unlike randomized, uniform accesses as in the synthetic graphs, Fin-B, User-B, and SQL-B all reach different volumes of accesses at different percentages of commonly used addresses (a CDF of the synthetic I/Os would result in linear lines with varying slopes depending upon working-set size). The high-performers tend to have very aggressive slopes in the first few percent of the most commonly used address space, and reach very near to 100% of all accesses performed prior to 30-40% of the address space. Further, also unlike the simplified environment of the synthetic tests, read-heavy workloads do demonstrate more significant lifetime improvements than their mixed brethren with a similar write address reuse curve.

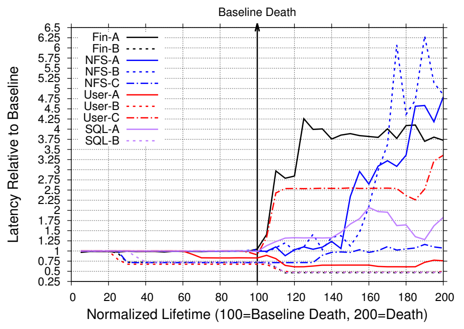
Moving onto the MLC trace-driven results, we are witness to expectedly lower relative improvements, but more consistency in terms lifetime enhancement across the traces. In fact,



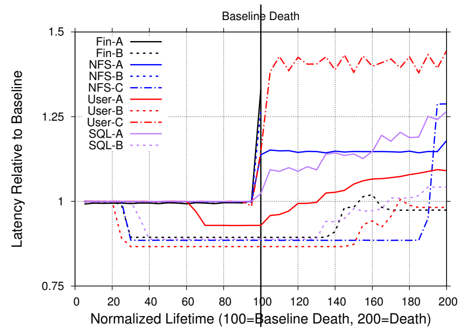
(a) Latency Change over Life (TLC 10% Reserved)



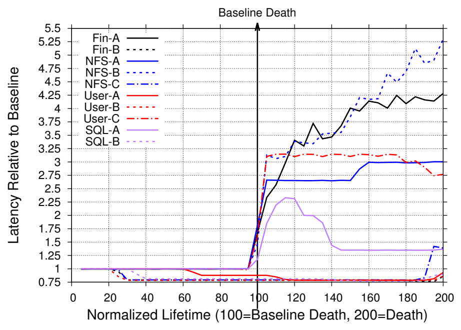
(b) Latency Change over Life (TLC 20% Reserved)



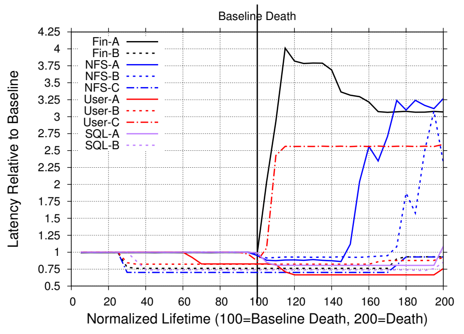
(c) Latency Change over Life (TLC 30% Reserved)



(d) Latency Change over Life (MLC 10% Reserved)



(e) Latency Change over Life (MLC 20% Reserved)



(f) Latency Change over Life (MLC 30% Reserved)

Fig. 6: Latencies over the entire lifetimes of the traces explored, broken down into 10, 20, and 30 percent reserve areas for both MLC- and TLC-based SSDs and normalized over a 200 point range. Vertical line at x-axis 100 demarcates death of the baseline, and, if our algorithm improves lifetime for that trace, x-axis 200 indicates death of the ZombieNAND-enhanced drive. Latencies all shown relative to the baseline, which has been normalized to 1.0.

with the exception of NFS-A and NFS-B, which still do not improve hardly at all (due to lack of write address reuse), all of the traces gain a significant amount of lifetime relative to the best case. Furthermore, the traces also more frequently take advantage of both the 20% and the 30% jump in reserve area, unlike TLC, and likely due to the relative space differences between MLC and TLC.

Finally, the time-series analysis of latency in Figure 6 sheds considerably more light on the latency situation for ZombieNAND than the average latency values did earlier. While we do not have space to fully explore the rich graphs there, two important take-aways exist: One, although ZombieNAND does a great job of leveraging improved latencies for deceased blocks early on in the life of the drive for those traces it can benefit (prior to baseline death), it does not hurt the traces which do not benefit as much from the scheme until the baseline is within 5% from death. Two, while increased reserve does tend to improve lifetime, it also seems to have a negative impact on the post-baseline-death latencies for some of the traces. This is likely due to increased contention for the zombified block pool and "spillover" accesses that stretch across zombified blocks and normal blocks.

V. Related Work

We divide related works into three categories: First, those which laid the hardware and physics foundation to perform this switching. Second, those which employ bit-switching preemptively for performance reasons. And third, those which employ bit switching for the same reasons as we do, but to lesser or different effect.

First, in [13] Taehee Cho et al. propose a dual-mode flash memory technology, which offers both SLC-mode operations and MLC-mode operations in the same NAND flash fabrication. While MLC mode operations use a low-voltage-based incremental step pulse programming (ISPP) method to tightly control the cell thresholds, SLC-mode employs three times higher voltage to reach high throughput. Though this dual-mode NAND flash demonstrates that multiple modes are possible on the same flash die device, they do not tackle latency variation, reliability issues, or side-effects resulting from this architecture.

Moving onto those works which use bit-switching preemptively for performance reasons, we see in [14] Moinuddin K. Qureshi et al. propose a Morphable Memory System (MMS) that utilizes different latency values as observed in MLC phase change random access memory (PCRAM). Based on incoming request and workloads, MMS uses different resistance values on PCRAM in an attempt to reduce memory operational latency. Even though MMS provides insights regarding bit-switching, it does not deal with reliability whatsoever, and furthermore the access granularities and resistance characteristics explored differ markedly from those in NAND flash making MMS inapplicable to this work. In [15], Sungjin Lee et al. propose a flexible flash file system (FlexFS), which partitions NAND flash into two sections of SLC and MLC, dynamically sizing each based on applications requirements. This mode switching is used to powerful effect to satisfy quality of service requirements, however, similar to MMS, FlexFS also ignores reliability issues. Last in this group, in [16], Laura M. Grupp, et. al., propose a flexible flash translation layer, which mimics FlexFS in a number of ways. Specifically, it schedules performance-critical operations and bursty workloads using SLC, and achieves such in a TLC/MLC/SLC device by revealing latency variation patterns for both SLC, MLC and TLC amongst pages. Yet again, this only handles performance characteristics, and importantly, all three of these works preemptively adapt the capacity of the drive, which renders it incompatible with traditional file systems and unaligned with manufacturer reliability guarantees.

Last, looking at a selection of works which most aligns with ours, we find three in particular that take a similar approach but either do so only superficially or do so for a different NVM type. In [17], Xavier Jimenez et. al. present, in their interactive poster, a topical exploration of how reviving dead blocks might impact lifetime. However, this work fails to capture all of the benefits of our approach for a number of reasons: First and foremost, they demonstrate very marginal improvements because they fail to fully explore the parameter space of modern SSDs, including, but not limited to, not looking at varying reserved areas, varying working set sizes, and a wide range of applications as we do in this work. Second, they do not examine the impact of adapting the wear-leveling schemes in modern SSDs to improve the longevity and performance of SSDs as they age. Last, rather than exploring this for a general SSD architecture, they only explore a narrow use-case: Hybrid FTL architectures. By bringing their dead MLC blocks back as SLC just for buffering for their log-structured FTL, this results in the healthier MLC still being written to when the logs are written out, resulting in write amplification compared to our use case and reads failing to leverage the vastly improved speeds in SLC. Moving onto [18], Azevedo et. al. propose "Zombie Memory," which on its surface sounds very similar to our work. However, their work actually explores an orthogonal approach to lifetime extension, and one which delivers no performance benefits for their MLC PCM. In fact, as their PCM ages and their techniques come to the fore, the performance degrades. Specifically, they look at using error correcting codes across partially dead blocks to extend one block's lifetime by sourcing parts of other dead blocks, and they never consider changing the bit mode from MLC to SLC; whatever bit-mode it starts as, it stays that way. This scheme could be applied in tandem with ours to multiplicative lifetime gains, a consideration of ours for future work. Last, in [19], Dong et. al. propose AdaMS, an adaptive PCM design to switch from MLC to SLC when PCM-specific failure events occur, and they design circuitry around this specific functionality to cope with the transition. Because the failure events, lifetime model used, circuitry designed, and algorithm proposed all rely on very specific PCM characteristics, which NAND flash do not share, their approach, while similar, is not applicable to NAND flash storage.

VI. Conclusion

As widespread adoption and new uses arise for NAND-flash-based SSDs, pressure for higher density will only increase. To achieve this, manufacturers very well may continue stuffing more bits into flash cells, or continue shrinking them further when they are already struggling to retain data at the current feature size generation. Without intervention, this will reignite worries over flash longevity people were finally beginning to get over. Enter ZombieNAND, our novel NAND-cell-reviving scheme with accompanying modified garbage-collection and "wear-unleveling" algorithm to do something with those otherwise deceased and unusable flash cells, and which does so without jeopardizing any manufacturer-specified guarantees of P/E cycles.

Using a heavily-modified simulator we added a flash physics engine to so we could predict lifetime of changing cells with high-fidelity, we evaluate and analyze ZombieNAND across over five-hundred synthetic and sixty application trace-driven experiments. We demonstrate that ZombieNAND succeeds in extending the lifetime of TLC and MLC SSDs, sometimes in excess of an order of a magnitude, and for most of the traces and synthetic configurations run over two times. Moreover, we show that it absolutely does not deteriorate the lifetime for workloads that it cannot help, and equally importantly, does not degrade the latency of runs until after a normal SSD without ZombieNAND on-board would already be dead. In fact, for around half of our experiments, it consistently delivers in excess of 25% faster latencies than the baseline. Finally, we provide thorough analysis of these results and detail the property of the traces used to gather them.

VII. Acknowledgements

This research is supported in part by NSF grants 1213052, 1302557, 1017882, 0937949 and 0833126.

References

[1] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches." in *FAST*.

[2] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling," ser. DATE '13.

[3] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan, "How i learned to stop worrying and love flash endurance," ser. HotStorage'10.

[4] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime management of flash-based ssds using recovery-aware dynamic throttling," ser. FAST'12.

[5] B. Godard, J.-M. Daga, L. Torres, and G. Sassatelli, "Evaluation of design for reliability techniques in embedded flash memories," ser. DATE '07.

[6] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual," 2008.

[7] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *USENIX ATC*, 2008.

[8] "NAND flash memory MT29F32G08ABAAA, MT29F64G08AFAAA SLC datasheet," Micron Technology, Inc, Tech. Rep.

[9] "NAND flash memory MT29F8G08MAAWC, MT29F16G08QASWC MLC datasheet," Micron Technology, Inc, Tech. Rep.

[10] "NAND flash memory MT29F64G08EBAA TLC datasheet," Micron Technology, Inc, Tech. Rep.

[11] K. Bates and B. McNutt. Umass Trace Repository. [Online]. Available: traces.cs.umass.edu/index.php/Main/Traces

[12] SNIA IOTTA repository. [Online]. Available: http://iotta.snia.org/tracetypes/3

[13] T. Cho, Y.-T. Lee, E.-C. Kim, J.-W. Lee, S. Choi, S. Lee, D.-H. Kim, W.-G. Han, Y.-H. Lim, J.-D. Lee, J.-D. Choi, and K.-D. Suh, "A dual-mode nand flash memory: 1-gb multilevel and high-performance 512-mb single-level modes," *Solid-State Circuits, IEEE Journal of*, 2001.

[14] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," ser. ISCA '10.

[15] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "Flexfs: A flexible flash file system for mlc nand flash memory," ser. USENIX'09.

[16] L. M. Grupp, J. D. Davis, and S. Swanson, "The harey tortoise: Managing heterogeneous write performance in ssds," ser. USENIX ATC'13.

[17] X. Jimenez, D. Novo, and P. Ienne, "Phœnix: reviving mlc blocks as slc to extend nand flash devices lifetime," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013.

[18] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie memory: Extending memory lifetime by reviving dead blocks," ser. ISCA '13.

[19] X. Dong and Y. Xie, "Adams: Adaptive mlc/slc phase-change memory design for file storage," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*.