Write Endurance in Flash Drives: Measurements and Analysis

Simona Boboila
Northeastern University
360 Huntington Ave.
Boston, MA 02115
simona@ccs.neu.edu

Peter Desnoyers
Northeastern University
360 Huntington Ave.
Boston, MA 02115
pjd@ccs.neu.edu

Abstract

We examine the write endurance of USB flash drives using a range of approaches: chip-level measurements, reverse engineering, timing analysis, whole-device endurance testing, and simulation. The focus of our investigation is not only measured endurance, but underlying factors at the level of chips and algorithms—both typical and ideal—which determine the endurance of a device.

Our chip-level measurements show endurance far in excess of nominal values quoted by manufacturers, by a factor of as much as 100. We reverse engineer specifics of the Flash Translation Layers (FTLs) used by several devices, and find a close correlation between measured whole-device endurance and predictions from reverse-engineered FTL parameters and measured chip endurance values. We present methods based on analysis of operation latency which provide a non-intrusive mechanism for determining FTL parameters. Finally we present Monte Carlo simulation results giving numerical bounds on endurance achievable by any on-line algorithm in the face of arbitrary or malicious access patterns.

1 Introduction

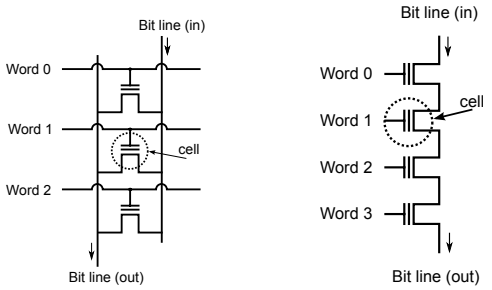
In recent years flash memory has entered widespread use, in embedded media players, photography, portable drives, and solid-state disks (SSDs) for traditional computing storage. Flash has become the first competitor to magnetic disk storage to gain significant commercial acceptance, with estimated shipments of 5×10^{19} bytes in 2009 [10], or more than the amount of disk storage shipped in 2005 [31].

Flash memory differs from disk in many characteristics; however, one which has particular importance for the design of storage systems is its limited *write endurance*. While disk drive reliability is mostly unaffected by usage, bits in a flash chip will fail after a limited number of writes, typical quoted at 10^4 to 10^5 depending on the specific device. When used with applications expecting a disk-like storage interface, e.g. to implement a FAT or other traditional file system, this results in over-use of a small number of blocks and early failure. Almost all flash devices on the market—USB drives, SD drives, SSDs, and a number of others—thus implement internal *wear-leveling* algorithms, which map application block addresses to physical block addresses, and vary this mapping to spread writes uniformly across the device.

The endurance of a flash-based storage system such as a USB drive or SSD is thus a function of both the parameters of the chip itself, and the details of the wear-leveling algorithm (or *Flash Translation Layer*, FTL) used. Since measured endurance data is closely guarded by semiconductor manufacturers, and FTL details are typically proprietary and hidden within the storage device, the broader community has little insight into the endurance characteristics of these systems. Even empirical testing may be of limited utility without insight into which access patterns represent worst-case behavior.

To investigate flash drive endurance, we make use of an array of techniques: chip-level testing, reverse engineering and timing analysis, whole device testing, and analytic approaches. Intrusive tests include chip-level testing—where the flash chip is removed from the drive and tested without any wear-leveling—and reverse engineering of FTL algorithms using logic analyzer probing. Analysis of operation timing and endurance testing conducted on the entire flash drive provides additional information; this is augmented by analysis and simulation providing insight into achievable performance of the wear-leveling algorithms used in conjunction with typical flash devices.

The remainder of the paper is structured as follows. Section 2 presents the basic information about flash memory technology, FTL algorithms, and related work. Section 3 discusses our experimental results, including chip-level testing (Section 3.1), details of reverse-engineered FTLs (3.2), and device-level testing (3.3).



(a) NOR Flash (b) NAND Flash

Figure 1: Flash circuit structure. NAND flash is distinguished by the series connection of cells along the bit line, while NOR flash (and most other memory technologies) arrange cells in parallel between two bit lines.

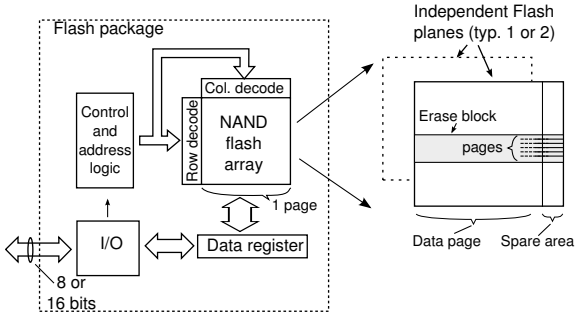


Figure 2: Typical flash device architecture. Read and write are both performed in two steps, consisting of the transfer of data over the external bus to or from the data register, and the internal transfer between the data register and the flash array.

Section 4 presents a theoretical analysis of wear-leveling algorithms, and we conclude in Section 5.

2 Background

NAND flash is a form of electrically erasable programmable read-only memory based on a particularly space-efficient basic cell, optimized for mass storage applications. Unlike most memory technologies, NAND flash is organized in *pages* of typically 2K or 4K bytes which are read and written as a unit. Unlike block-oriented disk drives, however, pages must be erased in units of *erase blocks* comprising multiple pages—typically 32 to 128—before being re-written.

Devices such as USB drives and SSDs implement a re-writable block abstraction, using a Flash Translation Layer to translate logical requests to physical read, program, and erase operations. FTL algorithms aim to maximize endurance and speed, typically a trade-off due to the extra operations needed for wear-leveling. In addition, an FTL must be implementable on the flash controller; while SSDs may contain 32-bit processors and megabytes of RAM, allowing sophisticated algorithms, some of the USB drives analyzed below use 8-bit controllers with as little as 5KB of RAM.

2.1 Physical Characteristics

We first describe in more detail the circuit and electrical aspects of flash technology which are relevant to system software performance; a deeper discussion of these and other issues may be found in the survey by Sanvido *et al* [29]. The basic cell in a NAND flash is a MOSFET transistor with a floating (i.e. oxide-isolated) gate. Charge is tunnelled onto this gate during write operations, and removed (via the same tunnelling mechanism) during erasure. This stored charge causes changes in V_T, the threshold or turn-on voltage of the cell transistor, which may then be sensed by the read circuitry. NAND flash is distinguished from other flash technologies (e.g. NOR flash, E^2PROM) by the tunnelling mechanism (Fowler-Nordheim or FN tunnelling) used for both programming and erasure, and the series cell organization shown in Figure 1(b).

Many of the more problematic characteristics of NAND flash are due to this organization, which eliminates much of the decoding overhead found in other memory technologies. In particular, in NAND flash the only way to access an individual cell for either reading or writing is *through* the other cells in its bit line. This adds noise to the read process, and also requires care during writing to ensure that adjacent cells in the string are not disturbed. (In fact, stray voltage from writing and reading may induce errors in other bits on the string, known as *program disturbs* and *read disturbs*.) During erasure, in contrast, all cells on the same bit string are erased.

Individual NAND cells store an analog voltage; in practice this may be used to store one of two voltage levels (*Single-Level Cell* or SLC technology) or between 4 and 16 voltage levels—encoding 2 to 4 bits—in what is known as Multi-Level Cell (MLC) technology. These cells are typically organized as shown in the block diagram in Figure 2. Cells are arranged in pages, typically containing 2K or 4K bytes plus a spare area of 64 to 256 bytes for system overhead. Between 16 and 128 pages make up an *erase block*, or *block* for short, which are then grouped into a flash *plane*. Devices may contain independent flash planes, allowing simultaneous operations for higher performance. Finally, a static RAM buffer holds data before writing or after reading, and data is transferred to and from this buffer via an 8- or 16-bit wide bus.

2.2 Flash Translation Layer

As described above, NAND flash is typically used with a *flash translation layer* implementing a disk-like interface of addressable, re-writable 512-byte blocks, e.g. over an interface such as SATA or SCSI-over-USB. The FTL maps logical addresses received over this interface (*Logical Page Numbers* or LPNs) to physical addresses in the flash chip (*Physical Page Numbers*, PPNs) and manages the details of erasure, wear-leveling, and garbage collection [2, 3, 17].

Mapping schemes: A flash translation layer could in theory maintain a map with an entry for each 512-byte logical page containing its corresponding location; the overhead of doing so would be high, however, as the map for a 1GB device would then require 2M entries, consuming about 8MB; maps for larger drives would scale proportionally. FTL resource requirements are typically reduced by two methods: *zoning* and larger-granularity mapping.

Zoning refers to the division of the logical address space into regions or *zones*, each of which is assigned its own region of physical pages. In other words, rather than using a single translation layer across the entire device, multiple instances of the FTL are used, one per zone. The map for the current zone is maintained in memory, and when an operation refers to a different zone, the map for that zone must be loaded from the flash device. This approach performs well when there is a high degree of locality in access patterns; however it results in high overhead for random operation. Nonetheless it is widely used in small devices (e.g. USB drives) due to its reduced memory requirements.

By mapping larger units, and in particular entire erase blocks, it is possible to reduce the size of the mapping tables even further [8]. On a typical flash device (64-page erase blocks, 2KB pages) this reduces the map for a 1GB chip to 8K entries, or even fewer if divided into zones. This reduction carries a cost in performance: to modify a single 512-byte logical block, this block-mapped FTL would need to copy an entire 128K block, for an overhead of $256\times$.

Hybrid mapping schemes [19, 20, 21, 25] augment a block map with a small number of reserved blocks (*log* or *update* blocks) which are page mapped. This approach is targeted to usage patterns that exhibit block-level temporal locality: the pages in the same logical block are likely to be updated again in the near future. Therefore, a compact fine-grained mapping policy for log blocks ensures a more efficient space utilization in case of frequent updates.

Garbage collection: Whenever units smaller than an erase block are mapped, there can be *stale* data: data which has been replaced by writes to the same logical address (and stored in a different physical location) but which has not yet been erased. In the general case recovering these pages efficiently is a difficult problem. However in the limited case of hybrid FTLs, this process consists of merging log blocks with blocks containing stale data, and programming the result into one or more free blocks. These operations are of the following types: *switch merges*, *partial merges*, and *full merge* [13].

A *switch merge* occurs during sequential writing; the log block contains a sequence of pages exactly replacing an existing data block, and may replace it without any further operation; the old block may then be erased. A *partial merge* copies valid pages from a data block to the log block, after which the two may be switched. A *full merge* is needed when data in the log block is out of order; valid pages from the log block and the associated data block are copied together into a new free block, after which the old data block and log block are both erased.

Wear-leveling: Many applications concentrate their writes on a small region of storage, such as the file allocation table (FAT) in MSDOS-derived file systems. Naïve mechanisms might map these logical regions to similar-sized regions of physical storage, resulting in premature device failure. To prevent this, *wear-leveling* algorithms are used to ensure that writes are spread across the entire device, regardless of application write behavior; these algorithms [11] are classified as either dynamic or static. *Dynamic wear-leveling* operates only on overwritten blocks, rotating writes between blocks on a free list; thus if there are m blocks on the free list, repeated writes to the same logical address will cause $m + 1$ physical blocks to be repeatedly programmed and erased. *Static wear-leveling* spreads the wear over both static and dynamic memory regions, by periodically swapping active blocks from the free list with static randomly-chosen blocks. This movement incurs additional overhead, but increases overall endurance by spreading wear over the entire device.

2.3 Related Work

There is a large body of existing experimental work examining flash memory performance and endurance; these studies may be broadly classified as either circuit-oriented or system-oriented. Circuit-level studies have examined the effect of program/erase stress on internal electrical characteristics, often using custom-fabricated devices to remove the internal control logic and allow measurements of the effects of single program or erase steps. A representative study is by Lee et al. at Samsung [24], examining both program/erase cycling and hot storage effects across a range of process technologies. Similar studies include those by Park et al. [28] and Yang et al. [32], both also at Samsung. The most recent work

Device	Size (bits)	Cell	Nominal endurance	Process
NAND128W3A2BN	128M	SLC	10^5	90nm
HY27US08121A	512M	SLC	10^5	90nm
MT29F2G08AAD	2G	SLC	10^5	50nm
MT29F4G08AAC	4G	SLC	10^5	72nm
NAND08GW3B2C	8G	SLC	10^5	60nm
MT29F8G08MAAWC	8G	MLC	10^4	72nm
29F16G08CANC1	16G	SLC	10^5	50nm
MT29F32G08QAA	32G	MLC	10^4	50nm

Table 1: Devices tested

in this area includes a workshop report of our results [9] and an empirical characterization of flash memory carried out by Grupp et at. [12], analyzing performance of basic operations, power consumption, and reliability.

System-level studies have instead examined characteristics of entire flash-based storage systems, such as USB drives and SSDs. The most recent of these presents uFLIP [7], a benchmark for such storage systems, with measurements of a wide range of devices; this work quantifies the degraded performance observed for random writes in many such devices. Additional work in this area includes [14],[27], and [1]

Ben-Aroyo and Toledo [5] have presented detailed theoretical analyses of bounds on wear-leveling performance; however for realistic flash devices (i.e. with erase block size $>$ 1 page) their results show the existence of a bound but not its value.

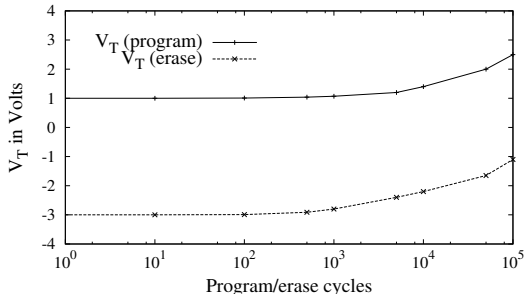


Figure 3: Typical V_T degradation with program/erase cycling for sub-90 nm flash cells. Data is abstracted from [24], [28], and [32].

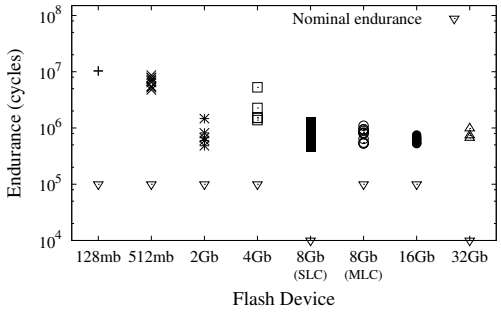


Figure 4: Write/Erase endurance by device. Each plotted point represents the measured lifetime of an individual block on a device. Nominal endurance is indicated by inverted triangles.

3 Experimental Results

3.1 Chip-level Endurance

Chip-level endurance was tested across a range of devices; more detailed results have been published in a previous workshop paper [9] and are summarized below.

Methodology: Flash chips were acquired both through distributors and by purchasing and disassembling mass-market devices. A programmable flash controller was constructed using software control of general-purpose I/O pins on a micro-controller to implement the flash interface protocol for 8-bit devices. Devices tested ranged from older 128Mbit (16MB) SLC devices to more recent 16Gbit and 32Gbit MLC chips; a complete list of devices tested may be seen in Table 1. Unless otherwise specified, all tests were performed at 25° C.

Endurance: Limited write endurance is a key characteristic of NAND flash—and all floating gate devices in general—which is not present in competing memory and storage technologies. As blocks are repeatedly erased and programmed the oxide layer isolating the gate degrades [23], changing the cell response to a fixed programming or erase step as shown in Figure 3. In practice this degradation is compensated for by adaptive programming and erase algorithms internal to the device, which use multiple program/read or erase/read steps to achieve the desired state. If a cell has degraded too much, however, the program or erase operation will terminate in an error; the external system must then consider the block *bad* and remove it from use.

Program/erase endurance was tested by repeatedly programming a single page with all zeroes (vs. the erased state of all 1 bits), and then erasing the containing block; this cycle was repeated until a program or erase operation terminated with an error status. Although nominal device endurance ranges from 10^4 to 10^5 program/erase cycles, in Figure 4 we see that the number of cycles until failure was higher in almost every case, often by nearly a factor of 100.

During endurance tests individual operation times were measured exclusive of data transfer, to reduce dependence on test setup; a representative trace is seen in Figure 5. The increased erase times and decreased program times appear to directly illustrate V_T degradation shown in Figure 3—as the cell ages it becomes easier to program and harder to erase, requiring fewer iterations of the internal write algorithm and more iterations for erase.

Additional Testing: Further investigation was performed to determine whether the surprisingly high en-

	Mean Endurance	Standard Deviation	Min. and max (vs. mean)
128mb	10.3 ($\times 10^6$)	0.003	+0.002 / -0.002
512mb	6.59	1.32	+2.09 / -1.82
2Gb	0.806	0.388	+0.660 / -0.324
4Gb	2.39	1.65	+2.89 / -1.02
8Gb SLC	0.827	0.248	+0.465 / -0.359
8Gb MLC*	0.783	0.198	+0.313 / -0.252
16Gb	0.614	0.078	+0.136 / -0.089
32Gb	0.793	0.164	+0.185 / -0.128

Table 2: Endurance in units of 10^6 write/erase cycles. The single outlier for 8 Gb MLC has been dropped from these statistics.

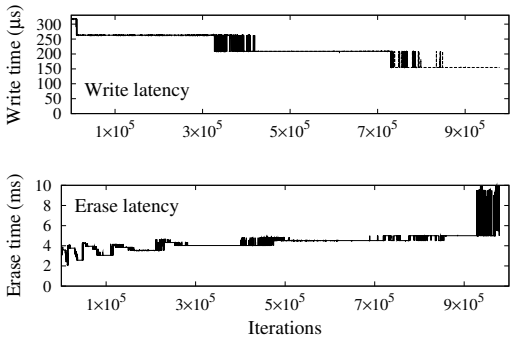


Figure 5: Wear-related changes in latency. Program and erase latency are plotted separately over the lifetime of the same block in the 8Gb MLC device. Quantization of latency is due to iterative internal algorithms.

durance of the devices tested is typical, or is instead due to anomalies in the testing process. In particular, we varied both program/erase behavior and environmental conditions to determine their effects. Due to the high variance of the measured endurance values, we have not collected enough data to draw strong inferences, and so report general trends instead of detailed results.

Usage patterns – The results reported above were measured by repeatedly programming the first page of a block with all zeroes (the programmed state for SLC flash) and then immediately erasing the entire block. Several devices were tested by writing to all pages in a block before erasing it; endurance appeared to decrease with this pattern, but by no more than a factor of two. Additional tests were performed with varying data patterns, but no difference in endurance was detected.

Environmental conditions – The processes resulting in flash failure are exacerbated by heat [32], although internal compensation is used to mitigate this effect [22]. The 16Gbit device was tested at 80° C, and no noticeable difference in endurance was seen.

Conclusions: The high endurance values measured were unexpected, and no doubt contribute to the measured performance of USB drives reported below, which achieve high endurance using very inefficient wear-leveling algorithms. Additional experimentation is needed to determine whether these results hold across the most recent generation of devices, and whether flash algorithms may be tailored to produce access patterns which maximize endurance, rather than assuming it as a constant. Finally, the increased erase time and decreased programming time of aged cells bear implications for optimal flash device performance, as well as offering a predictive failure-detection mechanism.

Device	Size	Chip Signature	USB ID
Generic	512Mbit	HY27US08121A	1976:6025
House	16Gbit	29F16G08CANC1	125F:0000
Memorex	4Gbit	MF12G2BABA	12F7:1A23

Table 3: Investigated devices

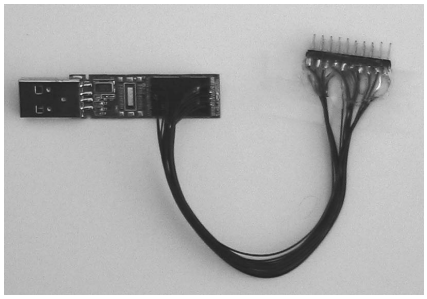


Figure 6: USB Flash drive modified for logic analyzer probing.

3.2 FTL Investigation

Having examined performance of NAND flash itself, we next turn to systems comprising both flash and FTL. While work in the previous section covers a wide range of flash technologies, we concentrate here on relatively small mass-market USB drives due to the difficulties inherent in reverse-engineering and destructive testing of more sophisticated devices.

Methodology: we reverse-engineered FTL operation in three different USB drives, as listed in Table 3: *Generic*, an unbranded device based on the Hynix HY27US08121A 512Mbit chip, *House*, a MicroCenter branded 2GB device based on the Intel 29F16G08CANC1, and *Memorex*, a 512MB Memorex "Mini TravelDrive" based on an unidentified part.

In Figure 6 we see one of the devices with probe wires attached to the I/O bus on the flash chip itself. Reverse-engineering was performed by issuing specific logical operations from a Linux USB host (by issuing direct I/O reads or writes to the corresponding block device) and using an IO-3200 logic analyzer to capture resulting transactions over the flash device bus. From this captured data we were then able to decode the flash-level opera-

	Generic	House	Memorex
Structure	16 zones	4 zones	4 zones
Zone size	256 physical blocks	2048 physical blocks	1024 physical blocks
Free blocks list size	6 physical blocks per zone	30-40 physical blocks per zone	4 physical blocks per zone
Mapping scheme	Block-level	Block-level / Hybrid	Hybrid
Merge operations	Partial merge	Partial merge / Full merge	Full merge
Garbage collection frequency	At every data update	At every data update	Variable
Wear-leveling algorithm	Dynamic	Dynamic	Static

Table 4: Characteristics of reverse-engineered devices

tions (read, write, erase, copy) and physical addresses corresponding to a particular logical read or write.

We characterize the flash devices based on the following parameters: *zone organization* (number of zones, zone size, number of free blocks), *mapping schemes*, *merge operations*, *garbage collection frequency*, and *wear-leveling algorithms*. Investigation of these specific attributes is motivated by their importance; they are fundamental in the design of any FTL [2, 3, 17, 19, 20, 21, 25], determining space requirements, i.e. the size of the mapping tables to keep in RAM (zone organization, mapping schemes), overhead/performance (merge operations, garbage collection frequency), device endurance (wear-leveling algorithms). The results are summarized in Table 4, and discussed in the next sections.

Zone organization: The flash devices are divided in zones, which represent contiguous regions of flash memory, with disjoint logical-to-physical mappings: a logical block pertaining to a zone can be mapped only in a physical block from the same zone. Since the zones function independently from each other, when one of the zones becomes unusable, other zones on the same device can still be accessed. We report actual values of zone sizes and free list sizes for the investigated devices in Table 4.

Mapping schemes: Block-mapped FTLs require smaller mapping tables to be stored in RAM, compared to page-mapped FTLs (Section 2.2). For this reason, the block-level mapping scheme is more practical and was identified in both Generic and multi-page updates of House flash drives. For single-page updates, House uses the simplified hybrid mapping scheme (which we will describe next), similar to Ban's NFTL [3]. The Memorex flash drive uses hybrid mapping: the data blocks are block-mapped and the log blocks are page-mapped.

Garbage collection: For the Generic drive, garbage collection is handled immediately after each write, eliminating the overhead of managing stale data. For House and Memorex, the hybrid mapping allows for several sequential updates to be placed in the same log block. Depending on specific writing patterns, garbage collection can have a variable frequency. The number of sequential updates that can be placed in a 64-page log block (before a new free log block is allocated to hold updated pages of the same logical block) ranges from 1 to 55 for Memorex and 1 to 63 for House.

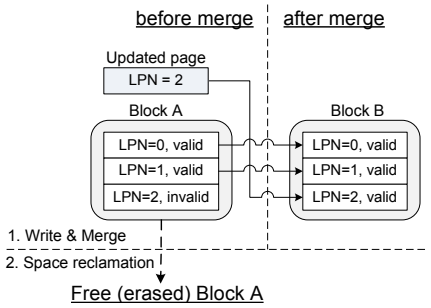


Figure 7: Generic device page update. Using block-level mapping and a partial merge operation during garbage collection. LPN = Logical Page Number. New data is merged with block A and an entire new block (B) is written.

We illustrate how garbage collection works after being triggered by a page update operation.

The Generic flash drive implements a simple page update mechanism (Figure 7). When a page is overwritten, a block is selected from the free block list, and the data to be written is merged with the original data block and written to this new block in a partial merge, resulting in the erasure of the original data block.

The House drive allows multiple updates to occur before garbage collection, using an approach illustrated in Figure 8. Flash is divided into two *planes*, even and odd (blocks B-even and B-odd in the figure); one log block can represent updates to a single block in the data area. When a single page is written, meta-data is written to the first page in the log block and the new data is written to the second page; a total of 63 pages may be written to the same block before the log must be merged. If a page is written to another block in the plane, however, the log must be merged immediately (via a full merge) and a new log started.

We observe that the House flash drive implements an optimized mechanism for multi-page updates, requiring 2 erasures rather than 4. This is done by eliminating the intermediary storage step in log blocks B-even and B-odd, and writing the updated pages directly to blocks C-even and C-odd.

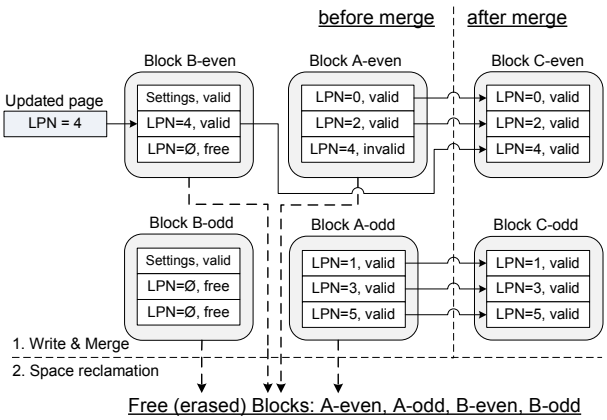


Figure 8: House device single-page update. Using hybrid mapping and a full merge operation during garbage collection. LPN = Logical Page Number. LPN 4 is written to block B, "shadowing" the old value in block A. On garbage collection, LPN 4 from block B is merged with LPNs 0 and 2 from block A and written to a new block.

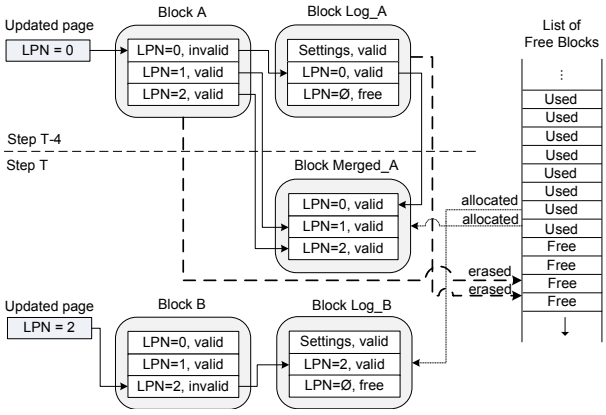


Figure 9: Memorex device page update. Using hybrid mapping and a full merge operation during garbage collection. LPN = Logical Page Number. LPN 2 is written to the log block of block B and the original LPN 2 marked invalid. If this requires a new log block, an old log block (Log_A) must be freed by doing a merge with its corresponding data block.

The Memorex flash drive employs a complex garbage collection mechanism, which is illustrated in Figure 9. When one or more pages are updated in a block (B), a merge is triggered if there is no active log block for block B or the active log block is full, with the following operations being performed:

- The new data pages together with some settings information are written in a free log block (Log_B).
- A *full merge* operation occurs, between two blocks (data block A and log block Log_A) that were accessed 4 steps back. The result is written in a free block (Merged_A). Note that the merge operation may be deferred until the log block is full.
- After merging, the two blocks (A and Log_A) are erased and added to the list of free blocks.

Wear-leveling aspects: From the reverse-engineered devices, static wear-leveling was detected only in the case of the Memorex flash drive, while both Generic and House devices use dynamic wear-leveling. As observed during the experiments, the Memorex flash drive is periodically (after every 138th garbage collection operation) moving data from one physical block containing rarely updated data, into a physical block from the list of free blocks. The block into which the static data has been moved is taken out of the free list and replaced by the rarely used block.

Conclusions: The three devices examined were found to have flash translation layers ranging from simple (Generic) to somewhat complex (Memorex). Our investigation provided detailed parameters of each FTL, including zone organization, free list size, mapping scheme, and static vs. dynamic wear-leveling methods. In combination with the chip-level endurance measurements presented above, we will demonstrate in Section 3.4 below the use of these parameters to predict overall device endurance.

3.3 Timing Analysis

Additional information on the internal operation of a flash drive may be obtained by timing analysis—measuring the latency of each of a series of requests and detecting patterns in the results. This is possible because of the disparity in flash operation times, typically 20μs, 200-300μs, and 2-4ms for read, write and erase respectively [9]. Selected patterns of writes can trigger differing sequences of flash operations, incurring different delays observable as changes in write latency. These changes offer clues which can help infer the following characteristics: (a) wear-leveling mechanism (static or dynamic) and parameters, (b) garbage collection mechanism, and (c) device end-of-life status.

Approach: Timing analysis uses sequences of writes to addresses $\{A_1, A_2, \ldots A_n\}$ which are repeated to provoke periodic behavior on the part of the device. The most straightforward sequence is to repeatedly write the same block; these writes completed in constant time for the Generic device, while results for the House device are seen in Figure 10. These results correspond to the FTL algorithms observed in Section 3.2 above; the Generic device performs the same block copy and erase for every write, while the House device is able to write to Block B (see Figure 8) 63 times before performing a merge operation and corresponding erase.

More complex flash translation layers require more complex sequences to characterize them. The hybrid

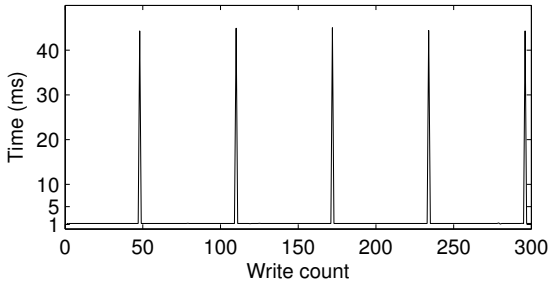


Figure 10: House device write timing. Write address is constant; peaks every 63 operations correspond to the merge operation (including erasure) described in Section 3.2.

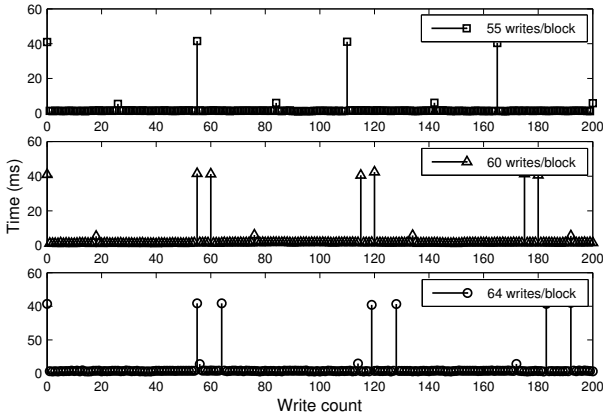


Figure 11: Memorex device garbage collection patterns. Access pattern used is $\{A_1 \times n, A_2 \times n, \ldots\}$ for $n = 55, 60, 64$ writes/block.

FTL used by the Memorex device maintains 4 log blocks, and thus pauses infrequently with a sequence rotating between 4 different blocks; however, it slows down for every write when the input stream rotates between addresses in 5 distinct blocks. In Figure 11 we see two patterns: a garbage collection after 55 writes to the same block, and then another after switching to a new block.

Organization: In theory it should be possible to determine the zones on a device, as well as the size of the free list in each zone, via timing analysis. Observing zones should be straightforward, although it has not yet been implemented; since each zone operates independently, a series of writes to addresses in two zones should behave like repeated writes to the same address. Determining the size of the free list, m, may be more difficult; variations in erase time between blocks may produce patterns which repeat with a period of m, but these variations may be too small for reliable measurement.

Wear-leveling mechanism: Static wear-leveling is indicated by combined occurrence of two types of peaks: smaller, periodic peaks of regular write/erase operations, and higher, periodic, but less frequent peaks that suggest

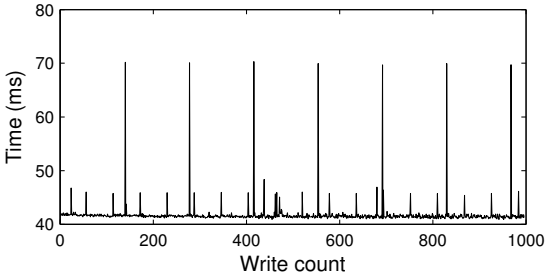


Figure 12: Memorex device static wear-leveling. Lower values represent normal writes and erasures, while peaks include time to swap a static block with one from the free list. Peaks have a regular frequency of one at every 138 write/erasure.

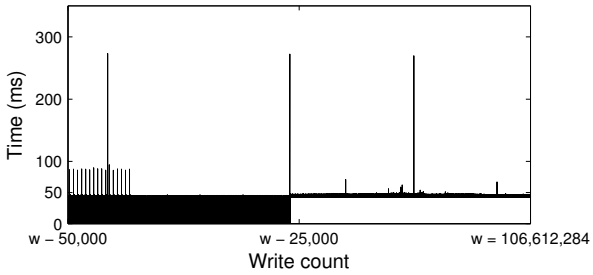


Figure 13: House device end-of-life signature. Latency of the final 5×10^4 writes before failure.

additional internal management operations. In particular, the high peaks are likely to represent moving static data into highly used physical blocks in order to uniformly distribute the wear. The correlation between the high peaks and static wear-leveling was confirmed via logic analyzer, as discussed in Section 3.2 and supported by extremely high values of measured device-level endurance, as reported in Section 3.3.

For the Memorex flash drive, Figure 12 shows latency for a series of sequential write operations in the case where garbage collection is triggered at every write. The majority of writes take approximately 45 ms, but high peaks of 70 ms also appear every 138th write/erase operation, indicating that other internal management operations are executed in addition to merging, data write and garbage collection. The occurrence of high peaks suggests that the device employs static wear-leveling by copying static data into frequently used physical blocks.

Additional tests were performed with a fourth device, *House-2*, branded the same as the House device but in fact a substantially newer design. Timing patterns for repeated access indicate the use of static wear-leveling, unlike the original House device. We observed peaks of 15 ms representing write operations with garbage collection, and higher regular peaks of 20 ms appearing at approximately every 8,000 writes. The 5 ms time differ-

Device	Parameters	Predicted endurance		Measured endurance
Generic	$m = 6, h = 10^7$	mh	6×10^7	$7.7 \times 10^7, 10.3 \times 10^7$
House	$m = 30, k = 64, h = 10^6$	between mh and mkh	between 3×10^7 and 1.9×10^9	10.6×10^7
Memorex	$z = 1024, k = 64, h = 10^6$(est.)	zkh	6×10^{10}	N/A

Table 5: Predicted and measured endurance limits.

ence from common writes to the highest peaks is likely due to data copy operations implementing static wear-leveling.

End-of-life signature: Write latency was measured during endurance tests, and a distinctive signature was seen in the operations leading up to device failure. This may be seen in Figure 13, showing latency of the final 5×10^4 operations before failure of the House device. First the 80ms peaks stop, possibly indicating the end of some garbage collection operations due to a lack of free pages. At 25000 operations before the end, all operations slow to 40ms, possibly indicating an erasure for every write operation; finally the device fails and returns an error.

Conclusions: By analyzing write latency for varying patterns of operations we have been able to determine properties of the underlying flash translation algorithm, which have been verified by reverse engineering. Those properties include wear-leveling mechanism and frequency, as well as number and organization of log blocks. Additional details which should be possible to observe via this mechanism include zone boundaries and possibly free list size.

3.4 Device-level Endurance

By device-level endurance we denote the number of successful writes at logical level before a write failure occurs. Endurance was tested by repeated writes to a constant address (and to 5 constant addresses in the case of Memorex) until failure was observed. Testing was performed on Linux 2.6.x using direct (unbuffered) writes to the block devices.

Several failure behaviors were observed:

- **silent:** The write operation succeeds, but read verifies that data was not written.
- **unknown error:** On multiple occasions, the test application exited without any indication of error. In many casses, further writes were possible.
- **error:** An I/O error is returned by the OS. This was observed for the House flash drive; further write operations to any page in a zone that had been worn out failed, returning error.
- **blocking:** The write operation hangs indefinitely. This was encountered for both Generic and House flash drives, especially when testing was resumed after failure.

Endurance limits with dynamic wear-leveling: We measured an endurance of approximately 106×10^6 writes for House; in two different experiments, Generic sustained up to 103×10^6 writes and 77×10^6 writes, respectively. As discussed in Section 3.2, the House flash drive performs 4 block erasures for 1-page updates, while the Generic flash drive performs only one block erasure. However, the list of free blocks is about 5 times larger for House (see Table 3), which may explain the higher device-level endurance of the House flash drive.

Endurance limits with static wear-leveling: Wearing out a device that employs static wear-leveling (e.g. the Memorex and House-2 flash drives) takes considerably longer time than wearing out one that employs dynamic wear-leveling (e.g. the Generic and House flash drives). In the experiments conducted, the Memorex and House-2 flash drives had not worn out before the paper was submitted, reaching more than 37×10^6 writes and 26×10^8 writes, respectively.

Conclusions: The primary insight from these measurements is that wear-leveling techniques lead to a significant increase in the endurance of the whole device, compared to the endurance of the memory chip itself, with static wear-leveling providing much higher endurance than dynamic wear-leveling.

Table 5 presents a synthesis of predicted and measured endurace limits for the devices studied. We use the following notation:

N = total number of erase blocks,
k = total number of pages in the erase block,
h = maximum number of program/erase cycles of a block (i.e. the chip-level endurance),
z = number of erase blocks in a zone, and
m = number of free blocks in a zone.

Ideally, the device-level endurance is Nkh. In practice, based on the FTL implementation details presented in Section 3.2 we expect device-level endurance limits of mh for Generic, between mh and mkh for House, and zkh for Memorex. In the following computations, we use the program/erase endurance values, i.e. h, from Figure 4, and m and z values reported in Table 4. For Generic, $mh = 6 \times 10^7$, which approaches the actual measured values of 7.7×10^7 and 10.3×10^7. For House, $mh = 3 \times 10^7$ and $mkh = 30 \times 64 \times 10^6 = 1.9 \times 10^9$, with the measured device-level endurance of 10.6×10^7 falling between these two limits. For Memorex, we do not have chip-level endurance measurements, but we will

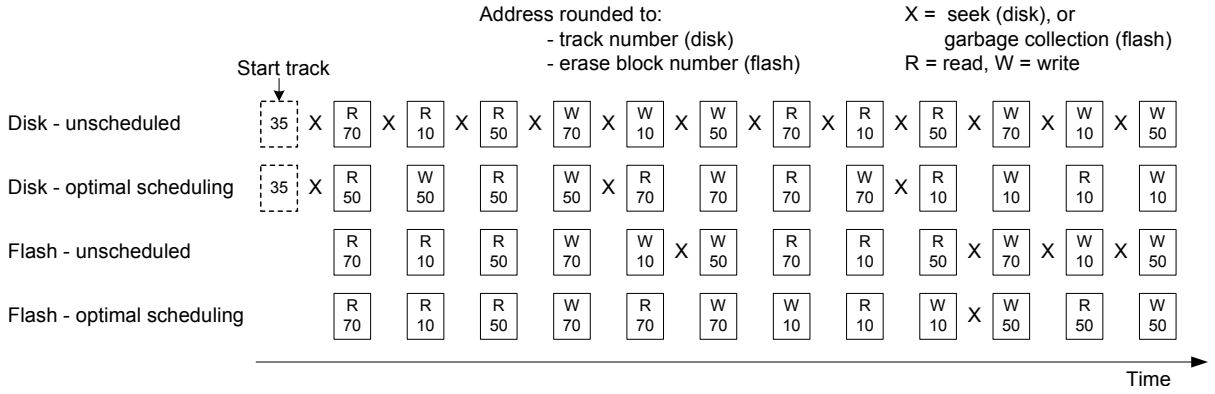


Figure 14: Unscheduled access vs. optimal scheduling for disk and flash. The requested access sequence contains both reads (R) and writes (W). Addresses are rounded to track numbers (disk), or erase block numbers (flash), and "X" denotes either a seek operation to change tracks (disk), or garbage collection to erase blocks (flash). We ignore the rotational delay of disks (caused by searching for a specific sector of a track), which may produce additional overhead. Initial head position (disk) = track 35.

use $h = 10^6$ in our computations, since it is the predominant value for the tested devices. We estimate the best-case limit of device-level endurance to be $zkh = 1024 \times 64 \times 10^6 \approx 6 \times 10^{10}$ for Memorex, which is about three orders of magnitude higher than for Generic and House devices, demonstrating the major impact of static wear-leveling.

3.5 Implications for Storage Systems

Space management: Space management policies for flash devices are substantially different from those used for disks, mainly due to the following reasons. Compared to electromechanical devices, solid-state electronic devices have no moving parts, and thus no mechanical delays. With no seek latency, they feature fast random access times and no read overhead. However, they exhibit asymmetric write vs. read performance. Write operations are much slower than reads, since flash memory blocks need to be erased before they can be rewritten. Write latency depends on the availability (or lack thereof) of free, programmable blocks. Garbage collection is carried out to reclaim previously written blocks which are no longer in use.

Disks address the seek overhead problem with scheduling algorithms. One well-known method is the elevator algorithm (also called SCAN), in which requests are sorted by track number and serviced only in the current direction of the disk arm. When the arm reaches the edge of the disk, its direction reverses and the remaining requests are serviced in the opposite order.

Since the latency of flash vs. disks has entirely different causes, flash devices require a different method than disks to address the latency problem. Request scheduling algorithms for flash have not yet been implemented in practice, leaving space for much improvement in this area. Scheduling algorithms for flash need to minimize garbage collection, and thus their design must be dependent upon FTL implementation. FTLs are built to take advantage of temporal locality; thus a significant performance increase can be obtained by reordering data streams to maximize this advantage. FTLs map successive updates to pages from the same data block together in the same log block. When writes to the same block are issued far apart from each other in time, however, new log blocks must be allocated. Therefore, most benefit is gained with a scheduling policy in which the same data blocks are accessed successively. In addition, unlike for disks, for flash devices there is no reason to reschedule reads.

To illustrate the importance of scheduling for performance as well as the conceptually different aspects of disk vs. flash scheduling, we look at the following simple example (Figure 14).

Disk scheduling. Let us assume that the following requests arrive: R 70, R 10, R 50, W 70, W 10, W 50, R 70, R 10, R 50, W 70, W 10, W 50, where R = read, W = write, and the numbers represent tracks. Initially, the head is positioned on track 35. We ignore the rotational delay of searching for a sector on a track. Without scheduling, the overhead (seek time) is 495. If the elevator algorithm is used, the requests are processed in the direction of the arm movement, which results in the following ordering: R 50, W 50, R 50, W 50, R 70, W 70, R 70, W 70, (arm movement changes direction), R 10, W 10, R 10, W 10. Also, the requests to the same track are grouped together, to minimize seek time; however, data integrity has to be preserved (reads/writes to the same disk track must be processed in the requested order, since they might access the same address). This gives an overhead of 95, which is 5x smaller with scheduling vs. no scheduling.

Flash scheduling. Let us assume that the same sequence of requests arrives: R 70, R 10, R 50, W 70, W 10, W 50, R 70, R 10, R 50, W 70, W 10, W 50, where R = read, W = write, and the numbers represent erase blocks. Also assume that blocks are of size 3 pages, and there are 3 free blocks, with one block empty at all times. Without scheduling, 4 erasures are needed to accommodate the last 4 writes. An optimal scheduling gives the following ordering of the requests: R 70, R 10, R 50, W 70, R 70, W 70, W 10, R 10, W 10, W 50, R 50, W 50. We observe that there is no need to reschedule reads; however, data integrity has to be preserved (reads/writes to the same block must be processed in the requested order, since they might access the same address). After scheduling, the first two writes are mapped together to the same free block, next two are also mapped together, and so on. A single block erasure is necessary to free one block and accommodate the last two writes. The garbage collection overhead is 4x smaller with scheduling vs. no scheduling.

Applicability: Although we have explored only a few devices, some of the methods presented here (e.g. timing analysis) can be used to characterize other flash devices as well. FTLs range in complexity across devices; however, at low-end there are many similarities. Our results are likely to apply to a large class of devices that use flash translation layers, including most removable devices (SD, CompactFlash, etc.), and low-end SSDs. For high-end devices, such as enterprise (e.g. the Intel X25-E [16] or BiTMICRO Altima [6] series) or high-end consumer (e.g. Intel X25-M [15]), we may expect to find more complex algorithms operating with more free space and buffering.

As an example, JMicron's JMF602 flash controller [18] has been used for many low-end SSDs with 8-16 flash chips; it contains 16K of onboard RAM, and uses flash configurations with about 7% free space. Having little free space or RAM for mapping tables, its flash translation layer is expected to be similar in design and performance to the hybrid FTL that we investigated above.

At present, several flash devices including low-end SSDs have a built-in controller that performs wear-leveling and error correction. A disk file system in conjunction with a FTL that emulates a block device is preferred for compatibility, and also because current flash file systems still have implementation drawbacks (e.g. JFFS2 has large memory consumption and implements only write-through caching instead of write-back) [26].

Flash file systems could become more prevalent as the capacity of flash memories increases. Operating directly over raw flash chips, flash file systems present some advantages. They deal with long erase times in the background, while the device is idle, and use file pointers (which are remapped when updated data is allocated to a free block), thus eliminating the second level of indirection needed by FTLs to maintain the mappings. They also have to manage only one free space pool instead of two, as required by FTL with disk file systems. In addition, unlike conventional file systems, flash file systems do not need to handle seek latencies and file fragmentation; rather, a new and more suited scheduling algorithm as described before can be implemented to increase performance.

4 Analysis and Simulation

In the previous section we have examined the performance of several real wear leveling algorithms under close to worst-case conditions. To place these results in perspective, we wish to determine the maximum theoretical performance which any such on-line algorithm may achieve. Using terminology defined above, we assume a device (or zone within a device) consisting of N erase blocks, each block containing k separately writable pages, with a limit of h program/erase cycles for each erase block, and m free erase blocks. (i.e. the physical size of the device is N erase blocks, while the logical size is $N - m$ blocks.)

Previous work by Ben-Aroya and Toledo [5] has proved that in the typical case where $k > 1$, and with reasonable bounds on m, upper bounds exist on the performance of wear-leveling algorithms. Their results, however, offer little guidance for calculating these bounds. We approach the problem from the bottom up, using Monte Carlo simulation to examine achievable performance in the case of uniform random writes to physical pages. We choose a uniform distribution because it is both achievable (by means such as Ban's randomized wear leveling method [4]) and in the worst case unavoidable by any on-line algorithm, when faced with uniform random writes across the logical address space. We claim therefore that our numeric results represent a tight bound on the performance of any on-line wear-leveling algorithm in the face of arbitrary input.

We look for answers to the following questions:

- How efficiently can we perform *static wear leveling*? We examine the case where $k = 1$, thus ignoring erase block fragmentation, and ask whether there are on-line algorithms which achieve near-ideal endurance in the face of arbitrary input.
- How efficiently can we perform *garbage collection*? For typical values of k, what are the conditions needed for an on-line algorithm to achieve good performance with arbitrary access patterns?

In doing this we use *endurance degradation* of an algorithm, or relative decrease in performance, as a figure

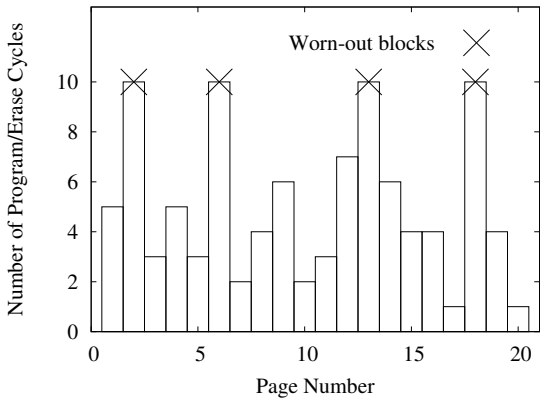


Figure 15: Trivial device failure ($N = 20$, $m = 4$, $h = 10$). Four blocks have reached their erase limit (10) after 100 total writes, half the theoretical maximum of Nh or 200.

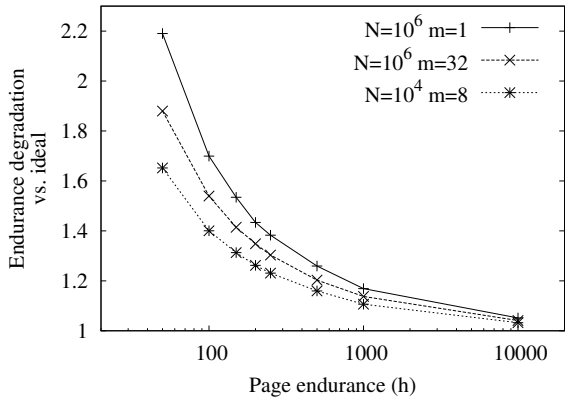


Figure 16: Wear-leveling performance. Endurance degradation (by simulation) for different numbers of erase blocks (N), block lifetime (h), and number of free blocks (m).

of merit. We ignore our results on block-level lifetime, and consider a device failed once m blocks have been erased h times—at this point we assume the m blocks have failed, thus leaving no free blocks for further writes. In the perfect case, all blocks are erased the same number of times, and the drive endurance is $Nkh + mS$ (or approximately Nkh) page writes—i.e. the total amount of data written is approximately h times the size of the device. In the worst case we have seen in practice, m blocks are repeatedly used, with a block erase and reprogram for each page written; the endurance in this case is mh. The endurance degradation for an algorithm is the ratio of ideal endurance to achieved endurance, or $\frac{Nk}{m}$ for this simple algorithm.

4.1 Static Wear Leveling

As described in Section 2.2, *static wear leveling* refers to the movement of data in order to distribute wear evenly across the physical device, even in the face of highly non-uniform writes to the logical device. For ease of analysis we make two simplifications:

- Erase unit and program unit are of the same size, i.e. $k = 1$. We examine $k > 1$ below, when looking at garbage collection efficiency.
- Writes are uniformly distributed across physical pages, as described above.

Letting $X_1, X_2, \ldots X_N$ be the number of times that pages $1 \ldots N$ have been erased, we observe that at any point each X_i is a random variable with mean w/N , where w is the total number of writes so far. If the variance of each X_i is high and $m \ll N$, then it is likely that m of them will reach h well before $w = Nh$, where the expected value of each X_i reaches h. This may be seen in Figure 15, where in a trivial case ($N = 20$, $m = 4$, $h = 10$) the free list has been exhausted after a total of only $Nh/2$ writes.

In Figure 16 we see simulation results for a more realistic set of parameters. We note the following points:

- For $h < 100$ random variations are significant, giving an endurance degradation of as much as 2 depending on h and m.
- For $h > 1000$, uniform random distribution of writes results in near-ideal wear leveling.
- N causes a modest degradation in endurance, for reasonable values of N; larger values degrade endurance as they increase the odds that some m blocks will exceed the erase threshold.
- Larger values of m result in lower endurance degradation, as more blocks must fail to cause device failure.

For reasonable values of h, e.g. 10^4 or 10^5, these results indicate that randomized wear leveling is able to provide near-optimal performance with very high probability. However the implementation of randomization imposes its own overhead; in the worst case doubling the number of writes to perform a random swap in addition to every logical write. In practice a random block is typically selected every d writes and swapped for a block from the free list, reducing the overhead to $1/d$.

Although this reduces overhead, it also reduces the degree of randomization introduced. In the worst case—repeated writes to the same logical block—a page will remain on the free list until it has been erased d times before being swapped out. A page can thus only land in the free list h/d times before wearing out, giving performance equivalent to the case where the lifetime h' is h/d. As an example, consider the case where $d = 200$ and $h = 10^4$; this will result in performance equivalent to $h = 50$ in our analysis, possibly reducing worst-case endurance by a factor of 2.

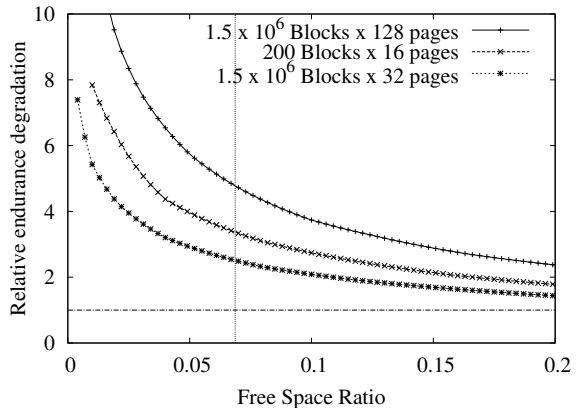


Figure 17: Degradation in performance due to wear-leveling for uniformly-distributed page writes. The vertical line marks a free percentage of 6.7%, corresponding to usage of 10^9 out of every 2^{30} bytes.

4.2 Garbage Collection

The results above assume an erase block size (k) of 1 page; in practice this value is substantially larger, in the devices tested above ranging from 32 to 128 pages. As a result, in the worst case m free pages may be scattered across as many erase blocks, and thus k pages must be erased (and $k-1$ copied) in order to free a single page; however depending on the number of free blocks, the expected performance may be higher.

Again, we assume writes are uniformly and randomly distributed across Nk pages in a device. We assume that the erase block with the highest number of stale pages may be selected and reclaimed; thus in this case random variations will help garbage collection performance, by reducing the number of good pages in this block.

Garbage collection performance is strongly impacted by the utilization factor, or ratio of logical size to physical size. The more free blocks available, the higher the mean and maximum number of free pages per block and the higher the garbage collection efficiency. In Figure 17 we see the degradation in relative endurance for several different combinations of device size N (in erase blocks) and erase block size k, plotted against the fraction of free space in the device. We see that the worst-case impact of garbage collection on endurance is far higher than that of wear-leveling inefficiencies, with relative decreases in endurance ranging from 3 to 5 at a typical utilization (for low-end devices) of 93%.

Given non-uniform access patterns, such as typical file system access, it is possible that different wear-leveling strategies may result in better performance than the randomized strategy analyzed above. However, we claim that no on-line strategy can do better than randomized wear-leveling in the face of uniformly random access patterns, and that these results thus provide a bound on worst-case performance of any on-line strategy.

For an ideal on-line wear-leveling algorithm, performance is dominated by garbage collection, due to the additional writes and erases incurred by compacting partially-filled blocks in order to free up space for new writes. Garbage collection performance, in turn, is enhanced by additional free space and degraded by large erase block sizes. For example, with 20% free space and small erase blocks (32 pages) it is possible to achieve an endurance degradation of less than 1.5, while with 7% free space and 128-page blocks endurance may be degraded by a factor of 5.[1]

5 Conclusions

As NAND flash becomes widely used in storage systems, behavior of flash and flash-specific algorithms becomes ever more important to the storage community. Write endurance is one important aspect of this behavior, and one on which perhaps the least information is available. We have investigated write endurance on a small scale—on USB drives and on flash chips themselves—due to their accessibility; however the values we have measured and approaches we have developed are applicable across devices of all sizes.

Chip-level measurements of flash endurance presented in this work show endurance values far in excess of those quoted by manufacturers; if these are representative of most devices, the primary focus of flash-related algorithms may be able to change from wear leveling to performance optimization. We have shown how reverse-engineered details of flash translation algorithms from actual devices in combination with chip-level measurements may be used to predict device endurance, with close correspondence between those predictions and measured results. In addition, we have presented non-intrusive timing-based methods for determining many of these parameters. Finally, we have provided numeric bounds on achievable wear-leveling performance given typical device parameters.

Our results explain how simple devices such as flash drives are able to achieve high endurance, in some cases remaining functional after several months of continual testing. In addition, analytic and simulation results highlight the importance of free space in flash performance, providing strong support for mechanisms like the TRIM command which allow free space sharing between file systems and flash translation layers. Future work in

[1] This is a strong argument for the new SATA *TRIM* operator [30], which allows the operating system to inform a storage device of free blocks; these blocks may then be considered free space by the flash translation layer, which would otherwise preserve their contents, never to be used.

this area includes examination of higher-end devices, i.e. SSDs, as well as pursuing the implications for flash translation algorithms of our analytical and simulation results.

References

[1] AJWANI, D., MALINGER, I., MEYER, U., AND TOLEDO, S. Characterizing the performance of flash memory storage devices and its impact on algorithm design. In *Experimental Algorithms*. 2008, pp. 208–219.

[2] BAN, A. Flash file system. United States Patent 5,404,485, 1995.

[3] BAN, A. Flash file system optimized for page-mode flash technologies. United States Patent 5,937,425, 1999.

[4] BAN, A. Wear leveling of static areas in flash memory. United States Patent 6,732,221, 2004.

[5] BEN-AROYA, A., AND TOLEDO, S. *Competitive Analysis of Flash-Memory Algorithms*. 2006, pp. 100–111.

[6] BITMICRO NETWORKS. Datasheet: e-disk Altima E3F4FL Fibre Channel 3.5". available from `www.bitmicro.com`, Nov. 2009.

[7] BOUGANIM, L., JONSSON, B., AND BONNET, P. uFLIP: understanding flash IO patterns. In *Int'l Conf. on Innovative Data Systems Research (CIDR)* (Asilomar, California, 2009).

[8] CHUNG, T., PARK, D., PARK, S., LEE, D., LEE, S., AND SONG, H. System Software for Flash Memory: A Survey. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing* (2006), pp. 394–404.

[9] DESNOYERS, P. Empirical evaluation of NAND flash memory performance. In *Workshop on Hot Topics in Storage and File Systems (HotStorage)* (Big Sky, Montana, October 2009).

[10] DRAMEXCHANGE. 2009 NAND flash demand bit growth forecast. `www.dramexchange.com`, 2009.

[11] GAL, E., AND TOLEDO, S. Algorithms and data structures for flash memories. *ACM Computing Surveys 37*, 2 (2005), 138–163.

[12] GRUPP, L., CAULFIELD, A., COBURN, J., SWANSON, S., YAAKOBI, E., SIEGEL, P., AND WOLF, J. Characterizing flash memory: Anomalies, observations, and applications. In *42nd International Symposium on Microarchitecture (MICRO)* (December 2009).

[13] GUPTA, A., KIM, Y., AND URGAONKAR, B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Washington, DC, USA, 2009), ACM, pp. 229–240.

[14] HUANG, P., CHANG, Y., KUO, T., HSIEH, J., AND LIN, M. The Behavior Analysis of Flash-Memory Storage Systems. In *IEEE Symposium on Object Oriented Real-Time Distributed Computing* (2008), IEEE Computer Society, pp. 529–534.

[15] INTEL CORP. Datasheet: Intel X18-M/X25-M SATA Solid State Drive. available from `www.intel.com`, May 2009.

[16] INTEL CORP. Datasheet: Intel X25-E SATA Solid State Drive. available from `www.intel.com`, May 2009.

[17] INTEL CORPORATION. Understanding the flash translation layer FTL specification. Application Note AP-684, Dec. 1998.

[18] JMICRON TECHNOLOGY CORPORATION. JMF602 SATA II to Flash Controller. Available from `http://www.jmicron.com/Product_JMF602.htm`, 2008.

[19] KANG, J., JO, H., KIM, J., AND LEE., J. A Superblock-based Flash Translation Layer for NAND Flash Memory. In *Proceedings of the International Conference on Embedded Software (EMSOFT)* (2006), pp. 161–170.

[20] KIM, B., AND LEE, G. Method of driving remapping in flash memory and flash memory architecture suitable therefore. United States Patent 6,381,176, 2002.

[21] KIM, J., KIM, J. M., NOH, S., MIN, S. L., AND CHO, Y. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics 48*, 2 (2002), 366–375.

[22] KIMURA, K., AND KOBAYASHI, T. Trends in high-density flash memory technologies. In *IEEE Conference on Electron Devices and Solid-State Circuits* (2003), pp. 45–50.

[23] LEE, J., CHOI, J., PARK, D., AND KIM, K. Data retention characteristics of sub-100 nm NAND flash memory cells. *IEEE Electron Device Letters 24*, 12 (2003), 748–750.

[24] LEE, J., CHOI, J., PARK, D., AND KIM, K. Degradation of tunnel oxide by FN current stress and its effects on data retention characteristics of 90 nm NAND flash memory cells. In *IEEE Int'l Reliability Physics Symposium* (2003), pp. 497–501.

[25] LEE, S., SHIN, D., KIM, Y., AND KIM, J. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. In *Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED)* (2008).

[26] MEMORY TECHNOLOGY DEVICES (MTD). Subsystem for Linux. JFFS2. Available from `http://www.linux-mtd.infradead.org/faq/jffs2.html`, January 2009.

[27] O'BRIEN, K., SALYERS, D. C., STRIEGEL, A. D., AND POELLABAUER, C. Power and performance characteristics of USB flash drives. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (2008), pp. 1–4.

[28] PARK, M., AHN, E., CHO, E., KIM, K., AND LEE, W. The effect of negative V_{TH} of NAND flash memory cells on data retention characteristics. *IEEE Electron Device Letters 30*, 2 (2009), 155–157.

[29] SANVIDO, M., CHU, F., KULKARNI, A., AND SELINGER, R. NAND flash memory and its role in storage architectures. *Proceedings of the IEEE 96*, 11 (2008), 1864–1874.

[30] SHU, F., AND OBR, N. Data Set Management Commands Proposal for ATA8-ACS2. ATA8-ACS2 proposal e07154r6, available from `www.t13.org`, 2007.

[31] SOOMAN, D. Hard disk shipments reach new record level. `www.techspot.com`, February 2006.

[32] YANG, H., KIM, H., PARK, S., KIM, J., ET AL. Reliability issues and models of sub-90nm NAND flash memory cells. In *Solid-State and Integrated Circuit Technology (ICSICT)* (2006), pp. 760–762.