Depletable Storage Systems

Vijayan Prabhakaran, Mahesh Balakrishnan, John D. Davis, Ted Wobber
Microsoft Research Silicon Valley

Abstract

Depletable storage systems use media such as NAND flash that has a limited lifetime, which decreases with more usage. Such systems treat write cycles, in addition to space, as a constrained resource of the system. Depletable storage systems must be equipped to monitor writes, attribute depletion to appropriate applications, and even control the rate of depletion. We outline the new functionalities enabled by depletion-aware mechanisms and discuss the challenges in building them.

1 Introduction

We have traditionally built storage systems that treat space as their primary resource constraint. The notion of "quota" typically applies to a fixed amount of storage that is allocated to a user. Recent cloud-based storage services charge users based on the amount of data stored and how long they are used for [2, 10]. Even in peer-to-peer storage systems, space is the primary consideration for fairness; for example, past research has focused on how to ensure fair storage consumption among the users of peer-to-peer storage systems [4].

However, with the emergence of non-volatile memory technologies such as NAND and NOR-based flash and phase change memory (PCM), treating space as the only limited resource of a storage system portrays an incomplete picture. NAND-based flash memory has certain unique limitations: first, a block must be erased before it is overwritten; second, a flash block becomes unusable after a finite number of erasures because of high raw bit error rate. Therefore, an SSD as a whole can accommodate only a fixed number of writes, before it runs out of *write cycles*, which are the raw physical writes that can be issued to the media. We call such a system, which uses storage media with limited write cycles, a *depletable storage system*.

We propose that a depletable storage system must consider write cycles, in addition to the storage space, as a constrained resource. Specifically, storage systems must be equipped with depletion-aware mechanisms to monitor the depletion caused by writes, attribute depletion to specific users or applications, and even limit the level or rate of depletion.

Building depletion-aware systems enables new functionalities in the context of depletable storage. For example, tracking the depletion caused by writes enables predictable device replacement schedules, which can reduce administrative costs and improve overall system availability. By attributing depletion to appropriate applications, cloud storage providers can charge users not only based on their space usage but also based on their write cycle consumption. Controlling the depletion caused by writes can prevent a buggy or even a malicious application from consuming more write cycles than it is allowed. We discuss these applications and others later, in detail.

Treating depletion as a first-class concern in modern storage systems is non-trivial. First, file and storage systems consist of several diverse layers, such as the buffer cache and I/O scheduler, each of which can delay, reorder, coalesce or fragment writes, causing the resulting depletion of the underlying media to be either amplified and reduced. In addition, writes not only traverse several software layers, but also hardware logic such as RAID controllers and device firmware. Ideally, any solution to keep track of depletion must transcend all the storage layers; however, such pervasive solutions are impractical given the variety of storage devices and vendors.

Second, even under identical conditions, different workloads consume varying amounts of write cycles. In this paper, we show that random workloads can consume as much as 5 times more write cycles on a commodity SSD than sequential workloads. Given a mix of applications concurrently issuing writes, it is not straightforward to compute the resulting write pattern or the write cycles consumed on the device.

Lastly, depletable storage systems are likely to use different types of memory with vastly different lifetime, performance, and cost characteristics. For example, MLC-based flash has a lower cost per GB compared to SLC-based flash, but due to the limited number of erasures it can tolerate, MLC has a higher cost per write cycle. Consequently, depletion on an MLC-based device cannot be treated as equivalent to depletion on an SLC-based device.

2 Depletable Storage

A depletable storage system uses media whose lifetime decreases with more usage. An important aspect of depletable storage is that its lifetime must be related to the write load imposed on it in measureable and predictable ways. Even though hard disk drives have a limited lifetime, it is hard to measure, predict, or even relate the lifetime to the workload.

2.1 Background

2.1.1 What Causes Depletion?

The underlying physical properties of certain storage technologies dictate that only a limited number of writes can be issued to the memory cells.

In NAND and NOR-based flash, when a write is issued, flash memory stores the charges inside the transistor gates of a block. A block can be overwritten only after flushing the pre-existing charges by a process known as an erasure. Erasing a flash block requires a large voltage to remove the electrons trapped inside the floating gates. Such high voltages can cause degradation, limiting the number of erasures and therefore, the number of writes that can be issued to the media.

Other memory technologies, such as phase change memory (PCM), can also degrade with more use, although for different reasons. PCM does not require an erasure before an overwrite and can sustain up to 100 million write cycles. Since PCM is considered a potential replacement for main memory, even such high write cycles may be consumed by the proportionally large number of `store` operations. Although PCM exhibits similar issues, in this paper we focus primarily on flash memory based systems.

2.1.2 Ideal Write-Lifetime

Given the size and other parameters of an SSD and an ideal workload that writes perfectly sequentially, it is straightforward to compute the relationship between writes sent to the device and the write cycles used up on it. Based on this relationship, we can estimate the ideal, total amount of data that can be written before the device stops accepting new writes; we call this the ideal *write-lifetime* of a device. For example, a 80 GB SSD with 5K erasure cycles can theoretically support about 400 TB (5000 times 80 GB) of data writes, which is its ideal write-lifetime. However in reality, the write-lifetime of a device can be much smaller than its ideal write-lifetime due to several factors.

A primary reason for reduced lifetime is write amplification caused by the Flash Translation Layer (FTL). Simple block-mapped FTLs use more write cycles for a given workload since they naively erase an entire block when writing to a part of it, performing a read-modify-erase-write operation. More sophisticated FTLs save write cycles by appending writes to the end of a log [5]. As a result, the number of write cycles used by a write may depend on the workload and firmware in complex ways.

FTLs also employ more complex background tasks such as cleaning and wear-leveling. Cleaning generates free flash blocks by moving valid pages together; wear-leveling ensures all the flash blocks are used uniformly by swapping blocks containing cold and hot data [1]. Such operations may consume write cycles even when there are no workloads running in the system.

2.2 Write-Lifetime in Practice

As explained above, even though the ideal write-lifetime can be calculated trivially, the real write-lifetime of an SSD is hard to estimate because of FTL idiosyncrasies. To address this issue, SSD manufacturers have proposed various endurance metrics as a way to gauge the lifetime of an SSD. For example, SanDisk's longterm data endurance (LDE) defines the total amount of data writes allowed in an SSD lifespan (assuming a specific workload pattern) [13]. Similarly, the Intel X25-M SSD [8] exports a "Media Wearout Indicator" (counter ID `0xE9`), which reports the number of write cycles used by the media [7]. This attribute is initialized to 100 when the device is new; as writes are issued and more erasures are performed, the counter drops in decrements of 1; it finally reaches 1 when the maximum rated erase cycles are used.

Although additional wear can be put on the Intel X25-M after the counter reaches 1, it is unclear if the device is guaranteed to remain reliable. For the purpose of this work, we assume that the device is worn out when the media wearout indicator reaches 1. The rate at which this attribute decreases depends upon several factors such as caching and workload access patterns.

2.2.1 Media Wearout With and Without Caching

Our first experiment was to measure the impact of workload and caching on the write-lifetime of an Intel X25-M SSD. Although the device's advertised capacity is 80 GB, the FTL reserves certain portion of the flash memory for cleaning and wear-leveling, and exports only

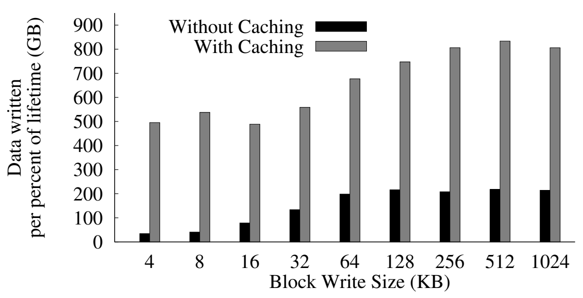

Figure 1: **SSD Write-Lifetime in Practice.**

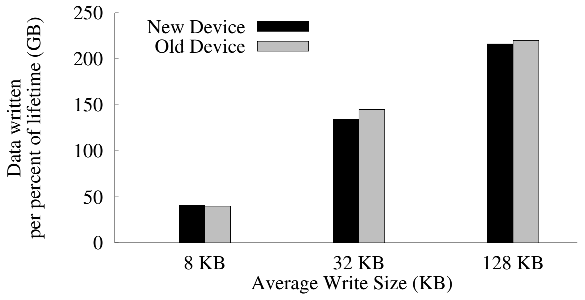

Figure 2: **Lifetime Depletion in New and Old SSD.**

74.5 GB for the file system. The device consists of MLC-based flash memory with 5K erasure cycles.

We created a 70 GB file and ran a write-intensive workload that issued random writes to the file. We varied the random write sizes from 4 KB to 1024 KB; the larger the write size, the more sequential the workload becomes. After every 1 GB of write, we read the media wearout indicator to see if it had dropped. We repeated the experiment with write-caching enabled (both at the file system level and device-level) and disabled.

Figure 1 presents the rate of decrease of the media wearout indicator for different write sizes. Y-axis presents the average total amount of data that was written for the indicator to drop by a count of 1 (*i.e.*, 1% of the device lifetime) whereas X-axis shows the write size.

We make the following observations from the figure. First, as expected, the device wears out faster without write-caching when compared to the system with caching turned on. This is because caching reduces the number of writes (for example, by coalescing overwrites) and therefore conserves more flash write cycles.

Second, short random writes consume more write cycles than large sequential writes. We believe that this is due to write amplification in the FTL. In unbuffered case, on an average, 1% of the write-lifetime is consumed by every 35 GB of random 4 KB writes. Therefore, for random 4 KB writes, the overall write-lifetime of Intel X25-M is about 3.5 TB, which is two orders of magnitude smaller than the ideal write-lifetime of 400 TB (80 GB times 5K erasure cycles). For buffered 4 KB random writes, the indicator drops every 495 GB, implying that the write-lifetime in this case is about 49.5 TB.

Finally, we see that the rate of media wear out becomes a constant after certain write sizes. For example, for unbuffered writes, between write sizes of 64 KB to 1024 KB, we see that media wearout indicator drops after a data write of 200 to 220 GB. This is likely because of the way the FTL maps the logical pages to physical pages; for example, if logical pages were mapped at the granularity of 64 KB, then a single 128 KB write and a two separate 64 KB writes are likely to consume same number of flash write cycles.

2.2.2 Media Wearout on New and Old Devices

Results from Figure 1 were obtained when the device was new. We repeated our workload (for the unbuffered case) after 60% of the device lifetime had been consumed (*i.e.*, the media wearout indicator dropped below 40) to understand if the rate of media wear is different between different stages of a device lifetime.

Figure 2 presents the rate of media wearout when the device was fresh and old. While there are some minor differences, we did not see a significant difference between the rate at which the media wears out when it was new versus old. We believe that the Intel X25-M is sophisticated enough to wear-level the device uniformly.

3 New Functionalities and Mechanisms

The above experiments show that the rate of depletion depends on the observed write pattern, which in turn depends on the workload and storage stack characteristics. Understanding further about the relationship between the workload, storage stack, and depletion can enable new functionalities and applications. We discuss some of them below.

3.1 Depletion-Aware Functionalities

• **Better Resource Utilization Metrics:** A depletable storage system that attributes depletion accurately to specific applications can be used to enable new quota systems for shared servers, as well as new pricing models for cloud storage. Current cloud storage providers charge users based on how much data they store, as opposed to how they use it. Such a model is clearly impractical for depletable systems where aggressive workloads can rapidly use up storage devices.

• **Preventing New Attacks:** Since excessive writes can deplete the write-lifetime of an SSD, they are prone to new types of "depletion-of-lifetime" attack. For example, an adversarial application can issue enough writes to drain the SSD lifetime. It is easy to show that even users with a modest amount of storage can use excessive write cycles and thereby drain the write-lifetime of a device.

• **Improving Reliability:** Traditionally, the lifetime of a storage device is considered independent of the workload that it hosts. This has resulted in some standard disk replacement practices; for example, disks are replaced either when they are completely failed or start exhibiting performance issues. In contrast, since the lifetime of a depletable storage is quantifiable, system administrators can predict when an SSD will run out of its lifetime and create a smooth device replacement pattern.

• **New Evaluation Metrics:** Depletion adds a new dimension when comparing different storage system designs, opening the door to systems that trade-off factors such as performance or durability against reduced depletion. It also leads designers to avoid components that impose excess write activity to achieve other goals (for example, defragmentation utilities).

3.2 Depletion-Aware Mechanisms

To enable the above mentioned functionalities, we need certain mechanisms in a depletable storage system.

• **Tracking:** The first step to treating depletion as a primary concern is to track the depletion caused by application-level writes. This goes beyond counting individual writes issued by applications; rather, we are concerned with the actual level of depletion caused on the media by any write, measured in write cycles.

• **Attribution:** We need new mechanisms that can attribute every write cycle used on the storage device to the corresponding write issued by an application.

• **Control:** A depletable storage system must provide mechanisms to control the rate of media wear. Commodity SSDs already provide ways to throttle writes issued to them. We argue that such techniques must be moved up the storage stack for better control in the software layer.

Applications sharing a depletable storage might have certain requirements; for example, they might want to proportionally share the available write cycles; or, they might desire "depletion isolation" (similar to performance isolation) from each other. To realize these requirements, we hope to leverage the algorithms and mechanisms proposed in the previous work [16, 3, 15].

4 Design Challenges

Implementing the depletion-aware mechanisms is non-trivial; it is made difficult by two significant challenges: storage layering and heterogeneity.

4.1 Layering

Layering poses two problems for tracking and attributing depletion. First, multiple software and hardware layers intervene between an application and the raw storage media, each of which can modify writes in different ways to increase or reduce depletion. Second, the actions of these layers are highly workload-specific; different workloads can result in vastly different levels of depletion. Below, we examine these problems in detail.

Write Amplification: As mentioned earlier, SSD firmware can cause write amplification [6] due to its internal wear-leveling logic. However, amplification occurs throughout the storage stack; for example, a single write to a file may be amplified by a journaling file system into several writes because of the writes to the journal and to the fixed location. Similarly, a software RAID volume manager might replicate a write for reliability reasons.

Amplification is usually workload-specific. For example, SSD firmware amplifies small random writes but not large sequential writes. Enough information may not be available at a higher layer (*e.g.*, the file system) to infer the write amplification that occurs lower in the stack.

Write Reduction: Application-level writes may even be reduced before they reach the media. For example, the buffer cache manager and IO scheduler can reduce writes by coalescing and merging multiple writes into a single write. Such performance optimizations can interfere with write tracking and attribution. Consider a process, which issues an asynchronous write to the buffer cache and exits; when the write is received by a lower layer, there may not be an owner for write attribution. Similarly, if writes from different applications are merged by an IO scheduler, write attribution must handle the merged write appropriately.

One approach for tracking the amplification and reduction of writes across layers involves virtual machines, where any modification that happens to the writes in the guest OS can be caught by monitoring writes going out to the virtual disk, which is just a file on the host OS. Capturing write modifications that happen in hardware (for example, in RAID controllers or SSD firmware) are more difficult. A white-box approach would consist of modifying the hardware to return explicit write cycle counts in response to writes; a black-box approach might involve modeling the behavior of the hardware under different workloads in order to predict the number of write cycles used up by any write.

Controlling depletion presents more challenges. Simply denying the writes of an application, which has exceeded its quota might crash the application or even the file system [12]. Writes of an application which has exceeded its quota may be delayed, but at the expense of using more memory for caches. Moreover, writes may not be always delayed because an application might request a synchronous write to the storage. New external synchrony mechanisms might be helpful in minimizing the synchronous writes [11]. We are examining the use of hybrid disk storage designs, where writes issued by applications which have exceeded their quota can be forwarded to disk drives [14].

4.2 Heterogeneity

An aspect of heterogeneous storage mentioned previously is that all write cycles are not equal; MLC-based flash costs less per GB compared to SLC-based flash, but is in fact more expensive per write cycle. Designers of depletable storage systems may need to make decisions based on not just the quantity of write cycles being used but also their cost. One possibility is a common currency that allows write cycles from different technologies to be compared.

We define a *write credit* as a common currency to measure the write-lifetime consumption of an application. A write credit is the cost of raw write cycles on an SSD; specifically, we define it as the ratio of write-lifetime to the cost of the SSD. As of March 2010, a 80 GB Intel X25-M with 5K erasure cycles costs about $225. The device has 400 TB of ideal write-lifetime and therefore costs $1 for every 1.78 TB of raw write cycles consumed.

Write credit presents a new axis by which two SSDs can be compared in terms of their cost per write. A 64 GB of SLC-based Intel X25-E [9] costs $745. With 100K erasure cycles, the device has 6400 TB of ideal write-lifetime. Assuming a similar firmware, the SLC device costs $1 for every 8.59 TB of write cycles. Based on the write credits (1.78 TB vs. 8.59 TB), the Intel X25-E is more cost effective than the X25-M.

Heterogeneous systems can include more complex trade-offs. For example, SSDs from different manufacturers may respond differently to the same workload, with one SSD using up more write cycles than the other. Once again, explicit feedback from the SSD to report the number of write cycles used – or black-box modeling to determine it – could be a potential solution.

5 Conclusion

SSDs offer several benefits over traditional HDDs; they have better read-write performance, power characteristics, and reliability. However, this comes with the price of limited lifetime. In this paper, we argue for depletion-aware systems that treat write cycles as a first-class concern. Depletable storage systems open the door for new functionalities such as better device replacement schedule and new pricing model. However, several challenges must be met before realizing these functionalities and we hope to address them in the future work.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of USENIX Annual Technical Conference*, pages 57–70, 2008.

[2] Amazon. Amazon Simple Storage Service (Amazon S3). `http://aws.amazon.com/s3/`.

[3] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 45–58, Feb. 1999.

[4] L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 120–132, Oct. 2003.

[5] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 229–240, Mar. 2009.

[6] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009.

[7] Intel Corporation. Intel Solid-State Drive Toolbox. `http://download.intel.com/support/ssdc/hpssd/sb/intel_ssd_toolbox_user_guide.pdf`.

[8] Intel Corporation. Intel X18-M/X25-M SATA Solid State Drive. `http://download.intel.com/design/flash/nand/mainstream/mainstream-sata-ssd-datasheet.pdf`.

[9] Intel Corporation. Intel X25-E SATA Solid-State Drive. `http://download.intel.com/design/flash/nand/extreme/319984.pdf`.

[10] Microsoft. Windows Azure Platform. `http://www.microsoft.com/windowsazure/`.

[11] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn. Rethink the sync. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 1–14, Nov. 2006.

[12] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON file systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 206–220, Oct. 2005.

[13] SanDisk. Longterm Data Endurance (LDE) for Client SSD. `http://www.sandisk.com/media/65675/LDE_White_Paper.pdf`.

[14] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending SSD Lifetimes with Disk-Based Write Caches. In *Proceedings of the File and Storage Technologies Conference*, pages 101–114, Feb. 2010.

[15] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: Performance Insulation for Shared Storage Servers. In *Proceedings of the File and Storage Technologies Conference*, Feb. 2007.

[16] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI '94)*, pages 1–11, Monterey, California, Nov. 1994.