Differential RAID: Rethinking RAID for SSD Reliability

Mahesh Balakrishnan[1], Asim Kadav[2], Vijayan Prabhakaran[1], Dahlia Malkhi[1]

[1]Microsoft Research Silicon Valley, Mountain View, CA, USA
[2]University of Wisconsin, Madison, WI, USA

{maheshba@microsoft.com, kadav@cs.wisc.edu, vijayanp@microsoft.com, dalia@microsoft.com}

Abstract

SSDs exhibit very different failure characteristics compared to hard drives. In particular, the Bit Error Rate (BER) of an SSD climbs as it receives more writes. As a result, RAID arrays composed from SSDs are subject to correlated failures. By balancing writes evenly across the array, RAID schemes can wear out devices at similar times. When a device in the array fails towards the end of its lifetime, the high BER of the remaining devices can result in data loss. We propose Diff-RAID, a parity-based redundancy solution that creates an age differential in an array of SSDs. Diff-RAID distributes parity blocks unevenly across the array, leveraging their higher update rate to age devices at different rates. To maintain this age differential when old devices are replaced by new ones, Diff-RAID reshuffles the parity distribution on each drive replacement. We evaluate Diff-RAID's reliability by using real BER data from 12 flash chips on a simulator and show that it is more reliable than RAID-5, in some cases by multiple orders of magnitude. We also evaluate Diff-RAID's performance using a software implementation on a 5-device array of 80 GB Intel X25-M SSDs and show that it offers a trade-off between throughput and reliability.

Categories and Subject Descriptors D.4.2 [*Operating Systems*]: Storage Management; D.4.4 [*Operating Systems*]: Reliability; D.4.8 [*Operating Systems*]: Performance

General Terms Design, Performance, Reliability

Keywords RAID, SSD, Flash

1. Introduction

Solid State Devices (SSDs) have emerged in the last few years as viable replacements for hard drives in many settings. Commodity SSDs can offer thousands of random reads and writes per second, potentially eliminating I/O bottlenecks in high-performance data centers while driving down power consumption. Though early SSDs were prohibitively expensive, the emergence of Multi-Level Cell (MLC) technology has significantly driven down SSD cost in the recent past.

However, MLC devices are severely hamstrung by low endurance limits. Individual flash pages within an SSD require expensive erase operations between successive writes. Each erasure makes the device less reliable, increasing the Bit Error Rate (BER) observed by accesses. Consequently, SSD manufacturers specify not only a maximum BER (usually around 10^{-14}, as with hard disks), but also a limit on the number of erasures within which this guarantee holds. For MLC devices, this *erasure limit* is typically rated at 5,000 to 10,000 cycles per block. As flash bit density continues to increase, the erasure limit is expected to decrease as well.

Device-level redundancy is currently the first line of defense against storage failures. Existing redundancy options – such as any of the RAID levels – can be used without modification to guard against SSD failures, and to mask the high BER of aging SSDs. Unfortunately, existing RAID solutions do not provide adequate protection for data stored on SSDs. By balancing writes across devices, they cause multiple SSDs to wear out at approximately the same rate. Intuitively, such solutions end up trying to protect data on old SSDs by storing it redundantly on other, equally old SSDs. Later in this paper, we quantify the ineffectiveness of such an approach.

We propose Differential RAID (Diff-RAID), a new parity-based technique similar to RAID-5 designed explicitly for reliable SSD storage. Diff-RAID attempts to create an age differential across devices, limiting the number of high-BER SSDs in the array at any point in time. In other words, Diff-RAID balances the high BER of older devices in the array against the low BER of younger devices.

To create and maintain this age differential, Diff-RAID modifies two existing mechanisms in RAID-5. First, Diff-RAID distributes parity blocks *unevenly* across devices; since parity blocks are updated more often than data blocks due to random access patterns, devices holding more parity receive more writes and consequently age faster. Diff-

EuroSys'10, April 13–16, 2010, Paris, France.

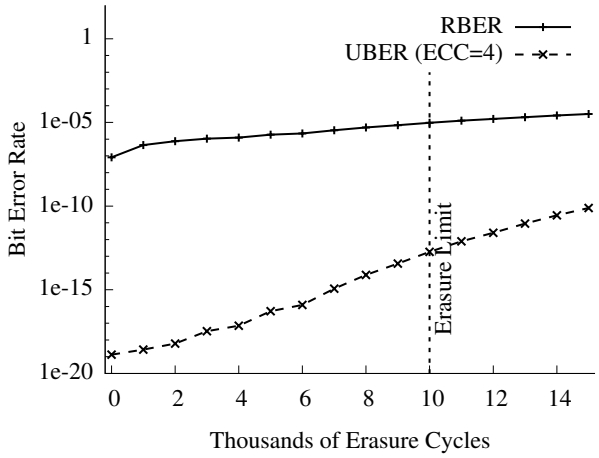

Figure 1. RBER (Raw Bit Error Rate) and UBER (Uncorrectable Bit Error Rate, with 4 bits of ECC) for MLC flash rated at 10,000 cycles; data taken from [6].

RAID supports arbitrary parity assignments, providing a fine-grained trade-off curve between throughput and reliability. Second, Diff-RAID reshuffles parity on drive replacements to ensure that the oldest device in the array always holds the maximum parity and ages at the highest rate. This ensures that the age differential created through uneven parity assignment persists when new devices replace expired ones in the array.

Diff-RAID's ability to mask high BERs on aging SSDs confers multiple advantages. First, it offers higher reliability than RAID-5 or RAID-4 while retaining the low space overhead of these options. Second, it opens the door to using commodity SSDs past their erasure limit, protecting the data on expired SSDs by storing it redundantly on younger devices. Third, it potentially reduces the need for expensive hardware Error Correction Codes (ECC) in the devices; as MLC density continues to increase, the cost of such ECC is expected to rise prohibitively. These benefits are achieved at the cost of some degradation in throughput and complexity in device replacement.

We evaluate Diff-RAID performance using a software implementation running on a 5-device array of Intel X25-M SSDs, on a combination of synthetic and real server traces. We also evaluate Diff-RAID reliability by plugging in real-world flash error rates into a simulator. We show that Diff-RAID provides up to four orders of magnitude more reliability than conventional RAID-5 for specific failure modes.

The remainder of this paper is organized as follows: Section 2 describes the problem of correlated SSD failures in detail. Section 3 describes Diff-RAID. Section 4 evaluates Diff-RAID reliability and performance. Section 5 summarizes related work, Section 6 describes future goals for Diff-RAID, and Section 7 concludes.

2. Problem Description

2.1 Flash Primer

The smallest unit of NAND-based flash storage that can be read or programmed (written) is a *page* (typically 4 KB in size). All bits in a blank page are set to 1s, and writing data to the page involves setting some of the bits within it to 0s. Individual bits within a page cannot be reset to 1s; rewriting the page to a different bit pattern requires an intermediate *erase* operation that resets all bits back to 1. These erasures are performed over large blocks (e.g., of 128 KB) spanning multiple pages. Blocks *wear out* as they are erased, exhibiting increasing BERs that become unmanageably high once the erasure limit is breached.

As a result of these fundamental constraints on write operations, early flash-based devices that performed in-place page modification suffered from very poor random write latencies; writing to a randomly selected 4 KB page required the entire 128 KB erase block to be erased and rewritten. In addition, imbalanced loads that updated some pages much more frequently than others could result in uneven wear across the device.

To circumvent these problems, modern SSDs implement a log-based block store, exposing a logical address space that is decoupled from the physical address space on the raw flash. The SSD maintains a mapping between logical and physical locations at the granularity of an erase block. A write to a random 4 KB page involves reading the surrounding erasure block and writing it to an empty, previously erased block, with no expensive erase operations in the critical path. In addition, the mapping of logical to physical blocks is driven by *wear-leveling* algorithms that aim for even wear-out across the device. SSDs typically include more raw flash than advertised in order to continue logging updates even when the entire logical address space has been occupied; for example, an 80 GB SSD could include an extra 10 GB of flash.

SSDs come in two flavors, depending on the type of flash used: Single-Level Cell (SLC) and Multi-Level Cell (MLC). A cell is the basic physical unit of flash, storing voltage levels that represent bit values. SLC flash stores a single bit in each cell, and MLC stores multiple bits. SLC provides ten times the erasure limit as MLC (100,000 cycles versus 10,000), but is currently 3-4 times as expensive. Current industry trends point towards MLC technology with more bits per cell.

Flash Error Modes: MTTF (Mean Time To Failure) values are much higher for SSDs than hard drives due to the absence of moving parts. As a result, the dominant failure modes for SSDs are related to bit errors in the underlying flash. Bit errors can arise due to writes (*program disturbs*), reads (*read disturbs*) and bit-rot over time (*retention errors*) [6, 10]. All these failure modes occur with greater frequency

as the device ages[1]. In practice, SSDs use hardware ECC to bring down error rates; the pre-ECC value is called the Raw Bit Error Rate (RBER), and the post-ECC value is called the Uncorrectable Bit Error Rate (UBER).

Figure 1 shows the RBER and UBER of flash rise with the number of per-block erasures. We assume 4-bit ECC per 512 byte sector (the current industry standard for MLC flash [10]); this means that enough ECC is used to correct 4 or less bad bits in each sector. The RBER data in the figure corresponds to the D-MLC32-1 flash chip in [6], rated at 10,000 cycles; we compute the UBER using the methodology described in [10].

2.2 The Problem with RAID for SSDs

Device-level redundancy has been used successfully for decades with hard drives. The standard block-based interfaces exposed by SSDs allow existing redundancy and striping solutions – such as the RAID levels – to work without modification. We examine the reliability of a specific RAID option (RAID-5), though our observations apply in different degrees to any of the RAID levels. We will first describe the operation of RAID-5 and then outline the concerns with its usage.

In an N-device RAID-5 array, data blocks are striped across N-1 devices and a parity block is computed for each stripe and stored on the Nth device. The role of the parity device rotates for each stripe; as a result, each of the N devices stores an equal fraction of the parity blocks in the array. Whenever one of N-1 data blocks in the stripe is written to, the corresponding parity block in the stripe must also be updated; consequently, parity blocks receive more write traffic than data blocks for random access workloads. By rotating the role of the parity device, RAID-5 aims to eliminate any parity bottlenecks in the array, spreading write load evenly across the array.

The key insight in this paper is that this **load-balancing of writes can cause correlated failures in SSD arrays.** Since all devices in the RAID-5 array receive similar write workloads, they use up their erasure cycles at similar rates. As a result, the array can be in a state where multiple devices have reached the end of their erasure limit and exhibit high UBERs, making correlated failures highly probable.

In particular, we are concerned with the case where the array experiences a drive failure, and subsequently cannot reconstruct the missing drive due to bit errors in the remaining drives. Since the array has no redundancy once a drive fails, any existing bit errors in the array will be uncorrectable. The corruption of single bits in critical data structures – such as filesystem super-blocks – can result in massive data loss. This failure mode has been of increasing concern with multi-terabyte hard drive arrays, where the sheer quantity of data makes a bit error likely [12]. To the best of our knowledge, we are the first to point out its significance for SSD arrays of any size (in an earlier version of this paper [8]).

[1] Through this paper, we define 'age' in erasure cycles and not in time

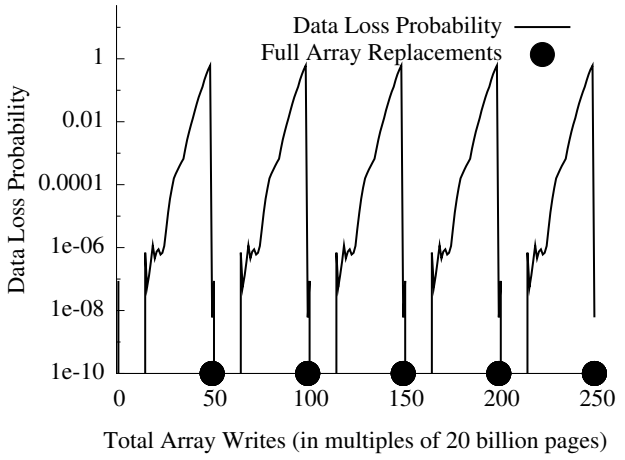

Figure 2. RAID-5 Reliability: the array becomes more unreliable as it ages, until all devices hit their erasure limit at the same time and are replaced with brand new devices.

Figure 2 shows the probability of data loss for a RAID-5 array (5 devices, 80 GB each) change as data is written to it; see Section 4 for more details on the simulation setup. At each point in the graph, we plot the probability that data loss will occur if one of the devices fails. Each device is replaced by a new device when it reaches its erasure limit of 10,000 cycles. In the beginning, all devices are brand new and have low UBERs, and hence the probability of data loss is very low. As the array receives more writes, the devices age in lock-step; the peaks on the curve correspond to points where all devices reach their erasure limits at the same time and are replaced, transitioning the array to a set of brand new devices.

Importantly, this phenomenon is not specific to RAID-5. For example, it occurs in a fixed-parity configuration (RAID-4) as well, since all data devices age at the same rate. It occurs to a lesser extent in mirrored configurations (RAID-1, RAID-10), since both mirrors age at the same rate. Having two parity devices (RAID-6) prevents data loss on one device failure; however, data loss is likely if two devices are lost. Essentially, any RAID solution used with SSDs offers less reliability than system administrators would like to believe.

3. Differential RAID

Diff-RAID is a parity-based RAID solution very similar to RAID-5. We believe a parity-based scheme to be a great fit for SSDs; their extremely high random read speed eliminates the traditional synchronous read bottleneck for random writes. On a 5-device array of Intel X25-Ms configured as RAID-5, we were able to achieve *14,000 random writes per second* to the array; a similar experiment on hard drives

would have been limited to a few hundred writes per second due to random seeks in the critical path.

Accordingly, our focus is on improving the reliability of a parity-based scheme. We start with the basic RAID-5 design and improve it in two key ways. First, *uneven parity distribution* allows Diff-RAID to age SSDs at different rates, creating an age differential in the array. Second, *parity-shifting drive replacement* allows Diff-RAID to maintain that age differential when old devices are retired from the array and replaced by new SSDs.

3.1 Uneven Parity Distribution

Diff-RAID controls the aging rates of different devices in the array by distributing parity blocks unevenly across them. Random writes cause parity blocks to attract much more write traffic than data blocks; on every random write to a data block, the corresponding parity block has to be updated as well. As a result, an SSD with more parity receives a higher number of writes and ages correspondingly faster.

In the following discussion, we measure the age of a device by the average number of cycles used by each block in it, assuming a perfect wear-leveling algorithm that equalizes this number across blocks. For example, if each block within a device has used up roughly 7,500 cycles, we say that the device has used up 7,500 cycles of its lifetime.

We represent parity assignments with n-tuples of percentages; for example, $(40, 15, 15, 15, 15)$ represents a 5-device array where the first device holds 40% of the parity and the other devices store 15% each. An extreme example of uneven parity assignment is RAID-4, represented by $(100, 0, 0, 0, 0)$ for 5 devices, where the first device holds all the parity. At the other extreme is RAID-5, represented by $(20, 20, 20, 20, 20)$ for 5 devices, where parity is distributed evenly across all devices.

For a workload consisting only of random writes, it is easy to compute the relative aging rates of devices for any given parity assignment. For an n device array, if a_{ij} represents the ratio of the aging rate of the ith device to that of the jth device, and p_i and p_j are the percentages of parity allotted to the respective devices, then:

$$a_{ij} = \frac{p_i * (n-1) + (100 - p_i)}{p_j * (n-1) + (100 - p_j)} \quad (1)$$

In the example of 5-device RAID-4, the ratio of the aging rate of the parity device to that of any other device would be $\frac{100*4+0}{0*4+100} = 4$; in other words, the parity device ages four times as fast as any other device. Since RAID-4 is an extreme example of uneven parity assignment, it represents an upper bound on the disparity of aging rates in an array; all other parity assignments will result in less disparity, with RAID-5 at the other extreme providing no disparity.

3.2 Parity-Shifting Drive Replacement

In the long run, uneven parity distribution is not sufficient to maintain an age differential. To understand this point, consider a RAID-4 array with 5 devices. With a workload of random writes, the data devices in the array will be at approximately 2,500 erase cycles each when the parity device hits 10,000 cycles and is replaced. If we allow the new replacement device to be a verbatim copy of the old device – as conventional RAID would do – it will hold all parity and continue to age four times as fast as the data devices. After the third such replacement of the parity device, the data devices will all be at 7,500 cycles, resulting in an aging and unreliable array. The fourth replacement of the parity device will occur when all the devices are at 10,000 cycles.

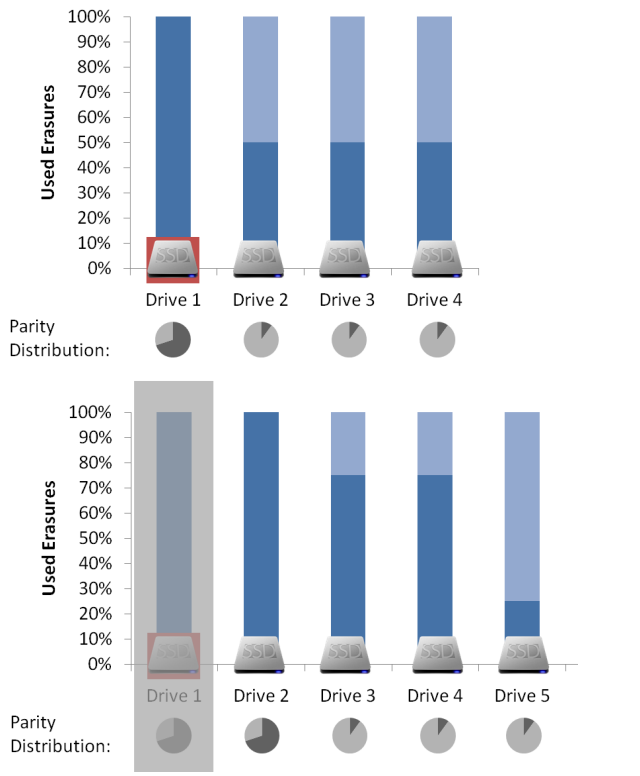

Figure 3. Parity-Shifting Drive Replacement for a 4-device array with parity assignment $(70, 10, 10, 10)$: age distribution of devices when Drive 1 is replaced (Top), and when Drive 2 is replaced (Bottom). Drive 2 receives 70% of the parity after Drive 1 is replaced.

Intuitively, after each replacement of the parity drive, the new replacement drive ages at a much faster rate than the older data devices and hence catches up with them. As a result, the age differential created by the uneven parity distribution is not maintained across drive replacements. In order to ensure that the age differential is maintained, we would like the aging rate of a device to be proportional to its age: the older a device, the faster it ages. If this were the case, any age differential would persist across drive replacements.

Another reason for aging older devices faster is that the presence of an old SSD makes the array less reliable, and hence we want to use up its erase cycles rapidly and remove

it as soon as possible. Conversely, young SSDs make the array more reliable, and hence we want to keep them young as long as possible. If two SSDs are at the same age, we want to age one faster than the other in order to create an imbalance in ages.

Consequently, on every device replacement, we shift the logical positions of the devices in the array to order them by age, before applying the parity assignment to them. In the simple example of RAID-4, each time a parity device is replaced, the oldest of the remaining data devices becomes the new parity device (breaking ties arbitrarily), whereas the new replacement device becomes a data device.

Figure 3 illustrates this scheme for a 4-device array with a parity assignment of $(70, 10, 10, 10)$; by Equation 1, this assignment results in the first device aging twice as fast as the other three devices. As a result, when Drive 1 reaches 10,000 erase cycles, the other three SSDs are at 5,000 cycles. When Drive 0 is replaced, Drive 1 is assigned 70% of the parity blocks, and the other SSDs (including the new one) are assigned 10% each.

Diff-RAID provides a trade-off between reliability and throughput: the more skewed the parity distribution towards a single device, the higher the age differential, and consequently the higher the reliability. A skewed parity assignment results in low throughput and high reliability, and a balanced parity assignment results in high throughput and low reliability. The Diff-RAID parity assignment corresponding to RAID-4 provides the least throughput – with the single parity device being a bottleneck – but the highest reliability, and the one corresponding to RAID-5 provides the highest throughput but the least reliability. Since Diff-RAID supports any arbitrary parity assignment, administrators can choose a point on the trade-off curve that fits application requirements.

3.3 Analysis of Age Distribution Convergence

Interestingly, if we continue shifting parity over multiple device replacements for a given parity assignment, the device ages observed at replacement time eventually converge to a stationary distribution. Figure 4 plots the distribution of device ages at replacement time for a 5-device array with a parity assignment of $(80, 5, 5, 5, 5)$. After a number of replacements, the ages of the devices at replacement time converge so that the oldest remaining device is always at 5,750 erase cycles, with the other devices at 4,312.5, 2,875 and 1,437.5 cycles respectively. Figure 5 shows the stationary distributions for a number of different parity assignments; on the x-axis is the percentage of parity on a single device, with the remaining spread evenly across the other devices. Importantly, *the number of replacements over time does not depend on the parity assignment*: Diff-RAID uses exactly as many replacements as conventional RAID-5.

We can show that any parity assignment eventually converges to a stationary distribution. Let us denote the different rates at which disks age via a vector $(t_1, t_2, ..., t_k)$. Recall that Equation 1 allows us to compute this vector given any parity assignment, assuming a perfectly random workload. This vector is normalized and sorted, so that $t_1 = 1$, and for all j, $t_j >= t_{j+1}$. This means that during a period of time that device 1 ages 100 units, device 2 ages $t_2 * 100$ units, device 3 $t_3 * 100$ units, etc. In practice, the aging vector is a function of both the workload and the parity assignment. In our analysis, we assume a random workload, with the aging-vector reflecting only the parity assignment; however, the analysis should work with any aging vector derived from a real workload and parity assignment.

Suppose that the expected life of a device is A aging-units. Now, think of a new device which is brought into the system when device 1 burns out. Let us number this new device $k + 1$. Device 2 replaces the role of device 1 when device $k + 1$ enters; at this point, it has aged $A * t_2$, and it continues aging at rate t_1; device 3 replaces device 2 at age $A * t_3$ and continues aging at rate t_2, etc. Denote by A_2 the number of aging units that device 2 suffers during the period until it, too, burns out. During this period, device $k + 1$ ages $A_2 * t_k$. Then device 3 becomes the first device, and burns out after A_3 aging units, during which time device $k + 1$ ages $A_3 * t_{k-1}$. And so on. The last aging period until device $k + 1$ needs to replace the first device in the array is A_k, and its aging increment is $A_k * t_2$.

In summary, when device $k + 1$ replaces the first device in the array it has age $A_2 * t_k + A_3 * t_{k-1} + ... + A_k * t_2$. We obtain the burnout time A_{k+1} remaining for device $k + 1$ from this value onward by putting $A_{k+1} = A - (A_2 * t_k + A_3 * t_{k-1} + ... + A_k * t_2)$. More generally, for any d, A_{k+d} is the burnout period remaining after it reaches age $A_{1+d} * t_k + A_{2+d} * t_{k-1} + ... + A_{k-1+d} * t_2$.

We want to design the aging vector so that A_{k+d} is a certain fixed, positive value, so that when a new device replaces the first one, it has a reasonable remaining life expectancy. In practice, we probably need to worry only about a few device replacements, since it is hard to imagine the same RAID of SSDs undergoing more than a handful of device replacements before the entire technology is upgraded.

Nevertheless, we can design a replacement strategy that keeps A_{k+d} fixed for any number of replacements. Denote by B the desired remainder life-expectancy for a replacement device. During the period that device 1 ages A units, device 2 should age at rate $r_{1,2} = (A - B)/A$. Device 3 should reach age $A - B$ after aging at rate $r_{1,3}$ for period A, and at rate $r_{2,3}$ for period B. That is, $A - B = A * r_{1,3} + B * r_{2,3}$. And so on.

After performing k device replacements, there is a steady-state allocation that fixes B forever, simply by fixing $t_2 + t_3 + ... + t_k = (A - B)/B$. We obtain $A_{k+d} = A - (A_{1+d} * t_k + A_{2+d} * t_{k-1} + ... + A_{k-1+d} * t_2) = A - B * (A - B)/B = B$, and the invariant continues to hold. Conversely, if the aging vector is fixed, then the remaining life of the second array-device upon replacement converges to B satisfying the

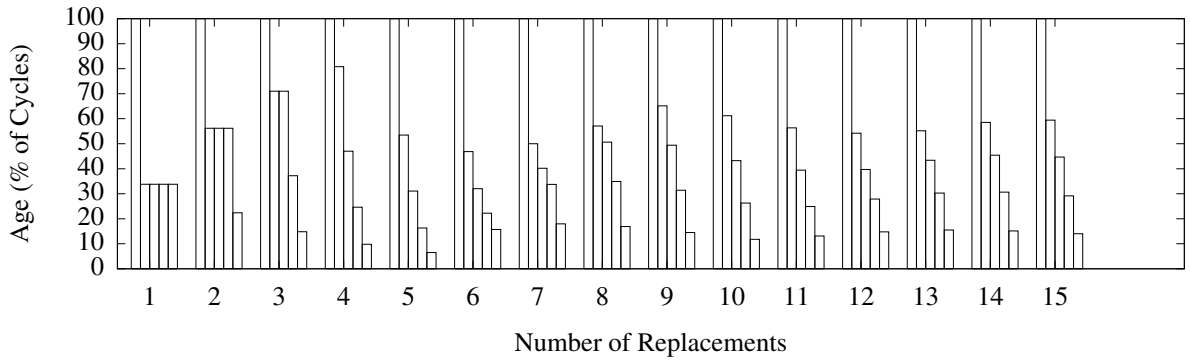

Figure 4. Distribution of device ages at replacement time for $(80, 5, 5, 5, 5)$ parity assignment: on each replacement, a new device is added to the end of the array and the second device moves into first position.

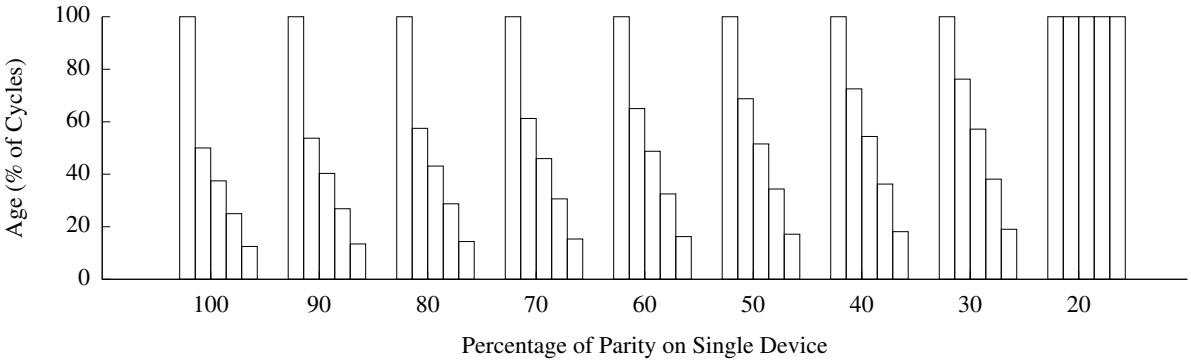

Figure 5. Convergent distribution of ages at replacement time for different parity assignments; on the x-axis is the percentage of parity on a single device, with the rest spread evenly across the other devices (e.g., $x = 60$ represents $(60, 10, 10, 10, 10)$).

equation above. Briefly, to see that this holds, observe that if the remaining lives of replacement devices exceed B by ϵ, then the remaining life upon replacement of device $k + 1$ is $A - (B + \epsilon)((A - B)/B) = B - \epsilon * (A - B)/B$. And conversely, if it was $B - \epsilon$, it becomes $B + \epsilon * (A - B)/B$. Hence, the life of a replacement device gradually converges to B.

For example, we could have three devices, with steady-state aging rates $(1, \frac{1}{2}, \frac{1}{2})$. In the first replacement period, we need to age device 3 at rate $\frac{1}{4}$. After device 2 becomes the first in the array, we continue aging at rates $(1, \frac{1}{2}, \frac{1}{2})$ forever. Each device replacement occurs when the first device reaches its full life period; the second device $\frac{1}{2}$, and the third $\frac{1}{4}$. The second device moves to be the first in the array, having suffered $\frac{1}{2}$ of its burnout load; the third device moves to be second, with $\frac{1}{4}$ burnout, and a fresh new device is deployed in the third array position. Then, after another $\frac{1}{2}$ life, the first device in the array burns out; the second device has age $\frac{1}{4} + \frac{1}{2} * \frac{1}{2}$ and moves to replace the first one; the third device has age $\frac{1}{2} * \frac{1}{2} = \frac{1}{4}$ and replaces the second. In numbers, the series of successive replacements is $(1, \frac{1}{2}, \frac{1}{4}), (\frac{1}{2} + \frac{1}{2}, \frac{1}{4} + \frac{1}{2} * \frac{1}{2}, \frac{1}{2} * \frac{1}{2})$, and so on.

3.4 Deployment Issues

Bootstrapping Convergence: The convergence property of Diff-RAID can be extremely useful in designing an array with stable reliability and performance properties. Given a predictable workload, it is possible to compute the stationary distribution for any parity assignment; if the array starts off its constituent devices already fitting the age distribution, it will reach that distribution every time the oldest device in the array hits the erasure limit. As a result, the worst case unreliability of the array is very easy to predict.

One possibility is that system administrators will bootstrap new Diff-RAID arrays by burning through flash cycles on the first set of devices used (or conversely, by throwing away the devices before they reach their erasure limit). At first glance, this seems like an expensive proposition, given the high cost of flash. However, the bootstrapping process represents a one-time cost involving a fixed amount of flash: in the example of a $(80, 5, 5, 5, 5)$ array, it involves wasting a little over 14K erasure cycles, or around a device and a half worth of flash.

For example, consider a 5-device Diff-RAID array with a 5-year lifetime, where a single SSD has to be replaced every year. Assuming that a single SSD costs 300$, the total cost of storage would be 3000$ over the five years (not taking

dropping SSD costs into account). The cost of the wasted flash in this case would be around 450$, or around 15% extra. Since this represents a one-time dollar cost, we believe it to be a practical solution.

Of course, it is entirely possible to run Diff-RAID without first putting it in a stationary distribution. As we will show in our evaluation, the resulting array will not be as predictably reliable, but it will still be orders of magnitude more reliable than conventional RAID-5 on average.

Implementation of Drive Replacement: We expect administrators to replace worn out devices by simply removing them and replacing them with new devices, and triggering RAID reconstruction. In addition to reconstruction, parity has to be redistributed across the devices. A naive way to implement the redistribution is to retain the same parity mapping function, but simply shift the logical order of the devices to the left; if the replaced device was at position 0, the new device now occupies position 3 (in a 4-device array), and all the other devices shift one position to the left. While this simplifies the logic in the RAID controller immensely, it requires all parity blocks in the array to be moved.

A faster option is to retain the logical order of devices, but change the parity mapping function to a new function that satisfies the parity assignment percentages with minimal movement of parity blocks. For example, if the parity assignment is $(70, 10, 10, 10)$ in a four device array, and device 0 is replaced by device 4, we need to move 60% of the parity blocks from device 4 to device 1; it needs 70% in all and already has 10%. Such a movement of parity blocks will leave 10% on device 4, and no changes occur on device 2 and device 3, which already have their allotted 10%. As a result, only 60% of the parity blocks need to be moved. This is Diff-RAID's default drive replacement implementation.

An alternative option is to delay parity redistribution and do it incrementally during the execution of actual writes to the array. Whenever a random write occurs, we simply write the new parity to its new location, moving the data block at that location to the old parity location. However, this option slows down writes to the array for an unspecified period of time. Importantly, none of the redistribution options use up too many flash cycles by redistributing parity blocks; rewriting all the parity blocks uses up a total of a single erasure cycle, summed across all the devices in the array.

Workload Sequentiality: Conventional RAID-5 controllers designed for hard drives do their utmost to extract full-stripe writes from the workload: writes that span the entire stripe and can be executed without first reading the old data and parity blocks. They do so using optimizations such as write caching within the controller to coalesce multiple random writes into single full-stripe writes. However, existing RAID-5 implementations have had limited success in this regard; existing enterprise applications such as database servers are notoriously non-sequential, leading to the growing unpopularity of RAID-5.

For SSD arrays in general, reads and writes are extremely fast and full-stripe writes are not required for performance; our evaluation will bear out this observation. For Diff-RAID in particular, full-stripe writes are in fact detrimental to reliability; each device gets written to in a full-stripe write, reducing the disparity in write workloads observed by the devices. Full-stripe writes do have the advantage of conserving flash cycles by performing fewer parity writes; however, the level of write caching required to extract them creates a window of unreliability while the data is cached on the controller, and also adds complexity to the controller. As a result, we don't implement any optimizations such as write caching to leverage full-stripe writes.

4. Evaluation

In this section, we evaluate the *reliability* and *performance* of Diff-RAID and compare them to traditional RAID-5 arrays. To measure reliability, we implemented various RAID configurations on a simulator and evaluated them using bit error rates from real devices. To measure performance, we implemented a software Diff-RAID that runs on an array of 5 Intel X25-M SSDs [7]; performance measurements were gathered by running synthetic benchmarks as well as real workloads collected from enterprise servers.

Diff-RAID Array Configurations: In the following evaluations, we use a 5-SSD array, where each SSD is 80 GB in size. All the SSDs in the array are homogeneous, using the same controller logic and flash chips. The SSDs have MLC chips rated at an erasure limit of 10,000 cycles. For reliability simulations, we assume that SSDs in the array are replaced when they hit this limit.

Given an array of SSDs, various Diff-RAID configurations are possible depending upon how much parity is assigned to each device. In order to narrow down the solution space, we focus on a subset of possible assignments that concentrate a particular percentage of parity on the first device and evenly distribute the rest of the parity over the other devices in the array. Specifically, we evaluate the following 9 different configurations: $(100, 0, 0, 0, 0)$ (*i.e.*, RAID-4), $(90, 2.5, 2.5, 2.5, 2.5)$, ... and $(20, 20, 20, 20, 20)$ (*i.e.*, RAID-5).

Bit Error Rates: To measure the reliability of a specific Diff-RAID configuration, we must first understand the bit error rates for the MLC flash chips. We use raw bit error rate curves for 12 different flash chips obtained from two published studies. The first study by Grupp, *et al.* [6] provides us with 10 different RBER curves. We obtain two more curves from the second study, by Mielke, *et al.* [10] (specifically, we use the Diamond and Triangle flash chips, in the nomenclature of that paper).

We assume that the data in the flash are protected with ECC. We model 4-bit ECC protection per 512-byte sector, which is the current industry standard for MLC flash. We use the UBER equation from the second study (page 1 of Mielke,

Flash Type	Raw Bit Error Rate		Uncorrectable Bit Error Rate	
	5 K	10 K	5 K	10 K
B-MLC8-1	2.99e-07	3.60e-07	2.26e-20	6.80e-20
B-MLC8-2	1.78e-07	3.58e-07	0.00e+00	0.00e+00
B-MLC32-1	1.92e-06	4.84e-06	6.31e-17	6.40e-15
B-MLC32-2	8.17e-07	2.62e-06	7.47e-19	2.99e-16
C-MLC64-1	1.12e-06	3.67e-06	4.36e-18	1.61e-15
C-MLC64-2	1.85e-06	9.51e-06	5.26e-17	1.85e-13
D-MLC32-1	6.70e-07	1.18e-05	4.40e-19	5.31e-13
D-MLC32-2	7.14e-07	2.16e-06	6.03e-19	1.14e-16
E-MLC8-1	3.07e-08	1.79e-07	2.82e-21	4.44e-20
E-MLC8-2	1.93e-07	6.70e-07	0.00e+00	4.40e-19
IM-1	5.50e-07	1.16e-06	1.17e-19	5.32e-18
IM-2	5.22e-08	1.09e-07	2.20e-20	1.28e-19

Table 1. Bit error rates for various flash types when used for 5K and 10K erasure cycles.

et al. [10]) on the RBER values to compute the UBER for the 12 flash chips. Table 1 presents the RBER values from these two previous studies and the UBER values that we calculated for various flash chips, when the chips have used up 5,000 and 10,000 erasure cycles. All but one of the flash chips are rated at 10,000 cycles; IM-1 is rated at 5,000 cycles. For our reliability simulations, we model the SSD array to consist of one of these flash types, whose UBER values are subsequently used to compute RAID reliability estimates, as explained below.

Data Loss Probability (DLP): UBER only gives the probability of an uncorrectable error in a bit. We use UBER to calculate the probability of a data loss in a single SSD and then use it to calculate the same for an entire Diff-RAID array. Specifically, given that a single SSD has failed (or is removed after all its erase cycles are used), we calculate the probability of one or more uncorrectable errors occurring during a RAID reconstruction event; we denote this by 'Data Loss Probability' (DLP). For example, a DLP of 10^{-6} means that 1 out of a million RAID reconstruction events will fail and result in some data loss.

We consider the DLP of an array into two cases: 'oldest' refers to the case where the oldest device in the Diff-RAID array has failed; 'any' refers to the case where a randomly chosen device in the array has failed. In typical scenarios, 'oldest' is the common case for SSD arrays, occurring routinely when a device hits the erasure limit and is replaced by a newer device. Note that in a RAID-5 array, since all the SSDs age at the same rate, no SSD is older than any other and therefore, 'oldest' and 'any' cases are the same.

4.1 Diff-RAID Reliability Evaluation

We measure the reliability of Diff-RAID configurations by running a simulator with UBER data for various flash chips. In most evaluations, we use a synthetic random write workload; we also evaluate the reliability under real-world enterprise workloads using a trace playback. In all the following measurements, we quantify the reliability through DLP; the lower the DLP, the better the reliability.

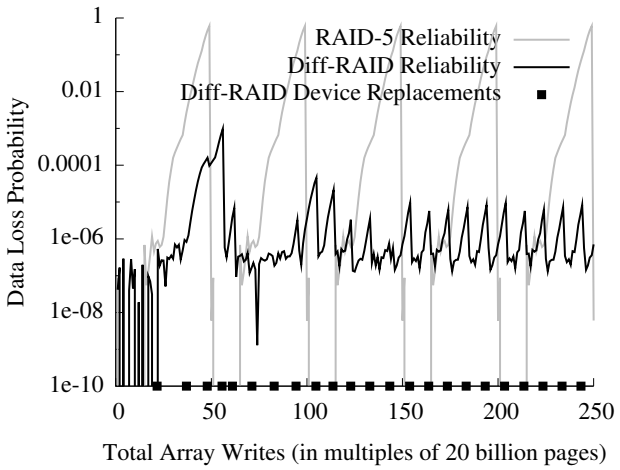

Figure 6. Diff-RAID reliability changes over time and converges to a steady state; the RAID-5 curve from Figure 2 is reproduced here for comparison.

The simulator operates by issuing simulated writes to random blocks in the address space exposed by the array; each of these array writes results in two device writes, one to the device holding the data block and the other to the device holding the parity block. The simulator then tracks the corresponding increase in erasures experienced by each device (assuming a simplistic model of SSD firmware that translates a single write to a single erasure). At any given point in the simulation, the DLP is computed by first assuming that a device has failed and then using the flash UBER to calculate the resulting probability of data corruption. If the 'oldest' failure mode is chosen, the failed device is the oldest one in the array. If the 'any' mode is chosen, the returned DLP value is an average across multiple DLP values, each of which is computed for the failure of a particular drive in the array.

4.1.1 Reliability of Diff-RAID

Figure 6 shows how the DLP of a Diff-RAID array changes as it is written to; in the process, SSDs get replaced at the end of their lifetime and new SSDs are added to the array. Eventually, the DLP converges to a steady state, where it is low and tightly bounded. For this experiment, we simulate a 5-SSD Diff-RAID array with a parity assignment of $(90, 2.5, 2.5, 2.5, 2.5)$ and use the D-MLC32-1 flash (we choose this flash as it has the highest UBER). We assume that each SSD is replaced when it hits the erasure limit of 10,000 cycles.

After 10 to 15 replacements, the reliability curve reaches a steady state; this is a direct result of the age distribution converging. Once the age distribution converges, the data loss probability is bounded between 10^{-7} and 10^{-5}. As mentioned previously, one may start the array in the steady state by using SSDs whose erase cycles have already been

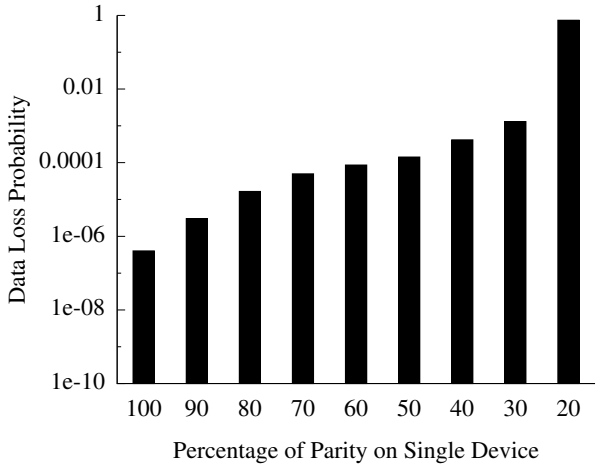

Figure 7. Diff-RAID provides tunable reliability. X-axis shows % of parity on first device, with remaining parity divided evenly across other devices. Left-most bar corresponds to $(100, 0, 0, 0, 0)$ and right-most to $(20, 20, 20, 20, 20)$.

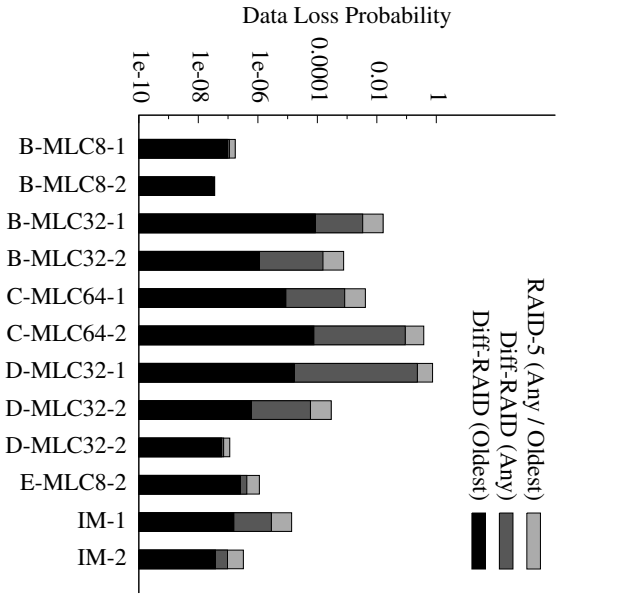

Figure 8. For various flash chips, Diff-RAID is more reliable than RAID-5 under two cases: when the oldest device fails and when any device fails. RAID-5 performs identically in both these cases.

used by appropriate amounts (or replacing them early by treating them as devices that have already used up erase cycles). For the remaining experiments in this section, we focus on the DLP of Diff-RAID at its steady state, ignoring the initial fluctuations.

4.1.2 Reliability of Diff-RAID Configurations

The reliability of Diff-RAID depends on its specific configuration. Figure 7 shows the DLP of different Diff-RAID parity assignments. The DLP is calculated on a model where each SSD in the array uses flash type D-MLC32-1 and the oldest device fails on a 5-SSD array. The left-most bar corresponds to an assignment with 100% of the parity on a single device (*i.e.*, similar to RAID-4). The right-most bar corresponds to an assignment of 20% of the parity to each device (*i.e.*, similar to RAID-5). Intermediate points concentrate some parity on a single device and spread the remainder across all other devices. As expected, the reliability of the array goes up as we concentrate more parity on a single device.

4.1.3 Reliability with Different Flash Types

Next, we want to understand how the reliability of the overall array varies depending on the type of flash chip used. In addition, we also want to estimate the system reliability when any device (not just the oldest device) fails.

Figure 8 compares the DLP of conventional RAID-5 to a specific Diff-RAID parity assignment of $(80, 5, 5, 5, 5)$, for different types of flash chips. For failures involving the oldest device in the array, Diff-RAID is one or more orders of magnitude more reliable than RAID-5 for 7 out of 12 flash types. For failures involving any device, Diff-RAID is twice as reliable as RAID-5 for 9 out of 12 flash types.

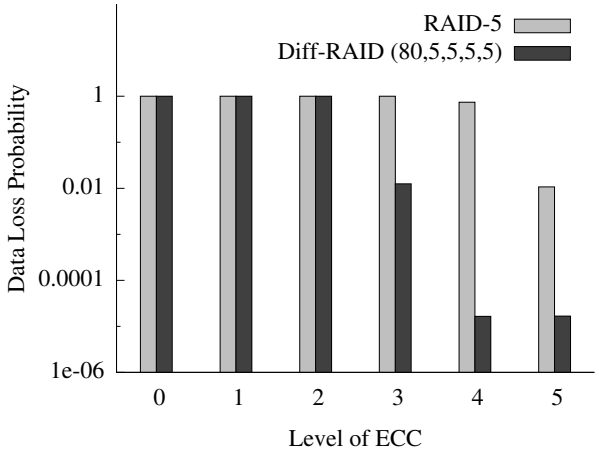

Figure 9. Diff-RAID can lower hardware ECC requirements. For example, Diff-RAID (with (80,5,5,5,5) parity) on 3-bit ECC flash is equivalent to RAID-5 on 5-bit ECC flash.

4.1.4 Reliability with Different ECC Levels

Since Diff-RAID provides better reliability than conventional RAID-5 with the same number of ECC bits, it is also possible that Diff-RAID provides similar reliability to RAID-5 while using fewer ECC bits. Such guarantees can mitigate the need for expensive ECC protection in hardware.

Figure 9 compares Diff-RAID and conventional RAID-5 reliability on flash type D-MLC32-1 with different levels of ECC. Diff-RAID uses a parity assignment of $(80, 5, 5, 5, 5)$, and we consider the failure of the oldest SSD. On the x-

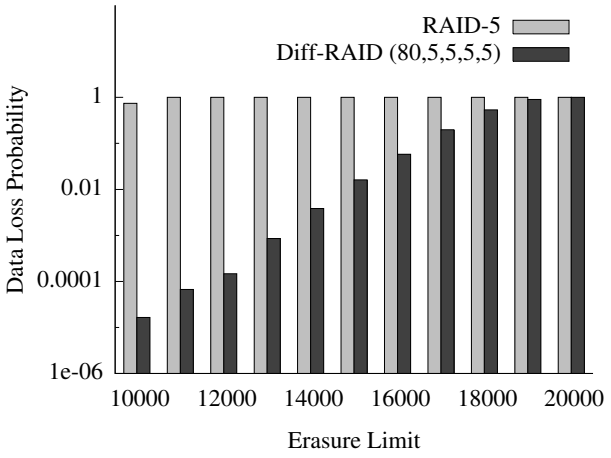

Figure 10. Diff-RAID can extend SSD lifetime by a few 1000s of cycles without high data loss probability.

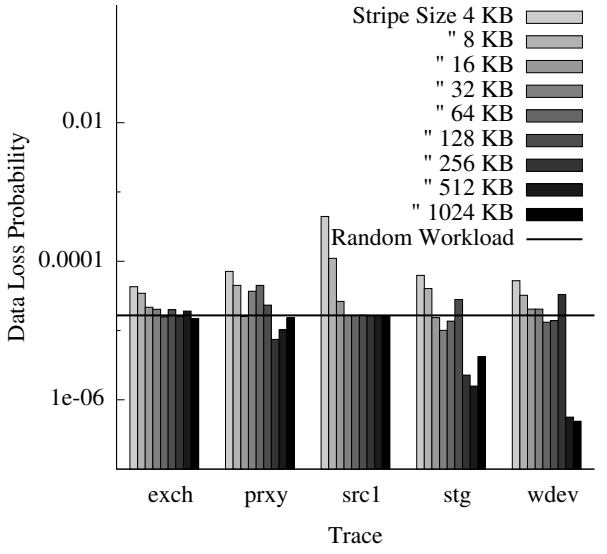

Figure 11. Diff-RAID reliability with different traces and stripe sizes shows that there's no universally good stripe size, but Diff-RAID works well with different workloads if the stripe size is selected carefully.

axis, ECC level 't' represents the ability to correct t bit errors in a 512-byte sector. The important point made by this graph is that Diff-RAID can be used to lower hardware ECC requirements; in this example, Diff-RAID on flash with 3-bit ECC is equivalent to conventional RAID-5 on the same flash with 5-bit ECC.

4.1.5 Reliability Beyond Erasure Limit

Figure 10 compares Diff-RAID and RAID-5 reliability on flash type D-MLC32-1 when the flash is used past its erasure limit of 10,000 cycles. Again, the Diff-RAID assignment used is $(80, 5, 5, 5, 5)$, the failure mode is 'oldest', and we assume 4 bits of ECC. For each bar in the graph, devices in the array are replaced when they hit the extended erasure limit represented by the value on the x-axis. RAID-5 data loss probability hits a value of 1.0 immediately beyond 10,000 cycles, while Diff-RAID's data loss probability climbs slowly and hits 1.0 at 20,000 cycles. It is clear that Diff-RAID allows flash to be used beyond its erasure limit; if each device in the array is used for 13,000 cycles, less than one out of 1000 failure events involving the oldest device will result in some data loss.

4.1.6 Reliability on Real Workloads

Thus far, we evaluated Diff-RAID on a synthetic random write workload. In order to evaluate Diff-RAID's reliability under different workloads, we used a number of enterprise traces taken from a previous study [11]. For real workloads, the stripe size (the size of an individual stripe chunk) is crucial in determining access patterns skews (i.e., a certain device being written to more often) and the frequency of full-stripe writes. We plot the reliability of our synthetic workload as a straight line; the RAID stripe size does not matter at all for purely random writes, since the fraction of full-stripe writes is zero irrespective of stripe size.

In Figure 11, each set of bars corresponds to a particular trace; within a set of bars, the lightest bar corresponds to the smallest stripe size used (4 KB), and each darker bar increases the stripe size by a factor of 2. The darkest bar corresponds to a stripe size of 1 MB. We can see that increasing the stripe size does not affect different traces consistently. Importantly, Diff-RAID continues to provide excellent reliability for multiple real workloads, across all stripe sizes.

Only one of the traces (src1) had a significant number of full-stripe writes; as the stripe size was increased to 16 KB, full-stripes were converted into random writes, resulting in a 30% increase in the number of flash page writes. For all other traces, the fraction of full-stripe writes were low even with a small stripe size, and the difference in the number of flash page writes for different stripe sizes was less than 10% in all cases.

4.2 Diff-RAID Performance Evaluation

To measure the performance trade-offs between RAID-5 and other Diff-RAID configurations, we implemented a software Diff-RAID controller, which uses the Windows raw device interface to access the SSDs. Our software Diff-RAID can run synthetic benchmarks (*e.g.*, random 4 KB writes) and also issue I/Os from trace files. Importantly, we disable the actual XOR parity computation to eliminate the potential CPU bottleneck; we believe that hardware implementations of Diff-RAID will compute XORs using specialized hardware. As mentioned previously, we don't implement write caching in the RAID controller, nor do we optimize half-stripe writes by reading missing data; these optimizations are

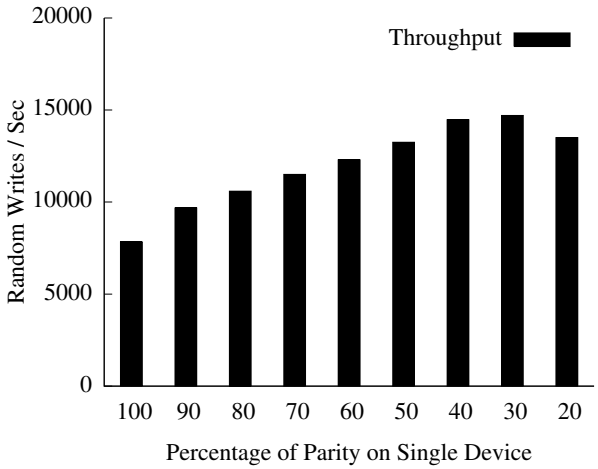

Figure 12. Diff-RAID throughput under various parity assignments.

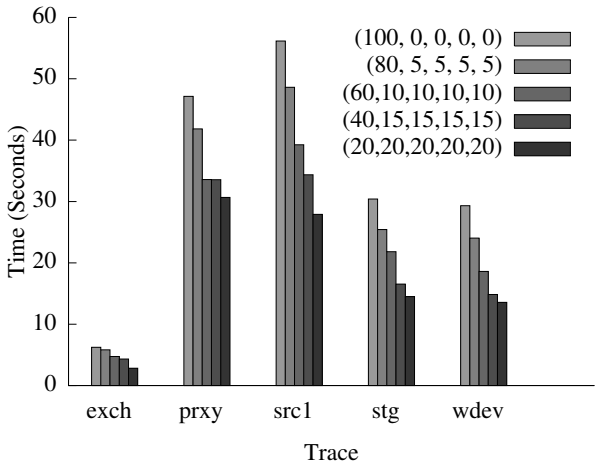

Figure 13. Full-speed trace playback times show that the throughput trade-off holds for non-random workloads.

usually used to improve RAID-5 performance on hard drive arrays, but are not required for SSDs and can hurt Diff-RAID reliability.

We use an array of 5 Intel X25-M SSDs each of 80 GB in size. According to Intel's data sheet, each SSD is specified to provide 3.3K IOPS of random write throughput, 35K IOPS of random read throughput, 70 MB/s of sequential write throughput and 250 MB/s of sequential read throughput [7]. In order to bring the SSDs performance to a steady state, we ran each device through a burn-in process that involved writing sequentially to the entire address space (80 GB) and subsequently wrote 1 million 4K pages at random locations (4 GB of data). In addition, we also ran many microbenchmarks on each device prior to the burn-in process; as a result, all the SSDs had used up between 3% and 8% of their erase cycles.

4.2.1 Diff-RAID Throughput

As explained earlier, Diff-RAID provides a trade-off between reliability and performance. Figure 12 presents the random write throughput on our SSD array setup. We evaluated various Diff-RAID parity assignments from $(100, 0, 0, 0, 0)$ (left most bar), corresponding to a RAID-4 configuration, to $(20, 20, 20, 20, 20)$ (right most bar), corresponding to a RAID-5 assignment. In between these two extremes, we assign a specific percentage of parity on one device and distribute the rest evenly on rest of the SSDs. We notice two points from Figure 12: first, Diff-RAID provides excellent random I/O throughput, varying from 8 K to 15 K IOPS, driving the point that parity-based schemes such as Diff-RAID are suitable for SSD arrays in enterprise settings; second, Diff-RAID is flexible, letting the administrator choose the reliability-performance trade-off suitable for the specific workload setting.

	RAID-5	Naive	Minimal
Reconstruction Time (minutes)	16.68	62.14	26.08

Table 2. Reconstruction time for an 80 GB SSD in a 5-device array. Minimal parity shifting uses only 56% more time than standard RAID-5 reconstruction.

4.2.2 Performance Under Real Workloads

In addition to measuring the performance of Diff-RAID under random writes, we also evaluated the system using the set of enterprise traces described previously. Figure 13 presents the trace playback time when the trace was issued at full speed under various Diff-RAID configurations. Since writes are crucial and take more time to complete, we only issued the writes from each trace. Diff-RAID configurations show a clear trend in the performance (RAID-5 performing the best, RAID-4 performing the worst, and other configurations in the middle) for all the traces.

4.2.3 Recovery Time

Finally, we evaluate the recovery performance of Diff-RAID compared to conventional RAID-5. Diff-RAID recovery time is expected to be higher than RAID-5's due to the required reshuffling of parity assignments. This parity shifting can be performed during RAID-5 reconstruction, either naively (by moving all parity) or using a minimal number of parity block movements. Table 2 presents the recovery time for RAID-5 and Diff-RAID's naive and minimal strategies. By shifting parity minimally, Diff-RAID can recover and move to the next parity configuration with just 56% more time than RAID-5. Note that reconstruction times for SSDs are much shorter than for hard drives, reducing the possibility of another device failure occurring during reconstruction.

5. Related Work

Commodity SSDs are already being used in enterprise settings. A recent case study by Fusion-io reports on the benefits when MySpace moved to solid state storage [4]. The results of our paper can assist administrators in assessing and improving the reliability of enterprise flash deployments.

An example of a research system that uses clustered flash is FAWN, a large array of low-power nodes with flash storage used as a low-power data intensive computing cluster [1]. The principle behind Diff-RAID could be used to enable redundancy schemes within FAWN-like systems that avoid correlated failures.

Since an SSD consists of an array of flash chips, it is possible to provide built-in redundancy within the SSD to handle failures at the chip level. Greenan *et al.* [5] propose erasure coded stripes which span pages from different chips. SSDs with such inherent redundancy can improve the DLP of the whole Diff-RAID array. On similar lines, RamSan-500 is a commercially available SSD that includes built-in redundancy [13]. Unlike commodity SSDs, it uses smaller ECC-based code words (64 bytes instead of the conventional 512 bytes) and applies RAID-3 internally across replaceable flash memory modules. While better than RAID-5, RAID-3 still suffers from correlated failures if the conventional replacement strategy is used.

This paper is a full-length version of an earlier workshop paper on the Diff-RAID system [8]. Several previous works have analyzed the bit error rates of flash chips and showed how they correlate to the age of the flash [3, 6, 10]. None of these studies look at the correlation of failures across multiple SSDs.

Multiple systems have been proposed that use SSDs in concert with hard drives, usually as read caches [2, 9]. While such systems are critical for introducing flash into the storage stack, we believe that there's much value in examining storage designs composed purely of commodity flash, especially if the associated reliability concerns can be countered with redundancy.

6. Future Work

An important avenue of future work involves extending Diff-RAID to multiple parity schemes such as RAID-6 or more general erasure coded storage. We believe that uneven parity placement could improve the reliability of such solutions. It is interesting to note that a simple version of the Diff-RAID technique can be applied to RAID-1, where one of the mirrors is burnt to 50% of its erase cycles; this places the two-device array in a convergent age distribution, with every device replacement occurring when the mirror is at 50% of its erase cycles.

We are currently implementing a prototype version of Diff-RAID within the Windows 7 volume manager; this will enable users to create and manage Diff-RAID arrays via standard administrative interfaces in Windows.

7. Conclusion

Diff-RAID is a new RAID variant designed specifically for SSDs. It distributes parity unevenly across the array to force devices to age at different rates. The resulting age differential is maintained across device replacements by redistributing parity on each device replacement. Diff-RAID balances the high BER of aging devices against the low BER of younger devices, reducing the chances of correlated failures. Compared to conventional RAID-5, Diff-RAID provides a higher degree of reliability for SSDs for the same space overhead, and also provides a fine-grained trade-off curve between throughput and reliability. It can potentially be used to operate SSDs past their erasure limit, as well as reduce the need for expensive ECC within SSDs.

References

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP'09)*, Big Sky, MT, October 2009.

[2] R. Bitar. Deploying Hybrid Storage Pools With Sun Flash Technology and the Solaris ZFS File System. Technical Report SUN-820-5881-10, Sun Microsystems, October 2008.

[3] P. Desnoyers. Empirical Evaluation of NAND Flash Memory Performance. In *The First Workshop on Hot Topics in Storage (HotStorage'09)*, Big Sky, MT, October 2009.

[4] Fusion-io. MySpace Case Study. `http://www.fusionio.com/case-studies/myspace-case-study.pdf`.

[5] K. M. Greenan, D. D. Long, E. L. Miller, T. J. E. Schwarz, and A. Wildani. Building Flexible, Fault-Tolerant Flash-based Storage Systems. In *The Fifth Workshop on Hot Topics in Dependability (HotDep'09)*, Lisbon, Portugal, June 2009.

[6] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *MICRO'09*, New York, December 2009.

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

[8] A. Kadav, M. Balakrishnan, V. Prabhakaran, and D. Malkhi. Differential RAID: Rethinking RAID for SSD Reliability. In *The First Workshop on Hot Topics in Storage (HotStorage'09)*, Big Sky, MT, October 2009.

[9] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud. Intel®turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. *Transactions on Storage*, 4(2):1–24, 2008.

[10] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill. Bit error rate in NAND Flash memories. In *IEEE International Reliability Physics Symposium (IRPS)*, pages 9–19, April 2008.

[11] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to SSDs: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, 2009.

[12] Robin Harris. Why RAID-5 stops working in 2009. `http://blogs.zdnet.com/storage/?p=162`.

[13] Woody Hutsell. An In-depth Look at the RamSan-500 Cached Flash Solid State Disk. `http://www.texmemsys.com/files/f000233.pdf`.