Warped Mirrors for Flash

Yiying Zhang, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
Department of Computer Sciences, University of Wisconsin – Madison
{yyzhang,dusseau,remzi}@cs.wisc.edu

Abstract— **Flash-based devices are cost-competitive to traditional hard disks in both personal and industrial environments and offer the potential for large performance gains. However, as flash-based devices have a high bit-error rate and a relatively short lifetime, reliability issues remain a major problem. One possible solution is redundancy; using techniques such as mirroring, data reliability and availability can be greatly enhanced. All standard RAID approaches assume that devices do not wear out, and hence distribute work equally among them; unfortunately, for flash, this approach is not appropriate as the life of flash cell depends on the number of times it is written and cleaned. Hence, identical write patterns to mirrored flash drives introduce a failure dependency in the storage system, increasing the probability of concurrent device failure and hence data loss.**

We propose *Warped Mirrors* as a solution to this endurance problem for mirrored flash devices. By carefully inducing a slight imbalance into write traffic across devices, we intentionally increase the workload of one device in the mirror pair, and thus increase the odds that it will fail first. Thus, with our approach, device failure independence is preserved. Our simulation results show that across both synthetic and traced workloads, little performance overhead is induced.

I. Introduction

A serious alternative to hard-disk drives has finally arisen. After many years and many promising yet unrealized technologies (e.g., holographic storage [4]), flash-based solid-state drives (SSDs) have gained a foothold in the persistent-storage marketplace and will likely continue to do so in the foreseeable future. While initially deployed in mobile devices and other less traditional computing platforms, lowered costs, increased capacities, and performance advantages have made flash a serious competitor to hard drives in current and future storage systems [8], [16].

One major difference, however, between flash SSDs and hard drives is found in their reliability characteristics. Hard-drive failure arises from a number of causes [21], including mechanical and electrical failure. The primary flash failure mode, however, is based upon usage; specifically, due to the intrinsic nature of the technology, flash drives *wear out* after repeated updates (i.e., program/erase cycles) [3], [17].

In a lone device, wear out is not of particular concern; for example, a manufacturer of a portable MP3 player would simply gauge the expected lifetime of the player and design the flash storage subsystem so as not to exceed said lifetime, perhaps by adding extra capacity or improved wear-leveling [28].

In contrast, when using multiple storage devices, wear out leads to new design challenges. With traditional hard drives, *RAID algorithms* are often employed to provide reliability in the face of device failure [20], [26]. These techniques, whether using simple mirroring [6], parity-based schemes [19], [20], [24], or more complex encodings [15], all make the same underlying assumption: device failure is *independent*. Although this assumption is not perfect [25], it has enabled RAID vendors to provide high-performance, high-capacity systems with excellent reliability characteristics.

Unfortunately, if wear out is the primary cause of failure, existing RAID schemes work quite poorly [17]: device failure becomes highly correlated. For example, consider a mirrored flash pair; the write load to each device is identical; presuming that the underlying technology is similar, the likelihood of wear out at or near the same time is high. If devices wear out simultaneously, the RAID becomes unavailable; worse, data loss is likely.

In this paper, we introduce *Warped Mirrors (WaM)*, a RAID approach for mirrored flash storage systems that addresses the failure dependence problem. We focus on mirroring since mirrored systems give best performance under variety of workloads. The basic idea of Warped Mirrors is simple: by carefully adding a minimal additional write load (via *dummy writes*) to one of the flash devices in the mirror pair, Warped Mirrors induce one device to fail slightly sooner than the other; this artificially induced *failure separation interval (FSI)* allows a system manager to *schedule* device repair and thus replace the faulty device before the second (catastrophic) failure occurs. Such scheduled maintenance is the key to lowering the total cost of ownership, one of the main goals of Warped Mirrors.

One key aspect of the design of Warped Mirrors is that the FSI is *configurable*. The party responsible for device replacement can set the FSI to as short or long of a time window as desired; Warped Mirrors then automatically adjust the imbalance based on both the desired FSI and the current workload to achieve the expected failure separation. For example, if an administrator would like to check and possibly replace a device once a week, the administrator might set the FSI to two or three weeks, thus ensuring that the first failure will be handled before the second failure occurs.

Through detailed simulation, we show that Warped Mirrors achieve the reliability targets with little performance overhead, while minimizing system cost under a wide range of synthetic workloads and real-world traces. Warped Mirrors achieve these goals while making little or no assumption about the underlying SSD (including details such as the exact algorithms of its flash-translation layer), and thus is readily deployable.

978-1-4799-0218-7/13/$31.00 © 2013 IEEE

The rest of this paper is organized as follows. In Section II, we give a brief background of flash memory and its endurance properties. We then describe the basic flash-based mirroring system setting and its endurance problem in Section III. In Section IV, we discuss the basic architecture of *Warped Mirrors*, its modeling component, and an algorithm used by it. In Section V, we describe our simulator, the workloads used and simulation results. Finally, we discuss the related work and present our conclusions and future work.

II. Background

Before presenting Warped Mirrors, we first provide some background information on the relevant aspects of NAND-based flash technology. Specifically, we discuss their internal structure, performance and reliability issues, and controlling firmware design.

There are three types of NAND flash that are typically available. The first is known as Single-Level Cell (SLC), which stores one bit per flash cell. The second is called Multi-Level Cell (MLC), which stores two bits per cell. Another flash type is Triple-Level Cell (TLC), which stores three bits per cell. MLC and TLC are denser than SLC but have a performance and reliability cost. A cell encodes information via voltage levels; thus, being able to distinguish between high and low voltage is necessary to differentiate a 1 from a 0 (for SLC; more voltage levels are required for MLC and TLC).

Modern flash-based SSDs appear to a host system as a storage device that can be written to or read from in fixed-sized units, much like modern hard drives. Internally, however, there are some important differences. A typical flash is organized into a set of *erase blocks*, which are usually between 16KB to 2MB in size. Before writing to the flash, an entire block must be erased, which sets all the bits in the block to 1. Writes (which change some of the 1s to 0s) can then be performed to the newly-erased block in units of *pages*, which are typically 2KB, 4KB, or 8KB; sometimes this step is known as *programming* the flash. In contrast to this intricate and expensive procedure, reads are relatively straightforward and can be readily performed in page-sized units.

Writing is thus a noticeably more expensive process than reading. For example, Grupp *et al.* report typical random read latencies of 12 μs (microseconds), write (program) latencies of 200 μs, and erase times of roughly 1500 μs [13]. Thus, in the worst case, both an erase and program are required, and a write will take more than $100\times$ longer than a read ($141\times$ in the example numbers above).

An additional problem with flash is its endurance; each program/erase (P/E) cycle does some damage to the cell, and over time it becomes increasingly difficult to differentiate a 1 from a 0 [2]. Thus, each block has a *lifetime*, which gives the number of P/E cycles that the device should be able to perform before it fails. Typical values reported by manufacturers are 100,000 cycles for SLC flash and 10,000 for MLC, though some devices begin to fail earlier than expected [13].

For both performance and reliability reasons, most flash devices include a *Flash Translation Layer* (FTL) that manages the underlying flash devices and exports the desired disk-like block interface as described above [14]. FTLs serve two important roles in flash-based SSDs; the first role is to improve performance, by reducing the number of erases required per write. The second role is to increase the lifetime of the device through *wear leveling*; by spreading erase load across the blocks of the device, the failure of any one block can be postponed (although not indefinitely).

Both of these roles are accomplished through the simple technique of indirection. Specifically, the FTL maps logical addresses (as seen by the host system) to physical blocks (and hence the name). Higher-end FTLs never overwrite data in place; rather, they maintain a set of “active” blocks that have recently been erased and write all incoming data (in page-sized chunks) to these blocks, in a style reminiscent of log-structured file systems [22]. Some blocks thus become “dead” over time and can be garbage collected; explicit cleaning can compact scattered live data and thus free blocks for future usage. Recently, techniques have been proposed to remove the indirection in flash-based FTL by storing block physical addresses directly in file systems [29].

III. Flash-based RAIDs

While single flash-based SSDs will be popular in many settings, collections of SSDs, deployed in a RAID-like configuration, will be important as well. In some cases, *performance* demands will be the reason for using multiple flash devices; in others, *capacity* is the issue, particularly given the relatively-low densities of flash as compared to disk.

Whatever the reason for use of multiple SSDs, some form of redundancy will likely be needed, in order to protect against data loss and maintain availability when a device fails. Indeed, reliability alone is sometimes reason enough to use multiple SSDs, even if performance and capacity demands do not require more than a single device.

A. The Problem: Failures Are Not Independent

The problem that arises in any flash-based RAID is a simple one: device failures are no longer independent. If wear out is the primary cause of failure of a flash, it is likely *all* the devices in the given RAID will wear out at nearly the same time, thus greatly increasing the chances of downtime or data loss or both. All current RAID schemes assume failure independence; when one device fails, it is thus presumed that the other devices will be available to service requests as well as reconstruct the data from the lost device.

One underlying assumption of RAID, that balancing load across devices is a good idea for performance, is also at the heart of the problem. For example, with simple striping (RAID level 0), writes to the RAID are generally spread evenly across the devices. While good in a typical RAID, perfect load balance exacerbates the problem with devices that wear out with usage.

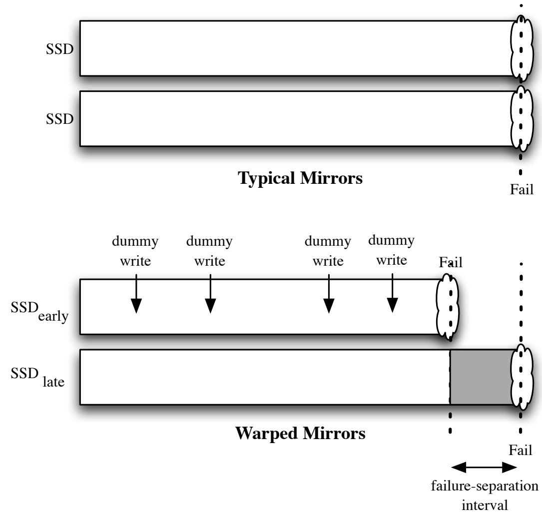

Fig. 1. Typical Mirrors vs. Warped Mirrors

B. The Solution: Warped Mirrors

Warped Mirrors (WaM) present one attractive solution to the failure-dependence problem. The approach is quite simple: by placing a small, carefully-controlled extra write load on one device in a mirror pair, WaM induces a (slightly) early failure of one device in the pair, and thus provides a window of opportunity to install a new device before the other device in the pair fails. We refer to each extra write as a *dummy write*, and the window between failures the *failure separation interval (FSI)*.

For example, see the timeline in Figure 1. In the topmost diagram in the figure, typical mirroring is used; the result is that two failures occur at nearly the same time. In the bottommost diagram, WaM applies dummy writes and thus one device (designated SSD$_{early}$) fails slightly before the other (SSD$_{late}$).

Central to the design of WaM is the *configurability* of the FSI. The responsible party for the health of the RAID (i.e., a storage administrator, or in a home setting, the user) can set this parameter according to the needs of the site, and the frequency with which replacement is possible; the longer the interval, the lower the cost. This choice presents a clear trade-off to the administrator: longer intervals lower the cost of maintenance but exhibit a performance cost (as we will show in some detail in the coming sections).

Further note that this entire process could be automated. For example, many high-end RAID systems have "phone home" capabilities that essentially allow a system, once failure is detected, to inform the RAID manufacturer of the problem. The manufacturer, in turn, could respond by sending a replacement drive (or drives) to the site of the deployment, where a local administrator (or employee of the RAID manufacturer) would simply follow simple instructions such as "install these drives in these slots." In such a scenario, the manufacturer would set the FSI accordingly, taking into account factors such as shipping delays and likely availability of a local presence to perform the repair.

Of course, mirrors are not the only important RAID configuration, but they attracted our focus for numerous reasons. First, mirrored systems give the best performance under a variety of workloads, particularly those with many "small" writes (*e.g.*, transactional workloads). Further, mirroring is the most conceptually simple of any approach to redundancy; storage systems like GFS [11] uses simple redundancy instead of more complex RAID constructions. Finally, and perhaps archaically, some finicky storage administrators (still) do not trust storage systems that store data in anything other than their plain raw form; even established technologies such as parity-encoding are viewed with a cautious eye by this suspicious lot.

C. Challenges

There are numerous challenges that must be overcome in building Warped Mirrors. We present and discuss said challenges below.

1) Subverting the FTL: Without an FTL, writes in the logical space of a flash SSD are mapped directly to the physical flash, and thus causing the device to perform an erase is straightforward; one simply needs to overwrite a recently-written block. In doing so, an erase must be first performed by the flash controller in order to re-program the block.

When an FTL is present (which we assume will be the common case), such a simple strategy is not guaranteed to generate an erase; the overwrite may be mapped to a block that has previously been erased. Thus, without information on the internal operation of the SSD (which is undoubtedly difficult or impossible to come by), any approach can only be at best approximate.

To generate a proper erase load despite the presence of an FTL and to thus operate upon any SSD regardless of underlying FTL details, we adopt the following approach: with some probability P_{dummy}, the system will issue a dummy write to SSD$_{early}$. The key to WaM is thus the control of this probability to achieve the desired FSI, and thus adjusting this value is critical; therefore, we will develop a detailed model of SSD operation and use said model to calculate P_{dummy} accordingly. The development of this model is intricate and thus we present it in its own section of the paper (§IV).

However, the exact method of issuing a dummy write remains undefined. One method we will explore simply repeats the last write that took place; we call this method the "repeat last" dummy write. The benefit of this approach is that it is simple to implement. A second approach picks a purely random page, reads it in first, and then issues a write to it (with the same data) as the dummy write. We call this approach "random page". Although this approach has a slightly higher cost (due to the extra read), it is generally more robust to the underlying wear-leveling algorithm.

2) Achieving Near-Perfect FSI: Because of the approximate nature of our approach, it is possible that WaM will not achieve the desired FSI through dummy writes alone. There are two cases we must consider: SSD$_{early}$ failing too early, and SSD$_{early}$ failing too late.

If SSD$_{early}$ fails too early, the system is in excellent shape from a reliability standpoint; the administrator, in this case, will have *extra* time to service the system and thus the chance of data loss is reduced. The cost of failing too early is performance, as the load imbalance affected by WaM was too great in this case; WaM can take this situation into account in the future to lessen the imbalance and more perfectly achieve the desired FSI.

If SSD$_{early}$ fails too late, the system has a problem: the FSI "guaranteed" by WaM may not be met, and thus the devices may fail too close in time. The result could be an unexpected data loss, as the service check comes too late to handle the failure. To remedy this situation and to achieve the specified FSI, WaM enters *degraded mode*, in which it begins to service writes to the remaining drive SSD$_{late}$ more slowly (reads are still served at normal speed). By slowing down writes during the failure window, WaM effectively delays the failure of the second flash in the pair and thus ensures that the FSI contract is properly fulfilled.

Deciding the degraded-mode slowdown factor, R_{delay}, also requires some care. Thus, we further develop the algorithm again in a subsequent portion of the paper (§IV).

3) The Restoration Process: Our goal, as described above, is to enable administrators to make service of flash-based RAIDs *schedulable*; by separating the failure time of each device in a pair, WaM allows administrators to follow a fixed service routine, thus only needing to be able to service devices at periodic intervals (say once every two weeks). However, this approach still leaves unanswered the following question: what is the exact restoration process?

Our approach is to replace the failed drive (SSD$_{early}$) with a new drive, restore its contents with a full copy from the existing drive (SSD$_{late}$), and then to replace the yet-to-fail drive (SSD$_{late}$) immediately and restore it from the newly-installed first replacement. This approach has many benefits, including its simplicity; when finished, the system is in the exact same state as before, and thus the entire WaM process can begin again.

An alternate approach would only replace SSD$_{early}$ at this point and wait for SSD$_{late}$ to fail before replacing it. The negative of this approach is that service of the pair now requires two interactions (not one), and thus increases maintenance cost. The slight benefit, however, is the small amount of extra lifetime one achieves from SSD$_{late}$; we feel that this small benefit is not worth the cost.

4) Generalizing to Striped Mirrors: Our discussion thus far has centered around a single pair of devices; however, high-end mirrored storage consists of many devices, arranged in either RAID-10 fashion (striping across mirrored pairs) or RAID-01 (mirroring across two striped arrays). Thus, we now describe how WaM operates in said environments.

Overall operation is quite similar, except now half of the drives are treated as SSD$_{early}$ was and the other half as SSD$_{late}$. Thus, half of the drives will fail ahead of schedule, and should thus be replaced as before, with the subsequent replacement of all SSD$_{late}$ drives. In this manner, our approach works equally well on large mirrored storage systems as it does on just a single mirror pair.

D. Alternate Approaches

Before closing this section, we discuss some alternate approaches and their strengths and weaknesses as compared to Warped Mirrors. One straightforward change to flash devices that would make building reliable storage on top of them is to export wear out information to hosts. If such information was available and could be trusted, it would to some extent obviate the need for our approach, as a RAID controller could simply monitor said information and replace SSDs as they neared their wear-out point.

We believe that such information may be hard to come by, as it would require all manufacturers to agree upon a standard interface to export it. Further complicating the issue is that the presence of such information exposes SSD manufacturers to detailed studies of their wear-leveling algorithms. As many of these algorithms are proprietary, manufacturers may be loathe to export an interface which enables their reverse engineering. Finally, there is little incentive for honesty; if one "honest" company's device reports failures accurately, whereas another "dishonest" company does not, the dishonest company may gain competitive advantage as a drive that does not fail as quickly.

Another approach would be to proactively replace both mirrors at a fixed time, well before the manufacturer-guaranteed wear out time. Unfortunately, without precise information (as above), this approach is likely to be costly, as one would likely have to replace devices very early to feel comfortable that they would not fail before expected. As others have shown, sometimes devices wear out before the manufacturer claims, particularly true for high-density MLC or TLC flash SSDs [13].

The SMART (Self-Monitoring, Analysis, and Reporting Technology) interface was designed to provide an early warning of hard disk drive failures using various indicators. It has been extended to indicate SSD life time as well. However, we believe that the SMART interface is insufficient for a couple of reasons. First, not all SSDs have SMART support. There are also compatibility issues with different softwares interpreting the SMART information from different hardwares. Second, the lifetime reported by SMART is an estimated normalized value. One has to either proactively replace SSD mirrors or risk the danger of data loss using the SMART interface.

IV. WaM Control Algorithms

We now explain how the control algorithms of Warped Mirrors operate. These control algorithms take as input the desired *failure separation interval (FSI)* and control the load imbalance to each mirror pair, with the goal of achieving the desired FSI.

A. Basic Control Algorithm

We now develop the WaM model to approximate how much load imbalance to induce. Recall that our goal is to calculate P_{dummy}, which is the probability that the controller will generate an extra dummy write to SSD$_{early}$; the higher this probability, the sooner SSD$_{early}$ will fail, and thus the larger the FSI will be.

Assume for simplicity that each SSD has a fixed lifetime (given by the manufacturer), which is determined by the number of P/E (program/erase) cycles per block. We call this value N_{worn}, as it is the number of erases to a block that causes it to become worn out. Note that a more advanced (and perhaps realistic) model could be used here (e.g. a probability distribution centered around N_{worn}), but as we will see, this approach is sufficient as we are only generating an approximation. We further assume the presence of an FTL that writes to the devices in a log-structured fashion and performs perfect wear-leveling. With these assumptions in place, we now develop a model that allows us to calculate P_{dummy}.

To do so, first we need to define a few new terms. The first is R_{erase}, the ratio of the number of erases performed on SSD$_{early}$ divided by the number of erases performed on SSD$_{late}$:

$$R_{erase} = \frac{N_{erases}^{early}}{N_{erases}^{late}}. \tag{1}$$

R_{erase} is thus always greater than 1, as we are placing an extra write (and thus erase) load onto SSD$_{early}$.

R_{erase} is clearly dependent on P_{dummy}. Here, because we assume that each additional dummy write is simply written to an "active block" that is the target of current writing activity within the device, we can conclude that:

$$R_{erase} \ = 1 + P_{dummy}. \tag{2}$$

Put another way, P_{dummy} just serves to increase the total write load on the device; because all writes are transformed into sequential writes to the active block, increasing P_{dummy} linearly increases the number of erases performed on the device; as an example, consider setting P_{dummy} to 1; in this case, each write generates an additional dummy write, and the number of erases is doubled.

The question then becomes: given a particular R_{erase}, how many erases are left remaining on SSD$_{late}$ once the failure of SSD$_{early}$ occurs? By definition, the number of erases remaining (per block), $N_{remaining}^{late}$, is equal to the number of maximum possible erases (per block) N_{worn} minus the number that have been issued, N_{erases}^{late}:

$$N_{remaining}^{late} = N_{worn} - N_{erases}^{late}. \tag{3}$$

However, we know that the early-failing device SSD$_{early}$ has now failed; thus $N_{remaining}^{early} = 0$ and $N_{erases}^{early} = N_{worn}$. Because the failing device SSD$_{early}$ has received proportionally R_{erase} times more erases, we can conclude that the number of erases performed thus far on SSD$_{late}$ is:

$$N_{erases}^{late} = \frac{N_{worn}}{R_{erase}}. \tag{4}$$

Thus, plugging Equation 4 into Equation 3, we find that the number of remaining erases (per block) on SSD$_{late}$ is:

$$N_{remaining}^{late} = N_{worn} - \frac{N_{worn}}{R_{erase}}. \tag{5}$$

Recall our goal is to have SSD$_{late}$ fail a specific amount of time after SSD$_{early}$; we call the time of this failure-separation interval T_{FSI}. Given this time, we can estimate the number of I/Os that take place during the time interval, using historical averages of the current workload. Once we know this value, we can estimate the number of erases that occur during the FSI, and, combining that result with Equation 5, solve for the desired P_{dummy}.

In any time T, we know that the number of I/Os ($N_{I/Os}$) that take place is:

$$N_{I/Os} = \frac{T}{T_r + T_i} \tag{6}$$

where T_r is the average I/O response time of the device and T_i is the average inter-arrival time of requests. To calculate the number of erases that occur per I/O, we need to know the percentage of I/O requests that are writes, P_{write}. With this value, we can directly estimate the total number of erases (across all blocks) that occur in time T as a function of T:

$$N_{erase}^{total}(T) = \frac{T}{T_r + T_i} \times P_{write} \times \frac{N_{page}}{N_{block}} \tag{7}$$

Here, we estimate the number of erases that occur per write by multiplying the I/O write rate by the ratio of page size to block size; because the FTL will turn all writes into sequential writes, only after $\frac{N_{block}}{N_{page}}$ writes will an erase have to be generated (in the steady state), and hence the equation above.

However, what we are interested in is the number of erases that occur per block. To calculate this value, we assume perfect wear-leveling, and assume the erase load is spread evenly across all N_{ssd} blocks in the SSD. Thus, the number of erases that occur to a single block (and indeed, to all blocks) in time T is:

$$N_{erase}^{single}(T) = \frac{T}{T_r + T_i} \times P_{write} \times \frac{N_{page}}{N_{block}} \times \frac{1}{N_{ssd}} \tag{8}$$

We can now solve for R_{erase} and thus for P_{dummy}, by combining Equations 5 and 8. From Equation 5 we know the number of remaining erases in SSD$_{late}$, and we can simply plug in T_{FSI} as our time T to T_{FSI} to find the number of erases generated during the failure-separation interval, as we do here:

$$N_{remaining}^{late} = N_{erase}^{single}(T_{FSI}), \tag{9}$$

which expands to:

$$N_{worn} - \frac{N_{worn}}{R_{erase}} = \frac{T_{FSI}}{T_r + T_i} \times P_{write} \times \frac{N_{page}}{N_{block}} \times \frac{1}{N_{ssd}}. \quad (10)$$

Solving for R_{erase}, we find that:

$$R_{erase} = \frac{N_{worn}}{N_{worn} - \frac{T_{FSI}}{T_r+T_i} \times P_{write} \times \frac{N_{page}}{N_{block}} \times \frac{1}{N_{ssd}}}. \quad (11)$$

Calculating P_{dummy} from this is trivial (recall from Equation 2 that $P_{dummy} = R_{erase} - 1$).

B. Handling Imperfection in FSI

As discussed earlier, because of the assumptions the model must make, the actual delivered T_{FSI} may not match the desired intent. If it is longer, there is nothing else to do. However, if it is shorter, there is a danger that the repair will not take place in time, and thus both unavailability and data loss may arise.

To handle this second case, the WaM control algorithm notices when a failure occurs not at the expected time in SSD$_{early}$ and subsequently slows down the rate of write requests to the device so as to achieve the desired FSI. We call this slowdown ratio R_{delay}.

$$R_{delay} = \frac{N^{late}_{remaining\_target}}{N^{late}_{remaining\_actual}} \quad (12)$$

C. Assumptions and Limitations

Our model makes a number of assumptions in order to calculate P_{dummy}. We now address the importance of these assumptions and how they affect the effectiveness of Warped Mirrors.

1) Imperfect Wear Leveling: One assumption our model above makes is that the wear-leveling of the FTL is nearly perfect. Of course, depending on the particular flash device, different wear-leveling algorithms may be used, and their success at spreading the erase load across the blocks of the flash may vary.

Our assumption of perfect wear-leveling can cause our estimation for T_{FSI} and thus P_{dummy} to be inaccurate. Fortunately, we can use R_{delay} in the degraded period to achieve the original targeted T_{FSI}. If a device does not spread erase load perfectly, it will cause the device to fail earlier than expected. In this case, both SSD$_{early}$ and SSD$_{late}$ may fail earlier than expected. In other words, our estimation is off by a factor that can be estimated by the expected time of failure of the SSD$_{early}$ divided by the actual time of failure; T_{FSI} thus will also be shorter than expected by this factor. We then set R_{delay} to this factor in the degraded period to achieve the desired T_{FSI}.

2) Workload Changes: Our model has a clear workload-dependent component, in that it uses both the rate of I/O (overall) as well as the percentage of writes in the workload to calculate P_{dummy}. Thus, our approach is sensitive to changes in the workload, both during normal operation and particularly during degraded mode when only SSD$_{late}$ is still operational.

To handle workload changes, WaM continually monitors the workload and adjusts P_{dummy} as need be. Thus, during normal operation, P_{dummy} may fluctuate up and down depending on the current write load. In the worst case, the change in workload occurs immediately after the failure of SSD$_{early}$. Even in this case, WaM can handle the problem; by slowing down writes appropriately (as described above), the desired FSI can be reached.

3) Incorrect SSD Lifetime: We use a fixed SSD block lifetime of maximum erases in our model. However, in reality such maximum erase count is only an estimation and may be incorrect. Also, flash bit error rates increase over time [13].

If we detect that SSD$_{early}$ fails earlier than expected, then SSD$_{late}$ will also likely fail earlier than expected. In this case, we use a proper R_{delay} to slow down writes to SSD$_{late}$ to achieve the desired T_{FSI}. If instead SSD$_{early}$ fails later than expected, then the actual T_{FSI} will likely to be longer than the expected target and we do not need to do anything.

4) Device Parameters: One further assumption our model makes is that the block size and the page size of the underlying devices are known. As most manufacturers are open about these numbers, we do not feel that this assumption is unreasonable.

However, if said parameters did become hidden in the future, it would not be overly difficult to discover them via offline profiling of the device, similar to Saavedra and Smith's cache hierarchy discovery tool [23]. By presenting the device with different I/O patterns and measuring the time, a good estimate of the likely block and page sizes could readily be generated.

5) What Happens Without An FTL?: One problem we do not address is what happens when you use Warped Mirrors upon a device without an FTL. In such a system, our estimation of how many erases are being generated by dummy writes can be quite off; specifically, each dummy write may trigger an erase (as opposed to one roughly every $\frac{N_{block}}{N_{page}}$ writes), since it writes to a new block different from the previous write.

In this scenario, SSD$_{early}$ will fail dramatically earlier than expected, which is not desirable. One possible solution would be to "detect" when one is running without an FTL through performance measurements of on-going I/O. For example, one could observe the latency of the immediate over-write of a recently-written block; if said latency is always "high" (*i.e.*, near the cost of an erase), one could conclude with high probability that the FTL is not performing remapping.

However, given that most current and perhaps all future SSDs will contain some kind of wear-leveling FTL, the benefits afforded by such a mechanism are not likely worth the costs. Moreover, a Warped Mirror manufacturer could solve this problem by construction, *i.e.*, never ship a mirrored system with anything but certified wear-leveling SSD components.

V. Evaluation

We now evaluate Warped Mirrors in terms of reliability, performance, and monetary cost. In this analysis, we use detailed simulation to see how WaM behaves under both

TABLE I

SIMULATION PARAMETERS.

Symbol	Value
Time for a page read	25 μs
Time for a page write	200 μs
Time for block erase	1.5 ms
Block Size	256 KB
Page Size	4 KB
N_{worn}	10,000
Cost (SSD)	$99

TABLE II

TRACED WORKLOAD PROPERTIES.

Trace	Number of Requests	Read Percent	Random Write Percent	Average Request Size
Postmark	62,257	83.2%	15.1%	222.32 KB
TPC-C	6,832,380	64.6%	35.4%	8.20 KB
WebSearch	1,055,448	99.9%	0.1%	15.14 KB

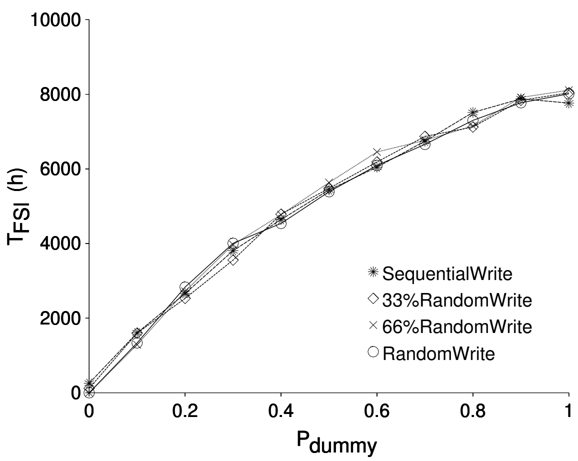

Fig. 2. T_{FSI} with workloads of different percentage of random and sequential writes with wear-leveling FTL.

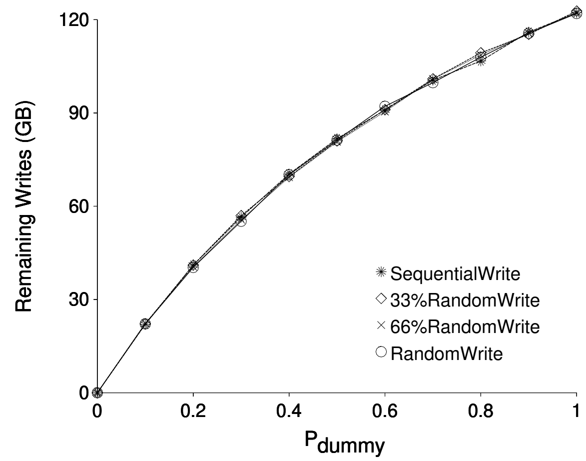

Fig. 3. Remaining writes on the surviving drive with workloads of different percentage of random and sequential writes with wear-leveling FTL.

synthetic and real workloads. As we will see, WaM delivers its reliability goals with low monetary cost and little performance overhead.

A. Simulator

We use the DiskSim simulator and its SSD extension as the base of our simulation environment [7]. We modified DiskSim in several ways to support Warped Mirrors. Specifically, we intercept write requests before the RAID mapping takes place and add dummy writes to one flash drive (SSD$_{early}$) if necessary. Dummy writes are either issued immediately after the completion of the write operation on SSD$_{early}$, or delayed until when there are enough read requests to be performed in parallel on SSD$_{late}$ together with the dummy write to SSD$_{early}$.

We consider a flash drive dead whenever its first block dies; with good wear-leveling, this behavior is a reasonable approximation of what should occur. After the first drive dies, we continue to serve requests from the remaining drive (SSD$_{late}$). To slow down service during this time (if needed), we simply delay the response from SSD as need be.

In our simulation, we use a pair of identical MLC SSDs. Table I gives basic configurations of SSDs used in our experiments. We set SSD effective storage space to 80 GB; each SSD is 20% overprovisioned, which means the raw disk capacity is 100 GB. The SSD price is taken from a typical online cost of an Intel 80GB MLC NAND SSD as of the date of submission [1].

B. Workloads

We use both synthetic and traced real workloads to analyze Warped Mirrors. To model a system under heavy load, we use zero think-time between requests. Thus, when a request finishes, the next request is issued immediately. We believe this to be a reasonable assumption in our target environment of large-scale data centers.

We change the percentage of random vs. sequential writes to generate synthetic workloads. The size of each request is a single page (4 KB). We don't include reads in these synthetic workloads, since only writes will cause erases and affect SSD failures. We show performance with both read and write requests using traced workloads.

We use workloads from two benchmarks, Postmark, TPC-C, and traces from a web search engine [27]. Instead of using the original arrival times, we again model these traces as if they were in a heavily-loaded system, *i.e.*, with no think time between requests. Properties of these traces are summarized in Table II.

C. Simulation Results

We now present the results of our analysis. We investigate how effective dummy writes are in achieving a failure separation and show their performance costs. We also explore the utility of degraded-mode delays. Although most of our investigations are with synthetic workloads, we also show that real workloads can successfully use WaM without too high of

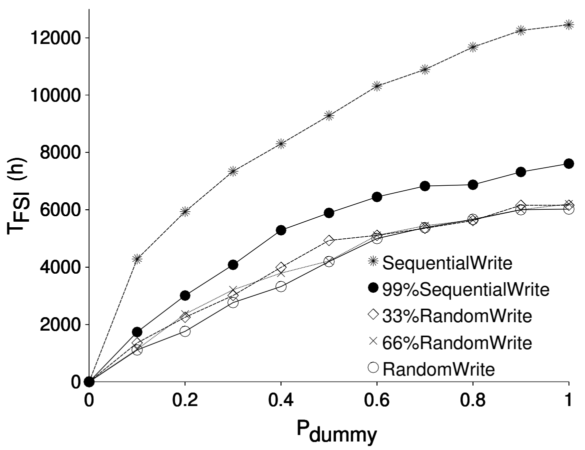

Fig. 4. T_{FSI} with workloads of different percentage of random and sequential writes without FTL.

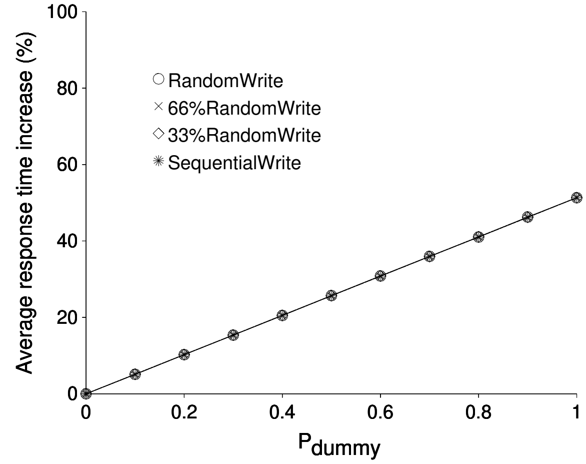

Fig. 5. Average response time (ms) with workloads of different percentage of random and sequential write requests.

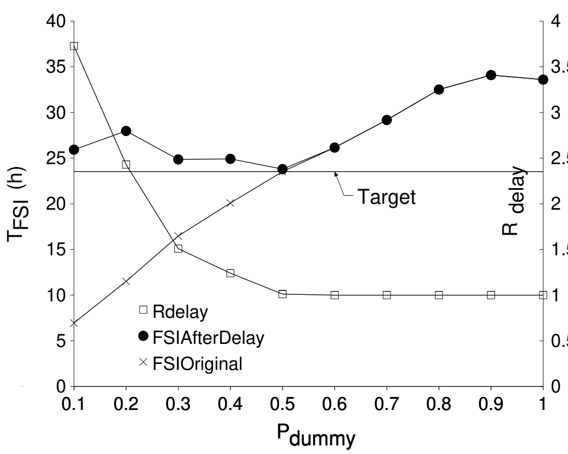

Fig. 6. Effect of adding delay in degraded period to adjust T_{FSI} when P_{dummy} is not acurate.

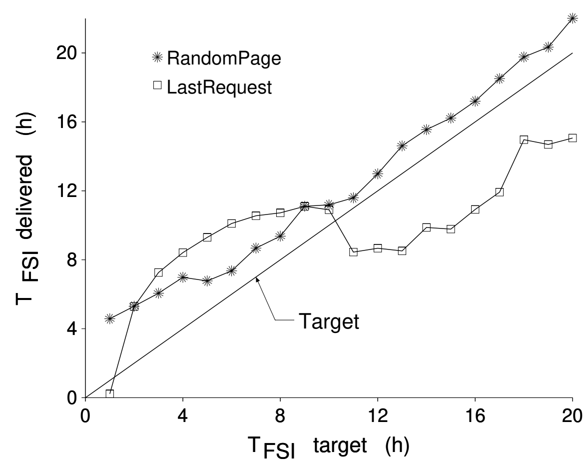

Fig. 7. Delivered T_{FSI} using dummy write page chosen at random and as last request page.

a performance cost. Finally, we investigate the monetary cost of our approach, and show that it is low.

1) Separating Drive Failures with P_{dummy}: We first show the effect of varying P_{dummy} on a modern flash and its reliability characteristics. Specifically, we answer the question: can we separate drive-failure times by increasing P_{dummy}?

Figure 2 plots the failure-separation interval (FSI) as a function of P_{dummy}. In this experiment, we place a synthetic load of all writes on the mirror pair, and vary whether the workload writes purely sequentially, purely randomly, or somewhere in-between. We continuously run the workload until both drives fail; we calculate FSI as the time between said flash-drive failures. We also calculate the number of writes that go to SSD$_{late}$ after SSD$_{early}$ dies and before SSD$_{late}$ dies; the results are shown in Figure 3.

As we can see from Figure 2 and Figure 3, on a modern flash with wear leveling, increasing P_{dummy} has the effect of increasing the FSI, as desired. Simply put, by placing an increased write load on one flash drive in the mirror pair, we can cause it to fail slightly earlier than the other drive in the pair as we had hoped.

Interestingly, these graphs also show how wear-leveling has the effect of homogenizing flash drives. Despite the different synthetic workloads presented to the drives, the delivered FSI is the same (given a particular P_{dummy}). The reason, of course, is the nature of the wear-leveling FTL; by transforming all writes into writes to an “active” block, writes are all performed as if they are sequential, and thus the failure properties become less workload dependent.

To confirm that this is the case, we also configured our SSD simulator to write directly to the drive (without a wear-leveling FTL). The results of this experiment, shown in Figure 4, show that without an FTL, FSI is quite dependent on workload. Clearly the presence of wear-leveling makes our task of separating drive-failure time easier.

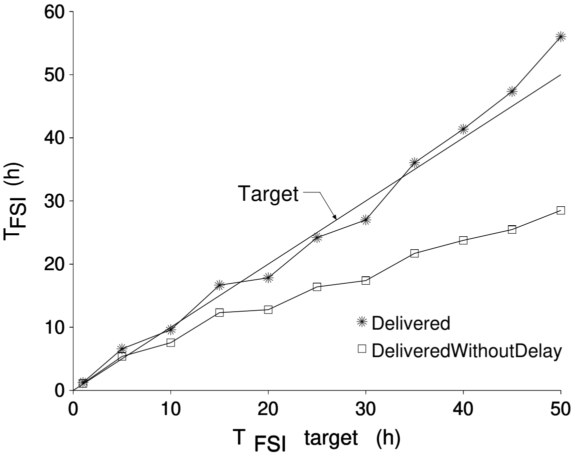

Fig. 8. Delivered T_{FSI} with and without delay with different target T_{FSI} (sequential workload).

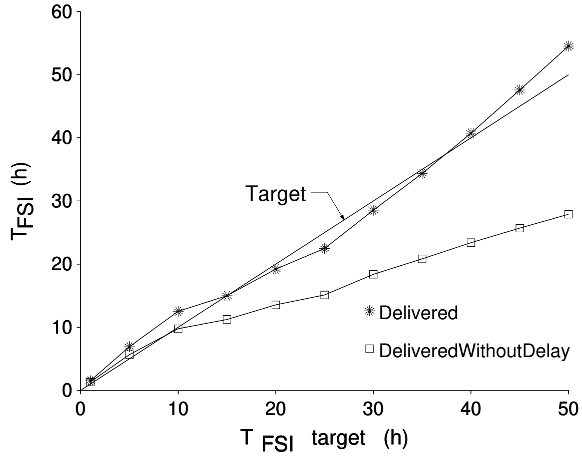

Fig. 9. Delivered T_{FSI} with and without delay with different target T_{FSI} (random workload).

2) Performance Effect of P_{dummy}: We now show the performance impact of increasing P_{dummy}. We use the same synthetic workloads as above. Figure 5 shows the slowdown percentage under an increasing fraction of dummy writes placed on SSD_{early}.

As we can see from the figure, the effect is quite linear; more dummy writes to SSD_{early} lead to slower overall performance for the workloads. Thus, our basic trade-off: the greater P_{dummy} is, the greater the FSI will be, but the worse the impact on performance will be.

3) Using Degraded Mode Delays: Despite our best intentions, the FSI delivered by our model when just using dummy writes will not always be perfect. In these cases, WaM uses degraded mode write delays (R_{delay}) to achieve the desired FSI. We now investigate the utility of this approach.

Figure 6 shows the delivered FSI as P_{dummy} is varied. In this experiment, P_{dummy} should be set to around 0.5 to achieve the desired FSI; when it is lower, the FSI will be too low, and when higher, the achieved FSI too high. The graph confirms this expected behavior.

The graph also shows that in the case where P_{dummy} is too low, increasing R_{delay} has the effect of increasing the FSI to the desired level. When P_{dummy} is too high, however, no such fix can be achieved; in these cases, the delivered FSI will simply be too high, and R_{delay} will be set to 1 to make it no worse.

4) The Importance of the Dummy-Write Algorithm: We now show the importance of the dummy-write algorithm as described earlier. Specifically, we show the effect of using the two different approaches on the ability of WaM to achieve the desired FSI. Our results are presented in Figure 7.

From the figure, we can see that the "repeat last request" approach is not as stable as the "random page" approach that picks a random page, reads it, and then writes it back to the flash. For this reason, the "random page" approach to generating dummy writes is used throughout these results.

5) Achieving the Correct FSI: Finally, we now investigate whether our model accurately chooses P_{dummy} in order to generate the desired FSI. To do so, we again run our synthetic write workloads and compare the input (desired) FSI to the output (actual) FSI. Figures 8 and 9 show the delivered FSI as a function of the input FSI under sequential- and random-write workloads, respectively.

In both cases, when just using dummy writes, our model does not quite generate the desired FSI. The reason for this is that our assumption about wear-leveling is too simplistic; exactly perfect wear-leveling is unlikely to occur in practice. Thus, without any extra action, the drives would fail too closely together and thus endanger the reliability of the storage system.

Fortunately, in both cases, using an additional delay during degraded mode (R_{delay}) allows us to quite closely deliver the desired FSI. WaM recognizes that the desired FSI is not going to be met; it then slightly increases the delay of writes during degraded mode (after the first drive failure), and in doing so achieve our goal.

6) Macro-benchmarks and Traces: We now show the reliability and performance effects on macro-benchmarks and real traces. Specifically, we use workloads from the Postmark benchmark, the TPC-C benchmark, and the WebSearch trace [27]. We measure the effect of adding dummy writes on the performance and reliability characteristics of said workloads.

Figure 10 shows that the real workloads behave much in the same way as our synthetics in terms of the effects of varying P_{dummy} on the delivered FSI. Thus, under real workloads, the desired failure-separation window can be achieved.

More interesting is the question of performance impact. As Figure 11 shows, the greater the write load of the workload, the greater the impact. For the WebSearch benchmark (which is nearly all reads), adding a few dummy writes has no visible impact on performance, whereas the impact on write-heavy Postmark and TPC-C are more noticeable. Fortunately, even

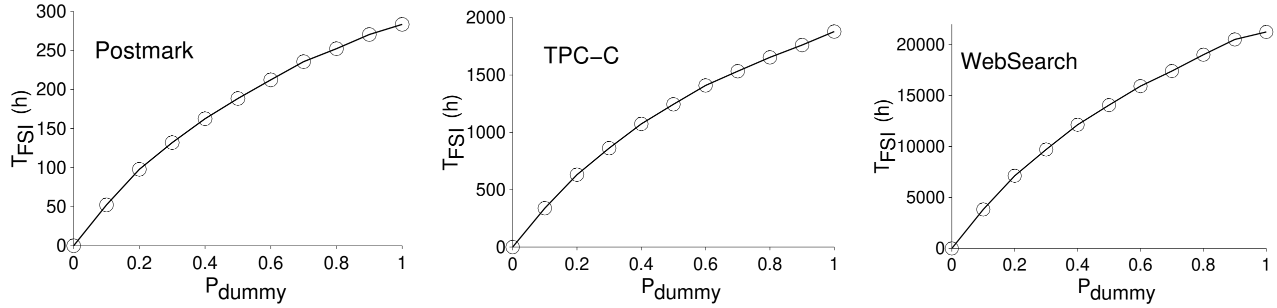

Fig. 10. T_{FSI} for the Postmark, TPC-C, and WebSearch traces with different dummy write frequencies.

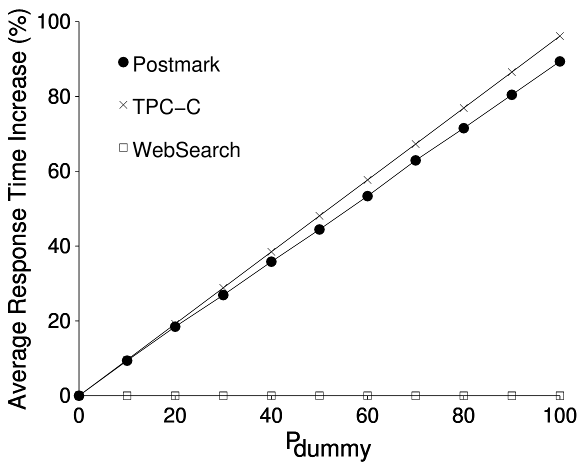

Fig. 11. Average response time (ms) for traced workloads with different dummy write frequencies.

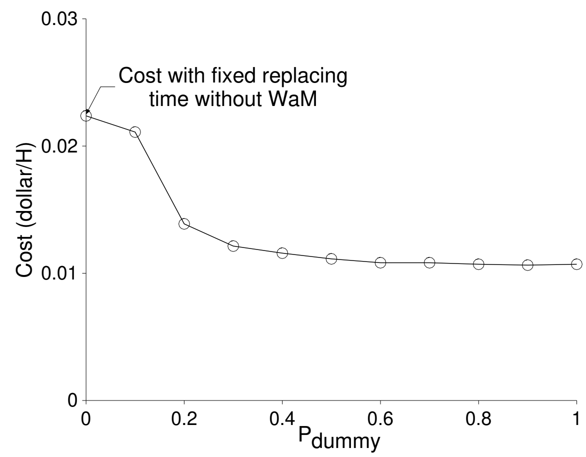

Fig. 12. Hourly dollar cost with fixed replacing strategy and with WaM of different dummy write frequencies.

very small dummy write percentages (*e.g.*, $P_{dummy} \leq 0.1$) generate satisfactory failure-separation intervals, and thus the overall performance impact, even on these write-intensive workloads, is likely to be quite small.

7) Monetary Cost: Finally, we show total system cost of WaM and compare it with a system that schedules replacement at a fixed time. System cost of WaM includes the cost of SSDs and the cost for periodically checking system health. We estimate the latter cost by a typical hourly rate for system administrators ($20) divided by the length of FSI. Hourly SSD cost is calculated by the cost of an Intel 80GB MLC NAND SSD divided by its measured SSD lifetime in WaM or a fixed time (e.g., one year) in a system without WaM. We omit all other costs such as power and cooling, since they are the same across these two systems. We found that to use a Mirrored Intel SSD pair for one year, it costs $198 if we use the fixed-replacing-time scheme and $122 if we use WaM (with 20% dummy writes). Figure 12 shows the hourly dollar costs of the fixed-replacing-time scheme and WaM with different P_{dummy}. We can see that the system that retires SSDs at fixed time always has a higher cost than WaM. Even though WaM pays an additional cost for maintenance, its total system cost is still noticeably lower. We can also see that increasing P_{dummy} or increasing FSI has the effect of reducing system cost, as a higher FSI results in lower maintenance cost.

VI. Related Work

In recent years, flash-based devices have been proposed to be used as whole-sale disk replacements in data-center environments. For example, flash-based systems like Gordon [9] and FAWN [5] have been introduced to provide a good balance of performance, power, and monetary cost. Gordon and FAWN both try to reduce the mismatch between CPU and storage performance in existing systems. Each node in the Gordon system contains a CPU, a set of flash chips, and DRAMs, with a special FTL designed for data-intensive applications. Wear-leveling is used to extend lifetime of Gordon nodes. However, correlated failures among nodes due to wear-out still exist in Gordon. FAWN uses a log-structure to store data of key-value pairs on flash devices and consistent hashing to distribute loads across nodes. Redundancy is also used in FAWN, which again could suffer from the correlated wear-out problem.

Soundararajan *et al.* proposed the use of hard drives as a write cache for SSDs in order to extend SSD lifetime and take advantage of the fast random-read speed of flash [12]. However, using hard drives or other storage devices as a cache for mirroring SSDs does not solve the problem of correlated

SSD failure, as long as writes issued to the underlying mirror pair are identical.

Keeping spare disks reduces recovery time and thus the length of FSI. However, when kept idle, these disks do not contribute to overall system operation. Distributed sparing was proposed to make use of spare disks [10]. By distributing spare capacity across all disks in a disk array, all disks contribute to system operation. This notion can be use to arrays of SSDs as well. By keeping sparing capacity, reconstruction can be performed once a failure is detected. However, we still need a FSI that is larger than reconstruction time and time to notice a failure.

Most relevant to our work is the work of Kadav *et al.*, who investigated a similar problem of using SSDs on parity-based RAID [17]. Since RAID-5 distributes parity and data blocks evenly, all SSDs in a RAID-5 collection wear out at the same rate. They propose Diff-RAID which assigns parity blocks unevenly across devices. The drive having the most parity blocks will die first. After each replacement, Diff-RAID reshuffles parity so that the oldest array has the maximum parity and dies first. Similar technique has been proposed by Mir *et al.* [18]. These techniques will not work for flash-based mirroring, since there is no parity and the amount written to the devices of a mirroring system is exactly the same. Warped Mirrors present a different approach to early failure induction based on load rather than layout; it would be interesting to compare both techniques more directly, something we leave for future work.

VII. Conclusions and Future Work

In this paper, we address the challenge of the reliability problem of mirrored flash. Warped Mirrors (WaM) present one solution to this problem. By carefully adding a minimal dummy write load to one flash in each mirror pair, WaM induces its early failure, thus providing a failure-separation interval (FSI) in which the device may be restored before the other fails.

We develop a model of flash wear out and use it to adjust our WaM load imbalance algorithms. Through simulation, we show that our approach is good at delivering the desired FSI and usually does so with little performance overhead. Our approach is also low at monetary cost, getting the most out of each flash before having to replace it.

In the future, it would be interesting to generalize our load-based approach to other RAID schemes. We are also interested in seeing how our technique works in real deployments and not just simulation. In both cases, further experience is required to gain a better understanding of the true nature of flash failures and thus whether these techniques can be applied in practice.

Acknowledgment

We thank the anonymous reviewers for their tremendous feedback and comments, which have substantially improved the content and presentation of this paper. We also thank the members of the ADSL research group for their insightful comments.

This material is based upon work supported by the National Science Foundation under the following grants: CCF-0811657 and CNS-0834392, as well as by generous donations from Google, NetApp, and Samsung.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other institutions.

References

[1] Intel X25-M 80 GB Mainstream SATA II MLC 2.5-Inch Solid State Drive OEM. htpp://www.amazon.com/gp/product/B003F8KT1O.

[2] A. Modelli, A. Visconti, and R. Bez. Advanced flash memory reliability. In *Proceedings of the IEEE International Conference on Integrated Circuit Design and Technology (ICICDT '04)*, Austin, Texas, May 2004.

[3] N. Agarwal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proceedings of the USENIX Annual Technical Conference (USENIX '08)*, Boston, Massachusetts, June 2008.

[4] Alcatel-Lucent. Lucent, Imation Developing Bell Labs Holographic Storage Technology. http://www.bell-labs.com/news/1999/august/11/1.html, 1999.

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

[6] D. Bitton and J. Gray. Disk shadowing. In *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB 14)*, Los Angeles, California, August 1988.

[7] J. S. Bucy and G. R. Ganger. The DiskSim Simulation Environment Version 3.0 Reference Manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, January 2003.

[8] Cade Metz. Flash Drives Replace Disks at Amazon, Facebook, Dropbox. htpp://www.wired.com/wiredenterprise/2012/06/flash-data-centers/all/, 2012.

[9] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV)*, Washington, DC, March 2009.

[10] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, New York, October 2003.

[12] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Griffin: Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST '10)*, San Jose, California, February 2010.

[13] 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 *Proceedings of MICRO-42*, New York, New York, December 2009.

[14] 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 (ASPLOS XIV)*, Washington, DC, March 2009.

[15] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson. Coding Techniques for Handling Failures in Large Disk Arrays. *Algorithmica*, 12(2):182–208, August 1994.

[16] Jeff Barr. New High I/O EC2 Instance Type - hi1.4xlarge - 2 TB of SSD-Backed Storage. http://aws.typepad.com/aws/2012/07/new-high-io-ec2-instance-type-hi14xlarge.html, 2012.

[17] Mahesh Balakrishnan, Asim Kadav, Vijayan Prabhakaran, and Dahlia Malkhi. Differential RAID: Rethinking RAID for SSD Reliability. In *Proceedings of the EuroSys Conference (EuroSys '10)*, Paris, France, April 2010.

[18] I. F. Mir and A. A. McEwan. A Fast Age Distribution Convergence Mechanism in an SSD Array for Highly Reliable Flash-based Storage Systems. In *Proceedings of the 3rd International Conference on Communication Software and Networks (ICCSN '11)*, Xi'an, China, May 2011.

[19] A. Park and K. Balasubramanian. Providing fault tolerance in parallel secondary storage systems. Technical Report CS-TR-057-86, Department of Computer Science, Princeton University, November 1986.

[20] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD Conference on the Management of Data (SIGMOD '88)*, Chicago, Illinois, June 1988.

[21] 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)*, Brighton, United Kingdom, October 2005.

[22] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992.

[23] R. H. Saavedra and A. J. Smith. Measuring Cache and TLB Performance and Their Effect on Benchmark Runtimes. *IEEE Transactions on Computers*, 44(10):1223–1235, 1995.

[24] K. Salem and H. Garcia-Molina. Disk Striping. In *Proceedings of the 2nd International Conference on Data Engineering (ICDE '86)*, Los Angeles, California, February 1986.

[25] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST '07)*, San Jose, California, February 2007.

[26] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID. In *Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST '04)*, San Francisco, California, April 2004.

[27] University of Massachusetts. Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage, 2009.

[28] Western Digital. NAND Evolution and its Effects on Solid State Drive (SSD) Useable Life. http://www.wdc.com/WDProducts/SSD/whitepapers/en/NAND_Evolution_0812.pdf, 2009.

[29] Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. De-indirection for flash-based ssds with nameless writes. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies (FAST '12)*, San Jose, California, February 2012.