Age Distribution Convergence Mechanisms for Flash Based File Systems

Alistair A. McEwan and Irfan F. Mir
Department of Engineering, University of Leicester, Leicester, LE1 7RH
Email: {aam19, im105} @le.ac.uk

***Abstract*—Solid state devices are common choices for data and systems storage in many high assurance application domains due to features such as no moving parts, shock/temperature resistance and low power consumption. On the other hand, they present new challenges in data reliability concerns. In multilevel cell NAND flash memories the bit error rate increases exponentially with reduced endurance limit as compared to single-level cell NAND flash memories. This can significantly reduce the data reliability and integrity of flash based storage systems. One solution to this is Redundant Array of Independent Disk (RAID) mechanisms. However these have an inherent problem as all flash memory chips wear out at the same time due to equal distribution of write operations. Recent solutions such as Diff-RAID partly solve this problem using uneven parity distribution mechanism, but suffer age-variation problems thereby decreasing the reliability of the array as well as increasing the cost.**

In this paper we present a fast age distribution convergence mechanism and page write control mechanism for a solid state device array. This mechanism solves the age convergence problem and uses fewer replacement devices. In the case of pure random write distribution the mechanism also saves page writes, thereby increasing the lifespan of each element in the array.

Keywords–SSD array, NAND flash, data reliability, high-assurance storage, RAID, bit error rates.

I. INTRODUCTION

Solid State Disk (SSD) architectures generally consist of a number of NAND flash memory chips [1]. Flash based SSD storage systems are increasingly used to replace traditional magnetic hard disk drives because of their increasing high performance, large amount of storage, and reliability [2]. However, the way that individual flash memory chips are managed internally to an SSD directly affects the reliability of a device. For instance, it is shown in [3] that sometimes even distribution of write operations across these flash memory chips can damage the reliability of SSDs.

The life of a flash memory chip depends upon the erase/write operations which, unlike magnetic disks, physically wear out the device. Life can be extended by using efficient wear-leveling algorithms [4]–[7] and Error Correction Codes (ECC). Recent developments, such as Multi Level Cell (MLC) and Triple Level Cell (TLC) devices [8] increase the available storage in a flash device, but at a cost of lifespan and reliability. However, reliability is a crucial consideration where a flash memory device is to be used in a high-integrity environment. The write endurance of a flash memory chip increases Bit Error Rates (BER) [9] as exampled in Figure 1 which is plotted using data taken from [10]. It is therefore important that we consider how flash memory based SSD storage systems can be built and deployed in high assurance environments in a reliable and cost-effective manner.

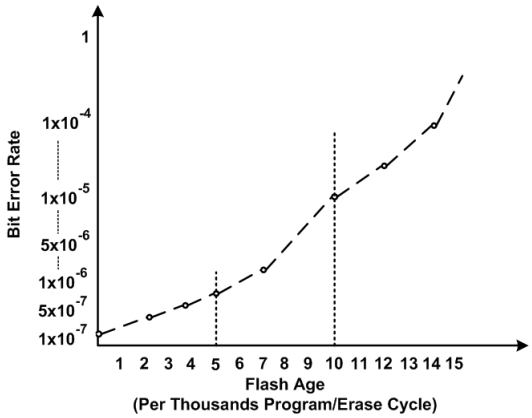

Fig. 1. Sample erasure cycles and bit error rate for a D-MLC32-1

Error Correction Code (EEC) is a technique used to check the bit-wise validity of data being read from a flash memory chip, and is generally implemented in hardware in the flash controller. However EEC does not protect a system against whole page, chip, or system failure. To achieve this, more fault-tolerance mechanisms are needed. [11] proposes data redundancy within a flash memory chip as a solution to provide additional reliability in flash based storage systems. Redundant Array of Independent Disk (RAID) systems [12] are commonly used in magnetic disk systems to enhance reliability in the case of individual component failure. However RAID techniques are not directly applicable to flash based SSD storage systems as they cause all the components in the flash system to reach their erasure (unreliability) limits at the same rate. Modifications of RAID techniques have been proposed for building reliable NAND flash based storage systems. In this context RAID-4 and RAID-5 are mostly exploited because of their parity-based data redundancy mechanisms.

A reliability enhancing mechanism for a large flash embedded satellite storage system is proposed in [13] based on a conventional RAID-5 mechanism—a multi-cluster RAID-5 architecture offering 120GB storage using

doi:10.4304/jcp.7.4.988-997

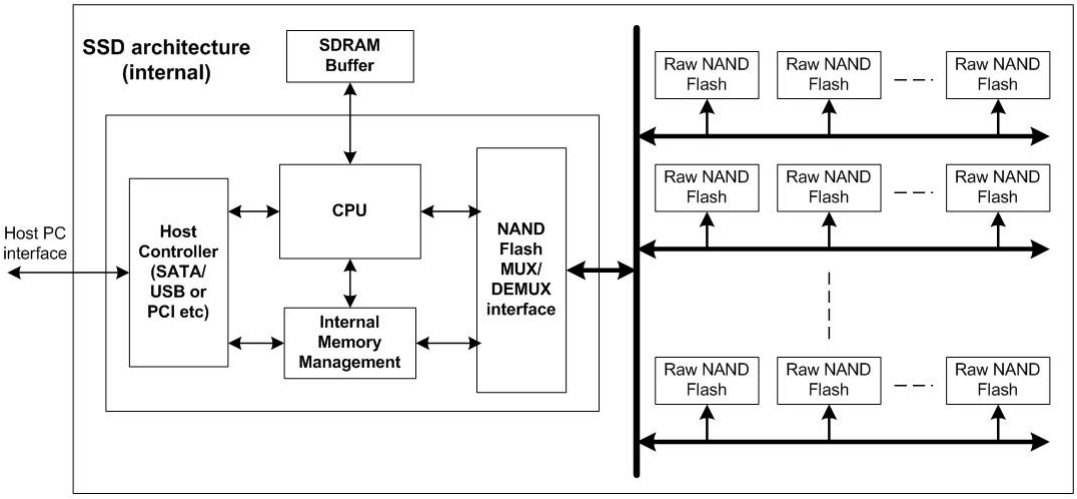

Fig. 2. A generic Solid State Disk architecture

2GB MLC NAND flash chips is presented. [14] presents a heterogeneous SSD RAID-4 system which eliminates the parity bottleneck by introducing a conventional hard disk for parity storage, and additionally introduces a wear leveling scheme. [11] presents a high-level architecture for reliable NAND flash which manages erasure-coded stripes at different granularities within a flash chip. Here a single host-based Flash Translation Layer (FTL) is used instead of individual FTL for each flash device. It is based on a conventional RAID-4 mechanism which can be exploited at page, block and bank levels across flash arrays. This approach gives enormous data protection in the case of whole-page and chip-level failures. [3] introduces Differential RAID (Diff-RAID), which achieves device-level redundancy by maintaining an age differential among flash devices. This ensures all devices have different bit error rates and avoids the situation where all devices wear out simultaneously by distributing parity unevenly. The result is a more reliable system than conventional RAID-4 or RAID-5 architectures.

The contribution of this paper is to present enhancements to Flash Diff-RAID architectures that firstly enhance reliability, secondly reduce cost by using fewer Flash devices, and thirdly extend the lifespan of each device using an enhanced page control mechanism.

The rest of this paper is structured as follows. In Section II we present the details of NAND-based storage systems. Section III highlights the reliability concerns in Flash based RAID systems. Our age distribution convergence mechanism and page control mechanism are presented in Section IV. We evaluate our mechanisms and demonstrate the enhanced reliability in Section V. Finally we draw conclusions in Section VI.

II. NAND FLASH BASED STORAGE SYSTEMS

Figure 2 shows the common architecture of SSD devices for massive data storage. A commodity SSD consists of a number of NAND flash chips, a controller, a multiplexor, a demultiplexor, DRAM, a processor, and a host interface such as SATA or PCI. The flash controller provides the low level signal access to each flash chip and the processor manages the traffic between the host computer and flash controller.

There are five aspects that need to be taken into consideration when building NAND flash SSD storage systems. These are: capacity, performance, reliability, endurance, and cost. Multi Level Cell flash memories typically have at least double the capacity with lower cost per memory cell than Single Level Cell memories; and this therefore increases their popularity in massive data storage systems. However MLC flash memories have lower performance and reliability–typically 10 times reduced endurance limit compared to SLC flash memories [10], [15], [16]. This is an important factor where high reliability demands are to be placed on the storage system.

FlexFS is a MLC flash based file system, presented in [15]. It can achieve comparable performance to SLC flash memories where it determines the requirement for data storage in defined MLC and SLC regions within MLC flash memories. The utilization of multiple memory banks in parallel is widely explored in flash based storage systems [13], [17], [18] to boost their IO performance.

High bit error rates in MLC flash memories can significantly reduce the overall reliability of MLC-based storage systems. To overcome this, high level Error Correction Codes are needed, and this is becoming increasingly important in the domain of high reliability storage systems [16], [19]. Conventional RAID configurations can also be exploited in MLC based storage systems [3], [11], [13], [14].

III. RAID MECHANISM FOR FLASH

RAID mechanisms have been successfully employed in magnetic disks for many years. As described in Section I there have been efforts to exploit these ideas in flash based SSD storage systems, however the ideas do not transpose over into this domain directly. In this section we highlight

some of the issues in exploiting RAID architectures in flash based SSD storage systems.

A. Limitations of RAID in SSDs

RAID mechanisms work by distributing the data to be stored (for instance a file) across a number of devices. A parity check is performed on the data to be written, and parity values are stored on an extra device. When one hardware component fails, the missing data–either a section of the file, or the parity data–can always be recalculated and recovered. The parity check mechanism assumes that only one component (one hardware device) needs recovered at any one time. Typically RAID does not work in the case where more than one device fails simultaneously. RAID mechanisms typically distribute writes and parity checks evenly across the storage mediums being employed. However, in the case of flash based SSD systems, this means that each of the devices will wear out, resulting in increased bit error rates, at the same rate [9].

In flash based RAID architectures, this means a situation can arise where data cannot be recovered. If the devices all wear out evenly, then the data stored on more than one device may be corrupted due to the high bit error rate. Consequently, there will be no data redundancy, and the corrupted data cannot be recovered. In the case of traditional disk based RAID mechanisms, the risk of two devices failing simultaneously is a risk that can be calculated given the assurances needed of the system. In the case of flash based systems it is highly likely due to the inherent wear properties of flash memory and the even distribution (and therefore lifespan) over the devices.

The technique adopted in Diff-RAID [3] relies upon two mechanisms. The first is uneven parity distribution, and the second is parity redistribution at the time of device replacement when a flash component wears out. However there are limitations to these techniques. Firstly the redistribution of parity amongst devices, especially during the replacement of the first few devices, can cause an increase in the bit error rate of the consecutive older devices. Secondly, in the case of full-stripe write operations it only provides the same reliability as RAID-5, and therefore relies upon random writes[1] in order to maintain the age differential. A further drawback of this technique is that the system itself is not immediately stable in terms of parity distribution. This means that several flash devices need to be consumed and replaced before the uneven parity distribution reaches a stable (optimal) state. this is a concern in high reliability applications where either cost, or physical space, is a concern.

IV. Reliability Enhancement Mechanisms

We use the term *age convergence* to refer to the mechanism where each flash device in our system is converging towards a state where we never have more than one device reaching an age where it can become potentially unreliable. We assume that when a given device reaches this unreliable age, it will be replaced with a new device. In this way, the constraints of RAID—that more than one device cannot be allowed to fail—are never violated. By *least aged* we mean the SSD that has had the fewest write operations and therefore the lowest BER. This corresponds to the device lowest down the curve in Figure 3. Correspondingly, by *most aged* we mean the device that has had the most write operations. This is displayed graphically in Figure 3 where we superimpose sample SSDs on our earlier graph in example positions based on the number of writes to each device. The most aged device is SSD6 as this is the device furthest up the curve and as such is the most unreliable in terms of BER.

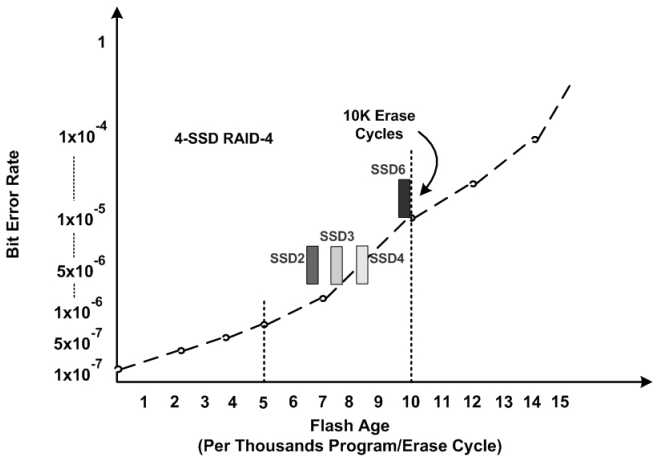

Fig. 3. Bit error rate versus write/erase cycles

In this section, we present a new fast age convergence mechanism and a page write control mechanism applicable to RAID based SSD arrays that enhances the overall reliability of a flash based SSD storage system. Our fast age convergence mechanism overcomes the problem of uneven—and potentially unreliable—age distribution in initial device replacements. The page write control mechanism increases the lifespan of an individual SSD by saving page writes during partial updates of full stripe data in the array.

A. Age distribution convergence

In our mechanism, parity is distributed unevenly in such a way that in a RAID array of n SSD devices, only 3 will hold parity blocks during the lifespan of the first $n-3$ device replacements. After $n-3$ device replacements the most aged device will hold the maximum parity blocks and rest will hold evenly distributed parity blocks.

Age distribution convergence can be achieved instantly by saving flash writes on the least aged SSD after the first (most aged) device replacement. For this purpose a spare SSD is swapped (and copied) in place of the least aged SSD at the time of parity redistribution during the first replacement phase; and remains in use until the second replacement phase. At the time of this replacement, the spare SSD is swapped and copied back to the previously

[1]The concept of *full stripe writes* and *random writes* is explained in Section IV-C.

least aged SSD (which will be used again in the array during subsequent replacement phases until it reaches its wear-out level). In this way the age differential among the devices is stable immediately after first device replacement to minimize the chance of unreliable data recovery or rising error rates in more than one device.

Definition 4.1: The SSD array

$$SSDArray : \mathbb{N} \pfun (Data \times \mathbb{R})$$
$$\#SSDArray >= 4$$

In our description of the mechanism, *SSDArray* represents the array of flash devices in our SSD. It is a function which maps natural numbers (each device index) onto some arbitrary set (the data contained on an individual device and a parity limit for that device represented by a real number. Notionally, this is similar in specification terms to an array in implementation terms: if one were to look at, for instance $SSDArray(1)$, one would have returned the data and parity limit stored on device 1 (we assume that the actual parity data itself is included in the data). The function is partial as not all numbers may physically represent devices in the system—although there must be at least 4 devices. The length of the array—i.e. the number of devices in the system—is given by $\#SSDArray$, which is the cardinality of the function.

Definition 4.2: Identifying individual SSD devices

$$SSD_i : \mathbb{N} \mapsto (Data \times \mathbb{R})$$
$$SSD_{sp} : \mathbb{N} \mapsto (Data \times \mathbb{R})$$
$$SSD_i \in SSDArray$$
$$SSD_{sp} \notin SSDArray$$

The variables SSD_i and SSD_{sp} are used to index individual SSD devices in our mechanism. SSD_i represents a particular device (index) in the array and SSD_{sp} represents the spare SSD device used for copying and temporary storage. Note that SSD_{sp} is not a constant, as the spare device is regularly swapped in and out for a specific device. As such, our system has an invariant that states SSD_{sp} is never in the array, i.e. $SSD_{sp} \notin SSDArray$.

Definition 4.3: Variables used for parity calculation

$$B : \mathbb{Z}$$
$$x : \mathbb{R}$$
$$y : \mathbb{R}$$
$$PCC : \mathbb{N}$$
$$B \leq (\#SSDArray * (\#SSDArray - 1))$$
$$1 > x \geq 0.5$$
$$x + y = 1$$

The variables B, x, and y are used to represent the amount of parity that we wish to store on the first 3 devices in the array. B is used to calculate the minimum amount of parity that must be stored on the first device in order to allow convergence to emerge; whilst x and y are used to control the split of the remaining parity on the other 2 devices in the early replacement phases. The value of y must be no greater than x as this ensures that more parity will be stored on the next most aged device in the system. These values can be thought of as *user configurable* within these constraints, but are fixed for the lifetime of a given system. The variable *PCC* is used to count the number of parity redistributions that have been performed as the behaviour of the mechanism is dependent upon this information.

Definition 4.4: Setting parity configuration

$$SetParityLimits == \mathbb{N} \times SSDArray \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow SSDArray \times \mathbb{N}$$

The the function *SetParityLimits* sets the limit of the parity that can be stored on each device. The function is called in each device replacement phase, and moves the limits of parity stored on each device further down the array—so, for instance it moves the parity limit for device 3 to device 4, and for device 4 to device 5, and so on.The ratio of parity distribution is set at configuration time using the values x, y, and B, which are parameters to this function along with the array itself and the current value of *PCC*. The function returns the amended array along with an incremented *PCC* counter. Note that we do not physically move the parity data—only the limit permitted on each device. The parity data itself *migrates* as the file system is being used.

Definition 4.5: Setting up a replacement SSD

$$SetupReplacement == SSDArray \times (\mathbb{N} \pfun Data \times \mathbb{R}) \rightarrow SSDArray$$

The function *SetupReplacement* takes a new SSD device and our SSD Array as parameters, and returns a new array that has the oldest (lowest indexed) device dropped off, and a new device (the highest indexed) added on—with the data from the oldest device copied over. In physical terms, this is equivalent to unplugging and throwing away the most worn out device, plugging in a new device in its place, and using the parity calculation to work out the data that was on the oldest device that must be stored on the new device.

Definition 4.6: Copying an SSD

$$Swap == (\mathbb{N} \mapsto Data \times \mathbb{R}) \times (\mathbb{N} \mapsto Data \times \mathbb{R}) \rightarrow (\mathbb{N} \mapsto Data \times \mathbb{R}) \times (\mathbb{N} \mapsto Data \times \mathbb{R})$$

The function *Swap* takes two devices as parameters, and copies the data on the first device onto the second device. As above, although we have only provided a specification

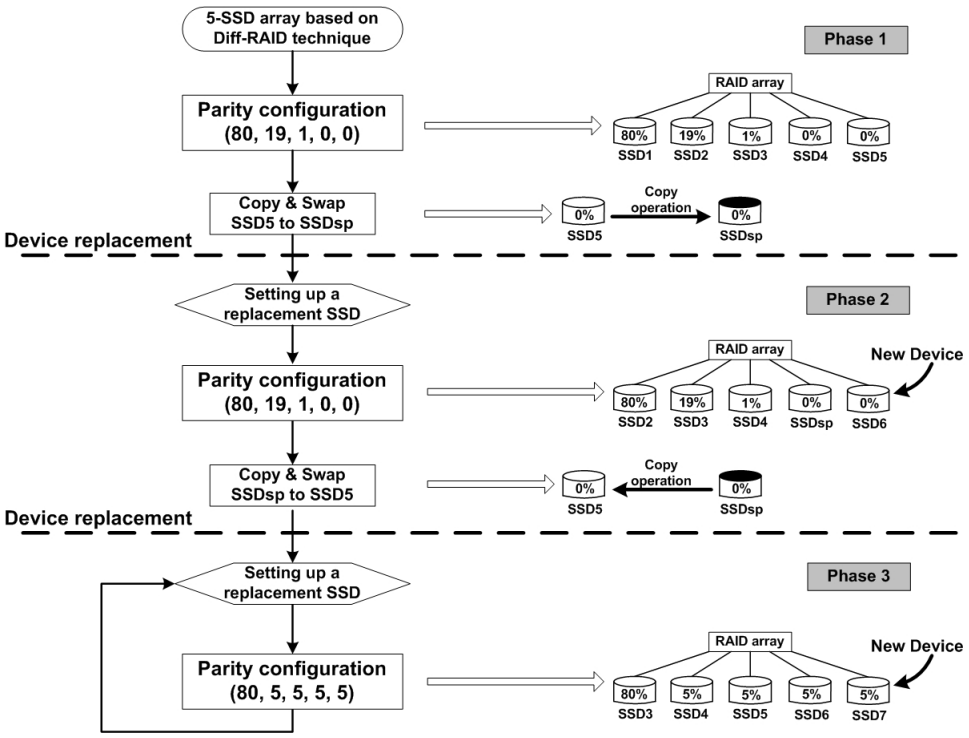

Fig. 4. Graphical example of fast age distribution convergence mechanism

of this function in this paper, in our implementation this function is responsible for physically copying data from one device to another.

Our mechanism to achieve this is given below. Initially the first, second and third devices are allocated parity of $100 - B$, xB, and yB respectively for the first $(\#SSDArray) - 3$ device replacements.

A key feature of our mechanism is that it saves writes on the least aged device by copying the data stored on this device into a spare SSD during the first device replacement process. At the point of the second device replacement the data on this spare device is copied back onto the least aged device in the array. This act of swapping a spare device in and out of the array to save writes continues until it reaches a critical bit error rate.

Mechanism 4.1: Fast age distribution convergence

while (1) *do*

if $(replacement\ needed)$

$Swap(SSD_n, SSD_{sp}) \lhd (PCC < 3) \rhd Skip$

$SetupReplacement(SSDArray, newdevice)$

$SetParityLimits(PCC, SSDArray)$

end if

end while

B. Case study using a 5 device RAID array

The phases of evolution of a 5 SSD system employing this mechanism is exampled graphically in Figure 4. Initially, in phase 1 of this example parity is distributed in such a way that the first 3 devices(SSD_1, SSD_2, SSD_3) hold 80%, 19%, and 1% of the parity respectively as values for B, x, and y are set at 20, 0.95 and 0.05. The spare device, SSD_{sp} is not used in the array at this point.

As the file system is used SSD_1 will be the first device to wear out as it holds the most parity (80% of writes to the file system will involve updating this parity). When it reaches its erasure limit and needs to be replaced, the the least aged device in the array (typically SSD_n, where n is the sizer of the array—in this case SSD_5) is copied to the spare device SSD_{sp}. These devices are now notionally swapped—that is the spare device is now an element of the array in position 5, and SSD_5 becomes SSD_{sp}. The most aged device SSD_1 is removed from the system and replaced with a brand new device that takes its place. The data that should be stored on this device is calculated and written using standard RAID techniques.

Parity limits are then redistributed around the system. SSD_2 is now the most aged device, so parity limits are redistributed across devices SSD_2,SSD_3, and SSD_4 as 80%, 19%, and 1% respectively, and no other devices hold parity. The actual parity bits migrate onto SSD_2 as the file system is used (rather than moving parity data as an action in the device replacement process); taking into account potential collisions with data and parity.

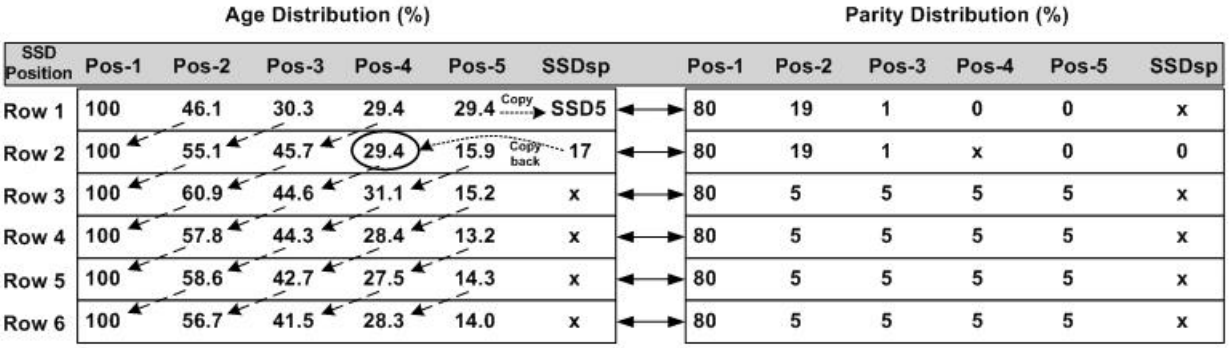

Fig. 5. Implication of our mechanism in 5-SSD Diff-RAID

When SSD_2 reaches its erasure limit, the same process is followed.

In the final phase, the ages of devices in our system have reached a stable state and do not fluctuate. When devices age and need replacement, parity is distributed evenly across the 4 least aged devices in order to maintain this stability. In the case of this example, the most aged device must hold a minimum of 80% parity to achieve this, so our distribution is set at 80%, 5%, 5%, 5%, 5%.

Definition 4.7: SSD Age Ratio

$$Age_{ij} \widehat{=} \frac{P_i*(n-1)+(100-P_i)}{P_j*(n-1)+(100-P_j)}$$

We evaluate the age distribution mechanism using the same technique as that in [3]. The age ratio Age_{ij} of i_{th} device with respect to device j is calculated using the formula in Definition 4.7, where P_i and P_j represents the parity percentage of devices i and j respectively in an array of size n, and non parity (data) writes are assumed equal in a random write scenario.

The implication of this mechanism in a 5 SSD Diff-RAID, adopting an initial parity assignment of (80, 19, 1) and a stable assignment of (80, 5, 5, 5, 5) is exampled in Figure 5. The mechanism is applied in each row among 5-SSD array with a spare SSD using differing parity assignments, and stable ageing rate is achieved after the first device replacement.

The cross arrows in Figure 5 show the device replacement process at the time of each replacement. In rows 1 and 2, the parity is distributed (80, 19, 1, 0, 0) in the 5 SSD array, where device number 5 is a spare device. It should be noted that the SSD in position 1 must be copied to the SSD in position 5 as the first action in the replacement process. In row 2, SSD 4 is unused, as it is replaced by the SSD in position 5 as part of the replacement process, thus maintaining the age distribution rate. Furthermore, as part of the second replacement (the transition from row 2 to row 3) the spare SSD is copied back into SSD 5. Finally, in row 3 and onwards, parity (80, 5, 5, 5, 5) is assigned and a stable state achieved.

C. Page writes control mechanism

In general parity-based RAID systems have poor write performance compared to read performance. In the case of an n-device RAID-5, at least, 2 write and n- 2 read operations are required for updating data. RAID-5 is a block-level striping architecture where parity is distributed across the devices evenly.

A *full stripe write* is a write operation where data is written across all devices in the array, as in Figure 6. The *full stripe* consists of a *stripe element* on each device. Each stripe element is made up of a number of *blocks*, and each block is made up of a number of *pages*.

Maximum performance is typically achieved when the file system sector size (I/O block) is a strict multiple of the strip size (i.e, sector size mod stripe size $= 0$).

For instance, in the case where we have a 5 element array, we may choose to allocate a stripe size of 48k. This may break down as 4 stripe elements of size 12k, each one located on a seperate device, with a further 12k being allocated on the remaining device for parity. Ideally, sector size would then be an integer multiple of 48k. The choice of stripe size also is affected by the size of pages and blocks, and strict multiples are beneficial here also.

When data is being updated in a given stripe, it is necessary to read all the data stored in that stripe in order to calculate and write the new parity. In a full stripe write there is no need to first read this data as the entire stripe is to be written anyway. However if we wish to efficiently only write to the pages that need updating, then it is necessary first to read all data and calculate the change in parity (which also must be re-written).

However there is a further refinement that may be observed. When we are updating a single stripe element (commonly known as *a random write*), it may be the case that we are only in fact updating a subset of the pages in that stripe element. We introduce the term *fractional random write* to describe the situation where a subset of the pages in a single stripe element need updating.

We present a simple mechanism below that enhances the lifespan of SSD devices in a RAID system by saving flash page writes in the case of fractional random writes, by only writing to the pages in a given stripe element that actually require updating.

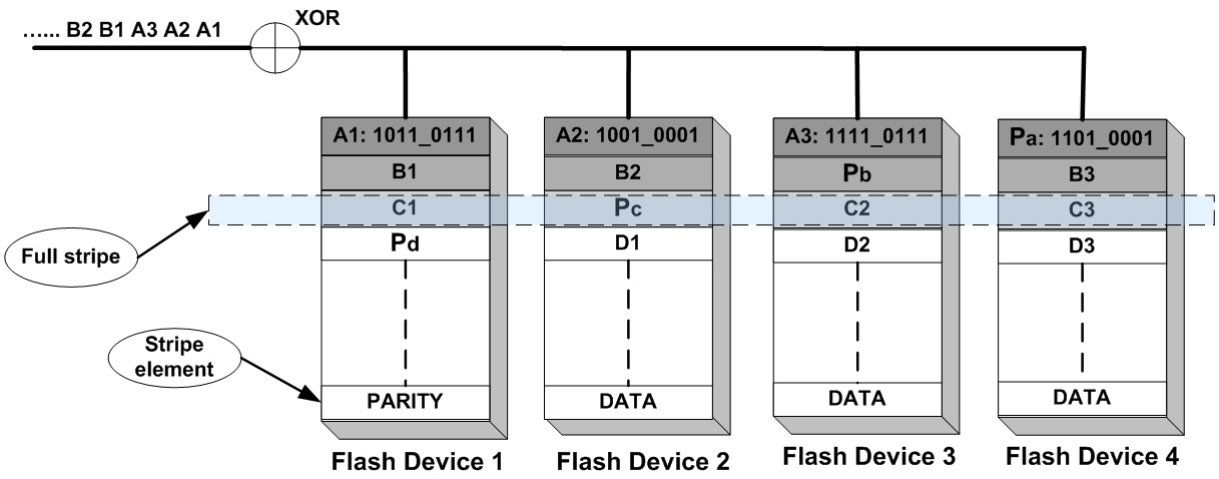

Fig. 6. Stripe writes across a RAID array

When we are in a data updating phase, the incoming data of a stripe element and its ECC per page are extracted from the incoming buffer block as are the related stored page data and corresponding ECC from the corresponding SSD. A comparison of the two sets of ECCs and the two data pages is performed. If the result of the comparisons is false (all zeros) then a page update is not required; otherwise it is; and this result is recorded.

At the end of this decision the page mapping table is updated, with only those pages that require updating actually written to the corresponding device.

Mechanism 4.2: Flash page writes control

while $PageNum < StripeSize - 1$ *do*

$(RxPageData, RxECC) :=$
$PageData(RxDataBuff, StripeSize, PageNum)$

$(StoredPageData, StoredECC) :=$
$PageData(SSD_n, StripeSize, PageNum)$

$UpdatePageMappingTable(SSD_n, PageNum)$

$UpdatePageNum[PageNum] :=$
$(\ Compare(0, RxECC, StoredECC) \vee$
$Compare(1, RxPageData, StoredPageData)\ )$

$PageNum := PageNum + 1$

end while

V. Reliability evaluation

We evaluate our mechanism using the same metrics as those in [3]. That is, we aim to demonstrate the reliability enhancement over the techniques in [3] in terms of age distributions and page writes. The results suggest that the mechanisms presented in this paper do indeed increase reliability whilst reducing cost.

A. Fast age convergence mechanism

Figure 7 compares the results of our mechanism against Diff-RAID [3]. The most significant difference shown in our results is that using Diff-RAID, in the early replacement phases there are often several devices above the 60% error rate threshold thereby risking the failure of the RAID mechanism as discussed earlier (see, for instance replacement phases 3 and 4). However our enhanced mechanism never has more than one device above this threshold. Figure 7 also demonstrates that higher reliability in terms of bit error rate of up to 5% is achieved during the initial device replacements. This bounds the average age of each SSD across the device, and ensures consistent and even usage of each SSD in terms of pure random writes.

Figure 8 shows the average ages of devices using the two techniques. It shows improved stability from our enhancements in that there is very little deviation from a safe average age of 50%, compared to Diff-RAID peaking at 60% average age, with the consequential increase in bit error rates.

Our mechanism can be applied to SSD arrays of larger sizes. Figure 9 shows results for arrays of length 6 and 7, and how these arrays quickly converge to a steady state. The results show that in the general case, age convergence to optimal levels happens approximately 7 times faster– leading to cheaper and more stable systems.

B. Page writes control mechanism

In evaluating the page writes, some assumptions are necessary. That is, we assume a pure random workload where 10% of the workload consists of evenly distributed random writes. We also assume that every fractional random write, at least, needs to update a maximum of half of the stripe element data in an n-SSD RAID.

The result is each SSD experiences the same number of fractional random writes across RAID-based storage systems. Our mechanism saves 5% data writes in our system.

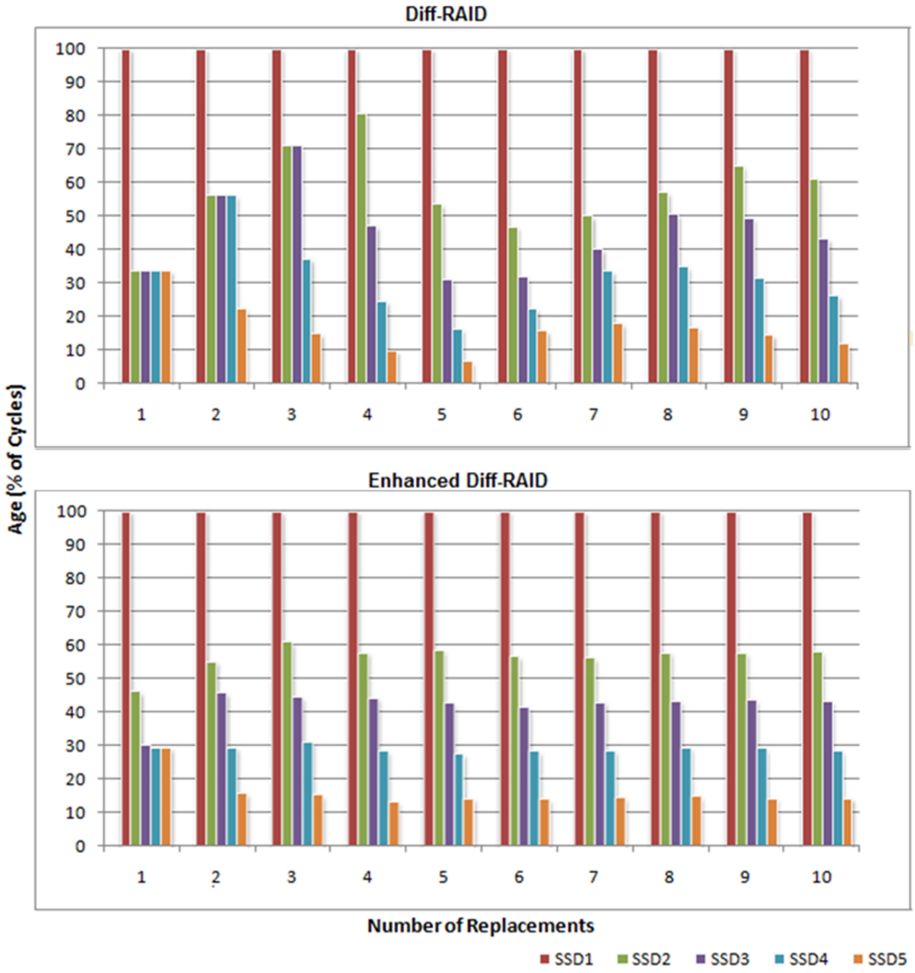

Fig. 7. Comparison of device ages at replacement phases of Enhanced Diff-RAID with Diff-RAID

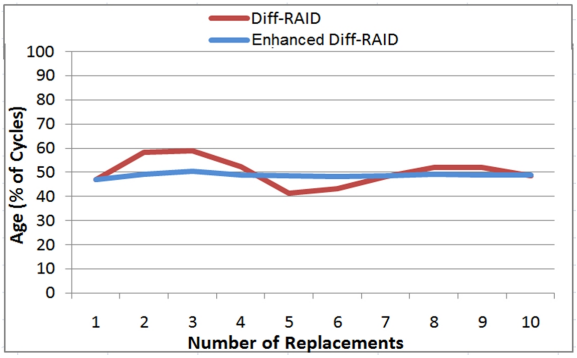

Fig. 8. Average age of 5-SSD RAID array during the first 10 device replacement processes

Our description of a page write control mechanism is relatively simple in that it essentially compares stored data with data to be stored. However a key point to note is that in our work we are producing a hardware implementation of the file translation layer into which this mechanism would go. As such, we can exploit concurrency and compare entire pages in a single operation—therefore the implementation of this mechanism is feasible in a way that would not be in other implementations. The cost of implementation is two registers, each of page size, plus an associated array of XOR comparators—this is approximately double the cost of the implementation of a system that does not employ this mechanism.

VI. Conclusion

In this paper we have presented two novel mechanisms that enhance the reliability of flash based storage systems. The fist mechanism was a fast age convergence mechanism for Diff-RAID that increased reliability during the early stages of the storage devices life. It achieved this by adopting different parity configurations across the SSD array. The second mechanism was a page write control mechanism that increased the lifespan of an SSD device. It achieved this by reducing the number of (unnecessary) writes to a flash chip during the fractional random writes based workload.

Currently these mechanisms are being implemented in a flash based SSD array which is being built. The controller in this array, and the processor, are soft-core FPGA-based. This allows us to investigate and implement new techniques in hardware. Our system is configurable in many ways, including, for instance stripe size. As part

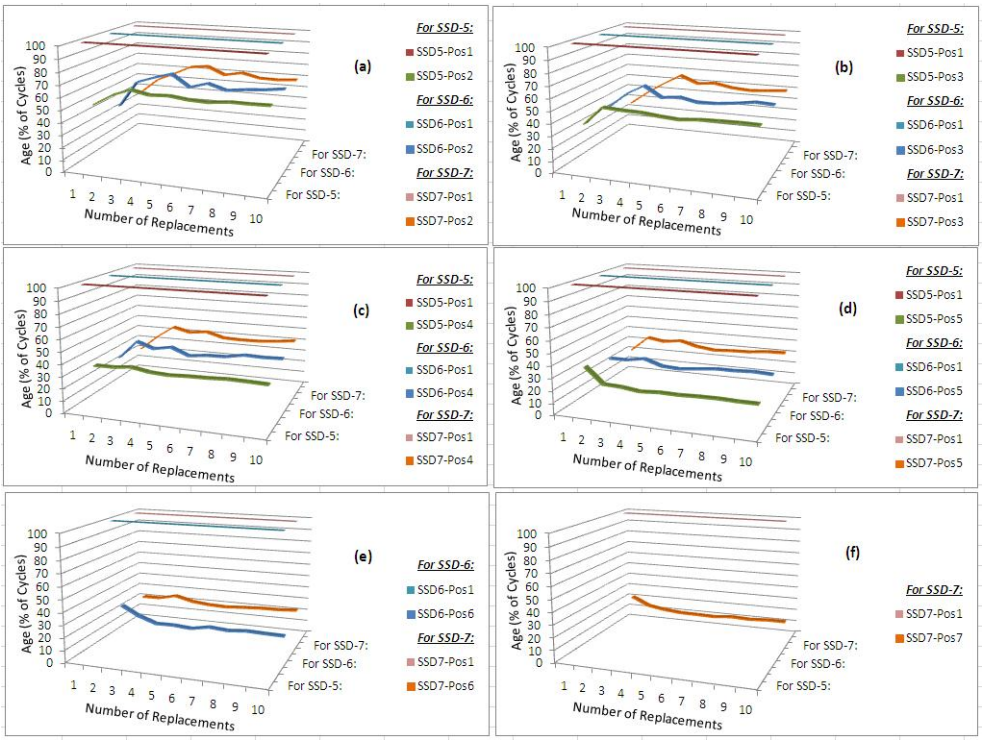

Fig. 9. Convergence extrapolation for longer arrays

of this process we are developing techniques that will allow us to exploit and increase the occurances of random write (and fractional random write) in general usage, and investigate real-time issues in our filesystem. Ultimately we aim to build a flash based SSD storage system that can be deployed in a number of high assurance applications under development.

REFERENCES

[1] J. D. Davis and L. L. Zhang, "FRP: A Nonvolatile Memory Research Platform Targeting NAND flash," in *Proceedings of the First Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, ser. WISH '09. ACM, 2009.

[2] J. Bowen, W. Hutsell, and N. Ekker, "RamSan-6300 14-Million IOPS Flash-based SSD System," Texas Memory Systems, Tech. Rep., November 2008, http://www.ramsan.com/products/73.

[3] M. Balakrishnan, A. Kadav, V. Prabhakaran, D. Malkhi, and Dahlia, "'Differential RAID: Rethinking RAID for SSD Reliability," *ACM Transactions on Storage*, vol. 4, no. 1, pp. 4–22, July 2010.

[4] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138–163, June 2005.

[5] I. Corporation, "Understanding the Flash Translation Layer (FTL) Specification," Intel Corporation, Application Note 684, December 1998.

[6] A. Ban, "Flash File System, US patent 5404485," April 1995.

[7] P.-L. Wu, Y.-H. Chang, and T.-W. Kuo, "A file-system-aware FTL design for flash-memory storage systems," in *Design, Automation and Test in EuropeConference and Exhibition*, ser. DATE '09, DATE '09. IEEE, 2009, pp. 393–398.

[8] Micron, "Intel, Micron First to Sample 3-Bit-Per-Cell NAND Flash Memory on Industry-Leading 25-Nanometer Silicon Process Technology," Press Release, August 2010, http://news.micron.com/releaseDetail.cfm?ReleaseID=499901.

[9] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill, "Bit Error Rate in NAND flash Memories," in *International Reliability Physics Symposium*. IEEE, April 2008, pp. 9–19, print ISBN: 978-1-4244-2049-0.

[10] 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 *42nd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, December 2009, pp. 24–33, print ISBN: 978-1-60558-798-1.

[11] K. M. Greenan, D. D. E. Long, L. Ethan, L. Miller, T. J. E. Schwarz, and S. J. A. Wildani, "Building Flexible, Fault-Tolerant Flash-based Storage Systems," in *The Fifth Workshop on Hot Topics in Dependability*. HotDep 2009, June 2009.

[12] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, H. Borla and P.-Å. Larson, Eds., vol. 17. ACM Press, June 1988, pp. 109–116.

[13] S. Zertal, "A Reliability Enhancing Mechanism for a Large Flash Embedded Satellite Storage System," in *Third International Conference on Systems*. IEEE, April 2008, pp. 345–250, print ISBN: 978-0-7695-3105-2.

[14] K. Park, D.-H. Lee, Y. Woo, G. Lee, J.-H. Lee, and D.-H. Kim, "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," in *9th International Symposium on Communications and Information Technology*, ser. ISCIT '09, September 2009, pp. 140–145, print ISBN: 978-1-4244-4521-9.

[15] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "FlexFS: A Flexible Flash File System for MLC NAND Flash Memory," in *Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2009, p. 9, http://dl.acm.org/citation.cfm?id=1855807.1855816.

[16] E. Deal, "Trends of NAND Flash Memory Error Correction," Cyclic Design, White Paper, June 2009, http://cyclicdesign.com/whitepapers/Cyclic_Design_NAND_ECC.pdf.

[17] J. Cooke, "The inconvenient truths about nand flash memory," Web presentation, 2007, http://download.micron.com/pdf/presentations/events/flash_mem_summit_jcooke_inconvenient_truths_nand.pdf.

[18] Y.-B. Chang and L.-P. Chang, "A self-balancing striping scheme for nand-flash storage systems," in *Proceedings of the 2008 ACM*

symposium on Applied computing, ser. SAC '08. ACM, 2008, pp. 1715–1719, iSBN: 978-1-59593-753-7.

[19] J. U. Kang, J. S. Kim, C. Park, H. Park, and J. Lee, "A Multi-Channel Architecture for High-Performance NAND Flash-based Storage System," *Journal of Systems Architecture*, vol. 53, no. 9, pp. 644–658, September 2007.

Alistair A. McEwan is a lecturer in Real-time systems and Software Engineering in the Embedded Systems Research Group at the University of Leicester, UK. He received his DPhil in Computer Science from the University of Oxford in 2006, and joined the Univeristy of Leicester in 2007.

He has worked on a number of projects involving thoeretical approaches to concurrent systems, systems specification and verification, modelling of safety-critical systems, and hardware/software co-design. Much of his current work is looking at the appication of formal Software Engineering techniques to the development, and safety-modelling of large systems involving software and bespoke hardware.

Irfan Mir completed his Masters degree in Electronics from Quaid-i-Azam University Islamabad, Pakistan in 2000. After completing his Masters he worked on reconfigurable technologies within local electronics industry and acquired 10 years practical experience on FPGA based system design and development. Currently he is a member of Embedded Systems Research Group (ESRG), University of Leicester as a PhD student. His research interests are flash based storage, flash file systems, FPGA based architectures and embedded systems.