DCS: Diagonal Coding Scheme for Enhancing the Endurance of SSD-Based RAID Arrays

Yubiao Pan, Yongkun Li, *Member, IEEE*, Yinlong Xu, and Biaobiao Shen

***Abstract*—To guarantee high reliability, solid-state drive (SSD)-based storage systems require data redundancy schemes, e.g., redundant array of independent disks (RAID) schemes. Traditional RAID-5, RAID-6, and Reed–Solomon codes can tolerate one, two, and an arbitrary number of device failures, respectively. However, some SSDs under those redundant configurations may age much faster than others because of the high skewness and locality of workloads. The uneven aging rates may make some SSDs wear out very quickly and decrease the endurance of SSD-based RAID arrays. To address this problem, we first come up with a diagonal coding scheme (DCS) by distributing the updating dependencies evenly among devices to improve the system-level wear-leveling. DCS can efficiently improve the array endurance if requests are aligned with the stripe size, i.e., when data symbols in the same stripe receive the same number of writes, while the number could be different for different stripes. To relax the above assumption, we further propose an enhanced scheme, DCS+. With a buffer design, DCS+ can improve the wear-leveling among devices under general access patterns via triggering different responses to different kinds of requests. We conduct extensive trace-driven evaluations based on real-world workloads, and results show that our design efficiently enhances the endurance of SSD-based RAID arrays.**

***Index Terms*—Endurance, redundant array of independent disks (RAID), solid-state drives (SSDs), system-level wear-leveling.**

I. Introduction

SOLID-STATE drives (SSDs) are being widely deployed in large-scale storage systems as they provide multiple improvements over hard disk drives (HDDs), such as higher input–output (I/O) performance, lower power consumption, and higher shock resistance.

However, SSDs also possess multiple constraints, one major concern is that each block in an SSD can only tolerate a limited number of erasures. The typical value of this number is 10k for multilevel cell (MLC) SSDs, and it may even drop to several thousands for triple-level cell (TLC) SSDs [7], [11]. Thus, single SSDs usually employ wear-leveling techniques to balance the erasures on blocks inside SSDs [4]–[6], [14]. On the other hand, bit errors are still common in SSDs, especially for MLC SSDs and TLC SSDs, and they could be caused by read disturb, write disturb, and even data retention [10], [11], [19]. In particular, bit error rate increases as the number of erasures performed on the SSD increases, and the increase may become sharp when SSDs are reaching their erasure limit. Even worse, as the density of flash cells increases so to enlarge the capacity of single SSDs, the endurance of SSDs continues decreasing, while the bit error rate further increases [11].

To enhance the endurance and reliability of SSDs, both wear-leveling techniques and error correction codes are developed, while they are only for single SSDs. In terms of SSD-based arrays, device-level redundancy schemes like redundant array of independent disks (RAID) [22] become necessary. In particular, RAID-5, RAID-6 (e.g., EVENODD [3], row-diagonal parity (RDP) [8], X-code [26]), and Reed–Solomon (RS) code [23] are deployed into RAID arrays for tolerating one, two, and even an arbitrary number of device failures.

We use RAID-5 as an example to illustrate the data organization of RAID arrays. RAID-6 and RS code have similar data organization as RAID-5, which will be shown in Section II. RAID-5 can not only improve the system reliability, but can also provide high I/O throughput as it achieves both load-balancing and I/O parallelism. A RAID-5 array is divided into many stripes, each of which consists of one parity symbol that is encoded from the data symbols in the same stripe. Symbols are striped across multiple devices in a round-robin manner. For the ease of presentation, we use C to represent a parity symbol, and use D to denote a data symbol in this paper. For example, in Fig. 1, data symbols 1, 2, 3, 4, 5, 6, 7, and 8 are located in $D_{0,0}$, $D_{0,1}$, $D_{0,2}$, $D_{0,3}$, $D_{1,0}$, $D_{1,1}$, $D_{1,2}$, and $D_{1,4}$, respectively. The corresponding two parity symbols are located in $C_{0,4}$ and $C_{1,3}$. Note that parity symbols receive more writes in an RAID array because updating a data symbol requires an update to the corresponding parity symbol. For instance, if $D_{0,0}$, $D_{0,1}$, $D_{0,2}$, and $D_{0,3}$ are updated once independently, then the parity symbol $C_{0,4}$ will be updated by four times.

Manuscript received June 8, 2015; revised August 30, 2015; accepted October 28, 2015. Date of publication November 30, 2015; date of current version July 15, 2016. The work of Y. Li was supported in part by the National Nature Science Foundation of China under Grant 61303048, and in part by the Anhui Provincial Natural Science Foundation under Grant 1508085SQF214. The work of Y. Xu was supported in part by the National Nature Science Foundation of China under Grant 61379038, and in part by the Huawei Innovation Research Program. An earlier conference version of the paper appeared in IEEE NAS 2014. In this journal version, we extended the diagonal coding scheme for general erasure codes, proposed a new buffer design for DCS+ to handle general access patterns, added more experiments to evaluate the input–output performance, and also studied the impact of request size, buffer size and the parameter S on the performance of our schemes. This paper was recommended by Associate Editor Y. Wang. *(Corresponding author: Yongkun Li.)*

Y. Pan and B. Shen are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: pyb@mail.ustc.edu.cn; ustcshen@mail.ustc.edu.cn).

Y. Li and Y. Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China, and also with the AnHui Province Key Laboratory of High Performance Computing, University of Science and Technology of China, Hefei 230026, China (e-mail: ykli@ustc.edu.cn; ylxu@ustc.edu.cn).

Digital Object Identifier 10.1109/TCAD.2015.2504333

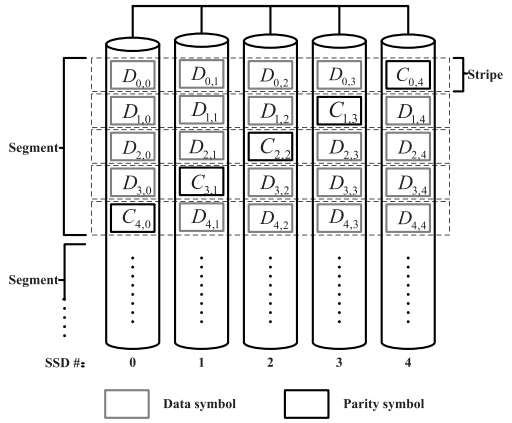


Fig. 1. Data organization in an RAID-5 system.

We call this dependent relationship as updating dependency. Here, we do not consider any optimization as the updating requests to the four data symbols may happen at different time instants. Even though those requests happen at the same time, they may be responded at different time points because the RAID array may receive concurrent requests from different users. Besides, some scenarios with a strong reliability requirement may also introduce this updating dependency.

If the accesses to data symbols are uniform, the number of writes to each device must be the same due to the even distribution of parity symbols. Unfortunately, real-world workloads are usually nonuniform, e.g., some data symbols are accessed much more frequently than others, which causes some SSDs to wear out much faster than others in SSD-based RAID arrays even if parity symbols are placed evenly across devices. To illustrate the unevenness of writes to SSDs under an RAID configuration, we still use Fig. 1 as an example. If data symbols $D_{0,0}$, $D_{0,1}$, $D_{0,2}$, and $D_{0,3}$ receive 50 update requests, while other data symbols receive ten updates only, then we can easily derive that SSD 4 receives 120 more update operations than other SSDs, and so SSD 4 wears out faster than others. From this example, we see that in an SSD-based RAID-5 array, SSDs may wear out at different rates under nonuniform workloads, which quickly triggers recovery and replacement operations due to the worn-out SSD. The updating dependency between data and parity symbols also makes SSDs reach different aging rates in RAID-6 and RS-based RAID.

Note that if SSDs wear out at different rates, the RAID system will trigger more data recovery and device replacement operations, which may introduce high system cost. First, the device recovery and replacement operation may take a lot of time and consume both central processing unit (CPU) and I/O resources, thus delay user requests. Second, it may also incur system management cost when replacing the worn-out SSD with a new one for handling device failures. At last, as pointed out in [27], if recovery is complex and time consuming, data loss will happen in a higher chance because any device failure at this moment may make data unrecoverable.

Therefore, we define the endurance of an SSD RAID system as the time period since the array was constructed until device replacement was triggered. Motivated from the statement above, it is of big significance to balance the writes to multiple devices in an SSD-based RAID array so as to improve the array endurance. To address this problem, we come up with an idea that the updating dependencies coming from different data symbols in the same stripe should be evenly distributed over all devices, and then we develop a diagonal coding scheme (DCS) based on this idea, this scheme can efficiently achieve wear-leveling among devices, and finally enhances the endurance of SSD RAID arrays. In particular, we make the following contributions in this paper.

1) We observe that nonuniformity of workloads have a negative impact on wear-leveling among devices under RAID configuration. Based on this observation, we develop a new DCS, which can achieve system-level wear-leveling across multiple devices for specific access patterns.
2) We design DCS5 and DCS-RS by adopting DCS for RAID-5 and RS-based RAID systems, respectively. DCS5 can tolerate single-device failure just like traditional RAID-5, and its encoding and decoding complexity are also the same as that of RAID-5. DCS-RS achieves the same level of reliability and complexity as RS. Besides, we discuss about the DCS for different codes in RAID-6 arrays.
3) We also propose an enhanced version of DCS with a buffer design, which is called DCS+, so to improve the system-level wear-leveling in SSD-based RAID arrays under general access patterns by triggering different actions to respond to different kinds of requests.
4) We validate the effectiveness of DCS+ in improving the endurance of SSD-based RAID arrays using trace-driven evaluations with real-world workloads. Besides, we show the I/O performance and encoding complexity of DCS+ in our experimental evaluations.

The rest of this paper is organized as follows. In Section II, we provide necessary background on SSDs and RAID, as well as a brief introduction to four erasure codes, e.g., RS, EVENODD, RDP, and X-code. We also illustrate the endurance issue of SSD-based RAID arrays in this section. In Section III, we first justify our motivation, and then describe the detailed design of our coding scheme. In Section IV, we use trace-driven evaluations to show the endurance improvement of our coding scheme over traditional RAID. We review related work in Section V and conclude the paper in Section VI.

II. Problem Formulation

In this section, we first provide some background on SSDs and RAID, then give an introduction of RS, EVENODD, RDP, and X-code. After that, we formulate the endurance problem in SSD-based RAID systems.

A. Background on SSDs and RAID

SSDs have three basic operations: 1) read, 2) write, and 3) erase. Read and write operations are performed in unit of pages, and the typical page size is 4 kB. Erase operations are performed in unit of blocks, and each block contains 64 or 128 pages in common configurations. For SSDs, data can only

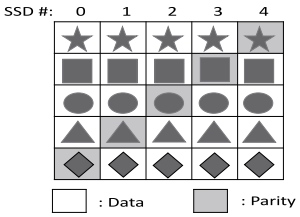


Fig. 2. Data layout of RAID-5.

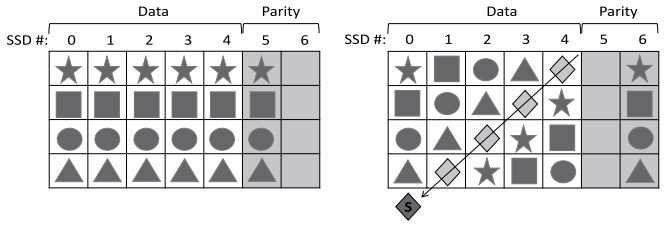


Fig. 3. Data layout of EVENODD.

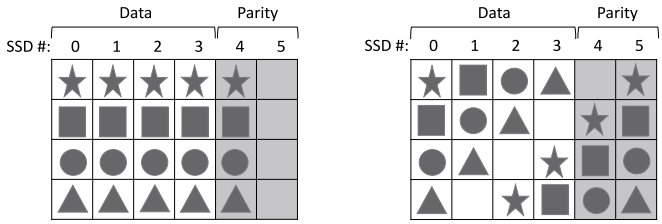


Fig. 4. Data layout of RDP.

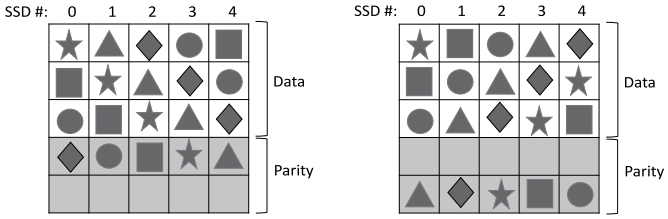


Fig. 5. Data layout of X-code.

be written to clean pages, and so erase operation is required to reset all pages in a block to the clean state. For performance consideration, SSDs adopt "out-of-place" overwrites. Precisely, when an update operation comes in, SSDs first write this new data to another clean page, which is termed as valid page, then mark the original page as invalid. Thus, SSDs continue generating invalid pages, and garbage collection (GC) must be triggered to reclaim the space of invalid pages when the number of clean blocks drops below a predefined threshold. In particular, the GC process first chooses a block according to the GC algorithm, then writes all valid pages in the chosen block to other clean pages, and finally erases the block.

Now we introduce the organization of an SSD-based RAID array. As we all know, RAID can provide different levels of data redundancy so to protect systems from a single device failure (e.g., RAID-5), double device failures (e.g., EVENODD, RDP, and X-code-based RAID-6), or a general number of device failures (e.g., RS-based RAID arrays). An SSD-based RAID array is composed of N devices, $(N-K)$ of which are data devices and K of which are redundant devices. For example, for RAID-5, we have $K = 1$. To generate data redundancy, K redundant symbols which are usually called parities are computed from $(N - K)$ data symbols, and then both the data symbols and the corresponding parities are striped across the N devices. In RAID systems, parity symbols are usually located across the whole array in a round-robin manner.

B. Overview of Different RAID Levels

1) RAID-5: RAID-5 is composed of N devices. The whole array is separated into multiple segments and each segment has the same data layout. A segment is further divided into N stripes. For traditional RAID-5, each stripe contains $N - 1$ data symbols and one parity symbol. The reader can refer to Fig. 2 for an example.

2) EVENODD-Based RAID-6: EVENODD code is defined as a $(p-1) \times (p+2)$ matrix in a segment, where p is a prime number. In fact, if p is not a prime number, the system may "add" some dummy data devices which store zero data. Thus, p will reach to the next larger prime number. Data symbols are located in the first p columns, and the last two columns contain parity symbols. EVENODD code uses the horizontal and diagonal parity-data relationships to tolerate up to double device failures. Note that an adjusting factor S is used for computation in each diagonal parity-data relationship. We illustrate the data layout of EVENODD in Fig. 3.

3) RDP-Based RAID-6: RDP code is defined as a $(p-1) \times (p+1)$ matrix in one segment, where p is a prime number. Data symbols are put into the first $p-1$ devices, while parity symbols are stored in the last two devices. RDP is an enhanced scheme of EVENODD as it removes the computation of the adjusting factor S for decreasing the computational complexity. RDP code also uses the horizontal and diagonal parity-data relationships to tolerate double device failures. The data layout of RDP is illustrated in Fig. 4.

4) X-Code-Based RAID-6: The data layout of X-code is a $p \times p$ matrix, where p is also a prime number. Data symbols are striped in the first $p-2$ rows across all devices, while parity symbols are stored in the last two rows. X-code tolerates double device failures by using both the diagonal and anti-diagonal parity-data relationships. Readers can refer to Fig. 5 for an example of X-code.

5) Reed–Solomon: RS code is the most popular erasure code and widely used in storage systems. It can tolerate a general number of device failures. Galois field arithmetic is used in the coding process of RS, which is different from the usage of XOR computation as in RAID-5 and erasure code-based RAID-6 described before. Fig. 6 shows the data layout of an RS-based RAID.

C. Problem Fomulation

As we stated in Section I, in an SSD-based RAID system, parity symbols suffer more writes than the corresponding data symbols in the same stripe, mainly because a parity symbol needs to be updated every time when a data symbol within the stripe gets updated.

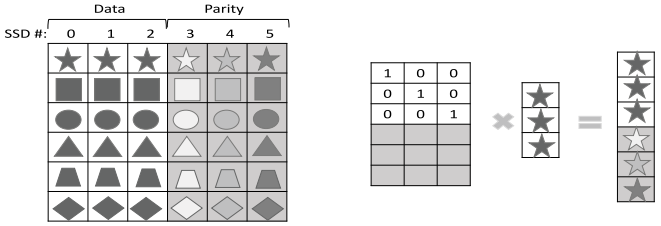


Fig. 6. Data layout of RS.

It is well known that real-world workloads are usually nonuniform, e.g., 80% of requests access only 20% of data, which is also called "The 80/20 rule" [9]. The 80/20 rule indicates that skewness exists in various I/O workloads. Skewness means that some data symbols are hot and accessed frequently, while others are cold and get accessed rarely. Besides of skewness, workloads also possess high spatial locality, which implies that when certain data symbols are requested, there is a high probability to access their surrounding data symbols. For example, in an online-editing scenario, successive user requests usually access the data symbols in a small hot region of the file. Skewness and locality within workloads make some parity symbols receive more update operations than others.

Note that the maximum number of erase operations that can be performed on each block in an SSD is limited, so when an SSD wears out, recovery and replacement process will be triggered immediately. The recovery process includes reading the corresponding symbols from survival SSDs, performing decoding operation, and writing the reconstructed data to a spare SSD. So it consumes a great amount of CPU resources and I/O operations to reconstruct data in the failed device, which delays user requests. In addition, device recovery and replacement also incur system management cost. Therefore, uneven wear of SSDs in an SSD RAID system may trigger the device recovery and replacement event quickly and frequently, and thus increases the system cost and decreases SSD RAID endurance.

From the statement above, we can conclude that ensuring wear-leveling across SSDs in an RAID array is significant to enhance the endurance of the whole array. There are two works addressing the wear-leveling problem among SSDs under RAID settings. Park *et al.* [21] developed a scheme for SSD-based RAID-5 system, which uses a table to record the write number of each parity symbol and then employ greedy algorithm to exchange hot parity symbols with cold ones. Another work of WeLe-RAID [27] proposed an addressing method for age-driven parity redistribution to balance write grade among SSDs in RAID-5 systems.

We address the problem of achieving system-level wear-leveling across devices in a general RAID array (e.g., RAID-5, RAID-6, and RS-based RAID) from a different angle. In particular, we develop a new DCS to enhance the array endurance. The detailed implementation of the coding scheme will be described in the next section. Note that since we focus on system-level wear-leveling across SSDs, we assume that each SSD already adopts some wear-leveling technique, and so it achieves perfect block-level wear-leveling. That is, all blocks inside an SSD should have the same number of erasures.

III. Design of DCS

In this section, we first describe the basic idea that motivates the design of our DCS, then we perform a simple analysis to show the rationale of the idea. After that, we illustrate on the details of our coding scheme, i.e., DCS5 for RAID-5 and DCS-RS for RS-based RAID, and discuss about the DCS for erasure-code-based RAID-6. Finally, we improve DCS for enhancing the array endurance under general access patterns.

A. Observations and Basic Idea

Our goal is to achieve system-level wear-leveling in an RAID array without lowering performance. We observe that skewness and locality of workloads have a negative impact on wear-leveling among devices under RAID configuration. To illustrate this impact, note that updating each data symbol corresponds to an update to the parity symbols within the same stripe. For ease of presentation, we call this dependent relationship as updating dependency. Under RAID configuration, the updating dependency caused by all data symbols in the same horizontal parity-data relationship (stripe) aggregates at the same parity symbol. Because of the skewness and locality within workloads, some stripes or sequential data symbols must be updated more frequently than others, and thus the corresponding parity symbols get even more writes because of the updating dependency, and it finally makes SSDs that contain a larger amount of these hot parity symbols wear out faster.

Based on the observation that the updating dependencies caused by all data symbols in the same stripe aggregate at the same parity symbol because of the horizontal parity-data relationship, we come up with a novel idea to balance writes across devices. Our basic idea is that the updating dependencies coming from different data symbols in the same stripe should be evenly distributed over all devices, not aggregating at only parity devices. To achieve this, we need to change the traditional coding scheme for RAID systems (e.g., RAID-5, RAID-6, and RS-based RAID), and design a new coding scheme. Note that we still need to keep the property of RAID systems in protecting the system from the same number of device failures. To describe the idea of our DCS, we take Fig. 7 as an example. Fig. 7(a) is an example of traditional coding scheme with horizontal parity-data relationship. A parity symbol is encoded from those data symbols in the same stripe (i.e., the same row). Fig. 7(b) is a simple example illustrating our idea. The data symbols are still striped across all SSDs just like traditional RAID, while the coding method becomes different. In particular, we use a DCS instead of the original row-based coding method. For example, the first data symbol in Fig. 7(b), $D_{0,0}$, has an updating dependency with parity symbol $C_{2,2}$ on SSD 2. Similarly, $D_{0,1}$, $D_{0,2}$, and $D_{0,3}$ have an updating dependency on SSD 0, SSD 3, and SSD 1, respectively. By using the diagonal coding method, we can see that the updating dependencies from data symbols in the same row, which may usually be accessed sequentially, are distributed over multiple SSDs.

B. Benefit of the Diagonal Coding Method

In this section, we perform theoretic analysis to show the benefit of the DCS in balancing writes to SSDs so as to

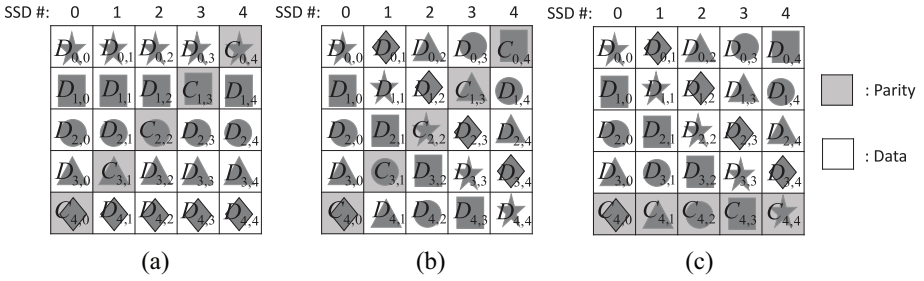


Fig. 7. Different data layouts in an SSD-based RAID-5 system. (a) Traditional RAID-5. (b) Diagonal method. (c) DCS5.

validate the effectiveness of our basic idea. In an $[(N-1)+1]$ RAID system, we suppose that different stripes (or rows) receive different numbers of update requests due to the skewness and locality of workloads, and we use S_i to denote the number of updates to stripe i. For simplicity, we assume that all data symbols within one stripe receive the same number of writes [i.e., each data symbol in stripe i receives $(S_i/(N-1))$ writes]. We relax this assumption to handle the case of general access patterns in Section III-D. We denote the number of write operations to the ith SSD using traditional RAID with horizontal parity-data relationship as in Fig.7(a) as N_a^i, and use N_b^i to denote the same number when our coding scheme is deployed as in Fig. 7(b). Under our assumption, we can easily derive N_a^i and N_b^i as follows:

$$\begin{aligned} N_a^i &= \sum_{j=0}^{N-1} \frac{S_j}{N-1} + (N-2)\times\frac{S_{N-1-i}}{N-1} \\ &= \frac{S}{N-1} + \frac{(N-2)S_{N-1-i}}{N-1} \\ N_b^i &= 2\times\sum_{j=0}^{N-1} \frac{S_j}{N-1} - 2\times\frac{S_{N-1-i}}{N-1} \\ &= \frac{2S}{N-1} - \frac{2S_{N-1-i}}{N-1} \end{aligned}$$

where $S = \sum_{j=0}^{N-1} S_j$. To compare the ability of the two coding schemes in achieving system-level wear-leveling, we derive the variance of the number of writes to each device under the two schemes, and denote them as $\mathrm{Var}(N_a)$ and $\mathrm{Var}(N_b)$, respectively. Note that the average number of writes to each SSD can be easily derived, and it is $E_N = (2/N)S$. We derive the variance as follows:

$$\begin{aligned} \mathrm{Var}(N_a) &= \frac{1}{N}\sum_{i=0}^{N-1}\left(N_a^i - E_N\right)^2 \\ &= \frac{1}{N}\sum_{i=0}^{N-1}\left[\frac{S}{N-1} + (N-2)*\frac{S_{N-1-i}}{N-1} - \frac{2}{N}S\right]^2 \\ &= \frac{(N-2)^2}{N(N-1)^2}\left(\sum_{i=0}^{N-1} S_i^2 - \frac{S^2}{N}\right) \end{aligned}$$

$$\begin{aligned} \mathrm{Var}(N_b) &= \frac{1}{N}\sum_{i=0}^{N-1}\left(N_b^i - E_N\right)^2 \\ &= \frac{1}{N}\sum_{i=0}^{N-1}\left[2\times\frac{S}{N-1} - 2\times\frac{S_{N-1-i}}{N-1} - \frac{2}{N}S\right]^2 \\ &= \frac{4}{N(N-1)^2}\left(\sum_{i=0}^{N-1} S_i^2 - \frac{S^2}{N}\right). \end{aligned}$$

Now we can compare the two coding schemes by computing the difference between $\mathrm{Var}(N_a)$ and $\mathrm{Var}(N_b)$. Mathematically, we have

$$\mathrm{Var}(N_a) - \mathrm{Var}(N_b) = \frac{(N-2)^2-4}{N(N-1)^2}\left(\sum_{i=0}^{N-1} S_i^2 - \frac{S^2}{N}\right).$$

Since we have $\sum_{i=0}^{N-1} S_i^2 \geq (1/N)(\sum_{i=0}^{N-1} S_i)^2$, according to the Cauchy–Schwarz inequality and $S = \sum_{i=0}^{N-1} S_i$, we can have the following inequality:

$$\mathrm{Var}(N_a) \geq \mathrm{Var}(N_b), \text{ if } N \geq 4. \quad (1)$$

Note that an RAID system is composed of N devices where N is usually larger than 4 under practical configurations. Thus, the above result implies that our idea can achieve better wear-leveling than traditional RAID with horizontal parity-data relationship. For example, if we set $N = 5$, $S_0 = 16A$ and $S_1 = S_2 = S_3 = S_4 = A$ according to the 80/20 rule, we can have $\mathrm{Var}(N_a) = (81/4)A^2$, $\mathrm{Var}(N_b) = (36/4)A^2$, and $\mathrm{Var}(N_a)$ is two times bigger than $\mathrm{Var}(N_b)$. We further validate the effectiveness of our idea in improving the endurance of SSD RAID arrays via experiments in Section IV.

C. Implementation of the Diagonal Coding Scheme

In previous sections, we describe our basic idea and theoretically validate its effectiveness. In this section, we describe the details of our coding scheme for different RAID codes, e.g., DCS5 for RAID-5 systems and DCS-RS for RS-based RAID systems. Finally, we discuss about the implementation of the DCS for erasure-code-based RAID-6 arrays.

1) DCS5 for RAID-5: DCS5 adopts a new data layout within a segment. Precisely, all parity symbols within a segment are located in the last stripe and a diagonal data set is used for encoding. A simple example of the data layout is shown in Fig. 7(c). DCS5 can make the number of writes

on different SSDs within a segment more even, and therefore makes SSDs wear out more evenly than traditional RAID-5 configuration.

a) Encoding function: Our DCS5 can be easily deployed by system designers, and the encoding function can be summarized as follows:

$$C_{N-1,j} = \oplus_{i=0}^{N-2} D_{i,(i+j+1) \bmod N} \tag{2}$$

where N is the number of devices in the system, and $\oplus$ indicates the XOR operation. The function above describes the relationship among data symbols and the corresponding parity symbols in the first segment. It tells the system which data symbols can be used to encode each parity symbol. In a segment, a symbol located in the ith row and the jth column, where i and j range from 0 to $N-1$, will be denoted by either $C_{i,j}$ or $D_{i,j}$. Now we use Fig. 7(c) as an example to describe the encoding process. The parity symbol $C_{4,0}$ is: $C_{4,0} = D_{0,1} \oplus D_{1,2} \oplus D_{2,3} \oplus D_{3,4}$. Due to the same data layout in all segments, the coding function also applies for other segments through modulo operation in the subscripts. Besides, We can easily decode the symbols according to the encoding function because decoding is just an inverse process of encoding.

b) Complexity: Compared with traditional coding scheme for RAID-5 systems, DCS5 has the same complexity for all encoding, decoding, and recovery operations. In particular, DCS5 needs $O(N^2)$ XORs per segment for encoding/recovery and $O(N)$ XORs for decoding a data symbol, which is the same as traditional RAID-5. We further show the complexity of our coding scheme with experiments, and compare the coding performance between DCS5 and traditional RAID-5 using the SSD simulator in Section IV.

c) MDS property: A coding set is defined as a set of symbols which contains its parity symbol(s) and related data symbols. The encoding function of DCS5 shows that all symbols in a coding set are located in different devices and there are no two symbols overlapping in all coding sets. Therefore, we can easily check that DCS5 still maintains the minimum detectable signal (MDS) property and it can tolerate any single device failure.

d) Benefits: Every row under the DCS5 setting [see Fig. 7(c)] has one more data symbol than that in Fig. 7(a) and (b) except for the last row. Moreover, all updates to the parity symbols in the case where the corresponding data symbols are in the same stripe will be distributed over all devices rather than only one device, thus system performance can be improved when serving sequential requests which span one or multiple stripes. More importantly, DCS5 achieves better system-level wear-leveling than RAID-5 if all data symbols in the same row receive the same number of writes. We extend the design of DCS5 in the next section so as to handle general access patterns.

2) DCS-RS for Reed–Solomon-Based RAID: RS code use horizontal parity-data relationship for data layout. Differently, DCS-RS adopts a new data layout within a segment which is an $N \times N$ matrix. All parity symbols which are located in the last K columns within a segment under the traditional RS code configuration are now located in the last K rows, and a diagonal parity-data relationship is used for encoding. A simple example of DCS-RS with $N = 6$ and $K = 3$ is shown in Fig. 8. DCS-RS actually makes SSDs wear out more evenly than traditional RS-based RAID configuration.

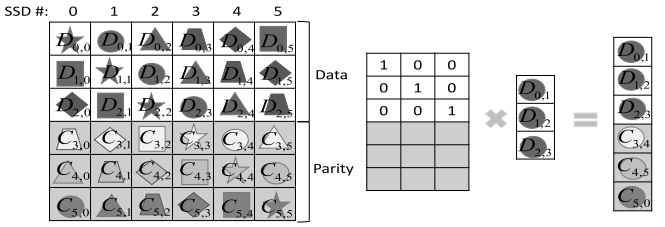


Fig. 8. DCS-RS: Diagonal coding scheme for Reed–Solomon code.

a) Encoding function: The encoding function of DCS-RS can be summarized as follows:

$$C_{N-1,j \bmod N}, \ldots, C_{N-K,(j-K+1) \bmod N} = \mathrm{GF}_{i=0}^{N-1-K} D_{i,(i+j+1) \bmod N} \tag{3}$$

where N is the total number of devices in the system and K is the number of redundant devices. GF means a Galois field arithmetic used by traditional RS code. The function indicates the system uses diagonal parity-data relationship for coding operation. For example, parity symbols $C_{5,0}$, $C_{4,5}$, and $C_{3,4}$ are calculated from diagonal data set $D_{0,1}$, $D_{1,2}$, and $D_{2,3}$ shown in Fig. 8. The coding function also applies for other segment through modulo operation to the subscripts because of the same data layout in all segments.

b) Complexity: Compared with traditional RS code, DCS-RS needs the same number of read/write operations for encoding, decoding, and updating process. So DCS-RS has the same complexity as traditional RS, and we also show this by experiments.

c) MDS property: The encoding function of DCS-RS above shows that all data and parity symbols in one diagonal coding set are distributed across all devices. This means that if RS code can tolerate K device failures, DCS-RS that only changes the data layout of original RS code can also tolerate K device failures, so DCS-RS still maintains the MDS property.

d) Benefits: Due to the similar reason as in DCS5 discussed above, the system under DCS-RS configuration will achieve better system-level wear-leveling and better performance.

3) DCS for RAID-6: As we analyzed at the beginning of this section, DCS is also suitable for RAID-6 arrays. We know that there are many kinds of erasure codes for RAID-6. EVENODD and RDP adopt horizontal and diagonal parity-data relationship for fault tolerance. So these two kinds of codes suffer from the problem caused by updating dependency of horizontal parity-data relationship. Besides, their updating complexity is high because updating one data symbol may cause more than two updates to the corresponding parity symbols, especially for EVENODD code, which may bring even more writes into the array than RDP.

Fortunately, we have X-code which tolerates double-device failure by using both diagonal and anti-diagonal parity-data

relationships. X-code can evenly distribute updating dependencies caused by data symbols in one stripe over all devices. Furthermore, its updating complexity is lower than EVENODD and RDP because no matter which data symbol needs to be updated, RAID-6 under X-code configuration will only update two parity symbols.

Hence, X-code can achieve better system-level wear-leveling than RDP and EVENODD. We will further show this via experiments in Section IV, and we also apply our scheme to further improve the system-level wear-leveling for X-codes.

D. Extending DCS for Handling General Access Patterns

In the last section, we present the design of our new coding scheme. DCS5 and DCS-RS have the same complexity as traditional RAID-5 and RS-based RAID, respectively, and achieve better system-level wear-leveling if requests span one or multiple stripes. In the following, we discuss about the impact of request size on system-level wear-leveling.

1) On the one hand, if write requests always span one segment or multiple segments, they will cover one or more segments and thus distribute the updating dependency evenly among SSDs. Therefore, traditional RAID coding schemes already make writes be evenly distributed across the whole system, which helps achieve system-level wear-leveling like the DCS. In this case, traditional RAID coding schemes and DCS have similar effect on system-level wear-leveling. On the other hand, if write requests span only one or multiple stripes, but not a full segment, then DCS can achieve better system-level wear-leveling than traditional coding schemes. Thus, we conclude that our design is more suitable for the workloads in which most requests cover only one or more stripes.
2) We find that requests under real-world workloads may span only a partial stripe and random updates to sequentially written data may also happen. If random updates usually happen, one or multiple data symbols in a stripe may become hotter and receive much more writes than other data symbols in the same stripe, which causes the corresponding parity symbols suffer more writes. In this scenario, neither DCS nor traditional coding scheme can achieve a desired system-level wear-leveling.

A real-world workload may contain various kinds of access patterns. To address this challenge, we propose a heuristic design, which is called DCS+, to improve system-level wear-leveling under general access patterns.

a) Basic idea: DCS+ can be deployed in file system level. Specifically, it divides the whole RAID array into two regions, which are termed as sequential region (SR) and random region (RR), respectively. The rough idea of DCS+ is that large accesses are directed to SR, and small accesses are directed to RR. We call a request a large request if it accesses at least S continuous data symbols, otherwise, we call it a small request. Here, S is a tunable parameter and $S \geq 2$.

b) Buffer design: In terms of the implementation, we allocate a buffer to differentiate small requests from large requests. In particular, we deploy the queueing technique in this buffer and use red–black tree data structure to manage requests, including the offset, size, and timestamp of requests. Specifically, before putting a write request into the queue, DCS+ will first check whether the offset of this request exists in the red–black tree. If it does, DCS+ will check whether the size of this request covers the size or overlaps a part of that recorded in the red–black tree. After that, DCS+ decides to remove the duplicate request in the queue or modify the partial duplicate one. This helps reduce the number of writes and reads to SSDs, and thus improves the I/O response time. Finally, we add the new request or partially modified one into the queue, and update the red–black tree. In our experiments, DiskSim is used as a system-level simulator, while the trace processor and this buffer design are implemented on the top of DiskSim. When the queue dispatches new write requests into system (i.e., DiskSim simulator), DCS+ separates large write and small write into two regions according to the data layout of DCS5 or DCS-RS. In order to issue update operations, DCS+ triggers different actions according to the access patterns. Specifically, when receiving a large update access, it directly sends the update request to the corresponding SSDs for data updating. On the other hand, when a small update access arrives, it will reconstruct a new stripe by appending the data to the next free space in RR, and mark the original data as invalid.

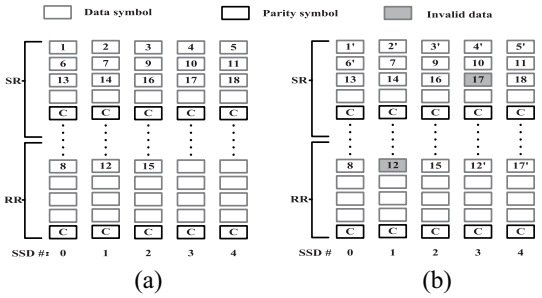


Fig. 9. Example of DCS5+. (a) Data placement for new writes. (b) Data placement for updates.

Fig. 9 shows an example of our solution for RAID-5 called DCS5+ in which we set S as 2. In Fig. 9(a), the large requests accessing the data symbol sets $(1, 2, 3, 4, 5, 6, 7)$, $(9, 10, 11)$, $(13, 14)$, and $(16, 17, 18)$, and the small requests accessing data symbols 8, 12, and 15 are directed separately to SR and RR. Now we assume that there is one large request accessing data symbols $(1, 2, 3, 4, 5, 6)$, and two small requests accessing data symbols 12 and 17, respectively. Then DCS5+ performs in-place update for the large request, while for the small requests, it first writes new data $12'$ and $17'$ to new location, and then marks the old data as invalid, as described in Fig. 9(b). One thing we would like to emphasize is that Fig. 9 shows the logical view from the file system.

We summarize the implementation of DCS+ as in Algorithm 1 stated below.

In terms of the implementation of DCS+, we further make the following discussions.

1) *The Choices of Buffer Size in DCS+:* Using a buffer in DCS+ can potentially reduce writes and reads to SSDs due to the skewness and locality of workloads,

Algorithm 1 Framework of DCS+

```
for Each request at the head of the queue in buffer do
  if It is a new write then
    if It accesses more than S symbols then
      Direct it to SR;
    else
      Direct it to RR;
    end if
  end if
  if It is an update then
    if It accesses more than S symbols then
      Update data in related SSDs;
    else
      Write the new data to RR;
      Mark the old location as invalid;
    end if
  end if
end for
```

so it is beneficial for erasure reduction and I/O performance improvement. The larger the queue size of the buffer is, the more improvement DCS+ will achieve. However, the larger the queue size is, the more memory space will be consumed. We further show and discuss this tradeoff in the experiment section. On the other hand, considering that the loss of requests stored in buffer at a sudden power off may cause reliability degradation. In order to avoid this disaster, we suggest to implement the buffer scheme with a nonvolatile random access memory (NVRAM).

2) *The Choices of S in DCS+:* Note that we take the requests which access at least S data symbols as large requests. If we let $S \geq N$, then a large access always spans at least one stripe, which leads to better system-level wear-leveling. However, the system will be busy in handling small updates, i.e., writing new data to the RR region and marking old locations as invalid, which also causes overheads and degrades system performance. On the other hand, if we set $S = 2$, a request accessing two or more data symbols is treated as a large request, which can also achieve better system-level wear-leveling than DCS, while the improvement is limited. We will further show the impact of parameter S on system-level wear-leveling and I/O performance in Section IV.

3) *How to Avoid Correlated Failures in DCS+:* If DCS+ makes all SSDs in an SSD RAID array reach the same aging rate, it may cause correlated failures, which means multiple SSDs will wear out at approximately the same time. In order to avoid correlated failures, we suggest that an SSD RAID system should monitor the lifetime of SSDs. Once the number of erasures on all SSDs exceeds a certain predefined threshold, the system should start migrating data and replacing all SSDs with new ones. In other words, instead of replacing all SSDs when they wear out, one can trigger the replacement operation earlier according to the reliability requirement of applications.

TABLE I
DISKSIM CONFIGURATION

Parameter	Value
page size	4KB
number of pages per block	64
number of blocks per package	16384
number of packages per SSD	1
page read latency	0.025ms
page write latency	0.200ms
block erase latency	1.500ms

4) *Garbage Collection in DCS+:* The process of handling small update accesses in DCS+ is done by the file system, and it can let SSDs treat small update accesses as write operations. However, the process may make SSDs fail to reclaim those symbols (pages) which are marked as invalid by file system. In addition, those invalid symbols must be retained for RAID coding, and thus consume extra storage space. The bigger S is, the more space DSC+ will consume. Therefore, GC is necessary to be deployed in the file system so as to help SSDs reclaim invalid symbols. The GC mechanism in the file system can use a table to record invalid symbols after small update accesses. When GC is triggered, e.g., when system is at idle or the space utilization of the RAID system exceeds a predefined threshold, it first chooses a segment containing the most invalid symbols by using a greedy algorithm, then reads out valid data symbols, writes them to a clean segment in the RR region, and finally sends TRIM commands to SSDs to explicitly delete the symbols in the selected segment. We note that GC operations indeed cause more writes into the system, and influence the lifetime of SSDs. Besides, this process will also demand extra bandwidth. However, no matter whether those invalid symbols are marked by SSDs or by DCS+, they always need to be reclaimed. That is, DCS+ just migrates some GC operations from the SSD level to the RAID level.

IV. EXPERIMENTAL EVALUATION

In this section, we conduct extensive evaluations to show the effectiveness and performance of our coding scheme by using the widely accepted trace-driven SSD simulator, DiskSim [13] with SSD extension [1]. In particular, we compare our coding scheme with traditional RAID (RAID-5, RAID-6, and RS-based RAID) from the aspects of endurance and I/O performance. In terms of the endurance, we compare system-level wear-leveling, which is denoted by the total and standard deviation of the number of erasures performed on each SSD. For comparing I/O performance, we collect the average response time of overall array system from output files after processing workloads. We also study the coding time of our scheme so as to show its complexity. In the following sections, we first introduce the DiskSim configuration and the I/O workloads used for driving our evaluations, then we present the evaluation results.

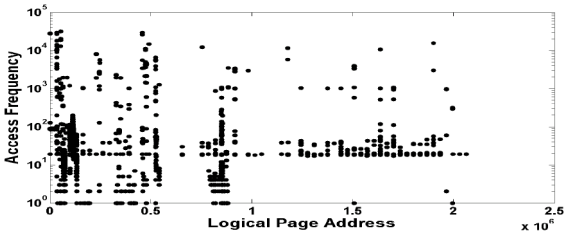


Fig. 10. Write-access frequency in online trace.

A. DiskSim Configuration

In our evaluation, we consider an SSD-based RAID system that is composed of N SSDs, each of which has 4 GB capacity and is associated to a controller. All controllers are attached to an I/O bus. Each SSD is configured to contain one package, each of which contains eight planes with 2048 blocks each. There are 64 pages of size 4 kB in each block. For timing parameters, reading one page of data from flash media to a register is set as 0.025 ms, and the time of programming a page data from a register to flash media is set as 0.2 ms. We configure the time to erase one block as 1.5 ms. The main parameters of DiskSim configuration are listed in Table I. DiskSim is used as a system-level simulator to respond requests for different data layout algorithms (e.g., traditional RAID-5, DCS5 for RAID-5, RS-based RAID, DCS-RS for RS-based RAID, or different codes for RAID-6) and I/O workloads.

B. Workloads

We consider the following five workloads in our evaluations.

1) *Src1_2 and Proj_0 [20]:* They are one-week block I/O traces of enterprise servers at Microsoft Research Cambridge. There are 36 I/O traces from 36 different volumes on 13 servers. We choose two of them for experiments.
2) *Online [25]:* It is an I/O trace that is collected from a course management system using Moodle in a university department.
3) *Synthetic:* It is a synthetic workload that accesses data with normal distribution.
4) *Fin1 [24]:* It is collected from a large financial institution running online transaction process application.

All workloads are with the write locality. We collect the write-access frequency in the range of logical page address from workloads and show the result under online trace in Fig. 10. From Fig. 10, we find that some logical pages receive tremendous write operations while others are rarely written or even get no write operation. Moreover, plenty of pages receive the same number of writes as those nearby pages do. Additionally, other workloads share the similar trend. Those workloads are write dominant. In particular, Src1_2 trace contains 1.9 million requests, and the average request size is 29 kB. Proj_0 gets 4.2 million requests with average size of 38 kB. All requests in online traces have a size of 4 kB. The synthetic workload which accesses data with normal distribution has five million requests with an average size of 48 kB, and the standard deviation of request size is 16 kB, which means that most requests has the size ranging from 32 to 64 kB. Fin1 trace contains 5.3 million requests, and most of them are not bigger than 8 kB. Readers can refer to Table II for the detailed statistics of different workloads.

TABLE II
STATISTICS OF DIFFERENT TRACES

Trace	Total # of requests	Avg. request size	Write ratio
Src1_2	1.9 M	29 KB	0.7463
Proj_0	4.2 M	38 KB	0.8752
Online	5.7 M	4 KB	0.7388
Synthetic	5.0 M	48 KB	1
Fin1	5.3 M	5.22 KB	0.7683

TABLE III
RATIO OF THE MAXIMUM NUMBER OF ERASURES TO THE MINIMUM ONE PERFORMED ON EACH SSD UNDER DIFFERENT RAID CONFIGURATIONS

Trace	# of SSDs	Ratio (MAX/ MIN) in RAID5	Ratio (MAX/ MIN) in DCS5	Ratio (MAX/ MIN) in DCS5+
Src1_2	11	1.121	1.053	1.034
	13	1.198	1.105	1.064
Proj_0	11	1.083	1.024	1.010
	13	1.057	1.021	1.014
Online	11	1.285	1.213	1.009
	13	1.408	1.236	1.005
Synthetic	11	1.002	1.0006	1.0003
	13	1.002	1.0004	1.0003
Fin1	11	1.287	1.207	1.033
	13	1.209	1.353	1.020
Trace	# of SSDs	Ratio (MAX/ MIN) in RS	Ratio (MAX/ MIN) in DCS-RS	Ratio (MAX/ MIN) in DCS-RS+
Src1_2	11	1.083	1.078	1.030
	13	1.115	1.045	1.033
Proj_0	11	1.046	1.019	1.014
	13	1.146	1.035	1.021
Online	11	1.263	1.127	1.006
	13	1.232	1.161	1.013
Synthetic	11	1.0012	1.0002	1.0001
	13	1.0019	1.0002	1.0001
Fin1	11	1.295	1.210	1.030
	13	1.218	1.155	1.005

C. System-Level Wear-Leveling

In this experiment, we set the size of the coding unit as 4 kB, and S as the array size. In addition, we configure a buffer with the size of 4 MB for DCS+. We deploy traditional RAID-5 which we denote as RAID5, as well as DCS5, and DCS5+ with different array size N (where $N = 7, 9, 11, 13$). We also deploy RS code, DCS-RS, DCS-RS+ (which adopts DCS+ for RS code), RDP, EVENODD, X-code, X-code+ (which applies DCS+ for X-code) in our evaluations. To compare the system-level wear-leveling of different coding schemes, we first collect the total number of erasures performed on each SSD, and then compute the standard deviation of these numbers. The results are shown in Fig. 11. Besides, we collect the ratio of the maximum number of erasures to the minimum one under different setups as shown in Table III.

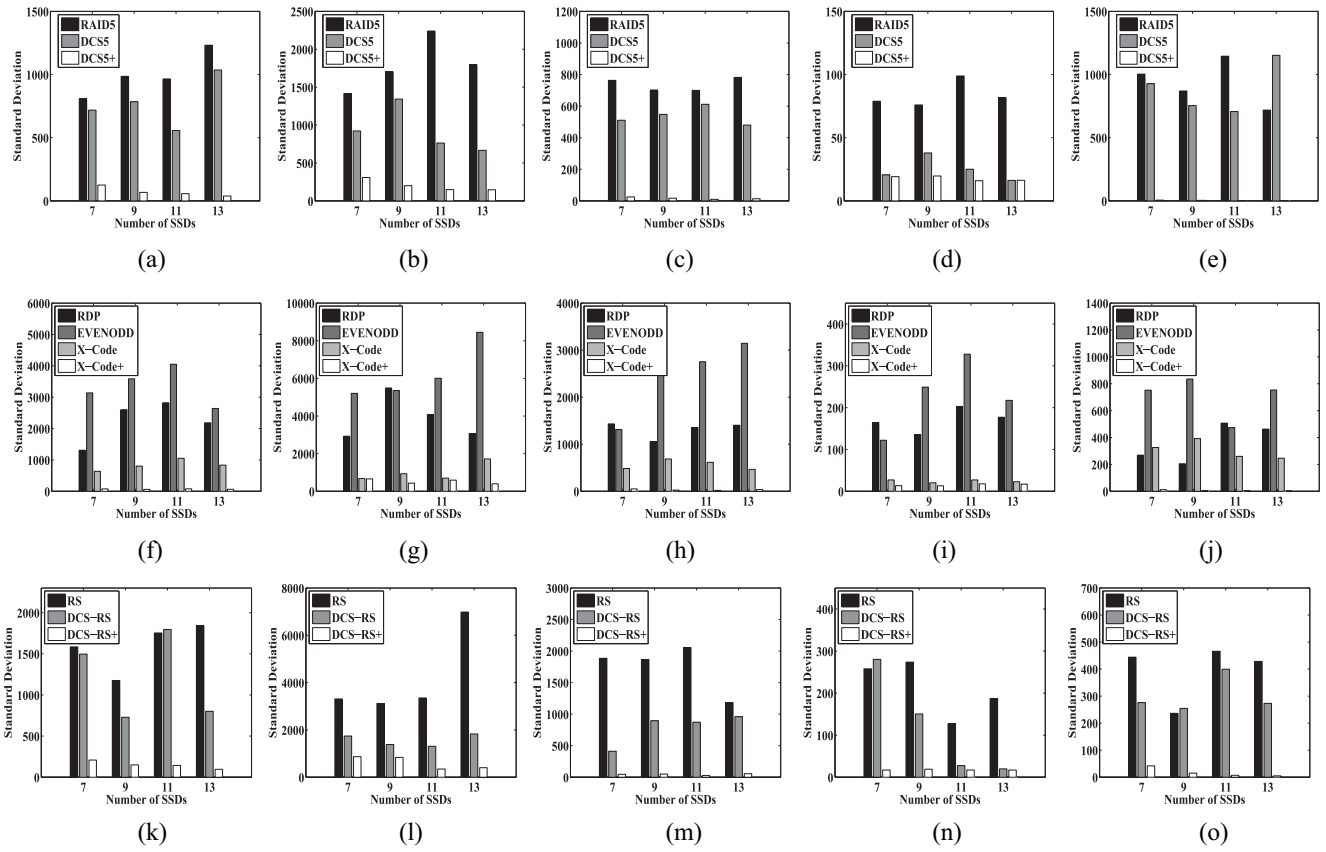


Fig. 11. Standard deviation of the number of erasures performed on each SSD under different traces and RAID configurations. (a) Src1_2 for RAID-5. (b) Proj_0 for RAID-5. (c) Online for RAID-5. (d) Synthetic for RAID-5. (e) Fin1 for RAID-5. (f) Src1_2 for RAID-6. (g) Proj_0 for RAID-6. (h) Online for RAID-6. (i) Synthetic for RAID-6. (j) Fin1 for RAID-6. (k) Src1_2 for RS RAID. (l) Proj_0 for RS RAID. (m) Online for RS RAID. (n) Synthetic for RS RAID. (o) Fin1 for RS RAID.

In Fig. 11, the horizontal axis represents the array size, i.e., the total number of SSDs containing in the array, and the vertical axis represents the standard deviation of the number of erasures. Each subfigure corresponds to a different I/O trace. In particular, the subfigures in different row in Fig. 11 are the results of different RAID levels with different coding methods under the five I/O workloads. For RAID-5 tests, the ratio of the standard deviation of RAID5 to that of DCS5+ ranges from around 80:1 to 4:1, and we obtain similar results from the subfigures in the middle row for RAID-6 and the subfigures in the last row for RS-based RAID. Small accesses dominate in online and Fin1 traces, and so DCS5+, X-code+, and DCS-RS+ achieve much better wear-leveling than those traditional methods as shown in Fig. 11(c), (h), and (m) and (e), (j), and (o). Src1_2 and Proj_0 contain many large writes spanning one or more stripes, and these requests lead to the results in Fig. 11(a), (f), and (k) and (b), (g), and (l). Synthetic workload contains few small requests, so DCS+ has a very small improvement compared with DCS in Fig. 11(d), (i), and (n).

To further show the difference of the number of erasures performed on each SSD, we show the ratio of the maximum one to the minimum one under different setups and I/O traces in Table III. The largest ratio is 1.408 in online trace under traditional RAID-5 containing 13 SSDs, which means that the youngest SSD has just been erased about half of the times as that of the oldest SSD. Note that our coding scheme, DCS5+ and DCS-RS+, can make the ratio be close to one, which validates the effectiveness of our coding scheme in evening the erasures on multiple SSDs.

The evaluation results show that DCS+ (DCS5+, DCS-RS+, and X-code+) always achieves better wear-leveling compared with corresponding traditional version and DCS version as the standard deviation is smaller. Therefore, SSDs wear more evenly by using our scheme DCS+. We also find that DCS (DCS5 and DCS-RS) also achieves better system-level wear-leveling than traditional RAID-5 and RS-based RAID in most scenarios, while the improvement is much smaller than DCS+. This evaluation shows that DCS may not be able to enhance the endurance of SSD-based RAID arrays for any kind of workloads, while DCS+ works well for all workloads.

D. Total Number of Erasures Performed on RAID

The wear-leveling experiments described above show that our DCS and DCS+ can achieve better wear-leveling among devices than traditional RAID. In this evaluation, we focus on examining how the durability of RAID arrays is affected by our DCS and DCS+.

We collect the number of erasures performed on the SSD RAID array under different configurations, where the array size is set as 7, 9, 11, and 13, and S equals to the array size.

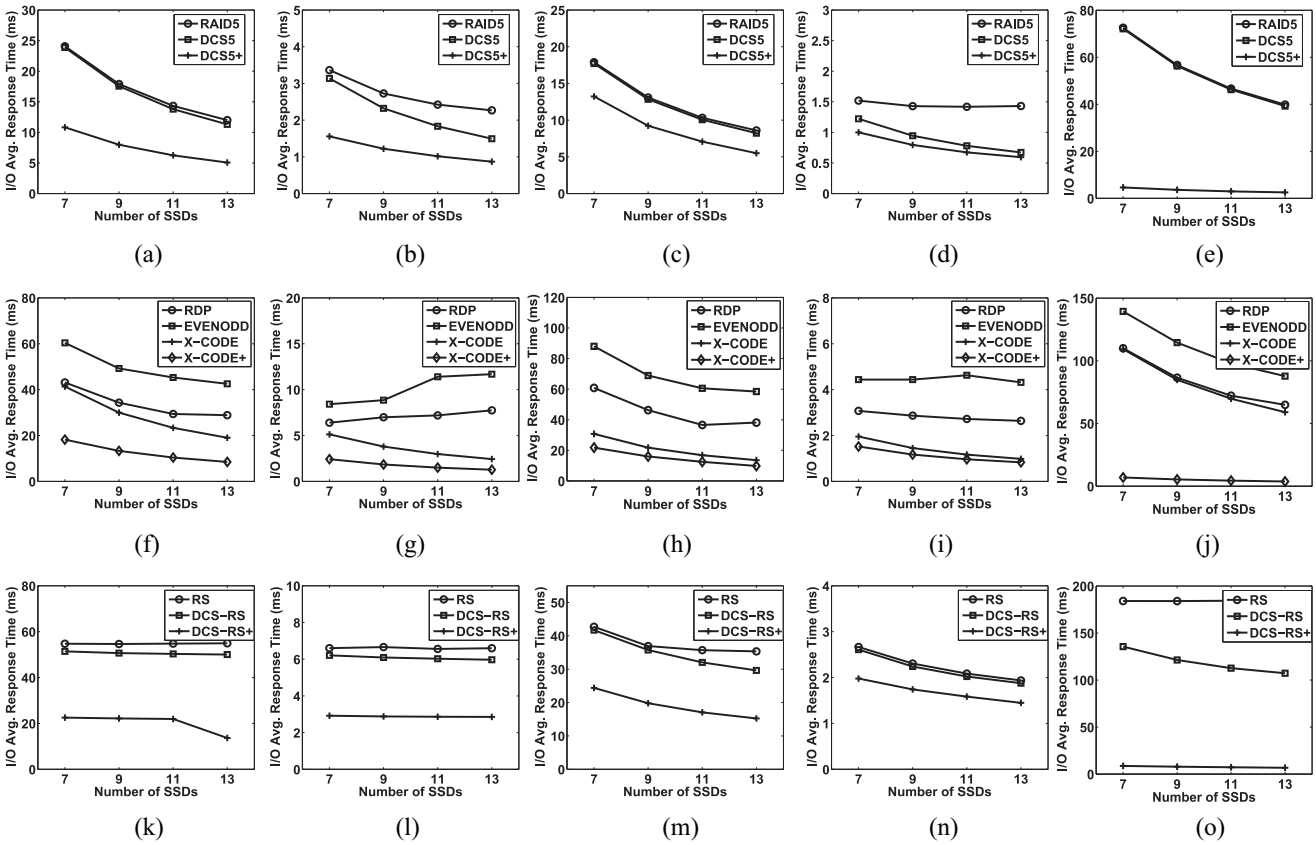


Fig. 12. Average response time of overall array system under different traces and RAID configurations. (a) Src1_2 for RAID-5. (b) Proj_0 for RAID-5. (c) Online for RAID-5. (d) Synthetic for RAID-5. (e) Fin1 for RAID-5. (f) Src1_2 for RAID-6. (g) Proj_0 for RAID-6. (h) Online for RAID-6. (i) Synthetic for RAID-6. (j) Fin1 for RAID-6. (k) Src1_2 for RS RAID. (l) Proj_0 for RS RAID. (m) Online for RS RAID. (n) Synthetic trace for RS RAID. (o) Fin1 for RS RAID.

TABLE IV
TOTAL NUMBER OF ERASURES PERFORMED ON SSD RAID UNDER ONLINE TRACE WITH DIFFERENT RAID CONFIGURATIONS

# of SSDs	Total # of erasures in RAID5	Total # of erasures in DCS5	Total # of erasures in DCS5+
7	122290	122290	114480
9	119028	119027	111119
11	115767	115767	107730
13	112502	112504	104491

Table IV shows the evaluation results under online trace. We can see that the difference between RAID5 and DCS5 is negligible under online trace. However, DCS5+ suffers much less erasures in the comparison with RAID5 and DCS5. Because the buffer in DCS5+ removes many duplicate or partial duplicate requests due to the locality of online workload, and finally reduces the number of write operations, which leads to less erasures. This implies that DCS5+ decreases the total number of erasures performed on the array, and it also balances the erasures on multiple devices. Since the results of different RAID levels with different coding methods under different traces are similar, we do not show them in the interest of space.

Based on the results in Fig. 11 and Table IV, we can conclude that our scheme with the heuristic solution (DCS+) greatly enhances the endurance of SSD-based RAID arrays as DCS+ not only decreases the total number of erasures, it also makes the erasures on multiple SSDs more even.

E. I/O Performance

In this section, we study the I/O performance of RAID arrays using different coding schemes under different RAID levels. After processing those workloads under different configurations, we collect the average response time of overall array system from output files. The results of comparison are shown in Fig. 12.

Each subfigure in Fig. 12 corresponds to one I/O trace. The *x*-axis represents the array size and the *y*-axis means the average response time. From the subfigures in the first row for RAID-5, we find that DCS5 which balances the write operations among SSDs can achieve better I/O performance than RAID5, but this improvement is not obvious. Additionally, DCS5+ further reduces the average response time and has a large improvement compared with RAID5. Because DCS5+ not only balances writes, it also reduces the number of requests via the buffer design. We can obtain similar results from the subfigures in the last row for RS-based RAID tests.

Besides, we obtain more useful information from the subfigures in the second row for RAID-6. EVENODD causes the largest average response time under all configurations. Because when updating one data symbol, EVENODD may bring more than two updates to the corresponding parity symbols, which

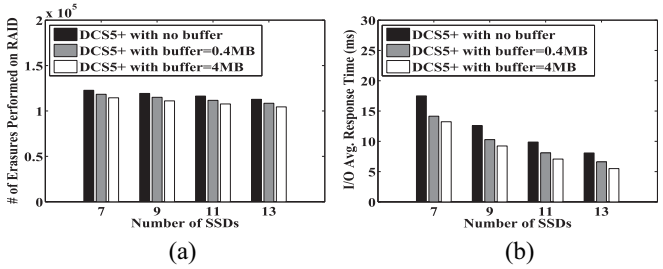


Fig. 13. Impact of the buffer size under online trace. (a) Endurance. (b) Average response time.

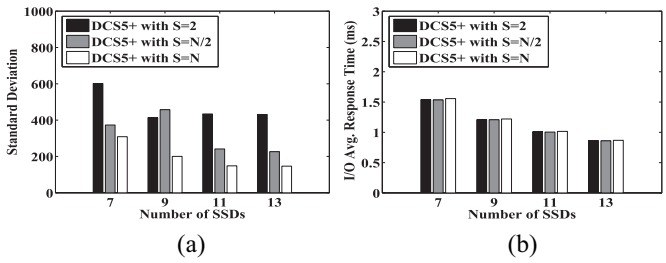


Fig. 14. Impact of the parameter S under Proj_0 trace. Note that N denotes total number of devices. (a) Endurance. (b) Average response time.

brings more operations into SSD RAID systems. Due to the same reason, average response time in RDP is higher than X-code and X-code+. X-code and X-code+ always cause two parity symbol updates once updating one data symbol, but X-code+ with the buffer will further improve the I/O performance.

From Fig. 12, we can see that DCS indeed improves the system performance under all settings. The main reason is that it can make the requests to multiple SSDs more even, and so each SSD receives almost the same number of requests. In addition, with the buffer design, DCS+ will have a much larger improvement.

F. Discussion on System Parameters

We first make a discussion about how different buffer size of DCS+ affects both the endurance and the I/O performance of an SSD RAID system. In this evaluation, we set S as the array size and study the impact of different buffer size (e.g., 0 MB, 0.4 MB, and 4 MB). We show the results of DCS5+ under online trace in Fig. 13.

With the buffer size increases, we can see that DCS5+ reduces the number of erasures under all settings from Fig. 13(a), and improves I/O performance from Fig. 13(b). In Fig. 13(a), DCS5+, with buffer size of 0 MB, suffers almost the same number of erasures as RAID5 and DCS5 do compared to Table IV, and achieves almost the same average response time as RAID5 and DCS5 do compared to Fig. 12. Therefore, the buffer design in DCS+ indeed helps enhance both the endurance and the I/O performance. However, memory is a scarce resource and we could not allocate much memory for the DCS+ buffer. Based on the results in Fig. 13, the buffer size of 4 MB, which is acceptable requirement, is enough to achieve a good improvement for DCS5+.

Then, we study how both the endurance and the I/O performance are affected by the parameter S. In this section, we set the buffer size as 4 MB, and vary S ($S = 2$, $S = (1/2)N$ and $S = N$, where N is set as 7, 9, 11, and 13). Note that we let $(N/2)$ be equal to $\lfloor (N/2) \rfloor$ in our implementation. In this experiment, we consider Proj_0 trace as an example because it has a large average request size (see Table II). Thus, the system could handle large accesses when S is large.

Fig. 14 shows the evaluation results of DCS5+. The x-axis is the number of SSDs, and the y-axis is the standard deviation of the number of erasures on each SSD. We see that the bigger S is, the better system-level wear-leveling DCS5+ achieves, but the larger average response time it causes, which validates the discussion about the choices of S in Section III-D.

TABLE V
COMPARISON OF CODING TIME BETWEEN RAID5 AND DCS5

# of SSDs	Coding time(ms) in RAID5	Coding time(ms) in DCS5	Coding time(ms) in RAID5	Coding time(ms) in DCS5
7	2035463.5	2061684.4	972959.9	963533.3
9	2067158.2	2028089.3	1005218.7	994807.7
11	2183578.0	2136415.4	1020035.3	993981.9
13	2207876.2	2260809.3	1005525.3	1023017.5

G. DCS Coding Complexity

As we have analyzed in Section III, DCS5 has the same complexity as traditional RAID-5 in terms of encoding, decoding, and recovery. DCS-RS also has the same complexity as traditional RS. In this section, we use DiskSim with SSD extension to validate the result. The number of SSDs in this experiment is set as 7, 9, 11, and 13, and we consider different coding units with size 4 and 8 kB. The comparison of the coding time between DCS5 and RAID5 is shown in Table V. The coding unit used in the second and third columns is set as 4 kB, and it is set as 8 kB in the fourth and fifth columns. We see that the coding performance of DCS5 is almost the same as that of the traditional RAID-5. Besides, the results of DCS-RS and traditional RS is similar, so we do not show them in the interest of space.

H. Comparison With WeLe-RAID

WeLe-RAID [27] aims at achieving system-level wear-leveling for SSD-based RAID-5 as DCS5+ does. Differently, WeLe-RAID balances the number of write operations among SSDs via calling a parity redistribution process when the difference of number of writes to SSDs exceeds a predefined threshold. This process lowers system performance due to the overhead caused by data migration and the uneven distribution of parities. Additionally, the effectiveness of system-level wear-leveling depends on the predefined threshold. The smaller the threshold is, the better system-level wear-leveling WeLe-RAID can achieve, but the lower performance it will suffer.

TABLE VI
SYSTEM-LEVEL WEAR-LEVELING AND PERFORMANCE COMPARISON BETWEEN WELE-RAID AND DCS5+ UNDER ONLINE TRACE

	# of SSDs	Wele-RAID (100000)	Wele-RAID (50000)	DCS5+ $S = N$
Standard Deviation of Erasured Number	7	619.4	370.6	25
	9	503.2	312.7	17
	11	519.4	359.7	9.5
	13	535.4	352.8	13.5
I/O Avg. Response Time (ms)	7	33.449699	33.583252	13.225518
	9	24.190365	24.22565	9.230509
	11	18.951007	18.967502	7.067279
	13	15.589729	15.597556	5.49416

TABLE VII
SYSTEM-LEVEL WEAR-LEVELING AND PERFORMANCE COMPARISON BETWEEN WELE-RAID AND DCS5+ UNDER PROJ_0 TRACE

	# of SSDs	Wele-RAID (100000)	Wele-RAID (50000)	DCS5+ $S = N$
Standard Deviation of Erasured Number	7	919.7	642.2	308.8
	9	704.5	563.3	200.4
	11	793.5	439.4	148.4
	13	883.8	500.2	146.5
I/O Avg. Response Time (ms)	7	3.365105	3.366667	1.556822
	9	2.731328	2.733376	1.220289
	11	2.427087	2.424396	1.015007
	13	2.254138	2.257918	0.867339

We compare system-level wear-leveling and average response time between WeLe-RAID and DCS5+ with different configurations under online trace in Table VI and Proj_0 trace in Table VII. WeLe-RAID(100 000) means that the predefined threshold of WeLe-RAID is 100 000. The third and fourth columns in the tables validate the above discussions about WeLe-RAID. The results in the last column implies that if WeLe-RAID achieves the same system-level wear-leveling as DCS5+ by configuring the parameters, it must cause a big performance degradation compared with DCS5+.

V. RELATED WORK

We first review the works targeting to reduce the update frequency of parity symbols in SSD-based RAID. HPDA [18] is designed for SSD-based RAID-4. It uses one HDD as the parity disk to absorb all parity writes and uses another HDD as a mirror to absorb small writes. Zeng *et al.* [28] proposed a similar idea and design HRAID6ML for SSD-based RAID-6. This two works both use different devices to reduce parity writes into SSDs, but they do not achieve system-level wear-leveling, while we aim to balance parity writes among SSDs.

There are some works [12], [15], [16] proposing to use a buffer, e.g., NVRAM, to delay parity updates so to reduce the writes into SSDs. However, they are designed for single SSDs while our work focuses on SSD-based RAID. Therefore, those techniques are orthogonal to our work, and so they can also be deployed with our scheme together.

We have already reviewed the related studies [21], [27] working on wear-leveling in SSD-based RAID systems in Section II. In Section IV, we also compare our work with WeLe-RAID[27], which has the same design goal with our work.

Diff-RAID [2] addresses the reliability of SSD-based RAID arrays from a different aspect. It places parities unevenly across SSDs and makes SSDs wear at different aging rate via parity reshuffle at each device replacement so as to avoid correlated failures. This method introduces overhead because of the parity reshuffle in each device replacement, and frequently replacing worn out devices may also degrade system performance. Warped mirrors [29] induces a slight imbalance into write traffic across SSDs to avoid correlated failures for SSD mirror-RAID systems. Li *et al.* [17] propose an analytical model to quantify the reliability dynamics of SSD-based RAID-5 arrays, and study how different parity distributions influence the reliability of SSD-based RAID-5 arrays.

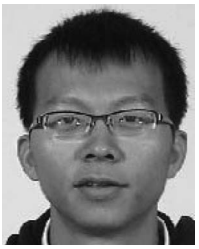
VI. CONCLUSION

Because of the nonuniformity of workloads and the updating dependency between data symbols and parity symbols, the number of writes to each SSD in an RAID configuration varies significantly, which makes some SSDs wear out very quickly and finally decreases the endurance of SSD-based arrays. In this paper, we propose a new DCS, which can efficiently balance the writes to different SSDs and increase system-level wear-leveling. Based on DCS, we design DCS5 for RAID-5 systems and DCS-RS for RS-based RAID systems, respectively. We also discuss about the DCS for different codes in RAID-6 arrays. However, DCS requires the number of writes to the data symbols in the same stripe to be equal, which may not get satisfied under all workloads. To relax the constraint of DCS, we further proposed an enhanced coding scheme, DCS+, which can improve both the endurance and the I/O performance of SSD-based RAID arrays under general access patterns. Trace-driven experiments using real-world workloads show that our solution achieves better system-level wear-leveling and thus better endurance than traditional coding schemes for SSD-based RAID systems. Besides, our coding scheme has the same coding complexity as traditional codes, and it also reduces the user response time.

REFERENCES

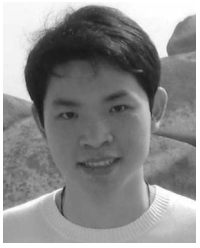
[1] N. Agrawal *et al.*, "Design tradeoffs for SSD performance," in *Proc. ATC*, Boston, MA, USA, 2008, pp. 57–70.
[2] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD reliability," *ACM Trans. Stor.*, vol. 6, no. 2, 2010, Art. ID 4.
[3] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.
[4] L.-P. Chang and C.-D. Du, "Design and implementation of an efficient wear-leveling algorithm for solid-state-disk microcontrollers," *ACM Trans. Design Autom. Electron. Syst.*, vol. 15, no. 1, 2009, Art. ID 6.
[5] L.-P. Chang and L.-C. Huang, "A low-cost wear-leveling algorithm for block-mapping solid-state disks," *ACM SIGPLAN Not.*, vol. 46, no. 5, pp. 31–40, 2011.
[6] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 53–65, Jan. 2010.
[7] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, Seattle, WA, USA, 2009, pp. 181–192.

[8] P. Corbett *et al.*, "Row-diagonal parity for double disk failure correction," in *Proc. FAST*, San Francisco, CA, USA, 2004, pp. 1–14.
[9] M. Gómez and V. Santonja, "Characterizing temporal locality in I/O workload," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst.*, San Diego, CA, USA, 2002, pp. 76–82.
[10] L. M. Grupp *et al.*, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. MICRO*, New York, NY, USA, 2009, pp. 24–33.
[11] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. FAST*, San Jose, CA, USA, 2012, p. 2.
[12] S. Im and D. Shin, "Flash-aware RAID techniques for dependable and high-performance flash memory SSD," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 80–92, Jan. 2011.
[13] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger. *The DiskSim Simulation Environment (v4.0)*. [Online]. Available: http://www.pdl.cmu.edu/DiskSim/, accessed Dec. 2015.
[14] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee, "A group-based wear-leveling algorithm for large-capacity flash memory storage systems," in *Proc. CASES*, Salzburg, Austria, 2007, pp. 160–164.
[15] S. Lee, B. Lee, K. Koh, and H. Bahn, "A lifespan-aware reliability scheme for RAID-based flash storage," in *Proc. ACM Symp. Appl. Comput.*, Taichung, Taiwan, 2011, pp. 374–379.
[16] Y. Lee, S. Jung, and Y. H. Song, "FRA: A flash-aware redundancy array of flash storage devices," in *Proc. 7th IEEE/ACM Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Grenoble, France, 2009, pp. 163–172.
[17] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Stochastic analysis on RAID reliability for solid-state drives," in *Proc. SRDS*, Braga, Portugal, 2013, pp. 71–80.
[18] B. Mao *et al.*, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Trans. Stor.*, vol. 8, no. 1, 2012, Art. ID 4.
[19] N. Mielke *et al.*, "Bit error rate in NAND flash memories," in *Proc. IEEE Int. Rel. Phys. Symp.*, Phoenix, AZ, USA, 2008, pp. 9–19.
[20] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage, *ACM Trans. Stor.*, vol. 4, no. 3, 2008, Art. ID 10.
[21] K. Park *et al.*, "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," in *Proc. CIT*, Icheon, Korea, 2009, pp. 140–145.
[22] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. SIGMOD*, Chicago, IL, USA, 1988, pp. 109–116.
[23] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, 1960.
[24] *Storage Performance Council*. (2002). [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[25] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "SRCMap: Energy proportional storage using dynamic consolidation," in *Proc. FAST*, San Jose, CA, USA, 2010, pp. 267–280.
[26] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.
[27] D. Yimo, L. Fang, C. Zhiguang, and M. Xin, "WeLe-RAID: A SSD-based RAID for system endurance and performance," in *Network and Parallel Computing*. Berlin, Germany: Springer, 2011, pp. 248–262.
[28] L. Zeng *et al.*, "HRAID6ML: A hybrid RAID6 storage architecture with mirrored logging," in *Proc. MSST*, San Diego, CA, USA, 2012, pp. 1–6.
[29] Y. Zhang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Warped mirrors for flash," in *Proc. MSST*, Long Beach, CA, USA, 2013, pp. 1–12.



Yubiao Pan received the B.S. degree from the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China, in 2010, where he is currently pursuing the Ph.D. degree.

His current research interests include solid-state devices, distributed storage system, and data deduplication.



Yongkun Li (M'13) received the B.Eng. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2008 and the Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2012.

He is currently an Associate Researcher with the School of Computer Science and Technology, University of Science and Technology of China. He was a Post-Doctoral Fellow with the Institute of Network Coding, Chinese University of Hong Kong. His current research interests include performance evaluation of networking and storage systems.

Yinlong Xu received the B.S. degree in mathematics from Peking University, Beijing, China, in 1983, and the M.S. and Ph.D. degrees in computer science from the University of Science and Technology of China (USTC), Hefei, China, in 1989 and 2004, respectively.

He was an Assistant Professor, a Lecturer, and an Associate Professor with the Department of Computer Science and Technology, USTC. He is currently a Professor with the School of Computer Science and Technology, USTC. He is currently leading a group of research students in doing some networking and high performance computing research. His current research interests include network coding, wireless network, combinatorial optimization, design and analysis of parallel algorithm, and parallel programming tools.

Prof. Xu was a recipient of the Excellent Ph.D. Advisor Award of Chinese Academy of Sciences in 2006.

Biaobiao Shen received the bachelor's degree in computer science from the Anhui University of Technology, Ma'anshan, China, in 2013. He is currently pursuing the master's degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.

His current research interests include storage system, especially NAND flash SSDs-based storage systems and distributed systems.