A Modeling Framework for Reliability of Erasure Codes in SSD Arrays

Mostafa Kishani, Saba Ahmadian, and Hossein Asadi*, Senior Member, IEEE

Data Storage, Networks, & Processing (DSN) Lab, Department of Computer Engineering

Sharif University of Technology

Abstract—Emergence of *Solid-State Drives* (SSDs) have evolved the data storage industry where they are rapidly replacing *Hard Disk Drives* (HDDs) due to their superiority in performance and power. Meanwhile, SSDs have reliability issues due to bit errors, bad blocks, and bad chips. To help reliability, *Redundant Array of Independent Disks* (RAID) configurations, originally proposed to increase both performance and reliability of HDDs, are also applied to SSD arrays. However, the conventional reliability models of HDD RAID cannot be intactly applied to SSD arrays, as the nature of failures in SSDs are totally different from HDDs. Previous studies on the reliability of SSD arrays are based on the deprecated SSD failure data, and *only* focus on limited failure types, device failures, and page failures caused by the bit errors, while recent field studies have reported other failure types including bad blocks and bad chips, and a high correlation between failures.

In this paper, we investigate the reliability of SSD arrays using field storage traces and real-system implementation of conventional and emerging erasure codes. The reliability is evaluated by statistical fault injection experiments that post-process the usage logs obtained from the real-system implementation, while the fault/failure attributes are obtained from the state-of-the-art field data by previous works. As a case study, we examine conventional RAID5 and RAID6 and emerging *Partial-MDS* (PMDS) codes, *Sector-Disk* (SD) codes, and *STAIR* codes in terms of both reliability and performance using an open-source software RAID controller, MD (in Linux kernel version 3.10.0-327), and arrays of Samsung 850 Pro SSDs.

Our detailed analysis on the data loss breakdown shows that a) emerging erasure codes fail to replace RAID6 in terms of reliability, b) row-wise erasure codes are the most efficient choices for contemporary SSD devices, and c) previous models overestimate the SSD array reliability by up to six orders of magnitude, as they just focus on the coincidence of bad pages (bit errors) and bad chips within a data stripe that holds the minority of root cause of data loss in SSD arrays. Our experiments show that the combination of bad chips with bad blocks is recognized as the major source of data loss in RAID5 and emerging codes (contributing more than 54% and 90% of data loss in RAID5 and emerging codes, respectively), while RAID6 remains robust under these failure combinations. Finally, the fault injection results reveal that SSD array reliability, as well as the failure breakdown is significantly correlated with SSD type.

I. Introduction

Solid-State Drives (SSDs) are predicted to replace *Hard Disk Drives* (HDDs) due to their performance and power consumption benefits [1]. While a big spectrum of *Non-Volatile Memory* (NVM) technologies are appeared in the recent years and struggle to find their place in industry [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], SSDs are still the most matured and promising high-performance storage devices. SSDs are intensively used a) as the main storage media in all-flash storage systems, b) as a caching media in *Input/Output* (I/O) cache layer [22], [23], [24], [22], [25], [26], [27], [28], and c) for tiering purposes [29], [30], [31] (Fig. 1). Meanwhile, SSDs have reliability issues due to wear-out[1], bit errors, bad blocks, and bad chips. These reliability issues can increase the chance of data unavailability and data loss in storage systems [32], [33], [34], [35].

To enhance reliability, *Redundant Array of Independent Disks* (RAID) configurations [36] are employed in storage systems to avoid data loss and data unavailability. However, the nature of failures and errors in SSDs are totally different from HDDs [37], [38], [39], [40]; SSDs have increasing *Bit Error Rate* (BER) with a distribution different from HDDs and there is a high correlation between bit errors in SSDs [37], [38], [39]. Due
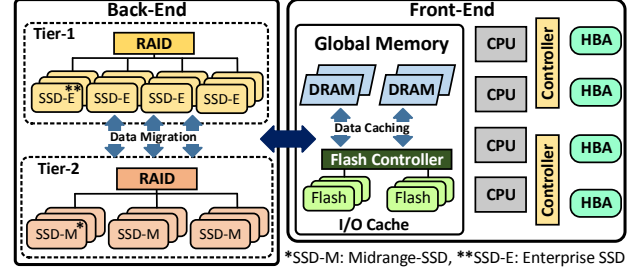

Fig. 1: Structure of an enterprise flash-based storage system.

to these differences, conventional reliability models of HDD arrays cannot be applied intactly to SSDs. Previous studies on the reliability of SSD arrays [41], [42], [43], [44], [45], [46], however, are based on old SSD failure studies [47], [48], [49], and just focus on a single failure type, page failures caused by bit errors, while recent field studies have reported other failure types including bad blocks and bad chips alongside page failures, and a high correlation between these failure types.

A recent work by Schreoder et al. [37] reports SSD failure data in Google datacenters and shows that alongside bit errors that result in the loss of one page of data, device failures, including bad blocks and bad chips also affect the reliability of SSD arrays, despite previous work that just reports *Raw Bit Error Rate* (RBER)[2] and *Uncorrectable Bit Error Rate* (UBER). This study also reports a high correlation between RBER and parameters such as prior *Program/Erase* (P/E) cycles, and a high correlation between bad chip and total number of bad blocks in a SSD chip. Meza et al. [38] and Narayanan et al. [39] have also reported SSD failure field data in the recent years. The reports of all the mentioned works contradict the old SSD failure studies [47], [48], [49], hence, discredit existing reliability models that are based on those data. Many studies try to model the reliability of SSD arrays and modify RAID configurations in favor of SSD failure characteristics [41], [42], [43], [44], [50], [51]. These works, however, come with very trivial or no reliability estimations, *or* are based on misleading SSD failure characteristics reported by old studies [47], [48], [49]. To the best of our knowledge, *none* of previous studies have modeled SSD array reliability based on valid field data and real-system implementation.

In this paper, we investigate the reliability of SSD arrays using the real-system implementation of conventional RAID and emerging erasure codes. The reliability is evaluated by statistical fault injection experiments that post-process the SSD usage logs obtained from the system run, while the fault/failure attributes are obtained from the state-of-the-art field data by previous works [37]. As a case study, we examine conventional RAID5 and RAID6 and emerging *Partial-MDS* (PMDS) codes [46], *Sector-Disk* (SD) codes [50], and *STAIR* codes [51] in terms of reliability, endurance, and performance. The experiments are conducted using an open-source software-RAID controller, *Multiple Device* (MD), in Linux kernel version 3.10.0-327 (CentOS 7 operating system), and arrays of Samsung 850 Pro SSDs.

---

*Corresponding Author

[1]Each flash cell can tolerate a limited number of writes (erasures) and wears out after a few thousand of erasures, depending on the technology and device variations.

[2]RBER is defined as the number of corrupted bits over the total number of read bits (including both correctable and uncorrectable errors) [37].

Thorough investigation of SSD arrays has revealed the following major observations: 1) Erasure codes mainly suffer from data loss caused by the combination of device failure and block failure. In the RAID5 arrays, bad chips combined with either bad pages or bad blocks are the major sources of data loss. The contribution of bad blocks combined by bad chip (two bad blocks and one bad chip) is also significant in total data loss of RAID6. 2) Unlike previous models which only focus on the coincidence of bad pages and bad chips, our study shows that this type of failure contributes the minority of data loss in SSD arrays. 3) SSD array reliability, as well as failure breakdown is significantly correlated with SSD type. 4) Time to scrub has a significant impact on array reliability, while the impact of time to recover from a device failure is of less significance. 5) RAID5 and RAID6 codes perform almost independent of stripe size. Emerging erasure codes, however, benefit from smaller stripe sizes and show a promising reliability improvement when reducing stripe size.

We can summarize the major contributions of this work as follows:

- We propose an analytic model for the reliability of SSD arrays, considering realistic SSD failure attributes from state-of-the-art studies in the field.
- We propose a generalized fault injection framework for evaluating the reliability of SSD arrays, using SSD usage logs obtained by real-system implementation and SSD failure characteristics obtained by state-of-the-art field data.
- We evaluate the reliability of different erasure codes under an extensive number of representative storage workloads.
- We compare the performance and endurance overhead of different erasure codes using the real storage stack, despite previous works that inadequately compare decode/encode complexity and ignore the endurance and I/O overhead.
- We develop an open-source framework for SSD array fault injection, which will be publicly available for the research community.[3]

The rest of this paper is organized as follows. Section II discusses related work on SSD reliability. Section III presents a background about examined erasure codes. Section IV discusses the proposed modeling framework. Section V presents the experimental setup, results, and the corresponding observations and discussions. Finally, Section VI concludes the paper.

II. Related Work

A. Field Studies on SSD Failure Characteristics

A recent work by Schreoder et al. [37] investigates SSD reliability by collecting six-year SSD failure data in Google datacenters. This study shows that alongside bit errors that result in the loss of one page of data, device failures, including bad blocks and bad chips are also of major reliability threats in SSD arrays, despite previous work that just report RBER and UBER. This study also reports a high correlation between RBER and parameters such as prior P/E cycles, SSD age, read count, write count, erase count, and prior RBER. Another field study by Meza et al. [38] reports that RBER does not monotonically increase with P/E cycles and also reports that RBER has an exponential growth in SSD useful life. However, this study reports a smooth linear increase of RBER with P/E cycles. This study also shows a high correlation between total number of bad blocks in a SSD chip and the number of bad blocks already developed. It also shows that in an over four year mission, more than 30-80% of SSDs experience bad blocks in the field. Another observation of this study is that 2-7% of SSDs experience bad chip within the first four year of their life.

A work by Meza et al. [38] conducts a deep study on the failure characteristics of flash memories using field data from Facebook datacenters. This work observes that SSD failure rate does not increase monotonically with flash chip wear. In turn, SSD failure rate has four phases of early detection, early failure, useful life, and wear-out [38]. Another observation is that UBER obtained in this work is 10 to 1000 times smaller than the raw BER of similar flash chips examined by Grupp et al. [47]. This is due to the fact that SSDs in this work correct small errors, perform wear leveling, and are not at the end of their rated life [38]. Meza et al. show that on average 10% of SSDs experience 80% of all uncorrectable errors, while in most of platforms 10% of SSDs experience 95% of all observed uncorrectable errors. It also shows that during two successive weeks, 98% of SSDs that experienced an error during the first week also had an error during the next week.

Grupp et al. [47] also report BER for *Single Level Cells* (SLC) and *Multi Level Cells* (MLC) flash of different feature size. In another research, Grupp et al. [49] show that BER increases by flash chip wear. However, this work does not consider the effect of optimizations in the SSD controller and buffering layers. Cai et al. [48] also examine the bit error patterns in MLC NAND flash and demonstrate its dependency to P/E cycles, physical location, and value. Finally, Mielke et al. [52] report BER and sector failure of MLC NAND flash in conjunction with P/E cycles, retention time, and number of reads.

B. Analysis and Modeling of SSD Array Reliability

A large body of research has investigated and tried to improve the reliability of disk arrays [45], [53], [54], [55], [56], [57], [58], [59], [60], [35], [61]. For the sake of brevity, here we focus on the studies concentrating on *SSD* arrays. Greenan et al. [45] propose a combination of inter-device and intra-device parity codes to cope with page failures, block failures, and device failures in SSD arrays. While the authors have a realistic assumption about failure types in SSD arrays, their reliability assessment approach is questionable, as it reports *Uncorrectable Page Error Rate* (UPER) using cumulative Binomial distribution as a function of RBER. The proposed method also necessitates the migration of *Flash Translation Layer* (FTL) from device to RAID controller. Hence, the method cannot be employed using *Commercial off-the-Shelf* (COTS) devices.

Balakrishnan et al. [44] propose Differential RAID as an alternative to conventional RAID5 to be applied in SSD arrays. The idea is based upon uneven parity distribution across array devices (in the most intense configuration, RAID5 is transformed to RAID4) to reduce time proximity of wear-out phenomenon in SSDs. This method is examined using statistical fault injections. Li et al. [41] compare RAID5 with Differential RAID [44] in SSD arrays. The work has a mathematical discussion, adopted from [62], to apply a variable failure rate to *Continuous Time Markov Chain* (CTMC) using *Kolmogorov* forward equation, *uniformization* [62], and *truncation*. This work validates the mathematical model by statistical fault injections using Microsoft SSD simulator [63] extended from DiskSim [64], and estimates the reliability as a function of erasures (SSD age). One important shortcoming of this work is considering equal failure rate for all devices using Weibull distribution of SSD bit error rate, and ignoring the correlation of errors.

Kim et al. [65] attempt to improve the reliability of RAID5 in SSD arrays by proposing *Dynamic Striping-RAID* (DS-RAID). This work compares the proposed method with conventional RAID5 in terms of response time and number of write operations, including original data writes and extra writes due to parity and garbage collection, as a representative for SSD lifetime. Finally, Kim et al. [42] propose *Elastic Striping and Anywhere Parity* (eSAP-RAID) as an alternative to RAID5 with higher performance and reliability in SSD arrays. This method tries to reduce the number of writes due to parity updates, by allowing flexible stripe size and parity placement. Both works [65], [42] use Microsoft SSD simulator [63] extended from DiskSim [64].

Moon and Reddy [43] investigate the reliability of RAID0, RAID1, and RAID5 in SSD arrays by considering the effect of garbage collection and show the trade-off between reliability and utilization in a SSD array. This work arguably uses Markov

---

[3]The framework is available in http://dsn.ce.sharif.edu/

TABLE I: Qualitative comparison of proposed framework with different SSD array reliability models by Balakrishnan et al. [44], Blaum et al. [46], Li et al. [41], and Moon and Reddy [43].

	[44]	[46]	[41]	[43]	Proposed
Coincidence of device and symbol failure	✓	✓		✓	✓
SSD age	✓		✓		✓
BER as a function of erasures	✓		✓		✓
Statistical fault injection	✓		✓		✓
Use mathematical reliability analysis		✓	✓	✓	
Emerging erasure codes		✓			✓
Valid SSD failure field data					✓
Correlation between errors					✓
Valid distribution for BER					✓
Real-system implementation					✓

models by considering constant bit error rate for SSDs and ignores the correlation between errors. Blaum et al. [46] propose a new family of erasure codes to cope with the coincidence device failures and symbol (page) failures. The proposed code is evaluated using probabilistic analysis of data loss. In summary, Table I makes a qualitative comparison between different SSD array reliability models and our proposed framework. As the table shows, unlike our proposed framework, previous works do not use real-system implementation and also make use of deprecated SSD failure data.

III. Background

RAID is proposed as a solution to cope with performance and reliability issues of single disks [36]. RAID5 and RAID6 configurations distribute the data to an array of disks while keeping the row-wise parity of devices in respectively one and two redundant devices. Hence, RAID5 and RAID6 can respectively tolerate one and two device failures. We can put both RAID5 and RAID6 in the category of *Maximum Distance Separable* (MDS) codes that offer the maximum correction capability, due to having the maximum hamming distance[4]. Blaum et al. [46] propose PMDS codes to handle the combination of both device failures and symbol (page) failures, by using the combination of row-wise parity and a new concept of *Global Parity* (GP) that is taken across the whole data stripe (Fig. 2).
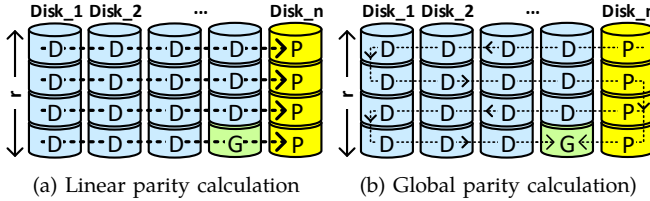

(a) Linear parity calculation   (b) Global parity calculation)

Fig. 2: Structure of a data stripe using $PMDS(1,1)$ code when the number of devices and rows are respectively n and r. Note D, P, and G respectively stand for data symbol, parity symbol, and Global parity symbol.

Fault tolerance of PMDS codes can be specified by m, number of tolerable drive failures (i.e., number of coding drives) and s, number of tolerable sector failures (equal to the number of global parities). For example, $m = 1$ and $s = 1$ says that one drive failure is tolerable while one of operating chunks can tolerate one sector failure. A specific configuration of PMDS codes is capable of tolerating one device and one sector failure, denoted as $PMDS(1,1)$, that is examined in our study. SD codes [50] and STAIR codes [51] also propose different methods for encoding and decoding of global parity with different computational complexities (but the same I/O overhead) [51].

We assume that each data stripe is composed of n devices (or n data chunks), including redundant devices, and r rows, where r stands for the number of symbols from each device in one stripe. Fig. 2 shows the structure of PMDS(1,1) while the data symbols are denoted by D, row-vise parity symbols are denoted by P, and the global parity symbol is denoted

[4]MDS codes have a big spectrum of alternatives such as Reed-Solomon [66], multi-dimensional codes [67], and simple linear hamming codes usually used in fast memory structures [68].

TABLE II: ERF and computation (number of XORs) needed for updating one data stripe of different erasure codes.

	RAID5	RAID6	STAIR(1,1)
XORs	$(n-1) \times r$	$(n-1) \times r \times 2$	$(n-1) \times r \times 2 + r - 1$
ERF	$\frac{n+1}{n}$	$\frac{n+2}{n}$	$\frac{(n+1) \times r}{(n \times r) - 1}$

TABLE III: Write/read operations needed for updating one page, one row, and one data stripe. W stands for the write operation, and R stands for read before write.

	RAID5	RAID6	PMDS(1,1)
Sector Update	W=2	W=3	W=4
	R=2	R=3	R=4
Row Update	W=$n+1$	W=$n+2$	W=$n+3$
	R=0	R=0	R=$n+2$
Stripe Update	W=$(n+1) \times r$	W=$(n+2) \times r$	W=$(n+1) \times r$
	R=0	R=0	R=0

by G. PMDS codes, SD codes and STAIR codes are systematic (separable) codes with homomorphic property. This property enables updating the codeword when the data is partially updated by an approach similar to updating normal parity bits, as shown in Equation 1.

$$Codeword_{new} = Codeword_{old} \oplus Data_{old} \oplus Data_{new} \quad (1)$$

Hence, PMDS(1,1), SD(1,1), and STAIR(1,1) codes perform similar in terms of I/O penalty, but are different in encoding/decoding computational complexity [51]. However, as encoding/decoding computation time is negligible compared to I/O penalty, in the rest of this work we note PMDS(1,1), SD(1,1), and STAIR(1,1) codes by PMDS(1,1) or simply PMDS. Table II shows the *Effective Replication Factor* (ERF) and computations (XORs) needed for encoding one stripe of RAID5, RAID6, and PMDS (discussed in detail by Li and Lee [51]). We also compare the I/O penalty of erasure codes in Table III. This table shows that in the case of sector and row update, PMDS has more number of write/read compared to RAID5 and RAID6, while in the case of stripe update both RAID5 and PMDS have an equal overhead lower than the overhead of RAID6. This analysis shows that in sequential workloads dominated by full stripe writes, we can expect a greater performance from PMDS, compared to RAID6. We further verify this hypothesis by examining different realistic workloads.

IV. Modeling Reliability of RAID5, RAID6, and PMDS Codes in SSD arrays

We model the reliability of SSD array for different erasure codes, by proposing a fault injection environment that uses the field data of SSD failure statistics from Schreoder et al. [37], alongside SSD operation log from arrays of Samsung 850 Pro SSDs, using the open-source software RAID controller MD in Linux kernel version 3.10.0-327. We consider three possible failure types in SSD arrays, reported by field studies [37], including *Bad Page*, *Bad Block*, and *Bad Chip*.

- **Bad Page (**BP**)** or **Bad Symbol (**BS**)** is the most prevalent failure type in an SSD array, caused by uncorrectable bit errors in SSD device. As in the storage systems, data is logically read/written/managed in the units of pages, the page is considered lost when it contains uncorrectable corrupted bits. We call this failure *Bad Page* or *Bad Symbol*, as the page is the smallest data symbol that different erasure codes are performed on. A bad symbol can result in the loss of one data stripe, if it is not correctable by the employed erasure code. Note bit errors that are correctable by the internal *Error Correction Code* (ECC) of the SSD device are not considered as bad symbol.
- **Bad Block (**BB**)** is reported by field studies as another common failure type in SSD arrays [37]. As each block contains multiple (tens or hundreds) pages, uncorrectable bad blocks can affect multiple data stripes in the SSD array, depending on the data striping protocol.
- **Bad Chip (**BC**)** is the last type of failure in the SSD arrays, reported by field studies [37]. Bad chip can result in the
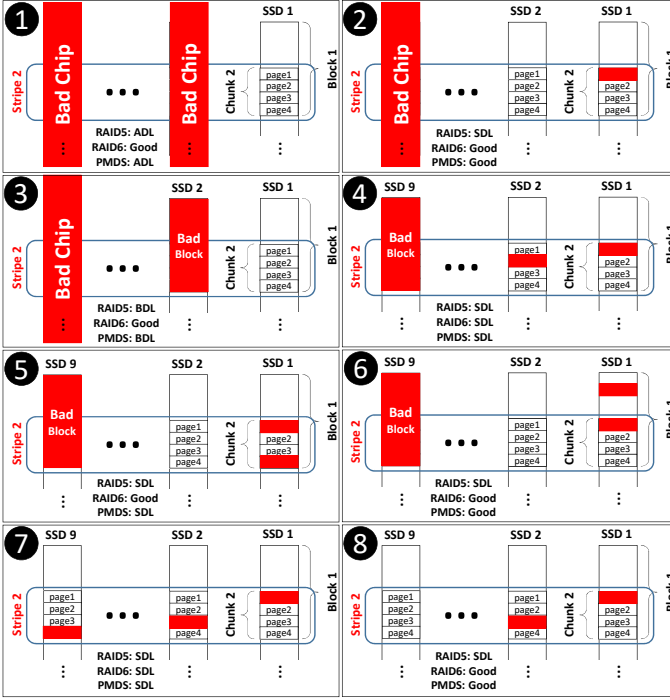
Fig. 3: Failure samples in different SSD arrays.

loss of whole array, for example when two bad chips on two different array devices coincide in the case of RAID5. It also can result in the loss of one or multiple data stripes, when it coincides with a bad symbol or bad block in another array device in the case of RAID5.

A. Correction Capability of RAID5, RAID6, and PMDS

Fig. 3 shows eight examples of the combination of bad symbol, bad block, and bad chip in an SSD array. In this figure, *Array Data Loss* (ADL) stands for the loss of whole SSD array, *Block Data Loss* (BDL) stands for the loss of all data stripes a block is shared upon, *Stripe Data Loss* (SDL) stands for the loss of one data stripe and *Good* stands for no data loss. In this figure, we consider a fully striped SSD array, in which a stripe is composed of data chunks from all SSD devices. Without loss of generality, here we assume each stripe contains the data from one chip of each SSD device. Taking other assumptions may affect the magnitude of data loss upon failure incidence. Each data chunk contains multiple pages (4 in this example) and each block is shared upon multiple stripes (2 in this example). We should note that erasure codes are performed on the stripe unit, hence, the uncorrectable loss of a single page is considered as the loss of whole stripe (SDL).

In example ❶, coincidence of two bad chips result in ADL in RAID5 and PMDS while it is recoverable in the case of RAID6. In example ❷, the combination of one bad chip and one bad symbol is correctable in the case of RAID6 and PMDS, but it results in SDL in RAID5. The combination of bad chip and bad block in example ❸ results in BDL in the case of RAID5 and PMDS, as bad block affects the entire data chunk rather than one symbol. In example ❹, all erasure codes face SDL, as three data chunks are corrupted. In example ❺, RAID5 experiences SDL due to the combination of a bad block and a bad symbol. PMDS also experiences SDL in example ❺, as it cannot tolerate more than one symbol failure coincided with a bad chip. Example ❻ is similar to example ❺, however, in this case PMDS can correct the failure incidence, as two symbol failures occur in two different stripes. The coincidence of three bit errors in three different symbols in example ❼ also results in SDL in all erasure codes. Finally, in example ❽ where two bit errors in different symbols coincide, PMDS and RAID6 can correct the failure, but RAID5 experiences SDL as it cannot tolerate multiple symbol failures in multiples chunks of a single

stripe. Here we can conclude that PMDS can tolerate multiple symbol failures in one data chunk alongside a single symbol failure in another data chunk. RAID6 can tolerate multiple symbol failures in two data chunks, and RAID5 can tolerate multiple symbol failures in one data chunk.

B. Analysis of RAID5, RAID6, and PMDS Reliability

Fig. 4 shows the state diagram of SSD array reliability, using different erasure codes, from error-free operation to failure incidence (ADL, BDL, and SDL). This analysis is used in our statistical fault injections to evaluate the reliability of different erasure codes. Field studies show that the failure characteristics of a SSD will change when it wears out, i.e., it passes its P/E Limit or *Wear Out Limit* (WOL) [37].

Storage systems may have different regulations when they face a worn-out SSD, such as replacing the SSD or continuing its operation up to the failure. However, in this work we model the most conservative assumption that worn-out SSD is replaced with a brand-new one. Replacing the worn-out SSD is also performed with two different regulations: 1) The first regulation removes the worn-out SSD, immediately replaces it with the brand-new SSD, and reconstructs the data of worn-out SSD on the brand-new one using the parity of the other SSDs. This approach, however, may result in data loss when a bad symbol exists in other operating SSDs. The reason is that the data of those stripes containing bad symbols cannot be reconstructed when the worn-out device is removed (and its data is unavailable) in the case of RAID5. 2) An alternative approach that prevents this data loss case is adding the brand-new SSD when the worn-out SSD is still operational, making a RAID1 configuration between the brand-new and worn-out SSD, waiting for all data of worn-out SSD to be copied into the brand-new one, and finally removing the worn-out SSD. In this study, for the sake of reliability we take the second approach.

1) RAID5: Fig. 4a shows the state diagram of RAID5 SSD array reliability. In the OP state, none of SSDs have bad symbol, bad block, or bad chip. When a SSD device wears out, the array moves to the OP_{WO} state in which one (or multiple) SSD device is worn out and is waiting to be replaced with brand-new one. We dedicate OP_{WO} state from OP state as field studies show that SSD failure characteristics change after wear out [37]. If we neglect this change, OP and OP_{WO} states can be merged (the same happens to other WO states).

Upon a bad chip in OP and OP_{WO} states, the array moves to EXP_BC state. In this state (and also EXP_BC_{WO} state when the array has worn out SSDs), the array moves back to operational state when the failed device is replaced and reconstructed on the brand-new one. However, any successive bad chip, bad block, and bad symbol results in data loss and moves the array to ADL, BDL, and SDL states, respectively. An operational array (in either OP or OP_{WO} states) moves to EXP_BB state when it faces a bad block. In this state (and also EXP_BB_{WO} state when the array has worn out SSDs), a successive bad chip (in another device) and coincidence of another bad block in the same stripe, called *Same Stripe Bad Block* (SSBB), results in BDL. Moreover, coincidence of a symbol failure in the same stripe, called *Same Stripe Bad Symbol* (SSBS), results in SDL. In EXP_BB and EXP_BB_{WO} states, the chip containing bad block is prone to fail (bad chip) that moves the array to either EXP_BC or EXP_BC_{WO} states, when no other chip contains bad block or bad symbol. In EXP_BB and EXP_BB_{WO}, bad block is detected by a final read error, write error, or erase error [37], and removed by reallocating the corrupted block to a safe block that moves the array back to operational state.

Finally, an operational array (in OP or OP_{WO} states) moves to EXP_BS state when it faces a bad symbol. In EXP_BS state (and also EXP_BS_{WO} state when the array has worn out SSDs), a successive bad chip, SSBB, and SSBS results in stripe data loss and moves the array to SDL state. However, in EXP_BS and EXP_BS_{WO} states, bad symbol can be detected by read error or scrubbing [69] and be removed by reconstructing the data from parity of other devices, that moves the array to the operational state.
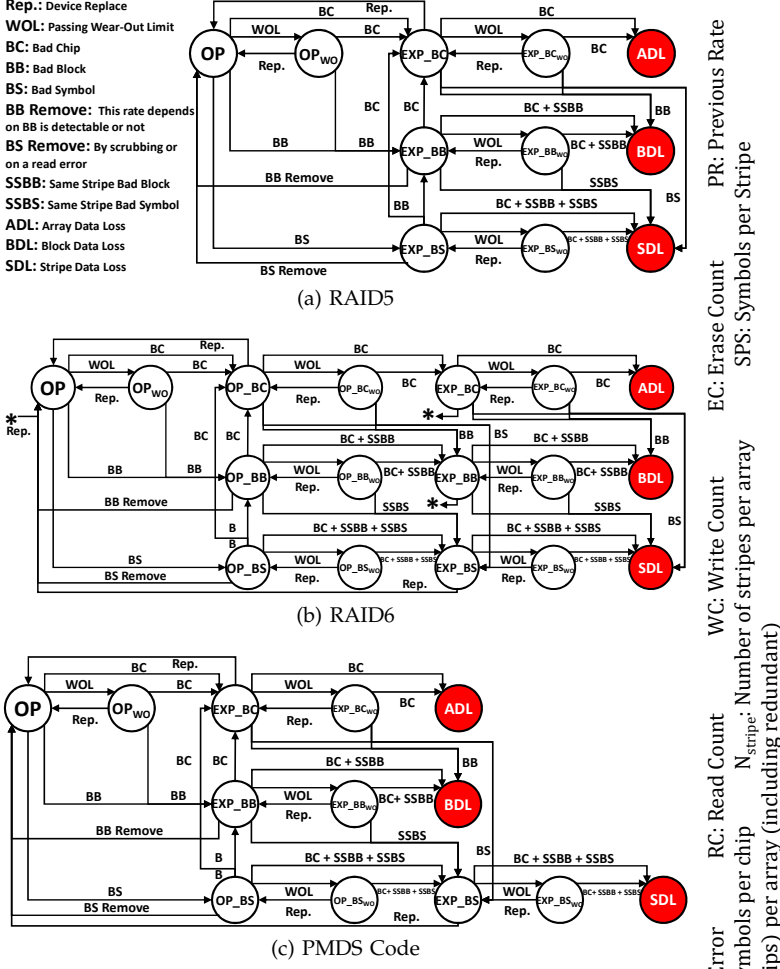
Fig. 4: State diagram of SSD array reliability for RAID5, RAID6, and PMDS code.

2) RAID6: Fig. 4b shows the state diagram of RAID6 SSD array reliability. As the figure shows, RAID6 can tolerate an extra failure compared to RAID5, due to having two redundant devices. The description of states and transitions is similar to RAID5. The only difference is six states (OP_BC, OP_BB, OP_BS, OP_BC_{WO}, OP_BB_{WO}, and OP_BS_{WO} states) added to RAID6 diagram. These states have the same transitions as EXP states in RAID5 diagram, by this difference that a successive failure in OP state moves the array to EXP state rather than data loss state.

3) PMDS: Fig. 4c shows the state diagram of SSD array reliability when employing PMDS code. As the figure shows, PMDS performs the same as RAID5 in the case two bad chips coincide (resulting ADL) and the case one bad chip coincides with bad block (BDL). The difference is that PMDS can handle the coincidence of bad chip with bad symbols, and coincidence of two bad symbols in one stripe. Our fault injection experiments using real failure statistics from the field (in Section V) show that this feature of PMDS code can dramatically decrease the number of data loss events compared to RAID5, at a performance overhead and negligible space overhead.

C. Statistical Fault Injection Environment

Fault injection can be implemented by injecting faults on SSD simulator (such as DiskSim [64]) in the runtime, as previous work [41] does. That approach, however, is very time-consuming as SSD simulators have much complexities and are very slow. We take another approach that extracts SSD usage log, including the number of reads, writes, erases, and P/E cycles from a real system (or simulator), and post-processes
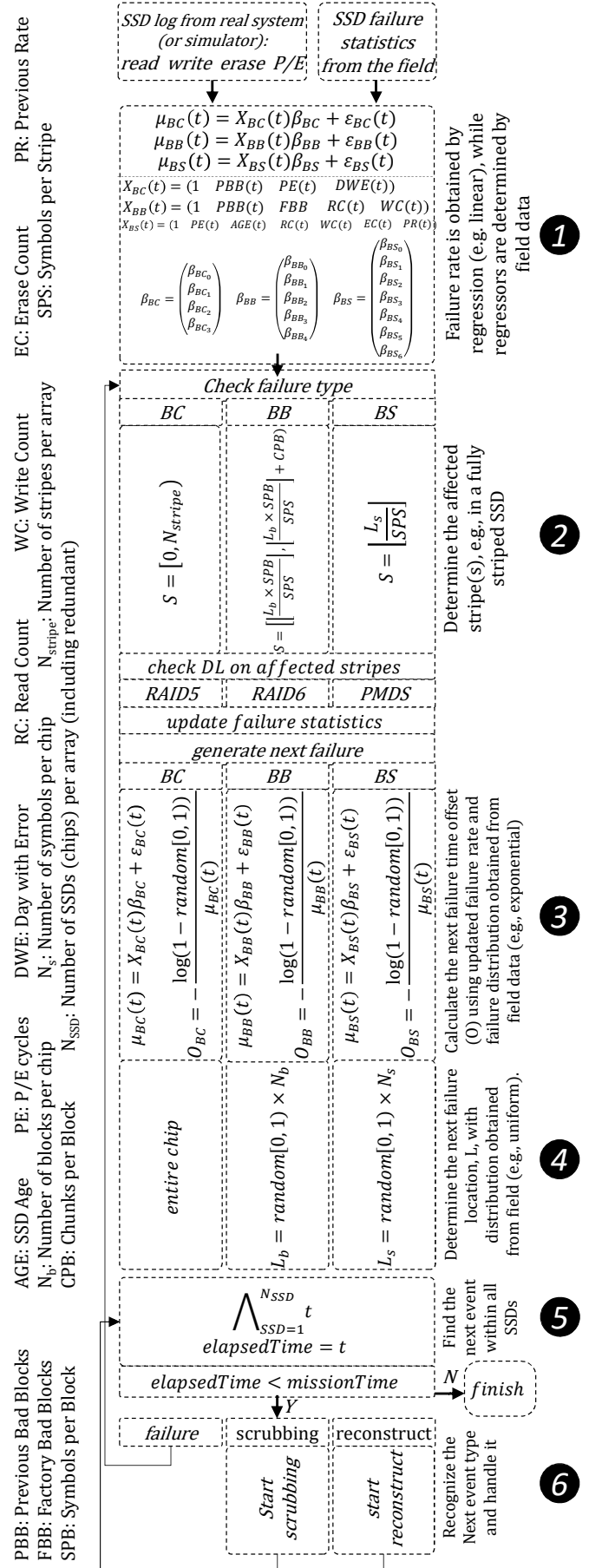

Fig. 5: Statistical fault injection environment.

this information to perform fault injection experiments. This approach has two advantages: **a)** It is very fast. **b)** We can use SSD usage logs from the real systems, rather than simulators, to obtain more realistic results. Accordingly, our failure model has two phases: a) *Capturing SSD Logs* and b) *Statistical Fault Injection* in respect with SSD failure statistics from the field and SSD usage logs obtained by our real-system run.

1) Capturing SSD Logs: In the phase of capturing SSD log, we have different benchmarks running on the desired array configuration. What we capture from benchmark running is the number of P/E cycles as well as write/read accesses. In specific, P/E cycles are calculated by capturing *Wear Leveling Count* parameter from S.M.A.R.T [70], before and after the benchmark run, following the instructions of SSD vendors. The number of writes confirmed to the SSD is also calculated by capturing *Total LBAs Written* from S.M.A.R.T [70], before and after the benchmark run. For each individual SSD, we also extract the full details of read/write request type, destination, size, and issue time using Blktrace [71].

2) Statistical Fault Injection: Fig. 5 shows the flowchart of our fault injection framework. We use the field data of SSD failure statistics from Schroeder et al. [37], alongside SSD operation log to dynamically evaluate the rate of bad chip, bad block, and bad symbol, per each individual chip. Schroeder et al. [37] show that failure (bad chip, bad block, and bad symbol) rate at time t highly correlates with parameters such as *Read Count* (RC), *Write Count* (WC), *Erase Count* (EC), *Previous Bad Blocks* (PBB), *Previous bit error Rate* (PR), *Days With Error* (DWE), and number of *Factory Bad Blocks* (FBB).[5] Hence, the failure rates (bad chip, bad block, and bad symbol rates) are evaluated by regression (e.g., linear regression) from the mentioned factors, while the *Parameter Vector*, β, should be determined by the field data. Accordingly, the rate of bad chip, $\mu_{BC}(t)$, is calculated as shown in Equation 2 (considering linear regression).

$$\mu_{BC}(t) = X_{BC}(t) \times \beta_{BC} + \epsilon_{BC}(t)$$
$$X_{BC}(t) = \begin{bmatrix} 1 & PBB(t) & PE(t) & DWE(t) \end{bmatrix}$$
$$\beta_{BC} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} \quad (2)$$

Where $X_{BC}(t)$ is the vector of regressors determined by field data, $\beta_{BC}(t)$ is the parameter vector (also determined by field data), and $\epsilon_{BC}(t)$ is the error vector. $\mu_{BB}(t)$ and $\mu_{BS}(t)$ are also calculated by the similar equations shown in Fig. 5. Note that Schroeder et al. [37] have reported a limited failure data including RBER as a function of EC, percentage of drives with bad blocks, median number of bad blocks, mean number of bad blocks, and percentage of drives with bad chips. We determine the failure rates by using the mentioned data. Fig. 7 and Table VII summarize the employed failure statistics by Schroeder et al. [37].

In the fault injection phase, the desired array of SSDs is constructed and for each individual SSD, the P/E cycles and accesses are imported from SSD logs captured by benchmark run. For each SSD, the rate of BS ($\mu_{BS}(t)$) is updated in specific time intervals, regarding P/E cycles at time t, following the field data appeared in Table VII and Fig. 7. The other two parameters of BB and BC are also considered for each individual SSD device. About BC, we have the rate of drives with bad chip (in a four-years mission) from the field [37]. The available detail on how bad chips correlate bad blocks is limited to the fact that 2/3 of all bad chips appear in those chips that have more than 5% of their blocks failed. We consider this correlation in our experiments by creating a pool of SSDs at the beginning of experiments, following the failure statistics reported by the field data.

Create SSD Pool: We build a pool of SSDs (in our experiments, 10,000 SSDs) regarding the failure attributes reported by [37]. In the next step, we construct the disk array

by randomly selecting n SSDs (e.g., 8 SSDs for an array of RAID5(7+1)) from the SSD pool. Following we describe how bad chip and bad block is considered in the failure model.

Bad Chip: At the start of fault injection experiments, the SSD pool is created in a way the BC and BB statistics conforms the field data reported by [37] (detailed results on how the BB and BC of SSDs in the failure model statistically conforms the field data are appeared in Section V-C). When the SSD pool is created, some chips are marked as to be failed within mission time. In the SSD array, constructed by randomly choosing n out of 10,000 SSDs in the pool, if a chip is marked to encounter bad chip, it is failed within mission time. No data is provided by [37] about the time distribution of BC, so we consider exponential distribution, following the conventional assumption on the time to failure of semiconductors. Following we describe how the correlation between BC and BB is considered in the failure mode.

Bad Block: The distribution of the number of mission-time bad blocks in the SSD population is not reported by field data. The field data only reports that the number of *factory* bad blocks is close to *Normal* distribution in the population of SSDs under study [37]. Hence, in creating the SSD pool we consider the number of mission-time bad blocks also follows the normal distribution. From the field data, we also have the percentage of drives with bad blocks, median number of bad blocks for drives having bad block, and mean number of bad blocks for drives having bad block. We create the SSD pool to conform the mentioned statistics obtained from the field, as shown in Section V-C. The field data also reports a correlation between bad chip and bad block [37]. Based on the field results, 2/3 of all bad chips happen in those chips that have more than 5% of all their blocks failed. We consider this correlation between BB and BC in creating the SSD pool, as shown in Section V-C. The last correlation reported by the field study is the correlation of BB with previous BB [37]. The field study reports the median number of bad blocks a drive will experience within mission time, as a function of number of bad blocks already experienced. We also consider this correlation in creating the SSD pool by increasing BB probability in those chips that have experienced a specific threshold of BB, as verified in Section V-C. Regarding the mentioned statistical attributes, we determine the occurrence of bad chip, as well as the number of bad blocks for each SSD in the SSD pool.

Constructing SSD Array and Starting Fault Injection: In the next step, we construct the SSD array by randomly choosing n out of 10,000 SSDs from the pool. For each SSD, we also have P/E cycles as well as accesses from benchmark run on the real system. As the fault injection time, t, passes, for each SSD the $RBER$ is updated regarding P/E cycles at time t, following the data appeared in [37] (Fig. 7). For time intervals of one hour, we estimate the number of BS by multiplying $RBER$ to the number of accessed bits. No information on the spatial characteristics of BS is reported by the field data, so we consider uniform distribution for BS. The time distribution of BS, BB, and BC is not clarified by the field data, so we consider exponential distribution for time to failure, following the conventional assumption on the time to failure of semiconductor devices.

At the beginning of fault-injection, the simulator is initiated by the first bad chip, bad block and bad symbol for each chip, and the next failure (the failure having minimum t) is issued to the failure handling queue. Thereafter, the simulator recognizes the failure type and determines the affected sectors, regarding the failure type and location. As we consider a fully-striped architecture for SSD array, in the case of bad chips all stripes are affected (stripes 0 to N_{stripe}, as shown in Fig. 5). In the case of bad blocks, the number of affected stripes is equal to *Chunks Per Block* (CPB) that depends on the array architecture, including block size, stripe size, and number of devices. The index of the first affected stripe is $\lfloor \frac{L_b \times SPB}{SPS} \rfloor$, where L_b is the location (index) of affected block, SPB is the number of symbols per block, and SPS is the number of symbols per stripe. In the case of bad symbols, a single stripe is affected which index is

---

[5]Bad blocks already exist on a brand-new SSD chip [37].

$\lfloor \frac{L_s}{SPS} \rfloor$, where L_s is the location (index) of the affected symbol and SPS is the number of symbols per stripe.

Afterwards, the simulator needs to check if any of the previous errors in the affected sectors have already removed.[6] In the next step, based on the analysis described in Section IV-B, the simulator checks data loss (ADL, BDL, and SDL) on the affected stripes, regarding the employed erasure code (RAID5, RAID6, or PMDS) and updates the failure statistics.

After handling a failure, it is needed to generate the next failure of that type. For example, after handling a bad symbol on chip c, it is needed to generate the next bad symbol incidence for chip c. To this end, the simulator determines the next failure time offset, O, using the dynamically evaluated failure rate, μ. Considering time-to-failure follows exponential distribution, the time offset of the next bad chip, O_{BC} is recognized as shown in Equation 3.

$$O_{BC} = \frac{log(1 - random[0,1))}{\mu_{BC}(t)} \qquad (3)$$

Where $random[0,1)$ is a uniformly generated random number between 0 and 1, and $\mu_{BC}(t)$ is the rate of bad chip at time t, dynamically evaluated using Equation 2. The time offset of the next bad block, O_{BB}, as well as the time offset of the next bad symbol, O_{BS}, are also calculated with the similar equations, as shown in Fig. 5. Thereafter, the simulator determines the location of failure (in the case of bad symbol and bad block), regarding the number of symbols and number of blocks per chip, with predefined distribution obtained from the field (e.g., uniform distribution within a single chip). Accordingly, the location of the next bad symbol event, L_s is determined as shown in Equation 4.

$$L_s = random[0,1) \times N_s \qquad (4)$$

Where N_s is the number of symbols per array. The location of the next bad block event, L_b, is determined by a similar equation, as shown in Fig. 5. In the next step, the next event (failure, scrubbing, or reconstruct) within all chips is issued for handling, and the simulation time is set to the next event time (the event with the minimum time offset, as shown in Fig. 5). If the total mission time is already passed, the simulation finishes. Otherwise, in the case of *reconstruct*, the reconstruction is starting, the possible data loss detected in the reconstruction process is collected, and the replaced SSD statistics is initiated. In the case of *scrubbing*, the possible data loss detected in the scrubbing process is collected and the correctable errors is removed. Finally, in the case the next event is *failure* the simulator turns back to the state of checking the failure type.

V. Results and Observations

In this section, we evaluate the reliability and performance of RAID5, RAID6, and PMDS array configurations using the test platform depicted in Fig. 6. We examine realistic application workloads on the SSD arrays under a real platform and track the block layer I/O traces as well as SSD usage statistics provided by S.M.A.R.T. The performance of different array configurations is collected from workload execution on the real platform, while the array reliability is obtained from our fault injection framework (presented in Section IV-C) by post-processing the SSD usage logs. In the following we first elaborate the details of test platform and examined SSDs. Afterwards, we provide the experimental results.

A. Experimental Setup

Our test platform is composed of *real-system* and *fault injection environment*. In the real-system part of the platform, we use an open-source software RAID controller, MD driver in

---

[6]Previous bad blocks and bad sectors are possibly detected after a read error, write error, or erase error, and removed by reallocating in the case of bad blocks, and rewriting in the case of bad symbol. Scrubbing and SSD reconstruct also remove the errors, but these tasks are handled in another procedures.
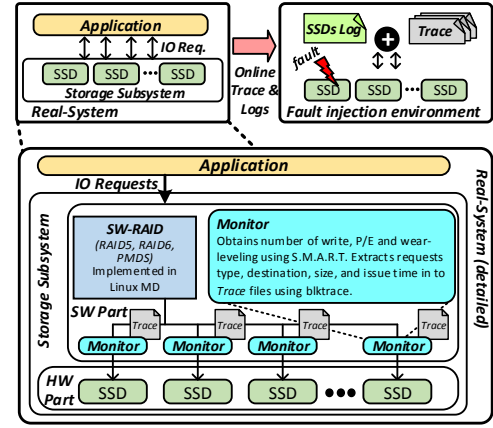
Fig. 6: The structure of test platform used in experiments.

TABLE IV: Hardware and software stack of the real-system part of our test platform, responsible for examining the performance of SSD array configurations and collecting SSD usage logs.

Software	
OS	CentOS 7
Kernel	3.10.0-327
SW RAID controller	*Multiple Device (MD) driver*
Hardware	
Under test SSDs	8x Samsung 850 Pro, 512GB, SATA
RAM	8GB from Hynix Semiconductor
CPU	16 core Intel (R) Xeon (R) E5-2620 @ 2.1GHz
Motherboard	Supermicro X10DRL-i

Linux kernel version 3.10.0-327 running on *CentOS 7* operating system, and arrays of Samsung 850 Pro SSDs to obtain the effect of erasure codes on the performance of SSD arrays, and capture the SSD usage logs. Table IV provides the details of hardware and software stack in real-system. This platform is responsible for array performance evaluation and collecting SSD usage logs. Afterwards, the array reliability is evaluated using the fault injection environment by post-processing the I/O traces and SSD usage logs collected from the real-system run (as detailed in Section IV-C). The statistical fault injection environment is developed from scratch in C++. The pseudo-code of the major functions of fault injection implementation is shown in Algorithm ?? in Appendix ??. The supplementary function definitions are also shown in Algorithm ?? in Appendix ??.

1) Workloads: The experiments are conducted using both synthetic benchmarks and realistic applications. For the synthetic experiments, we employ FIO tool [72] and run *Random Read* (RR), *Random Write* (RW), *Sequential Read* (SR), *Sequential Write* (SW), and mixture of random read/write requests ($Mixed$) workloads. We also employ *Filebench* [73] in order to commit realistic application I/O requests to the disk subsystem. We run various workloads, including *Webserver*, *Fileserver*, *Varmail*, *Copyfiles*, *Mongo*, and *Video server* from *Filebench* framework. In the following, we explain the characteristics of examined benchmarks.

- **FIO** is a powerful synthetic benchmark, capable of generating synthetic workloads with customized access pattern, request type, request size, locality of accesses, and *Working Set Size* (WSS). Using FIO, we examine five representative synthetic workloads as detailed in Table V.
- **Filebench** is a benchmarking tool that works on the filesystem level and can generate a big spectrum of application workloads. In our experiments, we employ six representative workloads including *Webserver*, *Varmail*, *Webproxy*, *Mongo*, *Video server*, and *Fileserver*. The *Webserver* workload creates millions of files with mean size equal to 64KB and the maximum request size equal to 1MB where more than 100 threads have access to the files at the same time.

Similarly, the *Varmail* workload creates files with mean size equal to 16KB with a maximum request size equal to 1MB, but the number of threads is equal to 16 which is significantly less than *Webserver*.

The *Fileserver* workload creates more than 600,000 files with mean size equal to 128KB and the maximum request size equal to 1MB while more than 50 threads have simultaneous access to the files. The *Mongo* workload which simulates the MongoDB I/O requests creates 100,000 files with mean size equal to 512KB and the maximum request size equal to 1MB while only one thread commits I/O request to the files. The last workload is *Video server* that creates files with the size of 10GB in average, including 6 active and 7 inactive videos. The event rate of this workload is equal to 96 and the average I/O size is equal to 256KB which only includes read requests. We should note that Filebench experiments are performed with both enabled and disabled buffer cache. We mainly disable buffer cache to evaluate the performance of the disk subsystem (i.e., array of SSDs) and to remove the impact of filesystem level cache on the I/O requests. Our initial experiments reveal that enabled buffer cache reduces the number of committed writes into the SSD array by 2X (on average). We observe high level of reduction in two types of workloads: a) read-intensive ones with few number of write operations such as videoserver and fileserver and b) write-intensive workloads with large number of Write-After-Write (WAW) sequences with small reuse distance. The former type of workloads has a limited number of write accesses that are mostly handled by DRAM in the presence of buffer cache while in the latter type of workloads, the write accesses and further modifications (second write operation on the same address) are mainly buffered in the buffer cache. Hence, by enabling the buffer cache we experience considerably fewer write operations on the disk subsystem (*varmail*, *randwrite*, and *randreadwrite* workloads best fit in this group). For the rest of workloads such as *mongo*, *copyfiles*, and *webserver*, since these workloads almost include equal number of read and write operations (with sequential pattern in *copyfiles* and random pattern in *mongo* and *webserver*), buffer cache is of less improving effect and helps to reduce the write operations respectively by 1.07X, 1.6X, and 1.5X in *mongo*, *copyfiles*, and *webserver*.

TABLE V: Parameters of the running workloads with FIO.

Workload	Req. Size	Req. Type	Access Pattern	I/O depth	Threads	I/O Engine
SR	4MB	Read	Sequential	16	1	libaio
SW	4MB	Write	Sequential	16	1	libaio
RR	4KB	Read	Random (distribution: zipf:1.2)	16	16	libaio
RW	4KB	Write	Random (distribution: zipf:1.2)	16	16	libaio
Mixed	4KB	Read/Write (read: 70%)	Random (distribution: zipf:1.2)	16	16	libaio

The array performance for each experiment is collected from the output of the FIO and Filebench (reported in Appendix **??**). We validate the performance output of the benchmarks using *iostat* tool from *sysstat* package of the Linux. The block-layer log of logical accesses to individual SSDs and the SSD array (i.e., virtual disk) is collected by using *blktrace* [71]. *blktrace* is a comprehensive I/O tracing tool of Linux kernel that monitors the I/O requests committed and responded by the SSDs and virtual disk. The exact number of writes to each SSD and the number of wear leveling and Program/Erase (P/E) operations performed on each SSD is also obtained by using S.M.A.R.T [70]. These statistics are used later in the fault injection process to dynamically evaluate the failure rates and also reported as endurance results in Appendix **??**. Table VI shows the basic configuration of examined SSD array. In some experiments, we have modified some parameters that are noted in case. Note that we examine different erasure codes under fixed physical capacity to have a fair performance comparison.

TABLE VI: SSD array basic configuration.

Parameter	value	Parameter	value
SSD Elements	8	SSD Page Size	4 KB
SSD Page Per Block	64	SSD Planes Per Element	8
SSD Block Per Element	16,384	SSD Stripe Size	128 KB
SSD Size	512 GB	SSD Blocks	131,072
Array Devices	8	Chunk Pages	4

TABLE VII: Summary of SSD Bad chip and bad block statistics in four years of mission time [37].

Model Name	MLC-A	MLC-B	MLC-C	MLC-D	SLC-A	SLC-B
% Drives w/ Bad Chips	5.6	6.5	6.6	4.2	3.8	2.3
% Drives w/ Bad Blocks	31.1	79.3	30.7	32.4	39.0	64.6
Median # Bad Blocks	2	3	2	3	2	2
Mean # Bad Blocks	772	578	555	312	584	570

Using the collected statistics, we conduct fault injection experiments for 4 years mission time, while 1000 experiments are conducted per configuration.

B. Fault Injection Parameters

We used SSD failure statistics by [37] that investigates the failure of six SSD chip models, including four chip with MLC technology and two with SLC technology. We use the median RBER reported by [37] (Fig. 7) as a function of P/E cycles for each SSD model. Within fault injection experiments, RBER for each chip is dynamically determined regarding the number of P/E cycles at time t, obtained from SSD usage log. Using RBER, the *Bit Error Rate* (BER) of each SSD device is dynamically evaluated regarding the number of accessed bits, obtained from SSD usage logs, as follows:

$$BER = RBER \times number\ of\ accessed\ bits(\Delta t) \qquad (5)$$

Where Δt is the time interval the BER is evaluated for. We conduct our experiments by considering $\Delta t = 1 hour$. We also use the percentage of drives with bad chips and bad blocks reported for each SSD model in a four year mission [37], as well as the mean and median number of bad blocks for each SSD model (appeared in Table VII) [37]. Regarding these statistics and a restriction reported by [37] that the chip is considered failed when 5% of its blocks are failed (happened in 2/3 of all bad chip cases), we determine bad chip time and bad block rate for each SSD chip.

C. Validating Regression Model

Table VIII shows the SSD failure statistics of 10,000 drives in our regression model. Comparing the output of the regression model with field data results, the regression model fully matches the field data [37] in the following parameters:

- Drives with bad chip
- Drives with bad blocks
- Median number of bad blocks

Mean number of bad blocks has a maximum 11% error (for MLCD) compared to the field data, as shown in Fig. 8. Another
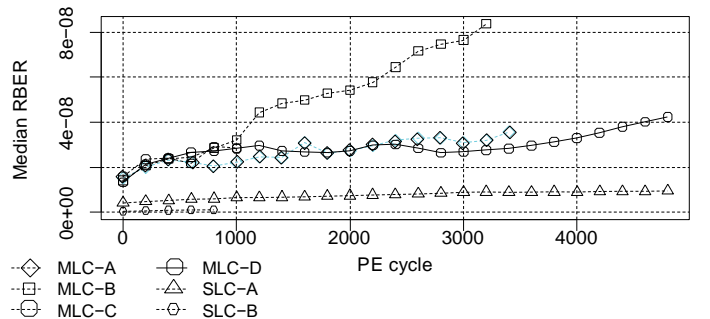

Fig. 7: Summary of SSD RBER statistics as a function of P/E cycles in 4 years of mission time [37].

TABLE VIII: Failure statistics of 10,000 SSD drives in the regression model.

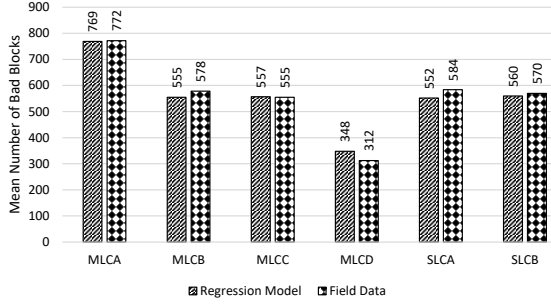|	MLCA	MLCB	MLCC	MLCD	SLCA	SLCB
Drives W Bad Blocks	3100	7930	3070	3240	3900	6460
Median # Bad Blocks	2	3	2	3	2	2
Mean # Bad Blocks	769	555.08	557.13	347.54	551.68	559.88
Drives W Bad Chips	560	650	660	420	380	230
Drives with BC and BB in more than 5% of all blocks	375	435	442	281	254	154
Rate of drives with BC and BB in more than 5% of all blocks over drives with BC	0.67	0.67	0.67	0.67	0.67	0.67
Drives with BC but no BB (not reported by the field data)	141	48	157	95	76	25

Fig. 8: Mean number of bad blocks reported by field data [37] and regression model.

Fig. 9: Failure breakdown for different erasure codes and SSD types (TTS = 10,000h, TTR = 10h)

constraint obtained by field data is the correlation between bad chip and bad block. Indeed, 2/3 of all bad chips happen in those chips that have more than 5% of all their blocks failed. Table VIII reports the rate of drives with BC and BB in more than 5% of all blocks, over drives with BC, showing that our regression model is loyal to this correlation.

Another correlation reported for bad blocks is the median number of bad blocks a drive will experience within mission time, as a function of number of bad blocks already experienced. The field data shows a very steep increase in median number of BB when the chip experiences more than one (in the case of MLC) and more than three (in the case of SLC) bad blocks. In specific in the case of MLC, the median number of bad blocks jumps to 200 after the second bad block is experienced. We consider this constraint in our regression model by increasing the rate of BB in those chips that have already experienced two bad blocks (four in the case of SLCs). However, as the field data does not distinguish the mentioned statistics for different MLC/SLC models (as appeared in Figure 8 of [37]), we employ the average values reported for SLCs and MLCs. Our empirical results show that the regression model is loyal to this constraint, as shown in table IX. Table IX shows the median number of bad blocks as a function of previous number of bad blocks experienced in the regression model output and field data (the average value reported for SLC and MLC is reported).

D. Data Loss Breakdown

Fig. 9 shows data loss root cause breakdown for different erasure codes and SSD types. The results are reported for aggregate of lost stripes in 12 examined workloads (including both synthetic and realistic application workloads), assuming

TABLE IX: Median number of bad blocks as a function of previous number of bad blocks experienced in the regression model and field data [37].

Previous Num. of Bad Blocks	2		3		4		5	
	Reg. Model	Field Data	Reg. Model	Field Data	Reg. Model	Field Data	Reg. Model	Field Data
MLC	143	143	155	151	159	158	183	183
SLC	5	5	20	20	43	45	77	75
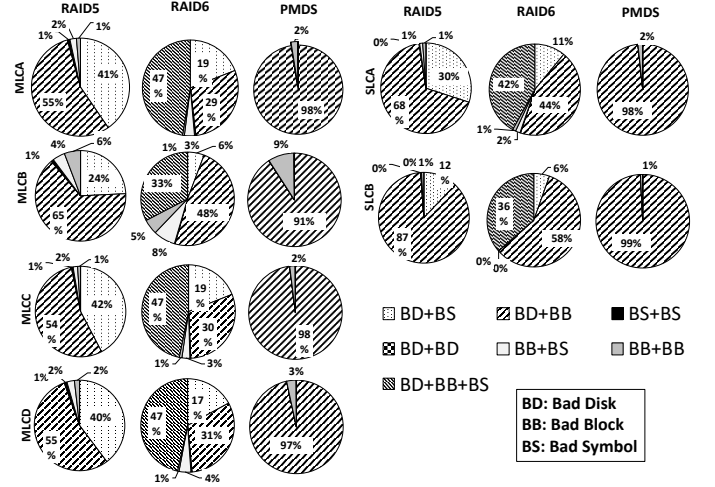
Time to Scrub (TTS)[7] equal to 10,000 Hours and *Time to Recover* (TTR)[8] equal to 10 Hours. The figure shows how different combinations of failures, including Bad Chip, Bad Block, and Bad Symbol, contribute to data loss.

As the figure shows, data loss breakdown significantly correlates with both erasure code and SSD type. The combination of bad disk and bad block (BD+BB) is the dominant source of data loss when using PMDS erasure code. A relatively smaller share of DL, less than 10% in all SSD types, is caused by coincidence of two bad blocks (BB+BB) in a data stripe. Hence, we can conclude that bad blocks are the dominant source of data loss when using PMDS codes. This observation is described by the fact that PMDS codes can correct the combination of bad chip with a bad symbol, and can also correct the coincidence of two bad symbols. Bad symbols leading to data loss just happen either in the case two bad symbols in a single stripe coincide with a bad chip (BD+BS), or three bad symbols coincide in a single stripe (BS+BS), that are not so probable (BD+BS and BS+BS failures happened respectively in 123 and two cases of data loss, compared to 12,194,444 cases caused by BD+BB). PMDS codes, however, fail to correct the combination of bad disk and bad block, as the bad blocks make an entire data chunk lost, rather than a single symbol (each data chunk includes 4 symbols, considering 4KB page size, 128KB stripe size, and 8 devices per stripe). Please note that other combinations of failures also have non-zero values, but are not reported as their contribution is less than 1%.

Data loss breakdown of RAID5 and RAID6 is more sensitive to SSD type. However, RAID5 failures are dominantly caused by the coincidence of bad chip with either bad block or bad symbol, while bad chips combined with bad blocks cause more than 50% of data loss in all SSD types. In the case of RAID6, the coincidence of bad chip, bad block and bad symbol (BD+BB+BS) has also a significant contribution in total data loss, even greater than BD+BB (caused by one bad chip combined with two bad blocks) for MLCA, MLCC, and MLCD.

E. Impact of Workload

Fig. 10 reports the number of lost stripes within 4 years mission time experienced in 1000 SSD arrays in both cases of enabled and disabled buffer cache. The results are reported for different SLC and MLC types. SSD arrays experience different magnitude of data loss depending on the examined workload and SSD type. This difference is mainly caused by workload

---

[7]TTS is the expected time between two array scrubbing processes.

[8]In the case of SSD failures, TTR is the expected time of device recovery process.

characteristics including number of P/E cycles and disk accesses which would be reduced in case of enabling buffer cache.

An important observation is that the relative reliability of workloads may change in different SSD types. For example in MLCA using RAID5 configuration, Fileserver workload experiences the most data loss, while in MLCB the most data loss is experienced in Varmail workload. Different rate of bad chip, bad block, and bad symbol in different SSD types and how they correlate with the workload characteristics is the major source of this observation. While the rate of bad chip and bad block is characterized by SSD type (Table VII) and determined in the start of simulation (discussed in Section V-B), the rate of bad symbol, determined by RBER, is also a function of P/E cycles (Fig. 7) and is highly correlated with workload and the impact of buffer cache where by enabling buffer cache we observe about 26.2%, 56.1%, and 29.5% smaller failure rate in RAID5, RAID6, and PMDS, respectively. Accordingly, the workloads characterized by large number of P/E cycles (i.e., the workloads dominated by write requests) experience a relatively greater data loss in SSD types with large RBER (MLCA, MLCB, MLCC, and MLCD).

F. Impact of SSD type

Fig. 11 compares the number of lost stripes in different SSD types. The reported values are aggregated from 11 synthetic and realistic application workloads. As the results show, MLCB is the least reliable SSD, experiencing one order of magnitude greater data loss than SLCB. Referring to the failure characteristics of MLCB (Fig. 7 and Table VII), this observation is described by MLCB having the greatest RBER, resulting in the highest rate of bad symbol between examined SSD types. While the mean number of bad blocks (per device) in MLCB is average, it has the greatest percentage of drives with bad blocks (79.5%), also describing the low reliability of this SSD type.

Another observation is considerable reliability benefits of SLC types over MLC types, specially in the case of SLCB. Both SLCA and SLCB, as Fig. 7 shows, have significantly lower RBER compared to MLC types. Moreover, percentage of drives with bad chips reported for SLCA and SLCB (Table VII) is considerably lower than MLC types (3.8% and 2.3%, respectively for SLCA and SLCB), helping the greater reliability in SLC types. While SLCB outperforms SLCA in terms of reliability, Table VII shows that it has greater percentage of drives with bad blocks than SLCA (64.6% vs 39.0%). This observation is described by the greater percentage of drives with bad chips in SLCA, and slightly greater RBER, compared to SLCB.

G. Impact of Time to Recover and Time to Scrub

Fig. 12 shows the impact of time to recover (TTR) and time to scrub (TTS) on the array reliability. One important factor that contributes array reliability is time to recover the array from a device failure, by reconstructing the failed device data to a brand-new device. Duration of this procedure, however, depends on the array architecture and is a function of parameters such as SSD performance and bandwidth of interconnections. Moreover, the reconstruction process is usually performed when the array is operational. Hence, the reconstruction time is also a function of workload (it takes longer under heavy workloads). Increased reconstruction time has a negative impact on the array reliability. The reason behind is the accumulation of bit errors within reconstruct process, possibly leading to stripe data loss. By increasing TTR from 10 to 100, the expected number of bit error within reconstruct process is increased by 10 times, leading to greater number of lost stripes.

Another important factor contributing array reliability is time to scrub. Scrubbing is performed on predefined periods to remove possible bit errors using array redundancies. This process reduces the chance of data loss by preventing the accumulation of bit errors, as well as the combination of device failure and bit errors. Scrubbing, however, is a costly process, as it mandates reading and verifying the entire array data.

Hence, time to scrub is defined to reach an effective trade-off between reliability and performance, depending on the policies of datacenter administrators.

As Fig. 12 shows, the impact of TTS on reliability is significant. In the case of RAID5 and PMDS, increasing TTS from 1000 to 10,000 has an ascending impact on data loss by almost 8 times. This impact is even more drastic in RAID6 and causes 56 times data loss increase. In the case of RAID5 and PMDS, increasing TTS from 100 to 10,000 results in 30 times greater data loss. It is also worth to mention that the impact of increased TTR is more drastic when having smaller TTS values. Under small TTS values, the total number of lost stripes is reduced, magnifying the impact of TTR increase. For example in the case of RAID5, increasing TTR from 10 to 100 leads to 10% data loss increase in the case of TTS=10,000. In the case of TTS=1000 and TTS=100, however, increasing TTR from 10 to 100 results in respectively 26% and 171% data loss increase.

Another important observation is that decreasing TTS improves RAID6 more than RAID5 and PMDS. Better explaining the case, we define *Late Scrub* (LS) a scrubbing process that comes too late to prevent Data Loss (DL) in a stripe and *Late Scrub Threshold* (LST) as the maximum TTS that can prevent DL for each data stripe. We expect weak erasure codes such as RAID5 having lower average LST and powerful erasure codes such as RAID6 having greater average LST. When LST is too low, even a big improvement (decrease) in TTS makes no difference. That is why we here observe decreasing TTS improves RAID6 more than RAID5 and PMDS.

H. Impact of Stripe Size

Fig. 13 shows the impact of stripe size on the array reliability. The minimum and maximum possible values for stripe size are determined by RAID controller manufacturers. For SSD RAID controllers, the minimum possible configuration is 64KB [74]. However, here we also examine 32KB stripe size configuration in our fault-injection experiments.

Regarding our analysis in Section IV, the stripe size has no impact on RAID5 and RAID6 codes, as those codes just employ row-wise parity codes. PMDS codes, however, benefit smaller stripe size. By reducing the stripe size, the global parity symbol would be responsible for error correction of a smaller number of data symbols, having less chance of fault accumulation leading to uncorrectable error. The empirical results also confirm our hypothesis and show a significant reliability improvement in PMDS codes when reducing stripe size to 32KB. Indeed, when reducing stripe size to 32KB, we observe 1002 stripe loss events in PMDS codes, versus 1754 events that we observe in RAID6. Theoretically, PMDS should not perform better than RAID6 in terms of reliability, but this observation is described by greater write overhead of RAID6 compared to PMDS; For the Mixed workload, due to having larger number of writes in RAID6 array (depicted in Fig. ??), the number of P/E cycles, as well as number of accesses is increased, leading to a greater RBER. For RAID5 and RAID6, however, reducing stripe size has no impact on reliability.

Please note that by reducing the stripe size, total number of stripes is doubled. Hence, the magnitude of data loss caused by two lost stripes in 32KB mode is equal to the magnitude of data loss caused by one lost stripe in 64KB mode. We observe that reducing stripe size by a factor of two almost doubles the number of lost stripes in RAID5 and RAID6, as expected. For example, by reducing stripe size from 64KB to 32KB, the number of lost stripes in RAID5 is increased from $1,810,334$ to $3,562,789$.

I. Comparison with Previous Models

Fig. 14 compares previous SSD array reliability models with the proposed model. The chart reports number of lost stripes normalized to proposed model results for TTS=10,000h and TTR=10h. In this chart, we classify the previous works into two categories. The models proposed by Balakrishnan et al. [44],
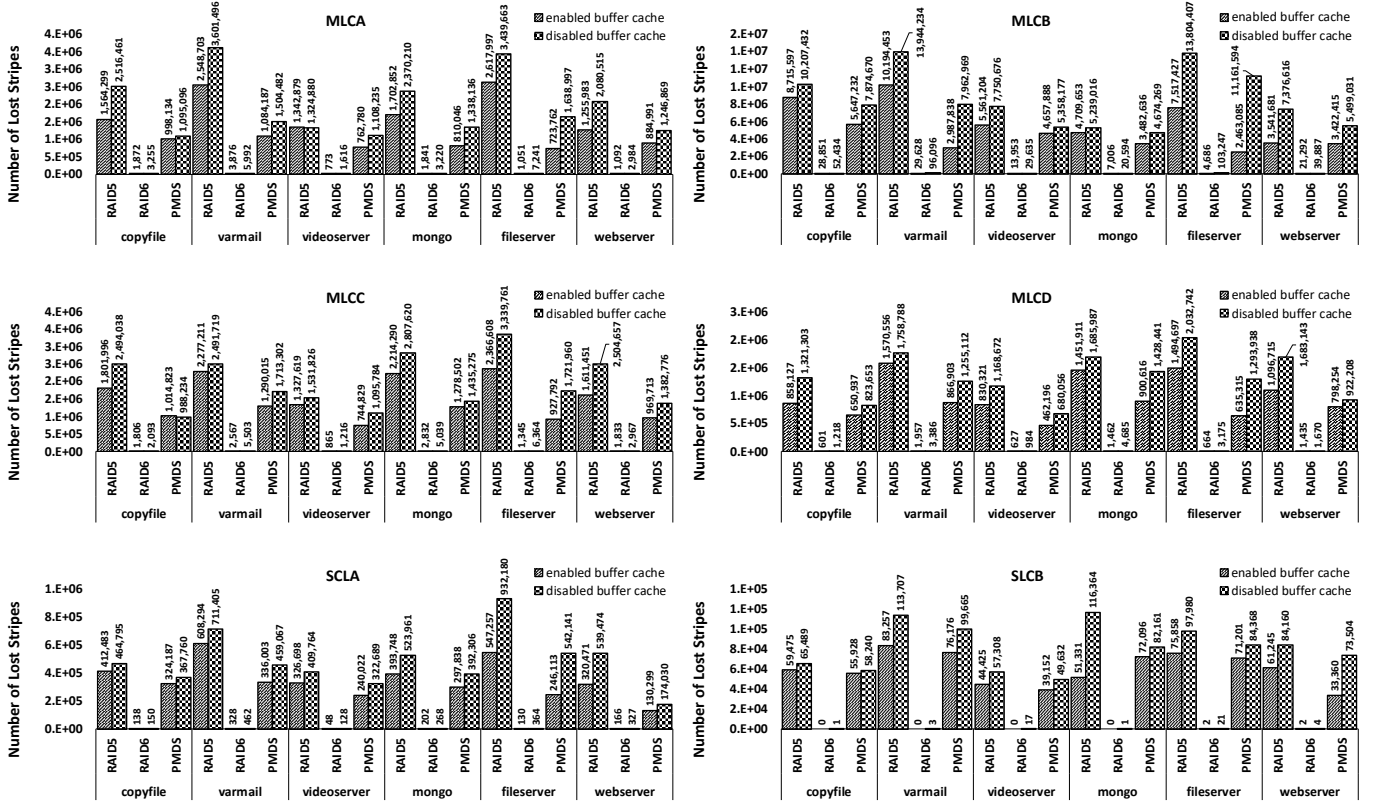
Fig. 10: Comparing reliability of different workloads (TTS = 10,000h, TTR = 10h).
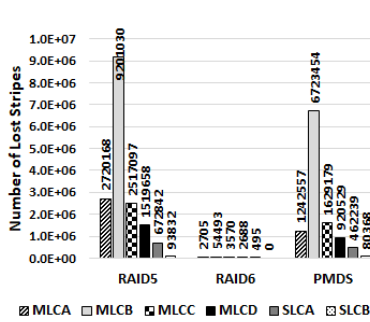
Fig. 11: Comparing reliability of different SSD types (TTS = 10,000h, TTR = 10h)

Fig. 13: Impact of Stripe Size (32KB, 64KB, and 128KB) on the reliability of SSD array, for MLCA under Mixed workload (TTS=10,000h, TTR=10h). The rest of SSD array parameters are appeared in Table VI.

Fig. 12: Impact of TTR (Hours) and TTS (Hours) on the reliability of SSD array (For MLCA under Mixed workload).

Blaum et al. [46], and Moon et al. [43] that consider the coincidence of bad chip and bad symbol and ignore the impact of bad block (as summarized in Table I) are classified as *Balakrishnan-Blaum-Moon*. The model of Li et al. [41] that just takes the coincidence of bad symbols into account (ignores bad chip and bad block) is classified as *Li*. The previous works, however, have also other sources of inaccuracy, neglected in this comparison, such as using deprecated SSD failure field data, using either Markov models or closed-form probability equations, and not using real-system implementation.

As the results show, *Li* provides the less accurate results due to ignoring both bad chips and bad blocks and just considering the coincidence of bad symbols in a data stripe. Depending on the SSD type, the results of *Li* underestimate reliability by at least two orders of magnitude. The results of *Balakrishnan-Blaum-Moon* is more accurate, due to considering

Fig. 14: Comparing previous SSD array reliability models with the proposed model. The chart reports number of lost stripes normalized to proposed model results for TTS=10,000h and TTR=10h. This chart reports the aggregate of copyfile, varmail, videoserver, mango, fileserver, and webserver workloads.

the coincidence of bad chips and bad symbols. However, in the case of SLCB arrays configured by PMDS codes, we observe *Balakrishnan-Blaum-Moon* underestimates reliability by five orders of magnitude.

J. Summary of Observations

Table X reports a summary of our observations comparing RAID5, RAID6, and PMDS in terms of source of failures and the impact of array parameters such as SSD type, TTS, TTR, and stripe size.

TABLE X: Summary of observations comparing RAID5, RAID6, and PMDS.

	Main Source of Failure	Dep. to SSD Type	Dep. to TTS	Dep. to TTR	Dep. to Stripe Size
RAID5	1) BD+BB 2) BD+BS	Yes	Significant	Less Significant	No
RAID6	1) BD+BB+BS 2) BD+BB	Yes	Significant	Less Significant	No
PMDS	BD+BB	Yes	Significant	Less Significant	Yes

The detailed observations as reported in Table X are as follows:

- Having slightly greater ERF than RAID5, PMDS(1,1) codes are proposed to offer a reliability close to RAID6 [50], [46] by correcting the combination of device and symbol failures. However, our analysis using recent field results show that PMDS reliability is far behind RAID6. The major source of misleading conclusions in previous works is taking deprecated assumptions about failure characteristics of SSD devices, falsified by state-of-the-art field data [37], [38].
- While PMDS(1,1) copes with the combination of device failure and symbol failure, it fails to correct errors combined by device failure and block failure, contributing more than 90% of total data loss.
- Even in RAID5 which can tolerate just a single device failure, the contribution of errors combined by device failure and block failure is more than those combined by device failure and symbol failure. While the rate of block failures is significantly lower than symbol failures, this observation is described by the greater magnitude of data loss imposed by block failures (in our experiments a single SSD block is shared upon 16 data stripes).
- The contribution of bad blocks combined by device failure (two bad blocks and one bad device) is also significant in total data loss of RAID6.
- In the resolution of one single data stripe where erasure codes take effect, the block failures manifest as device failure (they result in the loss of a full data chunk, rather than a single symbol). Hence, symbol-level protections (suggested by PMDS codes) are not effective in dealing with block failures. Regarding the significant contribution of bad blocks combined with bad devices in total data loss, we can conclude that device-level protections, such

as conventional RAIDs, are the most effective choices at least for the contemporary SSD architectures.

- The dark side of PMDS codes, ignored by its creators [46], [50], [51], is the negative effect of global parity write overhead on SSD endurance, that by itself violates reliability.
- SSD array reliability, as well as the failure breakdown, is significantly correlated with SSD type.
- Previous models on the reliability of SSD arrays just focus on the coincidence of bad symbols and bad chip. Our study, however, shows that this type of failure contributes the minority of data loss in SSD arrays and the previous models underestimate data loss by less than half.
- Time to scrub has a significant impact on array reliability, while the impact of time to recover from a device failure is of less significance.
- RAID5 and RAID6 codes which use row-wise parity, perform almost independent of stripe size. PMDS codes, however, benefit smaller stripe sizes and show a promising reliability improvement when reducing stripe size from 128KB to 32KB. This observation motivates us for further investigations of the effect of stripe size, under different workloads and array architectures.

VI. Conclusion

In this paper, we investigated the reliability of SSD arrays using real-system implementation of conventional and emerging erasure codes, under realistic storage traces. The reliability is evaluated by statistical fault injection experiments that post-process the SSD usage logs obtained from the real-system implementation, while the fault/failure attributes are obtained from the state-of-the-art field data by previous works. As a case study, we examined conventional RAID5 and RAID6 and emerging PMDS codes, SD codes and STAIR codes in terms of both reliability and performance using an open-source software RAID controller, MD (in Linux kernel version 3.10.0-327), and arrays of Samsung 850 Pro SSDs.

Our experiments showed that previous models underestimate the SSD array reliability by up to six orders of magnitude, as they just focus on the coincidence of bad pages (bit errors) and bad chips within a data stripe that holds the minority of data loss cause in SSD arrays. We observed the combination of bad chips with bad blocks as the major source of data loss in RAID5 and emerging codes (contributing more than 54% and 90% of data loss in RAID5 and emerging codes, respectively), while RAID6 remained robust under these failure combinations. We also observed that time to scrub is a significant contributor to array data loss, while the impact of time to recover from a device failure is of less significance. Finally, the fault injection results show that SSD array reliability, as well as the failure breakdown, is significantly correlated with SSD type. While our empirical results showed that emerging erasure codes fail to replace RAID6 in terms of reliability when having stripe sizes commonly used in enterprise RAID controllers (128KB and 64KB), for a speculative 32KB stripe size we observed a promising reliability improvement in emerging erasure codes, performing similar to RAID6.

References

[1] R. F. DeMara and P. Montuschi, "Non-volatile memory trends: Toward improving density and energy profiles across the system stack," *Computer*, no. 4, pp. 12–13, 2018.

[2] P. Wang, S. Li, G. Sun, X. Wang, Y. Chen, H. Li, J. Cong, N. Xiao, and T. Zhang, "Rc-nvm: Enabling symmetric row and column memory accesses for in-memory databases," in *High Performance Computer Architecture (HPCA)*. Vienna, Austria: IEEE, 2018, pp. 518–530.

[3] J. Henkel, "Emerging memory technologies," *IEEE Design & Test*, vol. 34, no. 3, pp. 4–5, 2017.

[4] Y. Huai, "Spin-transfer torque mram (stt-mram): Challenges and prospects," *Association of Asia Pacific Physical Societies (AAPPS) bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[5] C. Yang, H.-M. Chen, T. N. Mudge, and C. Chakrabarti, "Improving the reliability of mlc nand flash memories through adaptive data refresh and error control coding," *Journal of Signal Processing Systems*, vol. 76, no. 3, pp. 225–234, 2014.

[6] C. Yang, Y. Emre, C. Chakrabarti, and T. Mudge, "Flexible product code-based ecc schemes for mlc nand flash memories," in *Signal Processing Systems (SiPS)*. Beirut, Lebanon: IEEE, 2011, pp. 255–260.

[7] N. Khoshavi and R. F. DeMara, "Read-tuned stt-ram and edram cache hierarchies for throughput and energy optimization," *IEEE Access*, vol. 6, pp. 14 576–14 590, 2018.

[8] V. Fedorov, J. Kim, M. Qin, P. V. Gratz, and A. Reddy, "Speculative paging for future nvm storage," in *International Symposium on Memory Systems (MEMSYS)*. Alexandria, VA, USA: ACM, 2017, pp. 399–410.

[9] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "Mnftl: An efficient flash translation layer for mlc nand flash memory storage systems," in *Design Automation Conference (DAC)*. San Diego, CA, USA: ACM, 2011, pp. 17–22.

[10] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao, "Pcm-ftl: A write-activity-aware nand flash memory management scheme for pcm-based embedded systems," in *Real-Time Systems Symposium (RTSS)*. Vienna, Austria: IEEE, 2011, pp. 357–366.

[11] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "Demand-based block-level address mapping in large-scale nand flash storage systems," in *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Scottsdale, AZ, USA: ACM, 2010, pp. 173–182.

[12] ——, "A two-level caching mechanism for demand-based page-level address mapping in nand flash memory storage systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Chicago, IL, USA: IEEE, 2011, pp. 157–166.

[13] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," *Emerging Memory Technologies*, pp. 15–50, 2014.

[14] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," *ACM SIGARCH computer architecture news*, vol. 37, no. 3, pp. 34–45, 2009.

[15] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Design, Automation & Test in Europe (DATE)*. Dresden, Germany: IEEE, 2010, pp. 136–141.

[16] R. Zand, A. Roohi, and R. F. DeMara, "Energy-efficient and process-variation-resilient write circuit schemes for spin hall effect mram device," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2394–2401, 2017.

[17] Y. Guan, G. Wang, C. Ma, R. Chen, Y. Wang, and Z. Shao, "A block-level log-block management scheme for mlc nand flash memory storage systems," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1464–1477, 2017.

[18] Y. Kang, X. Zhang, Z. Shao, R. Chen, and Y. Wang, "A reliability enhanced video storage architecture in hybrid slc/mlc nand flash memory," *Journal of Systems Architecture*, vol. 88, pp. 33–42, 2018.

[19] Y. Wang, Y. Han, L. Zhang, H. Li, and X. Li, "Propram: exploiting the transparent logic resources in non-volatile memory for near data computing," in *Design Automation Conference (DAC)*. San Francisco, CA, USA: ACM, 2015, p. 47.

[20] Q. Zhao, M. Rajaei, I. Krivorotov, M. Nilsson, N. Bagherzadeh, and O. Boyraz, "Optical investigation of radiation induced conductivity changes in stt-ram cells," in *Conference on Lasers and Electro-Optics (CLEO)*. San Jose, CA, USA: IEEE, 2016, pp. 1–2.

[21] R. Salkhordeh, O. Mutlu, and H. Asadi, "An analytical model for performance and lifetime estimation of hybrid dram-nvm main memories," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1114–1130, 2019.

[22] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized ssd cache for primary storage." in *USENIX Annual Technical Conference*, Philadelphia, PA, USA, 2014, pp. 501–512.

[23] C. Li, P. Shilane, F. Douglis, and G. Wallace, "Pannier: Design and analysis of a container-based flash cache for compound objects," *ACM Transactions on Storage (TOS)*, vol. 13, no. 3, p. 24, 2017.

[24] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen, "Duracache: A durable ssd cache using mlc nand flash," in *Design Automation Conference (DAC)*. Austin, TX, USA: ACM, 2013, p. 166.

[25] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive i/o characterization," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1678–1691, 2016.

[26] R. Salkhordeh, S. Ebrahimi, and H. Asadi, "Reca: an efficient reconfigurable cache architecture for storage systems with online workload characterization," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PP, no. PP, pp. 1–1, 2018.

[27] S. Ahmadian, O. Mutlu, and H. Asadi, "Eci-cache: A high-endurance and cost-efficient i/o caching scheme for virtualized platforms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 2, no. 1, p. 9, 2018.

[28] R. Salkhordeh, M. Hadizadeh, and H. Asadi, "An efficient hybrid i/o caching architecture using heterogeneous ssds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1238–1250, 2018.

[29] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *International Conference on Supercomputing (ICS)*. Tucson, Arizona, USA: ACM, 2011, pp. 96–107.

[30] J. Guerra, H. Pucha, J. S. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Conference on File and Storage Technologies (FAST)*, vol. 11, San Jose, CA, USA, 2011, pp. 20–20.

[31] R. Salkhordeh, H. Asadi, and S. Ebrahimi, "Operating system level data tiering using online workload characterization," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1534–1562, 2015.

[32] J. G. Elerath and J. Schindler, "Beyond mttdl: A closed-form raid 6 reliability equation," *ACM Transactions on Storage (TOS)*, vol. 10, no. 2, p. 7, 2014.

[33] 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 *International Symposium on Communications and Information Technology (ISCIT)*. Incheon, Korea: IEEE, 2009, pp. 140–145.

[34] M. Kishani, M. Tahoori, and H. Asadi, "Dependability analysis of data storage systems in presence of soft errors," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 201–215, 2019.

[35] M. Kishani and H. Asadi, "Modeling impact of human errors on the data unavailability and data loss of storage systems," *IEEE Transactions on Reliability (TR)*, vol. 67, no. 3, pp. 1111–1127, 2018.

[36] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," vol. 17, no. 3, 1988.

[37] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected." in *Conference on File and Storage Technologies (FAST)*. Santa Clara, CA, USA: USENIX, 2016, pp. 67–80.

[38] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. Portland, Oregon, USA: ACM, 2015, pp. 177–190.

[39] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "Ssd failures in datacenters: What? when? and why?" in *International Systems and Storage Conference*. Haifa, Israel: ACM, 2016, p. 7.

[40] S. Ahmadian, F. Taheri, M. Lotfi, M. Karimi, and H. Asadi, "Investigating power outage effects on reliability of solid-state drives," in *Design, Automation and Test in Europe (DATE)*. Dresden, Germany: IEEE/ACM, March 2018.

[41] Y. Li, P. P. Lee, and J. C. Lui, "Analysis of reliability dynamics of ssd raid," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1131–1144, 2016.

[42] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Improving ssd reliability with raid via elastic striping and anywhere parity," in *Dependable Systems and Networks (DSN)*. Budapest, Hungary: IEEE, 2013, pp. 1–12.

[43] S. Moon and A. Reddy, "Does raid improve lifetime of ssd arrays?" *ACM Transactions on Storage (TOS)*, vol. 12, no. 3, p. 11, 2016.

[44] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential raid: Rethinking raid for ssd reliability," *ACM Transactions on Storage (TOS)*, vol. 6, no. 2, p. 4, 2010.

[45] K. M. Greenan, D. D. Long, E. L. Miller, T. Schwarz, and A. Wildani, "Building flexible, fault-tolerant flash-based storage systems," in *Proceedings of the 5th Workshop on Hot Topics in System Dependability (HotDep 2009)*, Yokohama, Japan, 2009.

[46] M. Blaum, J. L. Hafner, and S. Hetzler, "Partial-mds codes and their application to raid type of architectures," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4510–4519, 2013.

[47] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of nand flash memory," in *Conference on File and Storage Technologies (FAST)*. San Jose, CA, USA: USENIX, 2012, pp. 2–2.

[48] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in mlc nand flash memory: Measurement, characterization, and analysis," in *Design, Automation and Test in Europe*. Dresden, Germany: EDA Consortium, 2012, pp. 521–526.

[49] 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 *Microarchitecture, Annual IEEE/ACM International Symposium on (MICRO)*. New York, NY, USA: IEEE, 2009, pp. 24–33.

[50] J. S. Plank and M. Blaum, "Sector-disk (sd) erasure codes for mixed failure modes in raid systems," *ACM Transactions on Storage (TOS)*, vol. 10, no. 1, p. 4, 2014.

[51] M. Li and P. P. Lee, "Stair codes: A general family of erasure codes for tolerating device and sector failures," *ACM Transactions on Storage (TOS)*, vol. 10, no. 4, p. 14, 2014.

[52] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in nand flash memories," in *IEEE International Reliability Physics Symposium (IRPS)*. Phoenix, AZ, USA: IEEE, 2008, pp. 9–19.

[53] K. M. Greenan, E. L. Miller, T. J. Schwarz, and D. D. Long, "Disaster recovery codes: increasing reliability with large-stripe erasure correcting codes," in *ACM workshop on Storage security and survivability*. Alexandria, VA, USA: ACM, 2007, pp. 31–36.

[54] J.-F. Pâris, T. J. Schwarz, and D. D. Long, "Self-adaptive two-dimensional raid arrays," in *International Performance, Computing, and Communications Conference (IPCCC)*. New Orleans, Louisiana, USA: IEEE, 2007, pp. 246–253.

[55] C. C. A. Rincón, J.-F. Pâris, R. Vilalta, A. M. Cheng, and D. D. Long, "Disk failure prediction in heterogeneous environments," in *Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. Seattle, WA, USA: IEEE, 2017, pp. 1–7.

[56] T. Schwarz, A. Amer, T. Kroeger, E. Miller, D. Long, and J.-F. Pâris, "Resar: Reliable storage at exabyte scale," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. London, UK: IEEE, 2016, pp. 211–220.

[57] J.-F. Pâris, D. D. Long, and W. Litwin, "Three-dimensional redundancy codes for archival storage," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. San Francisco, CA, USA: IEEE, 2013, pp. 328–332.

[58] H.-W. Kao, J.-F. Pâris, S. T. Schwarz, and D. D. Long, "A flexible simulation tool for estimating data loss risks in storage arrays," in *Mass Storage Systems and Technologies (MSST)*. Long Beach, CA, USA: IEEE, 2013, pp. 1–5.

[59] K. Gopinath, J. Elerath, and D. Long, "Reliability modelling of disk subsystems with probabilistic model checking," University of California, Santa Cruz, Tech. Rep., 2009.

[60] J.-F. Pâris, T. J. Schwarz, D. D. Long, and A. Amer, "When mttdls are not good enough: Providing better estimates of disk array reliability," in *International Information and Telecommunication Technologies Symposium*, Foz do Iguaçú, Brazil, 2008, pp. 140–145.

[61] M. Kishani, R. Eftekhari, and H. Asadi, "Evaluating impact of human errors on the availability of data storage systems," in *Design, Automation and Test in Europe Conference (DATE)*. Lausanne, Switzerland: IEEE/ACM, 2017.

[62] E. d. S. e Silva and H. R. Gail, "Transient solutions for markov chains," in *Computational Probability*. Springer, 2000, pp. 43–79.

[63] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance." in *USENIX Annual Technical Conference*, Boston, Massachusetts, USA, 2008, pp. 57–70.

[64] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," *Parallel Data Laboratory*, p. 26, 2008.

[65] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Ds-raid: Efficient parity update scheme for ssds," in *Conference on File and Storage Technologies (FAST)*, vol. 4. San Jose, CA: USENIX, 2012, p. 2.

[66] S. B. Wicker, *Error control systems for digital communication and storage*. Prentice hall Englewood Cliffs, 1995, vol. 1.

[67] M. Kishani, H. R. Zarandi, H. Pedram, A. Tajary, M. Raji, and B. Ghavami, "Hvd: horizontal-vertical-diagonal error detecting and correcting code to protect against with soft errors," *Design Automation for Embedded Systems*, vol. 15, no. 3-4, pp. 289–310, 2011.

[68] M. Kishani, A. Baniasadi, and H. Pedram, "Using silent writes in low-power traffic-aware ecc," in *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Madrid, Spain: Springer, 2011, pp. 180–192.

[69] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. Long, A. Hospodor, and S. Ng, "Disk scrubbing in large archival storage systems," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*. Volendam, Netherlands: IEEE, 2004, pp. 409–418.

[70] M. S. Rothberg, "Disk drive for receiving setup data in a self monitoring analysis and reporting technology (smart) command," 2005, uS Patent 6,895,500.

[71] (2017) blktrace: A Block Layer IO Tracing Tool. [Online]. Available: https://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html

[72] (2017) Fio: Flexible I/O Tester Synthetic Benchmark. [Online]. Available: https://github.com/axboe/fio

[73] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *login: The USENIX Magazine*, vol. 41, no. 1, pp. 6–12, 2016.

[74] LSI, "LSI MegaRAID Controller Benchmark Tips," https://docs.broadcom.com/docs/12353177, Accessed: June 2018.

Mostafa Kishani received the B.S. degree in computer engineering from Ferdowsi University of Mashhad, Mashhad, Iran, in 2008, M.S. degree in computer Engineering from Amirkabir University of Technology (AUT), Tehran, Iran, in 2010, and PhD degree in computer engineering from Sharif University of Technology (SUT), Tehran, Iran, in 2018. He is currently a postdoctoral fellow in *Data Storage, Networks, and Processing* (DSN) Lab at SUT, Tehran, Iran. He was a hardware engineer in Iranian Space Research Center (ISRC) from 2010 to 2012. He was also a member of Institute for Research in Fundamental Sciences (IPM) Memocode team in 2010. From September 2015 to April 2016 he was a research assistant in Computer Science and Engineering department of the Chinese University of Hong Kong (CUHK), Hong Kong. He was also a research associate in the Hong Kong Polytechnic University (PolyU), Hong Kong, from April 2016 to February 2017.

Saba Ahmadian received the B.S. and M.S. degrees in computer engineering from SUT, Tehran, Iran, in 2013 and 2015, respectively. From 2011 to 2012, she was a member of *Energy Aware Systems* (EASY) Lab, SUT, where she researched on power reduction techniques on embedded CPUs. From 2012 to 2015, she was a member of *Embedded Systems Research* (ESR) Lab, SUT, where she researched on low power and reliability-aware techniques on Automata-based embedded systems. Currently, she is a Ph.D. candidate at *Data Storage, Networks, and Processing* (DSN) Lab at SUT under supervision of Dr. Hossein Asadi. Her research interests include storage systems design, virtualization platforms, fault tolerant design, and low power systems design.

Hossein Asadi (M'08, SM'14) received the B.Sc. and M.Sc. degrees in computer engineering from the SUT, Tehran, Iran, in 2000 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2007.

He was with EMC Corporation, Hopkinton, MA, USA, as a Research Scientist and Senior Hardware Engineer, from 2006 to 2009. From 2002 to 2003, he was a member of the Dependable Systems Laboratory, SUT, where he researched hardware verification techniques. From 2001 to 2002, he was a member of the Sharif Rescue Robots Group. He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently a tenured Associate Professor. He is the Founder and Director of the DSN Laboratory, Director of Sharif *High-Performance Computing* (HPC) Center, the Director of Sharif *Information and Communications Technology Center* (ICTC), and the President of Sharif ICT Innovation Center. He spent three months in the summer 2015 as a Visiting Professor at the School of Computer and Communication Sciences at the Ecole Poly-technique Federele de Lausanne (EPFL). He is also the co-founder of HPDS corp., designing and fabricating midrange and high-end data storage systems. He has authored and co-authored more than eighty technical papers in reputed journals and conference proceedings. His current research interests include data storage systems and networks, solid-state drives, operating system support for I/O and memory management, and reconfigurable and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup, a recipient of Best Paper Award at the 15th CSI International Symposium on *Computer Architecture & Digital Systems* (CADS), the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, and the Distinguished Technology Award from SUT in 2017. He is also recipient of Extraordinary Ability in Science visa from US Citizenship and Immigration Services in 2008. He has also served as the publication chair of several national and international conferences including CNDS2013, AISP2013, and CSSE2013 during the past four years. Most recently, he has served as a Guest Editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017).