Exploring Data-Level Error Tolerance in High-Performance Solid-State Drives

Xin Xu and H. Howie Huang, *Senior Member, IEEE*

***Abstract*—Flash storage systems have exhibited great benefits over magnetic hard drives such as low input-output (I-O) latency, and high throughput. However, NAND flash based Solid-State Drives (SSDs) are inherently prone to soft errors from various sources, e.g., wear-out, program and read disturbance, and hot-electron injections. To address this issue, flash devices employ different error-correction codes (ECC) to detect and correct soft errors. Using ECC induces non-trivial overhead costs in terms of flash area, performance, and energy consumption. In this work, we evaluate the feasibility of reducing the need for strong ECC while maintaining the correct execution of the applications. Specifically, we explore data-level error tolerance in various data-centric applications, and study the system implications for designing a low-cost yet high performance flash storage system, SoftFlash. We explore three key aspects of enabling SoftFlash. First, we design an error modeling framework that can be used in runtime for monitoring and estimating the error rates of real-world flash devices. Our experiments show that the error rate of SSDs can be modeled with reasonable accuracy (13%) using parameters accessible from operating systems. Second, we carry out extensive fault-injection experiments on a wide range of applications including multimedia, scientific computation, and cloud computing to understand the requirements and characteristics of data level error tolerance. We find that the data from these applications show high error resiliency, and can produce acceptable results even with high error rates. Third, we conduct a case study to show the benefits of leveraging data-level error tolerance in flash devices. Our results show that, for many data-centric applications, the proposed SoftFlash system can achieve acceptable results (or better in certain cases), with more than a 40% performance improvement, and a third of the energy consumption.**

***Index Terms*—Flash, solid-state drives, error correction codes.**

ACRONYMS AND ABBREVIATIONS

SSD	Solid-State Drive
ECC	Error Correction Code
P-E cycles	Program-Erase Cycles
BER	Bit Error Rate
RBER	Raw Bit Error Rate
UBER	Uncorrectable Bit Error Rate
SLC	Single-Level Cell
MLC	Multi-Level Cell
TLC	Triple-Level Cell
SECDEC	Single-Error Correction and Double-Error Detection
LDPC	Low-Density Parity-Check code
AIC	Akaike Information Criterion
MRE	Mean Relative Error
RS	Reed-Solomon ECC
BCH	Bose-Chaudhuri-Hocquenghem ECC
PSNR	Peak Signal to Noise Ratio
NAND flash	Flash memory named after Negated-AND logical gates
SMART	Self-Monitoring, Analysis and Reporting Technology
OOB	Out Of Band
SCSI	Small Computer System Interface

Manuscript received September 15, 2013; revised December 31, 2013 and February 09, 2014; accepted February 10, 2014. Date of publication September 24, 2014; date of current version February 27, 2015. This work was supported in part by NSF grants 1350766, 1124813, and 0937875. Associate Editor: W.-T. K. Chien

The authors are with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA (e-mail: xuxin@gwmail.gwu.edu; howie@gwu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2014.2354931

NOTATIONS

a, b, c, d, e	Coefficients
y_{model}	The modeled RBER
y_{real}	The measured RBER
N	The total number of bits in a codeword including data and metadata
n	The codeword length in bits
p	The number of parity bits
m	The number of user data bits
B_{cw}	The number of user data bits in a codeword
i, j	The horizontal and vertical positions of image pixels
$p(i, j)$	The pixel value at location (i, j) in the fault injection output
$P(i, j)$	The pixel value at location (i, j) in the reference image
H, V	The length, and width of image in terms of the number of pixels

I. Introduction

CURRENTLY, Solid-State Drives (SSDs) are used in a broad range of computers from mobile devices, personal computers, workstations, to supercomputers. Enterprise SSDs have been widely used in many data centers [1]. Compared to magnetic hard drives, SSDs built upon NAND flash memories have significant advantages in input-output (I-O) throughput and bandwidth [2], [3]. These flash drives are often considered to be rugged storage systems because they are shock and vibration resistant, and can work in a wider temperature range, thanks to the fact that SSDs do not contain any moving components like rotating platters and seeking arms as in magnetic hard drives.

Unfortunately, flash drives are error prone. Their lifetimes are usually limited by a certain number of program-erase (P-E) cycles. They are also subject to soft errors, which can be induced by various sources such as read or program disturbance, data retention, etc. The resultant raw bit error rates (RBER) can be as high as 10^{-5} for multi-level cell (MLC), or 10^{-7} for single-level cell (SLC) [4]. In contrast, the RBER for magnetic hard drives can be much lower at less than 10^{-9} [5].

The common practice to address the problem of high error rates is to employ error correcting codes (ECC) with the goal of ensuring 100% correctness on the data level [6]. For example, one well-known ECC is the Hamming code [7] that is designed for single-error correction and double-error detection (SECDEC) for cache or main memory. Unfortunately, flash memory has very high bit error rates, and requires stronger ECC, which leads to a higher overhead in area, performance, and energy. For example, the area overhead for parity bits in an 8 KB page can go from 5% to 12% [8]. Some ECC algorithms (e.g., low-density parity-check code (LDPC)) may incur even higher area overhead (e.g., up to 100% in [9]). The performance overhead for ECC decoding, a procedure required by every read operation, contributes to 38% of the overall access time [10]. The NAND flash memory with ECC can consume seven times more energy in read access, and four times more energy in write access, compared with ones without ECC [11].

In traditional storage systems that consist of magnetic hard drives, these overheads are less of a problem. A read operation in magnetic hard drives may have a long seek latency, which mitigates the performance degradation from the ECC decoding. However, this is not the case for SSDs. The read performance is critical for flash memory, as it is one of the major driving-forces for their wide adoption. Any performance degradation would not be desirable. Also, given the high price of flash chips, one would want to limit the areas that are used to store the error correction data. Furthermore, as technology scales, the recent trends have pointed to the need for more powerful ECCs in the future generations of flash memories [12], [13], resulting in even more significant overheads.

In this work, instead of proposing more powerful ECC algorithms, we take a different route to investigate the possibility of data-level error tolerance in flash memories. The key observation that has motivated this research is that the requirement for data integrity of storage systems may be too strict for certain types of applications (e.g., artificial intelligence, and multimedia processing). Errors in these applications do not necessarily affect software output at the user perceivable level. In this case, applications can be considered as correctly executed on the application level, even with errors. Based on this observation, applications with inherent error tolerance may not need very strong ECC to maintain reliability. For those applications, ECC strength can be lowered to reduce the overhead in performance, energy, and area. A new SSD architecture can be designed towards this goal. To effectively utilize this capability, the approach should be able to adjust ECC strength according to the error rate of the underlying SSD, and the error rate required by applications. To do so, there are three fundamental issues that must be understood thoroughly.

For the first issue, the new architecture should be able to accurately estimate actual error rates of SSDs after the systems have been deployed. This estimation is the basis for making decisions on adjusting ECC strength. However, this is a difficult task. There is no available model that allows us to estimate error rates after SSDs are deployed in systems. Current research usually focuses on measuring the RBERs of flash chips in laboratory environments. These RBERs cannot be used to estimate error rates of SSDs that are visible to applications. This limit is mainly because of two reasons. First, the error rates of individual chips may vary due to manufacturing process variations [14], and it is difficult to derive an accurate estimation of SSD error rate based on existing measurements on flash chips. Second, the SSD error rate is affected by the frequency of read, write, and erase operations.

This information must be collected at runtime.

For the second issue, the error tolerance capability of applications must be evaluated quantitatively, i.e., the target error rates required by applications must be obtained. This information, combined with estimated SSD error rates, can then be used for adjusting ECC strength. Applications may have different error tolerance capabilities. Even if some applications (e.g., multimedia applications) are known as error resilient, the capability of such resiliency must be quantitatively evaluated to confidently adjust ECC capabilities. This evaluation requires extensive experiments based on various representative applications.

For the third issue, it is important to quantitatively understand the potential benefits of utilizing the data-level error tolerance of applications. This quantification requires an evaluation of the impact of various ECC algorithms on performance, area, and energy. This information is the key to making the tradeoff between ECC capability and overhead.

Therefore, before implementing a new architecture, we need to carry out three critical tasks: 1) modeling RBERs for SSDs, 2) analyzing the error resiliency of applications, and 3) quantitatively studying the benefits and overhead of ECC algorithms in SSDs. We summarize our solutions to these tasks as follows.

- We model RBERs at the disk level to quantitatively analyze unique error behaviors in SSDs. This model can be used by operating systems to estimate the current error rate of the underlying SSD on-the-fly. To our knowledge, this is the first attempt to construct statistically accurate RBER models on real-world SSDs, extending prior works on analyzing RBERs at the flash chip level [15]–[17].
- We investigate data-level error tolerance for various applications under different error rates estimated by RBER models at the disk level. We identify the correctness requirements for different application data by analyzing fault

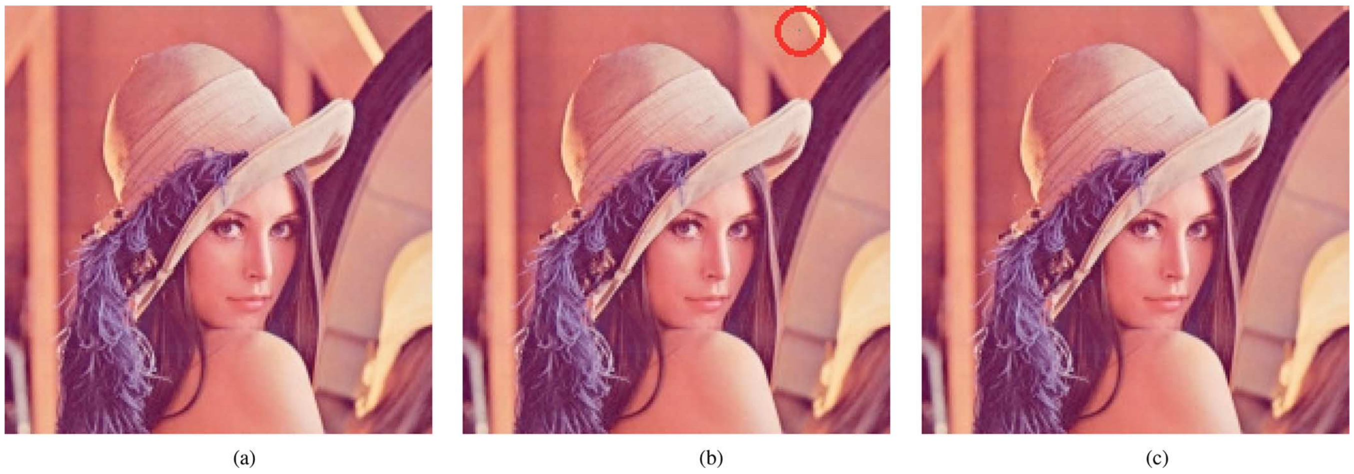

Fig. 1. Bitmap images with different error rates. Fig. (b) contains about 100 errors, but only one is visible as highlighted by the red circle. Figs. (a) and (c) have no visible distinctions. (a) Original image with no error, (b) Error rate 10^{-5}, (c) Error rate 10^{-7}.

injection results. We find that, for some data, it is possible to achieve the data-level error tolerance at the raw error rate of SSDs without using any ECC protections. Thus, one can potentially relax the correctness requirements of non-critical data for high-performance flash storage systems. These findings are essential for utilizing data level error tolerance capabilities.

- We explore the potential benefits of relaxing the requirements of data-level error tolerance. We conduct experiments by extending an SSD simulator [18] to quantify the overheads of various ECCs with different strengths. The results show that, for a number of benchmarks, ECC in flash storage can impose over 120% access latency, and up to 6 times the amount of energy consumption.

In this paper, we focus on studying the above three issues based on quantitative models and evaluation. The models and results can help to evaluate the feasibility and potential benefits of leveraging the error resiliency of applications. We discuss a new SSD architecture, SoftFlash, that takes advantage of the error resiliency of applications based on the results of our study. We discuss the major issues of designing and implementing SoftFlash.

This paper is organized as follows. Section II explains the data-level correctness in SSDs, and the proposed framework. Section III presents our approach for RBER modeling at the disk level. Section IV discusses various ECC codes and their overheads in terms of area, performance, and energy. Section V presents the data-level error tolerance, and the experimental results for a number of data-intensive applications in flash storage. We conclude in Section VII.

II. SoftFlash Architecture

A. Data-Level Error Tolerance

Ideally, data correctness should be preserved at all cost. However, some data are not as important as others. Different files such as binary executables, text files, multimedia, virtual machine images, backup and log files, etc., have different levels of importance when they are used in applications. Even within a file, not every bit is important. Some soft errors that occur in a certain portion of a file may not become visible to users. If one carefully examines the requirements of the data correctness within the context of applications, there are many potential cases where data are not required to be very accurate. Image storage is one of these examples. Fig. 1 shows the bitmap images with no error, as well as with an error rate of 10^{-5}, and 10^{-7} respectively. These two error rates usually represent the error rates at the end of the lifetimes of two major types of flash memory. The single-bit flip errors are randomly injected into those images with the specified error rates. Compared with the original image, the one with the error rate of 10^{-5} has a barely noticeable difference, a very small blue dot as marked by the red circle. The other one with the error rate of 10^{-7} does not have any visible difference from the original image because the bitmap image consists of many small pixels, and errors in just a few of the pixels may not change their color significantly. Even in the case that an error changes the color of some pixels, a user may not notice the differences because pixels are small. Clearly, in this example, the error rate can be as high as 10^{-7} without significantly affecting the correctness of an image, at least as long as the errors are not clustered. In comparison, the current standard for the acceptable error rate is 10^{-16}. This strict requirement significantly underestimates the inherent error tolerance capability in many applications. However we must note that there are applications involving image storage under which very high standards may be critical, such as legal matters.

This concept is related to soft computing, which applies to a wide range of applications, including artificial intelligence algorithms that are tolerant of approximate data or computations [19], multimedia applications [20], [21], and data mining [22]. Previous research investigated low cost error protection and detection mechanisms in microprocessor architecture, memory, and programming languages [23]–[26]. In this work, we focus on the data error tolerance in flash storage systems.

To make our discussion clear, we define the data-level error tolerance as the data in storage systems being sufficiently correct to ensure that applications can produce acceptable results. That is, although the errors may have occurred in the data in storage systems, the data will be deemed to be correct as long

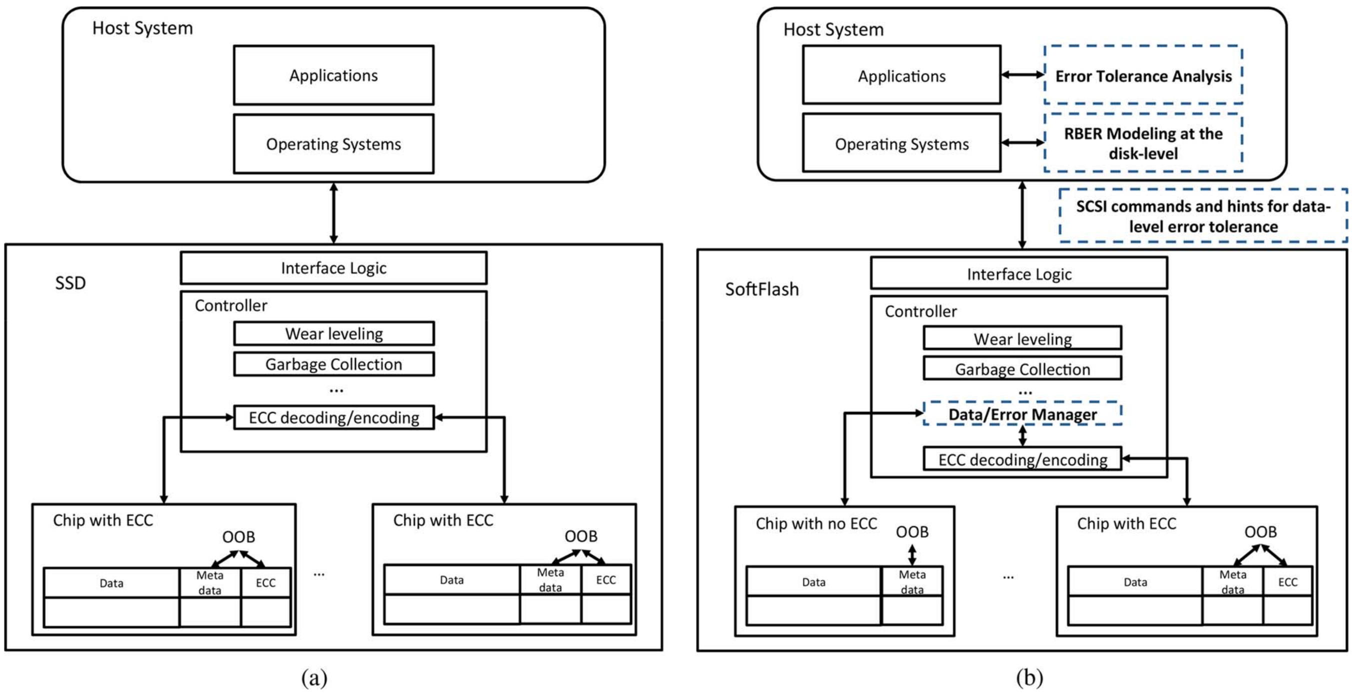

Fig. 2. Two SSD Architectures; the dotted rectangles highlight new SoftFlash components. (a) Traditional SSD. (b) SoftFlash.

as the application outputs are acceptable to users. The data-level error tolerance can be leveraged to achieve a high reliability of flash storage systems without using strong ECCs. Therefore, the ECC overhead in terms of performance and energy can be reduced. Note that the data-level error tolerance is not simply lowering the requirements of the application reliability. It is a tunable specification that can be adjusted by users according to their specific reliability requirements. It cannot be easily achieved. To allow users to make the tradeoff, systematic studies are required to understand the data-level error tolerance. The details of the study are described in Section V.

B. Solid-State Drives, and Soft Errors in Flash

As shown in Fig. 2(a), a traditional SSD is composed of flash memory packages, a flash controller, RAM, and interface logic.

Note that each flash page has an out-of-band (OOB) area, e.g., 256 bytes for a 2 KB page, or 512 bytes for a 4 KB page. The OOB area is used to hold metadata such as error-correcting coding (ECC) bits.

NAND flash can be categorized by the number of bits stored in each cell: single-level cell (SLC) flash that holds one bit per cell, multi-level cell (MLC) that holds two bits per cell, and triple-level cell (TLC) that holds three bits per cell. The specifications of SLC, MLC, TLC [27], [28], and magnetic hard drives [29] are shown in Table I. TLC has not been widely used in SSD products, and therefore their data are limited. The erase latency of TLC is estimated based on the read and write latencies of TLC and MLC. SLC is often used in high-end products, because of its better performance and longer lifetime. MLC and TLC, on the other hand, is mostly used for low-end and middle-level SSDs. In this paper, we focus on MLC NAND, as it is the most popular flash memory in the current market.

TABLE I
SSD AND MAGNETIC HARD DRIVES

	SLC	MLC	TLC	Magnetic Hard Disk
Bits per Cell	1	2	3	N/A
Price($/GB)	5	1.5	1	0.1
P/E Cycles	100k	5k - 10k	1K	N/A
Read Latency(μs)	25	60	100	8500
Write Latency(μs)	250	800	2,100	9500
Erase Latency(μs)	700	1,500	3,000	N/A
ECC (512 bytes)	1 bit	4+ bits	30 bits	1 bit
RBER	$<$ 2.5E-7	$<$ 6E-5	$<$ 1E-3	$<$ 1E-9

There are three types of operations in flash memory: read, write, and erase. The read operation fetches data from flash memory, the write operation stores new data to clean memory cells, and the erase operation resets the cells to the clean state. Different from direct overwriting in magnetic hard drives, original dirty data must be erased before new data can be written into flash memory. This unique operation is strongly correlated to high error rates in flash memory.

Flash memory usually has two types of errors: hard errors, and soft errors. Hard errors are permanently damaged memory cells that are more likely to occur at the end of the lifetime. Usually, when they occur, the flash chip has to be replaced. The lifetime of flash memory is defined by the number of program-erase cycles. As shown in Table I, the MLC's lifetime is usually only a tenth of the SLC's lifetime. Soft errors are transient faults that do not permanently damage memory cells. Soft errors occur during the entire lifetime. They are caused by a variety of factors such as disturbances from read, write, or erase operations, and charge loss during data retention. Research [16] also shows that, as the P-E cycle increases, the soft error rate in the flash memory increases exponentially. In particular, MLC devices have the raw bit-error rate at 6×10^{-5}, which is orders of magnitude worse than SLC devices. Note that TLC has recently been developed to store three bits of data per cell, and is expected to have fewer P-E cycles (only about 1 K) and higher error rates (10^{-3}) than MLC. Comparatively, magnetic hard disks usually have lower error rates (10^{-9}). Currently, SSD manufacturers usually aim to achieve a 10^{-16} uncorrectable BER (UBER) for enterprise SSDs, as an industry standard [30]–[33]. Because the raw bit error rates of flash memory are much higher, ECCs are employed to achieve this target error rate (10^{-16}).

C. SoftFlash Architecture

In SSDs, various ECC algorithms are implemented in the controllers to detect and recover soft errors. When there is a write access to an SSD, the ECC information will be encoded and stored in OOB. When there is a read access to an SSD, the ECC information in OOB will be decoded and verified. If the ECC detects an error, error correction procedures will be enabled to correct error bits. If the number of error bits is beyond the ECC correction capability, the operating system will be notified that these data are corrupted.

For many data-centric applications that are inherently error tolerant, we can relax the reliability requirements for SSDs, and take advantages of this opportunity to reduce the overhead costs imposed by the ECC mechanisms. Current operating systems and SSD architectures do not provide such support. To achieve this change, during the program execution, we need to obtain 1) the requirements of data-level error tolerance for data being used in applications, and 2) the current raw bit error rate of the underlying flash storage system. With this information, the operating system can instruct the flash storage system to provide adequate ECC protections.

Towards this goal, we proposed a new type of solid-state drive, SoftFlash, which tolerates soft errors up to a threshold with no or less powerful ECC, while maintaining the correct execution of the application. The architecture is shown in Fig. 2(b), while the traditional SSD is shown in Fig. 2(a). In traditional systems, operating systems and SSDs do not differentiate the data-level correctness requirements; all data are sent to flash memory with the same ECC protection. In comparison, the new architecture will process and send the data to SoftFlash in three steps: 1) the current error rate of the SSD is estimated by the operating system using an error model (Section III); 2) the operating system identifies the requirements of the data correctness, and provides this information to the SSD as a part of SCSI commands or hints (Section V); and 3) the underlying data manager in the SSD controller dispatches these data with adequate ECC protection based on the correctness requirements.

To keep the design and production cost low, the implementation of SoftFlash can be done within operating systems and firmware in SSD controllers without modifying current hardware. The system designers need to integrate the RBER model into the operating systems, and to specify the targeted error rate and ECC strength for application data. This integration can be done by developing a kernel module, and modifying the file system functions related to meta-data operations. Then, this information in meta-data should be sent to SSDs with write or read requests. This communication between operating systems and underlying SSDs is done by modified SCSI commands. Next, SSD manufacturers need to modify the firmware of SSD controllers to integrate a function that sends data to chips with proper ECC depending on the information sent through SCSI commands. These implementations do not require hardware modifications to current systems, and therefore can be deployed with relatively low cost. The details are discussed in Section VI.

SoftFlash mainly leverages the data-level error tolerance from applications to relax the reliability requirement for flash memory. Our work is related to two projects that also studied ECC overheads in flash memory: Kgil *et al.* [13] propose the hardware supports to accelerate the decoding procedures by parallelizing the computations, and Wu *et al.* [10] propose a BCH code with an adjustable code length. Their research is orthogonal to this study, and can be easily integrated into SoftFlash. In addition to the improvement in read performance, SoftFlash also aims to achieve significant savings in storage area and energy consumption.

III. Error Rate Models for Flash Storage

Before we go into data-level error tolerance in flash storage systems, we first need to understand the error characteristics of flash memory. Here we will use the bit error rate (BER) as a major metric. The raw bit error rate (RBER) is the ratio between the number of bits with errors and the total number of bits that have been checked by ECC. The uncorrectable bit error rate (UBER) is the ratio between the number of bits with errors that cannot be corrected by ECC and the total number of bits that have been checked by ECC. While prior research works have started the efforts of measuring and modeling flash error rates, they are mostly limited to individual flash chips [16]. In this section, we will begin with the discussion on RBERs at the chip level, and propose a black-box modeling approach for RBER at the disk level. The unique feature of our approach is that our method aims at bit error rates at the disk level, and leverages self-reported statistics from real-world SSDs, namely SMART attributes, to build a realistic, accurate estimation.

S.M.A.R.T. or SMART stands for Self-Monitoring, Analysis, and Reporting Technology [34], which is developed to monitor the health of magnetic hard drives, and to alert users when faults occur or a failure is imminent. In the event that a catastrophic failure is predicated, users can take necessary actions to avoid data loss, e.g., copying data to another storage, and replacing the indicated drive. For magnetic hard drive reliability, SMART attributes can be used for statistics (e.g., power cycle count), environmental monitoring (e.g., temperature, shock and vibration), drive performance measurement (e.g., spin up time), and error reporting (e.g., read-write error rate, and reallocated sections count). For each attribute, the raw, and normalized values are reported, and compared against pre-defined thresholds. If a threshold is exceeded, chances are high that the drive may encounter a failure in the future. Note that different manufacturers may define a slightly different set of attributes and interpretations.

For SSDs, a few SMART attributes are added to measure flash specific characteristics, including average, minimum, and maximum P-E counts; read, program, and erase errors; and the remaining lifetime with regard to the maximum P-E cycle in the

TABLE II
SMART ATTRIBUTES

Selected Magnetic Hard Drive Attributes	
Power Cycle Count	The total number of power on/off cycles for a magnetic hard drive
Spin Up Time	The mean time to spin up the spindle from still to full speed
Reallocated Sectors Count	The count of the sectors that have been replaced (remapped) with spare sectors
Read Error Rate	Hardware errors when reading data from a disk
Write Error Rate	Hardware errors when writing data to a disk
Temperature	Disk internal temperature
Selected SSD Attributes	
Read Failure Block Count	The number of blocks with flash read failures
Program Failure Block count	The number of blocks with flash program (write) failures
Erase Failure Block Count	The number of blocks with flash erase failures
Total count of error bits	Total number of corrected errors
Average erase count	The average erase count of all blocks
Remaining lifetime	Estimated remaining lifetime in percentage based on erase count

specification. Table II lists a number of common SMART attributes for magnetic hard drives and SSDs. In this work, we collect the SMART attributes on OCZ Vertex SSDs [35] to build error rate models at the disk level.

RBER at the chip level: At the flash chip level, soft errors are mostly caused by three factors: wear-out, disturbance, and data retention. Wear-out is a well-known failure mode that affects the lifetime of flash memory. P-E cycle can be used for a good indicator for wear-out. Chimenton *et al.* [15], and Sun *et al.* [17] measured the effect of P-E cycles on the raw bit error rate of flash memory, and showed that RBER can be modeled using an exponential function of P-E cycles.

RBERs at the chip level gradually change through the chips lifetime. Previous studies have measured RBERs of flash chips from several manufacturers [16]. Flash chips are repeatedly written till P-E cycles reach 10 000. Results show that RBERs gradually increase from about 10^{-9} to 10^{-6}. We measure RBERs in SSDs, and compare them with chip-level data. As shown in Fig. 3, for a relatively new SSD with a P-E cycle of under 1 000, the measured RBERs at the disk level show a range of 10^{-11} to 10^{-10}. For flash chips with the same P-E cycles, RBERs are from 10^{-8} to 10^{-7}. Because all chips and SSDs are in their early lifetimes, their measured RBERs are significantly lower than 6×10^{-5}, which is the maximum RBER for MLC, as shown in Table I. However, there is a clear difference (four orders of magnitude) between RBERs at the disk level and the chip level. We believe several factors contribute to the large disparity in RBERs at the disk and chip levels. First, RBERs at the chip level may be closely tied to the specific chip types and manufacturers. Second, due to process variance [14], RBERs of flash chips may vary even if they are from the same product line. This variation further limits the applicability of prior RBER models. Furthermore, the mechanisms causing errors are more complicated in SSDs than in flash chips after being deployed in real systems. Error rates of flash chips are affected by a combination of dynamic factors (e.g., read, write, and erase operations). SSDs contain a number of flash chips managed by SSD controllers using various techniques (e.g., wear-leveling). This variability further complicates the relationship between these factors and error rates. Therefore, a wide-scope approach at the SSD level is needed to accurately model RBERs.

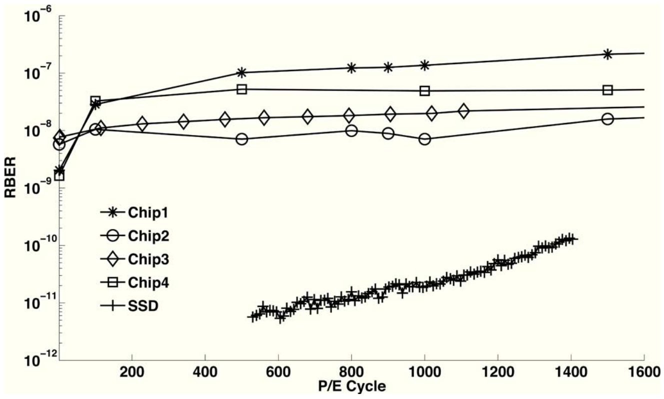

Fig. 3. RBERs at the chip level and the disk level.

RBER modeling at the disk level: Here we study error characteristics at the disk level, and propose to utilize the black-box approach to build a model of RBER at the disk level. This approach is particularly suitable for SSDs because their internal designs are kept confidential. Such an approach makes it possible to model the error behaviors of the SSDs, and to mathematically capture the interactions among internal drive components.

There are many factors that may cause errors in flash memory, such as wear-out, and read disturbance. To balance the model accuracy and the complexity, we choose the P-E cycles, and the total number of read sectors as the model parameters. Because RBERs of SSDs vary upon the SSD utilization, these factors must be dynamically captured to accurately estimate RBERs in real systems. Clearly, the P-E cycle is a major indicator for wear-out that causes errors, and therefore shall be included in our model. Because the number of P-E cycles already counts the program operations, we do not have to explicitly model the program disturbance. We also include the read count to characterize SSD utilization that cannot be directly captured by P-E cycles. It is another factor that may cause errors (e.g., read disturbance). Because data retention is closely related to the usage of users and applications, this effect can be modeled implicitly using P-E cycle and read count.

Note that there are some other factors such as temperature that are also related to error rates. We do not include the temperature in the current model for the following reason. In most common circumstances, unless the change in temperature is extreme, the effect of temperature on error rates would not be observed in the short time interval. Previous works on measuring the temperature effect on retention error rates usually conduct experiments for weeks or months. In contrast, our model can be updated in a much shorter interval (minutes or hours) to capture dynamic changes. To adapt to dynamic changes in RBERs that are caused by other factors that are not included in our model, we periodically monitor RBERs, and measure the accuracy of the model. If the accuracy decreases to a certain level due to changes in these factors, the model can be re-built to reflect these changes.

In this paper, we use the black-box modeling approach to evaluate different functions to build a model. In particular, we construct and evaluate three types of models: a linear model, a quadratic model, and an exponential model. In the following models, PE_Cycle represents the number of P-E operations that have been performed, $Read\_Count$ represents the total number of sectors that have been read so far, and RBER is the estimated raw bit error rate and the model output.

To evaluate the model fitness, we utilize the stepwise algorithm [36] that iteratively evaluates all possible combinations of given items in the model, and identifies the most suitable model configuration. The metric for comparison between different models is Akaike information criterion (AIC) [37]. AIC is based on information theory to quantitatively describe the model quality that is defined as $2 \times (number\ of\ variables) - 2 \times \ln(maximum\ likelyhood)$. A lower AIC value indicates a better fit. The stepwise algorithm iteratively calculates the AIC value for each model configuration to help choose a suitable configuration.

The Linear model assumes a linear relationship between the observed variables, PE_Cycle and $Read\_Count$, and the estimated variable, RBER. This model uses a first-order polynomial function that can be represented as

$$RBER = a + b \times PE\_Cycle + c \times Read\_Count \quad (1)$$

The Quadratic model uses a second-order polynomial function to better model the error behaviors that cannot be fully captured by the linear model. It can be represented as

$$RBER = a + b \times PE\_Cycle + c \times PE\_Cycle^2 + d \times Read\_Count + e \times Read\_Count^2 \quad (2)$$

We utilize the stepwise algorithm to evaluate the fitness of different model configurations. We find that this model can be further reduced to only three coefficients with similar AIC and MRE. The reduced quadratic model is represented as

$$RBER = a + b \times PE\_Cycle + c \times Read\_Count^2 \quad (3)$$

Here we employ an exponential function on PE_Cycle only because using an exponential function on $Read\_Count$ will lead to very large errors. The model can be written as

$$RBER = a + b \times PE\_Cycle^2 + c \times exp(PE\_Cycle) + d \times Read\_Count + e \times Read\_Count^2 \quad (4)$$

TABLE III
MODEL FITNESS

No.	Model	MRE	AIC
(1)	Linear	0.4879	121.8
(2)	Quadratic	0.2995	55.1
(3)	Reduced Quadratic	0.3023	55.6
(4)	Exponential	0.1306	26.8
(5)	Reduced Exponential	0.1393	27.9

We utilize the stepwise algorithm to evaluate the fitness of different model configurations. We find that this model can be further reduced to only four coefficients with similar AIC and MRE. The reduced exponential model is represented as

$$RBER = a + b \times PE\_Cycle + c \times exp(PE\_Cycle) + d \times Read\_Count^2 \quad (5)$$

In this study, we use two OCZ Vertex SSDs [35], which provide the SMART readings on average P-E cycles, and bit error counts that can be directly utilized in our modeling process. Our models are not limited to a specific type of SSD. As long as the SSD manufacturers report this status information in SMART attributes, we can easily train the model to adopt to their new SSDs.

To exercise the SSDs, we use a synthetic microbenchmark that iteratively writes a 512 GB file and reads it back. The S.M. A.R.T. attributes are measured after every 1TB write and 500 GB read to the SSD. We collect the data on two OCZ Vertex SSDs for about two weeks. For each 1TB write, we find that the P-E cycles of the SSDs will increase by around 10. We use 80% of the measured data to train the model, and the remaining 20% of the data to verify the model for each SSD respectively. The 80 and 20 separation is chosen to balance the distribution between training dataset and evaluation dataset. The model accuracy is evaluated with mean relative error (MRE), which is calculated as

$$MRE = \frac{|y_{model} - y_{real}|}{y_{real}}. \quad (6)$$

Table III lists the model fitness for different models. As one can see, the non-linear exponential model has a much smaller AIC and MRE than linear and quadratic models. Using an exponential model can reduce MRE to about 13%. A low AIC (26.8) indicates that the exponential model has better fitness than other linear and quadratic models. The stepwise algorithm helps us to identify the important items in the model, and reduce the model complexity. The reduced quadratic and exponential models are able to achieve similar AIC, and MRE with only three, and four coefficients respectively. Fig. 4 plots the estimated RBERs using the exponential model, where one can see that the measured values (dot) match closely with the estimated RBERs (line). We will use the exponential model in the following sections.

IV. ERROR CORRECTION CODES FOR FLASH STORAGE

Given their error-prone nature, flash memories use various error correction codes (ECC). For example, single-error-correct-double-error-detection (SECDED) coding schemes such as

TABLE IV
ECC COMPARISON

	Hamming code	RS	BCH	LDPC
Code	(4120, 4096)	(4240,4096)	(4304, 4096)	(8192, 4096)
Correctable error bits	1	8	16	4096
Storage overhead (%)	0.6	3.3	4.8	100 [9]
latency (μs)	34.5 [38]	44.3 [38]	41 [10]	41 [39]
Energy/byte (read/write) (pJ)	15/17 [38]	16/18 [38]	18/20 [11]	24/27 [9]

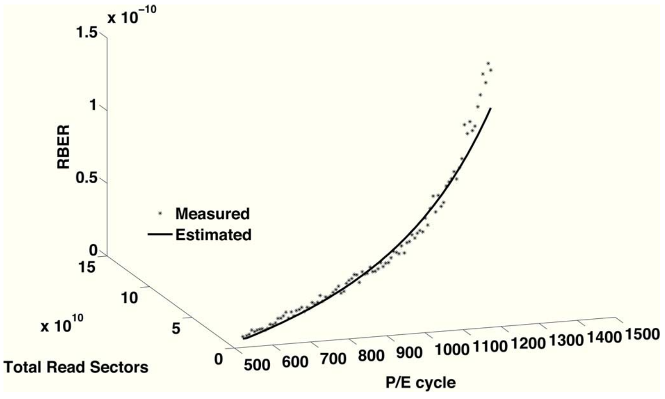

Fig. 4. Measured and Estimated SSD Error rate on OCZ Vertex SSD.

Hamming Codes may be sufficient for SLC. However, as technology scales, flash density continues to increase, and stronger ECCs are required to ensure data correctness.

In this section, we discuss four common ECCs in flash storage. Based on our error model, and measurements on real devices, we estimate the UBERs of SSDs under various types of ECC schemes. In addition, we analyze ECC overheads in terms of area, latency, and energy consumption. Those data will help provide a deep understanding of reliability characteristics of SSDs.

A. ECC Schemes

Here we briefly review four types of ECCs that are commonly used in flash memory. For consistency, we use n as the codeword length in bits, p as the number of parity bits, and m as the number of user data bits.

Hamming codes [7] are designed for single-error correction and double-error detection (SECDEC). Hamming codes can be presented as (n, m). The advantage of Hamming codes is that the encoding and decoding latency and energy are relatively low, but it comes with a limited single-error correction capability.

Reed-Solomon (RS) codes are used to handle multiple-bit corruption in flash memory, especially for MLC. RS codes can be represented as (n, m) with s-bit symbols, where s is the size of the symbol. The number of parity bits is usually $2t$, where t is the maximum number of correctable errors. Compared with Hamming codes, RS codes have higher latency and power consumption, and can handle the errors that occur in groups.

Bose-Chaudhuri-Hocquenghem (BCH) codes, based on Galois Fields, have lower encoding and decoding latency than RS codes. Binary BCH is usually represented as (n, t, m), where the required number of parity bits is usually $log(n) \times t$. One disadvantage of BCH codes is that, as the number of correctable error and code length increase, the decoding latency and area overhead are dramatically increased [12], [13]. Because read performance is critical to SSDs, this latency issue might become an issue.

Low-Density Parity-Check (LDPC) codes are a class of error correcting codes that can provide stronger error correction capabilities [41]. It has been shown that LDPC doubles the error correcting capability of BCH codes with the same number of parity bits when it is implemented in TLC flash memory [8]. Compared with other coding schemes, LDPC has a stronger protection capability, but a higher energy consumption.

We sum up the four types of ECCs in Table IV. We adopt the numbers from [11], which implements a single error correcting hamming code (536, 512). In this implementation, flash memory consumes 4 pJ/byte, and 9 pJ/byte for reads, and writes without ECC, respectively; and about 15 pJ/byte with ECC protection with an efficient embedded microprocessor. We estimate the energy consumption of both RS and BCH codes based on the results in [38].

B. ECC Performance and Overheads

As the next step, we utilize the experimental data obtained in Section III to estimate uncorrectable bit error rates under different ECC strengths. The relationship between raw bit error rates (RBER) and uncorrectable bit error rates (UBER) can be represented by

$$UBER = \sum_{E+1}^{N} \left(\begin{smallmatrix} N \\ n \end{smallmatrix} \right) \times RBER^{n} \times (1 - RBER)^{N-n} / B_{cw} \tag{7}$$

Usually, a codeword contains 512 bytes of user data, and a number of parity bits. Given an ECC scheme, and our proposed model, we are able to estimate UBER. Note that, with ECC protection, UBER is the actual error rate that is visible to applications. The results are shown in Fig. 5. SSD manufacturers often quote 10^{-16}, or even higher as the UBER in the specifications [33], [42], [30]. While RBER increases gradually during the lifetime of SSDs, our data show that a weak ECC can meet reliability requirements in the early lifetime of the SSD. However, as the device gradually wears out, soft error rates will be higher, and therefore stronger ECCs are required to provide a better protection, which as we will show later would incur significant performance and energy overheads.

To evaluate the performance overhead for different ECCs, we conduct experiments based on the SSD simulator SSDSim that was developed in Microsoft Research [18]. We model the latency of BCH codes based on the implementation in [10], and estimate the latency for other ECC schemes based on relative protection strengths. For example, we assume that the decoding

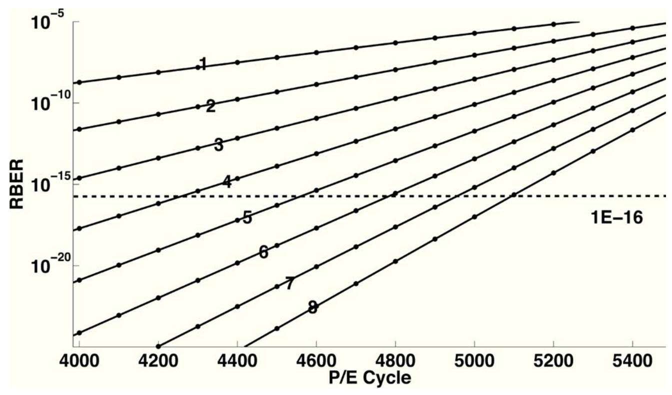

Fig. 5. UBER vs. P-E cycles, where lines represent ECCs with different capabilities varying from 1 bit-error to 8 bit-errors per 512 bytes.

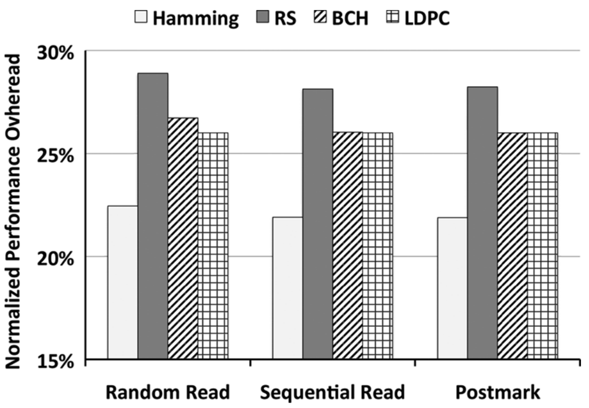

Fig. 6. Performance overheads for different ECCs.

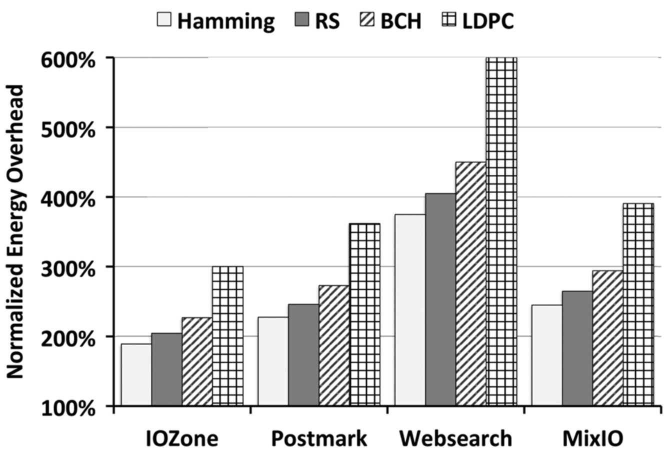

Fig. 7. Energy overheads for different ECCs.

latency of BCH and LDPC are comparable, as shown in [39]. In the tests, we use two microbenchmarks, sequential and random read, as well as Postmark that mimics the workload on an email server [43]. We focus on the read performance, and normalize the numbers to those without ECCs. The results are shown in Fig. 6. As we can see, all four codes may add over 20% overhead to read performance, and BCH and LDPC codes win out because they provide a reasonable tradeoff between access latency and protection strength.

We further estimate the energy overhead costs of running a number of workloads, including the MixIO microbenchmark that consists of a sequence of sequential read and write requests, IOZone [44] that has the most writes, WebSearch trace that has the most reads [45], and PostMark that also has a mix of reads and writes. The energy overhead is estimated based on the data in Table IV, and the traces generated by SSDSim. Similarly, the numbers are normalized to without ECC schemes. As shown in Fig. 7, one can see 1) all ECC codes would incur non-trivial energy overheads of at least twice as much; 2) a more complex ECC scheme like LPDC may consume up to six times more energy than without any data protection; and 3) read-intensive workloads will encounter much higher energy overheads than write-intensive workloads, mostly due to ECC decoding operations.

It is important to note that ECC overheads can be highly dependent on hardware and software implementation. Ideally, different ECC schemes shall be implemented and evaluated on the same platform. We believe that the numbers presented in this work represent a reasonable estimation of each ECC design. The results demonstrate the need for a flexible, reliable, yet balanced ECC design for flash storage. We leave hardware implementation and measurement to future work. In summary, various ECC schemes used in today's flash storage systems incur non-trivial overhead in I-O performance and energy consumption. The key challenge is how to design high-performance flash storage systems with suitable ECC schemes that satisfy application reliability requirements, and minimize the overheads.

V. Data-Level Error Tolerance in Flash Storage

In this section, we explore the data-level error tolerance in flash storage. In particular, we try to answer the following questions. Can the correctness requirement for flash storage systems be relaxed for data-centric applications? How will the errors in flash storage systems affect the application outputs? How can we improve the data-level error tolerance? Previous research uses fault injections to evaluate the reliability characteristics of target systems [23], [46], [47]. In this section, we conduct an extensive set of fault-injection experiments, and analyze the characteristics of data-level error tolerance in a number of data-intensive applications.

A. Data-Centric Applications

To evaluate the data-level error tolerance in flash based storage systems, we examine six data-intensive applications from different benchmark suites, including MediaBench [48], PARSEC [49], and MapReduce/Hadoop [50]. The benchmarks cover various fields, such as multimedia processing, artificial intelligence, scientific computing, and cloud computing. Table V lists the parameters for each application. In this paper, we study the errors that occur in the data files of these applications. We consider these files to be critical, and their correctness should be maintained. Note that errors may still occur in the program text, operating system files, and libraries, which is beyond the scope of this work.

For each experiment, we classify the results into five categories, from crash, bad, good, same, to better. An overview of evaluation metrics is shown in Table VI. The metrics with the *original* subscript are the results of fault-free execution or the

TABLE V
CONFIGURATIONS OF EXPERIMENTS

Name	Category	Description	Data (lg/sm)	File (lg/sm)	Metric
Cjpeg	Multimedia	Compress BMP to JPEG images	19GB/697MB	1494/100	PSNR
Djpeg	Multimedia	Decompress JPEG to BMP images	2.7GB/638MB	1494/100	PSNR
Kmeans	Data Mining	Partition n points into k clusters	77MB/7.7MB	1/1	Classification purity
Canneal	AI	Chip design using annealing method	98MB/3.4MB	1/1	Routing cost
MC	Statistics	Monte Carlo methods for π calculation	16GB/16MB	1/1	Relative errors
Text	Web	count occurrence of keywords in files	162GB/1000MB	148/1	Occurrences

TABLE VI
EVALUATION METRICS OF FAULT INJECTION RESULTS

Name	Better	Same	Good	Bad	Crash
Cjpeg	N/A	$PSNR_{fault} = PSNR_{original}$	$RE \leq 5\%$	$RE > 5\%$	no result
Djpeg	N/A	$PSNR_{fault} = Inf.$	$PSNR \geq 50db$	$PSNR < 50db$	no result
Kmeans	$Purity_{fault} > Purity_{original}$	$Purity_{fault} = Purity_{original}$	$RE \leq 5\%$	$RE > 5\%$	no result
Canneal	$Routing\_Cost_{fault} < Roung\_Cost_{original}$	$Routing\_Cost_{fault} = Roung\_Cost_{original}$	$RE \leq 5\%$	$RE > 5\%$	Wrong input format
MC	$EE_{fault} < EE_{original}$	$EE_{fault} < EE_{original} \times 10$	$EE_{fault} < EE_{original} \times 100$	$EE_{fault} \geq EE_{original} \times 100$	no result
Text	N/A	$String\_Count_{Fault} = String\_Count_{original}$	$RE \leq 0.1\%$	$RE > 0.1\%$	no result

ground truth values whenever available, and those with *fault* subscript are the results after fault injections.

The criteria of classifications are specified by studying prior work in each domain. 1) For the artificial intelligence applications canneal, and kmean, we use relative errors (RE) to quantify output quality. It is suggested that this type of application is inherently approximate. The outputs without any errors may be off by more than 15% compared with perfect results, and 5% relative errors on top of the outputs is acceptable [23], [24]. Therefore, in our paper, results are considered as good if RE is less than 5%. 2) For multimedia applications cjpeg and djpeg, PSNR is usually used to measure the output quality. For the image decompression process (djpeg), previous work considers output decompressed images with PSNR greater than 50 db to be good results [23], [24]. We adopt the same criteria for the djpeg. Image compression (cjpeg) is a lossy process, i.e., the output and input images are not exactly the same, and there is a quality degradation in output images. Therefore, we also use the relative errors (RE) at less than 5% as the criteria for good results. 3) Text search is an application similar to the top-k algorithm that is widely used in database. Some top-k algorithms allow a certain level of inaccuracy in results (2%–10% or even higher) to achieve better performance [51], [52]. Here we specify a higher accuracy requirement in the experiments. Results with less than a 0.1% of change in the count of words are good. If any key word is missing from the output, we consider this is a bad result. 4) MC is an application to estimate π. The accuracy is usually defined according to the number of decimal digits. Therefore, the criteria for classification are also defined in a similar way.

We conduct more than 30 000 fault injections on the benchmarks to gain in-depth understanding about the data-level error tolerance. Specifically, kmeans, MC, and canneal are conducted for 1000 repetitions with both large and small inputs. The other three applications, Text, djpeg, and cjpeg, are very time consuming. It takes about 5 to 24 hours for each run in a server, including generating input files, fault injections, and application executions. We have conducted at least 150 repetitions for djpeg, and cjpeg with small inputs, and 1000 repetitions for Text with small inputs. We have also run the tests for 10 times for djpeg, cjpeg, and Text with large inputs.

In this work, we consider that data-level error tolerance can be achieved when the experiments achieve good to better results. It is important to note that in some cases, because bit faults introduce randomness in data, results may be better than original ones.

Image Compression (Cjpeg) from the MediaBench suite compresses the bitmap images into the jpeg format images. Due to its lossy nature, the outputs of jpeg images do not have the same picture quality as the original bitmap images. The peak signal to noise ratio (PSNR) is used to measure the compression quality, and is calculated as

$$PSNR = 10 \times log_{10}(\frac{255^2 \times H \times V}{\sum[p(i,j) - P(i,j)]^2})$$

The jpeg images with PSNR larger than 50 db are acceptable in terms of quality [23]. In our experiments, we compare the PSNR between fault free output images and fault injection outputs, and calculate the relative error (RE) as

$$RE = |PSNR_{fault} - PSNR_{correct}/PSNR_{correct}|\,.$$

If a faulty jpeg image has the same PSNR as the correct jpeg image, this result is categorized as *same*. Note that the fault injection will not generate better images, thus the *better* category is not applicable to this benchmark. A jpeg image with less than 5% relative error in PSNR is considered as *good*. Images with greater than 5% error are considered as *bad*. If the image cannot be recognized by the program, it is categorized as *crash*. In our experiments, we use 100 bitmap images (with the total size of 697MB) as the small input, and about 1 494 bitmap images (19GB) as the large input. The faults are randomly injected among all images. This experiment is conducted about 150 times for small inputs, and 10 times for large inputs; and the average percentage of images with acceptable quality is reported.

Image Decompression (Djpeg), also from MediaBench, decompresses jpeg images to bitmap images. Because decompression is a lossless operation, the output images should have infinite PSNR if executed correctly. In our experiments, decompressed images with greater than 50 db PSNR are considered *good*. Otherwise, it is a *bad* image. If the image cannot be recognized by the program, it is considered *crash*. Similarly, we use 100 jpeg images (638 MB) as the small input, and 1 494 jpeg images (2.7 GB) as the large input. The experiment is conducted about 150 times for the small input, and 10 times for the large input. The final output is the average percentage of the number of decompressed images with an acceptable quality.

K-means clustering algorithms, commonly used for data-mining applications, group the n points with m features into k clusters, so that the points in each cluster have minimum mean distances to the center of the cluster. Here we use the *kmeans* function in MATLAB [53]. In this experiment, we test the inputs with two different sizes: a small input with 10K points and 10 features for each point, and a large input with 100k points and 100 features for each point. The output of this algorithm is the classification result. We examine the changes in classification purity [54]. We compare the classification results with the reference input. Each cluster in the classification output may have several groups of points that belong to different clusters in the reference input. In this case, we consider the group with the maximum number of points to determine the correct classification of this output cluster. Other points in this output cluster are incorrectly classified. Note that, due to the approximate nature of the K-means methods, even without any faults presented, MATLAB cannot classify input points with 100% accuracy. The purity (percent of points in the defining cluster) is usually around 70%. A result with higher purity is considered *better*. We consider that a result with less than 5% purity change is *good*. With the correct output purity, 5% variation does not significantly affect results. The experiments are repeated 1 000 times, and the percentage of acceptable results is reported.

Canneal uses the simulated annealing algorithm to optimize the routing cost of chip design. The input is the network graph in a text file. We use *native* as the large dataset (98 MB, 2.5 million nodes), and *simsmall* as the small dataset (3.4 MB, 100 K nodes). The output is a map containing the locations of all elements. The metric of evaluating the output map is the final routing cost. The final routing cost is calculated using the correct network graph based on the output map. Note that, due to the approximate nature of the simulated annealing algorithm, the output map of the correct execution is not necessarily the best result. If a fault injection result has a smaller routing cost than the correctly executed result, it is categorized as *better*. If a fault injection result has the same routing cost as the correctly executed result, it is categorized as *same*. If the relative error is less than 5%, it is *good*. Otherwise the result is *bad*. The experiments are repeated 1 000 times.

Monte Carlo (MC) methods use random sampling techniques to estimate the results. In this paper, we conduct fault injection experiments using the MapReduce implementation of the π estimator in Hadoop. The program randomly generates points in a unit square plane. Each point can be identified as inside or outside of the circle. Thus, π can be estimated by calculating the ratio of the number of points in the circle versus in the square. The user can specify the number of sampling points, and the number of iterations. A higher precision π can be obtained by specifying a larger number of points, more iterations, or both. In our experiments, we use one million sampling points, and one iteration, as the small input, which results in 16 MB of data. Also, we use 100 million sampling points, and 10 iterations, with 16 GB of data as the large input. The output of this program is the estimated value of π with various precisions. The results are evaluated by the estimation error (EE) to the ground truth value of π. EE is defined as $EE = \pi_{estimation} - \pi_{real}$. Note that even a correctly executed result of MC will have non-zero relative error. Therefore, for this benchmark, the fault injection result with EE_{fault}, that is lower than $EE_{original}$ of the fault-free result, is *better*. The result with ten times $EE_{original}$ is *same*, the result with 100 times $EE_{original}$ is *good*, and otherwise *bad*. The experiments are repeated 1 000 times.

Text search is another MapReduce benchmark in Hadoop that searches for matching strings in text files. Given a specified keyword, the number of occurrences of this keyword will be reported. If a regular expression is specified, multiple string patterns can be found. In this case, it will output sorted numbers of occurrence of all matched strings. We use one file with 1GB size as the small input, and use the entire Wikipedia repository (148 files and 162 GB) as the large input. In this paper, we search for a regular expression, and examine the top 100 frequent strings. If the word count of a string is not significantly changed (less than 0.1%), the search for this string is considered as *good*; otherwise it is considered as *bad*. The result of an experiment is the worst result of the 100 strings. For example, if any string is *bad*, the result of this experiment is *bad*. If a string is not in the top 100 results, it is also considered *bad*. The experiments are repeated 1 000 times with small inputs, and 10 times with large inputs to balance the amount of experiments and the accuracy.

B. Analysis of Data-Level Error Tolerance

To understand error behaviors of different applications, we evaluate all benchmarks on flash storage systems with a range of BER from 10^{-5} to 10^{-8}, representative as the raw bit error rate of flash memory [4], [16]. The results of large inputs are shown in Fig. 8. We found that the results for the small inputs are quite similar, except for K-means. The smaller input in K-means contains much fewer errors, therefore the results are improved. Although small inputs in Canneal also contain fewer errors, which happens because the Canneal input data format is very strict, the small number of errors can still easily affect the input data format, and lead to program crashes.

One can see from Fig. 8 that Cjpeg, Monte Carlo methods, and text searches are relatively robust to errors. It is not surprising that Cjpeg is more robust than Djpeg, because data in

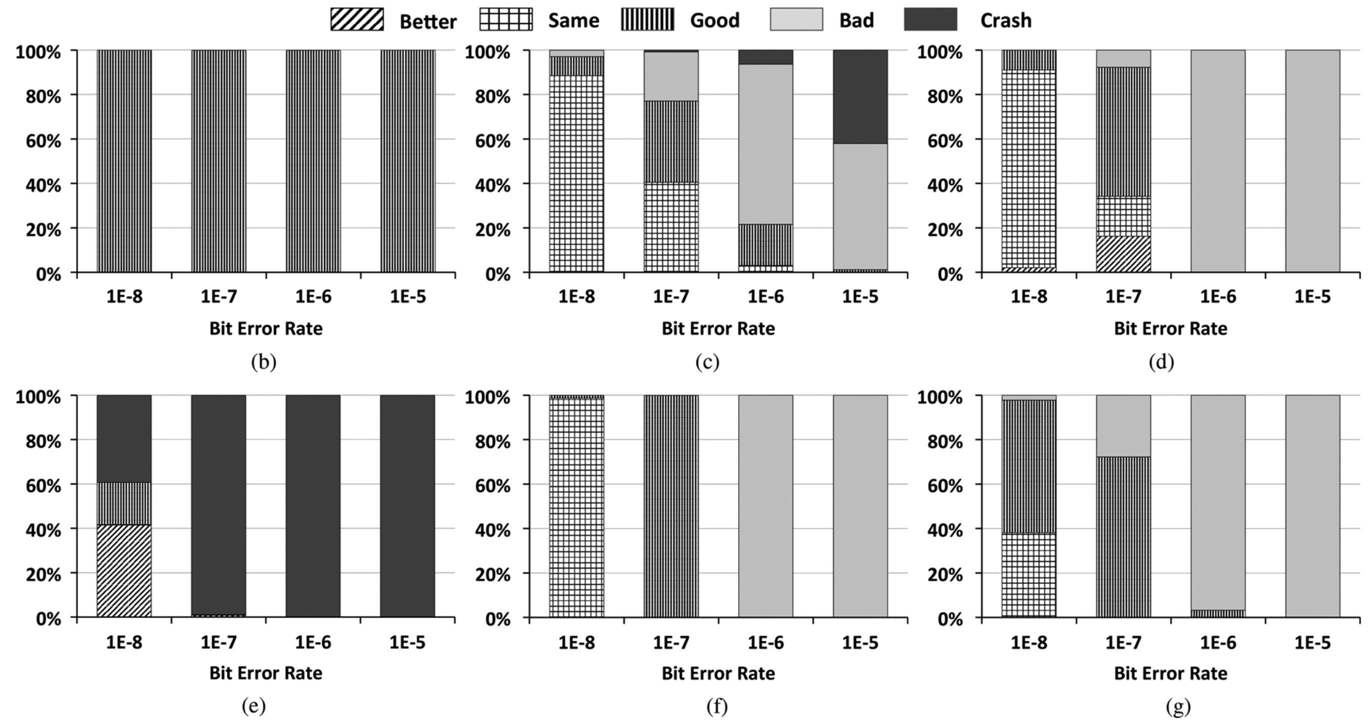

Fig. 8. The data level error tolerance of applications under various BERs. (b) Cjpeg, (c) Djpeg, (d) Kmeans, (e) Canneal, (f) MC, (g) Text.

bitmap images are uncompressed. K-means exhibits error tolerant capability under a relatively low error rate, but when the error rate goes higher, the acceptable rate drops significantly. With the approximate nature of the algorithm, we find that a certain number of changes in the data input do not degrade result quality, especially when changes in values are small. When a value is changed significantly, the algorithm may remove this value from computation and still produce acceptable results. Furthermore, text processing exhibits a good error tolerant capability when the error rate is higher than 10^{-8}. Note that our criteria for text processing are very strict (0.1% compared with 2%–10% for other top-k algorithms). The criteria may be varied depending on scenarios. In that case, the results can be significantly improved. Canneal and Djpeg are more error sensitive. However, this behavior can be easily improved with the help of a few simple techniques. Note that we envision that such techniques can be implemented as the OS and SoftFlash controller work together by passing the hints from high-level applications, as shown in Fig. 2(b).

We utilize progressive jpeg images as the inputs of Djpeg. Jpeg images consist of a sequence of segments. Each segment contains Huffman tables, quantization tables, or scans. Each segment begins with a marker. The segment of scan contains coded data that can be decoded with Huffman tables and quantization tables. Progressive images contain multiple scans so that the contour of an image can be decoded and displayed fast, and then the image progressively reaches its best quality. Progressive images are widely used on the Internet due to this feature. This feature also provides an opportunity to improve the error tolerance capability. The last segments of code data in the progressive images are less important than the other segments, because it is used to improve the image quality.

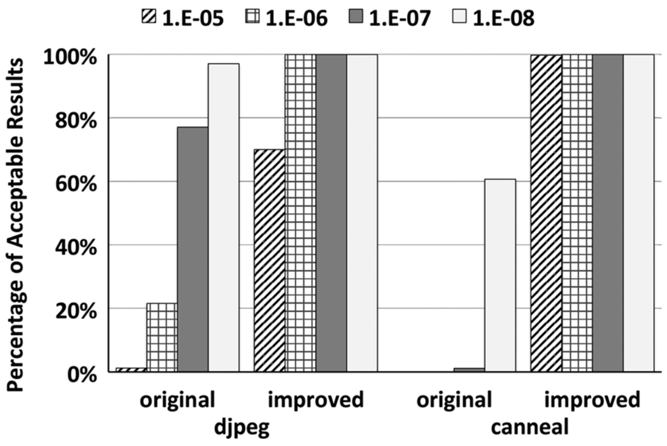

Fig. 9. Improvement in acceptable results under various error rates.

To verify this effect, we inject the faults only to the last scan of a progressive image, and compare it with results from injecting faults in all segments of its un-progressive version. As shown in the two left-most clusters of the bar graphs in Fig. 9, in the un-progressive image when the error rate reaches 10^{-5}, there are no acceptable results. Comparatively, the progressive Djpeg has a higher acceptable rate at each error rate, and the average acceptable percentage is improved by 32.5%. These results suggest that data in the same file may have different correctness requirements. By differentiating the requirements, and applying different level of protections, the overall error tolerance capability can be improved significantly.

In addition, we notice that there are a large number of crash cases in Canneal. In those crash cases, errors change the input data to invalid values or formats, which results in crashes. To fix this problem, we add a simple validation function to check the input data. If the data are out of the valid range, random and valid values will be generated to replace the erratic values so that the program will not crash due to erratic values. This function only takes several lines, so that it does not affect the overall Canneal performance. But it is very effective. As shown in Fig. 9, Canneal with this simple improvement becomes much more error tolerant, even under a high error rate. The original version cannot execute correctly with an error rate higher than 10^{-7}. However, the improved version achieves 100% acceptable results at all four error rates.

Through analyzing fault injection results, we summarize our observations as follows

- **A large amount of data can potentially tolerate much higher bit error rates than what the SSD provides**. Our fault injection results show that, for these applications, the bit error rate can be as high as 10^{-6}. Comparatively, current SSDs usually use 10^{-16} as the targeted bit error rate. With the error model constructed in Section III, it is possible to store data for these applications in SSDs without any ECCs or with weaker ECCs as long as their error rates are lower than the application requirements. We have conducted a case study, which will be presented shortly, to show the potential benefits of this approach.
- **Data have different levels of importance, and in some cases the importance can be easily identified**. For example, all data in the input files of K-means, Text, and MC are highly error-tolerant. These files as a whole can be easily operated when deploying SoftFlash. It is tricky for data with different importance in a single file. For example, the *jpeg* and *bmp* images contain some formatting and coding information that is more critical than coded data. Because the critical parts of these files are only relatively small portions, a more aggressive approach can even take advantage of these non-critical data by storing them into SoftFlash. Fortunately, these non-critical data can be identified by markers in the file, which can be utilized by SoftFlash as indicators.
- **Minor changes to some data-intensive applications can significantly improve their reliability**. For example, Canneal cannot tolerate even a single error if it has the wrong format. But an improved version with a simple modification shows a strong error tolerance capability (Fig. 9).

In summary, the applications exhibit strong error tolerance capabilities at very high error rates. We also show that some simple techniques can be applied to reduce the need for strong ECCs. Therefore, it is viable to relax the data correctness requirements in SoftFlash, with support from the operating system, and the SSD controller. As we have shown in the Fig. 9, after fixing a small bug in the program, Canneal shows great error tolerance ability. The application designers should write robust programs that minimize the potential impacts of the errors, which in turn will increase the data-level error tolerance of applications. The details about the implementation issues in operating systems and SSD controllers are discussed in Section VI.

C. Case Study

Various applications exhibit strong capabilities of the data-level error tolerance. In this section, we conduct a case study to demonstrate how to use SoftFlash to trade data-level correctness for area, performance, and energy.

TABLE VII
MC WITH DIFFERENT ECC SCHEMES

	No ECC (SoftFlash)	Hamming	RS	BCH
Error Rate	1.14E-7	2.69E-11	4.84E-23	3.63E-75
Accuracy	2.36E-6	2.88E-8	2.88E-8	2.88E-8
Storage Overhead	0	0.6%	3.3%	4.8%
Energy Overhead	1	3.75	4.05	4.5
Read Latency Overhead	1	1.41	1.47	1.45

We assume that the MC benchmark is running on a SoftFlash SSD at its P-E cycles of 3 400. This experiment can be easily carried out to the full lifetime of an SSD, which we omit here to save print space. First, the operating system uses the disk-level RBER models in Section III to estimate the current raw bit error rate of the underlying SSD. At the P-E cycle of 3 400, the estimated disk-level error rate is 1.14×10^{-7}. Second, the operating system obtains the data-level correctness requirement of MC, based on the results in Section V, which show that MC maintains the data-level correctness at the error rate of 10^{-7}. Third, the operating system passes the hint of the data-level error tolerance of MC to the underlying SoftFlash SSD. Because the MC shows a strong error tolerance capability at the error rate of 10^{-7}, and the modeled current error rate is close to this range, the data-error manager in the SoftFlash SSD controller dispatches the data inputs of MC to flash chips without ECC protections. In the traditional architecture, the input data will be stored in flash chips with ECC protections otherwise.

We conduct fault injection experiments in the MC benchmark with calculated uncorrectable bit error rates, and also calculate the overhead costs for traditional architectures with three ECC schemes. The quantitative results are presented in Table VII. The read performance and energy numbers are normalized to the SoftFlash numbers. One can see that the mean relative errors (accuracy in the table) are 2.63×10^{-6} in SoftFlash, drops to 2.88×10^{-8} with Hamming code, and does not further decrease even with stronger ECC. However, the energy, performance, and storage overhead continue to increase as the number of ECC bits increases. The energy consumption of ECC schemes can go up to 4.5 times that of SoftFlash. The storage overhead for ECC parity goes up to 4.8%, and the read performance overhead is increased by more than 40%. Though the last performance number is not as high as energy overhead, considering the importance of SSD read performance, 40% degradation is quite significant. Clearly, for this particular application, MC is error tolerant to a certain degree, and cannot benefit with additional ECC strength. As a result, the proposed SoftFlash system that aims to achieve data-level correctness can leverage this behavior to devise dynamic protection in an online fashion, and trade data correctness for I-O performance.

We conduct another case study on the improved version of *Djpeg*. We assume the same error rates and ECC algorithms are used as in *MC*. The results are shown in Table VIII. Using

TABLE VIII
IMPROVED DJPEG WITH DIFFERENT ECC SCHEMES

	No ECC (SoftFlash)	Hamming	RS	BCH
Error Rate	1.14E-7	2.69E-11	4.84E-23	3.63E-75
Results	Acceptable	Good	Good	Good
Storage Overhead	0	0.6%	3.3%	4.8%
Energy Overhead	1	3.75	4.05	4.5
Read Latency Overhead	1	1.41	1.49	1.47

ECC algorithms, results from *Djpeg* are all the same as in the original outputs. But the improved version of *Djpeg* can still achieve acceptable results, even at a high error rate, as shown in Section V-B. This case study demonstrates the opportunity of reducing the ECC strength in image storage. The result shows the potential benefits in reducing energy, and improving performance.

VI. DISCUSSION

Our experiments demonstrate the feasibility and benefits of deploying a new SSD architecture as shown in Fig. 2. The results and analysis on the data-level error tolerance can be used to derive the required error rates for applications. The RBER model at the disk level can be used to estimate the actual error rate in SSDs. Then, decisions can be made to provide suitable ECC to bridge the gap between the actual and required error rates.

To implement such an architecture, supports are required from operating systems and SSD controllers. In this section, we discuss these implementation issues.

Operating system supports. The RBER model is constructed in operating systems. A kernel module can be developed for this purpose. There are three major functions in this module: 1) identifying data for error resilient applications; 2) collecting SMART data from underlying SSDs; and 3) constructing a RBER model, and estimating RBERs using the constructed model. To support those functions, there are three major modifications that should be made in operating systems.

First, the kernel module must be able to differentiate data with different error tolerance requirements so that they can be stored with different ECC strength. For some applications, such as the *text, canneal, kmean,* and *MC* applications used in our experiments, the difference can be identified at the file level. All data in the files that are used as inputs for the same application can be considered with the same error tolerance requirement. Therefore, the target error rates obtained from our fault injection experiments can be stored with files in their meta data (e.g., inode). For some applications, a portion of data (e.g., pixels in *bmp* files used by *cjpeg*) in the same file are more resilient than others (e.g., metadata in *bmp* files). For these applications, the difference should be identified at the block level. The target error rate can still be stored in inode, but there is an extra bit assigned to each block to indicate if this target error rate is applied to this block. When the file system is performing operations on files, the error tolerance information in inode can be retrieved. To ensure the correctness of systems, the data used by operating systems and inode data should be protected with ECC.

Second, the kernel module should be able to obtain SMART data from SSDs for model construction. This is a relatively easy task because it can be implemented with existing tools such as *smartmontools* [55]. After SMART data are obtained, the model can be constructed to obtain the estimated error rate of SSDs. With the estimated error rate and the error resiliency requirements for each file or block, the kernel module can specify the required ECC strength, and can store this information into inode.

Third, when accessing data from SSDs, the kernel module must be able to send requests with the required ECC strength along with other information in inode to SSD controllers. This need can be done by modifying the SCSI command interface. The original read and write commands can be modified to add this information.

SSD controller supports. SSD controllers receive SCSI commands from the kernel module, and adjust current ECC strength accordingly. Note that software ECC can be implemented in the SSD controllers to provide flexibility of adjusting ECC strength. Software ECC is already supported by some flash manufacturers [56]. Therefore, the modification to SSD controllers is minimal, and can be done by modifying SSD controller firmware without involving hardware changes.

In summary, SoftFlash requires a kernel module to perform model construction, modifications to SCSI commands and inode, and modifications to SSD controller firmware. It does not involve modifications to the current hardware. Therefore, the design of SoftFlash can be adopted in real systems at a reasonably low cost.

VII. CONCLUSION

In this paper, we study the three key problems that enable a new SSD architecture SoftFlash: 1) find the error models for flash memory and solid-state drives; 2) conduct fault injection experiments on various types of applications; and 3) analyze potential benefits in performance, area, and energy efficiency. Our study shows that the error rate model of SSDs can be constructed by two inputs from SMART attributes that can be easily accessed from OS. Combined with our analysis of the data-level error tolerance of various applications, SoftFlash can potentially improve the read latency by more than 40%, and reduce energy overhead by up to 78%.

We have gained the insightful understandings of fault behaviors in flash based storage systems, identified a few key design challenges, and evaluated design feasibility and its potential benefits. We discuss the major implementation issues of deploying SoftFlash. The modifications to current systems are mainly within OS and SSD controller firmware, which we plan to carry out in future work. We believe that the proposed flash storage system with data-level error tolerance presents an interesting tradeoff between high-performance data access and data-level correctness for many applications.

Acknowledgment

We thank Dr. Jason Rupe, Dr. Wei-Ting Kary Chien, and anonymous reviewers for their valuable suggestions.

References

[1] "Micron P400m enterprise SATA SSD datasheet," [Online]. Available: http://www.micron.com/~/media/Documents/Products/Data%20Sheet/SSD/p400m_2_5.pdf

[2] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: Analysis of tradeoffs," in *Proc. 4th ACM Eur. Conf. Computer Systems*, 2009.

[3] F. Chen, D. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proc. 11th Int. Joint Conf. Measurement and Modeling of Computer Systems*, 2009.

[4] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009.

[5] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proc. 2007 ACM SIGMETRICS Int. Conf. Measurement and Modeling of Computer Systems*, 2007.

[6] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proc. IEEE*, vol. 91, no. 4, pp. 489–502, Apr. 2003.

[7] Spansion, "What Types of ECC Should be Used on Flash Memory?" Application Note, 2011.

[8] A. Yu, "Improving NAND reliability with low-density parity check codes (LDPC)," in *Proc. Flash Summit*, 2011.

[9] V. Gaudet, "Energy efficient circuits for LDPC decoding," Jul. 2007 [Online]. Available: http://www.cmoset.com/uploads/7.2.pdf

[10] G. Wu, X. He, N. Xie, and T. Zhang, "Diffecc: Improving SSD read performance using differentiated error correction coding schemes," in *Proc. Int. Symp. Modeling, Analysis, and Simulation of Computer Systems*, 2010.

[11] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks," in *Proc. 5th Int. Conf. Information Processing in Sensor Networks*, 2006.

[12] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in *Proc. 40th Asilomar Conf. Signals, Systems and Computers*, 2006.

[13] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *Proc. 35th Annu. Int. Symp. Computer Architecture*, 2008.

[14] A. Ferreira, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Impact of process variation on endurance algorithms for wear-prone memories," in *Proc. Design, Automation & Test in Europe Conf. Exhib. (DATE)*, 2011.

[15] A. Chimenton and P. Olivo, "Erratic erase in flash memories—Part I: Basic experimental and statistical characterization," *IEEE Trans. Electron Devices*, vol. 50, no. 4, pp. 1009–1014, Apr. 2003.

[16] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill, "Bit error rate in NAND flash memories," in *Proc. IEEE Int. Reliability Physics Symp.*, 2008.

[17] H. Sun, P. Grayson, and B. Wood, "Qualifying reliability of solid-state storage from multiple aspects," in *Proc. 7th IEEE Int. Workshop Storage Network Architecture and Parallel I/O*, 2011.

[18] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annual Technical Conf.*, 2008.

[19] Y. Jin, "A definition of soft computing," [Online]. Available: http://www.soft-computing.de/def.html

[20] M. Breuer, "Multi-media applications and imprecise computation," in *Proc. 8th Euromicro Conf. Digital System Design*, 2005.

[21] F. Kurdahi, A. Eltawil, A. Djahromi, M. Makhzan, and S. Cheng, "Error-aware design," in *Proc. 10th Euromicro Conf. Digital System Design Architectures, Methods and Tools*, 2007.

[22] J. Meng, S. Chakradhar, and A. Raghunathan, "Best-effort parallel execution framework for recognition and mining applications," in *Proc. IEEE Int. Symp. Parallel&Distributed Processing*, 2009.

[23] X. Li and D. Yeung, "Application-level correctness and its impact on fault tolerance," in *Proc. IEEE 13th Int. Symp. High Performance Computer Architecture*, 2007.

[24] X. Li and D. Yeung, "Exploiting application-level correctness for low-cost fault tolerance," *J. Instruction-Level Parallelism*, vol. 10, pp. 1–28, Sep. 2008.

[25] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," in *Proc. 16th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2011.

[26] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *Proc. 32nd ACM SIGPLAN Conf. Programming Language Design and Implementation*, 2011.

[27] J. Heidecker, "NAND flash qualification guideline," in *Proc. NEPP Electronic Technology Workshop*, 2012.

[28] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. 10th USENIX Conf. File and Storage Technologies*, 2012.

[29] Seagate Inc., "Barracuda data sheet," [Online]. Available: http://www.seagate.com/files/staticfiles/docs/pdf/datasheet/disc/barracuda-ds1737-1-1111us.pdf

[30] PNY Technologies Inc., PNY Prevail SSD [Online]. Available: http://www3.pny.com/SSDPDFs/Prevail-SSD.pdf

[31] Seagate, Establishing Industry Endurance Standards for Solid State Storage, Tech. Rep., 2010 [Online]. Available: http://www.seagate.com/files/staticfiles/docs/pdf/whitepaper/tp618-ssd-tech-paper-us.pdf

[32] JEDEC, "Solid state drive (SSD) requirements and endurance test method," [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd218a

[33] Samsung, Samsung SSD SM825 [Online]. Available: http://www.samsung.com/us/business/oem-solutions/pdfs/SM825_Product\%20Overview.pdf

[34] American National Standard of Accredited Standards Committee INCITS, ATA/ATAPI Command Set. S.M.A.R.T. [Online]. Available: http://www.t13.org/Documents/UploadedDocuments/docs2006/D1699r3f-ATA8-ACS.pdf

[35] OCZ, OCZ Vertex SSD Specification [Online]. Available: http://www.ocztechnology.com/products/flash_drives/ocz_vertex_series_sata_ii_2_5-ssd 2009

[36] N. R. Draper and H. Smith, *Applied Regression Analysis*. New York, NY, USA: Wiley, 1981.

[37] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.

[38] B. Fu and P. Ampadu, "Error control combining hamming and product codes for energy efficient nanoscale on-chip interconnects," *IET Comput. Digit. Techn.*, vol. 4, no. 3, pp. 251–261, May 2010.

[39] J. Kim, J. Cho, and W. Sung, "Error performance and decoder hardware comparison between EG-LDPC and BCH codes," in *Proc. 2010 IEEE Workshop on Signal Processing Systems (SIPS)*, 2010.

[40] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ, USA: Prentice Hall, 1983.

[41] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA, USA: MIT Press, 1963.

[42] Intel, Intel 510 SSD Specification [Online]. Available: http://download.intel.com/pressroom/kits/ssd/pdf/Intel_SSD_510_Series_Product_Specification.pdf

[43] J. Katcher, "Postmark: A new file system benchmark," [Online]. Available: http://www.netapp.com/tech_library/3022.html

[44] IOzone [Online]. Available: http://www.iozone.org

[45] "SPC trace file format specification," 2002 [Online]. Available: http://skuld.cs.umass.edu/traces/storage/SPC-Traces.pdf

[46] X. Xu and M.-L. Li, "Understanding soft error propagations using efficient vulnerability-driven fault injection," in *Proc. Int. Conf. Dependable Systems and Networks*, 2012.

[47] X. Xu, K. Teramoto, A. Morales, and H. H. Huang, "DUAL: Reliability-aware power management in data centers," in *Proc. IEEE Int. Symp. Cluster Computing and the Grid*, 2013.

[48] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, "Mediabench ii video: Expediting the next generation of video systems research," *Microprocess. Microsyst.*, vol. 33, pp. 301–318, Jun. 2009.

[49] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. Dissertation, Princeton University, Princeton, NJ, USA, Jan. 2011.
[50] Apache, Apache Hadoop Website, 2011 [Online]. Available: http://hadoop.apache.org/
[51] A. Metwally, D. Agrawal, and A. E. Abbadi, “Efficient computation of frequent and top-k elements in data streams,” in *Proc. 10th Int. Conf. Database Theory*, 2005.
[52] H. Huang, N. Zhang, W. Wang, G. Das, and A. Szalay, “Just-in-time analytics on large file systems,” in *Proc. 9th USENIX Conf. File and Storage Technologies (FAST)*, 2011.
[53] MathWorks, Matlab Website, 2011 [Online]. Available: http://www.mathworks.com/products/matlab/
[54] C. Manning, P. Raghavan, and H. Schutze*, Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[55] Smartmontools [Online]. Available: http://smartmontools.sourceforge.net/
[56] M. Mariano, Micron Technology, Inc., ECC Options for Improving NAND Device Reliability, Tech. Rep., 2012.

Xin Xu received the B.S. and M.S. degrees from East China University of Science and Technology in 2006 and 2009, respectively. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at George Washington University.

His research mainly focuses on characterizing and designing fault tolerance mechanisms in computer systems at the system and architecture levels.

H. Howie Huang received the Ph.D. in computer science from the University of Virginia in 2008.

He is an Assistant Professor in the Department of Electrical and Computer Engineering at the George Washington University. His research interests are in the areas of computer systems and architecture, including cloud computing, big data, high-performance computing and storage systems.

Prof. Huang received the NSF CAREER award in 2014, NVIDIA Academic Partnership Award in 2011, and IBM Real Time Innovation Faculty Award in 2008.