An Exploratory Study on Software-Defined Data Center Hard Disk Drives

YIN LI, Rensselaer Polytechnic Institute, USA
XUBIN CHEN, Rensselaer Polytechnic Institute, USA
NING ZHENG, Scaleflux, Inc., USA
JINGPENG HAO, Rensselaer Polytechnic Institute, USA
TONG ZHANG, Rensselaer Polytechnic Institute, USA

This paper presents a design framework aiming to reduce mass data storage cost in data centers. Its underlying principle is simple: Assume one may noticeably reduce the HDD manufacturing cost by significantly (i.e., at least several orders of magnitude) relaxing raw HDD reliability, we ensure the eventual data storage integrity via low-cost system-level redundancy. This is called system-assisted HDD bit cost reduction. In order to better utilize both capacity and random IOPS of HDDs, it is desirable to mix data with complementary requirements on capacity and random IOPS in each HDD. Nevertheless, different capacity and random IOPS requirements may demand different raw HDD reliability vs. bit cost trade-offs and hence different forms of system-assisted bit cost reduction. This paper presents a software-centric design framework to realize data-adaptive system-assisted bit cost reduction for data center HDDs. Aiming to improve its practical feasibility, its implementation is solely handled by the filesystem and demands only minor change of the error correction coding (ECC) module inside HDDs. Hence, it is completely transparent to all the other components in the software stack (e.g., applications, OS kernel, and drivers) and keeps fundamental HDD design practice (e.g., firmware, media, head, and servo) intact. We carried out analysis and experiments to evaluate its implementation feasibility and effectiveness. We integrated the design techniques into *ext4* to further quantitatively measure its impact on system speed performance.

CCS Concepts: • **Storage Management → Storage hierarchies**;

Additional Key Words and Phrases: Reliability, error-tolerance, filesystem design, local erasure coding

ACM Reference Format:
Yin Li, Xubin Chen, Ning Zheng, Jingpeng Hao, and Tong Zhang. 2019. An Exploratory Study on Software-Defined Data Center Hard Disk Drives. *ACM Trans. Storage* 1, 1, Article 1 (January 2019), 22 pages. https://doi.org/10.1145/3319405

1 INTRODUCTION

This paper focuses on the reducing bit cost of magnetic recording hard disk drives (HDDs) for data centers. Although flash memory has been rapidly penetrating into data centers, the substantial bit

Authors' addresses: Yin Li, Rensselaer Polytechnic Institute, Department of Electrical, Computer & Systems Engineering, 110 8th Street, Troy, NY, 12180, USA, liyin1985@gmail.com; Xubin Chen, Rensselaer Polytechnic Institute, Department of Electrical, Computer & Systems Engineering, 110 8th Street, Troy, NY, 12180, USA, chenx22@rpi.edu; Ning Zheng, Scaleflux, Inc. 97 East Brokaw Road Suite 260, San Jose, CA, 95112, USA, ningzhengrpi@gmail.com; Jingpeng Hao, Rensselaer Polytechnic Institute, Department of Electrical, Computer & Systems Engineering, 110 8th Street, Troy, NY, 12180, USA, haoj@rpi.edu; Tong Zhang, Rensselaer Polytechnic Institute, Department of Electrical, Computer & Systems Engineering, 110 8th Street, Troy, NY, 12180, USA, tzhang@ecse.rpi.edu.

1553-3077/2019/1-ART1 $15.00
https://doi.org/10.1145/3319405

cost advantage of HDDs and exploding data storage demands will keep HDDs as the technology of choice for mass data storage in data centers, at least in the foreseeable future [8]. Over the past decades, the steady HDD bit cost reduction has been solely enabled by the continuous progress of underlying device technologies [11, 26], e.g., thinner magnetic media, better and smaller write and read heads, more powerful read channel signal processing and error correction coding (ECC), and more accurate head positioning servo. However, as these device-level technologies are approaching their physical limits, it becomes increasingly difficult for HDD industry to maintain the historical bit cost reduction trend on its own. Recently, the concept of *data center HDDs* [8] has been proposed as an option to continue HDD bit cost reduction for data centers. The key is to re-think the design and architect of HDDs and I/O software stack by cohesively exploiting the characteristics of data centers and magnetic recording technology.

One essential aspect of data center HDDs is to exploit the bit cost vs. raw HDD storage reliability trade-off. In this work, we assume that HDD manufacturers could noticeably reduce HDD bit cost by significantly (i.e., several orders of magnitude) relaxing its reliability specs (i.e., read retry rate and sector read failure rate) [3]. By applying certain data redundancy at the system level, data centers could embrace the orders-of-magnitude per-HDD reliability degradation without sacrificing the eventual data storage integrity. If the system-level redundancy does not override the per-HDD cost saving and meanwhile does not cause intolerable system performance impact, it could be a viable option to complement with HDD technology advancements to push the bit cost reduction envelope. This is referred to as system-assisted bit cost reduction. A recent study [20] presented a design technique called *local erasure coding* to implement system-assisted bit cost reduction at the filesystem level. Its basic idea is to apply long erasure codes to the data being stored in one HDD, which tries to fix the sector read failures with existing redundancy instead of immediately triggering the slow internal read retries inside HDDs.

This paper studies the practical implementation of system-assisted bit cost reduction for data center HDDs. In order to better utilize both capacity and random IOPS of HDDs, it is desirable to mix data with complementary requirements on capacity and random IOPS in each HDD. It is not uncommon that resources (e.g., HDD storage space, CPU, etc.) assigned to dedicated workloads are underutilized during most of the time. Hence, there is a good potential to improve resource utilization (and hence reduce overall system cost) by consolidating multiple workloads on fewer servers. A prior study [10] demonstrated that it is possible to achieve absolute consolidation ratios ranging between 5.5:1 and 17:1 for some specific workloads. Even when dedicating the entire HDDs to specific workloads (e.g., HDFS in a large Hadoop cluster), there are still different read request characteristics (e.g., small request with 64kB vs. large requests with 1MB). Therefore, this work assumes that it is practically desirable to consolidate workloads with different IO patterns onto the same HDD. However, it will complicate the realization of system-assisted bit cost reduction because different types of data may demand different raw HDD reliability vs. bit cost trade-offs. This paper presents a software-centric design framework to address this challenge. To maximize its practical feasibility, this design framework has the following properties: (1) Within the software stack, it only requires modest changes of a filesystem and is completely transparent to all the other components (e.g., applications, OS kernel, and drivers). (2) Inside HDDs, it only requires minor changes to the ECC module and is completely transparent to all the other components (e.g., firmware, media, head, and servo). (3) It is completely transparent to HDD I/O protocols (i.e., it does not require any changes to the existing I/O protocols).

The proposed design framework is based upon a simple fact: Assume a segment (e.g., 512B) within one 4kB sector is all-zero, and the ECC module inside HDD can detect the location this all-zero segment. Then the intra-HDD ECC module could simply force this segment as all-zero during ECC decoding, which can reduce the decoding failure probability and hence improve the

storage reliability for this sector. This is referred to as *per-sector zero forcing* and the length of the all-zero segment is called the zero forcing number. The basic idea underlying this proposed design framework is to combine per-sector zero forcing and local erasure coding. By dynamically configuring the parameters of per-sector zero forcing and local erasure coding, we could achieve *data-adaptive* system-assisted bit cost reduction for data with different requirements on capacity and random IOPS.

To facilitate its practical implementation, we propose to group all the files on each HDD into three distinct categories: (1) immutable files that are accessed mostly by sequential read (e.g., image and video), (2) immutable files that are accessed mostly (or at least noticeably) by random read (e.g., 32kB or 64kB data blocks in BigTable and HBase), and (3) mutable files that essentially include all the other files. By fixing the parameters of per-sector zero forcing and local erasure coding for each category, we can largely simplify the process of design parameter configuration. Under this design framework, one major design challenge is how to practically realize per-sector zero forcing with minimal changes to the existing software and hardware infrastructure. To address this challenge, we develop a set of design techniques that only modestly change the filesystem and intra-HDD ECC module, while keeping everything else across the entire software and hardware stacks intact. We carried out a variety of analysis and experiments to quantitatively study the impact of the proposed design framework on system speed performance and overall system-level redundancy. Our study targets at HDDs whose storage reliability specs are relaxed by 3 4 orders of magnitude compared with today's commercial HDDs. To study the impact on system speed performance, we integrated the proposed design techniques into an *ext4* filesystem, and carried out experiments using synthetic workloads and representative applications including HBase and big data benchmark suite HiBench 3.0 [2]. Our results show that the proposed design techniques incur very small speed performance degradation (i.e., 4% and below). Our analysis results further show that the proposed design framework incurs small system-level redundancy (i.e., less than 9% and as low as 2%), which most likely could be easily off-set by the HDD manufacturing cost reduction enabled by the orders-of-magnitude relaxation of raw storage reliability.

2 BACKGROUND AND RATIONALE

2.1 HDD Sector Read Failures

HDDs are fundamentally subject to runtime positional off-set of write/read head, i.e., the position of write/read head is not well aligned with the center of the target track. Head off-set is mainly caused by environmental vibration that is particularly severe in data centers. Runtime head off-set strongly correlates with HDD sector read failures (i.e., HDD internal ECC fails to decode one sector). During normal operation, once a sector read failure occurs, HDDs switch from the normal mode into a so-called *retry mode*. During retry mode, HDDs repeatedly read the failed sector by adjusting various operational configurations/parameters (in particular the position of write/read head). Read retry operation continues until the sector has been successfully decoded or a time-out limit has been reached. In the former case (i.e., read retry success), the host receives the correct data but suffers from a (much) longer read latency, which could noticeably degrade the HDD random IOPS and read tail latency; while in the latter case (i.e., read retry failure), HDDs will report a sector read failure (i.e., a sector data loss) to the host. In this paper, we define *soft sector read failure rate* as the probability that HDDs encounter a sector read failure during the normal operation (note that today's HDDs switch to the retry mode upon a soft sector read failure), and define *hard sector failure rate* as the probability that one sector cannot be successfully recovered even after read retry. Prior study [33] shows that "high-fly" writes and off-track reads/writes are important factors causing sector read failures and we have further confirmed with our industrial liaisons that in

today's commercial HDDs, soft and hard sector failure rates are typically specified as $< 10^{-6}$ and $< 10^{-12}$, respectively. about this point.

2.2 System-Assisted Bit Cost Reduction

As conventional magnetic recording technology approaches its limit at around 1Tb/in^2, the HDD industry has been exploring several new recording technologies including heat-assisted magnetic recording (HAMR) [32, 36], shingled magnetic recording (SMR) [21, 25], and two-dimensional magnetic recording (TDMR) [17, 34, 38]. Unfortunately, in spite of significant investment over the years, none of these new recording technologies shows a viable and sustainable path to continue the historical bit cost reduction trend. This naturally opens the door for more actively involving computing systems to assist the continuation of HDD bit cost reduction. The basic concept is very simple: Any storage technology is fundamentally subject to a bit cost vs. reliability trade-off, and HDDs are of course no exception. It is reasonable to expect that HDD bit cost can be noticeably reduced if we significantly relax HDD sector failure rates (e.g., by at least 3 4 orders of magnitude) beyond today's specs. Such cost reduction can be realized in different ways, e.g., over-scaling areal storage density, and relaxing the specs of head/media manufacturing and the requirements of HDD testing/qualification. In order not to compromise the data storage integrity, the per-HDD reliability degradation must be compensated by system-level redundancy. This is referred to as system-assisted HDD bit cost reduction. Clearly, the extra cost of the system-level redundancy should not override the cost reduction of HDDs, and the associated system-level operations (e.g., erasure code decoding) do not cause an intolerable impact on the system performance.

The authors of [8] pointed out that the existing redundancy distributed within or even across data centers (e.g., in the form of replica and distributed erasure coding) may be leveraged to realize system-assisted HDD bit cost reduction. Nevertheless, when using distributed system-level redundancy to recover per-HDD sector read failures, it incurs extra network traffics. As a result, it may only allow relatively modest per-HDD reliability degradation (e.g., relaxing the soft sector read failure rate from 10^{-6} to 10^{-4} 10^{-5}) in order to avoid significant network traffic overhead. Aiming to push the limit of system-assisted HDD bit cost reduction, the authors of [20] proposed a scheme called *local erasure coding*, which is applicable to systems dominated by large-chunk sequential HDD write/read operations.

2.3 Local Erasure Coding

As discussed later in Section 3, local erasure coding [20] is an integral component in our proposed design solution. We choose to implement local erasure coding inside a filesystem at the software level other than inside RAID controller at the hardware level for the following reasons: First, RAID is being replaced by distributed erasure coding in data centers, thus there may not be a dedicated hardware RAID controller inside data center servers. Second, given the runtime workload characteristics, we may need to dynamically adjust the local erasure coding configurations (i.e., the number of sectors being protected by one local erasure coding group, and the number of redundant sectors in one local erasure coding group). Hence, it is preferred to implement local erasure coding at the software level. Finally, a recent study [20] has been demonstrated that software-level implementation of local erasure encoding/decoding could achieve more than 1GB/s on one CPU core. Hence we expect that software-level implementation may not incur prohibitive CPU usage overhead.

For the reference of the readers, this sub-section describes its basic design concept. Given an $(l\ ,\ l\ )$ erasure code (e.g., RS code), where each codeword protects l user data symbols with l redundant symbols. The filesystem applies the $(l\ ,\ l\ )$ erasure code to protect each group of l consecutive user data sectors at the cost of l redundant sectors, where all the l l sectors

in each coding group reside in the same HDD. Upon a sector read failure during normal HDD operation, the filesystem first tries to recover the failed sector through erasure code decoding, and only if the erasure code decoding fails, HDD switches to the retry mode to recover the failed sector. Let p denote the sector read failure rate, we can express the erasure code decoding failure rate as

$$P\ l\ ,l\ ,p \quad \sum_{m\ 1}^{m\ n} \binom{l\ \ l}{i} p \quad 1 \quad p^{\ n\ \ m} \quad . \tag{1}$$

A soft sector read failure occurs when an HDD fails to decode one sector during its normal operation. Such a failure will trigger the HDD switches from the normal operation mode into a retry mode to repeatedly read the failed sector by adjusting some configurations and parameters such as read head position. In today's commercial HDDs, read retry rate is relatively low (e.g., $10^{\ 6}$ and below). The probability that one sector cannot be correctly read even after the long-latency HDD read retry is called hard sector failure rate, which must be extremely low (e.g., $10^{\ 14}$ and below). Assume we can noticeably reduce the bit cost of HDDs with the soft sector read failure rate $p\ >> 10^{\ 6}$ and hard sector read failure rate $p\ >> 10^{\ 12}$. Since each HDD read retry involves few (and even many) additional disk rotations, it causes a very long read latency overhead (e.g., tens or hundreds of ms). In comparison, local erasure code decoding has a much less latency overhead, especially in the case of sequential read requests. The use of $(l\ ,\ l\ )$ erasure code can reduce the HDD read retry rate from $p\ $ to $P\ l\ ,l\ ,p\ $, where $P\ l\ ,l\ ,p\ $ can be orders of magnitude lower than $p\ $. Meanwhile, the hard sector failure rate reduces from $p\ $ to $P\ l\ ,l\ ,p\ $. However, local erasure coding on its own suffers from two problems: (1) In case of sector read failures during small read, it has to read a large number of extra sectors from HDDs for erasure code decoding. (2) In case of a small update, it has to carry out read-modify-&-encoding-write operations. Both scenarios could seriously degrade HDD random IOPS and read tail latency as aggressively relaxing the reliability specs.

2.4 Mixed Data Storage on HDDs

To minimize the storage system TCO (total cost of ownership), one should fully utilize not only the per-HDD capacity but also the per-HDD random IOPS. Hence, it is desirable to mix data with complementary requirements on capacity and random IOPS in each HDD. It is not unusual that *immutable data* dominate the capacity of HDD-based storage systems in data centers. Dependent upon their read access characteristics, immutable data can be categorized as: (1) Immutable data which are accessed mostly by sequential read (e.g., image and video): They are the most capacity-demanding but least IOPS-demanding. As pointed out in [8], the exploding volume of video data is the major driver for HDD capacity in data centers. (2) Immutable data which are accessed mostly (or at least noticeably) by random read (e.g., 32kB or 64kB data blocks in BigTable and HBase): They are modestly capacity-demanding and very IOPS-demanding. They become increasingly pervasive as big data analytics are being widely deployed in data centers. Besides immutable data, HDDs store a large variety of mutable data, which typically occupy much less storage capacity than mutable data. Some mutable data could be very IOPS-demanding.

By mixing data with complementary requirements on capacity and random IOPS, we improve the utilization of both per-HDD capacity and per-HDD random IOPS. Nevertheless, because different data may favor different raw HDD reliability vs. cost trade-offs, it can be difficult to practically realize system-assisted HDD bit cost reduction. For example, due to the latency overhead induced by handling sector read failures (especially for small read), data with higher random IOPS demands cannot tolerate HDDs with high sector read failure rates, while data with lower random IOPS demands may much more easily tolerate such HDDs. Unfortunately, existing solutions (e.g., the

local erasure coding design technique as discussed above) only gear to a single reliability vs. bit cost trade-off point and hence may not work well for a mixture of diverse data in the same HDD.

3 PROPOSED DESIGN SOLUTION

To embrace the mixture of different types of data in each HDD, this section presents a design strategy to implement system-assisted HDD bit cost reduction in a data-adaptive manner. To improve its practical feasibility and adaptability, this design strategy is developed with the following objectives:

Minimal and isolated changes in the software stack: It should constrain all the changes inside a filesystem (e.g., ext4 and XFS) and leaves all the other components across the OS and I/O hierarchy software stack completely intact.

Minimal and isolated changes inside HDDs: HDDs should maintain their current homogeneous design practice, i.e., all the sectors inside one HDD have the same capacity (e.g., 4kB) and are protected by the same ECC with the same amount of coding redundancy, and the zone/track/sector structure across the entire disk is designed solely for maximizing areal storage density. We should not demand any changes to all the major components inside HDDs, including firmware, servo, media, and head, and read channel. As discussed later, we only require minor changes to the ECC module inside HDDs.

No changes in the I/O protocol: It should not demand any changes to the existing I/O protocol (e.g., SATA and SAS). In particular, it should not demand any new I/O commands.

Table 1. Mathematical Symbols and Their Definitions.

Symbol	Definition (unit)
P	Erasure code decoding failure rate.
p	Sector read failure rate.
l	No. of consecutive user data sectors in one local erasure coding group.
l	No. of consecutive redundant sectors in one local erasure coding group.
n	Zero-forcing number ($Bytes$).
d	No. of primary sectors in one zero-forcing group.
f_1	The percentage of 1's within the end length-n segment on the HDD.
C : C : C	The capacity ratio of all the type-A/B immutable files and mutable files.
T	The latency of serving one request (ms).
f T	The probability mass function (PMF) of T.
l	The read request queue depth.
q	No. of sectors within each read request.
τ $_y$	The average latency to recover one sector during the mechanical read-retry mode.
τ	The latency for HDD to read one sector during its normal mode.
τ	the average latency to readjust the read head location.

3.1 Basic Design Concept

In current practice, all the 4kB sectors are protected by the same fixed amount of ECC redundancy inside HDDs. When reading one sector, HDD controller tries to utilize the internal ECC redundancy to reconstruct the entire 4kB user data. In order to facilitate the presentation, Table. 1 summarizes the important symbols and definitions being used in this paper. The proposed design solution is based upon the following observation: Assume HDDs internally use an (n, m) ECC code to protect

each sector, where n and m denote the amount of user data (e.g., 4kB) and coding redundancy (e.g., 400B) per sector. If a certain segment within one sector is forced to be all-zeros, Let n ($n < n$) denote the length of the all-zero segment, which is referred to as *per-sector zero forcing number*. As a result, when reading the sector, HDD controller only needs to reconstruct the remaining n n amount of true user data (i.e., 4kB-512B=3.5kB) using the internal ECC redundancy, instead of size-n data in the whole sector. With the less amount of user data to be reconstructed, the same amount of ECC redundancy can achieve stronger error correction strength. Therefore the ECC decoder can leverage this property to reduce the decoding failure probability by forcing zeros on certain locations during ECC decoding. The per-sector zero-forcing number n can be used to configure the trade-off between bit cost and sector failure rate: As n increases, the per-sector user data storage capacity (i.e., n n) degrades, while the ECC decoding failure rate reduces and hence storage reliability improves. It is possible that the zero-forcing space could be used for storing additional ECC redundancy inside HDDs, which it could achieve even stronger error correction strength hence further reduce sector failure rate. However, this demands the ECC engine inside HDDs must be able to support multiple code rates, which could noticeably increase the silicon implementation cost and power consumption. Hence we choose the simple zero-forcing strategy in this work.

For the purpose of demonstration, we constructed a rate-0.9 low-density parity-check (LDPC) code [39]. Code rate is defined as the ratio between the user data length and the codeword length. For further details about LDPC coding and basic ECC operations, interested readers are referred to [22]. We also note that the proposed per-sector zero forcing is applicable to any linear block codes such as Red-Solomon code, BCH code, and LDPC code. This work chooses LDPC code simply because modern HDDs almost pervasively use LDPC codes as ECC. We carried out decoding simulations under different values of n , including 0, 256B, 512B, and 1kB. Fig. 1 shows the simulated raw bit error rate (BER) vs. LDPC code decoding failure rate, assuming that the bit noise follows a white Gaussian distribution, to obtain each data point, we ran the LDPC decoding until at least 20 sector decoding failures have occurred. The results clearly show the impact of the zero-forcing number n on the decoding failure rate. As the same ECC redundancy could be used to protect $4kB$ n user data instead of $4kB$ in one codeword, the decoding failure rate is reduced by several orders of magnitude.

We further studied the benefit of the zero-forcing approach compared to decreasing the codeword length in the local erasure coding. Let us consider the zero-forcing case with n =512B, p 10^{-3}, and the local erasure coding case with l $7, l$ 1. According to Eq. 1, the local erasure decoding failure rate is $2.2 \quad 10^{-5}$, however if zero-forcing is used, based on the LDPC simulation result in Fig. 1, the soft sector read failure rate is far below $1 \quad 10^{-8}$. As one alternative to the proposed zero-forcing strategy, we may use the all-zero segment in user data portion to store additional ECC coding redundancy, which can achieve even higher error correction strength. However, this approach demands that the ECC engine inside HDDs must be able to support multiple code rates, which could noticeably increase the silicon implementation cost.

The above observation suggests that we can dynamically configure the per-sector bit cost vs. raw HDD reliability trade-off by adjusting the zero-forcing number n within each sector. Meanwhile, to maintain the same HDD data storage integrity (e.g., sector loss probability below 10^{-12}), we may further apply local erasure coding on top of per-sector zero forcing, as illustrated in Fig. 2. Recall that l and l denote the number of user data and redundant sectors within one local erasure coding group. By adjusting the set of design parameters (n , l , l), we could achieve different forms of system-assisted HDD bit cost reduction on the same HDD, geared to data with different demands on storage capacity and random IOPS.

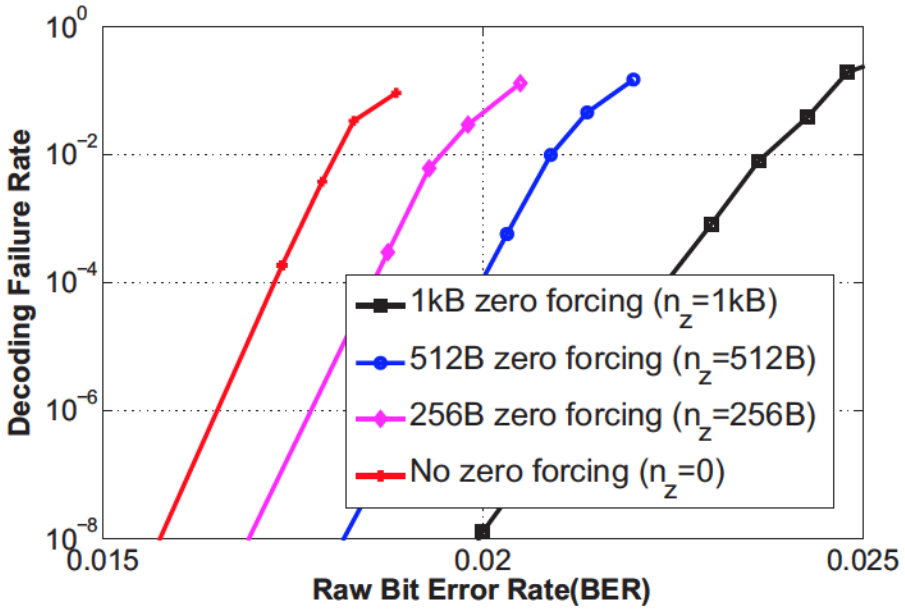

Fig. 1. Simulated BER vs. decoding failure rate under four different values of n , where the sector size n is 4kB

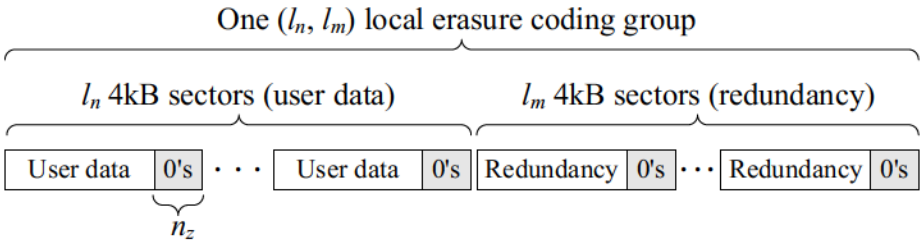

Fig. 2. Illustration of combining zero forcing and local erasure coding, where different sets of parameters (n , l , l) achieve different forms of system-assisted HDD bit cost reduction.

One may argue that HDD manufacturers could implement heterogeneous HDDs where different regions on the same disk exhibit different bit cost vs. raw reliability trade-offs. Although it is theoretically possible, its practical feasibility is very questionable since it demands significant changes on major components inside HDDs (e.g., firmware, media, and servo). Therefore, this work assumes that HDD manufacturing cost would be noticeably reduced by allowing all the sectors are subject to the same soft sector read failure rate of p >> 10^{-6} and hard sector read failure rate p >> 10^{-12}. Given per-sector zero-forcing number n , let p^{z} and p^{z} denote the corresponding soft and hard sector read failure rates. Clearly, we have p^{0} p >> 10^{-6} and p^{0} p >> 10^{-12}. Given the local erasure coding parameters (l , l) and using Eq. (1) in Section 2.3, we can calculate the HDD read retry rate as P l , l , p^{z} , which is the probability that the local erasure code fails to recover soft sector read failures. Similarly, we can calculate the eventual sector data loss probability as P l , l , p^{z} , which is the probability that the local erasure code fails to recover hard sector read failures. In order to ensure the overall storage integrity, we should have P l , l , p^{z} < 10^{-12}.

In this work we assume the sector error rate follows an independent binomial distribution. We noticed prior work [5, 31] has well studied the latent sector errors in HDDs, and tried to fit the latent sector error patterns into certain well-known statistical distributions (e.g., Geometric, Weibull, Rayleigh, Pareto, and Lognormal). In sharp contrast, this work proposes to largely relax HDD sector read failure rate by aggressively pushing the areal storage density. As a result, the HDD sector failures are mostly caused by degraded raw BER inside HDD, other than latent sector failures. Due to the randomness of raw BER inside HDD, we expect that there will be less spatial correlation among sector failures. Therefore, for our envisioned future HDDs with significantly relaxed raw BER, random and independent sector errors will be largely dominant over latent sector errors.

The design parameters (n , l , l) directly determine the bit cost vs. IOPS trade-off: As we increase the per-sector zero-forcing number n and reduce the codeword length l , we could reduce the occurrence of HDD read retry and reduce the latency incurred by local erasure coding, which contributes to reducing the impact on random IOPS. Nevertheless, larger n and smaller l come with more redundancy, leading to higher effective bit cost. Moreover, since local erasure coding must carry out read-modify-write operations to handle data write, its impact on write requests tend to be much more significant than read requests. In spite of its simple design concept, the practical implementation of this design strategy is not trivial. Throughout the remainder of this section, we present design solutions to address the following two major issues: (1) configuration of design parameters (n , l , l) on the entire HDD, and (2) modification of the filesystem and HDD to support per-sector zero forcing.

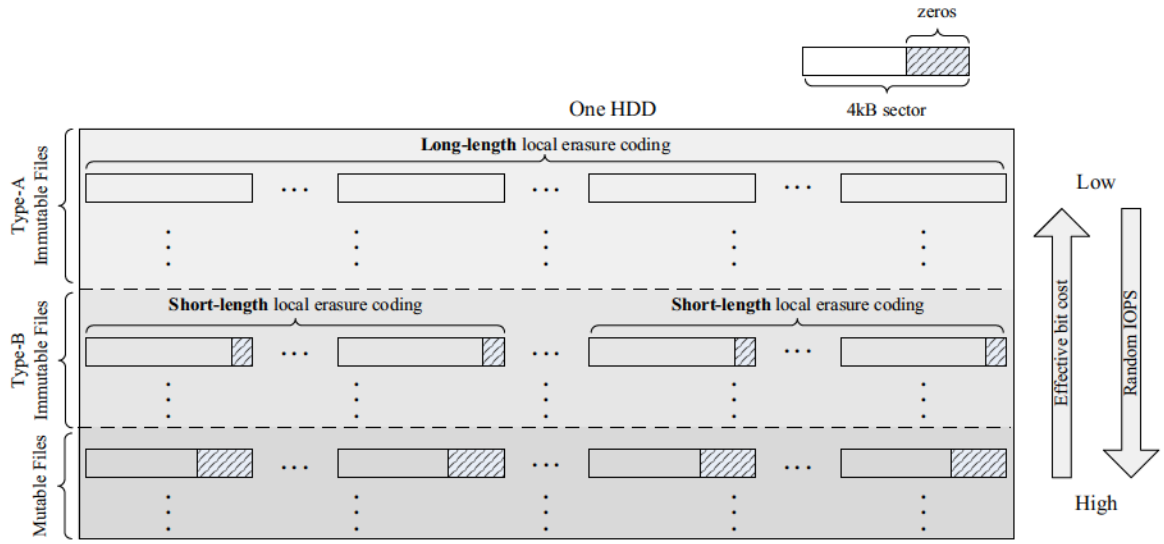

Fig. 3. Illustration of the proposed strategy to simplify the configuration of design parameters (n , l , l) for all the data on HDDs.

3.2 Configuration of Design Parameters

Following the discussions above in Section 2.4, we propose to put all the files on each HDD into three different categories, where each file category associates with one fixed set of design parameters (n , l , l). The files could be classified either statically or dynamically, e.g., dividing the whole HDD into static partitions for different categories. We could also classify the files dynamically and use little metadata (e.g., a 2-bit flag inside an inode) to indicate their categories when files are

created. As illustrated in Fig. 3, the three different categories of files and the configuration of their corresponding design parameters are described as follows:

Type-A immutable files: They contain only immutable data which are accessed mostly by sequential read (e.g., image and video). Such files tend to dominate the storage capacity usage and have minimal demands on random IOPS. Therefore, aiming to minimize the effective bit cost, we simply disable the use of per-sector zero forcing (i.e., n 0) in order to fully utilize the 4kB storage capacity in each sector. Hence, we fully rely on the local erasure coding to compensate for the raw HDD storage reliability. We construct an (l , l) local erasure code with a sufficiently large value of l (e.g., larger than 1000) so that it could restore the per-HDD storage integrity (e.g., data loss probability is below 10 ^{12}) with sufficiently small coding redundancy l l (e.g., 3% and below).

Type-B immutable files: They contain only immutable data which are accessed mostly (or at least noticeably) by random read (e.g., 32kB or 64kB per read). These files tend to occupy a noticeable amount of storage capacity and meanwhile have high demands on random IOPS. To strike a balance between bit cost and random IOPS, we combine per-sector zero forcing and local erasure coding. Since the local erasure code decoding overhead depends on the codeword length, we should use a modest codeword length. We propose to set l (much) less than the number of sectors per track in HDDs, hence local erasure code decoding only incurs at most one additional disk rotation. In today's HDDs, the number of sectors per track is 200 300 on average. Accordingly, we set the value of zero-forcing number n to achieve a sufficiently low soft sector read failure rate p ^{z} (e.g., below 10 ^{4}), and construct an (l , l) local erasure code to ensure the HDD retry rate P l , l , p ^{z} < 10 ^{6} and hard sector failure rate P l , l , p ^{z} < 10 ^{12}.

Mutable files: To simplify the practical implementation, we treat all the other files as mutable files with the highest demands on random IOPS. We do not use local erasure coding for those files, and only rely on per-sector zero forcing to restore their storage reliability, i.e., the zero-forcing number n should be large enough to ensure p ^{z} < 10 ^{6} and p ^{z} < 10 ^{12}. Logically, we could consider these files are being stored on a relatively high-cost HDD with sufficiently high per-sector storage reliability. Apparently, this approach is an over-kill for some (or even majority) files in this category. Fortunately, due to their relatively small storage footprint compared with immutable files, it is justifiable to trade insignificant storage capacity waste for the implementation simplicity.

3.3 Realization of Per-sector Zero Forcing

This sub-section presents techniques to address two main issues of practical implementing per-sector zero forcing: (1) how to embrace the mismatch between the OS page size (e.g., 4kB) and HDD per-sector true user data storage capacity (i.e., 3.5kB); and (2) how to detect and leverage the per-sector zero forcing inside HDDs.

3.3.1 Modification of a Filesystem. As stated above, we aim to constrain all the changes to a filesystem and keep the other components in the software stack intact. To simplify the implementation, we propose to modify the space allocation of a filesystem as follows: Define d $\frac{z}{z}$, where n is the HDD sector size (e.g., 4kB) and n is the per-sector zero-forcing number (e.g., 256B or 512B). For each type-B immutable file or mutable file, a filesystem allocates space in the unit of d 1 consecutive sectors, and uses d 1 sectors to store the content of d pages[1]. As illustrated in

[1]For the sake of simplicity, we assume the HDD sector size is the same as the OS memory page size (i.e., both are 4kB) in this paper.

Fig. 4, for one group of d pages, let p , r denote the content of one OS page being written to the HDD, where the size of p and r is n n and n , respectively. Among the d 1 sectors, each one of the first d sectors stores p , $\mathbf{0}$ and is called a *primary* sector, where $\mathbf{0}$ denotes a size-n all-zero vector, and the last sector stores $r_1, r_2,$, r , $\mathbf{0}$ and is called a *complementary* sector. Upon a read request, the corresponding complementary sector is always fetched from the HDD together with the primary sector(s). Since each group of d 1 sectors tend to consecutively reside on the same track and the value of d is very small (e.g., 7 or 15), fetching the additional complementary sector does not incur noticeable HDD read latency overhead. Once both primary and complementary sectors have been fetched, a filesystem needs to perform a memory copy operation (i.e., copy the size-n content r_1 from the complementary sector to a primary sector) to construct one entire size-n OS memory page.

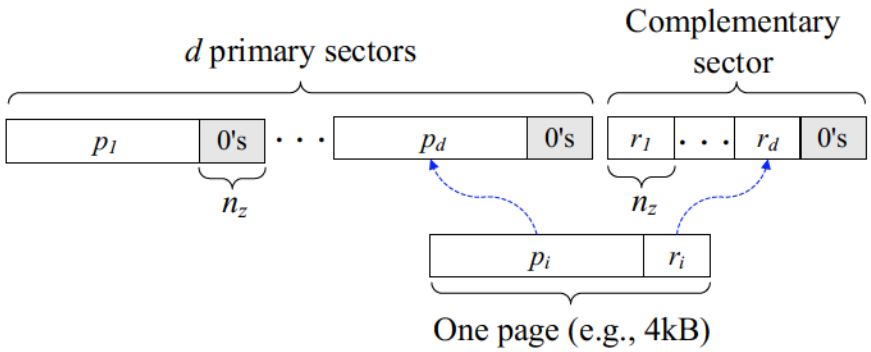

Fig. 4. Proposed the filesystem data placement using primary-complementary sector arrangement.

In the case of reading data from type-B immutable files, after a complementary sector has been loaded into one memory page and its content has been used for constructing other memory pages, a filesystem can simply discard this page. Nevertheless, in the case of accessing mutable files, since the data could be modified and written back to the HDD later on, a filesystem should leave the complementary sector page in OS page cache. To serve a write request, a filesystem must modify and write both the primary sector and the corresponding complementary sector. This, however, could be subject to a reliability risk for update-in-place filesystems (e.g., ext4) without using full data journaling. In particular, if an HDD write failure occurs on a complementary sector (e.g. due to sudden power loss), we will lose the r 's for all the associated d primary sectors. As a result, an HDD write failure when updating one page on the HDD may cause the loss of all the other d 1 pages within the same group.

We propose a simple replica-based method to address this issue. Since a write failure only corrupts the content of one sector on the HDD, we propose to keep two copies of the complementary sector for each group. In particular, the HDD uses d 2 sectors to store the content of d pages, where the other two sectors are complementary sectors storing the same content of $r_1, r_2,$, r , $\mathbf{0}$. A filesystem always updates the content of the two complementary sectors altogether with the primary sector(s). Upon an HDD write failure on one complementary sector, we can always use the other one to ensure the storage consistency. To handle the scenarios that the HDD fails to record multiple consecutive sectors (i.e., an HDD corrupts multiple consecutive sectors), we could put the two copies of the complementary sectors at the two ends of the sector groups. If the burst write failure is so severe that it corrupts all the sectors in one sector group, all the content in the group will be lost, no matter whether we use zero-forcing or not. For example, suppose a filesystem updates the content of one primary sector R and its associated complementary sectors C_1 and C_2:

(1) If a write failure occurs on the first complementary sector C_1, the content of C_2 remains intact and ensures the consistency of the other d 1 primary sectors, and only the page being written to the HDD is lost. (2) If a write failure occurs on the second complementary sector C_2, we can subsequently replicate the content of C_1 to C_2, and the data is stored in all the d primary sectors remain valid. (3) In case power fails during a write to C_2, after the system restarts from a crash, the contents of two complementary sectors C_1 and C_2 should always be re-synced before issuing any writes. If C_1 and C_2 have different content which is caused by a write failure, it will corrupt the sector ECC and hence makes the sector (C_1 or C_2) unrecoverable, HDDs can always identify which sector is corrupted. During a re-sync process we copy the correct sector content to the corrupted sector.

In terms of atomicity, after further consulting with our industry liaison at one major HDD manufacturer [3], we realized that, to guarantee per-sector write atomicity in the case of sudden power failure, HDDs rely on residual energy being stored on internal capacitors to finish the write. Moreover, latest HDDs (including shingled drives) use a small amount of embedded flash memory (and new non-volatile memory such as STT-RAM in the future) to further enhance the write durability. For HDDs that have embedded non-volatile memory, our proposed design strategy will not degrade the write atomicity. However, if HDDs do not have embedded non-volatile memory, then we have to rely on intra-HDD capacitors to ensure the write atomicity.

3.3.2 Calculation of System-level Redundancy. Before presenting a solution to address the issue of how to detect and leverage the per-sector zero forcing inside HDDs, we discuss the calculation of the overall system-level redundancy on the entire HDD, assuming the use of the above presented strategy for realizing per-sector zero forcing. Let C : C : C represent the ratio among the user data capacity of all the type-A/B immutable files and mutable files. Let n and n denote the per-sector zero-forcing number for type-B immutable files and mutable files. Recall n denote the HDD sector size (e.g., 4kB), and define d $\frac{B}{\frac{z}{B}}$ and d $\frac{M}{\frac{z}{M}}$. Let (l , l) and (l , l) denote the local erasure coding parameters for type-A and type-B immutable files. We can calculate the overall system-level redundancy as

$$C \quad \frac{l}{l} \quad C \quad \frac{1}{d} \quad \frac{l}{l} \quad C \quad \frac{2}{d}. \tag{2}$$

Section 4.1 will present redundancy analysis results under a variety of conditions, which shows that the proposed design framework tends to incur insignificant extra system-level redundancy.

3.4 Intra-HDD Support of Per-sector Zero Forcing

The proposed design strategy demands that HDDs can leverage per-sector zero forcing to reduce the intra-HDD ECC decoding failure probability. Aiming to minimize the changes inside HDDs and keep existing I/O protocol intact, this sub-section presents a design solution to enable intra-HDD support of per-sector zero forcing by only modestly changing intra-HDD ECC coding module. It does not require any extra metadata to record whether zero forcing occurs within one sector. The key idea is illustrated in Fig. 5 and described as follows:

Given one sector being written to the HDD, let d denote the length of the all-zero segment at the end of the sector, where d 0, n , n . Before the ECC module inside HDDs encodes the data, it applies a *randomization* process to randomize the data except the length-d all-zero segment. Data randomization can be done by simply XORing the data with a pre-generated random pattern. Before the ECC module decodes a sector, the ECC module first gathers the percentage of 1's within the length-n segment at the end of the sector and its proceeding length-(n n segment,

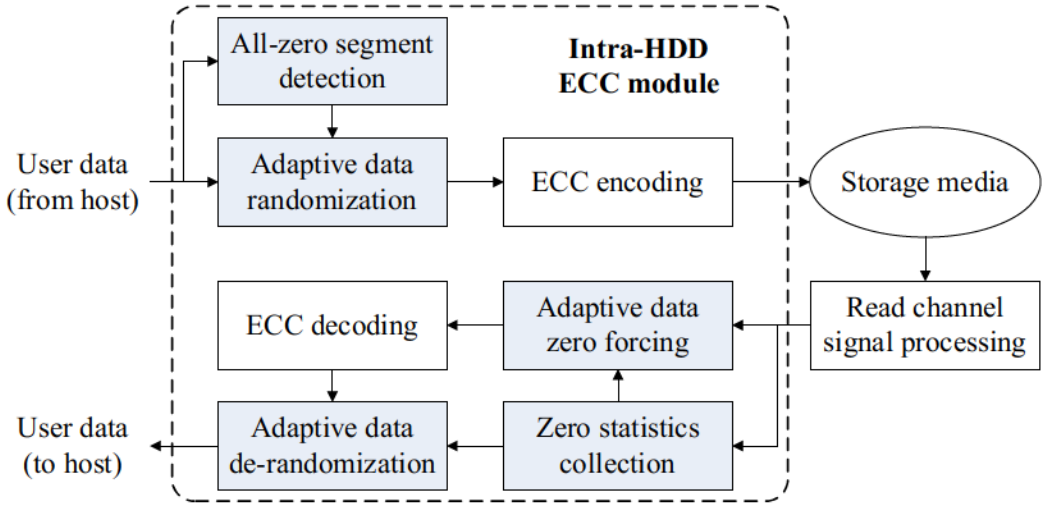

Fig. 5. Modification of the intra-HDD ECC module to support the realization of per-sector zero forcing, where the shaded blocks represent new functions being added into the ECC module.

denoted as f_1 and f_1 , respectively. Since the output of the HDD read channel (i.e., the signal equalization and trellis signal detection [19]) has reasonably low bit error rate (e.g., 3 10^{-2} and below), the ECC module could use the statistics information f_1 and f_1 to guess the type of this sector:

f_1 0.5, the ECC module predicts that this sector is a normal sector (i.e., the sector belongs to a type-A immutable file) and hence carries out ECC decoding without any zero forcing;
f_1 << 0.5 and f_1 0.5, the ECC module predicts that this sector has a zero-forcing number of n (i.e., the sector belongs to a type-B immutable file). Accordingly, it zero-forces the length-n segment at the end of the sector during ECC decoding;
If f_1 << 0.5 and f_1 << 0.5, the ECC module predicts this sector has a zero-forcing number of n (i.e., the sector belongs to a mutable file). Accordingly, it zero-forces the length-n segment at the end of the sector during ECC decoding.

Because of the use of randomization in the ECC encoding process, the prediction should have very low mis-prediction probability. For example, let us assume the output of the HDD read channel has a bit error rate of 3 10^{-2} ,which follows an independent binomial distribution. Given a type-B immutable file sector with a zero-forcing number of 256B, the probability that f_1 0.1 is as low as 1.3 10^{-47}. If extra guarantee is required against mis-prediction, in case of the ECC decoding failure, the ECC module could try the other two possible scenarios (e.g., if the ECC decoder predicts a sector belongs to a type-B immutable file but the decoding fails, it could repeat the decoding by assuming the sector belongs to a type-A immutable file or a mutable file).

4 EVALUATION RESULTS

We carried out a variety of experiments and analysis in order to evaluate the effectiveness of the proposed design framework and involved trade-offs. In our study, we assume one could noticeably reduce the HDD manufacturing cost by relaxing raw soft sector failure rates to 1 10^{-3} 1 10^{-2}, which represents at least 3 4 orders of magnitude degradation compared with current practice.

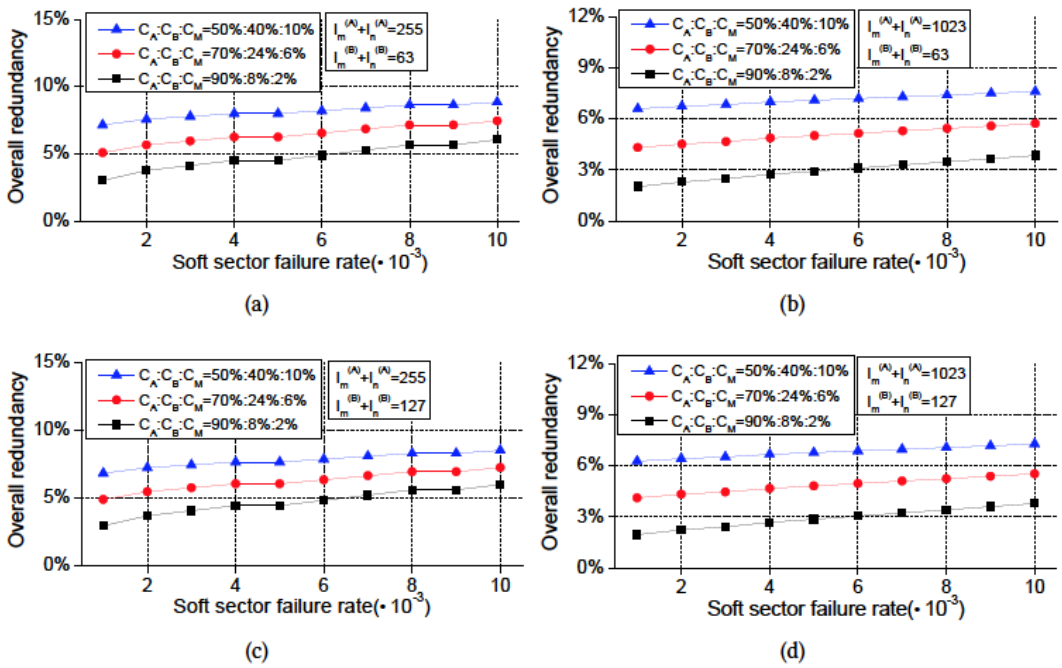

Fig. 6. Calculated system-level redundancy overhead under different raw HDD soft sector failure rates with (a) $l_m^{(A)} + l_n^{(A)} = 255, l_m^{(B)} + l_n^{(B)} = 63$; (b) $l_m^{(A)} + l_n^{(A)} = 1023, l_m^{(B)} + l_n^{(B)} = 63$; (c) $l_m^{(A)} + l_n^{(A)} = 255, l_m^{(B)} + l_n^{(B)} = 127$; and (d) $l_m^{(A)} + l_n^{(A)} = 1023, l_m^{(B)} + l_n^{(B)} = 127$.

Based upon the LDPC decoding performance simulation results shown in Fig. 1, we set the zero-forcing numbers for type-B immutable files $n_z^{(B)}$ and mutable files $n_z^{(M)}$ as 256B and 512B, respectively. We use the well-known RS codes [37] to construct local erasure codes. To experimentally evaluate the impact on system speed performance, we integrated the proposed design solutions into an *ext4* filesystem, and carried out various experiments on one PC with 3.30GHz CPU, 8GB DRAM, and 500GB 7200rpm HDD. We use HBase for type-B immutable data and HiBench suite for mutable data. We run each workload independently in order to compare the cases with and without using the proposed design techniques. To better present the experimental results without causing any confusions, we present the results of running each workload individually on the same machine with the HDD.

4.1 System-level Redundancy Overhead

We first studied the system-level redundancy overhead using the mathematical formulation presented in Section 3.3.2. Given the same coding redundancy, Reed-Solomon code can achieve better erasure correction strength than any other erasure codes. Therefore, we choose Reed-Solomon code to implement local erasure coding in order to minimize the system-level coding redundancy. The redundancy overhead depends upon several factors, including the ratio among the user data volume of all the type-A/B immutable files and mutable files $C_A : C_B : C_M$, zero-forcing number $n_z^{(B)}$ and $n_z^{(M)}$, and local erasure coding parameters $(l_n^{(A)}, l_m^{(A)})$ and $(l_n^{(B)}, l_m^{(B)})$. We fix the zero-forcing numbers $n_z^{(B)}$ and $n_z^{(M)}$ as 256B and 512B, respectively, and evaluate the impact of user data volume ratios and local erasure coding configurations under different sector failure rates on the overall redundancy

overhead. Fig. 6 shows the calculated system-level redundancy overhead under different raw HDD soft sector failure rates (i.e., 1 10 ^{3} to 1 10 ^{2}) with different parameter configurations.

First, to evaluate the impact of data volume ratio C : C : C on the overall redundancy overhead, we considered three different C : C : C ratios: 90% : 8% : 2%, 70% : 24% : 6% and 50% : 40% : 10%. The percentage of type-A immutable files C varies from 90% to 50%, and we keep the ratio C : C always equals to 4:1. The results show that the overall redundancy overhead is heavily affected by the data volume ratio C : C : C . As shown in Fig. 6, the overall system-level redundancy overhead could drop by about 4% (e.g., from 7% to 3%) once the percentage of type-A immutable files increases from 50% to 90%. This can be easily justified since type-A immutable files only use local erasure coding (i.e., n 0) with very large codeword length (hence very low coding redundancy). In comparison, type-B immutable files and mutable files demand (much) higher redundancy overhead because of the use of per-sector zero forcing (and short-length local erasure coding). Nevertheless, the overall system-level redundancy overhead appears to be reasonably low, e.g., even when the raw HDD soft sector failure rate is as high as 1 10 ^{2}, the overall redundancy overhead is still (much) less than 9%.

We further investigated the effects of local erasure coding parameters. For type-A immutable files, which are accessed mostly by sequential read, they can readily accommodate local erasure codes with large codeword length. In this study, we considered two different codeword length (i.e., l l), i.e., 255 and 1023. With the codeword length of 1023, one coding group contains about 4MB, while each access to images/videos tends to be at least a few MBs. For type-B immutable files, their random read access nature demands the use of short-length local erasure codes. As discussed above, we set the codeword length less than the number of sectors per track. Hence, we considered two different codeword length (i.e., l l) for type-B immutable files: 63 and 127. The four sub-figures in Fig. 6 correspond to the four combinations of different l l and l l . The results suggest that the codeword length for type-A immutable files (i.e., l l) has a more significant impact on the overall redundancy. This can be explained as follows: Dependence of coding redundancy on codeword length is highly related to the raw error rate: The higher the raw error rate it, the stronger the coding redundancy depends on the codeword length. Compared with type-B immutable files, type-A immutable files does not use per-sector zero forcing hence are subject to higher raw HDD soft sector failure rates. As a result, the overall coding redundancy is more noticeably affected by the codeword length for type-A immutable files.

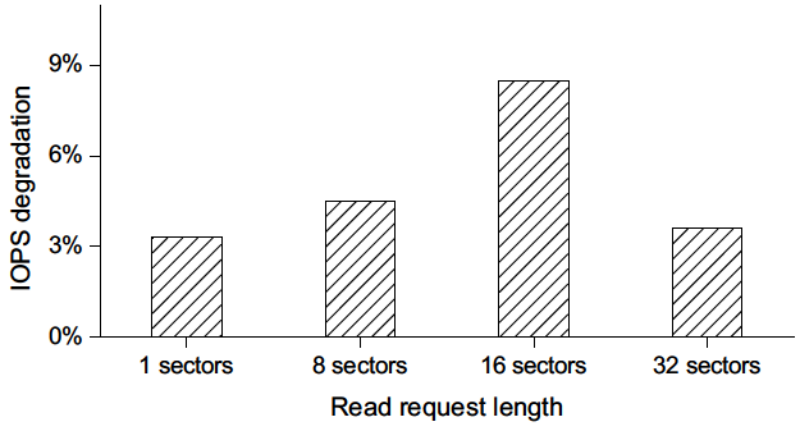

Fig. 7. HDD random read IOPS degradation when issuing small random reads to type-B immutable files.

4.2 Speed Performance Impact

For type-A immutable files, since local erasure code decoding can be much faster than HDD I/O throughput [20], they tend not to suffer from noticeable speed performance degradation. In this study, we carried out experiments to measure the impact on applications with type-B immutable files or mutable files.

In the context of applications with type-B immutable files, we first carried out experiments to directly measure the HDD IOPS using synthetic random read workloads, and then ran HBase [1] with Yahoo! Cloud Serving Benchmark (YCSB) [4] to measure the system-level performance impact. We set the raw HDD soft sector failure rate as high as 1×10^{-2} and the local erasure codeword length $l_n^{(B)} + l_m^{(B)}$ as 255. Because the per-sector zero-forcing number $n_z^{(B)}$ is 256B, every group of 15 primary sectors share one complementary sector. For the experiments, we first write total 2000 files to the HDD, where each file is 5MB and its starting LBA is randomly chosen from the entire LBA range of the HDD. Then we randomly read 1, 8, 16, or 32 consecutive sectors from each file, based upon which we measure the average IOPS. To evaluate the IOPS degradation, we repeat the same experiments without using per-sector zero forcing and local erasure coding, which is used as a baseline to quantify the IOPS degradation induced by the proposed design solution. Fig. 7 shows the random IOPS degradation normalized against the baseline. As we increase the request size, the IOPS degradation first increases (i.e., from 4% at the request size of 1 sector to 9% at the request size of 16 sectors), then reduces (i.e., the degradation drops back to 4% at the request size of 32 sectors). We suspect that this phenomenon may be caused by the cache and scheduling mechanisms inside HDDs.

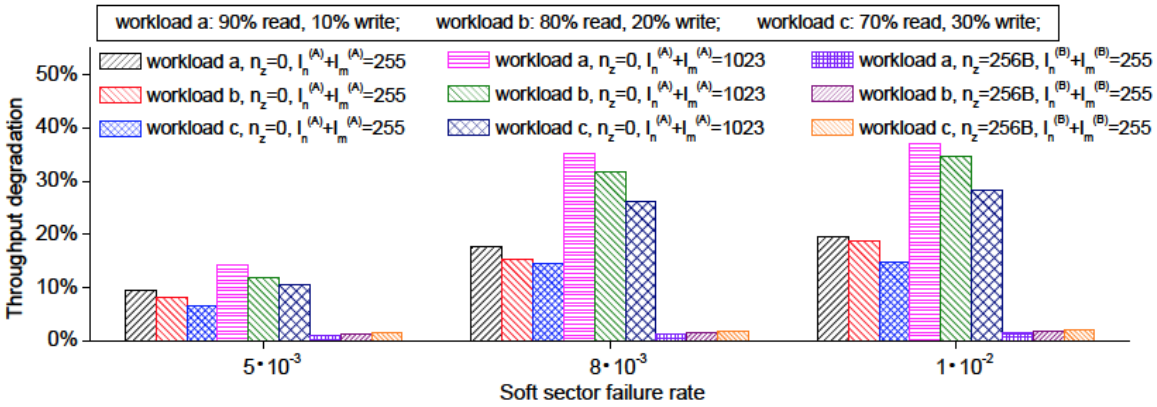

Fig. 8. Measured HBase throughput degradation under YCSB workloads with three different settings

.

Using HBase as a test vehicle, we further investigated the impact on average system speed performance for applications with type-B immutable files. We use YCSB with the following three different settings: (1) workload a: 90% read, 10% write; (2) workload b: 80% read, 20% write; (3) workload c: 70% read, 30% write. For the purpose of comparison, we also carried out experiments when using only local erasure coding without per-sector zero forcing (i.e., treat HBase files as type-A immutable files instead of type-B immutable files). We considered three different HDD soft sector failure rates: 5×10^{-3}, 8×10^{-3} and 1×10^{-2}. Table 2 shows the coding parameters under different codeword length and sector failure rates.

Fig. 8 shows the HBase throughput degradation normalized against the baseline. The results show that the combination of per-sector zero forcing and local erasure coding can significantly

Table 2. Parameters of RS-based local erasure codes under different codeword length and sector failure rates.

p	l l 255		l l 1023		l l 255	
	l	l	l	l	l	l
5 10^{-3}	9	246	19	1004	1	254
8 10^{-3}	12	243	25	998	2	253
1 10^{-2}	13	242	28	995	2	253

reduce the system speed performance degradation compared with the case of using local erasure coding only. The proposed design solution can keep the HBase performance degradation well below 5%.

To measure the speed performance impact on the applications with mutable files, we carried out experiments using the following applications from the benchmark suite HiBench 3.0 [2]: (1) Machine learning benchmarks Bayesian Classification (bayes) and K-means clustering (kmeans); (2) Job based micro benchmark WordCount (wordcount), which counts input text data generated by RandomTextWriter; (3) SQL benchmark hivebench that performs scan, join and aggregate operations, based upon the workload characteristics presented in [28]; (4) Web search benchmarks PageRank (pagerank) and Nutchindexing (nutch). As described above, we set the per-sector zero-forcing number as 512B for mutable files. Hence, each group of 7 primary sectors shares two complementary sectors. Fig. 9 shows the measured throughput degradation normalized against baseline without using per-sector zero forcing. The results show that the speed performance degradation is very insignificant with the average throughput degradation less than 1%.

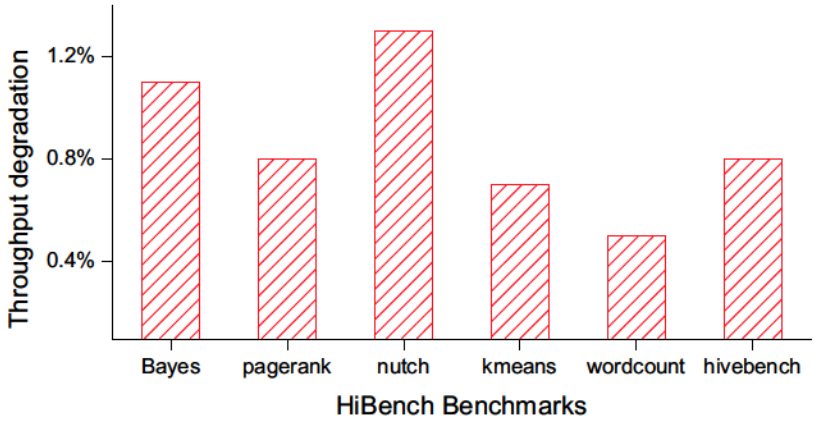

Fig. 9. Measured throughput degradation of HiBench benchmarks, where the files are treated as mutable files and protected by using per-sector zero forcing only.

4.3 Tail Latency Estimation

Besides the average throughput, read tail latency is also an important performance metric for data centers. In this work, we derived mathematical formulations to estimate the read tail latency in case of random small read requests with different queue depth. Let T denote the latency of serving one request, and f T denote its probability mass function (PMF), based upon which we could derive tail latency. Let p denote the soft sector failure probability. Let l denote the read request queue depth, and q denote the number of sectors within each read request. Let τ $_y$ denote the average

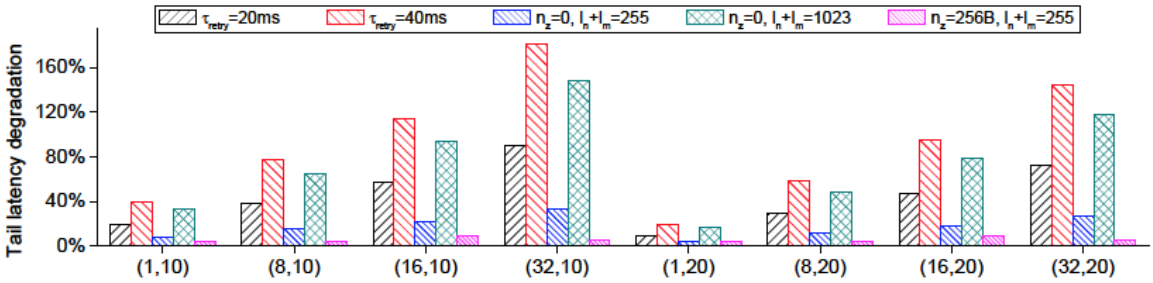

Fig. 10. Estimated 99-percent read tail latency degradation under different (q, l) parameters, where q denotes the number of sectors being read by each request, and l denote the read request queue depth.

latency to recover one sector during the mechanical read retry mode, τ_u denote the latency for the HDD to read one sector during its normal mode, and τ_{seek} denote the average latency to readjust the read head location. We can estimate the values of T and its PMF $f(T)$ as follows:

(1) Case I: The simplest case is all the sectors being read do not encounter soft sector failures:

$$\begin{cases} T = l \cdot (q \cdot \tau_u + \tau_{seek}) \\ f(T) = (1-p)^{q \cdot l} \end{cases} \tag{3}$$

(2) Case II: There are $j > 0$ read requests that encounter $e'_1,\ e'_2, \cdots, e'_j \le l_m$ soft sector failures and trigger local erasure code decoding. Let $e_1,\ e_2, \cdots, e_j \le l_m$ denote the total number of soft sector failures during the local erasure code decoding process, where $e'_i \le e_i$. Let p_i denote the probability that a (e'_i, e_i) failure pattern occur within one group, where $p_i = \binom{l_n+l_m-q}{e_i-e'_i} \cdot p^{e_i-e'_i} \cdot (1-p)^{(l_n+l_m-q-e_i+e'_i)} \cdot \binom{q}{e'_i} \cdot p^{e'_i} \cdot (1-p)^{(q-e'_i)}$. The other $(l-j)$ requests are correctly read from disk. we have

$$\begin{cases} T = j \cdot \left((l_n + l_m) \cdot \tau_u + \tau_{seek}\right) + (l-j) \cdot (q \cdot \tau_u + \tau_{seek}) \\ f(T) = \left(\dfrac{l!}{(l-j)!} \cdot \prod_{i=1}^{j} p_i\right) \cdot (1-p)^{q(l-j)} \end{cases} \tag{4}$$

Accordingly, we estimated the 99-percent read tail latency under different request size and request queue depth for the following three scenarios: (1) HDDs solely rely on internal read retry to handle soft sector read failures without using the proposed design framework, where we assume the average retry latency is 20ms or 40ms. (2) Only local erasure coding is applied to handle soft sector read failures, where the codeword length is 255 or 1023. (3) We combine the proposed per-sector zero forcing and local erasure coding, where the zero-forcing number is 256B and codeword length is 255. The raw HDD soft sector failure rate is set to 5×10^{-3}. Assuming the use of 7200 rpm HDDs, we set τ_u as 33us and τ_{seek} as 10ms. Fig. 10 shows the degradation of estimated 99-percentile read tail latency compared with the ideal baseline (i.e., HDDs without any sector read failures). The results show that the combination of per-sector zero forcing and local erasure coding is subject to very small tail latency degradation (i.e., less than 10%). In comparison, all the other schemes suffer from much larger tail latency degradation (e.g., even over 100%).

4.4 Effects of File Mis-categorization

Since the proposed design framework handles different categories of files in different ways, its effectiveness may degrade in the case of file mis-categorization at the filesystem level. Since type-A immutable files (e.g., image and video) should be easily distinguished from others, mis-categorization

more likely occurs between type-B immutable files and mutable files. If a type-B immutable file is mis-categorized as a mutable file, it will induce more redundancy and hence degrade the effective storage capacity. Meanwhile, the speed performance will improve since mutable files do not use local erasure coding.

On the other hand, if a mutable file is mis-categorized as a type-B immutable file, non-trivial storage reliability issue will arise. As discussed above, each group of primary sectors in type-B immutable files only associate with one complementary sector. If the file contains mutable data, an in-place data update operation will introduce the risk of data loss in case of the complementary sector write failure. Prior work [14, 18, 24] studied how to classify file types based on file properties and mathematical classification models, although they aimed at solving different problems. The cost of mis-prediction must be paid either by additional hardware resources (i.e. storage space) and/or performance degradation. In this work, to mitigate the issue of file misclassification, we can use the simple concept of undo logging: If a file categorized as type-B immutable is being updated, we first copy the previous version of the sector(s) being updated and associated complementary sector to a dedicated undo log, and then perform the in-place update. One may argue that the best solution is to rewrite the entire file according to the correct classification. This nevertheless can only occur in the background, especially for large files, since the entire file rewrite may take a relatively long time. The proposed use of undo-log can actually complement with the entire file rewrite, i.e., we can first use undo-log to absorb update in the foreground, and then rewrite the entire file according to the correct classification in the background. Fig. 11 shows the measured IOPS degradation caused by the use of undo logging, where each write request may contain 1, 8, 16, or 32 sectors. The results show that the IOPS degradation remains around 30% and is weakly dependent on the write size. This is because the latency of additional head seeks in undo logging tends to dominate the overall latency overhead.

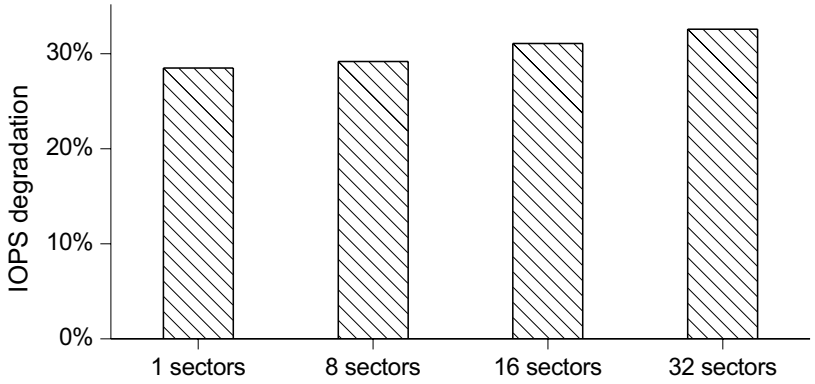

Fig. 11. Measured random data update IOPS degradation due to the use of undo logging in the case of mis-categorizing mutable files as type-B immutable file.

In addition to the raw HDD IOPS, we also investigated the impact on system-level speed performance of such file mis-categorization. In this study, we assume all the files being created by HiBench benchmarks are categorized as type-B immutable files. Whenever data updates occur, a filesystem invokes undo logging to ensure the data storage integrity. Fig. 12 shows the normalized throughput degradation for six different workloads within HiBench benchmark suite. The results show that the degradation is no more than 17%. Compared with the results shown in Fig. 9, mis-categorizing mutable files as type-B immutable files tends to worsen the system-level throughput degradation from about 1% up to 17%.

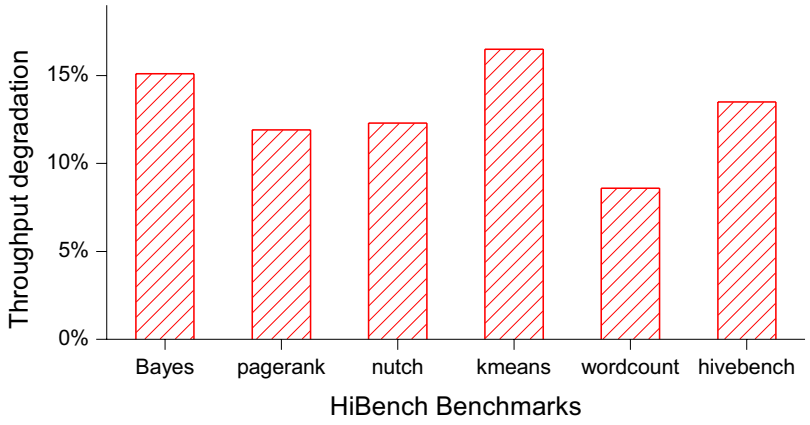

Fig. 12. Measured throughput degradation of HiBench benchmarks, where mutable files are mis-categorized as type-B immutable files.

5 RELATED WORK

This work is directly motivated by the collection views on data center HDDs presented in [8], and is based upon the concept of local erasure coding presented in [20]. As shown in this paper, local erasure coding on its own is only suitable for files dominated by sequential data access, hence cannot accommodate the mixed data types per HDD in data centers. The rationale of applying system-level redundancy to tolerate HDD operational failures is certainly not new, and extensive prior work has led to the wide real-life adoption of RAID [7, 9, 27] and distributed erasure coding [12, 15, 16, 30]. Both RAID and distributed erasure coding aim to accommodate catastrophic HDD failures at relatively very high redundancy (e.g., 20% to 50%). This work focuses on accommodating unconventionally high per-HDD sector read failures, which is essentially complementary to RAID and distributed erasure coding.

This work constrains all the software-level modification in filesystems. Prior work also studied how to enhance filesystems to better handle various HDD reliability issues. Prabhakaran et al. [29] presented an Internal RObustNess (IRON) filesystem design framework that includes a variety of HDD failure detection and recovery techniques. In particular, its transaction checksum technique inspired the implementation of journal checksum in the widely deployed *ext4* filesystem.

Prior work developed schemes to make HDDs more reliable through cross-sector error coding [23] and the use of drive motor to handle power failures [13]. Prior work also developed different methods to improve the reliability of solid-state drives (SSDs) by changing ECC engine inside the controllers, e.g., [35] proposed error-prediction low-density parity-check (EP-LDPC) and error-recovery schemes to design highly reliable SSDs, and [6] studied how to use the rate-adaptive LDPC codes to maximize the capacity of SSDs.

6 CONCLUSIONS

This paper presents a simple yet effective design framework to realize data-adaptive system-assisted HDD bit cost reduction. It enables the use of low-cost, low-reliability HDDs to host a variety of data with distinctively different demands on random IOPS and storage capacity. The key is to adaptively combine per-sector zero forcing and local erasure coding based upon different data characteristics. We have developed techniques to implement this design framework, in particular the practical implementation of per-sector zero forcing. This developed design framework only requires modest changes of a filesystem and a minor change of the intra-HDD ECC module, while

keeping everything else across the entire software/hardware stacks untouched. We carried out analysis and experiments to evaluate its effectiveness when assuming per-HDD reliability is relaxed by 3 4 orders of magnitude. Our analysis show that this design framework induces (much) less than 9% of system-level redundancy overhead. We integrated the design techniques into *ext4* to study its impact on various speed performance metrics, including HDD random IOPS, application throughput, and read tail latency. Using HBase and HiBench suite as test vehicles, we show that the proposed design solutions induce very small (e.g., even below 1%) of throughput degradation. We have further shown that the impact on the HDD random IOPS and tail latency are also small (i.e., less than 10%).

7 ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation (NSF) under grant CNS-1814890.

REFERENCES

[1] [n. d.]. *HBase.* https://hbase.apache.org/.
[2] [n. d.]. *HiBench 3.0.* https://github.com/intel-hadoop/Hi Bench/releases.
[3] [n. d.]. *Private communication with engineers of a major HDD manufacturer.*
[4] [n. d.]. *YCSB.* https://github.com/brianfrankcooper/YCSB.
[5] Lakshmi N Bairavasundaram, Garth R Goodson, Shankar Pasupathy, and Jiri Schindler. 2007. An analysis of latent sector errors in disk drives. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 35. ACM, 289–300.
[6] Stephen Bates. 2013. Using rate-adaptive LDPC codes to maximize the capacity of SSDs. In *Proc. Flash Memory Summit.* 1–12.
[7] M. Blaum, J. Brady, J. Bruck, and J. Menon. 1995. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans. Comput.* 44, 2 (Feb 1995), 192–202.
[8] E. Brewer, L. Ying, L. Greenfield, Robert Cypher, and T. T'so. 2016. *Disks for Data Centers.* Technical Report. Google.
[9] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. 1994. RAID: High-performance, Reliable Secondary Storage. *Comput. Surveys* 26, 2 (June 1994), 145–185.
[10] Carlo Curino, Evan PC Jones, Samuel Madden, and Hari Balakrishnan. 2011. Workload-aware database monitoring and consolidation. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data.* ACM, 313–324.
[11] E. Daniel, C. Mee, and M. Clark. 1999. *Magnetic recording: the first 100 years.* John Wiley & Sons.
[12] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. 2010. Network coding for distributed storage systems. *IEEE Transactions on Information Theory* 56, 9 (2010), 4539–4551.
[13] Timothy A Ferris and Robert P Ryan. 2015. Disk drive charging capacitor using motor supply voltage during power failure. (July 28 2015). US Patent 9,093,105.
[14] Gregory R Ganger, John D Strunk, and Andrew J Klosterman. 2003. *Self-* storage: Brick-based storage with automated administration.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE.
[15] K. Greenan, X. Li, and J. Wylie. 2010. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST).* 1–14.
[16] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. 2012. Erasure coding in windows azure storage. In *Proc. of USENIX Annual Technical Conference (ATC).* 15–26.
[17] A. R. Krishnan, R. Radhakrishnan, B. Vasic, A. Kavcic, W. Ryan, and F. Erden. 2009. 2-D Magnetic Recording: Read Channel Modeling and Detection. *IEEE Transactions on Magnetics* 45, 10 (Oct 2009), 3830–3836.
[18] Tom M Kroeger and Darrell DE Long. 1999. The case for efficient file access pattern modeling. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on.* IEEE, 14–19.
[19] E. Kurtas and B. Vasic. 2005. *Coding and signal processing for magnetic recording systems.* CRC Press.
[20] Y. Li, H. Wang, X. Zhang, N. Zheng, S. Dahandeh, and T. Zhang. 2017. Facilitating Magnetic Recording Technology Scaling for Data Center Hard Disk Drives through Filesystem-Level Transparent Local Erasure Coding. In *USENIX Conference on File and Storage Technologies (FAST).* 135–148.
[21] F. Lim, B. Wilson, and R. Wood. 2010. Analysis of shingle-write readback using magnetic-force microscopy. *IEEE Transactions on Magnetics* 46, 6 (Jun. 2010), 1548–1551.
[22] S. Lin and D. J. Costello. 2004. *Error Control Coding: Fundamentals and Applications (2nd Ed.).* Prentice Hall.
[23] Jaishankar Moothedath Menon and Krishnakumar Surugucchi. 2014. Method to protect data on a disk drive from uncorrectable media errors. (Feb. 4 2014). US Patent 8,645,622.

[24] Michael Mesnier, Eno Thereska, Gregory R Ganger, Daniel Ellard, and Margo Seltzer. 2004. File classification in self-* storage systems. In *null*. IEEE, 44–51.
[25] K. Miura, E. Yamamoto, H. Aoi, and H. Muraoka. 2009. Estimation of maximum track density in shingles writing. *IEEE Transactions on Magnetics* 45, 10 (Oct. 2009), 3722–3725.
[26] A. Moser, K. Takano, D. Margulies, M. Albrecht, Y. Sonobe, Y. Ikeda, S. Sun, and E. Fullerton. 2002. Magnetic recording: advancing into the future. *Journal of Physics D: Applied Physics* 35, 19 (2002), R157.
[27] D. Patterson, G. Gibson, and R. Katz. 1988. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 109–116.
[28] A. Pavlo, E. Paulson, A. Rasin, D. Abadi, D. DeWitt, S. Madden, and M. Stonebraker. 2009. A comparison of approaches to large-scale data analysis. In *Proc. of the ACM SIGMOD International Conference on Management of Data*. 165–178.
[29] V. Prabhakaran, L. Bairavasundaram, N. Agrawal, H. Gunawi, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2005. IRON File Systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 206–220.
[30] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur. 2013. Xoring elephants: Novel erasure codes for big data. In *Proc. of the VLDB Endowment*, Vol. 6. 325–336.
[31] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. 2010. Understanding latent sector errors and how to protect against them. *ACM Transactions on storage (TOS)* 6, 3 (2010), 9.
[32] M. A. Seigler, W. A. Challener, E. Gage, N. Gokemeijer, G. Ju, B. Lu, K. Pelhos, C. Peng, R. E. Rottmayer, X. Yang, H. Zhou, and T. Rausch. 2008. Integrated head assisted magnetic recording head: design and recording demonstration. *IEEE Transactions on Magnetics* 44, 1 (Jan. 2008).
[33] Sandeep Shah and Jon G Elerath. 2005. Reliability analysis of disk drive failure mechanisms. In *Annual Reliability and Maintainability Symposium, 2005. Proceedings.* IEEE, 226–231.
[34] Y. Shiroishi, K. Fukuda, I. Tagawa, H. Iwasaki, S. Takenoiri, H. Tanaka, H. Mutoh, and N. Yoshikawa. 2009. Future Options for HDD Storage. *IEEE Transactions on Magnetics* 45, 10 (Oct 2009), 3816–3822.
[35] Shuhei Tanakamaru, Yuki Yanagihara, and Ken Takeuchi. 2013. Error-prediction LDPC and error-recovery schemes for highly reliable solid-state drives (SSDs). *IEEE Journal of Solid-State Circuits* 48, 11 (2013), 2920–2933.
[36] D. Weller, G. Parker, O. Mosendz, E. Champion, B. Stipe, X. Wang, T. Klemmer, G. Ju, and A. Aian. 2014. A HAMR media technology roadmap to an areal density of 4 Tb/in^2. *IEEE Transactions on Magnetics* 50, 1 (Jan. 2014).
[37] S. B. Wicker and V. K. Bhargava. 1994. *Reed-Solomon Codes and Their Applications.* IEEE Press.
[38] R. Wood, R. Galbraith, and J. Coker. 2015. 2-D Magnetic Recording: Progress and Evolution. *IEEE Transactions on Magnetics* 51, 4 (April 2015), 1–7.
[39] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *Proc. of the USENIX Conference on File and Storage Technologies (FAST)*. 243–256.