Mitigate HDD Fail-Slow by Pro-actively Utilizing System-level Data Redundancy with Enhanced HDD Controllability and Observability

1st Jingpeng Hao
Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, USA
haoj@rpi.edu

2nd Yin Li
Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, USA
liyin1985@gmail.com

3rd Xubin Chen
Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, USA
chenx22@rpi.edu

4th Tong Zhang
Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, USA
tzhang@ecse.rpi.edu

***Abstract*—This paper presents a design framework aiming to mitigate occasional HDD fail-slow. Due to their mechanical nature, HDDs may occasionally suffer from spikes of abnormally high internal read retry rates, leading to temporarily significant degradation of speed (especially the read latency). Intuitively, one could expect that existing system-level data redundancy (e.g., RAID or distributed erasure coding) may be opportunistically utilized to mitigate HDD fail-slow. Nevertheless, current practice tends to use system-level redundancy merely as a safety net, i.e., reconstruct data sectors via system-level redundancy only after the costly intra-HDD read retry fails. This paper shows that one could much more effectively mitigate occasional HDD fail-slow by more pro-actively utilizing existing system-level data redundancy, in complement to (or even replacement of) intra-HDD read retry. To enable this, HDDs should support a higher degree of controllability and observability in terms of their internal read retry operations. Assuming a very simple form enhanced HDD controllability and observability, this paper presents design solutions and a mathematical formulation framework to facilitate the practical implementation of such pro-active strategy for mitigating occasional HDD fail-slow. Using RAID as a test vehicle, our experimental results show that the proposed design solutions can effectively mitigate the RAID read latency degradation even when HDDs suffer from read retry rates as high as 1% or 2%.**

***Index Terms*—HDD fail-slow, read latency, RAID, read retry, system-level redundancy**

I. Introduction

This paper studies how to effectively mitigate the fail-slow problem of hard disk drive (HDD). With the mechanical nature, HDDs are fundamentally sensitive to the variation of environmental factors such as vibration, temperature, and humidity [1]. This attributes to the widely observed HDD fail-slow phenomenon (i.e., HDDs occasionally operate at a speed much slower than their normal specs). A recent study [2] on fail-slow hardware reports that HDD speed could drop by even *3 orders of magnitudes* due to vibration. Driven by new magnetic recording technologies (e.g., heat assisted magnetic recording (HAMR) [3], [4], shingled magnetic recording (SMR) [5], [6], and two-dimensional magnetic recording (TDMR) [7]–[9]), the track pitch of HDDs will continue to shrink, which will make future HDDs inevitably more subject to environmental variations and hence more severe fail-slow problems.

This work particularly focuses on the read-fail-slow problem that is mainly caused by the read retry operations inside HDDs. Upon an internal sector read failure (e.g., due to environmental variation), today's HDDs always switch into a so-called *retry mode* during which HDDs repeatedly read the failed sector by adjusting various operational configurations/parameters (e.g., read head position, timing recovery, and read-back signal amplification). The read retry operation continues until the sector read succeeds or a timeout limit (e.g., tens of seconds) has been reached. Under significant environmental variations, HDDs could experience abnormally high read retry rate (e.g., 0.1% and above), leading to very poor speed performance (especially the tail latency) over a certain period. In comparison, read retry rate is typically $10^{-5} \sim 10^{-6}$ (and even below) under normal HDD specs. Very intuitively, in the presence of system-level data redundancy (e.g., RAID and distributed erasure coding), we could utilize such existing data redundancy to *indirectly* reconstruct the data instead of solely relying on intra-HDD read retry. In order to materialize this very simple concept, HDDs should support a higher degree of *controllability* (e.g., host can dynamically turn on/off intra-HDD read retry and/or adjust the read retry timeout limit) and *observability* (e.g., host can inquire HDDs about their current read retry statistics). The industry has of course recognized the potential benefits of this simple concept. For example, under the Open Compute Project (OPC), there is a

proposal on making cloud-HDDs to support the so-called *fast-fail read* [10], which reduces the HDD read retry timeout limit on the per-request basis. Nevertheless, current practice tends to be *passive* from the following two aspects: (1) It only *re-actively* invokes the system-level indirect data reconstruction, i.e., only after HDDs report sector read failures due to internal read retry timeout, the host will fetch the entire redundancy coding group from the HDD array to indirectly reconstruct the failed sectors. (2) It only opportunistically utilizes the existing data redundancy without actively enhancing the sector-failure-tolerance capability of the redundancy coding schemes (e.g., RAID and distributed erasure coding). Recall that current redundancy coding schemes are designed solely for tolerating catastrophic HDD failures (and server unavailability in the case of distributed erasure coding). Tolerating random sector read failures is essentially just a *by-product* of current redundancy coding schemes.

II. Background and Rationale

It is well documented that HDDs can occasionally operate at a speed much slower than their normal specs, which is called fail-slow [2]. Although the exact reason behind HDD fail-slow can be multi-fold and even is not fully understood (at least in the open literature), it is generally believed that fail-slow is mainly caused by abnormally high intra-HDD read retry rate. This is particularly true when deploying HDDs in a harsh operating environment such as data centers. Each read retry repeatedly reads the failed sector through additional disk rotations until the sector has been successfully read or a timeout 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; 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 the presence of system-level data redundancy (e.g., RAID and distributed erasure coding), we could possibly better mitigate HDD fail-slow by complementing (or even replacing) intra-HDD read retry with system-level indirect data reconstruction. This requires that HDDs have certain controllability and observability in terms of their internal read retry. The industry has of course well recognized the potential of this very simple intuition. For example, HDDs that are fully compliant with the T13 ATA-8 standard [11] allow the host to configure the read retry timeout limit on the per-drive basis through the time-limited error recovery (TLER) parameter. Nevertheless, per-drive configuration of the read retry timeout limit tends to be too coarse-grained and inflexible. A more recent effort is an OPC proposal [10] on introducing a new fast-fail read command that can adjust the retry timeout limit on the per-request basis. Nevertheless, as discussed earlier in Section I and further illustrated in Fig. 1(a), current practice always serves each read request with the *normal mode* first while keeping the *system-assisted mode* as a backup just in case of intra-HDD read retry timeout.

Under relatively severe HDD fail-slow with high read retry rate (e.g., 1% and above), such a passive current practice

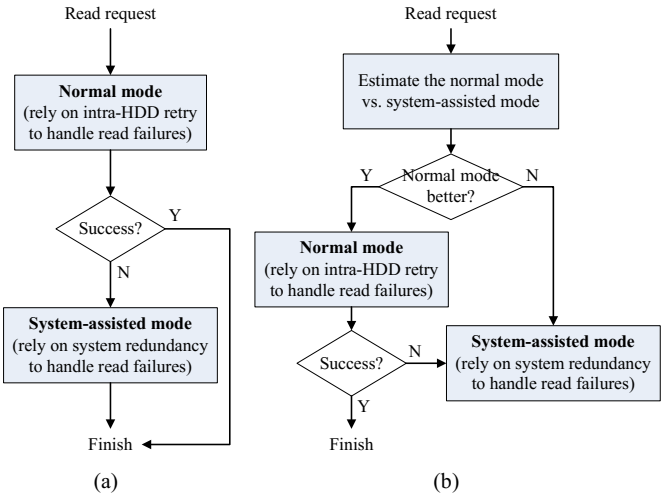

Fig. 1. Illustration of the operational flow on serving each request in the case of (a) current practice, and (b) envisioned pro-active design strategy.

may not necessarily be the best option. In particular, certain read requests may be (much) better served by directly starting with the *system-assisted mode*. Very intuitively, one may want to be more flexible on how each read request is served in order to achieve better overall system latency performance. This leads to a more pro-active design strategy as illustrated in Fig. 1(b). The objective of this work is to investigate the effective implementation of such a pro-active design strategy and quantitatively evaluate its potential advantage over current practice.

III. Proposed Design Solutions

Under the pro-active design strategy as illustrated in Fig. 1(b), for each read request, we must decide whether it is beneficial to directly serve this read request in the *system-assisted mode*. We can formulate this decision-making process as follows: For each read request r, let the set $\mathcal{R}_r$ contains all the read requests whose read latency will be affected by the decision on whether we serve the current request r in the *normal mode* or *system-assisted mode*. Obviously, all the requests in $\mathcal{R}_r$ are served after the request r has been served. For each read request $r_i \in \mathcal{R}_r$, let $\tau_i^{(N)}$ and $\tau_i^{(S)}$ denote its read latency if we serve the current request r in the *normal mode* and *system-assisted mode*, respectively. Without the loss of generality, we assume all the read requests have the same priority (i.e., their read latencies are equally important). Hence, we can use the aggregated net read latency

$$\tau = \sum_{\forall r_i \in \mathcal{R}_r} \left(\tau_i^{(N)} - \tau_i^{(S)}\right) \tag{1}$$

as the metric for the decision, i.e., if $\tau > 0$, then we should serve the current read request r directly in the *system-assisted mode*, otherwise we start with the *normal mode*. In this section, we will first present two techniques that can reduce the latency $\tau_i^{(S)}$ and hence increase the chance of using the *system-assisted mode* to improve overall read latency profile.

In particular, Section III-A presents a scheme that can improve the tolerance to random sector read failures, which can contribute to reducing $\tau_i^{(S)}$. Section III-B discusses the practical implementation of *system-assisted mode*, which may help to reduce $\tau_i^{(S)}$. Section III-C presents a mathematical framework that formulates the process of quantitatively estimating the aggregated net read latency τ. In particular, given each read request r, the mathematical formulation framework can be used to estimate the set $\mathcal{R}_r$ and each $\tau_i^{(N)} - \tau_i^{(S)}$. To simplify the discussions, this paper describes and evaluates the design solutions in the context of RAID, where each individual HDD is a fault domain.

A. Expanded RAID Encoding

Let us consider a RAID system over $m+k$ HDDs, which can tolerate the catastrophic failures of any k HDDs. In current practice, each codeword (e.g., $(m+1)$-bit parity codeword in RAID-5 with $k=1$ and $(m+2)$-byte Reed-Solomon codeword in RAID-6 with $k=2$) protects m user data symbols with k redundant symbols and can correct k symbol errors. Each codeword spans over $m+k$ sectors, where each sector is stored on one individual HDD. Hence, being covered by a large number of independent codewords, each group of $m+k$ sectors form a *sector coding group* and can tolerate up to k sector read failures within this $(m+k)$-sector coding group. Let p_f denote the sector read failure probability when intra-HDD read retry is disabled, we can calculate the probability of RAID decoding failure (i.e., one sector coding group experiences more than k sector read failures) as

$$P_{RAID\_fail} = \sum_{i=k+1}^{m+k} \binom{m+k}{i} \cdot p_f^i \cdot (1-p_f)^{m+k-i}.$$

In the case of RAID decoding failure, we have to re-fetch the data from HDDs by enabling the intra-HDD read retry. This however could significantly increase the read latency of the current read request. Therefore, it is highly desirable to reduce the RAID decoding failure probability. To achieve this objective, we propose an expanded RAID (eRAID) coding strategy, where the key is to expand the length of each codeword by an expansion factor $e \geq 1$ so that each codeword spans over e consecutive sectors from each HDD. Let s denote the number of sectors in each RAID stripe, and assume s is divisible by the expansion factor e. Each eRAID codeword (e.g., Reed-Solomon codeword) protects $e \cdot m$ user data symbols with $e \cdot k$ redundant symbols and can tolerate up to $e \cdot k$ symbol errors. Each eRAID codeword spans over $e \cdot (m+k)$ sectors, where e consecutive sectors are stored on one individual HDD. Therefore, each group of $e \cdot (m+k)$ sectors form a sector coding group and can tolerate up to $e \cdot k$ sector read failures within this $e \cdot (m+k)$-sector coding group. Assuming each stripe consists of 2 sectors (i.e., $s=2$) and meanwhile setting $m=2$ and $k=1$, Fig. 2 further illustrates and compares the conventional RAID and proposed eRAID coding. In the case of eRAID encoding, we set the expansion factor $e=2$. As shown in Fig. 2(a), with $m=2$ and $k=1$, each codeword in conventional RAID (i.e., RAID-5) is a simple 3-bit parity codeword and can correct a 1-bit error. Hence each group of 3 sectors form a coding sector group and can tolerate one sector failure. In comparison, as shown in Fig. 2(b), each codeword in the eRAID is a 6-byte Reed-Solomon (RS) codeword and can tolerate two 1-byte errors. Hence each group of 6 sectors form a coding sector group and can tolerate two sector failures within this 6-sector coding group.

Given the sector read failure rate of p_f, we can calculate the probability that one eRAID codeword experiences a decoding failure (i.e., one eRAID sector group experiences more than $e \cdot k$ read failures) as

$$P_{eRAID\_fail} = \sum_{i=e\cdot k+1}^{e\cdot(m+k)} \left[\binom{e\cdot(m+k)}{i} \cdot p_f^i \cdot (1-p_f)^{e\cdot(m+k)-i} \right].$$

We note that the above formula is valid only when the occurrence of read failure on one sector is statistically independent from any other sectors. Although this is true for sectors from different HDDs, it may not be necessarily true for adjacent sectors from the same HDD. Nevertheless, each eRAID sector coding group contains e adjacent sectors from each HDD. Under severe environmental vibration, if one sector experiences a read failure, its adjacent sector may more likely experience a read failure as well. To incorporate such adjacent-sector failure correlation, we can use a correlation factor c ($c \geq 1$) , i.e., under the sector read failure rate of p_f, if one sector experiences a read failure, a read failure may occur on its adjacent sector with a probability of $c \cdot p_f$. Among all the sector read failures, at most half of the failures occur with the probability of $c \cdot p_f$. Therefore, we can estimate the upper bound of the eRAID decoding failure probability as

$$P_{eRAID\_fail} \leq \sum_{i=e\cdot k+1}^{e\cdot(m+k)} \left[\binom{e\cdot(m+k)}{i} \cdot p_f^{\lfloor \frac{i}{2} \rfloor} \cdot (c\cdot p_f)^{\lceil \frac{i}{2} \rceil} \cdot (1-c\cdot p_f)^{e\cdot(m+k)-i} \right].$$

Fig. 3 shows the simulated decoding failure probability under different sector read failure rates with $m=5$ and $k=1$. For eRAID, we considered two different values of the expansion factor e (i.e., 2 and 3). The adjacent sector failure correlation factor c is set to be 1 (i.e., no correlation) and 2 (i.e., if one sector fails, the failure rate of its immediately adjacent sector will double). The results clearly show that the proposed eRAID can significantly reduce the decoding failure probability. For example, under the sector read failure rate of 1%, the conventional RAID decoding failure probability is 1.5×10^{-3}, and the eRAID decoding failure probability is 2.1×10^{-4} and 2.7×10^{-5} when the expansion factor e is 2 and 3, respectively, when there is no adjacent sector failure correlation (i.e., $c=1$). If we aggressively set the correlation factor as 2 , the eRAID decoding failure probability only slightly increases to about 3×10^{-4} and 4×10^{-5} when the expansion factor e is 2 and 3, respectively. Clearly, eRAID maintains the same level of tolerance to catastrophic HDD

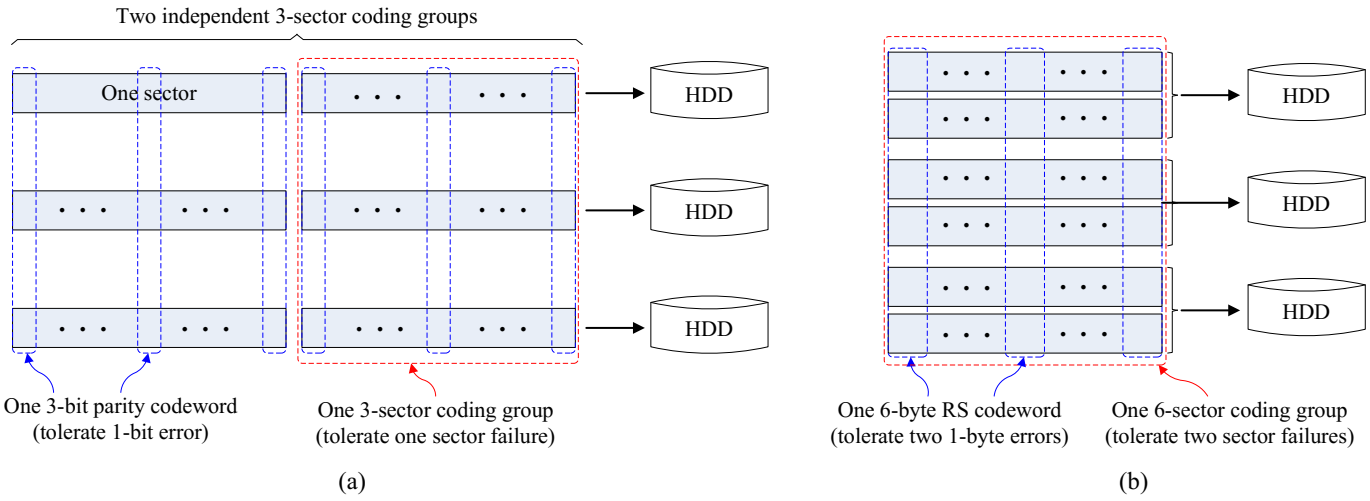

Fig. 2. Illustration of (a) conventional RAID and (b) proposed eRAID on 3 HDDs with $m = 2$ and $k = 1$.

failures as the conventional RAID, because each catastrophic HDD failure causes e sector failures within each eRAID coding sector group.

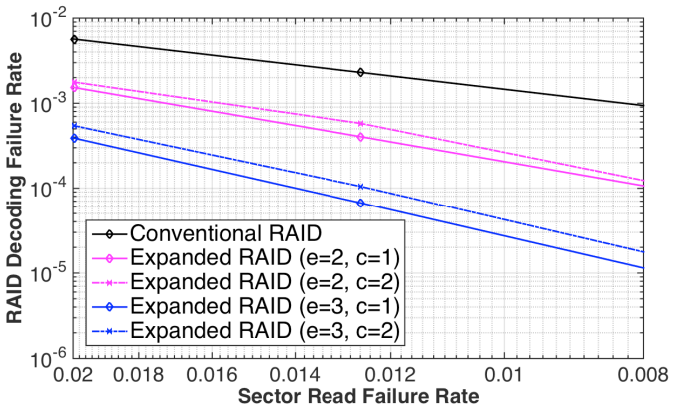

Fig. 3. RAID decoding failure rate under different sector read failure rate with $m = 5$ and $k = 1$.

It should be pointed out that, as the cost of achieving stronger tolerance to random sector read failures, the proposed eRAID apparently demands higher encoding and decoding computational complexity. Given an RS code with the codeword length n and error-correction strength of t, its encoding and decoding computational complexity is proportional to $n \cdot t$. Because of the relatively short codeword length (e.g., 10 or 20) and the small value of t (e.g., 2 or 4) in real-world systems, the computational complexity overhead of eRAID tends to be insignificant and can be easily accommodated by hardware-based RAID controller. In the case of software-based RAID controller, modern CPUs can also easily handle the eRAID encoding/decoding. For example, the RS encoding/decoding modules in Intel Intelligent Storage Acceleration Library (ISA-L) [12] can readily achieve above GB/s throughput on a single CPU core.

B. Implementation of System-Assisted Mode

When serving a read request in the *system-assisted mode*, we essentially intend to opportunistically leverage the existing coding redundancy to improve the read latency profile, especially in the presence of occasional HDD fail-slow with high read retry rates. At the first glance, one may intuitively expect to implement the *system-assisted mode* in a progressive manner: We first only fetch the requested data (with intra-HDD read retry disabled) from the RAID array. Only if one or multiple sector read failures occur, we fetch the other data in the same sector coding group (with intra-HDD read retry disabled) and reconstruct the requested data from the RAID decoding. However, this intuitive implementation option fails to exploit the full potential of the *system-assisted mode*. In particular, in the presence of existing coding redundancy, there are multiple ways of obtaining/reconstructing the requested data, among which directly fetching the requested data may not necessarily always be the best option. For example, suppose one read request aims to fetch data from l HDDs among the total $m + k$ HDDs. Besides directly fetching the data from these l HDDs, there are $\binom{m+k}{m}$ other ways of fetching data from HDDs to reconstruct the requested data. Considering the runtime variations among all the HDDs (e.g., in terms of sector read failure rate and request queue depth), the above progressive data fetching strategy may not always be the best option.

In this work, we propose to implement the *system-assisted mode* in a pro-active manner with *a posteriori* request removal. The basic idea is simple and can be described as follows. First, we note that, throughout the remainder of this paper, the term *read request* refers to the request being issued to the RAID array, and the term *read sub-request* refers to the request being dispatched to one individual HDD. To serve one read request being issued to the RAID array, we dispatch one or multiple sub-requests to one or multiple HDDs in the RAID array. Given a read request r covering data on l

HDDs, let $\{r_1, r_2, \cdots, r_l\}$ denote the corresponding l sub-requests. Meanwhile, let $\{r'_1, r'_2, \cdots, r'_{m+k}\}$ denote the sub-requests that fetch the entire coding group covering the read request r, where $r_i \subseteq r'_i$ for $1 \leq i \leq l$ (i.e., compared with the original sub-request r_i, the sub-request r'_i may read the same or more data dependent upon the boundary of the coding group). When serving the read request r in the *system-assisted mode*, we simultaneously dispatch the entire sub-request set $\{r'_1, r'_2, \cdots, r'_{m+k}\}$ to all the $m+k$ HDDs with intra-HDD read retry disabled. Due to the runtime variations among all the HDDs, the service on different sub-requests may finish at (largely) different time. Let the set $\{d_i\}$ denote the returned data of all the sub-requests that have been served by HDDs so far (note that a sector read failure will correspond to an erasure in $\{d_i\}$). Once the data set $\{d_i\}$ is enough to reconstruct the data being requested by r, we immediately remove the other outstanding sub-requests r'_j's that have not entered HDDs yet (i.e., those sub-requests that are still waiting in the host-side request queues). Clearly, if we can further enhance the controllability of HDDs so that host can inform HDDs to cancel requests that have been dispatched to HDDs, we can further improve the effectiveness of this design approach. In this work, to minimize the changes to HDDs, we do not assume the availability of such additional HDD controllability. Hence, we can only remove the sub-requests that have not entered HDDs yet.

In the case that we still cannot reconstruct the requested data even after all the sub-requests $\{r'_1, r'_2, \cdots, r'_{m+k}\}$ have been served by the HDDs (i.e., because of too many sector read failures), we have to re-dispatch one or more sub-requests to HDDs with intra-HDD read retry enabled. Based upon the above discussions, Fig. 4 further illustrates the entire operational flow of the proposed method for implementing the *system-assisted mode*.

C. Aggregated Net Read Latency Estimation

As discussed above, we can use the aggregated net read latency τ in Eq. (1) as a metric to decide whether it is beneficial to directly serve one read request in the *system-assisted mode*. This subsection presents a mathematical formulation framework for quantitatively estimating this metric. Given a read request being issued to the RAID array, we dispatch one or multiple sub-requests to HDDs, where each sub-request fetches data from one HDD. To simplify the mathematical formulation, we treat all the sub-requests independently from each other and count the read latency of each sub-request individually. Hence, for each read request, we re-define $\mathcal{R}_r$ as the set that contains all the sub-requests whose read latency will be affected by the decision whether we serve the current read request r in the *normal mode* or *system-assisted mode*. For each sub-request $r_i \in \mathcal{R}_r$, let $\tau_i^{(N)}$ and $\tau_i^{(S)}$ denote its read latency if we serve the current request r in the *normal mode* and *system-assisted mode*, respectively. As shown in Eq. (1), in order to estimate the aggregated net read latency τ, we must determine the set $\mathcal{R}_r$ and estimate the read latency difference $\left(\tau_i^{(N)} - \tau_i^{(S)}\right)$ for each sub-request within the set $\mathcal{R}_r$.

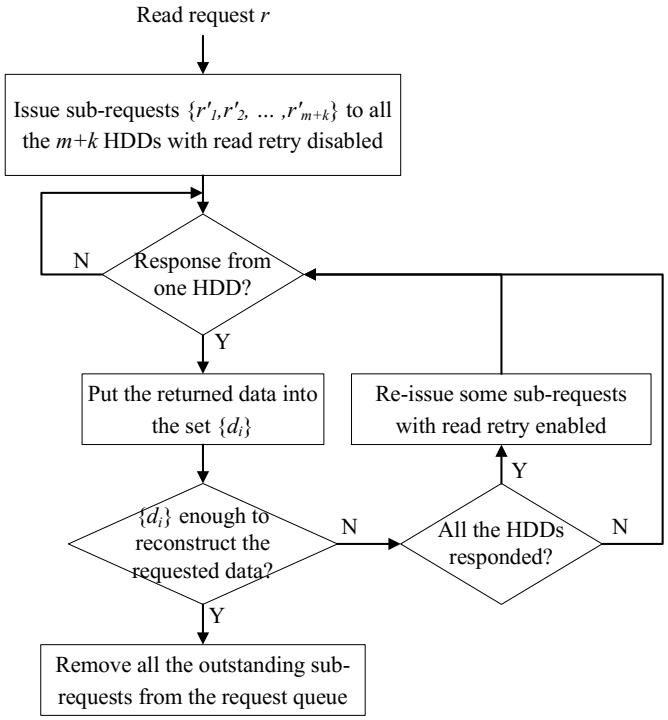

Fig. 4. Illustration of the operational flow of serving a read request in the *system-assisted mode* using the proposed implementation method.

We first discuss the estimation of $\left(\tau_i^{(N)} - \tau_i^{(S)}\right)$. Without the loss of generality, we assume the read request r spans over the first l HDDs (where $l \leq m+k$). When serving the read request r in the *system-assisted mode* with the method presented in Section III-B, we dispatch $m+k$ sub-requests $\{r'_1, r'_2, \cdots, r'_{m+k}\}$ to all the $m+k$ HDDs (with intra-HDD read retry disabled). Accordingly, we partition the set $\mathcal{R}_r$ into $m+k$ sub-sets $\mathcal{R}_r^{(j)}$ $(1 \leq j \leq m+k)$, where each $\mathcal{R}_r^{(j)}$ contains all the sub-requests dispatched to the j-th HDD. Let $\tau_{HDD}^{(j)}$ denote the average latency for the j-th HDD to serve one read without read retry (i.e., the latency of seeking the location and then flying over one or multiple sectors). For the j-th HDD, let $p_f^{(j)}$ denote the sector read failure probability, and $\tau_{retry}^{(j)}$ denote the average read retry latency once read retry is enabled. For each sub-request in $\mathcal{R}_r^{(j)}$, we have

$$\tau_i^{(N)} - \tau_i^{(S)} = \begin{cases} p_f^{(j)} \cdot \tau_{retry}^{(j)} + p_c^{(j)} \cdot \tau_{HDD}^{(j)}, & 1 \leq j \leq l \\ -(1 - p_c^{(j)}) \cdot \tau_{HDD}^{(j)}, & l < j \leq m+k \end{cases},$$

where $p_c^{(j)}$ denote the probability that the sub-request r'_j will be canceled from the host-side request queue.

Next, we will discuss how to determine the $m+k$ sub-sets $\mathcal{R}_r^{(j)}$ $(1 \leq j \leq m+k)$. Recall that each sub-set $\mathcal{R}_r^{(j)}$ contains all the future sub-requests being dispatched to the j-th HDD, whose read latency will be affected by the decision whether we serve current request r in the *normal mode* or *system-assisted mode*. When we just receive the read request r, future read requests have not arrived yet. Therefore, we have to estimate the size of each $\mathcal{R}_r^{(j)}$ through appropriate

probabilistic modeling. Let $\mathcal{Q}_j$ denote the host-side request queue consisting of all the outstanding sub-requests to be dispatched to the j-th HDD, and $q_j(t)$ denote its queue depth (i.e., the number of sub-requests waiting in the queue $\mathcal{Q}_j$) at the time of t. Given the read request r, assume we push its sub-requests into their corresponding queues at the time t_0. Let $t_e^{(j)}$ denote the time that the queue depth $q_j(t)$ drops to zero for the first time after t_0. Once the queue depth drops to zero, we can assume that the read latency of all the future subsequent requests will not be affected by any previous requests. Let $n_{t_0 \to t_e}^{(j)}$ denote the number of sub-requests that have been pushed into the queue $\mathcal{Q}_j$ between t_0 and $t_e^{(j)}$. Therefore, the decision of serving current read request r in either *normal mode* or *system-assisted mode* at the time t_0 will affect the $n_{t_0 \to t_e}^{(j)}$ future sub-requests in the queue $\mathcal{Q}_j$. Hence, we have that $|\mathcal{R}_r^{(j)}| = n_{t_0 \to t_e}^{(j)}$.

Given the queue depth of $q_j(t_0)$ at the time t_0, we have that the j-th HDD will serve total $q_j(t_0) + n_{t_0 \to t_e}^{(j)}$ sub-requests before the queue depth $q_j(t)$ drops to zero at the time $t_e^{(j)}$. Recall that $\tau_{HDD}^{(j)}$ denote the average latency for the j-th HDD to physically serve one read without read retry. Since intra-HDD read retry rate may at most be few percent, we can estimate that it will take

$$T_t^{(j)}(n)|_{n=n_{t_0 \to t_e}^{(j)}} = \big(q_j(t_0) + n_{t_0 \to t_e}^{(j)}\big) \cdot \tau_{HDD}^{(j)} \tag{2}$$

for the j-th HDD to serve all the sub-requests in the queue before the queue depth drops to zero. Meanwhile, let $P_n^{(j)}(t)$ denote the probability that n sub-requests enter the queue $\mathcal{Q}_j$ within a period of t. Therefore, we could estimate the average value of $n_{t_0 \to t_e}^{(i)}$ (i.e., the size of the sub-set $\mathcal{R}_r^{(j)}$) as

$$E\big(n_{t_0 \to t_e}^{(j)}\big) = \sum_{n=1}^{\infty} n \cdot P_n^{(j)}\big(T_t^{(j)}(n)\big). \tag{3}$$

Accordingly, based upon above discussions, we could estimate the aggregated net read latency τ as

$$\begin{aligned} \tau \;\; = \;\; & \sum_{j=1}^{l} E\big(n_{t_0 \to t_e}^{(j)}\big) \cdot \big(p_f^{(j)} \cdot \tau_{retry}^{(j)} + p_c^{(j)} \cdot \tau_{HDD}^{(j)}\big) - \\ & \sum_{j=l+1}^{m+k} E\big(n_{t_0 \to t_e}^{(j)}\big) \cdot (1 - p_c^{(j)}) \cdot \tau_{HDD}^{(j)}. \end{aligned} \tag{4}$$

In the runtime, HDDs with enhanced observability periodically update the host with the values of the sector read failure probability $p_f^{(j)}$, average read retry latency $\tau_{retry}^{(j)}$, and average read service time $\tau_{HDD}^{(j)}$. Meanwhile, host can estimate the sub-request cancellation probability $p_c^{(j)}$ based upon recent operational statistics. Host can estimate $P_n^{(j)}(t)$ based upon recent request arrival statistics. In particular, assuming that the request arrival interval can be modeled as a random Poisson process, we can easily estimate the Poisson distribution parameter based on recent request arrival statistics, and then accordingly derive the probability $P_n^{(j)}(t)$.

IV. Evaluations

We carried out experiments on a server with dual-socket Intel Xeon E5-2630 2.2GHz CPUs (10 cores per socket), 64GB DRAM, and six HDDs that form a RAID-5 with the stripe size of 8kB. Each HDD is a 2TB 7200rpm SATA 6.0Gb/s 3.5″ internal drive. To enable each HDD fully exploit the Native Command Queuing (NCQ), we run 32 threads in the user-space to concurrently dispatch read requests to each HDD, and each thread maintains its own request queue. We uniformly distribute requests among all the 32 queues associated with the same HDD. Therefore, for all the six HDDs, we run total 192 threads in parallel (hence total 192 queues) to dispatch requests to the six HDDs.

To emulate HDD fail-slow under different read retry rate and retry latency, we use a simple method described below: Let $p_f^{(j)}$ denote the read retry rate of the j-th HDD, and s_j denote the average number of disk rotations for the j-th HDD to finish one read retry. Let n_s denote the average number of 4kB sectors along each track. Given one sub-request that consists of b sectors and is dispatched to the j-th HDD with read retry enabled, we can estimate the probability that this sub-request experiences a read retry as $1 - \big(1 - p_f^{(j)}\big)^b$. In this work, we assume that HDDs can maximize their internal read retry efficiency, i.e., the same read retry operation can handle all the sector read failures encountered by one sub-request. Therefore, with the probability of $1 - \big(1 - p_f^{(j)}\big)^b$, we increase the size of this sub-request by $s_j \cdot n_s \cdot 4$kB, regardless the number of sector read failures encountered by this sub-request. Moreover, to simplify the experiments, we use a fixed small number (i.e., only 3 or 5 in this work) of disk rotations per read retry operation. In modern 3.5″ HDDs, the number of sectors per track ranges from about 200 to 400 dependent on the track position on the surface of the disk platter. In our experiments, we set that the average number of 4kB sectors per track is 300. Therefore, to emulate the occurrence of read retry when one HDD serves a read request, we increase the read request size by 3.6MB and 6MB for 3 and 5 rotations per read retry.

Moreover, to incorporate the adjacent sector failure correlation (as discussed in Section III-A), we assume that a sector read failure makes its immediately adjacent sector subject to a higher failure probability $c \cdot p_f^{(j)}$, and we set the factor c as 2 (i.e., the sector failure probability will double) in our experiments.

A. Impact of Read Retry

We first evaluated the effect of HDD fail-slow. In particular, we consider the scenarios where intra-HDD read retry rate temporarily increases from normal specs (i.e., $10^{-5} \sim 10^{-6}$) to 1% or 2% because of significant environmental variations. We carried out experiments under three different read request sizes (8kB, 24kB, and 40kB). To better reveal the impact of read request size, we ran each experiment using one request size. The read request arrival interval follows the Poisson distribution, and the starting LBA (logical block address) of

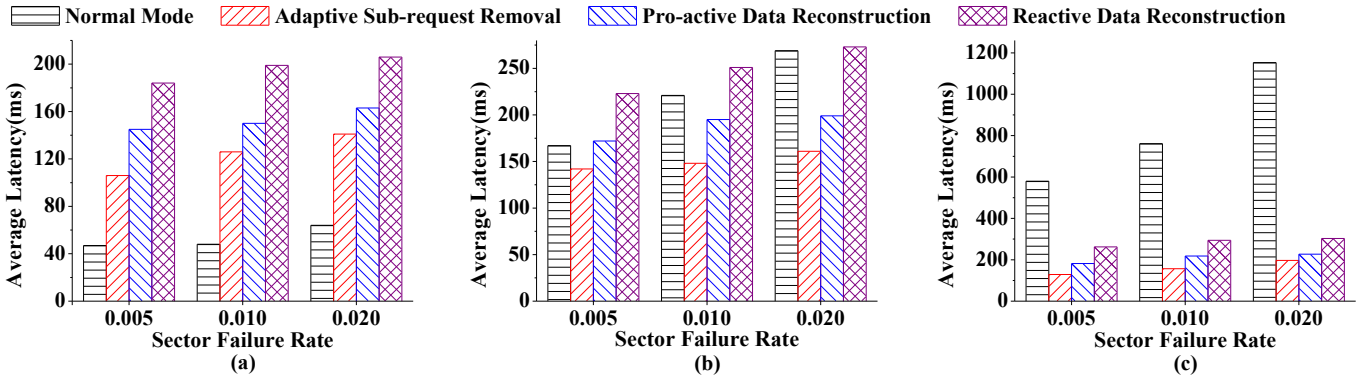

Fig. 5. Measured average read latency with *normal mode* and different *system-assisted mode* implementations when the read request size is (a) 24kB, (b) 40kB, and (c) 80kB. The mean of request arrival interval is 8ms.

each read request randomly distributes throughout the entire LBA range. Since each HDD can sustain up to around 200 random 4kB IOPS, we chose the average arrival interval of 8ms in order to create relatively high stress on HDDs and meanwhile obviate request queue overflow. Table I and Table II show the measured average and 99-percentile read latency, respectively, under different configurations. Given the same request average arrival interval, due to the absence of request queue overflow, all the different configurations have the same IOPS and only differ on read latency.

TABLE I
COMPARISON OF AVERAGE READ LATENCY.

Rotations per retry	Retry rate	Read request size		
		8kB	24kB	40kB
0	0	16ms	41ms	107ms
3	1%	18ms	48ms	221ms
	2%	19ms	64ms	269ms
5	1%	18ms	56ms	284ms
	2%	22ms	90ms	553ms

TABLE II
COMPARISON OF 99-PERCENTILE READ LATENCY.

Rotations per retry	Retry rate	Read request size		
		8kB	24kB	40kB
0	0	43ms	169ms	832ms
3	1%	63ms	236ms	1,712ms
	2%	68ms	512ms	2,190ms
5	1%	81ms	243ms	2,513ms
	2%	98ms	530ms	3,336ms

The results show the significant impact of high intra-HDD read retry rate, even though we assume each read retry only incurs 3 or 5 additional disk rotations. Its impact increases as the read request size increases. For example, with 3 additional rations per retry, when read retry rate increases from 0 to 1%, the average read latency only increases from 16ms to 18ms for 8kB request size, but it increases from 107ms to 221ms for 40kB request size. Under the read retry rate of 2% and 5 rotations per retry, the average read latency can degrade by $4\times$ for 40kB request size. Table II shows the same trend on the 99-percentile read latency. The measurement results quantitatively demonstrate that even few percentage of read retry rate with few disk rotations per retry could significantly degrade the HDD-based storage system performance, which well justifies the motivations of this work.

B. Effectiveness of Adaptive Sub-Request Removal

We carried out experiments to evaluate the effectiveness of adaptive sub-request removal when serving a request in the *system-assisted mode*. For the purpose of comparison, we considered the following three different cases on the implementation of the *system-assisted mode*:

1) *Adaptive sub-request removal*: As discussed in Section III-B, in order to better embrace the runtime variations among all the HDDs (e.g., in terms of sector read failure rate and request queue depth), we propose to implement the *system-assisted mode* by pro-actively invoking data reconstruction with adaptive sub-request removal from the host-side request queues. This case fully implements our proposed design approach.
2) *Pro-active data reconstruction*: Compared with the first case (i.e., adaptive sub-request removal), its only difference is the absence of the adaptive sub-request removal. It fetches the entire coding sector group from the RAID array and pro-actively leverages the RAID coding to reconstruct the requested data, but does not remove those unneeded sub-requests from the host-side request queues.
3) *Reactive data reconstruction*: It fetches the entire coding sector group from the RAID array but only reactively leverages the RAID coding to reconstruct the requested data and does not remove those unneeded sub-requests from the request queues. In particular, this case always first waits for the requested data from HDDs, and only if the requested data suffer from sector read failures, it will then reconstruct the requested data through RAID decoding.

We note that all the three cases use the conventional RAID-5 encoding (i.e., simple parity code). Fig. 5 and Fig. 6 show the

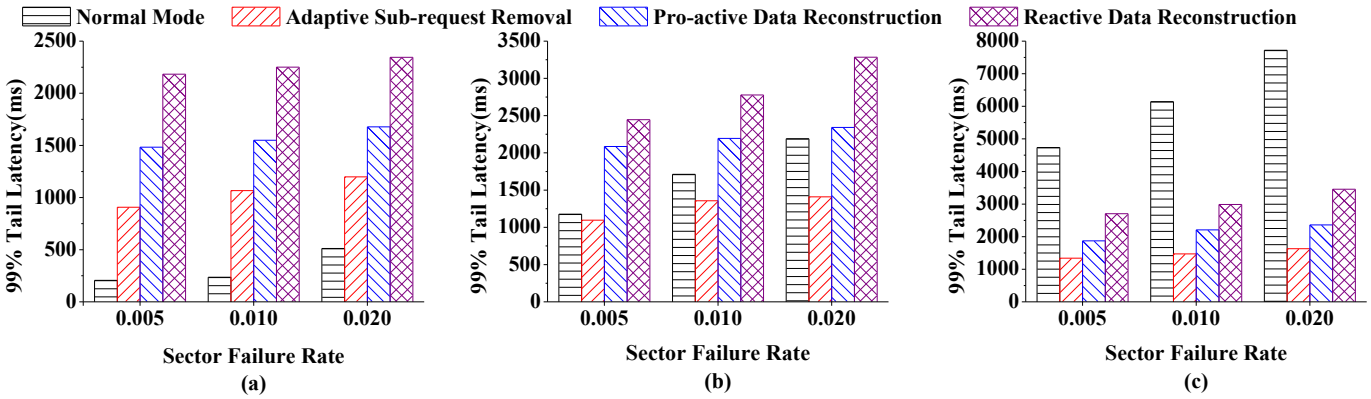

Fig. 6. Measured 99-percentile read latency with *normal mode* and different *system-assisted mode* implementations when the read request size is (a) 24kB, (b) 40kB, and (c) 80kB. The mean of request arrival interval is 8ms.

measured average and 99-percentile read latency, respectively, when using the above three different cases for implementing the *system assisted mode*. We also measured the average and 99-percentile read latency of a baseline that simply uses the conventional practice (i.e., always serve read requests in the *normal mode* with read retry enabled). The average request arrival interval is 8ms. For the baseline case, each read retry only consumes 3 additional disk rotations. The results show that small requests (i.e., 24kB) can be better served by the *normal mode* while larger requests (i.e., 40kB and 80kB) can be better served by the *system-assisted mode*. Meanwhile, the advantage of the *system-assisted mode* rapidly increases as the read request size increases (e.g., from 40kB to 80kB). This is because of the following two factors: (1) Data fetching overhead of the *system-assisted mode* reduces as we increase the request size. When the read request spans over all the HDDs (e.g., 80kB per request), the *system-assisted mode* does not incur any extra data fetching overhead. (2) As the read request size increases, the probability that one request suffers from sector read failures will increase.

As shown in Fig. 5 and Fig. 6, the proposed adaptive sub-request removal can noticeably improve the effectiveness of the *system-assisted mode*. For example, under 1% sector read failure rate and 40kB request size, the use of adaptive sub-request removal can reduce the average read latency by 24% and 41%, compared with the other two cases. Moreover, the second case (i.e., pro-active data reconstruction) can achieve noticeably better latency than the third case (i.e., reactive data reconstruction). This suggests that, even without adaptive request removal, it is still beneficial to more pro-actively leverage the RAID coding redundancy.

C. Effectiveness of eRAID

We further studied the proposed eRAID coding as presented in Section III-A. In the experiments, we set the expansion factor of eRAID as 2. Hence, for the RAID-5 over six HDDs, each eRAID codeword is an RS code that consists of 12 1-byte symbols and can correct up to 2 symbol errors. For the purpose of comparison, we also considered the baseline that uses the conventional RAID-5 over the six HDDs (i.e., each codeword is a 6-bit parity code). The mean of request arrival interval remains as 8ms. Fig. 7 shows the measured average and 99-percentile read latency under the read request size of 40kB. In the case of RAID/eRAID decoding failures, we should re-dispatch the failed sub-requests to HDDs with read retry enabled, and we set that all the re-dispatched sub-requests always experience read retries. Hence, Fig. 7 shows the results when each read retry takes 3 or 5 additional disk rotations. To minimize the latency of such re-dispatched sub-requests, we always insert the re-dispatched requests into the head of host-side request queues.

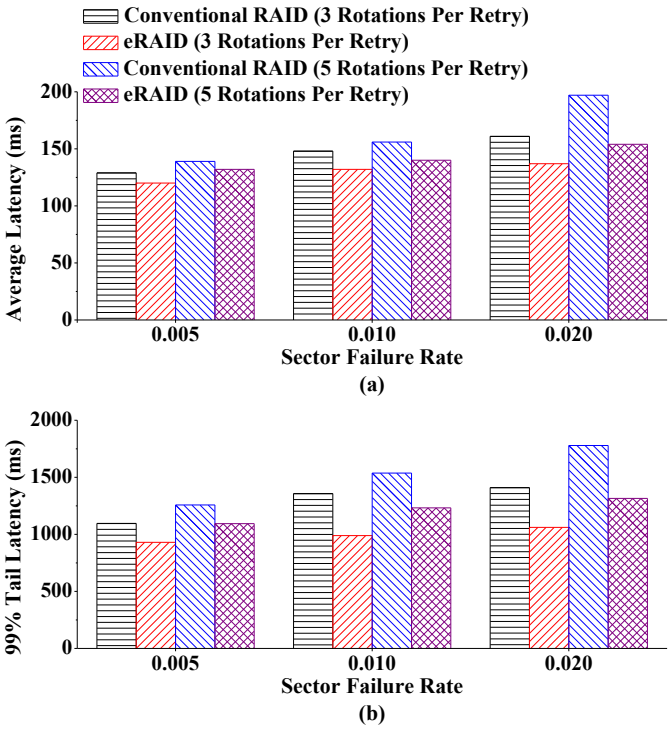

Fig. 7. Measured average and 99-percentile read latency with the read request size of 40kB.

The results show that the proposed eRAID coding strategy can noticeably reduce the read latency of the *system-assisted mode*, compared with the conventional RAID. For example, under the sector read failure rate of 1% and only 3 disk rotations per read retry, replacing conventional RAID with eRAID can reduce the average and 99-percentile read latency by 11% and 27%, respectively. The results also show that the benefit of eRAID improves as the sector read failure rates further increase.

D. Overall Performance Evaluations

We carried out further experiments under both synthetic and trace-based workloads. For synthetic workloads, we randomly mix read requests with ten different sizes i·8kB (where $1 \leq i \leq 10$), where requests with different sizes occur with the same probability. For trace-based workloads, we chose four traces from the MSR Cambridge trace set [13], denoted as MSR-1, MSR-2, MSR-3, and MSR-4, respectively. We chose two traces from the Systor'17 trace set [14], denoted as Systor-1 and Systor-2. All these six traces are read-intensive, where read requests account for at least 70% of all the requests in each trace. Table III summarizes the read request size statistics of these six traces. In Table III, the sum of each row is not 1 but the total read requests percentage among all requests. For example, the total read requests percentage for MSR-1 is 71.7%, and that means the write requests percentage is 28.3%.

TABLE III
TRACE READ REQUEST SIZE STATISTICS.

	≤8kB	(8kB, 16kB]	(16kB 32kB]	>32kB
MSR-1	56.9%	2.2%	2.1%	10.5%
MSR-2	80.5%	1.5%	1.7%	11.6%
MSR-3	28.9%	2.3%	8.0%	44.6%
MSR-4	4.8%	2.8%	0.4%	87.0%
Systor-1	30.9%	7.1%	18.0%	14.4%
Systor-2	35.9%	11.3%	15.0%	27.6%

1) Experimental Results under Synthetic Workloads: Fig. 8 shows the measured average and 99-percentile read latency when all the six HDDs experience the same sector read failure rate. For the purpose of comparison, the figure also includes the measured read latency when using the conventional practice (i.e., using the *normal mode* only to serve each request). The results well demonstrate the effectiveness of the proposed pro-active design strategy. For example, under the sector read failure rate of 1% and only 3 additional disk rotations per retry, the proposed design strategy can reduce the average and 99-percentile read latency by 44% and 46%, respectively.

In addition to assuming all the HDDs experience the same sector read failure rate during HDD fail-slow, we further carried out experiments by increasing the sector read failure rate of only one HDD. In each experiment, we randomly choose one HDD to increase its sector read failure rate while keeping the other five HDDs intact. We use the same read request statistics as above (i.e., randomly mixing read requests

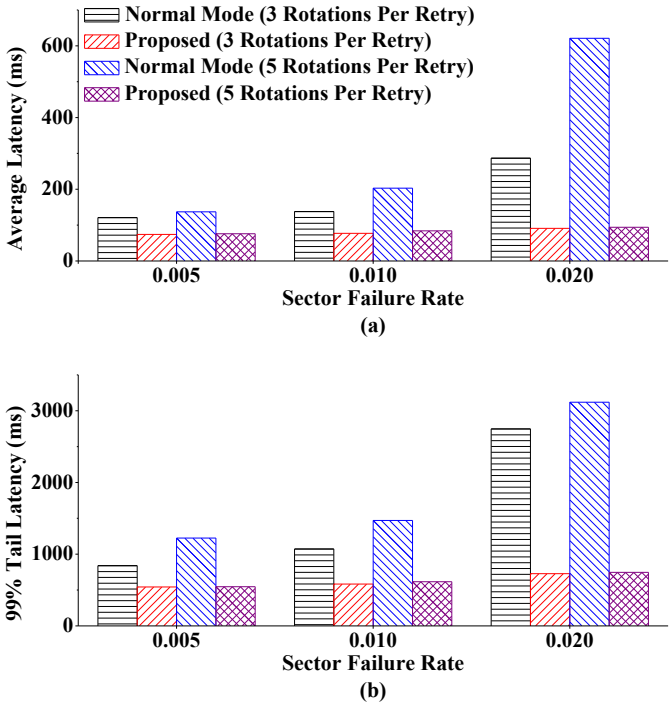

Fig. 8. Measured average and 99-percentile read latency with mixed request size under synthetic workloads. All the six HDDs experience the same sector failure rate.

with ten different sizes and equal probability). Fig. 9 shows the measured average and 99-percentile read latency. The results show that, even when only one HDD experiences high sector read failure rate, the proposed design approach can still noticeably reduce the read latency.

2) Experimental Results under Trace-based Workloads: In addition to the synthetic workloads, we carried out experiments using the six traces as described above. For all the experiments, we only considered the case of 3 additional rotations per retry, and sector failure rate of 0.5% and 2%.

Fig. 10 shows the measured average and 99-percentile read latency when all the six HDDs suffer from the same sector failure rate of 0.5% or 2%. The results clearly show the effectiveness of the proposed design solution on reducing the read latency. For example, under the sector failure rate of 2%, the proposed design solution can reduce the average and 99-percentile read latency by 34.2% and 35.1% for the trace MSR-2, and by 61.4% and 59.6% for the trace MSR-4. For the trace Systor-2, the proposed design solution can even reduce the average and 99-percentile read latency by 68.9% and 61.9%. Compared with the four MSR traces, the two Systor traces benefit more from the proposed design solution. This is because the Systor traces have smaller average request arrival time than MSR traces. Among the four MSR traces that have similar request arrival time, the benefit of the proposed solution closely relates to the percentage of large read requests. As shown in Table III, the percentage of >32kB read requests increases from the MSR-1 to MSR-4. Accordingly, the benefit of the proposed design solution improves from the MSR-1 to

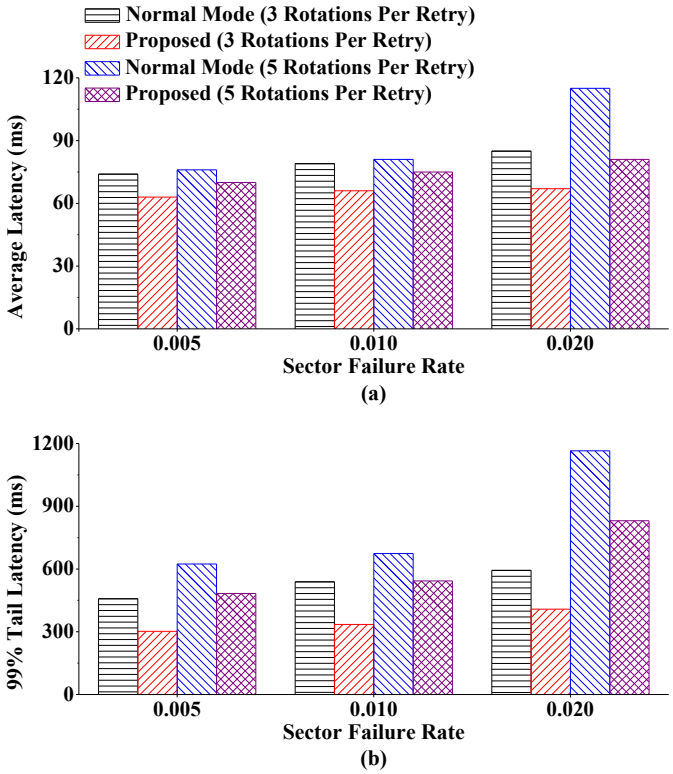

Fig. 9. Measured average and 99-percentile read latency with mixed request size under synthetic workloads. Only one HDD experiences the high sector failure rate.

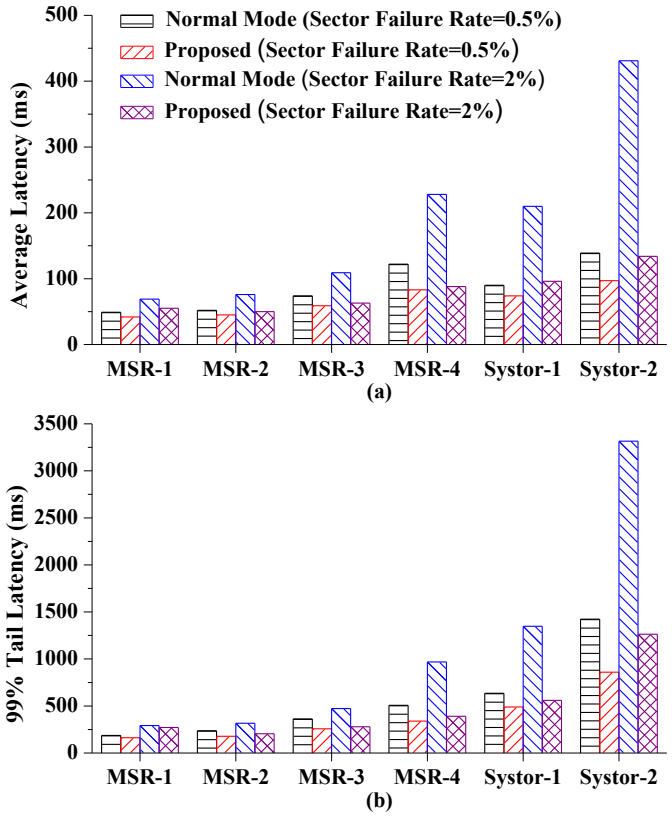

Fig. 10. Measured average and 99-percentile read latency under six different traces. All the six HDDs experience the same sector failure rate.

MSR-4, as shown in Fig. 10. The same conclusion can be drawn for the two Systor traces. Moreover, the results clearly show that the proposed design solution can make the read latency much less sensitive to the sector failure rate. When serving requests using the normal mode, the average and 99-percentile read latency of Systor-2 increase by 210.0% and 132.9% when the sector failure rate increases from 0.5% to 2%. In comparison, when using the proposed design solution, the increase drops to 38.1% and 47.0%.

Fig. 11 shows the measured average and 99-percentile read latency when only one HDD suffers from the abnormally high sector failure rate of 0.5% or 2%. The results show that the proposed design solution can still achieve noticeable or significant benefits. Similar to the scenario where all the HDDs suffer from high sector failure rate, the proposed design solution is more beneficial if the trace has a smaller arrival time and/or higher percentage of large read requests. For example, under the sector failure rate of 2%, the proposed can reduce the average and 99-percentile read latency by 17.9% and 21.1% for MSR-3, 17.5% and 39.8% for MSR-4, 35.1% and 53.1% for Systor-1, and 25.2% and 47.1% for Systor-2. The above results well demonstrate the effectiveness of the proposed design solution.

V. Related Work

Applying system-level data redundancy to tolerate HDD operational failures is of course not new. Extensive prior work has led to the pervasive real-life adoption of RAID [15]–[17] and distributed erasure coding [18]–[21]. Both RAID and distributed erasure coding aim to accommodate catastrophic HDD failures (and server unavailability in the case of distributed erasure coding) at relatively low redundancy (e.g., 20% to 50%). Most prior work around RAID mainly focused on the following two aspects: (1) Mathematically model the capability on tolerating HDD failures (e.g., see [22]–[26]) by taking into account of the more realistic failure characteristics (including the HDD latent errors). (2) Develop techniques to realize efficient data reconstruction in the presence of catastrophic HDD failures (e.g., see [27], [28]) and dynamically redistribute RAID when more HDDs have been added into the array or when the RAID system needs to be upgraded (e.g., see [29]–[31]). All the prior work tend to utilize the system-level data redundancy in a re-active manner and target at very low sector failure probabilities. To the best of our knowledge, no prior work has ever comprehensively studied the potential of pro-actively utilizing the existing system-level data redundancy to mitigate occasional HDD fail-slow with high sector read failure rates.

VI. Conclusions

This paper presents a simple yet effective design framework to mitigate occasional HDD fail-slow with high intra-HDD read retry rates. The key is to pro-actively utilize the existing

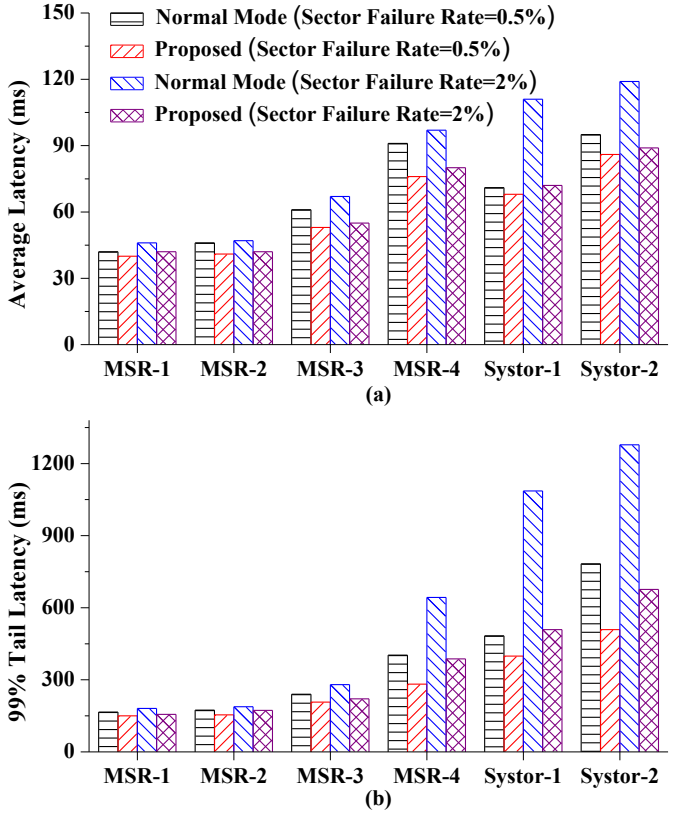

Fig. 11. Measured average and 99-percentile read latency under six different traces. Only one HDD experiences the high sector failure rate.

system-level data redundancy to minimize the impact of HDD fail-slow on storage speed performance (in particular read latency). Assuming future HDDs can support a simple form of enhanced controllability and observability regarding their internal read retry operations, this paper presents specific design techniques and a mathematical formulation framework in order to practically and effectively implement the envisioned pro-active design strategy. With a RAID-5 storage array over six 2TB HDDs, we carried out extensive experiments over a wide range of configurations and results well demonstrate the effectiveness of the proposed design framework on mitigating HDD fail-slow.

ACKNOWLEDGEMENT

This work is supported in part by the NSF grant CNS-1814890 and a grant from IDEMA/ASRC.

REFERENCES

[1] B. D. Strom, S. Lee, G. W. Tyndall, and A. Khurshudov, "Hard disk drive reliability modeling and failure prediction," *IEEE Transactions on Magnetics*, vol. 43, no. 9, pp. 3676–3684, Sept 2007.

[2] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," in *USENIX Conference on File and Storage Technologies (FAST)*, 2018, pp. 1–14.

[3] D. Weller, G. Parker, O. Mosendz, E. Champion, B. Stipe, X. Wang, T. Klemmer, G. Ju, and A. Aian, "A HAMR media technology roadmap to an areal density of 4 Tb/in^2," *IEEE Transactions on Magnetics*, vol. 50, no. 1, Jan. 2014.

[4] 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, "Integrated head assisted magnetic recording head: design and recording demonstration," *IEEE Transactions on Magnetics*, vol. 44, no. 1, Jan. 2008.

[5] K. Miura, E. Yamamoto, H. Aoi, and H. Muraoka, "Estimation of maximum track density in shingles writing," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3722–3725, Oct. 2009.

[6] F. Lim, B. Wilson, and R. Wood, "Analysis of shingle-write readback using magnetic-force microscopy," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 1548–1551, Jun. 2010.

[7] Y. Shiroishi, K. Fukuda, I. Tagawa, H. Iwasaki, S. Takenoiri, H. Tanaka, H. Mutoh, and N. Yoshikawa, "Future options for HDD storage," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3816–3822, Oct 2009.

[8] A. R. Krishnan, R. Radhakrishnan, B. Vasic, A. Kavcic, W. Ryan, and F. Erden, "2-D magnetic recording: Read channel modeling and detection," *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3830–3836, Oct 2009.

[9] R. Wood, R. Galbraith, and J. Coker, "2-D magnetic recording: Progress and evolution," *IEEE Transactions on Magnetics*, vol. 51, no. 4, pp. 1–7, April 2015.

[10] *OCP Storage Cloud HDD Fast Fail Read SubProject*. https://www.opencompute.org/wiki/Storage/CloudHDD.

[11] *AT Attachment 8 ATA/ATAPI Command Set (ATA8-ACS)*. http://www.t13.org/.

[12] *Intelligent Storage Acceleration Library (ISA-L)*. https://github.com/01org/isa-l.

[13] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.

[14] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proceedings of the 10th ACM International Systems and Storage Conference*. ACM, 2017, p. 13.

[15] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1988, pp. 109–116.

[16] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, Jun. 1994.

[17] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, Feb 1995.

[18] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proc. of the VLDB Endowment*, vol. 6, no. 5, 2013, pp. 325–336.

[19] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–14.

[20] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[21] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. of USENIX Annual Technical Conference (ATC)*, 2012, pp. 15–26.

[22] A. Ma, R. Traylor, F. Douglis, M. Chamness, G. Lu, D. Sawyer, S. Chandra, and W. Hsu, "RAIDshield: characterizing, monitoring, and proactively protecting against disk failures," *ACM Transactions on Storage (TOS)*, vol. 11, no. 4, p. 17, 2015.

[23] G. F. Hughes and J. F. Murray, "Reliability and security of RAID storage systems and D2D archives using SATA disk drives," *ACM Transactions on Storage (TOS)*, vol. 1, no. 1, pp. 95–107, 2005.

[24] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Improving storage system availability with D-GRAID," *ACM Transactions on Storage (TOS)*, vol. 1, no. 2, pp. 133–170, 2005.

[25] A. Dholakia, E. Eleftheriou, X.-Y. Hu, I. Iliadis, J. Menon, and K. Rao, "A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors," *ACM Transactions on Storage (TOS)*, vol. 4, no. 1, p. 1, 2008.

[26] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 9, 2010.

[27] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "PRO: A popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems." in *USENIX Conference on File and Storage Technologies (FAST)*, vol. 7, 2007, pp. 301–314.

[28] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/O workload outsourcing for boosting RAID reconstruction performance." in *USENIX Conference on File and Storage Technologies (FAST)*, vol. 9, 2009, pp. 239–252.

[29] A. Miranda and T. Cortes, "CRAID: online RAID upgrades using dynamic hot data reorganization." in *USENIX Conference on File and Storage Technologies (FAST)*, vol. 14, 2014, pp. 133–146.

[30] G. Zhang, Z. Huang, X. Ma, S. Yang, Z. Wang, and W. Zheng, "Raid+: deterministic and balanced data distribution for large disk enclosures," in *USENIX Conference on File and Storage Technologies*, 2018, p. 279.

[31] W. Zheng and G. Zhang, "Fastscale: Accelerate RAID scaling by minimizing data migration." in *USENIX Conference on File and Storage Technologies (FAST)*, 2011, pp. 149–161.