Tiger: Disk-Adaptive Redundancy Without Placement Restrictions

Saurabh Kadekodi, *Google;* Francisco Maturana and Sanjith Athlur, *Carnegie Mellon University;* Arif Merchant, *Google;* K. V. Rashmi and Gregory R. Ganger, *Carnegie Mellon University*

https://www.usenix.org/conference/osdi22/presentation/kadekodi

This paper is included in the Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation.

July 11–13, 2022 • Carlsbad, CA, USA

978-1-939133-28-1

Tiger: Disk-Adaptive Redundancy Without Placement Restrictions

Saurabh Kadekodi[*]
Google

Francisco Maturana[*]
Carnegie Mellon University

Sanjith Athlur
Carnegie Mellon University

Arif Merchant
Google

K. V. Rashmi
Carnegie Mellon University

Gregory R. Ganger
Carnegie Mellon University

Abstract

Large-scale cluster storage systems use redundancy (via erasure coding) to ensure data durability. Disk-adaptive redundancy—dynamically tailoring the redundancy scheme to observed disk failure rates—promises significant space and cost savings. Existing disk-adaptive redundancy systems, however, pose undesirable constraints on data placement, partitioning disks into subclusters that have homogeneous failure rates and forcing each erasure-coded stripe to be entirely placed on the disks within one subcluster. This design increases risk, by reducing intra-stripe diversity and being more susceptible to unanticipated changes in a make/model's failure rate, and only works for very large storage clusters fully committed to disk-adaptive redundancy.

Tiger is a new disk-adaptive redundancy system that efficiently avoids adoption-blocking placement constraints, while also providing higher space-savings *and* lower risk relative to prior designs. To do so, Tiger introduces the *eclectic stripe*, in which redundancy is tailored to the potentially-diverse failure rates of whichever disks are selected for storing that particular stripe. With eclectic stripes, pre-existing placement policies can be used while still enjoying the space-savings and robustness benefits of disk-adaptive redundancy. This paper introduces eclectic striping and Tiger's design, including a new mean-time-to-data-loss (MTTDL) approximation technique and new approaches for ensuring safe per-stripe settings given that failure rates of different devices change over time. In addition to avoiding placement constraints, evaluation with logs from real-world clusters shows that Tiger provides better space-savings, less bursty IO for changing redundancy schemes, and better robustness (due to increased risk-diversity) than prior disk-adaptive redundancy designs.

1 Introduction

“A Tiger never changes its stripes”, but can it be made to? In this context, the Tiger is a cluster storage system and its *stripes* are the erasure coded data that is placed across multiple disks in order to ensure data reliability. In today's cluster

[*]Equal contribution

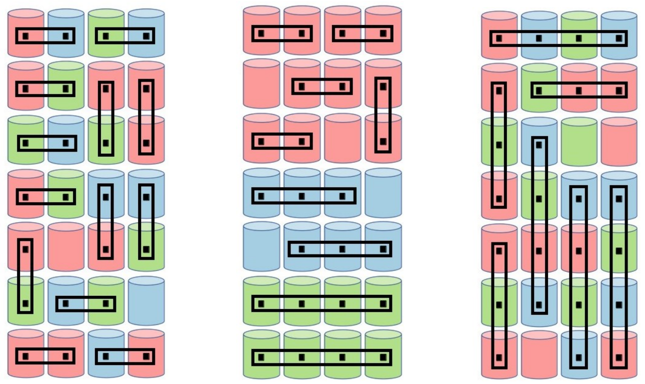

(a) Conventional cluster storage
(b) Pacemaker (subcluster-based)
(c) Tiger (this paper)

Figure 1: Stripe placements and configurations in different erasure coding systems: Disks of same color have similar annualized failure rates (AFRs), with red being least reliable (highest AFR), then blue, then green. Rectangles represent stripes with shorter stripes having higher redundancy. Conventional one-scheme-fits all designs (1a) impose no placement restrictions, but make no distinction of disk AFRs and therefore overprotect much of the data—all stripes use the widest redundancy scheme, shown as 2-wide for illustration. Pacemaker (1b) and Tiger (1c) tailor redundancy based on disk AFRs, resulting in different stripe widths in the illustration, and thereby reduce storage overhead. Pacemaker does this with rigid AFR-based subcluster boundaries, whereas Tiger requires no such boundaries.

storage systems, most of the data reliability is via erasure coding [13, 21, 37, 40, 50, 58].

Conventionally, a single cluster-wide redundancy scheme is selected for each data corpus (or for all data corpuses) [11, 14, 15, 21, 33]. This approach fails to account for the disk-reliability heterogeneity present in modern storage clusters, which consist of hundreds-of-thousands of hard disk drives (HDDs or just "disks") of multiple makes/models deployed at different times. This forces conventional storage clusters to use excessive redundancy (wasting capacity, and thus money and energy) to guarantee data safety, given that different disks have different failure rates. Absent other information, redundancy schemes are usually chosen to be safe for stripes fully stored on the least reliable disks (e.g., Fig. 1a). Recent research has showed that adapting redundancy scheme selection to the observed failure rates of specific disks can reduce the space (=cost) overhead of redundancy by up to 20% [23].

Existing disk-adaptive redundancy designs [24, 25], however, face several significant adoption hurdles. At their core, these designs rigidly partition a storage cluster into subclusters of disks (called redundancy groups or Rgroups) that have similar failure rates, so they can use a subcluster-wide redundancy scheme tailored to meet the required data reliability target (e.g., Fig. 1b). Key adoption hurdles include: (1) Since each stripe must be entirely within a single Rgroup, this subcluster-based design can interfere with other data placement considerations, such as enhancing risk-diversity by spreading data across fault domains and different makes/models/batches of disks. Indeed, many of the Rgroups consist of a single make/model. (2) To provide reasonable degrees of performance and reconstruction speed scalability, subclusters must be sizable, making these designs only suitable for very large storage clusters. (3) When failure rates rise for a given make/model, as it ages, the redundancy scheme for an entire Rgroup (potentially 100s of PBs) may need to change to maintain target data reliability levels—all at once. The Pacemaker design [24] proposes to predict such changes and start them early, but they need to predict a month or more in advance to avoid reliability problems given the huge amount of data being transitioned, which is inherently a risky proposition. (4) The subcluster-based designs assume full adoption of disk-adaptive redundancy, not allowing for selective adoption for some data corpuses but not for others.

We present Tiger, a disk-adaptive redundancy system that eliminates the placement constraints posed by subcluster-based disk-adaptive redundancy designs while providing equal or greater benefits. Tiger's core new abstraction is the *eclectic stripe*, in which disks of different AFRs can be used to store a stripe that has redundancy tailored to the set of AFRs for those disks. In terms of placement flexibility, eclectic stripes are identical to stripes in conventional (non-disk-adaptive redundancy) designs. But, unlike conventional stripes, eclectic stripes do not conservatively assume the worst-case AFR for all disks. Instead, with eclectic stripes, the redundancy scheme is dynamically set for each stripe based on the AFRs of the chosen disks (e.g., Fig. 1c). Tiger's eclectic stripe approach avoids all the adoption hurdles discussed above, while simultaneously increasing the effectiveness (higher space-savings) and robustness (lower burstiness of urgent transition IO) of disk-adaptive redundancy.

Efficiently incorporating the proposed new abstraction of eclectic stripes is challenging due to multiple reasons. Tiger introduces several new design elements to overcome these challenges. First, calculating the exact reliability in terms of mean-time-to-data-loss (MTTDL) of a stripe can be prohibitively expensive, since accounting for different failure rates can lead to an exponential number of states in the traditional Markov chain reliability model. To address this, we provide a novel approximation technique that speeds up MTTDL calculation by 2-4 orders of magnitude while always preserving accuracy of over 95%, and on average over 99.5%. Second, while disks for a stripe can be chosen based on pre-existing placement policies, the chosen disks may not form an adequately-reliable stripe for a planned redundancy scheme, since the reliability is dependent on the chosen disks' AFRs. Tiger uses an AFR-aware stripe-width-reduction policy to quickly achieve sufficient reliability. Third, disk AFRs change over time [25], which can require changing the redundancy schemes of some eclectic stripes. Keeping track of AFRs for each stripe and triggering the redundancy schemes can significantly increase the overhead for metadata and background operations. Tiger introduces an *eclectic volume* abstraction to reduce metadata overhead and make identification of required changes efficient. It also introduces policies to reduce transition IO: the IO involved with enacting changes to stripe redundancy schemes.

Evaluating the feasibility and efficacy of eclectic stripes requires analysis of long-term effects on huge storage clusters. We evaluate Tiger using the same logs as used to evaluate Pacemaker [24], enabling an apples-to-apples comparison. These logs contain all disk-deployment, failure, and decommissioning events from four production storage clusters: three 160K–450K-disk Google clusters and a $\approx$110K-disk cluster used for the Backblaze Internet backup service [3]. Simulation driven by production logs allows us to analyze reliability, space usage, and redundancy maintenance traffic for multiple clusters each with over 100K disks and over multiple years, which would be infeasible otherwise as part of a research setup. For all four clusters, Tiger provides equal or better space-savings than Pacemaker, while requiring at most 0.5% of daily IO bandwidth for transition IO. More importantly, the transition IO is both less bursty, in terms of when it is needed, and less urgent, in terms of how unsafe an unsafe stripe might be if the scheme transition were delayed. For instance, in response to a tiny rise in AFR ($< 0.25\%$) for disks of a given make/model, Pacemaker would need 196% of the total IO bandwidth from each of those disks in order to make the data safe—to avoid stealing more than 5% of IO bandwidth for transition IO, Pacemaker would have to know to start 40 days in advance—but Tiger would need $<$1.6% even for a 1% AFR increase because of the diversity of its eclectic stripes. And, most importantly, Tiger exhibits significantly better risk-diversity, stemming from removing placement constraints and allowing differently-reliable disks (and hence disks of different makes/models) to belong to the same stripe. For example, even with random selection of disks for each stripe, most of Tiger's eclectic stripes span most of a cluster's make/models; Pacemaker's strict Rgroup boundaries disallow use of more than one make/model for most stripes.

Contributions. In this paper, we make four main contributions. First, we introduce eclectic stripes as a tool for realizing disk-adaptive redundancy without the placement restrictions posed by prior designs. Second, we present a reliability model and its approximation to efficiently calculate the MTTDL of eclectic stripes. A surprising outcome is that a homogeneous stripe with the same scheme and average disk AFR as an eclectic stripe is less reliable! Third, we present the design and architecture of Tiger, the first disk-adaptive redundancy system for supporting and efficiently managing eclectic stripes. Fourth, we evaluate Tiger and compare it to the state-of-the-art, using logs from four large real-world storage clusters, demonstrating its effectiveness in realizing disk-adaptive redundancy without prior designs' adoption challenges and with greater space-savings and lower risk.

2 Background and Motivation

We first provide a primer on data redundancy done using erasure coding followed by the gist and importance of disk-adaptive redundancy. We then describe the problems with existing disk-adaptive redundancy systems, which is the motivation for this paper.

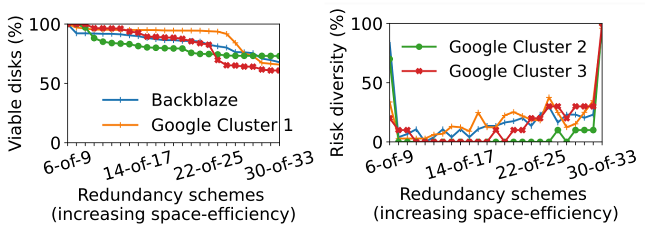
Erasure Coding for data durability. Modern storage clusters often comprise of hundreds-of-thousands of disks of multiple make/models deployed over time. The sheer scale of the storage clusters makes disk failures a common occurrence [15], which necessitates some form of redundancy to ensure data durability and availability. While replication is popular for availability of hot data, erasure coding (a more space-efficient alternative to replication) is more common for the durability of colder data, which forms the majority of the stored data. In erasure coding (EC), data is split into k chunks, and $n-k$ *parity* chunks are subsequently generated to form a *stripe* with n chunks. Each chunk is stored on a separate disk. This k-of-n EC scheme (also called "redundancy scheme") can withstand up to $n-k$ failures with a storage overhead of $\frac{n}{k}$. Any k chunks of an n-chunk stripe are sufficient to construct the original data.

Reliability Metrics: MTTDL and AFR. The reliability of a stripe is determined by its *mean-time-to-data-loss* (MTTDL). A stripe's MTTDL is calculated using a continuous-time Markov chain shown in the left side of Fig. 3. Each state represents the number of simultaneously lost chunks in a stripe. The MTTDL is the mean time to reach state DL (where $n-k+1$ chunks are simultaneously lost) from state 0; this is when data is irrecoverably lost. This model assumes a *homogeneous stripe*, where all disks fail with the same rate λ. Downward transitions denote failures, which happen with a rate of λ times the number of available chunks, while upward transitions denote repairs, which happen with a fixed rate μ. Failure rates are commonly expressed as an *annualized failure rate* (AFR), which is defined as the expected fraction of failed disks in a year, assuming that failed disks are replaced and the disk population remains fixed.

Disk-adaptive redundancy. Storage clusters have conventionally been using a one-scheme-fits-all redundancy scheme by assuming that all disks fail similarly. Prior work has shown that disk AFRs are highly correlated with their vintage [26, 35]. With modern clusters having a mix of disk makes/models/batches, there can be over an order of magnitude difference between AFRs of different groups of disks [25]. Additionally, over their lifetime, disk AFRs follow a "bathtub curve" with multiple failure regimes: infancy (high AFR) followed by useful life with potentially multiple phases (piecewise linear phases with low AFR that increases gradually) and finally wearout (high AFR) [24].

Disk-adaptive redundancy capitalizes on differences in disk AFRs and dynamically tailors data redundancy to observed disk failure rates [23]. Disk-adaptive redundancy systems take into account various constraints including the reconstruction costs when making the decision of a target stripe width to adapt to. Specifically, wide schemes are used only when a stripe's average AFR is low enough to keep the reconstruction cost contained below a configured limit. More generally, wide stripes provide cost savings in terms of smaller storage overhead at the cost of higher reconstruction costs and higher degraded mode reads. We know from conversing with architects of large-scale storage clusters that the cost of the excess byte footprint matters more than the cost of excess IO required in the context of redundancy, given existing workloads. This is especially so since, in general, large-scale capacity-tier storage cluster workloads tend to be cold (have low IO/s per byte). Additionally, cold data experiences fewer reads, and therefore has very few costly degraded mode reads. Backblaze is an example where, for archival data that has low IO access rates, administrators have publicly confirmed use of wide redundancy schemes such as 17-of-20 [4]. By using more space-efficient redundancy schemes during low AFR regimes, disk-adaptive redundancy can provide substantial space-savings ($> 20\%$) in clusters with over 100K disks.

Prior disk-adaptive redundancy systems. Two disk-adaptive redundancy systems have been proposed in the literature: HeART [25] and Pacemaker [24]. In HeART, the authors propose a tool to statistically learn the AFRs of different disk groups and identify change-points for safe redundancy transitions. By transitioning to an encoding scheme with minimum storage overhead that still meets the target MTTDL, HeART was able to obtain $\approx 20\%$ space-savings when tailoring erasure codes, and $\approx 33\%$ space-savings when tailoring replication. Although lucrative, HeART overlooked an important practical hurdle in performing disk-adaptive redundancy: *transition overload*, i.e. the IO overhead of performing redundancy transitions. Crippling transition overload when thousands of disks require simultaneous redundancy transitions forms the basis for Pacemaker [24]. The gist of Pacemaker is to convert urgent redundancy transitions into schedulable ones by making conservative predictions of the rise in AFR and proactively issuing redundancy transitions. This allows the transition overload to be spread out over time, such that it can be completed within tolerable IO limits without compromising data safety.



(a) Placement constraints **(b)** Risk-diversity

Figure 2: 2a shows Pacemaker's placement constraints by highlighting the fraction of the disk fleet that is viable for different schemes exercised on four production clusters. Fig. 2b shows the risk-diversity obtained by the same clusters on particular dates in their lifetime. A risk-diversity of 100% implies at least one chunk stored on every possible make/model, whereas a 0% risk-diversity implies that the particular scheme was not feasible in the cluster. Pacemaker performs poorly in both placement constraints and risk-diversity.

2.1 Existing designs are impractical

Despite remarkable space-savings and low IO costs, existing disk-adaptive redundancy systems remain impractical in real-world settings.

Placement restrictions. The primary hurdle stems from the placement restrictions posed by reliance on redundancy groups (*Rgroups*). An Rgroup is a set of disks with similar AFRs, such that they can use the same redundancy scheme. Prior systems redundancy management techniques rigidly partition the cluster's disks into Rgroups, and every stripe must be stored entirely within a single Rgroup. Fig. 2a shows the percentage of disks that are rendered infeasible for various redundancy schemes Pacemaker can employ on a particular day in four large storage clusters. More than 30% of the disks are deemed infeasible for space-efficient schemes beyond 22-of-25, because their AFRs are not low enough for those disks to participate in an Rgroup for which schemes beyond 22-of-25 can meet the target MTTDL. Furthermore, in order to maintain proper redundancy, stripes are typically constrained to span across different racks, servers, power lines, etc. Adding another placement constraint may be close to impossible.

Lower risk-diversity. Due to high correlation of AFRs and makes/models/batches [26,35], and in order to enable efficient transitioning mechanisms, many Rgroups contain disks from just one make/model. This is undesirable from a risk-diversity perspective. Fig. 2b shows the fraction of makes/models that are covered for the same stripe configurations in the same four clusters described above. Higher risk-diversity is valuable for mitigating consequences of bulk failure situations (e.g., from rapid degradation due to manufacturing defects), especially in a disk-adaptive redundancy system where redundancy is tuned rather than regularly excessive.

Reliance on AFR prediction. With lower risk-diversity, Pacemaker's Rgroups are already susceptible to data loss due to bulk failures in a single make/model (uncommon, but not impossible). Furthermore, Pacemaker's IO cost reduction is highly dependent on being able to accurately predict an AFR rise well in advance. Currently AFR is calculated only on the basis of age. Prior work has highlighted that it is dependent on various factors such as vintage, temperature, vibration, etc. [7, 26, 27, 35]. This makes an already difficult task of accurate AFR prediction even harder.

All-or-nothing. Current disk-adaptive redundancy designs depend on forming Rgroups, and work efficiently if entire Rgroups perform redundancy transitions together (for step-deployed disks). This implies that the entire cluster must commit to performing disk-adaptive redundancy for all of their data stored on all disks. Such a restriction makes disk-adaptive redundancy unusable without a major overhaul of the architecture of the existing storage cluster.

The key takeaway is that additional data placement restrictions create adoption-blocking limitations and risks. In order have have both placement flexibility and disk-adaptivity, we need a new approach that includes the ability to reason about and tune the reliability of stripes that span disks with different AFRs. We achieve this via ***eclectic stripes***.

3 *Eclectic Stripes* and their challenges

Eclectic stripes are central to Tiger's approach of providing disk-adaptive redundancy without placement restrictions. An eclectic stripe is an EC stripe placed on a collection of disks that can have different failure rates. The reliability model of conventional EC stripes forces them to be allocated on disks having (or worse, assumed to be having) the same failure rate. In terms of composition an eclectic stripe is no different than what a conventional EC stripe would be. Specifically, the same disks that make up a conventional stripe can also make up an eclectic stripe, just that eclectic stripes are cognizant of the AFR differences of the underlying disks and can accurately reason about the resulting reliability. A disk-adaptive redundancy system that supports eclectic stripes has to overcome several challenges.

1. Ensure efficient creation of sufficiently reliable eclectic stripes. Taking AFR differences of all disks in a stripe into account makes exact MTTDL calculation of eclectic stripes prohibitively expensive (see §4.1.1). Since stripe creation is a critical-path operation, it is imperative that a disk-adaptive redundancy system supporting eclectic stripes reasons about its reliability in an efficient and accurate manner.

2. Ensure efficient management of eclectic stripes. All underlying disks of an eclectic stripe will not experience an AFR rise or fall together. A system supporting eclectic stripes must efficiently identify which stripes need to change their redundancy in response to changing AFRs.

3. Support unchanged placement policies. While tweaking the placement policies might provide additional optimizations, a system that supports eclectic stripes must support existing placement policies without any change.

4. Retain key benefits of disk-adaptive redundancy. Dynamic redundancy adaptation at a low transition IO cost; continuously providing adequate reliability; providing space-savings by using more space-efficient redundancy schemes in low-AFR regimes are the key benefits of disk-adaptive redundancy. Any proposed disk-adaptive redundancy system should strive to maintain these benefits.

5. Ensure an adoption-friendly design. Apart from placement restrictions, existing disk-adaptive redundancy system designs require that the entire cluster commits entirely to perform disk-adaptive redundancy, or it cannot gain any of its benefits. Moreover, only the very large-scale storage clusters can use existing disk-adaptive redundancy designs, whereas the small and medium sized clusters are outside their scope. High emphasis on usability and showcasing a way for easy adoption of disk-adaptive redundancy in existing storage clusters of all shapes and sizes is an important design challenge.

4 Mechanisms to enable eclectic stripes

In this section, we address the two main challenges of eclectic stripes: their reliability and their management.

4.1 Interpreting reliability of eclectic stripes

We first shed light on key takeaways from our study of the reliability of eclectic stripes and then provide the detailed theory and the associated analysis.

Calculating MTTDL of eclectic stripes is efficient and accurate. The exact calculation of the MTTDL of an eclectic stripe is computationally expensive. We provide a novel approximation that provides the MTTDL with over 99.5% accuracy (on average), and always provides over 95% accuracy in our tests. In practice, a difference of 5% in MTTDL typically translates into a difference of around 0.1% AFR for a homogeneous stripe, which is negligible. The exact MTTDL calculation and the approximation are detailed in §4.1.1, 4.1.2.

Eclectic stripes are more reliable than homogeneous stripes. When comparing the MTTDL of an eclectic stripe with a homogeneous stripe having the same EC scheme and same avg. AFR, the MTTDL of the eclectic stripe is always higher than the MTTDL of the corresponding homogeneous stripe for typical system parameters (§4.2, Fig. 4).

Eclectic stripes are robust to AFR changes of individual disks. The MTTDL of the eclectic stripes does not react abruptly to the increase in AFR of a few disks. Compared to the conventional approach of treating stripes as homogeneous with AFR equal to the maximum AFR in the stripe, MTTDL of eclectic stripes react very gradually to AFR changes.

Eclectic stripes are more robust to AFR misestimations. Due to the nature of empirical data, any system that measures AFR has to estimate it. Since the AFRs of different disk make/models are estimated independently, it is unlikely that there will be simultaneous underestimation of the AFR of every disk in an eclectic stripe, and hence the impact of estimation errors is smaller (Fig. 5) and may even cancel each other out. Furthermore, disk-adaptive redundancy systems are made even more robust against misprediction by the use of confidence intervals. Thus, eclectic stripes are more robust to AFR misestimations compared to homogeneous stripes.

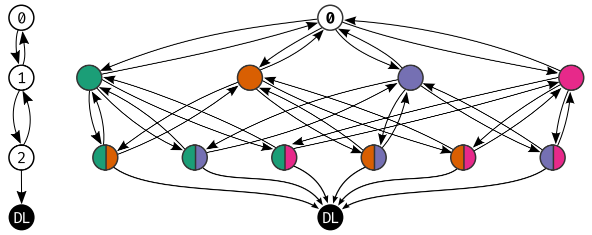


Figure 3: Left: Classic Markov chain model for the MTTDL of a 2-of-4 homogeneous stripe. Right: Markov chain model for the MTTDL of a 2-of-4 eclectic stripe.

4.1.1 Exact MTTDL calculation is costly

Using a Markov chain model to calculate the MTTDL of storage systems is a classic approach [16]. A generalization of this approach helps us take into account disks with different failure rates. Consider an EC stripe of a k-of-n scheme, placed over n disks with failure rates $\lambda_i (i \in [n])$ and a disk repair rate of μ. The state of the system is given by an n-length vector $\mathbf{s} = (s_0, \ldots, s_n)$ with $s_i = 1$ if disk i has failed, and $s_i = 0$ otherwise $(i \in [n])$. The state space is given by states $(s_i)_{i=1}^n$ such that the total number of failure $\sum_{i=0}^n s_i$ is at most the number of parities $n-k$, and a data loss state labeled DL. Therefore, the total number of states is $1 + \sum_{i=0}^{n-k} \binom{n}{i}$. The rate of transition from state $\mathbf{s}$ to $\mathbf{s}'$ is defined as:

- λ_i if $s_i = 0, s'_i = 1$, and $s_j = s'_j$ for $i \neq j$ (i^{th} disk fails),
- μ if $s_i = 1, s'_i = 0$, and $s_j = s'_j$ for $i \neq j$ (i^{th} disk repaired),
- $\sum_{i=1}^n (1 - s_i)\lambda_i$ if $\sum_{i=1}^n s_i = n-k$ and $\mathbf{s}' = DL$ (any disk fails when $n-k$ disks have failed and are not repaired).

The MTTDL is defined as the mean time to state DL from the initial state $\mathbf{0} = (0, \ldots, 0)$.

Given the values of $n, k, (\lambda_i)_{i=1}^n$, and μ, one can compute the MTTDL by using the standard approach of solving a system of equations. However, this approach is not tractable, due to the exponential explosion on the number of states with respect to $n-k$ (see Fig. 3 to compare conventional Markov chain with that of an eclectic stripe). For example, the Markov chain of a 10-of-14 eclectic stripe has 1472 states, compared to 6 states in the case of a 10-of-14 homogeneous stripe. Reasoning about this model can be hard too, since it is not directly clear how disk AFRs affect MTTDL. Furthermore, this approach tends to be numerically unstable, which makes obtaining precise MTTDLs hard. We find that computing a single MTTDL using this approach with realistic parameters can take up to several seconds using the Mathematica 12

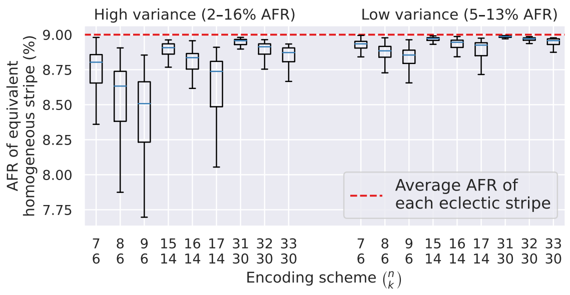


Figure 4: Reliability of eclectic stripes compared to homogeneous stripes. For each scheme, we sample 1000 eclectic stripes and for each stripe we compute its MTTDL ρ and then compute the AFR λ of a homogeneous stripe with the same scheme and MTTDL equal to ρ. The boxes show the distribution of λ over the 1000 stripes. The AFR of the first $n-1$ disks in a eclectic stripe are sampled uniformly at random from the range 2–16% (high variance) or 5–13% (low variance), and the AFR of the last disk in a stripe is chosen to ensure that the average AFR of the disks *in each stripe* is fixed at 9%. E.g. the median 6-of-9 eclectic stripe from the high-variance group is as reliable as a 6-of-9 homogeneous stripe with AFR 8.5%, despite having an average AFR of 9%.

software [52] on a desktop PC. This is too slow in practice, because not only do we need to compute the MTTDL when creating new stripes, but we also need to periodically compute the MTTDL of every stripe in the system (typically billions) as device AFRs change. The next section describes an efficient approximation that makes the MTTDL calculation of eclectic stripes computationally tractable and highly accurate.

4.1.2 Efficient and accurate MTTDL approximation

In order to compute and better understand the MTTDL of eclectic stripes, we propose an approximation formula, building on the approach presented in [2] for homogeneous stripes. This approximation is extremely good when $\mu \gg \max_i \lambda_i$, which is true for modern cluster storage systems.

The main idea behind this approximation is to note that (in the steady state) disk i will be available a fraction $A_i = \mu/(\mu+\lambda_i)$ of the time, and that the system will reach the *DL* state when exactly $k-1$ of the disks are available. Therefore, the MTTDL can be approximated with the following formula (see appendix A for the full derivation):

$$\text{MTTDL} \approx \left(\mu(n-k+1)\,\text{PBin}(k-1;n,(A_i)_{i=1}^{n})\right)^{-1}, \quad (1)$$

where $\text{PBin}(k;n,(p_i)_{i=1}^{n})$ is the probability of obtaining exactly k heads when flipping n biased coins with probability of heads p_i for coin i. PBin is known as the Poisson-binomial distribution, and it can be efficiently evaluated [12, 19].

We tested this approximation against the Markov chain approach over all values of $6 \le k \le 30$, $1 \le n-k \le 3$, and AFRs of 1–16%. The relative difference between the two output MTTDLs never exceeded 5% and was less than 0.5% on average[*]. As a benefit, the approximation is 2–4 orders of magnitude faster to evaluate (in the order of milliseconds), more numerically stable, significantly simpler to implement, and gives direct insight into how the parameters affect MTTDL.

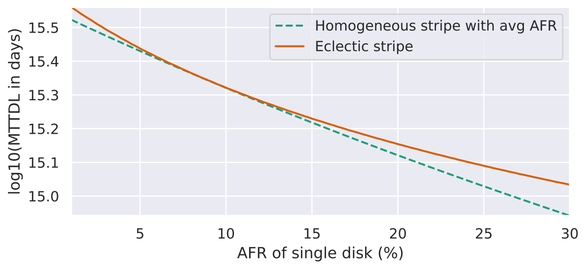


Figure 5: Reliability of a 6-of-9 eclectic stripe when the AFR of a single disk varies. The eclectic stripe is composed of 8 devices with AFR 9%, and one device whose AFR varies from 1% to 30% (x axis). The dashed line denotes the MTTDL of a 6-of-9 homogeneous stripe with the same average AFR as the eclectic stripe. The solid line denotes the MTTDL of the eclectic stripe. Reliability of the eclectic stripe is always above the corresponding homogeneous stripe.

4.2 Understanding MTTDL of eclectic stripes

The main difference between the reliability of an eclectic stripe and a homogeneous stripe is given by the Poisson-binomial factor in Eq. 1, which becomes Binomial when all probabilities are equal. Notice that the difference between A_i in Eq. 1 will be small because $\mu \gg \max_i \lambda_i$, and therefore the corresponding Poisson-Binomial distribution will not deviate too much from a Binomial distribution with trial success probability $A = \sum_{i=1}^{n} A_i/n$ [6]. Furthermore, we have:

$$\sum_{i=1}^{n} \frac{A_i}{n} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{1+\lambda_i/\mu} \approx \frac{1}{n}\sum_{i=1}^{n}\left(1-\frac{\lambda_i}{\mu}\right) = 1 - \frac{\sum_{i=1}^{n}\lambda_i/n}{\mu},$$

where we use the approximation $1/(1+x) \approx 1-x$ for small x. This means that the reliability of an eclectic stripe will tend to be close to the reliability of a homogeneous stripe with AFR equal to the average AFR of the eclectic stripe.

To measure how close the MTTDL of an eclectic stripe will be to that of a homogeneous stripe with the same scheme and average AFR, we conduct two numerical experiments. Fig. 4 compares eclectic stripes against homogeneous stripes that have the same MTTDL, across different schemes and AFR ranges. In this experiment, instead of directly showing an MTTDL ρ (which is hard to interpret) in the y-axis, we show the AFR λ of a homogeneous stripe that has MTTDL equal to ρ (under the relevant scheme). The results show that eclectic stripes are *more* reliable than homogeneous stripes with the same scheme and average AFR. In other words, for a homogeneous stripe composed of disks with AFR λ to match

[*]The median relative difference between the exact and approximated eclectic stripe MTTDL was 0.1%, the 90th percentile error was 0.5%, and the 95th percentile error was 0.7%.

the reliability of an eclectic stripe with AFRs $(\lambda_i)_{i=1}^n$, the disks in the homogeneous stripe have to be more reliable on average, i.e., $\lambda < \sum_{i=1}^n \lambda_i/n$. The difference, however, becomes small when the ratio n/k is small, or the range of AFRs is small. Fig. 5 shows the reliability of an eclectic stripe when the AFR of a single disk in the eclectic stripe varies in the range 1–30%. This experiment shows that eclectic stripes provide a dampening effect against AFR rises of a small number of devices in two ways: (1) a small number of devices have a smaller impact on the average AFR of the stripe (slope of the dashed line), and (2) the convex shape of the curve shows that the eclectic stripe is even more reliable than a homogeneous stripe with the same scheme and average AFR.

Checking if a stripe is safe: Typically, a minimum level of reliability is set in the cluster by setting a *MTTDL threshold* that all stripes must satisfy in order to be deemed safe. Given the results presented in this section, we now describe a simple method to determine whether a stripe is safe. We define the *critical AFR* of a k-of-n scheme and MTTDL threshold θ as the highest AFR that disks in a homogeneous k-of-n stripe can attain while still having an MTTDL of at least θ. The critical AFRs for the different schemes that are used in a system can be precomputed and stored. Then, a simple andx efficient way of checking whether an eclectic stripe under some scheme is safe is to check whether the average AFR in the stripe is less than the critical AFR for that scheme. Since an eclectic stripe is at least as reliable as a homogeneous stripe with the same scheme and average AFR, if the stripe passes this check, then we can be certain that the stripe is safe. If the stripe does not pass the check, then it *may* be unsafe, which can determined by computing its MTTDL. This test can help greatly reduce the amount of work needed in checking whether stripes are still safe, and it also provides a simple way of understanding the reliability of eclectic stripes.

4.3 Eclectic Volumes

Disk AFR changes may trigger redundancy transitions. Prior designs performed disk-adaptive redundancy at the disk level. Thus, if a disk's AFR changed, either all or none of the stripes on that disk required a redundancy transition. With eclectic stripes, each disk may store chunks of stripes with different reliabilities. An AFR change might only require redundancy transitions for a subset of those stripes. With millions of eclectic stripe chunks being stored on each disk, a linear search through all of them for each AFR change is impractical.

An eclectic volume is a collection of eclectic stripes that use the same EC scheme and are stored on the same set of disks. A disk can contain multiple volume fragments identified by their globally unique volume ID. Each disk maintains a map of stripe ID to eclectic volume ID. Since each eclectic volume spans the exact same disks, whenever a disk's AFR changes, Tiger only needs to check whether the EC scheme used for each of the disk's constituent volumes still meets the required MTTDL target. There is no need to check the reliability of each of the individual eclectic stripes within a volume since they are all identically reliable. The details of how Tiger manages eclectic volumes is described in §5.3.

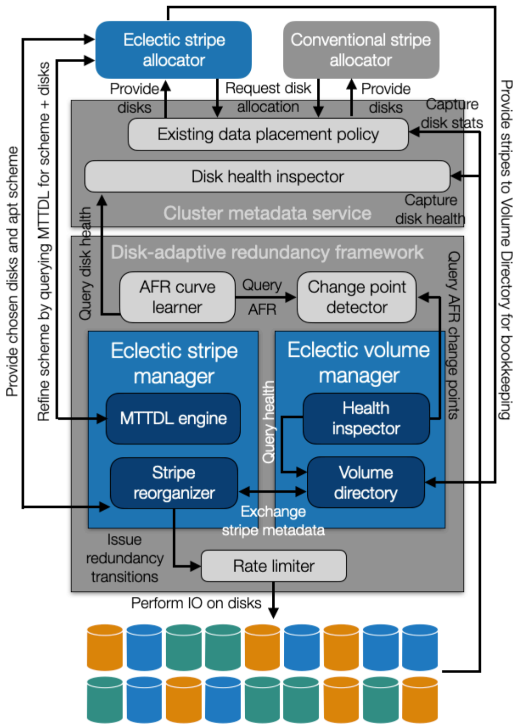


Figure 6: Architecture of Tiger. The blue boxes correspond to Tiger's components. The gray boxes correspond to existing components in cluster storage system architecture and components present in existing disk-adaptive redundancy systems.

Eclectic volumes prove to be efficient only if they represent a large number of eclectic stripes. Therefore, in Tiger the default size of an eclectic volume is set to 1 TeraByte (TB). This way, even though Tiger performs reliability monitoring at the volume granularity it ensures that each eclectic stripe is always sufficiently reliable.

5 Design and working of Tiger

Tiger is a practical disk-adaptive redundancy system designed to overcome the challenges described in §3. Fig. 6 shows the architectural components of Tiger (colored boxes) and how they interact with existing cluster storage system components and common disk-adaptive redundancy components.

5.1 Data flow in Tiger

We overview Tiger by explaining the lifecycle of eclectic stripes. An eclectic stripe is created via the *Eclectic Stripe Allocator* (ESAllocator), which identifies a set of disks and the corresponding scheme on which this data is to be stored. The ESAllocator uses the existing and unmodified data placement policy to obtain a set of disks. That placement policy uses whatever knowledge designers choose (e.g., available freespace, load balance, and fault domain constraints) in selecting the set of disks. The ESAllocator then queries the *Eclectic Stripe Manager's MTTDL Engine* (ESMTTDLEngine) with the AFRs of the chosen disks, and a stripe configuration, to verify that the planned stripe's MTTDL meets the required target MTTDL. If it does not, the ESAllocator boosts the MTTDL by changing the stripe configuration until an appropriately safe redundancy scheme is found. §5.2 details this process.

Once created, the ESAllocator passes the stripe to the Eclectic Volume Manager (EVManager, see §5.3) to either add the stripe to an existing volume, or create a new volume which will contain the new stripe. The Eclectic Volume Health Inspector (EVHInspector) continuously monitors the reliability of the eclectic volume by querying the change point detector, which identifies significant AFR changes in the data from the AFR curve learner. The AFR curve learner, change point detector and the rate limiter can be reused without change from any existing disk-adaptive redundancy system[*]. In reaction to a significant AFR change (rise or fall), the EVHInspector alerts the EVManager, which fetches the eclectic stripe metadata from the EVDirectory and provides both the AFR change and the metadata to the Eclectic Stripe Reorganizer (ESReorganizer; see §5.2). The ESReorganizer includes techniques to efficiently perform redundancy transitions. If eclectic stripes must change, the ESReorganizer consults the ESAllocator in forming them. Non-urgent redundancy transitions (when the target MTTDL is not at risk of being violated) are throttled by the rate limiter in order to not overwhelm the storage cluster.

Tiger's stripe-by-stripe disk-adaptive redundancy approach enables incremental adoption by allowing data to be stored either as an eclectic stripe or a homogeneous stripe. This is in contrast to subcluster-based designs that are all-or-nothing.

5.2 The Eclectic Stripe Manager

The Eclectic Stripe Manager (ESManager) handles construction, maintenance and reorganization of eclectic stripes.

Constructing eclectic stripes In the absence of an existing eclectic volume that has space (described later in §4.3), the ESAllocator asks the existing data placement policy for disks to store each new eclectic stripe. Since that placement policy is unaware of disk-adaptive redundancy, it may return a set of disks whose AFRs produce an MTTDL that either fails to meet or far exceeds the target MTTDL. Algorithm 1 describes the process to build a space-efficient, yet adequately reliable eclectic stripe.

To give itself flexibility, ESAllocator asks the placement policy to provide a set of disks for the maximum-width-allowed stripe (e.g., 33 for 30-of-33). The ESAllocator then queries the ESMTTDLEngine with the provided disks and its planned scheme to get the MTTDL value. If the MTTDL does not meet the target MTTDL, ESAllocator discards a disk from the set and increases the redundancy of the corresponding scheme (e.g., 29-of-32 instead of 30-of-33) to boost the stripe's MTTDL, repeating this process until sufficient MTTDL is achieved. This process is guaranteed to terminate, since the least space-efficient scheme in a storage cluster must meet the target MTTDL. Moreover, by iterating from the most space-efficient scheme allowed, the algorithm terminates at the most space-efficient scheme for the provided disks.

[*]Tiger reuses the Ruptures change-point detection library [47, 48], the AFR curve-learner and the rate-limiter from HeART [25] and Pacemaker [24].

Algorithm 1

```
$\theta_{\text{MTTDL}} \leftarrow$ target MTTDL
$n_{\max} \leftarrow max\{n \mid (n,k) \in \text{schemes}\}$
$(d_1,\ldots,d_{n_{\max}}) \leftarrow n_{\max}$ randomly sampled devices
for $(n,k) \in$ schemes in order of increasing $n/k$ do
    if MTTDL$(n,k,(d_1,\ldots,d_n)) \geq \theta_{\text{MTTDL}}$ then return $(n,k)$
```

Ensuring reliability amid disk failures. The reliability of each eclectic stripe is a function of the AFRs on the disks on which it is stored. So, when a disk fails, the reconstructed data cannot simply be placed on a randomly chosen disk, since its AFR might be high enough to cause the eclectic stripe's MTTDL to exceed the target. Recall, from §4.2, that the critical AFR of an EC scheme is the highest AFR that a homogeneous stripe of that scheme can reliably support, and a simple way to test that an eclectic stripe is safe is to check that its average AFR is below the critical AFR for its EC scheme. Therefore, we can ensure that reliability will be preserved if we choose a disk that keeps the average AFR of the affected stripes under their respective critical AFRs.

When a disk in Tiger fails, the EVManager is notified. This triggers a lookup in the EVDirectory for eclectic stripes whose chunks need to be reconstructed. The EVManager forwards the list of chunks to the ESReorganizer. For each stripe, the ESReorganizer asks the ESAllocator for disks to replace the failed disks, providing the critical AFR for the stripe. The ESAllocator returns suitable disks, if they are found, otherwise, it allocates (one or more) new eclectic stripes and moves the prior stripe's data (including any reconstructed data) to the new stripes. Finding sufficiently reliable disks to store the reconstructed data results in lower transition IO than allocating new eclectic stripes, since the latter involves moving data of disks that did not fail. After the reconstruction process (whether or not new eclectic stripes are formed), ESReorganizer informs the EVManager of the changes, which then updates the EVDirectory accordingly.

Dealing with AFR changes over time A disk's AFR is not constant throughout its lifetime [9, 10, 23, 56]. In addition to building and maintaining eclectic stripes, ESManager must also ensure that data is kept safe when a disk's AFR changes.

Ensuring data reliability with increasing AFRs. The EVManager monitors AFR by querying the change point detector.

Whenever the AFR rises, the EVManager identifies any eclectic volumes whose data is at a risk of becoming under-reliable. It alerts the ESReorganizer, with the necessary stripe metadata of such stripes, which calls the ESAllocator with the current and previous disk AFR values and the number of chunks that need reallocation onto safer disks.

As with failed data reconstruction, ESAllocator prefers finding suitable disk alternates whose AFRs are less than or equal to previous AFRs values of the disks whose AFRs rose. If ESAllocator cannot find suitable disks, new eclectic stripes are formed and data is moved, as described previously.

Reducing data over-protection with reducing AFRs. When a disk's AFR decreases, there is no reliability threat to the data stored on that disk, but there may be an opportunity to reduce redundancy and obtain space-savings.

The simplest way (that also entails no transition IO cost) of reducing a stripe's redundancy is by deleting excess parities[*]. However, deleting parities is rarely an option for two reasons. First, most storage clusters have a minimum requirement on the number of parities per stripe, set by the system administrator. Second, adding/deleting a parity has a much higher impact on the MTTDL value of a stripe than adding/deleting a data chunk— deleting even a single parity usually makes the stripe miss the target MTTDL. When ESReorganizer receives metadata of possibly over-redundant stripes from the EVManager, it queries the ESMTTDLEngine whether reducing parities is feasible and, if so, enacts the change.

When deleting parities is not an option, there are two additional ways redundancy can be reduced. First, the ESAllocator could find candidate disks with AFR higher than the current disk's AFR, but low enough that the mean AFR is below the stripe's critical AFR. This method is cost-effective, since it involves only reading and writing those chunks that are on over-protected disks. Second, if the ESAllocator cannot find suitable disks, it performs new stripe allocations if it can find a new eclectic stripe with lower storage overhead. Although re-allocation has a high IO overhead (since it involves copying data over to the new stripe), it is not urgent when lowering redundancy and can be throttled by the rate limiter without putting any data at risk.

The eclectic stripe reorganizer (ESReorganizer). The ESReorganizer uses several techniques to ensure adequate reliability and provide maximum space-savings.

At any given time, the ESReorganizer might be dealing with multiple eclectic stripes seeking possible changes. ESReorganizer processes requests in priority of maintaining reliability: failed data reconstruction, then near-risk stripes that need to increase their redundancy, then requests of decommissioning disks to move data off of them, and then stripes seeking a redundancy reduction. It processes eclectic stripes that are requesting reduction in redundancy in descending order of their storage overhead.

[*]Deleting parities may not work reducing redundancy of non-MDS codes.

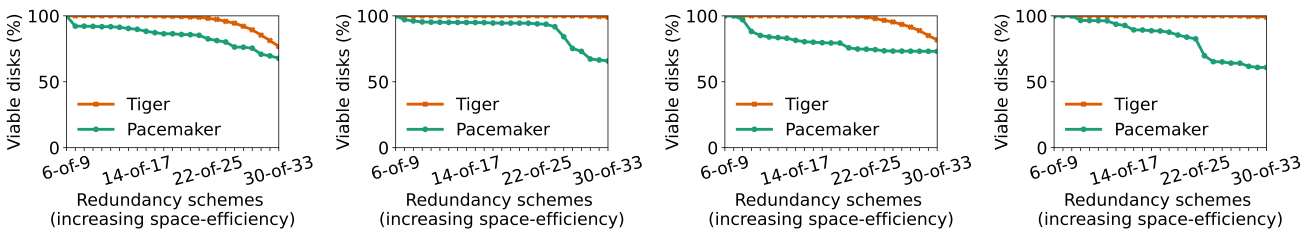
5.3 The Eclectic Volume Manager

The EVManager is responsible for creating, maintaining and monitoring the health of eclectic volumes. Recall (from §4.3) that an eclectic volume (typically in TBs) contains hundreds-of-thousands of eclectic stripes (typically in MBs). Along with health, the EVManager maintains usage statistics (e.g., freespace and load) for each eclectic volume.

Constructing and populating eclectic volumes. Similar to how ESManager manages eclectic stripes, EVManager dynamically creates and destroys eclectic volumes. The construction of the first eclectic stripe forces the creation of the first eclectic volume on the same set of disks that are chosen by the ESAllocator. When creating subsequent eclectic stripes, the ESAllocator first queries the EVManager to check if there are eclectic volumes that are conducive for storing new stripes. The EVManager does this by maintaining capacity and load-balancing metrics for each eclectic volume. Thus, the EVManager also avoids hot-spotting within eclectic volumes by spreading hot data evenly across multiple eclectic volumes. Once the target eclectic volume is identified, the set of disks comprising the eclectic volume are returned to the ESAllocator. If there is no space available, the ESAllocator gets a new set of disks from the placement policy which causes EVManager to create a new eclectic volume atop those disks. Tiger's eclectic volumes operate similar to Ceph's placement groups [51].

The Eclectic Volume Directory. Recall from §4.3 that eclectic volumes are simply a logical grouping of all the eclectic stripes with the same redundancy scheme on the same set of disks. Each eclectic volume has a unique entry in the EVDirectory and stored against the eclectic volume ID are the disks on which the eclectic volume is stored. In addition, the EVDirectory also contains a mapping from disk serial number to list of volume IDs whose fragments are stored on that disk. Note that the size of this metadata is very small. With TB-sized volume fragments, even a 100K disk storage cluster with 20TB disks will have an EVDirectory less than 100MB.

The tiny size of the EVDirectory also implies that it is unlikely to be a bottleneck. The EVDirectory will typically be queried and updated whenever disks fail, or their AFR increases significantly (in order to fetch the eclectic volumes IDs stored on the affected disks). It might also be queried to fulfill an allocation request in order to get the disks on which an eclectic volume is stored, if the eclectic-volume-to-disks mapping is not cached. Even a cluster with 500K disks has at most a few hundred disk failures in a day and typically not more than 10 makes/models, thus limiting the EVDirectory updates to less than 1000 per day. Although allocations are more frequent, caching can filter most queries for them, and their rate is also much lower than the rate of file metadata lookups in a cluster with billions of files. And, if necessary, traditional metadata scaling techniques can be employed to prevent EVDirectory from becoming a bottleneck.



(a) Backblaze **(b)** Google Cluster 1 **(c)** Google Cluster 2 **(d)** Google Cluster 3

Figure 7: Placement constraints posed by Tiger compared to Pacemaker by observing the percentage of the disk fleet that is viable for the different redundancy schemes. Tiger has lower placement constraints than Pacemaker. Tiger has over >75% disks being viable for all four clusters for all scheme configurations. Pacemaker's placement constraints are more pronounced in Google clusters since they are mostly step-deployed. This results in strict Rgroup boundaries disallowing disks from different makes/models being a part of the same Rgroup.

Reacting to failures and AFR changes. The EVHInspector continuously polls the change point detector and the cluster metadata service to gather information about disk failures and significant AFR changes. For all significant changes, the EVHInspector reconfirms the MTTDL of the affected volumes by querying the ESMTTDLEngine with the changed AFRs. Even though it is technically not a stripe, a EVDirectory has all information required to calculate the reliability of an eclectic volume, viz. the AFRs of the disks on which the volume resides, and the redundancy scheme configuration. Due to its small metadata footprint, EVHInspector can check the health of billions of stripes by checking the reliability of only thousands of eclectic volumes.

Whenever a disk fails, or a disk's AFR increases, the EVHInspector looks up the EVDirectory to find the volumes affected due to this failure / AFR rise. If the disk in question is alive, the volume manager queries the disk to obtain the stripe IDs belonging to that volume ID. If the disk has failed, the EVHInspector queries other disks of that particular eclectic volume and gathers the stripe IDs from them. Note that all disks storing a particular eclectic volume have the same list of eclectic stripe IDs in common (but they also each may have other stripes as well from non-overlapping eclectic volumes).

The EVHInspector then forwards the list of stripe IDs to the ESReorganizer along with the updated and previous AFR information and the action to be taken (reconstruct data, increase redundancy or reduce redundancy). On performing the appropriate task, the ESReorganizer communicates the metadata changes back to the EVManager, and the EVManager subsequently reflects it in the EVDirectory. For reconstruction and increase in redundancy, if a replacement disk is found, and has enough capacity to accommodate all chunks of the failed disk / disks whose AFR has increased, the eclectic volume of all constituting eclectic stripes after the operation remains the same. For redundancy reductions, or in case of not finding a replacement disk, or not finding one with enough capacity, the eclectic stripes depart from their original eclectic volume (unlike Ceph's placement groups) since they will now be stored on potentially different subset of disks.

6 Evaluation of Tiger

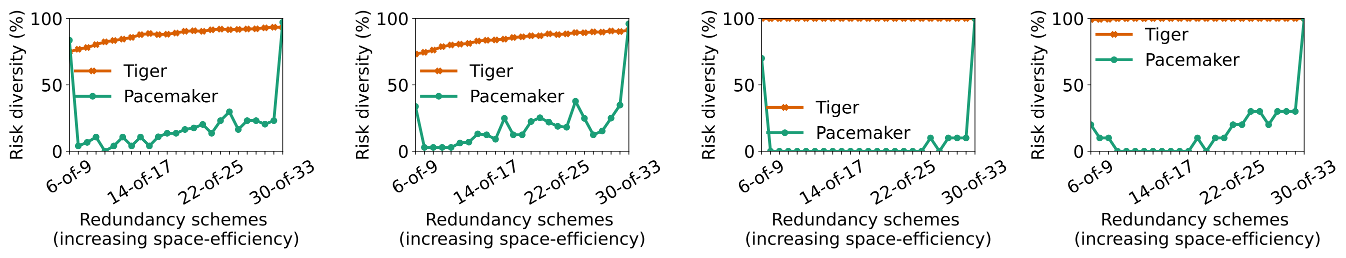
We now evaluate how Tiger performs on real-world data, and show how it fulfills the challenges laid out in §3. Tiger is evaluated using real-world deployment and failure logs from four production clusters at two different organizations (Google and Backblaze). Each cluster has a multi-year lifetime and disks from multiple makes/models/batches. Backblaze uses *trickle*-deployed disks. These disks are added to the cluster every few days in the tens or hundreds. Google Cluster 2 and Cluster 3 have *step*-deployed makes/models where disks are introduced into the cluster in large batches of tens-of-thousands of disks within a very short span of time. Google Cluster 1 is a mix of step- and trickle-deployed disks.

The highlights of our evaluation are (1) Tiger significantly lowers placement restrictions posed by Pacemaker (existing state-of-the-art disk-adaptive redundancy system); (2) Tiger's eclectic stripes provide much higher risk-diversity compared to Pacemaker; (3) Tiger is closer to the target MTTDL, and thus more efficient than existing disk-adaptive redundancy approaches; (4) Tiger outperforms Pacemaker in space-savings while keeping the average transition IO $<= 0.5\%$ and peak transition IO $< 5\%$ of cluster IO bandwidth and (5) Tiger's eclectic stripes are less sensitive to rising AFR and provide better data safety.

6.1 Tiger enables flexible data placement

We capture the flexibility in data placement by measuring the percentage of the disk fleet that is considered viable for storing data using a particular redundancy scheme. The viability is decided by whether the data stored on those disks will meet the target MTTDL. The X-axis in Fig. 7a shows the various schemes that can be supported in each storage cluster[*]. For estimating Tiger's viable disk candidates, we perform a Monte-Carlo simulation on specific days in each

[*]The narrowest scheme is set to 6-of-9 and widest is set to 30-of-33. Schemes with higher width have lower redundancy since the number of parities are kept the same. This is based on reference to prior work [24, 25], and also on the basis of communication with storage administrators of large-scale cluster storage systems at various organizations.



(a) Backblaze risk-diversity **(b)** Google Cluster 1 risk-diversity **(c)** Google Cluster 2 risk-diversity **(d)** Google Cluster 3 risk-diversity

Figure 8: Risk-diversity achieved by Tiger over three large-scale cluster storage systems. All three plots are average risk-diversity measurements taken over 5 days spread equally over the lifetime of the clusters. Pacemaker due its Rgroup based design has much lower risk-diversity compared to Tiger, more evident in Fig. 8c and 8d which are entirely step-deployed clusters.

of the cluster's lifetime. We allocate 1000 eclectic stripes by picking disks uniformly at random and check how many of the possible schemes can use the chosen disks. For Pacemaker, we bin the disks by AFRs to mimic Rgroups and measure the ratio of the population of the Rgroups to the entire disk fleet.

Tiger has almost all disks available for allocation for any scheme in Google Clusters 1 and 3 (Figs. 7b, 7d), whereas in Backblaze and Google Cluster 2 (Figs. 7a, 7c) at most 25% disks are deemed not viable for the widest schemes (beyond 22-of-25). When a large fraction of disks of the cluster have a high AFR (as is the case with Backblaze and Google Cluster 2 for the chosen dates), formation of eclectic stripes ends up with mostly high AFR disks. In such situations, Tiger cannot employ a very space-efficient redundancy scheme. Pacemaker's strict Rgroup boundaries, on the other hand, limit all disks in an Rgroup to a single scheme that may not be very wide. Therefore, for Pacemaker, all clusters see a significant drop in viable disks as the width increases.

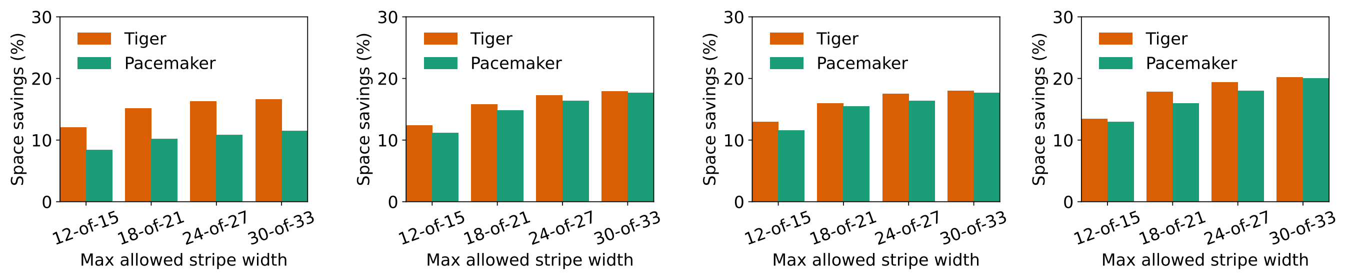
6.2 Tiger achieves high risk-diversity

Risk-diversity of a stripe is directly proportional to the number of unique makes/models participating in that stripe. If all makes/models in the storage cluster have representation in the stripe, its risk-diversity is defined to be 100%. A 0% risk-diversity implies that there were no disks in the cluster that could be used for the particular scheme. The setup used for evaluating risk-diversity is a Monte-Carlo simulation, where 100 stripes were allocated for each scheme configuration by choosing disks uniformly at random. For Tiger, we measure risk-diversity by capturing the average number of unique disk makes/models on which the chunks of an eclectic stripe are stored for each stripe configuration. For Pacemaker, we again bin the disks by AFR to form Rgroups, and count the unique number of makes/models within each Rgroup. We take the average of this simulation performed on five equally spaced days in the cluster lifetime to get an overall sense of risk-diversity of both systems.

Tiger significantly outperforms Pacemaker in providing high risk-diversity. Fig. 8 captures the risk-diversity achieved by Tiger vs Pacemaker. Since Tiger has no partitioning of disks, all disks of any make/model are viable for allocating any scheme. The minimum risk-diversity achieved by Tiger is 60% across all four clusters, that too for the narrowest scheme (6-of-9) for Backblaze (Fig. 8a) and Google Cluster 1 (Fig. 8b) clusters. Both these clusters have seven makes/models, and it is unlikely that seven out of nine chunks will be across different makes/models. As the stripe width increases, Tiger's risk-diversity also improves. Entirely step-deployed clusters, Google Cluster 2 (Fig. 8c) and Google Cluster 3 (Fig. 8d) have four and three makes/models respectively. Tiger achieves perfect risk-diversity for all possible schemes in those clusters. For Pacemaker, it is more likely that clusters where all makes/models are trickle-deployed will have a better risk-diversity because multiple makes/models can be a part of the same Rgroup so long as their AFRs are in the same range, for e.g. Backblaze (Fig. 8a). Nevertheless, even clusters with all trickle-deployed disks do not see perfect (or even good) risk-diversity since different makes/models are deployed at different times, and they go through different phases of life at different dates. Risk-diversity is poorer for Pacemaker in clusters with step-deployed makes /models as seen in Figs. 8c and 8d. This is because Rgroups and steps have a 1:1 mapping and each step only contains disks of a single make/model. The reason Pacemaker has 100% risk-diversity for 30-of-33 is because when averaging over multiple days (5 for this experiment), all makes/models on some date belonged to an Rgroup with the 30-of-33 redundancy scheme.

6.3 Tiger adapts redundancy efficiently

The efficacy of disk-adaptive redundancy performed by Tiger is evaluated using three metrics. First, we discuss the MTTDL distribution of data stored using Tiger. Subsequently, using the same four clusters used by Pacemaker we evaluate the resulting space-savings obtained by Tiger because of disk-adaptive redundancy, and finally we measure the IO overhead needed to perform necessary redundancy transitions. For fair comparison, when evaluating Tiger, we employ the same configurations (such as the IO constraints and permitted redundancy schemes) and tools (such as the AFR curve learner and the change-point detector) that are used in Pacemaker.



(a) Backblaze space-savings **(b)** Google Cluster 1 space-savings **(c)** Google Cluster 2 space-savings **(d)** Google Cluster 3 space-savings

Figure 9: Space-savings achieved by Tiger for disk-adaptive redundancy simulated on four production clusters compared to Pacemaker over conventional one-scheme-fits-all redundancy approaches. Figs. 9a–9d show that across all clusters with different maximum stripe width configurations, Tiger provides up to 5% higher average space-savings compared to Pacemaker.

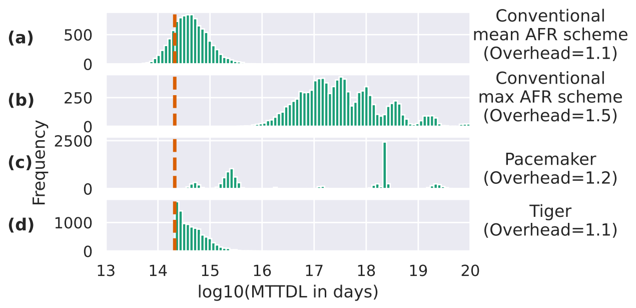


Figure 10: Comparison of MTTDL distributions for different approaches. We form 10000 random stripes for each approach using the AFRs from Google Cluster 1 (notice the different scales in the Y-axis). In a conventional system, a single scheme is chosen for all stripes based on the average AFR (*a*) or maximum AFR (*b*). (*c*) In Pacemaker, stripes must reside within an Rgroup, and the scheme depends on the Rgroup. (*d*) In Tiger, the scheme for each stripe is chosen based on the AFRs in the stripe. The dashed vertical line denotes the target MTTDL.

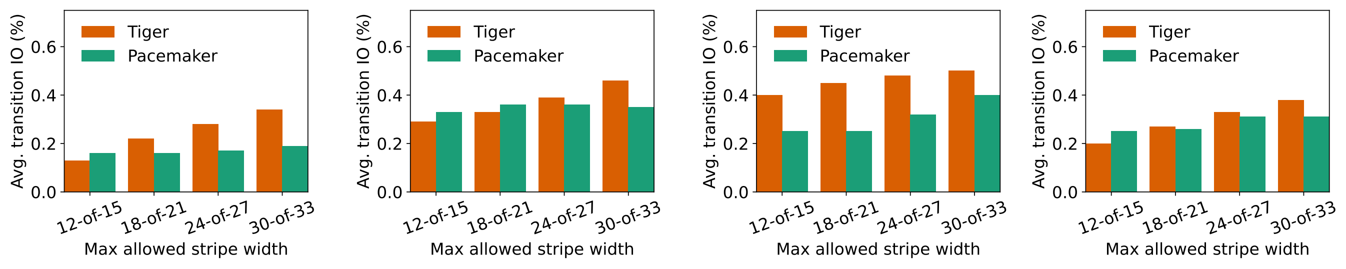
Tiger's achieves tight reliability. Storage clusters have to ensure that all data in the cluster always meets a specified target level of reliability typically specified as a MTTDL value. Tiger's target MTTDL is set as the lowest acceptable MTTDL in the system. This is calculated using the MTTDL of the most conservative homogeneous stripe possible (6-of-9) having the maximum possible AFR (16%). These settings are borrowed from Pacemaker's evaluation for a fair comparison with Tiger.

Fig. 10 shows a comparison in the distribution of stripe MTTDL with different approaches to redundancy selection for a specific day in Google Cluster 1. Fig. 10(a) shows conventional systems choosing the redundancy scheme based on the avg. AFR, which results in small storage overhead, but puts a big fraction of the stripes at risk. Fig. 10(b) shows conventional systems that choose the redundancy scheme on the basis of max AFR. Although all stripes are sufficiently protected, the storage overhead is the highest among all four alternatives. Fig. 10(c) shows Pacemaker where the different MTTDL clusters represent different Rgroups with different redundancy schemes. Pacemaker achieves good reduction in storage overhead, and keeps all stripes above the target MTTDL. In fact, some Rgroups (with higher MTTDL values) are too over-protected and denote lost opportunities for space-savings. Finally, Fig. 10(d) shows Tiger's MTTDL distribution. Despite all its eclectic stripes being above the MTTDL threshold, Tiger has least storage overhead.

Tiger achieves attractive space-savings. Akin to Pacemaker, by dynamically tailoring redundancy to disk AFRs, Tiger's eclectic stripes can use more space-efficient redundancy schemes to meet the required MTTDL target. Fig. 9 shows that Tiger achieves equal or better average space-savings compared to Pacemaker in all four clusters. For Google Clusters 1, 2 and 3 (Figs. 9b, 9c, 9d), the highly cost-efficient redundancy transitions of Pacemaker allows a large step-deployed make/model to spend more time in lower redundancy. This boosts Pacemaker's overall space-savings for these clusters and prevents Tiger from surpassing it easily.

In the Backblaze cluster (Figs. 9a), the reason for Tiger achieving better space-savings is because eclectic stripes allow high AFR disks to be mixed with low AFR disks and yet use an optimized redundancy scheme. In Pacemaker, high AFR disks cannot be mixed with other disks, resulting in lower space-savings. In the Backblaze cluster, all the seven makes/models are trickle-deployed. This results in a non-trivial fraction of disks constantly being in high-AFR regimes of infancy or wearout. While Pacemaker is forced to use the default, most conservative redundancy scheme on these disks, Tiger can use these disks for more space-efficient redundancy schemes by combining them with other, more robust disks. As a result, Tiger is able to achieve up to 5% higher space-savings compared to Pacemaker.

Tiger has very low IO overhead. Fig. 11 shows the IO overhead comparison between Pacemaker and Tiger. Although both systems are capped at 5% and in general require very low IO (compared to background tasks such as scrubbing that requires $\approx 7\%$ [5]), our evaluation shows that Tiger can achieve all its benefits with an average IO bandwidth required for redundancy transitions of at most 0.5%. In an absolute sense, Tiger's low IO overhead is mainly attributed to Tiger's efficient redundancy transitions for an AFR rise (detailed in §5.2), where Tiger moves the potentially risky chunk from



(a) Backblaze IO overhead **(b)** Google Cluster 1 IO overhead **(c)** Google Cluster 2 IO overhead **(d)** Google Cluster 3 IO overhead

Figure 11: IO overhead of redundancy scheme transitions of Tiger versus Pacemaker. In most configurations, Tiger has a higher IO overhead compared to Pacemaker due to Pacemaker leveraging its IO-efficient transitioning mechanisms. Despite being higher, the average IO overhead of Tiger is still at most 0.5% of the overall cluster's IO bandwidth; much lower than existing background tasks such as scrubbing, that require approximately 7% IO bandwidth [5]

an unsafe disk to a safe disk rather than re-encoding it or reallocating it; both having a significantly higher IO cost.

Compared to Pacemaker, Tiger still incurs slightly higher IO overhead. This is due to Tiger's mechanism of coalescing space-inefficient (high-redundancy) eclectic stripes into new space-efficient (low-redundancy) eclectic stripes in response to AFR reduction by moving all chunks. It leads to more data movement compared to moving just the chunks of the high-AFR disks (as is the case when AFR rises). This is a conscious design choice made in Tiger in order to maximize space-savings for non-urgent redundancy transitions at the expense of a minor increase in the IO overhead. Moreover, Pacemaker's IO-efficient redundancy transitioning mechanisms (that are more suitable for its Rgroup-based design) further help in reducing its IO overhead.

Tiger does not experience urgent IO bursts. In order to understand the burstiness of the IO that can be experienced by Tiger compared to Pacemaker, we artificially increase the AFR of a make/model and measure the resulting transition IO load for maintaining data reliability. Fig. 12 shows the comparison of IO loads experienced by Pacemaker vs Tiger for three instances of increasing AFR of a single step-deployed make/model. Performed on three different dates in two Google clusters (Cluster 1 and Cluster 2), we observe that Pacemaker needs orders of magnitude higher IO bandwidth than Tiger to achieve the required transitions. In fact for Google Cluster 2, in both instances none of Tiger's stripes needed transitioning despite observing a 1% rise in AFR.

We explain Pacemaker's high IO requirement with an example. Suppose a 20TB disk, which can perform 100MB/s needs to transition away from using a 30-of-33 scheme. Despite using Pacemaker's optimized Type 2 transitions[*], simply reading the data to recalculate new parities would require 196% of the disk's possible IO bandwidth in a day (assuming 90% fullness to match Pacemaker's setup). In a step-deployed make/model all disks of an Rgroup transition together. In order to spread out the resulting IO burst over time, Pacemaker relies on predicting the AFR rise well in advance. To maintain a 5% IO cap, Pacemaker would need to know the AFR rise at least 40 days in advance. Long-term AFR predictions are both non-robust and non-trivial.

[*]In Type 2 transitions, Pacemaker re-encodes data from one scheme to another without re-writing any data. It simply recalculates new parities, writes them and deletes the old ones.

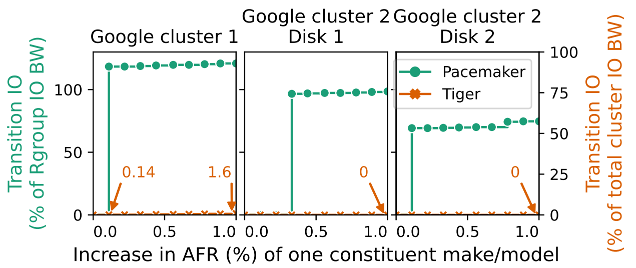


Figure 12: IO cost of redundancy transitions associated with the increase of AFR for one constituent make/model. IO cost is measured as a percentage of the total IO bandwidth of the Rgroup for Pacemaker, whereas it is the total cluster IO bandwidth for Tiger. It is calculated by scaling up a simulation of 1000 random stripes in each system and measuring the number of stripes that become unsafe after the given increase in AFR.

In contrast, Tiger for the same transition does not suffer from any IO bursts. Firstly, because of eclectic stripes, even if the disk AFR increases, only a limited fraction of data stored on it will need a redundancy transition, since other stripes might be residing on more robust disks and might continue to meet the target MTTDL. Secondly, other disks over which the eclectic stripes needing an increase in redundancy are spread need not (and probably will not) belong to the same make/model/batch. Therefore, they will not require a simultaneous increase in redundancy and can assist in transitioning data from the affected stripes. Thus disks in Tiger are spared from any sudden IO bursts.

6.4 Challenging situations for Tiger

There are certain situations that create fundamental challenges for Tiger and other disk-adaptive redundancy systems.

Sudden rise in AFRs mimicking bulk failures. Although

Fig. 12 shows that Tiger is robust to AFR rises in any make/model in a cluster, there could be bulk failure scenarios where large fraction of the disks in the cluster fail together. On such occasions, any system (including Tiger) that depends on redundancy will suffer from potential data loss unless the system includes cross-cluster redundancy.

A cluster with a single step-deployed make/model. Suppose a cluster had only one make/model, deployed in a step-deployed manner (note: we have not come across such an example for the large clusters targeted): there would be no diversity to exploit and all disks of the cluster would undergo redundancy transitions together. Not only would this produce bursty IO, but also will potentially result in a capacity crunch (when increasing redundancy). Such clusters would either need to keep some space unutilized to account for the bulk redundancy-increasing transitions, or will need to make provisions to add more disks to the cluster before the redundancy-increasing transitions are issued.

7 Additional Related Work

The closest related works, HeART and Pacemaker, are discussed in §2 together with other background. Additional related works can be divided into works that study the reliability of disks and distributed storage, and systems that manage multiple EC schemes and transitions between them. One essential part of disk-adaptive redundancy is the monitoring of disk AFRs, which are used by Tiger to assess the reliability of stripes. Many works have studied the behavior of disk AFRs and their impact on distributed storage reliability [5, 8, 18, 22, 26, 34, 35, 41–44]. Also, multiple works have studied the prediction of disk AFRs based on different features [1, 17, 27, 32, 45, 49, 59].

Many existing distributed storage systems allow for multiple EC schemes to coexist in the same cluster [11, 14]. There are systems that propose choosing different EC schemes for different data [46, 55]. The problem of transitioning data from one EC scheme to another has been widely studied in the Coding Theory literature, with many works studying its cost, as well as proposing special code designs that reduce the cost of transitions [20, 28–31, 36, 38, 39, 53–55, 57, 60]. Such designs could be used with Tiger, though our evaluations indicate that transition IO is not a significant problem.

8 Conclusion

Tiger enables disk-adaptive redundancy without the placement restrictions and associated problems that plague prior designs. Tiger's eclectic stripes tailor redundancy to whichever disks are chosen for each stripe. Our evaluations indicate that it reduces risk in two major ways: by increasing disk-type diversity in stripes and by reducing burstiness of transition IO urgency. Taken together, Tiger makes disk-adaptive redundancy practical for adoption in real storage clusters.

9 Acknowledgements

We thank our shepherd Gala Yadgar and the anonymous reviewers for their valuable feedback and suggestions. We extend special thanks to Larry Greenfield and numerous other researchers and engineers at Google. This research is supported in part by NSF grants CNS1956271 and CNS1901410. We also thank the members and companies of the PDL Consortium (Amazon, Google, HPE, Hitachi, IBM, Intel, Meta, Microsoft, NetApp, Oracle, Pure Storage, Salesforce, Samsung, Seagate, Two Sigma, Western Digital) and VMware for their interest, insights, feedback, and support.

A Derivation of approximation of MTTDL of eclectic stripes

In order to approximate the MTTDL of an eclectic stripe, we will assume that the stripe can be repaired in the data loss state and we will approximate the MTTDL as the mean time between visits to the data loss state. In particular, we will analyze the stripe as an alternating renewal process. Let A_s be the stripe availability (i.e., the fraction of the time that the stripe is not in the data loss state), μ_s be the repair rate in the data loss state, and λ_s the stripe data loss rate. As described above, the MTTDL is approximately λ_s^{-1}. For an alternating renewal process, we have that:

$$A_s = \frac{\mu_s}{\mu_s + \lambda_s} \iff \frac{1}{\lambda_s} = \frac{A_s}{\mu_s(1 - A_s)} \tag{2}$$

The repair rate in the data loss state is simply the number of failed disks in that state:

$$\mu_s = (n - k + 1)\mu. \tag{3}$$

We assume that each disk in the stripe fails independently from the rest, and that it is repaired with rate μ if it fails. Then, in steady state, disk i is available with probability:

$$A_i = \frac{\mu}{\mu + \lambda_i}. \tag{4}$$

Let $P(j)$ be the probability that we find the stripe in a state where exactly j disks are available in the stripe. Since there are no states with more than $n - k + 1$ failed disks, we have that:

$$P(j) = \frac{Q(j)}{Q(k-1) + \cdots + Q(n)}, \text{for } k - 1 \leq j \leq n, \tag{5}$$

where $Q(j)$ is the probability that exactly j disks are available. Since disks are independent, $Q(j)$ is equal to a Poisson-binomial distribution, with probabilities $(A_i)_{i=1}^{n}$. Given this, the availability of stripe is given by:

$$A_s = P(k) + \cdots + P(n). \tag{6}$$

Thus, we have:

$$\frac{1}{\lambda_s} = \frac{Q(k) + \cdots + Q(n)}{\mu(n-k+1)Q(k-1)} \approx \frac{1}{\mu(n-k+1)Q(k-1)}. \tag{7}$$

Where the approximation comes from the fact that $Q(n) \approx 1$ because $\mu \gg \max_i \lambda_i$ and thus all A_i are close to 1.

In summary, we have that:

$$\mathrm{MTTDL} \approx \frac{1}{\mu(n-k+1)Q(k-1)}. \tag{8}$$

References

[1] Preethi Anantharaman, Mu Qiao, and Divyesh Jadav. Large Scale Predictive Analytics for Hard Disk Remaining Useful Life Estimation. In *IEEE International Conference on Big Data*, 2018.

[2] John E Angus. On computing MTBF for a k-out-of-n: G repairable system. *IEEE Transactions on Reliability*, 37(3):312–313, 1988.

[3] Backblaze. Disk Reliability Dataset. https://www.backblaze.com/b2/hard-drive-test-data.html, 2013-2018.

[4] Backblaze. Erasure coding used by Backblaze. https://www.backblaze.com/blog/reed-solomon/, 2013-2018.

[5] Lakshmi N Bairavasundaram, Garth R Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. In *ACM SIGMETRICS Performance Evaluation Review*, 2007.

[6] Werner Ehm. Binomial approximation to the Poisson binomial distribution. *Statistics & Probability Letters*, 11(1):7–16, 1991.

[7] Nosayba El-Sayed, Ioan A Stefanovici, George Amvrosiadis, Andy A Hwang, and Bianca Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 163–174, 2012.

[8] Jon Elerath. Hard-disk drives: The good, the bad, and the ugly. *Communication of ACM*, 2009.

[9] Jon G Elerath. AFR: problems of definition, calculation and measurement in a commercial environment. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2000.

[10] Jon G Elerath. Specifying reliability in the disk drive industry: No more MTBF's. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2000.

[11] Erasure code Ceph Documentation. https://docs.ceph.com/docs/master/rados/operations/erasure-code/, (accessed September 25, 2019).

[12] Manuel Fernández and Stuart Williams. Closed-form expression for the Poisson-binomial probability density function. *IEEE Transactions on Aerospace and Electronic Systems*, 46(2):803–817, 2010.

[13] Daniel Ford, François Labelle, Florentina I Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in Globally Distributed Storage Systems. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[14] Apache Software Foundation. HDFS Erasure Coding. https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html, 2017 (accessed November 5, 2020).

[15] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, 2003.

[16] Garth Alan Gibson. *Redundant disk arrays: Reliable, parallel secondary storage*. PhD thesis, University of California, Berkeley, 1991.

[17] Greg Hamerly, Charles Elkan, et al. Bayesian approaches to failure prediction for disk drives. In *International Conference on Machine Learning (ICML)*, 2001.

[18] Eric Heien, Derrick Kondo, Ana Gainaru, Dan LaPine, Bill Kramer, and Franck Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *ACM / IEEE High Performance Computing Networking, Storage and Analysis (SC)*, 2011.

[19] Yili Hong. On computing the distribution function for the Poisson binomial distribution. *Computational Statistics & Data Analysis*, 59:41–51, 2013.

[20] Yuchong Hu, Xiaoyang Zhang, Patrick P. C. Lee, and Pan Zhou. Generalized optimal storage scaling via network coding. In *IEEE International Symposium on Information Theory (ISIT)*, 2018.

[21] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin, et al. Erasure Coding in Windows Azure Storage. In *USENIX Annual Technical Conference (ATC)*, 2012.

[22] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *ACM Transactions on Storage (TOS)*, 2008.

[23] Saurabh Kadekodi. *DISK-ADAPTIVE REDUNDANCY: tailoring data redundancy to disk-reliability heterogeneity in cluster storage systems*. PhD thesis, Carnegie Mellon University, 2020.

[24] Saurabh Kadekodi, Francisco Maturana, Suhas Jayaram Subramanya, Juncheng Yang, KV Rashmi, and Gregory R Ganger. PACEMAKER: Avoiding heart attacks in storage clusters with disk-adaptive redundancy. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[25] Saurabh Kadekodi, K V Rashmi, and Gregory R Ganger. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity. In *USENIX File and Storage Technologies (FAST)*, 2019.

[26] Ao Ma, Rachel Traylor, Fred Douglis, Mark Chamness, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. RAIDShield: characterizing, monitoring, and proactively protecting against disk failures. *ACM Transactions on Storage (TOS)*, 2015.

[27] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *USENIX Annual Technical Conference (ATC)*, 2017.

[28] Francisco Maturana, V. S. Chaitanya Mukka, and K. V. Rashmi. Access-optimal linear MDS convertible codes for all parameters. In *IEEE International Symposium on Information Theory (ISIT)*, 2020.

[29] Francisco Maturana and K. V. Rashmi. Bandwidth cost of code conversions in distributed storage: Fundamental limits and optimal constructions. *arXiv preprint arXiv:2008.12707*, 2020.

[30] Francisco Maturana and K. V. Rashmi. Convertible codes: new class of codes for efficient conversion of coded data in distributed storage. In *Innovations in Theoretical Computer Science Conference, (ITCS)*, 2020.

[31] Sara Mousavi, Tianli Zhou, and Chao Tian. Delayed parity generation in MDS storage codes. In *IEEE International Symposium on Information Theory (ISIT)*, 2018.

[32] Joseph F Murray, Gordon F Hughes, and Kenneth Kreutz-Delgado. Hard drive failure prediction using non-parametric statistical methods. In *Springer Artificial Neural Networks and Neural Information Processing (ICANN/CONIP*, 2003.

[33] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Mike Shuey, Richard Wareing, Monika Gangapuram, Guanglei Cao, et al. Facebook's tectonic filesystem: Efficiency from exascale. In *19th {USENIX} Conference on File and Storage Technologies ({FAST} 21)*, pages 217–231, 2021.

[34] David A Patterson, Garth Gibson, and Randy H Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM International Conference on Management of Data (SIGMOD)*, 1988.

[35] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure Trends in a Large Disk Drive Population. In *USENIX File and Storage Technologies (FAST)*, 2007.

[36] Brijesh Kumar Rai, Vommi Dhoorjati, Lokesh Saini, and Amit K. Jha. On adaptive distributed storage systems. In *IEEE International Symposium on Information Theory (ISIT)*, 2015.

[37] K V Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers. *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2014.

[38] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar. Enabling node repair in any erasure code for distributed storage. In *IEEE International Symposium on Information Theory (ISIT)*, 2011.

[39] KV Rashmi, Nihar B Shah, and Kannan Ramchandran. A piggybacking design framework for read-and download-efficient distributed storage codes. *IEEE Transactions on Information Theory*, 2017.

[40] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: Novel erasure codes for big data. In *International Conference on Very Large Data Bases (VLDB)*, 2013.

[41] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. Understanding latent sector errors and how to protect against them. *ACM Transactions on Storage (TOS)*, 2010.

[42] Bianca Schroeder and Garth A Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *USENIX File and Storage Technologies (FAST)*, 2007.

[43] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*. IOP Publishing, 2007.

[44] Sandeep Shah and Jon G Elerath. Disk drive vintage and its effect on reliability. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2004.

[45] Brian D Strom, SungChang Lee, George W Tyndall, and Andrei Khurshudov. Hard disk drive reliability modeling and failure prediction. *IEEE Transactions on Magnetics*, 2007.

[46] Eno Thereska, Michael Abd-El-Malek, Jay J Wylie, Dushyanth Narayanan, and Gregory R Ganger. Informed data distribution selection in a self-predicting storage system. In *IEEE International Conference on Autonomic Computing (ICAC)*, 2006.

[47] Charles Truong, Laurent Oudre, and Nicolas Vayatis. A review of change point detection methods. In *arXiv:1801.00718v1 [cs.CE]*, 2018.

[48] Charles Truong, Laurent Oudre, and Nicolas Vayatis. ruptures: change point detection in python. In *arXiv:1801.00826v1 [cs.CE]*, 2018.

[49] Yu Wang, Eden WM Ma, Tommy WS Chow, and Kwok-Leung Tsui. A two-step parametric method for failure prediction in hard disk drives. *IEEE Transactions on industrial informatics*, 2014.

[50] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Springer International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[51] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[52] Wolfram. Wolfram Mathematica. https://www.wolfram.com/mathematica.

[53] Si Wu, Zhirong Shen, and Patrick P. C. Lee. Enabling I/O-efficient redundancy transitioning in erasure-coded KV stores via elastic Reed-Solomon codes. In *39th Symposium on Reliable Distributed Systems, SRDS 2020, Shanghai, China, September 21-24, 2020*, 2020.

[54] Si Wu, Yinlong Xu, Yongkun Li, and Zhijia Yang. I/O-efficient scaling schemes for distributed storage systems with CRS codes. *IEEE Transactions on Parallel and Distributed Systems*, 2016.

[55] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A tale of two erasure codes in HDFS. In *USENIX File and Storage Technologies (FAST)*, 2015.

[56] Jimmy Yang and Feng-Bin Sun. A comprehensive review of hard-disk drive reliability. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 1999.

[57] Xiaoyang Zhang, Yuchong Hu, Patrick P. C. Lee, and Pan Zhou. Toward optimal storage scaling via network coding: from theory to practice. In *IEEE Conference on Computer Communications, (INFOCOM)*, 2018.

[58] Zhe Zhang, Amey Deshpande, Xiaosong Ma, Eno Thereska, and Dushyanth Narayanan. Does erasure coding have a role to play in my data center. *Microsoft research MSR-TR-2010*, 52, 2010.

[59] Ying Zhao, Xiang Liu, Siqing Gan, and Weimin Zheng. Predicting disk failures with HMM-and HSMM-based approaches. In *Springer Industrial Conference on Data Mining (ICDM)*, 2010.

[60] Weimin Zheng and Guangyan Zhang. Fastscale: accelerate RAID scaling by minimizing data migration. In *USENIX File and Storage Technologies (FAST)*, 2011.