

November 5, 2020

DRAFT

**DISK-ADAPTIVE REDUNDANCY:
tailoring data redundancy to
disk-reliability-heterogeneity in cluster
storage systems**

Saurabh Kadekodi

December 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Gregory R. Ganger, Co-chair

K. V. Rashmi, Co-chair

Garth A. Gibson (CMU and Vector Institute)

Arif Merchant (Google Inc.)

Remzi Arpacı-Dusseau (University of Wisconsin-Madison)

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

Large-scale cluster storage systems typically consist of a heterogeneous mix of storage devices with significantly varying failure rates. Despite having reliability differences of over an order of magnitude in the same storage cluster for the same storage tier, redundancy settings are generally configured in a one-scheme-for-all fashion. This dissertation paves the way for exploiting disk reliability heterogeneity to tailor redundancy settings to different disk groups for cost-effective, arguably safer redundancy.

The first contribution is the heterogeneity-aware redundancy tuner (HeART), an online tuning tool that guides selection of different redundancy settings for long-term data reliability, based on observed reliability properties of each disk group. By processing disk failure data over time, HeART identifies the boundaries and steady-state failure rate for each deployed disk group by make/model. Using this information, HeART suggests the most space-efficient redundancy option allowed that will achieve the specified target data reliability. HeART is evaluated via simulation on longitudinal disk failure logs from a large production cluster with over 100,000 disks. Guided by HeART, the cluster could meet target data reliability levels with much fewer disks than one-scheme-for-all approaches: 11–16% fewer compared to erasure codes like 10-of-14 or 6-of-9 and up to 33% fewer compared to 3-way replication.

While HeART promised substantial space-savings, it is rendered unusable in production settings of real-world clusters, because the IO load of transitions between redundancy schemes overwhelms the storage infrastructure (termed *transition overload*). The follow-on work conducts an in-depth analysis of millions of disks from production systems at Google, NetApp, and Backblaze to expose and understand transition overload as a roadblock to disk-adaptive redundancy. Building on the insights drawn from this analysis, Pacemaker is the second contribution of this dissertation; a low-overhead disk-adaptive redundancy orchestrator that realizes HeART’s dream in practice. Pacemaker mitigates transition overload by (1) proactively organizing data layouts to make future transitions efficient, and (2) initiating transitions proactively in a manner that avoids urgency while not compromising on space-savings. Evaluation of Pacemaker with traces from four large (110K–450K disks) production clusters show that the transition IO requirement decreases to never needing more than 5% cluster IO bandwidth (only 0.2–0.4% on average). Pacemaker achieves this while providing overall space-savings of 14–20% (compared to using a static 6-of-9 scheme) and never leaving data under-protected.

The final contribution of this dissertation is the design and implementation of disk-adaptive redundancy techniques from Pacemaker in the widely used Hadoop Distributed File System (HDFS). This prototype re-purposes HDFS’s existing architectural components for disk-adaptive redundancy, and successfully leverages the robustness and maturity of the existing code. Moreover, the components that are re-purposed are fundamental to any distributed storage system’s architecture, and thus, this prototype also serves as a guideline for other systems to support disk-adaptive redundancy.

Contents

1	Introduction	1
1.1	Goals	3
1.2	Contributions	3
2	Background and motivation	5
2.1	Overview of cluster storage systems	5
2.1.1	Hard disk drives are the primary storage devices	5
2.1.2	Disk failures are common	6
2.1.3	Storage clusters have heterogeneous disk populations	6
2.2	Data reliability is achieved using data redundancy	6
2.2.1	Replication	6
2.2.2	Erasure coding	7
2.2.3	Metrics of data reliability	7
2.3	Characterizing disk reliability over lifetime	8
2.3.1	The disk bathtub curve	8
2.4	Scheme selection problem for scalable storage	8
3	A case for disk-adaptive redundancy	10
3.1	The Heterogeneity-Aware Redundancy Tuner (HeART)	10
3.2	Having HeART can make you rich	12
3.2.1	The Backblaze dataset	12
3.2.2	Disk group formation and varying AFRs	13
3.2.3	Space savings from heterogeneous AFRs	16
3.3	The ways of the HeART	19
3.3.1	Challenges	19
3.3.2	HeART architecture	20
3.3.3	Online anomaly detection	20
3.3.4	Online change point detection	21
3.4	Measuring HeART	24
3.4.1	Implementation of the components	24
3.4.2	Evaluation on the Backblaze dataset	24
3.4.3	Sensitivity analysis	28
3.5	HeART-less alternatives (related work)	30

4 Disk reliability analysis in production environments	32
4.1 Longitudinal disk reliability datasets	32
4.2 Observations and insights	32
4.2.1 Modern disaggregated storage clusters have over 100K HDDs	32
4.2.2 Useful life AFRs are wildly heterogeneous	33
4.2.3 Disk have two distinct deployment patterns	34
4.2.4 AFRs rise gradually over time with no clear wearout	35
4.2.5 Useful life could have multiple phases	35
4.2.6 Infancy often short-lived	36
5 Combating transition overload in disk-adaptive redundancy systems	38
5.1 Pacemaker: avoiding HeART attacks	38
5.2 Simple re-encoding cannot reduce transition overload	40
5.3 Design goals	41
5.3.1 Disk lifecycle under Pacemaker	41
5.3.2 Key decisions	42
5.3.3 Constraints	42
5.3.4 Designing IO constraints on transitions	43
5.3.5 Design goals	43
5.4 Design of Pacemaker	44
5.4.1 Proactive-transition-initiator	45
5.4.2 Rgroup-planner	46
5.4.3 Transition-executor	48
5.5 Evaluation	50
5.5.1 Pacemaker on Google Cluster1 in-depth	51
5.5.2 Pacemaker on the other three clusters	53
5.5.3 Sensitivity analyses and ablation studies	58
5.6 Related work	59
6 Implementing disk-adaptive redundancy in HDFS	61
6.1 Background on HDFS architecture	61
6.1.1 Realizing Dgroups in HDFS	61
6.1.2 Realizing Rgroups in HDFS	62
6.2 Incorporating transition-executor in HDFS	62
6.2.1 Type 1 (transitioning by emptying disks)	62
6.2.2 Type 2 (transitioning by bulk parity re-calculation)	62
6.2.3 Purging Rgroups	63
6.3 Designing IO constraints in Pacemaker	63
6.3.1 Handling the average-IO constraint	63
6.3.2 Handling the peak-IO constraint	64
6.4 Salient features of this architecture	64
6.4.1 Right level of control and view of the system	64
6.4.2 Minimizing changes to the HDFS architecture and maximizing re-purposing of existing HDFS mechanisms	64

6.4.3	Flexible Rgroup-specific customizations are possible	65
6.4.4	Cost of maintaining multiple DatanodeManagers is small	65
6.4.5	File access interface remains unchanged	65
6.5	Evaluating HDFS + Pacemaker	65
7	Discussion and future direction	68
7.1	Disk-adaptive redundancy is a self-correcting mechanism for poor disks	68
7.2	Striping data across different Rgroups	68
7.3	Incorporating availability in disk-adaptive redundancy	70
7.4	Adaptive redundancy is not restricted to HDDs	70
8	Conclusions	71
A	Failure rate estimation details	73
Bibliography		75

List of Figures

I.1	(a) illustrates the assumption all disks fail similarly. Two 4-of-6 stripes are shown. (b) shows the make-up of a modern storage cluster. Each color represents a make/model with a distinct AFR, so the two 4-of-6 stripes may have very different reliability.	2
I.2	AFR heterogeneity observed in over 60 makes/models with a total disk population of over 5.3 million HDDs deployed across Google, NetApp and Backblaze.	3
2.1	The disk hazard curve (also known as the bathtub curve) showing disk reliability over its lifetime.	9
3.1	Annualized failure rate (AFR) for the six disk groups that make up >90% of the 100,000+ HDDs used for the Backblaze backup service [10]. Details of each disk group are given in Section 3.2.	11
3.2	AFR comparison between all 4TB disks grouped together and disk groups broken down by make/model. The AFR differences in make/model-based grouping enables HeART to perform finer-grained specialization leading to higher benefits.	13
3.3	The canonical bathtub curve used to represent disk failure characteristics.	14
3.4	An abstract timeline of a disk group from deployment to failure or decommissioning, with the three distinct periods. The notations below the timeline (r_{de}) and (r_{DG}) denote the redundancy scheme employed during the respective stage.	14
3.5	Cumulative raw AFR versus age (in days) for all six disk groups being analyzed.	15
3.6	Schematic diagram of HeART. Components include an anomaly detector, an online change point detector, and a redundancy tuner.	19
3.7	Raw and HeART-curated AFR curves for the H-4B disk group. Five spikes in AFR (points A-E), which correspond to four (anomalous) bulk failure events, are automatically filtered out by HeART.	21
3.8	Total number of disks and number of disk failures by date for H-4B disks. The step-wise jumps in the black curve represent incremental deployments. The largest red spike represents the disks that failed on July 23, 2017, causing anomalies A and B in Fig. 3.7.	22
3.9	HeART in action on all disk groups, showing successful identification of infant mortality, useful life and wearout periods as well as automatic removal of anomalies	23

3.10 AFR of the S-4 disk group using a sliding window of 30 days. The determined useful life AFR value by HeART is conservative enough to subsume even the 30-day AFR values which vary more than the cumulative AFRs.	25
3.11 Overall space reduction achieved by HeART on the Backblaze dataset over the complete lifetime of every disk group, for erasure codes as the default scheme. For a maximum code dimension of up to $2 \times r_{def}$, we observe between 6–7.5% space reduction and for a maximum code dimension of up to $4 \times r_{def}$, we observe between 10–12% space reduction, translating to actual space savings of 40–80 PBs.	28
3.12 The effect of varying T_{flat} (AFR flatness threshold) on the H-4B disk group's AFR curve. Larger T_{flat} implies a higher useful life AFR along with a larger useful life period. The default value for T_{flat} in HeART is 0.5.	29
4.1 AFR spread for over 60 makes/models from NetApp, Google and Backblaze binned by the age of the oldest disk. Each box corresponds to a unique make/model, and at least 10000 disks of each make/model were observed (outlier AFR values omitted).	33
4.2 Backblaze cluster showing trickle deployments.	34
4.3 Google Cluster2 showing step deployments.	34
4.4 Distribution of AFR calculated over consecutive non-overlapping six-month periods for NetApp disks, showing the gradual rise of AFR with age (outliers omitted).	35
4.5 Approximation of useful life length for NetApp disks for 1-5 consecutive phases of useful life and three different tolerance levels. Each useful-life phase is approximated by an uninterrupted sequence of days during which AFR remains within the specified (atop graph) ratio between maximum and minimum AFRs. Boxes represent the distribution of the combined length of all phases for different makes/models. The box labeled “age” represents the distribution of the age of the oldest disk for different makes/models, which is an upper bound the length of useful life.	36
5.1 Fraction of total cluster IO bandwidth needed to use disk-adaptive redundancy for a Google storage cluster’s first three years. The state-of-the-art proposal [41] shown in (a) would require up to 100% of the cluster bandwidth for extended periods, whereas Pacemaker shown in (b) always fits its IO under a cap (5%). The light gray region shows the disk count (right Y-axis) over time.	39
5.2 Pacemaker architecture.	44
5.3 Detailed IO analysis and space savings achieved by Pacemaker-enabled adaptive redundancy on Google Cluster1.	52
5.4 Google Cluster2 transition IO, space-savings and individual Dgroup AFR curves.	54
5.5 Google Cluster3 transition IO, space-savings and individual Dgroup AFR curves.	55
5.6 Backblaze cluster transition IO, space-savings and individual Dgroup AFR curves.	56
5.7 Pacemaker’s sensitivity to the peak-IO constraint.	57

5.8	Multiple useful-life phases	58
5.9	Transition type distribution	58
6.1	Pacemaker-enhanced HDFS architecture	62
6.2	DFS-perf reported throughput for baseline, with one Datanode failure and one Rgroup transition	66
6.3	DFS-perf reported latency for baseline, with one Datanode failure and one Rgroup transition	66
7.1	Continuous Time Markov Chain for calculating MTTDL of 3-replication with cross-disk-group striping	69

List of Tables

3.1	The disk groups identified from the Backblaze dataset for reliability heterogeneity analysis. The disk group shorthand above is used to represent the respective makes/models throughout the chapter.	13
3.2	A sample of the estimated savings achievable through HeART. The space reductions obtained on H-4A disks by using redundancy schemes with lower storage overhead while meeting the reliability target set by applying the default redundancy scheme (r_{def}) on S-4 disks.	18
3.3	Disk space saved by HeART by tuning the redundancy in the useful life of a disk group according to the observed disk group-specific AFRs. The units for MTTDLs is years. The cost savings are calculated for 3 default schemes: (14, 10) on AFR 4.01% disks, (9, 6) on AFR 4.01% disks and 3-replication (i.e. (3, 1)) on AFR 4.01% disks. Thus, the target reliability is the MTTDL of the respective default redundancy schemes using a 4.01% AFR (the r_{def} table header). The max dimension of the scheme permitted during useful life for each disk group has at most twice the dimension of default redundancy scheme, i.e. 20 data chunks for (14, 10), 12 data chunks for (9, 6) and 2 data chunks for 3-replication.	26
5.1	Definitions of Pacemaker's terms.	42

Chapter 1

Introduction

Distributed storage systems within large data centers are primary components of cloud, Internet service, and data analytics infrastructures. With the explosion of machine learning and rapid rise in big data analytics, data science, edge computing and real-time systems, these distributed storage systems are continuously pressured for storage space. For architects of such large-scale storage systems, a petabyte (PB) of free space translates to a critically low free space warning. In fact analysts estimate that 175 zettabytes ($1 \text{ ZB} = 1024 \text{ PB}$) of data will be generated annually by 2025, most of which will be stored in data centers [73]. Although new data and hot/warm data is now often stored on Flash SSDs, cost considerations will lead to the majority of data continuing to be stored on mechanical disks (HDDs) for the foreseeable future [4, 15, 73].

In order to keep up with data growth, modern storage architectures often scale to huge capacities by combining up to hundreds of thousands of storage devices into a single storage system [30, 75, 85]. At such scales device failure are common, and yet, data has to remain reliable despite these failures. Data redundancy is the most common method to protect against data loss in the face of failures [27, 30, 39, 61, 75]. While replication is often used to improve performance for hot data, erasure coding has become common for more cost-effectively storing most data [25, 29, 39].

A primary goal, when selecting a data redundancy scheme (e.g., 3-way replication vs 2-way replication, or 6-of-9 erasure code vs. 6-of-8 erasure code), is ensuring that it satisfies the data reliability goal. Well-established and often-used equations [56, 80] exist for computing mean-time-to-data-loss (MTTDL) for a redundancy scheme from the per-device annualized failure rate (AFR; the commonly used alternative for mean-time-to-failure (MTTF)¹), the number of devices, and the mean-time-to-repair (MTTR). Redundancy settings are generally chosen as if all of the devices have the same AFR. This is largely true for RAID arrays (tens of disks), or even sizable NFS/CIFS file servers, or smaller cluster storage systems (hundreds of disks), which often contain a uniform set of disks deployed all-at-once and removed before wearout. Under this setting, any two stripes (see Fig. □a) experience the same level of data reliability.

Unfortunately, the same reliability math doesn't hold for modern large-scale storage systems because all devices don't have the same AFR. With 100K+ disks deployed over time and

¹AFR is the expected number of device failures in a typical year. AFR is a rate quantity, inversely proportional to MTTF.

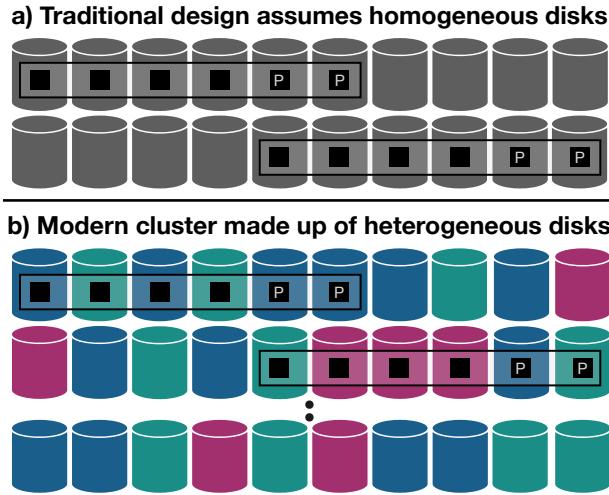


Figure 1.1: (a) illustrates the assumption all disks fail similarly. Two 4-of-6 stripes are shown. (b) shows the make-up of a modern storage cluster. Each color represents a make/model with a distinct AFR, so the two 4-of-6 stripes may have very different reliability.

per-acquisition optimization of which disks to buy (e.g., lowest-cost-per-byte at each acquisition time), there is a lot of heterogeneity. That is, the constituent devices in such systems are of multiple makes/models and multiple device ages, and AFR values vary as a function of both attributes [41, 47, 57]. This leads to different MTTDLs which means different levels of reliability.

The MTTDL equations can still be used to guide decisions, as long as a sufficiently high AFR value is used. For example, if the highest AFR value possible for any deployed make/model at any age allowed (before decommissioning) is used, the computed MTTDL will be a lower bound. Unfortunately, the range of possible AFR values in a deployed system is quite large [41, 47, 57, 70]. For example, as shown in Fig. 4.1, we observe the range to be over an order of magnitude in both the NetApp, Google and Backblaze logs we are using for this dissertation, from double-digit AFRs to low single digit AFRs. Since the overall average is much closer to the lower end of the AFR range, the highest value is a very conservative over-estimate for most of the devices. The resulting MTTDLs are thus loose lower bounds, leading decision-makers to use excessive redundancy.

Nonetheless, we are told by admins of large storage deployments that such approaches, and other ad hoc conservative over-protection, are the norm. Such approaches are problematic in two significant ways. First, they result in much higher cost overheads (in the form of more disks with the associated dollar, environmental consequences) than might otherwise be necessary. Second, and more insidious, they are still risky, because it is unclear (until too late) how conservative one needs to be. For example, suppose that the redundancy scheme shown in Fig. 4.1b is designed using the AFR of green disks. If the AFR of purple disks is higher than the green disks, and the AFR of the blue disks is lower than the green disks. Then, the data in the top stripe is over-protected, whereas simultaneously, the data in the bottom stripe is under-protected and at a higher risk of data loss.

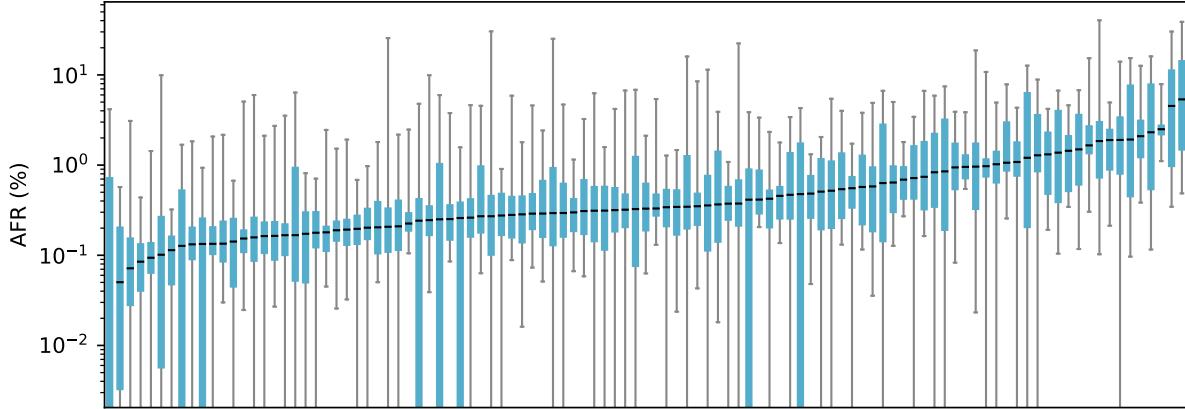


Figure 1.2: AFR heterogeneity observed in over 60 makes/models with a total disk population of over 5.3 million HDDs deployed across Google, NetApp and Backblaze.

1.1 Goals

The overarching goal of this thesis is to enable highly-reliable distributed storage that handles the varying AFRs of different underlying devices, without excessive redundancy or hidden risk of under-protecting data. The system must fulfill the following goals:

- **Always meet data reliability target:** ensure that data striped across any given set of disks is always covered enough to meet MTTDL in light of the specific AFRs of those disks at the moment.
- **Minimize overhead:** keep the IO work associated with adaptive redundancy management from interfering with foreground IO.
- **Maximize space efficiency:** avoid wasting space by over-protecting data relative to target MTTDL and the specific disk AFRs.

1.2 Contributions

This dissertation makes four primary contributions. The first contribution is making the case for disk-adaptive redundancy using the **Heterogeneity-Aware Redundancy Tuner (HeART)**. HeART is an adaptive redundancy system that replaces ad hoc "one-size-fits-all-devices" approaches to choosing redundancy parameters by dynamically selecting them on the basis of continuously monitoring underlying disk AFR behavior. HeART's algorithms are data-driven, based on the disk reliability analysis of a production storage cluster with over 100K disks. HeART could achieve target data reliability levels with fewer disks than traditional approaches: 11–16% fewer compared to erasure codes like 10-of-14 or 6-of-9 and 33% fewer compared to 3-way replication. The design of HeART and associated analyses are described in Chapter 3.

The second contribution is an **in-depth reliability analysis** of over 5 million disks belong-

ing to over 60 makes/models across several clusters belonging to Google, NetApp and Backblaze. This analysis reconfirms the highly varying AFRs across different disk makes/models. In addition we also highlights several important characteristics that directly affect the design of a disk-adaptive redundancy system. Some of the insights were previously unknown, whereas others redefine existing insights to provide a modern, and more comprehensive understanding of disk reliability in modern large-scale storage clusters. The datasets and insights are described in detail in Chapter 4.

The third contribution is the design of **Pacemaker**: a low-overhead disk-adaptive redundancy orchestrator that overcomes the IO load of dynamic redundancy transitions between schemes, termed *transition overload*. Pacemaker mitigates transition overload by (1) proactively organizing data layouts to make future transitions efficient, and (2) initiating transitions proactively in a manner that avoids urgency while not compromising on space-savings. Evaluation of Pacemaker with traces from four large (110K-450K disks) production clusters show that the transition IO requirement decreases to never needing more than 5% cluster IO bandwidth (only 0.2-0.4% on average Pacemaker achieves this while providing overall space-savings of 14-20% and never leaving data under-protected. Chapter 5 describes Pacemaker in detail.

The fourth and final contribution is the **design and implementation of disk-adaptive redundancy in the popular Hadoop Distributed File System (HDFS)**. This design repurposes existing machinery in order to carry out efficient redundancy transitions as outlined in Pacemaker. The details of this design are presented in Chapter 6.

We start the rest of the dissertation with Chapter 2 giving the required background and motivating disk-adaptive redundancy. The rest of the dissertation describes the above mentioned contributions in detail.

Chapter 2

Background and motivation

2.1 Overview of cluster storage systems

Large-scale distributed storage systems such as the Google File System [30], Colossus, Amazon S3 [2], Microsoft Azure [6], etc. are composed of multiple cluster storage systems (also called storage clusters). Each cluster is typically divided into multiple storage tiers, each with a distinct purpose, such as the caching tier (also known as the hot tier for short-lived or high-performance data), the primary storage tier (also known as the warm tier where the bulk of the data that is actively in use and has moderate churn resides), or the archival tier (also known as the cold tier meant for backups), etc. Large-scale storage clusters can have anywhere between tens to hundreds of thousands of storage devices in their primary storage tier. Furthermore, most clusters in today's large-scale distributed systems are designed in a disaggregated manner, wherein the space on all the storage devices in the same tier is exposed as a single large addressable storage pool [4]. Data stored in such clusters is stored across a number of storage devices. In most such clusters, clients access any given device directly (via a storage network) or indirectly (via whichever networked storage-node it is attached to) after contacting a logically-separate metadata service that maintains data location and other information.¹ While details vary, this basic architecture enables large and time-varying numbers of devices to be used effectively.

2.1.1 Hard disk drives are the primary storage devices

Owing to the economical price-point calculated in terms of dollars-per-terabyte (\$/TB), the primary storage tier of most of the large-scale storage clusters today is made up of hard disk drives (HDDs). Throughout this dissertation, any reference to a storage device or a disk implies a reference to a HDD. The storage capacity (space) of commonly used disks ranges from anywhere between 4TB-14TB. The price dynamics are unlikely to change drastically in the foreseeable future, ensuring that the bulk of the bytes in large-scale distributed systems and cloud computing platforms will continue to be stored on HDDs [4, 15].

¹For simplicity, the discussion below will discuss things in terms of “device” or “disk”, even though a storage-node may contain multiple devices.

2.1.2 Disk failures are common

In large-scale storage clusters disk failures happen all common. In fact, modern distributed storage systems are designed for failures, which means that they treat failures as the norm, and not an exception from the norm [30]. Storage devices exhibit two kinds of failures: *partial failures* and *fail-stops* (complete failures). Partial failures might involve a particular read or write failing because of a sector error or checksum failure. The disk as a whole is still functional, and usually it remaps failed sectors to spare locations in order to continue operation. In the case of fail-stop, the disk stops functioning altogether. This dissertation focuses on data redundancy optimizations for fail-stop failures where all the data from a disk is permanently lost when the disk fails.

2.1.3 Storage clusters have heterogeneous disk populations

A per-acquisition optimization of which make/model to acquire (such as targeting the lowest \$/TB option available at the time, or requiring the largest capacity disks) leads to a diverse population of disks in any given large-scale storage cluster. The storage clusters we evaluated had between 3-10 makes/models of disks deployed over a span of a few years. As disks age, or as technology advances, it is common to retire or decommission older disk models and replace them with newer, more advanced variants. Chapter 4 has more details about the composition of the different clusters evaluated as a part of this research.

2.2 Data reliability is achieved using data redundancy

Under above mentioned operating conditions where disk failures are very common. Data redundancy is the most common approach to protect against permanent data loss amongst continuous disk failures [10, 30, 39, 57, 61, 62, 66]. There are two common ways of performing data redundancy.

2.2.1 Replication

Replication involves creating exact copies of the data on different disks as backup. Usually, data is either duplicated (2-way replicated) or triplicated (3-way replicated). Replication not only makes the data more durable, but also improves its availability since the redundant copies of the data can be used to service reads in parallel.

The degraded mode read algorithm (i.e. the algorithm followed in case one of the disks on which data is stored fails) for replication is very straightforward. The other copies of data are referenced until the lost copy is replenished. There is a background process that continuously monitors if data is under-redundant, and replenishes lost copies using the healthy copies of data that are stored on other disks.

For hot data with certain access patterns, replication may be viewed as the right option for performance, with the replication factor chosen to provide sufficient reliability. But, of course, the cost consequence is a doubling or tripling of the number of disks needed. This makes replication infeasible for majority of the data in modern storage clusters.

2.2.2 Erasure coding

Erasure code is a more space-efficient redundancy mechanism compared to replication. For erasure coding a file, it is first broken into identically sized chunks of data where each chunk is at most a few MBs. Every k chunks of data are then combined with $n - k$ identically sized parity chunks to form an n chunk stripe. For the popularly used maximum distance separable (MDS) codes, any k chunks can be used to recover any of the n chunks. In this dissertation we use the notation k -of- n scheme to indicate an erasure code with k data chunks and $n - k$ parity chunks per stripe.

The degraded mode read algorithm for erasure coding is more costly and complex compared to that of replication. When a read is requested to a chunk that is missing because the disk on which it was stored has failed, it triggers a process called data reconstruction. Data reconstruction (for MDS codes) involves reading any k of the remaining chunks from the stripe that the missing chunk belongs to, to a common location (either the client or one of the server nodes). Then, the missing data is reconstructed using the combination of parity and data chunks that are read. The reconstructed data is then provided to the requesting client. Similar to replication, there is a background monitor process that keeps checking for under-redundant data and issues reconstructions proactively without being triggered by client reads.

It is increasingly common, in large-scale systems, to use erasure-coding (with $k > 1$) for bulk and colder data. With erasure-coding, space efficiency ($\frac{n}{k}$) for tolerating a given number of failures is better. But, additional considerations arise regarding the amount of work involved in recovering from a device failure—since k remaining chunks must be read to reconstruct each lost chunk, too high a k will result in more work than desired and, potentially, full recovery time that is too long. Schemes like 6-of-9 and 10-of-14 have been reported for real deployments [25, 26, 27, 61, 62, 66], and Backblaze reports use of 17-of-20 [11].

2.2.3 Metrics of data reliability

Annualized failure rate (AFR)

Annualized failure rate (AFR) is the standard metric used to describe a disk's fail-stop rate [19]. As the name suggests, it is the expected percentage of disks that will fail-stop in a given year from a population of disks. We calculate AFR using the following formula:

$$AFR (\%) = \frac{f_d}{n_1 + n_2 + \dots + n_d} \times 365 \times 100 \quad (2.1)$$

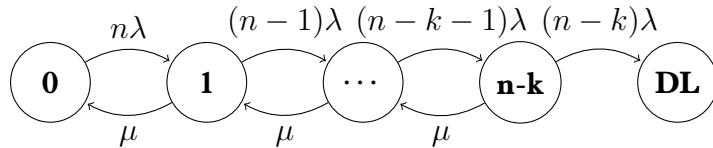
where f_d is the number of disks failed in d days and n_i is the number of disks operational during day i . Next, we will show how the AFR is used in the reliability calculation of a redundancy scheme.

Mean time to data loss (MTTDL)

The standard metric used for reliability of data is *mean time to data loss* (MTTDL). MTTDL is calculated based on two rates – *mean time to failure* (AFR) and *mean time to repair* (MTTR) [56].

AFR is a function of the disk's AFR, while the MTTR is the time it takes to reconstruct the lost data from the failed disk. In a large-scale cluster storage system, multiple copies of the data exist, that too on distinct disks. Thus, the effort of reconstructing the lost data of a failed disk is shared by numerous disks, which can be so efficient that the time it takes to detect that a disk has failed (approximately 15 minutes) dominates the time taken to reconstruct all of its data [27]. This decouples the MTTR of the disk from its storage capacity. Thus, in large cluster storage systems, the data reliability is only a function of the disk's AFR (which is a function of its AFR).

When using erasure coding for fault tolerance, MTTDL is calculated at a stripe granularity. A stripe is a set of fixed sized chunks that include both data and parity chunks. For a given redundancy scheme, let n be the total chunks in a stripe, for k data chunks.



2.3 Characterizing disk reliability over lifetime

2.3.1 The disk bathtub curve

Disk reliability varies over its lifetime as shown in Fig. 21. When the disk is in its infancy, it suffers from *infant mortality*. Infant mortality is also referred to as “burn-in” failure rate, and is an increased failure rate usually triggered due to new environment, workload pattern, etc. Eventually the disk AFR plateaus into a low, stable failure rate regime. This regime is called *useful life*. Infancy lasts 3–5 months whereas useful life lasts 3–4 years. Once the disk is heavily used for approximately 4 years, it suffers from another phase of high AFR similar to infancy. This phase of life is called *wearout*. When a disk is old enough to be considered to be in its wearout phase, it is usually decommissions before it fails. This way the data on the disk can be copied over to other disks rather than performing the IO and time intensive data reconstruction once the disk fails. Moreover, during reconstruction the data is vulnerable to being permanently lost if many such disks fail together.

2.4 Scheme selection problem for scalable storage

Generally, administrators determine the redundancy scheme to be used based on several factors, including cost, performance, and data reliability. For long-term storage, protecting data from loss is a critical constraint. Given the MTTDL and per-make/model AFR and MTTR, the subset of redundancy scheme options that would achieve a target MTTDL can be identified; the final selection can be based on the other considerations. Unfortunately, this simple-sounding approach doesn't work well for large-scale systems. Specifically, the MTTDL equations relies on devices all conforming to the same stationary AFR value and the

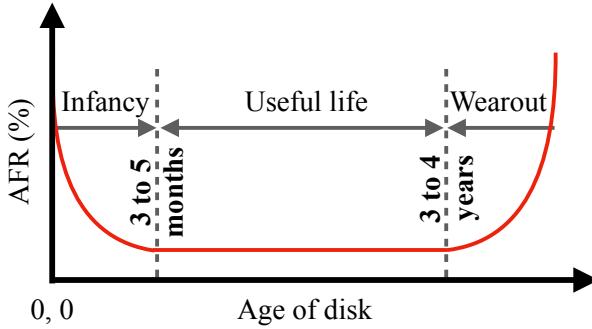


Figure 2.1: The disk hazard curve (also known as the bathtub curve) showing disk reliability over its lifetime.

administrator knowing that value; neither of which are true. Although it may be tempting to use AFR values taken from manufacturer’s specifications, studies have shown that failures rates observed in practice often do not match those [67]. More importantly, AFR values vary significantly between makes/models and for devices of different ages [41, 47, 57, 67]. Deployed disks roughly age through three phases of life: infancy, when the failure rate may be high and unstable; useful life, when the failure rate is lower and more stable; and wearout, where the failure rate climbs and grows more unstable.

The MTTDL equations can still be used to guide decisions, as long as a sufficiently high AFR value is used. For example, if the highest AFR value possible for any deployed make/model at any age allowed (before decommissioning) is used, the computed MTTDL will be a lower bound. Unfortunately, the range of possible AFR values in a deployed system is generally quite large [41, 47, 57, 70]. For example, we observe the range to be well over an order of magnitude in the NetApp and Google logs discussed later, from rare occurrences of double-digit AFRs to more commonly observed low single digit AFRs. Since the overall average is much closer to the lower end of the AFR range, the highest value is a very conservative overestimate for most of the devices. The resulting MTTDLs are thus very loose lower bounds, leading decision-makers to use excessive redundancy.

Nonetheless, we are told by admins of large storage deployments that such approaches, and other ad hoc approaches to conservative over-protection, are the norm. Such approaches are problematic in two significant ways. First, they result in much higher cost overheads than might otherwise be necessary. Second, and more insidious, they are still risky, because it is unclear (until too late) how conservative one needs to be.

Chapter 3

A case for disk-adaptive redundancy

3.1 The Heterogeneity-Aware Redundancy Tuner (HeART)

Large cluster storage systems almost always include a heterogeneous mix of storage devices, even when using devices that are all of the same type (e.g., Flash SSDs or mechanical HDDs). Commonly, this heterogeneity arises from incremental deployment combined with per-acquisition optimization of which make/model to acquire, such as targeting the lowest cost-per-byte option available at the time. As a result, a given cluster storage system can easily include several makes/models, each in substantial quantity.

Beyond performance and capacity differences, different makes/models can also have substantially different reliabilities. For example, Fig. 3.1 shows the average *annualized failure rates (AFRs)* during the useful life (stable operation period) for the 6 HDD make/model-based disk groups that make up more than 90% of the cluster storage system (with 100,000+ disks) used for the Backblaze backup service [10]. The highest failure rate is over 3.5 \times greater than the lowest, and no two are the same. Schroeder et al. [70] recently showed that different Flash SSD makes/models similarly exhibit substantial failure rate differences.

Despite such differences, the degree of redundancy employed in cluster storage systems for the purpose of long term data reliability (e.g., the degree of replication or erasure code parameters) is generally configured as if all of the devices have the same reliability. Unfortunately, this approach leads to configurations that are overly resource-consuming, overly risky, or a mix of the two. For example, if the redundancy settings are configured to achieve a given data reliability target (e.g., a specific *mean time to data loss (MTTDL)*) based on the highest AFR of any device make/model (e.g., S-4 from Fig. 3.1), then too much space will be used for redundancy associated with data that is stored fully on lower AFR makes/models (e.g., H-4A). Continuing this example, our evaluations show that the overall wasted capacity can be up to 16% compared to uniform use of erasure code settings stated as being used in real large-scale storage clusters [27, 61, 62, 66] and up to 33% compared to using 3-replication for all data—the direct consequence is increased cost, as more disks are needed. If redundancy settings for all data are based on lower AFRs, on the other hand, then data stored fully on higher-AFR devices is not sufficiently protected to achieve the data reliability target.

This chapter presents HeART (Heterogeneity-Aware Redundancy Tuner), an online tool

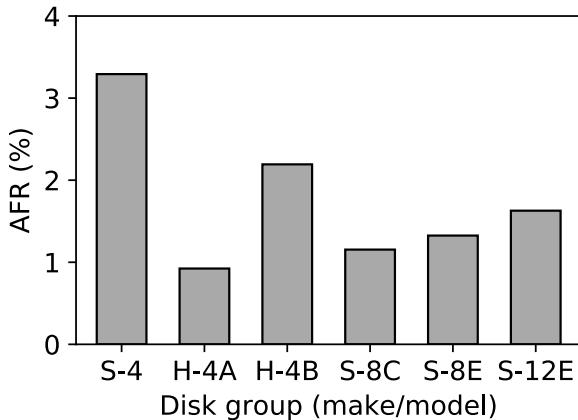


Figure 3.1: Annualized failure rate (AFR) for the six disk groups that make up $>90\%$ of the 100,000+ HDDs used for the Backblaze backup service [10]. Details of each disk group are given in Section 3.2.

for guiding exploitation of reliability heterogeneity among disks to reduce the space overhead (and hence the cost) of data reliability. HeART uses failure data observed over time to empirically quantify each disk group’s reliability characteristics and determine minimum-capacity redundancy settings that achieve specified target data reliability levels. For the Backblaze dataset of 100,000+ HDDs over 5 years, our analysis shows that using HeART’s settings could achieve data reliability targets with 11–33% fewer HDDs, depending on the baseline one-scheme-for-all settings. Even when the baseline scheme is a 10-of-14 erasure code whose space-overhead is already low, HeART further reduces disk space used by up to 14%.

Online (real-time) use of observed device reliability requires careful design. HeART uses robust statistical approaches to identify not only a steady-state AFR estimate for each disk group, but also the transitions between deployment stages: infancy \rightarrow useful life \rightarrow wearout, as in bathtub curve visualizations. HeART assumes that administrators have a baseline redundancy configuration that would be used in HeART’s absence; that same configuration should be used for a disk group, when it is initially deployed. HeART processes failure data for that disk group, during this initial period of 3–5 months, to determine both when infancy ends and a conservative AFR estimate for the useful life period. It also suggests the most space-efficient redundancy settings supported by the storage system that will achieve the specified data reliability target.

Naturally, the useful life period does not last forever. HeART continues to process failure data for each disk group, automatically identifying the onset of the wearout period. At this point, a transition to more conservative redundancy (e.g., the original baseline), and possibly decommissioning, is warranted. Importantly, HeART distinguishes between anomalous failure occurrences (e.g., one-time device-independent events, like a power surge, in which many devices fail together) and true changes in the underlying AFR.

This chapter makes four primary contributions. First, it highlights an often overlooked aspect of device heterogeneity (reliability) that should be exploited in cluster storage systems, and quantifies potential cost-and/or-reliability benefits. Second, it confirms the above ob-

servation and quantification with analysis of multi-year reliability data from a sizable cluster storage deployment (Backblaze), showing up to 11–33% reduction in the overall number of disks needed to achieve target data reliability. Third, it describes an online tool (HeART) that automatically determines per-disk-group useful life AFRs and durations, and identifies the right redundancy scheme settings for each. Fourth, it shows that the algorithms are effective using data from a large-scale production cluster (Backblaze) and are able to expose the expected capacity savings opportunities without compromising data reliability.

3.2 Having HeART can make you rich

This section builds a case for HeART by showing the benefits of using different redundancy schemes for disk groups exhibiting different reliability characteristics in the same commercially used cluster storage system. To support the case, we quantify space overhead reductions that can be achieved by adopting the different redundancy schemes.

3.2.1 The Backblaze dataset

Our analysis is based on an open source dataset from a data backup organization, Backblaze [10]. This dataset consists of over 5 years of disk reliability statistics from a production cluster storage system with over 100,000 HDDs.

We use the standard metric, *annualized failure rate (AFR)*, to describe a disk’s fail-stop rate¹ [19, 72]. As the name suggests, it is the expected percentage of disks that will fail-stop in a given year from a population of disks. AFR is calculated on day d , based on the past d days of reliability data, using the following formula:

$$AFR (\%) = \frac{f_d}{n_1 + n_2 + \dots + n_d} \times 365 \times 100 \quad (3.1)$$

where f_d is the number of disks failed in the past d days and n_i is the number of disks operational during day i .

Note that the AFR calculation is dependent on the number of days a disk was in operation. This can be tricky to estimate from the Backblaze dataset since the “death” of a disk in this dataset may also indicate its decommissioning, which may or may not imply its failure. We argue that, in the case of Backblaze, the date of decommissioning a disk only affects the absolute date at which it would have fail-stopped, but does not affect its *rate of failure*. Backblaze adopts a proactive disk replacement strategy that is driven by monitoring a combination of five S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) statistics.² The increased probability of failure indicated by grown defects in a disk is supported by several previous

¹Storage devices can exhibit *partial failures* and *fail-stops (complete failures)*. Partial failures might involve a particular read or write failing because of a sector error or checksum failure, while the disk as a whole is still functional. In the case of fail-stop, the disk stops functioning altogether.

²Backblaze uses S.M.A.R.T. 5 (Reallocated Sectors Count), S.M.A.R.T. 187 (Reported Uncorrectable Errors), S.M.A.R.T. 188 (Command Timeout), S.M.A.R.T. 197 (Current Pending Sector Count) and S.M.A.R.T. 198 (Uncorrectable Sector Count) as indicators that a disk is about to fail. [9]

Make/Model	Disk group shorthand	# of disks	Oldest disk age
Seagate ST4000DM000	S-4	37015	5 yrs
HGST HMS5C4040ALE640	H-4A	8715	4.77 yrs
HGST HMS5C4040BLE640	H-4B	15048	4.2 yrs
Seagate ST8000DM002	S-8C	9885	1.99 yrs
Seagate ST8000NM0055	S-8E	14395	1.2 yrs
Seagate ST12000NM0007	S-12E	21581	8 mts

Table 3.1: The disk groups identified from the Backblaze dataset for reliability heterogeneity analysis. The disk group shorthand above is used to represent the respective makes/models throughout the chapter.

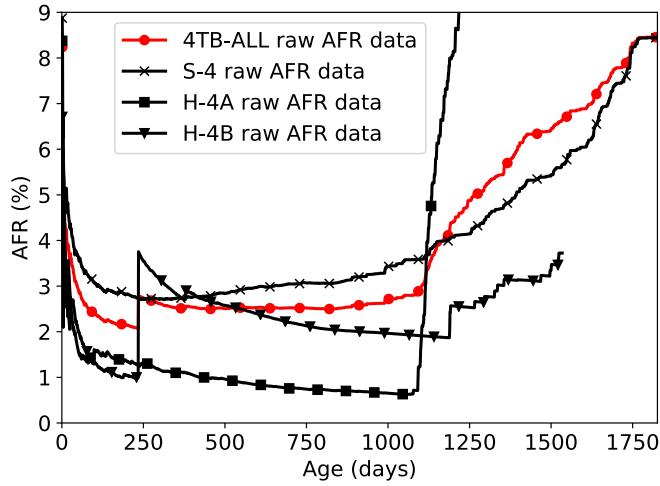


Figure 3.2: AFR comparison between all 4TB disks grouped together and disk groups broken down by make/model. The AFR differences in make/model-based grouping enables HeART to perform finer-grained specialization leading to higher benefits.

studies [2, 47, 57, 69]. In fact, Pinheiro et al. [67] show that the critical threshold for several S.M.A.R.T. attributes before their imminent failure is one—that is, the probability of failure of a disk in the next two months increases manifold when any of these S.M.A.R.T. attributes show a value greater than zero. Ma et al. [42] also show the high likelihood of disk failure by monitoring the reallocated sectors count (S.M.A.R.T. attribute 5), which is one of the signals used by Backblaze as a disk replacement indicator. Therefore, we believe that Backblaze’s proactive disk replacement rate is a reasonable approximation for the actual disk failure rate.

3.2.2 Disk group formation and varying AFRs

To effectively exploit heterogeneity in AFRs of different disk groups, we need to categorize the disks using some parameter that (1) groups disks with similar AFRs together and (2) has substantially different AFRs across groups. In whichever manner we choose to group the

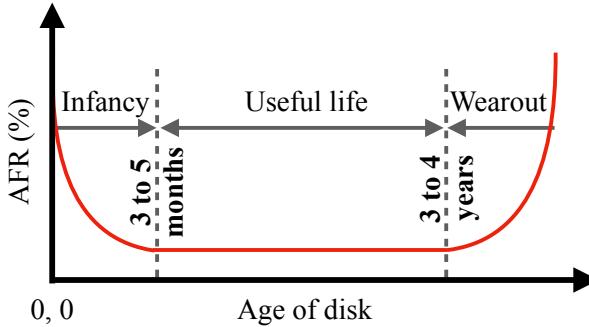


Figure 3.3: The canonical bathtub curve used to represent disk failure characteristics.

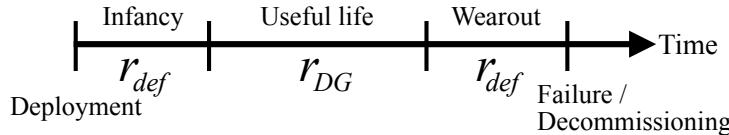


Figure 3.4: An abstract timeline of a disk group from deployment to failure or decommissioning, with the three distinct periods. The notations below the timeline (r_{def} and r_{DG}) denote the redundancy scheme employed during the respective stage.

disks, in order to gain statistical confidence in the AFR value, we need to ensure that each disk group has a sizeable population. Our definition of a sizeable population is approximately 10,000 or more disks. This is in line with disk populations considered in previous reliability studies [48]. We identify the following four ways to categorize disks:

- **By make/model:** Economies of scale result in large quantities of disks being purchased from the same vendor. Prior studies have shown that AFR may vary significantly by vintage [21, 47, 57].
- **By capacity:** Grown defects can be a function of disk capacity, thus causing disks of similar capacity to fail at a similar rate.
- **By operational conditions:** Disks that share similar vibration or temperature experiences may cause them to fail similarly. Thus, chassis placement and other operational conditions may influence failure rates.
- **By usage:** Increased space utilization or higher I/O rates may result in different disks showing different failure characteristics.

Unfortunately since we do not have access to the operational conditions or usage patterns, we can only analyze grouping on the basis of make/model or capacity.

Fig. B.2 shows the AFR by considering all 4TB disks as one disk group (red curve with circular marks) and the AFRs of the three make/models of 4TB disks as individual disk groups (black curves). We see significant differences between AFRs when disks are grouped by make/model, suggesting that grouping by capacity is insufficient. HeART groups disks by make/model.

Table B.1 shows the six make/model disk groups that make up over 90% of the Backblaze

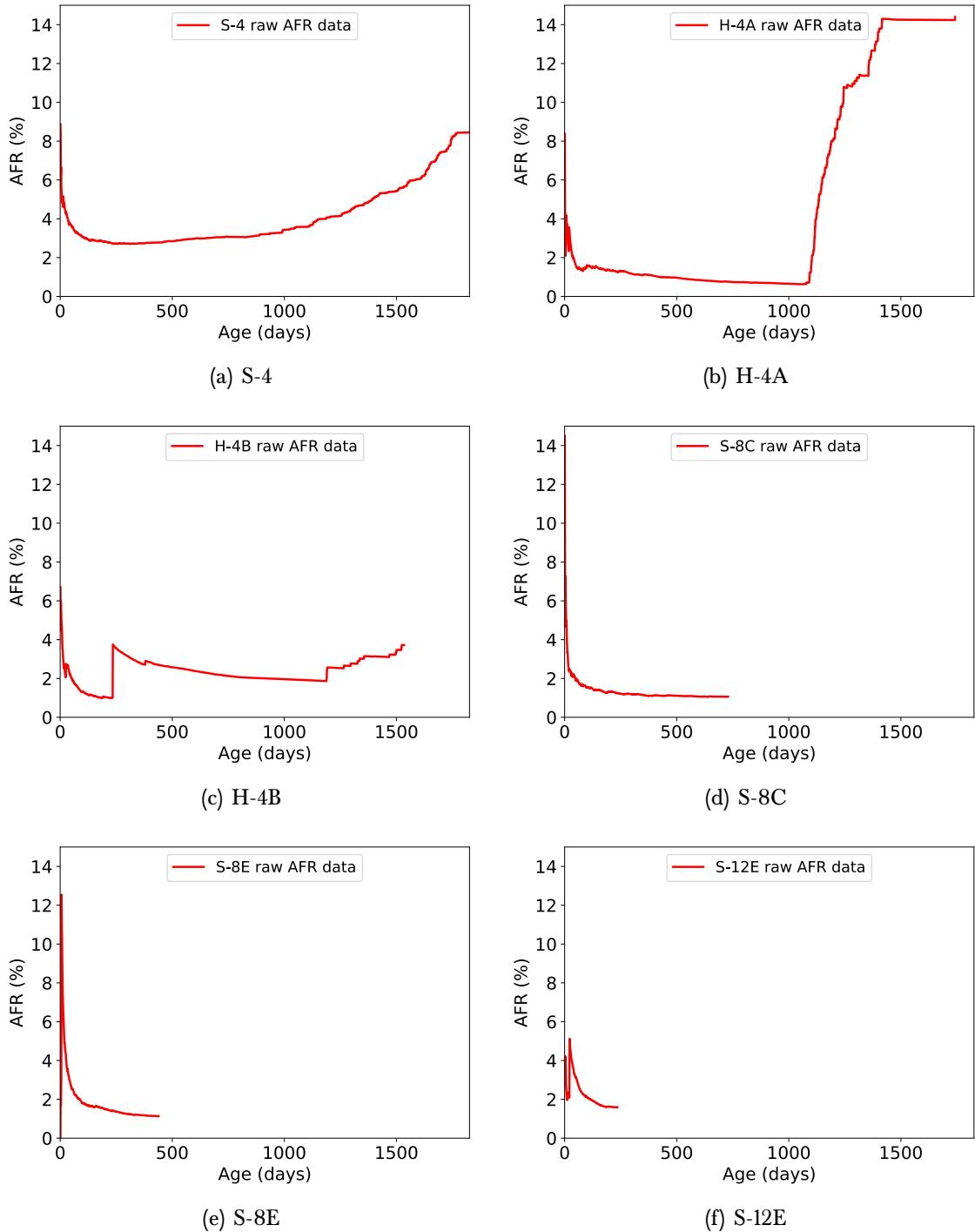


Figure 3.5: Cumulative raw AFR versus age (in days) for all six disk groups being analyzed.

deployment, with their population size, the age of their oldest disk, and the shorthand names we will use throughout the chapter.

AFR variation over time.

As expected, AFR values of each disk group vary over the lifetime of disks. It is well known that the AFR values over a disk’s lifetime follow a *bathtub curve* [22, 23, 89]. Fig. 3.3 shows the canonical representation of a bathtub curve. The lifetime is typically divided into three distinct periods:

- **Infant mortality:** A higher failure rate in the early days after deployment. This is also called the *burn-in period*.
- **Useful life period:** The stable region of operation, where rate of failure is lower.
- **Wearout stage:** A higher failure rate towards the disks’ end of life due to wear and aging.

Fig. 3.5 shows the AFR behavior versus age of the six disk makes/models. The three disks in the top row clearly exhibit all three stages of the bathtub curve.³ This is because the oldest disks from the S-4, H-4A and H-4B disk groups in the dataset are old enough to have entered their wearout stages. Since the deployment of S-8C, S-8E and S-12E disks has been more recent, these disk groups have ended their infant mortality, but are yet to enter their wearout stages.

3.2.3 Space savings from heterogeneous AFRs

Our goal is to reduce storage overhead by tailoring the redundancy scheme employed to the failure rate of a disk group during its useful life period. We parameterize a redundancy scheme using two parameters n (called “length”) and k (called “dimension”), and call it a (n, k) scheme.⁴ For any replication based scheme, $k = 1$ and n represents the total number of replicas. For any erasure coding based scheme, k represents the number of data chunks and $(n - k)$ is the number of parity chunks, thus resulting in n chunks in total.⁵ For an (n, k) redundancy scheme, the storage overhead is given by $\frac{n}{k}$.

achieves reduction in storage overhead by explicitly factoring in the group-specific AFR values in deciding the appropriate redundancy scheme for each disk group. Based on the canonical bathtub curve (Fig. 3.3), and the AFR curves shown in Fig. 3.5, we conclude that the safest stage to apply lower redundancy (without risking not meeting the reliability target) during a disk group’s lifetime is in its useful life period. Fig. 3.4 shows the abstract timeline of a disk group, where r denotes the redundancy scheme applied in each stage. Since all cluster storage systems today use some redundancy scheme whose resilience is acceptable to them,

³Fig. 3.5(c) corresponding to disk H-4B does not completely conform to the bathtub shape. We will discuss this case later in detail.

⁴This notation follows the standard notation employed in the coding theory literature.

⁵Although the description of the notation applies only to “systematic” codes, and most of the codes employed in storage systems are indeed systematic, HeART is applicable to storage systems employing non-systematic codes as well.

we assume that to be the *default* redundancy scheme. Since infancy and wearout periods have higher and less stable AFRs compared to useful life, for every disk group, HeART employs the default redundancy scheme for all infancy and wearout periods. This is shown as r_{def} in Fig. 3.4.

HeART suggests lower redundancy than the default scheme only during the useful life period, during which AFR values are relatively stable.

We use the standard metric for reliability of data employed in storage systems, *mean time to data loss* (MTTDL). MTTDL is calculated based on two rates – *mean time to failure* (AFR) and *mean time to repair* (MTTR) [56, 80]. AFR is directly related to the disk’s AFR. MTTR is the time it takes to reconstruct the lost data on the failed disk. Following prior work, we model the time to repair based on the time it takes to detect that a disk has failed (which is approximately 15 minutes) [27, 89]. We note that, by choosing the failure detection time as a proxy for the repair time, we are effectively choosing a lower bound on the repair time. Reliability differences between redundancy schemes are higher when repair times are higher, leading to even greater potential for space saving through HeART.

When a disk group enters its useful life period, HeART chooses a redundancy scheme (r_{DG}) that meets the following conditions:

1. is as reliable as r_{def} , i.e. $MTTDL^{r_{DG}} \geq MTTDL^{r_{def}}$
2. tolerates at least as many failures as r_{def}

According to condition 1 above, we need to set a target MTTDL in order to compare the resilience of different redundancy schemes. Although prior studies have shown MTTDL targets to be as low as 10,000 years [65], in order to ensure that we do not regress on reliability that disks in our dataset can currently offer, we set the target MTTDL to be the MTTDL of the default redundancy scheme applied on the disk group with the highest AFR. S-4 is the disk group with highest useful life AFR in the Backblaze dataset (refer Fig. 3.1). Therefore, for every default redundancy scheme, we will use S-4’s MTTDL for that scheme as the target MTTDL.

Multiple redundancy schemes can achieve the same or similar MTTDL values. These schemes can differ in their dimension (k) or the number of parity chunks per stripe ($n - k$) or both. It is well known that, generally speaking, codes with a longer dimension can provide the same MTTDL with lower space overhead compared to shorter codes. However, long codes consume significantly higher cluster bandwidth for reconstruction, since many more disks have to be accessed when performing reconstruction of failed data [89, 61, 62, 66]. The cluster bandwidth consumed during reconstruction is a major concern in erasure-coded storage systems. This has been highlighted in several works in the past [89, 61, 62, 66] and is consistent with our discussions with cluster storage system administrators. We, therefore, limit our cost reduction analysis to codes with at most $2\times$ the dimension (i.e., parameter k) of the default redundancy scheme.

Table 3.2 shows space savings for one disk group (H-4A) as an example. We will first highlight the space reduction when erasure coding schemes are used as the default, focusing on the (14, 10) and (9, 6) schemes known to have been used in large data centers [27, 61, 62, 66]. For (14, 10) as the default scheme, the MTTDL difference between H-4A and S-4 disks is over 580×. Thus, we can choose a weaker redundancy scheme (a scheme with

Disk groups		$r_{def} = (14, 10)$		$r_{def} = (9, 6)$		$r_{def} = (3, 1)$	
DG	AFR	r_{DG}	Cost↓	r_{DG}	Cost↓	r_{DG}	Cost↓
S-4	3.29%	(14, 10)	NA	(9, 6)	NA	(3, 1)	NA
H-4A	0.92%	(24, 20)	14%	(15, 12)	16%	(4, 2)	33%

Table 3.2: A sample of the estimated savings achievable through HeART. The space reductions obtained on H-4A disks by using redundancy schemes with lower storage overhead while meeting the reliability target set by applying the default redundancy scheme (r_{def}) on S-4 disks.

lower storage overhead $\frac{n}{k}$), so long as conditions 1 and 2 above are fulfilled. In fact, the high AFR differences allow us to use the longest allowed optimized code ($2 \times$ the dimension of the default redundancy scheme) for H-4A disks, i.e. (24, 20) leading to a useful life space reduction of 14%. Similarly, when using (9, 6) as the default scheme, the MTTDL difference between H-4A and S-4 is over 160×. This again allows us to choose the longest code for H-4A when $r_{def} = (9, 6)$, i.e. (15, 12), providing a space reduction of 16%.

For $r_{def} = 3$ -replication (recall that, under the (n, k) notation introduced above, 3-replication is denoted as the (3, 1) erasure code), we can tune the redundancy on H-4A disks to (4, 2) to respect our $2 \times$ default stripe dimension limit and still achieve an MTTDL that is approximately 11× that of S-4’s MTTDL. Using a (4, 2) scheme leads to a 33% reduction in disk space.

Large internet services companies try very hard to minimize free space (as low as 5%, according to some administrators) in order to minimize capital and operating costs. We are told that space savings translate directly into reduced numbers of disks needed, and even modest space savings (e.g., 10%) would build a solid case for tailoring redundancy schemes to heterogeneous disk AFRs.

We note that much of the reduction in storage overhead arises from allowing codes up to $2 \times$ in dimension (i.e., parameter k). However, simply employing an erasure code with twice the dimension for all data is not generally a suitable solution. First, the AFR for certain disk groups might be high enough to make codes with $2 \times$ dimension not acceptable causing them to miss the target reliability. Second, and more broadly, the reconstruction overheads can be unacceptable. For popular codes employed in practice, the amount of cluster bandwidth required for reconstruction is proportional to $k \times$ AFR, where k is the dimension of the code. The stable and lower AFR during a disk group’s useful life period allows the I/O generated for reconstruction to be contained even if longer codes are employed, which is why HeART optimizes redundancy schemes *only* during a disk group’s useful life. Using longer codes on data stored on disk groups in their infancy and wearout stages would exacerbate the cluster bandwidth consumption for reconstruction due to higher failure rates in these stages.

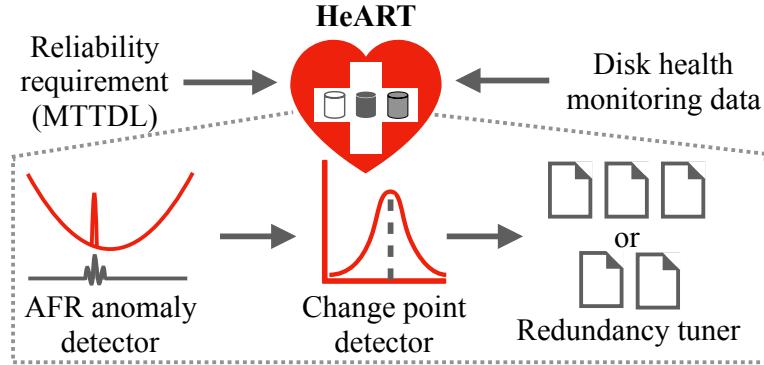


Figure 3.6: Schematic diagram of HeART. Components include an anomaly detector, an online change point detector, and a redundancy tuner.

3.3 The ways of the HeART

This section describes the challenges, design and implementation of HeART. We also quantify the cost reductions achieved by HeART for the Backblaze dataset.

3.3.1 Challenges

There are several challenges in taking the idea presented in Section 3.2 to practice.

Challenge 1: Function online and be quick

In making our case for HeART, we made use of the complete failure information (e.g., the full bathtub curve) for the disk groups. This helped in clearly identifying the 3 stages of a disk group's lifetime and AFR values in each of the stages. In practice, however, AFR values for disk groups deployed in cluster storage systems can only be known in an online fashion (i.e., as a continuous stream of reliability data, as it is observed). Furthermore, the crux of the cost reduction from HeART comes from quickly tuning the redundancy scheme as soon as we are confident of a disk group having entered its useful life period. Thus, our first challenge in building HeART is that it needs to function in an online fashion taking a continuous stream of disk health data as input and quickly react to the changes in the failure rate.

Challenge 2: Be accurate

It is important to correctly identify the three different stages of the bathtub curve for each disk group (recall Fig. B.3). If we are hasty in declaring the end of the infancy period or lax in identifying end of useful life, we might not meet the reliability target because of having tailored the redundancy to a relatively low failure rate during the useful life period. In contrast, if we are too lax about declaring end of infancy or too hasty in declaring onset of the wearout stage, the opportunity of cost reduction will diminish.

Challenge 3: Filter-out anomalies

Events such as power outages, natural disasters or human error can cause large numbers of disks to fail at once. It is important to distinguish between an accidental rise in AFR due to such anomalous events versus the rise in AFR due to onset of the wearout stage. Our third challenge is to perform AFR anomaly detection to avoid prematurely declaring end of useful life, consequently reducing the window of opportunity for cost reduction. At the same time, HeART needs to exercise caution so as to not treat a genuine rise in AFR as an anomaly, which risks not meeting reliability targets.

3.3.2 HeART architecture

Fig. 8.6 shows the primary components of HeART. HeART assumes the existence of a disk health monitoring/logging mechanism already in place, which is common in large-scale cluster storage deployments. From the time of deployment till the end of infancy, the default redundancy scheme (r_{def}) is used to protect the data stored on a disk group. Periodically, disk health data for each disk group is passed through an *anomaly detector*. Following an anomaly check, the cumulative AFR of every disk group is passed through a *change point detector*, which checks if a transition to different phase of life has occurred. Once the change point detector announces start of the useful life period, HeART suggests a new redundancy mechanism for the useful life of the disk group (r_{DG}). It computes a *determined useful life AFR* (AFR_{DG}), which is the AFR at the end of infancy padded with a tunable buffer, and uses it to calculate $MTTDL^{r_{DG}}$ for different redundancy scheme (r_{DG}) options. The buffer is introduced to tolerate the fluctuation of AFR during the useful life period (see Section 8.4.3). HeART keeps checking for anomalies and change points throughout the useful life period. When the change point detector marks the end of useful life, HeART raises an alert to reset the redundancy scheme to r_{def} to handle the increased AFR during wearout, as was handled in the absence of HeART.⁶

The remainder of this section describes our approach to addressing the above mentioned challenges. We leverage established tools and algorithms from online services and time-series analysis literature. While other options may perform even better, our evaluations indicate that these established tools are effective. We show the efficacy of HeART using the Backblaze dataset in Section 5.5.

3.3.3 Online anomaly detection

Incidents like losing power to a rack of disks, a natural disaster, or an accident, can cause a large number of failures resulting in a sudden rise in AFR. Such bulk failures can easily exceed the limits of any reasonable redundancy scheme, so administrators seek to mitigate them by defining appropriate failure domains and spreading data+redundancy across the failure domains [61, 62]. Such failures are not reflective of the true rise in AFR because of wearout, and therefore HeART considers these incidents as anomalies. It is important to note

⁶We note that the current architecture of HeART determines one useful life AFR for all disks belonging to a disk group and does not handle changes in the intra-disk-group reliability distribution over time.

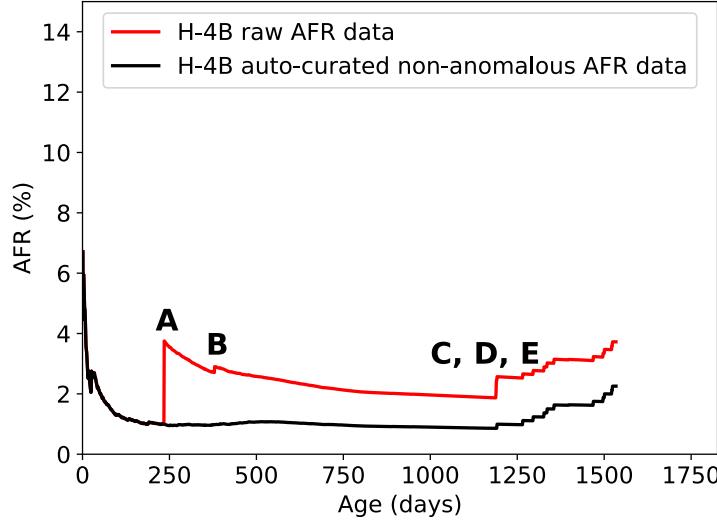


Figure 3.7: Raw and HeART-curated AFR curves for the H-4B disk group. Five spikes in *AFR* (points A–E), which correspond to four (anomalous) bulk failure events, are automatically filtered out by HeART.

that the benefits we extract from exploiting the reliability heterogeneity are proportional to the length of the useful life period, and therefore prematurely announcing wearout stage due to an anomaly would significantly diminish achievable gains.

We use the H-4B disks as a motivating example for anomaly detection (shown in Fig. 3.7). The raw AFR curve (red curve) shows that just after a few days into its useful life, there are large spikes in the AFR curve for drives that are about 235 days old (point A) and 380 days old (point B). Further along, we observe three more spikes that are in succession for disks that are about 1200 days old (points C, D and E). The failures corresponding to points A and B are all caused because of 322 drives failing on one particular date. Here, failure of disks of two different ages correspond to a failure event on the same day because these disks were deployed on different dates. Fig. 3.8 shows the total number of disks running and the per-day number of disk failures of H-4B as a function of the date. The left y-axis shows the cumulative disks of H-4B running on each day. The steps in the black curve show the incremental deployments of H-4B disks. The right y-axis shows the number of H-4B disks failing on each day (red curve). The tallest red spike in Fig. 3.8 corresponds to points A and B from Fig. 3.7. Points C, D and E occurred because of disks failing on different days.

In the absence of anomaly detection, HeART would have incorrectly concluded that the disk group's wearout stage began as early as point A.

3.3.4 Online change point detection

We refer to a transition in the AFR curve of a disk group as a *change point*. There are two major change points for each disk group: end of infant mortality stage and the onset of the wearout stage. This subsection describes our methods of identifying the two change points.

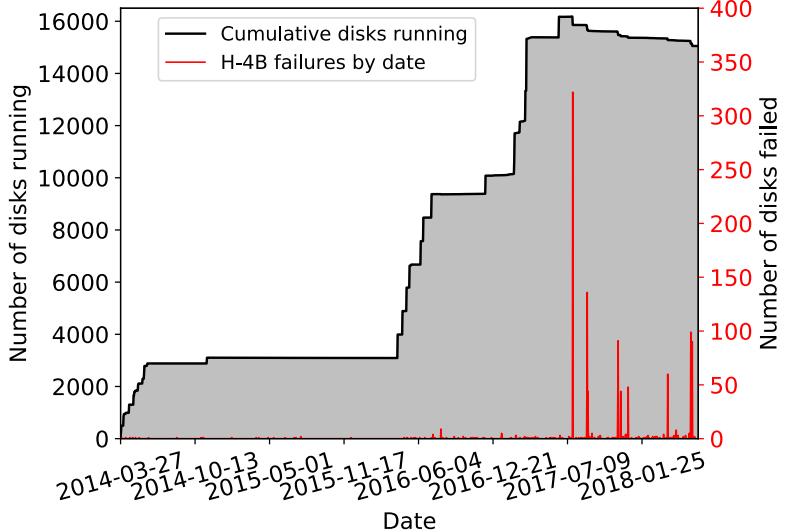


Figure 3.8: Total number of disks and number of disk failures by date for H-4B disks. The step-wise jumps in the black curve represent incremental deployments. The largest red spike represents the disks that failed on July 23, 2017, causing anomalies A and B in Fig. 3.7.

Onset of useful life period. HeART uses prior studies about infant mortality in HDDs along with change point detection to decide a disk group’s end of infancy. Prior studies performed on the Google and EMC disk fleets [42, 52] have shown that infant mortality lasts for approximately one quarter. Therefore, in order to be conservative, HeART exempts the first quarter from being assessed for end of infant mortality. Since disk reliability data is collected periodically, each time data is collected after the first 90 days, we run change point detection on the AFR curve generated by a sliding window of the past 30 days. HeART declares end of infancy if the last change point marked by the detector is over 30 days old, and the failure rate during the last 30 days is relatively constant. More precisely, HeART declares end of infancy when the difference between the observed maximum and minimum AFR values in at least 30 days past the last change point is less than a certain threshold T_{flat} . T_{flat} is the threshold for flatness and is a tunable parameter in HeART. Sensitivity to T_{flat} is evaluated in Section 3.4.3. Note that HeART takes a conservative approach in declaring the onset of the useful life period of a disk group in order to increase confidence about reducing redundancy for data stored on that disk group.

End of useful life period. Being lax in declaring the end of useful life period (i.e., onset of wearout) can risk in HeART not meeting the intended reliability target. Hence, HeART takes a conservative approach and marks the end of useful life for the first AFR observed that is greater than the determined useful life AFR. Since HeART checks for anomalous AFR fluctuations before checking for change points, if the anomaly detection phase does not filter out an increase in AFR, HeART assumes it to be a true increase in AFR. Thus, here too HeART takes a conservative approach and errs on the side of exiting the useful life period early and reverting to the default redundancy scheme.⁷

⁷Although the H-4A graph in Fig. 3.5(b) appears to show a sudden, huge rise in AFR, we believe that it

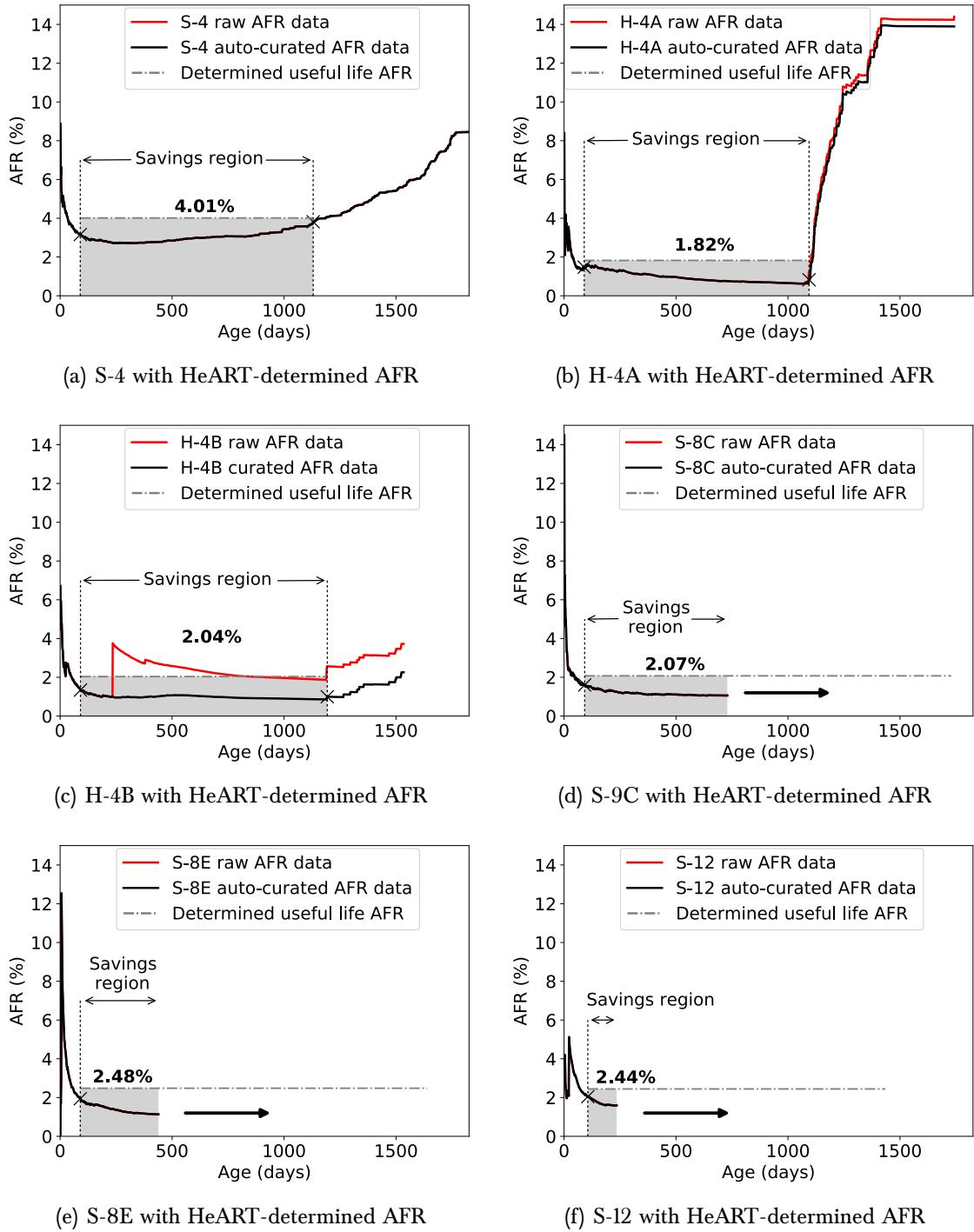


Figure 3.9: HeART in action on all disk groups, showing successful identification of infant mortality, useful life and wearout periods as well as automatic removal of anomalies.

3.4 Measuring HeART

This section describes implementation details of various components that make up HeART and presents an evaluation of HeART on the Backblaze dataset.

3.4.1 Implementation of the components

Our current implementation of HeART leverages existing, standard algorithms for anomaly detection and change point detection. Employing more sophisticated algorithms might lead to even better results.

Anomaly detector: For anomaly detection, our current implementation of HeART uses the RRCF algorithm [4] exposed by Amazon’s data analytics service offering called Kinesis [3].⁸ The anomaly detector acts on a reliability data stream made available by the disk health monitoring system. The output from the anomaly detector is also a data stream containing anomaly scores produced by the RRCF algorithm. Potential anomalies identified by RRCF have a higher anomaly score than data that the algorithm considers non-anomalous. RRCF generates the anomaly score based on how different the new data is compared to the recent past. For consistency with change point detection, we set the window size of the recent past to be one month. If the anomaly score is above a certain threshold, HeART considers that snapshot of reliability data as anomalous. RRCF advises to only consider the highest anomaly scores as true anomalies [4]. The anomaly score threshold is a tunable parameter in HeART. Lowering the score makes HeART more sensitive to fluctuations in AFRs.

Change point detector: Our current implementation of HeART uses a standard window-based change point detection algorithm, which compares the discrepancy between adjacent *sliding windows* within the AFR curve to determine if a change point has been encountered. In particular, we employ the *Ruptures* library for online change point detection [81, 82]. We set the sliding window size to one month, because AFRs at a lower granularity than a month are jittery.

3.4.2 Evaluation on the Backblaze dataset

Identifying useful life period

Fig. B.9 shows the results from HeART running on all 6 disk groups of the Backblaze dataset. HeART accurately identifies the infancy, useful life and wearout stages of the S-4, H-4A and H-4B disk groups shown in Figs. B.9(a), B.9(b) and B.9(c), respectively. For the S-8C, S-8E and S-12E disk groups (Figs. B.9(d), B.9(e) and B.9(f)), HeART identifies the end of infancy and correctly shows that they are still in their useful life. The width of the shaded region of each disk group highlights the “savings region”, i.e. the useful life period determined by HeART

is an artifact of Backblaze’s recording of decommissioned disks as failed, based on the device removal pattern seen in the failure data. Data from more sources are needed to confirm this hypothesis. If some disks do exhibit such transitions, then strategies for predicting failures (and wearout onset), such as by using S.M.A.R.T. statistics [6, 48, 88, 94], will be needed to use any but the most conservative redundancy schemes.

⁸We use Amazon’s service so as to avoid re-implementing a state-of-the-art algorithm.

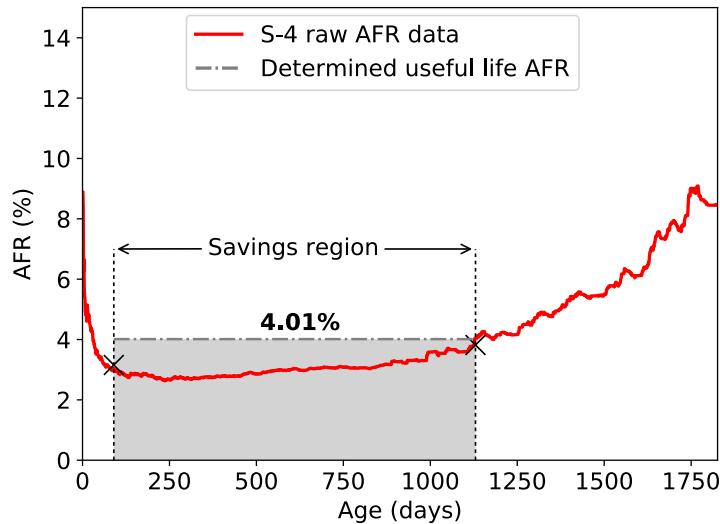


Figure 3.10: AFR of the S-4 disk group using a sliding window of 30 days. The determined useful life AFR value by HeART is conservative enough to subsume even the 30-day AFR values which vary more than the cumulative AFRs.

for which HeART potentially suggests a lower redundancy scheme. The height of the shaded region in Fig. 3.9 denotes the AFR values protected by the useful life AFR value determined by HeART for that disk group.

It is important to note that even though Fig. 3.9 shows cumulative AFR behavior, HeART performs anomaly detection and online change point detection on AFRs calculated using monthly sliding windows. Thus, not only is the cumulative AFR always inside the shaded region, but the instantaneous failure rate for any 30-day period is also less than the determined AFR value. In fact, the first rise in the instantaneous failure rate is what determines the end of the useful life period. Fig. 3.10 shows the instantaneous failure rate of S-4 disks being lower than the determined useful life AFR value throughout the useful life period.

In contrast to S-4 (Fig. 3.9(a)), the H-4A (Fig. 3.9(b)) and H-4B (Fig. 3.9(c)) disk groups have a sudden occurrence of their respective wearout stages. The quick reactivity requirement explained in Section 3.3 comes into effect for these disk groups. How quickly HeART reacts to changes in the AFR is determined by how quickly failure data is provided to HeART. Since Backblaze maintains daily snapshots of disk health, the quickest reaction to an increased failure rate is on the day that the failures occur. In our evaluation, HeART successfully identifies the increased AFR on the very day it was provided with the increased AFR data.

Anomaly detection

As explained in Section 3.3, the anomaly detector successfully detects five anomalies in the lifetime of H-4B disks. Additionally, two anomalies are also detected for the H-4A disks. Correctly identifying anomalous events increased the identified useful life period of H-4B disks by over 5 \times . In the absence of anomaly detection, the end of useful life period would have been incorrectly identified at age 235 days (shown by point A in Fig. 3.7).

Disk groups		$r_{def} = MTTDL_{4.01\% AFR}^{(14,10)} = 1.46E + 21$			
DG	AFR	$MTTDL^{r_{def}}$	r_{DG}	$MTTDL^{r_{DG}}$	Cost↓
S-4	4.01%	$1.46E + 21$	(14, 10)	$1.46E + 21$	NA
H-4A	1.82%	$7.57E + 22$	(24, 20)	$3.56E + 21$	14%
H-4B	2.04%	$4.28E + 22$	(24, 20)	$2.01E + 21$	14%
S-8C	2.07%	$3.98E + 22$	(24, 20)	$1.87E + 21$	14%
S-8E	2.48%	$1.61E + 22$	(21, 17)	$1.58E + 21$	11%
S-12E	2.44%	$1.75E + 22$	(21, 17)	$1.72E + 21$	11%

Disk groups		$r_{def} = MTTDL_{4.01\% AFR}^{(9,6)} = 3.31E + 16$			
DG	AFR	$MTTDL^{r_{def}}$	r_{DG}	$MTTDL^{r_{DG}}$	Cost↓
S-4	4.01%	$3.31E + 16$	(9, 6)	$3.31E + 16$	NA
H-4A	1.82%	$7.80E + 17$	(15, 12)	$7.20E + 16$	16%
H-4B	2.04%	$4.94E + 17$	(15, 12)	$4.56E + 16$	16%
S-8C	2.07%	$4.66E + 17$	(15, 12)	$4.30E + 16$	16%
S-8E	2.48%	$2.26E + 17$	(13, 10)	$3.99E + 16$	13%
S-12E	2.44%	$2.41E + 17$	(13, 10)	$4.26E + 16$	13%

Disk groups		$r_{def} = MTTDL_{4.01\% AFR}^{(3,1)} = 6.36E + 12$			
DG	AFR	$MTTDL^{r_{def}}$	r_{DG}	$MTTDL^{r_{DG}}$	Cost↓
S-4	4.01%	$6.36E + 12$	(3, 1)	$6.36E + 12$	NA
H-4A	1.82%	$6.80E + 13$	(4, 2)	$1.70E + 13$	33%
H-4B	2.04%	$4.83E + 13$	(4, 2)	$1.21E + 13$	33%
S-8C	2.07%	$4.62E + 13$	(4, 2)	$1.16E + 13$	33%
S-8E	2.48%	$2.69E + 13$	(4, 2)	$6.72E + 12$	33%
S-12E	2.44%	$2.82E + 13$	(4, 2)	$7.06E + 12$	33%

Table 3.3: Disk space saved by HeART by tuning the redundancy in the useful life of a disk group according to the observed disk group-specific AFRs. The units for MTTDLs is years. The cost savings are calculated for 3 default schemes: (14, 10) on AFR 4.01% disks, (9, 6) on AFR 4.01% disks and 3-replication (i.e. (3, 1)) on AFR 4.01% disks. Thus, the target reliability is the MTTDL of the respective default redundancy schemes using a 4.01% AFR (the r_{def} table header). The max dimension of the scheme permitted during useful life for each disk group has at most twice the dimension of default redundancy scheme, i.e. 20 data chunks for (14, 10), 12 data chunks for (9, 6) and 2 data chunks for 3-replication.

Cost savings per disk group

Table B.3 summarizes the cost savings of employing disk group specific redundancy in their respective useful lifespans. Disk groups with similar AFRs are grouped together. As discussed in Section B.2, we restrict the dimension (k) of the optimized code to at most $2\times$ that of the default redundancy scheme (r_{def}). In each case of r_{def} , we set the target reliability to the MTTDL achieved by using the highest-AFR disk group, which in the case of Backblaze are the S-4 disks.

It is important to note that the useful life AFRs determined by HeART are higher than the useful life AFRs shown in Fig. B.1. Recall from Section B.2, that HeART adds a (tunable) buffer above the useful life AFR determined at the end of infancy (which is an additional 25% by default). HeART chooses to be conservative in determining a useful life AFR value to ensure that reliability targets are comfortably met and to elongate the length of the useful life period to maximize benefits.

As in Section B.2, we exemplify the space reduction for erasure coding schemes using (14, 10) and (9, 6) schemes, which are known to have been employed in large-scale data centers [22, 61, 62, 66].

First, we evaluate using (14, 10) as the default redundancy scheme. (14, 10) has the lowest storage overhead ($1.4\times$) among the default redundancy schemes we evaluate, making it the hardest to find codes that meet the target MTTDL and reduce overhead even further. Despite these constraints, HeART enables a 14% space reduction for H-4A, H-4B and S-8C disks by suggesting a (24, 20) code and a reduction of 11% for S-8E and S-12E disks by suggesting a (21, 17) code.

Next, we measure HeART’s performance when using (9, 6) as the default redundancy scheme. We observe a space reduction of 16% on H-4A, H-4B and S-8C disks by using the maximum allowed (15, 12) redundancy scheme. For S-8E and S-12E disks, HeART suggests shorter (13, 10) code lengths compared to the above three disk groups in order to address their relatively higher determined AFR values, leading to a space reduction of 13%.

Finally, we also include the cost reduction for the canonical redundancy scheme, 3-replication, for completeness. We see that HeART enables 33% space reduction for all disk groups. We note that if replication is employed primarily for availability, that data may not be a candidate for tuning redundancy through .

For H-4A, H-4B and S-8C disks, HeART chose the $2\times$ max stripe-length for all three evaluated default redundancy schemes, extracting the maximum cost reduction (as explained in Table B.2). Even with the maximum allowed stripe length, the MTTDLs for the above disks are approximately $2.5\times$ higher than the target MTTDL value, suggesting further storage cost reductions if one is allowed even longer codes.

Overall cost reduction

To highlight the overall cost reduction achieved on the Backblaze disk fleet, we show the capacity-weighted cost savings in Fig. B.11. This cost reduction is over the whole lifetime of the disks (including the unoptimized infancy and wearout periods) and for all six disk groups (including the unoptimized S-4 disks). We only show the benefits for the erasure

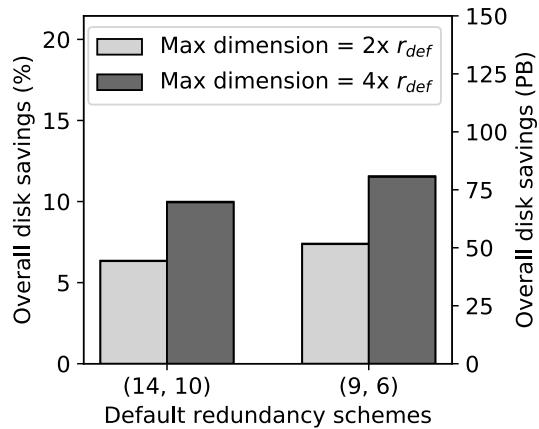


Figure 3.11: Overall space reduction achieved by HeART on the Backblaze dataset over the complete lifetime of every disk group, for erasure codes as the default scheme. For a maximum code dimension of up to $2 \times r_{def}$, we observe between 6 – 7.5% space reduction and for a maximum code dimension of up to $4 \times r_{def}$, we observe between 10 – 12% space reduction, translating to actual space savings of 40 – 80 PBs.

coding schemes we evaluated, leaving out 3-way replication, since erasure codes are the more popular choice for data durability. The overall cost reduction achieved with the maximum stripe dimension being $2 \times$ the default redundancy scheme is approximately 6% when using (14, 10) and approximately 7.5% when using (9, 6) as the default. If we relax the constraint of the maximum stripe dimension to $4 \times$ the dimension of the default redundancy scheme, we can expect to achieve between 10 – 12% overall space reduction. These modest percentage savings translate to significant savings in terms of actual storage space in large-scale clusters. For example, as shown on the right-side y-axis in Fig. 3.11, savings in storage space for the the Backblaze cluster range between 40 – 80 PBs.

3.4.3 Sensitivity analysis

There are several configuration parameters that govern the behavior of HeART, of which most are dependent on the ready-made tools we have used for different components of our system (e.g., the threshold for anomaly scores when using RRCF for anomaly detection). There are, however, two fundamental parameters that are independent of which anomaly detector or change point detector is used.

Before going into the details about the two parameters, we note that the modulation of both the parameters only has an effect on the gains that our optimization can yield. *Neither of them affects correctness* of our framework or protection of data in any way. This allows operators of cluster storage systems to start with conservative values, observe the AFR behavior of their disks and accordingly choose apt values to minimize their costs without risking not reaching their reliability target. We next discuss the two parameters.

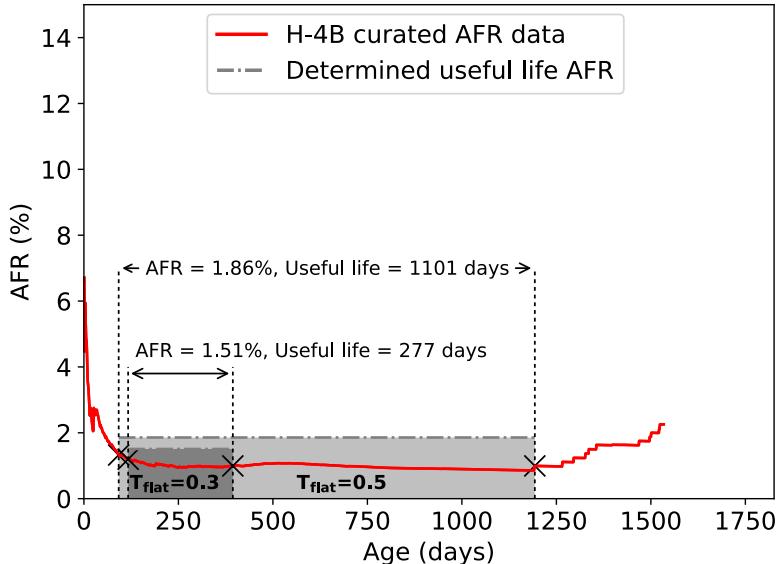


Figure 3.12: The effect of varying T_{flat} (AFR flatness threshold) on the H-4B disk group’s AFR curve. Larger T_{flat} implies a higher useful life AFR along with a larger useful life period. The default value for T_{flat} in HeART is 0.5.

Flatness parameter (T_{flat})

T_{flat} is used to deduce the end of the infant mortality period. As mentioned in Section 3.3.4, the end of infancy is defined as the first 30+ day period beyond the change point detected after the first quarter such that the difference between maximum and minimum observed AFR is below the threshold T_{flat} . Thus, T_{flat} essentially determines the flatness in the AFR curve for a given period. Currently, we define T_{flat} to be 0.5. A larger T_{flat} value will reduce the length of infancy until it reaches 90 days, beyond which it will have no effect. A lower T_{flat} will enforce a stricter flatness criteria, typically causing end of infancy to be declared late. Ending infancy sooner potentially causes HeART to choose a larger value as the determined useful life AFR. This, in turn causes HeART to choose a stronger redundancy scheme (with higher space overhead) compared to one that would have been chosen with the determined AFR value derived as a result of a lower T_{flat} value. This reduces the achievable savings within the useful life period of the disk group. As a tradeoff, we get a larger useful life period with a larger T_{flat} , since not only does infancy end sooner, but also the onset of wearout stage is postponed, since the increased useful life AFR now has higher tolerance to AFR variances throughout the useful life.

Fig. 3.12 shows the effect of varying T_{flat} on H-4B disks. We show the results for two different values $T_{flat} = 0.3$ and $T_{flat} = 0.5$. When T_{flat} was set to 0.3, we can see HeART declaring end of infancy at close to 100 days. Despite the buffer added to the determined useful life AFR at the end of infancy, the fluctuation in monthly AFRs caused a spike on day 394 to rise above the determined useful life AFR, causing HeART to announce end of useful life. In contrast, when $T_{flat} = 0.5$, infancy was declared to end on day 91, and the determined AFR value was high enough to tolerate the spike on day 394, increasing the useful life period

by a significant amount.

Useful life AFR buffer

The AFR buffer is the conservative padding added to the useful life AFR determined at the end of infancy. Currently, the useful life AFR is determined as the AFR value at the end of infancy *plus an additional buffer*, the tunable AFR buffer parameter. The choice of the buffer value has similar tradeoffs to the flatness parameters discussed above. A high buffer value implies a more conservative approach to setting the determined useful life AFR. This will prolong the useful life period, but restrict the tuning of the redundancy scheme due to the high useful life AFR value determined (and thus reducing benefits). In contrast, setting a low buffer value will shorten the useful life period but allow more cost reductions during the useful life. Operators can set the buffer based on AFR fluctuations observed in their storage systems, which can stem anywhere from workload patterns to operational conditions.

3.5 HeART-less alternatives (related work)

The closest related work can be classified into disk reliability studies that identify reliability heterogeneity, techniques to predict disk failures using reliability data, and systems that automate redundancy scheme selection.

Numerous studies have been conducted to characterize disk failures [12, 21, 37, 40, 47, 56, 57, 67, 68, 69, 74]. Among the studies conducted on large production systems, Shah and Elerath [21, 74], Pinheiro et al. [57] and Ma et al. [47] independently verify that failure rates are highly correlated with disk manufacturers. These studies were conducted on the NetApp, Google and EMC disk fleets, respectively. Schroeder and Gibson also conducted a similar reliability study on disks from a high performance computing environment [62], not only highlighting reliability heterogeneity between disks deployed across systems, but also pointing out that disk datasheet reliability is very different from reliability observed in the field. Recently, Schroeder et al. [70] highlighted the heterogeneity in the reliability of different SSD technologies from four different manufacturers. Also, Schroeder et al. [69] reported heterogeneity of partial disk failures (sector errors) across makes/models for NetApp’s disk fleet.

There have been numerous works that predict disk failures [35, 53, 77, 84, 92]. Among the more recent ones, Mahdisoltani et al. [48] use machine learning techniques to predict occurrence of partial disk errors using S.M.A.R.T. data. Anantharaman et al. [6] use random forests and recurrent neural networks to predict remaining useful life for HDDs. Both studies were performed on the Backblaze dataset.

Thereska et al. [79] built a self-prediction capability in cluster storage systems to assist in making informed redundancy and data placement decisions by answering *what-if* questions. It differs from HeART in that it does not perform and adapt to online analysis of reliability characteristics, relying on pre-knowledge of reliability metrics. Keeton et al. [42] built an optimization framework that automatically provided data dependability solutions to protect against site-level disasters by using information like workload patterns, and cost of recovery.

This work also assumes prior knowledge of failure rates. Tempo [76] is a system that proactively creates replicas to ensure high durability in wide-area network distributed systems. It does this economically by allowing the user to specify a maximum maintenance bandwidth, and its design revolves around the efficient use of a distributed hash table. Carbonite [18] is a replica maintenance solution for distributed storage systems spread over the Internet, which makes efficient use of bandwidth in maintaining redundancy in the face of transient failures.

Chapter 4

Disk reliability analysis in production environments

We examine disk reliability and deployment characteristics of over 5 million HDDs, covering over 60 makes/models, by analyzing multi-year logs from real-world environments. This chapter presents key insights that inform our understanding of the sources of transition overload and challenges/opportunities for a robust disk-adaptive redundancy solution which is described in the next chapter.

4.1 Longitudinal disk reliability datasets

The largest dataset comes from NetApp and contains information about disks deployed in filers (file servers). Each filer reports the health of each disk periodically (typically once a fortnight) using their AutoSupport [43] system. We analyzed the data for a subset of their deployed disks, which included over 50 makes/models and over 4.3 million disks total.

The other datasets come from large storage clusters deployed at Google and the Backblaze Internet backup service. Although the basic disk characteristics (e.g., AFR heterogeneity and the AFR behavior discussed below) are similar to the NetApp dataset, these datasets also capture the evolution and behavior in our target context (large-scale disaggregated storage clusters). The particular Google clusters shown were selected based on their longitudinal data availability, but were not otherwise screened for favorability. For each cluster, the multi-year log records (daily) all disk deployment, failure, and decommissioning events from birth of the cluster until the date of the log snapshot.

4.2 Observations and insights

4.2.1 Modern disaggregated storage clusters have over 100K HDDs

We show detailed analysis on four large-scale disaggregated storage clusters – three from Google and one from Backblaze. Google Cluster1’s disk population over three years included \approx 350K disks of 7 makes/models. Google Cluster2’s population over 2.5 years included \approx 450K

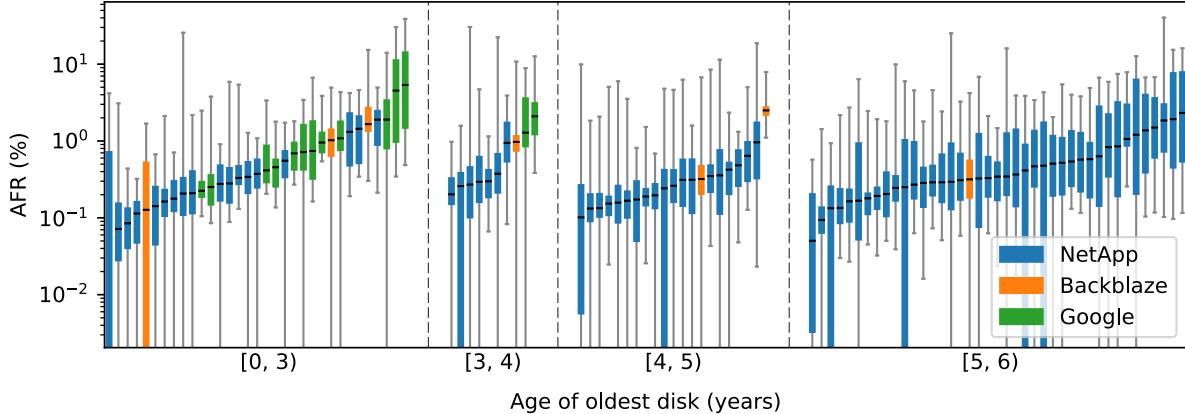


Figure 4.1: AFR spread for over 60 makes/models from NetApp, Google and Backblaze binned by the age of the oldest disk. Each box corresponds to a unique make/model, and at least 10000 disks of each make/model were observed (outlier AFR values omitted).

disks of 4 makes/models. Google Cluster3’s population over 3 years included $\approx 160K$ disks of 3 makes/models. The Backblaze cluster’s population since 2013 has included over 110K disks of 7 makes/models ranging from 4TB to 12TB. A subset of this dataset was also evaluated by HeART as shown in Chapter 3.

The insight for disk-adaptive redundancy systems

At such large scales space-savings of even 1% affects at least 1000 disks. Thus, optimizations such as disk-adaptive redundancy, which promise close to 20% space-savings directly translate to requiring tens-of-thousands of fewer disks, which dramatically reduces energy and storage costs.

4.2.2 Useful life AFRs are wildly heterogeneous

The highest useful life AFRs observed are close to two orders of magnitude compared to the lowest useful life AFRs across the 60 makes/models we analyzed across three organizations; NetApp, Google and Backblaze. Analyzed disks consist of both consumer and enterprise grade disks (most disks are enterprise). The proprietary nature of our data prevents us from disclosing the actual make/model or population of each disk type. Fig. 4.1 captures the range of AFRs that we observed in our analysis over disks of various ages.

The insight for disk-adaptive redundancy systems

The concept of disk-adaptive redundancy rests on exploiting reliability heterogeneity to have different redundancy for different sub-populations of disks. The observation that AFRs across different makes/models differs by orders of magnitude, and that this AFR heterogeneity is

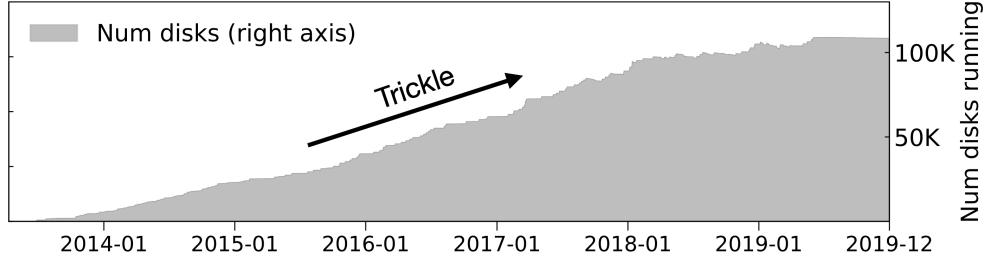


Figure 4.2: Backblaze cluster showing trickle deployments.

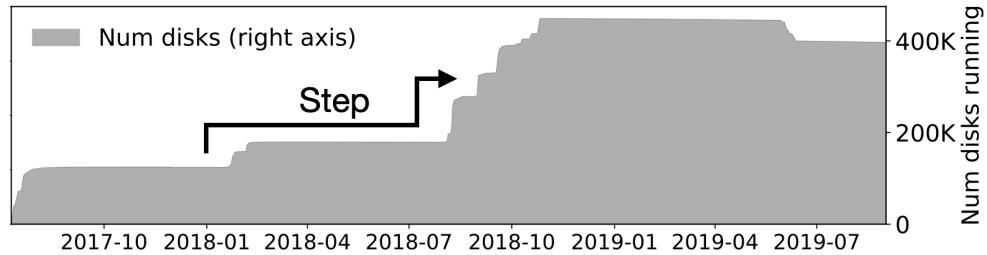


Figure 4.3: Google Cluster2 showing step deployments.

ubiquitous suggests that disk-adaptive redundancy is a useful approach that is applicable for most large-scale storage clusters.

4.2.3 Disk have two distinct deployment patterns

Trickle deployment

Trickle-deployed disks are added to a cluster frequently (weekly or even daily) over time by the tens and hundreds. Trickle-deployed disks may belong to different makes/models. Fig. 4.2 shows the Backblaze storage cluster which is entirely trickle-deployed.

Step deployment

Step-deployment introduces many thousands of disks into the cluster “at once” (over a span of a few days), followed by potentially months of no new step-deployments. Disks of a step are typically of the same make/model. Fig. 4.3 shows one of the Google clusters that is step-deployed.

A cluster may be entirely trickle-deployed (like the Backblaze cluster) or entirely step-deployed (like one of the Google clusters), but more often than not, it is usually a mix of the two (the remaining Google clusters we analyzed).

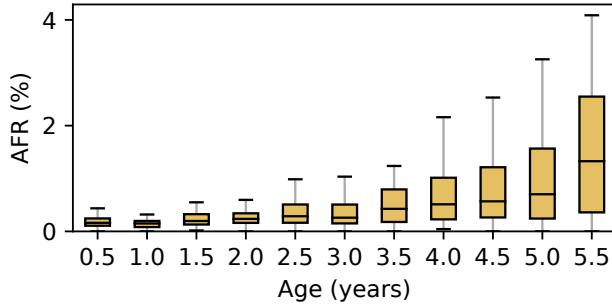


Figure 4.4: Distribution of AFR calculated over consecutive non-overlapping six-month periods for NetApp disks, showing the gradual rise of AFR with age (outliers omitted).

The insight for disk-adaptive redundancy systems

The nature of disk deployment affects the nature of disk transitions. Specifically, since trickle-deployed disks are deployed a-few-at-a-time, and since redundancy transitions are a function of disk age (as explained in Section 3.2.2) they also transition a-few-at-a-time. In contrast, since step-deployed disks are deployed in bulk, they also transition in bulk.

4.2.4 AFRs rise gradually over time with no clear wearout

The folklore of a single, flat useful life transitioning to a steep wearout phase was not observed in the disk populations we studied. Rather, in general, it was observed that AFR curves rise gradually as a function of disk age. Fig. 4.4 shows the gradual rise in AFR over six month periods of disk lifetimes. Each box represents the AFR of disks whose age corresponds to the six-month period denoted along the X axis. AFR curves for individual makes/models (e.g., Figs. 5.3(c) and 5.3(d)) are consistent with this aggregate illustration. Importantly, none of the over 60 makes/models from Google, Backblaze and NetApp displayed sudden onset of wearout.

The insight for disk-adaptive redundancy systems

Gradual increases in AFR, rather than a sudden rise in AFR suggests that one could anticipate disks approaching an AFR threshold. This enables disk-adaptive redundancy systems to act proactively, rather than reactively to increase redundancy such that data is never left under-protected.

4.2.5 Useful life could have multiple phases

Given the gradual rise of AFRs, useful life can be decomposed into multiple, piece-wise constant phases. Fig. 4.5 shows an approximation of the length of useful life when multiple phases are considered. Each box in the figure represents the distribution over different make/models of the approximate length of useful life. Useful life is approximated by considering the longest period of time which can be decomposed into multiple consecutive phases (number of phases

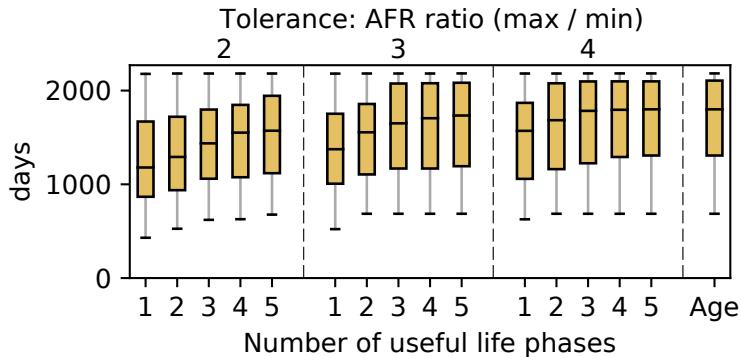


Figure 4.5: Approximation of useful life length for NetApp disks for 1-5 consecutive phases of useful life and three different tolerance levels. Each useful-life phase is approximated by an uninterrupted sequence of days during which AFR remains within the specified (atop graph) ratio between maximum and minimum AFRs. Boxes represent the distribution of the combined length of all phases for different makes/models. The box labeled “age” represents the distribution of the age of the oldest disk for different makes/models, which is an upper bound to the length of useful life.

indicated by the bottom x-axis) such that the ratio between the maximum and minimum AFR in each phase is under a given tolerance level (indicated by the top x-axis). The last box indicates the distribution over make/models of the age of the oldest disk, which is an upper bound to the length of useful life. As shown by Fig. 4.5, the length of useful life can be significantly extended (for all tolerance levels) by considering more than one phase. Furthermore, the data show that a small number of phases suffice in practice, as the approximate length of useful life changes by little when considering four or more phases.

The insight for disk-adaptive redundancy systems

Having multiple phases of useful life implies potentially multiple redundancy transitions to take adequate advantage of lower redundancy when in phases of life where AFR is low. At the same time, each additional transition has more IO associated with it, and therefore whether to optimize redundancy for each phase of useful life is a trade-off between IO and the resultant space savings.

4.2.6 Infancy often short-lived

Disk may go through (potentially) multiple rounds of so-called “burn-in” testing. The first tests may happen at the manufacturer’s site. There may be additional burn-in tests done at the deployment site allowing most of the infant mortality to be captured before the disk is deployed in production. For the NetApp and Google disks, we see the AFR drop sharply and plateau by 20 days for most of the makes/models. In contrast, the Backblaze disks display a slightly longer and higher AFR during infancy, which can be directly attributed to their less aggressive on-site burn-in.

The insight for disk-adaptive redundancy systems

A short-lived infancy implies a larger fraction of the disk lifetime (once it is deployed) to be spent in optimized redundancy, and thus an opportunity for higher space-savings.

Pacemaker's design is heavily influenced from these learnings, as will be explained in the next chapter.

Chapter 5

Combating transition overload in disk-adaptive redundancy systems

5.1 Pacemaker: avoiding HeART attacks

Storage clusters are made up of disks from a mix of makes/models acquired over time, and different makes/models have highly varying failure rates[41, 47, 57]. Despite that, storage clusters employ a “one-size-fits-all-disks” approach to choosing redundancy levels, without considering failure rate differences among disks. Hence, space overhead is often inflated by overly conservative redundancy levels, chosen to ensure sufficient protection for the most failure-prone disks in the cluster. Although tempting, the overhead cannot be removed by using very “wide” codes (which can provide high reliability with low storage overhead) for all data, due to the prohibitive reconstruction cost induced by the most failure-prone disks (more details in § 5.2). An exciting alternative is to dynamically adapt redundancy choices to observed failure rates (AFRs) for different disks, which recent proposals suggest could substantially reduce the space overhead [41].

Adapting redundancy involves dynamic transitioning of redundancy schemes, because AFRs must be learned from observation of deployed disks and because AFRs change over time due to disk aging. Changing already encoded data from one redundancy scheme to another, for example from an erasure code with parameters k_1 -of- n_1 to k_2 -of- n_2 can be exorbitantly IO intensive. Existing designs for disk-adaptive redundancy such as HeART are rendered unusable by overwhelming bursts of urgent transition IO when applied to real-world storage clusters. Indeed, as illustrated in Fig. 5.1(a), our analyses of production traces show extended periods of needing 100% of the cluster’s IO bandwidth for transitions. We refer to this as the *transition overload* problem. At its core, transition overload occurs whenever an observed AFR increase for a subset of disks requires too much urgent transition IO in order to keep data safe. Existing designs for disk-adaptive redundancy perform redundancy transitions as a reaction to AFR changes. Since prior designs are reactive, for an increase in AFR, the data is already under-protected by the time the transition to increase redundancy is issued. And it will continue to be under-protected until that transition completes. For example, around 2019-09 in Fig. 5.1(a), data was under-protected for over a month, even though the entire

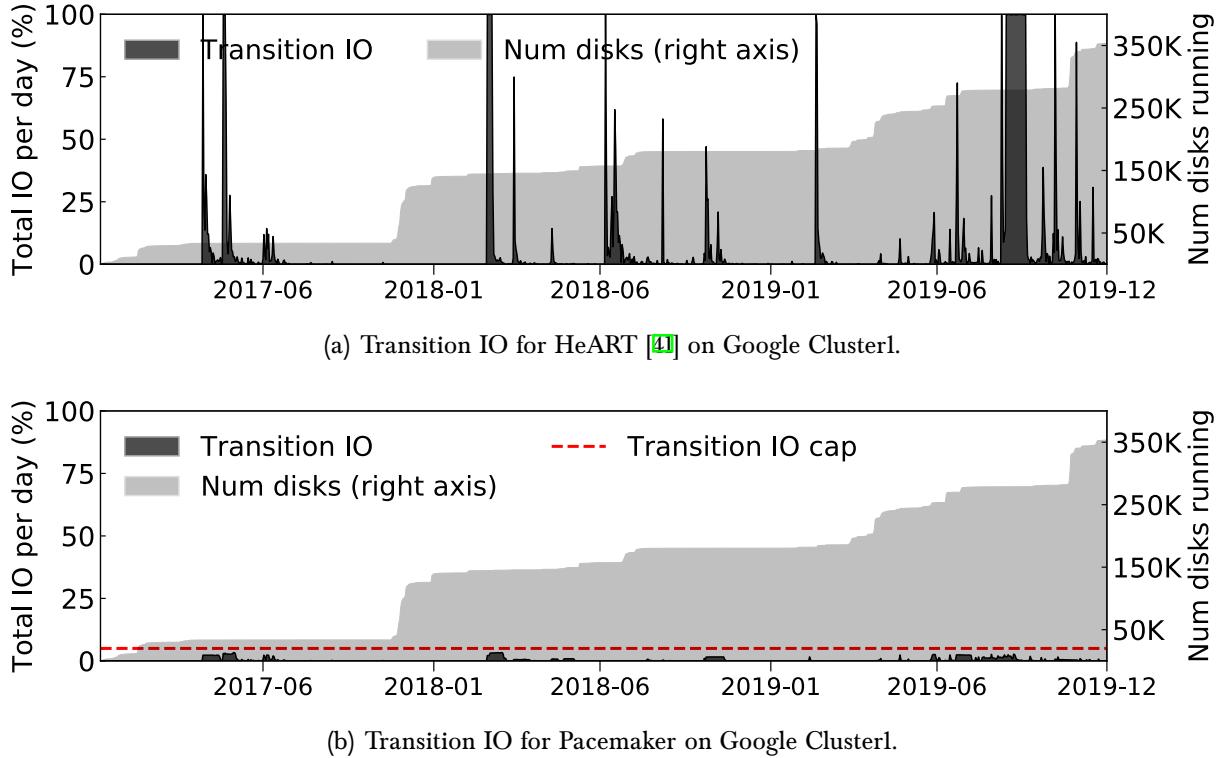


Figure 5.1: Fraction of total cluster IO bandwidth needed to use disk-adaptive redundancy for a Google storage cluster’s first three years. The state-of-the-art proposal [4] shown in [a] would require up to 100% of the cluster bandwidth for extended periods, whereas Pacemaker shown in [b] always fits its IO under a cap (5%). The light gray region shows the disk count (right Y-axis) over time.

cluster’s IO bandwidth was used solely for redundancy transitions. Simple rate-limiting to reduce urgent bursts of IO would only exacerbate this problem causing data-reliability goals to be violated for even longer.

To understand the causes of transition overload and inform solutions, we analyse multi-year deployment and failure logs for over 5.3 million disks from Google, NetApp and Backblaze. Two common transition overload patterns are observed. First, sometimes disks are added in tens or hundreds over time, which we call *trickle* deployments. A statistically confident AFR observation requires thousands of disks. Thus, by the time it is known that AFR for a specific make/model and age is too high for the redundancy used, the oldest thousands of that make/model will be past that age. At that point, all of those disks need immediate transition. Second, sometimes disks are added in batches of many thousands, which we call *step* deployments. Steps have sufficient disks for statistically confident AFR estimation. However, when a step reaches an age where the AFR is too high for the redundancy used, *all* disks of the step need immediate transition.

This chapter introduces *Pacemaker*, a new disk-adaptive redundancy orchestration system that exploits insights from the aforementioned analyses to eliminate the transition overload

problem. Pacemaker proactively organizes data layouts to enable efficient transitions for each deployment pattern, reducing total transition IO by over 90%. Indeed, by virtue of its reduced total transition IO, Pacemaker can afford to use extra transitions to reap increased space-savings. Pacemaker also proactively initiates anticipated transitions sufficiently in advance that the resulting transition IO can be rate-limited without placing data at risk. Fig. 5.1(b) provides a peek into the final result: Pacemaker achieves disk-adaptive redundancy with substantially less total transition IO and never exceeds a specified transition IO cap (5% in the graph).

We evaluate Pacemaker using logs containing 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 [10]. On all four clusters, Pacemaker provides disk-adaptive redundancy while using less than 0.4% of cluster IO bandwidth for transitions on average, and never exceeding the specified rate limit (e.g., 5%) on IO bandwidth. Yet, despite its proactive approach, Pacemaker loses less than 3% of the space-savings as compared to an idealized system with perfectly-timed and instant transitions. Specifically, Pacemaker provides 14–20% average space-savings compared to a one-size-fits-all-disks approach, without ever failing to meet the target data reliability and with no transition overload. We note that this is substantial savings for large-scale systems, where even a single-digit space-savings is worth the engineering effort. For example, in aggregate, the four clusters would need \approx 200K fewer disks.

We also implement Pacemaker in HDFS, demonstrating that Pacemaker’s mechanisms fit into an existing cluster storage system with minimal changes. Complementing our longitudinal evaluation using traces from large scale clusters, we report measurements of redundancy transitions in Pacemaker-enhanced HDFS via small-scale cluster experiments. Prototype of HDFS with Pacemaker is open-sourced and is available at <https://github.com/thesys-lab/pacemaker-hdfs.git>.

This chapter makes five primary contributions. First, it demonstrates that transition overload is a roadblock that precludes use of previous disk-adaptive redundancy proposals. Second, it presents insights into the sources of transition overload from longitudinal analyses of deployment and failure logs for 5.3 million disks from three large organizations. Third, it describes Pacemaker’s novel techniques, designed based on insights drawn from these analyses, for safe disk-adaptive redundancy without transition overload. Fourth, it evaluates Pacemaker’s policies for four large real-world storage clusters, demonstrating their effectiveness for a range of deployment and disk failure patterns. Fifth, it describes integration of and experiments with Pacemaker’s techniques in HDFS, demonstrating their feasibility, functionality, and ease of integration into a cluster storage implementation.

5.2 Simple re-encoding cannot reduce transition overload

Although HeART proposed an exciting approach to exploit disk-reliability heterogeneity, its design overlooks a crucial element: the IO cost associated with changing the redundancy schemes. Changing already encoded data under one erasure code to another can be exorbitantly IO intensive. We simulated HeART on real-world storage clusters, and simulating simple read-re-encode-write to re-encode data revealed extended periods where data safety

is at risk and where 100% cluster IO bandwidth is consumed for scheme changes. We call this problem *transition overload*.

An enticing solution that might appear to mitigate transition overload is to adapt redundancy schemes only by removing parities in low-AFR regimes and adding parities in high-AFR regimes. While this solution eliminates transition IO when reducing the level of redundancy, it does only marginally better when redundancy needs to be increased, because new parity creation cannot avoid reading all data chunks from each stripe. What makes this worse is that transitions that increase redundancy are time-critical, since delaying them would miss the MTTDL target and leave the data under-protected. Moreover, addition / removal of a parity chunk massively changes the stripe’s MTTDL compared to addition / removal of a data chunk. For example, a 6-of-9 MTTDL is $10000\times$ higher than 6-of-8 MTTDL, but is only $1.5\times$ higher than 7-of-10 MTTDL. AFR changes would almost never be large enough to safely remove a parity, given default schemes like 6-of-9, eliminating almost all potential benefits of disk-adaptive redundancy.

This chapter analyzes disk deployment and failure data from large-scale production clusters to discover sources of *transition overload* and informs the design of a solution. It then describes and evaluates Pacemaker, which realizes the dream of safe disk-adaptive redundancy without transition overload.

5.3 Design goals

Pacemaker is an IO efficient redundancy orchestrator for storage clusters that support disk-adaptive redundancy. Before going into the design goals for Pacemaker, we first chronicle a disk’s lifecycle, introducing the terminology that will be used in the rest of the chapter (defined in Table 5.1).

5.3.1 Disk lifecycle under Pacemaker

Throughout its life, each disk under Pacemaker simultaneously belongs to a *Dgroup* and an *Rgroup*. There are as many Dgroups in a cluster as there are unique disk makes/models. Rgroups on the other hand are a function of redundancy schemes and placement restrictions. Each Rgroup has an associated redundancy scheme, and its data (encoded stripes) must reside completely within that Rgroup’s disks. Multiple Rgroups can use the same redundancy scheme, but no stripe may span across Rgroups. The Dgroup of a disk never changes, but a disk may transition through multiple Rgroups during its lifetime. At the time of deployment (or “birth”), the disk belongs to *Rgroup0*, and is termed as an *unspecialized disk*. Disks in Rgroup0 use the default redundancy scheme, i.e. the conservative one-scheme-fits-all scheme used in storage clusters that do not have disk-adaptive redundancy. The redundancy scheme employed for a disk (and hence its Rgroup) changes via *transitions*. The first transition any disk undergoes is an *RDn transition*. A RDn transition changes the disk’s Rgroup to one with lower redundancy, i.e. more optimized for space. Whenever the disk departs from Rgroup0, it is termed as a *specialized disk*. Disks depart from Rgroup0 at the end of their infancy. Since infancy is short-lived (§4.2.6), Pacemaker only considers one RDn transition for each disk.

<i>Term</i>	<i>Definition</i>
Dgroup	Group of disks of the same make/model.
Transition	The act of changing the redundancy scheme.
RDn transition	Transition to a lower level of redundancy.
RUp transition	Transition to a higher level of redundancy.
peak-IO-cap	IO bandwidth cap for transitions.
Rgroup	Group of disks using the same redundancy with placement restricted to the group of disks.
Rgroup0	Rgroup using the default one-scheme-fits-all redundancy used in storage clusters today.
Unspecialized disks	Disks that are a part of Rgroup0.
Specialized disks	Disks that are not part of Rgroup0.
Canary disks	First few thousand disks of a trickle-deployed Dgroup used to learn AFR curve.
Tolerated-AFR	Max AFR for which redundancy scheme meets reliability constraint.
Threshold-AFR	The AFR threshold crossing which triggers an RUp transition for step-deployed disks.

Table 5.1: Definitions of Pacemaker’s terms.

The first RDn transition occurs at the start of the disk’s useful life, and marks the start of its specialization period. As explained in §4.2.5, a disk may experience multiple useful life phases. Pacemaker performs a transition at the start of each useful-life phase. After the first (and only) RDn transition, each subsequent transition is an *RUp transition*. An RUp transition changes the disk’s Rgroup to one with higher redundancy, i.e. less optimized for space, but the disk is still considered a specialized disk unless the Rgroup that the disk is being RUp transitioned to is Rgroup0. The space-savings (and thus cost-savings) associated with disk-adaptive redundancy are proportional to the fraction of life the disks remain specialized for.

5.3.2 Key decisions

To adapt redundancy throughout a disk’s lifecycle as chronicled above, three key decisions related to transitions must be made

1. *When should the disks transition?*
2. *Which Rgroup should the disks transition to?*
3. *How should the disks transition?*

5.3.3 Constraints

The above decisions need to be taken such that a set of constraints are met. An obvious constraint, central to any storage system, is that of data reliability. The *reliability constraint*

mandates that all data must always meet a predefined target MTTDL. Another important constraint is the *failure reconstruction IO constraint*. This constraint bounds the IO spent on data reconstruction of failed disks, which as explained in §5.2 is proportional to AFR and scheme width. This is why wide schemes cannot be used for all disks all the time, but they can be used for low-AFR regimes of disk lifetimes (as discussed in §5.2).

Existing approaches to disk-adaptive redundancy make their decisions on the basis of only these constraints [41], but fail to consider the equally important *IO caused by redundancy transitions*. Ignoring this causes the transition overload problem, which proves to be a showstopper for disk-adaptive redundancy systems. Pacemaker treats transition IO as a first class citizen by taking it into account for each of its three key decisions. As such, Pacemaker enforces carefully designed constraints on transition IO as well.

5.3.4 Designing IO constraints on transitions

Apart from serving foreground IO requests, a storage cluster performs numerous background tasks like scrubbing and load balancing [12, 54, 69]. Redundancy management is also a background task. In current storage clusters, redundancy management tasks predominantly consist of performing data redundancy (e.g. replicating or encoding data) and reconstructing data of failed or otherwise unavailable disks. Disk-adaptive redundancy systems add redundancy transitions to the list of IO-intensive background tasks.

There are two goals for background tasks: Goal 1: they are not too much work, and Goal 2: they interfere as little as possible with foreground IO. Pacemaker applies two IO constraints on background transition tasks to achieve these goals: (1) *average-IO constraint* and (2) *peak-IO constraint*. The average-IO constraint achieves Goal 1 by allowing storage administrators to specify a cap on the fraction of the IO bandwidth of a disk that can be used for transitions over its lifetime. For example, if a disk can transition in 1 day using 100% of its IO bandwidth, then an average-IO constraint of 1% would mean that the disk will transition at most once every 100 days. The peak-IO constraint achieves Goal 2 by allowing storage administrators to specify the peak rate (defined as the *peak-IO-cap*) at which transitions can occur so as to limit their interference with foreground traffic. Continuing the previous example, if the peak-IO-cap is set at 5%, the disk that would have taken 1 day to transition at 100% IO bandwidth would now take at least 20 days. The average-IO constraint and the peak-IO-cap can be configured based on how busy the cluster is. For example, a cluster designed for data archival would have a lower foreground traffic, compared to a cluster designed for serving ads or recommendations. Thus, low-traffic clusters can set a higher peak-IO-cap resulting in faster transitions and potentially increased space-savings.

5.3.5 Design goals

The key design goals are to answer the three questions related to transitions such that the space-savings are maximized and the following constraints are met: (1) reliability constraint on all data all the time, (2) failure reconstruction IO constraint on all disks all the time, (3) peak-IO constraint on all disks all the time, and (4) average-IO constraint on all disks over time.

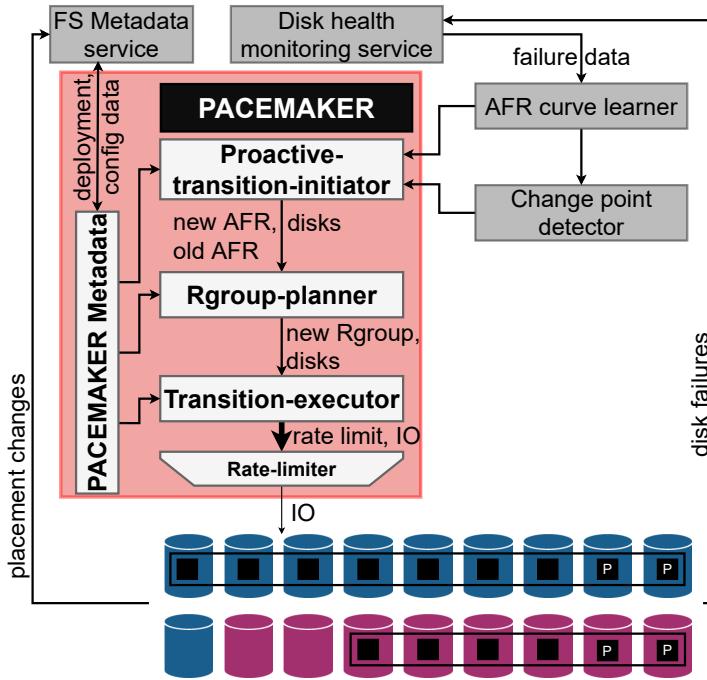


Figure 5.2: Pacemaker architecture.

5.4 Design of Pacemaker

Fig. 5.2 shows the high level architecture of Pacemaker and how it interacts with some other components of a storage cluster. The three main components of Pacemaker correspond to the three key decisions that the system makes as discussed in §5.3. The first main component of Pacemaker is the *proactive-transition-initiator* (§5.4.1), which determines when to transition disks using the AFR curves and the disk deployment information. The information of the transitioning disks and their observed AFR is passed to the *Rgroup-planner* (§5.4.2), which chooses the Rgroup to which the disks should transition. The Rgroup-planner passes the information of the transitioning disks and the target Rgroup to the *transition-executor* (§5.4.3). The transition-executor addresses how to transition the disks to the planned Rgroup in the most IO-efficient way.

Additionally, Pacemaker also maintains its own *metadata* and a simple *rate-limiter*. Pacemaker metadata interacts with all of Pacemaker's components and also the storage cluster's metadata service. It maintains various configuration settings of a Pacemaker installation along with the disk deployment information that guides transition decisions. The rate-limiter rate-limits the IO load generated by any transition as per administrator specified limits. Other cluster components external-to-Pacemaker that inform it are the *AFR curve learner* and the *change point detector*. As is evident from their names, these components learn the AFR curve

of each Dgroup and identify change points for redundancy transitions. The AFR curve learner receives failure data from the *disk health monitoring service*, which monitors the disk fleet and maintains their vitals.

5.4.1 Proactive-transition-initiator

Proactive-transition-initiator's role is to determine *when to transition the disks*. Below we explain Pacemaker's methodology for making this decision for the two types of transitions (RDn and RUp) and the two types of deployments (step and trickle).

Deciding when to RDn transition a disk

Recall that a disk's first transition is an RDn transition. As soon as proactive-transition-initiator observes (in a statistically accurate manner) that the AFR has decreased sufficiently, and is stable, it performs an RDn transition from the default scheme (i.e., from Rgroup0) employed in infancy to a more space-efficient scheme. This is the only RDn transition in a disk's lifetime.

Deciding when to RUp transition a disk

RUp transitions are performed either when there are too few disks in any Rgroup such that data placement is heavily restricted (which we term *purging an Rgroup*), or when there is a rise in AFR such that the reliability constraint is (going to be) violated. Purging an Rgroup involves RUp transitioning all of its disks to an Rgroup with higher redundancy. This transition isn't an imminent threat to reliability, and therefore can be done in a relaxed manner without violating the reliability constraint as explained in §[5.4.3](#).

However, most RUp transitions in a storage cluster are done in response to a rise in AFR. These are challenging with respect to meeting IO constraints due to the associated risk of violating the reliability constraints whenever the AFR rises beyond the AFR tolerated by the redundancy scheme (termed *tolerated-AFR*).

In order to be able to safely rate-limit the IO load due to RUp transitions, Pacemaker takes a *proactive* approach. The key is in determining when to initiate a proactive RUp transition such that the transition can be completed before the AFR crosses the tolerated-AFR, while adhering to the IO and the reliability constraints without compromising much on space-savings. To do so, the proactive-transition-initiator assumes that its transitions will proceed as per the peak-IO constraint, which is ensured by the transition-executor. Pacemaker's methodology for determining when to initiate a proactive RUp transition is tailored differently for trickle versus for step deployments, since they raise different challenges.

Proactive-transition-initiator for trickle-deployed disks

For trickle-deployed disks, Pacemaker considers two category of disks: (1) first disks to be deployed from any particular trickle-deployed Dgroup, and (2) disks from that Dgroup that are deployed later.

Pacemaker labels the first C deployed disks of a Dgroup as *canary* disks, where C is a configurable, high enough number of disks to yield statistically significant AFR observations. For example, based on our disk analyses, we observe that C in low thousands (e.g., 3000) is sufficient. The canary disks of any Dgroup are the first to undergo the various phases of life for that Dgroup, and these observations are used to learn the AFR curve for that Dgroup. The

AFR value for the Dgroup at any particular age is not known (with statistical confidence) until all canary disks go past that age. Furthermore, due to the trickle nature of the deployment, the canary disks would themselves have been deployed over weeks if not months. Thus, AFR for the canary disks can be ascertained only in retrospect. Pacemaker never changes the redundancy of the canary disks to avoid them from ever violating the reliability constraint. This does not significantly reduce space-savings, since C is expected to be small relative to the total number of disks of a Dgroup (usually in the tens of thousands).

The disks that are deployed later in any particular Dgroup are easier to handle, since the Dgroup's AFR curve would have been learned by observing the canaries. Thus, the date at which a disk among the later-deployed disks needs to RUp to meet the reliability constraints is known in advance by the proactive-transition-initiator, which it uses to issue proactive RUp transitions.

Proactive-transition-initiator for step-deployed disks

Recall that in a step deployment, most disks of a Dgroup may be deployed within a few days. So, canaries are not a good solution, as they would provide little-to-no advance warning about how the AFR curve's rises would affect most disks.

Pacemaker's approach to handling step-deployments is based on two properties: (1) Step-deployments have a large number of disks deployed together, leading to a statistically accurate AFR estimation; (2) AFR curves based on a large set of disks tend to exhibit gradual, rather than sudden, AFR increases as the disk ages (§4.2.4). Pacemaker leverages these two properties to employ a simple *early warning* methodology to predict a forthcoming need to RUp transition a step well in advance. Specifically, Pacemaker sets a threshold, termed *threshold-AFR*, which is a (configurable) fraction of the tolerated-AFR of the current redundancy scheme employed. For step-deployments, when the observed AFR crosses the threshold-AFR, the proactive-transition-initiator initiates a proactive RUp transition.

5.4.2 Rgroup-planner

The Rgroup-planner's role is to determine *which Rgroup should disks transition to*. This involves making two *interdependent* choices: (1) the redundancy scheme to transition into, (2) whether or not to create a new Rgroup.

Choice of the redundancy scheme

At a high level, the Rgroup-planner first uses a set of selection criteria to arrive at a set of viable schemes. It further narrows down the choices by filtering out the schemes that are not worth transitioning to when the transition IO and IO constraints are accounted for.

Selection criteria for viable schemes. Each viable redundancy scheme has to satisfy the following criteria in addition to the reliability constraint:

1. must satisfy the minimum number of simultaneous failures per stripe (i.e., $n - k$).
2. must not exceed the maximum allowed stripe dimension (k).

3. must have its expected failure reconstruction IO ($AFR \times k \times \text{disk-capacity}$) be no higher than was assumed possible for Rgroup0 (since disks in Rgroup0 are expected to have the highest AFR).
4. must have a recovery time in case of failure (MTTR) that does not exceed the maximum MTTR (set by the administrator when selecting the default redundancy scheme for Rgroup0).

Determining if a scheme is worth transitioning to. Whether the IO cost of transitioning to a scheme is worth it or not and what space-savings can be achieved by that transition is a function of the number of days disks will remain in that scheme (also known as *disk-days*). This, in turn, depends on (1) when the disks enter the new scheme, and (2) how soon disks will require another transition out of that scheme.

The time it takes for the disks to enter the new scheme is determined by the transition IO and the rate-limit. When the disks will transition out of the target Rgroup is dependent on the future and can only be estimated. For this estimation, the Rgroup-planner needs to estimate the number of days the AFR curve will remain below the threshold that forces a transition out. This needs different strategies for the two deployment patterns (trickle and step).

Recall that Pacemaker knows the AFR curve for trickle-deployed disks (from the canaries) in advance. Recall that step-deployed disks have the property that the AFR curve learned from them is statistically robust and tends to exhibit gradual, as opposed to sudden AFR increases. The Rgroup-planner leverages these properties to estimate the future AFR behavior based on the recent past. Specifically, it takes the slope of the AFR curve in the recent past¹ and uses that to project the AFR curve rise in the future.

The number of disk-days in a scheme for it to be worth transitioning to is dictated by the IO constraints. For example, let us consider a disk running under Pacemaker that requires a transition, and Pacemaker is configured with an average-IO constraint of 1% and a peak-IO-cap of 5%. Suppose the disk requires 1 day to complete its transition at 100% IO bandwidth. With the current settings, Pacemaker will only consider an Rgroup worthy of transitioning to (assuming it is allowed to use all 5% of its IO bandwidth) if at least 80 disk-days are spent after the disk entirely transitions to it (since transitioning to it would take up to 20 days at the allowed 5% IO bandwidth).

From among the viable schemes that are worth transitioning to based on the IO constraints, the Rgroup-planner chooses the one that provides the highest space-savings.

Decision on Rgroup creation.

Rgroups cannot be created arbitrarily. This is because every Rgroup adds *placement restrictions*, since all chunks of a stripe have to be stored on disks belonging to the same Rgroup. Therefore, Rgroup-planner creates a new Rgroup only when (1) the resulting placement pool created by the new Rgroup is large enough to overcome traditional placement restrictions such

¹Pacemaker uses a 60 day (configurable) sliding window with an Epanechnikov kernel, which gives more weight to AFR changes in the recent past [36].

as “no two chunks on the same rack”², and (2) the space-savings achievable by the chosen redundancy scheme is sufficiently greater than using an existing (less-space-efficient) Rgroup.

The disk deployment pattern also affects Rgroup formation. While the rules for whether to form an Rgroup remain the same for trickle and step-deployed disks, mixing disks deployed differently impacts the transitioning techniques that can be used for eventually transitioning disks out of that Rgroup. This in turn affects how the IO constraints are enforced. Specifically, for trickle deployments, creating an Rgroup for each set of transitioning disks would lead to too many small-sized Rgroups. So, for trickle-deployments, the Rgroup-planner creates a new Rgroup for a redundancy scheme if and only if one does not exist already. Creating Rgroups this way will also ensure that enough disks (thousands) will go into it to satisfy placement restrictions. Mixing disks from different trickle-deployments in the same Rgroup does not impact the IO constraints, because Pacemaker optimizes the transition mechanism for few disks transitioning at a time, as is explained in §5.4.3. For step-deployments, due to the large fraction of disks that undergo transition together, having disks from multiple steps, or mixing trickle-deployed disks within the same Rgroup, creates adverse interactions (discussed in §5.4.3). Hence, the Rgroup-planner creates a new Rgroup for each step-deployment, even if there already exists one or more Rgroups that employ the chosen scheme. Each such Rgroup will contain many thousands of disks to overcome traditional placement restrictions. Per-step Rgroups also extend to the Rgroup with default redundancy schemes, implying a per-step Rgroup0. Despite having clusters with disk populations as high as 450K disks, Pacemaker’s restrained Rgroup creation led to no cluster ever having more than 10 Rgroups.

Rules for purging an Rgroup.

An Rgroup may be purged for having too few disks. This can happen when too many of its constituent disks transition to other Rgroups, or they fail, or they are decommissioned leading to difficulty in fulfilling placement restrictions. If the Rgroup to be purged is made up of trickle-deployed disks, the Rgroup-planner will choose to RUp transition disks to an existing Rgroup with higher redundancy while meeting the IO constraints. For step-deployments, purging implies RUp transitioning disks into the more-failure-tolerant RGroup (RGroup0) that may include trickle-deployed disks.

5.4.3 Transition-executor

The transition-executor’s role is to determine *how to transition the disks*. This involves choosing (1) the most IO-efficient technique to execute that transition, and (2) how to rate-limit the transition at hand. Once the transition technique is chosen, the transition-executor executes the transition via the rate-limiter as shown in Fig. 5.2.

Selecting the transition technique

Suppose the data needs to be conventionally re-encoded from a k_{cur} -of- n_{cur} scheme to a k_{new} -of- n_{new} scheme. The IO cost of conventional re-encoding involves reading-re-encoding-

²Inter-cluster fault tolerance remains orthogonal to and unaffected by Pacemaker.

writing all the stripes whose chunks reside on each transitioning disk. This amounts to a read IO of $k_{cur} \times$ disk-capacity (assuming almost-full disks), and a write IO of $k_{cur} \times$ disk-capacity $\times \frac{n_{new}}{k_{new}}$ for a total IO $> 2 \times k_{cur} \times$ disk-capacity for each disk.

In addition to conventional re-encoding, Pacemaker supports two new approaches to changing the redundancy scheme for disks and selects the most efficient option for any given transition. The best option depends on the fraction of the Rgroup being transitioned at once.

Type 1 (Transition by emptying disks). If a small percentage of an Rgroup’s disks are being transitioned, it is more efficient to retain the contents of the transitioning disks in that Rgroup rather than re-encoding. Under this technique, the data stored on transitioning disks are simply moved (copied) to other disks within the current Rgroup. This involves reading and writing (elsewhere) the contents of the transitioning disks. Thus, the IO of transitioning via Type 1 is at most $2 \times$ disk-capacity, independent of scheme parameters, and therefore at least $k_{cur} \times$ cheaper than conventional re-encoding.

Type 1 can be employed whenever there is sufficient free space available to move the contents of the transitioning disks into other disks in the current Rgroup. Once the transitioning disks are empty, they can be removed from the current Rgroup and added to the new Rgroup as “new” (empty) disks.

Type 2 (Bulk transition by recalculating parities). If a large fraction of disks in an Rgroup need to transition together, it is more efficient to transition the entire Rgroup rather than only the disks that need a transition at that time. Most cluster storage systems use systematic codes³ [16, 27, 29, 52], wherein transitioning an entire Rgroup involves only calculating and storing new parities and deleting the old parities. Specifically, the data chunks have to be only read for computing the new parities, but they do not have to be re-written. In contrast, if only a part of the disks are transitioned, some fraction of the data chunks also need to be re-written. Thus, the IO cost for transitioning via Type 2 involves a read IO of $\frac{k_{cur}}{n_{cur}} \times$ disk-capacity, and a write IO of only the new parities, which amounts to a total IO of $\frac{n_{new}-k_{new}}{k_{new}} \times \frac{k_{cur}}{n_{cur}} \times$ disk-capacity for each disk in the Rgroup. This is at most $2 \times \frac{k_{cur}}{n_{cur}} \times$ disk-capacity, which makes it at least $n_{cur} \times$ cheaper than conventional re-encoding.

Selecting the most efficient approach for a transition

For any given transition, the transition-executor selects the most IO-efficient of all the viable approaches. Almost always, trickle-deployed disks use Type 1 because they transition a-few-at-a-time, and step-deployed disks use Type 2 because Rgroup-planner maintains each step in a separate Rgroup.

Choosing how to rate limit a transition

Irrespective of the transitioning techniques, the transition-executor has to resolve the competing concerns of maximizing space-savings and minimizing risk of data loss via fast transitions,

³In systematic codes, the data chunks are stored in unencoded form. This helps to avoid having to decode for normal (i.e., non-degraded-mode) reads.

and minimizing foreground work interference by slowing down transitions so as to not overwhelm the foreground IO. Arbitrarily slowing down a transition to minimize interference is only possible when the transition is not in response to a rise in AFR. This is because a rising AFR hints at the data being under-protected if not transitioned to a higher redundancy soon. In Pacemaker, a transition without an AFR rise occurs either when disks are being RDn transitioned at the end of infancy, or when they are being RUp transitioned because the Rgroup they belong to is being purged. For all the other RUp transitions, Pacemaker carefully chooses how to rate limit the transition.

Determining how much bandwidth to allow for a given transition could be difficult, given that other transitions may be in-progress already or may be initiated at any time (we do observe concurrent transitions in our evaluations). So, to ensure that the aggregate IO of all ongoing transitions conforms to the peak-IO-cap cluster-wide, Pacemaker limits each transition to the peak-IO-cap within its Rgroup. For trickle-deployed disks, which share Rgroups, the rate of transition initiations is consistently a small percentage of the shared Rgroup, allowing disk emptying to proceed at well below the peak-IO-cap. For step-deployed disks, this is easy for Pacemaker, since a step only makes one transition at a time and its IO is fully contained in its separate Rgroup. The transition-executor’s approach to managing peak-IO on a per-Rgroup basis is also why the proactive-transition-initiator can safely assume a rate-limit of the peak-IO-cap without consulting the transition-executor. If there is a sudden AFR increase that puts data at risk, Pacemaker is designed to ignore its IO constraints to continue meeting the reliability constraint—this safety valve was never needed for any cluster evaluated.

After finalizing the transitioning technique, the transition-executor performs the necessary IO for transitioning disks (read, writes, parity recalculation, etc.). We find that the components required for the transition-executor are already present and adequately modular in existing distributed storage systems. In §6, we show how we implement Pacemaker in HDFS with minimal effort.

Note that this design is for the common case where storage clusters are designed for a single dedicated storage service. Multiple distinct distributed storage services independently using the same underlying devices would need to coordinate their use of bandwidth (for their non-transition related load as well) in some way, which is outside the scope of this dissertation.

5.5 Evaluation

Pacemaker-enabled disk-adaptive redundancy using is evaluated on production logs from four large-scale real-world storage clusters, each with hundreds of thousands of disks. We also experiment with a proof-of-concept HDFS implementation on a smaller sized cluster. This evaluation has four primary takeaways: (1) Pacemaker eliminates transition overload, never using more than 5% of cluster IO bandwidth (0.2–0.4% on average) and always meets target MTTDL, in stark contrast to prior work approaches that do not account for transition IO load; (2) Pacemaker provides more than 97% of idealized-potential space-savings, despite being proactive, reducing disk capacity needed by 14–20% compared to one-size-fits-all; (3) Pacemaker’s behavior is not overly sensitive across a range of values for its configurable parameters; (4) Pacemaker copes well with the real-world AFR characteristics explained in

Chapter 4. For example, it successfully combines the “multiple useful-life phases” observation with efficient transitioning schemes. This evaluation also shows Pacemaker in action by measuring disk-adaptive redundancy in Pacemaker-enhanced HDFS.

Evaluation methodology.

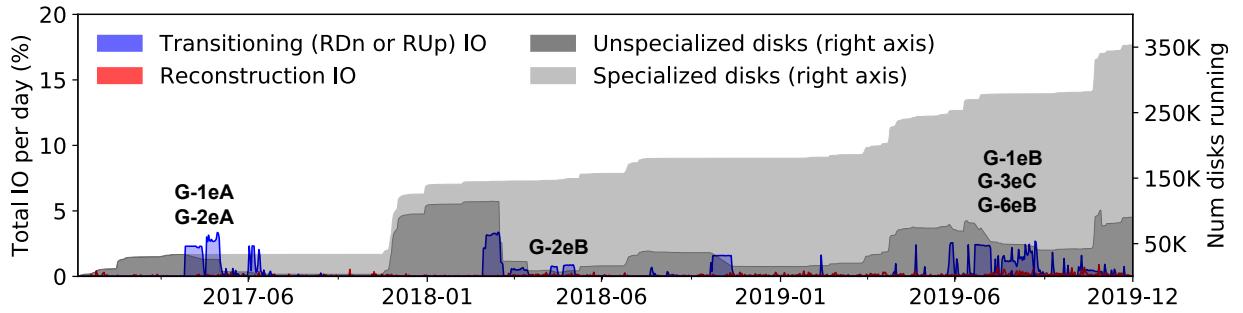
Pacemaker is simulated chronologically for each of the four cluster logs described in §4: three clusters from Google and one from Backblaze. For each simulated date, the simulator changes the cluster composition according to the disk additions, failures and decommissioning events in the log. Pacemaker is provided the log information, as though it were being captured live in the cluster. IO bandwidth needed for each day’s redundancy management is computed as the sum of IO for failure reconstruction and transition IO requested by Pacemaker, and is reported as a fraction of the configured cluster IO bandwidth (100MB/sec per disk, by default).

Pacemaker was configured to use a peak-IO-cap of 5%, an average-IO constraint of 1% and a threshold-AFR of 75% of the tolerated-AFR, except for the sensitivity studies in §5.5.3. For comparison, we also simulate (1) an idealized disk-adaptive redundancy system in which transitions are instantaneous (requiring no IO) and (2) the prior state-of-the-art approach (HeART) for disk-adaptive redundancy. For all cases, Rgroup0 uses 6-of-9, representing a one-size-fits-all scheme reported in prior literature [22]. The required target MTTDL is then back-calculated using the 6-of-9 default and an assumed tolerated-AFR of 16% for Rgroup0. These configuration defaults were set by consulting storage administrators of clusters we evaluated.

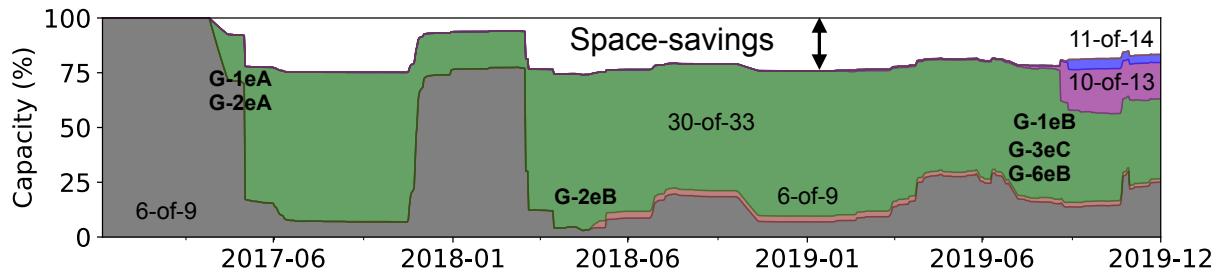
5.5.1 Pacemaker on Google Cluster1 in-depth

Fig. 5.3(a) shows the IO generated by Pacemaker (and disk count) over the \approx 3-year lifetime of Google Cluster1. Over time, the cluster grew to over 350K disks comprising of disks from 7 makes/models (Dgroups) via a mix of trickle and step deployments. Fig. 5.3(c) and Fig. 5.3(d) show AFR curves of 2 of the 7 Dgroups (obfuscated as G-1 and G-2 for confidentiality) along with how Pacemaker adapted to them at each age. G-1 disks are trickle-deployed whereas G-2 disks are step-deployed. The other 5 Dgroups are omitted due to lack of space. Fig. 5.3(b) shows the corresponding space-savings (the white space above the colors).

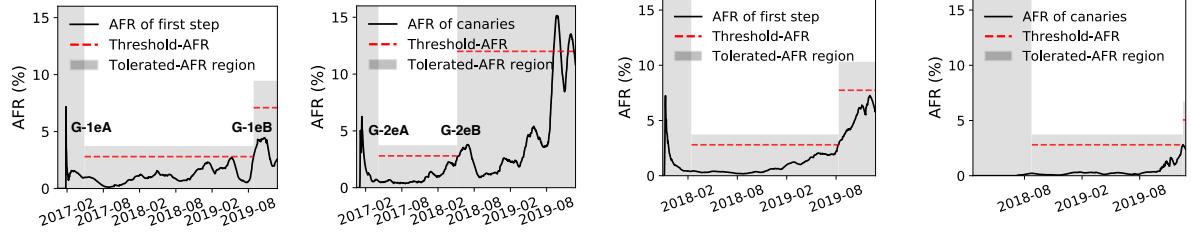
All disks enter the cluster as unspecialized disks, i.e. Rgroup0 (dark gray region in the Fig. 5.3(a) and left gray region of Figs. 5.3(c) and 5.3(d)). Once a Dgroup’s AFR reduces sufficiently, Pacemaker RDn transitions them to a specialized Rgroup (light gray area in Fig. 5.3(a)). Over their lifetime, disks may transition through multiple RUp transitions over the multiple useful-life phases. Each transition requires IO, which is captured in blue in Fig. 5.3(a). For example, the sudden drop in the unspecialized disks, and the blue area around 2018-04 captures the Type 2 transitions caused when over 100K disks RDn transition from Rgroup0 to a specialized Rgroup. The light gray region in Fig. 5.3(a) corresponds to the time over which space-savings are obtained, which can be seen in Fig. 5.3(b).



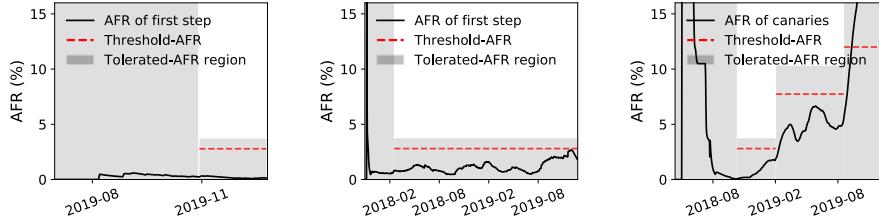
(a) Redundancy management IO due to Pacemaker over its 2.5+ year lifetime broken down by IO type. This is identical to Fig. 5.1(b) with the left Y-axis only going to 20% to show the detailed IO activity in the cluster.



(b) Space-savings due to Pacemaker. Each colored region represents the fraction of cluster capacity that is using a particular redundancy scheme. 6-of-9 is the default redundancy scheme (Rgroup0's).



(c) G-1 (step) AFR curve (d) G-2 (trickle) AFR curve (e) G-6 (step) AFR curve (f) G-4 (trickle) AFR curve



(g) G-8 (step) AFR curve (h) G-5 (step) AFR curve (i) G-3 (trickle) AFR curve

Figure 5.3: Detailed IO analysis and space savings achieved by Pacemaker-enabled adaptive redundancy on Google Cluster1.

Many transitions with no transition overload

Pacemaker successfully bounds all redundancy management IO comfortably under the configured peak-IO-cap throughout the cluster’s lifetime. This can be seen via an imaginary horizontal line at 5% (the configured peak-IO-cap) that none of the blue regions goes above. Recall that Pacemaker rate-limits the IO within each Rgroup to ensure simultaneous transitions do not violate the cluster’s IO cap. Events **G-1eA** and **G-2eA** are examples of events where both G-1 and G-2 disks (making up almost 100% of the cluster at that time) request transitions at the same time. Despite that, the IO remains bounded below 5%. **G-3eC** and **G-6eB** also show huge disk populations of G-3 and G-6 Dgroups (AFRs not shown) requesting almost simultaneous RUp transitions, but Pacemaker’s design ensures that the peak-IO constraint is never violated. This is in sharp contrast with HeART’s frequent transition overload, shown in Fig. 5.1(a).

Disks experience multiple useful-life phases

G-1, G-3, G-6 and G-7 disks experience two phases of useful life each. In Fig. 5.3(a), events **G-1eA** and **G-1eB** mark the two transitions of G-1 disks through its multiple useful lives as shown in Fig. 5.3(c). In the absence of multiple useful-life phases, Pacemaker would have RUp transitioned G-1 disks to Rgroup0 in 2019-05, eliminating space-savings for the remainder of their time in the cluster. §5.5.3 quantifies the benefit of multiple useful-life phases for all four clusters.

MTTDL always at or above target

Along with the AFR curves, Figs. 5.3(c) and 5.3(d) also show the upper bound on the AFR for which the reliability constraint is met (top of the gray region). Pacemaker sufficiently protects all disks throughout their life for all Dgroups across evaluated clusters.

Substantial space-savings achieved

Pacemaker provides 14% average space-savings (Fig. 5.3(b)) over the cluster lifetime to date. Except for 2017-01 to 2017-05 and 2017-11 to 2018-03, which correspond to infancy periods for large batches of new empty disks added to the cluster, the entire cluster achieves \approx 20% space-savings. Note that the apparent reduction in space-savings from 2017-11 to 2018-03 isn’t actually reduced space in absolute terms. Since Fig. 5.3(b) shows relative space-savings, the over 100K disks deployed around 2017-11, and their infancy period makes the space-savings appear reduced relative to the size of the cluster.

5.5.2 Pacemaker on the other three clusters

Figs. 5.4(a), 5.5(a), 5.6(a) compares the transition IO incurred by Pacemaker to that for HeART [4] for Google Cluster2, Google Cluster3 and Backblaze respectively, along with the corresponding space-savings achieved by Pacemaker. While clusters using HeART would suffer transition overload, the same clusters under Pacemaker always had all their transition

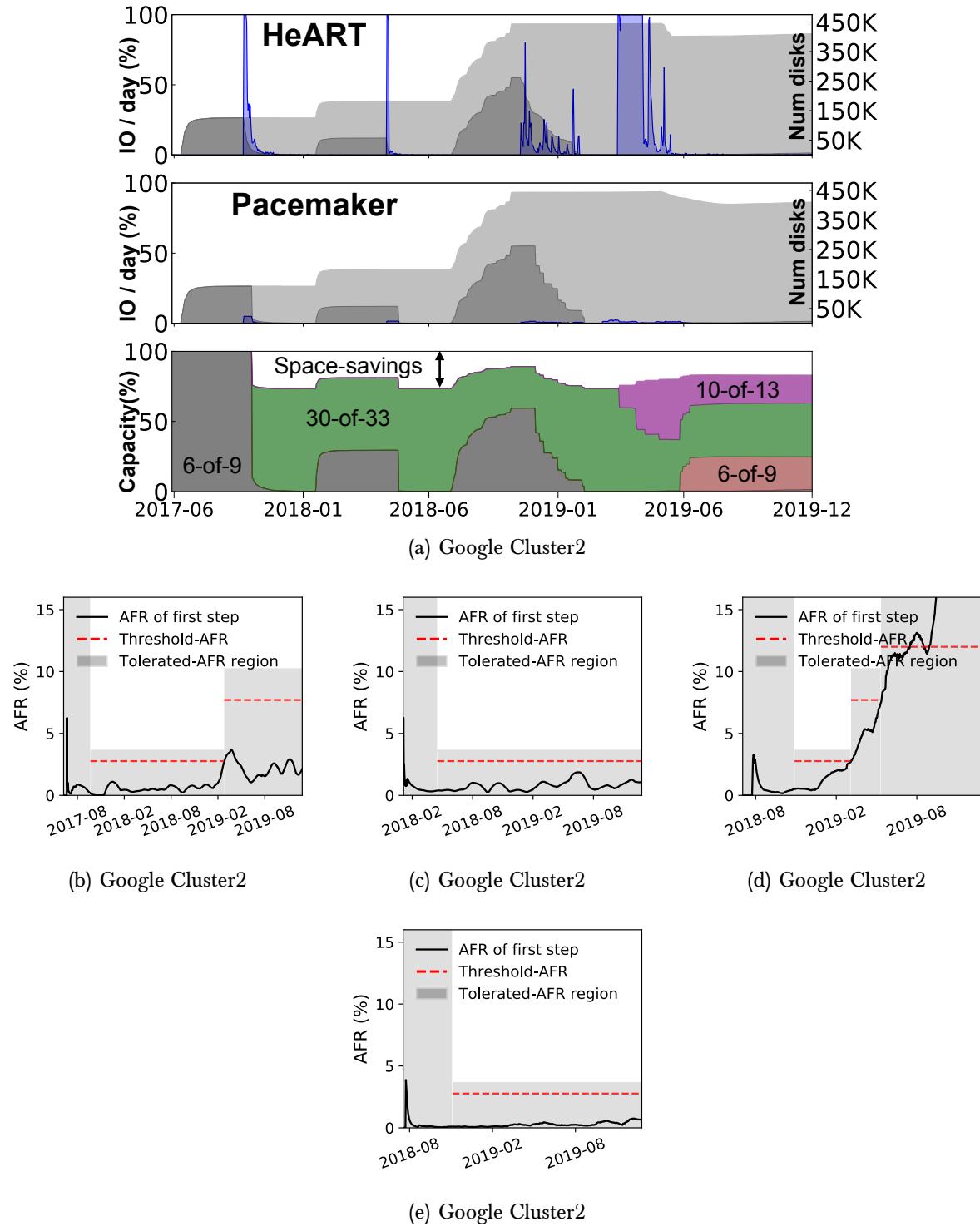
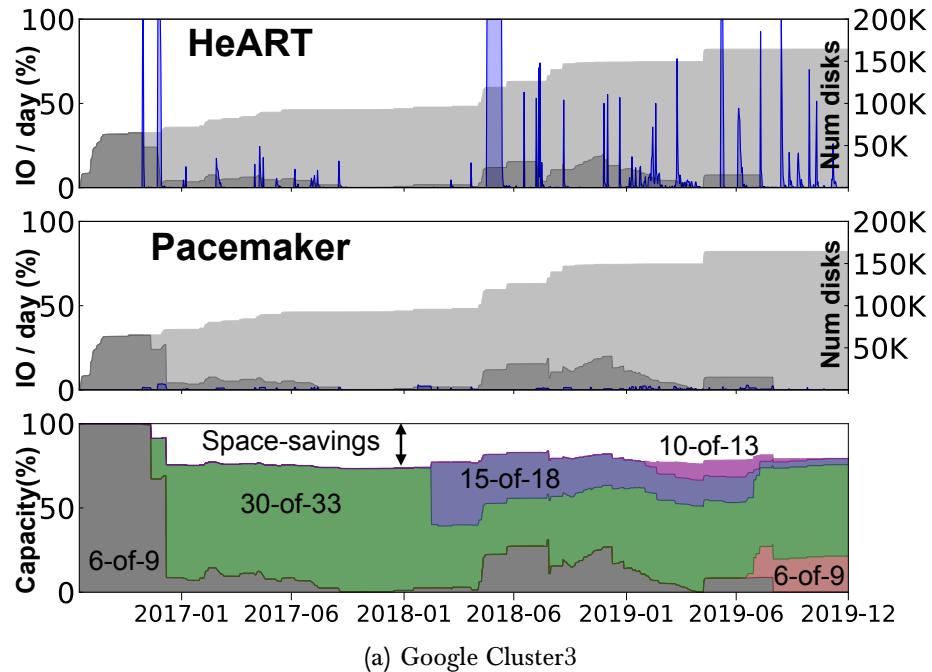
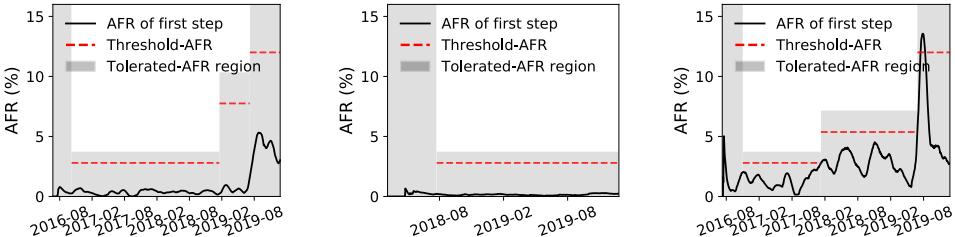


Figure 5.4: Google Cluster2 transition IO, space-savings and individual Dgroup AFR curves.



(a) Google Cluster3

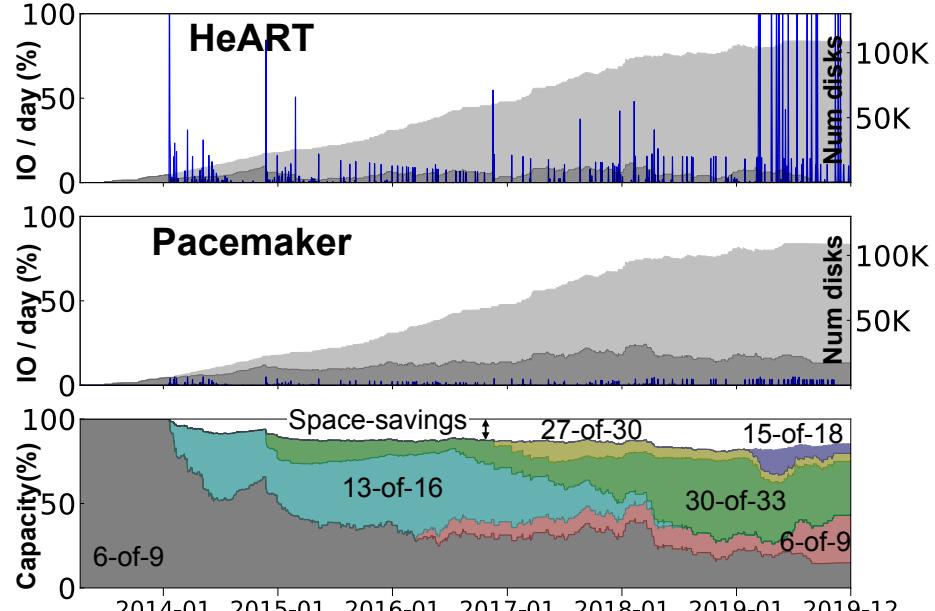


(b) G-2 (step) AFR curve

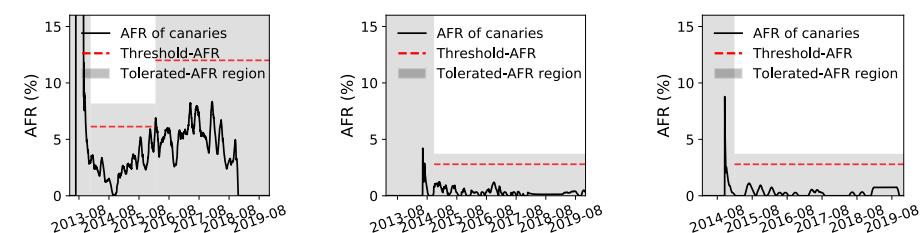
(c) G-4 (step) AFR curve

(d) G-1 (step) AFR curve

Figure 5.5: Google Cluster3 transition IO, space-savings and individual Dgroup AFR curves.



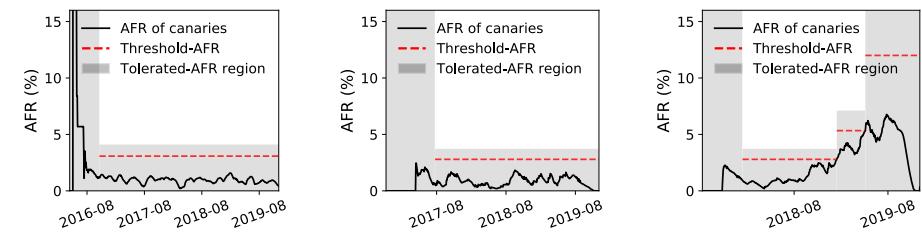
(a) Backblaze Cluster



(b) S-4 (trickle)

(c) H-4A (trickle)

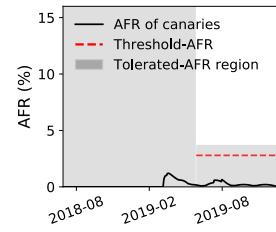
(d) H-4B (trickle)



(e) S-8C (trickle)

(f) S-8E (trickle)

(g) S-12E (trickle)



(h) H-12E (trickle)

Figure 5.6: Backblaze cluster transition IO, space-savings and individual Dgroup AFR curves.

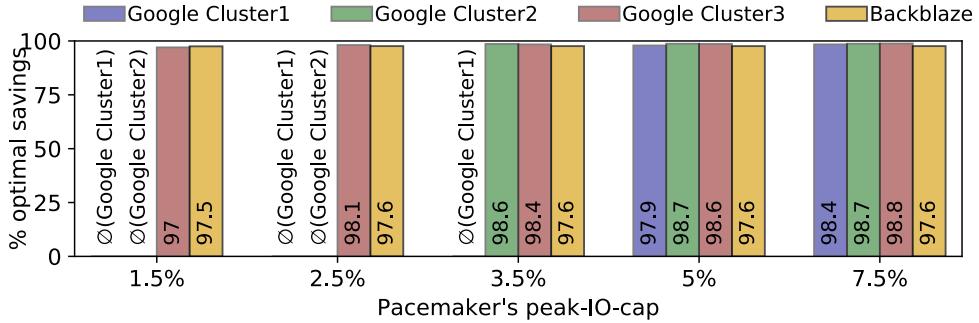


Figure 5.7: Pacemaker’s sensitivity to the peak-IO constraint.

IO under the peak-IO-cap of 5%. In fact, on average, only 0.21–0.32% percent of the cluster IO bandwidth was used for transitions. The average space-savings for the three clusters are 14–20%.

Google Cluster2

Fig. 5.4(a) shows the transition overload and space-savings in Google Cluster2 and the corresponding space-savings. All Dgroups in Google Cluster2 are step-deployed. Thus, it is not surprising that Fig. 5.9 shows that over 98% of the transitions in Cluster2 were Type 2 transitions (bulk parity recalculation). Cluster2’s disk population exceeds 450K disks. Even at such large scales, Pacemaker obtains average space-savings of almost 17% and peak space-savings of over 25%. This translates to needing 100K fewer disks.

Google Cluster3

Google Cluster3 (Fig. 5.4(a)) is not as large as Cluster1 or Cluster2. At its peak, Cluster3 has a disk population of approximately 200K disks. But, it achieves the highest average space-savings (20%) among clusters evaluated. Like Cluster2, Cluster3 is also mostly step-deployed.

Backblaze Cluster

Backblaze (Fig. 5.6(a)) is a completely trickle-deployed cluster. The dark grey region across the bottom of Fig. 5.6(a)’s Pacemaker plot shows the persistent presence of canary disks throughout the cluster’s lifetime. Unlike the Google clusters, the transition IO of Backblaze does not produce bursts of transition IO that lasts for weeks. Instead, since trickle-deployed disks transition a-few-at-a-time, we see transition work appearing continuously throughout the cluster lifetime of over 6 years. The rise in the transition IO spikes in 2019, for HeART, is because of large capacity 12TB disks replacing 4TB disks. Unsurprisingly, under Pacemaker, most of the transitions are done using Type 1 (transitioning by emptying disks) as shown in Fig. 5.9. The average space-savings obtained on Backblaze are 14%.

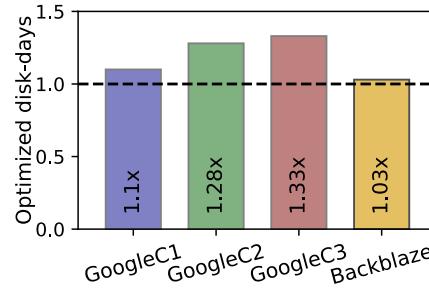


Figure 5.8: Multiple useful-life phases

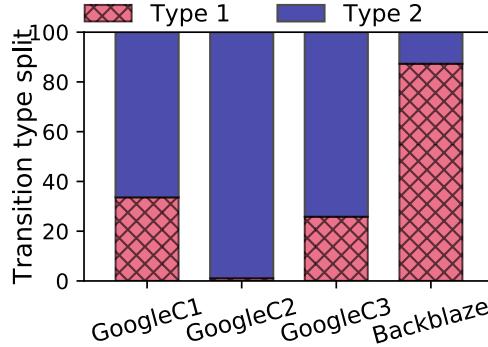


Figure 5.9: Transition type distribution

5.5.3 Sensitivity analyses and ablation studies

Sensitivity to IO constraints

The peak-IO constraint governs Fig. 5.7, which shows the percentages of optimal space-savings achieved with Pacemaker for peak-IO-cap settings between 1.5% and 7.5%. A peak-IO-cap of up to 7.5% is used in order to compare with the IO percentage spent for existing background IO activity, such as scrubbing. By scrubbing all data once every 15 days [12], the scrubber uses around 7% IO bandwidth, and is a background work IO level tolerated by today’s clusters.

The Y-axis captures how close the space-savings are for the different peak-IO-caps compared to “Optimal savings”, i.e. an idealized system with infinitely fast transitions. Pacemaker’s default peak-IO-cap (5%) achieves over 97% of the optimal space-savings for each of the four clusters. For peak-IO constraint set to $\leq 2.5\%$, some RUp transitions in Google Cluster1 and Cluster2 become too aggressively rate-limited causing a subsequent AFR rise to violate the peak-IO constraints. We indicate this as a failure, and show it as “”. The same situation happens for Google Cluster1 at 3.5%.

Sensitivity to threshold-AFR

The threshold-AFR determines when proactive RUp transitions of step-deployed disks are initiated. Conceptually, the threshold-AFR governs how risk-averse the admin wants to be. Lowering the threshold would trigger an RUp transition when disks are farther away from the tolerated-AFR (more risk-averse), and vice-versa. We evaluated Pacemaker for threshold-AFRs

of 60%, 75% and 90% of the respective Rgroups' tolerated-AFRs. We found that Pacemaker's space-savings is not very sensitive to threshold-AFR, with space-savings only 2% lower at 60% than at 90%. Data remained safe at each of these settings, but would become unsafe with higher values.

Contribution of multiple useful-life phases

Fig. 5.8 compares the increased number of disk-days spent in specialized Rgroups because of considering multiple useful-life phases. In the best case, Google Cluster2 spent 33% more disk-days in specialized redundancy, increasing overall space-savings from 16% to 19%. Note that in large-scale storage clusters, even 1% space-savings are considered substantial as it represents thousands of disks.

Contribution of transition types

By proactively keeping step-deployed disks in distinct Rgroups and using specialized transitioning schemes whenever possible, instead of using simple re-encoding for all transitions, Pacemaker reduces total transition IO by 92–96% for the four clusters. Fig. 5.9 shows what percentage of transitions were done via Type 1 (disk emptying) vs. Type 2 (bulk parity recalculation). As expected, Google clusters rely more on Type 2 transitions, because most disks are step-deployed. In contrast, the Backblaze cluster is entirely trickle-deployed and hence mostly uses Type 1 transitions. The small percentage of Type 2 transitions in Backblaze occur when Rgroups are purged.

5.6 Related work

The closest related work [41] proposes a redundancy adaptation tool called HeART that categorizes disks into groups and suggests a tailored redundancy scheme for each during its useful life period. As discussed earlier, while [41] showcased potential space-savings, it ignored transition overload and hence is made impractical (Fig. 5.1(a)). Pacemaker eliminates transition overload by employing IO constraints (specifically the peak-IO and average-IO constraints) that cap the transition IO to a tiny fraction cluster bandwidth. While HeART was evaluated only for the trickle-deployed Backblaze cluster, our evaluation of Pacemaker for Google storage clusters exposes the unique challenges of step-deployed clusters. Several design elements were added to Pacemaker to address the challenges posed by step-deployed disks.

Various systems include support for multiple redundancy schemes, allowing different schemes to be used for different data [24, 20]. Tools have been created for deciding, on a per-data basis, which scheme to use [79, 87]. Keeton et al. [42] describe a tool that automatically provides disaster-resistant solutions based on budget and failure models. Pacemaker differs from such systems by focusing on efficiently adapting redundancy to different and time-varying AFRs of disks.

Reducing the impact of background IO, such as for data scrubbing, on foreground IO is a common research theme. [5, 8, 45, 46, 54, 71]. Pacemaker converts otherwise-urgent bursts of transition IO into proactive background IO, which could then benefit from these works.

Disk reliability has been well studied, including evidence of failure rates being make/model dependent [12, 21, 37, 40, 47, 56, 57, 67, 68, 69, 74]. There are also studies that predict disk failures [6, 35, 48, 53, 77, 84, 92], which can enhance any storage fault-tolerance approach.

While several works have considered the problem of designing erasure codes that allow transitions using less resources, existing solutions are limited to specific kinds of transitions and hence are not applicable in general. The case of adding parity chunks while keeping the number of data chunks fixed can be viewed [51, 59, 63] as the well-studied reconstruction problem, and hence the codes designed for optimal reconstruction (e.g., [20, 33, 55, 60, 63, 83]) would lead to improved resource usage for this case. Several works have studied the case where the number of data nodes increases while the number of parity nodes remains fixed [38, 58, 86, 90, 93]. In [87], the authors propose two erasure codes designed to undergo a specific transition in parameters. In [49], the authors propose a general theoretical framework for studying codes that enable efficient transitions for general parameters, and derive lower bounds on the cost of transitions as well as describe optimal code constructions for certain specific parameters. However, none of the existing code constructions are applicable for the diverse set of transitions needed for disk-adaptive redundancy in real-world storage clusters.

Chapter 6

Implementing disk-adaptive redundancy in HDFS

We have implemented a prototype of Pacemaker for the Hadoop distributed file system (HDFS) [25]. HDFS is a popular open source distributed file system, widely employed in the industry for storing large volumes of data. We use HDFS v3.2.0, which natively supports erasure coding. Prototype of HDFS with Pacemaker is open-sourced and is available at <https://github.com/thesys-lab/pacemaker-hdfs.git>. We note that this prototype is solely from a functional perspective, and not from the performance perspective. The intent of implementing Pacemaker in HDFS is to show how existing components of a distributed storage system can be repurposed to realize a distributed storage system with disk-adaptive redundancy.

6.1 Background on HDFS architecture

HDFS has a central metadata server called Namenode (Namenode, akin to the master node) and a collection of servers containing the data stored in the file system, called Datanodes (Datanode, akin to worker nodes). Clients interact with the Namenode only to perform operations on file metadata (containing a collection of the Datanodes that store the file data). Clients directly request the data from the Datanodes. Each Datanode stores data on its local drives using a local file system.

6.1.1 Realizing Dgroups in HDFS

This design makes a simplifying assumption that all disks belonging to a Datanode are of the same Dgroup and are deployed together (this could be loosened easily). Under this simplifying assumption, conceptually, an Rgroup would consist of a set of Datanodes that need to be managed independent of other such sets of Datanodes as shown in Fig 6.1.

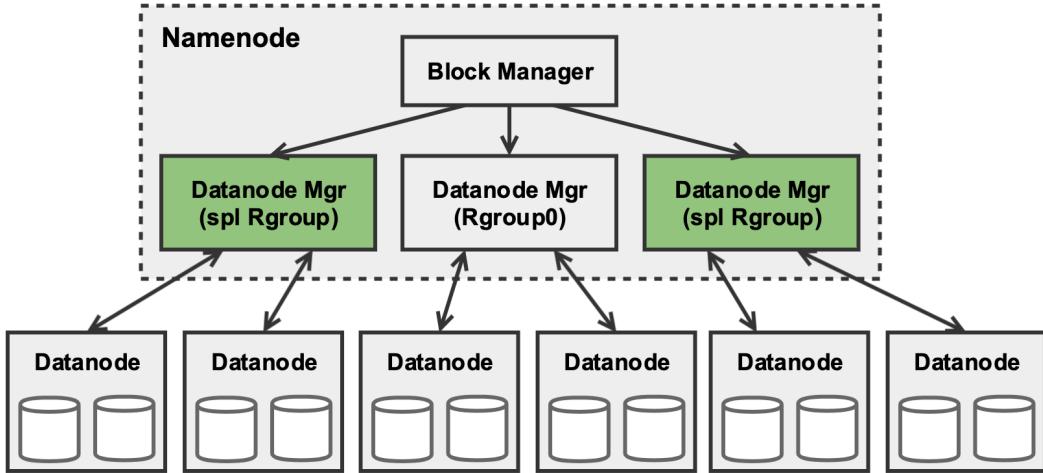


Figure 6.1: Pacemaker-enhanced HDFS architecture.

6.1.2 Realizing Rgroups in HDFS

The Namenode maintains a DatanodeManager, which is a gateway for the Namenode to interact with the Datanodes. The DatanodeManager maintains a list of the Datanodes, along with their usage statistics. The DatanodeManager also contains a HeartBeatManager which handles the periodic keepalive heartbeats from Datanodes. A natural mechanism to realize Rgroups in HDFS is to have one DatanodeManager per Rgroup. Note that the sets of Datanodes belonging to the different DatanodeManagers are mutually exclusive. Implementing Rgroups with multiple DatanodeManagers has several advantages.

6.2 Incorporating transition-executor in HDFS

6.2.1 Type 1 (transitioning by emptying disks)

An important part of Pacemaker functionality is transitioning Datanodes between Rgroups. Recall from §5.4.3 that one of Pacemaker's preferred way of transitioning trickle-deployed disks across Rgroups is by emptying the disks. In HDFS, the planned removal of a Datanode from a HDFS cluster is called decommissioning. Pacemaker re-uses decommissioning to remove a Datanode from the set of Datanodes managed by one DatanodeManager and then adds it to the set managed by another, effectively transitioning a Datanode from one Rgroup to another.

6.2.2 Type 2 (transitioning by bulk parity re-calculation)

Type 2 transitions are meant to transition the entire Rgroup from one redundancy scheme to another. Type 2 transitions are usually observed when transitioning step-deployed disks as mentioned in §5.4.3. Type 2 transitions in HDFS has two parts: add Datanodes in Rgroup A into Rgroup B and change the encoding scheme of files and directories in Rgroup A to that

of Rgroup B. There are some challenges in implementing this in HDFS. First, in the currently supported erasure coding version in HDFS (called “Striped”) HDFS data blocks are split into smaller “cells”. A collection of these cells across HDFS data blocks in different Datanodes form a stripe. This complicates the process of transitioning at the HDFS block level because doing so might affect multiple stripes together causing data inconsistency. Second, the transitioning of the entire file should reflect as though it happened atomically even though the actual transitioning happens stripe-by-stripe. Third, changing the erasure coding scheme should respect the block placement policy of the target Rgroup. Finally, throughout the transitioning process there should be continued forward progress, i.e. the transitioning and transitioned data should be able to be read by clients at all times.

To overcome those challenges, we come up with the following design. First, we set the cell size to be the HDFS block size. That means the whole block is a single cell, which simplifies the transitioning implementation, which (conceptually) can now happen at the HDFS block level. This is a reasonable assumption, and is only a stop-gap requirement since an erasure coding implementation where an entire HDFS block is an erasure coded block is proposed (as the “Contiguous” method) and is the development pipeline [28]. Second, we create temporary shadow inode using the snapshot feature in HDFS. That can ensure the atomicity of the entire operation and will not affect the client requests on that file during the transfer. Third, we only transfer those necessary blocks to form new stripes and respect the requirement of the block placement policy in target Rgroup. That can minimize the data transfer during Type 2 transitions. Fourth, we use the reconstruction mechanism to compute new parity blocks according to the erasure coding scheme in the target Rgroup. Finally, we update the metadata in Namenode and delete those blocks are not useful including old parity blocks and the old copy of transferred blocks.

6.2.3 Purging Rgroups

When a Dgroup’s Datanode population drops below a threshold, Pacemaker purges the Rgroup and moves the Datanodes to the next most space efficient Rgroup. The Rgroup purging process is conceptually the same as transitioning all Datanodes to another Rgroup as discussed above. A caveat is that because newly introduced Datanodes are empty when they join an Rgroup, bulk Datanode retirement may cause load imbalance. Pacemaker suggests explicitly suggesting calling the HDFS load balancer after every retirement (and even periodically if afforded), whose job is to balance the data between data-rich Datanodes to the data-poor Datanodes.

6.3 Designing IO constraints in Pacemaker

6.3.1 Handling the average-IO constraint

Recall from §5.3.4 that Pacemaker has a average-IO constraint that caps the average IO spent on redundancy transitions over the lifetime of a disk. The average-IO constraint can be easily incorporated into HDFS through the DatanodeManager. Each DatanodeManager maintains

per-Datanode metrics such as the amount of IO performed, the load (im)balance, etc. By adding “transition-IO” as another metric that is tracked by the DatanodeManager, we can easily keep track of how much IO is spent over the lifetime of a Datanode (and thus the disks attached to that Datanode).

6.3.2 Handling the peak-IO constraint

The large data movement involved in both Type 1 and Type 2 result in significant use of the network bandwidth. Recall from §~~6.3.4~~ that Pacemaker’s design includes a peak-IO constraint that puts a cap on the rate at which transitions can happen so as to minimize interference with foreground IO traffic, which if not done could negatively affect both throughput and latency.

To implement the peak-IO constraint in HDFS, this design uses its wait-based data transfer throttler, which is originally used in block data sending and receiving, journaling, and image transferring. To make use of the throttler in Type 1 and Type 2 transitions, a custom decorator is developed on top of Java’s output stream which has a rate-limiter embedded in it. This design wraps the output stream that is used to perform Type 1 and Type 2 transitions by this custom throttled decorator. Finally, given a configured peak-IO cap, the throttled output stream can automatically control the data transfer speed to fulfill the configured bandwidth requirement to meet the peak-IO constraint and ensure predictable performance.

6.4 Salient features of this architecture

6.4.1 Right level of control and view of the system.

Since the DatanodeManager resides below the block layer, when the data needs to be moved for redundancy adaptations, the logical view of the file remains unaffected. Only the mapping from HDFS blocks to Datanodes gets updated in the inode. The statistics maintained by the DatanodeManager can be used to balance load across Rgroups.

6.4.2 Minimizing changes to the HDFS architecture and maximizing re-purposing of existing HDFS mechanisms

This design obviates the need to change HDFS’s block placement policy, since it is implemented at the DatanodeManager level. Block placement policies are notoriously hard to get right. Moreover, block placement decisions are affected by fault domains and network topologies, both of which are orthogonal to the goals of Pacemaker, and thus best left untouched. Likewise, the code for reconstruction of data from a failed Datanode need not be touched, since all of the reads (to reconstruct each lost chunk) and writes (to store it somewhere else) will occur within the set of nodes managed by its DatanodeManager. Existing mechanisms for adding and decommissioning nodes managed by the DatanodeManager can be re-purposed for Pacemaker’s Type 1 transitions (described below).

6.4.3 Flexible Rgroup-specific customizations are possible

This design allows for flexibility to add Rgroup-specific functionality in the future. For example, reliability tiering could be performed by allowing sets of Rgroups have different reliability targets. No architectural change would have to be done in order to enable this functionality.

6.4.4 Cost of maintaining multiple DatanodeManagers is small

Each DatanodeManager maintains two threads: a HeartbeatManager and a DatanodeAdmin-Manager. The former tracks and handles heartbeats from each Datanode, and the latter monitors the Datanodes for performing decommissioning and maintenance. The number of DatanodeManager threads in the Namenode will increase from two to $2 \times$ the number of Rgroups. Fortunately, even for large clusters, we observe that the number of Rgroups would not exceed the low tens (§6.5). The Namenode is usually a high-end server compared to the Datanodes, and an additional tens of threads shouldn't affect performance.

6.4.5 File access interface remains unchanged

Pacemaker does not change the file manipulation API or client access paths. But, there is one corner-case related to transitions when file reads can be affected internally. To read a file, a client queries the Namenode for the inode and caches it. Subsequently, the reads are performed directly from the client to the Datanode. If the Datanode transitions to another Rgroup while the file is still being read, the HDFS client may find that that Datanode no longer has the requested data. But, because this design uses existing HDFS decommissioning for transitions, the client software knows to react by re-requesting the updated inode from the Namenode and resuming the read.

6.5 Evaluating HDFS + Pacemaker

This section describes basic experiments with the Pacemaker-enabled HDFS, focusing on its functioning and operation. Note that Pacemaker is designed for longitudinal disk deployments over several years, a scenario that cannot be reproduced identically in laboratory settings. Hence, these HDFS experiments are aimed to display that integrating Pacemaker with an existing storage system is straightforward, rather than on the long-term aspects like overall space-savings or transition IO behavior over cluster lifetime as evaluated via simulation above.

The HDFS experiments run on a PRObE Emulab cluster [31]. Each machine has a Dual-Core AMD Opteron Processor, 16GB RAM, and Gigabit Ethernet. We use a 21-node cluster running HDFS 3.2.0 with one Namenode and 20 Datanodes. Each Datanode has a 10GB partition on a 10000 RPM HDD for a total cluster size of 200GB. We statically define the cluster to be made up of two Rgroups of ten Datanodes each, one using the 6-of-9 erasure coding scheme and the other using a 7-of-10 scheme. DFS-perf [34], a popular open-source HDFS benchmark is used, after populating the cluster to 60% full. Each DFS-perf client sequentially reads one file over and over again (size=768MB), for a total read size of about

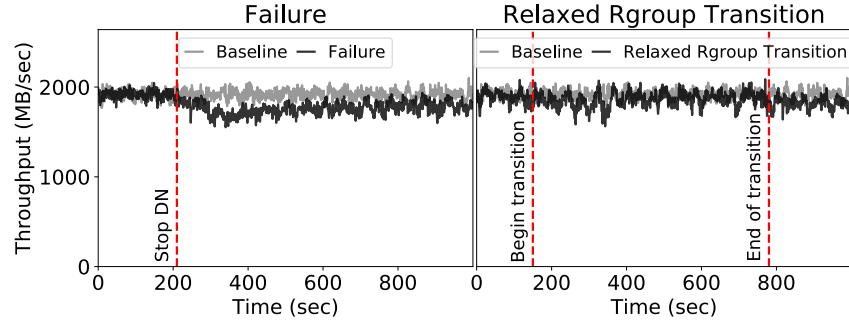


Figure 6.2: DFS-perf reported throughput for baseline, with one Datanode failure and one Rgroup transition.

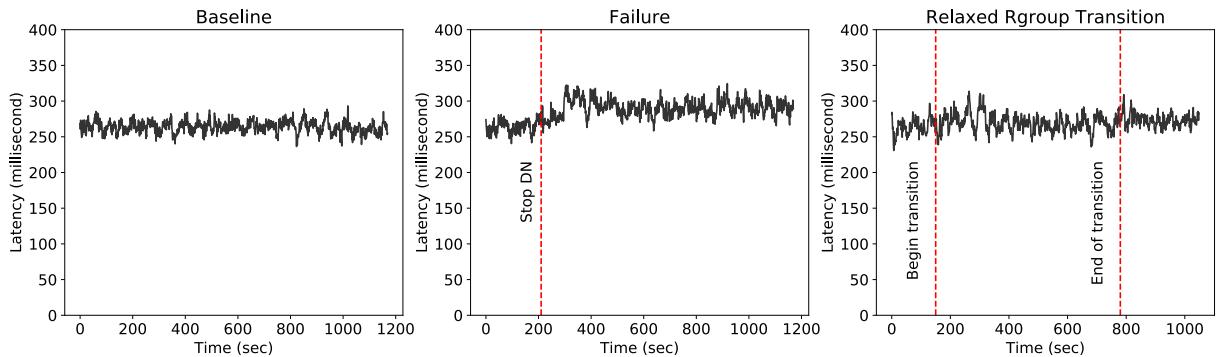


Figure 6.3: DFS-perf reported latency for baseline, with one Datanode failure and one Rgroup transition.

1.75TB over 40 iterations. We use 60 DFS-perf clients, running on 20 nodes separate from the HDFS cluster.

We focus on the behavior of a Datanode as it transitions between Rgroups, compared with baseline HDFS performance (where all Datanodes are healthy) and its behavior while recovering from a failed Datanode. Fig. 6.2 shows the client throughput after the setup phase, followed by a noticeable drop in client throughput when a Datanode fails (emulated by stopping the Datanode). This is caused by the reconstruction IO that recreates the data from the failed node. Read latency exhibits similar behavior as shown in Fig. 6.3. Eventually, throughput settles at about 5% lower than prior to failure, since now there are 19 Datanodes. The Relaxed Rgroup Transition refers to the customized throttled output stream that implements the peak-IO constraint as explained in §6.3.2. The impact of throttling seen in both throughput and latency when comparing the sudden non-throttled recovery (which is an emulation of sudden transitions shown via the “Failure” plots in Fig. 6.2 and Fig. 6.3) to the throttled transitions (shown via the “Relaxed Rgroup Transition” plots in the same figures). The gradual reduction of throughput and gradual increase of latency maintains predictable performance for on-going foreground tasks.

Fig. 6.2 also shows client throughput when a node is RDn transitioned from 6-of-9 to 7-of-10. There is minor interference during the transition, which can be attributed to the data

movement that HDFS performs as a part of decommissioning. The transition requires less work than failed node reconstruction, yet takes longer to complete because Pacemaker limits the transition IO. Eventually, even though 20 Datanodes are running, the throughput is lower by $\approx 5\%$ (one Datanode's throughput). This happens because Pacemaker empties the Datanode before it moves into the new Rgroup, and load-balancing data to newly added Datanodes happens over a longer time-frame. Experiments with RUp transition showed similar results.

Chapter 7

Discussion and future direction

This chapter discusses issues surrounding disk-adaptive redundancy and also outlines its future research directions that can be explored.

7.1 Disk-adaptive redundancy is a self-correcting mechanism for poor disks

Today's storage clusters have a static redundancy scheme selection process as explained in 2. This places an inherent upper bound on the AFR that can be tolerated across all deployed disks. Specifically, the highest AFR tolerated by the most conservative scheme that still meets the target MTTDL acts as the upper bound of the AFR that can be tolerated in that cluster. If a disk make/model violates the AFR upper bound because of a manufacturing defect, or because of a bad firmware upgrade, it causes panic. Such situations can cause data loss if the storage administrators aren't quick enough to ensure data has been copied out and the problematic disks have been replaced promptly. Large-scale storage cluster administrators have told us that this situation does occur in practice, and quick reaction of administrators and systems is paramount to ensure data safety. A disk-adaptive redundancy system would notice an increase in AFR and automatically increase the redundancy without any human intervention. This helps significantly reduce the risk of data loss.

7.2 Striping data across different Rgroups

In the absence of disk-adaptive redundancy, storage clusters tend to place a stripe of width n across *any n disks*, uniformly at random after taking into account fault domain considerations such as "no two chunks on the same rack". By explicitly differentiating between reliability of different disk sub-populations, disk-adaptive redundancy disallows the freedom to place chunks at random, since different stripes would then end up with different amount of reliability. There are two major disadvantages of forcing all data of a stripe being stored on similarly failing disks: a) data often requires expensive redundancy transitioning as shown in Chapter 5 and b) this technique lends itself to correlated failures that might arise because disks having the

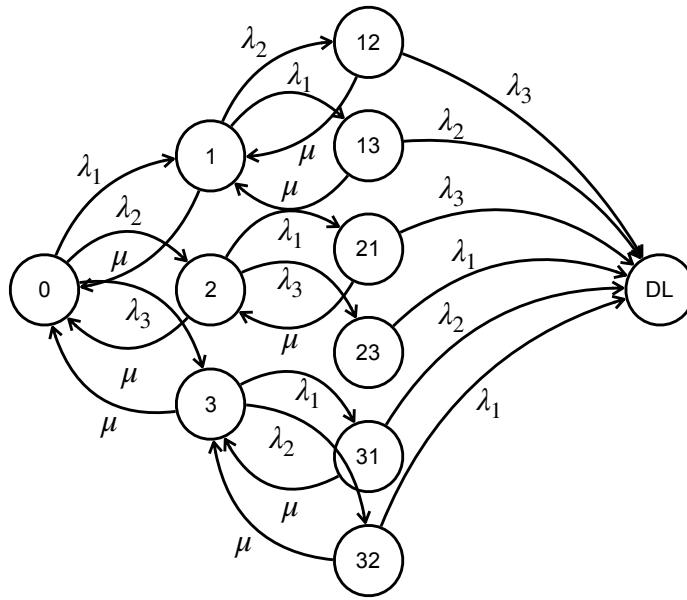


Figure 7.1: Continuous Time Markov Chain for calculating MTTDL of 3-replication with cross-disk-group striping.

same redundancy may often be disks belonging to the same make/model which are deployed together (step-deployment).

A variant of might choose to place data intelligently across differently reliable disks, while still monitoring the MTTDL of each stripe such that it meets the defined target. We term this approach of disk-adaptive redundancy as “cross-group striping”. When the underlying disks change their reliability, a cross-group striping system will shuffle data around to ensure that the target reliability is always met. MTTDL calculations for a stripe spanning differently reliable disks is much more complicated compared to traditional MTTDL calculations. For example, Fig. 7.1 shows the continuous-time Markov chains required to calculate the MTTDL of 3-replication when replicas reside on disks with different reliability. Cross-group striping systems can overcome the above mentioned problems of disk-adaptive redundancy systems such as HeART and Pacemaker because: a) moving data is much cheaper than transitioning; and b) striping across differently reliable disks reduces the risk of data loss due to correlated failures. These systems might be the right disk-adaptive redundancy solution for storage clusters that want to minimize interference from background IO activity. Cross-group striping systems promise a rich optimization space for re-imagining disk-adaptive redundancy without requiring any redundancy transitions.

7.3 Incorporating availability in disk-adaptive redundancy

Data is kept redundant for two reasons: durability and availability. This dissertation has explored in-depth disk-adaptive redundancy for durability. The purpose of redundancy for availability is to ensure data access even when servers are inaccessible due to events such as software upgrades or networking partitioning. More often than not redundancy for the purpose of availability is done for hot data. Moreover, the level of redundancy kept for adequate availability overshoots the redundancy required for adequate durability.

Incorporating availability in disk-adaptive redundancy systems such as HeART and Pacemaker is straightforward. The scheme selection process follows a constraint based approach in filtering out infeasible schemes that disks could transition to. In addition to the existing IO and reliability constraints, an availability-aware disk-adaptive redundancy system would also have an availability constraint for scheme selection. Locally Reconstructible Codes (LRC) are examples of erasure codes that have both local and global parities which are amenable to high availability without compromising durability [B9].

7.4 Adaptive redundancy is not restricted to HDDs

The concept of adaptive redundancy transcends beyond HDDs. The obvious extension is to other storage devices such as solid state disks (SSD) and non-volatile memory (NVM). SSDs have been shown to have a different bathtub curve shape compared to HDDs [B0], and NVMs are still too young for us to know of how their AFR varies over their lifetime. No matter what how the AFR changes, adaptive redundancy systems can help keep data safe on these devices. Redundancy is also ensured at the level of servers, where data is consciously spread across disks that are attached to multiple servers. Adaptive redundancy can also aid in ensuring low cost and robust data safety at the server level.

Chapter 8

Conclusions

Data reliability techniques are unaware of disk reliability heterogeneity present in today's cluster storage systems. In an attempt to keep data adequately safe, viable redundancy schemes are chosen on the basis of perceived disk failure rates, which are typically conservative upper bounds on observed disk failure rates (AFR). This makes data reliability much more costly than required. On occasion, data redundancy, inadequate estimate AFR might even render data under-reliable and thus at a risk of permanent data loss.

My thesis proposes disk-adaptive redundancy as a practical solution to ensure adequate data reliability in storage clusters that have disk reliability heterogeneity. disk-adaptive redundancy is the act of dynamically tailoring data redundancy to observed disk failure rates while always meeting a desired reliability target. This dissertation lays the foundation of disk-adaptive redundancy via four primary contributions. In the first contribution we design and implement *HeART* – the heterogeneity-aware redundancy tuner, which makes the case for disk-adaptive redundancy. It shows that disk-adaptive redundancy has the potential of reduced storage costs by up to 33% compared to 3-replication and up to 16% compared to popular erasure coding schemes such as 6-of-9 or 10-of-14. The second contribution is the in-depth analysis of over 5 million disks across three production environments of NetApp, Google and Backblaze spanning over 60 makes/models. This analysis provides several new insights (and re-defines our existing understanding) about disk reliability distribution over its lifetime. This analysis drives our next contribution – *Pacemaker*; a disk-adaptive redundancy orchestrator that overcomes transition overload (re-encoding overheads) to make disk-adaptive redundancy systems practically feasible. It reduces the IO overhead of redundancy transitions by over 90% compared to traditional read-re-encode-write approaches. Furthermore, Pacemaker-enabled disk-adaptive redundancy requires an overall IO of less than 0.5% of the cluster IO bandwidth on average for all its redundancy transitions while keeping all transition IO capped by a reasonable threshold (say 5%) of the cluster's IO bandwidth. The final contribution describes algorithms to implement disk-adaptive redundancy in a widely used distributed file system (HDFS) by re-purposing many of its existing features for disk-adaptive redundancy.

Data in data centers is growing at an alarming rate and will continue to do so with large migrations of businesses and data repositories to the cloud [73]. This puts serious space pressure on many (if not all) of the large-scale storage clusters. Furthermore, most of the data being stored is warm or cold data, which is accessed occasionally, but needs to be highly

durable. Lowering the storage and energy costs of storing this data without compromising data reliability is one of the main motivations behind public clouds offering low-cost data archival solutions such as Amazon Glacier [1] and Google Cold Storage [32]. The hardware industry too is witnessing a lot of ongoing research in newer high-density storage devices such as shingled magnetic recording disks (SMR), heat-assisted magnetic recording disks (HAMR) and bit-patterned media recording disks (BPMR). More radical hardware inventions for solving the cold data storage problem include projects such as Project Silica that aims to use glass as a high-density storage medium for cold data [7, 17, 64], and DNA storage [13], which aims to store data in synthesized DNA strands.

Disk-adaptive redundancy is a completely software driven solution for the aforementioned problem that provides sizeable space-savings ($\approx 20\%$ fewer disks; tens-of-thousands of fewer storage devices per cluster) while ensuring adequate data reliability. Moreover, disk-adaptive redundancy is an optimization technique that is applicable to all storage devices currently present, and those that will be invented in the future. Theoretically, it even supports redundancy performed on tiered storage clusters where different storage media are used to service different performance requirements, but the data on any of those tiers has to maintain a target reliability. This makes disk-adaptive redundancy a robust solution for a timely problem.

Appendix A

Failure rate estimation details

This section describes how we calculate failure rates for each Dgroup based on the disks' age using empirical data.

In the storage device reliability literature, the failure rate over a period of time is typically expressed in terms of Annualized Failure Rate (AFR), and calculated as:

$$AFR (\%) = \frac{d}{E} \times 100, \quad (A.1)$$

where d is the number of observed disk failures, and E is the sum of the exposure time of each disk, measured in years. The exposure time of a disk is the amount of time it was on operation (i.e., deployed and had not failed nor been retired) during the period in consideration, and it is typically measured at the granularity of days.

If the time to failure is exponentially distributed, then Formula A.1 corresponds to the maximum likelihood estimate for the rate parameter of the exponential distribution. Due to the memoryless property of this distribution, such a formula would be appropriate only if we assume that failure rate is constant with respect to time or device age. Thus, Eq. A.1 is useful for estimating AFR over long and stable periods of time, but makes it hard to reason about changes in AFR over time. Therefore, in this work, we estimate AFR using the following approach.

Assume that the lifetime (time from deployment to failure) of each disk is an i.i.d. discrete random variable T with cumulative density function F and probability mass function f . The failure rate (also known as *hazard rate*) [80] of this distribution is given by:

$$h(t) = f(t)/(1 - F(t)). \quad (A.2)$$

The *cumulative hazard* defined as $H(t) = \sum_{i=0}^t h(i)$ is commonly estimated using the Nelson-Aalen estimator:

$$\hat{H}(t) = \sum_{i=0}^t \frac{d_i}{a_i} \quad \text{for } t \in \{0, \dots, m\}, \quad (A.3)$$

where d_i is the number of disks that failed during their i -th day, a_i is the number of disks that were in operation at the start of their i -th day, and m is the age in days of the oldest observed

disk drive. An estimate for the failure rate can be obtained by applying the so-called *kernel method* [28]:

$$\hat{h}(t) = \sum_{i=0}^m \frac{d_i}{a_i} K(t - i), \quad \text{for } t \in \{0, \dots, m\}, \quad (\text{A.4})$$

where $K(\cdot)$ is a kernel function. Formula A.4 can be considered as a smoothing over the increments of Formula A.3. For our calculations, we utilized an Epanechnikov kernel [36] with a bandwidth of 30 days (the Epanechnikov kernel is frequently used in practice due to its good theoretical properties).

A big advantage of this approach is that it is *nonparametric*, meaning that it does not assume that the lifetime T follows any particular distribution. This allows Pacemaker to adapt and work effectively with a wide arrange of storage devices with vastly different failure rate behaviors.

Bibliography

- [1] Amazon. Amazon Glacier. <https://aws.amazon.com/glacier>, . 8
- [2] Amazon. Amazon Simple Storage Service (S3). <https://aws.amazon.com/s3>, . 21
- [3] Amazon. Kinesis. <https://aws.amazon.com/kinesis>, . 34]
- [4] Amazon. Robust Random Cut Forest. <https://docs.aws.amazon.com/kinesisanalytics/latest/sqlref/sqlrf-random-cut-forest.html>, . 34]
- [5] George Amvrosiadis, Angela Demke Brown, and Ashvin Goel. Opportunistic storage maintenance. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2015. 56
- [6] Preethi Anantharaman, Mu Qiao, and Divyesh Jadav. Large Scale Predictive Analytics for Hard Disk Remaining Useful Life Estimation. In *2018 IEEE International Congress on Big Data (BigData Congress)*, 2018. 7, 35, 56
- [7] Patrick Anderson, Richard Black, Ausra Cerkauskaite, Andromachi Chatzieleftheriou, James Clegg, Chris Dainty, Raluca Diaconu, Rokas Drevinskas, Austin Donnelly, Alexander L Gaunt, et al. Glass: A new media for a new era? In *10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018. 8
- [8] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *ACM SIGMETRICS Performance Evaluation Review*, 2002. 56
- [9] Backblaze. HDD SMART Stats. <https://www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures>. 2
- [10] Backblaze. Disk Reliability Dataset. <https://www.backblaze.com/b2/hard-drive-test-data.html>, 2013-2018. [document], 22, 31, 81, 321, 51
- [11] Backblaze. Erasure coding used by Backblaze. <https://www.backblaze.com/blog/reed-solomon/>, 2013-2018. 222
- [12] 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. 321, 35, 534, 553, 56
- [13] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649, 2016. 8
- [14] Eric Brewer. Spinning Disks and Their Cloudy Future. <https://www.usenix.org/node/194391>, 2018. 1, 211

- [15] Eric Brewer, Lawrence Ying, Lawrence Greenfield, Robert Cypher, and Theodore T'so. Disks for data centers. Technical report, Google, 2016. [I](#), [2.11](#)
- [16] Brad Calder, Ju Wang, Aaron Ogas, Niranjan Nilakantan, Arild Skjolsvold, Sam McElvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011. [2.1](#), [5.4.3](#)
- [17] Andromachi Chatzieleftheriou, Ioan Stefanovici, Dushyanth Narayanan, Benn Thomsen, and Antony Rowstron. Could cloud storage be disrupted in the next decade? In *12th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*, 2020. [8](#)
- [18] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient Replica Maintenance for Distributed Storage Systems. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006. [3.5](#)
- [19] Gerry Cole. Estimating drive reliability in desktop computers and consumer electronics systems. *Seagate Technology Paper TP*, 2000. [2.2.3](#), [3.2.1](#)
- [20] Alexandros G. Dimakis, Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 2010. [5.6](#)
- [21] Jon Elerath. Hard-disk drives: The good, the bad, and the ugly. *Communication of ACM*, 2009. [3.2.2](#), [3.5](#), [5.6](#)
- [22] Jon G Elerath. AFR: problems of definition, calculation and measurement in a commercial environment. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2000. [3.2.9](#)
- [23] Jon G Elerath. Specifying reliability in the disk drive industry: No more MTBF's. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2000. [3.2.2](#)
- [24] erasure code ceph documentation. Erasure code Ceph Documentation. <https://docs.ceph.com/docs/master/rados/operations/erasure-code/>, (accessed September 25, 2019). [5.6](#)
- [25] Facebook. HDFS RAID. <http://www.slideshare.net/ydn/hdfs-raid-facebook>. [I](#), [2.2.9](#)
- [26] Andrew Fikes. Storage architecture and challenges. *Talk at the Google Faculty Summit*, 535, 2010. [2.2.9](#)
- [27] 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. [I](#), [2.2.9](#), [2.2.3](#), [3.1](#), [3.2.3](#), [3.2.3](#), [3.4.2](#), [5.4.3](#), [5.5](#)
- [28] Apache Software Foundation. HDFS Erasure Coding Phase II – EC with contiguous layout. <https://issues.apache.org/jira/browse/HDFS-8030>, 2016 (accessed November 5, 2020). [6.2.2](#)
- [29] Apache Software Foundation. HDFS Erasure Coding. <https://hadoop.apache.org/>

[docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html](https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html), 2017
(accessed November 5, 2020). [1](#), [5.4.3](#), [5.6](#)

- [30] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003. [1](#), [2.1](#), [2.12](#), [2.2](#)
- [31] Garth Gibson, Gary Grider, Andree Jacobson, and Wyatt Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX; login*, 38(3), 2013. [6.5](#)
- [32] Google. Google Cold Storage. <https://cloud.google.com/storage/archival>. [8](#)
- [33] Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin. On the locality of codeword symbols. *IEEE Transactions on Information Theory*, 2012. [5.6](#)
- [34] Rong Gu, Qianhao Dong, Haoyuan Li, Joseph Gonzalez, Zhao Zhang, Shuai Wang, Yihua Huang, Scott Shenker, Ion Stoica, and Patrick PC Lee. DFS-PERF: A scalable and unified benchmarking framework for distributed file systems. *EECS Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-133*, 2016. [6.5](#)
- [35] Greg Hamerly, Charles Elkan, et al. Bayesian approaches to failure prediction for disk drives. In *International Conference on Machine Learning (ICML)*, 2001. [3.5](#), [5.6](#)
- [36] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Kernel smoothing methods. In *The elements of statistical learning*, pages 191–218. Springer, 2009. [1](#), [A](#)
- [37] 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. [3.5](#), [5.6](#)
- [38] Yuchong Hu, Xiaoyang Zhang, Patrick P. C. Lee, and Pan Zhou. Generalized optimal storage scaling via network coding. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 956–960. IEEE, 2018. doi: 10.1109/ISIT.2018.8437684. [5.6](#)
- [39] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogas, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin, et al. Erasure Coding in Windows Azure Storage. In *USENIX Annual Technical Conference (ATC)*, 2012. [1](#), [2.2](#), [3.2.3](#), [3.2.3](#), [7.3](#)
- [40] 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. [3.5](#), [5.6](#)
- [41] 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. [\[document\]](#), [1](#), [2.4](#), [5.1](#), [5.1\(a\)](#), [5.3.3](#), [5.5.2](#), [5.6](#)
- [42] Kimberly Keeton, Cipriano A Santos, Dirk Beyer, Jeffrey S Chase, John Wilkes, et al. Designing for disasters. In *USENIX File and Storage Technologies (FAST)*, 2004. [3.5](#), [5.6](#)
- [43] Larry Lancaster and Alan Rowe. Measuring real-world data availability. In *LISA*, pages 93–100, 2001. [4.1](#)
- [44] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, An-

- drew Douglas, Nathanaël Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, et al. Understanding rack-scale disaggregated storage. In *9th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, 2017. [\[21\]](#)
- [45] Christopher R Lumb, Jiri Schindler, Gregory R Ganger, David F Nagle, and Erik Riedel. Towards higher disk head utilization: extracting free bandwidth from busy disk drives. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2000. [\[56\]](#)
- [46] Christopher R Lumb, Jiri Schindler, Gregory R Ganger, et al. Freeblock scheduling outside of disk firmware. In *USENIX File and Storage Technologies (FAST)*, 2002. [\[56\]](#)
- [47] 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. [\[1\]](#), [\[24\]](#), [\[32.1\]](#), [\[32.2\]](#), [\[3.3.4\]](#), [\[3.5\]](#), [\[51\]](#), [\[56\]](#)
- [48] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *USENIX Annual Technical Conference (ATC)*, 2017. [\[32.2\]](#), [\[7\]](#), [\[3.5\]](#), [\[56\]](#)
- [49] Francisco Maturana and K. V. Rashmi. Convertible codes: new class of codes for efficient conversion of coded data in distributed storage. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPICS*, pages 66:1–66:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ITCS.2020.66. [\[56\]](#)
- [50] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. *ACM SIGMETRICS Performance Evaluation Review*, 43(1): 177–190, 2015. [\[7.4\]](#)
- [51] Sara Mousavi, Tianli Zhou, and Chao Tian. Delayed parity generation in MDS storage codes. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 1889–1893. IEEE, 2018. [\[56\]](#)
- [52] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, et al. f4: Facebook’s warm BLOB storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, 2014. [\[54.3\]](#)
- [53] 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. [\[3.5\]](#), [\[56\]](#)
- [54] Alina Oprea and Ari Juels. A Clean-Slate Look at Disk Scrubbing. In *USENIX File and Storage Technologies (FAST)*, 2010. [\[5.3.4\]](#), [\[56\]](#)
- [55] Dimitris S. Papailiopoulos and Alexandros G. Dimakis. Locally repairable codes. *IEEE Transactions on Information Theory*, 2014. [\[56\]](#)
- [56] 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. [\[1\]](#), [\[22.3\]](#), [\[32.3\]](#), [\[3.5\]](#), [\[56\]](#)

- [57] 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. [1](#), [22](#), [24](#), [3.2.1](#), [3.2.2](#), [3.3.4](#), [3.5](#), [5.1](#), [5.6](#)
- [58] 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, Hong Kong, China, June 14-19, 2015*, pages 1482–1486. IEEE, 2015. doi: 10.1109/ISIT.2015.7282702. [5.6](#)
- [59] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar. Enabling node repair in any erasure code for distributed storage. In Alexander Kuleshov, Vladimir M. Blinovsky, and Anthony Ephremides, editors, *2011 IEEE International Symposium on Information Theory Proceedings, ISIT 2011, St. Petersburg, Russia, July 31 - August 5, 2011*, pages 1235–1239. IEEE, 2011. doi: 10.1109/ISIT.2011.6033732. [5.6](#)
- [60] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *IEEE Trans. on Information Theory*, 2011. [5.6](#)
- [61] K V Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2013. [1](#), [22](#), [2.2.2](#), [3.1](#), [3.2.3](#), [3.3.3](#), [3.4.2](#)
- [62] 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. [22](#), [2.2.2](#), [3.1](#), [3.2.3](#), [3.3.3](#), [3.4.2](#)
- [63] 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*, 63(9):5802–5820, 2017. [5.6](#)
- [64] Microsoft Research. Project Silica. <https://www.microsoft.com/en-us/research/project/project-silica>. [8](#)
- [65] Yasushi Saito, Svend Frølund, Alistair Veitch, Arif Merchant, and Susan Spence. FAB: building distributed enterprise disk arrays from commodity components. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004. [3.2.3](#)
- [66] 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*, 2013. [22](#), [2.2.2](#), [3.1](#), [3.2.3](#), [3.4.2](#)
- [67] 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. [24](#), [3.5](#), [5.6](#)

- [68] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*. IOP Publishing, 2007. [B.5](#), [B.6](#)
- [69] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. Understanding latent sector errors and how to protect against them. *ACM Transactions on Storage (TOS)*, 2010. [B.2.1](#), [B.5](#), [B.3.4](#), [B.6](#)
- [70] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *USENIX File and Storage Technologies (FAST)*, 2016. [I](#), [B.2.4](#), [B.1](#), [B.5](#)
- [71] Thomas JE Schwarz, Qin Xin, Ethan L Miller, Darrell DE Long, Andy Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, 2004. [B.6](#)
- [72] Seagate. Hard disk drive reliability and MTBF / AFR. http://knowledge.seagate.com/articles/en_US/FAQ/174791en. [B.2.1](#)
- [73] Seagate. The Digitization of the World From Edge to Core. <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>, 2018. [I](#), [B.8](#)
- [74] Sandeep Shah and Jon G Elerath. Disk drive vintage and its effect on reliability. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 2004. [B.5](#), [B.6](#)
- [75] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop distributed file system. In *IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, 2010. [I](#), [B.6](#)
- [76] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Robert Tappan Morris, M Frans Kaashoek, and John Kubiatowicz. Proactive Replication for Data Durability. In *USENIX International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006. [B.5](#)
- [77] Brian D Strom, SungChang Lee, George W Tyndall, and Andrei Khurshudov. Hard disk drive reliability modeling and failure prediction. *IEEE Transactions on Magnetics*, 2007. [B.5](#), [B.6](#)
- [78] Martin A. Tanner and Wing Hung Wong. The estimation of the hazard function from randomly censored data by the kernel method. *Ann. Statist.*, 11(3):989–993, 09 1983. doi: 10.1214/aos/1176346265. URL <https://doi.org/10.1214/aos/1176346265>. [A](#)
- [79] 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. [B.5](#), [B.6](#)
- [80] Kishor Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Wiley, 2001. [I](#), [B.2.3](#), [A](#)
- [81] Charles Truong, Laurent Oudre, and Nicolas Vayatis. A review of change point detection methods. In *arXiv:1801.00718v1 [cs.CE]*, 2018. [B.4.1](#)
- [82] Charles Truong, Laurent Oudre, and Nicolas Vayatis. ruptures: change point detection

- in python. In *arXiv:1801.00826v1 [cs.CE]*, 2018. [\[3.4.1\]](#)
- [83] Myna Vajha, Vinayak Ramkumar, Bhagyashree Puranik, Ganesh Kini, Elita Lobo, Biren-jith Sasidharan, P Vijay Kumar, Alexandar Barg, Min Ye, Srinivasan Narayananamurthy, et al. Clay codes: Moulding {MDS} codes to yield an {MSR} code. In *USENIX File and Storage Technologies (FAST)*, 2018. [\[5.6\]](#)
- [84] 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. [\[3.5\]](#), [\[5.6\]](#)
- [85] 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. [\[1\]](#)
- [86] 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*, 27(9):2639–2652, 2016. doi: 10.1109/TPDS.2015.2505722. [\[5.6\]](#)
- [87] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A tale of two erasure codes in HDFS. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST’15, pages 213–226, Berkeley, CA, USA, 2015. USENIX Association. ISBN 978-1-931971-20-1. URL <http://dl.acm.org/citation.cfm?id=2750482.2750499>. [\[5.6\]](#)
- [88] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchang Zhang, Jian-Guang Lou, et al. Improving service availability of cloud systems by predicting disk error. In *USENIX Annual Technical Conference (ATC)*, 2018. [\[7\]](#)
- [89] Jimmy Yang and Feng-Bin Sun. A comprehensive review of hard-disk drive reliability. In *IEEE Reliability and Maintenance Symposium (RAMS)*, 1999. [\[3.2.2\]](#)
- [90] Xiaoyang Zhang, Yuchong Hu, Patrick P. C. Lee, and Pan Zhou. Toward optimal storage scaling via network coding: from theory to practice. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 1808–1816. IEEE, 2018. doi: 10.1109/INFOCOM.2018.8485961. [\[5.6\]](#)
- [91] Zhe Zhang, Andrew Wang, Kai Zheng, G Uma Maheswara, and B Vinayakumar. Introduction to hdfs erasure coding in apache hadoop. *blog.cloudera.com*, 2015.
- [92] 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. [\[3.5\]](#), [\[5.6\]](#)
- [93] Weimin Zheng and Guangyan Zhang. Fastscale: accelerate RAID scaling by minimizing data migration. In Gregory R. Ganger and John Wilkes, editors, *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*, pages 149–161. USENIX, 2011. URL <http://www.usenix.org/events/fast11/tech/techAbstracts.html#Zheng>. [\[5.6\]](#)
- [94] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive drive failure prediction for large scale storage systems. In *IEEE/NASA Goddard*

November 5, 2020
DRAFT

Conference on Mass Storage Systems and Technologies (MSST), 2013. 