University of Nevada, Reno

# Probabilistic Approach to Avoid Uncorrectable Bit Errors in Storage Systems.

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science and Engineering

by

Masudul Hasan Masud Bhuiyan

Dr. Engin Arslan - Thesis Advisor
December 2020

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

**MASUDUL HASAN MASUD BHUIYAN**

Entitled

**Probabilistic Approach to Avoid Uncorrectable Bit Errors in Storage Systems**

be accepted in partial fulfillment of the
requirements for the degree of

Master of Science

Engin Arslan, Ph.D., Advisor

Frederick C Harris, Jr., Ph.D., Committee Member

Mehmet Gumus, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

December, 2020

**Abstract**

Silent data corruption in storage system poses a significant risk to the integrity of data. While error correction codes (ECC) can recover the majority of the errors, a non-negligible portion of them escape ECC, referred to as uncorrectable errors. As the scale of storage systems increases, the mean time between uncorrectable errors is reduced from months to hours, necessitating efficient ways to detect and handle them. In this thesis, we propose prediction models for uncorrectable errors by analyzing $150M$ daily SMART logs from $143K$ hard drives collected over the period of five years. The models achieve up-to 97% accuracy in uncorrectable bit error prediction while keeping false positive rates less than 3%. We further introduce two use cases to utilize highly accurate error prediction models to (i) mitigate the I/O overhead of file transfer integrity verification on file systems and to (ii) reduce the amount of I/O that is processed by disks with uncorrectable errors. Evaluation results show that running integrity verification only for disks with high error probability allows up to 97% decrease in I/O overhead of file transfers while avoiding more than 90% of uncorrectable errors. Moreover, diverting I/O operations from high-risk disks to low-risk disks can reduce the amount of data exposed to an uncorrectable error by 80% while keeping the overhead on low-risk disks less than 5%.

# Acknowledgments

First and foremost, I am thankful to the Creator who has granted me knowledge and ability to complete this thesis successfully. I want to thank my adviser Dr. Engin Arslan who was an excellent mentor and role model. He guided every step of my master's journey and helped me to grow as an independent researcher. Thank you, Engin, I am forever grateful for your immense trust and support.

I also thank my committee members, Dr. Frederick C Harris, Jr., and Dr. Mehmet Gumus, for taking the time to review this thesis and providing guidance on my research.

Next, I want to thank the current and past members of High Performance Computing And Networking (HPCN) Lab. Be it late afternoon chat or helping with running experiments, they were always there to help me. In particular, I am very grateful to Hemanta Sapkota, Md Arifuzzaman, and Ahmed Alhussen for their continuous support.

Finally, I want to thank my most important supporters. Firstly, My mother Mahmuda Shafiq, to whom I owe everything, for the unconditional love and support. My father, Shafiqul Islam, whose prayers and thoughts were the main enablers for all my achievements. My siblings, for always being there for me. Last, but not least, my wonderful wife, Saima, for supporting me throughout this journey and making our long-distance marriage work.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Advancements in computing and sensing technologies paved the way for many science applications to generate a massive amount of data. For example, Hardware/Hybrid Accelerated Cosmology Code is an extreme-scale cosmology simulation that produces 20PB data in a single particle simulation [11]. Similarly, soon-to-be-operational cosmology project Large Synoptic Survey Telescope will use a 3.2 gigapixel camera to take pictures of the southern sky and is expected to produce 15TB data every night. This huge volume of data is often needed to be stored in long-term storage systems to enable offline data analysis and on-demand access to remote collaborators. Thus, high-performance computing clusters are bundled with high-capacity, high-throughput parallel file systems that are composed of hundreds of storage servers and thousands of disks.

As a result, the scale of storage systems is growing rapidly to accommodate the increasing data generation rates of scientific applications. When combined with the fact that the reliability of disk drives has not changed significantly over the past years, storage systems experience bit errors more frequently, putting sensitive data at risk. While many of such errors can be recovered at disk level through common error

correcting codes (ECC), a non-negligible portion of them happen in a way that ECC falls short to correct, referred as uncorrectable errors or latent sector errors [5], [12], [14]. A previous study found that uncorrectable errors take place once in every $10^{-12}$ ($125GB$) to $10^{-15}$ ($125TB$) bits of I/O operation [24], [33]. A more recent study also revealed that 11-25% of all hard disks are exposed to at least one uncorrectable error in their lifetime [18]. Considering that file systems in many high performance computing clusters consist of tens of thousands of disks and handle hundreds of terabytes of I/O workload daily [17], these rates indicate the frequent occurrence of uncorrectable errors. In fact, our own analysis reveals that, on average, four uncorrectable errors take place every day in a cluster with around 38K disks.

Uncorrectable errors, if not handled timely, can lead to permanent data loss even when parity-enabled disk arrays are used [12]. Although the use of file system level checksumming together with parity or mirror-enabled disk arrays (e.g., RAID-Z) can help to detect and recover from many types of uncorrectable errors through RAID reconstruction, the cost of recovery could significantly deteriorate user response time [6], [30], [31]. Hence, we propose models to predict the occurrence of uncorrectable errors using Self-Monitoring, Analysis and Reporting Technology (SMART) [2] metrics and show that the models can be used to avoid them or at least minimize their impact on the system performance. SMART logs are, by default populated once a day and used to keep track of the status of storage devices [23]. By analyzing $150M$ SMART logs from $143K$ disks collected over the period of 68 months, from February 2014 to September 2019 [4], we developed models to predict the occurrence of uncorrectable errors with up-to 97% accuracy with as low as 3% misclassification rate.

Furthermore, we leverage high accuracy prediction models to introduce two application scenarios where early prediction of uncorrectable errors can be used to reduce the workload on storage system or minimize the impact of uncorrectable errors. First,

we demonstrate that one can reduce I/O workload of storage systems by eliminating integrity-verification for file transfers when disks-in-use are not expected to develop an error. In a nutshell, integrity verification– upon the completion of the file transfer– requires both sender and receiver to read files back from storage to calculate and compare their checksum in order to capture data corruption that might happen during network transmission or disk write. This process, however, incurs significant I/O overhead to file systems in addition to slowing down the transfer process by up to $2x$ [8], [15]. We, therefore, introduce probabilistic integrity verification to calculate file checksum on cache data when disks– that are used to store file data– do not exhibit the symptoms of uncorrectable errors. By doing so, one can reduce I/O load on storage systems significantly in addition to reducing the runtime of integrity verification process. Experimental results show that probabilistic integrity verification can save up to 97% of integrity verification-related I/O of file transfers while capturing more than 90% of uncorrectable errors.

Second, we take advantage of high accuracy prediction of uncorrectable errors to exclude high-risk disks from taking part in write operations to reduce the workload on them, thereby lower the likelihood of uncorrectable bit errors. Even if some errors cannot be avoided, redirecting write operations from high-risk ones to low-risk ones will reduce the amount of data at risk, alleviating the impact of the errors. On the other hand, excluding too many disks imposes the risk of overloading the remaining disks and potentially increasing their error probability. Our analysis shows that one can reduce the amount of I/O on disks with uncorrectable errors by up to 80% while increasing load on the low-risk disks by less than 5%.

In summary, we make the following contributions in this thesis:

- We process $150M$ SMART logs from $143K$ disks over the period of 68 months and present analysis results on the characteristics of uncorrectable errors.

- We apply several machine learning models to predict the occurrences of uncorrectable errors with up-to 97% accuracy while keeping false classification rate below 3%.

- We propose two use case scenarios where high-precision uncorrectable error prediction models can be used to minimize I/O load on storage systems and to alleviate the impact of errors. The simulation results show that 97% of integrity verification-related I/O load on storage systems can be eliminated while ensuring 90% coverage for uncorrectable errors. The results also show that one can save up to 80% of I/O operations from being issued to disks with uncorrectable bit errors while increasing the load on the remaining disks by less than 5%.

The rest of the thesis is organized as follows: Chapter 2 presents the related work and background for uncorrectable errors. Chapter 3 analyzes the characteristics of uncorrectable errors. Chapter 4 presents machine learning models to predict them using SMART metrics. Chapter 5 introduces probabilistic integrity verification for file transfers and discusses simulation results. It also describes I/O redirection proposal to avoid uncorrectable bit errors and presents simulation results for different disk models. Finally, we conclude the thesis with a summary and potential future directions in Chapter 6.

# Chapter 2

# Background And Related Work

## 2.1 Uncorrectable Disk Errors

Uncorrectable errors (aka latent sector error (LSE)) occur when data on disk sectors is corrupted beyond what ECC can repair. They are typically caused by undetected write errors or media imperfections. Rozier *et al.* [24] estimated that undetected write errors take place once in $10^{-12}$ to $10^{-13}$ I/O operations due to several types of write errors such as dropped, near-off track, and far-off track writes. Unlike other errors, undetected write errors are "silent" by definition, so they are not identified until a read request is issued subsequent to write. This, in turn, poses data loss risk as Hafner at al. showed that undetected write errors when followed by disk failures may result in complete data loss even in parity-based RAID arrays [12].

Bairavasundaram *et al.* analyzed LSEs for 1.53M hard disks over a period of 32 months [6] and found that 3.25% of all disks developed at least one LSE. Moreover, recent studies based on Facebook and Google datacenters statistics reported that a significant portion of solid state drives (20-57%) also developed at least one LSE

during their lifetime [19], [27]. Since ECC falls short identify and recover from many types of undetected write errors, most modern file systems rely on checksumming, which computes and stores a unique hash value for each data block (or set of blocks) separate from data block [34]. However, Krioukov *et al.* showed that checksumming, if not integrated into file system design properly, can fail to guarantee protections against complex failures such as lost writes and misdirected writes [14].

Although combining file system-level checksumming with parity or mirror-enabled disk arrays can find and fix most undetected write errors through disk scrubbing [21], an ability to predict errors before they happen provides an opportunity to detect and fix them quickly [18] or to completely avoid by reducing the load on high-risk disks.

Previous work on modeling disk reliability mainly focus on disk failure prediction using various machine learning models including support vector machine, neural network, and random forest [20], [32]. For example, Xu *et al.* proposed Cloud Disk Error Forecasting, a cost-sensitive machine learning system to predict disk failures [32]. In the context of error prediction for hard drives, Mahdisoltani *et al.* applied various machine learning models (e.g., neural network and random forest) to predict several types of disk errors using BackBlaze dataset [18]. Despite similarities, our work differs in three main ways: (i) While [18] used three years data of 68K disks, we do not only analyze a much larger dataset (5.5 years data of $143K$ disks) but also provide detailed information about the characteristics of uncorrectable errors such as its ratio and distribution over years. (ii) Our models by using a bigger dataset and larger feature size, and fine-tuning model hyperparameters, obtain significantly higher accuracy (95% vs 80%) while keeping misclassification rates low (5% vs 20%). (iii) More importantly, we propose to integrate uncorrectable error prediction models into two application scenarios and develop models to determine the set of disks to manage which will reduce the impact of uncorrectable bit errors while eliminating a

significant amount of overhead on storage systems caused by integrity verification of file transfers.

## 2.2 Integrity Verification for File Transfers

Data integrity is crucial for many science applications whose computations are extremely sensitive to data manipulation such as Hardware/Hybrid Accelerated Cosmology Code (HACC) simulations [11]. Since many science workflows are distributed in nature– data collection and processing facilities are geographically separated–, integrity of data transfers needs to be protected. Although there are built-in integrity verification mechanisms to detect data corruption, they are either weak or available only in a subset of systems. For example, TCP uses checksum to detect data corruption in the network, however, its 16-bit checksum is unable to detect errors once in 16 million to 10 billion packets [29]—a fairly small value in today's high-speed networks that operate at the speed of hundreds of gigabits-per-second. Besides network corruptions, file transfers are also vulnerable to uncorrectable disk errors that can happen during read (at the source) and write (at the destination) I/O operations. As a result, an extensive field study based on 40 billion Globus data transfer logs shows that one data corruption (happened either at network or disk) in every 1.26TB transfers goes undetected by built-in error detection mechanisms [16]. In another study, researchers executed HACC simulations in a distributed environment and noticed that 5% of all transfers experienced a data corruption that is not detected by existing integrity verification methods [13]. Therefore, high-speed transfer applications (e.g., GridFTP) adopt application-layer end-to-end integrity verification to increase the robustness of data transfers against silent data corruption. End-to-end integrity verification uses secure cryptographic hash functions (e.g., MD5 and SHA256) to calculate and compare

the checksum of files at the source and destination to detect data corruptions that would otherwise go undetected by existing mechanisms. On the other hand, it induces significant I/O overhead to file system while lowering overall transfer throughput due to requiring to reread files after their transfer to compute checksum [3], [7].

Although several approaches have been proposed to optimize the execution time for end-to-end integrity verification [1], [7], [8], [15], its I/O overhead has not been addressed. To give an example, transferring a 200GB file with integrity verification incurs 200GB disk read overhead both on the source and destination sites for checksum calculation apart from 200GB disk read (on source) and write (on destination) load as part of the transfer. Although it is possible that checksum of files can be calculated ahead of time and saved to avoid repetitive calculations [1], receiver needs to compute it at the time of transfer to ensure the integrity of transfers. Moreover, checksum of files cannot be always computed beforehand at the source facility if they are transferred right after they are created for real-time analysis.

Globus, a widely-adopted data transfer service for large-scale scientific data movements [10], supports end-to-end integrity verification, but uses a sequential approach– completes the integrity verification of a file before starting the transfer of next file– when transferring multiple files thus fail to utilize network resources efficiently. Liu *et al.* proposed block-level pipelining to overlap the transfer of one file with the integrity verification (i.e., checksum calculation and comparison) of another file to enhance transfer speed [16]. Charyyev *et al.* proposed RIVA to increase the robustness of end-to-end integrity verification by enforcing checksum calculation to bypass cache memory in order to capture disk errors [7]. Yet, none of the previous work tried to minimize the I/O overhead caused by the end-to-end integrity verification. This work therefore takes on this problem and proposes to use cache data to calculate file checksum when storage disks are deemed to be low-risk, significantly reducing the

I/O overhead of end-to-end integrity verification on storage systems.

# Chapter 3

# Analysis of Uncorrectable Errors

Previous studies show that both commodity and enterprise disks are exposed to a significant number of bit errors stemming from hardware malfunctions and buggy firmware [5], [24], [33]. As a result, it is estimated that on average one uncorrectable error is observed in every $10^{-12}$ and $10^{-15}$ bits of disk I/O for hard drives. As the capacity of modern disk drive increases along with the scale of storage systems, the mean time between uncorrectable errors are reduced to hours, putting sensitive data at risk. Undetected write errors, random bit rotting and media scratches are among the main reasons behind uncorrectable errors as they corrupt the data to the point that it is impossible for error correction codes to recover [22], [28]. Hafner *et al.* show that when undetected write errors occur along with disk failures, it can cause permanent data loss even in parity-enabled RAID arrays (i.e., RAID 5 and 6) [12]. Although file system-level checksumming when used together with parity or mirror-enabled RAID arrays can detect and recover from many types of undetected disk errors, recovery operation induces RAID reconstruction overhead which, if done at the time of I/O request, incurs performance penalty to I/O operations. minimizing uncorrectable errors incidents when possible is a worthy approach.

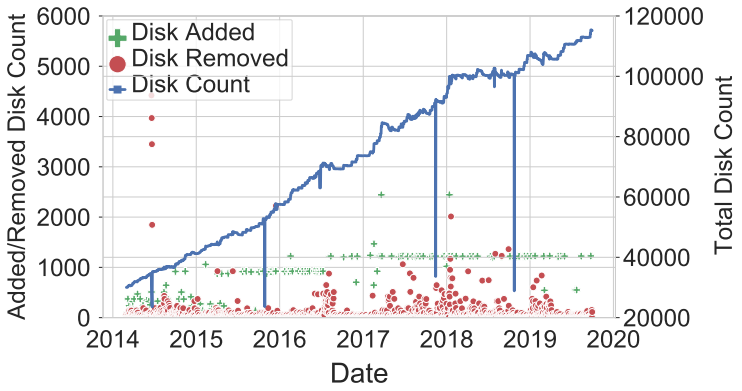



Figure 3.1: The number of added, removed, and total disk counts in Backblaze dataset.

## 3.1 SMART Dataset

To gain insights into uncorrectable disk errors, we utilized publicly available Backblaze dataset which contains SMART logs for $143K$ disks over the period of 68 months from February 2014 to September 2019 [4]. SMART logs are populated once a day for each disk to report their status, thus the dataset consists of $150M$ SMART logs in total. Although there were $143K$ unique disks in total, at most $120K$ of them were active at any given day. Figure 3.1 shows the number of active disks each day as well as added/removed disk counts. The number of disks increased from around 32K to around 110K between 2014 and 2019. We also observe that more than 2,000 disks were removed from the cluster on 13 occasions during this period. Unlike removed disk counts, the number of added disks follows a more steady pattern where 500-1500 disks are added at almost regular intervals.

## 3.2 Distribution of Errors

Among $143K$ disks, $6.6K$ of them had at least one uncorrectable error, bringing the percentage of disks with error to 3.2%. A total of $740K$ uncorrectable errors were

Table 3.1: Overview of disk models used in this work.

| Disk model | Capacity | Total Disk | Ratio of Disks with Uncorrectable Error |
|---|---|---|---|
| ST12000NM0007 | 12 TB | 38,272 | 2.2% |
| ST4000DM000 | 4 TB | 36,950 | 7.1% |
| ST8000NM0055 | 8 TB | 14,811 | 2.2% |
| ST8000DM002 | 8 TB | 10,161 | 2.7% |
| ST3000DM001 | 3 TB | 4,286 | 33.2% |

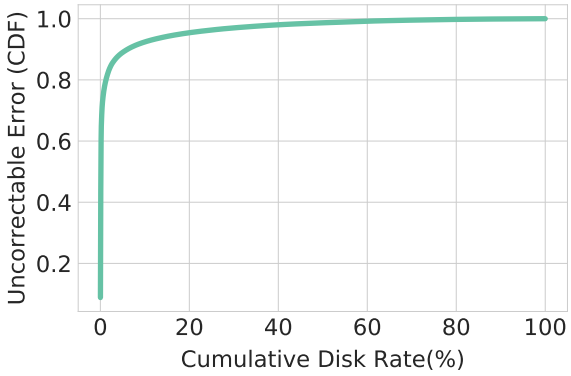



Figure 3.2: Distribution of Uncorrectable Errors to Disk Population

reported, whose distribution to $6.6K$ disks is shown in Figure 3.2. It is clear that a small number of disks contribute to the most of the errors. Specifically, less than 5% of disks had 89% of all errors and less than 10% of disks had 92.4% of all errors. Figure 3.3 demonstrates the distribution of disks by the number of uncorrectable errors that they developed. 14.5% of disks with error raised only one uncorrectable error and 21.7% of them raised more than 20 uncorrectable errors. Moreover, 6% of the disks with error reported more than 100 uncorrectable errors. Figure 3.4 presents the ratio of disks with uncorrectable error over time. While the ratio was as high as 0.15% in July 2014, it is stabilized at around 0.02% after removing one of the disk models from the cluster in late 2014 likely due to exhibiting too many errors.

Although the dataset contains logs from more than 80 disk models, we excluded the ones with less than thousand disks to avoid misleading results due to lack of data.

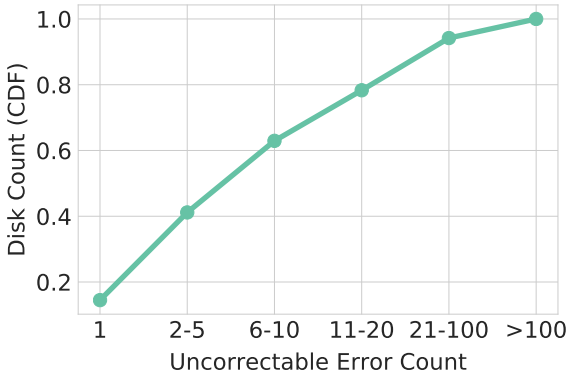



Figure 3.3: Error Count Distribution for Disks with at Least One Uncorrectable Error

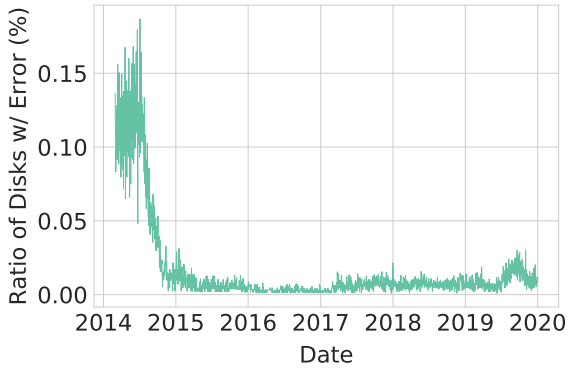



Figure 3.4: Ratio of Disks with Uncorrectable Errors

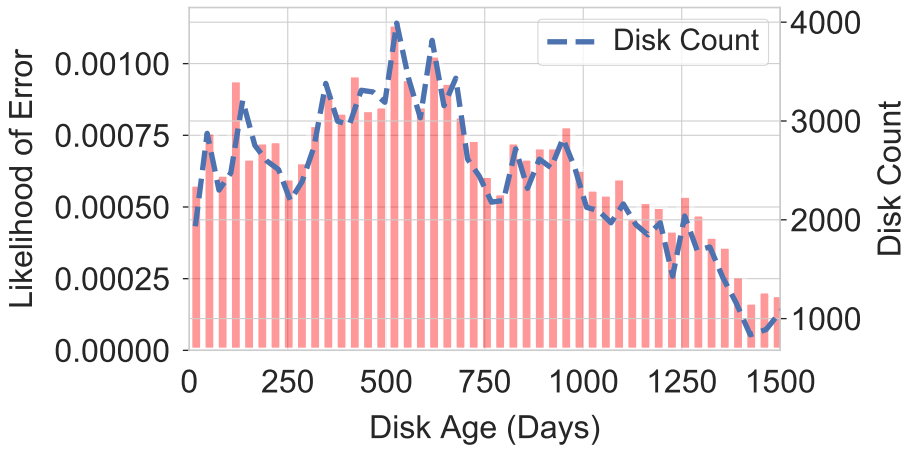



Figure 3.5: The impact of disk age on the occurrence of uncorrectable disk errors

This filtering left a total of 104,480 disks from five disk models whose details are given in Table 3.1. The disk models ST12000NM0007 and ST4000BM000 constitute 72% of all disks as there are more than $36K$ disks for each of these models. Despite having similar number of disks, $3x$ more disks developed an uncorrectable error in disk model ST4000DM000 compared to ST12000NM0007. Moreover, while the ratio of disks with uncorrectable error is less than 7.1% for four disk models, it reaches to 33.2% for ST3000DM001, indicating a significant variation between disk models with regard to the likelihood of developing uncorrectable errors.

## 3.3 Impact of disk age

We also analyzed the impact of disk age on uncorrectable errors in Figure 3.5. Although SMART logs do not contain disk age information, we measured disk age based on the first date SMART logs were published for the disks. We excluded the disks that were present in the first date of the data collection (Feb 1, 2014) since it is possible that those disks were used even before data collection was enabled. The likelihood of error follows a relatively stable pattern until disks are around 700 days old at which

point it starts to decrease. The probability of developing an uncorrectable error is 2-4x lower for older disks (more than four years old) compared to younger ones (less than 2 years old). This surprising observation can be attributed to the fact that old disks with uncorrectable errors might have been removed from the cluster as a precautionary measure to avoid future uncorrectable errors, leaving only the healthy ones. This can be observed by the decreasing disk count– the dashed line in the Figure 3.5. We leave more detailed analysis of this observation as a future work.

# Chapter 4

# Modeling Uncorrectable Errors

As SMART logs are populated every day for each disk, we aim to develop a model that can, by looking at SMART logs of any day, can predict if a disk will develop an uncorrectable error within next 24 hours. The accuracy of the model is then can be checked by looking at the next day's SMART logs to see if the disk raised an uncorrectable error.

## 4.1 Feature Selection

SMART logs contain more than 60 attributes, however, most of them are not reported consistently for all disk models. Filtering out those attributes reduces the total number of attributes to 16. We ran feature selection to find the attributes that can explain the variation in uncorrectable errors using Random Forest model and in the end settled with 13 features. Figure 4.1 demonstrates the contribution of each of 13 features in explaining uncorrectable error attribute. Although we find that five features (uncorrectable error count (#187), load cycle count (#193), power-on hours (#9), seek error rate (#7), and temperature (#194)) are sufficient to explain 90%

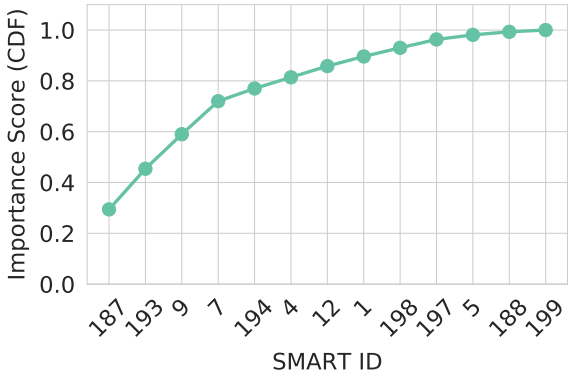



Figure 4.1: Importance score of SMART metrics for Random Forest model. Despite having over 60 SMART features in total, 13 of them are sufficient to explain 99% of variation for uncorrectable errors.

of variance for uncorrectable errors, we settled with 13 features to increase the variance coverage to 99%. The selected attributes are given in Table 4.1, sorted by the importance score.

Once the attributes are identified, we applied several machine learning models to find a relationship between the attributes and uncorrectable error. Note that since our goal is to predict errors before they take place, we marked the SMART logs with a positive label (i.e., uncorrectable error) if the next day's SMART log for the same disk reported an uncorrectable error and marked with a negative label otherwise. True Positive Rate (TPR) and False Positive Rate (FPR) metrics are used to evaluate the performance of prediction models. TPR indicates the portion of the actual positive samples that is correctly identified by a model. In the context of uncorrectable bit errors, it represents the percentage of disks with uncorrectable errors (i.e., positive class) that is correctly identified by the prediction models. The false positive rate, on the other hand, refers to the portion of actual negative cases that is misidentified as positive cases by a model. It represents the percentage of disks without uncorrectable error (i.e., negative class) that are *incorrectly* classified as disks with uncorrectable

Table 4.1: SMART attributes selected as input features for the prediction model

| **ID** | **Attribute Name** | **Importance Score** | **Description** |
|---|---|---|---|
| 187 | Uncorrectable Error | 0.294 | Count of uncorrectable errors |
| 193 | Load Cycle Count | 0.160 | The count of load/unload cycles into head landing zone position |
| 9 | Power-On Hours | 0.136 | Total hours the drive was in power on state |
| 7 | Seek Error Rate | 0.130 | Rate of seek errors of the magnetic heads |
| 194 | Temperature | 0.050 | Device temperature |
| 4 | Start/Stop Count | 0.044 | Count of spindle start/stop cycle. |
| 12 | Power Cycle Count | 0.044 | Count of full hard disk power on/off cycles. |
| 1 | Read Error Rate | 0.038 | Hardware read error count that occurred when reading data from a disk surface |
| 198 | Uncorrectable Sector Count | 0.034 | Total count of uncorrectable errors when reading/writing a sector. |
| 197 | Current Pending Sector Count | 0.033 | Count of unstable sectors waiting to be remapped |
| 5 | Reallocated Sectors Count | 0.018 | Count of the bad sectors that have been found and remapped. |
| 188 | Command Timeout | 0.012 | The count of aborted operations due to HDD timeout. |
| 199 | UltraDMA CRC Error Count | 0.007 | Count of errors in data transfer via the interface cable |

errors by the models. Hence, TPR and FPR values are calculated as follows:

$$TPR = \frac{\#true\_positives}{\#true\_positives + \#false\_negatives}$$

$$FPR = \frac{\#false\_positives}{\#false\_positives + \#true\_negatives}$$

## 4.2 Handling Data imbalance

The imbalance between negative and positive samples makes it challenging to develop accurate models. Specifically, unique uncorrectable errors (counting multiple uncorrectable errors in a day from a disk as one error) constitute less than 0.01% of all SMART logs. Thus, training a model using a dataset with 1:10,000 imbalance ratio will produce a biased model that is likely classify positive samples in negative class, thus missing uncorrectable errors. The typical approach to balance multiclass datasets is to resample the minority or majority classes through undersampling or oversampling. Oversampling duplicates the observations of the minority class and undersampling drops the observations of the majority class to obtain a balanced dataset.

Undersampling is typically preferred over oversampling as it significantly shortens training time and reduces false positive rate. We, therefore, applied undersampling using Synthetic Minority Oversampling Technique [9]. To avoid the bias in the random sampling process, we cross-validated the performance of machine learning models using different undersampled dataset.

Figure 4.2 compares the TPR for different disk models when oversampling and undersampling is used when training a Random Forest model. It is clear that oversampling degrades the TPR for all disk models. While TPR is always above 80% with

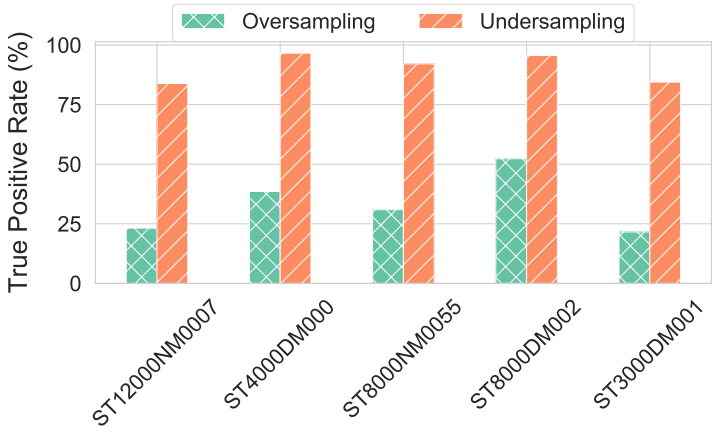



Figure 4.2: True Positive Rate for different disk models using one-to-one oversampling and undersampling ratios to balance dataset.

undersampling for all disk, it is below 52% with oversampling. In addition to degrading TPR, oversampling also almost doubles the size of training dataset by replicating the minority class samples by nearly 10,000 times. This, in turn, lead to considerable slow down in training times for machine learning models.

While outperforming oversampling in achieved TPR, undersampling introduces new challenges as it may disturb the distribution of the majority class and lead to misclassification of the majority class. Thus, FPR is an important metric to keep in mind since it increases the cost of taking any action to mitigate the potential errors. For instance, while only 5-10 disks develop uncorrectable errors at any given day, 1% false positive rate will require additional 250 disks to check when the number of total disks is around 25,000. As a result, it is important to find a sweet spot between TPR and FPR scores.

Figure 4.3 demonstrates the impact of undersampling ratio on weighted TPR and FPR scores, where weights of disk models are proportional to unique uncorrectable error counts. X-axis represents the sampling ratio of the majority class compared to the minority class. For example, sampling ratio 9 refers to a dataset with $1:9$ ratio for positive and negative samples, respectively. As we increase the sampling

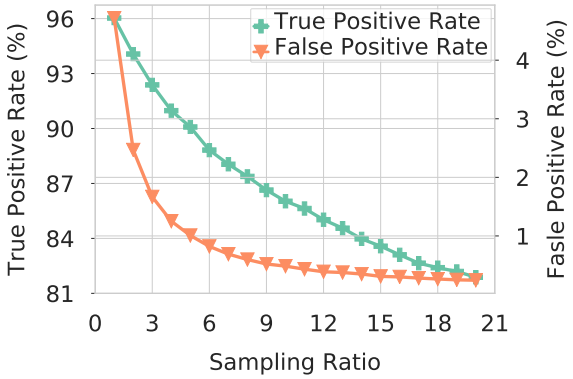



Figure 4.3: True positive and false positive rates with varying undersampling ratios.

ratio, FPR decreases from around 5% to 0.5%. However, this comes at the cost of decreasing TPR in addition to increasing training time. As an example, while $1:1$ undersampling yields more than 96% TPR, $1:21$ undersampling achieves less than 85%. Since we aim to detect and avoid uncorrectable errors as much as possible, high TPR is critical to achieve this mission. Although high FPR rates can adversely affect the feasibility of some of the proposed solutions such as marking some disks as "offline" for write operations as explained in Section 5.2, the FPR rate is still below 5% for the highest TPR rate. Therefore, we settled with $1:1$ undersampling ratio to be able to identify uncorrectable errors with high accuracy.

## 4.3 Model Training

To ensure that uncorrectable errors are sufficiently represented in training and test dataset, we performed following steps when setting up training and test data: We first separate the data into positive and negative groups as shown in Figure 4.4. We then split both classes into 80% training data and 20% test data. Since negative class is the majority class, we ndersample it before merging with positive class to build the training dataset. Similarly, we combine the test data from both classes to develop the test dataset. To ensure the validity of our models, we use 5-fold cross

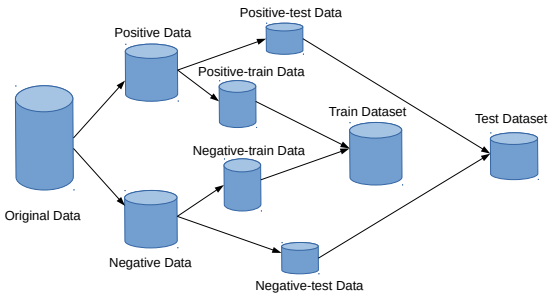



Figure 4.4: Preparation of training and test datasets

validation. for cross-validation, we at first divided the data into positive and negative classes. Both of these classes are then divided into 5 equal-size groups. We combined 4 folds of positive data with 4 folds of negative data to build the training dataset. The rest portion of the data was used as test data. We run the models 5 times with one portion used test set while the remaining data is used for training. We found out cross-validation is very important to remove sampling bias from the model. It also helps the model to avoid temporal bias.

## 4.4 Model Evaluation

We applied seven machine learning models that are commonly used for classification problems. They are Logistic Regression, Support Vector Machines, Decision Tree, Random Forest, Neural Networks, Gaussian Naive Bayes, and LightGBM. Since hyperparameters have a significant impact on the performance of these models, we used scikit-optimize library – with Gaussian-based Bayesian optimization – to discover optimal hyperparameter set for each model. The search focuses on identifying maximum tree depth for Decision Tree, LightGBM and Random Forest, the number of trees for the Random Forest, and the number of hidden layers and learning rate for the Neural Network.

Figure 4.5 presents Receiving Operating Characteristics (ROC) curve for different

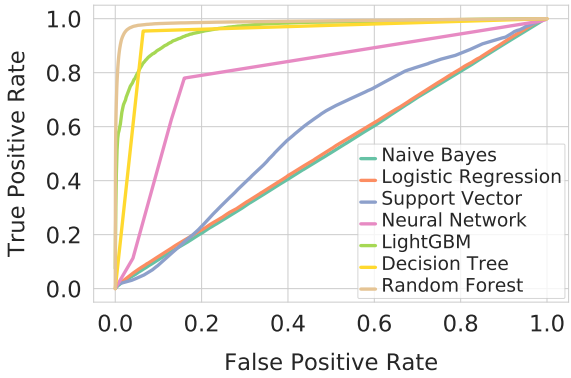



Figure 4.5: ROC curves of machine learning models for uncorrectable error prediction. Random Forest yields the best performance with 95% true positive rate with less than 5% false positive rate.

classification models. ROC curve evaluates the performance of the machine learning classifiers under different threshold values. Even though default threshold of 0.5 is used to determine the class of a sample in binary classifier– a sample with 0.51 error probability is marked with error label– one can configure the threshold value to discover a sweet spot for TPR and FPR scores. Since we trained the machine learning models for each disk model separately, weights are used to calculate the average result for ROC curve. The weights are proportional to total number of uncorrectable errors in disk models.

Naive Bayes, Logistic Regression, and Support Vector Machine perform poorly as they cause 85% FPR to achieve 90% TPR. High FPR value is a major impediment in the implementation of potential solutions in most classification problems as it increases the cost of actions. As we aim to reduce the I/O overhead of end-to-end integrity verification for file transfer by calculating file checksum on disk data only when a disk is considered to be high-risk, high value of FPR would cause a significant reduction in I/O savings. Therefore, it is important to keep FPR close to zero to maximize the benefit of probabilistic integrity verification for file transfers.

While Neural Network yields better performance than Support Vector Machine,

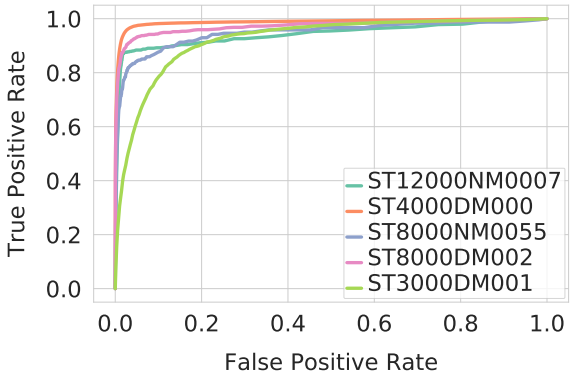



Figure 4.6: ROC curve for different disk models using Random Forest classifier.

its FPR score is still more than 60% for 90% TPR. It, however, can achieve close to 80% TPR with less than 15% FPR. On the other hand, Random Forest classifier yields 95% TPR with less than 5% FPR, outperforming other models (except Decision Tree) by a significant margin. Surprisingly, Decision Tree also yields competitive results compared to its ensemble counterpart. For instance, it acquires 93% FPR with 6% FPR. In the rest of the analysis, we use Random Forest classifier due to its high TPR and low FPR performance.

We further investigate the performance of the Random Forest (RF) classifier for different disk models in Figure 4.6. It is clear that the RF classifier achieves more than 95% TPR for most disk models while keeping FPR less than 5%. It yields more than 90% TPR with less than 3% FPR for the largest two disk models, ST12000NM007 and ST4000DM000. On the other hand, 90% TPR causes 14% and 20% FPR for ST80000NM0055 and ST30000DM001 disk models, respectively. Moreover, as mentioned Section 4, ST30000DM001 disk model was removed from the cluster in 2015 after developing a large number of uncorrectable errors.

# Chapter 5

# Avoiding Uncorrectable Errors

In this chapter, we leverage high accuracy prediction models to introduce two application scenarios where early prediction of uncorrectable errors can be used to reduce the workload on storage system or minimize the impact of uncorrectable errors. First, we demonstrate that one can reduce I/O workload of storage systems by eliminating integrity-verification for file transfers when disks-in-use are not expected to develop an error. In a nutshell, integrity verification– upon the completion of the file transfer– requires both sender and receiver to read files back from storage to calculate and compare their checksum in order to capture data corruption that might happen during network transmission or disk write. This process, however, incurs significant I/O overhead to file systems in addition to slowing down the transfer process by up to $2x$ [8], [15]. We, therefore, introduce probabilistic integrity verification to calculate file checksum on cache data when disks– that are used to store file data– do not exhibit the symptoms of uncorrectable errors. By doing so, one can reduce I/O load on storage systems significantly in addition to reducing the runtime of integrity verification process. Experimental results show that probabilistic integrity verification can save up to 97% of integrity verification-related I/O of file transfers while capturing

more than 90% of uncorrectable errors.

Second, we take advantage of high accuracy prediction of uncorrectable errors to exclude high-risk disks from taking part in write operations to reduce the workload on them, thereby lower the likelihood of uncorrectable bit errors. Even if some errors cannot be avoided, redirecting write operations from high-risk ones to low-risk ones will reduce the amount of data at risk, alleviating the impact of the errors. On the other hand, excluding too many disks imposes the risk of overloading the remaining disks and potentially increasing their error probability. Our analysis shows that one can reduce the amount of I/O on disks with uncorrectable errors by up to 80% while increasing load on the low-risk disks by less than 5%.

## 5.1 Probabilistic Integrity Verification

File transfers are vulnerable to silent data corruption as existing resiliency control mechanisms lack the robustness needed to capture silent errors that can happen while transmitting data in the network or while writing it to disk. For instance, TCP uses 16-bit checksum to capture data corruption but it fails to detect errors once in 16 million to 10 billion packets [29], which is not rare in today's high-speed networks that operate at the speed of hundreds of gigabits per second. Moreover, data can also be corrupted at the end systems due to memory and disk corruptions. For instance, undetected write errors can alter multiple bits at unintended disk sectors due to misaligned write head [6], [12], [24], [26], causing corrupted data to be stored in disk. Although Data Integrity Field has been proposed by T10 committee (aka T10 DIF) to ensure the integrity of disk write operations, it requires hardware support thus it is not necessarily available in all storage systems. Yet, it does not protect all types of disk errors, such as dropped writes [12].

Therefore, end-to-end integrity verification is proposed to ensure the integrity of file transfers, which works as follows: A file is first read from the disk at the sender and transferred to the receiver using a transfer application such as FTP or GridFTP. Then, the checksum of the original file at the sender and the transferred copy at the receiver are computed using a secure cryptographic hash function, such as SHA256. Finally, the receiver sends the checksum value it calculated to the sender to compare against the sender version. If they match, then the transfer is marked as successful. Otherwise, the transferred copy of the file on the receiver side is deemed corrupt and the file is transferred again.

### 5.1.1 I/O Overhead Analysis

End-to-end integrity verification offers a robust solution against uncorrectable disk errors by catching undetected write errors that might happen while writing files to disks on the receiver side. However, it imposes significant I/O and computation cost due to checksum calculation. For example, transfer of 100TB file with integrity verification will incur 100 TB read I/O load on source and destination file system in addition to 100TB read on the sender and 100TB write on receiver caused by transfer operation.

To demonstrate this, we transferred a $200GB$ file between two servers in the same local area network using Globus transfer service, which, by default, checks the integrity of file transfers. We monitored read I/O throughput on the sender and receiver sides using `sar` utility and plotted cumulative values in Figure 5.1.

Even though we submitted the transfer job at $t = 0s$, no network activity is observed until $t = 400s$, during which disk read I/O value for the sender gradually increases to $200GB$. When the network transfer starts, the sender read I/O rate continues to increase and hit to $400GB$ at $t = 712s$. Once the transfer of the file is

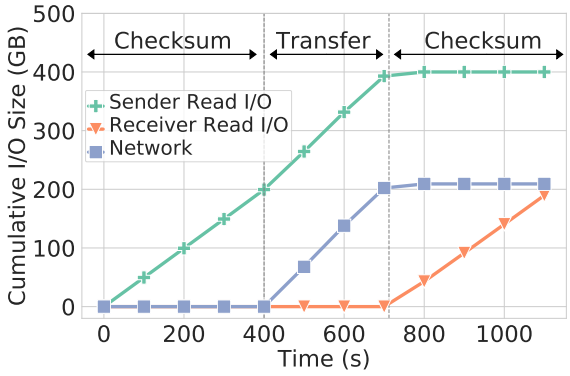



Figure 5.1: Cumulative disk and network I/O rates for the transfer of a 200GB file using Globus transfer service.

completed, receiver disk read I/O rate starts to rise and reaches $200GB$ at $t = 1123s$. As a result, both sender and receiver servers experience $200GB$ I/O load on file system due to integrity verification as it involves reading files from disk after (or before for sender) the transfer to calculate their checksum. Considering the fact that many science projects move terabytes of data every day between high-performance computing facilities, integrity verification introduces a non-negligible amount of I/O pressure on file systems.

While it is possible to compute the checksum of files while reading them to transfer at the sender side (or while writing them at the receiver side) to avoid rereading files for checksum calculation, it will prevent the detection of undetected write errors (on receiver), thus it is not considered "true" end-to-end integrity verification. Moreover, some data repositories store the checksum of files to not to recalculate them every time files are transferred. Although this will eliminate the need to calculate the checksum on the sender, receiver still needs to read the transferred files back from disk to compute their checksum to capture undetected write errors that might have happened while writing the files. Furthermore, calculating the checksum of files ahead of time may not be a feasible option when files are transferred right after they are produced– a typical scenario for many streaming workflows to enable near real time

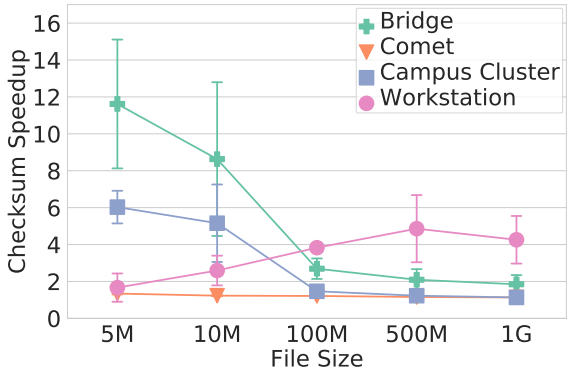



Figure 5.2: Checksum calculation speedup when it is performed on memory copy of data over disk copy.

data processing. In addition, it increases transfer times considerably due to requiring to compute and verify checksum of files after they are transferred.

Figure 5.2 compares the ratio of speedup of checksum calculations for different file sizes when it is performed over cache data over disk data. We can observe that running integrity verification on memory copy of data can speed up the process by 1.5-8.5 times. Thus, running checksum computation on memory before writing data to disk if the disk does not show error symptom would significantly lower the cost of integrity verification for file transfers. As a result, while is it critical to capture silent errors to prevent permanent data loss, integrity verification incurs significant overhead by creating read I/O load on storage system and degrading effective transfer rates.

It is therefore important to address I/O overhead of end-to-end integrity verification process of file transfers to reduce I/O workload of file systems and dedicate limited I/O bandwidth to computing jobs. To achieve this goal, we introduce probabilistic integrity verification that, instead of reading data from disk, uses cache data on memory to calculate file checksums when disks are predicted to have low risk of uncorrectable errors by the models. For example, if receiver file system splits a file into two disks with 0.8 and 0.01 uncorrectable error probability values, the proba-

bilistic integrity verification can use cache-based checksum calculation for the portion of the file on the second disk (the one with probability of 0.01) due to its low error probability, reducing the overhead of integrity verification. On the other hand, since the first disk has a high error probability, integrity verification can be executed using the disk copy of data for checksum calculation to capture potential errors.

### 5.1.2 Problem Formulation

In this section, we define the problem of selecting disks to run integrity verification on disk data to find and recover from uncorrectable bit errors while reducing the I/O overhead of integrity verification process. We first estimate the error probability of a write operation when probability of error is unknown for disks. To do so, we rely on the average undetected bit error rate of disk drives, referred as $UBER$. Although this information is not released by disk manufacturers, previous studies estimate $UBER$ (stemmed from undetected write error) to be around $10^{-15}$ (once in $125TB$) for enterprise disks and flash drives [24], [33]. Then, the probability of an uncorrectable write error to take place when writing $b$ bits to a disk becomes $b * UBER$[8]. As striping is commonly used in parallel file systems (both RAID and file system level), files are typically distributed over multiple disks. Thus, error probability for a file transfer that involves $N$ disks can be calculated by

$$E(T) = 1 - \prod_{i=1}^{N}(1 - b_i * UBER) \tag{5.1}$$

which can be approximated to

$$E(T) \approx \sum_{i=1}^{N} b_i * UBER \tag{5.2}$$

where $b_i$ is the amount of data written to disk $i$. We aim to take advantage of the error prediction models to estimate the likelihood of uncorrectable errors for each of $N$ disks and execute integrity verification on disk data only when the estimated error probability is higher than a user-defined threshold. It is important to note that although we check for undetected write errors only for high-risk disks, integrity verification can still be executed on cache data (on memory) for low-risk disks to capture possible data corruption that might occur during the network transfer.

Let $p_i$ be the probability of $i^{th}$ disk to develop an uncorrectable error due to undetected write error within next day, $i = 1, \ldots, N$ as estimated by the *Random Forest model*. Let $b_i$ be the data size we write on disk $i$ and $D_i$ is the total write I/O size for disk $i$ in a given day, where $b_i \leq D_i$ for $i = 1, \ldots, N$. Therefore, the probability of a transfer operation to be exposed to an uncorrectable error on disk $i$ becomes

$$E(i)_{model} = b_i \frac{p_i}{D_i} \tag{5.3}$$

where $\frac{p_i}{D_i}$ is analogous to $UBER$ as it represents the probability of uncorrectable error to affect a unit write operation (i.e., one bit) as we assume $p_i$ to represent exactly one uncorrectable error to happen in a day during a total of $D_i$ bits are processed. We then multiply this by the data size that the transfer operations issued on the disk, $b_i$. Therefore, the probability of at least one uncorrectable error to affect the given transfer $T$ can be estimated by

$$E(T)_{model} = 1 - \prod_{i=1}^{N} (1 - b_i \frac{p_i}{D_i}) \approx \sum_{i=1}^{N} b_i \frac{p_i}{D_i} \tag{5.4}$$

Note that $E(T)_{model}$ will be smaller than $E(T)$ if disks used for a file transfer are on average predicted to be low risk (i.e., $p < 0.5$) by the model. As a result, one can calculate the difference between baseline error prediction and model-based error

prediction by

$$k = \frac{E(T) - E(T)_{model}}{E(T)_{model}} \tag{5.5}$$

then determine to execute integrity verification for some disks if the *improvement ratio*, $k$, is smaller than a desired value. Although executing integrity verification for all disk would meet user defined threshold, it will increase I/O overhead on file system. We, therefore, aim to strike a balance between mitigating uncorrectable errors and reducing the overhead of integrity verification by finding a right set of disks.

For a given set $S = \{1, \ldots, N\}$, the set of disks used to write data in given transfer task $T$, we define $E_{S_c}(T)_{model}$ to be the probability of error obtained by running disk-based integrity verification on a subset of $S$, $S_c$, and calculate by

$$E_{S_c}(T)_{model} = \sum_{i \in S \setminus S_c} b_i \frac{p_i}{D_i} \tag{5.6}$$

where $S_c \subset S$. Let $S_1, S_2, \ldots, S_m$ be the subsets of $S$ which can be found by iterating over all the subsets of $S$. As multiple subsets of $S$ can satisfy the improvement ratio requirement, we define a cost function to find the one that yields maximum I/O saving as

$$C(S_c) = \alpha \sum_{i \in S_c} b_i + (1 - \alpha) \sum_{i \in S \setminus S_c} b_i \frac{p_i}{D_i} \tag{5.7}$$

where $0 \leq \alpha \leq 1$ for each subset $S_c$ of $\{1, \ldots, N\}$. The cost function consists of two parts as total data size we run the integrity verification on disk data, $\sum_{i \in S_c} b_i$, and predicted uncorrectable error rate, $\sum_{i \in S \setminus S_c}$ for disks that we do not run integrity verification on disk data. $\alpha$ is then used to find a balance between I/O saving rate and improvement ratio. When $\alpha$ is close to 1, the cost function will prefer a subset that runs integrity verification on a minimum amount of data that can satisfy the improvement factor. On the contrary, when it is close to 0 will choose a subset that

returns minimum error probability. Although we only evaluate the cost of disk subsets that satisfy the improvement factor, a cost function with $\alpha < 1$ gives an opportunity to reduce the error probability further down. Once the $\alpha$ is set to a desired value in accordance with the application requirement, we can then search for a set of disks, $S_c$, that returns the minimum cost value.

**Brute Force Solution:** A straightforward solution to the problem of finding the optimal set of disks to run integrity verification on disk data to minimize the cost function is to iterate over all subsets of a set $S$ and find the one with the minimum cost. However, this approach takes a long time to execute when the number of disks used in a transfer task, $N$ is more than 20. Specifically, while brute-force finds a solution in 0.18 seconds when 8 disks used, it takes 123 seconds for 24 disks. Given that a significant portion of scientific transfers involve tens of files and use hundreds of disks to store the data [16], brute force solution is impractical to apply in real-time. We therefore introduce a greedy solution to find a close-to-optimal solution in a polynomial time.

**Greedy Solution:** Instead of checking all possible subsets, the greedy solution follows a step by step approach to find the optimal. It first checks if running integrity verification on cache data for all disks ($S_0 = \{\}$) meets the user-defined improvement ratio. If it does, then the search will be terminated and the empty set will be returned as a solution. Otherwise, the greedy will select a disk for which running the integrity verification on disk data would satisfy the improvement ratio while lowering the cost function the most using

$$S_1 = \underset{S_c \in \{\{i\} \mid i \in S\}}{\arg\min} C(S_c). \tag{5.8}$$

If $E_{S_1}(T)_{model}$ meets the improvement ratio condition, then the greedy stops; other-

wise, continues the process by searching another disk that minimizes $C(S_c)$ together with the disk indexed by the element of $S_1$. It will continue the search until a set to satisfy the improvement ratio is found or all disks are selected. Since the greedy approach only selects one disk at any step by evaluating all available disks, its time complexity becomes $O(N^2) = N + (N-1) + (N-2) + \cdots + 2$, where $N$ is the number of disks used in the transfer. The execution time of the greedy solution is 0.18 seconds for 24 disks and 0.25 seconds for 100 disks.

### 5.1.3 Simulation Results

We evaluate the performance of probabilistic integrity verification using file transfer logs of Comet supercomputer located at San Diego Supercomputing Center in California. The logs report start time, file size, user id, and end time of transfers that took place in March of 2017. Figure 5.3 to 5.5 presents the characteristics of transfer logs. On average, there are $150K$ file transfers each day carrying up-to $66TB$ data. We find that 80% of all transfer jobs include less than 200 files while around 5% of all transfer jobs contain more than $1,000$ files. We also notice that 90% of all transfers carry files that are smaller than $10MB$. These findings match with earlier reports regarding the file transfer characteristics between HPC clusters [16], thus the logs can be used to represent the characteristics of file transfers between HPC facilities.

Since we do not have access to SMART reports for Comet, we simulated transfer logs on Backblaze cluster. To simulate the transfers, we first pick a transfer from the transfer log and randomly select $N$ disks from the BackBlaze dataset. The number of disks, $N$, depends on file system settings such as RAID configurations (e.g., RAID level, number of drives in a RAID array, and RAID stripe size) and file system striping settings (i.e., stripe count which refers to the number of storage servers used to distribute a file). Without loss of generality, we set the RAID level to 0 (i.e., no

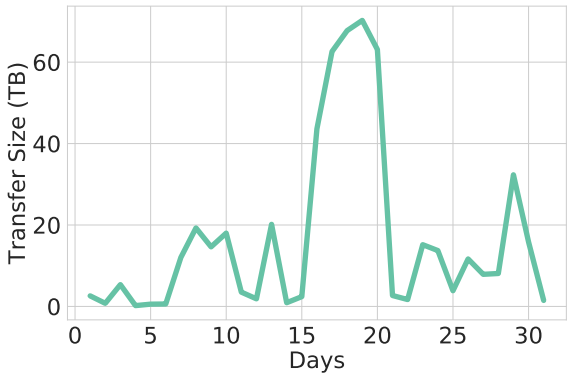



Figure 5.3: Daily Transfer Rate in Comet Dataset

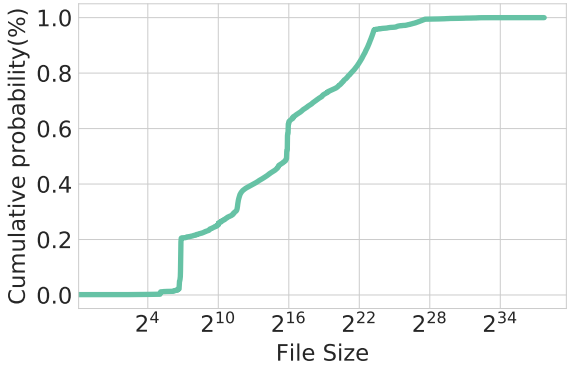



Figure 5.4: File Size Distribution in Comet Dataset

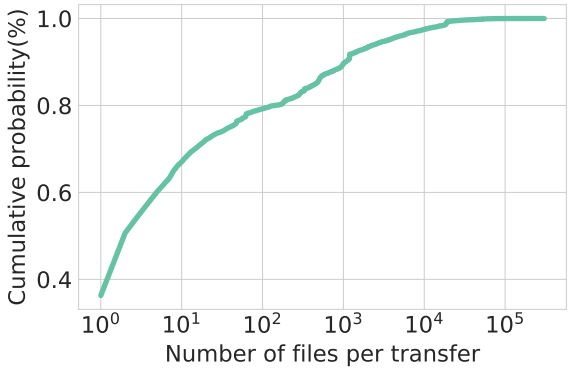



Figure 5.5: File Count Distribution in Comet Dataset

parity or mirroring), the number of disks in RAID array to 8, RAID stripe size to $128KB$, and file system stripe count to 1, as default settings in simulations. Thus, the number of disks used for each file transfer is by default set to 8. Note that since we do not evaluate the recoverability of corrupted data, RAID level does not affect the performance of the proposed solution. Once the disks are selected for a file transfer, we then use probabilistic integrity verification to determine whether or not to execute integrity verification on disk data or cache data for each disk. If the transfer task involves multiple files, we select $N$ disk for each file separately. We repeat this process for all transfers in the transfer log. As the transfer log contains around $150K$ file transfers in a day, we select 1.2M (150K transfer $\times 8$) disks in total, causing each disk to be used by 31 ($\frac{150K \times 8}{38K}$) to 300 ($\frac{150K \times 8}{4K}$) times each day based on the disk model[1]. Simulating the transfer logs for an entire month leads each disk to be used by $240 - 2,400$ times.

To evaluate the performance of probabilistic integrity verification, we define two metrics as *coverage ratio* and *I/O save ratio*. The coverage ratio represents the percentage of data size that is protected against undetected write errors by running integrity verification on disk data. Note that it only considers the portion of data that is directed to disks with uncorrectable errors. Since the number of disks with error only constitutes 0.02% of all disks, this results in approximately 0.02% transfer data to be written to disks with error, assuming equal distribution of workload over available disks. I/O save ratio, however, calculates the percentage of integrity verification-related I/O that is avoided by executing integrity verification on cache data in memory. As an example, if integrity verification is executed on disk data for all disks, the coverage ratio will become 100% while I/O save ratio becomes 0%. The goal of probabilistic integrity verification is therefore to increase I/O save ratio while

[1]Total number of disks in disk models vary between $4,000$ and $38,000$

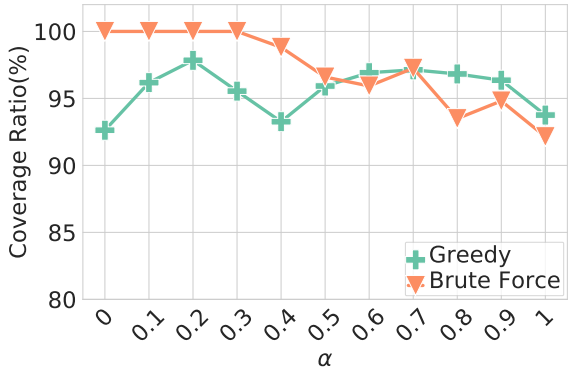



Figure 5.6: Comparison of brute force and greedy-based probabilistic integrity verification in terms of coverage ratio

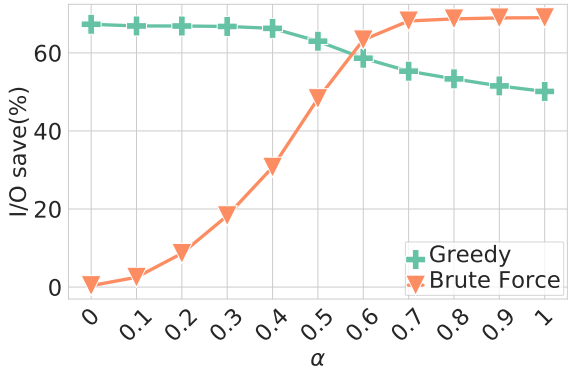



Figure 5.7: Comparison of brute force and greedy-based probabilistic integrity verification in terms of I/O save ratio

keeping coverage ratio as high as possible to minimize the likelihood of transfers to be exposed to uncorrectable errors as formalized in Equation 5.7. Unless otherwise specified, we set the improvement ratio, $k$, to 90%, which requires finding a set of disks, $S_c$ that will lower the risk of uncorrectable errors by at least 10 times compared to baseline error probability, $E(T)$ as described in Equation 5.2.

Figure 5.6 and 5.7 compares the performance of brute force and greedy approaches for varying $\alpha$ values as defined in Equation 5.7. We can see that $\alpha$ value has a limited impact on the coverage ratio of the greedy solution as improvement ratio 90% is sufficient enough to check high-risk disks against uncorrectable bit errors. On the

other hand, the brute force approach does not save any I/O when $\alpha = 0$ as it prefers the solution with the smallest $E(T)_{model}$, which happens when integrity verification is executed on disk data for all write operations. In exchange, it yields 100% coverage ratio by capturing all uncorrectable errors with disk-level integrity verification. It, however, achieves 69% I/O saving while offering 92% coverage ration when $\alpha$ is set to 1. Larger $\alpha$ values lead to decrease in I/O save rate by around 15% for the greedy method. This is because the cost function will first run integrity verification for small I/O writes to keep the data size, $\sum_{i \in S_c} b_i$, small, but they are unlikely to satisfy the improvement ratio criteria. Consequently, the greedy solution will then move to disks with larger transfer I/O size, resulting in a suboptimal solution. As an example, if we are given a list of integers {1,2,10} and asked to find a subset whose sum of elements is the smallest value that is greater than 5, then the greedy approach will end up with a subset that contains all three integers due to selecting the smallest number in each step. However, the optimal solution would be the subset with only one element, {10}. As a result, the greedy solution might yield suboptimal results when $\alpha$ value is close to 1. We leave the optimization of the greedy under such circumstances as a future work and use $\alpha = 0$ in the rest of the experiments. The greedy approach with $\alpha = 0$ yields competitive results compared to brute force with $\alpha = 1$ (67% vs 69% in I/O save ratio and 92% vs 93% in coverage ratio), therefore we used the greedy approach in the rest of the analysis due to the prohibitive computational cost of the brute force solution.

Figure 5.8 compares the execution time to find a solution for a file transfer with increasing number of used disks. Note that although the number of disks used for a single file is set to 8 by default, the number of disks used for a transfer job could be large if the job contains multiple files or file system striping is used. The computation time for brute force reaches to 100 seconds when $M = 24$, whereas the greedy solution

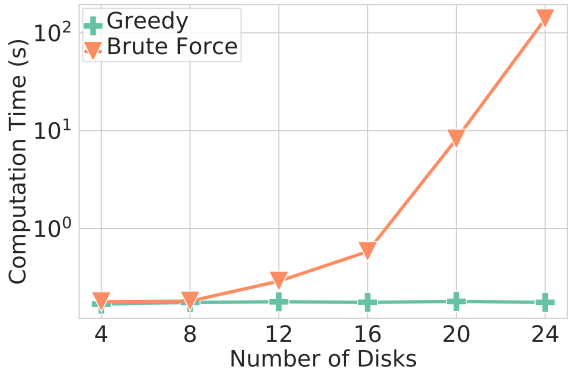
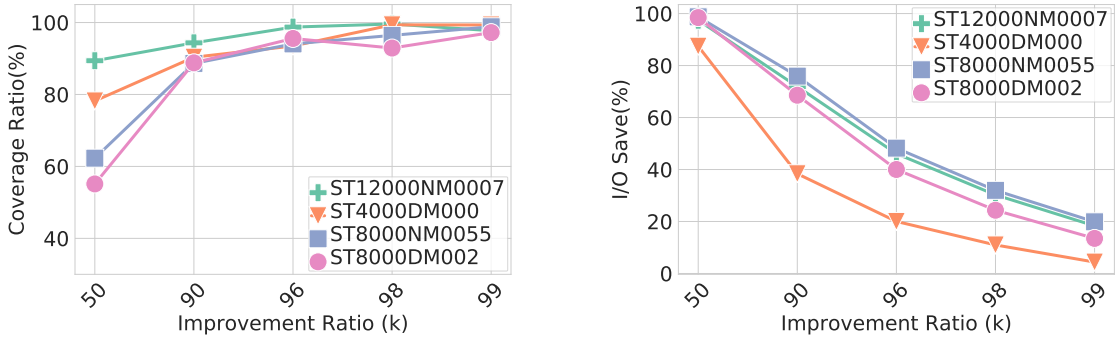



Figure 5.8: Comparison of brute force and greedy-based probabilistic integrity verification in terms of execution time




(a) Coverage Ratio

(b) Saved I/O Ratio

Figure 5.9: The performance analysis of probabilistic integrity verification algorithm when tested with various improvement ratios.

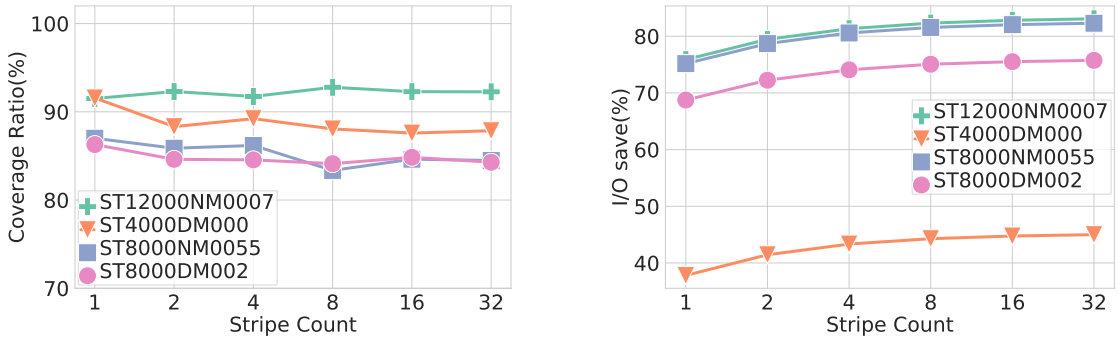
can return a solution in the order of microseconds.

Figure 5.9 demonstrates the impact of improvement ratio on the performance of probabilistic integrity verification. It is clear that increasing improvement ratio increases the coverage ratio as we are running disk-level integrity verification for more disks to lower the error rate. Checking more disks, in turn, leads to smaller I/O saving ratios. We observe that probabilistic integrity verification with improvement ratio of 90% achieves $85\% - 95\%$ coverage ratio while saving $40\% - 75\%$ integrity verification-related I/O for different disk models. When the improvement ratio is set to 50%, I/O saving ratio increases to 97% while still achieving 90% coverage ratio for

disk model ST120000NM0007. For the same disk model, the greedy solution can save 30% I/O while covering more than 99% transfers against uncorrectable errors. The difference between the performance of probabilistic integrity verification for different disk models can be attributed to the performance of Random Forest classifier as it is able to yield smaller FPR score for the same TPR score for ST120000NM0007 compared to the other models.

We also assessed the impact of file system striping on the performance of probabilistic integrity verification. The striping count defines the number of storage servers used to distribute a file. It complements RAID striping by increasing the number of disks used to store files. Lustre file system defines default Striping count is set to 1, but allows users to define striping count for files [25]. Stripe count of 2 or higher values are typically suggested for large files, so we evaluated its impact for files whose size ranges between 1GB and 300GB in Figure 5.10. We observe that probabilistic integrity verification save 10% higher I/O compared to the default striping count 1. This can be attributed to the fact that as large are split to more disks, it allows probabilistic integrity verification to make more precise disk selection decisions to meet the improvement factor. Coverage ratio, however, declines $7-8\%$ for most disk models as striping count is increased.

In addition to lowering I/O overhead, the probabilistic integrity verification can also speedup file transfers especially when integrity verification process is the bottleneck of whole process. To confirm this, we transferred a dataset with 10 1GB files between two servers in a network with 10Gbps bandwidth and 30ms round trip time. We measure the transfer time when integrity verification is executed with various I/O saving ratios. For instance, 10% I/O saving ratio will run integrity verification on disk data for 90% of transfer data, while the rest will be computed using cache data. The results (as given in Table 5.1) show that probabilistic integrity verification does




(a) Coverage Ratio

(b) Saved I/O Ratio

Figure 5.10: Performance analysis of probabilistic integrity verification when file system-level striping count is varied between 1 and 32. Large stripe counts increase I/O save ratio by around 10% while causing 5-7% decrease in coverage ratio for some disks.

Table 5.1: Impact of probabilistic integrity verification on transfer time

| | I/O Saving Ratio | | | |
|---|---|---|---|---|
| | 0% | 10% | 50% | 90% |
| Time (sec) | 360.8 ± 5.3 | 353.6 ± 6.8 | 263.0 ± 1.7 | 168.5 ± 2.0 |

not only help to reduce I/O overhead on storage system but also lowers transfer times significantly; 53% for 90% I/O save.

## 5.2 I/O Redirection to Avoid Uncorrectable Errors

Probabilistic integrity verification improves the resilience of file transfer I/O by verifying the integrity of write operations, however, it does not help to prevent uncorrectable errors. In this section, we propose to mark high-risk disks as "unavailable" for new write operations to reduce I/O load on them. Although eliminating read I/O requests would also help to minimize I/O load, that would require moving data out of high-risk disks (or recreating missing blocks in the case of RAID storage), thus would cause significant overhead on the system. Disabling high-risk disks for write I/O does not only help to lower the risk of developing uncorrectable errors but also protects

write operations against silent errors by issuing them to healthy disks. It is important to note that redirecting traffic from high risk disks to low risk disks can increase the failure probability of low risk disks, thus it is important to keep the overhead on the "online" disks low.

### 5.2.1 Problem Formulation

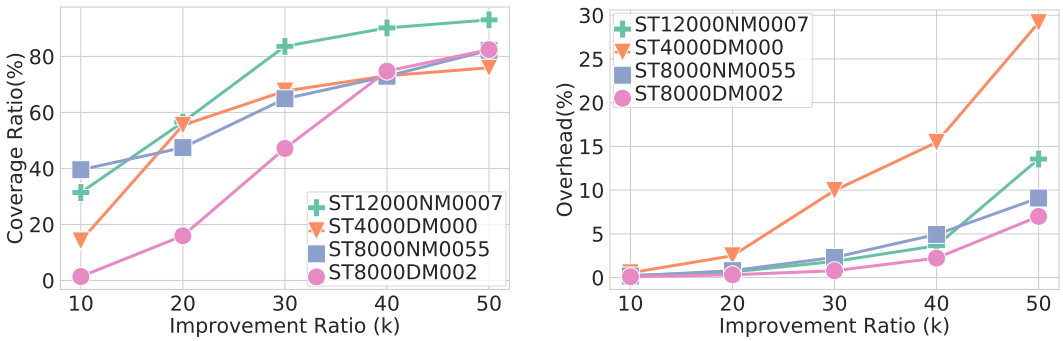
We first formulate the error probability for a cluster of $N$ disks that handles $D$ bits of daily write I/O load. Then, the probability of having at least one uncorrectable error in the cluster can be calculated as

$$E(C) = 1 - \prod_{i=1}^{N} p_i \tag{5.9}$$

where $p_i$ is the error probability for disk $i$ as estimated by the Random Forest model and $D_i$ is the amount of write I/O issued to the disk. Note that different from Equation 5.3, the term $(\frac{b_i}{D_i})$ does not exist anymore as it returns 1 when I/O in consideration, $b_i$, equals to total I/O on disk, $D_i$. Moreover, while Equation 5.3 only iterates over the disk that are involved in any particular transfer task, Equation 5.9 considers all disks in the cluster hence estimated error values are significantly higher. To lower $E(C)$, we can deactivate a set of high-risk disks, $S = \{1, \ldots, M\}$ for write operations until we can reevaluate their risk next day using the SMART metrics. Taking some disks offline will increase the load on the online disks by

$$T = \frac{\sum_{i \in M} D_i}{N - M} \tag{5.10}$$

bit and potentially increases their error probability. Although it is hard to calculate the impact of increased workload on error probability on active disks, we incorporate




(a) Coverage Ratio of Uncorrectable Errors (b) I/O Overhead on Low-Risk Disks

Figure 5.11: The performance analysis of I/O redirection experiments for various improvement ratio targets.

it by increasing their error probability by the percentage of increase in the I/O load. Thus, the uncorrectable bit error probability for the cluster becomes

$$\hat{E}(C) = 1 - \prod_{i \notin S} p_i \frac{D_i + T}{D_i} \tag{5.11}$$

with improvement ratio

$$k = \frac{E(C) - \hat{E}(C)}{E(C)} \tag{5.12}$$

We implemented a solution to find the set of disks to satisfy the improvement ratio condition. It works by sorting the disks in descending order by their error probability and adding them to the offline list, $S$, one by one until the estimated error rate $\hat{E}(C)$ satisfies the target improvement ratio. Note that while the cost function of probabilistic integrity verification considers data size (Eq 5.7), it is irrelevant in the I/O redirection scenario since total I/O size will not be affected by the proposed actions.

### 5.2.2 Simulation Results

Figure 5.11 shows the coverage ratio and overhead results when I/O redirection is applied for target improvement ratio of 10% to 50%. The coverage ratio indicates the percentage of I/O that is redirected from disks with uncorrectable error to disks without error. Figure 5.11(b) shows the ratio of increased I/O on the low risk disks due to excluding high risk disks from write operations. The results show that one can save more than 50% I/O from being exposed to uncorrectable error while increasing the overhead of low risk disks by less than 5%. For ST12000NM007, 80% I/O coverage ratio can be achieved with less than 5% increase in overhead on the active disks.

## 5.3 Deployment Challenges and Opportunities

In this section, we lay out potential deployment options for the proposed application scenarios. The error prediction models we drive in this thesis rely on SMART monitoring system which is supported by the most disk manufacturers. Thus, system administrators can configure to collect and publish daily SMART metrics such that pre-trained models can be used to decide whether or not a disk is likely to develop an uncorrectable error within next 24 hours. Since SMART reports contain less than 60 attributes (actual number depends on the disk model and SMART version), the storage footprint to store daily SMART logs would be negligible compared to the scale of today's large-scale parallel file systems. As an example, the size of daily SMART reports of BackBlaze dataset for $120K$ disks is 32.5MB.

A potential deployment challenge would be to collect and train a model for each disk type. While this is a common challenge for any supervised learning-based prediction model, we show in Figure 5.12 that one can use a prediction model trained for one disk type to make predictions for another disk type. In the figure, we eval-

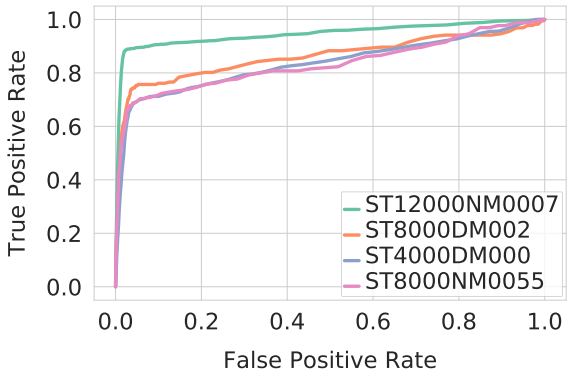



Figure 5.12: ROC curve for the Random Forest model when trained with ST12000NM0007 disk model and tested with others.

uate the performance of Random Forest model when it is trained with disk model ST12000NM0007 and tested against other disk models. It is clear that the model achieves more than 70% TPR with less than 5% FPR for all three disk models. For 90% TPR, the model returns 70% FPR for all disk models, which can be used to save up to 30% of I/O overhead induced by file transfer integrity verification. Although a more detailed analysis is required using more disk models from different manufacturers, these preliminary results indicate that one can potentially start with a pre-trained prediction model and re-train it as data from actual disks becomes available over time.

Another challenge regarding the application of I/O redirection is the implementations of marking high-risk disks unavailable for write operations until their risk declines. We believe that this can be implemented in two ways. First, when RAID arrays are used with mirroring or parity, then it is possible to mark high-risk disks as write-protected to prevent write I/O without affecting the operation of the RAID array. Second, the storage servers in parallel file systems can also be marked read-only to prevent write operations. For example, Object Storage Servers in Lustre file system can be temporarily deactivated to only allow read I/O. Thus, this option can be considered when multiple disk drives of the same storage server are selected to

mark read-only by the proposed I/O redirection algorithm.

# Chapter 6

# Conclusion and Future Work

Ensuring end-to-end data integrity while keeping the system over-head low is one of the crucial requirements of the exascale computing era. It is most significant for applications that can not tolerate silent data corruption. Uncorrectable errors pose a threat to data integrity in storage systems as they may result in complete data loss when not handled properly. Even with proper control mechanisms (i.e., file system level checksumming), the ability to predict uncorrectable errors allows users and systems administrators to take precautionary actions such as taking high-risk disks offline or verifying the integrity of I/O operations. In this work, we analyzed $150M$ SMART logs from $143K$ disks over the period of 68 months to gain insights into the characteristics of uncorrectable errors. We further derived high accuracy prediction models to estimate the occurrence of uncorrectable errors using daily SMART logs. Among several machine learning models, we find that Random Forest classifier yields the best performance compared with over 95% true positive rate in exchange of less than 5% false negative rate. We then incorporated the prediction model into two application scenarios. In the first use case, high-accuracy error prediction led to 97% reduction in I/O overhead caused by the integrity verification process of wide area

file transfers. In the second application scenario, we introduced I/O redirection solution to divert write I/O operation from high-risk disks to low-risk one to avoid write operations being affected by uncorrectable errors. We find that one can lower I/O issued on disks with error by up to 80% while increasing the load on the other disks by less than 5%.

As future work, we will look deeper into the impact of disk age on on the occurrence of uncorrectable disk errors such that this information can be utilized tom improve the accuracy of prediction models. Moreover, we will evaluate the impact of using different $\alpha$ values in out cost function to see how the proposed algorithms work in these scenarios. We will also work on prediction models that support transfer learning to facilitate the deployment of proposed application scenarios in cases historical data is not available. Finally, we will test the proposed model and associated application scenarios using actual devices to validate our findings and pave the way for deployment to production systems.

# References


[1] A. Alhussen and E. Arslan, "RIVAChain: Blockchain-based integrity verification for file transfers," in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, 2020.

[2] B. Allen, "Monitoring hard disks with smart," *Linux Journal*, vol. 2004, no. 117, p. 9, 2004.

[3] E. Arslan and A. Alhussen, "A low-overhead integrity verification for big data transfers," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 4227–4236. DOI: 10.1109/BigData.2018.8622116.

[4] *Backblaze*, https://www.backblaze.com, 2019.

[5] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, and B. Schroeder, "An analysis of data corruption in the storage stack," *ACM Trans. Storage*, vol. 4, no. 3, Nov. 2008, ISSN: 1553-3077. DOI: 10.1145/1416944.1416947. [Online]. Available: https://doi.org/10.1145/1416944.1416947.

[6] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 289–300, Jun. 2007, ISSN: 0163-5999. DOI: 10.1145/1269899.1254917. [Online]. Available: https://doi.org/10.1145/1269899.1254917.

[7] B. Charyyev, A. Alhussen, H. Sapkota, E. Pouyoul, M. H. Gunes, and E. Arslan, "Towards securing data transfers against silent data corruption," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 262–271. DOI: 10.1109/CCGRID.2019.00040.

[8] B. Charyyev and E. Arslan, "RIVA: Robust integrity verification algorithm for high-speed file transfers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1387–1399, 2020. DOI: 10.1109/TPDS.2020.2966616.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002, ISSN: 1076-9757. DOI: 10.1613/jair.953. [Online]. Available: http://dx.doi.org/10.1613/jair.953.

[10] I. Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011. DOI: 10.1109/MIC.2011.64.

[11] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W.-k. Liao, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016, ISSN: 1384-1076. DOI: https://doi.org/10.1016/j.newast.2015.06.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138410761500069X.

[12] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao, "Undetected disk errors in raid arrays," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 413–425, 2008. DOI: 10.1147/rd.524.0413.

[13] R. Kettimuthu, Z. Liu, D. Wheeler, I. Foster, K. Heitmann, and F. Cappello, "Transferring a petabyte in a day," *Future Generation Computer Systems*, vol. 88, pp. 191–198, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.05.051. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18302280.

[14] A. Krioukov, L. N. Bairavasundaram, G. R. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea, "Parity lost and parity regained," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08, San Jose, California: USENIX Association, 2008.

[15] S. Liu, E. Jung, R. Kettimuthu, X. Sun, and M. Papka, "Towards optimizing large-scale data transfers with end-to-end integrity verification," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 3002–3007. DOI: 10.1109/BigData.2016.7840953.

[16] Z. Liu, R. Kettimuthu, I. Foster, and N. S. V. Rao, "Cross-geography scientific data transferring trends and behavior," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18, Tempe, Arizona: Association for Computing Machinery, 2018, pp. 267–278, ISBN: 9781450357852. DOI: 10.1145/3208040.3208053. [Online]. Available: https://doi.org/10.1145/3208040.3208053.

[17] G. K. Lockwood, S. Snyder, S. Byna, P. Carns, and N. J. Wright, "Understanding data motion in the modern hpc data center," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, 2019, pp. 74–83. DOI: 10.1109/PDSW49588.2019.00012.

[18] F. Mahdisoltani, I. Stefanovici, and B. Schroeder, "Proactive error prediction to improve storage system reliability," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, Jul. 2017, pp. 391–402, ISBN: 978-1-931971-38-6. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/mahdisoltani.

[19] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 1, pp. 177–190, Jun. 2015, ISSN: 0163-5999. DOI: 10.1145/2796314.2745848. [Online]. Available: https://doi.org/10.1145/2796314.2745848.

[20] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "Ssd failures in datacenters: What, when and why?" *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 407–408, Jun. 2016, ISSN: 0163-5999. DOI: 10.1145/2964791.2901489. [Online]. Available: https://doi.org/10.1145/2964791.2901489.

[21] A. Oprea and A. Juels, "A clean-slate look at disk scrubbing," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10, San Jose, California: USENIX Association, 2010, p. 5.

[22] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Iron file systems," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 206–220, Oct. 2005, ISSN: 0163-5980. DOI: 10.1145/1095809.1095830. [Online]. Available: https://doi.org/10.1145/1095809.1095830.

[23] M. S. Rothberg, *Disk drive for receiving setup data in a self monitoring analysis and reporting technology (smart) command*, US Patent 6,895,500, May 2005.

[24] E. W. D. Rozier, W. Belluomini, V. Deenadhayalan, J. Hafner, K. Rao, and P. Zhou, "Evaluating the impact of undetected disk errors in raid systems," in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, 2009, pp. 83–92. DOI: 10.1109/DSN.2009.5270353.

[25] S. Saini, J. Rappleye, J. Chang, D. Barker, P. Mehrotra, and R. Biswas, "I/o performance characterization of lustre and nasa applications on pleiades," in *2012 19th International Conference on High Performance Computing*, 2012, pp. 1–10. DOI: 10.1109/HiPC.2012.6507507.

[26] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," *ACM Trans. Storage*, vol. 6, no. 3, Sep. 2010, ISSN: 1553-3077. DOI: 10.1145/1837915.1837917. [Online]. Available: https://doi.org/10.1145/1837915.1837917.

[27] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, Santa Clara, CA: USENIX Association, Feb. 2016, pp. 67–80, ISBN: 978-1-931971-28-7. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/schroeder.

[28] T. J. E. Schwarz, Qin Xin, E. L. Miller, D. D. E. Long, A. Hospodor, and S. Ng, "Disk scrubbing in large archival storage systems," in *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings.*, 2004, pp. 409–418. DOI: 10.1109/MASCOT.2004.1348296.

[29] J. Stone and C. Partridge, "When the crc and tcp checksum disagree," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 309–319, Aug. 2000, ISSN: 0146-4833. DOI: 10.1145/347057.347561. [Online]. Available: https://doi.org/10.1145/347057.347561.

[30] J. Wan, J. Wang, Q. Yang, and C. Xie, "S2-raid: A new raid architecture for fast data recovery," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–9. DOI: 10.1109/MSST.2010.5496980.

[31] S. Wu, H. Jiang, L. Tian, and B. Mao, "Workout: I/o workload outsourcing for boosting RAID reconstruction performance," in *7th USENIX Conference on File and Storage Technologies (FAST 09)*, San Francisco, CA: USENIX Association, Feb. 2009. [Online]. Available: https://www.usenix.org/conference/fast-09/workout-io-workload-outsourcing-boosting-raid-reconstruction-performance.

[32] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou, M. Chintalapati, and D. Zhang, "Improving service availability of cloud systems by predicting disk error," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA: USENIX Association, Jul. 2018, pp. 481–494, ISBN: 978-1-939133-01-4. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/xu-yong.

[33] Y. Zhang, D. S. Myers, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Zettabyte reliability with flexible end-to-end data integrity," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, 2013, pp. 1–14. DOI: 10.1109/MSST.2013.6558423.

[34] Y. Zhang, A. Rajimwale, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "End-to-end data integrity for file systems: A zfs case study," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10, San Jose, California: USENIX Association, 2010, p. 3.