ARTIFACT EVALUATED
USENIX
AVAILABLE

Kratos: A Robust and Comprehensive Pipeline for Detecting Various Patterns of Disk Fail-Slow Behaviors at Scale

Ruidan Li*, Erci Xu†,*, Xikang Song, Kiran Srinivasan‡, Haryadi S. Gunawi

University of Chicago †Shanghai Jiao Tong University ‡NetApp

Abstract

This paper presents a comprehensive study of fail-slow behaviors using the largest dataset available to date. We identified five fail-slow patterns, four of which were previously unexamined, and found that all state-of-the-art approaches fail to capture them fully. Hence, we build KRATOS, a comprehensive, robust, and practical detection method that accurately identifies all patterns, remains effective across diverse environments without fine-tuning, and operates efficiently at scale. On multiple RAID and non-RAID datasets (including SSDs and HDDs), KRATOS detects 32.8-50% more cases, achieves over 95% accuracy across a broad threshold range, and runs 3-128× faster than existing methods.

1 Introduction

Performance is critical in system design, as it influences efficiency, scalability, and user experience across all layers. As a result, performance monitoring has gained increasing attention in both industry and academia. For instance, both cloud and HPC communities seek fine-grained visibility at per-operation level [1–3], at I/O level [4–6], and at hardware level [7, 8], driving extensive research into performance regression prevention, detection, and mitigation [9–21].

As a result, disk fail-slow failures, where disks degrade unpredictably rather than fail outright, are receiving growing attention in both industry and academia due to the detection challenges, prevalence in production, and potential cascading impact [22–31]. Existing fail-slow detection efforts largely focus on individual case studies [32], white-box analysis [23], and identifying only the most severe cases [33, 34]. *However, few studies have attempted a systematic, quantitative characterization of fail-slow behaviors.* One reason is that fail-slow failures are often silent, misclassified as benign performance variation, or obscured by background noise. Thus, it requires a comprehensive understanding of fail-slow patterns at scale to achieve general and robust detection.

Currently, we have access to Company X's (COMX) large-scale performance data, which, to the best of our knowledge, is one of the first large-scale datasets from a major storage provider available for research.[1] The disk-level performance metrics are recorded at fine granularity (every 10 seconds), similar to the recently published Alibaba dataset with 15-second granularity [34]. Note that COMX's dataset is approximately 92 times larger in the disk performance data collection period than Alibaba's.

The availability of the new dataset allows us to close the existing gap by answering *many research questions that have not been fully addressed in the past*: (1) How do fail-slow disks behave in real-world deployments? (2) Do they exhibit similar or diverse patterns? (3) Can state-of-the-art detection methods reliably capture all of them? (4) If not, what are their limitations? (5) Can we build a more robust fail-slow detection pipeline? To answer these questions, this paper makes the following contributions.

Fail-slow pattern study. We conduct a large-scale study of fail-slow cases within COMX's dataset, covering over 55 million drive days. By examining the dataset from spatial and temporal aspects, we uncover a broader and more intricate range of fail-slow behaviors than previously recognized (§3). Our study reveals *five* major fail-slow patterns (4 more than prior work [34]), spanning from clear, self-evident anomalies to subtle, hard-to-detect failures. Moreover, we observe two noisy patterns that make many methods ineffective. Similar trends are found in the Alibaba dataset (the Open dataset), which includes both SSDs and HDDs.

Failures of existing detection. We adapt and refactor nine existing detection frameworks to assess their effectiveness against the fail-slow patterns that we have discovered. These include IASO and Perseus [33, 34], the state-of-the-art fail-slow detection methods; LSTM and PatchTST [35, 36], advanced machine learning models for time-series analysis; VAE and GVAE [9], methods used for performance debugging; CSR and Multi-Pred [37, 38], frameworks validated for disk failure prediction; and Threshold [39], a simple yet widely used approach in industry. Although each method excels within its intended scope, our analysis of this larger dataset uncovers a critical limitation: no existing algorithms, including state-of-the-art methods, can reliably detect all five fail-slow patterns. This finding highlights that fail-slow detection remains an unresolved challenge at scale.

*Corresponding authors: Ruidan Li (ruidanli@uchicago.edu) and Erci Xu (jostep90@gmail.com)

[1] The dataset is available to academic researchers under a research agreement with COMX, but is not public.

KRATOS. Given this challenge, deploying multiple frameworks to compensate is impractical, which raises a crucial question: Can a single solution address it? Since all prior methods have failed, *can we develop a comprehensive, robust, and practical method that identifies all patterns, remains effective across environments, and operates efficiently at scale?* To achieve this, we design KRATOS, a novel fail-slow detection method that overcomes these challenges via three key components: lightweight and general feature, extreme case isolation, and noise reduction for the "in-between" cases (§4).

(a) The first challenge we address is to identify the minimal feature set that can effectively capture various fail-slow patterns. We do not want to rely on very specific information or deploy multiple algorithms. For this, we performed careful feature engineering and discovered Z-Score [40] as an effective initial transformation. However, as Z-Score only captures spatial characteristics, our feature engineering combines it with temporal signals and transforms it into a 2-sigma count, leading to a lightweight and overarching feature that ensures coverage across all fail-slow patterns.

(b) The 2-sigma count itself is not actionable; system operators still need to decide thresholds before acting. Thus, the next challenge is to convert the feature (a sequence) into high-quality alerts ("true positives") for storage operators. We explored various divergence methods to compress the sequence and adopted the Kullback-Leibler divergence for its capability to preserve separation and computational efficiency in our problem domain. We then discretize the divergence into distinct risk levels based on its distribution, effectively isolating high-risk cases for prioritization.

(c) Finally, we handle the "in-between" cases (between normal and extreme) that often lead to false positives and negatives (that many existing works fail to address in the COMX dataset). We identified three distinct edge cases to handle: transient slowness (where the disk returns to normal performance within a short time), masked cases (where the disk that is next slowest in the RAID is hidden by the slowest disk), and inflated cases (from side effects of divergence-based comparisons). To handle these, we leverage the ratio of non-zero 2-sigma counts to eliminate the noise and construct an adjusted Z-Score on top of the divergence to distinguish the masked cases, thereby reducing false negatives.

We conducted a thorough evaluation by benchmarking KRATOS against 9 classic and state-of-the-art solutions on four different datasets, covering 23,628 SSDs and 674,447 HDDs in total (§5). We first deployed KRATOS on the COMX proprietary dataset. To avoid COMX-overfitting concern, we further included the Open fail-slow dataset [34]. To eliminate operational bias, we then asked another company (COMY) to *independently deploy* KRATOS in their production systems (*i.e.*, without our access to their dataset or the ability to tune KRATOS). Finally, to assess if KRATOS detects wide-spectrum fail-slow behaviors, we reproduced 9 faults in testbed and deployed KRATOS on the collected data.

On the Open dataset, KRATOS is as effective as Perseus (the best of the 9), but on the COMX, COMY and fault-injection datasets, KRATOS is superior to Perseus by detecting 32.8-50% more of the total cases. On more challenging fail-slow patterns, KRATOS successfully identifies 90% and 73.8% more cases. KRATOS is the most stable, *i.e.*, it can sustain near-perfect accuracy (>95%) for 37.3% of threshold ranges, while other algorithms can only do so within 0-5% threshold ranges. Furthermore, in 86.7% of threshold ranges, KRATOS remains the highest-performing algorithm. In terms of execution time, KRATOS is 3-128× (43× on average) faster than all the algorithms. Overall, these results show that KRATOS *is the only approach capable of detecting all fail-slow patterns while maintaining stable and high performance across all datasets*.

2 Dataset and Motivation

2.1 Dataset

Our analysis is based on 323,835 disks across 4200 hosts over a year, collectively covering 55 million drive days and 39.7 billion telemetry data points.

2.1.1 Components. A **host** is a system that interacts with the storage infrastructure managed by COMX. Each host contains multiple RAID groups. The majority of hosts (over 90%) have 2 to 14 RAID groups each. A **RAID group** consists of physical disks (or disk partitions) directly attached to a controller, collectively enhancing performance. We focus on disk-based RAID-6 groups with non-rotated parity blocks and we only include data disks.[2] The interquartile range of RAID group width, excluding parity disks, is 9 to 19.

2.1.2 How the Dataset Is Collected. Performance-related telemetry data (e.g., latency, blocks read/write) is collected **every ten seconds** at both disk and RAID levels from production systems at client sites. Data collected at the same time from the same RAID group is called an **epoch**. To minimize interference, data is recorded daily in a rotating 2-hour window (*e.g.*, 2-4 pm Monday, then 4-6 pm Tuesday) to better capture workload variations.

2.1.3 Important Features. In every epoch, 2 key metrics are captured: **(1) per-block latency**, which is the average latency per block within the 10-second epoch; and **(2) block count**, which is the total number of 4 KiB blocks read per epoch. **Throughput** equals block count times block size divided by 10. We focus on read latency since most writes in this deployment are handled by non-volatile memory, which ensures stable performance. Hence, most fail-slow issues were reported in read operations.

[2] Since parity disks do not serve read I/Os, we exclude them from our analysis to avoid misclassification.

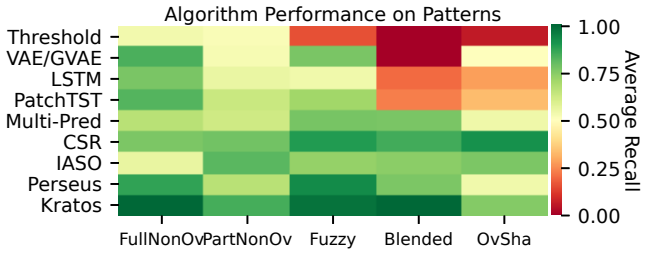


Figure 1: **Algorithm performance on patterns (§3).** *This heatmap shows algorithms' performance on the patterns in Section 3. Green and red cells mean high and low recalls respectively.*

2.2 Study Methodology

To begin an in-depth study of disk fail-slow behaviors in such a large-scale dataset, we first confront the question: *How can we systematically study fail-slow cases in the absence of labeled data?* While performance complaints are tracked, critical details are often missing. Related work like Perseus [34] published 297 fail-slow cases, but only after deploying their algorithm. There is a lack of formalized methodology for systematically understanding and classifying disk fail-slow cases, a concern echoed by industry engineers we spoke to.

To address this question, we employ a two-phase methodology: statistical analysis to identify suspicious patterns indicative of fail-slow disks, followed by validation on a subset of cases confirmed through system logs and engineer assessment (§5.1). Our analysis leverages raw performance metrics such as latency, throughput, and block access patterns, along with their temporal variations. We use the convex hull and Sigma Rules [41, 42] to measure cluster separation in spatial (latency-vs-throughput) trends, and we check the cluster centroid and interquartile range (IQR) [43, 44] for gauging temporal similarity in per-block latency.

We would like to emphasize that, in our experience, studying patterns using these methods proved more valuable than rushing to find a bulletproof solution. Two years ago, we tried various approaches but all failed (*e.g.*, deep learning was ineffective). Our journey led us to conclude that it is important to first understand the different patterns of fail-slow cases, which we will unpack in the next section (§3) before designing a more advanced fail-slow detection pipeline (§4).

3 Fail-Slow and Noisy Patterns

We define a drive-level fail-slow symptom as recurring or persistent latency degradation of one disk relative to comparable peer disks, without requiring the disk to fail outright. In this section, we attempt to answer two important research questions: **(1)** Do fail-slow failures share common traits? If not, how do they differ? **(2)** Can state-of-the-art solutions detect all fail-slow failures? If not, why?

For question (1), we show five fail-slow patterns (§3.1) ordered by detection difficulty, ranging from clear, self-evident anomalies to more subtle, harder-to-detect failures. Additionally, we identified two noise-inducing patterns that make many other methods ineffective (§3.2). For question (2), as we study the patterns, we also port and refactor nine existing detection frameworks (from existing fail-slow detection methods to advanced machine learning techniques) to evaluate their effectiveness on our discovered patterns, as summarized in **Figure 1**. *All the patterns have been identified in the Open dataset (covers both SSDs and HDDs) as well.*

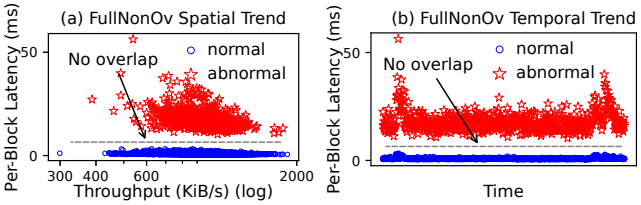


Figure 2: **Fully non-overlap cluster (§3.1.1).** *This figure shows the spatial and temporal trends of the FullNonOv type for a RAID group over a 2-hour window. The y-axis represents per-block latency, while the x-axis shows throughput in (a) and time in (b). The normal (blue) and abnormal (red) clusters are fully separable.*

3.1 Fail-Slow Patterns

This section presents the five major fail-slow patterns, ranked by how likely they are to remain undetected by existing methods. For each pattern, we call the group of data from the abnormal disk an *abnormal cluster*, and the ones from the normal disks a *normal cluster*. Figures for each pattern in this section follow the same style.

3.1.1 Fully Non-Overlap Clusters. We begin with the most commonly known behavior: *fully non-overlapped clusters* (FullNonOv), commonly called outlier clusters, in **Figure 2a**. Most disks (in blue) maintain low latency (in the y-axis) across varying throughput (in the x-axis), while one disk (in red) exhibits unexpectedly high latencies. While Figure 2a focuses on the spatial trend (latency vs. throughput), its fail-slow behavior is also clear in the temporal trend (**Figure 2b**), where the problematic disk (in red) *consistently* shows abnormally high per-block latency in every epoch. These latency spikes may stem from firmware bugs or accumulated device errors causing permanent slowdowns. For example, failing to update a bug-prone firmware version can degrade performance, while irreparable device errors can further deteriorate disk performance [32, 45–47].

These disks can be easily isolated due to their strong spatial and temporal signals. The former is how far a disk deviates from peers per epoch; the latter is how persistently it deviates. Even the off-the-shelf methods can identify them. However, methods prone to false positives (*e.g.*, IASO and CSR) may overlook them when reducing false alarms.

3.1.2 Partially Non-Overlap Clusters. Unlike the previous category, where outliers are consistently obvious, the abnormal cluster here is only *partially* separable. We call these *partially non-overlapped clusters* (PartNonOv). In

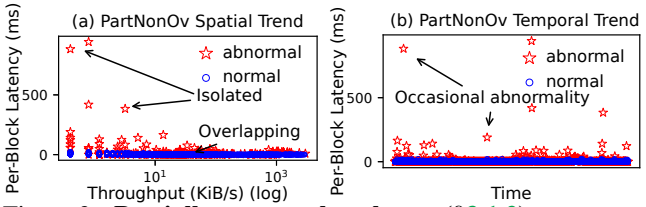


Figure 3: **Partially non-overlap cluster (§3.1.2).** *In (a), the lower part of the abnormal cluster (red) overlaps with the normal cluster (blue), while the upper part is separable. In (b), the temporal trend reveals sporadic spikes in the abnormal disk's latency.*

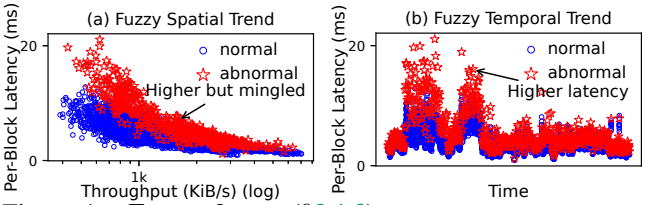


Figure 4: **Fuzzy cluster (§3.1.3).** *The spatial trend in (a) shows overlapping clusters without clear separation. The temporal trend in (b) reveals consistent higher latency from the abnormal disk, with clearer separation when RAID-wide latency is high.*

Figure 3a, the disk showing unexpectedly high latencies (in red) has some fully isolated points, while the rest mingles with the normal cluster (in blue). The temporal trend in **Figure 3b** confirms the disk's mixed behavior. Hardware wear (*e.g.*, bad sectors) can cause this inconsistent slowdown [32]. In Figure 3a, latency spikes occur across varying throughput, suggesting they are unrelated to bursty workloads. If one or a few blocks in a large I/O encounter a media error (*e.g.*, unrecovered read error) requiring multiple retries, latency spikes [48, 49].

This type of disk is harder to detect. While outliers have higher latency (strong spatial signal), temporal signals are weak since abnormalities occur sporadically, especially when issues appear only in specific I/O types (*e.g.*, large I/Os). Due to this inconsistency, the detection effectiveness highly depends on how they preserve temporal information. IASO reliably detects it by accumulating temporal signals.

3.1.3 Fuzzy Clusters. We found it more challenging when the abnormal cluster "mingles" with the normal cluster, as shown in **Figure 4a**. The abnormal cluster (red) has higher latency across throughput ranges but largely overlaps with the normal one (blue), *preventing* clear separation [50]. Thus, it's named *fuzzy* clusters (Fuzzy). Although spatial signal is weak, its temporal signal is strong. In **Figure 4b**, red spikes is noticeable when RAID-wide latency increases. One main cause is model/vendor differences [51]. As Figure 4a and 4b show, lower throughput correlates with higher latency, hinting that performance differences are more apparent during non-throughput-bound periods (*e.g.*, small I/Os).

In this pattern, most data points fall "in the middle," making classification difficult. While the boundary between clusters is unclear, their centroids differ, providing a moderate spatial signal. The abnormal disk is consistently slower during small I/Os, offering some temporal signal as well. However, due to the weak temporal signal, time-series methods like IASO and PatchTST struggle to produce stable results. Regression-based approaches like Perseus can detect spatial trends, but the signals remain weak due to cluster fuzziness. The impact of weak signals is discussed in Section 5.6.1.

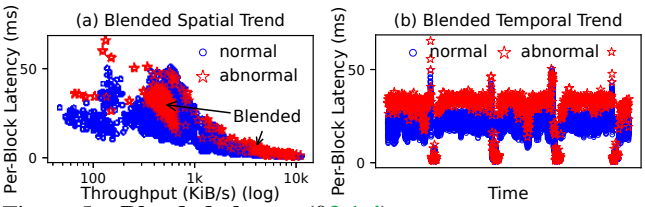


Figure 5: **Blended cluster (§3.1.4).** *The spatial trend in (a) shows the fully intertwined clusters since the periodic spikes of the normal peers in (b) obscure the consistently elevated latency.*

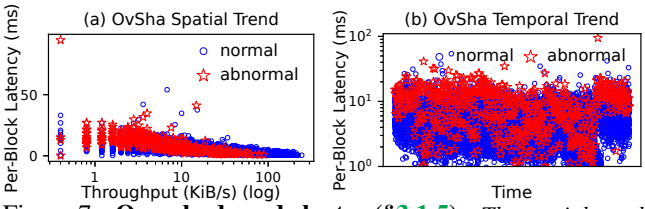


Figure 7: **Overshadowed cluster (§3.1.5).** *The spatial trend in (a) and the temporal trend in (b) show no distinction between the normal and abnormal clusters.*

3.1.4 Blended Clusters. Although similar to the prior type, this one differs as the abnormal cluster is *completely inseparable* from the normal cluster in spatial analysis. Hence, we call it *blended* clusters (Blended). In **Figure 5a**, the trend resembles the previous type, but the normal (blue) and abnormal (red) clusters are fully intertwined. Fortunately, the temporal signal remains strong. As shown in **Figure 5b**, the abnormal disk consistently exhibits higher latency than its peers, despite occasional spikes in normal disks. The most common cause is model differences, though software bugs like scheduling issues can also contribute [34, 51]. Unlike the temporary latency variations in the previous pattern, the temporal separation here is more persistent.

While Perseus can detect the previous pattern, it struggles to identify this case reliably. This is because the centroids of the two clusters in Figure 5a are closer than in Figure 4a, further weakening the spatial signals.

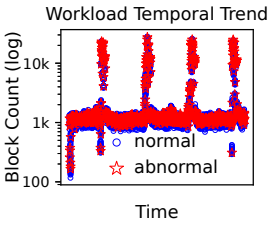


Figure 6: **Blended.**

Further investigation shows that the blending results from periodic bursty workloads (indicated by the block count in **Figure 6**), which create RAID-wide periodic latency spikes, including the blue dots in Figure 5b. Consequently, high-latency blue dots appear in the upper-left of Figure 5a, burying the red dots and obscuring the fail-slow behavior.

3.1.5 Overshadowed Clusters. Finally, the most chal-

lenging pattern that we observed is *overshadowed clusters* (OvSha), where an abnormal disk's behavior is overshadowed by its peers in *both* spatial and temporal analyses. In **Figure 7a** and **7b**, the abnormal disk (red) appears benign. In **Figure 8**, it processes more small I/Os, indicating an imbalanced workload. This is evident from its low total block reads and high average latency. Ideally, all data disks in a RAID group should receive balanced I/Os to ensure even utilization and maximize aggregate bandwidth, making such a persistent imbalance abnormal. This often results from software bugs (*e.g.*, I/O scheduling issues) [34, 52, 53].

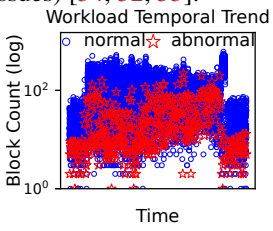


Figure 8: **OvSha.**

Due to the subtleness of this pattern, our experiments (§5) show that many detection methods struggle here, as summarized in Figure 1. This highlights the importance of studying fail-slow patterns at scale. In fact, this pattern emerged during KRATOS's iterative development. While refining KRATOS with key metrics from earlier patterns, these refinements ultimately revealed this new pattern.

3.2 Noise Patterns

While the previous section discusses types prone to false negatives, this section shows how transient anomalies and inconsistent workloads create noise in detection.

3.2.1 Transient. The most common noise is transient failures. When these failures self-resolve without a clear cause, operators often ignore them due to the high debugging cost. Transient noise can also stem from bursty workloads [34]. Based on our datasets, accounting for this pattern reduces false positives by 64% due to its prevalence.

3.2.2 Small Load. At scale, even deployed machines experience periods of low load. Like small sample sizes in statistics, low load reduces the reliability and accuracy of trend estimation. The scattered latency data under these conditions complicates trend analysis for all detection methods, weakening their effectiveness. For instance, Perseus's regression approach fails due to the absence of a clear trend. Identifying this issue could further eliminate 25% of the false positives.

3.3 Takeaways for Fail-Slow Detection

Detecting fail-slow issues involves finding deviation in spatial and temporal trends. We map these patterns onto a Cartesian plane, with spatial signals on the x-axis and temporal signals on the y-axis, categorized as strong, moderate, and weak. In **Figure 9**, FullNonOv appears in the upper right (strong in both), while noise and normal cases are in the lower left. We make three key observations. First, this mapping highlights the comprehensiveness of our study. More importantly, the impact of individual patterns shows that none are negligible, making inclusive detection essential. Second, this visualization clarifies the strengths and weaknesses of existing algorithms. For example, Perseus, which mostly relies on spatial signals, excels at detecting Fuzzy but struggles with Blended. In contrast, IASO accumulates temporal signals but is spatially weak, leading to missed detections in Fuzzy. Third, an ideal detection approach should integrate both signals, forming a "sweet curve" in Figure 9. Depending on the setup and sensitivity, this curve may lean toward spatial or temporal signals to maximize fail-slow coverage while minimizing false positives.

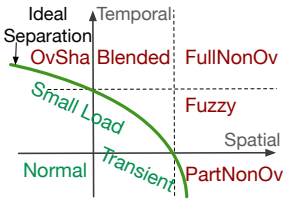


Figure 9: **Fail-slow axes.**

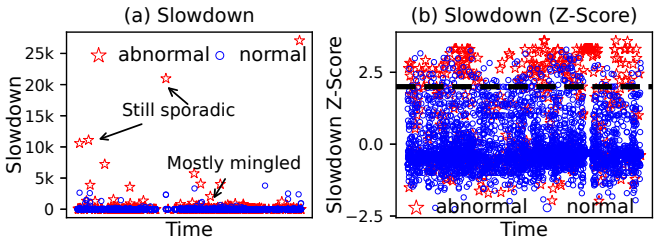


Figure 10: **Slowdown measurement on partially non-overlap clusters (§4.1.1).** *The figure shows the simple slowdown and Z-Score slowdown of the partially non-overlap cluster pattern case discussed in Section 3.1.2.*

4 Design

The discovery of varying fail-slow patterns and the identification of the "(un)comfort zones" in existing methods drive us to develop KRATOS with the following principles: it should be *comprehensive*, covering all fail-slow patterns and deviation severities; *general*, requiring only common telemetry rather than specialized features; *efficient*, keeping both computation and storage overhead low; and *practical*, producing interpretable and actionable results for operators.

At a high level, KRATOS uses statistical methods to turn noisy latency traces into operator-facing evidence. Z-Score asks whether a disk is slower than its peers per epoch; 2-sigma count tells whether that slowdown persists over time; divergence shows if the disk's daily slowdown pattern differs from peer disks; and the non-zero ratio and adjusted Z-Score handle transient noise and masked cases before alerts fire.

4.1 Feature Engineering: 2-Sigma Count

The first challenge we address is identifying a feature that can effectively capture various fail-slow patterns. We aim to avoid inefficiently tracking excessive information or deploying multiple algorithms for *efficiency*.

4.1.1 Slowdown with Z-Score. From Section 3.1, we know that regardless of the patterns, all of them exhibit some degree of deviation. Hence, we start with metrics that are of-

ten used to detect deviations and outliers. One such metric is the widely used slowdown ratio, defined as the latency of an individual disk relative to the RAID-wide median latency [54]. For example, in **Figure 10a**, we directly apply the method on PartNonOv, a case previously shown to have vague temporal signals. *Sporadically*, a disk can look over 1000x slower (elevated red dots) compared to its peers (blue dots). However, slowdown *fails* to provide more separation between normal and abnormal disks (in most of the epochs blue and red dots are still mingled). This is because the median, although widely used, is overly rough and volatile to represent the normal cluster. Hence, we seek other peer-based statistical approaches to highlight the slowness.

After testing various approaches, including interquartile range (IQR) [55], percentile-based methods [56], and the Z-Score (standard score) [40], we found the Z-Score most effective. It accounts for variance in comparisons, making subtle deviations more noticeable when overall variance is low. For instance, in **Figure 10b**, rather than displaying sporadic high-latency points, the Z-Score highlights a consistent pattern where most values from the abnormal disk (red) exceed 2, clearly distinguishing it from normal behavior.

To systematically set the Z-Score threshold, we first constrain search space to integers, a common practice [54] for fast convergence and *practicality*. We then analyze the Z-Score distribution over a month (**Figure 11**). The tail is defined the same as in [54], where the slowest disk at the epoch is Tail 1, the second slowest is Tail 2, and so on. The CDF shows Tail 1 is significantly shifted to higher slowdown regions, while over 95% of Tail 2 and all of Tail 3 remain below a Z-Score of 2. The gap between Tail 1 and 2 is maximized when Z-Score is 2 (gray dotted line), making it the optimal threshold. Therefore, we classify per-epoch outliers as disks with a slowdown Z-Score above 2 (2-sigma rule).

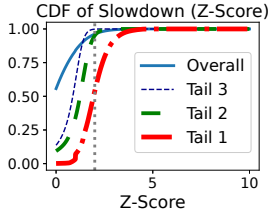


Figure 11: **Slowdown.**

4.1.2 2-Sigma Count. While the 2-sigma rule captures *spatial signals*, using it alone can be noisy. As shown earlier in Figure 10b, many normal disks *occasionally* have high Z-Scores as well. However, unlike raw slowdown metrics, the Z-Score provides a more consistent separation (*e.g.*, the abnormal disk is *consistently* above 2), leading us to the next step: accumulating fail-slow deviations over time. To quantify deviation consistency (*i.e.*, *temporal signal*) with minimal overhead, we decide to track its *cumulative occurrence*. Thus, we define *2-sigma count* as a counter that increments by 1 when the Z-Score exceeds

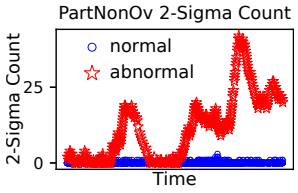


Figure 12: **2-sigma count.**

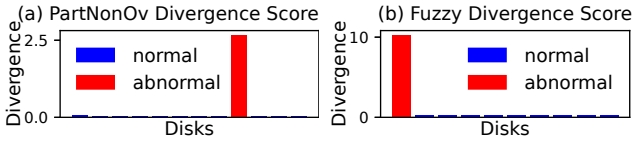


Figure 13: **Divergence score of PartNonOv and Fuzzy cases (§4.2.1).** *This figure shows the disk's 2-sigma count trend and divergence score of the PartNonOv and Fuzzy example cases.*

2 and decrements by 1 when it falls below. This decrement makes KRATOS resistent to SSD performance fluctuations. We thus have our final feature: *2-sigma count*, a cumulative counter that tracks how many consecutive epochs a disk remains 2-sigma away. On the same pattern used earlier, **Figure 12** shows the effectiveness of the 2-sigma count. The y-axis represents the count, showing a *noticeably enhanced* separation between the normal and abnormal disks with persistent spikes across hundreds of epochs.

4.2 Isolating Positives: The Extremes

With the feature identified, we start carving out the cases slowly. Ideally, we seek a clear binary separation between the positive and normal cases, but it is very challenging as we describe more at the end of this section. Thus, we use a tiering approach to describe the *risk level* on a daily basis. The purpose of this section is to provide the risk level definition so that the extreme cases can be easily isolated.

4.2.1 Dimension Reduction. The first step in risk level assessment is dimension reduction. While the 2-sigma count tracks per-epoch cumulative deviation, its sequence is not operator-friendly as it is difficult to set alert conditions. Hence, we compress the sequence into a single daily report, answering the question: "Every day, what is the risk level of this disk?" Our goal is to reduce the sequence to a score per disk while preserving, or preferably even enhancing, the separation between the normal and abnormal disks.

To ensure computational efficiency, we evaluate statistical metrics such as Kullback-Leibler divergence (KL) [57], Jensen-Shannon divergence [58], and Wasserstein metric [59], and choose KL for its separation preservation. We then define the per-disk *divergence score* as: $div(disk) = average_kl(disk, peer_disk)$. This represents the disk's average deviation from its peers. For example, **Figure 13a** shows the divergence score for the partially non-overlap (PartNonOv) case, whose 2-sigma count trend appears earlier in Figure 12. The divergence score "squashes" the sequence to a number while highlighting separation: the score of the abnormal disk (in red) is *significantly higher* than its peers' (in blue). Besides the sporadically abnormal case, the divergence score shows clearer separation for patterns with more persistent deviation. **Figure 13b** depicts the fuzzy clusters (Fuzzy) case (discussed in Section 3.1.3), where the abnormal disk scores 10, compared to 2.5 in the PartNonOv case. This demonstrates that *consistency is rewarded* in this

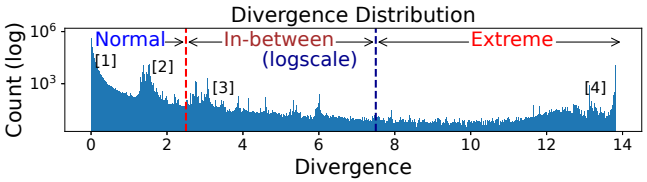


Figure 14: **Divergence score distribution (§4.2.2).** *This figure presents the divergence score distribution from a dataset repeatedly sampled. Instead of a smooth decline from left to right, we observe four peaks overall.*

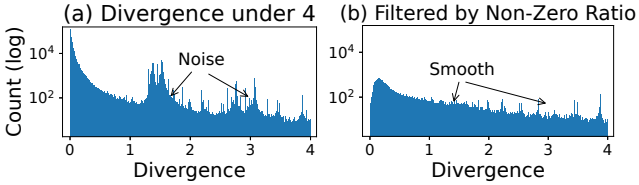


Figure 15: **Effectiveness of non-zero ratio (§4.3.1).** *This figure shows the effectiveness of using the non-zero ratio to filter out noise (i.e., the peaks highlighted in (a)).*

count-based approach, and preserved in the divergence score.

4.2.2 Risk Levels. The next step is categorizing the divergence score into risk levels, as the raw score alone is not actionable. Operators need clear thresholds for decision-making. To establish these thresholds, we first analyze how different 2-sigma count sequences map to divergence scores. **Figure 14** shows the score distribution from a randomly sampled dataset (repeatedly sampled five times, the distribution remains the same). There are four peaks noted. After investigation, we found that the second peak corresponds to normal disks with abnormal peers. Since the score is averaged across peers, normal disks in these cases receive slightly inflated, nonzero values. The third peak represents the *in-between* cases like PartNonOv, where abnormal disks occasionally show high latency. Their scores are higher than normal but remain on the lower end. The fourth peak is related to the *extreme* cases with consistent deviations, such as Fuzzy, where abnormal disks maintain high divergence scores.

With a clear understanding of the peaks, we define the following risk levels: *extreme* (fourth peak), *high* (third peak), *moderate* (between the second and third), *low* (second peak), and *minor* (leftmost peak). The gray line separates extreme from high levels, while the red line distinguishes these from the rest. At this stage, extreme cases are effectively isolated.

4.3 Improve Accuracy: The In-Betweens

While the previous section separates the extreme from the normal, the final challenge is refining the "in-between" categories as indicated in Figure 14. That is, ideally, we reduce the false positives and false negatives, and let the likely positive cases remain intact.

4.3.1 Non-Zero Ratio. The most common source of noise is transient slowness (§3.2.1). Using divergence alone is insufficient, as transient cases can blend with in-between cases

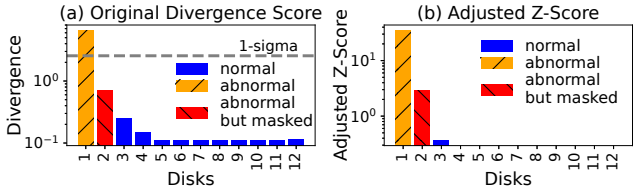


Figure 16: **Effectiveness of adjusted Z-Score (§4.3.2).** *This plot shows the original divergence score and the adjusted Z-Score of a set of peer disks.*

(high-divergence peers inflate normal disks' scores). To prevent false positives, we denoise using the *percentage of non-zero 2-sigma count* (non-zero ratio for short) as the filter. This ensures only disks with sustained anomalous slowness are considered, while normal disks with inflated scores or transient issues are excluded. **Figure 15a** (log-scaled y-axis) shows a zoomed-in divergence score distribution (under 4), where the peaks corresponds to transient and normal cases with inflated scores. Applying a 0.1 non-zero ratio filter, which excludes disks with zero 2-sigma counts over 90% of the time, removes this peaks entirely, as seen in **Figure 15b**.

In addition to the non-zero ratio, we eliminate inactive disks by excluding those with a daily maximum block read close to zero, filtering out small load noise.

4.3.2 Adjusted Z-Score. While transient slowness introduces noise, another observed issue is the presence of multiple abnormal disks. The symptoms of the less severe disk can be *masked* by the more anomalous one, leading to false negatives. For example, in **Figure 16a**, both disks 1 and 2 are labeled abnormal, but disk 1 exhibits a more severe issue while disk 2 shows only subtle deviation. How can disk 2 be isolated? Note that disk 2's divergence score is nearly identical to inflated normal disks (as discussed in Section 4.2) because disk 1 increases per-epoch variance, raising the 2-sigma threshold. We tested sigma-based outlier detection on the divergence score, but disk 1's influence inflates the sigma so much that even the 1-sigma rule (gray line) is ineffective. The divergence score gap between the normal and masked disks remains, the question is how to maximize it. To properly isolate disk 2, we compute the *adjusted Z-Score*: the Z-Score of divergence score excluding disk 1. After removal, disk 2 is clearly revealed, as shown in **Figure 16b**, where its Z-Score more distinctly stands out from normal peers (blue).

Removing transient noises by non-zero ratio, excluding inactive workloads by filtering out disks with near-zero blocks read, and highlighting masked disks via adjusted Z-Score, we can further carve out the in-between cases from the normal ones, and extend the lower bound of the divergence scores to include all the peaks in the risk level calculation.

4.4 Putting It All Together

This section covers how KRATOS works as a whole. KRATOS runs on a daily basis: every day, when new 2-hour data is available, KRATOS will run on a centralized server

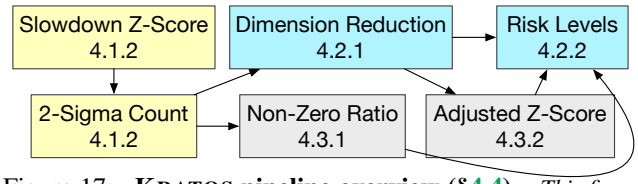


Figure 17: **KRATOS pipeline overview (§4.4).** *This figure shows the key components of* KRATOS *and how they are connected. We adopt the risk level-to-risk score mapping in Perseus (i.e., 100, 25, 10, 5, and 1 for the five risk levels with 100 assigned to high).*

Dataset	HDD/SSD	RAID	#Cases	#Drive	#Method
COMX	HDD	Yes	506	87,326	7
Open [34]	Both	No	297	42,828	9
COMY	Both	No	83	567,921	3

Table 1: **Evaluation dataset (§5.1).** *This table shows the details of the three production datasets used for the evaluation, along with the number of detection methods evaluated on.*

to perform the detection. Detection consists of three steps, as shown in **Figure 17**. First, it converts the raw metrics to 2-sigma count. Then, it quantifies risk via dimension reduction and labels each disk with a risk level. Before sending an alert, it prunes noises. We adopt the risk level-to-score mapping used in Perseus. Alerts for positive cases, defined as disks with cumulative scores above the alerting threshold (detailed in Section 5), will be reported to operators.

5 Evaluation

5.1 Setup

We show the performance of KRATOS and how other algorithms work on three production datasets (listed in **Table 1**) and the fault-injection dataset (details in **Table 2**).

The COMX dataset has 506 positive cases and 87,326 disks (2 weeks for each) in total. The positive cases are cross-verified with additional system footprints (*e.g.*, logs) by engineers, while the negative cases are confirmed to exhibit no noticeable deviations.

To avoid dataset bias, we also evaluate KRATOS on the Open dataset, which has 297 positive cases with a total of 42,828 disks, each having two weeks' data. For each day, the data contains 3-hour write latency and throughput metrics at a 15-second interval. All the labels are created by Open engineers, each at disk granularity.

To eliminate operational bias, KRATOS, Perseus, and Threshold are further evaluated on the COMY dataset, which shares similar data format with Open and COMX. It covers 83 verified cases over 567,921 drive weeks. All the cases are validated by log signals, SMART metrics, and manually confirmed by the engineers at COMY.

To assess the generalizability and sensitivity of KRATOS, we injected 9 faults [34, 60–73] into our testbed [74] following the same protocol in [75] with varying fault severity and collected the latency data.

Fault Type	Fault Root Cause	Pattern
Hardware error	Latent sector errors, ECC retry cycles, Voltage retry loops, Slow page writes [49, 60–65]	PartNonOv, Fuzzy
Firmware error	Firmware bugs, Firmware throttling [54, 66, 67]	FullNonOv
Background maintenance	GC pauses, Wear-leveling issues [67–72]	PartNonOv
Front-end scheduling	Scheduler preemption [34]	Blended, OvSha

Table 2: **Evaluation faults (§5.1).** *This table shows the details of nine faults injected into our testbed for evaluation. The faults are chosen to cover a wide range of root causes and symptoms, and are commonly observed in production systems.*

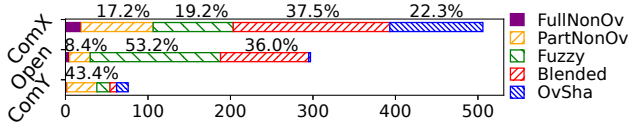


Figure 18: **Dataset pattern ratio (§5.1).** *This figure shows the pattern ratio difference between the three datasets. The length of the block represents the number of cases falling into the pattern.*

Backblaze dataset [76] is not used because it lacks the necessary per-block latency data for our analysis.

In **Figure 18**, we classified the positive cases of the three datasets following the five patterns in Section 3.1. The y-axis is the dataset and the x-axis is the case count. The pattern ratio differs across all three datasets.

Methods compared. We compare KRATOS with 9 methods: Threshold [39], VAE [77], GVAE [9], LSTM [35], PatchTST [36], Multi-Pred [38], CSR [37], IASO [33], and Perseus [34]. For Threshold, a disk is considered anomalous if its average latency is over a threshold for the past 5 minutes. For VAE, GVAE, LSTM and PatchTST, we train on normal disks and alert based on MSE of the predicted latency following the guide in [9, 78]. For Multi-Pred and CSR, we adapt their approach and ensure the core component (Random forest and XGBoost) is used [79, 80]. For IASO and Perseus, we follow the implementation in each paper.

5.2 On the COMX Dataset

This section shows the performance of KRATOS and other algorithms on the COMX dataset. We discuss the true positives, false negatives, and false positives individually. An ideal algorithm should have high true positives, low false negatives, and low false positives. For each algorithm, we use the parameters that yield the highest Matthews Correlation Coefficient (MCC [81]).

5.2.1 True Positives. We first show the true positives in **Figure 19a**. Each bar shows the true positive cases (in the y-axis, from 0 to 506) detected by various fail-slow detection algorithms (in the x-axis). For example, the leftmost bar shows that Threshold has 39 detected true positive cases whereas Perseus (second from the right) has 340 cases identi-

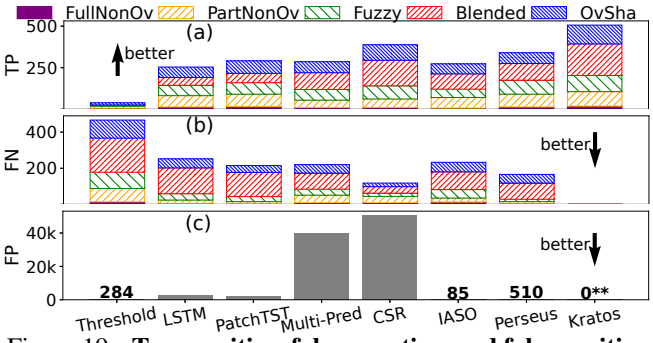


Figure 19: **True positive, false negative, and false positive case count (§5.2).** *This figure shows the true positive (TP), false negative (FN), and false positive (FP) case counts on the* ComX *dataset. For TP and FN, we color the bar based on patterns.*

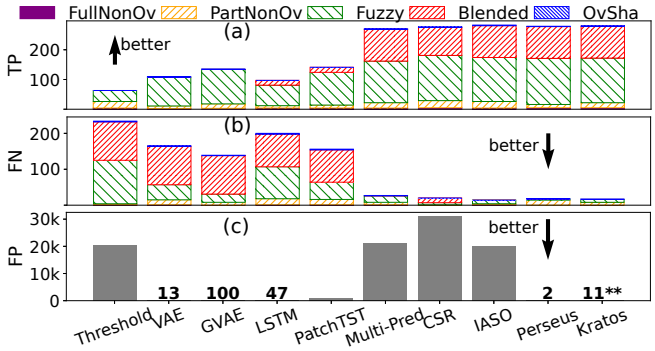


Figure 20: **True positive, false negative, and false positive case count (§5.3).** *This figure shows the true positive (TP), false negative (FN), and false positive (FP) case counts from each algorithm on the Open dataset.*

fied. The color segment indicates the proportion of different patterns. Perseus detects 77 PartNonOv (yellow), 84 Fuzzy (green), 100 Blended (red), and 65 OvSha (blue) cases.

[Key result #1] *While* Kratos *successfully works on the Open dataset, Perseus does not show similar performance on the* ComX *dataset, as it only detects 67.2% of the cases.* Among the rest, Threshold detects the fewest number of cases, showing that it's overly conservative. CSR has the highest number of cases detected next to Kratos. Kratos outperforms the rest, particularly in Blended and OvSha, the two types that are challenging to detect. For Blended, Kratos has a 23.4% and 90% higher detection rate than CSR and Perseus, respectively; for OvSha, Kratos is 21.5% and 73.8% more accurate than CSR and Perseus.

5.2.2 False Negatives. We then show the false negatives in **Figure 19b**. Each bar shows the number of missed cases and is colored with the associated pattern. Taking Perseus as an example, it misses 5 out of 19 FullNonOv, 10 out of 87 PartNonOv, 13 out of 97 Fuzzy, and roughly half of the Blended (90/190) and OvSha (48/113) cases.

[Key result #2] *Except* Kratos*, all other techniques fail to detect the spatially challenging patterns, as the average miss rate of Blended and OvSha for the other approaches (excluding Threshold) is 51.8% and 37.8%, respectively.* The miss rate of PartNonOv is particularly high for Multi-Pred and CSR, showing that simply borrowing from a similar domain is impractical. **[Key result #3]** *LSTM and PatchTST miss, on average, 72.9% of Blended, indicating that the advanced time-series approaches can hardly differentiate spatially subtle fail-slow symptoms from normal variation, making them unsuitable for fail-slow detection.*

5.2.3 False Positives. For the false positives, we show the total count per algorithm in gray bars in **Figure 19c**, with numbers noted above the nearly invisible bars. For example, Threshold has 284 false positives while Multi-Pred and CSR have over 40,000 false positives.

Kratos can reliably eliminate noise better than all other fail-slow detection and adapted algorithms. Among all the possible alerting thresholds that Kratos can choose from, we found that there is a 1.7% "sweet spot" that can perfectly balance the false positives and false negatives(**), yielding the highest MCC. We attribute it to the homogeneity of RAID. Unfortunately, such a "sweet spot" cannot be found for any other algorithms as they are either overly sensitive or too lenient. In particular, **[Key result #4]** *the overly high false positives from Multi-Pred and CSR confirms our prior observation that adapting algorithms from a similar field is not viable for fail-slow detection, as the precision is no better than random.* Another interesting finding is that the false positives can vary significantly across the two datasets. For example, Threshold and IASO have high false positives (over 20,000) in the Open dataset but the number is low (under 300) in the ComX dataset, which shows the difficulty of developing a general approach that can show reliable performance across different environments.

5.3 On the Open Dataset

This section shows the performance of Kratos and all other algorithms on the Open dataset to avoid dataset bias. For each algorithm, we continue to use the parameters that yield the highest MCC on the dataset.

5.3.1 True Positives. The true positives are in **Figure 20a**. The y-axis ranges from 0 to 297 and the algorithms follow the same order as Section 5.2. For example, the leftmost bar shows that Threshold has 63 true positiveas cases detected and the two rightmost bars show the total cases identified by Perseus and Kratos (279 and 280 respectively). The bar is also colored based on the pattern. For example, 58.7% of the cases detected by Threshold are Fuzzy (green).

[Key result #5] Kratos *is as effective as Perseus on the Open dataset, detecting even one more true positive case.* Among the nine evaluated methods, Multi-Pred, CSR, IASO, Perseus, and Kratos identify a similar number of cases, with IASO achieving the highest detection count.

5.3.2 False Negatives. We next analyze the false negatives in **Figure 20b**. The y-axis, x-axis, and color segment re-

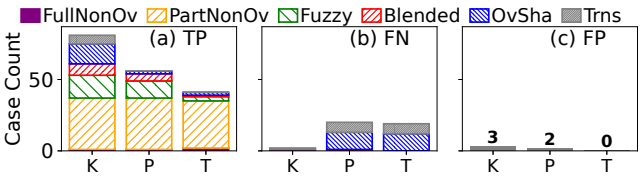


Figure 21: **COMY case count (§5.4).** *This figure shows the evaluation result from* KRATOS *(K), Perseus (P) and Threshold (T) on the* COMY *dataset.*

main the same as in Figure 20a. Each bar shows the number of missed cases and the associated patterns. For example, Perseus and KRATOS miss 18 and 17 cases respectively, with Perseus missing mostly PartNonOv (yellow) and KRATOS missing half PartNonOv and half Fuzzy (green).

[Key result #6] *Perseus performs poorly on temporally challenging patterns, as it misses 56% of PartNonOv.* All the methods miss some PartNonOv cases, showing the challenge in detecting cases with unstable temporal signals. Another key observation is that Perseus exhibits the lowest miss rate for Fuzzy cases, which comprises 53.2% of the total dataset (shown in Section 5.1), suggesting that its algorithm may be primarily optimized for this failure pattern. Threshold, VAE, GVAE, LSTM, and PatchTST miss almost all Blended cases, which are subtle to detect.

5.3.3 False Positives. We now examine false positives as depicted in **Figure 20c**. Each bar shows the number of false positives by each algorithm and we list numbers on the top of the small bars that are nearly invisible. For example, LSTM has 47 false positives under its ideal parameters and Threshold has over 20,000 false positives.

Perseus has the lowest false positives (2) and KRATOS classifies 11 of the "normal" disks as fail-slow disks. However, *engineers may mislabel the case(**).* After further investigation, 8 of them look highly suspicious as they display fail-slow symptoms (*e.g.*, visually slower than peers for over 2 hours). We have shared the cases with Alibaba's SRE and received confirmation that they are indeed suspicious. The cases are currently under investigation. **[Key result #7]** *Threshold, Multi-Pred, CSR, and IASO yield at least 47.1% false positives, making them unusable in production.*

5.4 On the COMY Dataset

This section shows the performance of KRATOS, Perseus, and Threshold on the COMY dataset. To eliminate operational bias, *the* COMY *evaluation was conducted entirely by* COMY *engineers, without our access to their dataset or the ability to tune* KRATOS so no tuning was performed.

5.4.1 True Positives. As shown in **Figure 21a**, KRATOS consistently detects more cases than both Perseus and Threshold across patterns. **[Key result #8]** *Specifically,* KRATOS *catches 50% more true positive cases than Perseus, and 97.6% more than Threshold.* This result highlights the strength of KRATOS in maintaining high sensitivity as it is

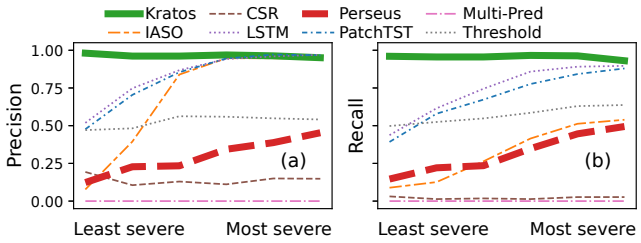


Figure 22: **Precision and recall trend on the injected faults (§5.5.1).** *This figure shows the precision (a) and recall (b) of each method on the injected faults. The x-axis is ordered by increasing fault severity, and the y-axis is the macro precision and recall across all five traces.*

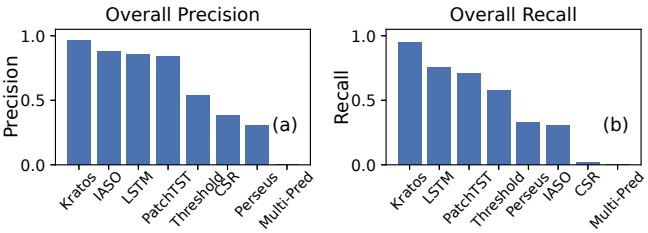


Figure 23: **Overall precision and recall on the injected faults (§5.5.2).** *This figure shows the overall precision (a) and recall (b) for each method on the injected faults.*

applied to the COMY dataset without tuning, showing its resilience. On the contrary, Perseus struggles to adapt to the new dataset when fine-tuning was not applied.

5.4.2 False Positives and Negatives. As shown in **Figure 21b** and **c**, all methods yield near-zero false positives. **[Key result #9]** *This indicates that* KRATOS *can achieve a higher detection rate without introducing noise, even without tuning*, making it preferable for real-world deployment.

KRATOS has higher coverage in Blended and OvSha. There are several borderline cases that are statistically considered transient but, later on, are confirmed to be faulty, and only KRATOS is able to detect such cases despite the subtleness of the symptom. The cases missed by Perseus and Threshold are mostly due to challenges in tuning the thresholds that achieve high accuracy, showing the lack of flexibility in both approaches.

5.5 On the Fault Injection Dataset

All three datasets are symptom-based, containing only traces of observed symptoms without root-cause information. We further evaluate the methods in a controlled setting by reproducing nine representative faults in our testbed [74]. These faults span a wide range of scenarios, from hardware errors to front-end scheduling issues, allowing us to test whether KRATOS covers a broad spectrum of cases. Following the same protocol as [75], we run five traces under different workloads [82, 83]. This setup complements the production-study results by enabling repeatable evaluation under progressively harsher fault configurations. Configuration details

are provided in the supplemental material.[3]

5.5.1 Precision and Recall Trend. From the macro precision and recall results in **Figure 22**, **[Key result #10]** KRATOS *is the most stable approach across the injected configurations*. As the injected faults become more severe from left to right, KRATOS consistently sustains near-perfect precision and recall, showing that its behavior is robust to changes in fault intensity. IASO is also reasonably strong, especially as severity increases, although its performance is still less consistent than KRATOS. We also observe that most of the residual inaccuracy of KRATOS is concentrated in one fault (scheduler preemption) rather than being broadly spread across all injected cases, which suggests that its remaining weakness is narrow and specific instead of systemic.

5.5.2 Overall Precision and Recall. The overall precision and recall analysis in **Figure 23** further highlights the trade-off between coverage and alert quality. Both KRATOS and IASO achieve high precision (as shown in Figure 23a), indicating that they are effective at surfacing genuinely problematic cases. However, IASO has substantially lower recall (Figure 23b), suggesting that it misses many fault cases despite the high quality of its alerts. In contrast, KRATOS maintains strong precision while also achieving much higher recall, resulting in better overall fault coverage and a more effective balance between alert quality and completeness.

5.6 Other Evaluation

In this section, we show how the performance is affected by the threshold sensitivity, from the most sensitive threshold (alerting as many as possible) to the most lenient threshold (alerting only if the confidence is high), on both COMX and Open datasets. We then show the computational overhead of KRATOS and the detailed parameter tuning for KRATOS.

5.6.1 Threshold Sensitivity. **[Key result #11]** *Among all the algorithms,* KRATOS *is the only one that could always maintain high true positives while having minimal false positives.* In **Figure 24a** and **24b**, only KRATOS (in green) can maintain an MCC > 0.95 and > 0.8 for 37.3% and 22.7% of the threshold range on the COMX and the Open dataset, respectively. For the others, either the MCC is constantly low (*e.g.*, Threshold (in gray), with a max 0.22 MCC) or it can only maintain a high MCC for a small range (*e.g.*, Perseus (in red) has > 0.95 MCC for only 5% of the threshold range on the Open dataset).

[Key result #12] KRATOS *is the only one that can reliably capture all five patterns while all the others have their own "comfort zone."* The reason KRATOS is stable is because it performs similarly well across *all patterns*. In **Figure 25**, we show the per-pattern recall trend on the COMX dataset. The x-axis is the threshold sensitivity, and the y-axis is recall. The recall of Perseus (in red) decreases drastically

[3]The fault-injection details can be found in the supplemental material.

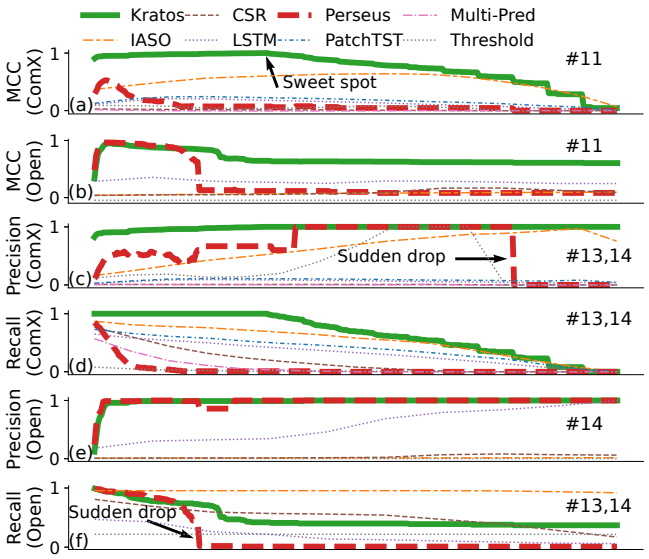


Figure 24: **Metric trends (§5.6.1).** *This figure shows the metric trends (MCC, precision, and recall) for all algorithms on both datasets when the threshold changes. From left to right, the threshold goes from the most sensitive to the most lenient. The related key result number is noted in each figure.*

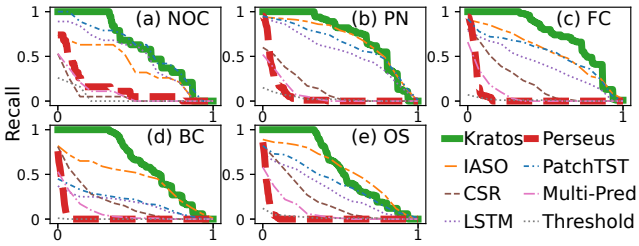


Figure 25: **Recall trend per pattern (§5.6.1).** *This figure shows the per-pattern recall trend of each algorithm when the threshold changes from the most sensitive (leftmost) to the most lenient (rightmost) on the* COMX *dataset.*

(especially for OvSha) as the threshold is tightened for most patterns (except FullNonOv, which is the most obvious fail-slow type). However, for KRATOS (in green), each recall has a large plateau at the top, making it possible to find the common threshold that covers all the patterns.

[Key result #13] *Perseus is overly sensitive to threshold changes, even for its own dataset.* From **Figure 24c**, the precision of Perseus increases as the threshold changes but suddenly drops to 0 near the end. It shows that it lacks the capability to effectively reduce false positives while maintaining a high true positive rate via its weak output signals. *In* **Figure 24f**, *the recall of Perseus goes from 0.4 to 0.02 within a 0.3% threshold change*, and a similar trend can be observed in **Figure 24d** as well. Such instability makes the algorithm hard to be widely applied for fail-slow detection.

[Key result #14] KRATOS *is reliable even with suboptimal thresholds.* In **Figure 24c** to **Figure 24f**, the precision and recall of KRATOS remain high until near the mid-point (49% and 21.7% of the threshold range for > 0.7 recall and

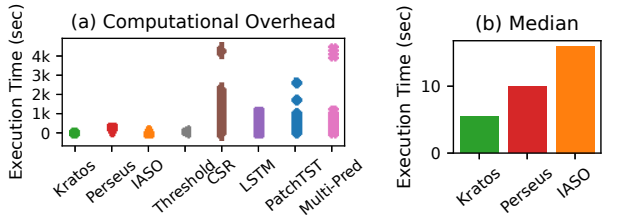


Figure 26: **Computational overhead (§5.6.2).** *This figure shows computational overhead. (a) shows the execution time of all algorithms, with the line showing the standard deviation. (b) shows the median execution time of* KRATOS*, Perseus, and IASO.*

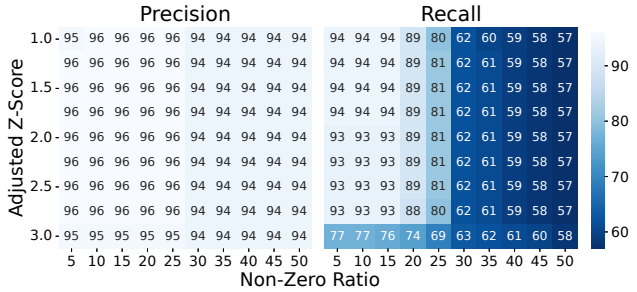


Figure 27: **Sensitivity evaluation (§5.6.3).** *This figure shows the precision and recall of* KRATOS *on the Open dataset under different non-zero ratios (x-axis) and adjusted Z-Scores.*

precision on the COMX and the Open dataset, respectively), which brings two benefits. First, the operators have a wider range of thresholds to choose from. Any threshold falling within that range will yield good accuracy. Second, if the monitored environment changes, re-evaluation of the threshold would not be urgent since KRATOS can still achieve reasonable accuracy with suboptimal thresholds.

5.6.2 Computational Overhead. We measure the execution time for all the algorithms in Figure 26. The y-axis is the time required to process a RAID group and the x-axis is the algorithm. The line that crosses through the dot is the standard deviation. From **Figure 26a**, we can see that there are two groups, fast (KRATOS, Perseus, IASO, and Threshold) and slow (CSR, LSTM, PatchTST, and Multi-Pred). The average execution time of the slow group is 19× slower than the fast group. Among the fast group, we show the median execution time in **Figure 26b**, KRATOS (green) has the lowest overhead. *The low overhead is a result of using solely statistical approaches.*

5.6.3 Parameter Tuning. To find the proper threshold for non-zero ratio and adjusted Z-Score (§4.3), we conduct a sensitivity evaluation on the Open dataset. In **Figure 27**, we show the precision and recall of KRATOS when fine-tuning. The y-axis is the adjusted Z-Score (from 1 to 3), and the x-axis is the non-zero ratio (from 5 to 50). For example, the leftmost bottom cell means when filtering out disks with adjusted Z-Score lower than 3 and non-zero ratio lower than 5%, the precision is 0.95. Combining precision and recall, we choose 1.5 and 10 as the thresholds for adjusted Z-Score and non-zero ratio in our experiments. Precision and recall

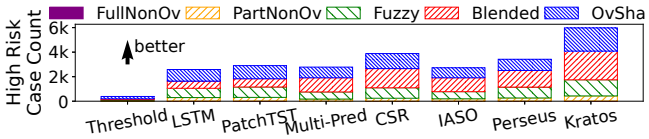


Figure 28: **Detected high-risk case count (§6).** *This figure shows the high-risk cases detected by each algorithm on the* COMX *full production data. The bars are colored by pattern category.*

remain high for a wide range of parameter choices, further confirming the stability of KRATOS.

6 Use Case and Discussion

In addition to detecting ongoing fail-slow symptoms, KRATOS can also be used to prioritize historical traces for operator investigation. KRATOS does not require logs as input and does not perform root-cause diagnosis by itself; instead, it flags drives whose latency behavior is abnormal relative to peers, after which operators can correlate these candidates with logs, SMART data, and other system signals. Across the full COMX dataset (*i.e.*, 3,939,960 disks), KRATOS identifies around 6K high-risk cases with high confidence (Figure 28) using less than one day on one server. This reduces the search space for manual diagnosis and enables system administrators to correlate traces collected months, or even years earlier, with other evidence to investigate potential root causes behind cascading failure events.

Another opportunity for KRATOS lies in its applicability to heterogeneous environments. Due to the quasi fail-slow symptoms, existing approaches (*e.g.*, Perseus) require homogeneous setups, such as identical SSD models under balanced workloads, to reduce noise. In contrast, KRATOS captures fail-slow behavior from both spatial and temporal perspectives without relying on tightly matched workloads or hardware configurations. This makes KRATOS applicable beyond large-scale cloud data centers, including small clusters with mixed workloads and heterogeneous SSDs.

7 Conclusion

In this paper, we present a large-scale study of fail-slow failures, revealing their diverse patterns and the limitations of existing methods. Based on the observations, we further develop KRATOS, a detection method that can reliably capture fail-slow cases under varying patterns, even the subtle ones.

8 Acknowledgments

We thank Andi Quinn, our shepherd, and the anonymous reviewers for their tremendous feedback and comments. We also would like to thank Jiahong Liu for the helpful discussions on this paper. This material was supported by funding from NSF (grant Nos. CCF-2119184 and CNS-2402327).

References

[1] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F. Wenisch. The Mystery Machine: End-to-end Performance Analysis of Large-scale Internet Services. In *Proceedings of the 11th Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[2] Jaewon Lee, Changkyu Kim, Kun Lin, Liqun Cheng, Rama Govindaraju, and Jangwoo Kim. WSMeter: A Performance Evaluation Methodology for Google's Production Warehouse-Scale Computers. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[3] Korakit Seemakhupt, Brent E. Stephens, Samira Khan, Sihang Liu, Hassan M. G. Wassel, Soheil Hassas Yeganeh, Alex C. Snoeren, Arvind Krishnamurthy, David E. Culler, and Henry M. Levy. A cloud-scale characterization of remote procedure calls. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

[4] Phitchaya Mangpo Phothilimthana, Saurabh Kadekodi, Soroush Ghodrati, Selene Moon, and Martin Maas. Thesios: Synthesizing Accurate Counterfactual I/O Traces from I/O Samples. In *Proceedings of the 29th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[5] Mihailo Isakov, Eliakin del Rosario, Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert B. Ross, and Michel A. Kinsy. HPC I/O Throughput Bottleneck Analysis with Explainable Local Models. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[6] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, Weiguo Liu, and Wei Xue. End-to-end I/O Monitoring on a Leading Supercomputer. In *Proceedings of the 16th Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[7] Jovan Stojkovic, Prateek Misra, Íñigo Goiri, Scott Whitlock, Eiman Choukse, Mayukh Das, Chaitanya Bansal, Jung Ho Lee, Zizheng Sun, Haoran Qiu, Roger Zimmermann, Sandeep Samal, Bhavani Warrier, Anshuman Raniwala, and Ricardo Bianchini. Smartoclock: Workload- and risk-aware overclocking in the cloud. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2024.

[8] Reza Zamani and Ahmad Afsahi. A study of hardware performance monitoring counter selection in power modeling of computing systems. In *Proceedings of the 2012 International Green Computing Conference (IGCC)*, pages 1–6, 2012.

[9] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[10] Dong Young Yoon, Yang Wang, Miao Yu, William Wang, Ayichew Hailu, Rohan Bopardikar, Bin Zhang, Jialiang Qu, David Meisner, Santosh Sonawane, Yunqi Zhang, Rodrigo Paim, Mack Ward, Ivor Huang, Matt McNally, Daniel Hodges, Zoltan Farkas, Caner Gocmen, Elvis Huang, and Chunqiang Tang. FBDetect: Catching tiny performance regressions at hyperscale through in-production monitoring. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*, pages 522–540, 2024.

[11] Jonathan Kaldor, Jonathan Mace, Michal Bejda, Edison Gao, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Venkataraman, Kaushik Veeraraghavan, and Yee Jiun Song. Canopy: An End-to-End Performance Tracing And Analysis System. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[12] Fred Lin, Matt Beadon, Harish Dattatraya Dixit, Gautham Vunnam, Amol Desai, and Sriram Sankar. Hardware Remediation At Scale. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2018.

[13] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *Proceedings of the 13th Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[14] Chengqiang Huang, Geyong Min, Yulei Wu, Yiming Ying, Ke Pei, and Zuochang Xiang. Time Series Anomaly Detection for Trustworthy Services in Cloud Computing Systems. In *IEEE Transactions on Big Data*, 2017.

[15] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, Youjiang Wu, Ken

Hsieh, Kaixin Sui, Xin Meng, Yaohai Xu, Wenchi Zhang, Furao Shen, and Dongmei Zhang. Cross-dataset Time Series Anomaly Detection for Cloud Systems. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, 2019.

[16] Fred Lin, Antonio Davoli, Imran Akbar, Sukumar Kalmanje, Leandro Silva, John Stamford, Yanai Golany, Jim Piazza, and Sriram Sankar. Predicting Remediations for Hardware Failures in Large-Scale Datacenters. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2020.

[17] Areeg Samir and Claus Pahl. Anomaly Detection and Analysis for Clustered Cloud Computing Reliability. In *CLOUD COMPUTING*, 2019.

[18] Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, Qingwei Lin, Gil Lapid Shafriri, and Murali Chintalapati. Predictive and Adaptive Failure Mitigation to Avert Production Cloud VM Interruptions. In *Proceedings of the 14th Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[19] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: Practical & Scalable ML-Driven Performance Debugging in Microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 629–644, 2021.

[20] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[21] Mike Chow, Yang Wang, William Wang, Ayichew Hailu, Rohan Bopardikar, Bin Zhang, Jialiang Qu, David Meisner, Santosh Sonawane, Yunqi Zhang, Rodrigo Paim, Mack Ward, Ivor Huang, Matt McNally, Zoltan Farkas Daniel Hodges, Caner Gocmen, Elvis Huang, and Chunqiang Tang. ServiceLab: Preventing Tiny Performance Regressions at Hyperscale through Pre-Production Testing. In *Proceedings of the 18th Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

[22] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC)*, 2014.

[23] Peng Huang, Chuanxiong Guo, Lindong Znhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randonph Yao. Gray Failure: The Achilles' Heel of Cloud Scale Systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS XVI)*, 2017.

[24] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*, 2016.

[25] Thanh Do, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, and Haryadi S. Gunawi. Limplock: Understanding the Impact of Limpware on Scale-Out Cloud Systems. In *Proceedings of the 4th ACM Symposium on Cloud Computing (SoCC)*, 2013.

[26] Aishwarya Ganesan, Ramnatthan Alagappan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to Single Errors and Corruptions. In *Proceedings of the 15th USENIX Symposium on File and Storage Technologies (FAST)*, 2017.

[27] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2016.

[28] Michael P. Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Black-Box Problem Diagnosis in Parallel File Systems. In *Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST)*, 2010.

[29] Andy A. Hwang, Ioan A. Stefanovici, and Bianca Schroeder. Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 111–122, 2012.

[30] Xiaodong Li, Matthew C. Huang, Kai Shen, and Ling Chu. An empirical study of memory hardware errors in

a server farm. In *Proceedings of the 3rd Workshop on Hot Topics in System Dependability (HotDep)*, 2007.

[31] Jacob Alter, Ji Xue, Alma Dimnaku, and Evgenia Smirni. SSD failures in the field: symptoms, causes, and prediction models. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2019.

[32] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST)*, 2018.

[33] Biswaranjan Panda, Deepthi Srinivasan, Huan Ke, Karan Gupta, Vinayak Khot, and Haryadi S. Gunawi. IASO: A Fail-Slow Detection and Mitigation Framework for Distributed Storage Services. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, 2019.

[34] Ruiming Lu, Erci Xu, Yiming Zhang, Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Jiwu Shu, Minglu Li, and Jiesheng Wu. Perseus: A Fail-Slow detection framework for cloud storage systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 49–64, Santa Clara, CA, February 2023. USENIX Association.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[36] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

[37] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, Murali Chintalapati, and Dongmei Zhang. Improving Service Availability of Cloud Systems by Predicting Disk Error. In *Proceedings of the 2018 USENIX Annual Technical Conference (ATC)*, 2018.

[38] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive Error Prediction to Improve Storage System Reliability. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC)*, 2017.

[39] Using Amazon CloudWatch alarms , 2013.

[40] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society A*, 185:71–110, 1893.

[41] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[42] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury, 2nd edition, 2002.

[43] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[44] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[45] 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 *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST)*, 2007.

[46] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. In *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST)*, 2008.

[47] Juan Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '15)*, pages 177–190. ACM, 2015.

[48] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *Proceedings of the 14th USENIX Symposium on File and Storage Technologies (FAST)*, 2016.

[49] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *Proceedings of the 2007 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[50] M.-S. Yang. A survey of fuzzy clustering. *Mathematical and Computer Modelling*, 18(11):1–16, 1993.

[51] Neda Tavakoli, Dong Dai, and Yong Chen. Client-side straggler-aware i/o scheduler for object-based parallel file systems. *Parallel Computing*, 82:3–18, 2019.

[52] Zhibing Sha, Jun Li, Zhigang Cai, Min Huang, Jianwei Liao, and François Trahay. Degraded mode-benefited i/o scheduling to ensure i/o responsiveness in raid-enabled ssds. *ACM Transactions on Design Automation of Electronic Systems*, 27(6):1–24, 2022.

[53] Youngjae Kim and Raghul Gunasekaran. Understanding i/o workload characteristics of a peta-scale storage system. *The Journal of Supercomputing*, 71(3):761–780, 2015.

[54] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *Proceedings of the 14th USENIX Symposium on File and Storage Technologies (FAST)*, 2016.

[55] H. P. Vinutha, B. Poornima, and B. M. Sagar. Detection of outliers using interquartile range technique from intrusion dataset. In *Information and Decision Sciences*, pages 511–518. Springer, 2018.

[56] Olumuyiwa Ibidunmoye, Ali-Reza Rezaie, and Erik Elmroth. Adaptive anomaly detection in performance metric streams. *IEEE Transactions on Network and Service Management*, 15(1):217–231, 2018.

[57] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[58] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[59] Cédric Villani. *Optimal Transport: Old and New*. Springer, 2009.

[60] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. Understanding Latent Sector Errors and How to Protect Against Them. In *Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST)*, 2010.

[61] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. In *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST)*, 2008.

[62] Laura M. Grupp, John D. Davis, and Steven Swanson. The Bleak Future of NAND Flash Memory. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies (FAST)*, 2012.

[63] Laura M. Grupp, John D. Davis, and Steven Swanson. The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs. In *Proceedings of the 2013 USENIX Annual Technical Conference (ATC)*, 2013.

[64] Qiao Li, Min Ye, Yufei Cui, Liang Shi, Xiaoqiang Li, and Chun Jason Xue. Sentinel cells enabled fast read for NAND flash. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, Renton, WA, July 2019. USENIX Association.

[65] Bryan S. Kim, Jongmoo Choi, and Sang Lyul Min. Design tradeoffs for SSD reliability. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 281–294, Boston, MA, February 2019. USENIX Association.

[66] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. A study of SSD reliability in large scale enterprise storage deployments. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 137–149, Santa Clara, CA, February 2020. USENIX Association.

[67] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. *ACM Transactions on Storage (TOS)*, 14(3), 2018.

[68] Jaeho Kim, Donghee Lee, and Sam H. Noh. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of the 13th USENIX Symposium on File and Storage Technologies (FAST)*, 2015.

[69] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, 2019.

[70] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. *ACM Transactions on Storage (TOS)*, 13(3), 2017.

[71] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. Towards an unwritten contract of intel optane SSD. In *11th USENIX Workshop on Hot Topics*

in Storage and File Systems (HotStorage 19), Renton, WA, July 2019. USENIX Association.

[72] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. Operational characteristics of SSDs in enterprise storage systems: A Large-Scale field study. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 165–180, Santa Clara, CA, February 2022. USENIX Association.

[73] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *Proceedings of the 2007 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[74] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[75] Daniar H. Kurniawan, Rani Ayu Putri, Peiran Qin, Kahfi S. Zulkifli, Ray A. O. Sinurat, Janki Bhimani, Sandeep Madireddy, Achmad Imam Kistijantoro, and Haryadi S. Gunawi. Heimdall: Optimizing storage i/o admission with extensive machine learning pipeline. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, pages 1109–1125. ACM, 2025.

[76] Backblaze Drive Stats: Hard Drive Reliability Test Data , 2026.

[77] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[78] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 89–94, 2015.

[79] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[80] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 785–794. ACM, 2016.

[81] Brian Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405:442–451, 1975.

[82] Ricardo Koller and Raju Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Transactions on Storage*, 6(3):13:1–13:26, 2010.

[83] Alibaba Group. Alibaba block traces. https://github.com/alibaba/block-traces, 2020. Block I/O trace dataset from Alibaba Cloud elastic block storage, sampled from 1000 virtual disks in January 2020.