PERSEUS: A Fail-Slow Detection Framework for Cloud Storage Systems

Ruiming Lu, *Shanghai Jiao Tong University;* Erci Xu, *Alibaba Inc. and Shanghai Jiao Tong University;* Yiming Zhang, *Xiamen University;* Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, and Zongpeng Zhu, *Alibaba Inc.;* Guangtao Xue, *Shanghai Jiao Tong University;* Jiwu Shu, *Xiamen University;* Minglu Li, *Shanghai Jiao Tong University and Zhejiang Normal University;* Jiesheng Wu, *Alibaba Inc.*

https://www.usenix.org/conference/fast23/presentation/lu

This paper is included in the Proceedings of the 21st USENIX Conference on File and Storage Technologies.

February 21–23, 2023 • Santa Clara, CA, USA

978-1-939133-32-8

Open access to the Proceedings of the 21st USENIX Conference on File and Storage Technologies is sponsored by

PERSEUS: A Fail-Slow Detection Framework for Cloud Storage Systems

Ruiming Lu[1]*, Erci Xu[3,1]*, Yiming Zhang[2]†, Fengyi Zhu[3], Zhaosheng Zhu[3],
Mengtian Wang[3], Zongpeng Zhu[3], Guangtao Xue[1]†, Jiwu Shu[2], Minglu Li[1,4], and Jiesheng Wu[3]

[1]Shanghai Jiao Tong University, [2]Xiamen University,
[3]Alibaba Inc., and [4]Zhejiang Normal University

Abstract

The newly-emerging "fail-slow" failures plague both software and hardware where the victim components are still functioning yet with degraded performance. To address this problem, this paper presents PERSEUS, a practical fail-slow detection framework for storage devices. PERSEUS leverages a light regression-based model to fast pinpoint and analyze fail-slow failures at the granularity of drives. Within a 10-month close monitoring on 248K drives, PERSEUS managed to find 304 fail-slow cases. Isolating them can reduce the (node-level) 99.99th tail latency by 48%. We assemble a large-scale fail-slow dataset (including 41K normal drives and 315 verified fail-slow drives) from our production traces, based on which we provide root cause analysis on fail-slow drives covering a variety of ill-implemented scheduling, hardware defects, and environmental factors. We have released the dataset to the public for fail-slow study.

1 Introduction

Large-scale storage systems are susceptible to various failures [2, 3, 5, 7, 21, 32, 34, 35, 41, 42, 48]. Both academia and industry have made great efforts on identifying [26, 28, 39], detecting [9, 10, 20, 27], and fixing [8, 13, 44, 46] different kinds of failures (e.g., fail-stop [2, 18, 31, 37], fail-partial [6, 38, 40], and Byzantine [14]) in the field.

Recently, the fail-slow failures [19], also known as gray failures [24] or limpware [16], have been receiving an increasing amount of attention [23, 29, 36]. In fail-slow failures, a software or hardware component (while functioning) delivers lower-than-expected performance. With faster hardware devices (e.g., Optane SSD [49] and Z-SSD [11]) and software stack (e.g., kernel bypassing [47]), the impact of fail-slow failures (e.g., caused by malfunctioning NANDs or unfit scheduling), which might be masked as noise previously, are more likely to be noticed. Recent studies [30, 36] indicate that annual fail-slow occurrences can be as frequent as annual fail-stop events (1%∼2%).

Accurately detecting fail-slow failures is challenging. Performance variations caused by internal factors (e.g., SSD garbage collection) or external factors (e.g., workload burst) can have similar symptoms as fail-slow failures. Unlike fail-stop failures where the criteria (e.g., software crash [2], data loss [34]) are well-defined, determining fail-slow failures is usually empirical in practice and thus inherently inaccurate. Moreover, fail-slow failures are often transient [19], making it difficult for on-site engineers to identify, let alone reproduce or reason the root causes.

Although several work [23, 29, 36] has attempted to detect fail-slow failures, they are impractical and inefficient for large-scale deployment in our production cloud environment. First, these techniques require source code access (e.g., static analysis in OmegaGen [29]) or software modification (e.g., modifying software timeouts in IASO [36]), while cloud vendors like us do not touch tenants' code. Even for in-house infrastructures, inserting certain code segments is still time-consuming, as the systems can run dozens of internal services with different software stacks. Second, existing techniques can only detect fail-slow failures at the node level (e.g., IASO), thus still requiring nontrivial manual efforts to locate the culprits [19].

In this paper, we share our experiences in developing a practical, fine-grained, and general fail-slow detection framework that is applicable to a wide range of of services and devices (with minor or no adjustment) in the Alibaba cloud data centers. We start with analyzing the characteristics of the known fail-slow failures in our fleet. We then discuss three unsuccessful attempts at identifying fail-slow failures in the field, including using an empirical threshold, performing peer-evaluation-based detection [22], and refactoring IASO [36].

With the lessons learned from our earlier efforts, we design and implement PERSEUS, a non-intrusive fail-slow detection framework. We first leverage classic machine learning techniques (PCA [1], DBSCAN [43], and polynomial regression [17]) to establish a mapping between latency variation and workload pressure. With the mapping, PERSEUS can automatically derive an accurate and adaptive threshold for each node to identify slow entries within the monitoring traces. Further, based on the slow entries, PERSEUS constructs the corresponding fail-slow events and utilizes a scoreboard mechanism to evaluate the severity of such events.

PERSEUS has been deployed in our cloud for over ten months, monitoring an increasing number of drives up to around 300K by now. PERSEUS has already identified more

*Equal contribution.
†Corresponding authors.

than 300 fail-slow drives. By isolating and/or replacing the identified fail-slow drives, we significantly reduce the node-level tail latency. The 95th, 99th, and 99.99th write latencies drop by 31%, 46%, and 48%, respectively.

We compare PERSEUS to previous fail-slow detection methods as follows. We assemble a large-scale fail-slow dataset (including 315 verified fail-slow drives and around 41K of their cluster-wise peer drives) from our production traces, and build a test benchmark based on the dataset. The benchmark evaluations indicate that PERSEUS outperforms all previous methods, achieving a precision of 0.99 and a recall of 1.00. We also evaluate the effectiveness of components and the sensitivity of parameters in PERSEUS. The results show that PERSEUS can serve as a non-intrusive (based on monitoring traces), fine-grained (per-drive), general (one set of parameters fits all) and accurate (high precision and recall) fail-slow detection framework for the cloud storage systems.

We have also analyzed the reasons for fail-slow failures and discover a wide variety of root causes including ill-implemented scheduling (e.g., unnecessary resource contention), hardware flaws (e.g., bad sectors for HDDs), and environmental factors (e.g., temperature and power).

This paper makes the following contributions.

- We share our lessons on detecting fail-slow failures in large-scale data centers from three unsuccessful attempts.
- We propose the design of PERSEUS, a non-intrusive, fine-grained and general fail-slow detection framework.
- We assemble a large-scale fail-slow dataset[1] and build a fail-slow test benchmark.
- We provide an in-depth root cause analysis of fail-slow failures from the perspective of various factors.

2 Background

2.1 System Architecture

In this paper, we explore fail-slow detection methods on a subset of Alibaba Internet Data Centers (IDCs). These IDCs span across the globe and each IDC includes multiple storage clusters. Atop each cluster, a distributed file system (DFS) is deployed to support a dedicated service (e.g., block storage, NoSQL, or big data analysis). Each cluster consists of tens of racks (at most 200), and each rack contains dozens of nodes (at most 48). There are three types of storage nodes: (1) *All-flash*: a node contains 12 NVMe SSDs to store data; (2) *Hybrid*: a node contains 60-120 HDDs for data storage and 2 SSDs as write cache; (3) *All-HDD*: a node contains 70-80 HDDs to store data. By default, data storage drives in each node are of the same model. At most three drives from the same node can be taken down for repairing at a time. Table 1 lists the basic information and distribution of the drives. Note that we name drive models as vendor-model. For example, I-A stands for model A of vendor I.

[1]We release our dataset at https://tianchi.aliyun.com/dataset/144479.

Class	Model	Vendor%	Total%	Layer /Type	Cap. (GB)
NVMe SSD	I-A	3.58	1.80	32L	1920
	I-B	6.96	3.49	64L	1920
	I-C	0.76	0.38	32L	3840
	I-D	82.57	41.38	64L	4000
	I-E	6.13	3.07	64L	7680
	II-A	52.24	2.40	48L	1920
	II-B	47.76	2.19	48L	3840
SATA HDD	III-A	100	13.28	CMR	12000
	IV-A	100	32.01	CMR	12000

Table 1: Summary of drive statistics in our dataset. (§2.1). ***Vendor%****: percentage of drive models in the same vendor;* ***Total%****: percentage of drive models in the total population;* ***Layer/Type****: number of stacking layers for 3D TLC NAND SSD, or recording type for HDD;* ***Cap.****: capacity.*

Service	#Entries (M)	Total%
Log service	0.58	0.16
Big data	2.05	0.57
E-commerce	4.04	1.13
Table storage	9.32	2.61
Stream processing	12.77	3.58
Database	13.61	3.81
Object storage	30.13	8.44
Data warehouse	31.86	8.92
Block storage	252.80	70.78
Total	**357.16**	**100.00**

Table 2: Cloud services and daily entries (§2.2). ***#Entries (M)****: number of entries in millions. Each entry has five fields (i.e., avg_latency, avg_throughput, drive_ID, node_UID, timestamp).*

2.2 Dataset Description

In our data centers, a monitoring daemon is placed in each node to collect operational statistics, mainly the latency and throughput of each drive. The daemons calculate the average statistics every 15 seconds and record them as time-series data entries. The daemons run three hours a day (from 9PM to 12AM). A drive generates 720 entries (= 180 min $\times$ 4 entries/min) per day. In total, we have compiled around 100 billion entries as our dataset. Table 2 lists the daily distribution of 9 cloud services.

2.3 Impact of Fail-Slow Failures

Fail-slow failures, especially the transient ones, can often be ignored or misinterpreted as performance variations. Additionally, storage stacks usually have multiple levels (software, firmware, and hardware) of fault tolerance, such as retry and redundancy, silently masking the fail-slow failures. However, fail-slow failures can have a much more significant impact on I/O performance in the wild. Next, we will use a representative example from our object storage service to demonstrate

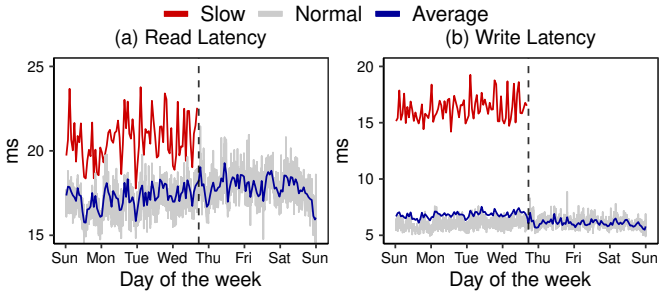

Figure 1: Fail-slow impact on I/O latency (§2.3). *The figures show (a) read and (b) write latency three days before and after isolating the fail-slow HDD (in red) in one node. Lines in grey refer to the latency distribution of normal peers from the same node. The vertical dashed line refers to the time when the fail-slow drive was taken down for replacement.*

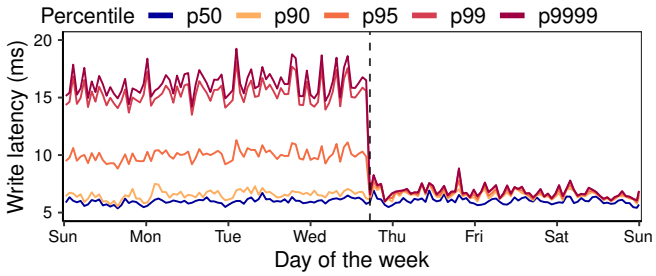

Figure 2: Fail-slow impact on tail latency (§2.3). *The figure shows the node-level write latency (of the same node as in Figure 1) at different percentiles. The vertical dashed line refers to the isolation time.*

the effect of fail-slow failures and consequently the importance of fail-slow failure detection.

Figures 1a and 1b show the read and write latency of a fail-slow HDD (the red line) against the latency of the peer HDDs (the gray lines) from the same node. The vertical dashed line indicates when the fail-slow drive was isolated (and taken down for replacement). While the fail-slow drive shows much higher read/write latency than the mean (the blue line) of all the drives (2.06∼3.65× higher for write and 1.01∼1.50× higher for read), its other metrics, such as IOPS, remain normal.

It was the utilization rate rather than the latency that directly led us to identify this fail-slow failure. Our monitoring system indicated that the victim node had been receiving much fewer writes than other nodes. The log analysis further suggested that the load balancer of the distributed object storage system prefers not to allocate writes to the victim node due to its abnormally high retry rate caused by high tail latencies. Figure 2 presents the 50^{th}, 90^{th}, 95^{th}, 99^{th}, and 99.99^{th} percentile latency of the victim node before and after the dashed line when the fail-slow drive was isolated. Before isolating the fail-slow drive, the 99.99^{th} tail latency is 2.43∼3.29× that of the median at the node level.

This example, along with other similar cases, shows that fail-slow failures impact not only the victim drive but also the entire node for a long period of time. This motivates us to explore effective measures for detecting fail-slow failures in our cloud.

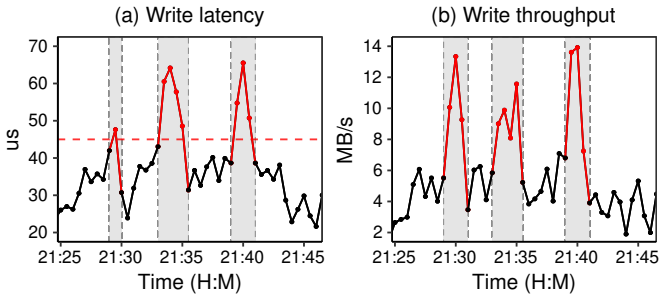

Figure 3: Sudden latency increase due to heavy load (§3.2). *The figures present the time series of (a) write latency and (b) write throughput of an NVMe SSD. Red points in grey boxes refer to time segments where drive latency is higher than a naive alarming threshold (i.e., 45 µs).*

3 Unsuccessful Attempts & Lessons

In this section, we first describe the design goals of the detection framework, and then discuss three unsuccessful early attempts. We conclude this section with a series of lessons to guide the design of PERSEUS.

3.1 Design Goals

From our perspective, a practical fail-slow detection framework should have the following properties.

- *Non-intrusive.* As cloud vendors, neither can we alter users' software nor require them to run specific modified versions of software stacks. Therefore, we can only rely on external performance statistics (e.g., drive latency) for detection.
- *Fine-grained.* Fail-slow root cause diagnosis can often be time-consuming (e.g., days to even weeks [19, 36]). We expect the framework to pinpoint the culprit.
- *Accurate.* The framework should have satisfying precision and recall to avoid unnecessary diagnosis on false positives.
- *General.* The framework can be deployed on both SSD and HDD clusters and quickly applied to different services (e.g., block/object storage and database) with minor adjustments.

3.2 Attempt 1: Threshold Filtering

Methodology. Intuitively, we can set up a hard threshold on drive latency to identify fail-slow drives based on the Service Level Objectives (SLOs). To avoid mislabeling due to one-off events such as SSD internal GC, we further specify a minimum slowdown span for a suspicious drive to be considered as fail-slow.

Limitation. Enforcing a hard threshold on device latency is clearly non-intrusive and fine-grained. However, the accuracy of threshold-identified fail-slow is low, as the latency is highly influenced by the workloads. Here, we use the latency and throughput traces of an NVMe SSD from the block storage service as an example. The left of Figure 3 illustrates the

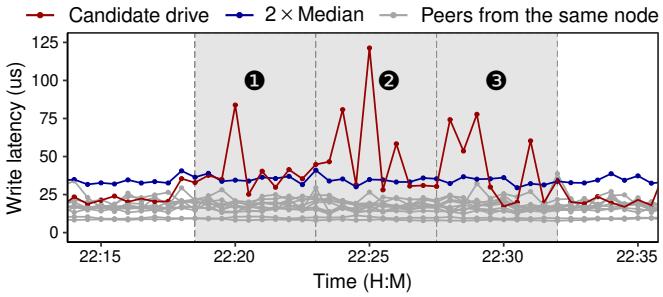

Figure 4: Peer evaluation (§3.3). *The figure shows peer evaluation results by comparing the candidate drive latency (in red) with node-level median latency (in blue) using a 5-minute sliding window. The candidate is slow (> 50% latency records higher than 2×Median) in three time windows (labeled with numbers in grey boxes).*

latency variation of the drive where the horizontal dashed line indicates the threshold (45μs). The right of Figure 3 is the corresponding throughput. We can see that three latency spikes occur at around 21:29, 21:34, and 21:40 as the latency increases up to 65μs. By comparing latency with throughput, it is clear that the workload pressure causes these spikes.

Hence, the dilemma is as follows. Setting a relaxed threshold easily mislabels normal performance variations as fail-slow events. Meanwhile, a strict one could leave many fail-slow cases undiscovered. Further, using a set of thresholds for different scenarios through fine-tuning can be fairly time-consuming as our experiments show that latency variation is a factor of drive models and workloads. In practice, we use threshold-based detection as a fail-safe measure like timeout.

3.3 Attempt 2: Peer Evaluation

Methodology. The problem of the first attempt is not having an adaptive threshold for detection. To address this problem, we explored the idea of peer evaluation [22, 30]. The rationale behind this approach is that, with load balancing across the distributed storage system, drives from the same node should receive similar workload pressure. Since fail-slow failures are relatively rare [19] and the majority of drives in a node should be normal, we can identify the fail-slow drive by comparing the performance between drives from the same node.

Specifically, we first calculate the node-level median latency at each entry timestamp (every 15 seconds). We then evaluate whether there are drives constantly (more than half of the time) delivering abnormal performance—twice slower than the median in our case—during the monitoring window (e.g., 5 minutes). If so, the detection framework reports a fail-slow event, and the monitoring window moves forward to start the next round of evaluation.

Figure 4 provides an example of peer evaluation detection, including the fail-slow drive (the red line), the adaptive threshold (2×median, the blue line), and the normal drives (grey lines). The three shaded regions (numbered ❶, ❷, and ❸) indicate three monitoring windows when the victim drive experiences a fail-slow event.

Limitation. Peer evaluation can obtain an adaptive threshold (the blue line in Figure 4), but it requires more empirical parameters than threshold filtering, such as the slowdown degree and the monitoring window span, for tuning. Although it is possible to fine-tune the parameters on a few clusters for specific storage services, the effort would be prohibitively large if we want to extend peer evaluation to other drive models of different services. For example, it took on-site engineers two hours to fine-tune a cluster with around 300 nodes, and this set of parameters fails to work on another cluster even under the same service with the exact same models of drives.

3.4 Attempt 3: IASO-Based Model

Methodology. IASO is a fail-slow detection framework focusing on identifying performance-degrading nodes [36]). The design principle of IASO is to leverage software timeouts and convert them into informative metrics to benchmark fail-slow. However, directly using IASO is not suitable for us. First, IASO requires code changes (i.e., intrusive monitoring) to insert or modify certain code snippets of the running instances (e.g., Cassandra and ZooKeeper), thus leveraging software-level timeouts to identify fail-slow incidents[2]. Second, IASO is node-level detection, whereas our goal is device-level. Nevertheless, we re-factor IASO with our best effort. To avoid modifying the software, we reuse the fail-slow event reporting by peer evaluation (i.e., Attempt 2, see §3.3).

Limitation. The IASO-based model delivers rather unsatisfactory performance (see §5.4 for details) on our assembled benchmark, with a precision rate of only 0.48. We suspect the main reason is that using the fail-slow event reporting to replace the software timeout might not be effective. Moreover, we have explored other possible alternatives, such as replacing software timeouts with thresholds. However, the results are still unsatisfactory. Therefore, we assume IASO, even with refactoring, may not achieve our goals.

3.5 Guidelines for PERSEUS

The aforementioned methods are either labor-intensive (requiring extensive tuning) or ineffective in the field. In this subsection, we use a series of research questions as guidelines for designing our next fail-slow detection framework.

RQ1: What metrics should we use?

Throughout the early development of previous attempts, we mostly focused on the write performance (i.e., the latency/throughput of write) for two reasons. First, among the verified fail-slow cases, more than half of them only have a notable influence on writes. Even for the rest, the impact on read performance is always much smaller (similar to Figure 1). Second, fail-slow failures have more severe impacts on writes. In our storage systems, most clusters require the

[2]Note that the definition of "non-intrusive" is different in IASO (meaning low overhead introduced) from ours (meaning no code changes).

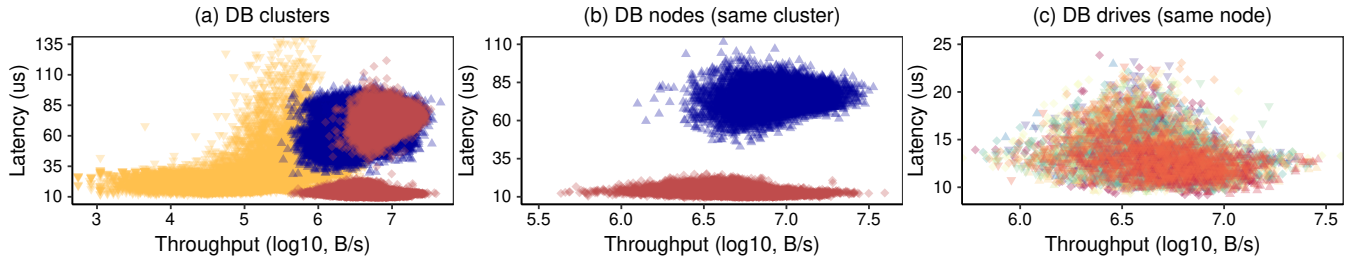

Figure 5: Distinct LvT distribution (§3.5). *The figures show the latency-vs-throughput (LvT) distribution of (a) three clusters (in yellow, blue and red) from database service, (b) two representative nodes (in blue and red) with clear-cut distribution from the same cluster (also from database service), and (c) drives (in distinct colors) in one node from database service. Throughput (unit: B/s) is scaled by log base 10 here and in related figures hereafter.*

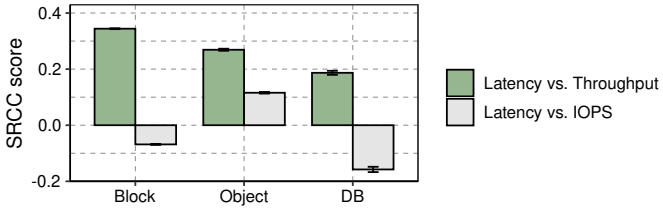

Figure 6: SRCC scores (§3.5). *The figure shows the SRCC scores between drive write latency and throughput/IOPS from three major services. Error bars refer to 95th percentile confidence intervals.*

write request to return after all three replicas ACKed (only <20% of our clusters require 2 replicas). Meanwhile, a read request will return as soon as one replica returns. In this case, fail-slow failure impacts the write request more often as one fail-slow write can lead to a slow write request in most cases while only 1/3 of the chances for a read request. Note that write reallocation cannot remedy for fail-slow write requests as the reallocation is triggered by a timeout (usually much longer than a fail-slow event). Nevertheless, we still evaluate fail-slow detection based on the read performance (see §5.5), and the results confirm the above assumptions.

RQ2: How to model workload pressure?

Simply depending on latency to detect fail-slow failures is unreliable. Our previous exploration has shown that workload pressure can significantly influence the latency variation. Further, we explore other indexes to model workload pressure to better understand the latency variation.

Analyzing Figure 3a and Figure 3b inspires us to check the possibility of using throughput or IOPS to model workload. Currently, per-drive I/O throughput (unit: byte/sec) and IOPS (unit: count/sec) are both available and stored in the same fashion as drive latency (see Section 2.2). In Figure 6, we measure the per-drive latency correlation with throughput and IOPS using Spearman's Rank Correlation Coefficient (SRCC [15]) across three representative services. A higher SRCC value indicates a stronger correlation. We can see that latency is more closely related to throughput than IOPS. Moreover, in certain services, latency is even negatively correlated with IOPS (e.g., block and database). Therefore, we decide to use throughput for modeling the workload pressure.

RQ3: How to automatically derive adaptive thresholds?

In Attempt 2 (§3.3), we discover that, though peer-evaluation can provide adaptive thresholds, this solution requires time-consuming tuning for different service types and drive models. Now, with workload pressure modeled by throughput, we are able to build the latency-vs-throughput (LvT) distribution. Then, we can use regression models on such distribution to define a statistically *normal* drive and subsequently use its upper bound as the adaptive thresholds for various environments.

To build such regression models, we need to determine the scope of drives to be included in the LvT distribution. The tradeoff is that including more samples (e.g., all drives from the service) can be more statistically confident but subject to a more diverse distribution—difficult to derive a clear upper bound. Therefore, we plot the distribution at three different scales in Figure 5 and discuss their pros and cons as follows.

Service-wise. In Figure 5a, we plot the LvT distribution of drives from three clusters (marked as red, yellow and blue) in the database service. First, we can observe that the clusters from the same service can have drastically different LvT distributions. For example, samples from the red cluster rarely overlap with those from the yellow cluster. This indicates that directly using all drives from the service to build the distribution is not applicable.

Cluster-wise. In Figure 5a, we notice that even samples from the same cluster can have clear-cut distributions (i.e., the two red regions). After statistical analysis, we discover that the disparity is widespread. For clarity, in Figure 5b, we plot the LvT distribution of drives from two different nodes (in red and blue) from the same cluster. The huge gap between distributions indicates that we also cannot rely on cluster-wise peer evaluation.

Node-wise. Finally, in Figure 5c, we plot the LvT distribution of drives from one all-flash node (12 SSDs). Each drive is represented with a color. We can see the colors are well clustered together which indicates drives from the same node follow a similar LvT distribution. Note that we also examine

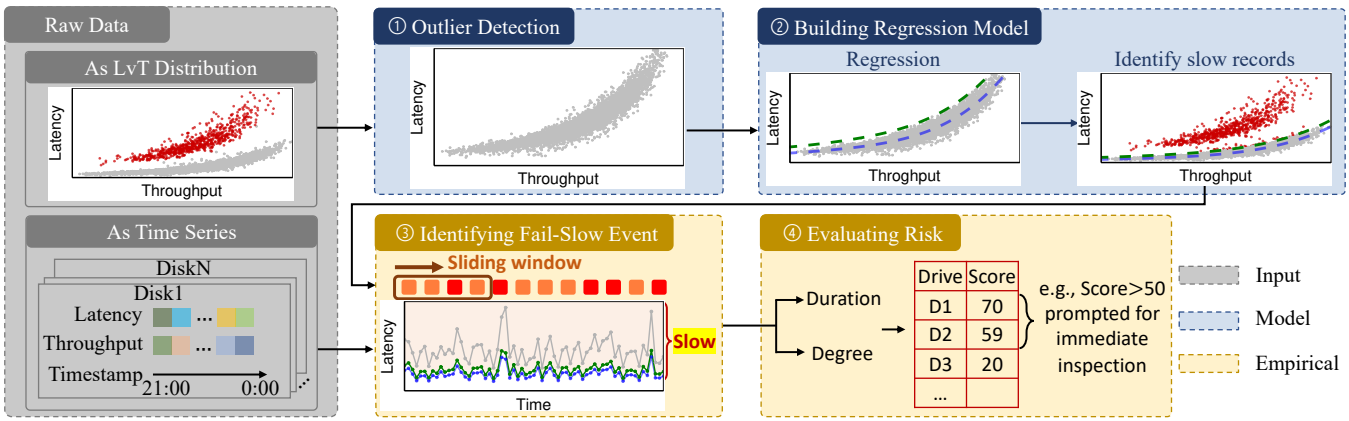

Figure 7: PERSEUS design diagram (§4.1). *From the raw data,* PERSEUS *seeks to distinguish the slow (in red) from the normal (in grey) by building a regression model (②) with preliminary outlier detection (①). With a sliding window,* PERSEUS *formulates consecutive slow records into slowdown events (③), and assigns risk scores based on slowdown duration and degree (④).*

nodes from other services and confirm that such behaviors persist across node configurations (e.g., all-flash or hybrid) and services (e.g., block or object storage). Thus, we decide to use the node-wise samples to build the LvT distribution.

RQ4: How to identify fail-slow without a criterion?

Unlike fail-stop failures, there are no clear criteria for detecting fail-slow drives. First, both the device (e.g., an SSD) and the software (i.e., users' code) can be a blackbox to the on-site engineers. Second, fail-slow failures can be temporal with varying symptoms. Moreover, the root causes of fail-slow failures can be too obscure to diagnose. As a result, we cannot exclude the possibility of mislabeling fail-slow failures.

Therefore, we rethink our strategy on fail-slow detection. Instead of relying on the framework to output a binary result (fail-slow or not), the detection tool should describe the likelihood of a drive to fail-slow. With sufficient accuracy, on-site engineers can focus on the most severe ones. While this may still leave some fail-slow drives undiscovered, it is acceptable as they behave like normal performance variations.

4 PERSEUS

With lessons from previous attempts, we propose PERSEUS, a non-intrusive, fine-grained and general fail-slow detection framework. The core idea of PERSEUS is building a polynomial regression on the node-level LvT distribution to automatically derive an adaptive threshold for each node. PERSEUS can use the threshold to formulate fail-slow events and further use a scoreboard mechanism to single out the drives with severe fail-slow failures. In this section, we first introduce the high-level workflow and then discuss the design of each step at length.

Our dataset can be viewed as a time-series dataset and each entry has five fields (i.e., avg_latency, avg_throughput, drive_ID, node_UID, timestamp). Every day, the monitoring proxy would gather 720 entries (180 minutes × 4 entries/min) from each drive (as raw dataset) and send them to PERSEUS for a four-step detection procedure.

4.1 High-Level Workflow

1. Outlier detection. For each node, PERSEUS first collects all the entries. Then, we use a combination of Principal Component Analysis (PCA [1]) and Density-Based Spatial Clustering of Applications with Noise (DBSCAN [43]) to identify and then discard outlier entries.
2. Building regression model. Based on the clean dataset (i.e., excluding the outliers), PERSEUS performs a polynomial regression to obtain the model and uses the prediction upper bound as a fail-slow detection threshold. Then, PERSEUS applies the model onto the raw dataset (i.e., including the outliers) to identify out-of-bound entries and mark them as slow entries.
3. Identifying fail-slow events. PERSEUS uses a sliding window and a slowdown ratio to identify consecutive slow entries and formulate corresponding fail-slow events.
4. Evaluating risk. Based on a risk-score mechanism [25], PERSEUS estimates the duration and degree of fail-slow events and assigns each drive a risk score based on daily accumulated fail-slow events. On-site engineers can then investigate the cases based on the severity.

4.2 Outlier Detection

Before applying regression models, a necessary pre-process is to root out noisy samples (i.e., outliers). While the LvT samples (i.e., <latency, throughput> pairs) are usually clustered together within a node (see RQ3 in §3.5), entries from fail-slow drives or under normal performance variations (e.g., internal GC) can still be deviating. Therefore, before building a polynomial regression model, we first screen out the outliers.
Using DBSCAN. Density-based clustering algorithms (measuring the spatial distance) are promising approaches for identifying the potentially distinctive groups (i.e., normal vs. slow). Initially, we employ DBSCAN [43] to label outliers.

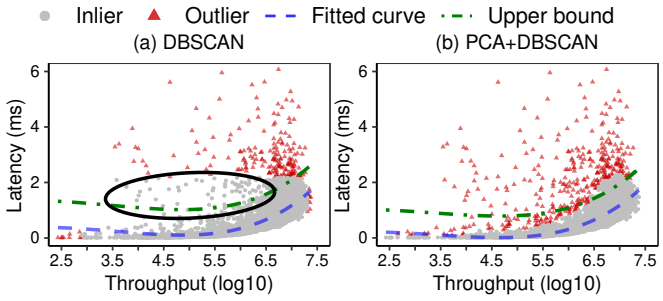

Figure 8: Outlier detection (§4.2). *The figures show the performance of the regression model based on two outlier detection schemes: (a) only DBSCAN and (b) PCA prior to DBSCAN. All data records come from the same node with one known fail-slow drive during the day.*

In a nutshell, DBSCAN groups points that are spatially close enough—distances between points is smaller than a minimum value. Note that <latency, throughput> pairs from long-term or permanent fail-slow drives can be clustered together but far away from the main cluster. Hence, we only keep one group with the most points for further modeling.

Adding PCA. Unfortunately, using the DBSCAN to sift through the raw dataset can have limited effectiveness. Here, we choose a sample node with one confirmed fail-slow drive to illustrate the limitation. In Figure 8a, we apply fine-tuned DBSCAN (outliers in red points) to the node's daily raw dataset and fit the rest of the data (grey points) to polynomial regression (with a fitted curve in blue and a 99.9% prediction upper bound in green dashed line). In this example, the DBSCAN algorithm only identifies 63.83% of slow entries.

The root cause is that the throughput and latency are positively correlated. Thus, the samples (i.e., <latency, throughput> pairs) can be skewed towards a particular direction. Hence, outliers (i.e., samples from fail-slow drives) can be mislabeled as inliers (see the black circle in Figure 8a). Therefore, we leverage Principal Component Analysis (PCA [1]) to transform the coordinates and penalize the outliers perpendicular to the skewed direction in order to reduce mislabeling. As a result, applying DBSCAN with PCA effectively detects 92.55% of slow entries (see Figure 8b).

Usage of outliers. Recall that in RQ4, we have discussed that just using binary detection cannot reflect the extent of slowdown. Therefore, we do not directly use the binary results of outlier detection, such as simply labeling outliers as slow entries (i.e., skipping §4.3) or fail-slow events (i.e., skipping §4.3 and §4.4). Rather, we exclude the outliers to build a better-fitted model for measuring slowdown degree of entries.

4.3 Regression Model

As normal drives within a node can have similar latency-vs-throughput mapping (i.e., well clustered together), we can use a regression model to describe the behavior of a "normal" drive and delineate the scope of variation for fail-slow detection. Classic regression models include linear, polynomial, and advanced ones like kernel regression. We do not use linear regression as the latency dependency on throughput is obviously nonlinear (e.g., see Figure 8). Moreover, advanced models (e.g., kernel regression) are unnecessary as the latency-vs-throughput mapping is primarily monotonic (i.e., latency increases along with the throughput). Polynomial regression is preferable as it handles nonlinearity while retaining model parsimony (i.e., achieving the desired goodness of fit with just enough parameters).

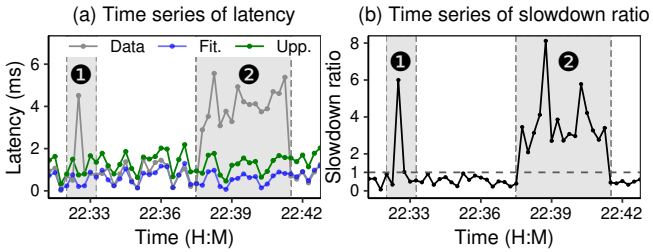

Figure 9: Identifying slowdown event (§4.4). *Figure (a) shows the original ("Data") time series of drive latency, together with the corresponding fitted values ("Fit.") and upper bounds ("Upp.") calculated from the regression model. Figure (b) shows the time series of slowdown ratio by dividing upper bound by original data. Records with a slowdown ratio higher than 1 are deemed as slow. The two grey boxes refer to a latency spike (❶) and a transient slowdown event (❷).*

4.4 Identifying Fail-Slow Event

Distinguishing slow entries. After obtaining the regression model, we can calculate a prediction upper bound to distinguish the slow entries, and use it to detect fail-slow events. For example, a 99.9% upper bound means that 99.9% of the variations are deemed normal. In practice, we use a combination of loose (i.e., 95%) and strict (i.e., 99.9%) upper bounds to avoid overfitting while identifying as many fail-slow drives as possible.

Formulating events. Next, we use both real and made-up examples to illustrate how to formulate fail-slow events. Figure 9a presents the drive latency (grey line), fitted values (blue line) and the 99.9% upper bound (green line). Slowdown Ratio (SR) is obtained from dividing drive latency by the upper bound, entry by entry (every 15 seconds). For example, let a drive's latency entries in one minute be [15, 20, 25, 10, 5] and the corresponding upper bound be [5, 5, 5, 5, 5]. The SR series would be [3, 4, 5, 2, 1]. Figure 9b presents the SR series of the candidate drive.

Next, we formulate fail-slow events by using a sliding window (similar to Attempt 2 in §3.3). The sliding window has a fixed length (i.e., a *minimum span*) and starts at the first entry. Within the span, if a certain *proportion* of SR series has a median SR value exceeding the *threshold*, PERSEUS would record that the drive has encountered a fail-slow event within the span and see if the event should be extended to the next entry.

Slowness	Duration (min)		
	Long-term ≥120	Moderate [60, 120)	Temporal [30, 60)
Severe (SR≥5)	Extreme	High	Moderate
Moderate (SR∈[2, 5))	High	Moderate	Low
Mild (SR∈[1, 2))	Moderate	Low	Minor

Table 3: Fail-slow risk matrix (§4.5). PERSEUS *assigns risk levels based on daily accumulated slowdown events. For example, drives at extreme risk for one day should experience a long-term slowdown (for 120-180 minutes in total) with a severe slowdown ratio (SR) on average (SR ≥5).*

As for the example above (i.e., an SR series of [3, 4, 5, 2, 1]), we set the *minimum span* as one minute (i.e., four entries), the *proportion* to be 50%, and the *threshold* to be 1. Then, the first four SR entries can form a fail-slow event as more than 50% of the SR values (i.e., 3, 4, 5, 2) have a higher median (i.e., 3.5) than the threshold (i.e., 1). For the same reason, this fail-slow event should include the fifth entry (i.e., 1).

In practice, we set the *minimum span* as 5 minutes, the *proportion* to be 50%, and the *threshold* to be 1, meaning the event should be slower than the upper bound. Our rationale is to only formulate fail-slow events under a persistent series of slowdown entries as one-off spike entries are likely to be acceptable performance variations. In Figure 9b, while both ❶ and ❷ have high SR values, only ❷ would be marked as a fail-slow event.

4.5 Risk Score

Recall our discussion on RQ4 and the usage of outlier detection, we also do not simply rely on the existence of fail-slow events to label the corresponding drive as "fail-slow." In fact, if we simply mark drives with one fail-slow event as fail-slow, we can easily obtain 6K such "fail-slow" cases on a bad day.

Therefore, we adopt the idea of establishing a risk score mechanism from performance regression testing [25]. In Table 3, slowdown duration and severity are classified into different risk levels (in shades of grey). For example, in our case, according to the daily slowdown span, the duration of fail-slow is classified into temporal (from 30 to 60 minutes), moderate (from 60 to 120 minutes) and long-term (from 120 to 180 minutes). Besides, based on the average slowdown ratio of the day, the slowness of fail-slow is evaluated as mild (1≤SR<2), moderate (2≤SR<5), or severe (SR≥5).

To examine fail-slow, a per-drive risk score is calculated by assigning different weights to risk levels:

$$\begin{aligned} \textit{Risk Score} = {} & N_{extreme} \times 100 + N_{high} \times 25 + \\ & N_{moderate} \times 10 + N_{low} \times 5 + N_{minor} \times 1 \end{aligned} \quad (1)$$

$N_{extreme}$ *refers to #days at extreme risk level.*

If a drive whose risk scores exceed a minimum value (i.e., *min_score*) within the most recent N days, the drive will be recommended for immediate isolation and hardware inspection. Note that all drives in our fleet, HDDs and SSDs, use the same scoring mechanism.

Service	Device	#Node	#Fail-slow
Stream processing	NVMe SSD	47	1
Table storage	NVMe SSD	87	1
Big data	NVMe SSD	119	1
Data warehouse	NVMe SSD	663	1
Database	NVMe SSD	96	2
E-commerce	NVMe SSD	223	6
Log service	SATA HDD	34	36
Object storage	SATA HDD	1426	42
Block storage	NVMe SSD	734	225
Total	**-**	**3429**	**315**

Table 4: Test dataset size (§5.1).

5 Evaluation

5.1 Fail-slow Benchmark

One significant challenge of testing fail-slow detection frameworks is the lack of a benchmark. Existing fail-slow datasets [19, 36] only record high-level administrative information of fail-slow incidents and thus cannot be used for evaluation. Therefore, we build and release a large-scale fail-slow detection benchmark based on verified fail-slow drives and production-level traces.

Benchmark size. Table 4 presents a summary of our benchmark. Specifically, our dataset includes 886 million operational traces of 15 consecutive days from 41K drives and 25 clusters. Among them, 315 drives (237 SSDs and 78 HDDs) are verified fail-slow and thus labeled as positive; the rest are normal peer drives from the same clusters. Among the verified cases, 304 are detected by PERSEUS. All fail-slow drives are verified by either on-site engineers or manufacturers. Their root causes include software scheduling bugs, hardware defects, and environmental factors.

Workload heterogeneity. The benchmark covers 9 major services (see Table 4). These services can have drastically different I/O accessing patterns and subsequently various LvT distributions.

Benchmark setup. In our dataset, 252 fail-slow drives are caused by software scheduling (see Section 6.1). To avoid potential concerns that PERSEUS may be specific to the Alibaba stack, we set up two scopes: (1) the full test dataset, and (2) a subset excluding traces from clusters with software-induced fail-slow drives. We only show results with the highest evaluation scores in Table 5 and Table 7.

5.2 Test Candidates

We compare PERSEUS with three models based on our early explorations, namely threshold filtering (§3.2), peer evaluation (§3.3), and the IASO-based model (§3.4). In this section, we introduce their implementation and configuration details.

5.2.1 Threshold Filtering

We include both statistical and empirical thresholds.

Statistical bound. We derive the following statistics as the upper bounds: (1) an X^{th} percentile where X ranges from 75

to 99; or (2) an interquartile range ($IQR = 3rd\_quartile - 1st\_quartile$) [30]. Drives are classified as fail-slow if their median latency during the three-hour monitoring exceeds the upper bound.

Empirical bound. We manually set a latency upper bound for each node setup of each service based on the Service Level Objectives (SLOs) or the suggestions from on-site engineers (e.g., 300 μs for the *all-flash* setup in block storage service).

5.2.2 Peer Evaluation

Recall that the peer evaluation approach identifies a fail-slow drive if its latency is at least X times that of the node median for a minimum duration (denoted *min_dur*). To obtain the best performance, we explore different sets of parameters (i.e., X from 1.5 to 3 and *min_dur* from 0 to 150 minutes).

5.2.3 IASO-Based Model

We re-implement IASO [36] strictly following its original design and only modify parts when necessary. Since IASO detects fail-slow on a per-node basis, we label the results as true positive if the node contains a fail-slow drive. We list key implementation details as follows.

Epoch. The size of each `epoch`, instead of 5 seconds, is adjusted to 15 seconds (the finest granularity of our raw dataset).

Timeout. The `timeout` signals are converted to the number of *slow* drives in a node. During each `epoch`, the *slow* drives refer to the drives whose latency records (i) exceed pre-defined empirical bounds (Attempt 1 in §3.2) or (ii) are at least 2× the median latency of all drives from the same node (Attempt 2 in §3.3). The `response` is set as the total number of drives in each node.

DBSCAN configuration. IASO records peer scores (the higher the slower) among nodes, and only keeps one outlier with the highest score for its further mitigation procedure. Here, since we only evaluate the detection part of IASO, we retain all outliers classified by the DBSCAN and set different score thresholds to fine-tune IASO.

5.2.4 PERSEUS

The deployed PERSEUS adopts outlier detection (§4.2) and uses the combination of two prediction upper bounds (i.e., 95% and 99.9%) to formulate fail-slow events (§4.4). The monitoring period (N) is set as 15 days for both upper bounds, and the alert score (*min_score*) is set as 90 for the former and 40 for the latter (§4.5). Note that PERSEUS uses the same set of parameters for all node configurations (e.g., all-flash and hybrid) across different services (e.g., block/object storage and big data).

5.3 Evaluation Metrics

We adopt three evaluation metrics: precision rate, recall rate, and Matthews Correlation Coefficient (MCC [33]). The precision indicates the percentage of drives identified by a method is indeed a fail-slow one. The recall is the percentage of real fail-slow drives identified by a method. Since our test dataset is highly unbalanced (i.e., the positive-to-negative ratio is 1:137), we further adopt MCC as it evaluates binary classification models more fairly on imbalanced datasets and can offer a more informative and convincing score compared to other widely adopted metrics like accuracy and F1-score [12].

Metric	Thresh-Stat	Thresh-Emp	Peer Eval	IASO-Based	PERSEUS-Deployed
Full-set					
Precision	1.00	1.00	0.98	0.48	0.99
Recall	0.52	0.02	0.57	0.24	1.00
MCC	0.72	0.14	0.74	0.32	0.99
Subset (excluding software-induced)					
Precision	1.00	1.00	1.00	0.45	0.94
Recall	0.71	0.09	0.65	0.61	1.00
MCC	0.84	0.30	0.80	0.52	0.97

Table 5: Overall evaluation results (§5.4). *The table shows the best evaluation scores of threshold filtering based on statistical (**ThreshStat**) and empirical (**ThreshEmp**) bounds, peer evaluation (**PeerEval**), and the* IASO*-based model. For* PERSEUS*, we list the results of the deployed version. Each method is evaluated on both the full-set and the subset.*

5.4 Evaluation Results

Table 5 summarizes the performance results. For previous attempts (Thresh-Stat to IASO-Based columns), we choose the best performance. For PERSEUS, we use the results of the deployed version. The upper half includes the results from the full benchmark tests and the lower half includes results from the benchmark excluding the scheduling-induced failures.

Threshold-based. We can see that, in both the full-set and the subset, the two threshold-based approaches can achieve a precision of 100%. However, the recalls are subpar, especially the empirical threshold method. This is understandable as strict thresholds, on the one hand, can expose extremely slow drives which are highly likely to be caused by fail-slow failures. On the other hand, such methods can leave drives with only mild fail-slow symptoms undiscovered.

Peer-evaluation. Using peer evaluation to detect fail-slow failures also yields high precision but low recall. The reason is that it generally adopts an adaptive threshold (i.e., X times the node median) to formulate fail-slow events. Thus, it faces the same problem as threshold-based methods.

IASO-based. While we have tried our best effort on refactoring and fine-tuning IASO, its performance is rather disappointing. With an MCC score of 0.32, IASO even falls behind using a simple statistical threshold or using a peer-evaluation method. We believe there are two aspects of reasons. First, IASO heavily relies on software timeouts which can not be simply replaced with other metrics (e.g., node-level slow drives). Second, its algorithm is designed for node-level detection where a finer grained event (e.g., a fail-slow drive) may not trigger enough alerts.

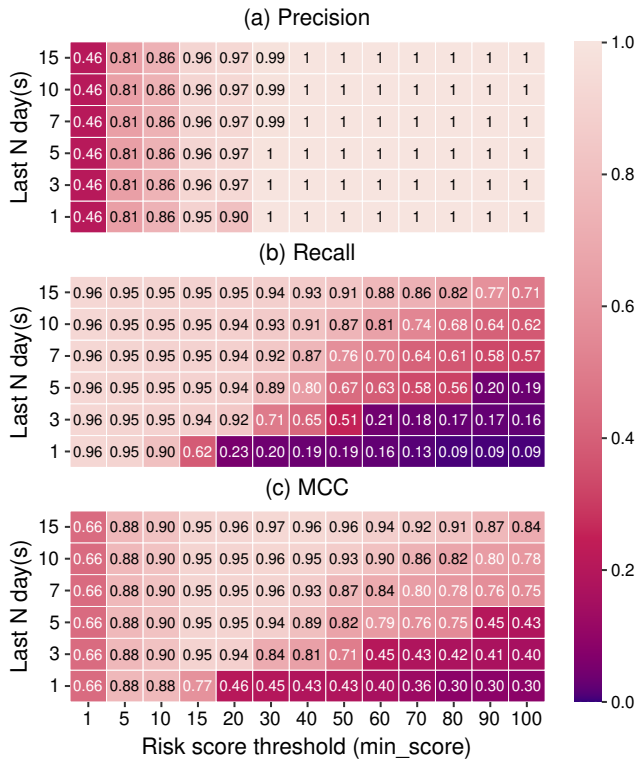

Figure 10: ***min_score* and *N* of PERSEUS (§5.5).** *The heatmaps show evaluation scores of* PERSEUS*-p999 in Table 7 under different min_score and N. The lighter the color, the higher the score, the better the result.*

PERSEUS. Table 5 shows that PERSEUS outperforms all previous attempts. The high precision and recall indicate that PERSEUS can successfully detect all fail-slow drives while rarely mislabeling normal as fail-slow. Therefore, we conclude that PERSEUS achieves our design goals as a fine-grained (per-drive), non-intrusive (no code changes), general (same set of parameters for different setups) and accurate (high precision and recall) fail-slow detection framework.

5.5 Effectiveness of PERSEUS Design

Now, we take a closer look at the effectiveness of procedures and sensitivity of parameters in PERSEUS. In Table 6, we list the main options or ranges of configurable parameters in PERSEUS. Next, we discuss the effectiveness of PERSEUS's procedures by enabling or disabling particular functionalities and explore different sets of parameters. The evaluation results are listed in Table 7. Note that the deployed version uses a combination of p95 and p999 upper bounds.

Outlier detection. By disabling the outlier detection, we can see that the precision is approximately the same, while the recall plummets to only 0.51. This indicates that, without outlier detection, PERSEUS may fail to distinguish samples from fail-slow drives, thus yielding a low recall.

PCA. Surprisingly, if we enable outlier detection but disable PCA, we find out the performance becomes even worse than simply without outlier detection. This experiment confirms the importance of correcting mislabeling samples via PCA.

Parameter	Range	Description
S1: Outlier detection (§4.2)		
PCA	On/Off	Transform the coordinates w.r.t. the principal components.
DBSCAN	On/Off	Density-based outlier detection.
S3: Identifying fail-slow event (§4.4)		
X	95∼99.9	Use the *X*% prediciton upper bound as the latency upper bound.
S4: Evaluating risk (§4.5)		
min_score	1∼100	Risk score threshold.
N	1∼15	Evaluate the risk score of the most recent *N* days.

Table 6: Evaluating PERSEUS's design choices (§5.5). *The table summarizes the parameter settings of evaluating* PERSEUS*'s design choices. PCA and DBSCAN are switched on and off to demonstrate their effectiveness.*

Metric	w/o Outlier	w/o PCA	p95	p99	p999	Deployed
Full-set						
Precision	0.98	0.55	0.99	1.00	1.00	0.99
Recall	0.51	0.43	0.99	0.93	0.93	1.00
MCC	0.71	0.49	0.99	0.96	0.96	0.99
Subset (excluding software-induced)						
Precision	0.95	0.36	0.94	0.98	1.00	0.94
Recall	0.82	0.91	0.95	0.92	0.95	1.00
MCC	0.88	0.57	0.95	0.95	0.98	0.97

Table 7: Evaluation results of PERSEUS's design choices in Table 6 (§5.5). *Only results based on the best sets of parameters with the highest MCC scores are shown.*

Prediction upper bounds. We set various prediction upper bounds from p95 to p999. The optimal one on the full-set is the p95 upper bound, with nearly perfect precision and recall both at 0.99. For the subset, a stricter bound of p999 is preferred as fail-slow in the subset is usually with severe slowdowns. In practice, the deployed version uses a combination of p95 and p999 upper bounds to strike a better balance on both benchmarks.

Risk score mechanism. Figure 10 evaluates the scoring mechanism. With a larger *min_score* and *N*, the precision usually increases while the recall decreases. This is because, as *N* becomes larger, drives—that have occasional but less severe slowdowns—could be misidentified with enough scores counted from more days.

Evaluating the read performance. Fail-slow failures impact the write performance more often than the read. Among the 315 fail-slow drives in our dataset, 49 are fail-slow in both write and read while 223 are only fail-slow in the write. Unfortunately, we do not have read traces of the remaining 43 fail-slow ones.

Runtime overhead. On Intel Xeon 8-core CPU 2.5GHz with 16GB RAM, the per-node execution time of PERSEUS is measured as 0.21±0.14 seconds. With low overheads, PERSEUS can detect fail-slow drives in tens of thousands of nodes each day on a single machine.

5.6 Benefit of Deployment

The most direct benefit of deploying PERSEUS is reducing tail latency. By isolating the fail-slow, node-level 95th, 99th and 99.99th write latencies are reduced by 30.67% (±10.96%), 46.39% (±14.84%), and 48.05% (±15.53%), respectively.

6 Root Cause Analysis

We further analyze the root causes of the 315 fail-slow drives in the test dataset. Among them, 216 SSDs and 36 HDDs are impacted by ill-implemented software scheduling. The remaining 42 HDDs and 21 SSDs are verified by our on-site engineers as hardware-related fail-slow failures and further sent back to vendors for detailed analysis. Due to the lengthy diagnosis process, we only obtain root causes for 15 drives (9 HDDs and 6 SSDs).

6.1 Ill-Implemented Scheduler

6.1.1 Case 1: In Open-Channel SSD Cluster

Symptom. In two clusters, PERSEUS has identified a total of 216 fail-slow drives constantly showing abnormal performance. This is unconventional as hardware-related fail-slow occurrences are usually rare and independent. Further investigation reveals that all detected fail-slow drives are with the same logical IDs, i.e., $disk_1$ and $disk_2$. After checking the per-node latency time series, we discover that the latency of individual drives in these nodes is positively correlated with their logical IDs, i.e., latency level: $disk_1 > disk_2 > \cdots > disk_{12}$. In other words, among those nodes, $disk_1$ is always the slowest, followed by $disk_2$ and so on.

Root cause. Each node in these two clusters is equipped with 12 open-channel SSDs (OC SSDs), whose Flash Translation Layer (FTL) is managed by the host. For each node, the host allocates 12 CPU cores to manage 12 OC SSDs, respectively (e.g., $core_0$-$core_{11}$ for SSD_0-SSD_{11}). The root cause is that the OS scheduler places system tasks on CPU cores by domain (a domain includes 6 CPU cores). Upon receiving a new system task (e.g., `ps` command), the scheduler first checks if the current core and last-selected core are idle. If not, starting from the first core in the domain, the scheduler iterates through all cores in order and attempts to preempt a core for running the task. As a result, OC drives with smaller IDs (e.g., $disk_1$) are more likely to be preempted and encounter fail-slow failures.

Fix. We modify the scheduler to no longer preempt the CPU cores assigned to the OC SSDs. After the fix, OC SSDs in these clusters no longer suffer fail-slow failures.

6.1.2 Case 2: In All-HDD Cluster

Symptom. In one cluster, PERSEUS has identified 36 fail-slow HDDs. This cluster is of an all-HDD setup with 76 HDDs in each node. There are three interesting facts about the distribution of the fail-slow HDDs from this cluster. First, the fail-slow failures always show up in fixed combinations of pairs. For example, if there are two fail-slow HDDs in a node, they would be $disk_0$ and $disk_{75}$. If there are 6 fail-slow HDDs, they would be $disk_{0-2}$ and $disk_{73-75}$. Second, all fail-slow HDDs are experiencing a similar level of slowdown. Third, in each node, the number of fail-slow drives is always twice the number of offline drives.

Root cause. In each node of this cluster, the OS assigns each HDD a thread to manage its I/O. The assignment follows a simple algorithm:

$$Thread\_ID = Disk\_ID \ \ mod \ \ \#Drives. \tag{2}$$

Therefore, as each node has 76 HDDs, normally $thread_0$ manages $disk_0$ and so on. However, when a drive is put offline, the number of drives changes and the assignment acts accordingly. For example, assume $disk_{20}$ crashes and the total number of drives now becomes 75. Then, $disk_0$ and $disk_{75}$ would share $thread_0$ ($0 \equiv 75 \bmod 75$) and thus suffer from fail-slow failures due to I/O contention. Similarly, two or three drives crash can result in corresponding two or three pairs of fail-slow occurrences.

Fix. We modify the assignment policy to only allow one HDD per thread and thus avoid the fail-slow occurrences.

6.2 Hardware Defects

Bad sector. Bad sectors are usually an artifact of physical damage (e.g., manufacturing defect or scratched by the read-/write head) [40]. To deal with them, disk firmware maintains a pool of spare sectors to reallocate the original data on bad sectors. Moreover, firmware remaps the logical address of bad sectors to the physical address of spare sectors. Upon host requests on an unmarked bad sector, the disk will suffer from long seek time (i.e., time spent for reallocating data). According to field events, three HDDs are reported to have a large number of bad sectors, resulting in repeated remapping and reallocating, and obviously fail-slow failures. Note that one can not simply infer the root causes based on the occurrences of such errors. The reason is that the hardware defects are usually neither necessary nor sufficient conditions for fail-slow failures.

Rotor eccentricity. Disk motor spins the platter at high speed (e.g., 7200 RPM for consumer-level HDDs). If the rotor in the spindle motor rotates with eccentricity, it will cause a lot of noise and vibration. For such disks, the read/write heads would frequently fail to locate targeted positions, resulting in considerable I/O delay. Two fail-slow HDDs are reported to suffer from slight rotor eccentricity in our field events.

Bad capacitors. SSD adopts a small amount of DRAM as an internal write-back cache to boost both read and write performance. If the DRAM capacitors are malfunctioning, SSD will be forced to stop using the cache since it is volatile (i.e., data loss upon power down), causing severe performance degradation. Instead, SSDs now only ACK after data have been directly flushed to the NAND successfully, incurring long latencies on writes. In total, four SSDs are found to have bad capacitors.

Read-only mode. Drives with severe errors (e.g., in the face of too many bad sectors) are reset to (temporal) read-only mode to prevent further data loss. Upon read-only mode, drives are blocked from executing any write command. As a result, two SSDs are found to be stuck in read-only mode.

6.3 Environment

Temperature and power are common sources of fail-slow incidents [19, 48]. According to field events, one fail-slow HDD suffered from temperature throttling due to high environmental temperature. Another three HDDs were related to insufficient power supply events.

7 Limitation

Multiple fail-slow occurrences. PERSEUS leverages an important precondition that fail-slow failures should be rare in the field. However, if a key component on the critical data path (e.g., HBA card) breaks down, all drives would be impacted and result in severe delays. In this case, PERSEUS may not be able to detect the performance anomalies as the LvT distribution can be skewed for all drives. At the moment, we are investigating the possibility to perform inter-node LvT distribution to enhance PERSEUS's ability on discovering multiple fail-slow occurrences within the same node.

Generalizability. Utilizing the LvT distribution to identify fail-slow drives also depends on the fact that all drives within the node have the same drive models and similar workloads. In our storage systems, drives have the same configuration, and multiple levels of load balancing assure that the workload on each drive is similar within the same node. While this is a common practice for large-scale storage systems [30, 32, 35, 48], it might not be the case for small-scale servers (e.g., private cloud), where drives in the same node can have drastically different workloads and configurations. Under such circumstances, the accuracy of PERSEUS can be affected.

Comprehensiveness. PERSEUS currently uses traces from 9PM to 12AM each day to reduce interference (see Section 2.2). It is possible that some fail-slow failures could only be triggered during a particular time window or under heavier workloads. We are working on designing a more efficient daemon to collect traces during busy hours for PERSEUS. Moreover, we are exploring other device-level metrics to enrich what PERSEUS can take as key inputs.

8 Related Work

Fail-slow failure study and diagnosis. As an emerging failure mode, fail-slow failure has received growing attention from academia and industry. Early literature mainly focuses on diagnosing fail-slow as an overlooked failure mode [16, 19, 24]. For example, Huang et al. define gray failure in the cloud with an abstract model [24]. Do et al. [16] measure system-level performance degradation brought by limpware, and address the necessity to develop limpware-tolerant systems. Gunawi et al. [19] perform qualitative analysis on 101 hardware-incident reports from various institutions and reveal the underlying fail-slow root causes in various types of hardware. Our motivational study in Section 2.3 specifically evaluates the fail-slow impact on drive performance (i.e., I/O latency). Moreover, our work provides a more diverse root cause analysis on fail-slow in storage devices.

Fail-slow failure detection. There have been a few studies addressing fail-slow detection and localization [4, 23, 29, 36, 45, 50]. For example, Panda et al. [36] convert software-level timeout signals into fail-slow metrics and adopt peer evaluation to detect fail-slow nodes. Huang et al. design Panorama to detect production failures by increasing the in-site observability [23]. Different from the above, PERSEUS detects fail-slow specifically in storage devices and is merely based on performance metrics like latency and throughput.

9 Conclusion

In this paper, we first share our unsuccessful attempts in developing robust and non-intrusive fail-slow detection for large-scale storage systems. We then introduce the design of PERSEUS, which utilizes classic machine learning techniques and scoring mechanisms to achieve effective fail-slow detection. Since deployment, PERSEUS has covered around 250K drives and successfully identified 304 fail-slow drives.

Acknowledgements

We would like to thank our shepherd Sangyeun Cho, and the anonymous reviewers for their insightful comments and suggestions. This research was supported by NSFC (62102424, 62072306), the Alibaba Innovation Research (AIR) program, National Key R&D Program of China (2022YFB4500302), Program of Hunan Postdoc Innovation (2021RC2069), and Program of Shanghai Academic Research Leader (20XD1402100). The authors would also like to thank Amber Bi and Shiming Wang for their feedbacks on early versions of this paper.

References

[1] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Computational Statistics*, 2010.

[2] Ramnatthan Alagappan, Aishwarya Ganesan, Yuvraj Patel, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Correlated crash vulnerabilities. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

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

[4] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[5] Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Garth R. Goodson, and Bianca Schroeder. An analysis of data corruption in the storage stack. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[6] 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 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[7] Lakshmi N. Bairavasundaram, Meenali Rungta, Nitin Agrawa, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Michael M. Swift. Analyzing the effects of disk-pointer corruption. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2008.

[8] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.

[9] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 1996.

[10] Wei Chen, S. Toueg, and M. Kawazoe Aguilera. On the quality of service of failure detectors. In *Proceedings of the 30th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2000.

[11] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, Jaehong Kim, Jaechun Park, Ki-Whan Song, Ki-Tae Park, Sangyeun Cho, Hwaseok Oh, Daniel D.G. Lee, Jin-Hyeok Choi, and Jaeheon Jeong. A flash memory controller for 15μs ultra-low-latency ssd using high-speed 3d nand flash with 3μs read time. In *Proceedings of the IEEE International Solid State Circuits Conference (ISSCC)*, 2018.

[12] David Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 2020.

[13] Brian Choi, Randal Burns, and Peng Huang. Understanding and dealing with hard faults in persistent memory systems. In *Proceedings of the 16th European Conference on Computer Systems (EuroSys)*, 2021.

[14] Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[15] GW Corder and DI Foreman. *Nonparametric Statistics: A Step-by-Step Approach.* Wiley, 2014.

[16] 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 Annual Symposium on Cloud Computing (SoCC)*, 2013.

[17] Norman R Draper and Harry Smith. *Applied Regression Analysis*. Wiley, 1998.

[18] 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.

[19] 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 Conference on File and Storage Technologies (FAST)*, 2018.

[20] Trinabh Gupta, Joshua B. Leners, Marcos K. Aguilera, and Michael Walfish. Improving availability in distributed systems with failure informers. In *Proceedings*

of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2013.

[21] Shujie Han, Patrick P. C. Lee, Fan Xu, Yi Liu, Cheng He, and Jiongzhou Liu. An in-depth study of correlated failures in production SSD-based data centers. In *Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST)*, 2021.

[22] 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 Conference on File and Storage Technologies (FAST)*, 2016.

[23] Peng Huang, Chuanxiong Guo, Jacob R. Lorch, Lidong Zhou, and Yingnong Dang. Capturing and enhancing in situ system observability for failure detection. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[24] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[25] Peng Huang, Xiao Ma, Dongcai Shen, and Yuanyuan Zhou. Performance regression testing target prioritization via performance risk analysis. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 2014.

[26] Volodymyr Kuznetsov, Vitaly Chipounov, and George Candea. Testing Closed-Source binary device drivers with DDT. In *Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC)*, 2010.

[27] Joshua B. Leners, Hao Wu, Wei-Lun Hung, Marcos K. Aguilera, and Michael Walfish. Detecting failures in distributed systems with the falcon spy network. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[28] Jiaxin Li, Yuxi Chen, Haopeng Liu, Shan Lu, Yiming Zhang, Haryadi S. Gunawi, Xiaohui Gu, Xicheng Lu, and Dongsheng Li. Pcatch: Automatically detecting performance cascading bugs in cloud systems. In *Proceedings of the 13th European Conference on Computer Systems (EuroSys)*, 2018.

[29] Chang Lou, Peng Huang, and Scott Smith. Understanding, detecting and localizing partial failures in large system software. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[30] Ruiming Lu, Erci Xu, Yiming Zhang, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Minglu Li, and Jiesheng Wu. NVMe SSD failures in the field: the Fail-Stop and the Fail-Slow. In *In Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022.

[31] Ao Ma, Fred Douglis, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. RAIDShield: Characterizing, monitoring, and proactively protecting against disk failures. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, 2015.

[32] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. A study of SSD reliability in large scale enterprise storage deployments. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST)*, 2020.

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

[34] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. In *Proceedings of the 2015 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2015.

[35] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *Proceedings of the 9th ACM International on Systems and Storage Conference (SYSTOR)*, 2016.

[36] 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 (USENIX ATC)*, 2019.

[37] Thanumalayan Sankaranarayana Pillai, Ramnatthan Alagappan, Lanyue Lu, Vijay Chidambaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Application crash consistency and performance with CCFS. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, 2017.

[38] Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Iron file systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[39] Matthew J. Renzelmann, Asim Kadav, and Michael M. Swift. SymDrive: Testing drivers without devices. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[40] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. Understanding latent sector errors and how to protect against them. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST)*, 2010.

[41] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash reliability in production: The expected and the unexpected. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*, 2016.

[42] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: A large-scale field study. In *Proceedings of the 2009 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2009.

[43] Erich Schubert, Jörg Sander, Martin Ester, Hans Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Transactions on Database Systems*, 2017.

[44] Riza O. Suminto, Cesar A. Stuardo, Alexandra Clark, Huan Ke, Tanakorn Leesatapornwongsa, Bo Fu, Daniar H. Kurniawan, Vincentius Martin, Maheswara Rao G. Uma, and Haryadi S. Gunawi. Pbse: A robust path-based speculative execution for degraded-network tail tolerance in data-parallel frameworks. In *Proceedings of the 8th ACM Symposium on Cloud Computing (SoCC)*, 2017.

[45] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. NetBouncer: Active device and link failure localization in data center networks. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[46] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[47] Benjamin Walker. Spdk: Building blocks for scalable, high performance storage applications. In *Storage Developer Conference*, 2016.

[48] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. Lessons and actions: What we learned from 10k ssd-related storage system failures. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC)*, 2019.

[49] Jinfeng Yang, Bingzhe Li, and David J. Lilja. Exploring performance characteristics of the optane 3d xpoint storage technology. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2020.

[50] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. Deepview: Virtual disk failure diagnosis and pattern detection for azure. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.