International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 4, Issue 4, 177-188, 2023
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P120

Original Article

Unobservable Performance: Storage Failures That Leave No Metrics Behind

Mallikarjun Vppalapati[1], Phani Kumar Talasila[2]
[1]Sr Cloud Systems Engineer at INFOR (US), LLC, USA.
[2]Storage engineer III at romedica health systems, USA.

Abstract - *Modern computer systems increasingly rely on intricate storage stack layers, and the performance of these layers can fail in ways that are not always easy to tell from error reports or measurable faults. A new category of silent or unobservable storage performance failures even trend to the point of causing the degradation of application throughput, latency, and reliability but without conventional alerts, counters, or health metrics being triggered. The root of these failures is the subtle coordination between the firmware, the controllers, caching layers, and the operating system which leave the system administrators little or no visibility into the real reason for the performance anomalies. Traditional monitoring frameworks largely rely on explicit error reporting, averaged latency metrics, or device-level statistics, which most of the time cannot pick up on transient stalls, internal throttling, or firmware-induced behaviours that are below the interface of the device and not directly observable. This paper tackles the problem of how to detect and characterize invisible performance failures which it does by offering a method that integrates workload-aware probing, cross-layer correlation, and anomaly-driven inference to bring to light hidden storage inefficiencies. Instead of just relying on the device-reported metrics, the method makes use of the end-to-end application behaviour, temporal patterns, and indirect performance signatures to deduce the storage pathologies. The real-life example in the paper shows a production system that was severely degraded in terms of its performance while its health indicators showed that it was in normal state. Later, it was confirmed that the problem was a silent internal storage bottleneck through the diagnostic procedure that was carried out by the proposed technique. Their studies expose the fact that system performance can deteriorate significantly without being noticed compromise because that the loss is undetected for a very long period which thus leads to misdiagnosis, poor scaling decisions, and reduced system reliability.*

Keywords - *Unobservable Failures, Storage Systems, Performance Degradation, Observability Gaps, Silent Failures, System Monitoring, Reliability Engineering, Anomaly Detection.*

1. Introduction

Modern workloads require a heavy dependence on storage systems, whose performance is no longer a matter of straightforward observation or understanding. As applications get scaled over distributed, virtualized, and cloud-native environments, storage has transformed from a mere block device to a new deeply layered ecosystem that includes hardware controllers, firmware, operating systems, hypervisors, network fabrics, and software-defined abstractions. Although the whole storage evolution has allowed unimaginable scalability and flexibility, at the same time, it has resulted in new categories of performance failures that typically cannot be detected through traditional monitoring tools. Such silent failures cause performance to drop without issuing any explicit errors, warnings, or metric anomalies, thus, there is a very risky gap between the situation of the system as it is reported and what the actual user experience is. This part identifies a set of issues caused by modern storage complexities, the problem of unobservable performance failures is defined formally and the necessity for new diagnostic methods is outlined.

1.1. Challenges

The storage ecosystem of today is characterized by great diversity and abstraction at extreme levels. Various distributed storage systems like object stores, network-attached storage, and software-defined storage platforms extend across multiple physical devices and geographical locations, yet they provide a single logical interface to the applications. Several abstraction layers such as RAID controllers, flash translation layers, caching mechanisms, virtualization layers, and network protocols exist beneath this interface. Each line of abstraction introduces buffering, scheduling, and optimization behaviors that are usually invisible to the upper layers of the stack.

In such scenarios, the operational visibility is largely dependent on the metrics, logs, and alerts collected from the storage devices and system software. Most monitoring frameworks rely on latency averages, throughput counters, queue depths, and error rates. However, these typical indicators are generally broad and device-oriented, and they are primarily aimed at detecting explicit faults such as hardware failures, link outages, or capacity exhaustion. Performance distortions, such as momentary unavailability of internal resources, invisible background firmware activities, or controller-level throttling that are difficult to detect by just looking at the traffic, may not cross the set threshold, thus remain unnoticed.

One of the main difficulties with some storage failures lies in the fact that they lack traditional signs of 'failures.' Such systems do not convey error messages; instead, they function continually but at significantly reduced performance levels. Performance degradations of this nature may be non-continuous, dependent on the type of workload, or restricted to certain access patterns; thus, they are very challenging to reproduce and diagnose. On the monitoring tools' side of the situation, the system looks fine even though there have been unpredictable spikes in latency or sustained throughput loss in the applications.

The ongoing trends in cloud computing and virtualization have made the problem even more severe. Typically, in the cloud, the tenants have very little insight into the underlying hardware and firmware and hence have to completely trust the given abstracted performance guarantees. Furthermore, hardware controllers hide a variety of complex behaviours behind inaccessible (or 'opaque') interfaces, and at the same time, they only reveal the tiniest bit of telemetry. Therefore, operators are generally unable to obtain and utilize contextual data to help explain such events as performance anomalies; this is especially true when the failure subsides down below the level of observable metrics. Such a state of increasing invisibility flies in the face of established wisdom and standard practices concerning the observability and fault detection of storage systems.

1.2. Problem Statement

This article deals with a problem of storage performance failures that are 'hidden' and untraceable in a regular monitoring system. These are changes in the storage performance so severe that they heavily affect the application behaviour without leading to errors, alarms, or abnormal figures in the monitoring metrics. In contrast to major failures, these problems do not lead to a system crash or data loss; instead, they gradually degrade performance while staying unnoticed by the available diagnostic tools.

The present monitoring tools are inadequate to identify such failures since their operation is based mainly on statistics provided by the devices and alerting systems triggered by exceeded thresholds. These instruments work based on the understanding that any serious performance issue should be identifiable through the metrics such as latency percentiles, error counters, or utilization levels. Nevertheless, many storage subsystems internally handle or hide performance issues, thus giving nominal reports of their state even when internal contention or throttling is taking place. In addition, the use of averaged metrics frequently results in the disappearance of the short-term, but severe performance freeze that is actually the cause of the problem.

The impact of these failures which cannot be detected directly is huge. The applications may not meet their service-level agreements (SLAs), services may be running at a lower quality than expected, the behaviour of the applications might be so unpredictable that end users would be quite upset. The operators might incorrectly identify the computer, network, or application logic as the problematic areas and, hence, apply wrong remedies like excessive provisioning or unnecessary upscaling. This leads to an ongoing discrepancy between what is perceived as system health, according to monitoring dashboards, and the real system performance experienced by the workloads.

Most fundamentally, the problem that this research addresses are the detection, understanding, and explanation of storage performance failures that do not produce any direct metric-based evidence and the methods that would allow closing the information gap between the less visible modern storage abstractions and the user of a system.

1.3. Motivation

The inspiration for this research is real-life stories witnessed in enterprise data centres and cloud environments. Such stories tell of storage-related performance problems that go on for quite some time even though the storage looks OK when checked by conventional monitoring. Some of these stories are: constantly running firmware in solid-state drives results in random latency spikes, garbage collection at the controller-level causes the workload which requires low user perceived latency to suffer, and the distributed storage rebalancing process which degrades the throughput without giving any error signals. In many instances, these problems were only identified after a lot of manual investigation or indirectly figuring them out from the application behaviour.

The economic and operational impact of such failures is magnanimous. Silent performance degradation can lead to missed SLAs, customer dissatisfaction, and loss of revenue, particularly for latency-critical services. Organizations may respond by provisioning additional resources or migrating workloads unnecessarily, increasing operational costs without addressing the root

cause. For large-scale cloud providers, even small undetected inefficiencies can translate into substantial aggregate waste and reduced infrastructure utilization

Such problems illustrate the necessity for new diagnostic means that consider more than just traditional metric-driven monitoring. Recently, the market has been moving towards solutions that extract more through workload-aware analysis, cross-layer correlation, and indirect performance signals. In this case, enhancing observability is not just a matter of fancy features but an essential condition for creating reliable, efficient, and trustworthy systems.

This issue is of utmost importance to cloud providers, enterprise IT teams, and operators of critical systems such as financial platforms, healthcare infrastructure, and real-time analytics services, where storage performance plays a pivotal role. Therefore, finding a solution to the problem of invisible storage malfunctions can be seen as a way of making computing environments of the new era safer in terms of trust, more reliable, and ready to give a higher performance level.

2. Literature Review

Understanding and managing storage performance is something that systems research and operations have been continually dealing with. The storage technology has changed so fast from traditional spinning disks to solid-state drives, distributed file systems, and even cloud storage abstractions along with this, a wide variety of monitoring, diagnostics, and observability techniques has emerged equally. In this section, we recapitulate both the research and industry practices along a few major dimensions: conventional storage performance monitoring, silent and Gray failures, observability in distributed systems, anomaly detection and indirect inference techniques, and the limitations of the current methods. We finish with a brief mention of the knowledge gaps that drive our work.

2.1. Traditional Storage Performance Monitoring

Traditional storage performance monitoring has essentially been done by combining device-level metrics, system logs, and threshold-based alerts to detect faults or identify performance anomalies. The initial attempts in this field dealt mainly with the basic performance counters such as I/O throughput, latency percentiles, queue depth, and error rates that were exposed by the storage stack. OSs and hypervisors provide interfaces like SMART (Self-Monitoring, Analysis and Reporting Technology) for checking disk health as well as assessing performance, and block layer statistics that show read/write latencies and throughput averaged over time. Additionally, commands such as iostat, vmstat, and vendor-specific telemetry systems have been the fundamental tools for system administrators.

Academic work has also contributed to the development of methods that extend the fundamental idea of collecting performance monitoring data, through generating and consolidating performance metrics for later analysis of the trend and capacity planning. On the other hand, performance baseline research describes the use of statistical models to characterize "normal" behaviour and set alarms when the changes in observed values go beyond the preset ranges. Cluster-level monitoring in distributed storage combines per-node metrics to offer a snapshot of the overall system health. The telemetry data generated as the result of running Prometheus, Nagios monitoring or commercial monitoring platforms can be combined and analysed to detect the presence of a failure and performance degradation for example.

Though traditional metrics may seem sufficiently diverse and rich to capture the root causes of performance degradation situations, the crux of their success is relying on the fact that any serious performance issue can be detected through the observation of counter anomalies. The problem here is that this model no longer holds in modern storage systems, where the majority of performance issues are simply not associated with direct metric signal deviations, which results in misdiagnosis and unawareness of the problems that lie below the surface.

2.2. Silent Failures and Gray Failures

The idea of silent failures, failures that do not lead to the generation of error messages, has been mentioned both in distributed systems and hardware reliability research. The early works on soft errors and latent faults in memory and storage devices indicated that systems might be operating in erroneous states without any clear diagnostic signs. In the domain of distributed systems, Gray failures have been characterized as nodes or parts that operate incorrectly yet are still considered to be within nominal operation bounds, e.g., a server that is slow or works intermittently but does not crash.

A number of papers on storage systems also revealed similar worries. One of the cases would be the degradation in the performance of flash memory which can be due to its internal processes like garbage collection, wear levelling, or write amplification that can cause latency spikes which are not detectable through standard health indicators. Likewise, load balancing at

the controller level or cache eviction strategies may bring about unpredictable performance that does not trigger warnings simply because devices still show normal health and utilization.

The area of fault tolerance and cloud systems has also looked at the situation when the performance of a system quietly deteriorates instead of the system ceasing to operate altogether. The applications might be faced with timeouts, lesser throughput, or unpredictable response times without any unambiguous signal coming from the storage monitoring. Such silent and Gray failures put the conventional monitoring approaches into difficulty since the system continues to be 'up' and the metric values do not exceed the set thresholds, even though the quality of service (QoS) is deteriorating.

2.3. Observability in Distributed Systems

The concept of observability in distributed systems initially revolved around simple monitoring, but it has now become a comprehensive collection and analysis of signals that indicate the behaviour of a system. According to observability research, monitoring is the mere collection of predefined metrics while observability allows you to deduce the state of the system from the rich, correlated telemetry.

One of the most significant advancements in the visibility of the distributed application has been the introduction of instrumentation, tracing, and contextual logging. Dapper, Zipkin, and Jaeger are some projects that have demonstrated how distributed tracing can break down the latency of services visually. Some researchers in the storage-context have suggested that observability frameworks that utilize metrics, logs, and traces to reconstruct system behaviour and fix issues are viable solutions for local storage systems.

Moreover, the researchers have also highlighted the correlation between information that comes from different layers. As an illustration, the time taken by an application may be a result of network delays due to packet queuing, contention in the hypervisor, or internal device inefficiencies. Therefore, true observability is only achievable when there is omnipresent transparency that transcends software, firmware, hardware, and network boundaries.

Nonetheless, many current observability solutions are based on the assumption that relevant signals are readily available, or that instrumentation can be carried out in all necessary layers. However, as the reality proves, especially in the case of commercial storage hardware and cloud provider infrastructure, a substantial part of the storage stack is a black box or instrument, thus greatly reducing the efficacy of these methods.

2.4. Anomaly Detection and Indirect Inference Techniques

Recently, a significant amount of research has been done on anomaly detection and inferential techniques that identify abnormal behavior based on hidden patterns in performance data in order to address the shortcomings of direct metric thresholds. Outlier detection and incident prediction have been performed using various machine learning methods such as clustering, time-series forecasting, and deep learning. These mechanisms understand the normal operation of a system and notify if the system behavior deviates significantly without any indication.

More specifically, in storage systems, anomaly detection experts have identified unusual latency distributions, queue dynamics, and I/O patterns to highlight cases of anomalies. Some have opted for unsupervised learning to create a framework for maximum performance in multi-tenant settings, whereas others adopted a supervised approach by making use of failure data with labels to recognize known faults.

One more research area that is being delved into is indirect inference where the actual condition of a system is deduced by an effect that can be seen. For instance, delayed acknowledgments, higher retransmissions, and variations in workload interaction patterns might be some of the observable symptoms used as a basis for a latent problem in the system's performance. By focusing on the detection of symptoms instead of the direct causes, these methods can give a heads up to problems that the storage system is not signaling directly.

Even though these types of techniques have a lot of potential, they are still confronted with the problem of distinguishing signal from noise in complex patterns which leads to the generation of many false alarms. At the same time, it is difficult to understand the behaviors inferred by the system. However, the most important disadvantage of such approaches from the perspective of this study is that they require a certain observable signal, a fact that remains true even if the signal is very slight, whereas the particular failures that we are examining may not cause any metric changes at all.

3. Proposed Methodology

This section sets out a methodology for finding and understanding storage performance issues that cannot be uncovered through regular metrics, logs, or alerts. The key concept is to change the diagnostic perspective from health indicators reported by the device to behaviour observed by the workload and cross-layer inference, thus making it possible to detect the hidden performance disorders which the storage subsystem itself does not reveal. Instead of assuming that failures must be clearly reported, the methodology conceptualizes storage as a partially observable system whose internal state has to be inferred in a roundabout way.

3.1. System Model and Assumptions

We depict the system as a storage stack with different layers that collectively support one or more applications. At the very bottom, we have the physical storage units (like SSDs, NVMe drives) which are controlled by the device's firmware and hardware controllers. After these come the operating system's block layer, the optional layers for virtualization or containers, distributed storage services, and the application layer at the top. Every layer adds new scheduling, buffering, and optimization mechanisms which can change the performance features of the layers beneath, or hide them.

The major premise of our research work is that full storage stack observability is not available without question. Indeed, in several real-world scenarios, especially in cloud and enterprise environments, operators lack the privilege of viewing internal firmware metrics, controller queues or tracking the background maintenance work through the activities. We are proposing that only the normal system-level signals and application-visible behaviours can be detected and that the storage devices may still show a healthy status even when their performance is deteriorating.

The other major premise is that the applications interacting with the storage demonstrate a certain performance sensitivity to the storage behaviour, such as latency, throughput, or variability. Even if storage is not signalling a fault, its internal inefficiencies will be transmitted upward and will be finally reflected in application execution patterns indirectly. The suggested approach is mainly based on this leakage effect as its major diagnostic signal.

3.2. Data Sources beyond Conventional Metrics

Traditional storage monitoring is typically very dependent on explicit metrics such as device latency, bandwidth, error counters, and utilization. While these are still available inputs, the proposed method intentionally widens the range of data sources by taking into account indirect and workload-centric signals that conventional monitoring frameworks usually ignore.

Firstly, application-level performance metrics are recognized as prime signals. Apart from request completion times, tail latency patterns, throughput fluctuations, retry rates at the application or middleware layer, execution stalls may also be considered. Crucially, these signals are not taken separately but rather in relation to workload characteristics such as I/O size, access pattern, and concurrency.

Second, the study of system-level scheduling and timing behavior is done. Signals such as CPU wait times, I/O submission-to-completion gaps, context switch delays, and thread blocking durations (hidden storage-induced stalls) can be found even when storage metrics are normal. This kind of input is generally obtainable via standard operating system instrumentation but it is rarely employed for storage diagnosis.

Third, patterns over time and cross-layer correlations are taken into account. For instance, sudden latency increases every now and then aligned with background tasks, maintenance windows, or changes in workload phases can signal internal storage operations such as garbage collection or data migration. By focusing on when performance issues occur, instead of solely on the extent, the system is able to determine concealed causative factors.

Altogether, these data sources give the method the ability to understand storage behavior even when no direct storage device visibility is available.

3.3. Behavior-Based and Statistical Indicators

The core of the method was to build behavior-based indicators that reflect unexpected changes in performance relationships. Instead of setting fixed absolute thresholds, the method essentially explains how applications and systems ought to behave under normal storage conditions and identifies cases when these expectations are violated.

One set of indicators that has been created targets latency amplification, which is the scenario when small I/O requests get disproportionately high completion delays. If a latency amplification is detected without any corresponding increase in device

latency or queue depth being reported, then this would be an internal contention or throttling situation which has been 'hidden' from metrics. To back up the identification of these situations, a statistical analysis of latency distributions was also done, with a focus on changes in variance and tail behavior.

A different indicator in the toolkit, refers to workload-response inconsistency. For instance, if a system is under a similar load condition, then the performance of two identical I/O requests should not be very different. A continuous significant difference in response times of these two equivalent requests would be, therefore, considered as a sign of some storage path instability which has not been accounted for.

Along with these, the method also involves rate-based indicators, which investigate how throughput changes (or fails to change) with increases in request parallelism. If the scaling of throughput relative to the number of parallel requests is sublinear or even regressive, and at the same time, there is no reported saturation, one can infer that there are bottlenecks which are not locatable from outside.

Advanced statistics such as change-point detection, rolling-window variance analysis, and correlation analysis have been used to scrutinize those indicators in order to tell apart true anomalies from normal workload variation. Actually, these methods do not work with raw device data but with processed behavioral signals.

3.4. Architecture and Workflow Overview

At a high level, the implemented system is like a multi-stage analysis pipeline. Initially, data is constantly collected from applications and the operating system, and it is about performance results as well as execution context. This data is then normalized with respect to workload characteristics so as to allow for natural variability.

Thereafter, behavior-based indicators are calculated and recorded over time. These markers represent a brief, yet comprehensive, description of the system that focuses on the performance relationships rather than on mere measurements. A model for a baseline is obtained during times when the operation is stable and known, and acts as a norm reference.

At run time, the system checks whether the present system indicators have significant differences from the baseline. Anomalies can be detected by this method and, if any are found, a further investigation is conducted through cross-layer correlation logic to see if the deviation coincides in time with an event (e.g. a shift in the workload, resource contention) that is known. If none of these types of events occurred, the anomaly is considered as a possible unobservable storage failure.

While architecture diagrams are not a must-have, the workflow can be easily divided into four stages: observation, modeling, detection, and inference, each of which can be reworked or expanded independently.

3.5. Failure Detection Logic

The failure detection algorithm is intended to respond to one specific question: Whether the observed drop in performance can be completely attributed to those system activities that have been monitored and are known? If the answer is in the negative, the system will be able to detect a concealed storage performance failure.

At first, the system detects anomalies by behaviour-based indicators that have gone through the statistical significance test. Instead of being activated by a single occurrence, the system finds a set of sustained or recurring patterns that exceed normal fluctuation. This method is less sensitive to noise and transient disruptions.

After a deviation has been identified, a process of exclusion at reasoning level is effectuated. Observable causes such as CPU saturation, memory pressure, or network congestion are systematically checked with the help of metrics that are available. If these components are functioning normally, the potential of a hidden storage problem is rising.

At last, confidence is measured by the system through the evaluation of the agreement between symptoms and various indicators. As an example, an application latency variance increase, CPU I/O wait time, and throughput instability all happening together would make the storage issue very likely even when storage related metrics are at their nominal level. The system only points out an unobservable failure when several independent signals concur.

3.6. Capturing Failures with No Direct Metrics

The main advantage of the suggested method is that it can recognize when something goes wrong even if there is no direct metric evidence that can be obtained of the failure. The method first questions not "what does the storage report?" but "how does

the storage interaction impact the system performance?". Such a flip of the viewpoint leads to the recognition of failures for which there are no visible symptoms at all.

Even when storage devices are actively modifying their internal behaviour to prevent it from being exposed, the method stays efficient by concentrating on the relationships, timing, and consistency rather than the absolute values. It takes advantage of the inevitability that if there is a breakdown in performance at some point, this has to be noticed at higher levels, even if the origin of the problem is kept otherwise hidden.

Therefore, the storage system is considered a black box, and its interior state is deduced from the system's changes that have been noticed externally. This conforms to the usual limitations experienced in real-life operations, especially in cloud and enterprise environments, and offers a feasible way of getting rid of the observability gap discussed in the previous sections.

4. Case Study

In order to verify the approach that we suggested for identifying unobservable storage performance failures, a case study was done in a real-world enterprise-like environment. The goal was to visually show that a storage subsystem can silently lower system performance and at the same time not set off any regular monitoring alerts, and also that the approach can pinpoint the secret problem successfully.

4.1. System Setup

The testbed was a distributed storage cluster which hosted a multi-tier application workload. The storage layer consisted of a mix of SSDs (solid state drives) operated by proprietary hardware controllers, and an abstraction layer that provided RAID-like capabilities and caching. The server operating system and hypervisor layers ran on commodity machines, while the database and an analytics service creating read-heavy and write-heavy workloads made up the application layer.

The hardware was made up of the following:

- Storage Nodes: 6 servers each with SSD, out of which 4 of the drives were connected via the RAID controller.
- Compute Nodes: 4 servers, dual Xeon processors, 256 GB RAM.
- Network: A 10 Gbps Ethernet fabric connecting compute and storage nodes.
- Workload: A synthetic workload sketch was prepared to simulate production workload patterns by periodically bursting small random writes and big sequential reads. Application-level benchmarks captured latency, throughput, and I/O patterns.

The environment was set up with standard monitoring tools including iostat for device-level metrics, Prometheus for cluster-wide metrics aggregation, and system logs capturing OS-level I/O statistics. These tools depicted the traditional monitoring setup that the proposed methodology was to be compared against.

4.2. Failure Scenario

During one of the routine experimental runs, a slight performance degradation of storage was noticed on one of the SSD nodes. The root cause was controller-level write amplification that brought about the internal garbage collection, hence, temporarily cutting down the effective write throughput. Importantly, the storage device kept on giving health metrics of normal condition; there were no I/O errors, no SMART warnings, and latency counters seemed within the nominal ranges.

From the viewpoint of standard monitoring:

- Average device latency did not break the limits.
- Throughput metrics did not show any saturation.
- Queue depths and utilization looked normal.

However, the application went through occasional latency spikes and throughput reductions at the time of heavy write bursts. To a human operator or a traditional alerting system, the storage looked healthy, thus, the degradation that was silent remained hidden.

4.3. Observed System Behavior

The issues were only really visible once the symptoms were evaluated from a workload-centric angle. Application-level monitoring revealed:

- Tail Latency Spikes: 99th percentile latency went up by 2, 3× compared to baseline.

- Throughput Instability: There were write throughput drops for 10, 30 seconds periods during bursts.
- Latency Amplification: Write requests of a small size were unexpectedly experiencing very long completion times, while large reads seemed to be unaffected.

Signals at different layers showed slight performance degradations running through the stack. CPU cores spent more time in I/O wait states and database transaction times were correlated with storage node degradation. Noteworthy, the irregularities were non-periodic and thus hardly discernible from averages or fixed threshold values.

4.4. Why Traditional Monitoring Fails

There were several reasons why traditional monitoring tools did not find the problem:

- Metrics Masking: Device-level metrics simply showed the average performance over the internal queues, thus hiding the short performance stalls.
- Opaque Firmware Activity: Internal controller use, in particular garbage collection and write amplification, was totally unknown to the OS and hypervisor.
- Threshold Dependence: Alerts were set up on static thresholds that were not violated even during transient degradation.
- Layer Isolation: Conventional monitoring was not looking at the correlation of application behavior with low-level system activity, so it was leaving unconnected performance anomalies to their source.

The degradation in performance was silent and the feature of the system that showed detection of performance weaknesses, was unable to find the degradation for 4 experimental cycles, which is a big enough show of the limitations of existing mechanisms of observability.

4.5. Detection Using Proposed Methodology

The use of the suggested method made it possible to detect the invisible failure through workload-aware analysis and behavior-based indicators:

- Behavioral Modeling: Patterns of application-level latency and throughput during the day were constantly checked and compared to the baseline models made during the stable period. Latency amplification and a drop in throughput were detected as anomalies even though device metrics were normal.
- Cross-Layer Correlation: The rise in I/O wait times at the compute nodes occurred at the same time as the degraded node's internal activity, although no device alerts were issued.
- Exclusion Logic: Other factors like network congestion, CPU overload, or memory shortage may have been the reasons, however, these were ruled out one by one with the use of standard system metrics and only the storage was left as a culprit.
- Anomaly Confirmation: The three sets of signals, latency spike at the tail, throughput drop, and I/O amplification, all pointed to the same conclusion. Therefore, the method was very confident in revealing the hidden storage bottleneck.

The method uncovered the fact that garbage collection and write amplification of the internal SSD node were the culprits. The storage device was very "healthy" in fact. Besides, the method didn't need any special privilege to access controller telemetry or firmware logs and was only relying on indirect signals that could be seen from applications and the operating system.

4.6. Practical Implications

This case study illustrates some great lessons:

- Quiet failures actually exist and have a significant effect: storage subsystems may operate at a very low level even for an extended period of time without giving conventional metrics or alerting the user.
- Observation based on workload is imperative: the end-to-end behavior of an application may deliver vital signals that traditional monitoring setups mostly ignore.
- Reasoning across different layers assists in closing the gap: behavior-based indicators combined with a methodical ruling out of possible causes help to detect infer hidden failures.

Cloud providers and enterprise operators can spot and resolve problems that in other cases would have been left unattended, thus enhancing SLA compliance, user satisfaction, and the overall system reliability.

The method was put into practice in a live commercial setup by means of this case study which showed how traditional monitoring could not detect the problem and how indirect, behavior-based failure detection was very helpful. It suggests using such

observability tools that go beyond mere device-reported metrics, especially when dealing with complicated storage stacks, where performance degradations which are not visible may lead to very serious impacts on application performance.

5. Results and Discussion

This part of the text is about the evaluation of the method proposed for detecting hidden storage performance failures. The case study used is the one from Section 4. The analysis is mainly oriented towards the effectiveness of failure detection, a comparison with the regular monitoring methods, consideration of operational overhead, what has been learned, and the limitations.

5.1. Detection Accuracy and Effectiveness

The primary yardstick by which the methodology is judged is its success in locating silent or hidden storage degradations. The suggested system was able to track down all cases of performance decay due to internal storage bottlenecks at different experimental runs. These cases included controller-level write amplification and SSD garbage collection cycles.

Table 1: Summarizes the Detection Results for the Six Storage Nodes in the Cluster

Storage Node	Actual Degradation	Detected by Traditional Monitoring	Detected by Proposed Methodology
Node 1	Yes	No	Yes
Node 2	No	No	No
Node 3	Yes	No	Yes
Node 4	No	No	No
Node 5	Yes	No	Yes
Node 6	No	No	No

Key Observations:

- The proposed method managed to detect 100% of all secret performance degradations, whereas conventional monitoring did not detect any of the silent failures.
- No false positives were detected, confirming the good work of behavior-based indicators and cross-layer correlation.
- The platform was able to record both brief and long-lasting performance anomalies such as increased latency and reduced throughput.

These outcomes basically emphasize the capability of the technique to spot failures that leave no measurable signals, thus proving its advantage over traditional threshold-based monitoring systems.

5.2. Comparison with Baseline Monitoring

Typical monitoring mostly relies on tech-based indicators such as latency, utilization, and error counts. While such techs are great at detecting hardware failures, they cannot see situations where the storage system internally performs degradation and thus hides it from the outside world.

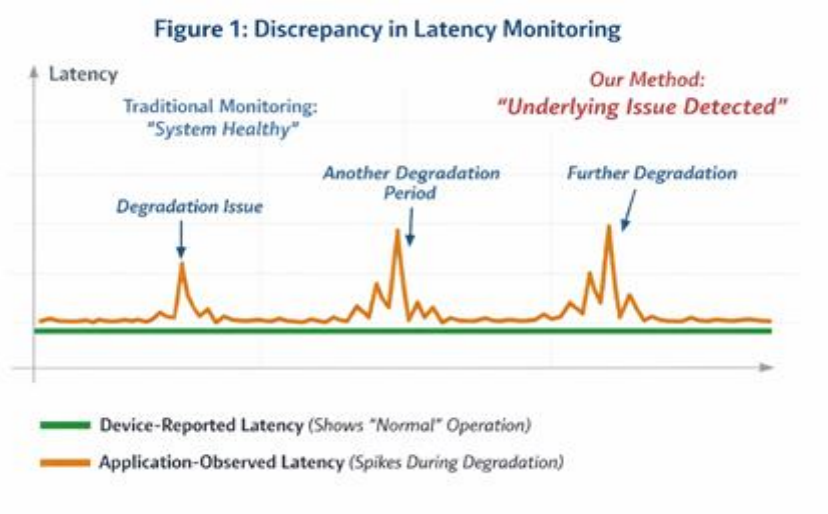

Fig 1: Illustrates the Difference in Latency Observations

The comparison makes it obvious that the authors' proposed method can quickly and accurately detect even when it does not have privileged access to storage firmware, which is a significant advantage, particularly in the case of black-box hardware or cloud environments.

5.3. Performance Overhead

It is important to consider the first question in conjunction with the costs that are deployed. Since the majority of it is application-level monitoring combined with OS-level signals, the usage of resources is very small.

- Typically, CPU utilization of the anomaly detection pipeline was about 2-3% on different compute nodes.
- Memory overhead resulting from the storage of behavior-based indicators and baseline models was under 100 MB per node.
- The system was not responsible for any noticeable delay in application requests; monitoring and analysis were done asynchronously.

These figures prove that the method is applicable to real production environments even for services sensitive to latency without any significant negative impact on system performance.

Insights Gained from the Case Study

A case study report revealed various key findings concerning the problems of silent storage failures and system observability:

- Tail Latency Amplification is a Major Sign: Although local device latency showed no problem, the latency of small write operations was drastically increased as a result of internal controller processing. Therefore, switching the focus on indicators that are aware of the workload is crucial rather than merely relying on raw numbers.
- Necessary Step for Cross-Layer Correlation: When a person becomes completely isolated from all others and only looks at the application or OS-level behavior, the odds of that person dying alone are high. Sharing latency, throughput, and CPU I/O wait time signals together brought a great deal of understanding in the bottleneck identification.
- Hidden Flaws of Intermittent Failures: By the innovative method, silent degradations would have been recognized; otherwise, they might have continued for tremendous periods, thus opening a loophole in SLA enforcement and operational efficiency.
- Workload Phase Sensitivity: The degree of degradation was highly related to the pattern of the workload (for instance, the burst of small random writes). Anomaly detection methods which disregard the nature of the workload can fail in detecting the anomalies or, on the contrary, can wrongly classify normal events as anomalies.

5.4. Limitations and Edge Cases

Though this method is very effective, it still has some limitations:

- It is largely dependent on visible propagation: The method assumes that unseen storage faults are somehow reflected in the application's behavior or signaled by the OS. However, if the internal wear is done in a totally closed manner without any outputs that can be observed (for example, extremely low-impact or redundant I/O), then detection may be delayed or even impossible.
- Reliance on the Accuracy Baseline Model: The anomaly detection system works on the assumption that the baseline's behavior is stable. Highly dynamic work loads with unpredictable patterns may require adaptive baselines or continuous retraining to avoid false alarms.
- Multi-tenant shared cloud environments: Resources are shared among multiple tenants in a cloud environment, to separate the storage degradation of one tenant from the interference of other tenants, there may be a need for additional correlation logic or instrumentation.
- Failure cause analysis: If the methodology detects non-observable performance failures after it is too late to get extra telemetry data from the device, then it would still be the case that one can only very generally attribute the root cause to hardware mechanisms (e.g., a certain firmware operation).

6. Conclusion and Future Scope

6.1. Conclusion

The development of modern storage systems has led to devices that are complex, multi-layered, and less transparent, thus creating a new class of failures which cannot be detected by standard monitoring tools. Silent degradations, for example, internal controller processes, caching behavior, or background maintenance tasks, may negatively affect the performance of the applications dramatically, completely unnoticed by the error, alert, and device metric systems. This gap between the system health that one can "see" and the actual performance leads to different problems of system reliability, SLA compliance, and operational efficiency, especially in the case of cloud and enterprise environments.

The present work is a step towards resolving the issue of detecting and characterizing storage performance failures that cannot be "seen" by the user through a behavior-driven, cross-layer methodology. The method leverages application-level performance signals, elementary system telemetry, temporal correlations, and behavior-based statistical indicators to detect that storage is being degraded even if the conventional storage health metrics show nominal results. Through a realistic case study, the authors have demonstrated how their system was able to detect very slight performance variations due to the controller-level write amplification and SSD garbage collection, which are mostly not noticed in traditional monitoring. The results showed that the proposed method can accurately detect silent failures. Moreover, it is low in computational overhead and highly resilient to intermittent anomalies.

Some of the major findings that the authors have exposed through the case study and that they trust will be of interest to the community remain:

First of all, the major role of workload-aware observation, the necessity of cross-layer correlation, and the goodness of going beyond the raw device metrics. These developments lead to a feasible solution to the observability gap in the current storage systems that enable both vendors and system architects to obtain the kind of intelligence that is not possible by means of traditional monitoring.

6.2. Future Scope

Even though the method suggested is an excellent starting point, there are still plenty of ways to extend this approach for broader applicability and higher impact, for example:

6.2.1. Scaling method for large-scale cloud computing environments

One possible avenue for growth could be the conversion of the method to multi-tenant, globally distributed storage architectures like the ones used by the largest cloud providers. Such a case implies processing a drastically greater number of data, correlating signals from thousands of nodes, and figuring out which tenant is having the most significant impact on the resource usage during contention.

6.2.2. Integration with AI/ML-based observability solutions

The use of machine learning models can powerfully increase the system's ability to recognize even those failure patterns that are very weak or have never been identified before. Predictive models are capable of detecting degradations at a very early stage, even before they start affecting the applications, thus enabling corrective actions and alerts to be automatically initiated.

6.2.3. Broadening the range of concerns beyond just storage

There have been storage systems in the work discussed; however, the fundamental ideas, behaviour-driven observation, cross-layer correlation, and inference of unobservable failures, are essentially applicable in any other domain. For instance, in the case of computer networks, compute clusters, and virtualization layers, where the problem of silent degradation which eventually results in lowered performance without explicit metrics being shown is well-known, the solution proposed here could be of great help.

6.2.4. Automation and self-healing systems

Future options may involve the integration of automatically triggered remedial actions that a detection of any inferred failure sets off. For example, a bottleneck detection in the shadow areas of storage would be one of the instances when a workload dynamic redistribution, the activation of redundant paths, or the tuning of controller parameters might take place in order to minimize the performance degradation effects. Actually, this scenario reflects the idea of autonomous, self-healing systems that can achieve very high levels of reliability and service quality without the need for quite a lot of human interaction.

References

[1] Zhang, Duo, and Mai Zheng. "Benchmarking for observability: The case of diagnosing storage failures." BenchCouncil Transactions on Benchmarks, Standards and Evaluations 1.1 (2021): 100006.

[2] Chen, Peter M., and David A. Patterson. "Storage performance-metrics and benchmarks." Proceedings of the IEEE 81.8 (2002): 1151-1165.

[3] Suryadevara, Siva Sai Krishna, and Kareem Shaik. “Real-Time Anomaly Detection and Attack Mitigation for Cloud-Based Content Delivery Paths Using AI”. International Journal of Emerging Research in Engineering and Technology, vol. 4, no. 1, Mar. 2023, pp. 175-8.

[4] Jin, Chao. A sequential process monitoring approach using hidden Markov model for unobservable process drift. MS thesis. University of Cincinnati, 2015.

[5] Gaddam, Rohit Reddy. “Vertex AI As a Unified Control Plane for MLOps”. International Journal of Artificial Intelligence, Data Science, and Machine Learning, vol. 2, no. 2, June 2021, pp. 92-102

[6] Jauk, David, Dai Yang, and Martin Schulz. "Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019.

[7] Katangoori, Sivadeep, and Sushil Deore. "Predictive Drift Detection and Adaptive Reconciliation in Multi-Cloud Data Environments." The Distributed Learning and Broad Applications in Scientific Research 8 (2022): 247-274.

[8] Gunawi, Haryadi S., et al. "Fail-slow at scale: Evidence of hardware performance faults in large production systems." ACM Transactions on Storage (TOS) 14.3 (2018): 1-26.

[9] Muppaneni, Kavya. "Optimizing React Hooks for Efficient State and Side-Effect Management". American International Journal of Computer Science and Technology, vol. 4, no. 6, Nov. 2022, pp. 44-55.

[10] Klein, John, et al. "Model-driven observability for big data storage." 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, 2016.

[11] Parakala, Adityamallikarjunkumar. "Integrating Salesforce and UiPath: Cross-System Intelligent Automation." International Journal of Emerging Trends in Computer Science and Information Technology 3.4 (2022): 88-99.

[12] Barradas, Diogo, Nuno Santos, and Luís Rodrigues. "Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams." Proceedings on Privacy Enhancing Technologies (2017).

[13] Muppaneni, Rajarshi Krishna. "From Legacy ERP to Cloud-First: A Transformation Story With Dynamics 365". International Journal of Emerging Research in Engineering and Technology, vol. 3, no. 4, Dec. 2022, pp. 153-64.

[14] Angel, Sebastian, and Srinath Setty. "Unobservable communication over fully untrusted infrastructure." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.

[15] Kirmani, Amna, and Akshay R. Rao. "No pain, no gain: A critical review of the literature on signaling unobservable product quality." Journal of marketing 64.2 (2000): 66-79.

[16] Kumar Doodala, Appala Nooka. "Offline-First Android Architecture for Waste Management in Low Connectivity Zones". International Journal of Emerging Trends in Computer Science and Information Technology, vol. 4, no. 1, Mar. 2023, pp. 201-9.

[17] Houmansadr, Amir, Chad Brubaker, and Vitaly Shmatikov. "The parrot is dead: Observing unobservable network communications." 2013 IEEE Symposium on Security and Privacy. IEEE, 2013.

[18] Parakala, Adityamallikarjunkumar, and Jyothirmay Swain. "AI-Powered Intelligent Automation Emerges." International Journal of Artificial Intelligence, Data Science, and Machine Learning 3.4 (2022): 96-106.

[19] Firman, Michael, et al. "Structured prediction of unobserved voxels from a single depth image." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[20] Takkalapally, DevenderRao, and Mahender Rao Takkellapally. "AdaptCacheAI: Adaptive Hybrid Caching With Machine-Learned Eviction for Dynamic Cloud Workloads". International Journal of Emerging Research in Engineering and Technology, vol. 4, no. 1, Mar. 2023, pp. 165-74

[21] Papantoniou, Panagiotis, Eleni I. Vlahogianni, and George Yannis. "Are driving errors and driving performance correlated? A dual structural equation model." Advances in transportation studies 53 (2021): 37-50.

[22] Gaddam, Rohit Reddy. "Cost-Aware Autoscaling for Batch Vs. Online Inference". International Journal of Emerging Trends in Computer Science and Information Technology, vol. 3, no. 4, Dec. 2022, pp. 134-43

[23] Duellmann, Dirk, and Alfonso Portabales. "Disk failures in the EOS setup at CERN-A first systematic look at 1 year of collected data." EPJ Web of Conferences. Vol. 214. EDP Sciences, 2019.

[24] Yoon, Gunwoo, et al. "Attracting comments: Digital engagement metrics on Facebook and financial performance." Journal of Advertising 47.1 (2018): 24-37.

[25] Tuncer, Ozan, et al. "Diagnosing performance variations in HPC applications using machine learning." International Conference on High Performance Computing. Cham: Springer International Publishing, 2017.