International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 2, Issue 2, 107-116, 2021
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V2I2P113

Original Article

The Geometry of Redundancy: Why Physically Separate Storage Paths Behave as One System

Mallikarjun Vppalapati
Sr Storage Engineer at Vsion Technologies, USA.

Abstract - *Nowadays,storage systems are, in most cases, based on the redundancy concept so that physically separate paths, controllers, and devices are considered to produce fault isolation and independent behavior by default. However, in reality, such systems often have failure patterns that are tightly coupled, performance bottlenecks, and recovery behaviors contradicting the physical separation that is visible. This paper reveals that there is a large distance between the storage design perception of redundancy and the systemic behavior witnessed in real-life environments. Therefore, it is argued that redundancy is more of a geometry than a function in most cases. We discuss geometric redundancy, meaning that components are placed in different physical or logical paths, but they are still limited by a common control plane, synchronization mechanisms, queuing dynamics, or workload characteristics, and hence, they behave like one system when being stressed. Though there have been plenty of observations of cascading failures and correlated performance degradation, few have formally analyzed why physically independent paths in storage fail together at the end of the day. In this paper, a method is described that involves architectural decomposition, failure-mode mapping, and workload-driven stress analysis to locate those coupling points that are hidden between different layers of the storage system. The study extends to a real-life example of an enterprise storage environment with dual-fabric connectivity and redundant controllers, where empirical data showed that latency spikes were synchronized, failover was coordinated, and recovery thresholds were shared even though the physical separation was complete. The main points indicate that the efficiency of redundancy depends more on control symmetry, feedback loops, and shared operational geometry than on the physical topology of the system.*

Keywords - *Redundant Storage Systems, Fault Domains, Storage Architecture, System Coupling, High Availability, Failure Correlation, Resilience Engineering.*

1. Introduction

1.1. Challenges in Redundant Storage Architectures

Redundancy has been a key principle of storage system design for a very long time. By adding completely separate physical components, such as two controllers, fabrics, network paths, and storage device copies, system designers intend to remove single points of failure and provide high availability. The usual logic is simple: if components are physically separated, then the failure of one should not affect the other. This mode of thinking, inherited from early fault-tolerant systems, still dictates how storage architectures are documented, certified, and procured.

But the reality is that today's storage environments have gone way beyond these simple assumptions. Modern storage stacks are very complicated, covering hardware, firmware, operating systems, hypervisors, container platforms, orchestration layers & observability tooling. Even though physical separation may be at the level of cabling or devices, logical & operational dependencies usually cross these layers. Shared metadata services, centralized configuration stores, common scheduling algorithms, and global state synchronization mechanisms cause independent paths to be coupled in a subtle or powerful way.

A large part of this coupling is due to shared control planes. Storage controllers may be coordinated in active-active or active-passive configurations, with joint decision-making, heartbeat protocols, and synchronized caches. Firmware versions, bug conditions, and recovery logic are usually uniform across redundant components, which means that the probability of simultaneous or cascading failures is increased. In a similar fashion, redundant network fabrics may share upstream switches, routing policies, congestion control behaviors, or timing dependencies that will result in correlated performance degradation when put under load.

All these interdependencies make failure predictions and performance modeling more difficult. Traditional reliability models are based on the assumption that redundant components are independent and therefore, availability can be calculated with simple probability rules. In reality, components are seldom independent. Latency spikes, throughput collapses, or control-plane stalls often occur simultaneously across multiple paths when subjected to stress conditions such as peak workloads, partial outages, or

recovery events. Thus, there is a disconnect between what the architects and operators can explain and the actual situation when systems that are "fully redundant on paper" behave like a single fragile entity during real incidents.

So, the problem is not a lack of redundancy; the challenge is that the concept of redundancy does not align with its actual manifestations. Recognizing this mismatch involves looking beyond physical diagrams and into the behavioral geometry of storage systems.

1.2. Problem Statement

Organizations, despite spending large amounts on redundant storage architectures, are still facing situations where failures happen and the benefits of physical separation (isolation) are not realized. Multiple-fabric, controller, and data path systems, which are usually multi-layered to avoid a single point of failure, often show the same failure modes, performance degradation in lockstep, or recovery in a coordinated manner. These findings demonstrate that there is a significant mismatch problem between the theoretical models used to design redundant systems and the operational reality of production environments.

This can be explained by the fact that at the root of the problem lies the mistaken belief that physical isolation guarantees independent failures. Typical fault-domain definitions are historically based on physical tangible boundaries, e.g., a controller, disk shelf, network link, or data center. While these boundaries provide some level of abstraction, they are missing the shared logical constructs: control logic, timing dependencies, configuration state, and workload feedback loops. Therefore, a number of "highly redundant" architectures, which is most of them, are actually susceptible to correlated failure when the fault domain is broader than assumed.

Storage paths that are physically separate, but decided by the same processes will, most likely, behave like a single system. For example, load-balancing algorithms may respond - in the exact same way - to congestion signals. Failover mechanisms may be activated simultaneously because of shared thresholds or timeouts. Firmware bugs at the controller level may become apparent at all the controllers. With the observability and alerting systems, there is even a risk that coupling be reinforced by driving operations and response synchronization. These commonly shared behaviors essentially result in multiple paths being merged into a single behavioral unit, more so when the system is under stress or is out of its normal state.

1.3. Motivation and Research Objectives

Organizations continue to experience failure situations despite the fact that they spend huge sums on redundant storage architectures. Besides that, a failure happens, and the benefits of physical separation (isolation) are not realized. Multiple-fabric, controller, and data path systems, which are often multi-layered to prevent a single point of failure, frequently exhibit the same failure modes and performance degradation in lockstep or recovery in a coordinated manner. These findings reveal that the theoretical models used to design redundant systems are completely out of line with the operational reality of production environments.

This can be explained by the fact that a core of the problem is that people wrongly believe that physical isolation automatically guarantees independent failures. Typical fault-domain definitions are historically based on the physical tangible boundaries, e.g., a controller, disk shelf, network link, or data center. Although these boundaries give a certain level of abstraction, they lack the shared logical constructs: control logic, timing dependencies, configuration state, and workload feedback loops. Hence, a number of "highly redundant" architectures, which are most of them, turn out to be vulnerable to correlated failure when the fault domain is broader than assumed.

Existing architectural frameworks and vendor documentation, for the most part, are silent about these shifts. Fault trees and availability calculations leave out control-plane coupling, take uniform recovery behavior for granted, and see performance degradation as a localized and not systemic problem. Thus, a false sense of security is being created, where redundancy exists in structure but not in function.

The problem has been identified as one in which physically redundant storage architectures collapse into unified systems of failure or degradation over and over again without a framework for explaining it. Without such a framework, organizations will always be reactive rather than proactive incident-wise. They will not be able to devise architectures that will most likely limit the blast radius and failure correlation.

2. Literature Review

There has been extensive research on redundancy and fault tolerance in computer systems. The first models strictly depended on the assumption that physical duplication of a device can isolate its failure. Classical fault- tolerance theory was mostly

hardware-centered, assuming independent failures of redundant components like a power supply, a processor, or a disk. The presence of N-modular redundancy, fail-stop processors, and reliability block diagrams made availability a mathematical property of independent components. Although these methods were capable of solid mathematical justification, they depended heavily on making failure isolation assumptions that have become hard to keep nowadays, especially in modern storage systems.

When storage systems were only made of direct-attached disks and started to be networked, the focus of research gradually shifted towards Storage Area Networks (SANs) and multipath I/O frameworks. Writing SAN research path redundancy was the main focus: multiple host bus adapters, fabrics, switches, and storage controllers - any combination of these components would fail, but data access should never fail. Multipath I/O (MPIO) methods were developed to hide the complexity of path operations, load distribution, and fault recovery from the higher levels of the system, thus supporting applications without knowing whether a path is good or not, with the underlying assumption that multiple paths will always guarantee resilience. Nevertheless, most of these studies have concentrated on correctness and availability in cases of isolated failures, such as a one-link or one-controller failure, rather than discussing correlated performance problems or behavioral convergence among different paths.

Later research articles dealt with performance improvements in multipath environments, including adaptive load balancing, queue-depth tuning, and path selection algorithms. These studies showed that the path's behavior depends not only on its physical features but also on the sharing of scheduling logic and feedback mechanisms. Although such studies recognized inter-path interactions, they usually considered them as solving optimization problems rather than fundamental problems that threaten redundancy. Hence, the presumption of independence was largely retained.

The notion of failure correlation has been more explicitly delved into in reliability engineering and research on large-scale distributed systems. In the case of common-mode failures multiple components failing because of one shared cause power systems, avionics, and cloud infrastructure domains have all recorded such instances. Studies on computing systems have proven that software bugs, configuration errors, environmental conditions, and operator mistakes can all be factors that trigger simultaneous failures of redundant components. The analyses of large-scale outages conducted by the cloud providers have confirmed that correlated failures are very common, especially those that are caused by sharing the same control software or operational processes.

In spite of this acknowledgment, a large part of the failure correlation literature considers common-mode failures only as rare events and not as structural properties of a system design. In storage research, correlated failures are frequently linked to external factors such as a firmware defect or a misconfiguration rather than to the very layout of the architecture. Such a perspective hinders one's capacity to use correlated behavior as an expected consequence of design decisions.

For example, research in software-defined networking (SDN) highlights the use of a centralized control with a distributed data path, while the authors of the paper still highlight the danger of control-plane failures. At the same time, the analogy between control-data plane separation and storage control and data planes has been drawn to describe the former in terms of managing metadata, user-requested policy enforcement (control), and actual block IO (data).

Nevertheless, works that go in-depth to analyze the resilience of the control plane in storage systems rarely discuss the behavioral coupling aspect but rather the availability and correctness aspects. Adding controllers and distributing metadata services are seen as good measures to increase fault tolerance but these additions alongside the synchrony requirement may result in shared decision-making and timing dependencies. The articles generally take it for granted that a triple-mated logic element box (redundant control plane) will guarantee the robustness of the entire system to one component failure, thus masking the vulnerability coming from the fact that identical logic and shared state can cause physically separate units to coordinate their actions.

Table 1: Summary of Literature Related to Redundancy, Fault Tolerance, and System Coupling

Authors / Year	Primary Focus	Key Contribution	Relevance to This Study
Pinheiro et al., 2006	Storage redundancy & energy efficiency	Shows redundancy can be exploited beyond fault tolerance, introducing systemic trade-offs	Highlights that redundancy has behavioral consequences, not just structural benefits
Klein & Keller, 2009	Secure storage architectures	Integrates redundancy with integrity and encryption	Treats redundancy as structural; does not address behavioral coupling
Orenstein, 1989	Redundancy in spatial databases	Early conceptual treatment of redundancy beyond hardware	Supports abstraction of redundancy beyond physical components

Gibson, 1991	RAID architectures	Formalizes disk-level redundancy and fault isolation	Foundational but assumes independent failure domains
Sheaffer et al., 2007	Hardware redundancy in GPUs	Demonstrates coordinated recovery mechanisms	Illustrates common-mode behavior despite redundancy
Ivaldi et al., 1988	Kinematic redundancy (biological systems)	Introduces redundancy as a geometric and behavioral concept	Conceptual inspiration for "geometric redundancy"
Chirikjian & Burdick, 2002	Hyper-redundant manipulators	Models redundancy in configuration space	Provides mathematical analogy for behavioral convergence
Rhea et al., 2001	Global distributed storage	Focus on maintenance-free redundancy	Reveals hidden coordination costs in distributed systems
Mussa-Ivaldi & Hogan, 1991	Control of redundant systems	Shows shared control leads to coordinated behavior	Supports control-plane coupling argument
Chirikjian & Burdick, 1994	Hyper-redundant robotics	Explores redundancy collapse under constraints	Analogy for storage paths collapsing into one behavior
Zhang et al., 2004	Physical storage schemes	Optimizes path evaluation using shared structures	Demonstrates logical coupling despite physical separation
Morris & Truskowski, 2003	Evolution of storage systems	Tracks shift from hardware to software-centric control	Shows increasing importance of control-plane behavior
Rowstron & Druschel, 2001	Peer-to-peer storage systems	Introduces coordinated storage and caching	Early evidence of systemic behavior under load
Ackoff & Emery, 2005	Purposeful systems theory	Systems behave as unified entities due to shared goals	Theoretical grounding for systemic behavior in redundancy
Cowan, 1988	Human memory systems	Studies shared constraints in parallel systems	Cross-domain analogy for resource contention and coupling

3. Proposed Methodology

3.1. Conceptual Model: Geometry of Redundancy

This study introduces geometric redundancy as a theoretical framework for explaining the unified behavior that physically redundant storage architectures normally reveal. Geometric redundancy, unlike standard redundancy models which emphasize component duplication and physical separation, portrays redundancy as a behavioral space of the system components - their responses, interactions, and convergence under different operational conditions.

According to this framework, redundancy is not just presence or absence but rather spatial in nature. Even if two storage paths are physically different, they could be located in the same area of behavioral space if they have the same operating logic, share a state, or are synchronized through feedback mechanisms. On the other hand, parts sharing a physical infrastructure may show different behavior if their operational geometry is different. So, geometric redundancy checks for independence not from the physical layout but from the differentiation in responses.

The pivotal point separating these two models is the difference between logical separation and physical separation. Physical separation is taken as tangible isolation - different controllers, fabrics, devices, or network links. Logical separation means autonomy in decision-making, timing, state management, and recovery behavior. Storage systems of the present time typically feature maximum physical separation but minimum logical separation due to shared firmware versions, centralized configuration, uniform timeout thresholds, and coordinated failover policies. This merging results in a geometry where different paths fold into one single effective failure surface.

Failure surface is a term that characterizes the boundary conditions at which a system no longer operates normally but in a degraded or failed state. In systems with geometric redundancies, these failure surfaces tend to affect several components at the same time. A sudden increase in load, a control-plane freeze, a fight for metadata, or a burst of recovery can force all redundant routes over the same failure limit simultaneously and thus result in system-wide effects that are not attributable to a single component.

As a formal description of multi-level systems such as storage, the dependency graph is a Device-Tree-like structure that captures both explicit and implicit relationships inside and outside the storage layers. Nodes are components or services, whereas

edges depict dependency types such as control authority, shared state, synchronization, or resource contention. These are not your usual fault trees, these graphs are directional and contextual, meaning the same components can interact in different ways depending on the type of workload or failure mode.

Through the geometrical representation of redundancy, the method enables the study of stress-induced redundancy degradation, as well as the impact of design decisions on the behavioral landscape of storage systems.

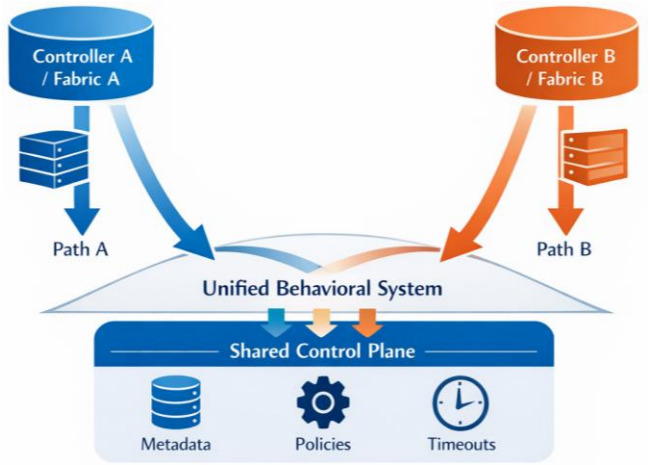

Fig 1: Geometry of Redundancy in Storage Systems

3.2. Analytical Framework

Continuing from the conceptual model, the analytical framework takes geometric redundancy and makes it measurable through an orderly dependency mapping and failure analysis. The framework covers different layers of the storage stack, such as hardware, firmware, network, operating system, multipath drivers, storage services, and orchestration or management plans. Besides components, each layer is examined for how it limits or coordinates behavior on parallel redundant paths.

It is aimed at identifying shared elements along supposedly independent paths. At the hardware layer, these may be shared power domains, clock sources, backplanes, or PCIe fabrics. The firmware layer involves shared codebases, upgrade cycles, and error-handling routines that are mapped. Dependencies at the network layer encompass shared switches, routing policies, congestion control mechanisms, and timing sensitivities. At the software layer, the dependencies encompass multipath algorithms, queue management, retry logic, metadata services, and centralized configuration stores.

As an illustration, a brief firmware stall can make the I/O queues blocked, activate multipath failover logic, overload alternate paths, and finally lead to system-wide latency spikes. The framework mainly focuses on propagation speed, amplification factors, and feedback loops which usually explain the fast behavior convergence of redundant paths.

In order to measure these effects the framework introduces observability metrics and fault isolation ones. Observability metrics consist of latency correlation coefficients between paths, synchronization of error counters, alignment of failover events, and temporal clustering of alerts. Fault isolation metrics evaluate the system's capacities to limit disturbances, such as time-to-divergence between paths, degree of load redistribution, and recovery asymmetry.

The most important thing is that the framework does not consider behavior to be static. Various operational regimes such as steady-state load, peak load, partial failure, and recovery are considered while taking measurements. This makes the analysis able to identify how redundancy geometry alters dynamically and why the systems that look independent under normal conditions become dependent under the stress. In a nutshell, these analytical components make it possible to measure geometric redundancy, which was merely a descriptive notion, as a property of the system.

3.3. Validation Approach

Stress scenarios artificially increase load, change access patterns, or simulate background tasks like rebuilds or rebalancing. Fault injection scenarios imitate partial failures such as link flaps, controller restarts, delayed acknowledgments, and control-plane stalls. The scenarios are gradually combined to see how the coupling develops.

Metrics that are instrumental to the verification process mainly revolve around independence against coupling. Examples are path-level latency divergence, throughput asymmetry, failover desynchronization, error localization, and recovery differentiation. When a system exhibits a high level of geometric redundancy, it responds divergently and the impact is localized, whereas the system with a low level of geometric redundancy shows synchronized degradation and unified recovery behavior.

Hardware/software setup and monitoring are of paramount importance. The data is gathered through storage-native telemetry, OS stats, multipath logs, network counters, and distributed tracing when available. Great care is taken at the time of synchronization of different data sources to enable proper correlation analysis. The approach does not rely on having insights into vendor proprietary internals, which is representative of the situation of most practitioners.

Some of the hypotheses supporting the verification are that, firstly, the use of identical configurations for redundant components is considered a realistic baseline instead of being an unusual case. Secondly, it is assumed that the workloads are representative but not necessarily exhaustive. Thirdly, the emphasis is on the relative behavior of paths rather than their absolute performance.

The verification process, which is carried out by this approach, shows how geometric redundancy can be directly seen, quantified, and compared across different architectures practically, thus serving as a basis for major architectural decisions.

4. Case Study

4.1. Environment Overview

The case study details a situation where a business-level storage system was set up to fulfill the stringent availability and performance requirements of the most critical running workloads. The setup is based on a dual-controller storage array that the active–active mode has been chosen for, and from there hosts get access through two physically separate storage fabrics. Every host is fitted with redundant host bus adapters, and multipath I/O software is employed for spreading the traffic and handling the failover scenarios by the paths already available. Structurally speaking, the setup is very much in tune with the highly regarded best practice principles of redundant storage design.

The two storage fabrics are completely apart physically as each of them has its own: switches, cabling, and zoning configurations. Storage controllers link similarly to both fabrics, and each controller provides hosts with the same logical unit mappings. The design of power supplies, cooling systems, and rack location is based on the idea of minimizing any shared physical dependencies. To operate more easily and to be more certain, things such as the firmware versions, configuration parameters, and the multipath policies are kept the same throughout the different components.

Both vendor documentation and internal architectural reviews describe the environment as one that has been fully redundant at which level there is not a single point of failure when it comes to component or connectivity. The failure modes of the two paths are taken to be statistically independent in the availability models which is why one can set quite aggressive service-level performance figures with respect to uptime and latency. In a similar vein but at a more detailed level, operational runbooks also indicate that loss or performance degradation on one of the paths should be completely masked by the use of other paths and the normal operation of the applications should not be disturbed in any significant way.

However, the operational data gathered over time revealed the case of a regular violation of a good practice where, at the very least, either performance degradation or failover behavior was observed on all paths simultaneously. Such instances triggered a reappraisal of the environment which was not prepared to be a rebellion against the physical design but rather to bring out the knowledge of how some form of behavioral coupling can appear in spite of structural redundancy. Therefore, it is the stress and failure scenarios that the case study traces rather than the normal one mode its nominal setup.

4.2. Failure Scenarios and Experiments

To assess the behavior of redundancy, several experiments that were planned as well as retrospective analyses were carried out. The planned fault injections were meant to be non-destructive and reversible, and they targeted the different layers of the storage stack, but they did not involve data loss. Besides the experiments, investigations of prior incident cases that have been captured by operational logs and monitoring systems were carried out.

The planned fault injections entailed, among other things, controlled link disruptions on a single fabric, temporary suspension of data services of a single controller, injection of artificial latency at the firmware level, and backend disk operation throttling to simulate internal contention. Each time, direct impact was limited to only one component or path, and the assumption was that the redundant paths would make up for it.

At the same time, past incident data were examined, and the focus was on events that had been explained by a single problem, like network congestion, controller overload, or temporary firmware anomalies. These incidents were helpful as a context because they showed how the system operated when controlled conditions were absent, including during high workload times and recovery events that had not been planned.

4.3. Observations and Data Collection

Data obtained through experimentations and incident investigations showed various behavioral patterns that were consistent. Consistency of latency on different redundant paths was the most noticeable pattern. Stress, whether artificially created or natural, causes path-level latency metrics which under normal conditions load diverge to rapidly align thus, latency on different redundant paths becoming consistent. This was a clear indication that the paths have shared bottlenecks or that the paths behave in throttling synchronously.

The spreading of errors also led to the strengthening of this sentiment. Counters measuring errors, events of timeout, and abrupt increases of retries happened almost at the same time on different paths, even when one path was the only one to experience an injected fault. This synchronization is an indication that the error-handling logic was working on common thresholds or global state thus, the localized disturbances being communicated system-wide.

Insight was also gained through the observation of recovery behavior. Instead of slow and localized recovery, the system showed performance improvements via step-functions, that is, all paths moved from degraded to stable states together in a very short time interval. These state changes were more often linked to control-plane events like the completion of cache rebalancing, release of metadata lock, or health-state normalization internally, rather than the restoration of individual physical components.

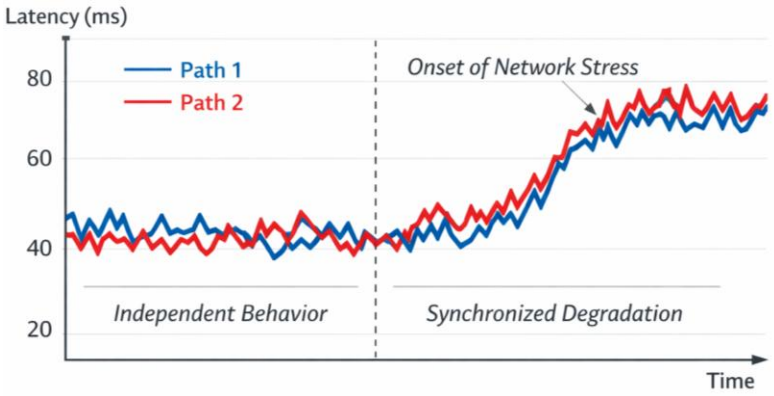

Fig 2: Correlated Latency Across Physically Redundant Paths

5. Results And Discussion

5.1. Key Findings

The case study results indicate quite clearly that the storage architecture checked so thoroughly for physical redundancy nevertheless was very systemic consistently across stress and failure. Instead of using redundant paths to isolate problems, these paths frequently converged in terms of performance, error response, and recovery timing. This convergence was not accidental; it went on to show up time and time again in both planned fault injections and real-world incidents, thus pointing to a system feature rather than an anomaly.

A major revelation was that latency and error signals across multiple paths became more and more synchronized. Queue depth, response time, and the number of retries - these metrics were expected to diverge - ended up displaying a high temporal correlation. In any case, alternate paths suffered a simultaneous or almost immediate degradation even when only one fabric or controller was faulty. Sometimes, non-faulted paths were more impacted than directly affected ones, which runs contrary to traditional redundancy theories.

When analyzing dependency mappings it turned out that it was mainly the result of shared components and shared decision logic that causes this behavior rather than simply physical dependencies. Common firmware execution paths, centralized metadata management, uniform timeout thresholds, and synchronized cache and queue management have led to a dominant role. Multipath

I/O software, which was intended to make the system more resilient, ended up increasing dependency since it always reacts the same way to a congestion or an error signal thus failover or throttling behavior is triggered across all paths concurrently.

These discoveries demonstrate a collapse of the classical redundancy concepts particularly those concerning the assumption of independence. Physical separation limited exposure to some classes of hardware failures but it could not result in behavioral isolation. When stressed the system acts like a single unit in which failure surfaces span beyond one nominal fault domain. Thus, this is the reason why failure models focusing only on component counts and physical topology tend to give an availability level that is too high compared to the actual one.

What is more, the study does not indicate that redundancy is useless but rather that the validity of its effectiveness is dependent. Redundancy was able to prevent the lack of data entirely but it did not manage to stop the systemic degradation and the unified recovery behavior. This difference brings out the fact that one must carefully separate survivability from performance isolation when s/he is assessing redundant storage systems.

5.2. Implications for Storage Architecture Design

The implications of the study's findings extend directly to how storage architectures identify and deploy fault domains. Historically, system component fault isolations demarcated boundaries around different hardware pieces—controllers, fabrics or devices neglecting the behavioral coupling via control planes and logic synchronization. The findings of this study suggest that fault domains should hence be characterized in behavioral terms, taking into account the communal response mechanisms of components when subjected to stress.

Besides simply reproducing hardware, the essence of designing for actual independence incorporates a measure of control-plane diversity, which stands out as a vital criterion. Techniques may involve, for instance, timeouts that are differently set, recovery procedures that are dissimilar, metadata channels that are kept apart, or firmware contexts that are differentiated. Indeed, such diversity can make the operation a bit complicated; however, the probability of all routes crossing the same failure surface simultaneously is drastically decreased.

An additional implication is related to observability and testing. Systems should not only be assessed via the simulation of component faults but also through correlation-oriented testing that evaluates the degree of path divergence during stress and recovery. Correlation is such a critical factor that it acts like a sonar signal, indicating that you have low geometric redundancy, signaling a strong physical separation.

Finally, these outcomes cast some doubt on the very premise of uniform configuration being a desirable feature. Managing a system becomes simpler with consistency; however, excessive conformity might strengthen coupling. Controlled heterogeneity achieved at the level of algorithms or policies could, therefore, be a way to enhance the robustness of the system by altering its redundancy geometrical arrangement.

Table 2: Indicators of Low vs High Geometric Redundancy

Dimension	High Geometric Redundancy	Low Geometric Redundancy
Latency behavior	Divergent across paths	Highly correlated
Failover timing	Asynchronous	Synchronized
Recovery pattern	Localized, gradual	Unified, step-function
Control-plane logic	Differentiated	Uniform
Failure surface	Narrow, localized	Broad, systemic

5.3. Limitations

The present research has several drawbacks that should be taken into account when the results are interpreted. Firstly, the case study is limited to only one enterprise storage environment. Although the behaviors observed are in line with the reports of the industry at large, the findings might not totally apply to all storage platforms, especially those with entirely different architectures or control-plane designs.

Secondly, the work is limited to the visibility provided by the standard operational tools used. The research does not depend on proprietary internals, which enhances its practical relevance but restricts the insight into certain low-level controller behaviors. Some coupling mechanisms, therefore, may still be guesses rather than direct observations.

Thirdly, workload patterns, although representative, do not encompass the whole spectrum of real-world access behaviors. A different combination of applications or ways of failure could lead to different redundancy geometries.

Nevertheless, the agreement of facts brought up in different situations significantly supports the main point of the discussion: just having physical redundancy is not enough to ensure the independence of behavioral characteristics. Further investigations with several platforms and more in-depth instrumentations will provide better and more detailed support for the model proposed.

6. Conclusion And Future Scope

6.1. Conclusion

Through this study, the authors wanted to find out how people's perception of redundancy in modern storage systems did not match the actual level of redundancy. They also came up with a new idea called geometric redundancy, which explains why when two or more physically separate paths are combined, in reality, they behave as if they are a single system. They further conducted conceptual modeling, developed analytical frameworks, and did an empirical case study to show that it is not only physical isolation that determines redundancy effectiveness but also other factors like the control planes being shared, logic being synchronized, and feedback-based interaction.

The case study demonstrated that there is a latency convergence, error propagation gets synchronized, and recovery is unification in a manner that even if the environment is for full physical redundancy, there is still a mismatch or structural problem between the traditional fault-domain assumptions and the operational reality.

The author posits that if you think of redundancy as a behavior of the whole system rather than just its structure, then the work serves as a toolkit for practitioners and researchers alike to critically evaluate and come up with storage architectures. In particular, for practitioners, the message of the finding is to take into account behavioral coupling in all aspects, like defining fault domains, bringing recovery plans to life, and configuring control-plane policy. Whereas for researchers, the idea of geometric redundancy is a key to examining failure surfaces, dependency graphs, and system dynamics at several storage layers; thereby, it is a way to unite the theory of reliability, performance modeling, and operational observations.

In summary, the investigation emphasizes the fact that redundancy, hence, is not independence. Hence, by physically separating different components, the chances of a catastrophic outage are made lesser but correlated performance degradations can still be there. Figuring out and quantifying the geometry of redundancy brings storage system resilience into focus more sensitively and real-world, not idealized, assumptions become the basis for design decisions informed by the explained behavior.

6.2. Future Research Directions

This research has found several avenues for future work that would be really interesting to explore. Firstly, if geometric redundancy could be quantitatively modeled, it might facilitate predictive assessment of systemic risk by combining probability-based fault models with behavioral dependency graphs. These kinds of predictive tools could be very useful for system architects who would be able to spot potential failure correlations even before deployment.

Secondly, the development of AI-assisted dependency discovery might be a very valuable contribution towards identifying hidden coupling in hardware, firmware, network, and software layers. Machine learning applied to telemetry, logs, and configuration data could reveal interactions that may be overlooked by manual analysis due to their subtle nature.

Lastly, taking the notion of geometric redundancy further into cloud-native and disaggregated storage architectures deserves to be a primary area of focus. In these setups, the control plane, orchestration layers, and multi-tenant workloads add new coupling mechanisms, thereby making conventional physical redundancy models even less predictive. Figure out geometric redundancy scenarios here would give a big leg up to companies aiming at creating highly resilient systems that structurally and operationally fit each other.

Work in this line of research can, therefore, sharpen both the theory and practice of redundancy, eventually leading to storage systems that are not only highly available but also behaviorally robust under real-world conditions.

References

[1] Pinheiro, Eduardo, Ricardo Bianchini, and Cezary Dubnicki. "Exploiting redundancy to conserve energy in storage systems." Proceedings of the joint international conference on Measurement and modeling of computer systems. 2006.

[2] Klein, Henning, and Jorg Keller. "Storage architecture with integrity, redundancy and encryption." 2009 IEEE International Symposium on Parallel & Distributed Processing. IEEE, 2009.
[3] Orenstein, Jack A. "Redundancy in spatial databases." ACM SIGMOD Record 18.2 (1989): 295-305.
[4] Gibson, Garth Alan. Redundant disk arrays: Reliable, parallel secondary storage. University of California, Berkeley, 1991.
[5] Sheaffer, Jeremy W., David P. Luebke, and Kevin Skadron. "A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors." Graphics Hardware. Vol. 2007. 2007.
[6] Ivaldi, FA Mussa, P. Morasso, and R. Zaccaria. "Kinematic networks: A distributed model for representing and regularizing motor redundancy." Biological cybernetics 60.1 (1988): 1-16.
[7] Muppaneni, Rajarshi Krishna. “Retail Reimagined: How Dynamics 365 Commerce Is Driving Omnichannel Experiences”. International Journal of AI, BigData, Computational and Management Studies, vol. 1, no. 1, Mar. 2020, pp. 49-59
[8] Rhea, Sean, et al. "Maintenance-free global data storage." IEEE internet computing 5.5 (2001): 40-49.
[9] Mussa-Ivaldi, Ferdinando A., and Neville Hogan. "Integrable solutions of kinematic redundancy via impedance control." The International Journal of Robotics Research 10.5 (1991): 481-491.
[10] Chirikjian, Gregory S., and Joel W. Burdick. "A hyper-redundant manipulator." IEEE Robotics & Automation Magazine 1.4 (1994): 22-29.
[11] Zhang, Ning, Varun Kacholia, and M. Tamer Ozsu. "A succinct physical storage scheme for efficient evaluation of path queries in XML." Proceedings. 20th International Conference on Data Engineering. IEEE, 2004.
[12] Morris, Robert JT, and Brian J. Truskowski. "The evolution of storage systems." IBM systems Journal 42.2 (2003): 205-217.
[13] Rowstron, Antony, and Peter Druschel. "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility." Proceedings of the eighteenth ACM symposium on Operating systems principles. 2001.
[14] Ackoff, Russell L., Russell Lincoln Ackoff, and Fred E. Emery. On purposeful systems: An interdisciplinary analysis of individual and social behavior as a system of purposeful events. Transaction Publishers, 2005.
[15] Cowan, Nelson. "Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system." Psychological bulletin 104.2 (1988): 163.
[16] Chirikjian, Gregory S., and Joel W. Burdick. "Kinematically optimal hyper-redundant manipulator configurations." IEEE transactions on Robotics and Automation 11.6 (2002): 794-806.