Unified Storage Health Monitoring on macOS

SMART-Aware & SMART-Fallback Architecture

Tien Hanprab
tienhanprab@gmail.com

A Technical Implementation of Adaptive Disk Health Monitoring

Portfolio Project Documentation

Abstract

This paper presents a unified storage health monitoring system for macOS that addresses the fundamental challenge of partial hardware visibility in modern computing environments. The system implements a dual-path architecture that provides reliable health assessments for all storage devices, regardless of SMART data availability. By employing graceful degradation principles, the system maintains operational effectiveness even when direct hardware telemetry is inaccessible, making it suitable for production monitoring environments with heterogeneous storage configurations.

Keywords: Storage Monitoring, SMART, macOS, System Architecture, Health Scoring, Observability

1. Introduction

Storage health monitoring is a critical component of system reliability engineering. Traditional approaches rely heavily on Self-Monitoring, Analysis and Reporting Technology (SMART) data, which provides direct hardware telemetry. However, in real-world deployments—particularly on macOS systems—SMART data is frequently unavailable due to USB bridge limitations, vendor-specific protocols, or hardware abstraction layers.

This project addresses this operational reality by implementing a monitoring system that treats SMART as an optimization rather than a dependency. The system automatically detects all physical storage devices, determines SMART capability on a per-device basis, and generates consistent health reports using the best available data source for each device.

1.1 Problem Statement

On macOS, external USB storage devices often do not expose SMART data despite having healthy underlying hardware. This creates a visibility gap: conventional monitoring tools either fail entirely or return incomplete data, leaving system operators without reliable health signals for a significant portion of their storage infrastructure.

1.2 Design Objectives

The system was designed to achieve four primary objectives:

1. Detect all physical storage devices attached to the system
2. Identify SMART capability on a per-device basis
3. Provide meaningful health assessments for every device
4. Avoid false confidence or silent failures in reporting

2. System Architecture

The system implements a pipeline-based architecture with decision gates, allowing for modular component design and future extensibility. Each stage in the pipeline is responsible for a specific aspect of the health assessment process.

2.1 Pipeline Stages

The monitoring pipeline consists of five distinct stages:

Stage 1	**Disk Detection**	**Enumerate all physical storage devices**
Stage 2	SMART Verification	Probe SMART capability for each device
Stage 3	Data Collection	Execute appropriate health assessment path
Stage 4	Health Scoring	Normalize metrics to unified 0-100 scale
Stage 5	Reporting	Generate JSON and CLI outputs

2.2 Architecture Diagram

Figure 1 illustrates the complete system flow, highlighting the decision gate at the SMART capability detection stage that routes each device through the appropriate health assessment path.

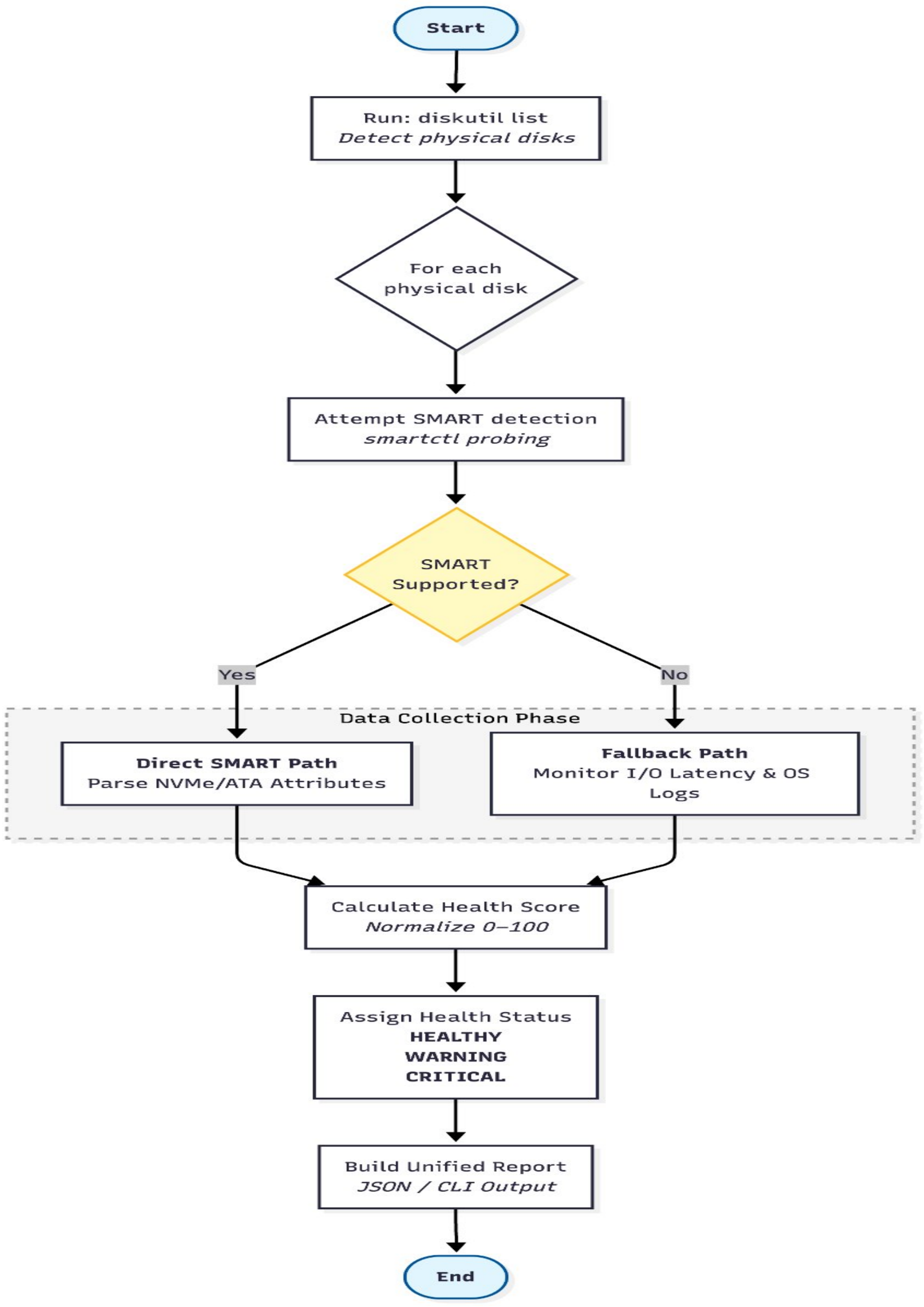

Figure 1: SMART Disk Health Monitoring Pipeline

3. Component Design

The system is organized into six primary modules, each with clearly defined responsibilities. This modular design facilitates testing, maintenance, and future enhancement.

3.1 Main Orchestrator (main.py)

Serves as the system entry point and orchestration layer. Controls execution flow, routes each disk through the appropriate health assessment path, and ensures consistent output formatting regardless of disk type or data availability.

3.2 Disk Detector (disk_detector.py)

Executes `diskutil list` to enumerate all storage devices, then filters logical volumes and containers to isolate physical disks only. This prevents false health signals from virtual or logical storage layers that represent abstractions of underlying hardware.

3.3 SMART Capability Checker (smart_checker.py)

Probes each physical disk using `smartctl` to determine SMART support. Implements the critical decision gate that determines which health assessment strategy to employ. This component recognizes that on macOS, many USB bridges block SMART passthrough despite healthy underlying hardware.

3.4 SMART-Based Health Analyzer (health_score.py)

Executed when SMART data is available. Parses NVMe and ATA SMART attributes to evaluate wear level, error counters, and temperature behavior. Normalizes these hardware-level metrics into a unified health score, providing high-fidelity insight into device condition.

3.5 Indirect Health Estimator (indirect_health.py)

Activated when SMART data is unavailable. Uses system-observable signals such as disk usage pressure, I/O latency, and OS-level error indicators to infer device health probabilistically. This component embodies the design principle: if internal telemetry is inaccessible, assess health by observing external behavior.

3.6 Unified Report Generator (report.py)

Consolidates health assessment results across heterogeneous devices and generates dual-format output: JSON for automation and machine integration, and CLI table format for human operators. Ensures operational clarity regardless of the data source used for each device.

4. Health Scoring Model

All devices are evaluated using a common 0-100 health score scale, regardless of whether the assessment used SMART data or indirect system metrics. This normalization enables fair comparison and consistent operational decision-making across heterogeneous storage configurations.

4.1 Score Interpretation

Score Range	Status	Interpretation
80 - 100	HEALTHY	Normal operation, no intervention required
50 - 79	WARNING	Early degradation signs, monitor closely
< 50	CRITICAL	Immediate attention required, consider replacement

4.2 Scoring Methodology

For SMART-enabled devices, the score is computed from weighted combinations of wear indicators, error counters, and thermal metrics. For devices without SMART, the score is derived from I/O performance characteristics, system error logs, and usage patterns. Both methodologies map to the same 0-100 scale using validated thresholds.

5. Implementation Highlights

5.1 Graceful Degradation Strategy

The system's core innovation is its graceful degradation approach. Rather than treating SMART unavailability as a failure condition, the system transparently switches to alternative assessment methods. This ensures continuous monitoring capability across the entire storage infrastructure, eliminating blind spots that conventional SMART-only tools would create.

5.2 Conditional Data Collection

The decision gate at the SMART verification stage implements a binary routing strategy:

Direct SMART Path: When SMART is supported, the system parses hardware-level attributes including wear indicators (percentage used, media errors), thermal metrics (current and maximum temperatures), and integrity counters (uncorrectable errors, reallocated sectors).

Fallback Path: When SMART is unavailable, the system monitors I/O latency patterns using `iostat`, analyzes disk utilization pressure, and scans system logs for storage-related errors. These indirect signals are aggregated to produce a probabilistic health estimate.

5.3 Dual-Format Reporting

The system generates two complementary output formats from the same underlying data:

JSON Format: Structured data suitable for integration with monitoring platforms, alerting systems, or automated decision-making workflows. Includes full metadata about assessment methodology and confidence levels.

CLI Table Format: Human-readable tabular display optimized for terminal viewing. Uses color coding (when supported) to highlight health status and provides at-a-glance operational awareness.

6. Known Limitations

The system explicitly acknowledges several operational constraints that inform appropriate use and interpretation of results:

1. SMART data is frequently inaccessible over USB on macOS due to bridge chip limitations
2. Indirect health assessment is inferential rather than diagnostic
3. Vendor-specific SMART attributes may vary across device manufacturers
4. The system performs read-only monitoring without destructive testing
5. Current implementation is macOS-specific due to <font name='Courier'>diskutil</font> dependency

These limitations are transparently communicated rather than hidden, ensuring users maintain appropriate confidence levels when interpreting health reports.

7. Engineering Design Rationale

This architecture reflects real-world operational constraints where perfect hardware telemetry cannot be assumed. The design philosophy prioritizes operational resilience over theoretical purity.

Rather than failing when SMART is unavailable, the system detects the limitation, adapts its assessment strategy, and communicates confidence transparently. This approach makes it suitable for production monitoring environments where heterogeneous storage configurations are the norm rather than the exception.

7.1 Key Design Decisions

Pipeline Architecture: Modular stages enable independent testing and future extensibility (e.g., Linux support, additional data sources).

Decision Gates: Conditional routing based on capability detection rather than hard dependencies prevents cascade failures.

Normalized Scoring: Common scale across heterogeneous devices enables consistent operational decision-making.

Dual Output: Simultaneous support for automation and human operators maximizes operational utility.

8. Operational Use Cases

The system addresses several practical monitoring scenarios:

Heterogeneous Storage Environments: Organizations with mixed internal and external storage can obtain unified health visibility.

USB Storage Monitoring: External drives that don't expose SMART still receive meaningful health assessments.

Automated Health Checks: JSON output enables integration with monitoring platforms like Prometheus, Grafana, or Nagios.

Preventive Maintenance: Early warning signals enable proactive disk replacement before failures occur.

9. Future Enhancements

The modular architecture facilitates several potential extensions:

Platform Support	Extend to Linux using equivalent system tools
Additional Metrics	Integrate filesystem-level health indicators
Historical Tracking	Implement time-series storage for trend analysis
Predictive Analysis	Apply machine learning to failure prediction
Alert Integration	Direct webhook support for monitoring platforms
Network Storage	Extend monitoring to NAS and SAN devices

10. Conclusion

This project demonstrates an observability-first engineering approach to storage health monitoring. By treating SMART as an optimization rather than a dependency, the system maintains operational effectiveness across heterogeneous storage configurations that would defeat conventional monitoring tools.

The graceful degradation strategy ensures no device is left unmonitored due to technical limitations of the monitoring infrastructure. This design philosophy—adapting to constraints rather than failing because of them—makes the system suitable for production deployment in environments where perfect hardware visibility cannot be assumed.

The modular architecture provides a foundation for future enhancements while the normalized health scoring enables consistent operational decision-making today. This balance between immediate utility and extensibility reflects practical engineering trade-offs in real-world system design.

Technical Summary

Language: Python 3.x

Key Dependencies: smartctl, diskutil, iostat

Architecture: Pipeline-based with decision gates

Output Formats: JSON, CLI table

Platform: macOS (extensible to Linux)

Design Pattern: Graceful degradation