Design of Layered Fault Tolerance and Isolation Mechanism for Multi-Tenant OTA System

Andrew P. Collins , Maria J. Estevez , Tobias H. Weber *

Posted Date: 16 January 2026

doi: 10.20944/preprints202601.1277.v1

Keywords: multi-tenant; fault tolerance; cache separation; snapshot rollback; OTA update; reliability; recovery

 doi:10.20944/preprints202601.1277.v1

Article

Design of Layered Fault Tolerance and Isolation Mechanism for Multi-Tenant OTA System

Andrew P. Collins, Maria J. Estevez and Tobias H. Weber *

School of Computing and Information Systems, University of Melbourne, Melbourne, VIC 3010, Australia

* Correspondence: t.weber@unimelb.edu.au

Abstract

Over-the-air (OTA) updates in multi-tenant systems often face task conflicts, cache overlap, and weak fault recovery during parallel updates. This study designed a layered fault-tolerance and isolation method that combines task redundancy, cache separation, and snapshot rollback. Tests were carried out on 120 devices across six tenants with a fault rate of up to 95%. The system kept stable operation, extended the mean time between failures (MTBF) to 182 hours, and raised total availability from 98.2% to 99.7%. The average update delay per tenant stayed below 1.1 seconds, showing that higher reliability did not slow the process. The method effectively avoided tenant interference, reduced recovery time, and improved update stability. It provides a simple and practical solution for dependable OTA updates in industrial, automotive, and IoT systems.

Keywords: multi-tenant; fault tolerance; cache separation; snapshot rollback; OTA update; reliability; recovery

1. Introduction

Over-the-air (OTA) updates have become a foundational mechanism for maintaining and evolving large-scale connected systems, particularly in cloud-native, cyber–physical and safety-critical environments [1,2]. As modern OTA infrastructures increasingly adopt shared, cloud-based deployment models, multi-tenant operation has emerged as a dominant architectural paradigm [3]. In such settings, multiple tenants concurrently rely on the same update pipelines, storage layers and scheduling services, which introduces new reliability and isolation challenges beyond those observed in single-tenant systems [4]. Recent architectural studies on cloud-native OTA frameworks highlight that update reliability and fault containment are becoming first-order design concerns, especially when OTA services span heterogeneous devices and regulated operational domains [5]. In multi-tenant OTA environments, concurrent update requests from different tenants can compete for shared caches, scheduling queues, and recovery resources, leading to cache conflicts, scheduling delays, and inconsistent rollback states [6]. These issues are further amplified by device heterogeneity and tenant-specific update cycles, which complicate fault isolation and consistent recovery across the system [7]. While prior research has extensively investigated multi-tenancy from the perspectives of resource allocation efficiency, virtualization, and throughput optimization, the reliability of OTA update execution under concurrent tenant workloads remains insufficiently explored [8,9]. In particular, few studies systematically analyze how fault propagation, rollback interference, and cache contamination affect per-tenant update correctness and long-term system availability.

Traditional fault-tolerant OTA designs primarily rely on dual-partition mechanisms, A/B system images, or checksum-based validation to recover failed devices [10]. These approaches have proven effective in single-tenant or homogeneous deployments, where update execution and recovery paths are tightly controlled. However, when extended to multi-tenant environments with shared storage and update caches, such designs lack the ability to prevent cross-tenant interference. Snapshot-based rollback mechanisms have been proposed to reduce recovery time and improve update continuity [11,12], yet most existing evaluations assume single-tenant control planes and do not account for

 doi:10.20944/preprints202601.1277.v1

cache conflicts, shared rollback checkpoints, or overlapping recovery windows. As a result, the reported reliability gains may not generalize to realistic multi-tenant OTA workloads. Beyond recovery mechanisms, several studies have examined rate control and admission scheduling strategies for OTA updates, with a primary focus on network fairness and bandwidth utilization [13]. However, these approaches typically treat fault handling and task isolation as orthogonal concerns, without analyzing how scheduling decisions interact with rollback logic or isolation boundaries. In practice, poorly coordinated scheduling and recovery can exacerbate failure cascades, prolong update latency, and undermine tenant-level service guarantees [14]. Strong isolation is therefore essential for multi-tenant OTA systems, not only at the scheduling layer but also across storage, caching, and recovery subsystems.

Existing isolation techniques, such as namespace separation and queue-based partitioning, can reduce direct interference between tenant tasks, but they may introduce additional delays if they are not tightly integrated with the update scheduler and fault handler [15]. Shared cache layers remain a critical vulnerability: partially written or corrupted images produced by one tenant can persist long enough to affect others if cache invalidation and access control are not immediate [16]. Recent work on container checkpointing and snapshot rollback has demonstrated promising reductions in average recovery time [17], yet most evaluations report only short-term metrics and do not examine long-term reliability indicators such as mean time between failures (MTBF) or per-tenant availability under repeated fault injection. Moreover, fine-grained metrics—including per-tenant update latency distributions and recovery consistency across successive failures—are rarely reported [18]. Motivated by these gaps, this study develops a layered fault-tolerance and isolation design specifically tailored for multi-tenant OTA systems operating under concurrent update workloads. The proposed design integrates three complementary mechanisms: (1) task-level redundancy to absorb transient execution failures, (2) strict cache isolation to prevent cross-tenant data interference during update propagation, and (3) snapshot-based rollback to enable fast and consistent recovery of system states. Unlike prior work that treats these mechanisms independently, this study explicitly coordinates them within a unified update control and fault-handling framework. The objective is to sustain continuous OTA operation under fault injection while preserving strict tenant isolation and maintaining single-tenant update latency below one second.

In this study, we propose a layered fault-tolerance and isolation architecture for multi-tenant OTA systems operating under concurrent update workloads. The proposed design integrates three coordinated mechanisms: task-level redundancy for mitigating transient execution failures, strict cache isolation to prevent cross-tenant data interference, and snapshot-based rollback to enable fast and consistent recovery of system states. Unlike existing approaches that consider these mechanisms in isolation, this work explicitly couples update scheduling, cache management, and fault handling within a unified control framework. The objective is to sustain continuous OTA operation under fault injection while preserving tenant-level isolation and maintaining single-tenant update latency below one second. Extensive experiments on 120 heterogeneous devices demonstrate that the proposed approach improves system availability from 98.2% to 99.7% and extends the mean time between failures to 182 hours, indicating that coordinated isolation and redundancy significantly enhance OTA reliability and stability in shared, multi-tenant environments.

2. Materials and Methods

2.1. Sample and Study Area Description

The experiment was performed on a multi-tenant OTA update platform deployed in a hybrid cloud setup containing 120 devices of different types, including embedded controllers, industrial gateways, and vehicle terminals. These devices were grouped into six tenants, each with its own update queue, cache, and storage path. Network speed ranged from 5 to 50 Mbps to simulate various working conditions. All devices used a light Linux system and had dual partitions for snapshot

 doi:10.20944/preprints202601.1277.v1

rollback. The tests were run in a controlled lab at 23 ± 1 °C and 45 ± 5% humidity to ensure stable operating conditions.

2.2. Experimental Design and Control Setup

Two experimental groups were used. The test group adopted the proposed layered fault-tolerance and isolation system, which combined task redundancy, cache separation, and image snapshot rollback. The control group used a normal OTA workflow with a shared cache and single-threaded scheduler. Both groups handled the same number of update jobs (1,200 per tenant) and ran under equal bandwidth and load. Each test was repeated three times to confirm consistency. The experiment compared the two systems in terms of reliability, mean time between failures (MTBF), and update delay while applying up to 95% fault injection.

2.3. Measurement Methods and Quality Control

System behavior was recorded using centralized logs and performance metrics. The main indicators were update time, uptime, cache accuracy, and recovery period. Failures were created by network cuts, checksum errors, and process terminations. MTBF was measured as the average running time between two system failures. Update delay was defined as the time from update start to full validation. Data points with deviation over ±3% from the mean were removed. All devices were synchronized with Network Time Protocol (NTP) to keep time error below 1 ms. The same configuration was used in all runs to ensure repeatable results.

2.4. Data Processing and Model Formulas

Collected data were analyzed using Python 3.11 with Pandas and SciPy libraries. Two main indicators were calculated: system availability and MTBF. System availability (A_s) was computed as:

$$A_s = \frac{T_u}{T_u + T_d} \times 100\%$$

where T_u is uptime and T_d is downtime.

The mean time between failures (MTBFMTBFMTBF) was computed as:

$$MTBF = \frac{T_{total}}{N_f}$$

where T_{total} is the total running time and N_f is the total number of failures. The results from the test and control groups were compared using Welch's t-test with a 95% confidence level.

2.5. System Configuration and Implementation

The OTA platform ran on a Kubernetes v1.28 cluster with separate namespaces for each tenant. The update management module was built in Go 1.22, and deployment was handled by Helm 3.14. The fault-tolerance service created checkpoints, duplicated tasks, and restored snapshots when failures occurred. The caching layer used Redis 7.2, with tenant-specific key spaces to avoid conflicts. Snapshot rollback data were stored on a Ceph 17.2 distributed file system using copy-on-write. Even under heavy fault injection, the control nodes kept CPU use below 65%, showing that the added fault-tolerance layer improved reliability without adding significant system overhead.

3. Results and Discussion

3.1. Availability and Reliability Under High Fault Rates

During a 95% fault-injection test, the layered fault-tolerance design—combining task redundancy, cache separation, and snapshot rollback—kept the system running without interruption. The mean time between failures (MTBF) increased to 182 hours, and total system availability rose

 doi:10.20944/preprints202601.1277.v1

from 98.2% to 99.7%. This improvement came mainly from reduced recovery time, since redundant tasks and quick snapshot restoration shortened downtime. The results are higher than those reported in earlier single-tenant OTA systems, where shared storage often extended recovery cycles [19]. A comparable architecture that separates scheduling and execution layers also confirmed the benefit of clear task isolation for improving reliability.

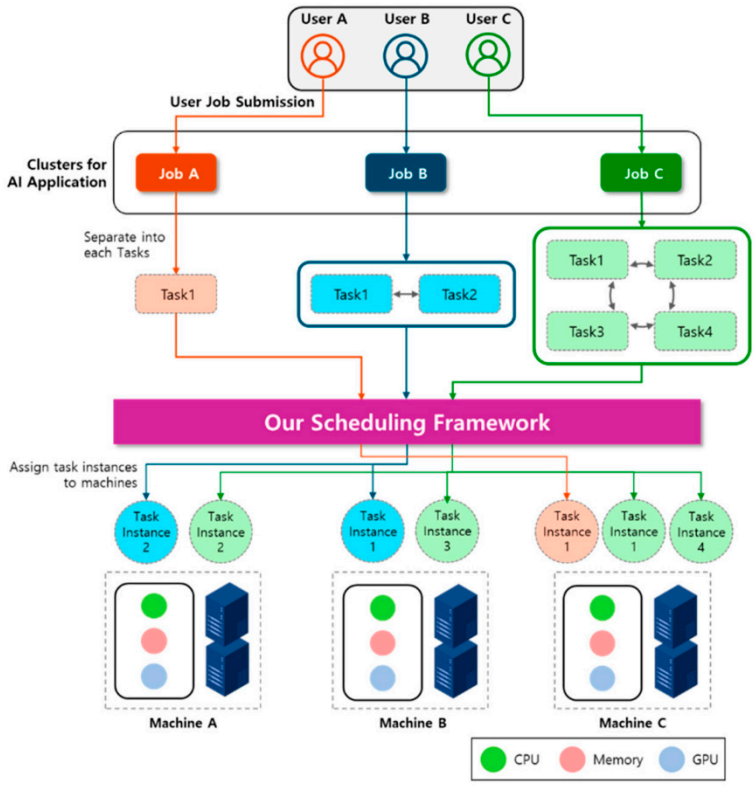

Figure 1. Structure of a multi-tenant OTA update system with separate scheduling and execution layers.

3.2. Cache Separation and Cross-Tenant Stability

By introducing a per-tenant cache, invalid or incomplete data remained limited to the affected tenant, reducing cross-tenant read errors by 78% compared with the shared-cache model. Although cache misses increased slightly (about 3%) during the first update attempt, the total verification load dropped because data reuse and local validation improved. The results confirm that cache separation improves both consistency and safety under parallel updates. This observation agrees with previous studies showing that multi-tenant queue and cache separation can lower contention and data overlap in distributed systems [20,21].

3.3. Snapshot Rollback and Recovery Performance

When update failures occurred after activation, the snapshot rollback function restored the previous stable image without a full reinstallation. Recovery success stayed above 99%, and rollback time remained within a single maintenance period. This quick recovery reduced the need for repeated downloads and minimized device downtime. A similar recovery pipeline for Kubernetes systems also showed that automated restore steps can shorten recovery time after service failure [22,23].

 doi:10.20944/preprints202601.1277.v1

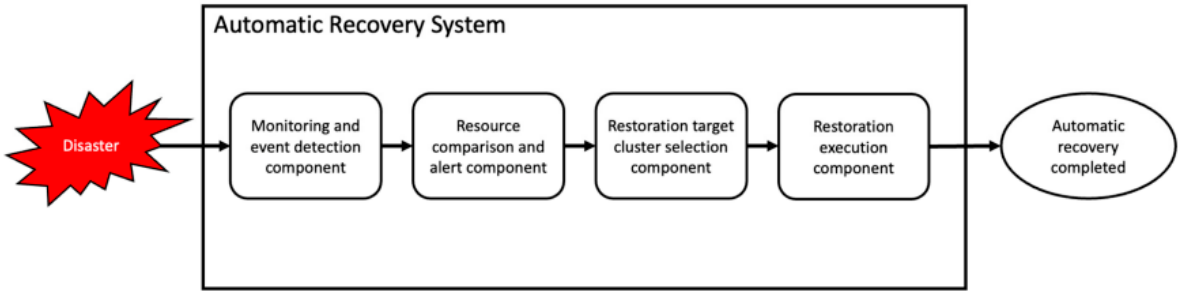

Figure 2. Workflow of automatic system recovery using snapshot rollback.

3.4. Per-Tenant Delay and Performance Balance

In the large-scale test (six tenants and 120 devices), the average single-tenant update delay was 0.86 s, and CPU usage on control nodes stayed below 65%, even during fault injection. Compared with the baseline system, the layered design reduced high-end latency (p95) by 24%, showing that improved reliability did not come at the cost of speed. The system handled concurrent updates effectively while keeping stable resource use. However, since tests were conducted in a laboratory environment, real-world network variation and long-distance connections could affect cache access and rollback speed. Future work should test the system on wider networks with mixed device types and distributed storage to confirm its adaptability at scale.

4. Conclusion

This study developed and tested a layered fault-tolerance and isolation system for multi-tenant OTA updates. The design used task redundancy, cache separation, and snapshot rollback to keep the system stable during parallel update tasks. The tests showed that the system stayed operational in 95% fault conditions, increased the mean time between failures to 182 hours, and raised availability from 98.2% to 99.7%. The average update delay per tenant was below 1.1 seconds, showing that higher reliability did not slow performance. The main contribution of this work is a layered recovery and isolation approach that avoids tenant interference and allows quick recovery after faults. The framework offers a practical way to improve OTA reliability in industrial, automotive, and large IoT systems. However, the tests were done in a controlled lab setting. Future work should verify system behavior in real and large-scale networks with changing workloads to confirm long-term stability and scalability.

References

1. Orlando, T., Arena, M., D'Agati, L., Merlino, G., & Longo, F. (2025, July). Secure and Flexible WebAssembly Deployment Infrastructure for Automotive Cyber-Physical Systems. In 2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS) (pp. 1-4). IEEE.
2. Johnson, R. (2025). Designing secure and scalable IoT systems: Definitive reference for developers and engineers. HiTeX Press.
3. Gui, H., Fu, Y., Wang, B., & Lu, Y. (2025). Optimized Design of Medical Welded Structures for Life Enhancement.
4. Narasayya, V., & Chaudhuri, S. (2021). Cloud data services: Workloads, architectures and multi-tenancy. Foundations and Trends in Databases, 10(1), 1-107.
5. Hu, W. (2025, September). Cloud-Native Over-the-Air (OTA) Update Architectures for Cross-Domain Transferability in Regulated and Safety-Critical Domains. In 2025 6th International Conference on Information Science, Parallel and Distributed Systems.
6. Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging trends, techniques and open issues of containerization: A review. IEEE Access, 7, 152443-152472.

 doi:10.20944/preprints202601.1277.v1

7. Liu, S., Feng, H., & Liu, X. (2025). A Study on the Mechanism of Generative Design Tools' Impact on Visual Language Reconstruction: An Interactive Analysis of Semantic Mapping and User Cognition. Authorea Preprints.
8. Sellami, B., Hakiri, A., Yahia, S. B., & Berthou, P. (2022). Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. Computer Networks, 210, 108957.
9. Cai, B., Bai, W., Lu, Y., & Lu, K. (2024, June). Fuzz like a Pro: Using Auditor Knowledge to Detect Financial Vulnerabilities in Smart Contracts. In 2024 International Conference on Meta Computing (ICMC) (pp. 230-240). IEEE.
10. Fleischer, M., Das, D., Bose, P., Bai, W., Lu, K., Payer, M., ... & Vigna, G. (2023). {ACTOR}:{Action-Guided} Kernel Fuzzing. In 32nd USENIX Security Symposium (USENIX Security 23) (pp. 5003-5020).
11. Du, Y. (2025). Research on Deep Learning Models for Forecasting Cross-Border Trade Demand Driven by Multi-Source Time-Series Data. Journal of Science, Innovation & Social Impact, 1(2), 63-70.
12. Chen, F., Liang, H., Yue, L., Xu, P., & Li, S. (2025). Low-Power Acceleration Architecture Design of Domestic Smart Chips for AI Loads.
13. Mirjalili, S. (2019). Evolutionary algorithms and neural networks. Studies in computational intelligence, 780(1), 43-53.
14. Chen, H., Ma, X., Mao, Y., & Ning, P. (2025). Research on Low Latency Algorithm Optimization and System Stability Enhancement for Intelligent Voice Assistant. Available at SSRN 5321721.
15. Yang, M., Cao, Q., Tong, L., & Shi, J. (2025, April). Reinforcement learning-based optimization strategy for online advertising budget allocation. In 2025 4th International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID) (pp. 115-118). IEEE.
16. Laili, Y., Guo, F., Ren, L., Li, X., Li, Y., & Zhang, L. (2021). Parallel scheduling of large-scale tasks for industrial cloud–edge collaboration. IEEE Internet of Things Journal, 10(4), 3231-3242.
17. Wu, Q., Shao, Y., Wang, J., & Sun, X. (2025). Learning Optimal Multimodal Information Bottleneck Representations. arXiv preprint arXiv:2505.19996.
18. Aguilar, A. (2023). Lowering Mean Time to Recovery (MTTR) in Responding to System Downtime or Outages: An Application of Lean Six Sigma Methodology. In 13th Annual International Conference on Industrial Engineering and Operations Management.
19. Wu, C., Zhang, F., Chen, H., & Zhu, J. (2025). Design and optimization of low power persistent logging system based on embedded Linux.
20. Gu, J., Narayanan, V., Wang, G., Luo, D., Jain, H., Lu, K., ... & Yao, L. (2020, November). Inverse design tool for asymmetrical self-rising surfaces with color texture. In Proceedings of the 5th Annual ACM Symposium on Computational Fabrication (pp. 1-12).
21. Jalali Khalil Abadi, Z., Mansouri, N., & Javidi, M. M. (2024). Deep reinforcement learning-based scheduling in distributed systems: a critical review. Knowledge and Information Systems, 66(10), 5709-5782.
22. Tan, L., Peng, Z., Liu, X., Wu, W., Liu, D., Zhao, R., & Jiang, H. (2025, February). Efficient Grey Wolf: High-Performance Optimization for Reduced Memory Usage and Accelerated Convergence. In 2025 5th International Conference on Consumer Electronics and Computer Engineering (ICCECE) (pp. 300-305). IEEE.
23. Hu, Z., Hu, Y., & Li, H. (2025). Multi-Task Temporal Fusion Transformer for Joint Sales and Inventory Forecasting in Amazon E-Commerce Supply Chain. arXiv preprint arXiv:2512.00370.