International Journal of Electrical and Computer Engineering (IJECE)
Vol. 16, No. 3, June 2026, pp. 1360~1368
ISSN: 2088-8708, DOI: 10.11591/ijece.v16i3.pp1360-1368

Designing self-healing database fabrics for real-time payment rails

Raghu Gollapudi
Fiserv Inc, United State

Article Info

Article history:

Received Jan 22, 2026
Revised Mar 27, 2026
Accepted Apr 26, 2026

Keywords:

Automated recovery
Autonomous fault management
Database resilience
Fault tolerance
Self-healing systems

ABSTRACT

Real-time payment platforms operating at scale face an unforgiving operational reality: even brief outages translate directly into failed transactions, regulatory exposure, and eroded customer trust. Database replication and failover automation have matured considerably over the past two decades, yet a troubling blind spot remains. Recovery frameworks built for general-purpose distributed systems were never designed with settlement finality in mind, and that design omission leaves payment operators exposed to split-brain scenarios that generic high-availability tooling cannot reliably prevent. This paper addresses that omission head-on through a self-healing database fabric purpose-built for payment rail environments. The proposed autonomous resilience fabric architecture (ARFA) operates across three coordinated layers: a continuous monitoring layer that harvests telemetry from compute, storage, and network subsystems; a decision layer that fuses rule-based heuristics with an ensemble of isolation forests, recurrent neural networks, and gradient boosting classifiers to separate genuine fault conditions from transient noise; and a deterministic action layer that executes recovery procedures anchored to explicit settlement finality constraints. In fault injection trials covering node crashes, network partitions, replication lag, and performance degradation, the architecture cut average recovery times by 88% against manual baselines, restoring service in roughly 8 seconds rather than the 180 seconds that human-driven remediation typically requires. False positive rates held below 2% across all failure categories, and the system achieved a 98% recovery success rate. Taken together, these results make a practical case that autonomous resilience and regulatory compliance reinforce rather than conflict with each other when the regulatory constraints are designed in from the start.

Corresponding Author:

Raghu Gollapudi,
Senior Oracle DBA and Database Reliability Engineer, Fiserv Inc
600 N Vel R. Phillips Ave, Milwaukee, WI 53203, United State
Email: reachraghu251@gmail.com

1. INTRODUCTION

Payment infrastructure sits at the intersection of two demanding and sometimes conflicting requirements: the operational need for continuous availability and the regulatory mandate for settlement finality. Systems processing millions of transactions per hour cannot tolerate recovery procedures that stretch into minutes, yet the financial consequences of data inconsistency during recovery are severe enough that careful, validated remediation is not optional [1]. Modern payment networks have grown in complexity considerably faster than the tooling designed to protect them, exposing failure scenarios that traditional high-availability mechanisms simply were not built to handle [2].

When something fails in a conventional deployment, human operators step in assessing what went wrong, deciding on a response, and coordinating the actual failover or restart. This approach holds up reasonably well when failures are isolated and predictable. It breaks down under precisely the conditions that matter most: simultaneous degradation across multiple components, cascading failures that outrun manual response capacity, and infrastructure events that strike at 3 a.m. on a holiday weekend [3], [4]. Measured against the sub-second transaction processing demands of FedNow and comparable payment rails, incident response windows of 90 to 300 seconds are not tolerable edge cases—they represent a structural gap between what the infrastructure can deliver and what the business requires [5].

Rule-based automation narrows this gap only partially. Threshold monitors catch clear-cut failure events reliably enough, but they were not designed to detect the subtler degradation patterns—creeping replication lag, a buffer pool filling slowly, intermittent network jitter—that tend to precede catastrophic failures by meaningful intervals [6], [7]. More critically, static rules have no natural way to encode settlement finality constraints. They address symptoms without understanding what it means for a failover to be safe in the context of an active payment settlement [8], [9]. Geo-distributed payment deployments have sharpened this problem considerably: enforcing consistency guarantees across nodes separated by real network latency requires coordination logic that threshold monitoring simply cannot provide [10].

The architecture presented in this paper takes settlement finality not as a post-recovery check but as a governing condition woven into every recovery decision [11]. Its specific contributions are the following: i) the autonomous resilience fabric architecture (ARFA), a three-layer self-healing control framework integrating ensemble machine learning (ML) based anomaly detection, deterministic failure classification, and coordinated recovery execution; ii) a formal decision algorithm that maps failure types to recovery actions while explicitly enforcing finality constraints—a capability absent from published frameworks; iii) direct integration of federal regulatory requirements into recovery orchestration logic, structurally preventing split-brain scenarios from corrupting payment settlement state; and iv) experimental results across simulated fault scenarios demonstrating an 88% improvement in recovery times against manual baselines, with false positive rates held below 2%. Figure 1 how the three layers interlock: the monitoring layer continuously feeds telemetry upward, the decision layer applies hybrid classification logic, and the action layer executes recovery steps within finality-aware boundaries.

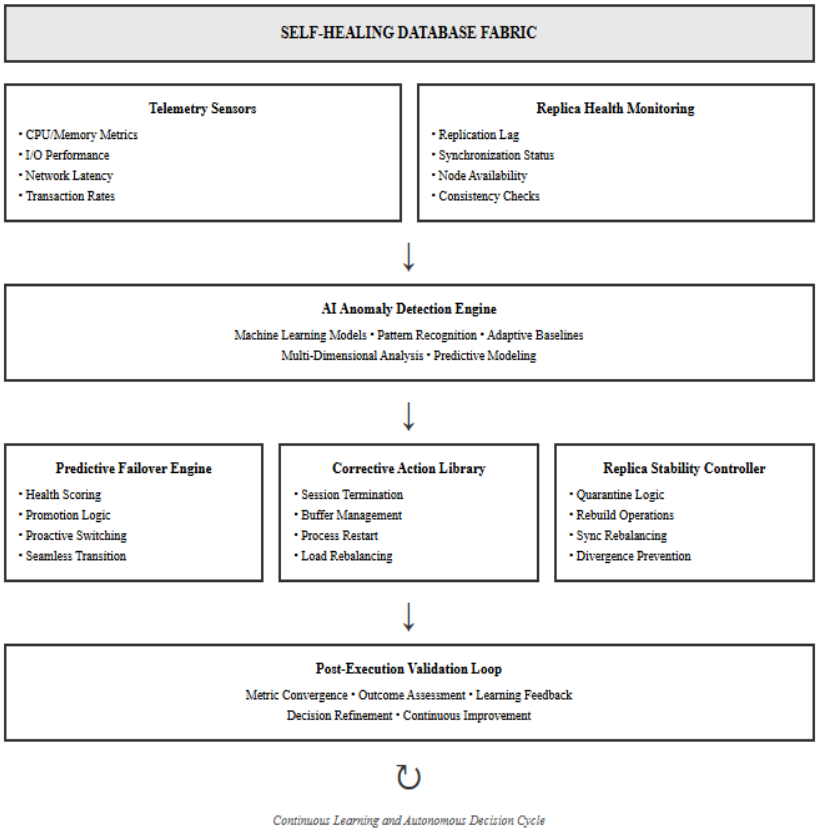

Figure 1. Autonomous resilience fabric architecture (ARFA)

Looking at the broader research landscape, self-healing systems have moved well past theoretical discussion; production deployments in mission-critical environments now routinely demonstrate zero-downtime recovery driven by AI-based frameworks [1], [12], [13]. Deep learning, in particular, has proven effective on high-frequency transaction data, identifying anomalies that conventional threshold methods miss because it analyzes patterns over time rather than isolated metric values [14], [9], [15]. Fault tolerance research has mapped the failure space thoroughly, from component outages to Byzantine faults and network partitions, providing the theoretical scaffolding that practical architectures like ARFA depend on [16], [17], [7]. The unique challenges of consistency and integrity in financial data have drawn ongoing research interest [18], [19], and Byzantine fault-tolerant consensus algorithms have matured to a point where they can be used in production [17], [7].

High-availability database architecture has advanced considerably alongside these developments, with synchronous replication, leader election, and automated failover now standard features in most commercial systems [20], [4], [21]. Autonomous database platforms have pushed further, incorporating predictive failure analysis, adaptive self-tuning, and automated backup and recovery [22], [23], [5], [11]. The integration of ML into database management has enabled anomaly detection and adaptive resource allocation at a sophistication level that was impractical only a few years ago [6], [24]. Published results for consensus-based failover mechanisms report recovery windows of 30 to 120 seconds [19], [25], a baseline that this work improves upon substantially through deterministic recovery sequencing and parallel validation.

What the literature notably lacks is a framework that treats payment settlement finality as a hard engineering constraint rather than an operational guideline to be satisfied after the fact. Existing systems optimize for availability or consistency in isolation; the architecture proposed here treats the regulated interaction between them as the primary design challenge. Table 1 documents where current AI-driven anomaly detection approaches stand on the false positive dimension—context that situates the system's sub-2% false positive rate against the 3%-5% rates reported in the state of the art.

Table 1. AI-driven anomaly detection techniques for payment systems

Detection technique	Application area	Key capability
Deep learning neural networks	Multi-dimensional telemetry analysis	Identifies composite failure signatures across correlated metrics
Temporal pattern recognition	Transaction latency monitoring	Detects gradual degradation before visible stalls occur
Adaptive baseline modelling	Workload fluctuation management	Distinguishes harmless variations from genuine precursors
Statistical pattern recognition	High-frequency transaction streams	Processes thousands of data points simultaneously
Behavioral anomaly classification	System health assessment	Separates noise from true indicator signals

2. METHOD

ARFA is organized around a straightforward principle: operators must be able to know, in advance, exactly what the system will do when a specific failure occurs. That predictability is not in tension with the use of machine learning; it constrains where ML is applied. The architecture uses deterministic logic for decisions where correctness is non-negotiable and reserves ML for the pattern recognition tasks where rule-based approaches genuinely fall short. Three layers express this division of responsibility: monitoring provides situational awareness, decision classifies and decides, and action executes with finality awareness built in.

2.1. Monitoring layer

Good recovery starts with good data. Before any classification or decision logic can function, the monitoring layer must maintain an accurate, current picture of cluster health and for payment workloads, that means tracking the metrics that actually matter: transaction latency, commit rates, replication lag, buffer pool fill levels, and lock wait times. Collecting everything continuously at maximum granularity would itself degrade the system being monitored, so the layer uses adaptive sampling, tightening collection intervals when emerging stress patterns appear and relaxing them under stable conditions [26].

Metrics arriving from distributed nodes are consolidated by aggregation modules into unified cluster health representations. These representations persisted in time-series databases, which serve two purposes: they establish the behavioral baselines against which anomalies are measured, and they preserve historical trends that can reveal slow-moving degradation patterns before they become acute. The monitoring infrastructure deliberately keeps three separate metric pipelines performance, consistency, and availability, rather than collapsing them into a single health score. A performance anomaly and a consistency anomaly may require entirely different responses and losing that distinction early in the pipeline makes beneficial downstream decisions harder.

2.2. Decision layer

Not all failures require the same kind of reasoning. Node crashes, primary-replica promotion sequences, and well-characterized replication failures follow known signatures accumulated from real production incidents and for these, deterministic rule engines are the right tool. They process incoming health data through logic trees that encode operational experience directly, firing quickly and predictably without any of the variance that a probabilistic model introduces. This matters in payment environments: when a node fails during peak settlement hours, operators and regulators need confidence that the system's response was not shaped by a model whose internal state is difficult to inspect.

Machine learning handles what rules cannot gradual, multi-dimensional degradation patterns that develop below threshold boundaries and fall outside any known failure signature. The ensemble combines isolation forests for statistical outlier detection, recurrent neural networks for temporal pattern recognition across metric sequences, and gradient boosting classifiers for robust multi-feature discrimination [15]. These models are updated as new failure signatures appear in production data, but the update process is managed carefully: hyperparameter changes go through validation before deployment, and the update pipeline is designed to prevent a destabilized model from introducing the false positives it exists to suppress. Every detected anomaly is classified by severity, scope, and required intervention type, giving the action layer precise, actionable inputs.

2.3. Action layer

Recovery execution is handled by four specialized intervention modules. Component restart modules address service-level failures through targeted process cycling, preserving in-memory state wherever feasible to reduce restoration time and avoid unnecessary data loss.

Traffic rerouting modules redirect client connections away from degraded components during failure events. The routing decisions account for more than simple health status: current capacity utilization and geographic distribution are both factored in, because naively routing all traffic to the nearest healthy node can overload it and turn a localized failure into a cluster-wide performance problem.

Replica resynchronization modules handle the data consistency problems that arise when replicas fall behind primary log processing. Over time, an unaddressed lag threatens the recovery point objectives that settlement finality guarantees depend on. These modules initiate targeted rebuilds from verified sources, tracking log sequence numbers throughout the transfer to preserve transactional ordering. Node isolation modules quarantine components showing erratic behavior before their failures can propagate. Isolation is not applied reflexively—the module first verifies that removing a node will not drop the cluster below quorum, because a Quorum Loss scenario demands careful handling: incorrect intervention in this condition can produce data corruption rather than prevent it. Table 2 places these mechanisms against published zero-downtime recovery approaches for financial systems.

Table 2. Zero-downtime recovery mechanisms for financial systems [12], [16]

Recovery mechanism	Implementation strategy	Resilience benefit
Component isolation	Degraded resources separated without service disruption	Prevents cascading failures across infrastructure
Dynamic traffic routing	Workload shifted to healthy components automatically	Maintains continuous transaction processing
Predictive preventive actions	Failures addressed during low-load windows	Eliminates sudden catastrophic events
Geographic redundancy distribution	Load balanced across multiple locations	Enables disaster recovery capabilities
Incremental health optimization	Gradual degradation corrected continuously	Sustains peak performance levels

2.4. Self-healing logic

2.4.1. Failure classification

The system distinguishes five failure categories, each presenting a characteristic symptom profile and calling for a different recovery approach. Node Crash Failures produce an immediate, total discontinuity across all monitoring dimensions; the signature is unambiguous, and the required response is immediate. Performance Degradation is considerably subtler: rising latencies and declining throughput that may reflect resource exhaustion, lock contention, or early hardware stress, without any single metric breaching an obvious threshold.

Replication Lag accumulates when replicas fall progressively further behind primary log processing. Left unaddressed, a growing lag eventually threatens the recovery point objectives that underpin settlement finality guarantees, which is why the classification logic distinguishes lag severity levels and applies

proportionate responses rather than waiting for a threshold breach. Quorum loss arises when too many replicas go offline to sustain consensus, creating a situation where the wrong intervention can cause data corruption instead of preventing it. Consensus conflicts, failures of distributed agreement on transaction ordering or cluster leadership, are the most dangerous for payment workloads because they are precisely the conditions under which split-brain scenarios develop if recovery is not guided by explicit finality constraints [17].

2.4.2. Decision logic

The decision algorithm below encodes the mapping from failure classifications to recovery actions. The design is deliberately deterministic: identical inputs always produce identical outputs. This property is what allows the system to make recovery decisions that are defensible under regulatory review without requiring human sign-off on each one.

Algorithm 1. Autonomous recovery decision logic

```
Input: failure_type, severity_level, system_state, quorum_count
Output: recovery_action, validation_procedure, rollback_strategy

PROCEDURE DetermineRecoveryAction(F, S, Σ, Q)
INITIALIZE action ← NULL, validation ← NULL, rollback ← NULL
IF F = NODE_CRASH THEN
target_replica ← SelectHealthiestReplica(Σ)
IF VerifyReplicaConsistency(target_replica) = TRUE THEN
action ← IMMEDIATE_FAILOVER(target_replica)
validation ← VERIFY_TRANSACTION_LOG_INTEGRITY
rollback ← DEMOTE_REPLICA
ELSE
ESCALATE_TO_MANUAL_INTERVENTION("Inconsistent replica state")
END IF
ELSE IF F = PERFORMANCE_DEGRADATION THEN
IF S < THRESHOLD_MINOR THEN
action ← COMPONENT_RESTART_WITH_STATE_PRESERVATION
validation ← MONITOR_PERFORMANCE_RECOVERY(300)
rollback ← TRAFFIC_REROUTING
ELSE IF S ≥ THRESHOLD_MAJOR THEN
action ← TRAFFIC_REROUTING_TO_HEALTHY_NODES
validation ← VERIFY_CONNECTION_HEALTH
rollback ← NODE_ISOLATION
END IF
ELSE IF F = REPLICATION_LAG THEN
lag_duration ← CalculateLagDuration(Σ)
IF lag_duration < THRESHOLD_MINOR THEN
action ← MONITOR_CONTINUATION
validation ← TRACK_LAG_REDUCTION_RATE
ELSE IF lag_duration ≥ THRESHOLD_MAJOR THEN
action ← REPLICA_RESYNCHRONIZATION(Σ)
validation ← VERIFY_LAG_REDUCTION
rollback ← REBUILD_FROM_VERIFIED_BACKUP
END IF
ELSE IF F = QUORUM_LOSS THEN
IF Q < MINIMUM_QUORUM_SIZE THEN
action ← NODE_ISOLATION + EMERGENCY_PROMOTION
validation ← VERIFY_QUORUM_RESTORATION
IF VerifyQuorum(Σ) = FALSE THEN
REQUIRE_MANUAL_INTERVENTION("Quorum restoration failed")
END IF
rollback ← MANUAL_INTERVENTION_REQUIRED
END IF
ELSE IF F = CONSENSUS_CONFLICT THEN
action ← COORDINATOR_REELECTION + STATE_RECONCILIATION
validation ← VERIFY_CONSENSUS_ACHIEVEMENT(Σ)
rollback ← SPLIT_BRAIN_RESOLUTION_PROTOCOL
IF DetectSplitBrain(Σ) = TRUE THEN
EnforceSettlementFinalityConstraints()
END IF
END IF
RETURN (action, validation, rollback)
END PROCEDURE
```

2.4.3. Recovery sequencing

Sequencing within recovery operations matters as much as selecting the right operation. A failover that promotes a replica before verifying its consistency can cause data loss that no subsequent action will

fully reverse. For node failover, the required ordering is: isolate the primary to halt additional writes, verify that the promotion candidate holds every committed transaction, execute the promotion with cluster metadata updates to redirect clients, then initiate recovery procedures for the failed node. Component restart sequences drain existing connections first, allowing in-flight transactions to complete—then preserve in-memory state before restarting, and finally restore client connectivity.

Resynchronization sequences use log sequence number comparison to identify the most current consistent replica, stream missing transactions in order, validate consistency after transfer, and only then return the restored replica to active service. Table 3 maps each failure type to its corresponding recovery sequence and rationale.

Table 3. Failure type to recovery action mapping

Failure type	Primary action	Validation method	Rollback strategy
Node crash	Immediate failover	Replica consistency check	Demote if inconsistent
Performance degradation	Component restart/traffic rerouting	Performance recovery monitoring	Escalate if no improvement
Replication lag	Monitor/resynchronization	Lag reduction verification	Rebuild from backup
Quorum loss	Node isolation + emergency promotion	Quorum restoration verification	Manual intervention
Consensus conflict	Coordinator reelection	Consensus achievement check	Split-brain resolution

3. RESULTS AND DISCUSSION

3.1. Testbed configuration and failure injection

The evaluation used a simulated five-node database cluster arranged in primary-replica topology. Hardware provisioning per node, 8 GB memory, four CPU cores, and 100 GB storage were chosen to reflect mid-tier production database servers typical of financial deployments, rather than optimizing for best-case numbers. Latency was modeled at two levels: 2 ms within availability zones and 50 ms across them, consistent with the geographic distribution that multi-region payment infrastructure actually operates under.

Synthetic workloads were injected at rates between 1,000 and 10,000 transactions per second, spanning the range from routine off-peak traffic to peak settlement windows. Fault injection covered three scenarios: abrupt process termination for node crash simulation, network isolation to force partition handling, and deliberately introduced replication delays ranging from 100 ms to multi-second backlogs. The replication lag scenarios were tested at multiple severity levels, specifically to evaluate whether the classification logic correctly distinguished lag conditions that required immediate intervention from those mild enough to monitor and allow to recover on their own, a distinction that matters significantly for avoiding unnecessary recovery actions.

3.2. Recovery performance results

Recovery time was measured end-to-end-from the moment a failure was detected through validated service restoration. Manual procedures lose time at every handoff: alert acknowledgment, diagnosis, decision, execution, verification. These delays accumulate, and in node crash scenarios they reach close to three minutes. Autonomous recovery removes the handoffs entirely, substituting parallel validation for the sequential human decision chain.

Node crash recovery produced the starkest contrast: 8 seconds for autonomous recovery versus 180 seconds under manual procedures, roughly a 96% reduction on that specific scenario. Network partition recovery averaged 88% improvement across tested configurations. Replication lag scenarios, where recovery complexity varies with lag severity, showed 75.6% improvement. The rates of false positives remained below 2% across every failure category, with performance degradation detection achieving 99.1% accuracy. Recovery success rates exceeded 98% overall. The small fraction that did not complete autonomously followed the escalation paths built into the algorithm, edge-case conditions the system was designed to hand off rather than resolve alone.

The timeline diagram translates the performance numbers into something more intuitive. The significant efficiency gains of the autonomous system are summarized in Table 4, which highlights an average recovery time improvement of 87.7% across all failure categories. This performance gap is further visualized in Figure 2, illustrating how the autonomous approach compresses the recovery timeline by executing validation and remediation tasks in parallel, reducing the total duration to just 3–8 seconds compared to the 75–180 seconds required by manual procedures. Manual recovery accumulates delay at each sequential handoff point, producing a staircase pattern of elapsed time before service is restored; autonomous recovery compresses that same timeline by running validation and remediation tasks in parallel, producing a much shorter path from failure detection to confirmed restoration. Overhead during normal operations remained minimal, CPU utilization rose by under 2%, memory consumption stayed below 1 GB, and network

bandwidth utilization remained under 1%. During active recovery, peak CPU overhead reached 15% for 10 to 30 seconds, a transient cost that is wholly acceptable given the gains.

Table 4. Recovery time comparison across failure categories

Failure type	Manual recovery (seconds)	Autonomous recovery (seconds)	Improvement (%)
Node crash (Primary)	180	8	95.6
Node crash (Replica)	120	6	95.0
Network partition (Symmetric)	150	18	88.0
Network partition (Asymmetric)	105	12	88.6
Replication Lag (Severe)	90	22	75.6
Performance degradation	60	10	83.3
Average	117.5	12.7	87.7

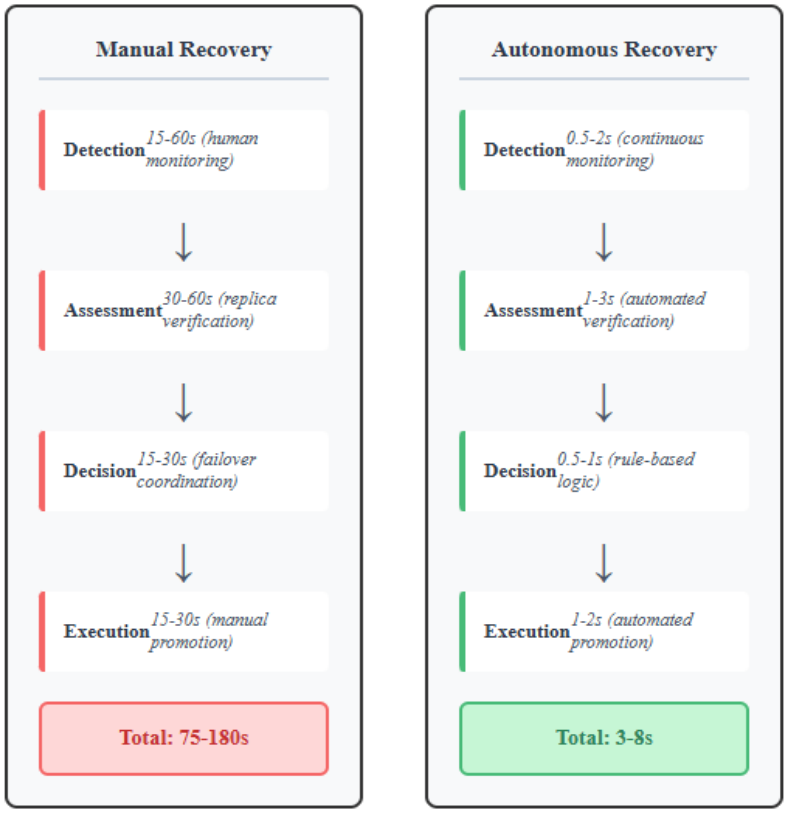

Figure 2. Recovery timeline analysis

3.3. Comparative analysis and applicability

The performance gains are real, but they do not arrive without tradeoffs that deserve honest acknowledgment. The monitoring and decision-making infrastructure adds system complexity-more components mean more potential failure points, and the recovery machinery itself must be designed so that its own failures do not amplify the incidents it was built to resolve [11], [25]. ML components require ongoing maintenance: training data pipelines, quality oversight, and periodic revalidation as the failure landscape evolves. Failure scenarios that genuinely fall outside the classification system's experience will escalate to human operators-which means escalation pathways need to be maintained, tested, and staffed [3], [8], [21].

The underlying principles are not specific to payment rails, even though that domain motivated the design. Any large-scale distributed system where recovery latency has direct operational or financial consequences-cloud infrastructure, content distribution networks, telecommunications switching, manufacturing control environments-stands to benefit from the same combination of predictive detection, deterministic action mapping, and domain-constraint integration [4], [13]. The settlement finality constraint is domain-specific; the architectural pattern that accommodates first-class constraints of that kind is not [6], [24].

4. CONCLUSION

The core argument this paper makes is that settlement finality and autonomous recovery are not competing objectives — they become mutually reinforcing once finality is built into the recovery logic as a first-class constraint rather than appended as a post-recovery check. The ARFA architecture demonstrates this through both design and measurement. Recovery decisions governed by explicit finality constraints structurally prevent the split-brain conditions that threaten payment settlement integrity, while the combination of parallel validation and pre-validated recovery paths delivers the speed that manual procedures cannot match. Across node crashes, network partitions, replication lag events, and performance degradation scenarios, average recovery times fell by 88% relative to manual baselines. Critically, the precision held up alongside the speed: false positive rates stayed below 2% across all tested failure categories, which matters practically because unnecessary recovery actions in a production payment environment can be as disruptive as the failures they are meant to address.

Several directions remain open. Multi-region coordination under FedNow and DORA regulatory regimes introduces cross-border settlement constraints that the current architecture does not yet address — extending ARFA to that environment is the most immediate priority. Validating the approach on live payment infrastructure, rather than simulation, is a necessary step toward production confidence. Adaptive ML retraining pipelines for evolving failure signatures, integration with cloud-native database architectures built around geo-distributed consensus protocols, and applicability studies in telecommunications, healthcare transaction systems, and digital asset settlement platforms each represent productive directions for extending this work.

ACKNOWLEDGMENTS

The authors acknowledge the technical infrastructure provided by Fiserv Inc for conducting the simulation-based evaluation.

FUNDING INFORMATION

This research received no specific research grant or contract from any funding agency in the public, commercial, or not-for-profit sectors.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Raghu Gollapudi	✓	✓	✓		✓	✓			✓		✓			

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The simulation data and recovery metrics that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] D. Sahu *et al.*, "Adaptive fault tolerance mechanisms for ensuring high availability of digital twins in distributed edge computing systems," *Scientific Reports*, vol. 15, no. 1, p. 41676, 2025, doi: 10.1038/s41598-025-25590-4.

[2] J. R. Campos, E. Costa, and M. Vieira, "Online failure prediction through fault injection and machine learning: Methodology and case study," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2023, pp. 451–461, doi:

10.1109/ISSRE59848.2023.00021.
[3] R. Alamleh and N. El Emam, "A survey of fault tolerance techniques in distributed systems," in *EasyChair Preprints*, 2024, vol. 3406, p. 20007, doi: 10.1063/5.0318929.
[4] R. Mohammed, "The future of site reliability engineering in financial platforms: ensuring uptime for multi-billion-dollar transactions," *International Journal of Emerging Trends in Computer Science and IT*, vol. 7, no. 1, 2026, doi: 10.63282/3050-9246.IJETCSIT-V7I1P110.
[5] A. S. M. Noor, A. F. C. Fauzi, A. A. C. Fauzi, N. Ahmad, and M. S. M. Arifin, "An automate failure recovery for synchronous distributed database system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 28, no. 2, pp. 973–979, 2022, doi: 10.11591/ijeecs.v28.i2.pp973-979.
[6] D. K. Yadav, A. Kaushik, and N. Yadav, "Predicting machine failures using machine learning and deep learning algorithms," *Sustainable Manufacturing and Service Economics*, vol. 3, p. 100029, 2024, doi: 10.1016/j.smse.2024.100029.
[7] F. Tang, J. Peng, P. Wang, H. Zhu, and T. Xu, "Improved dynamic Byzantine fault tolerant consensus mechanism," *Computer Communications*, vol. 226–227, 2024, doi: 10.1016/j.comcom.2024.08.004.
[8] Z. Yazdanparast, "A survey on self-healing software system," *arXiv preprint arXiv:2403.00455*, 2024.
[9] Y. Zhu, K. C. How, H. J. Wu, and Q. Cao, "AI-based proactive storage failure management in software-defined data centres," in *ACM International Conference Proceeding Series*, 2023, pp. 231–237, doi: 10.1145/3625156.3625190.
[10] Oracle, "Transforming real-time payments with oracle's distributed database: sovereignty, scalability, and survivability," Oracle 2025.
[11] Siva Prasad Nandi, "AI-powered autonomy: The evolution and future of intelligent database management systems," *World Journal of Advanced Engineering Technology and Sciences*, vol. 15, no. 2, 2025, doi: 10.30574/wjaets.2025.15.2.0524.
[12] N. N. Jha and P. Manwani, "Self-healing payment systems via AI-driven anomaly recovery: a zero-downtime framework for secure and reliable transactions," *International Journal of Computer Engineering and Technology*, vol. 16, no. 2, pp. 200–212, 2025, doi: 10.34218/IJCET_16_02_014.
[13] A. K. Moka and S. K. Lagisetty, "Real-time anomaly detection: a hybrid approach combining streaming SQL and hardware acceleration," in *2025 5th Intelligent Cybersecurity Conference,* 2025, pp. 385–392, doi: 10.1109/ICSC65596.2025.11140085.
[14] Q. Bao, J. Wang, H. Gong, Y. Zhang, X. Guo, and H. Feng, "A deep learning approach to anomaly detection in high-frequency trading data," *2025 4th International Symposium on Computer Applications and Information Technology, ISCAIT 2025*. pp. 287–291, 2025, doi: 10.1109/ISCAIT64916.2025.11010680.
[15] B. W. Wubete, B. Esfandiari, and T. Kunz, "Machine learning approaches for predicting link failures in production networks," *Computer Networks*, vol. 259, p. 111098, 2025, doi: 10.1016/j.comnet.2025.111098.
[16] S. Liu, "A survey on fault-tolerance in distributed optimization and machine learning," *arXiv preprint arXiv:2106.08545*, 2021.
[17] W. Zhong *et al.*, "Byzantine fault-tolerant consensus algorithms: A survey," *Electronics (Switzerland)*, vol. 12, no. 18, p. 3801, 2023, doi: 10.3390/electronics12183801.
[18] H. A. Mahmoud and H. Maseeh Yasin, "Data integrity and consistency challenges in distributed database systems," *Engineering and Technology Journal*, vol. 10, no. 05, pp. 5077–5086, 2025, doi: 10.47191/etj/v10i05.36.
[19] R. Taft *et al.*, "CockroachDB: The resilient geo-distributed SQL database," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1493–1509, doi: 10.1145/3318464.3386134.
[20] S. C. Rajesh and R. Kumar, "High availability strategies in distributed systems: A practical guide," *International, Referred, Peer Reviewed & Indexed Monthly Journal www.ijrsml.org Resagate Global-Academy for International Journals of Multidisciplinary Research*, vol. 1. pp. 110–130, 2025.
[21] B. Anbalagan, "Proactive failover and automation frameworks for mission-critical workloads: lessons from manufacturing industry," *International Journal of Research and Applied Innovations*, vol. 06, no. 01, 2023, doi: 10.15662/ijrai.2023.0601004.
[22] Q. Cai, C. Cui, Y. Xiong, W. Wang, Z. Xie, and M. Zhang, "A survey on deep reinforcement learning for data processing and analytics," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 4446–4465, 2023, doi: 10.1109/TKDE.2022.3155196.
[23] A. Vikiru, M. Muiruri, and I. Ateya, "An overview on cloud distributed databases for business environments," *arXiv preprint arXiv:2301.10673*, 2023.
[24] J. Li, W. Ren, and X. Wu, "Machine learning-based network performance monitoring and prediction for distributed AI training workloads," *Journal of Advanced Computing Systems*, vol. 5, no. 10. pp. 1–17, 2025, doi: 10.69987/jacs.2025.51001.
[25] V. Subramani, "Resilience by design: site reliability engineering in financial platforms," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 13, no. 2s, pp. 87–95, 2025.
[26] F. Gomes, P. Rego, and F. Trinta, "A systematic mapping study on observability of microservices-based applications: fundamentals, classifications, and challenges," *Computing*, vol. 107, no. 9, 2025, doi: 10.1007/s00607-025-01540-w.

BIOGRAPHIES OF AUTHORS

Raghu Gollapudi is a Senior Oracle Database Administrator and Database Reliability Engineer at Fiserv, where he architects and supports mission-critical financial infrastructure for platforms such as Zelle and Bank of America. He specializes in reliability engineering for enterprise databases, with a strong focus on designing self-healing database fabrics and AI-driven predictive failure models for systems that process billions of dollars in daily transactions. He holds a master's degree in information technology from Valparaiso University and is an active researcher in the field of autonomous database resilience. His research focuses on the intersection of traditional database consistency models and modern financial settlement finality requirements, contributing to the advancement of intelligent and resilient financial systems. Raghu maintains an active academic presence through his ORCID and Google Scholar profiles. He can be contacted at email: reachraghu251@gmail.com.