Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 3, Issue 2

Open Access | PP: 69-76

DOI: https://doi.org/10.70315/uloap.ulirs.2026.0302013

Architectural Solutions for Ensuring Fault Tolerance of Cloud IT Infrastructures

Abdul Nadeem Mohammed

Full Stack Developer, Techlance Solutions Inc, Michigan, USA.

Abstract

The article explores architectural solutions for ensuring fault tolerance in cloud IT infrastructures by analyzing how resilience emerges from the interaction of predictive, optimization, and orchestration mechanisms across multiple layers of the cloud stack. The article used a structured analytical review methodology based on comparative synthesis of peer-reviewed studies, extracting fault models, operational mechanisms, and optimization objectives, and integrating them into a unified cross-layer architectural framework. The main results are that fault tolerance in cloud systems is best understood as a closed-loop control process linking failure prediction, decision-making (e.g., placement and scheduling), and execution-level recovery (e.g., migration, checkpointing, and repair), and that different architectural layers optimize distinct but interdependent reliability objectives. The synthesis further reveals key trade-offs between availability and cost, prediction accuracy and latency, replication overhead and repair efficiency, and resilience and orchestration complexity. The article will be useful to researchers and practitioners in cloud computing, distributed systems, and reliability engineering by providing a coherent architectural perspective that connects fragmented fault-tolerance approaches into an integrated design framework for improving system-level reliability under dynamic and heterogeneous conditions.

Keywords: *Cloud Fault Tolerance; Cloud Architecture; Reliability Engineering; Availability Optimization; VM Placement; Task Scheduling; Microservices Orchestration; Erasure Coding; Failure Prediction; Cloud-Native Systems.*

INTRODUCTION

Fault tolerance has become a central architectural requirement for contemporary cloud IT infrastructures because cloud services increasingly underpin time-sensitive business processes, safety-relevant applications, and large-scale data management workloads. The literature consistently frames service availability and reliability as economically and operationally consequential: outages can originate from failures across interdependent layers (application components, virtual machines, hosts, networks), and even well-provisioned systems remain vulnerable to correlated faults and dynamic workload shifts. High-availability targets in such environments commonly reach 99.999% ("five nines"), while service downtime in large-scale web systems can incur losses of up to $300,000 per hour, emphasizing that fault tolerance is a critical economic constraint on cloud architecture design [1].

At the same time, cloud architectures are evolving in two directions that intensify fault-tolerance challenges while also creating new design opportunities. Multi-cloud deployments distribute workloads across different providers, reducing reliance on a single vendor. The platforms involved vary in interfaces, resource models, and configuration constraints. This results in portability issues and added operational overhead. Effects on resilience vary with how cross-platform interactions are handled in practice [2]. Second, cloud-native engineering emphasizes end-to-end automation and resilience across the full stack, expanding the fault-tolerance design space beyond classic redundancy to include observability-driven operations and systematic stack extensibility [3].

The purpose of this study is to develop a structured architectural synthesis of fault-tolerance mechanisms in cloud IT infrastructures, with emphasis on how resilience emerges from the interaction of predictive, optimization, and orchestration layers. The study organizes fault-tolerance mechanisms by architectural layer and operational function, including placement, scheduling, recovery, and storage repair. It also examines how outputs from predictive models

Citation: Abdul Nadeem Mohammed, "Architectural Solutions for Ensuring Fault Tolerance of Cloud IT Infrastructures", Universal Library of Innovative Research and Studies, 2026; 3(2): 69-76. DOI: https://doi.org/10.70315/uloap.ulirs.2026.0302013.

are translated into system-level actions such as migration, checkpointing, or topology reconfiguration. In contrast to studies centered on individual mechanisms (e.g., VM placement, task scheduling, or storage repair), the present work brings these elements together by linking predictive components, decision logic, and execution processes within a single analytical view, allowing trade-offs between availability, cost, scalability, and operational complexity to be examined in combination.

The study proceeds from the assumption that fault tolerance at the system level arises from the interaction of replication and prediction within a feedback-oriented structure connecting sensing, decision, and execution. Under these conditions, methods confined to a single layer tend to show reduced effectiveness when applied independently, particularly in multi-cloud and cloud-native settings with heterogeneous resources and continuously changing workloads.

MATERIALS AND METHODS

This study adopts a structured analytical review paradigm in which fault-tolerance approaches are as components of an integrated architectural system. Methodologically, the work is based on comparative synthesis and abstraction of peer-reviewed studies, combining elements of qualitative literature analysis with cross-layer architectural modeling. Each source is analyzed to extract (i) the underlying fault model, (ii) the operational mechanism (e.g., placement, prediction, scheduling, recovery, or repair), and (iii) the optimization objectives and performance trade-offs associated with the approach. These elements are then mapped onto a unified architectural framework that organizes fault-tolerance mechanisms according to their functional roles across the cloud stack and their interdependencies.

Yanal Alahmad and Anjali Agarwal introduced a multi-objective dynamic VM placement framework (formulated as an integer nonlinear program and solved with heuristic optimization) to improve application service availability while also reducing power consumption and resource waste, complemented by deep-learning-based failure prediction for proactive protection [1]. Juncal Alonso and colleagues conducted a PRISMA-informed systematic literature review (SLR) of multi-cloud native applications, classifying definitions, architectural models, and DevOps lifecycle challenges across studies published between 2011 and 2021 [2]. Jian Lin and co-authors proposed a cloud-native "light-cone" dimensional analysis model and used it to analyze extensibility challenges in enterprise cloud-native service stacks, linking architecture design to properties such as resiliency and observability, and reporting implementation-oriented insights via an extensible stack (OMStack) [3]. Sudheer Mangalampalli and co-authors designed a fault-tolerant, trust-oriented task scheduling algorithm using Harris Hawks Optimization and evaluated it via cloud simulation against alternative metaheuristics on synthetic distributions and real workload traces [4]. Joonseok Park and colleagues proposed a "private cloud bespoke orchestrator" emphasizing Infrastructure-as-Code (IaC), reuse of VM specifications via software product line concepts, and similarity-driven VM recovery, demonstrated through a case study on OpenStack [5].

Mikael Sabuhi and co-authors introduced Micro-FL, a fault-tolerant microservice-based federated learning platform evaluated on Google Cloud Platform using a managed Kubernetes environment, with experiments assessing behavior under injected faults and varying client scale [6]. Muhammad Asim Shahid and co-authors proposed a hybrid approach combining machine-learning-based fault classification/prediction and delta-checkpointing, evaluated using a secondary HPC fault dataset from ETH Zurich (trace data during fault injection) and a primary dataset generated via virtual machines [7]. Tengku Nazmi Tengku Asmawi and co-authors performed a comparative evaluation of traditional ML and deep learning models for job/task failure prediction using Google Cluster Traces, including performance metrics, feature importance, and scalability analysis [8]. Yangwenting Xu and colleagues proposed a whale-optimization-based, multi-objective method for geo-distributed erasure-coded storage repair, integrating loop elimination with shortest-path reasoning and reporting simulation-based reductions in repair latency and link-utilization imbalance [9]. Li Zhu and co-authors proposed a proactive reliability-aware failure recovery method for cloud-based automatic train supervision services using deep reinforcement learning (actor–critic with LSTM and Age of Information), evaluated via simulation against baseline recovery approaches [10].

Despite this breadth, a recurring limitation across the body of work is architectural fragmentation: individual mechanisms are often optimized within one layer (e.g., VM placement, task scheduling, microservice recovery, or storage repair) while the cross-layer coupling that determines end-to-end resilience is treated implicitly or delegated to operational practice. Multi-cloud and cloud-native paradigms amplify that gap by increasing heterogeneity and the number of control planes that must coordinate to prevent small faults from escalating into service-level outages [2–3]. In addition, several studies highlight the need for proactive approaches (prediction, preventive actions) but do not consistently formalize how prediction outputs are translated into architecturally safe actuation (placement changes, reconfiguration, checkpointing, or repair topology decisions) under uncertainty and cost constraints [1,7–10]. The present article addresses this limitation by synthesizing the reviewed evidence into a cohesive set of architectural solution categories that explicitly connect predictive models, optimization-based decision layers, and orchestration/repair workflows into an end-to-end fault-tolerance perspective grounded in the corpus.

RESULTS

The reviewed literature converges on an architectural view in which fault tolerance emerges from closed-loop control

across infrastructure layers: sensing and prediction feed decision and placement logic, which triggers orchestrated recovery or repair actions, which in turn reshape system state and future risk. This closed-loop perspective is explicit in works that combine prediction with proactive action (e.g., failure prediction informing protective actions or reducing recovery time), and it is implicit in optimization-driven placement/scheduling research that targets availability while managing power, resource waste, and fault exposure [1,7–10]. Figure 1 represents fault tolerance in cloud systems as a closed-loop control process in which system resilience emerges from continuous interaction between prediction, decision-making, and execution layers.

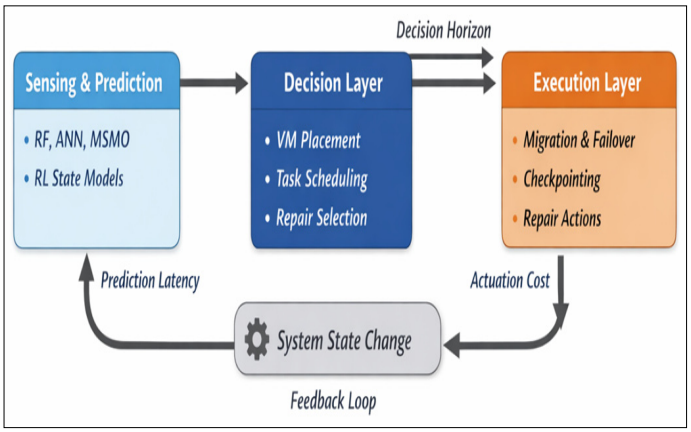

Figure 1. Closed-loop architectural model of fault tolerance in cloud IT infrastructures (the author's illustration)

On the left side of Figure 1, the sensing and prediction stage applies machine learning and state-aware models (e.g., RF, ANN, MSMO, RL) to system telemetry in order to identify conditions associated with upcoming failures, with performance limited in part by prediction latency. The resulting outputs are passed to a decision layer, where actions such as VM placement, task scheduling, or repair strategy selection are chosen within a given decision horizon. At the execution level, these actions take the form of operations including migration, checkpointing, and microservice failover, each introducing additional actuation overhead. Updates in system state are then returned to the prediction stage, allowing subsequent decisions to be adjusted based on observed outcomes. This closed-loop structure highlights that fault tolerance is a dynamic, feedback-driven process governed by trade-offs between timeliness, decision accuracy, and execution overhead.

A first cluster of architectural solutions centers on availability-aware resource placement and scheduling as fault-containment. In dynamic VM placement, service availability is treated as an objective that must be balanced against energy efficiency and resource utilization; the proposed MoVPAAC framework structures the problem into modules for deployment, proactive failure detection, and reconfiguration during scaling or failure events, indicating an architectural separation between (i) optimization logic and (ii) monitoring/prediction-driven triggers for adaptation [1]. At the infrastructure level, server availability is defined as

$$AV(s_j) = \frac{MTTF_j}{MTTF_j + MTTR_j}. \quad (1)$$

In formula (1), $MTTF_j$ and $MTTR_j$ denote the mean time to failure and mean time to repair of server s_j, respectively. Application-level availability is then computed as a multiplicative composition across all required functionalities:

$$AV_{app} = \prod_{f \in F} AV_{func_f}. \quad (2)$$

In formula (2), AV_{app} is overall availability of the application F is the set of functionalities required for the application to work, AV_{func_f} and is the availability of a single functionality f. Each functionality depends on one or more virtual machines, and failure is expressed as the complement:

$$Fail_{s_j} = 1 - AV(s_j). \quad (3)$$

Formula (3) has $Fail_{s_j}$ as probability that the server is down, s_j as VM host, and $AV(s_j)$ as availability of that server.

This formulation highlights that availability is dependent on placement decisions and fault-domain separation. Empirical scenarios demonstrate that baseline deployment configurations may yield availability levels around 0.90, while migration of virtual machines across heterogeneous servers can increase availability to approximately 0.94, and the introduction of standby virtual machines further improves availability to approximately 0.95. These results indicate that availability-aware placement operates as a proactive fault-tolerance mechanism, where deployment topology, redundancy strategy, and resource distribution jointly determine resilience, while also introducing trade-offs between availability, resource utilization, and operational cost. This optimization introduces a trade-off between redundancy and resource use. Increasing availability through additional replicas or standby virtual machines (e.g., dispersed placement) improves fault tolerance but raises operational cost and can limit application admissibility. At the same time, reducing resource allocation lowers cost but increases the risk of SLA violations and service disruption.

Figure 2 depicts an availability-aware virtual machine (VM) placement architecture in which applications are decomposed into multiple VM instances distributed across heterogeneous physical servers, with fault tolerance achieved through a combination of primary deployments and standby protection.

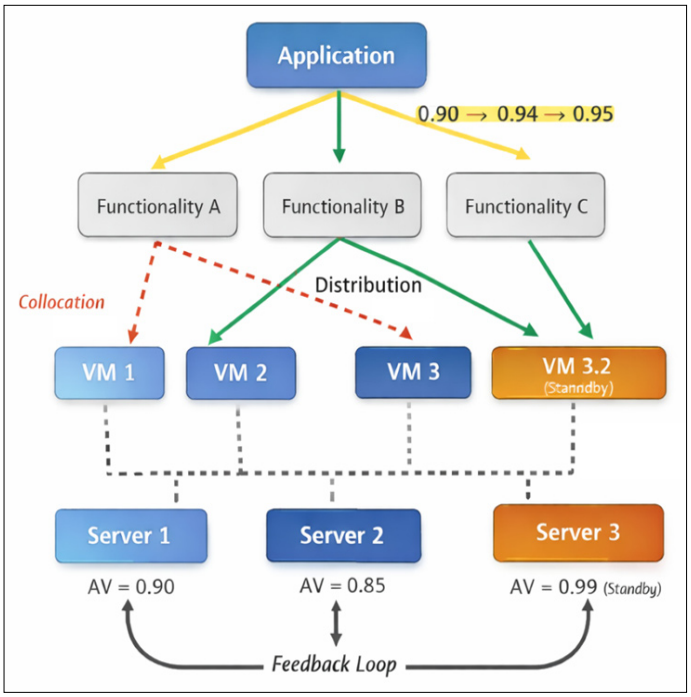

Figure 2. Availability propagation and optimization through VM placement and redundancy strategies (the author's illustration)

Figure 2 shows how application components are deployed on separate virtual machines (VM1, VM2, VM3.2), with placement spread across different physical servers instead of being concentrated on a single host. This distribution reduces dependence on a single failure domain. In addition, standby VMs are provisioned as inactive replicas that can be brought online when required. The figure also reflects that availability at the application level depends on how these VMs are arranged across failure domains, as well as on mechanisms such as migration, replication, and standby activation, which determine how the system reacts to node-level disruptions

At the task scheduling layer, fault tolerance is further handled through policies that incorporate trust and failure-related factors, where scheduling decisions influence the likelihood of SLA violations under changing workload conditions. The proposed trust-based scheduler assigns priorities to both tasks and virtual machines and evaluates system behavior across multiple QoS-oriented metrics, including makespan, failure rate, availability, and turnaround efficiency. Quantitative evaluation demonstrates that such scheduling mechanisms can reduce makespan by approximately 24–33%, decrease failure rates by around 60–65%, and improve availability by approximately 28–35% compared to conventional metaheuristic approaches [4]. These results indicate that scheduling policies act as an active resilience mechanism, shaping system reliability through workload distribution, priority control, and deadline-aware execution under heterogeneous cloud conditions. The diagram on Figure 3 represents scheduling as an active resilience mechanism, where task allocation decisions directly influence system reliability.

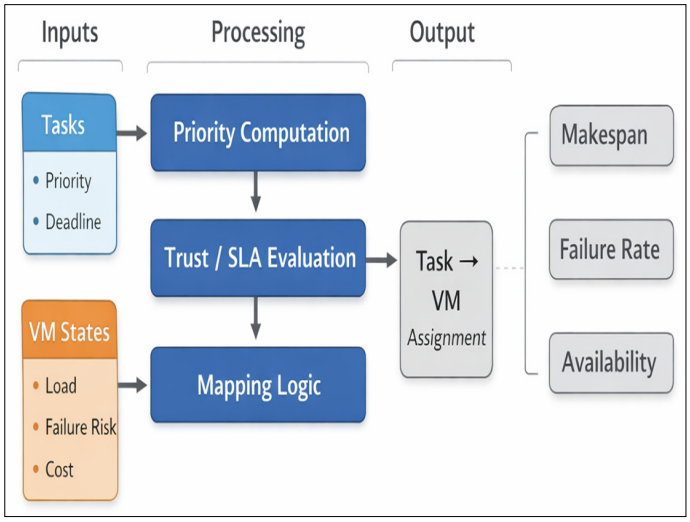

Figure 3. The author's illustration of fault-tolerant task scheduling pipeline as a resilience mechanism

Incoming tasks in Figure 3, defined by parameters such as priority and deadline, are evaluated together with the current state of virtual machines, including load level, estimated failure risk, and operating cost. At the processing stage, the scheduler computes a composite priority score and checks SLA-related attributes, such as VM reliability and historical success rates. These inputs guide task assignment to virtual machines, with consideration given to both execution efficiency and the likelihood of failure. In applied settings, this results in placements that steer workloads away from overloaded or unstable nodes, instead of selecting resources based on performance alone. Makespan, failure frequency, and availability highlight the underlying trade-offs: configurations that reduce execution time may increase the probability of failure, whereas more conservative allocations tend to preserve service continuity at the expense of longer runtimes.

A second cluster focuses on proactive failure prediction as a component of closed-loop resilience control. Empirical analysis based on Google Cluster Trace data illustrates the scale and structure of the prediction task, involving on the order of tens of millions of jobs (over 48 million records) and failure frequencies approaching 30%, which imposes requirements on models to process high-dimensional, temporally structured, and class-imbalanced inputs [8]. In this setting, neural-network-based approaches, including standard feedforward architectures, have been reported to reach prediction accuracy levels near 93%, allowing the identification of tasks with elevated failure likelihood. However, model selection is constrained by inference latency and scalability requirements: lightweight models such as logistic regression (LR) can perform inference over millions of tasks in sub-second time, making them more suitable for real-time scheduling and control-plane integration [8]. These results indicate that predictive fault tolerance must be understood as a pipeline where the operational feasibility of the model is determined by the trade-off between accuracy, computational overhead, and response time within large-

scale cloud environments. Figure 4 presents a closed-loop predictive fault-tolerance pipeline for large-scale cloud environments, showing how failure management evolves from raw data to automated system actions.

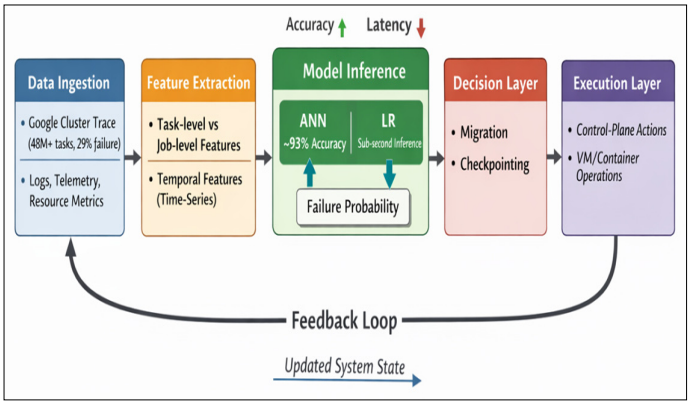

Figure 4. Predictive fault-tolerance pipeline for large-scale cloud environments based on the data by Tengku Asmawi et al. [8]

Figure 4 begins with data ingestion (e.g., Google Cluster Traces, logs, telemetry), which feeds into feature extraction where task-level, job-level, and temporal characteristics are derived. These features are processed in the model inference layer, where different models introduce a key trade-off: ANNs provide higher prediction accuracy (~93%), while logistic regression enables sub-second, large-scale inference. The output (failure probability) drives the decision layer, where actions such as migration, checkpointing, resource reallocation, and scheduling adjustments are now correctly integrated within the same control block. These decisions are executed in the execution layer through control-plane operations on VMs and containers. Finally, the system forms a feedback loop, where updated system state is continuously fed back into the pipeline, emphasizing that fault tolerance is an adaptive, real-time control process governed by accuracy–latency trade-offs.

In a further extension, availability-aware placement research integrates deep learning to predict application task failures as a basis for proactive protective actions, indicating a pattern in which prediction is architecturally coupled to placement and protection decisions [1]. A third cluster emphasizes orchestration and recovery workflows for virtualized and microservice-based systems as the operational backbone of fault tolerance. In private cloud environments, the PCBO approach frames failure handling as part of the VM lifecycle: standardized VM specifications and IaC-driven automation support repeatable provisioning, while log monitoring/purification and similarity-based reconstruction support recovery when issues occur, demonstrated through a case study deployment on OpenStack [5].

At the microservice layer, the literature operationalizes fault tolerance through decomposition plus automated restart/rescheduling capabilities: Micro-FL positions microservices as independently scalable units, with Kubernetes-backed behavior enabling continued operation even when a communication microservice fails, and evaluates fault scenarios in a managed Kubernetes environment with replicated messaging infrastructure (e.g., replication factor settings for brokers and coordination services) [6]. For high-stakes cloud services, proactive reliability-aware failure recovery is formalized as a sequential recovery procedure (launching backups, flow reconfiguration, and state synchronization) and then optimized using deep reinforcement learning that accounts for information freshness via Age of Information and temporal sensitivity via LSTM, aiming to reduce service interruption relative to reactive recovery that begins only after failure [10]. Figure 5 illustrates a fault-tolerant microservice orchestration architecture in a Kubernetes environment, where resilience is achieved through decomposition, replication, and automated control.

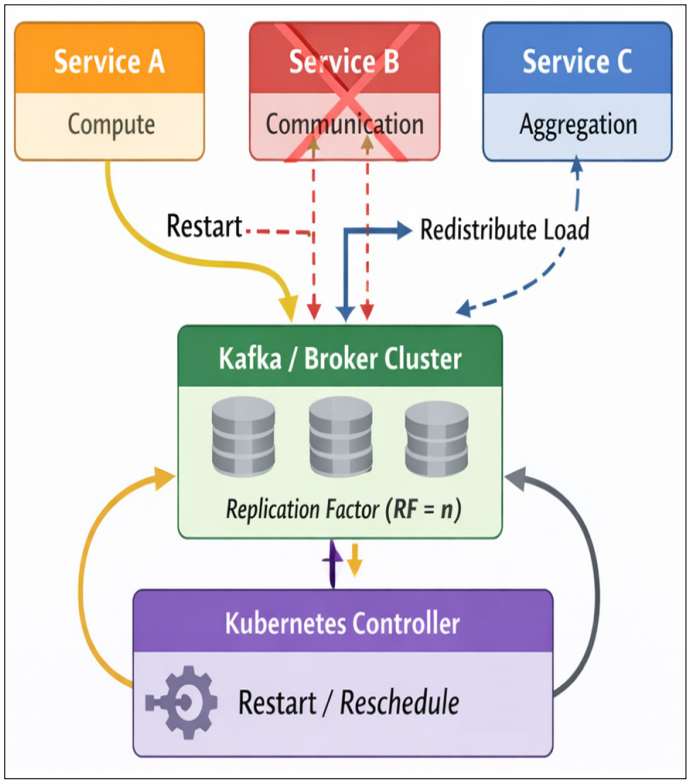

Figure 5. Fault-tolerant microservice orchestration in Kubernetes environments (author's illustration)

Figure 5 depicts three microservices: Service A (compute), Service B (communication), and Service C (aggregation). They are connected through a Kafka-based broker layer with replication factor $RF = n$, which maintains message persistence under failure. In the scenario shown, Service B becomes unavailable (marked by a cross), yet system operation continues. The Kubernetes controller initiates recovery by restarting the failed instance and reallocating it to an available node, while message exchange is temporarily sustained through the broker layer. At the same time, incoming load is redistributed across the remaining services. In this configuration, the messaging subsystem absorbs short-term disruption, enabling recovery to proceed without immediate propagation of failure. The example illustrates how decomposition limits fault scope, but also makes visible

a dependency on the control plane: both orchestration and message coordination must remain operational for recovery actions to succeed.

A related line of work addresses fault tolerance in geo-distributed storage from the perspective of repair topology. Here, resilience depends on how efficiently lost data fragments are reconstructed across the network. Erasure coding lowers storage overhead relative to replication, but recovery after node failure requires coordinated transfers from multiple locations, increasing cross-site traffic and repair time. As a result, the design problem is often formulated with competing objectives, where latency of repair, network load, and node-level balance must be considered jointly. Quantitative evaluations demonstrate that topology-aware repair strategies can reduce average repair latency by up to 41.11% and decrease link-utilization imbalance (standard deviation) by approximately 86.43% compared to baseline approaches [9]. Critically, repair time directly determines the system's vulnerability window, during which additional failures may lead to irreversible data loss, while network congestion emerges as the dominant bottleneck due to cross-data-center traffic amplification. These findings indicate that storage-level fault tolerance is a network optimization problem, where repair path design, bandwidth heterogeneity, and load balancing jointly define system resilience.

Finally, two sources widen the architectural frame by analyzing the structural conditions under which the above mechanisms can be composed at scale. The multi-cloud SLR reports ambiguity in multi-cloud definitions and differentiates replicated vs distributed multi-cloud applications, highlighting "white areas" such as interoperability/portability, security aspects, and lifecycle (DevOps) challenges that complicate consistent resilience engineering across heterogeneous providers [2]. In parallel, a cloud-native service-stack modeling approach emphasizes that cloud-native practices aim for automation and resiliency (alongside manageability and observability), and that extensible stacks must handle heterogeneous infrastructure and support microservice scheduling concerns such as service discovery, traffic routing, and high availability, including support for hybrid/multi-cloud via extensible IaC provider designs [3].

DISCUSSION

Taken together, the corpus suggests that fault tolerance in cloud IT infrastructures is an architectural alignment problem: prediction, optimization, orchestration, and data repair must be aligned so that local resilience actions do not create new vulnerabilities. The most technically mature mechanisms in the reviewed studies are those that formalize trade-offs and operationalize them through automation. Examples include multi-objective placement frameworks that jointly optimize availability and efficiency [1], scheduling mechanisms that explicitly incorporate failure and trust/SLA-related objectives [4], DRL-based recovery policies that formalize proactive decisions under dynamically changing service states [10], and storage repair optimization that treats repair latency and link-load imbalance as concurrent objectives [9].

However, the studies also imply several architectural tensions that remain insufficiently unified. First, proactive prediction is repeatedly justified as a way to reduce recovery time or prevent outages, yet prediction effectiveness depends on (i) operationally available features, (ii) computational scalability, and (iii) the ability to convert predictions into safe actions. Comparative results on Google Cluster Traces underline that model choice and feature importance differ by prediction granularity (job vs task) and that scalability is a distinct evaluation axis, limiting one-size-fits-all adoption [8].

Hybrid reliability approaches further demonstrate that prediction is often entwined with protective mechanisms such as checkpointing, so that architectural benefits arise only when prediction and state-preservation policies are co-designed [7]. In placement-focused work, prediction is integrated into a broader availability-aware framework, but such integration raises governance questions (e.g., when to migrate, when to allocate standby capacity) that can propagate risk if executed without cross-layer constraints [1]. Second, orchestration-centered resilience mechanisms introduce their own dependencies and potential single points of failure. Microservices and Kubernetes-backed restart/rescheduling can reduce the blast radius of component faults; still, platforms rely on control-plane correctness and on state-handling strategies (e.g., whether to replicate an aggregator or rely on restart semantics for a synchronous step) that require careful architectural justification [6].

In domain-specific critical services, proactive recovery involves multiple coordinated steps (backup placement, flow changes, state synchronization) whose timing is constrained by SLA tolerances; optimizing these decisions via reinforcement learning illustrates promise, but also indicates complexity in ensuring predictable behavior and verifiable safety in operational settings [10]. Private cloud orchestration work underscores a parallel point: automation and reuse (IaC plus standardized specifications) can raise consistency and reduce operator load, but reliable recovery also depends on observability pipelines (log monitoring/purification) and on similarity metrics that can reconstruct or reuse assets under failure conditions [5].

Third, multi-cloud evolution complicates fault tolerance because resilience is no longer bounded by one provider's primitives, and architectural models must handle heterogeneous services, portability gaps, and distributed deployment patterns. The SLR indicates that multi-cloud native applications are variably defined and that many studies use the term without precise meaning; this definitional ambiguity constrains how fault tolerance is specified, verified, and operationalized across providers [2].

The cloud-native stack analysis complements this by arguing for systematic modeling and extensible stack mechanisms that can handle heterogeneity and support hybrid/multi-cloud resource access, linking resilience to stack design choices [3].

A synthesis consistent with the corpus is that "architectural solutions" for fault tolerance should be judged by how well they couple three elements: (a) predictive awareness (failure likelihood and service state), (b) decision logic (placement/scheduling/repair optimization under multiple objectives), and (c) automated execution (orchestration workflows that complete recovery within acceptable interruption windows while controlling cost). The reviewed studies each provide partial realizations of this loop, but integrated reference architectures that rigorously connect these elements across compute, microservices, and storage remain comparatively under-specified within the ten-source set.

CONCLUSION

The reviewed studies indicate that fault tolerance in cloud infrastructures is realized through the joint operation of several components. In the analyzed cases, predictive modules, decision logic based on optimization, and execution-level recovery mechanisms function as interdependent elements. This observation suggests that resilient system design involves configurations where placement, scheduling, and repair actions are adjusted in response to evolving system states. Such arrangements allow operators to regulate availability alongside resource utilization and cost under SLA-constrained, heterogeneous workloads.

The results are consistent with the initial assumption that fault tolerance arises at the system level through coordination across layers within a feedback-driven structure linking monitoring, decision processes, and execution. Methods implemented at only one level show reduced effectiveness in the examined studies. By comparison, solutions that establish explicit links between these functions demonstrate greater ability to anticipate disruptions, limit their propagation, and restore service under changing operating conditions.

The analysis leads to an integrated architectural perspective that organizes fault-tolerance mechanisms across the cloud stack and clarifies the interdependencies through which they influence overall system reliability. This framework clarifies how predictive models translate into actionable system adaptations and how these adaptations propagate through infrastructure, service, and data layers to determine overall system reliability. Framing fault tolerance in terms of interacting layers within the cloud stack makes it possible to examine how availability, performance, scalability, and cost are negotiated.

Further work should address the still limited linkage between predictive components and the mechanisms that execute corrective actions, especially in settings characterized by uncertainty and strict timing constraints, and should include empirical validation of cross-layer designs under production-scale workloads. Concrete next steps can be formulated at the level of system design and evaluation. One line of work concerns the definition of interface contracts that allow components deployed across different cloud providers to exchange state and control signals without ad hoc integration. Another involves refining models of failure propagation so that dependencies between compute, network, and storage subsystems are represented with sufficient granularity to capture cascade effects. A related task is to embed energy consumption and operating cost directly into decision rules used for recovery, replication, and scheduling. Progress on these questions depends heavily on better empirical support: operational traces, incident records, and benchmark environments that expose systems to representative workload dynamics and plausible failure scenarios.

REFERENCES

1. Alahmad, Y., & Agarwal, A. (2024). Multiple objectives dynamic VM placement for application service availability in cloud networks. *Journal of Cloud Computing, 13*(46), 1-20. https://doi.org/10.1186/s13677-024-00610-2-
2. Alonso, J., Orue-Echevarria, L., Casola, V., Torre, A. I., Huarte, M., Osaba, E., & Lobo, J. L. (2023). Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review. *Journal of Cloud Computing, 12*(6), 1-34. https://doi.org/10.1186/s13677-022-00367-6
3. Lin, J., Xie, D., Huang, J., Liao, Z., & Ye, L. (2022). A multi-dimensional extensible cloud-native service stack for enterprises. *Journal of Cloud Computing, 11*(83), 1-18. https://doi.org/10.1186/s13677-022-00366-7
4. Mangalampalli, S., Karri, G. R., Gupta, A., Chakrabarti, T., Nallamala, S. H., Chakrabarti, P., Unhelkar, B., & Margala, M. (2023). Fault-tolerant trust-based task scheduling algorithm using Harris Hawks optimization in cloud computing. *Sensors, 23*(8009), 1-33. https://doi.org/10.3390/s23188009
5. Park, J., Jeong, S., & Yeom, K. (2025). Private cloud bespoke orchestrator: Techniques for constructing and operating bespoke-private cloud virtual machine environments for cloud users. *Journal of Cloud Computing, 14*(34), 1-20. https://doi.org/10.1186/s13677-025-00760-x
6. Sabuhi, M., Musilek, P., & Bezemer, C.-P. (2024). Micro-FL: A fault-tolerant scalable microservice-based platform for federated learning. *Future Internet, 16*(34), 1-20. https://doi.org/10.3390/fi16030070
7. Shahid, M. A., Alam, M. M., & Su'ud, M. M. (2023). Achieving reliability in cloud computing by a novel hybrid approach. *Sensors, 23*(1965), 1-55. https://doi.org/10.3390/s23041965

8. Tengku Asmawi, T. N., Ismail, A., & Shen, J. (2022). Cloud failure prediction based on traditional machine learning and deep learning. *Journal of Cloud Computing, 11*(47), 1-19. https://doi.org/10.1186/s13677-022-00327-0

9. Xu, Y., Wang, Y., Jiang, F., Zhou, F., & Chen, J. (2026). Data repair optimization method for geo-distributed fault-tolerant storage systems based on whale optimization algorithm. *Journal of Cloud Computing, 15*(3), 1-21. https://doi.org/10.1186/s13677-025-00807-z

10. Zhu, L., Zhuang, Q., Jiang, H., Liang, H., Gao, X., & Wang, W. (2023). Reliability-aware failure recovery for cloud computing based automatic train supervision systems in urban rail transit using deep reinforcement learning. *Journal of Cloud Computing, 12*(147), 1-14. https://doi.org/10.1186/s13677-023-00502-x