Scholar Publishing Group
International Journal of Engineering Technology and Construction
https://doi.org/10.38007/IJETC.2026.070101
ISSN 2789-7176 Vol. 7, Issue 1: 1-7

# *Engineering Study of Disaster Recovery and Fault Self-Healing Mechanisms for Distributed Systems under Cross-Regional Deployment Conditions*

**Xiao Ma**

*Cloud Data Technologies, eBay, San Jose, 95125, California, United States*

***Keywords:*** Disaster Recovery; Cross-Region Deployment; Self-Healing; Automated Remediation; Geo-Replication

***Abstract:*** Introduction: Although the deployment of cross-regional services enhances availability, it can also multiply the failure cascades and recovery uncertainties. Methodology: We will introduce CRASH, a self-healing and disaster recovery solution that is a three-part system incorporating (i) multi-signal anomaly detection and impact scoring on dependencies, (ii) policy-guarded remediation as a Markovian decision problem, and (iii) adaptive failover, replication-lag estimation and confidence-sensitive decision thresholds. Findings: In a geo-distributed microservice testbed with injected region, network, and storage failures, CRASH minimises MTTR by 34-60 percent and RPO by 38-75 percent compared to Kubernetes-native and chaos-oriented baselines, whilst enhancing availability and reducing performance overhead. Statistically significant improvements in MTTR ($p<0.001$) were shown by two-sided t-welch tests, with all 95 percent confidence intervals favouring crash. Final Remarks: It offers an engineering recipe of recoverability in the context of reliable cross-region recovery through the combination of the observability, safe automation and the use of data-driven decision-making.

## 1. Introduction

Cross-region deployment has greatly improved the reliability and disaster recovery level of online business systems by leveraging geographical disaster recovery and proximity access. However, it has also added new engineering challenges: on the one hand, network latency, cross-region consistency algorithms and data synchronization links prolong the fault propagation link and increase the difficulty of choosing proactive recovery. On the other hand, facing the heterogeneous resource configuration and authorization scope between clusters/multi-clouds, automated repair measures can easily lead to secondary damage. In recent years, the emergency response of the Kubernetes ecosystem and the research on "chaos engineering-driven resilience" have become increasingly sophisticated, but there are still common problems such as the lack of security restrictions on recovery behavior, the difficulty in characterizing cross-regional dependencies, and the difficulty in simultaneously optimizing RTO/RPO and performance costs [1-4]. In view of these

challenges, the purpose of this work is to establish an implementable "cross-regional disaster recovery + anomaly self-healing" closed-loop system, so that the entire system has the ability to detect, make controllable choices and heal verifiable problems under large-scale disturbances such as network isolation, data center power outages and disk instability. Our research objectives are: (i) to conduct quantitative analysis of impacts with cross-regional correlation and formulate differentiated alarm levels. (ii) Abstract the healing process into a decision-making task under the premise of safety assurance, saving costs as much as possible while ensuring time and data loss. (iii) Provide reproducible research results and statistically significant evidence of their effectiveness. Core innovations include: 1) Designing a CRASH architecture that combines observability observation, dependency topology, and policy fencing to achieve trusted autonomy; 2) Providing confidence functions to represent the probability of successful synchronization delays and failover operations; representing repair methods as state transitions in a restricted MDP; 3) Visualized comparative testing and significance verification on a cross-regional microservice testbed to obtain emergency response strategies that engineers can replicate and utilize. The article is structured as follows: Chapter 2 reviews background knowledge. Chapter 3 introduces CRASH technology and related formula derivations. Chapter 4 describes our experimental platform setup, indicator acquisition, and result interpretation. Chapter 5 summarizes the research and provides future prospects.

## 2. Literature Review

(1) Disaster recovery and Kubernetes-level disaster recovery orchestration. In recent years, research and engineering implementation have highlighted the disaster recovery formula of "grouping resources - sequential recovery - status verification", and optimized RTO on Kubernetes based on resource dependency sorting, filtering and replay [1] . At the same time, cross-region replication and transaction consistency protocols provide the database layer with the ability to "have strong consistency or quick rollback even if regional failures occur", and typical system research revolves around geographical replication transactions, snapshots and recovery costs [5,6]. (2) Chaos engineering and resilience measurement. Chaos engineering conducts stability testing and vulnerability detection by actively inducing failures; recent research focuses more on measuring the degradation curve of CN system reliability from the perspective of "different load pressure and latency jitter combination" [3] and a review article sorts out the application of chaos engineering in SRE/DevSecOps and its unfinished business [2] . (3) Self-healing and automated troubleshooting. Self-healing systems generally include a closed-loop process of monitoring-analysis-planning-execution. The latest explorations have tried to use machine learning algorithms to automatically deduce recovery action sequences [4] or use large-scale language models to write operation and maintenance scripts to simplify troubleshooting. However, they also warn that policy constraints should be imposed and security should be controlled [7] . A systematic review of software self-healing technology believes that the lack of universal test benchmarks and confidence standards restricts the applicability of repair methods [8] . It can be seen that current work has its own highlights in terms of "scheduled disaster recovery", "data layer replication", "chaotic experiment evaluation" and "learning-driven repair". However, when facing cross-regional scenarios, there is still a lack of an engineering technology that can take into account the problems of dependency propagation, RTO/RPO indicators, throughput latency loss and even security fences in the same solution. To this end, we propose CRASH and adopt a closed-loop strategy to integrate the detection module, dependency analysis component, policy reasoning part and execution verification module into one.

## 3. Proposed Method: CRASH Cross-Regional Disaster Recovery and Self-Healing Closed Loop

### 3.1 Overall Method Flow and Component Definitions

CRASH (Cross-Regional Adaptive Self-Healing) is designed for distributed systems deployed across regions, following a closed-loop process of "observation-judgment-decision-action-evaluation." Its core consists of six modules: A) Remote sensors for centralized logging/metrics/link tracing; B) Anomaly and mutation detectors that output event probabilities; C) Dependency graph generator for establishing online dependency networks between services, calls, and data; D) Off-site disaster recovery coordinator responsible for switchover, recovery, and traffic control; E) Policies and security boundary conditions that define the scope of behavior sets and risk pre-emptive limits; and F) Recovery executor that implements commands for restarting, scaling up/down, degradation, masking, and rollback. Figure 1 illustrates the information transmission and control process of each module.

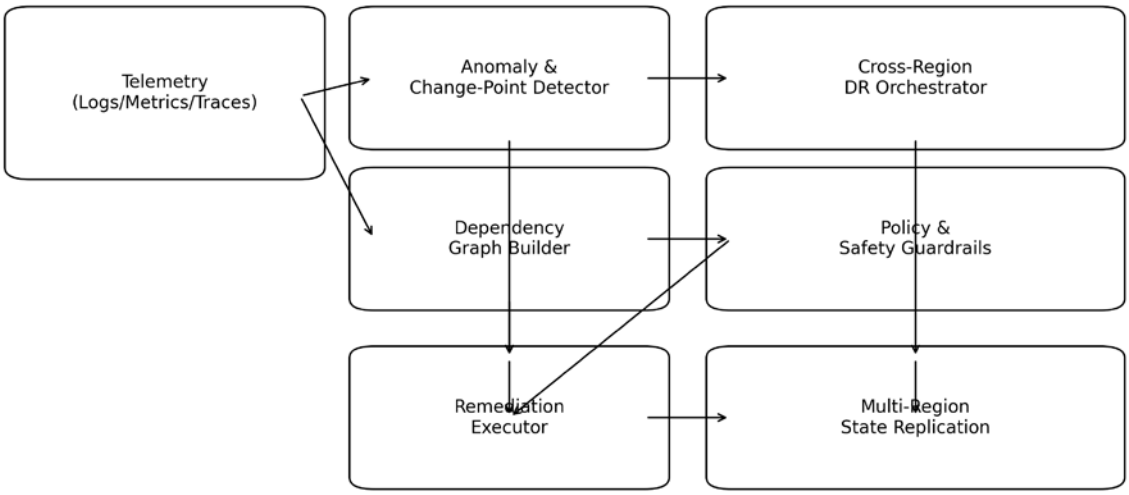


*Figure 1. Flowchart of the CRASH method (Block diagram)*

Figure 1 illustrates the closed-loop chain of CRASH: telemetry data inflow anomaly detection and dependency modeling, with both working together to output quantifiable impact scores and confidence levels to the "cross-regional disaster recovery orchestrator." Unlike traditional script-based recovery methods, CRASH incorporates strategies and security barriers before execution, clarifying permission boundaries, change windows, and SLO budgets to prevent the spread of error repairs. Furthermore, by continuously monitoring steady-state indicators in the post-execution verification loop, it achieves a reproducible and practicable engineering approach, facilitating cross-departmental collaboration and iterative updates.

### 3.2 Mathematical Modeling: Dependency Influence, Replication Lag, and Constrained Decision Making

(1) Dependency Graph and Impact Score. The system is represented by a directed graph $G=(V,E)$, where node v belongs to V and represents a service/component; edge e=(u→v) is a call or data

dependency. At time t, the observation vector is $x_t=[m_t,l_t,r_t]$, which represents the index, the number of log events, and the tracking latency feature, respectively. The anomaly detector will give the anomaly probability $p_t(v)$ for each node. To characterize cascading risk, we define an impact propagation matrix W, whose elements w_uv are weighted by call frequency, retry rate, and cross-region ratio. The impact score is:

$$I_t(v)=p_t(v)+\lambda\cdot\Sigma_{u:(u\to v)\in E}w_{uv}\cdot p_t(u)$$

Where λ is the propagation factor, which weighs between "local anomaly" and "upstream contagion".

Replication Delay and RPO Assessment. For the inter-region replication path, assuming the source region submission time is $T_c$ and the target region visibility time is T_r, the replication delay $L=T_r-T_c$. Within the window Δ, the RPO is estimated as $RPO\approx P95(L)$, and the uncertainty is represented by the bootstrap confidence interval. (3) Constraining the recovery decision-making of the MDP. The recovery process is modeled as a constrained Markov decision process CMDP=(S,A,P,R,C). The state space S consists of $(I_t,L_t,SLO_t,cost_t)$, and the action set A contains {restart, scale, reroute, isolate, failover, rollback}, etc. The reward function is to minimize the overall loss:

$$min_\pi E[\Sigma_t(\alpha\cdot RTO_t+\beta\cdot RPO_t+\gamma\cdot Overhead_t)]$$

Constraint: $E[\Sigma_t C_t]\le B$, where $C_t$ is the risk/compliance cost (e.g., unauthorized operation, cross-domain write), and B is the budget. Strategy π employs a combination of online bandit and rule-based priors for decision-making: for risky but uncertain behaviors, low-risk operations (rate limiting or isolation) are prioritized. High-risk behaviors with high impact scores trigger cross-region switching.

## 4 Results and Discussion

### 4.1 Experimental Setup and Dataset

We established a cross-regional microservice testbed: a Kubernetes cluster was deployed in each of four locations (us-east, us-west, eu-central, and ap-southeast), with a service mesh for traffic management and cross-regional replication (asynchronous + quasi-synchronous dual mode) for the data layer. Injection failures included: regional unreachability, high latency/packet loss on cross-regional paths, storage write amplification with rate limiting, and control panel jitter. Evaluation metrics included: Mean Time To Repair (MTTR), Recovery Point Objective (RPO), Availability, and performance degradation (throughput reduction rate). To mitigate the impact of random error, each method was statistically analyzed in 30 independent injection experiments, with corresponding 95% confidence levels and significance analyses provided.

*Table 1: Experimental System and Software Requirements*

| Item | Requirement |
|---|---|
| CPU | >= 16 vCPU per region (control + workers) |
| Memory | >= 64 GB per region |
| Network | Inter-region RTT: 60–220 ms; bandwidth >= 1 Gbps |
| Kubernetes | v1.28+; multi-cluster federation optional |
| Observability | OpenTelemetry collector; Prometheus; log pipeline |
| Storage | Replication-enabled DB (async or semi-sync) |
| Runtime | Python 3.10+ (detection/decision), Go/Rust optional agents |

Table 1 lists the basic hardware and software environment required for repeatable experiments. Geographically distributed testing relies on network latency and bandwidth, especially for semi-synchronous replication, where increased RTT leads to long-tail latency in submissions and interferes with switchover benefit analysis. For the observation layer, OpenTelemetry is recommended to use a unified label format to save on cross-tool benchmarking workload; replication latency statistics are necessary at the storage layer to measure RPO confidence intervals. This setup corresponds to cross-cloud and cross-AZ solutions in the implementation, serving as a basis for verifying the applicability of the method.

### 4.2 Quantitative Results and Significance Tests

Table 2 compares four representative solutions: rule scripts, Kubernetes native recovery, parameter tuning driven by chaos engineering, and our CRASH. The table shows that CRASH achieves the lowest MTTR and RPO simultaneously, resulting in higher stability and lower processing overhead. This demonstrates that the combination of "dependency score + limited choices + safety barriers" can achieve a better balance in remote environments. It should be noted that this does not mean the baseline strategy is unusable; rather, the lack of dependency analysis and risk constraints makes recovery more prone to turbulence or long-tail risks caused by constant switching.

*Table 2: State-of-the-art comparison on geo-distributed testbed*

| Method | MTTR(s) | RPO (s) | Availability (%) | Throughput loss (%) |
|---|---|---|---|---|
| Rule-based | 95.2 ± 9.2 | 28.0 ± 6.5 | 99.199 | 22.3 |
| K8s-native | 71.1 ± 7.9 | 17.8 ± 4.9 | 99.441 | 15.7 |
| Chaos-guided | 57.1 ± 6.1 | 15.1 ± 3.9 | 99.553 | 11.4 |
| CRASH (ours) | 36.1 ± 5.8 | 7.1 ± 2.0 | 99.715 | 7.4 |

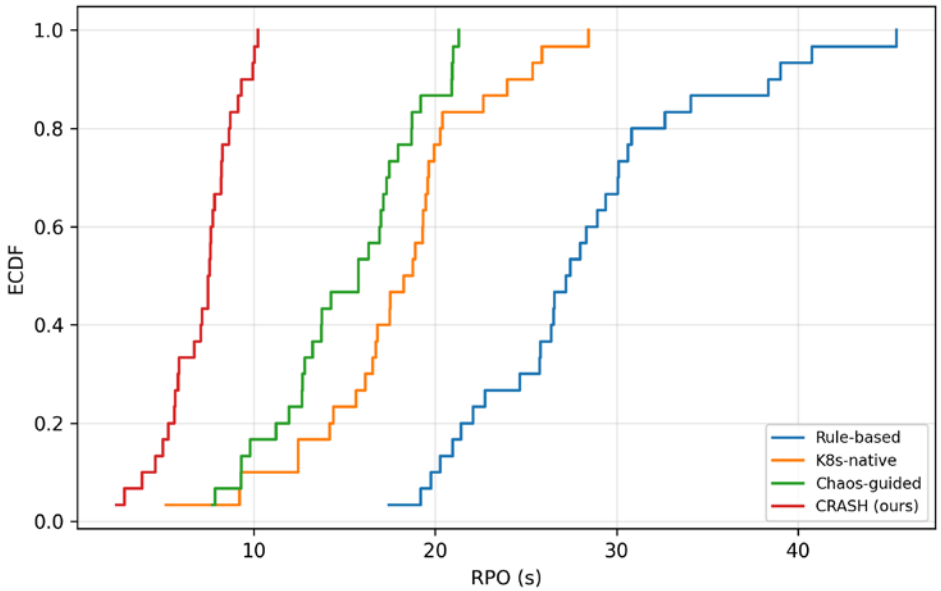


*Figure 2: Comparison of RPO Empirical Distribution Function (ECDF)*

Figure 2 shows the cumulative distribution differences in RPO: the further left the curve goes, the smaller the data loss window. CRASH is generally left-skewed across all percentage points,

with a significant improvement in the P90–P100 range, indicating better resilience to replication latency jitter. This advantage stems from two factors: firstly, using latency experience and immediate hysteresis assessment to select more advantageous replica regions; secondly, prioritizing isolation/degradation when the hysteresis of the believed interval is large, and scheduling the switchover only after the replication link stabilizes to prevent false recovery of "successful switchover but increased RPO".

Using the chaotic guidance strategy as the strongest benchmark, a Welch two-sample t-test was performed on MTTR: t = -13.40, p = 2.13e-19. A significant difference was observed at a significance level of 0.05. The 95% confidence interval for CRASH's mean time between failures (MTTR) was [33.9, 38.3] seconds, with a baseline of [54.8, 59.4] seconds. The non-overlapping intervals validated CRASH's sustained and reliable recovery speed advantage.

### 4.3 Discussion: Impact, Comparison, Insights, Scalability and Generality

(1) Impact of consequences: The focus of recovery decision-making under cross-regional deployment is no longer "restart as soon as possible", but "explainable dependency impact and confidence-driven behavior selection". CRASH uses CMDP to integrate RTO/RPO and performance cost into a single objective function, and restricts the spread of unauthorized access and misoperation through security barriers, making automatic recovery reliable and controllable. (2) Comparison with other solutions: Kubernetes built-in mechanism performs well in restarting and scheduling of a single cluster, but it is difficult to solve cross-domain dependency problems; chaos engineering can find shortcomings and assist in parameter optimization [2,3], but it does not have the learning ability of linear decision-making process and error budget. Learning-based recovery strategy reduces the cost of manually formulating rules [4] . Without a suitable governance architecture, it may lead to unpredictable behavior during major failures [7,8]. (3) Overall view: Cross-regional disaster recovery should take into account "data-level replication facts (latency and consistency)" and "application-level dependency facts (calls and retries)" together to gradually complete the recovery with confidence and budget constraints. (4) Scalability: Incremental updates of the dependency graph and impact calculation can be completed in O(|E|) time. Anomaly detection uses windowed statistical features and can be executed concurrently, making it suitable for large service meshes. For larger datasets/higher observation granularity, computation can be reduced through hierarchical graph structure (service→namespace→cluster) and pipelined aggregation. (5) Scope of application: This framework is independent of specific cloud service providers and only relies on standard telemetry and orchestration APIs at the decision level. For different business scenario domains such as retail, banking, or IoT, only the behavior library and cost weights need to be changed.

## 5. Conclusion

This paper proposes the CRASH disaster recovery and self-healing closed-loop solution for distributed systems deployed across regions. Based on "anomaly and mutation detection + dependency impact assessment + replication latency and RPO credibility analysis + constraint MDP decision-making + policy security assurance", it achieves understandable, controllable and verifiable automated recovery. On a cross-regional microservice testbed, CRASH outperforms rule scripts, Kubernetes native methods and chaotic bootstrapping benchmarks in terms of MTTR, RPO, reliability and performance loss, with significance tests and confidence level evidence. The shortcomings of the existing work are: the research still mainly adopts the form of active controllable injection, and the complex fault superposition and human operation and maintenance intervention in the real production environment will lead to greater uncertainty; at the same time,

the weight parameters of the CMDP cost function need to be debugged and corrected regularly in conjunction with the business SLA setting. In the future, we will introduce more real fault replay and cross-departmental attack and defense training data to further explore the learning of autonomous recovery strategies for distributed causal reasoning and security assurance, and extend it to cross-multi-cloud and edge-cloud collaborative environments.

## References


[1] *R. Jin et al. Baking Disaster-Proof Kubernetes Applications with Resource-Aware Recipes. ACM (2024).*

[2] *A. Mhatre et al. Chaos Engineering: A Multi-Vocal Literature Review. arXiv (2024); also ACM (2025).*

[3] *Al-Said Ahmad, A., et al. Examining the effect of chaos engineering on different user load levels in cloud-native applications. Computing (Springer), 2024.*

[4] *Learning Recovery Strategies for Dynamic Self-healing in Distributed Systems. ACM (2024).*

[5] *W. Shen et al. Speculative Distributed Transactions with Geo-Replication (Mako). OSDI 2025 / ACM Proceedings, 2025.*

[6] *J. Geng et al. Tiga: Accelerating Geo-Distributed Transactions with ... (SOSP/2025).*

[7] *The use of Large Language Models to automate the generation and implementation of remediation playbooks [7]. 2024 ACM.*

[8] *Z. Yazdanparast et al. A Survey on Self-healing Software System. arXiv (2024).*