International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 5, Issue 1, 160-166, 2024
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P115

Original Article

The Adaptive Intelligence in Cloud Systems: A Unified Architecture for AI Enhanced Observability and Automated Root Cause Analysis

Indrasena Manga[1], Sai Dheeraj Sivva[2], Vamshi Krishna Manga[3]
[1,3]Independent Researcher Dallas, Texas, USA.
[2]Software Engineer, Independent Researcher Charlotte, NC, USA.

Abstract - *Modern cloud systems generate high volume telemetry across logs traces metrics events and configuration state. Observability platforms can collect and visualize these signals yet interpretation triage and remediation still depend heavily on human expertise. Manual root cause analysis static dashboards and rule based alerts struggle with the complexity of microservices event driven pipelines and distributed runtime variability. This paper proposes a unified architecture for adaptive intelligence in cloud systems that integrates an observability pipeline lightweight learning models and autonomous decision policies to achieve continuous self analysis fault localization and performance optimization. The architecture enables applications to learn behavioral baselines detect degradations early identify root causes through dependency aware reasoning and apply corrective actions such as dynamic throttling configuration tuning resource scaling and workflow rerouting. We provide a concrete implementation blueprint including data normalization feature engineering diagnosis scoring and policy guardrails. We also describe an evaluation plan that targets improved mean time to detect and mean time to recover while reducing alert noise and controlling automation risk. The results argue that embedding adaptive intelligence directly into the operational stack is becoming essential for predictable resilience in large scale cloud services.*

Keywords - *Adaptive Intelligence, Aiops, Observability, Distributed Tracing Log Analytics, Automated Root Cause Analysis, Predictive Resilience, Kubernetes, Cloud Operations.*

1. Introduction

Enterprise software has shifted from bounded deployments to continuously evolving distributed systems. Elasticity and rapid delivery enable faster iteration but they also introduce complex interactions across compute networking storage middleware and application logic. Incidents often arise from combinations of partial failure saturation and configuration drift rather than from a single obvious fault. As service graphs grow large and as dependencies change frequently reliability depends on responding to tail latency and cascading timeouts rather than on average behavior. The Tail at Scale highlights why latency variability drives user experience in large online services and why tail tolerant design techniques are required [1].

Observability has become the standard method to capture evidence about runtime behavior. Metrics summarize rate and latency logs capture events and traces capture end to end request paths. However collecting evidence is not the same as understanding it. On call engineers still perform diagnosis by searching dashboards filtering logs and correlating trace anomalies with recent changes. This manual workflow is slow during peak impact and it does not scale with high dimensional telemetry. As a result mean time to detect and mean time to recover remain high in many environments and reliability depends on scarce expert intuition.

AIOps research proposes using machine learning to automate detection correlation and diagnosis. Recent surveys emphasize anomaly detection event correlation and root cause analysis while also highlighting the need for governance reproducibility and safe integration with operational workflows [2]. At the same time the industry has moved toward standard telemetry and programmable control planes. Together these trends enable a practical design goal: adaptive intelligence embedded into the operational stack that continuously learns normal behavior detects degradation localizes likely causes and recommends or applies corrective actions with bounded risk.

This paper proposes a unified architecture for adaptive intelligence in cloud systems. The architecture combines an observability data plane a context and topology model a learning plane for baseline and anomaly detection a diagnosis plane for dependency aware root cause localization and a policy plane that selects actions such as scaling throttling tuning or rerouting. Unlike approaches that rely on heavyweight deep learning as the primary mechanism the proposed design emphasizes lightweight models and interpretable reasoning so adoption is feasible for enterprises with limited labeled data and

strict change control. Deep learning methods for logs and combined trace and log modeling remain important references and baselines [3] [4] but the focus here is pragmatic adaptive operations at enterprise scale.

2. Background and Related Work

2.1. Observability signals and distributed tracing

Observability enables operators to infer internal system state from external signals. In cloud systems the common telemetry signals are metrics logs traces events and configuration state. Distributed tracing is particularly valuable because it captures causal ordering and latency across service calls which enables dependency mapping and critical path analysis. Trace based anomaly detection work shows that distributed traces can be mined to detect anomalous request executions and classify incidents [5]. Modern localization approaches incorporate span attributes and multi dimensional reasoning to identify performance root causes in complex microservice systems [6].

Despite its benefits tracing introduces overhead and cost which motivates sampling compression and selective instrumentation. Therefore an adaptive intelligence architecture must work with imperfect telemetry and must quantify confidence under sampling. It must also integrate tracing with logs metrics and changes so that diagnosis does not depend on any single data source.

2.2. Log parsing and log anomaly detection

Log analytics pipelines commonly begin with parsing. Drain provides an efficient online log parsing method using a fixed depth tree to structure free form logs into templates which accelerates downstream modeling [7]. Once structured logs are available anomaly detection can use template frequency shifts sequence modeling or clustering. DeepLog models sequences of log events using recurrent networks and demonstrates accurate anomaly detection and diagnosis from system logs [3]. While DeepLog is effective it requires careful preprocessing and may be costly for large multi tenant environments which motivates lightweight alternatives for many enterprises.

2.3. Trace and multimodal anomaly detection

Microservice incidents often manifest simultaneously across traces logs and metrics. DeepTraLog demonstrates that combining trace structure with log information through graph based deep learning can improve detection of microservice anomalies [4]. Trace based methods also use graph representations to capture the non linear structure of request paths. TraceGra proposes a graph representation for microservice traces and uses an encoder decoder model to detect anomalies [8]. Graph variational autoencoder methods have been applied to trace anomaly detection as well [9]. These approaches demonstrate the value of structure aware representations yet enterprise deployment still requires drift handling sampling robustness and interpretability.

2.4. Automated root cause analysis in microservices

Root cause analysis aims to identify the service instance dependency resource or configuration that most likely caused observed symptoms. MicroRCA proposes real time localization for performance issues using causal inference across service graphs and metrics [10]. TraceRCA proposes a practical approach that ranks suspicious services by comparing abnormal traces with normal traces and by mining a suspicious microservice set [11]. TraceContrast localizes multi dimensional root causes by combining contrast sequential pattern mining with spectrum analysis over trace representations [6]. Recent work also emphasizes interpretability and operator alignment. Chain of Event proposes an interpretable model that learns weighted causal relations between events derived from multimodal observation data [12]. Graph neural approaches such as Sleuth model causal impact across spans to perform trace based root cause analysis at large scale [13].

2.5. Anomaly detection and RCA surveys and evidence on adoption

Surveys of anomaly detection and root cause analysis in microservice based cloud applications highlight a diverse set of techniques and underline common challenges such as data quality lack of labels and evaluation gaps [14]. Industrial studies also show that telemetry adoption and analysis remain challenging due to organizational boundaries tooling fragmentation and evolving architectures [15]. These observations motivate a unified architecture that treats data normalization governance and integration as first class concerns rather than as afterthoughts.

3. Problem Statement and Design Goals

Enterprises typically operate an observability stack with dashboards alert rules and incident workflows. Yet three gaps prevent reliable automation at scale.

First there is fragmentation of evidence. Logs traces metrics and events live in different tools with inconsistent identifiers and ownership metadata. This makes correlation slow and it prevents model features from capturing full context.

Second there is limited automation of diagnosis. Alerts identify symptoms but root cause analysis remains manual and repetitive. Even when anomaly detectors exist they often output scores without actionable localization or without explaining dependencies and recent changes that contributed to risk.

Third there is weak linkage between insights and actions. Teams may have playbooks but execution is manual and inconsistent. When autonomous actions exist they are often narrow for example fixed scaling rules and they may conflict with reliability goals or cost goals under variable load.

The proposed architecture targets an integrated solution that closes these gaps. The design goals are: traceability from telemetry to services to ownership and configuration, lightweight modeling that works with limited labels, dependency aware diagnosis that produces ranked root cause candidates with evidence and policy driven action selection with guardrails audit logs and closed loop verification.

4. Unified Architecture for Adaptive Intelligence

4.1. Architecture overview

The architecture is organized into six cooperating planes: telemetry ingestion context enrichment topology and state modeling learning and detection diagnosis and localization and policy and actuation. The intent is to treat adaptive intelligence as an operational service that sits alongside the platform. It consumes telemetry and change data then continuously emits assessments decisions and evidence bundles that can be reviewed by engineers or consumed by automation.

4.2. Telemetry ingestion plane

The telemetry ingestion plane collects logs traces metrics events and configuration snapshots. To enable correlation the ingestion layer enforces a common identity model including service name environment region cluster namespace workload instance and deployment version. When request identifiers are available they are propagated through traces and logs to support joining multiple signals for the same request. Ingested events are time aligned using a unified clock reference and stored in a scalable time series store and search store.

This plane also performs loss aware sampling. For traces it supports head based sampling and tail based sampling while tracking sampling probability so anomaly estimates can be debiased. For logs it supports template driven sampling where common templates may be sampled more aggressively while rare templates are retained to preserve diagnostic value.

4.3. Context enrichment plane

Context enrichment adds ownership metadata business criticality service level objectives and dependency tags. It integrates with service catalogs deployment systems and configuration management databases. Each event is attached to an ownership record and to an SLO record. This enables routing of incidents and it allows policy decisions to depend on service tier.

Context enrichment also tracks change events. Each deployment emits a change record with commit identifier rollout stage feature flag state and configuration diff hash. When configuration is managed by infrastructure automation the system records an infrastructure change identifier. These signals are essential for change attribution during diagnosis and for safe rollback automation.

4.4. Topology and state modeling plane

Topology modeling constructs a dynamic dependency graph across services and resources. Edges are derived from trace parent child relations service discovery data and network flow telemetry. The graph is updated continuously and versioned so that drift and newly introduced dependencies can be detected. The architecture maintains both a service level graph and a workload instance graph. This enables localization at the service level for communication and at the instance level for remediation.

State modeling maintains a vector of system state per node that includes latency distributions error distributions saturation metrics configuration version and recent change markers. These state vectors are used by both detection and diagnosis components.

4.5. Learning and detection plane

The learning plane maintains baselines and detectors for key signals. Baselines are computed per service and per critical journey. For example an API operation may have separate baselines for weekday and weekend patterns and for regional traffic patterns. Because labeled incident data is often limited the architecture favors unsupervised and semi supervised methods.

For scalar time series such as error rate and latency percentiles the baseline model uses robust seasonal decomposition with median absolute deviation thresholds. For multidimensional features it uses robust principal component analysis to learn

normal subspaces. For heterogeneous feature sets isolation forest provides a fast anomaly score and supports incremental retraining. For structured logs the system uses template frequency shifts and optional sequence prediction as a baseline referencing DeepLog [3]. For traces it uses graph derived features such as call counts critical path latency span variance and fan out then applies one class detection.

Detectors emit anomaly events with confidence and persistence score. Persistence ensures that alerts represent sustained degradation rather than transient spikes. Confidence combines detector score sampling probability and baseline variance so operators can reason about reliability of the alert.

4.6. Diagnosis and localization plane

The diagnosis plane correlates anomalies and localizes root causes. It starts with anomaly clustering by time window service neighborhood and shared request attributes. The dependency graph enables propagation reasoning. If downstream services show elevated latency while an upstream dependency shows saturation the system models the likely direction of impact. Temporal lag analysis helps distinguish cause from symptom.

Localization produces a ranked list of candidate causes with evidence. The engine combines anomaly contribution change attribution and causal explanation. Anomaly contribution estimates how strongly a node explains downstream anomalies using propagation likelihood on graph edges. Change attribution increases prior probability for nodes with recent changes when anomaly onset aligns with rollout. Causal explanation constructs an event causal graph similar in spirit to Chain of Event where nodes represent anomalies changes and state transitions and edges represent learned influence strength [12].

The scoring design borrows from practical trace ranking in TraceRCA [11] and from causal reasoning in MicroRCA [10]. It can incorporate trace structure differences using methods similar to TraceContrast when multi dimensional causes such as environment coordination or communication overhead are involved [6]. The output includes top candidates a causal path narrative and direct links to supporting telemetry.

4.7. Policy and actuation plane

The policy plane converts diagnosis outputs into decisions and actions. Policies define allowed actions per service tier and per environment. For a high criticality service policies may allow automated rollback and scaling but require human approval for configuration changes. For a batch service policies may allow throttling rerouting and resource tuning with lower approval thresholds.

Each candidate action is evaluated by preconditions and by a cost risk model. Preconditions include current SLO risk saturation signals dependency stability and recent change constraints. The cost risk model estimates impact on latency error and cloud cost and it penalizes actions that expand blast radius. Actions are executed via platform APIs such as orchestrator scaling controls traffic management controls and deployment rollback controls.

Closed loop verification is required. After execution the system monitors whether SLO aligned metrics improve. If improvement does not occur within a bounded window the system can revert the action and escalate to human operators with an evidence bundle. This ensures automation is reversible and auditable.

5. Feature Engineering and Algorithms

5.1. Metrics feature set

Metrics features are derived from latency distributions error distributions throughput and saturation indicators such as CPU memory and queue depth. Instead of relying only on averages the architecture uses distribution percentiles and tail focused features since tail behavior drives user experience [1]. Features include p50 p90 p99 latency error rate rolling variance and slope and saturation ratios. When histograms are available features also include distribution shift metrics such as Wasserstein distance between current and baseline windows.

5.2. Trace feature set

Traces are represented as graphs where nodes are spans and edges represent parent child relations and cross service calls. The system computes per trace features including total duration critical path duration number of services fan out and span level outlier counts. It also computes service graph features aggregated over a time window including edge latency inflation and edge error propagation ratios. This representation supports detection and localization and it aligns with structure aware trace methods [8] [9].

To keep models lightweight the system does not require end to end deep trace embedding for all services. Instead it uses engineered graph features and uses simple detectors such as one class models. Deep or graph neural models can be used as optional modules for selected critical services when the value justifies the cost as shown in Sleuth and DeepTraLog [4] [13].

5.3. Log feature set

Logs are parsed into templates using an online parser such as Drain [7]. Features include template frequency changes template co occurrence patterns and sequences for selected components. In many incidents a single rare template burst indicates an exception path or a retry storm. The system highlights such bursts and correlates them with trace anomalies. When sequence modeling is desired DeepLog provides a strong baseline for modeling normal sequences and predicting anomalous patterns [3].

5.4. Correlation algorithm

Correlation operates in two stages. First anomaly events are grouped by time window and by shared identity attributes such as service environment and deployment version. Second groups are expanded on the dependency graph to include neighborhood anomalies. A correlation score is computed using temporal overlap and graph proximity weighted by propagation likelihood on edges. This produces incident candidates that represent coherent multi signal evidence rather than isolated alerts.

5.5. Root cause scoring algorithm

Given an incident candidate the system computes a suspiciousness score for each node in the neighborhood. The score includes: anomaly strength measured as normalized residual, abnormal trace ratio compared to baseline following TraceRCA principles [11], change prior based on recent deployment timing and causal path likelihood based on learned event influence weights inspired by Chain of Event [12]. Nodes are ranked and the top k nodes are presented with supporting evidence.

The algorithm can support fine grained localization when a service has multiple candidate resource metrics. Causal inference based approaches such as CausalRCA show how weighted causal graphs can be used to identify specific metrics as root causes [16]. In the proposed architecture such fine grained localization can be enabled for selected services by adding metric level nodes under a service node.

6. Operational Scenarios

6.1. Scenario 1 latency regression after deployment

Consider an API gateway service that experiences elevated p99 latency immediately after a deployment. Metrics show increased request latency and a slight increase in error rate. Traces show that the critical path expanded within a downstream authentication service. Logs show a burst of templates related to cache misses and token validation retries. The system correlates these signals and creates an incident candidate.

The diagnosis engine evaluates change attribution and finds that the authentication service had a configuration change at the same time window. The dependency graph shows that the gateway depends on authentication for most requests. Propagation likelihood is high because downstream latency increases align with upstream calls to authentication. The root cause ranking places authentication as top candidate and highlights the causal path from configuration change to cache miss burst to increased validation latency to gateway tail latency.

The policy plane evaluates allowed actions. For this tier one service the policy permits rollback of the last change and a temporary capacity scale out to protect user experience. The system recommends rollback and optionally executes a controlled scale out while waiting for approval. After rollback the system verifies improvement in p99 latency and reduces the scale out when stability is confirmed.

6.2. Scenario 2 dependency failure and retry storm

Consider a payment processing flow where a downstream third party dependency becomes slow. Traces show increased duration on the dependency edge and increased fan out due to retries. Metrics show rising queue depth and CPU saturation in upstream services due to retry storms. The system detects anomaly in edge latency and identifies retry patterns from logs. Localization ranks the dependency call edge as the most likely initiating factor while also identifying which upstream services are most impacted.

The policy plane may decide to apply adaptive throttling or circuit breaking to reduce amplification and preserve overall system health. Actions are bounded by policies to ensure that throttling respects business rules and does not exceed a configured rate limit. Closed loop verification checks whether saturation reduces and whether error budgets stabilize.

6.3. Scenario 3 resource contention in a multi tenant cluster

In multi tenant environments a noisy neighbor can cause resource contention. Metrics show elevated CPU steal time and increased latency across multiple services within the same cluster node pool. Traces show that latency inflation is distributed and not limited to a single dependency. The topology model highlights that affected services share the same node pool and that a new batch workload started recently. Localization identifies the batch workload as a likely cause and recommends moving it

to a separate node pool or applying resource limits. This scenario illustrates the need to model infrastructure as part of the dependency graph rather than focusing only on service calls.

7. Governance Security and Reliability of Automation

Automation raises governance concerns. First actions must be auditable. Each decision includes an evidence bundle with links to telemetry summaries change records and the causal path explanation. Second policies must align with organizational boundaries and approval workflows. The architecture supports recommendation mode approval mode and autonomous mode so teams can adopt gradually.

Security and privacy are also critical. Telemetry may include sensitive identifiers therefore the ingestion plane should support field redaction tokenization and access control. The context enrichment plane should integrate with identity and access management so only authorized users can view incident details for a given service. Model artifacts should be stored securely and changes to models should follow the same change management practices as code. Reliability of the adaptive intelligence system itself must be addressed. It should be deployed as a highly available service and should fail safe. If telemetry is missing or if models are unhealthy the system should degrade to recommendation mode or no action. Chaos engineering principles can be applied to validate resilience assumptions and to test whether automated actions remain safe under partial failures [17].

8. Evaluation Plan

Evaluation should combine offline replay and online experiments. Offline replay uses historical telemetry and incident labels when available to compute precision recall and localization rank. Online experiments use controlled fault injection in a staging environment and progressive rollout in production for low risk actions.

Key metrics include mean time to detect mean time to recover change failure rate and alert volume. Diagnosis quality is measured by top k accuracy and mean rank. Action quality is measured by success rate of automated remediation and by rollback rate. Cost impact is measured by resource utilization and by additional telemetry processing overhead. Baselines include rule based alerting traditional dashboard driven response and individual ML detectors. DeepLog is used as a log anomaly baseline [3]. DeepTraLog is used as a combined trace and log baseline and TraceGra and Graph VAE methods represent trace anomaly baselines [4] [8] [9]. Localization baselines include MicroRCA and TraceRCA and interpretability baselines include Chain of Event and large scale baselines include Sleuth [10] [11] [12] [13].

9. Conclusion

Adaptive intelligence in cloud systems requires more than anomaly detection. It requires an integrated architecture that unifies telemetry context dependency models diagnosis and safe action policies. This paper proposed a unified architecture that combines observability pipelines lightweight learning models dependency aware localization and policy driven actuation. By connecting evidence to diagnosis and diagnosis to bounded actions the architecture aims to reduce operational toil improve reliability outcomes and support predictable resilience in large scale cloud services. Future work includes large scale empirical validation improved causal inference under drift and deeper study of policy effectiveness for automated remediation.

References

[1] J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74 80, 2013, doi: 10.1145/2408776.2408794.

[2] Notaro, P., Cardoso, J., & Gerndt, M. (2021). A Systematic Mapping Study in AIOps. In F. Hacid et al. (Eds.), Service-Oriented Computing – ICSOC 2020 Workshops (pp. 110–123). Springer, Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-030-76352-7_15

[3] M. Du, F. Li, G. Zheng and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2017, doi: 10.1145/3133956.3134015.

[4] C. Zhang et al., "DeepTraLog: Trace Log Combined Microservice Anomaly Detection through Graph based Deep Learning," in Proceedings of the International Conference on Software Engineering, 2022, doi: 10.1145/3510003.3510180.

[5] Gunda, S. K. G. (2023). The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126-129. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113

[6] Wu, L., Tordsson, J., Elmroth, E., & Kao, O. (2020). MicroRCA: Root cause localization of performance issues in microservices. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS) (pp. 1–9). IEEE. https://doi.org/10.1109/NOMS47738.2020.9110353

[7] P. He, J. Zhu, Z. Zheng and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in 2017 IEEE International Conference on Web Services, 2017, doi: 10.1109/ICWS.2017.13.

[8] J. Chen et al., "TraceGra: A Trace based Anomaly Detection for Microservice using Graph Deep Learning," Computer Communications, 2023, doi: 10.1016/j.comcom.2023.03.028.
[9] Z. Xie et al., "Unsupervised Anomaly Detection on Microservice Traces through Graph VAE," in Proceedings of The Web Conference, 2023, doi: 10.1145/3543507.3583215.
[10] L. Wu, J. Tordsson, E. Elmroth and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," in 2020 IEEE IFIP Network Operations and Management Symposium, 2020, doi: 10.1109/NOMS47738.2020.9110353.
[11] Z. Li et al., "Practical Root Cause Localization for Microservice Systems via Trace Analysis," in 2021 IEEE ACM International Symposium on Quality of Service, 2021, doi: 10.1109/IWQOS52092.2021.9521340.
[12] Ding, R., Zhang, C., Wang, L., Xu, Y., Ma, M., Wu, X., … & Rajmohan, S. (2023). TraceDiag: Adaptive, interpretable, and efficient root cause analysis on large-scale microservice systems. arXiv. https://doi.org/10.48550/arXiv.2310.18740
[13] Liu, D., He, C., Peng, X., Lin, F., Zhang, C., Gong, S., … & Wu, Z. (2021). MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. arXiv. https://doi.org/10.48550/arXiv.2103.01782
[14] Gunda SK, Yettapu SDR, Bodakunti S, Bikki SB. Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management, 2023 Mar. 30;4(1):102-8. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112
[15] B. Li et al., "Enjoy your Observability: An Industrial Survey of Microservice Tracing and Analysis," Empirical Software Engineering, 2022, doi: 10.1007/s10664-021-10063-9.
[16] R. Xin, P. Chen and Z. Zhao, "CausalRCA: Causal Inference Based Precise Fine grained Root Cause Localization for Microservice Applications," Journal of Systems and Software, 2023, doi: 10.1016/j.jss.2023.111724.
[17] A. Basiri et al., "Chaos Engineering," IEEE Software, vol. 33, no. 3, pp. 35 41, 2016, doi: 10.1109/MS.2016.60.