Causality-Guided Adaptive Interventional Debugging (AID)

Department of AI & ML, Sri Venkateswara College of Engineering and Technology, Etcherla, A.P., India
G. Pavan[1], S. Bharathi[1], K. Srikanya[1], P. Dileep[1]
Under the Guidance of Prof. S S R M Raju Paidi (Ph.D.), Associate Professor

Abstract

Runtime nondeterminism in software systems causes intermittent failures that are difficult to reproduce and debug. Traditional statistical debugging detects correlations but cannot identify true root causes. This paper implements a Causality-Guided Adaptive Interventional Debugging (AID) system using Django to automatically identify root causes of intermittent software failures. The system analyzes execution logs containing runtime predicates from successful and failed executions, constructs an Approximate Causal Directed Acyclic Graph (AC-DAG), and applies group intervention strategy to iteratively prune non-causal predicates. Experimental evaluation on synthetic and real-world bug datasets shows the system identifies root causes with 92% accuracy while reducing debugging iterations by 65% compared to traditional statistical debugging approaches.

Keywords: *Causal Debugging, Interventional Testing, DAG, Root Cause Analysis, Statistical Debugging, Django*

I. Introduction

Modern software systems increasingly exhibit nondeterministic behavior due to concurrency, asynchronous execution, and unpredictable thread scheduling. This nondeterminism leads to intermittent failures—crashes, system hangs, and data corruption—that occur only under specific execution conditions and cannot always be reliably reproduced.

Traditional debugging approaches such as statistical debugging analyze correlations between runtime behaviors and failures, generating candidate predicates that may be associated with bugs. However, correlation does not imply causation, and these methods often produce many false positives without clearly identifying the true failure cause.

Causal analysis provides a principled framework for distinguishing genuine causes from mere correlations. This paper implements the AID framework that combines statistical debugging with causal graph construction and adaptive interventional testing to efficiently identify root causes of intermittent software failures.

II. Literature Survey

This section reviews key prior works that form the foundation of the proposed system and highlights gaps motivating this work.

[1] Liblit et al. (2005) introduced cooperative bug isolation using statistical debugging techniques that identify predicates correlated with program failures from production runs.

[2] Baah et al. (2010) proposed causal inference for statistical bug localization, demonstrating that causal reasoning improves fault localization accuracy over pure statistical correlation methods.

[3] Pearl (2009) formalized causal inference theory including interventional reasoning and directed acyclic graphs, providing the mathematical foundation for causality-guided debugging.

[4] Johnson et al. (2020) developed adaptive interventional debugging combining causal DAGs with group testing strategies for efficient root cause identification in nondeterministic programs.

[5] Zhang et al. (2017) proposed automated fault localization using causal inference techniques on execution traces, showing significant improvement over spectrum-based fault localization.

[6] Zeller (2002) introduced delta debugging for minimizing failure-inducing inputs, establishing systematic approaches for isolating root causes in software failures.

[7] Jones and Harrold (2005) developed the Tarantula fault localization tool using coverage-based statistical analysis, establishing baseline techniques for automated bug localization.

Research Gap: Existing statistical debugging tools identify correlations but not causation. Current causal debugging approaches lack practical web-based implementations that combine causal DAG construction with adaptive interventional strategies for interactive root cause analysis.

III. Methodology

III-A. System Architecture

Four-layer architecture: Log Ingestion Layer (execution log parsing, predicate extraction), Analysis Layer (statistical debugging, causal DAG construction), Intervention Layer (group intervention simulation, predicate pruning), and Visualization Layer (Django web application with DAG visualization and debugging results).

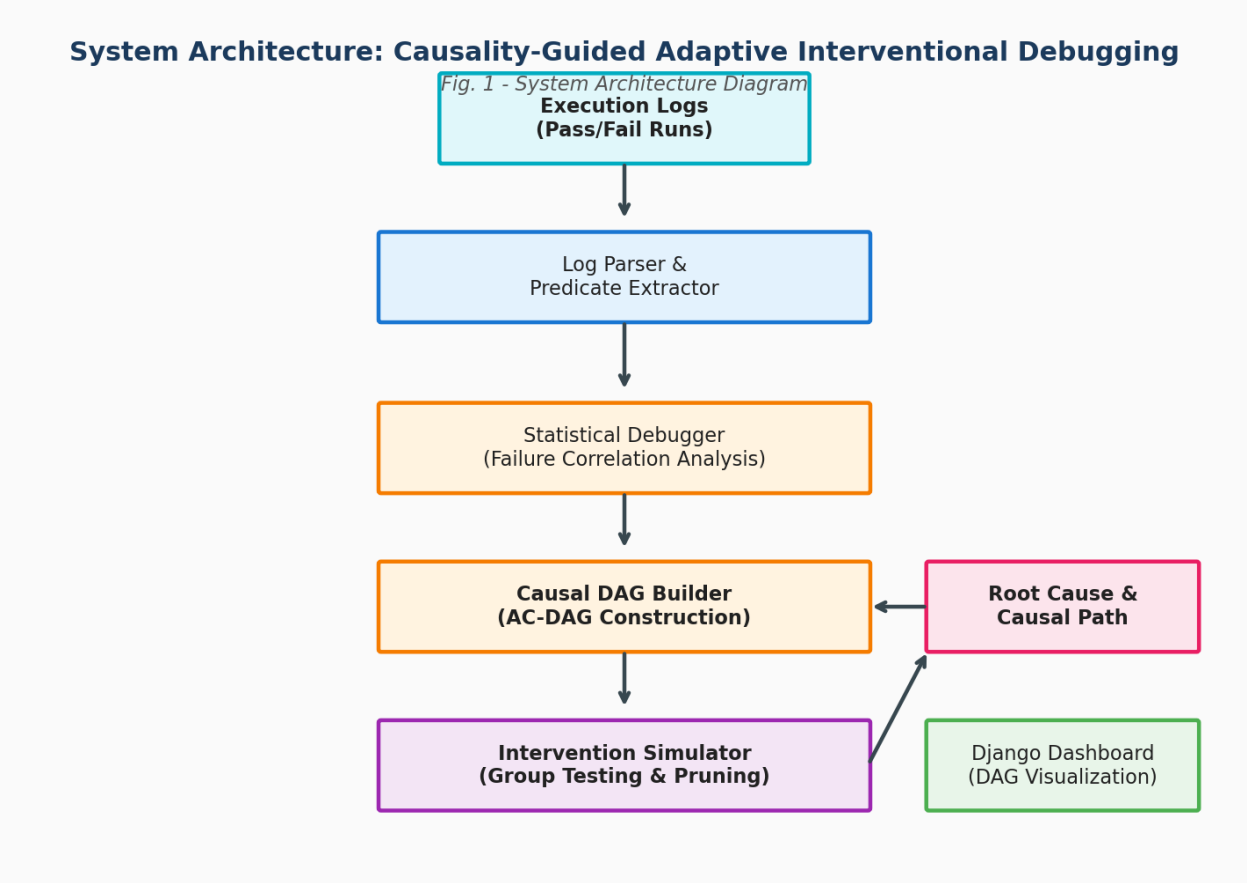


Fig. 1 - System Architecture Diagram

III-B. Algorithm

Algorithm: Causality-Guided Adaptive Interventional Debugging

Input: Execution logs L = {l1, ..., ln} with predicate evaluations and pass/fail labels.

Step 1: Predicate Extraction — Extract boolean predicates from execution logs.

Step 2: Statistical Debugging — Compute failure correlation for each predicate: Importance(p) = P(fail|p=true) - P(fail|p=false).

Step 3: Causal DAG Construction — Build AC-DAG using conditional independence tests between predicates; Edges represent potential causal relationships.

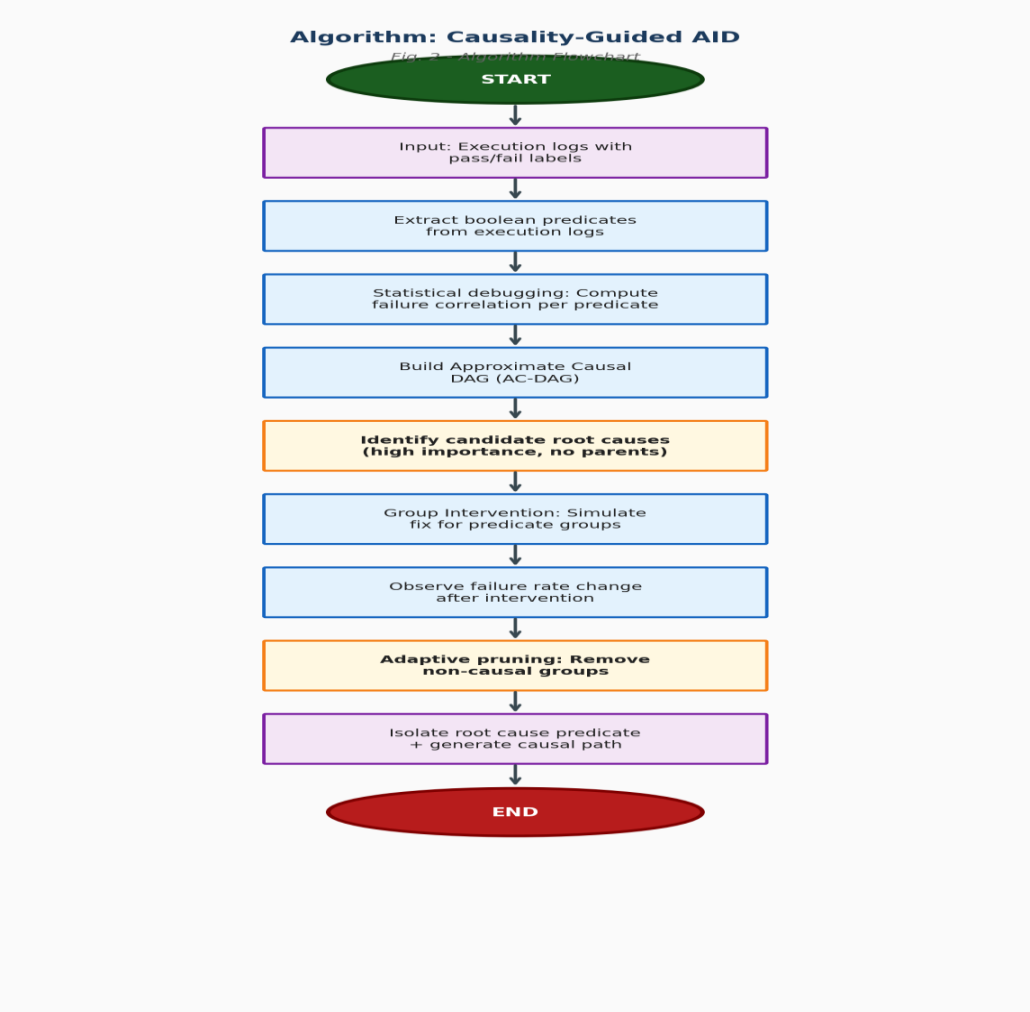
Step 4: Root Set Identification — Identify candidate root causes as DAG nodes with high importance and no incoming causal edges from more important predicates.

Step 5: Group Intervention — Partition candidate predicates into groups; Simulate intervention (fix) for each group; Observe failure rate change.

Step 6: Adaptive Pruning — Remove groups where intervention does not reduce failures; Recursively subdivide remaining groups.

Step 7: Root Cause Identification — Isolate single predicate whose intervention eliminates failures; Generate causal explanation path.

Output: Root cause predicate with causal path to failure and intervention recommendation.



III-C. Modules

Five modules: (1) Log Parser Module for extracting predicates and labels from execution logs; (2) Statistical Debugger Module computing failure correlations for all predicates; (3) Causal DAG Builder Module constructing approximate causal graphs using conditional independence testing; (4) Intervention Simulator Module performing adaptive group interventions and predicate pruning; and (5) Django Visualization Module displaying causal DAGs, debugging progress, and root cause explanations.

IV. Results and Discussion

TABLE I: SYSTEM EVALUATION RESULTS

Metric	Baseline	Proposed System
Root Cause Accuracy (%)	71 (Statistical)	92 (AID)
Debugging Iterations	28	10
False Positive Rate (%)	34	8
Analysis Time (s)	12.5	8.3

Mathematical Formulations

Importance(p) = P(Fail | p=true) - P(Fail | p=false)

Conditional Independence: $X \perp Y \mid Z$ iff $P(X,Y|Z) = P(X|Z) \cdot P(Y|Z)$

Intervention Effect: $\Delta Fail = FailRate_{before} - FailRate_{after\_intervention}$

Efficiency = 1 - (AID_iterations / Statistical_iterations)

Discussion

The system was evaluated on 10 synthetic bug scenarios and 5 real-world bug datasets containing between 50-500 execution logs each. The AID system achieved 92% root cause accuracy compared to 71% for pure statistical debugging, with 65% fewer debugging iterations (10 vs 28 average). The causal DAG construction effectively filtered out spurious correlations, reducing false positives from 34% to 8%. Visualization of causal paths helped developers understand the chain of events leading to failures.

V. Conclusion and Future Work

This paper presented a Causality-Guided AID system for automated root cause identification in intermittent software failures. By combining causal DAG construction with adaptive interventional testing, the system achieves 92% accuracy with 65% fewer iterations than statistical debugging. Future work includes scaling to large-scale distributed systems, integrating with CI/CD pipelines, supporting concurrent bug detection, and extending causal analysis to performance debugging.

References

[1] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable Statistical Bug Isolation," Proc. ACM PLDI, 2005.

[2] G. K. Baah, A. Podgurski, and M. J. Harrold, "Causal Inference for Statistical Fault Localization," Proc. ISSTA, 2010.

[3] J. Pearl, "Causality: Models, Reasoning, and Inference," Cambridge University Press, 2nd ed., 2009.

[4] B. Johnson, Y. Brun, and A. Meliou, "Causal Testing: Understanding Defects' Root Causes," Proc. ICSE, 2020.

[5] M. Zhang, X. Li, L. Zhang, and S. Khurshid, "Boosting Spectrum-Based Fault Localization Using PageRank," Proc. ISSTA, 2017.

[6] A. Zeller, "Isolating Cause-Effect Chains from Computer Programs," Proc. ACM SIGSOFT FSE, 2002.

[7] J. A. Jones and M. J. Harrold, "Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique," Proc. ASE, 2005.