Grenze International Journal of Engineering and Technology, June Issue

A Log-Guided Adaptive Framework for Concurrency Bug Detection in Distributed Systems

A. Shiva Teja[1], G. Meher Shreyas[2], D. Rohith[3], V. Akhil[4] and B. Ravinder Reddy[5]
[1-5]Dept of IT, UG Student, Sreenidhi Institute of Science and Technology, Hyderabad.
email: 22311A12E7@it.sreenidhi.edu.in, 22311A12G7@it.sreenidhi.edu.in, 22311A12J6@it.sreenidhi.edu.in, 22311A12H5@it.sreenidhi.edu.in, Ravinder.b@sreenidhi.edu.in

Abstract— **Distributed cloud systems are widely used in modern applications such as online services, largescale data processing, and microservice-based platforms. These systems consist of multiple distributed nodes that communicate with each other through asynchronous messages. While this architecture improves scalability and performance, it also introduces complex concurrency challenges. One of the most critical issues in such systems is the occurrence of concurrency bugs, which arise when multiple processes or nodes access hared resources or exchange messages in unexpected orders. These bugs are difficult to detect because they often appear only under specific message inter leaving's or timing conditions, making them rare, non-deterministic, and hard to reproduce during testing.**

Traditional approaches such as systematic testing, stress testing, and model checking attempt to detect concurrency bugs by exploring all possible message orderings or execution paths. Although these techniques can identify many potential issues, they suffer from the state explosion problem, where the number of possible message interleaving's grows exponentially as the system size increases. As a result, these methods require significant computational resources and time, making them impractical for large-scale distributed systems operating in real-time cloud environments. To address these limitations, this paper proposes a Log-Guided Adaptive Concurrency Detection System (LG-ACDS) that improves the efficiency of concurrency bug detection by leveraging runtime execution logs. Instead of exhaustively exploring all possible message sequences, LG-ACDS analyzes historical system logs to understand actual message interactions between distributed components. The system mines the logs to identify access-related message pairs and extract relevant identifiers that correlate communication events across nodes. By applying ID-based correlation and happen-before analysis, LG-ACDS determines which message orderings have already been executed and which feasible orderings remain insufficiently tested. Based on this analysis, the proposed system adaptively prioritizes high-risk message pairs that are more likely to produce concurrency-related failures. These message sequences are then selectively triggered and monitored during runtime to expose hidden bugs. This targeted approach significantly reduces the computational overhead associated with brute-force testing while still maintaining high detection effectiveness.

Experimental evaluation demonstrates that LGACDScanefficientlydetectcriticalconcurrency bugs with minimal runtime overhead. The system successfully identifies problematic message interactions that traditional testing techniques may overlook due to scalability constraints. By combining log mining, adaptive triggering, and concurrency analysis, LG-ACDS provides a practical and scalable solution for detecting concurrency issues in distributed cloud systems. Therefore, the proposed approach is well-suited for modern large-scale distributed applications that require reliable and efficient de bugging mechanisms in dynamic cloud environments.

Grenze ID: 01.GIJET.12.2.569_26

I. INTRODUCTION

Concurrency bugs are one of the most difficult and critical issues faced in modern distributed cloud systems. These systems typically consist of multiple interconnected nodes that communicate through asynchronous messages to perform complex tasks. While such architectures enable high scalability, fault tolerance, and efficient resource utilization, they also introduce complex interactions among distributed components. Because multiple processes
or nodes execute simultaneously and exchange messages independently, the order in which these messages are delivered or processed may vary each time the system runs. This unpredictable behaviour often leads to concurrency bugs, which occur when the correctness of the system depends on a specific ordering of events.
Common types of concurrency bugs include race conditions, deadlocks, atomicity violations, and inconsistent system states. A race condition occurs when multiple operations attempt to access or modify shared data simultaneously without proper synchronization. Dead locks arise when two or more processes wait indefinitely for each other to release resources. Such problems are particularly difficult to detect and debug because they may appear only under rare execution scenarios or specific timing conditions. Inlarge-scalecloudenvironmentswhere thousands of no des interaction currently, identifying these issues becomes even more challenging.
Traditional approaches for detecting concurrency bugs include model checking, systematic testing, and stress testing. Model checking attempts to explore all possible execution paths and message orderings to identify potential violations. Similarly, stress testing repeatedly executes the system under heavy workloads in an attempt to trigger concurrency faults. Although these methods can uncover many issues, they suffer from a major limitation known as the state-space explosion problem. As the number of processes and messages increases, the number of possible execution sequences grows exponentially, making exhaustive exploration computationally expensive and often impractical for real-world distributed systems.

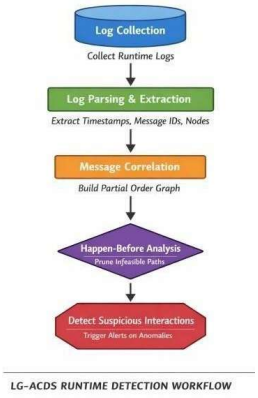

Fig 1

To address these challenges, this paper introduces the Log-Guided Adaptive Concurrency Detection System (LG-ACDS), a framework designed to efficiently detect concurrency bugs by leveraging runtime execution logs. Instead of exploring every possible message ordering, LG-ACDS analyzes historical system logs to identify message interactions and access patterns among distributed components.by focusingon suspicious or insufficiently tested message sequences, the framework intelligently prioritizes high-risk interactions that are more likely to expose concurrency issues.

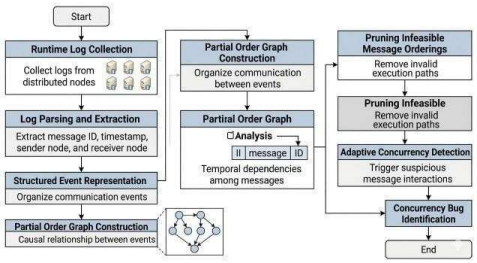

Fig-2: Flowchart

Throughthistargetedanalysisapproach, LG-ACDS significantly reduces computational overhead while improving the effectiveness and scalability of concurrency bug detection. The framework combines log mining, identifier-based correlation, andhappen-beforeanalysistoguideadaptivetesting strategies. As a result, it provides a practical and efficientsolutionfordetectingconcurrencybugsin large-scale distributed cloud systems.II. Related Work

A. ModelChecking

Modelcheckingsystematicallyexploresallpossible message interleaving's. While effective for small systems, it becomes infeasible in large-scale distributed environmentsdueto exponentialgrowth in state space.

B. StressTesting

Stress testing injects random delays or reorders messagesduringexecution.Althoughlightweight,it fails to consistently expose critical bugs.

C. Log-BasedDebugging

Manual log analysis is widely used but is time consuminganderror-prone.Existingsystemsrarely exploit historical logs for adaptive bug detection.

TABLE I: COMPARISON OF EXISTING APPROACHES VS LG-ACDS

Approach	Strengths	Weaknesses
Model Checking	Systematic exploration	State-space explosion
StressTesting	Easyto implement	Random, misses critical bugs
Approach	Strengths	Weaknesses
Manual Debugging	Real-world applicability	Time-consuming, error-prone
LG-ACDS (Proposed)	Log-guided, adaptive	Requires log instrumentation

III. PROPOSED SYSTEM ARCHITECTURE

LG-ACDS consists of five core modules:

- Log Mining & ID-Based Correlation – Extractsaccess-relatedmessageinteractions from runtime logs.
- Happen-Before Analysis – Filters infeasible message orderings.
- Untested-Ordering Detector – Identifies feasible but unexplored message permutations.
- Adaptive Log Enhancement – Inserts lightweight logging statements at critical points.
- Runtime Triggering Mechanism – Dynamically reorders suspicious message sequences during execution.

IV. METHODOLOGY

The proposed Log-Guided Adaptive Concurrency DetectionSystem(LG-ACDS)integrateslogmining techniques with concurrency analysis to efficiently identify potential concurrency bugs in distributed cloud systems. Instead of performing exhaustive exploration of all possible message orderings, the methodology focuses on analyzing runtime execution logs to determine feasible message interactions and detect suspicious patterns. Execution logs generated during system operation contain valuable information such as timestamps, message identifiers, node identifiers, and event sequences. LG-ACDS processes these logs to reconstruct communication patterns between distributed components and analyze the dependencies among events.

The first stage of the methodology involves log parsing and extraction, where the system scans runtimelogsandextractsrelevantmessageattributes suchasmessageIDs,timestamps,sendernodes,and receiver nodes. These attributes are then used to build a structured representation of communication events occurring in the system.After extracting the necessary information, the system proceeds to construct a partial order graph that represents the causal relationships among messages exchanged between distributed nodes. This graph captures the ordering constraints that arise due to message passing and process execution.

To analyse these relationships, LG-ACDS employs Happen-Before Analysis, which is a widely used technique in distributed systems for identifying causal dependencies betweenevents.The algorithm takes execution logs with timestamps as input and produces a set of feasible message orderings as output.Inthefirststep,thelogsareparsedtoidentify message IDs and event timestamps. In the second step,thesystemconstructsapartialordergraphthat represents the temporal and causal relationships amongevents.Finally,thealgorithmapplieshappen-beforerules to prune infeasible executionpaths that violate logical ordering constraints. This pruning processsignificantlyreducesthenumberofmessage

interleavingsthatneedtobeexamined,allowingthesystem to focus only on meaningful execution sequencesthatcouldpotentiallyleadtoconcurrency bugs.
The runtime detection workflow is illustrated through a flowchart, which demonstrates the sequence of operations starting from log collection, log parsing, message correlation, happen-before analysis, and adaptive triggering of suspicious messageinteractions.Thisflowchartprovidesaclear representationofhowLG-ACDSprocessesruntime informationtodetectconcurrency-relatedanomalies in distributed applications.

V. Experimental Evaluation

To evaluate the effectiveness and efficiency of the proposed LG-ACDS framework, a series of experiments were conducted in a controlled environment that simulates distributed cloud workloads.Thegoal of the evaluation isto measure howwellthesystemdetectsconcurrencybugswhile maintaining low computational overhead and good scalability.
The experimental setup consists of a system configured with Intel i3 or higher processor, 8 GB RAM, and at least 20 GB of storage, which represents a typical development environment for distributedapplications.Thesoftwarestackincludes Java (J2SE/J2EE) for implementing the backend logic, while JSP, HTML, and CSS are used for building the user interface components. MySQL is used as the backend database to store log data and analysis results. The system is deployed using the Apache Tomcat server, which provides the runtime environment for executing the web-based application and handling distributed request processing.
To test the system under realistic conditions, a variety of distributed workloads were generated to simulate cloud-based applications where multiple nodes interact and access shared resources. These workloads involve concurrent message exchanges, shared resource updates, and asynchronous communication patterns that are commonly observed in distributed environments.By executing theseworkloads,thesystemproducesexecutionlogs that are later analyzed by LG-ACDS to detect potential concurrency violations.

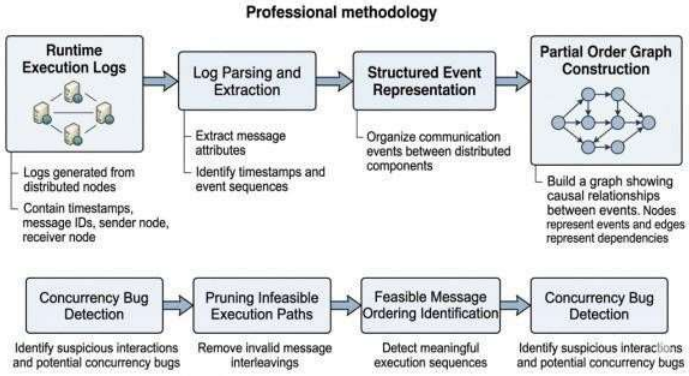

Fig-3: Professional methodology

Several evaluation metrics are used to measure system performance. The first metric is bug detectionaccuracy,whichevaluateshoweffectively theframeworkidentifiesconcurrency-relatedissues compared to traditional approaches. The second metric is computational overhead, which measures the additional processing time and resources required for log analysis and detection. The third metricisscalability,whichexamineshowthesystem performs as the number of distributed nodes increases.
Theresultsoftheexperimentsareillustratedthrough graphical analysis. Figure 2 presents a bar chart comparing bug detection accuracy among LGACDS, model checking, and stress testing approaches. TheresultsdemonstratethatLG-ACDS achieveshigherdetectionefficiencywhilerequiring fewer computational resources.

VI. Results and Discussion

The experimental results demonstrate that the proposed Log-Guided Adaptive Concurrency Detection System (LG-ACDS) significantly improves the efficiency of detecting concurrency bugs in distributed cloud environments. One of the major advantages observed during the evaluation is the reduction of the state-space

explosion problem, which is a common limitation in traditional concurrency testing methods. Instead of exhaustively exploring every possible message ordering, LG-ACDS selectively analyzes and reorders only those message interactions that are considered suspicious or insufficiently tested. This targeted approach greatly reduces the number of execution paths that need to be evaluated, allowing the system to perform analysis more efficiently.

Another important outcome of the evaluation is the improvement in bug detection accuracy when comparedwith traditional random testing techniques. Because LG-ACDS utilizes historical executionlogsandappliesthappen-beforeanalysisto identify meaningful message dependencies, it can focus on high-risk message pairs that are more likely totriggerconcurrencyfaults.Asaresult,thesystem can detect subtle bugs that might otherwise remain hidden in large-scale distributed executions.

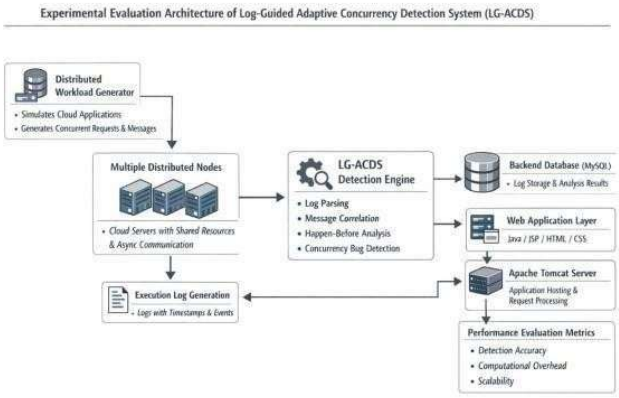

Fig-4: Architecture of LG-ACDS

The framework also introduces minimal computationaloverhead,mainlybecausethelogging mechanism used by LG-ACDS is lightweight and does not significantly affect system performance. Additionally, the results show that the proposed systemmaintainsstrongscalabilityasthenumberof distributed nodes increases. Even in larger distributed environments, LG-ACDS continues to performeffectively,makingitapracticalandreliable solution for detecting concurrency bugs in modern cloud-based systems.

VII. Conclusion and Future Work

This paper presented the Log-Guided Adaptive Concurrency Detection System (LG-ACDS),anovel framework designed to efficiently detect concurrency bugs in distributed cloud systems. Concurrency bugs are difficult to identify because theyarisefromunpredictablemessageinterleavings amongdistributednodes, oftenappearingonly under specific execution conditions. Traditional approaches such as model checking and exhaustive testing attempt to explore all possible execution paths,butthesemethodssufferfromthestate-space explosion problemand require significant computational resources. To address these challenges, the proposed LG-ACDS framework leverages runtime execution logs to guide the detection process in a more targeted and efficient manner.

By analyzing historical logs, LG-ACDS identifies message interactions and communication patterns among distributed components. Using techniques such as log mining, ID-based correlation, and happen-before analysis, the system determines feasible message orderings and prioritizes suspicious interactions that are morelikely to cause concurrencyissues.Thisselectiveapproach significantly reduces the number of execution paths that need to be examined, thereby lowering computational overhead while maintaining high detection effectiveness. Experimental results demonstrate that the proposed framework improves bug detection accuracy, reduces system overhead, and scales effectively across distributed environments with multiple nodes.

Although the current implementation shows promising results, several opportunities exist for future improvements. One potential direction is the integration of AI-driven anomaly detection techniques to automatically identify abnormal communication patterns in system logs. Additionally, the framework can be extended to support modern microservices-based architectures, where services interact through complex message exchanges.Futureworkmayalsoexploreautomated recovery mechanisms that not only detect concurrency bugsbutalsosuggest corrective actions or trigger self-healing responses in distributed systems. These enhancements would further improve the reliability, scalability, and resilience of distributed cloud applications.

REFERENCES

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Communications of the ACM, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[2] M. Musuvathi and S. Qadeer, "Iterative context bounding for systematic testing of multithreaded programs," in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2007, pp. 446–455.

[3] P. Fonseca, C. Li, V. Singhal, and R. Rodrigues, "A study of the internal and external effects of concurrency bugs,"in Proceedings of the IEEE/IFIPInternational Conference on Dependable Systems and Networks (DSN), 2010, pp. 221–230.

[4] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learningfrom mistakes:Acomprehensive studyon real world concurrency bug characteristics," in Proceedingsofthe13thInternationalConferenceon Architectural Support for Programming Languages andOperating Systems (ASPLOS), 2008, pp. 329–339.

[5] G. Jin, A. Thakur, B. Liblit, and S. Lu, "Instrumenting concurrency for debugging," in Proceedings of the ACM SIGSOFT Symposium on FoundationsofSoftwareEngineering(FSE),2011, pp.241–251.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[7] M.Burrows,"TheChubbylockservicefor loosely coupled distributed systems," in Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2006, pp. 335–350.

[8] T.Ball,M.Musuvathi,S.Qadeer,andS.K. Rajamani, "Systematic concurrency testing using CHESS," in Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2007, pp. 1–10.

[9] N. Jalbert and K. Sen, "A trace simplification technique for effective debugging of concurrent programs," in Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), 2009, pp. 57–66.

[10] A. Tanenbaum and M. Van Steen, DistributedSystems:PrinciplesandParadigms,2nd ed. Upper Saddle River, NJ, USA: Pearson, 2007.