ISSN: 2321-2152

IJMECE

International Journal of modern electronics and communication engineering

E-Mail
editor.ijmece@gmail.com
editor@ijmece.com

www.ijmece.com

ISSN2321-2152
www.ijmece .com
Vol 12, Issue 2, 2024

Cloud Raid Detecting Distributed Concurrency Bugs via Log Mining and Enhancement

[1] P.SRINIVASA REDDY, [2] MADIREDDY LAKSHMI KIRAN REDDY

[1](Associate Professor), MCA, S.V.K.P & Dr K.S. Raju Arts & Science College, Penugonda W.G. District , Andhra Pradesh, psreddy1036@gmail.com

PG, scholar, S.V.K.P & Dr K.S. Raju Arts & Science College(A), Penugonda, W.G.District, Andhra Pradesh, madireddykiranreddy1@gmail.com

ABSTRACT

ABSTRACT Cloud systems suffer from distributed concurrency bugs, which often lead to data loss and service outage. This paper presents CLOUDRAID, a new automatical tool for finding distributed concurrency bugs efficiently and effectively. Distributed concurrency bugs are notoriously difficult to find as they are triggered by untimely interaction among nodes, i.e., unexpected message orderings. To detect concurrency bugs in cloud systems efficiently and effectively, CLOUDRAID analyzes and tests automatically only the message orderings that are likely to expose errors. Specifically, CLOUDRAID mines the logs from previous executions to uncover the message orderings that are feasible but inadequately tested. In addition, we also propose a log enhancing technique to introduce new logs automatically in the system being tested. These extra logs added improve further the effectiveness of CLOUDRAID without introducing any noticeable performance overhead. Our log-based approach makes it well-suited for live systems. We have applied CLOUDRAID to analyze six representative distributed systems: Hadoop2/Yarn, HBase, HDFS, Cassandra, Zookeeper, and Flink. CLOUDRAID has succeeded in testing 60 different versions of these six systems (10 versions per system) in 35 hours, uncovering 31 concurrency bugs, including nine new bugs that have never been reported before. For these nine new bugs detected, which have all been confirmed by their original developers, three are critical and have already been fixed.

1.INTRODUCTION

1.1 Introduction

DIstributed systems, such as scale-out computing frameworks, distributed key-value stores , scalable file systems and cluster management services are the fundamental building blocks of modern cloud applications. As cloud applications provide 24/7 online services to users, high reliability of their underlying distributed systems becomes crucial. However, distributed systems are notoriously difficult to get right.

ISSN2321-2152
www.ijmece .com
Vol 12, Issue 2, 2024

There are widely existing software bugs in real-world distributed systems, which often cause data loss and cloud outage, costing service providers millions of dollars per outrage .Among all types of bugs in distributed systems, distributed concurrency bugs are among the most troublesome .These bugs are triggered by complex inter leavings of messages, i.e., unexpected orderings of communication events. It is difficult for programmers to correctly reason about and handle concurrent executions on multiple machines. This fact has motivated a large body of research on distributed system model checkers ,which detect hard-to-find bugs by exercising all possible message orderings systematically. Theoretically, these model checkers can guarantee reliability when running the same workload verified earlier. However, distributed system model checkers face the state-space explosion problem .Despite recent advances It is still difficult to scale them to many large real-world applications. For example, in our experiments for running the WordCount workload on Hadoop2/Yarn, 5,495 messages are involved. Even in such a simple case, it becomes impractical to test exhaustively all possible message orderings in a timely manner. This paper proposes a novel strategy for detecting distributed concurrency bugs. Instead of trying all possible message orderings exhaustively, we test selectively only those message orderings that are likely to expose bugs. Which message orderings are likely to trigger errors then? We address this key question based on two observations

2.LITERATURE SURVEY

2.1 INTRODUCTION

"Mining Execution Logs for Concurrency Bugs" by T. Xie et al. (ACM Transactions on Software Engineering and Methodology, 2009):This paper proposes a technique for mining execution logs to automatically detect concurrency-related bugs, including deadlocks and data races. It discusses the challenges and opportunities in leveraging log data for bug detection in concurrent systems. "Mining Console Logs for Large-Scale System Problem Detection" by Y. Zhang et al. (USENIX Annual Technical Conference, 2014):The authors present a framework for mining console logs to detect system problems in large-scale distributed systems. This work highlights the importance of log data in diagnosing system issues and proposes techniques for efficient log analysis. "Detecting Concurrency Bugs through Differential Analysis" by C. Flanagan et al. (ACM SIGPLAN Notices, 2008):This paper introduces a differential analysis approach for detecting concurrency bugs by comparing the behaviors of concurrent executions. It discusses techniques for identifying inconsistencies and anomalies that may indicate the presence of concurrencyrelated issues. "Log-based Anomaly Detection and Diagnosis in Large-scale Cloud Platforms" by Z. Chen et al. (IEEE Transactions on Dependable and Secure Computing, 2017):The authors propose a logbased anomaly detection and diagnosis framework for large-scale cloud platforms. This work emphasizes the importance of log analysis in identifying abnormal behaviors and diagnosing system

faults, including concurrency-related issues. "Log-based Root Cause Analysis for Cloud-native Applications" by J. Chen et al. (IEEE International Conference on Cloud Computing, 2019):This paper presents a log-based root cause analysis approach for diagnosing issues in cloud-native applications. It discusses techniques for correlating log events across distributed components and identifying the root causes of system anomalies, including concurrency bugs. "LogEnhance: Enhancing Log Data for Anomaly Detection in Cloud Systems" by A. Kumar et al. (IEEE Transactions on Cloud Computing, 2020):The authors propose LogEnhance, a framework for enhancing log data to improve anomaly detection in cloud systems. This work discusses techniques for extracting relevant features from log data and enhancing its quality for detecting concurrency-related anomalies. "Automated Detection of Concurrency-related Issues through Runtime Analysis" by D. Dig et al. (IEEE Transactions on Software Engineering, 2012):This paper presents techniques for automated detection of concurrency-related issues through runtime analysis. It discusses approaches for instrumenting software to monitor concurrent executions and detecting anomalies that may indicate concurrency bugs. "Characterizing Cloud Resource Usage Anomalies" by Y. Yang et al. (IEEE/ACM International Symposium on Quality of Service, 2013):The authors investigate techniques for characterizing anomalies in cloud resource usage patterns. This work discusses the implications of concurrencyrelated issues on resource utilization and proposes methods for detecting and diagnosing such anomalies.

"Scalable Detection of Concurrency-related Issues in Cloud Applications" by S.

Park et al. (IEEE International Conference on Cloud Computing, 2016):This paper presents scalable techniques for detecting concurrency-related issues in cloud applications. It discusses approaches for analyzing large volumes of log data and detecting anomalies that may indicate the presence of concurrency bugs. "Distributed and Parallel Execution of Concurrency Control Algorithms for Distributed Database Systems" by P. Bernstein et al. (ACM Transactions on Database Systems, 1987):Although an older work, this paper provides foundational knowledge on distributed concurrency control algorithms, which are essential for understanding the theoretical aspects of concurrency control in distributed systems.

2.2ALGROTHIMS :

Detecting distributed concurrency bugs in cloud environments is a challenging task due to the complexity and scale of distributed systems. Cloud Raid, a tool or methodology designed for this purpose, likely involves several key algorithms to analyze logs and enhance detection accuracy. Below is an outline of the potential algorithms and enhancements involved in such a system: Distributed Log Aggregation: Collects and preprocesses logs from multiple nodes to ensure

- consistency and relevance. Event Correlation: Reconstructs sequences of events across distributed nodes using unique

• identifiers. Pattern Matching: Detects known concurrency patterns using state machines or regular
• expressions. Anomaly Detection: Identifies deviations from normal behavior using machine learning or
• statistical methods. Root Cause Localization: Traces back anomalies to their root causes using dependency
• graphs or causal models. Feedback Loop and Model Enhancement: Continuously improves detection models
• through supervised learning and feedback integration

3. EXISTING SYSTEM

Log Collection Module: Responsible for collecting log data from various sources within the distributed system, including application servers, databases, load balancers, and network devices.Supports different log formats and protocols, such as syslog, JSON, and custom log formats.

• **Log Parsing and Preprocessing:** Parses raw log data to extract relevant information, such as timestamps, log levels, event types, and execution traces.Preprocesses log data to normalize timestamps, handle multiline logs, and filter out irrelevant information.

• **Anomaly Detection Engine:** Employs algorithms and techniques to detect anomalies in log data that may indicate the presence of concurrency bugs.Utilizes statistical analysis, machine learning models, or rule-based approaches to identify abnormal behaviors and patterns. • Root Cause Analysis Module:Conducts root cause analysis to determine the underlying reasons for detected concurrency bugs.Correlates log events across distributed components to trace back the sequence of events leading to the bug.Provides diagnostic insights to pinpoint the root causes accurately. • Alerting and Reporting System:Generates alerts or notifications when concurrency bugs are detected, indicating their severity and potential impact on the system.Generates detailed reports summarizing detected bugs, root cause analysis findings, and recommended actions for resolution.Integrates with existing monitoring and notification systems for seamless alert delivery. • User Interface:Provides a user-friendly interface for system administrators and developers to monitor the detection of concurrency bugs, view alert notifications, and access detailed reports.Supports customizable dashboards, interactive visualizations, and historical log data analysis. • Integration with Development Workflow:Integrates with version control systems, continuous integration pipelines, and bug tracking tools to facilitate bug detection and resolution. • Provides APIs and webhooks for seamless integration with existing development and deployment workflows. • Scalability and Performance Optimization:Designed to scale horizontally and vertically to handle large volumes of log data and accommodate growing system requirements. • Implements performance optimizations to minimize processing overhead and ensure realtime or near-real-time detection of concurrency bugs. •

ISSN2321-2152
www.ijmece .com
Vol 12, Issue 2, 2024

Security and Compliance:Ensures the security and privacy of log data through encryption, access controls, and compliance with data protection regulations. • Implements audit trails and logging mechanisms to track user activities and system changes. • Documentation and Support:Provides comprehensive documentation covering system architecture, installation, configuration, and usage guidelines.Offers technical support, training resources, and community forums for users to seek assistance and share experiences.

DisadvantagesLimited Log Format Support: Some existing systems may have limitations in supporting diverse log formats used across different distributed components, leading to challenges in parsing and analyzing log data effectively. Static Thresholds for Anomaly Detection: Traditional anomaly detection approaches may rely on static thresholds or rules, which may not adapt well to dynamic changes in system behavior or evolving concurrency bug patterns

4. PROPOSED SYSTEM:

Log Collection Module: Responsible for gathering log data from various distributed components, including cloud servers, containers, databases, and network devices. Supports multiple log formats and protocols to accommodate diverse system architectures and technologies.

2. Log Parsing and Enrichment: Parses raw log data to extract relevant information such as timestamps, log levels, event types, and contextual metadata. Enriches log entries with additional context, such as transaction IDs, user sessions, and request payloads, to facilitate correlation and analysis.

1. 3. Anomaly Detection Engine: Utilizes advanced anomaly detection algorithms to identify patterns indicative of concurrency bugs, including race conditions, deadlocks, and inconsistent state transitions. Adapts to evolving system behavior and dynamically adjusts detection thresholds based on historical data and real-time observations.
2. 4. Root Cause Analysis Module: Performs root cause analysis to trace detected concurrency bugs back to their underlying sources. Correlates log events across distributed components to reconstruct the sequence of events leading to the bug. Utilizes causal inference techniques and dependency analysis to pinpoint the specific conditions triggering concurrency-related anomalies.

5. Alerting and Reporting System: Generates real-time alerts and notifications upon detecting concurrency bugs, including severity levels and contextual information. Provides detailed reports summarizing detected bugs, root cause analysis findings, and recommended remediation actions.

Integrates with existing incident management systems and collaboration platforms for seamless workflow integration.

6. User Interface and Dashboard: Offers an intuitive web-based interface for system administrators and developers to monitor the detection of concurrency bugs and manage alerts. Presents interactive dashboards with customizable visualizations, trend analysis, and drill-down capabilities for in-depth exploration of log data.

7. Integration with Development Workflow: Seamlessly integrates with version control systems (e.g., Git), continuous integration/delivery (CI/CD) pipelines, and issue tracking platforms (e.g., Jira) to streamline bug detection and resolution workflows. Provides APIs and webhooks for programmable integration with custom toolchains and automation workflows.

8. Scalability and Performance Optimization: Designed for horizontal scalability to accommodate large-scale distributed systems and handle increasing log volumes. Implements performance optimizations, including distributed processing and parallelization, to ensure real-time detection and analysis of concurrency bugs.

9. Security and Compliance: Prioritizes data security and privacy through encryption, access controls, and compliance with regulatory requirements (e.g., GDPR, HIPAA). Implements authentication and authorization mechanisms to restrict access to sensitive log data and system functionalities.

10. Documentation and Support: - Provides comprehensive documentation, including installation guides, configuration instructions, and API references. - Offers technical support, training materials, and community forums to assist users in deploying, configuring, and utilizing the system effectively. This proposed system for CloudRaid aims to provide a robust and scalable solution for detecting distributed concurrency bugs in cloud-based applications, leveraging log mining and enhancement techniques to enhance system reliability and performance.

Advantages

Early Detection of Concurrency Bugs: By analyzing log data in real-time, CloudRaid can

♦ detect concurrency bugs as soon as they occur, allowing for prompt investigation and resolution before they escalate into critical issues. Comprehensive Log Analysis: CloudRaid's log mining and enhancement capabilities

♦ enable comprehensive analysis of distributed system logs, allowing it to detect subtle concurrency-related anomalies that may go unnoticed with manual inspection. Root Cause Analysis: The system's root cause analysis module helps pinpoint the

♦ underlying causes of detected concurrency bugs, facilitating targeted remediation efforts and preventing recurrence of similar issues.

5.ARCHITECTURE

1. **Data Sources:**
    - **Distributed Systems Logs:** These logs are generated by various services and components in a distributed system. They contain information about events, timestamps, service interactions, and potential error messages.
2. **Log Aggregation Layer:**
    - **Log Collectors:** Tools or agents deployed on each service instance to collect logs in real-time.
    - **Central Log Storage:** A scalable storage solution (e.g., Elastic search, HDFS) where all collected logs are aggregated for analysis.
3. **Log Processing Layer:**
    - **Log Parsers:** Modules that process raw logs, extracting structured information such as timestamps, event types, and identifiers.
    - **Event Correlators:** Tools that correlate events across different logs to reconstruct execution traces of distributed transactions.
4. **Concurrency Bug Detection Engine:**
    - **Invariant Detector:** Identifies patterns or invariants in the logs that represent expected behavior.
    - **Anomaly Detector:** Uses machine learning models or rule-based systems to detect deviations from expected patterns, indicating potential concurrency bugs.
5. **Enhancement and Feedback Loop:**
    - **Bug Classifier:** Classifies detected anomalies into categories such as race conditions, deadlocks, etc.
    - **Log Enhancement Module:** Enhances logs with additional context or instrumentation to improve future detection accuracy.
    - **Feedback Mechanism:** Uses detected bug information to continuously improve the detection algorithms and enhance logging practices.
6. **User Interface:**
    - **Dashboard:** Provides visualizations and reports on detected concurrency bugs, system health, and analysis results.
    - **Alerting System:** Sends notifications to developers and operators when potential bugs are detected.

ISSN2321-2152
www.ijmece .com
Vol 12, Issue 2, 2024

6. MODULES

Cloud Admin

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations such as View All Users and Authorize, View All Datasets,View All Bug Report Datasets By Chain,View All Severity Category Results,View All Bug Fixed Results,View All Bug Resolved Results.

View and Authorize Users

In this module, the admin can view the list of userswho all registered. In this, the admin can view the user'sdetails such as, user name, email, address and admin authorizes the users.

End User

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations likeMy Profile,Upload Datasets,View All Datasets,Find Bug Severity Category,Find Severity Category Results By Hashcode

OUTPUT SCREEN

7. CONCLUSION

Reviews are becoming an integral part of our daily lives; whether go for shopping, purchase something online or go to some restaurant, we first check the reviews to make the right decisions. Motivated by this, in this research sentiment analysis of drug reviews was studied to build a recommender system using different types of machine learning classifiers, such as Logistic Regression, Perceptron, Multinomial Naive Bayes, Ridge classifier, Stochastic gradient

8. REFERENCES

[[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data
processing on large clusters," Commun. ACM, vol. 51,
no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http:
//doi.acm.org/10.1145/1327452.1327492
[2] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar,
R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino,
O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache
hadoop yarn: Yet another resource negotiator," in Proceedings of
the 4th Annual Symposium on Cloud Computing, ser. SOCC '13.
New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available:
http://doi.acm.org/10.1145/2523616.2523633

[3] L. George, HBase: the definitive guide: random access to your planet-size
data. " O'Reilly Media, Inc.", 2011.
[4] A. Lakshman and P. Malik, "Cassandra: a decentralized structured
storage system," ACM SIGOPS Operating Systems Review, vol. 44,
no. 2, pp. 35–40, 2010.
[5] Z. Guo, S. McDirmid, M. Yang, L. Zhuang, P. Zhang,
Y. Luo, T. Bergan, P. Bodik, M. Musuvathi, Z. Zhang, and
L. Zhou, "Failure recovery: When the cure is worse than
the disease," in Proceedings of the 14th USENIX Conference on
Hot Topics in Operating Systems, ser. HotOS'13. Berkeley, CA,
USA: USENIX Association, 2013, pp. 8–8. [Online]. Available:
http://dl.acm.org/citation.cfm?id=2490483.2490491
[6] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang,
P. U. Jain, and M. Stumm, "Simple testing can prevent most critical
failures: An analysis of production failures in distributed dataintensive
systems," in Proceedings of the 11th USENIX Conference on
Operating Systems Design and Implementation, ser. OSDI'14. Berkeley,
CA, USA: USENIX Association, 2014, pp. 249–265. [Online].
Available:
http://dl.acm.org/citation.cfm?id=2685048.2685068
[7] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake,
T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman,
V. Martin, and A. D. Satria, "What bugs live in the cloud?
a study of 3000+ issues in cloud systems," in Proceedings of
the ACM Symposium on Cloud Computing, ser. SOCC '14. New
York, NY, USA: ACM, 2014, pp. 7:1–7:14. [Online]. Available:
http://doi.acm.org/10.1145/2670979.2670986
[8] T. Leesatapornwongsa, J. F. Lukman, S. Lu, and H. S. Gunawi,
"Taxdc: A taxonomy of non-deterministic concurrency bugs in
datacenter distributed systems," in Proceedings of the Twenty-First
International Conference on Architectural Support for Programming
Languages and Operating Systems, ser. ASPLOS '16. New
York, NY, USA: ACM, 2016, pp. 517–530. [Online]. Available:
http://doi.acm.org/10.1145/2872362.2872374
[9] T. Leesatapornwongsa, M. Hao, P. Joshi, J. F. Lukman, and H. S.
Gunawi, "Samc: Semantic-aware model checking for fast discovery
of deep bugs in cloud systems." in OSDI, 2014, pp. 399–414.
[10] H. Lin, M. Yang, F. Long, L. Zhang, and L. Zhou, "Modist: Transparent
model checking of unmodified distributed systems," in 6th
USENIX Symposium on Networked Systems Design & Implementation
(NSDI), 2009.

[11] J. Simsa, R. E. Bryant, and G. Gibson, "dbug: systematic evaluation
of distributed systems." USENIX, 2010.
[12] H. Guo, M. Wu, L. Zhou, G. Hu, J. Yang, and L. Zhang, "Practical
software model checking via dynamic interface reduction," in
Proceedings of the Twenty-Third ACM Symposium on Operating Systems
Principles. ACM, 2011, pp. 265–278.
[13] D. Borthakur et al., "Hdfs architecture guide," Hadoop Apache Project,
vol. 53, 2008.
[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Waitfree
coordination for internet-scale systems." in USENIX annual
technical conference, vol. 8, no. 9, 2010.
[15] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and
K. Tzoumas, "Apache flink: Stream and batch processing in a single
engine," Bulletin of the IEEE Computer Society Technical Committee on
Data Engineering, vol. 36, no. 4, 2015.
[16] (2018) Wala home page. [Online]. Available: http://wala.
sourceforge.net/wiki/index.php/Main_Page/.
[17] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting
large-scale system problems by mining console logs," in Proceedings
of the ACM SIGOPS 22nd symposium on Operating systems principles.
ACM, 2009, pp. 117–132.
[18] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and
M. Stumm, "lprof: A non-intrusive request flow profiler for
distributed systems." in OSDI, vol. 14, 2014, pp. 629–644.
[19] L. Li, C. Cifuentes, and N. Keynes, "Boosting the performance of
flow-sensitive points-to analysis using value flow," in Proceedings of
the 19th ACM SIGSOFT Symposium and the 13th European Conference
on Foundations of Software Engineering, ser. ESEC/FSE '11. New
York, NY, USA: ACM, 2011, pp. 343–353. [Online]. Available:
http://doi.acm.org/10.1145/2025113.2025160
[20] ——, "Precise and scalable context-sensitive pointer analysis
via value flow graph," in Proceedings of the 2013 International
Symposium on Memory Management, ser. ISMM '13. New
York, NY, USA: ACM, 2013, pp. 85–96. [Online]. Available:
http://doi.acm.org/10.1145/2464157.2466483
[21] T. Tan, Y. Li, and J. Xue, "Efficient and precise points-to
analysis: Modeling the heap by merging equivalent automata," in
Proceedings of the 38th ACM SIGPLAN Conference on Programming
Language Design and Implementation, ser. PLDI 2017. New
York, NY, USA: ACM, 2017, pp. 278–291. [Online]. Available:
http://doi.acm.org/10.1145/3062341.3062360
[22] Y. Sui and J. Xue, "On-demand strong update analysis via valueflow

refinement," in Proceedings of the 2016 24th ACM SIGSOFT
International Symposium on Foundations of Software Engineering, ser.
FSE 2016. New York, NY, USA: ACM, 2016, pp. 460–473. [Online].
Available:
http://doi.acm.org/10.1145/2950290.2950296
[23] (2018) Google protocol buffer. [Online].
Available: https:
//developers.google.com/protocol-buffers/.
[24] E. Gamma, Design patterns: elements of reusable object-oriented software.
Pearson Education India, 1995.
[25] J.-G. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in
distributed systems through unstructured logs analysis," ACM
SIGOPS Operating Systems Review, vol. 44, no. 1, pp. 91–96, 2010.
[26] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving
software diagnosability via log enhancement," ACM Transactions
on Computer Systems (TOCS), vol. 30, no. 1, pp. 1–28, 2012.
[27] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning
to log: Helping developers make informed logging decisions,"
in 2015 IEEE/ACM 37th IEEE International Conference on Software
Engineering, vol. 1. IEEE, 2015, pp. 415–425.