CLOUD RAID DETECTING DISTRIBUTED CONCURRENCY BUGS VIA LOG MINING AND ENHANCEMENT

MRS.S.KALYANI[1], RAGIPINDI HARI SANKARA REDDY[2], D CHIRAG RAO[3], N ROHIT[4]

ASSISTANT PROFESSOR[1], UG SCHOLAR[2,3&4]

DEPARTMENT OF CSE, CMR INSTITUTE OF TECHNOLOGY, KANDLAKOYA VILLAGE, MEDCHAL RD, HYDERABAD, TELANGANA 501401

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.

Index Terms— Distributed Systems, Concurrency Bugs, Bug Detection, Cloud Computing.

I. INTRODUCTION

Disturbed 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. 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 trouble-some. 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 Word Count 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.

II. LITERATURE SURVEY

A) 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

Depression MapReduce is a programming model designed to simplify data processing on large clusters. It abstracts the complexity of distributed computing and allows for efficient processing of vast amounts of data in parallel across a large number of machines. The model is based on two fundamental operations: Map, which applies a function to each element of the input data, and Reduce, which aggregates the results from the Map phase. This simple yet powerful model enables developers to focus on writing the core application logic, without having to manage the complexities of parallelization, fault tolerance, and load balancing. This paper discusses the design and implementation of MapReduce, the benefits of using this model, and the challenges of applying it to large-scale data processing tasks. The system has been proven to be highly scalable, fault-tolerant, and suitable for various real-world applications.

B) 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 *Proc. 4th Annu. Symp. Cloud Comput.*, ser. SOCC '13, New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

Apache Hadoop YARN (Yet Another Resource Negotiator) is a resource management layer for the Apache Hadoop framework. YARN is designed to address the limitations of the original MapReduce programming model by enabling multiple resource-intensive applications to run concurrently on the same Hadoop cluster. The paper introduces the architecture of YARN and discusses its design goals, which include improved scalability, enhanced resource management, and support for non-MapReduce applications. It provides a novel approach to resource negotiation by decoupling resource management and job scheduling, which enhances cluster utilization and ensures that workloads are managed efficiently. The paper also highlights YARN's ability to scale to larger cluster sizes and its support for a variety of applications beyond MapReduce, such as real-time processing and iterative algorithms. Overall, YARN represents a significant advancement in Hadoop's ability to support diverse big data applications and workloads.

C) L. George, *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. O'Reilly Media, Inc., 2011.

HBase: The Definitive Guide provides an in-depth introduction and comprehensive reference to HBase, an open-source, distributed, and scalable database designed for random access to large amounts of data. This book covers the architecture and internal mechanics of HBase, including its key features such as data model, storage structure, and the way it scales horizontally across clusters. It explains how HBase integrates with the Hadoop ecosystem and how it differs from traditional relational databases, particularly in its ability to handle high-throughput, low-latency operations on huge datasets. The guide also walks through practical examples of HBase installation, configuration, and performance tuning, offering insights into its usage in real-world applications. The book is aimed at developers and administrators working with big data technologies, providing them with the knowledge to harness the power of HBase for efficient data storage and retrieval in large-scale distributed systems.

IMPLEMENTATION

Modules

Service Provider

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 Login, Browse Data Sets and Train & Test, View Trained and Tested Accuracy in Bar Chart, View Trained and Tested Accuracy Results, View All Antifraud Model for Internet Loan Prediction, Find Internet Loan Prediction Type Ratio, View Primary Stage Diabetic Prediction Ratio Results, Download Predicted Data Sets, View All Remote Users.

View and Authorize Users

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

Remote 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 like REGISTER AND LOGIN, PREDICT PRIMARY STAGE DIABETIC STATUS, VIEW YOUR PROFILE.

CONCLUSION

We present CLOUDRAID, a simple yet effective tool for detect- ing distributed concurrency bugs. CLOUDRAID achieves its efficiency and effectiveness by analyzing message orderings that are likely to expose errors from existing logs. Our evaluation shows that CLOUDRAID is simple to deploy and effective in detecting bugs. In particular, CLOUDRAID can test 60 versions of six representative systems in 35 hours, finding successfully 31 bugs, including 9 new bugs that have never been reported before.

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 *Proc. 4th Annu. Symp. Cloud Comput.*, 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 *Proc. 14th USENIX Conf. 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 data-intensive systems," in *Proc. 11th USENIX Conf. 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 *Proc. ACM Symp. Cloud Comput.*, 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 *Proc. 21st Int. Conf. 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 Symp. 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 *Proc. 23rd ACM Symp. 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: Wait-free coordination for internet-scale systems," in *USENIX Annu. Technical Conf.*, 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," *Bull. IEEE Comput. Soc. Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[16] 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 *Proc. ACM SIGOPS 22nd Symp. 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 *Proc. 19th ACM SIGSOFT Symp. 13th European Conf. 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 *Proc. 2013 Int. Symp. Memory Management*, ser. ISMM '13, New York, NY, USA: ACM, 2013, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/2464157.2466483