Hardware Based Method to Diagnose Software Bugs

Hongyu Liu
Computer Science, University of Texas at San Antonio
xhe341@my.utsa.com

Abstract

Software bugs are notorious. Before we release a software, test work should be done to ensure no failures in production runs. However some bugs could be escaped from test. This may compromise end-user benefits. Unfortunately, to find out production-run bugs is challenging. Existing tools cannot meet all requirements, such as privacy, performance overhead, diagnosis latency and so on.

To build a ideal tool, we design it based on two observations[1]. First, many bugs have short propagation which means we can find out root cause right before failure sites. Second, collecting such information is much cheaper than monitoring the whole program execution. Software based methods have lots of shortcomings, while hardware based methods can be efficient and effective. We can use branch-tracing hardware and performance counters to collect useful information.

Many researches shows this method is feasible.
keywords: bug detection, LBR, Hardware

1 Introduction

Although we use sophisticated tools and approaches to detect bugs, many software bugs cannot be found out. But this bugs may incur significant influence. For example, Knight company lost $440 million due to a software glitch which could let computers execute a series of automatic orders in a short time. The company almost went bankrupt[3].

Unfortunately, it is very difficult to diagnose production-run bugs. In production-run situation, we need to preserve user's privacy and reduce performance overhead, which may increase diagnosis latency.

Existing tools could be classified in to three categories. The first one is to monitor the whole program in execution. Although these tools can find root cause of bugs easily, it would add much performance overhead. The more information it collects, the more overhead it adds. It is impossible to use these tools in production runs. The second one is to collect failure sites information. Due to collecting only failure sites information, the overhead is lowest. Users cannot notice these tools exist. Little information means we may not find out the root cause or it cannot capture the root cause information. This would cost long time to analyze and recollect information[8]. The third one is a tradeoff between the above two methods. In order to obtain more information and add low overhead, the tools use sampling to acquire data[4]. The obstacle to use this method is when we do sampling, there is no bugs, while bugs occur at sampling interval. It is so efficient to collect information related to bugs.

To build a ideal tool, we design based on two observations[1]. First, we can capture root cause right before failure sites. Second, recording this information is much cheaper than monitoring the whole program execution.

According to low overhead of hardware, we can use existing hardware to record runtime information. Intel processors provide hardware performance monitoring unit which is used for performance profiling. In provided facilities, we can use *Last Branch Record* (LBR) and *Branch Trace Store* (BTS) to record program branch information. LBR can be configured to record different types of branch instructions, including conditional branches, unconditional jumps, calls, returns, and others. We can use *perf_event* to collect cache-coherence events which may be useful for us to identify root cause of bugs.

2 Related work

Many works have been done on hardware based method to monitor program execution. THeME uses hardware instrumentation for test coverage analysis[7]. This is not for bugs detection. Intel GNU GDB tool uses hardware facilities for debugging. LBRLOG[1] uses LBX to store branches before program failure. This tool just could diagnose sequential bugs. LCRLOG[1]

can find out root cause of concurrent bugs. But this tool is a conceptional tool since it uses hardware extension to record cache-coherence event which means there is no such hardware so far. Like the LCRLOG, ECMon[5] proposes a hardware extension to deal with cache events in the whole program execution. This incurs more overhead than LCRLOG.

Triage[6] uses record and replay mechanism to detect bugs automatically. The tool modified OS kernel. It is impractical to use, since we do n't know whether this may compromise our system. CBI[2] was proposed for failure diagnosis. However the tool uses sampling method to collect information. The method is not efficient and precise. It may re-execute program many times to obtain enough information for diagnosis.

3 Conclusion

Based on the works which have been done, it indicates it is feasible to use hardware for failure diagnosis. We can design a tool which has low performance overhead and low diagnosis latency. This can be deployed in production-run program. Not only this can get enough information for bugs detection, but it can preserve user's privacy. We can combine record and replay mechanism with hardware based method to detect bugs and eliminate bugs automatically.

References

[1] J. Arulraj, G. Jin, and S. Lu. Leveraging the short-term memory of hardware to diagnose production-run software failures. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 207–222, New York, NY, USA, 2014. ACM.

[2] P. Arumuga Nainar and B. Liblit. Adaptive bug isolation. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 255–264, New York, NY, USA, 2010. ACM.

[3] CNN. Knight expensive software glitches.

[4] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 15–26, New York, NY, USA, 2005. ACM.

[5] V. Nagarajan and R. Gupta. Ecmon: Exposing cache events for monitoring. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 349–360, New York, NY, USA, 2009. ACM.

[6] J. Tucek, S. Lu, C. Huang, S. Xanthos, and Y. Zhou. Triage: Diagnosing production run failures at the user's site. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 131–144, New York, NY, USA, 2007. ACM.

[7] K. Walcott-Justice, J. Mars, and M. L. Soffa. Theme: A system for testing by hardware monitoring events. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 12–22, New York, NY, USA, 2012. ACM.

[8] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 293–306, Hollywood, CA, 2012. USENIX.