Groundwork for the Development of Testing Plans for Concurrent Software

Eileen Kraemer and Laura K. Dillon
University of Georgia, Michigan State University
eileen@cs.uga.edu, ldillon@cse.msu.edu

Abstract

While multi-threading has become commonplace in many application domains (e.g., embedded systems, digital signal processing (DSP), networks, IP services, and graphics), multi-threaded code often requires complex co-ordination of threads. As a result, multi-threaded implementations are prone to subtle bugs that are difficult and time-consuming to locate. Moreover, current testing techniques that address multi-threading are generally costly while their effectiveness is unknown. The development of cost-effective testing plans requires an in-depth study of the nature, frequency, and cost of concurrency errors in the context of real-world applications. The full paper will lay the groundwork for such a study, with the purpose of informing the creation of a parametric cost model for testing multi-threaded software. The current version of the paper provides motivation for the study, an outline of the full paper, and a bibliography of related papers.

1. Introduction

The use of multi-threading has become commonplace in domains, such as embedded systems, digital signal processing (DSP), networks, IP services, and graphics, where software must respond asynchronously and in a timely manner to events produced by autonomous agents in the execution environment. More recently, its use is also being fueled by the increasing prevalence of multi-core processors, integrated circuits to which two or more individual processors (cores) have been attached. The use of multi-threading is thus motivated largely by the promise of performance benefits.

However, performance benefits of multi-threading rely on the degree to which software algorithms can be designed and implemented to execute in parallel, and unfortunately such multithreaded code often requires complex co-ordination of threads. As a result, multi-threaded software is prone to subtle bugs that are difficult and time-consuming to locate. The unpredictable and asynchronous nature of the execution environment necessitates reasoning about either partially ordered or fully interleaved execution models. Many developers find the former models difficult to reason about, and the latter models produce a combinatorial explosion of potential orderings. In any case, timing-dependent errors may appear intermittently. Most debugging techniques rely on reproducing test executions that reveal errors, but such executions are not easily reproduced.

The development of cost-effective testing plans requires an in-depth study of the nature, frequency, and cost of concurrency errors in the context of real-world applications. In this paper we 1) survey the existing literature on concurrency bugs; 2) survey prior work on approaches to testing; 3) provide background on related work and the state-of-the-art in bug reporting and tracking; and 4) propose a structured reporting and classification mechanism for concurrency bugs, the execution environment in which they are found, their resolution, and the associated approach and cost of such resolution. This structured mechanism will promote improved diagnosis, simplify large-scale collection and analysis of real-world concurrency bugs, and, applied to a sufficiently large and representative collection of concurrency bugs, inform the creation of a parameterized cost model for concurrency testing.

2. Prior studies of concurrency bugs

Survey prior studies of concurrency bugs, describing approach of each and results.

Comment on effects of methodology and data set on differences in results.

Discuss themes that emerge and effect on direction of continuing collection and analyses of data on concurrency bugs.

Dagstuhl Seminar Proceedings 10111
Practical Software Testing : Tool Automation and Human Factors
http://drops.dagstuhl.de/opus/volltexte/2010/2621

3. Related work in testing and debugging of concurrent software

Survey major approaches to testing (and debugging) of concurrent software.

Discuss costs and benefits of each, and how they apply to current computing environments.

4. Prior work in bug reporting and tracking

Survey major approaches to bug reporting and tracking.

Discuss costs and benefits of each, and evaluate their suitability for reporting and tracking of concurrency-related bugs.

5. Benefits of structured reporting

We believe that the development and use of appropriate structured representations of concurrency bug data should both enable improved diagnosis of individual bugs and permit systematic analysis of collections of reports.

Expand on and provide support for this idea.

Provide examples of other domains in which such structured reporting has facilitated analysis and resulted in improved process e.g., MIAME data in genomics domain, other examples.

6. Proposed reporting structure

What data should be collected? What is necessary to diagnose individual bugs? What additional info do we need for larger analysis? How was the bug fixed? How long did it take to locate? How many interleavings/swapped memory accesses/ etc. would it have taken/did it take to track this down and then to fix it? … etc.

7. Cost model and testing plan

Present proposed approach to specification and parameterization of cost model

Describe elements of cost model, preliminary data set, fitting procedure, etc.

Describe how cost model would be used to help formulate testing plans.

8. Conclusion

Recommend use of structured reporting for concurrency bugs. Describe plan to apply methodology to large data set and discuss expected benefit of doing so.

9. References

An initial draft of potential references.

[1] A.-R. Adl-Tabatabai, C. Kozyrakis, and B. Saha. Transactional programming in a multi-core environment. In *PPOPP*, 2007.

[2] C. S. Ananian, K. Asanovic, B. C. Kuszmaul, C. E. Leiserson, and S. Lie. Unbounded transactional memory. In *HPCA*, 2005.

[3] C. Boyapati, R. Lee, and M. Rinard. Ownership types for safe programming: Preventing data races and deadlocks. In *OOPSLA*, 2002.

[4] A. Bron, E. Farchi, Y. Magid, Y. Nir, and S. Ur. Applications of synchronization coverage. In *PPoPP*, 2005.

[5] B. D. Carlstrom, A. McDonald, H. Cha_, J. Chung, C. C. Minh, C. Kozyrakis, and K. Olukotun. The atomos transactional programming language. In *PLDI '06*, 2006.

[6] S. Chandra and P. M. Chen. Whither generic recovery from application faults? A fault study using open-source software. In *DSN*, 2000.

[7] J.-D. Choi et al. Efficient and precise datarace detection for multithreaded object-oriented programs. In *PLDI*, 2002.

[8] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. R. Engler. An empirical study of operating system errors. In *SOSP*, 2001.

[9] J. Corbet (May 14 2008). "Distributed bug tracking". LWN.net. http://lwn.net/Articles/281849/. Retrieved 2010-05-10.

[10] O. Edelstein, E. Farchi, Y. Nir, G. Ratsaby, and S. Ur. Multi-threaded java program test generation. *IBM Systems Journal*, 2002.

[11] D. Engler and K. Ashcraft. RacerX: Effective, static detection of race conditions and deadlocks. In *SOSP*, 2003.

[12] E. Farchi, Y. Nir, and S. Ur. Concurrent bug patterns and how to test them. In *IPDPS*, 2003.

[13] M Fischer, M Pinzger, H Gall. Populating a release history database from version control and bug tracking systems . *In Proc. International Conference on Software Maintenance ICSM 2003).* Amsterdam, Netherlands, September 2003. IEEE.

[14] C. Flanagan and S. N. Freund. Atomizer: a dynamic atomicity checker for multithreaded programs. In *POPL*, 2004.

[15] P. Godefroid. Model checking for programming languages using verisoft. In *POPL*, 1997.

[16] Godefroid, P. and Nagappan, N. "Concurrency at Microsoft - An Exploratory Survey Proceedings of (EC)^2 (CAV 2008 Workshop on "Exploiting Concurrency Efficiently and Correctly"), Princeton, July 2008.

[17] W. Gu, Z. Kalbarczyk, R. K. Iyer, and Z. Yang. Characterization of linux kernel behavior under errors. In *DSN*, 2003.

[18] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis,
B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional memory coherence and consistency. In *ISCA*, 2004.

[19] T. Harris and K. Fraser. Language support for lightweight transactions. In *OOPSLA*, 2003.

[20] T. Harris, S. Marlow, S. Peyton-Jones, and M. Herlihy. Composable memory transactions. In *PPoPP '05*, 2005.

[21] R. Hastings and B. Joyce. Purify: Fast detection of memory leaks and access errors. In *Usenix*, 1992.

[22] J. Hess (6 April 2007). "Integrated issue tracking with Ikiwiki". LinuxWorld.com. IDG. http://www.linuxworld.com/news/2007/040607-integrated-issue-tracking-ikiwiki.html. Retrieved 2010-05-10.

[23] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: A tool for finding copy-paste and related bugs in operating system code. In *OSDI*, 2004.

[24] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai. Have things changed now?: an empirical study of bug characteristics in modern open source software. In *Proceedings of the 1st workshop on Architectural and system support for improving software dependability (ASID'06)*, 2006.

[25] S. Lu,W. Jiang, and Y. Zhou. A study of interleaving coverage criteria. In *FSE*, 2007.

[26] S. Lu, S. Park, C. Hu, X. Ma,W. Jiang, Z. Li, R. A. Popa, and Y. Zhou. Muvi: Automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *SOSP07*, 2007.

[27] Lu, S., Park, S., Seo, E., and Zhou, Y. 2008. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. SIGARCH Comput. Archit. News 36, 1 (Mar. 2008), 329-339. DOI= http://doi.acm.org/10.1145/1353534.1346323

[28] S. Lu, J. Tucek, F. Qin, and Y. Zhou. Avio: Detecting atomicity violations via access interleaving invariants. In *ASPLOS*, 2006.

[29] B. McCloskey, F. Zhou, D. Gay, and E. Brewer. Autolocker:
synchronization inference for atomic sections. In *POPL*, 2006.

[30] M. Moir. Transparent support for wait-free transactions. In *11th International Workshop on Distributed Algorithms*, 1997.

[31] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. Logtm: Log-based transactional memory. In *HPCA*, 2006.

[32] J. E. B. Moss and A. L. Hosking. Nested transactional memory: model and architecture sketches. *Sci. Comput. Program.*, 2006.

[33] Multiple(wiki) "Bug report". Docforge. http://docforge.com/wiki/Bug_report. Retrieved 2010-05-10.

[34] M. Musuvathi and S. Qadeer. Iterative context bounding for systematictesting of multithreaded programs. In *PLDI*, 2007.

[35] G. C. Necula, S. McPeak, and W. Weimer. CCured: Type-safe retrofitting of legacy code. In *POPL*, 2002.

[36] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *ENTCS*, 2003.

[37] R. H. B. Netzer and B. P. Miller. Improving the accuracy of data race detection. In *PPoPP*, 1991.

[38] T. Ostrand, E. Weyuker, and R. Bell. Predicting the location and number of faults in large software systems. *TSE*, 2005.

[39] M. Prvulovic and J. Torrellas. ReEnact: Using thread-level speculation mechanisms to debug data races in multithreaded codes. In *ISCA*, 2003.

[40] S. Qadeer and D. Wu. Kiss: keep it simple and sequential. In *PLDI*, 2004.

[41] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou. Rx: Treating bugs as allergies - a safe method to survive software failures. In *SOSP*, 2005.

[42] H. E. Ramadan, C. J. Rossbach, D. E. Porter, O. S. Hofmann, A. Bhandari, and E. Witchel. Metatm/txlinux: transactional memory for an operating system. In *ISCA*, 2007.

[43] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM TOCS*, 1997

[44] M. Sullivan and R. Chillarege. A comparison of software defects in database management systems and operating systems. In *FTCS*, 1992.

[45] R. N. Taylor, D. L. Levine, and C. D. Kelly. Structural testing of concurrent programs. *IEEE Trans. Softw. Eng.*, 1992.

[46] M. Vaziri, F. Tip, and J. Dolby. Associating synchronization constraints with data in an object-oriented language. In *POPL*, 2006.

[47] M. Xu, R. Bodik, and M. D. Hill. A serializability violation detector for shared-memory server programs. In *PLDI*, 2005.

[48] Y. Yu, T. Rodeheffer, and W. Chen. Racetrack: Ef_cient detection of data race conditions via adaptive tracking. In *SOSP*, 2005.

[49] Z. Li et al. Have things changed now? An empirical study of bug characteristics in modern open source software. In *ASID workshop in ASPLOS*, 2006.