Improving Developers' Confidence in Test Results of Multi-threaded Systems

Avoiding Early and Late Assertions

Ayla Dantas
Universidade Federal de Campina Grande - UFCG
ayla@dsc.ufcg.edu.br

Abstract

Testing multi-threaded systems is challenging, especially due to their inherent non-determinism. As a result, considering the same test, we can have success and failure results for different test executions. This may happen due to bugs that are only revealed on certain thread interleavings, and which are difficult to reproduce. However, this may also be caused by a problem with the test. When the latter case happens, developers may waste a lot of time searching a bug that does not exist, or they may lose trust in the test, maybe not investigating real bugs. With this work, we want to support tests development through an approach to avoid test failures caused by a common problem with multi-threaded systems testing: assertions performed too early or too late. The approach's basic idea is to use thread monitoring and other techniques to avoid such problems during the development of tests and hence increase the developers' confidence in test results.

Categories and Subject Descriptors D.2.5 [*Software Engineering*]: Testing and Debugging

General Terms Verification

Keywords Multi-threaded systems testing, software monitoring, aspect-oriented programming.

1. Motivation

Multi-threaded systems are becoming more and more popular, especially with the advent of multicore processors. However, this has increased the complexity of software development demanding, as a result, better programming tools to systematically find defects, help debug programs, find performance bottlenecks, and aid in testing (Sutter and Larus 2005). In this context, testing is especially challenging due to the inherent nondeterminism of multi-threaded systems (MacDonald et al. 2005). Tests that pass in some test runs and fail in others may be the result of intermittent bugs. As these bugs are difficult to find, when the test is the problem, developers may waste too much time trying to find bugs that do not exist. Another negative effect is that developers may no longer trust in test results. Once a test failure happens, they do not believe it was caused by a bug.

Testing activities consist of the following steps (Harrold 2000): designing test cases, executing the software with these test cases, and examining the results produced by these executions. While testing some distributed systems in our lab (LSD), we have noticed that many of their test failures were due to the following test problem: the results were being examined at inappropriate times. For instance, this examination process (test assertions) could be performed too early (when the system is still carrying out the desired operation) or too late (when the state being verified is no longer valid).

This happens because some tests exercise asynchronous operations. This way, it is not always simple to determine when all threads have finished processing and thus when it is safe to perform the test assertions (Goetz and Peierls 2006). Some common wait approaches being used are *explicit delays* (e.g. `Thread.sleep` calls) or busy waits (e.g. `Thread.sleep` calls inside loops where a condition is checked from time to time). Although being commonly used, they may lead to the following problems: long test runs (Meszaros 2007) or test failures due to insufficient delays or because of the miss of opportunity to fire effect (Lea 2000) (the assertion was performed too late). Another cause of test failures is that periodic threads may wake up and change the system state while assertions are being performed, thus producing non-deterministic results.

Trying to increase developers' confidence in test results, we intend to answer the following research question:

- How can we avoid failures in multi-threaded systems tests which are caused by assertions performed at inappropriate times?

OOPSLA'08, October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

2. Approach and its Evaluation

In order to answer this question, we propose an approach to determine the earliest time assertions can be safely performed. This approach is based on the monitoring and control of the system threads and on simple primitives made available to test developers. The general service provided by these primitives is the ability to make the test wait until an appropriate time where assertions should be performed (e.g. all threads are waiting) and to avoid main system state changes when this is achieved, which could otherwise lead to undesired test failures (because the system state changed while assertions were done). Figure 1 illustrates the approach. It shows the monitoring of the system under test, or SUT (considering execution points reached by threads) in order to provide the primitives to be used in test cases, such as the `waitUntil(sysState)` operation. For instance, whenever there is a call to the `Object.wait` method, a state change notification may be generated.

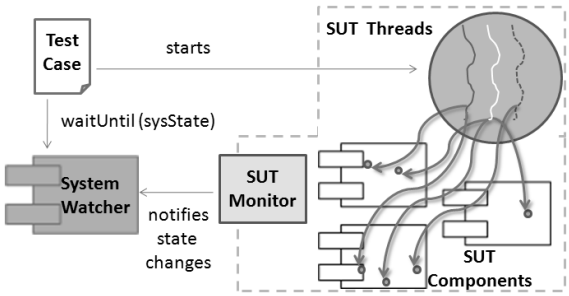

Figure 1. Approach Overview

In our previous work (Dantas et al. 2008), a testing framework to support our approach and based on Aspect-Oriented Programming (AOP) (Kiczales et al. 1997) has been developed and evaluated. Evaluation used synthetic tests that presented failures that were not due to application bugs. This evaluation compared this framework with the other approaches commonly used: explicit delays and busy waits. The factors considered were: i) percentage of test failures; and ii) test duration time. This evaluation results showed that using our test framework we could avoid all the failures that could happen with the other approaches. We could also observe that the amount of time taken for test executions of the versions using busy waits and explicit delays could be a lot greater than with our approach depending on the delay times chosen by developers. Later, we started to perform further experiments with the same evaluation purposes, but using the real tests of a distributed system under development in our lab (OurBackup (Oliveira 2007)).

The current version of the framework just supports tests that involve threads within a single process. A solution for distributed components and its evaluation are part of our future work. Besides that, we also plan to investigate other techniques, besides (or in addition to) AOP for threads monitoring in the tool to support tests development. AOP was used in order to better modularize threads monitoring and control, a challenging concern to be implemented. An alternative technique is to use a proposed design pattern which could be compared and maybe combined with the approach proposed by Meszaros (Meszaros 2007) of making tests synchronous. Besides, as threads monitoring may affect threads scheduling, possibly obscuring bugs that could otherwise be revealed, we also plan to evaluate this effect and the combination of our approach with tools based on noise making (Stoller 2002; Copty and Ur 2005) in order to avoid that.

3. Final Remarks

We plan to conclude the activities planned above by March 2010. By this time, we expect to have more results to show that our approach can be used to avoid test failures due to assertions performed too early or too late. We believe that several projects would be able to benefit from our approach, especially the ones that use Agile Methodologies, in which automatic tests play an important role and in which it is very important to have trustworthy test results.

References

Shady Copty and Shmuel Ur. Multi-threaded Testing with AOP is Easy, and It Finds Bugs. *Proc. 11th International Euro-Par Conference, LNCS*, 3648:740–749, 2005.

Ayla Dantas, Francisco Brasileiro, and Walfredo Cirne. Improving Automated Testing of Multi-threaded Software. *ICST*, 0:521–524, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/ICST.2008.38.

B. Goetz and T. Peierls. *Java concurrency in practice*. Addison-Wesley, 2006.

M.J. Harrold. Testing: a roadmap. *Proceedings of the Conference on The Future of Software Engineering*, pages 61–72, 2000.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect–Oriented Programming. In *European Conference on Object–Oriented Programming, ECOOP'97*, LNCS 1241, pages 220–242, Finland, June 1997. Springer–Verlag.

Doug Lea. *Concurrent Programming in Java (tm): Design Principles and Patterns*. Addison-Wesley, 2000.

S. MacDonald, J. Chen, and D. Novillo. Choosing Among Alternative Futures. *Proc. Haifa Verification Conference, LNCS*, 3875: 247–264, 2005.

G. Meszaros. *XUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007.

Marcelo Iury S. Oliveira. OurBackup: Uma Solução P2P de Backup Baseada em Redes Sociais. Master's thesis, Universidade Federal de Campina Grande, 2007.

Scott D. Stoller. Testing concurrent Java programs using randomized scheduling. In *Proc. Second Workshop on Runtime Verification (RV)*, volume 70(4) of *Electronic Notes in Theoretical Computer Science*. Elsevier, July 2002.

H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, 2005.