**Learning from Mistakes — A Comprehensive Study on Real World Concurrency Bug Characteristics**

对于现实世界的并发bug特征的一个全面的研究

一些有趣的发现：

（1）1/3被检测的非死锁并发错误，是由于程序员安排的操作时序被破坏，而这一点是早期的同步技术无法监测到的

（2）34%由多变量引起，无法检测

（3）92%的检测到bug可以使用不超过4个memory操作就能够重现

（4）73%的错误难以被程序员准确的定位出原因

传统的方法存在着不少的问题：

**Concurrency bug detection ：虽然以前已经有很多人针对data race和deadlock做出了很多方案，但是这些方法都不够的全面，例如无法找到更多的bug，对bug的修复没有什么帮助等等。**

**Concurrent program testing and model checking ：很多的bug需要特定的输入和特定的执行序列才能触发，很难设计Test input**

**Concurrent programming language design ：到底能找到哪一类的bug，能不能更加有效的帮助程序员理解bug的出现原因**

本文研究主要集中在bug的这些特征：bug pattern，manifestation（表现形式）fix strategy（修复策略），建立目录如下：

算机生成了可选文字:
Dimension 
Bug 
Pattern* 
Dimension 
Bug 
festation 
Dimension 
Non- 
deadlock 
Fix Strategy 
Deadlock 
Fix Strategy 
Concerns in 
Transactional 
Memory 
Category 
Atomicity 
Violation 
Order 
Violation 
Other 
Term 
Manifestation 
Condition 
# of threads involved 
# of variable involved 
# of accesses involved 
Category 
Condition Check 
Code Switch 
Design Change 
Lock Strategy 
Other 
Give up resource 
Split Resource 
Change acquisition order 
Other 
Very long code 
Rollback Problem 
Code Nature 
Definitions Related to Bug Pattern Study 
Description 
The desired serializability among multiple memory accesses is violated. 
(i.e. a code region is intended to be atomic, but the atomicity is not enforced during execution.) 
The desired order between two (groups of) memory accesses is flipped. 
(i.e. A should always be executed before B, but the order is not enforced during execution.) 
Concurrency bugs other than the atomicity violation and order violation. 
Definitions Related to Bug Manifestation Study 
Definition 
A specific execution order among a smallest set (S) of memory accesses. 
As long as that order is enforced, no matter how, the bug is guaranteed to manifest. 
The number of distinct threads that are included in 
The number of distinct variables that are included in S. 
The number of accesses that are included in 
Definitions Related to Bug Fix Study 
Description 
(1) While-flag; or (2) optimistic concurrency with consistency check. 
Switch the order of certain statements in the source code. 
Change the design of data structures or algorithms. 
(1) Add/change locks; or (2) adjust the region of critical sections. 
Strategies other than the above ones. 
Not acquiring a resource (lock, etc.) for certain code region. 
Split a big resource to smaller pieces to avoid competition. 
Switch the acquisition order among several resources. 
Strategies other than the above ones. 
A code region is too long to be put into a transaction. 
Some I/O and system calls are hard to roll back. 
Source code with certain design is hard to turn to transaction. 
Abbr. 
Atomicity 
Order 
Other 
Abbr. 
COND 
Switch 
Design 
Lock 
Other 
GiveUp 
Split 
AcqOrder 
Other 
Long 
Rollback 
Nature 

寻找bug，用比较常用的程序进行寻找，Mysql等，然后从bug database里面挑出典型的bug（随机），再由人工确认，但是仅限于Server & Client base（MySQL, Apache, Mozilla, and OpenOffice

）的应用程序，其他程序的并没有被纳入到实验当中给（例如操作系统）

发现：

bug pattern：

大部分的bug 都是由原子性和顺序被破坏这两个bug pattern导致的，因而detector需要将主要精力放在这两个pattern身上

三分之一的bug都是全新的order bug，之前的detector没有检测到，因而需要增添新的检测方法

bug manifestation：

尽管现实应用中包括了许许多多进程，但是实际引发bug的时候，绝大部分都是包括不多于两个线程

三分之一的死锁bug是由单个线程引起的，鉴于比较简单，建议detector尝试尽早将这些bug消除

66%的非死锁bug是由单个变量引起，其余是由多变量引起的，需要应用新的技术来检测

97%的死锁bug由不超过两个资源引起

大部分bug，无论死锁还是非死锁问题，在不超过四个access的情况下（非死锁对应的是memory access，死锁对应的是resource access），只要将顺序固定，就能够很准确的被表现出来。。。。。。不太懂。。。。。。原文：

**Finding (8.1):** 90% (67 out of 74) of the examined non-deadlock bugs can deterministically manifest, if certain orders among at most four memory accesses are enforced. **Finding (8.2):** 97% (30 out of 31) of the examined deadlock bugs can deterministically manifest, if certain orders among at most four resource acquisition/release operations are enforced.

**Implications:** Concurrent program testing can focus on the par- tial order among every small groups of accesses. This simplifies the interleaving testing space from exponential to polynomial re- garding to the total number of accesses, with little loss of bug exposing capability.

Bug fix：

只有三分之一不到的技术在fix bug的时候选择采用加锁或者改变锁

三分之二的死锁fix strategy都是让线程放弃对资源的acquire，但是这个会引发其他的非死锁bug，所以需要注意用这个strategy fix好的死锁bug是否真的没问题了，是不是还会引起其他的非死锁错误

Mistakes during bug fixing：

Transaction memory 能够帮助避免40%的并发错误

20%的时序性错误无法用TM来避免，因为TM只能用来保证原子性，因而需要程序语言来进行时序性支持

其他发现：

很多bug会导致很严重的问题，例如程序崩溃等，影响程序的可靠性

很多bug很难被重现

test case的设计对于bug的诊断很关键

程序员在开发的时候，缺乏诊断器的支持（如果有bug diagnosis的支持，将在开发时就已经避免很多错误）