Handling Production Run Concurrency-Bug Failures

Shan Lu

THE UNIVERSITY OF CHICAGO

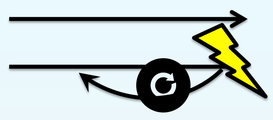
The Problem

- ❖Concurrency bugs
  - ➢ Synchronization problems in multi-threaded programs
  - ➢ Widely exist in production runs
    - ❑ Multi-threaded programs are pervasive
    - ❑ In-house testing is ineffective
  - ➢ Production-run failures are costly

The Solution

- ❖Solution 1: a reactive approach
  - ➢ Replay after a failure
- ❖Solution 2: a proactive approach
  - ➢ Perturb before a failure
- ❖Challenges
  - ➢ Functionality: how to make the failure disappear?
  - ➢ Performance: how to keep the overhead low?

Our Reactive Tool: ConAir [1,3]



Ideas

- ❖Rollback & replay 1 thread at failure
  - ➢ Delaying the too-fast thread

→Help recover all major types of concurrency buys

- ❖Rollback & replay idempotent regions
  - ➢ Requiring no checkpoints
  - → Negligible run-time overhead

Examples

```
Thread 1                          Thread 2
if (proc){          //A
  tmp=*proc;        //C      proc = NULL;        //B
}
MySQL
```

replay Thd 1 when failure occurs @ C

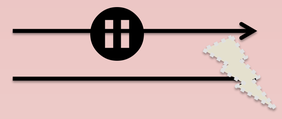
```
Thread 1                          Thread 2
   /*mThd is unintialized*/
_state=mThd→s;//A
                                  mThd = CreateThd(); //B
MySQL
```

replay Thd 1 when failure occurs @ A

Experimental Results

- ❖Performance
  - ➢ < 0.2% run-time overhead
- ❖Functionality
  - ➢ Works for 16 out of 26 real-world concurrency bugs
  - ➢ Caveats: assuming output correctness specifications

Our Proactive Tool: AI [2,3,4]



Ideas

- ❖Temporarily stall 1 thread at selected moments
  - ➢ Delaying the too-fast thread

→Help recover all major types of concurrency buys

- ❖Using *AI invariants* to identify "selected moments"
  - ➢ Concurrency bugs happen when a shared-variable access *i* follows an abnormal remote predecessor
  - ➢ Stalling before *i* so that the invariant is not violated

Examples

```
Thread 1                          Thread 2
if (proc){          //A
  tmp=*proc;        //C      proc = NULL;        //B
}
MySQL
```

stall Thd 2 when B is abnormally preceded by A

```
Thread 1                          Thread 2
   /*mThd is unintialized*/
_state=mThd→s;//A
                                  mThd = CreateThd(); //B
MySQL
```

stall Thd 1 when A is abnormally preceded by ϕ

Experimental Results

- ❖Performance
  - ➢ < 1% run-time overhead for desktop/server programs
  - ➢ >10X slowdown for scientific parallel programs
- ❖Functionality
  - ➢ Works for 35 out of 35 real-world concurrency bugs
  - ➢ Caveats: requires training

Summary: ConAir VS. AI

	ConAir	AI
Performance	Great	Poor when there are intensive shared-memory accesses
Functionality	Poor when failure thread is too slow Poor when error propagation is long	Not clear for more complicated concurrency bugs

Future: ConAir + AI?

References:

[1]. W. Zhang, M. Kruijf, A. Li, S. Lu, and K. Sankaralingam. ConAir: Featherweight Concurrency Bug Recovery Via Single-Threaded Idempotent Execution. In ASPLOS, 2013.

[2]. M. Zhang, Y. Wu, S. Lu, S. Qi, J. Ren, and W. Zheng. AI: A Lightweight System for Tolerating Concurrency Bugs. In FSE, 2014 (***ACM SIGSOFT Distinguished Paper Award***).

[3]. D. Deng, G. Jin, M. Kruijf, A. Li, et. al. Fixing, Preventing, and Recovering from Concurrency Bugs. In Science China Information Sciences, May 2015.

[4]. M. Zhang, Y. Wu, S. Lu, S. Qi, J. Ren, and W. Zheng. AI: A Lightweight System for Detecting and Tolerating Concurrency Bugs. Invited submission to Transaction of Soft. Eng.

For NSF XPS Workshop 2015