

Recovering From Concurrency-bug Failures

1

The challenges for failure recovery

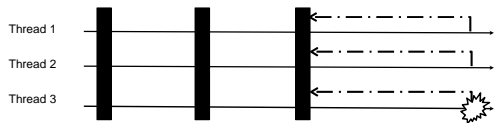
- What is a correct program state?
- How to go back to that state?
- How to by-pass the failure during re-execution?



2

Traditional rollback-recovery

Concurrency bugs are easier to recover from!



Triage: Diagnosing Production Run Failures at the User's Site, SOSPO7

Examples

```

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

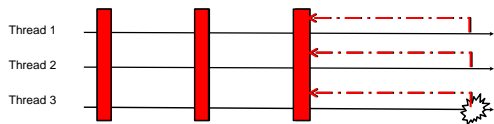
```

Thread 1 (child)  Thread 2 (parent)
                 =CreateThd();
_state = mThd->state;
                 mThd=
Mozilla
    
```

4

Are we done?

- What about the performance?



Modify OS/HW and Slow

5

How can we do better?

Do we have to roll back all threads?

Do we have to make whole-memory checkpoint?

ConAir: Featherweight Concurrency Bug Recovery Via Single-Threaded Idempotent Execution, ASPLOS'13

Bug example revisit

Do we need to roll back all threads?

```

Thd1          Thd2
if(ptr){
    tmp = *ptr;
}
ptr = NULL;
    
```

7

Bug example revisit

```

Thd1          Thd2          Timeline
if(ptr){
    tmp = *ptr;
}
if(ptr){
    tmp = *ptr;
}
ptr = NULL;
    
```

Rollback one thread is enough

8

Bug example revisit

Do we need periodic checkpoint?

```

Thd1          Thd2
if(ptr){
    tmp = *ptr;
}
ptr = NULL;
    
```

Failure site is (somewhat) predictable, guided checkpoint could work

9

Bug example revisit

Do we need memory checkpoint?

```

if(ptr)
    Idempotent {
        PC1: mov 0x80496f4, %eax
        PC2: test %eax,%eax
        PC3: je PCn
    }
    
```

An **idempotent** region is a code region that can be **reexecuted** for any number of times **without changing** the program **semantics**

Re-execution region is small, no need to checkpoint

10

Generalize the bug example

Single-threaded error propagation → roll back the failure thread is mostly enough
 Simple failure/error patterns → guided checkpoint-recovery is mostly enough
 Short error propagation → re-executing an idempotent region is often enough

11

Is single-threaded re-exec enough?

- Atomicity violation bugs

```

Thd1          Thd2
R1 if(ptr){
    tmp = *ptr;
}
ptr = NULL;
R1 if(ptr){
    tmp = *ptr;
}
    
```

12

Is single-threaded re-exec enough?

- Atomicity violation bugs (~100%)
- Order violation bugs

```

Thd1                                Thd2
_state = mThd->state;                mThd = CreateThd();
    
```

13

Is single-threaded re-exec enough?

- Atomicity violation bugs (~100%)
- Order violation bugs

```

Thd1                                Thd2
_state = mThd->state;                mThd = CreateThd();
_state = mThd->state;                mThd = CreateThd();
_state = mThd->state;                mThd = CreateThd();
    
```

Failure thread executes too fast

Failure thread executes too slow

14

Is single-threaded re-exec enough?

- Atomicity violation bugs (~100%)
- Order violation bugs (50%)

```

Thd1                                Thd2
_state = mThd->state;                mThd = CreateThd();
_state = mThd->state;                mThd = CreateThd();
_state = mThd->state;                mThd = CreateThd();
    
```

Failure thread executes too fast

Failure thread executes too slow

15

Is single-threaded re-exec enough?

- Atomicity violation bugs (~100%)
- Order violation bugs (50%)
- Deadlocks (?)

16

Is single-threaded re-exec enough?

- Atomicity violation bugs (~100%)
- Order violation bugs (50%)
- Deadlocks (100%)

17

A simplified con. bug failure recovery

Step1: Locate potential failures Recover many failures

Step2: Identify the re-exec region Negligible cost

Step3: Generate rollback re-exec code No change to semantics

18

Step 1 Identify potential failure sites

Failure Type	Potential Failure Site
Assertion Failures	Call to <code>__assert_fail</code> etc
Error Messages	Call to <code>fprintf(stderr,...)</code> , <code>NS_WARNING</code> in Mozilla, <code>tr_err</code> in Transmission, etc.
Incorrect outputs	Call to <code>(f)printf</code> , <code>BinLog::Write</code> in MySQL, etc.
Illegal mem. accesses	Dereferencing
Deadlocks	Call to <code>pthread_mutex_lock(...)</code>





Number of potential failure sites in MySQL: ~13000

Step 2 identify re-execution region





- Re-execution region = idempotent region

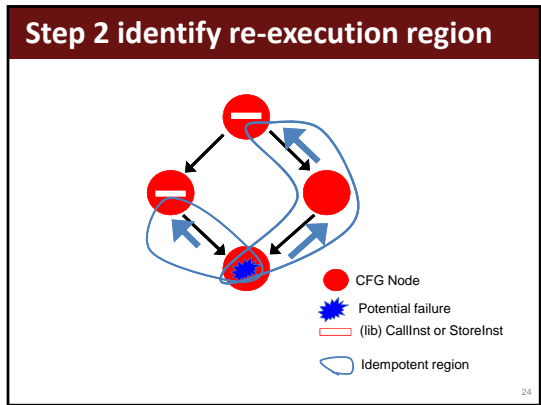
What code region is idempotent?

What makes code non-idempotent ?

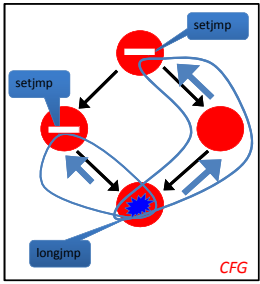
- I/O operation 
- Shared memory write 
- Local memory writes
 - `X = X + 1;` 
 - `Z = X + Y;`
 - `Y = X + 1;` 
 - `Z = X + Y;`
- Writes to registers

What makes code non-idempotent ?

- I/O operation  `CallInst(to lib)`
- Shared memory write  `StoreInst to non-Alloca`
- Local memory writes
 - `X = X + 1;`  `StoreInst to Alloca`
 - `Z = X + Y;`
 - `Y = X + 1;`  `Write to virtual register`
 - `Z = X + Y;` `+ setjmp/longjmp`
- Writes to registers



Step 3 generate code



```

setjmp(...);
//start of idempotent region
...
if(e){
}else{
    while(retryCnt++ < MAX){
        longjmp(...);
    }
    //potential failure site
    _assert_fail();
}
    
```

CFG

Code

Maximize legit idempotent region

- Handle some library functions (e.g. malloc, lock)
 - During execution: timestamp
 - Upon failure: undo most recent library functions
- Inter-procedural analysis
 - Configurable max level of function calls (e.g. 3)
- Optimization
 - Some recovery attempts are doomed to fail

Summary for failure recovery

- How to recovery from concurrency bug failures?
 - Rollback-replay
 - Different types of replay ...
- What are the remaining challenges?
 - Coverage vs. Overhead/System-Support
 - Can we prevent failures at run time?


Deadlock Immunity: Enabling Systems to Defend Against Deadlocks. OSDI 2008
 Cooperative Empirical Failure Avoidance for Multithreaded Programs. ASPLOS13

Failure Prevention

“AI: a Lightweight System for Tolerating Concurrency Bugs”
 Mingxing Zhang, Yongwei Wu, Shan Lu, Shanxiang Qi, Jinglei Ren, Weimin Zheng, FSE 2014

What is con. bug failure prevention?

- How to predict a failure?
- How to change the execution and avoid the failure?
 - Pause to change the timing!



Challenge

- How to predict a failure?
 - Not too early
 - Too early will lead to unnecessary performance losses
 - Not too late
 - Too late will make failures inevitable

Example

- Where should we predict the failure and pause?

Thread 1 (MySQL):
`if (proc) {
 tmp = *proc;
}`

Thread 2 (Mozilla):
`proc = NULL;`

Thread 1 (child):
`_state = mThd->state;`

Thread 2 (parent):
`mThd=CreateThd();`

31

How to generalize?

- Stop before every shared-variable write?
- Stop before every shared-variable read?

32

How to generalize?

- Stop before every shared-variable write?
 - When the previous access is abnormal?
- Stop before every shared-variable read?
 - When the previous access is abnormal?

33

How to generalize?

- Stop before every shared-variable write?
 - When the previous access is abnormal?
- Stop before every shared-variable read?
 - When the previous access is abnormal?

34

Our solution – Anticipating Invariant

For an instruction *i*,
 a fixed set of instructions *P* are expected to precede it and touch the same variable from a different thread (for correct execution)

35

Example

Thread 1 (MySQL):
`if (proc) {
 tmp = *proc;
}`

Thread 2 (Mozilla):
`proc = NULL;`

Thread 1 (child):
`_state = mThd->state;`

Thread 2 (parent):
`mThd=CreateThd();`

36

