CFix Automated Concurrency-Bug Fixing

Guoliang Jin, Wei Zhang, Dongdong Deng, Ben Liblit, and Shan Lu.

University of Wisconsin–Madison
Bugs Need to be Fixed

- Buggy software is an unfortunate fact;
  - There are many bugs to be fixed.

- Software quality does not improve until bugs are fixed;
  - Human patches are often buggy.

Automated fixing is desired
Automated Fixing is Difficult

What is the correct behavior?
- Usually requires developers’ knowledge.

How to get the correct behavior?
- Change program states under bug-triggering inputs;
- No change to program states under other inputs.
Concurrency Bugs

- Synchronization mistakes in multi-threaded programs.

- They are severe:
  - Caused real-world problems: Facebook IPO, etc;
  - Increasingly an issue in this multi-core era.

- They are difficult to reason about and fix manually:
  - Concurrency-bug fixing is time consuming [LuASPLOS’08];
  - Concurrency-bug patches are the most error-prone [YinFSE’11].
What is the correct behavior?
- The program state is already correct as long as the buggy interleaving does not occur.

How to get the correct behavior?
- Only need to disable failure-inducing interleavings;
- Can leverage well-defined synchronization operations.
Challenges

 How to handle the huge variety of bugs?

 How to know the root cause of a bug?

 How to ensure correctness?

 How to maintain performance?

 How to achieve patch simplicity?

 How to effectively test the patches?
CFix Overview

Bug Understanding
Fix-Strategy Design
Synchronization Enforcement
Patch Testing & Selection
Patch Merging
Run-time Support

Buggy binary + Input
Bug report
Bug report
Atomicity Problem
Atomicity Problem
Order Problem
Order Problem
Patched binary
Patched binary
Patched binary
Selected binary
Selected binary
Merged binary
Final patched binary
Contributions

- A technique that enforces order relationship: OFix;
- A framework that assembles a set of techniques to automate the whole bug-fixing process: CFix.
CFix: Automated Concurrency-Bug Fixing

Challenge:
- Concurrency bugs are difficult to reason about.

Solution:
- Leverage existing bug detectors.
Bug Understanding

- Use four types of different detectors:
  - **atomicity violation detectors**
    - ParkASPLOS’09, FlanaganPOPL’04, LuASPLOS’06, ChewEuroSys’10
  - **order violation detectors**
    - ZhangASPLOS’10, LuciaMICRO’09, YuISCA’09, GaoASPLOS’11
  - **data race detectors**
    - SenPLDI’08, SavageTOCS’97, YuSOSP’05, EricksonOSDI’10, KasikciASPLOS’10
  - **abnormal data flow detectors**
    - ZhangASPLOS’11, ShiOOPSLA’10

Bug report could be imperfect.
CFix: Automated Concurrency-Bug Fixing

Challenges:
- Huge variety of bugs;
- Inaccurate root cause.

Solution:
- Design a set of fix strategies using a combination of mutual exclusion enforcement; order relationship enforcement.
Fix-Strategy for Atomicity-Violation Detectors

Example 1

Thread 1

```
if (ptr != NULL) {
    ptr->field = 1;
}
```

Thread 2

```
ptr = NULL;
```

```
ptr = NULL;
```
Fix-Strategy for Atomicity-Violation Detectors

Example 2

Thread 1

\texttt{ptr->field = 1;}

\texttt{ptr->field = 1;}

\texttt{ptr = NULL;}

Thread 2

\texttt{ptr = NULL;}

\texttt{ptr->field = 1;}

\texttt{ptr = NULL;}

\texttt{ptr->field = 1;}

\texttt{ptr = NULL;}
Fix-Strategies

AV Detector

OV Detector

Race Detector

DU Detector

A

B

I₁

I₂

Wₜ

Wₔ
**CFix: Automated Concurrency-Bug Fixing**

- **Bug Understanding**
- **Fix-Strategy Design**
- **Synchronization Enforcement**
- **Patch Testing & Selection**
- **Patch Merging**
- **Run-time Support**

**Challenges:**
- Correctness, performance, simplicity.

**Solution:**
- Mutual exclusion enforcement: AFix [JinPLDI’11]
- Order relationship enforcement: OFix.
Order Relationship Enforcement: OFix

A_1 \rightarrow B \\
\vdots \\
A_j \rightarrow B \\
A \rightarrow ? \\
B

allA-B

use

destroy

initialization

read

firstA-B
OFFix all A-B Enforcement: Principles

- **signal** in A-threads:
  - A-thread signals when it will not execute more A;
  - Each A-thread signals only once;
  - Each A-thread signals as soon as possible.

- **wait** before B:
  - B Proceeds when each A-thread has signaled.
OFix allA-B Enforcement: A side

How to identify the last A instance in one thread

- Each thread that execute A:
  - exactly once as soon as it can execute no more A.
OFix all A-B Enforcement: A side
How to identify the last thread that executes A

```c
void main() {
    for (...) {
        thread_create(thr_main);
        ...;
    }
}

void thr_main() {
    for (...) {
        ...; // A
        ...;
    }
}

void ofix_signal() {
    mutex_lock(L);
    --counter_for_signal_threads;
    mutex_unlock(L);
    if (counter_for_signal_threads == 0) {
        cond_broadcast(con);
    }
}
```
Safe to execute only when \( O \) is 0.

Give up if OFix knows that it introduces new deadlock.

Timed wait-operation to mask potential deadlocks.

```c
void ofix_wait() {
    mutex_lock(L);
    if (O != 0)
        cond_timedwait(con, L, t);
    mutex_unlock(L);
}
```
is not needed if

B does not execute in the same run as

```c
void main() {
    if (...) {
        OFixSignal;
        exit(1);
    }
    thread_create(thr_main, ...); // A
    ...; // B
}
```

Significantly improve patch simplicity.
O Fix first A-B

- Basic enforcement

- When A may not execute:
  - Add a safety-net of signal with all A-B algorithm.

```c
void ofix_wait_b() {
    mutex_lock(L);
    if (flag != true) {
        cond_timedwait(con, L, t);
    }
    mutex_unlock(L);
}

void ofix_signal_b() {
    if (flag != true) {
        flag = true;
        mutex_lock(L);
        cond_broadcast(con);
        mutex_unlock(L);
    }
}
```
Cfix: Automated Concurrency-Bug Fixing

- Bug Understanding
- Fix-Strategy Design
- Synchronization Enforcement
- Patch Testing & Selection
- Patch Merging
- Run-time Support

Challenge:
- Effective multi-thread software testing

Solution:
- CFix-patch oriented testing
Patch Testing Principles

- Prune incorrect patches:
  - Patches causing failures due to incorrect root causes, etc.
- Prune slow patches
- Prune complicated patches

- Not exhaustive testing, but patch oriented testing.
Run Once without External Perturbation

- Reject if there is a time-out or failure.
- Patches fixing wrong root cause:
  - Make software to fail deterministically.

Thread 1

```c
ptr->field = 1;
ptr->field = 1;
```

Thread 2

```c
ptr = NULL;
```
Implicit Bad Patch

- A failure in patch_b implies a failure in patch_a:
  - If patch_a is less restrictive than patch_b.

- Helpful to prune patch_a:
  - Traditional testing may not find the failure in patch_a.
One programming mistake usually leads to multiple bugs.

- Heuristics to merge patches for related bugs.
CFix: Automated Concurrency-Bug Fixing

- Bug Understanding
- Fix-Strategy Design
- Synchronization Enforcement
- Patch Testing & Selection
- Patch Merging
- Run-time Support

- To understand whether there is a deadlock underlying time-out.
- Low-overhead, and suitable for production runs.
## Methodology

<table>
<thead>
<tr>
<th>APP.</th>
<th>AV Detector</th>
<th>OV Detector</th>
<th>RA Detector</th>
<th>DU Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBZIP2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTrack</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZSNES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cherokee</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Evaluation

<table>
<thead>
<tr>
<th>APP.</th>
<th>AV Detector</th>
<th>OV Detector</th>
<th>RA Detector</th>
<th>DU Detector</th>
<th># of Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBZIP2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>x264</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>FFT</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>5</td>
</tr>
<tr>
<td>HTTrack</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Mozilla-1</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>transmission</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>2</td>
</tr>
<tr>
<td>ZSNES</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>3</td>
</tr>
<tr>
<td>Apache</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>3</td>
</tr>
<tr>
<td>MySQL-1</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>5</td>
</tr>
<tr>
<td>MySQL-2</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
<td>9</td>
</tr>
<tr>
<td>Mozilla-2</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Cherokee</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Mozilla-3</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
Conclusion

- CFix uses some heuristics, with good results in practice:
  - Fixing root cause without requiring detectors to report it;
  - Small overhead;
  - Good simplicity.

- Concurrency bugs are feasible to be fixed automatically:
  - By removing bad interleavings;
  - Must be careful in the details.
Questions on CFix?

- Bug Understanding
- Fix-Strategy Design
- Synchronization Enforcement
- Patch Testing & Selection
- Patch Merging
- Run-time Support

It’s 2012. Is this the end of my world?