Handling Production Run Concurrency-Bug Failures

Shan Lu

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 bugs
- Rollback & replay idempotent regions
  - Requiring no checkpoints
  - Negligible run-time overhead

Examples

```java
Thread 1
if (proc)
    tmp=proc; //A
Thread 2
//B

replay Thd 1 when failure occurs @ C
```

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 bugs
- 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

```java
Thread 1
if (proc)
    proc = NULL; //A
Thread 2
//B

stall Thd 2 when B is abnormally preceded by A
```

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

<table>
<thead>
<tr>
<th></th>
<th>ConAir</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Great</td>
<td>Poor when there are intensive shared-memory accesses</td>
</tr>
<tr>
<td>Functionality</td>
<td>Poor when failure thread is too slow Poor when error propagation is long</td>
<td>Not clear for more complicated concurrency bugs</td>
</tr>
</tbody>
</table>

Future: ConAir + AI?

References: