Handling Production-Run Concurrency-Bug Failures

Shan Lu
University of Chicago

Karu Sankaralingam
University of Wisconsin - Madison
Reliability is crucial

• Production-run failures are costly

• Reliability affects other aspects of parallel systems

“We don’t use thread-level parallelism because that is difficult to get correct“
Reliability is challenging

• Concurrency bugs
  – Synchronization problems in multi-threaded s/w

• Concurrency bugs widely exist in production
  – In-house testing is ineffective for concurrency bugs

How to handle production-run failures caused by concurrency bugs?
Rollback recovery

Thread 1
Thread 2
Thread 3
Problems of the traditional approach
Bug examples

Do we need to roll back all threads?

Do we need memory checkpoint?

ConAir: Featherweight Concurrency Bug Recovery Via Single-Threaded Idempotent Execution [ASPLOS13]
ConAir system

Thread 1

Thread 2

Thread 3

• Guarantee no change to program semantics
• No change to OS/Hardware
• No prior bug knowledge required

• Negligible overhead (<0.2%)
• Work for 16 out of 26 real-world bugs
The problems

• What if the error propagation distance is long?

• What if the failure thread was already too slow?

MySQL
Proactive prevention

Thread 1

Thread 2

Thread 3
Proactive prevention

Thread 1

Thread 2

Thread 3
Bug examples

```
if (proc){
    tmp=*proc;
}

proc = NULL;

MySQL_state = mThd->state;

mThd = CreateThd();
```

Do we need to perturb multiple threads?

Do we perturb at random place?
A simple and generic scheme

- The best perturbation point is

  Right before a memory access that has an incorrect/abnormal remote predecessor
AI system

Thread 1

\[
\text{if (proc)}\{
  \text{tmp=*proc;}
\}
\]  
\text{MySQL}

Thread 2

\[
\text{proc = NULL;}
\]

Thread 1

\[
\text{state=mThd\rightarrow state;}
\]
\text{MySQL}

Thread 2

\[
\text{mThd = CreateThd();}
\]

- Guarantee no change to program semantics
- No change to OS/Hardware
- No prior bug knowledge required
- Negligible overhead (<0.2%)
- Work for 16 out of 26 real-world bugs

Training required

1% ~ 10X

35 out of 35

AI: A Lightweight System for Tolerating Concurrency Bugs [*FSE14*] ACM SIGSOFT Distinguished Paper Award
### 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&lt;br&gt;Poor when error propagation is long</td>
<td>Great, but require training</td>
</tr>
</tbody>
</table>

- Can we combine ConAir and AI?
Summary & Other efforts

• Reactive approach to production-run failures
  *ConAir: Featherweight Concurrency Bug Recovery Via Single-Threaded Idempotent Execution [ASPLOS13]*

• Proactive approach to production-run failures
  *AI: A Lightweight System for Tolerating Concurrency Bugs [FSE14]*
  *ACM SIGSOFT Distinguished Paper Award*

• How much can developers contribute?
  *What change history tells us about thread synchronization [FSE15]*

• How much can hardware contribute?

Thank XPS!