Lockout: Efficient Testing for Deadlock Bugs

Ali Kheradmand, Baris Kasikci, George Candea
Deadlock

• Set of threads
  • Each holding a lock needed by another thread
  • Waiting for another lock to be released by some other thread
Why Do Deadlocks Matter?

- Common in modern software
- Hard to detect manually
- Occur rarely during execution
Deadlock Detection

• Traditional testing
  – Deadlocks manifest rarely ✗

• Static detection
  – Fast (run offline) ✓
  – Few false negatives ✓
  – Many false positives ✗

• Dynamic detection
  – Slow (high runtime overhead) ✗
  – Many false negatives ✗
  – Few false positives ✓
Best of Two Worlds

• Normal tests can’t discover enough deadlocks
• Deadlock avoidance or fixing tools tools (Dimmunix [OSDI’08]) take a long time
  • Need to find the schedules that lead to a deadlock
• How to increase the probability of encountering a deadlock?

Steer the program towards schedules that are likely to cause a deadlock
Lockout

- Systematic deadlock testing
- Increases deadlock probability
  - By steering the scheduling
- Leverages past program executions

Trigger more deadlocks with the same test suite
Outline

• Lockout architecture
• Deadlock triggering algorithm
• Preliminary results
• Summary and future work
Lockout Architecture

Static Phase

cout << “Hello, World!\n”;

Source Code

Static Analysis

Instrumentation

Instrumented Binary

RaceMob [SOSP’13]
Lockout Architecture

Dynamic Phase

Instrumented Program

Events

Runtime

End of execution

Previous executions info

Schedule perturbation

Subsequent executions

DeadlockFuzzer [PLDI’09]
Outline

• Lockout architecture
• Deadlock triggering algorithm
• Preliminary results
• Summary and future work
Runtime Lock Order Graph (RLG)

Thread 1

lock(a)
...
lock(b)
...
lock(c)
Deadlock Triggering

- Selects a directed cycle in RLG
- Delays threads accordingly
- Improves simple preemption (CHESS [OSDI’08])
  - Preemption before each lock

![Diagram showing a directed cycle with nodes d, c, b, a and actions lock(a) and lock(b).]
Deadlock Triggering

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lock(a)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lock(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unlock(a)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lock(b)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lock(c)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unlock(c)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unlock(b)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lock(c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lock(a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unlock(a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unlock(c)</td>
</tr>
</tbody>
</table>
Deadlock Triggering

Thread 1

- RT()
- lock(a)
- RT()
- lock(b)
- RT()
- lock(c)
- RT()

Thread 2

- Continue
- Before lock (b)
- Continue
- Before lock (c)
- Continue
- Delay

Thread 3

- Continue
- Before lock (a)
- Continue
- Before lock (b)
- Continue
- Delay
- Continue

Before lock (a)

Before lock (b)

Before lock (c)

Deadlock Triggering

Thread

1

Time

Thread

2

Thread

3

Runtime

a

b

c

lock(a)

lock(b)

lock(c)
Race Dependent Deadlocks

- Preempt before memory accesses
  - Ideally only shared memory accesses
- Can be approximated by preempting after locks
- Can be improved using static analysis
- Can be improved using data race detection
Outline

• Lockout architecture
• **Deadlock triggering algorithm**
• Preliminary results
• Summary and future work
## Lockout Effectiveness

<table>
<thead>
<tr>
<th>Program</th>
<th>Fraction of executions resulting in deadlock (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Native</td>
</tr>
<tr>
<td>Microbench</td>
<td>0.00066 %</td>
</tr>
<tr>
<td>SQLite 3.3.0</td>
<td>0.00064 %</td>
</tr>
<tr>
<td>HawkNL 1.6b3</td>
<td>23 %</td>
</tr>
<tr>
<td>Pbzip2 1.1.6</td>
<td>0 %</td>
</tr>
<tr>
<td>Httpd 2.0.65</td>
<td>0 %</td>
</tr>
</tbody>
</table>

Fraction of executions with deadlocks increased up to three orders of magnitude
Outline

• Lockout architecture
• Deadlock triggering algorithm
• Preliminary results
• Summary and future work
Lockout

• Increases deadlock probability
• Leverages past program executions
• Effective
  • Up to 3 orders of magnitude more deadlock prone
• Open source:
  • https://github.com/dslab-epfl/lockout
Future Work

• Increasing effectiveness with low overhead
  • Static analysis (data races, shared variables)
• Lockout + Automatic failure fixing/avoidance
  (Dimmunix [OSDI’08], CFix [OSDI’12], Aviso
  [ASPLOS’13])
  • In production
  • For testing
• Crowdsourcing (Aviso [ASPLOS’13], RaceMob
  [SOSP’13])