AVIO: Detecting Atomicity Violation Bugs via Access Interleaving Invariants

Professor: Yuanyuan Zhou                  Graduate Students: Shan Lu, Joseph Tucek, Feng Qin

www.iti.uiuc.edu

Motivation

- Concurrent programs are prone to bugs
  - Programmers are used to sequential thinking
- Concurrency bugs have notorious characteristics
  - Hard to trigger during in-house testing
  - Hard to repeat during debugging
- Multi-core hardware trend worsens the problem

Atomicty Violation Bugs

- Synchronization bugs are not just races
  - Intended atomic region
- Programmers want atomicity (serializability)
  - Locks (or transactions) are ways to ensure atomicity
  - Bugs occur when atomicity intention is violated

How to capture atomicity intention?

Previous Solutions

- Manual annotation?
  - Huge manual effort
  - Miss unconscious intentions

Special code structure pattern?
- Can not cover all cases

Our Solution

- Use invariant to capture intention
- Access-Interleaving (AI) Invariant
  - An instruction holds AI invariant, if it should be atomic with its preceding access in order for the program to be correct

AVIO bug detection

- Statistically learn AI-Invariant from training (correct runs)
  - Correct runs reflect programmers intention
  - Since concurrency bugs rarely occur, it is easy to conduct training
- Detect invariant violations at run time

Implementation

- Hardware support AVIO (AVIO-H)
  - Simple extension on cache-coherence protocol
  - Negligible overhead
- Pure software AVIO (AVIO-S)
  - Cheaper
  - More accurate
  - Slower

Results

- AVIO detects more atomicity violation bugs
  - AVIO has less false positives

<table>
<thead>
<tr>
<th>#Bugs Detected</th>
<th>AVIO</th>
<th>Lockset</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment results on Apache, MySQL, Mozilla (with 6 tested real-world atomicity violation bugs)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average False Positives</th>
<th>AVIO-H</th>
<th>AVIO-S</th>
<th>Lockset*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server app.</td>
<td>3.25</td>
<td>51.5</td>
<td>694X</td>
</tr>
<tr>
<td>SPLASH2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conclusions

- AI-Invariant is a good way to reflect programmers atomicity intention
- AVIO is effective in detecting atomicity violations
  - Can distinguish benign races
  - Not rely on any annotation
  - Can be easily implemented on hardware with negligible overhead

* Valgrind implementation

Published in the 12th Architectural Support for Programming Languages and Operating Systems (ASPLOS’06)
Selected into IEEE Micro Special Issue on Top Picks from Architecture Conferences, 2006

Atomicity Violation Bugs

- Synchronization bugs are not just races
  - Intended atomic region
- Programmers want atomicity (serializability)
  - Locks (or transactions) are ways to ensure atomicity
  - Bugs occur when atomicity intention is violated

How to capture atomicity intention?

Previous Solutions

- Manual annotation?
  - Huge manual effort
  - Miss unconscious intentions

Special code structure pattern?
- Can not cover all cases

Our Solution

- Use invariant to capture intention
- Access-Interleaving (AI) Invariant
  - An instruction holds AI invariant, if it should be atomic with its preceding access in order for the program to be correct

AVIO bug detection

- Statistically learn AI-Invariant from training (correct runs)
  - Correct runs reflect programmers intention
  - Since concurrency bugs rarely occur, it is easy to conduct training
- Detect invariant violations at run time

Implementation

- Hardware support AVIO (AVIO-H)
  - Simple extension on cache-coherence protocol
  - Negligible overhead
- Pure software AVIO (AVIO-S)
  - Cheaper
  - More accurate
  - Slower

Results

- AVIO detects more atomicity violation bugs
  - AVIO has less false positives

<table>
<thead>
<tr>
<th>#Bugs Detected</th>
<th>AVIO</th>
<th>Lockset</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment results on Apache, MySQL, Mozilla (with 6 tested real-world atomicity violation bugs)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average False Positives</th>
<th>AVIO-H</th>
<th>AVIO-S</th>
<th>Lockset*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server app.</td>
<td>3.25</td>
<td>51.5</td>
<td>694X</td>
</tr>
<tr>
<td>SPLASH2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conclusions

- AI-Invariant is a good way to reflect programmers atomicity intention
- AVIO is effective in detecting atomicity violations
  - Can distinguish benign races
  - Not rely on any annotation
  - Can be easily implemented on hardware with negligible overhead

Published in the 12th Architectural Support for Programming Languages and Operating Systems (ASPLOS’06)
Selected into IEEE Micro Special Issue on Top Picks from Architecture Conferences, 2006