RaceMob: Crowdsourced Data Race Detection
By: Baris Kasikci, Cristian Zamfir, and George Candea
Presentation by: Jeremy Bradford
Data Race – when 2 or more threads in a program access data in an undetermined order and at least one of these accesses is a write
Motivation

• Data races are some of most costly and difficult to find bugs in multithreaded systems

• Exponential number of interleavings means impractical to test them all, so bugs can remain hidden
Solutions?

- Static detectors
  - Problem: many false positives (e.g. RELAY 84%)
  - Cannot accurately infer what is multithreaded
  - Handling of synchronization primitives

- Dynamic detectors
  - False positives are rare
  - Problems: high runtime overhead, false negatives
False Negative – Happens Before

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = 1$</td>
<td></td>
<td>$x = 1$</td>
<td></td>
</tr>
<tr>
<td>Lock(l)</td>
<td></td>
<td>Lock(l)</td>
<td>Lock(l)</td>
</tr>
<tr>
<td>Unlock(l)</td>
<td></td>
<td>Unlock(l)</td>
<td>Unlock(l)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x = 2$</td>
<td>$x = 2$</td>
</tr>
</tbody>
</table>
RaceMob

- 2-phase data race detector
- Uses static checking and dynamic checking
- Crowdsources validation of statically determined potential data races
  - Why crowdsourced?
  - Reduced overhead and real user execution
Phase 1: Static Detection

- Racemob uses RELAY, a lockset-based detector
- Could use any static detector, preferably complete
  - RELAY complete when no pointers, inline assembly
RaceMob

List of Races
- Unknown
- True Race
- Likely False Positive

“Hive”
- Assignment of Tasks
- Updating List

User Site
- Dynamic Context Inference
- On-demand Detection
Phase 2: Dynamic Validation

- Dynamic Context Inference (DCI) – lightweight initial verification (always on)
- On-demand data race detection
- Schedule Steering
Dynamic Context Inference

- Validates the statically determined races
- Checks for 2 conditions:
  - Concrete instance of aliasing
  - Access from different threads
- Negligible runtime overhead (0.01%)
- Small memory footprint (12 bytes per race)
On-demand Race Detection

- Starts tracking happens-before relationships after first potentially racing access is made
  - No Race: happens-before relationship established between first accessing thread and all other threads
  - Race: Access in another thread before happens-before relationship
Minimal Monitoring in RaceMob

Thread 1
- First Access
- Barrier
- ...
- ...

Thread 2
- ...
- Barrier
- ...
- ...

Thread 3
- ...
- Barrier
- Second Access
- ...
- ...
Minimal Monitoring in RaceMob

Thread 1
- First Access
- ...
- Barrier
- ...

Thread 2
- ...
- Barrier
- ...

Thread 3
- ...
- Second Access
- Barrier
- ...

Schedule Steering

- Tries to force different orders of execution for greater coverage
  - Pauses thread that is about to access data if not “scheduled thread”
  - If incorrect order, reports a timeout to the Hive, which may increase pause time up to a maximum
- Timeout generally kept small for low overhead
- Successful: found races otherwise undetected
Dynamic Validation

- Timeout delta < max
- Race detected
- No Race detected
- Timeout delta >= max
- True Race
-race detected
- Likely False Positive

Unknown
Results

- 106 total data races in 10 programs
- 0% false positive for detected races
- Efficiency
  - Runtime overhead average 2.32%, maximum 4.54%
- Found 2 previously undiscovered hangs in SQLite
## Comparison: Reported Races

<table>
<thead>
<tr>
<th>Program</th>
<th>Apache</th>
<th>SQLite</th>
<th>Fmm</th>
<th>Aget</th>
<th>Pfscan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RaceMob</td>
<td>8</td>
<td>3</td>
<td>58</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>TSAN</td>
<td>8</td>
<td>3</td>
<td>58</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>RELAY</td>
<td>118</td>
<td>88</td>
<td>176</td>
<td>256</td>
<td>17</td>
</tr>
</tbody>
</table>
## Comparison: Total Overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Apache</th>
<th>SQLite</th>
<th>Fmm</th>
<th>Aget</th>
<th>Pfscan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RaceMob Aggregate Overhead</td>
<td>339%</td>
<td>282%</td>
<td>1598%</td>
<td>144%</td>
<td>103%</td>
</tr>
<tr>
<td>TSAN Average Overhead</td>
<td>25,208%</td>
<td>1429%</td>
<td>47888%</td>
<td>184%</td>
<td>13402%</td>
</tr>
</tbody>
</table>
Issues with RaceMob

- Additional overhead for client
- Less in-house testing/releasing buggy software?
- Privacy implications
- Crowdsourcing with dishonest or malicious users
Final Thoughts

- Innovative combination of static and dynamic methods
- Much more accurate and with lower overhead than many of today’s standard tools
- Questions concerning privacy with crowdsourcing