Lightweight Data Race Detection for Production Runs

Swarndhu Biswas, UT Austin
Man Cao, Ohio State University
Minjia Zhang, Microsoft Research
Michael D. Bond, Ohio State University
Benjamin P. Wood, Wellesley College

CC 2017
A Java Program With a Data Race

Object $X = \text{null}$;
boolean done = false;

Thread T1

$X = \text{new } \text{Object}();$
done = true;

Thread T2

while (!done) {}  
$X.\text{compute}();$
Object X = null;
boolean done= false;

Thread T1
X = new Object();
done = true;

Thread T2
while (!done) {}
X.compute();

Data race

Conflicting accesses – two threads access the same shared variable where at least one access is a write

Concurrent accesses – accesses are not ordered by synchronization operations
Thread T1

X = new Object();

done = true;

Thread T2

temp = done;

while (!temp) {}

Infinite loop

Thread T1

done = true;

Thread T2

while (!done) {}

X.compute();

NPE
Data Races are Evil

- Challenging to reason about the correctness of racy executions
  - May unpredictably break code

- Lack of semantic guarantees in most mainstream multithreaded languages

- Usually indicate other concurrency errors
  - Atomicity, order, or sequential consistency violations

---

Far-Reaching Impact of Data Races

Therac-25 accidents, 1985-87

Technical Perspective

Data Races are Evil with No Exceptions

By Sarita Adve

How to miscompile programs with "benign" data races

Hans-J. Boehm
HP Laboratories
Get Rid of Data Races!!!

Avoiding and/or eliminating data races efficiently is a challenging and unsolved problem.
Data Race Detection on Production Systems

Avoiding and/or eliminating data races efficiently is a challenging and unsolved problem

No satisfactory solution to date
Data Race Detection Techniques

Static and predictive analyses

- Too many false positives, do not scale
## Data Race Detection Techniques

<table>
<thead>
<tr>
<th>Dynamic analysis</th>
<th>Lockset analysis</th>
<th>Happens-before analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expensive, reports many false positives</td>
<td>Sound and precise</td>
</tr>
<tr>
<td>sound</td>
<td>Expensive, not scalable, incurs space overhead</td>
<td>Coverage limited to observed executions</td>
</tr>
<tr>
<td>precise</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Existing Approaches for Data Race Detection on Production Runs

• **Happens-before-based sampling approaches**
  • E.g., LiteRace\textsuperscript{1}, Pacer\textsuperscript{2}
  • Overheads are still too high for a reasonable sampling rate
    • Pacer with 3% sampling rate incurs 86% overhead!!!
Existing Approaches for Data Race Detection on Production Runs

• Happens-before-based sampling approaches
  • E.g., LiteRace\textsuperscript{1}, Pacer\textsuperscript{2}
  • Overheads are still too high for a reasonable sampling rate
    • Pacer with 3% sampling rate incurs 86% overhead!!!

• RaceMob\textsuperscript{3}
  • Optimizes tracking of happens-before relations
  • Monitors only one race per run to minimize overhead
  • Cannot bound overhead, limited scalability and coverage

Existing Approaches for Data Race Detection on Production Runs

- **DataCollider**\(^4\)
  - Tries to collide racy accesses, synchronization oblivious
  - Samples accesses, and uses hardware debug registers for performance
  - Dependence on debug registers
    - Not portable, and may not scale well
    - Few debug registers
    - Cannot bound overhead

---

Outline

Data Races

Problems and Challenges

Data Race Detection in Production Systems

Drawbacks of existing approaches

Our contribution: efficient, complementary analyses

RaceChaser: Precise data race detection

Caper: Sound data race detection
Our Insight

Decouple data race detection into two lightweight and complementary analysis
Our Contributions

Decouple data race detection into two lightweight and complementary analysis

RaceChaser – Precise data race detector
• Under-approximates data races

Caper – Dynamically sound data race detector
• Over-approximates data races

Can miss true data races
Can report false data races
RaceChaser: Precise Data Race Detection

<table>
<thead>
<tr>
<th>Desired Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Performance and Scalability ?</td>
</tr>
<tr>
<td>● Bounded time and space overhead ?</td>
</tr>
<tr>
<td>● Coverage and Portability ?</td>
</tr>
</tbody>
</table>

**Design:**
- Monitor one data race (two source locations) per run
- Use collision analysis
- Bound overhead introduced
RaceChaser Algorithm

Two static sites involved in a potential data race

Max overhead 5%

avrora.sim.radio.Medium:access$302() byte offset 0
avrora.sim.radio.Medium:access$402() byte offset 2

True data race!

Data race not reproduced
Instrumenting Racy Accesses

- Limited to one potential race pair

avrora.sim.radio.Medium: access$302() byte offset 0

avrora.sim.radio.Medium: access$402() byte offset 2
Randomly Sample Racy Accesses

- Use frequency of samples taken
- Compute overhead introduced by waiting

```
avrora.sim.radio.Medium: access$302() byte offset 0
```
```
avrora.sim.radio.Medium: access$402() byte offset 2
```

Dynamic instance 992
Dynamic instance 993

Sampled
Try to Collide Racy Accesses

- Block thread for some time

Dynamic instance 992

Dynamic instance 993

Sampled

avrora.sim.radio.Medium: access$302() byte offset 0

avrora.sim.radio.Medium: access$402() byte offset 2
Collision is Successful

Dynamic instance 992

Dynamic instance 993

Sampled

True data race detected

Dynamic instance 215

avrora.sim.radio.Medium: access$302() byte offset 0

avrora.sim.radio.Medium: access$402() byte offset 2
Collision is Unsuccessful

- Thread unblocks, resets the analysis state, and continues execution

```
avrora.sim.radio.Medium: access$302() byte offset 0
```

```
avrora.sim.radio.Medium: access$402() byte offset 2
```

Dynamic instance 992

Dynamic instance 993

Sampled

Next instruction
Evaluation of RaceChaser

- Implementation is publicly available
- Jikes RVM 3.1.3
- Platform: 64-core AMD Opteron 6272
- Benchmarks:
  - Large workload sizes of DaCapo 2006 and 9.12-
  - Fixed-workload versions of SPECjbb2000 and SPECjbb2005

- Benchmarks:
  - Large workload sizes of DaCapo 2006 and 9.12-
  - Fixed-workload versions of SPECjbb2000 and SPECjbb2005
Run-time Overhead (%) of RaceChaser

Effectiveness of RaceChaser

• Collision analysis can potentially detect data races that are hidden by spurious happens-before relations

• Data race coverage of collision analysis depends on the perturbation and the delay
  • Prior studies seem to indicate that data races often occur close in time

• RaceChaser did as well as RaceMob/LiteHB over a number of runs
Outline

Data Races

Problems and Challenges

Data Race Detection in Production Systems

Drawbacks of existing approaches

Our contribution: efficient, complementary analyses

RaceChaser: Precise data race detection

Caper: Sound data race detection
Sound, Efficient Data Race Detection

Options

- Use static analysis offline
  - Too many false positives

- Use dynamic analysis online
  - Efficient enough for production runs?
Caper: Sound Data Race Detection

Input program → multiple runs → Caper algorithm

Static analysis

Dynamic analysis

dynamically sound

Set of potential race pairs
Caper Algorithm

Input program → Static analysis → Dynamic analysis → Set of potential race pairs

Input program → Static data race detector → Static race pairs (spPairs) → Dynamic analysis → multiple runs → dynamic race pairs (dpPairs)
Sound Dynamic Escape Analysis for Data Race Detection

Reachability-based analysis

\[ q.f = p \]
Caper’s Dynamic Analysis

deSites = \{ s \mid ( \exists s' \mid \langle s, s' \rangle \in spPairs \cup dpPairs) \land s \text{ escaped in an analyzed execution} \} 

dpPairs = \{ \langle s_1, s_2 \rangle \mid s_1 \in deSites \land s_2 \in deSites \}
Run-time Overhead (%) of Caper

- DEA
- Caper (first run)
- Caper (steady state)
## Effectiveness of Caper

<table>
<thead>
<tr>
<th></th>
<th>Sound static data race detector</th>
<th>Caper</th>
<th>Dynamic alias analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsqldb6</td>
<td>212,205</td>
<td>1,612</td>
<td>757</td>
</tr>
<tr>
<td>lusearch6</td>
<td>4,692</td>
<td>302</td>
<td>292</td>
</tr>
<tr>
<td>xalan6</td>
<td>83,488</td>
<td>1,241</td>
<td>581</td>
</tr>
<tr>
<td>avrora9</td>
<td>61,193</td>
<td>19,941</td>
<td>570</td>
</tr>
<tr>
<td>luindex9</td>
<td>10,257</td>
<td>192</td>
<td>193</td>
</tr>
<tr>
<td>lusearch9</td>
<td>7,303</td>
<td>34</td>
<td>39</td>
</tr>
<tr>
<td>sunflow9</td>
<td>28,587</td>
<td>200</td>
<td>1,086</td>
</tr>
<tr>
<td>xalan9</td>
<td>20,036</td>
<td>1,861</td>
<td>600</td>
</tr>
<tr>
<td>pjbb2000</td>
<td>29,604</td>
<td>11,243</td>
<td>1,679</td>
</tr>
<tr>
<td>pjbb2005</td>
<td>2,552</td>
<td>984</td>
<td>447</td>
</tr>
</tbody>
</table>
Efficiency vs Precision

- Static data race detector, e.g., Chord
- Caper
- Dynamic alias analysis
Usefulness of Caper

• Improve performance of analyses whose correctness relies on knowing all data races
  
  Record and replay systems
  Atomicity checking
  Software transactional memory

• Generate potential data races for analyses like RaceChaser/RaceMob/DataCollider
Lightweight Data Race Detection for Production Runs

Swarwendu Biswas, UT Austin
Man Cao, Ohio State University
Minjia Zhang, Microsoft Research
Michael D. Bond, Ohio State University
Benjamin P. Wood, Wellesley College

CC 2017