Do you have to reproduce the bug on the first replay attempt?

PRES: Probabilistic Replay with Execution Sketching on Multiprocessors

Soyeon Park, Yuanyuan Zhou
University of California, San Diego

Weiwei Xiong, Zuoning Yin, Rini Kaushik, Kyu H. Lee, Shan Lu
University of Illinois at Urbana Champaign
Concurrency bugs are important

- Writing concurrent program is difficult
  - Programmers are used to sequential thinking
    → Concurrent programs are prone to bugs

- Concurrency bugs cause severe real-world problems
  - Therac-25, Northeast blackout

- Multi-core trend worsens the problem
Characteristics of Concurrency Bugs

- A concurrency bug may need a special thread interleaving to manifest

```c
if ( buf_index + len < BUFSIZE )
    memcpy(buf[buf_index], log, len);
```

Thread 1

Thread 2

```
buf_index += len;
```

Crash!

Two implications:

- Hard to expose a concurrency bug during testing
- Difficult to reproduce a concurrency bug for diagnosis
Recording non-deterministic factors and re-execution

- Inputs (keyboards, networks, files, etc)
- Thread scheduling
- Return values of system calls

Deterministic Replay of Uniprocessor

< Production run >

< Replay run >
Deterministic Replay for Multiprocessors

- Much more difficult
  - Multi-threads execute simultaneously on different processors

- Extra source of non-determinism:
  - Interleaving of shared memory accesses

```
S1: if (buf_index + len < BUFFSIZE);
S2: buf_index += len;
S3: memcpy (buf [buf_index], log, len);

Crash!
```
State of the Art on Multiprocessor Replay

- Hardware-assisted approach
  - Recording all thread interactions with new hardware extension
    - ex) Flight Data Recorder, BugNet, Strata, RTR, DMP, Rerun, etc.
    - None of them exists in reality!

- Software-only approach
  - High production-run overhead (> 10-100X) ➡ Not practical!
    - due to capturing the global order of shared memory accesses
    - ex) InstantReplay, Strata/s, etc.

- Recent work: SMP-Revirt
  - use page protection mechanism to optimize memory monitoring
  - >10X production-run overhead on 2 or 4 processors
  - has false sharing and page contention issues (scalability)
Contrast between Common Practice & Existing Research Proposals

Production run

Common practice

0% overhead

Existing research proposals

10-100 X slowdown

Impractical!

Diagnosis phase

>1000 replay attempts*

the 1st replay attempt

* : according to our experimental results
1) **Production run performance** is more critical than replay time.

2) **We do NOT** need to reproduce a bug on the **1st replay attempt**.
Our Idea

Probabilistic Replay with Execution Sketching (PRES)

- Record only partial information during production run
  - Low recording overhead

- Push the complexity to diagnosis time

- Leverage feedback from unsuccessful replays
Probabilistic Replay via Execution Sketching (PRES)

- Sketch recording during production run
- Partial-information based replay (PI-Replay)
- Diagnosis phase

- Recording partial information (sketch) during production run
- Reproducing a bug, not the original execution
Contents

- Introduction
- Our approach
- Overview of PRES
  - Sketch recording
  - Bug reproduction
    - Partial-Information based replayer
    - Monitor
    - Feedback generator
- Evaluation
- Conclusion
Sketch Recording

- BASE: Uni-processor deterministic replay
- Existing s/w only deterministic replay for multi-processors
- Thread deterministic events including
- System global order of shared memory access
- Sync global order of shared memory access
- BASE + global order of basic-blocks
- BASE + global order of function calls
- BASE + global order of shared memory read/write accesses
- BASE + global order of synchronization operations
- SYNc + global order of system calls
- SYNC + global order of shared memory access

```c
if (myid==0) result = data;
```

```c
worker()
{
 lock (L);
 myid = gid;
 gid = myid+1;
 unlock (L);
 ...
}
```

```c
if (myid==0) result = data;
```
Contents

- Introduction
- Our approach
- Overview of PRES
- Sketch recording
- **Bug reproduction**
  - Partial-Information based replayer (PI-Replayer)
  - Monitor
  - Feedback generator
- Evaluation
- Conclusion
Partial Information-based Replay

- Process of bug reproduction phase

Monitor is used for:
- Detecting successful bug reproduction
- Detecting off-sketch path: deviates from sketches
Partial-Information based replayer

- Consults the execution sketch to enforce observed global orders
- Right before re-executing a sketch point, make sure that all prior points from other threads have been executed

< Production run >

T1 : lock A, global order 1
T2 : lock B, global order 2

< Replay run >

T1 : lock A, global order 1
T2 : lock B, global order 2

Wait for T1 to execute lock A first
Monitor

- **Detect successful bug reproduction**
  - Crash failure - PRES can catch exceptions
  - Deadlock - a periodic timer to check for progress
  - Incorrect results - programmer needs to provide conditions for checking
  - Can leverage testing oracles and existing bug detection tools

- **Detect unsuccessful replay**
  - Compare against the execution sketch from the original execution
  - Prevent from giving useless replay efforts on a wrong path
What if a replay attempt fails?

- Replay it again!
  - Restart from the beginning or the previous checkpoint

- Shall we do something different next time?
  - Random approach: just leave it to fate
  - Systematic approach
    - Actively learn from previous mistakes
Why previous replays cannot reproduce a bug?

- Some un-recorded data races execute in different orders

---

Production run

```c
worker()
{
    ...
}

if (myid==0)
    result = data;

tmp = result;
printf("%d\n", tmp);
```

Thread 1

1st replay attempt

```c
worker()
{
    ...
}

if (myid==0)
    result = data;

tmp = result;
printf("%d\n", tmp);
```

Thread 2

fail to reproduce the bug!

This original order is not recorded in the sketch
Feedback Generator (2/2)

Steps

- replay recorder
  - R/W traces
  - Identifying dynamic race pairs
    - initial candidates
  - Filtering order-determined races
    - suspect races
  - Selecting suspect
    - one race
  - Starting next replay

- use happens-before race detector
  - sketch
  - unrecorded races
  - The race order is already implied by the order of sketch points
    - close-to-failure-first, depth-first

- deterministically execute until the suspect race pair
- flip the race order
Contents

- Introduction
- Our approach
- Overview of PRES
- Sketch recording
- Bug reproduction
  - Partial-Information based replayer
  - Monitor
  - Feedback generator
- Evaluation
- Conclusion
Methodology

- Implement PRES using Pin
- 8-core Xeon machine
- 11 evaluated applications
  - 4 server applications (Apache, MySQL, Cherokee, OpenLDAP)
  - 3 desktop applications (Mozilla, PBzip2, Transmission)
  - 4 scientific computing applications (Barnes, Radiosity, FMM, LU)
- 13 real-world concurrency bugs
  - 6 Atomicity violation bugs (single- and multi-variable bugs)
  - 4 Order violation bugs
  - 3 Deadlock bugs
Recording Overhead (1/2)

- **Non-server applications**

<table>
<thead>
<tr>
<th>Application</th>
<th>PIN</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Applications (overhead %)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla</td>
<td>32.5</td>
<td>59.5</td>
<td>60.6</td>
<td>83.8</td>
<td>598.0</td>
<td>858.8</td>
<td>1213.9</td>
<td>3093.5</td>
</tr>
<tr>
<td>PBZip2</td>
<td>17.4</td>
<td><strong>18.0</strong></td>
<td><strong>18.0</strong></td>
<td>18.4</td>
<td>595.4</td>
<td>1066.6</td>
<td>1977.9</td>
<td><strong>27009.7</strong></td>
</tr>
<tr>
<td>Transmission</td>
<td>5.9</td>
<td>14.5</td>
<td>21.3</td>
<td>32.7</td>
<td>30.3</td>
<td>33.9</td>
<td>41.9</td>
<td>71.8</td>
</tr>
<tr>
<td>Scientific Applications (overhead %)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barnes</td>
<td>2.5</td>
<td>6.5</td>
<td>6.8</td>
<td>427.7</td>
<td>424.2</td>
<td>1122.3</td>
<td>2351.0</td>
<td>28702.2</td>
</tr>
<tr>
<td>Radiosity</td>
<td>16.0</td>
<td>16.1</td>
<td>16.1</td>
<td>779.0</td>
<td>480.3</td>
<td>1181.7</td>
<td>2425.5</td>
<td>27209.6</td>
</tr>
</tbody>
</table>

- RW : up to around 280 times slowdown
- SYNC, SYS : Good for performance critical applications
  - 6-60% overhead
Recording Overhead (2/2)

- Server application

MySQL

- SYNC, SYS : 3 times higher than RW throughput
## Number of Replay Attempts

NO: not reproduced within 1000 tries

<table>
<thead>
<tr>
<th>Application</th>
<th>Bug type</th>
<th>Base</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td>Atom.</td>
<td>NO</td>
<td>96</td>
<td>28</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MySQL</td>
<td>Atom.</td>
<td>NO</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cherokee</td>
<td>Atom.</td>
<td>NO</td>
<td>33</td>
<td>22</td>
<td>24</td>
<td>25</td>
<td>8</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>Deadlock</td>
<td>NO</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Desktop Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla</td>
<td>Multi-v</td>
<td>NO</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>PBZip2</td>
<td>Atom.</td>
<td>NO</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Transm.</td>
<td>Order</td>
<td>NO</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Scientific Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barnes</td>
<td>Order</td>
<td>NO</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>74</td>
<td>19</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Radiosity</td>
<td>Order</td>
<td>NO</td>
<td><strong>NO</strong></td>
<td><strong>NO</strong></td>
<td><strong>152</strong></td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

12 out of all 13 bugs mostly within 10 attempts

Reproduce all 13 bugs mostly within 5 attempts
**Benefit of Feedback Generation**

<table>
<thead>
<tr>
<th>Application</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apache</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>96</td>
<td>28</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>w/o</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>754</td>
<td>1</td>
</tr>
<tr>
<td><strong>PBZip2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>w/o</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td><strong>Barnes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>74</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>w/o</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

- Low-overhead sketches (SYNC and SYS) can be used to reproduce concurrency bugs **only because of our PI-replayer that leverages feedback**

- The random approach does NOT work!
Effects of Race Filtering

The number of dynamic benign data races to be explored

<table>
<thead>
<tr>
<th>Applications</th>
<th>BASE</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>54390</td>
<td>1072</td>
<td>274</td>
<td>33</td>
<td>25</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>MySQL</td>
<td>39983</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cherokee</td>
<td>133</td>
<td>86</td>
<td>58</td>
<td>16</td>
<td>36</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Mozilla</td>
<td>36258</td>
<td>317</td>
<td>310</td>
<td>14</td>
<td>72</td>
<td>60</td>
<td>42</td>
</tr>
<tr>
<td>PBZip2</td>
<td>667</td>
<td>326</td>
<td>318</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Transmission</td>
<td>225</td>
<td>240</td>
<td>172</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

- Filters out the races whose order can be inferred from sketches

- Significantly shrinks the unrecorded non-deterministic space to be explored
## Bugs vs. Execution Reproduction

The number of replays for bug reproduction (BR) vs. those for execution reproduction (ER)

<table>
<thead>
<tr>
<th>Application</th>
<th>SYNC</th>
<th>SYS</th>
<th>FUNC</th>
<th>BB-5</th>
<th>BB-2</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ER</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>PBZip2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ER</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

- Reproducing the exact same execution path requires 1.6-5 times more attempts with SYS and SYNC.
Scalability

**Radiosity**

- BASE
- PIN
- SYNC
- SYS
- FUNC
- BB-5
- BB-2
- BB
- RW

<table>
<thead>
<tr>
<th>number of processors</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>25</th>
<th>273</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized elapse time (the lower, the better)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2-core: SYNC 6.2%, SYS 11.8% (770% by SMP-Revirt)

**MySQL**

<table>
<thead>
<tr>
<th>number of processors</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized throughput of a server application (the higher, the better)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- BASE
- PIN
- SYNC
- SYS
- FUNC
- BB-5
- BB-2
- BB
- RW
Conclusions

- Software solutions can be **practical** for reproducing concurrency bugs on multiprocessor

- PRES: Probabilistic Replay via Execution Sketch
  - No need to reproduce the bug at the first attempt
  - Trade replay-time efficiency for lower recording overhead
  - PI-Replayer (that leverages feedback) makes low-overhead sketches (SYS, SYNC) useful for bug reproduction