Efficient Online Validation with Delta Execution

Yuke Liao: liaoyuke@umich.edu
Shenghua Guan: shenguan@umich.edu
Agenda

- Motivation
- Feasibility & Challenges
- Implementation Details
- Evaluation Methodology
- Experimental Results
- Conclusion
- Discussion
Motivation

Lines of Code:

- Windows 7: 40 Million
- Mac OS X “Tiger”: 85 Million
- Google Internet Service: 2 Billion

Source: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/
Motivation

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL INITIALIZATION FAILED
Motivation

November’s Patch Tuesday

IS THIS BIG
How to Validate the Patches?

- Nearly 70% of patches are buggy in first release
- Validation
  - Correctness
  - Efficiency
Why Not Offline Validation?

Although simple enough... the test environment may not reflect actual production usage.

-> Low Accuracy
How About Traditional Online Validation?

- Performance degradation
- Expensive
- Non-determinism output
Key Observations

MAREs: multiple almost redundant executions

Patch: tar null pt
tar/src/incremen.c
....
+if(dirp){
  if(children!=NO_CHILDREN)
    for(entry=dirp; ...){
      //main loop
    }
  free(dirp);
+}

Patch: CAN-2004-0493
httpd-2.0/server/protocol.c
....
+if((fold_len-1)>
  + r->server->limit){
  + r->status = BAD_REQUEST;
  + return;
+}

Patch: CAN-2004-0811
httpd-2.0/server/core.c
....
+if(new->satisfy[i] !=
  + SATIFY_NOSPEC){
    conf->satisfy[i]=new->satisfy[i];
  +}else{
    + conf->satisfy[i]=base->satisfy[i];
+}
Delta Execution

Run only the differences separately
Feasibility of Delta Execution

**Intuition**: programmers are encouraged to provide small patches

**Case Study**: 

Effective for 83% -> delta execution is promising!
Why Delta Execution?

- Low overhead - Efficient
- Reduce non-determinism - Accurate
Challenges

- When to split?
- When to merge?
- How to track modified data (IO)?
- Threads...
Implementation Details

- Basic Delta execution
  - Splitting:
    - Δ code -- fork()
    - Δ data -- mprotect deny access
  - Running split: mprotect to re-enable
  - Merging
    - when -- check every function return
    - how -- Δ pages
Dealing with I/O

- Problem: non-idempotent
- Solution: instrument read and write
  - validate writes
  - non-idempotent read once
  - expensive operation once
Maximizing Merged Execution

With aligning allocations

| 10 | 15 | 5 | 2 |

| 10 | 15 | 5 | 2 |

Original

| 10 | 5 | 2 |

Patched

| 15 | 2 |

Legend

Requested allocation  Extra allocation
Reducing Split/Merge Overhead

- Save processor context
- Run unmodified version
- Run modified version
- Comparison

=> avoid synchronization time
Handling Small $\Delta$ in Large Functions

DELTA_START
//This is common code
D_ORIG_START
//This was removed in the patch
D_ORIG_END
D_MOD_START
//This was added in the patch
D_MOD_END
DELTA_STOP

(b) Supporting small delta segments
Adaptive Mechanism for Worst Case Situation

Competitive analysis for worst case situation

Type changes:

\[
\frac{\epsilon \cdot \text{time\_merged} + \text{time\_split}}{\text{epoch\_length}} \geq 1
\]
Dealing with Threads

Challenges:

- How to split -- duplicate threads
- Thread creation / destruction during split
Challenge 1: fork() not duplicating threads

Solution:

- Modified fork() call -- intrusive
- Instrumentation -- recreate threads
Challenge 1: fork() not duplicating threads

Solution:

- Modified fork() call -- intrusive
- Instrumentation -- recreate threads
- **Disable other threads -- signal**
  - pro: no need to recreate
  - con: deadlock (rare)
Challenge 2: Thread creation/destruction mismatch

- Extremely rare
- Solution:
  - Side-by-side validation
Evaluation Methodology

Comparison based evaluation:

Baseline: traditional side-by-side online validation

V.S.

Delta execution with Pin instrumentation tool
Overhead Breakdown

Overhead

- Dynamic instrumentation
- Run split code

A nullpin tool is injected to achieve this!
# Experimental Results - Effectiveness

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Program</th>
<th>Change Description</th>
<th>Baseline</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>crafty</td>
<td>Chess Program</td>
<td>Code refactoring</td>
<td>fails–kibitzes differ</td>
<td>pass</td>
</tr>
<tr>
<td>raytrace</td>
<td>Raytracer</td>
<td>Fixed bug in result reporting</td>
<td>large fail–nondeterminism</td>
<td>pass</td>
</tr>
<tr>
<td>tar</td>
<td>Archive Utility</td>
<td>Fixed incremental archiving</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Apache 1</td>
<td>Web Server</td>
<td>Fixed overflow in mod_alias</td>
<td>fails–randomized etags</td>
<td>pass</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Web Server</td>
<td>Fixed overflow in mime parser</td>
<td>fails–randomized etags</td>
<td>pass</td>
</tr>
<tr>
<td>ATPhttpd</td>
<td>Web Server</td>
<td>Fixed overflow in HTTP parsing</td>
<td>small fails–timestamps</td>
<td>pass</td>
</tr>
<tr>
<td>DNSSCache</td>
<td>DNS Cache</td>
<td>Behavior change</td>
<td>fails–timing &amp; random Tx IDs</td>
<td>pass</td>
</tr>
<tr>
<td>MySQL 5.0</td>
<td>Database Server</td>
<td>Extra permission checks</td>
<td>large fail–nondeterminism</td>
<td>pass</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>Security Library</td>
<td>Added bug in TLS handling</td>
<td>MAJOR fail–cannot validate</td>
<td>pass</td>
</tr>
<tr>
<td>squid</td>
<td>Web Cache</td>
<td>Fixed overflow in FTP parsing</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>
Experimental Results - Performance

- Overall: 12% faster
- 74% not considering Pin tool
## Detailed Time Breakdown

<table>
<thead>
<tr>
<th></th>
<th>crafty</th>
<th>raytrace</th>
<th>tar</th>
<th>Apache1</th>
<th>Apache2</th>
<th>ATPhttpd</th>
<th>DNSScache</th>
<th>MySQL</th>
<th>OpenSSL</th>
<th>squid</th>
</tr>
</thead>
<tbody>
<tr>
<td>splits/sec</td>
<td>.005</td>
<td>.037</td>
<td>5.36</td>
<td>.368</td>
<td>6.560</td>
<td>19.10</td>
<td>9.638</td>
<td>.520</td>
<td>11.70</td>
<td>.903</td>
</tr>
<tr>
<td>%merged</td>
<td>99.996</td>
<td>99.134</td>
<td>45.40</td>
<td>94.500</td>
<td>72.900</td>
<td>12.10</td>
<td>55.691</td>
<td>87.827</td>
<td>59.500</td>
<td>88.200</td>
</tr>
<tr>
<td>%split</td>
<td>.001</td>
<td>.696</td>
<td>10.50</td>
<td>.002</td>
<td>.081</td>
<td>3.85</td>
<td>2.164</td>
<td>5.122</td>
<td>.229</td>
<td>.358</td>
</tr>
<tr>
<td>%splitting</td>
<td>.011</td>
<td>.130</td>
<td>3.08</td>
<td>.072</td>
<td>1.800</td>
<td>16.30</td>
<td>17.178</td>
<td>.469</td>
<td>6.290</td>
<td>0.896</td>
</tr>
<tr>
<td>%merging</td>
<td>.035</td>
<td>.055</td>
<td>40.60</td>
<td>2.420</td>
<td>25.20</td>
<td>67.60</td>
<td>24.917</td>
<td>7.440</td>
<td>33.900</td>
<td>10.500</td>
</tr>
</tbody>
</table>
Conclusions

- Reduced overhead of online validation
- Reduced spurious differences
- More to be done
  - complex threading behaviors
  - widen applicability (e.g. configuration)
- Resources: Automatic Exploit Generation
Discussion

- This paper is wrote 6 years ago, does the mechanism work well now? What factors may affect its effectiveness?
- Whether applicable for configuration changes?
- Will the adaptive mechanism work well in practice?