Insight: In-situ Online Service Failure Path Inference in Production Computing Infrastructures

Hiep Nguyen,
Daniel J. Dean, Kamal Kc, Xiaohui Gu
North Carolina State University
Online Services

Virtual Computing Lab (VCL)

Reservation servers

Resource pool

Setup

Copy image

Check image

Image storage

Database

Poll

Push

Web server

Request
Online Service Non-crashing Failures

- Online services are prone to non-crashing failures
  - Number of non-crashing failures in VCL: 154 in one month (April 2014); 1813 in one year (2013)
  - Non-crashing failures often go unnoticed
  - Error message does not tell why failures occur

lighttpd failure:

```
```

Authentication failure in VCL:

```
|2836|66:58|new| ---- WARNING ----
new.pm:process(295)|failed to load vmsk1 with kvmlinux-v0
...
|2836|66:58|new| ---- CRITICAL ----
State.pm:reservation_failed(213)| reservation failed on vmsk1: process failed after trying to load or make available
```
Approach 1: Offline Failure Reproduction

- Offline failure reproduction is hard
  - Lacking environment information
  - Missing interacting components
  - Absent third-party libraries
Approach 2: Record and Replay

- Intrusive system recording
  - High overhead
  - Privacy concerns
  - Deployment challenges
Approach 3: Onsite Failure Diagnosis

- **Triage:** diagnosing production run failures at the user’s site [Tucek et al. SOSP’07]
  - Relies on repeated replays to compare good and bad runs
  - Requires runtime checkpointing for replays
  - Performs diagnosis directly on the production server
  - Can incur long service downtime
  - Does not fully leverage runtime environment data
Our Key Observations

- **Production environments provide lots of clues**
  - Environment data: inputs, configuration files, interacting components
  - Runtime outputs: console logs, system call traces

- **Onsite failure path search is more efficient**
  - Significantly smaller search scope
  - Does not require intrusive recording for replay

- **Decouple failure analysis from the production service execution**
  - Capture runtime state using dynamic virtual machine cloning
  - Minimize production service downtime
Our Approach

Virtual Computing Lab (VCL)

Reservation servers

Shadow server

Matched failure paths

Matched failure paths

Resource pool

Image storage

Web server

Database

Clone

Binary execution exploration

Web server
Challenges

- **No source code access**
  - Binary-based approach

- **Fast failure path inference**
  - Leverage “fresh” environment data at the failure moment

- **Low overhead**
  - No intrusive recording

- **Different programming languages**
  - Compiled programs (e.g., C/C++)
  - Interpreted programs (e.g., Perl, Java)
Building Blocks

- **Dynamic** shadow server creation
  - Use live VM cloning
  - Decouple analysis from the production run

- **Guided** binary execution exploration
  - Leverage the production environment data and runtime outputs as guidance to search the failure paths
  - No source code required
Dynamic Shadow Server Creation

- **Automatic reconfiguration**
  - Reset the IP address for the shadow server
  - Reconfigure firewall

- **Policy-driven output filtering proxy**
Console log:

1. Checking request state in database
2. Start processing reservation

```
1  log("Checking request state in database");
2  my @selected_rows = database_select($select_statement);
3  if ((scalar @selected_rows) == 0) {
   log("0 rows returned from request state select statement, request was probably deleted, returning 0");
   return 0;
} else {

4  if ((scalar @selected_rows) > 1) {
   log("More than 1 row returned from request state select statement, returning 0");
   return 0;
5   } else {
6   log("Start processing reservation");
7   } 
8 }
9 }
10 } else {
11 log("Start processing reservation");
12 }
13 }
```
Problem with Sparse Console Log

- Too many possible paths

Console log:

1. Create hard link `./dir1/file1' to `./file1'

2. `.': hard link not allowed for directory

Create hard link `./dir1/file1' to `./file1'

Diagram:

```
  B_1
  /\    /
 True/  \
 False/   \
 B_2
  /\    /
 True/  False/   \
 B_3
 /\    /
 True/  False/   \
 B_4
 /\    /
 True/  False/   \
 B_5
 /\    /
 True/  False/   
```

`.': hard link not allowed for directory
Leveraging System Call Sequences

Console log:

1. Create hard link `./dir1/file1` to `./file1`
   - `sys_poll`
   - `sys_read`
   - `sys_read`

2. `.`: hard link not allowed for directory
Implementation

- Currently support Perl and C/C++ programs
  - Modified Perl interpreter for Perl programs
    - Add handling for branch opcodes (e.g., \texttt{OP\_COND\_EXPR})
    - Intercept interpreter’s execution stack to change branch conditions
  - Pin tool for C/C++ programs
    - Intercept branch statements (e.g., \texttt{JZ}, \texttt{JNE}, \texttt{JE})
    - Modify the branch conditions by changing EFLAGS register
  - Uses SystemTap for monitoring system calls
## Tested Failure Cases

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
<th>Failure path length (Num. of functions)</th>
<th>Failure name</th>
<th>Num. of console log messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCL</td>
<td>145K</td>
<td>112</td>
<td>Overlapping reservation failure</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td></td>
<td>299</td>
<td>Network failure</td>
<td>290</td>
</tr>
<tr>
<td></td>
<td></td>
<td>298</td>
<td>Authentication failure</td>
<td>409</td>
</tr>
<tr>
<td></td>
<td></td>
<td>147</td>
<td>Image corruption failure</td>
<td>178</td>
</tr>
<tr>
<td>Apache</td>
<td>176K</td>
<td>176</td>
<td>Authentication failure</td>
<td>1</td>
</tr>
<tr>
<td>httpd</td>
<td></td>
<td>164</td>
<td>CGI failure</td>
<td>1</td>
</tr>
<tr>
<td>Squid</td>
<td>110K</td>
<td>588</td>
<td>Non-crashing stop failure</td>
<td>195</td>
</tr>
<tr>
<td>Lighttpd</td>
<td>38K</td>
<td>730</td>
<td>Proxy failure</td>
<td>3</td>
</tr>
<tr>
<td>PBZIP2</td>
<td>3.9K</td>
<td>41</td>
<td>Decompression failure</td>
<td>14</td>
</tr>
<tr>
<td>aget</td>
<td>1.5K</td>
<td>2</td>
<td>Download failure</td>
<td>1</td>
</tr>
<tr>
<td>rmdir</td>
<td>0.2K</td>
<td>2</td>
<td>Option failure</td>
<td>2</td>
</tr>
<tr>
<td>ln</td>
<td>0.6K</td>
<td>1</td>
<td>Option failure</td>
<td>1</td>
</tr>
<tr>
<td>touch</td>
<td>0.5K</td>
<td>1</td>
<td>Time failure</td>
<td>1</td>
</tr>
</tbody>
</table>

Production failures

Reported open source software failures
# Call Path Difference in VCL Failures

<table>
<thead>
<tr>
<th>Failure name</th>
<th>Complete environment data</th>
<th>Partial environment data</th>
<th>No environment data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlapping reservation failure</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Network failure</td>
<td>0</td>
<td>0</td>
<td>Failed</td>
</tr>
<tr>
<td>Authentication failure</td>
<td>0</td>
<td>0</td>
<td>Failed</td>
</tr>
<tr>
<td>Image corruption failure</td>
<td>0</td>
<td>0</td>
<td>Failed</td>
</tr>
</tbody>
</table>
Call Path Difference in Open Source Software Failures

- Reproduced failure paths always cover root cause functions and branches

<table>
<thead>
<tr>
<th>Failure name</th>
<th>Original input</th>
<th>Same input type + console log</th>
<th>Same input type + console log + system call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache (authentication failure)</td>
<td>0</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Apache (CGI failure)</td>
<td>0</td>
<td>140</td>
<td>9</td>
</tr>
<tr>
<td>Squid (non-crashing stop failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lighttpd (proxy failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PBZIP2 (decompression failure)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>aget (download failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rmdir (option failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ln (option failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>touch (time failure)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Failure Reproduction Time

VCL failures

Open source software failures
Overhead

Performance impact

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow system</td>
<td>&lt; 0.3%</td>
</tr>
<tr>
<td>System call tracing</td>
<td>&lt; 1.5%</td>
</tr>
<tr>
<td>Shadow creation time</td>
<td>&lt; 30s</td>
</tr>
<tr>
<td>Stop-and-copy time</td>
<td>&lt; 70ms</td>
</tr>
</tbody>
</table>

Logging overhead (1 day)

<table>
<thead>
<tr>
<th>System</th>
<th>Input log</th>
<th>Console log</th>
<th>System call log</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCL</td>
<td>130MB</td>
<td>490MB</td>
<td>N/A</td>
</tr>
<tr>
<td>Apache</td>
<td>20MB</td>
<td>0.3MB</td>
<td>12MB</td>
</tr>
</tbody>
</table>
Conclusion

- **Insight: In-situ failure path inference system**
  - Enable failure path inference *inside* the production environment
  - Use *shadow component* to decouple failure analysis from the production run
  - *Guided* binary execution exploration to find high fidelity failure paths quickly without source code

Thank you!
Acknowledgement

- This work was sponsored in part by NSF CNS0915567 grant, NSF CNS0915861 grant, U.S. Army Research Office (ARO) under grant W911NF-10-1-0273, and Google Research Awards