Rx: Treating Bugs As Allergies: A Safe Method to Survive Software Failures

Feng Qin, Joseph Tuccek, Jagadeesan Sundaresan and Yuanyuan Zhou
University of Illinois at Urbana Champaign
Presented at SOSP 05
Outline

- Introduction
- Main idea
- Architecture
- Design and Implementation Issues
- Evaluation and Results
- Related work
- Conclusion
Introduction
Motivation

- An hour of downtime for a financial company costs $6mil
- Software failures reduce system availability
- Software defects - 40%
- Memory-related+concurrency bugs - 60%
- Cannot get rid of bugs
- Need highly available applications
Previous Solutions

Four categories:
- Rebooting
- checkpointing, rollback, re-execute
- Application-specific recovery
- Speculate on programmer intentions
Allergies are an inspiration

When a person suffers from an allergy, the most common treatment is to remove the allergens from their living environment.

In software, many bugs resemble allergies: their manifestation can be avoided by changing the execution environment.

The idea:

- Rollback the program to a recent checkpoint when a bug is detected.
- Dynamically change the execution environment based on the failure symptoms.
- Re-execute the buggy code region in the new environment.
Examples of allergen bugs

- Memory corruption
- Buffer overrun
- Un-initialized reads
- Data races
- Malicious request
Rx does it better

- Comprehensive
- Safe
- Noninvasive
- Efficient
- Informative
Main idea
Main Idea

- Checkpoint
- Sense bug
- Analyze symptoms and determine cure
- Re-execute from checkpoint
  - New environment
- Repeat until it goes away
  - Or time out
The execution environment

Definition: Almost everything that is external to application:
- Low level: hardware devices, processor architecture..
- Mid level: OS kernel scheduling, virtual memory manager, drivers, file system, network
- High level: standard libraries, third party libraries

Requirement for environmental change
- Correctness-preserving: execute according to the APIs
- Useful: potentially avoid software bugs
Categorizing useful changes

<table>
<thead>
<tr>
<th>Category</th>
<th>Environmental Changes</th>
<th>Potentially-Avoided Bugs</th>
<th>Deterministic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>delayed recycling of freed buffer</td>
<td>double free, dangling pointer</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>padding allocated memory blocks</td>
<td>dynamic buffer overflow</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>allocating memory in an alternate location</td>
<td>memory corruption</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>zero-filling newly allocated memory buffers</td>
<td>uninitialized read</td>
<td>YES</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>scheduling</td>
<td>data race</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>signal delivery</td>
<td>data race</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>message reordering</td>
<td>data race</td>
<td>NO</td>
</tr>
<tr>
<td>User-Related</td>
<td>dropping user requests</td>
<td>bugs related to the dropped request</td>
<td>Depends</td>
</tr>
</tbody>
</table>

Table 1: Possible environmental changes and their potentially-avoided bugs
Working with the changes

- Successful change - record
- Failure - see if it occurred before
- Else
  - Try low overhead changes first
- If failure doesn’t go away with useful change
  - keep rollback to previous checkpoint OR
  - Make another change
Architecture
Rx Design

Figure 2: Rx architecture
Sensors

- dynamically monitoring application execution
  - Exception sensors
  - Bug-specific sensor
- Dynamic bug detection tools
- send failure signature to Control Unit
Checkpoint & Rollback

- Memory snapshots
- File versioning
- Less checkpoint maintenance
Environment Wrappers

- Perform changes in the execution environment (re-execution)
  - Memory wrapper
  - Message wrapper
  - Process scheduling
  - Signal delivery
  - Dropping user requests
Proxy

(a) Proxy behavior in normal mode

(b) Proxy behavior in recovery mode
Control Unit

- Coordinates all of the components in the Rx
- Three functions
  - Directs the checkpointing and rollback process
  - Diagnose failure based on symptoms and experiences
  - Provides feedback to programmer
Design and Implementation Issues
Design and Implementation Issues

- Inter-Server Communication
- Multi-threaded Process Checkpointing
- Unavoidable Bug/Failure of Rx
Evaluation and Results
## Evaluation

<table>
<thead>
<tr>
<th>App</th>
<th>Ver</th>
<th>Bug</th>
<th>#LOC</th>
<th>App Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>4.1.1.a</td>
<td>data race</td>
<td>588K</td>
<td>a database server</td>
</tr>
<tr>
<td>Squid</td>
<td>2.3.s5</td>
<td>buffer overflow</td>
<td>93K</td>
<td>a Web proxy cache server</td>
</tr>
<tr>
<td>Squid-ui</td>
<td>2.3.s5</td>
<td>uninitialized read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squid-dp</td>
<td>2.3.s5</td>
<td>dangling pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td>2.0.47</td>
<td>stack overflow</td>
<td>283K</td>
<td>a Web server</td>
</tr>
<tr>
<td>CVS</td>
<td>1.11.4</td>
<td>double free</td>
<td>114K</td>
<td>a version control server</td>
</tr>
</tbody>
</table>

**Table 2: Applications and Bugs** (App means Application. Ver means Version. LOC means lines of code).
# Effectiveness

<table>
<thead>
<tr>
<th>apps</th>
<th>Bugs</th>
<th>Symptoms</th>
<th>Changes</th>
<th>Restart recovery</th>
<th>Rx recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squid</td>
<td>buffer overflow</td>
<td>SEGV</td>
<td>Padding</td>
<td>5.113s</td>
<td>0.095s</td>
</tr>
<tr>
<td>Squid-ui</td>
<td>uninit. read</td>
<td>SEGV</td>
<td>Zero All</td>
<td>5.000s</td>
<td>0.126s</td>
</tr>
<tr>
<td>Squid-dp</td>
<td>dangling ptr</td>
<td>SEGV</td>
<td>Delay Free</td>
<td>5.006s</td>
<td>0.113s</td>
</tr>
<tr>
<td>Apache</td>
<td>stack overflow</td>
<td>Assert</td>
<td>Drop Req.</td>
<td>1.115s</td>
<td>0.026s</td>
</tr>
<tr>
<td>CVS</td>
<td>double free</td>
<td>SEGV</td>
<td>Delay Free</td>
<td>0.010s</td>
<td>0.017s</td>
</tr>
<tr>
<td>MySQL</td>
<td>data race</td>
<td>SEGV</td>
<td>Sched. Change</td>
<td>3.500s</td>
<td>0.161s</td>
</tr>
</tbody>
</table>
Performance

Figure 5: Throughput and average response time with different bug arrival rates
Performance

(a) Throughput

(b) Average Response Time
Related Work
Related Work

- Recovery-Oriented Computing
- Shadow drivers
- Noisemakers
Conclusion

-safe, non-invasive and informative method for quickly surviving software failures
- Caused by common software defects
- Like all approaches it has its limitations
- It can effectively and efficiently recover from many software failures, but not all