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

Motivation

- Applications require high availability
- Server application downtime leads to lost productivity and lost business
- Average cost of an hour of downtime can exceed six million dollars
- Almost every organization in today’s e-commerce world is dependent on their systems being highly available

General checkpointing and recovery

- Idea: Checkpoint -> Rollback upon failure -> Re-execute
- Problems:
  - Similar problems to restarting techniques, such as inability to handle deterministic bugs

Rebooting Techniques

- Idea: Restart program or parts of program (microreboot) after it crashes
- Problems:
  - Designed for hardware failures, not software
  - Deterministic software failures cannot be dealt with as they will occur every time
  - Restarting takes time

Application specific recovery mechanisms

- Idea: Multi-process model, each client connection is new process, kill process if it fails
- Problems:
  - Still has issues with dealing with deterministic errors
  - If shared data is the problem, killing and restarting processes will not restore it to consistent state
Other methods

- Failure-oblivious computing
  - Idea: Provide artificial values for out-of-bound reads
- Reactive immune system
  - Idea: Creates emulators to run “faulty” regions of a program
  - Problems:
    - Considered by authors as “unsafe” because they mask behaviors and speculate as to what the program wants to achieve
    - Immune system has large overheads

Rx real-world metaphor

- Idea: Treat software bugs as real-world allergies
  - In real life allergens can be dealt with by changing living environment
    - Removing cat hair from area allows me to breathe better
    - Successfully removing allergen from environment allows one to determine cause of allergy
      - No cat hair = no sneezing → allergic to cats

Rx metaphor implemented

- Bugs resemble allergies
- Bugs can be dealt with by changing execution environment
  - When a bug is detected, rollback to checkpoint and alter execution environment to deal with detected issues
  - Least-intrusive changes can be tried first and more drastic changes can be implemented until a good execution environment is found

The Main Idea

Rx Architecture

- Dynamically monitor applications execution to determine software failures
- Sends information to control unit
- Two types of sensors
  - Sensor to monitor software errors (assertion failures, access violations)
  - Sensor to monitor software bugs (buffer overflows, access to freed memory)
Checkpoint and Rollback

- CR component takes a snapshot of application and stores it in main memory
- Stores memory and file states
- During rollbacks, all of these states can be re-implemented and the program can be continued from this previous checkpoint
- Multiple checkpoints can be stored in case Rx needs to rollback to an earlier checkpoint
- Keeps enough to be "2-competitive"

Execution Environment Changes

- Memory management-based
  - Addresses bugs that are memory-based such as buffer overflows, dangling pointers, etc.
  - Ex: Padding to prevent buffer overflows, zero-filling new buffers
  - Textual debug
  - Addresses bugs that are related to asynchronous events like data races
  - Ex: Increasing length of scheduling time slot can avoid context switches in buggy critical sections
- User request-based
  - Deals with the fact that it is impossible to test every possible user request
  - Ex: Dropping user requests during re-execution to deal with unexpected requests
- Interception-based
  - Intercepts memory-related library calls, adjusts according to what is known of failures

Environment Wrappers

- Perform environmental changes for application during re-execution
  - Memory wrapper
    - Intercepts memory-related library calls, adjusts according to what is known of failures
  - Message wrapper
    - Changes message delivery environment
    - Process scheduling
    - Changes processes' priority to deal with scheduling issues
    - Signal delivery
      - Keeps track of signals in order to control when they are sent
      - Dropping user requests
      - Drops requests that may be causing errors

Proxy

- Handles re-execution of requests, making crashes oblivious to clients
- In normal mode, the proxy simply relays messages between client and server, keeping track of them
- In recovery mode, handles three tasks:
  - Replays requests from client's last checkpoint
  - Implements message-related environmental changes
  - Buffers client requests until server has come back from software failure

Control Unit

- Controls the whole Rx system
  - Performs three functions:
    - Directs CR to rollback at software failures
    - Diagnoses failures based on "symptoms" and previous knowledge of failures
    - Provides information on failures for programmers
  - The control unit stores information on failures and what recoveries worked for future reference

Design and Implementation Issues

- Inter-server communication
  - Server communication is key so that multiple servers can be rolled back to achieve system stability
- Multi-threaded process checkpointing
  - Force all threads to be at user level to ensure accurate checkpointing due to threads running simultaneously
Evaluation

- Tested on 4 server applications (Apache httpd, MySQL, Squid, CVS)

<table>
<thead>
<tr>
<th>App</th>
<th>Ver</th>
<th>Bug</th>
<th>Leak</th>
<th>App Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>4.1.1a</td>
<td>data race</td>
<td>188K</td>
<td>a database server</td>
</tr>
<tr>
<td>Squid</td>
<td>2.3.35</td>
<td>traffic overload</td>
<td>99K</td>
<td>a Web proxy server</td>
</tr>
<tr>
<td>Squid</td>
<td>2.3.35</td>
<td>connection refused</td>
<td>100K</td>
<td>a Web proxy server</td>
</tr>
<tr>
<td>Apache</td>
<td>2.0.64</td>
<td>stuck-overwrite</td>
<td>235K</td>
<td>a Web server</td>
</tr>
<tr>
<td>CVS</td>
<td>1.1.14</td>
<td>double time</td>
<td>114K</td>
<td>a version control server</td>
</tr>
</tbody>
</table>

Throughput and Avg Response Time

<table>
<thead>
<tr>
<th>App</th>
<th>Thru</th>
<th>Avg Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>1000</td>
<td>0.5 ms</td>
</tr>
<tr>
<td>Squid</td>
<td>1500</td>
<td>1.2 ms</td>
</tr>
<tr>
<td>Apache</td>
<td>2000</td>
<td>2.0 ms</td>
</tr>
<tr>
<td>CVS</td>
<td>2500</td>
<td>2.7 ms</td>
</tr>
</tbody>
</table>

Overall Results

<table>
<thead>
<tr>
<th>App</th>
<th>Thru</th>
<th>Avg Recovery Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>1000</td>
<td>1.2 ms</td>
</tr>
<tr>
<td>Squid</td>
<td>1500</td>
<td>2.0 ms</td>
</tr>
<tr>
<td>Apache</td>
<td>2000</td>
<td>2.8 ms</td>
</tr>
<tr>
<td>CVS</td>
<td>2500</td>
<td>3.5 ms</td>
</tr>
</tbody>
</table>

Rx Advantages

- Comprehensive
- Can survive many common software defects
- Safe
- Does not change program, only environment it runs in
- Noninvasive
  - Few to no modifications required in software (no mods in any of the tested systems)
- Efficient
  - No rebooting (mostly) with little overhead
  - Learns from previous solutions
- Informative
  - Bugs are shown and details are given on the nature of the bug

Issues

- Unavoidable Bug/Failures
  - Accumulative memory leaks cannot be detected by Rx
  - Only solution is program restart
- Worst case scenario 2x time for normal restart
- Did not happen in any of the tests
Questions/Complaints?

What do they mean with “execution environment”?

- “almost everything that is external to the target application but can affect the execution of the target application”
- 3 levels:
  - Lowest: Hardware (processor, devices)
  - Middle: OS kernel (scheduling, virtual memory management, device drivers)
  - Highest: libraries (standard, third-party)

Throughput and Avg Response Time

Avg Space Overhead per Checkpoint

Different bug arrival rates