Record and Replay

Xiao Guangrong
<guangrong.xiao@linux.intel.com>
Agenda

• Introduction
• Challenges
• Solutions
• QEMU record & replay
• Cicada?
Introduction

• It allows capturing the execution of a running system for later replay
• Good for bug analysis
• Good for malware analysis
• Good for post-attack analysis
• Etc.
Challenges

• Modern CPUs are not deterministic
  • Out-of-order execution
  • Previous status based speculate algorithm
  • Unpredictable Cache miss
  • So, choose a probe checkpoint
  • Depends on Interrupt to flush its pipe line
  • Memory-chunk based

• Accurate time scale
  • Higher resolution than instruction execution
  • Interrupt based time-keeping is not suitable for Record & Replay.
  • Hardware performance counter (e.g., instructions count)

• Non-deterministic event record
  • Interrupt
  • Context-switch
  • Race conditions
  • Etc. ......
Solutions: Single process based record & Replay

• A example: Jockey: A user-space library for record-replay debugging
  • it intercepts system calls and rewrites all CPU offending instructions
    • To intercept system call
    • To rewrites CPU instructions, e.g., rdtsc
      1. Fetch text section by reading ELF header
      2. Scan text section and rewrite the inst.
• Advantage: it is simple enough
• Disadvantage
  • Single application only
  • Event based rather than inst. based
Solutions: Single process based record & Replay (Cont.)

• The methods to solve the challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>The way to solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-deterministic CPU</td>
<td>Memory barrier can be applied in the handler</td>
</tr>
<tr>
<td>Time scale</td>
<td>Event based</td>
</tr>
<tr>
<td>Non-deterministic event record</td>
<td>None is recorded</td>
</tr>
</tbody>
</table>
Solutions: multi-threaded process based Record & Replay

• A example: Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging
• Make checkpoints for the whole process status where it can be replayed later
• Record memory status – by forking a new process
• Log system calls – by intercepting system calls in kernel
• Handle shared memory by forcing page-fault
• Record signals
• Advantage: multi-thread workable
• Disadvantage: require kernel change, can not record context switch (can not replay race conditions)
Solutions: multi-threaded process based Record & Replay

• The methods to solve the challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>The way to solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-deterministic CPU</td>
<td>Memory barrier can be applied in the handler</td>
</tr>
<tr>
<td>Time scale</td>
<td>Event based</td>
</tr>
<tr>
<td>Non-deterministic event record</td>
<td>None is recorded</td>
</tr>
</tbody>
</table>
Solutions: Virtual Machine based Record & Replay

• Checkpoint system status
• Replay storage
• Log non-deterministic events
• Advantage: replay for the whole system
• Disadvantages:
  • 1) Over configure (e.g., dedicated VM images)
  • 2) Overload (e.g., log everything)
Solutions: Virtual Machine based Record & Replay (Cont.)

- The methods to solve the challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>The way to solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-deterministic CPU</td>
<td>Based on interrupt which can flush CPU pipe lines automatically</td>
</tr>
<tr>
<td>Time scale</td>
<td>Instruction count (e.g, instruction-count hardware performance count)</td>
</tr>
<tr>
<td>Non-deterministic event record</td>
<td>Record and associate it with the executed instruction sequence</td>
</tr>
</tbody>
</table>
Solutions: hardware-based Record & Replay

• Log the execution chunk which is not collisional with other CPUs and associated with a sequence number and CPU ID, consequently, these chunks can be replayed.
  • Instruction & memory reorder are allowed in the chunk
  • External interrupt can truncate the chunk
• Can be based on hardware transaction memory
• Advantage: full record & replay support
• Disadvantage: expensive for dedicated hardware and no real CPU supports it so far (cicada?)
Solutions: hardware-based Record & Replay (Cont.)

- Architecture
Solutions: hardware-based Record & Replay (Cont.)

• Work flow
Solutions: hardware-based Record & Replay (Cont.)

• The methods to solve the challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>The way to solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-deterministic CPU</td>
<td>Hardware guarantee (e.g., flush CPU pipe line at the boundary of chunk)</td>
</tr>
<tr>
<td>Time scale</td>
<td>Based on the chunk size which is predefined or recorded by the hardware (CS-LOG)</td>
</tr>
<tr>
<td>Non-deterministic event record</td>
<td>Hardware record them at the boundary of chunk</td>
</tr>
</tbody>
</table>
QEMU Record & Replay

- Only works on !KVM & !Xen, e.g., no hardware-based virtualization (there has some researches for KVM based RR)

```
Virtual machine ➔ QEMU ➔ Log file ➔ QEMU

Log recording phase

Log replaying phase

Results of debugging, trace capturing, taint analysis, and so on

Analysis results

Contains OS and program for analysis

Non-deterministic events in VM are saved into the external file

Offline analysis tool
```
QEMU Record & Replay (Cont.)

• Use ‘icount’ (instruction count) as its time scale

```assembly
check tcg_exit_req
000f2e0e: push %ebx
++icount
000f2e0f: sub $0x2c,%esp
++icount
000f2e12: movl $0xf64bc,0x4(%esp)
++icount
000f2e1a: movl $0xf4d50,(%esp)
++icount
000f2e21: call 0xf1ca0
++icount
```

I’m going to create my own icount with black jack and hookers.
QEMU Record & Replay (Cont.)

• ‘icount’ limitations
  • Only supports single vCPU
  • Different counting for REP instructions in single step and normal modes
    – icount is not incremented for the last (ecx=0) iteration in normal mode
  • Incorrect when using breakpoints through gdb
    – icount is incremented for non-executed instruction, which located at the breakpoint address

Hmm, no memory barrier is used to avoid instruction reorder?
QEMU Record & Replay (Cont.)

• Non-deterministic events
  • Only save non-deterministic to make log file smaller
  • Clock
  • The input of from peripheral devices, e.g.
    • Networking packets
    • Mouse & keyboard input
    • Etc

• Deterministic event
  • Not logged
  • It includes
    • Memory
    • Software interrupt
    • Instruction execution
QEMU Record & Replay (Cont.)

• Checkpoint
  • Replaying of the execution of virtual machine is bound by sources of non-determinism.

• Block device replay
  • It is inserted between disk image and virtual driver controller. Therefore all disk requests may be recorded and replayed.

• Networking replay
  • Qemu puts all packets coming from the outer world into a log. In replay mode packets from the log are injected into the network device.
Cicada?
Q & A?
References

• Jockey: A user-space library for record-replay debugging

• Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging

• Deterministic Replay of System’s Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging
  (https://www.computer.org/csdl/proceedings/csmr/2012/4666/00/4666a553.pdf)

• Samsara: Efficient Deterministic Replay with Hardware Virtualization Extensions

• DeLorean: Recording and Deterministically Replaying Shared-Memory Multiprocessor Execution Efficiently
  (http://iacoma.cs.uiuc.edu/iacoma-papers/isca08_rep.pdf)

• Deterministic Replay and Reverse Debugging for QEMU
  (http://www.linux-kvm.org/images/d/d0/02x06b-DeterministicReplay.pdf)