Distributed Real-Time Fault Tolerance on a Virtualized Multi-Core System

Eric Missimer*, Richard West and Ye Li

*Computer Science Department
Boston University
Boston, MA 02215
Email: {missimer, richwest, liye}@cs.bu.edu
*VMware, Inc.
Quest-V Background:
- Boston University’s in house operating system + hypervisor
- Developed for real-time and high-confidence systems

Key Features:
- Virtualized Separation Kernel
- Simplified Hypervisor:
  - **Sandboxes** are pinned to cores at boot, no need for scheduling
  - I/O devices are partitioned amongst sandboxes, not shared or emulated
  - Virtualization used for **encapsulation**
- Assume hypervisor is a trusted code base
- Communication through explicit shared memory channels
Motivation

- Safety critical systems require component isolation and redundancy
  - Integrated Modular Avionics (IMA), Automobiles
- Multi-/many-core processors are increasingly popular in embedded systems
- Multi-core processors can be used to consolidate redundant services onto a single platform
Motivation

- Many processors now feature hardware virtualization
  - ARM Cortex A15, Intel VT-x, AMD-V
- Hardware virtualization provides opportunity to efficiently partition resources amongst guest VMs
- Not trying to remove all hardware redundancy – just lessen it
Many processors now feature hardware virtualization
   ARM Cortex A15, Intel VT-x, AMD-V
Hardware virtualization provides opportunity to efficiently partition resources amongst guest VMs
Not trying to remove all hardware redundancy – just lessen it

Motivation

- Focusing on hardware transient faults and software timing faults
  - Random bit flips from caused by radiation
  - Asynchronous bugs in faulty device drivers
N redundant copies of a program, one per sandbox (at least three)
At least one voter
Hash based fault detection and recovery
Virtualized separation kernel platform provides new n-modular redundancy configurations
Software based dual core lock step (DCLS)
N-Modular Redundancy

Sandbox 1  Sandbox 2  ...  Sandbox N

Redundant Application Instances

Application Results/Voter Input

Voter

Voter Output (I/O Device Input)
N-Modular Redundancy for Real-Time Applications

Read Sensor Input

Perform Computation

Send Result and Hashes To Voter

Hash Memory

Sensor Data

Sync Operation
Fault Detection

- Typical n-modular redundancy compares the output of the computation
  - Pro: Fast
  - Con: Don’t know what went wrong
- Proposed detection method: compare application memory on a per page basis via hashes
  - Pro: Faster and generic recovery for complicated applications (discussed later)
  - Con: Must hash memory state of process (slow)
  - Can speed on comparison using a “summary” hash
Fault Detection

Sandbox 1
Summary
Page 1
Page 2

Sandbox 2
Summary
Page 1
Page 2

Sandbox 3
Summary
Page 1
Page 2

Voter
N-Modular Redundancy Configurations

- Voting mechanism and device driver in the hypervisor
- Voting mechanism and device driver in one sandbox
- Voting mechanism distributed across sandboxes and device driver is shared
Voting Mechanism and Device Driver in the Hypervisor

- **Guest**
  - Emulated Device

- **Hypervisor**
  - Voter
  - Device Driver
  - Physical Device
Voting Mechanism and Device Driver in the Hypervisor

Pros:
- No need to modify operating system - could apply to Linux as well as Quest
- Need only \( n \) sandboxes

Cons:
- Conflicts with Quest-V hypervisor design
- Faulty device driver could jeopardize the entire system
- Need to duplicate the entire guest
Voting Mechanism and Device Driver in One Sandbox

Private Communication Channels

Arbitrator Sandbox  Redundant Sandbox  Redundant Sandbox  Redundant Sandbox

Voter

Device Driver

Physical Device

Hypervisor
Voting Mechanism and Device Driver in One Sandbox

Pros:
- Simpler hypervisor
- Application level redundancy, don’t need to copy the entire sandbox

Cons:
- Need $(n+1)$ sandboxes
- Need to modify guest

Eric Missimer, Richard West and Ye Li
Voting is Distributed and Device Driver is Shared

Guest

Private Communication Channels

Voter

Device Driver

Guest

Voter

Device Driver

Guest

Voter

Device Driver

Shared Physical Device

Hypervisor
Voting is Distributed and Device Driver is Shared

Pros:

- Need only $n$ sandboxes
- Application level redundancy, don’t need to copy the entire sandbox

Cons:

- Need to modify guest
- Complicated shared device driver
Recovery

- Want recovery to be as generic as possible
- Simple applications – rebooting might be sufficient
- Complicated applications – rebooting could cause important state to be lost
- Perform live migrations of either application or guest machine
All performed within the context of the thread’s sporadic server
Quick Summary - Key Points to Take Away

- Per-page hash based fault detection and recovery
- Three n-modular redundancy configurations in a virtualized separation kernel

Hypervisor Voting

Sandbox Voting

Distributed Voting
Conclusion

So what’s left?
Conclusion

So what’s left?

Further implementation and comparison
Conclusion

So what’s left?

Further implementation and comparison

Figure out solution for voter single point of failure:
Possibilities include arithmetic encoding and memory scrubbing
More Info: www.questos.org
Conclusion

- More Info: www.questos.org
- Questions?