Hypervisor-Based Fault-Tolerance

THOMAS C. BRESSOUD
Isis Distributed Systems
and
FRED B. SCHNEIDER
Cornell University


Presented by Yeh Tsung-Yu
Outline

- Introduction
- Replica Coordination Protocol
- Performance
- Conclusion and future work
Introduction

- We propose a software layer between the hardware and the operating system - Hypervisor.

- Our fault-tolerant computing system does not require modifications of hardware, operating system, or any application software.
Introduction

- What is Hypervisor?
  - *Virtual machines which* having the same instruction-set architecture as the hardware on which the hypervisor executes.

- We run protected VM in primary physical host and backup VM in backup physical host.
  - We keep these VMs sync in order for app (run in VM) to survive processor failure.
Introduction

- When primary hardware fails, the VM in backup hardware will take over as soon as possible.
Replica Coordination Protocol

- Idealistically, we hope primary VM is state machine, and every instruction is deterministic.

- Since we could make backup VM read the same sequence of instructions, in order to make backup VM reach the same state.

- But there is still some non-deterministic instruction to be made by VM.
  - See next slide.
Replica Coordination Protocol

- **Deterministic instruction:**
  - E.g. ADD, DIV.
  - As the same argument is given, the same result is produced.

- **There is still some non-deterministic choice to make:**
  - E.g. reading the time-of-day clock.
  - E.g. VM’s interrupt.
Replica Coordination Protocol

- **Our generalized assumption 1:**
  - Environment Instruction Assumption: VM is invoked to simulate when E.I is going to be executed.
  - What is Environment Instruction?
    - E.g. reading time-of-day clock, reading disk block.
  - Actually E.I is only executed in primary, then result is transferred to backup.
Replica Coordination Protocol

- Our generalized assumption 2:
  - Instruction Stream Interrupt Assumption:
    - A mechanism is available to invoke the VM when a specified point in the instruction stream is reached.
  
  - We could support this assumption by recovery register (HP’s PA-RISC) which decrement each time an instruction done, and cause interrupt as content is zero.
Replica Coordination Protocol

- Our generalized assumption 2:
  - Recovery register is to separate instruction stream into epochs.
  - In every epoch, primary VM buffer interrupts, and forward these to backup VM in the epoch’s end.
  - Interrupts at backup VM are ignored.
Replica Coordination Protocol

- **Scenario (primary):**
  - P0: if primary VM execute Env. Instruction at pc
    - Send $[E(p), pc, Val]$ to backup and wait for ack.
  - P1: if primary VM receives a interrupt
    - Buffer INT for delivery later.
  - P2: if primary’s epoch ends
    - Primary send to backup all buffered INT during $E(p)$ and wait for ack.
    - Primary delivers all INT.
    - $E(p) = E(p) + 1$, and primary start epoch $E(p)$.  


Replica Coordination Protocol

- **Scenario (backup):**
  - P3: if backup VM execute Env. Instruction at pc
    - Wait receipt of \([E(b), pc, Val]\) from primary.
    - If \([E(b), pc, Val]\) is received, then ack primary.
  - P4: if backup VM receives a interrupt
    - It’s ignored.
  - P5: if epoch ends
    - Wait for all buffered INT from primary, if received, then ack primary.
    - Backup VM delivers all INT.
    - \(E(b) = E(b) + 1\), and backup start epoch \(E(b)\).
Replica Coordination Protocol

- If failure is detected (message timeout), backup VM execute Env. Instruction as if it’s primary.
  - In next epoch, backup is promoted to primary.

- Unavoidably, INT might be lost when primary fails transferring INT to backup. (discussed later)
Replica Coordination Protocol

- Interaction with the environment
  - I/O instructions executed by a backup are absorbed by backup’s VM.
  - Clock syn : at the end of epoch.
    - Make newly promoted primary are consistent with the failed primary.
Replica Coordination Protocol

- Interaction with the environment (const.)
  - backup VM can tolerate not receiving interrupts buffered by the primary VM.
    - This newly promoted primary simply delivers “uncertain interrupts” for outstanding I/O operations.
  - For disks and networks, driver will reissue its last I/O instruction upon receiving an uncertain interrupt.
    - the state of a disk is insensitive to repetitions of I/O operation.
    - network protocols themselves send and ignore duplicate messages.
### Protocol Example

#### Primary VM

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read clock</td>
</tr>
<tr>
<td>Request Read disk block A</td>
</tr>
<tr>
<td>Request Write disk block A, y</td>
</tr>
<tr>
<td>Request Write disk block A, x</td>
</tr>
<tr>
<td>ADD r0,r1,r2</td>
</tr>
<tr>
<td>ADD r3,r0,r2</td>
</tr>
<tr>
<td>DIV r3,r0,r2</td>
</tr>
<tr>
<td>Read clock</td>
</tr>
</tbody>
</table>

#### Backup VM

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read clock</td>
</tr>
<tr>
<td>Request Read disk block A</td>
</tr>
<tr>
<td>Request Write disk block A, y</td>
</tr>
<tr>
<td>Request Write disk block A, x</td>
</tr>
<tr>
<td>ADD r0,r1,r2</td>
</tr>
<tr>
<td>ADD r3,r0,r2</td>
</tr>
<tr>
<td>DIV r3,r0,r2</td>
</tr>
<tr>
<td>Read clock</td>
</tr>
</tbody>
</table>

- **Compute and transfer**
- **Ack**
- **Forward all buffered INT and accompany data**
- **End of epoch**
- **Absorbed by VM**
Protocol Example, Const

Primary VM ISR (read request INT)

Read disk block A

....

Backup VM ISR (read request INT)

Read disk block A

....
Fig. 2. CPU-intensive workload.
Performance

Fig. 3. Input/output options.
Performance

Predicted Performance

Normalized Performance

Epoch Length (EL)

Fig. 4. Faster communication.
Conclusion and future work

- VM is not the only way to use our approach, for example, one might modify microkernel.

- When time-to-market and cost is sensitive, our design is easier than the hardware-design (e.g., HP’s NonStop).