Remus: High Availability via Asynchronous Virtual Machine Replication

Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield

Department of Computer Science
The University of British Columbia


Presented by Yeh Tsung-Yu
Outline

- Introduction
- Design and Implementation
- Evaluation
- Conclusion and future work
Introduction

- It’s hard and expensive to design highly available system to survive hardware failure
  - Using redundant component.
  - Customized hardware and software.
  - E.g. HP Nonstop Server.

- Our goal is to provide high availability system, and it’s:
  - Generality and transparency
    - Without modification of OS and App.
  - Seamless hardware failure recovery
Introduction

- **Our approach**
  - VM-based whole system replication
    - Frequently checkpoint whole Virtual Machine state.
    - Protected VM and Backup VM is located in different Physical host.
  - Speculative execution
    - We buffer state to syn backup later, and continue execution ahead of syn point.
  - Asynchronous replication
    - Primary VM execution is overlap state transmission
Figure 1: Speculative execution and asynchronous replication in Remus.
Design and Implementation

Our implementation is based on:

- Xen virtual machine monitor.
- Xen’s support for live migration to provide fine-grained checkpoints.
- Two host machines are connected over redundant gigabit Ethernet connections.

The virtual machine does not actually execute on the backup host until a failure occurs.
Design and Implementation

Figure 2: Remus: High-Level Architecture
Pipelined checkpoint (see figure 1,p-5)

1. Pause the running VM and copy any changed state into a buffer. (talk later more detail)

2. With state changes preserved in a buffer, VM is unpaused and speculative execution resumes.

3. Buffered state is transmitted to the backup host.

4. When complete state has been received, ack to the primary.

5. Finally, buffered network output is released. (talk later more detail)
Design and Implementation

- Next we talk about how to checkpoint CPU state, memory, disk, network.

- CPU & memory state
  - Checkpointing is implemented above Xen’s existing machinery for performing live migration.

- What is live migration?
  - A technique by which a VM is relocated to another physical host with slight interruption.
How live migration work?

1. Memory is copied to the new location while the VM continues to run at the old location.

2. During migration, writes to memory are intercepted, and dirty pages are copied to the new location in rounds.

3. After a specified number of intervals, the guest is suspended and the remaining dirty page and CPU state is copied out. (final round)
Design and Implementation

- By hardware MMU, page protection is used to trap dirty page.

- Actually, Remus implements checkpointing as repeated executions of the final round of live migration.
Design and Implementation

- Disk buffering
  - Requirement
    - All writes to disk in VM is configured to write though.
    - As a result, we can achieve crash-consistent when active and backup host both fail.
  - On-disk state don’t change until the entire checkpoint has been received, see figure4 in p13.
Design and Implementation

Figure 4: Disk write buffering in Remus.
Network buffering

– Most networks cannot be counted on for reliable data delivery.

– Therefore, networked app use reliable protocol as TCP to deal with packet loss or duplication.

  - This fact simplifies the network buffering problem considerably: transmitted packets do not require replication
Design and Implementation

- In order to ensure packet transition atomic and checkpoint consistency:
  - outbound packets generated since the previous checkpoint are queued.
  - Until that checkpoint has been acknowledged by the backup site, outbound packets is released.
Design and Implementation

- Detecting Failure

  - We use a simple failure detector that is directly integrated in the checkpointing stream:
    - a timeout of the backup responding to commit requests.
    - a timeout of new checkpoints being transmitted from the primary.
  - Timeout event represents the host’s failure.
Figure 5: Checkpoint time relative to pages dirtied.

Figure 9: The effect of disk replication of Postmark performance.
Evaluation

Figure 6: Kernel build time by checkpoint frequency.

Figure 7: SPECweb scores by checkpoint frequency (native score: 305)
Conclusion and future work

- We develop a novel method to provide high availability to survive hardware failure, and it have low cost and transparency.

- But outbound packet latency lower network throughput and still is waited for future work to solve.