Live Migration of Direct-Access Devices

Asim Kadav and Michael M. Swift
University of Wisconsin - Madison

Live Migration

• Migrating VM across different hosts without noticeable downtime
• Uses of Live Migration
  – Reducing energy consumption by hardware consolidation
  – Perform non-disruptive hardware maintenance
• Relies on hypervisor mediation to maintain connections to I/O devices
  – Shared storage
  – Virtual NICs
Live Migration with virtual I/O

Source Host

Guest OS

Virtual Driver

Hypervisor + OS

Hardware Device

Destination Host

Device state available to hypervisor.

VMM abstracts the hardware

Hypervisor + OS

Hardware Device

Access to I/O Devices

Virtual I/O

- Guest accesses virtual devices
- Drivers run outside guest OS
- Hypervisor mediates all I/O

- Moderate performance
- Device independence
- Device sharing
- Enables migration
Direct access to I/O Devices

Direct I/O
• Drivers run in Guest OS
• Guest directly accesses device
  ▪ Near native performance
  ▪ No migration

Direct I/O Architecture
(Pass-through I/O)

Live Migration with Direct I/O

Source Host
Destination Host

Device State lost
No Heterogeneous Devices
Live migration with Direct I/O

• Why not both performance and migration?
  – Hypervisor unaware of device state
  – Heterogeneous devices/drivers at source and destination

• Existing Solutions:
  – Detach device interface and perform migration [Xen 3.3]
  – Detach device and divert traffic to virtual I/O [Zhai OLS 08]
  – Modify driver and device [Varley (Intel) ISSOO3 08]

Overview

• Problem
  – Direct I/O provides native throughput
  – Live migration with direct I/O is broken

• Solution
  – Shadow drivers in guest OS capture device/driver state
  – Transparently re-attach driver after migration

• Benefits
  – Requires no modifications to the driver or the device
  – Supports migration of/to different devices
  – Causes minimal performance overhead
Outline

- Introduction
- Architecture
- Implementation
- Evaluation
- Conclusions

Architecture

- Goals for Live Migration
  - Low performance cost when not migrating
  - Minimal downtime during migration
  - No activity executing in guest pre-migration

- Our Solution
  - Introduce agent in guest OS to manage migration
  - Leverage shadow drivers as the agent [Swift OSDI04]
Shadow Drivers

- Kernel agent that monitors the state of the driver
- Recovers from driver failures
- Driver independent
- One implementation per device type

Shadow Driver Operation

- Normal Operation
  - Intercept Calls
  - Track shared objects
  - Log state changing operations
- Recovery
  - Proxy to kernel
  - Release old objects
  - Restart driver
  - Replay log
Shadow Drivers for Migration

- **Pre Migration**
  - Record driver/device state in driver-independent way
  - Shadow driver logs state changing operations
    - configuration requests, outstanding packets
- **Post Migration**
  - Unload old driver
  - Start new driver
  - Replay log to configure driver

Shadow Drivers for Migration

- **Transparency**
  - Taps route all I/O requests to the shadow driver
  - Shadow driver can give an illusion that the device is up

- **State Preservation**
  - Always store only the absolute current state
  - No history of changes maintained
  - Log size only dependent on current state of the driver
Migration with shadow drivers

Outline

- Introduction
- Overview
- Architecture
  - Implementation
- Evaluation
- Conclusions
Implementation

- Prototype Implementation
  - VMM: Xen 3.2 hypervisor
  - Guest VM based on linux-2.6.18.8-xen kernel
- New code: Shadow Driver implementation inside guest OS
- Changed code: Xen hypervisor to enable migration
- Unchanged code: Device drivers

Changes to Xen Hypervisor

- Migration code modified to allow
  - Allow migration of PCI devices
  - Unmap I/O memory mapped at the source
  - Detach virtual PCI bus just before VM suspension at source and reconnect virtual PCI bus at the destination
  - Added ability to migrate between different devices
Modifications to the Guest OS

• Ported shadow drivers to 2.6.18.8-xen kernel
  – Taps
  – Object tracker
  – Log
• Implemented shadow driver for network devices
  – Proxy by temporarily disabling device
  – Log ioctl calls, multicast address
  – Recovery code

Outline

• Introduction
• Architecture
• Implementation
• Evaluation
• Conclusions
Evaluation

1. Cost when not migrating
2. Latency of Migration

Evaluation Platform

• Host machines
  – 2.2GHz AMD machines
  – 1 GB Memory
• Direct Access Devices
  – Intel Pro/1000 gigabit Ethernet NIC
  – NVIDIA MCP55 Pro gigabit NIC
• Tests
  – Netperf on local network
  – Migration with no applications inside VM
  – Liveness tests from a third physical host
Throughput

Throughput in Mbits/second

Direct I/O

Virtual I/O

Direct I/O with shadow throughput within 1% of original

CPU Utilization

CPU Utilization in the guest VM

Direct I/O with shadow CPU utilization within 1% of original
Migration Time

Significant migration latency (56%) is in driver uptime.

Conclusions

- Shadow Drivers used as an agent to perform live migration of VMs performing direct access
- Supports heterogeneous devices
- Requires no driver or hardware changes
- Minimal performance overhead and latency during migration
- Portable to other devices, OS and hypervisors
Questions

Contact: {kadav, swift} @cs.wisc.edu

More details:

http://cs.wisc.edu/~swift/drivers/

http://cs.wisc.edu/~kadav/

Backup Slides
Complexity of Implementation

- ~19000 LOCs
- Bulk of this (~70%) are wrappers around functions.
  - Can be automatically generated by scripts

Migration Time

Total downtime = 3.97 seconds

Significant migration latency (56%) is in driver uptime.
Migration with shadow drivers

Tracker

Log

Guest OS Kernel

Shadow Driver

Destination Hypervisor + OS

Source Network Driver

Dest. Network Driver

Source Network Card

Destination Network Card

Taps

Source Hypervisor + OS

Destination Hypervisor + OS