Bootstrapped Migration for Linux OS (BOMLO)

Jui-Hao Chiang,
Maohua Lu, Tzi-cker Chiueh
Stony Brook University, NY, USA
Jun. 2011 at ICAC
Outline

- Motivation
- Challenge
- Tuxonice
- BOMLO
- Performance and Future work
Motivation - Problem Statement

- Virtual Machine State Migration (\textbf{VMSM}) is an essential management primitive in modern data centers for
  - Resource Consolidation
  - Software/hardware maintenance
- But Virtualization is not universally advocated due to its
  - Sub-optimal performance and
  - Reliability issues
- What to do with machines without Virtualization?
  - BOMLO
Motivation - Problem of Virtualization

- **Sub-optimal performance**
  - VM (Virtual Machine) cannot fully leverage the capability of physical hardware
  - Some applications choose to run on bare metal
    - I/O-intensive applications, e.g., DBMS
    - Applications with high hardware demanding, e.g. Microsoft SharePoint
  - Shadow-driver research project supports migration for VMs with non-virtualized I/O devices (direct I/O is desirable)

- **Reliability issues**
  - Single point failure of physical machine (PM) fails all VMs on it

- No migration facility for architecture without virtualization support
  - ARM-based server is desirable for data center because of its low-power characteristic
Motivation - Move At Once

- BOMLO-like technology can enable the hypervisor to move all VMs on a PM to another PM in one shot
  - KVM is a built-in linux kernel component for hosting VMs as normal processes/threads
  - BOMLO sits directly on top of bare metal so that it can migrate the states of entire KVM plus all its running VMs
Motivation - Commercial Trend

- BOMLO is a desired feature in commercial product
  - Novell PlateSpin® Migrate
    - Supports offline physical machine migration for Linux
    - Offline data copying and deploying from source to destination machine
    - Requires application-level knowledge and configuration
  - Double-Take® MOVE
  - Vizioncore® vConverter
Challenge

- **VM migration v.s. BOMLO**
  - Done by hypervisor v.s. done by kernel itself
  - Hypervisor is not part of the states for migration
- Devices detach and attach (Hardware abstraction)
- Reduce service disruption (Live migration)
  - Tracking changed/dirtied memory states on source machine
VMSM

Before migration

Detach device on source;
Tracking memory changes;
Iteratively copy memory state;
Attach devices on destination

After migration
Tuxonice - Linux Suspend and Resume

- Suspend requires "Freeze"
  - Processes
  - Devices
- Resume requires "Thaw"
  - Reverse operation of Freeze
BOMLO

- Swap disk-based migration (Suspend)
Swap disk-based migration (Resume)
Memory-to-Memory Migration

- **Direct** memory copy without any temporary space
Memory-to-Memory Migration (cont.)

Before allocation

After allocation

After transmission

M1: Source Machine
M2: Destination Machine

Pages to be copied from M1
Occupied by M2's Boot Kernel
Allocated memory on M2

atomic-restore
Iterative Memory-To-Memory Migration

- Similar to Live Migration in VMSM
- Problem: while saving states to destination machine (e.g., 3 sec. for 250MB), all processes are suspended
  - Applications can no longer serve user's requests
- Desire: continuously run applications when transferring states
- Proposed Solution
  - Mark all pages as read-only
  - Intercept page write and put to a list, `dirty_list`
  - Send all memory in `dirt_list` to destination machine iteratively
- For kernel important data structures, e.g. page cache, we use `checksum` method to detect dirtiness
Performance and Future Work

● Migrate between two identical physical machine
  ○ Apache+AB, NFS+SPEC SFS, and MySQL+TPC-C
  ○ Service disruption time of benchmark is around 6 seconds

● Hardware Heterogeneity
  ○ CPU flag masking and hotplug support in Linux
  ○ Memory hotplug support in Linux
  ○ I/O devices: Shadow driver intercepts the interactions between native device drivers and the kernel's I/O subsystems
Related Work

● VMSM techniques
  ○ Xen live migration
  ○ VMware VMotion
  ○ NomadBIOS built on top of the L4 micro-kernel
  ○ Self-migration built on top Xen

● Migration without virtualization [Kozuch et al.]
  ○ Only idea and rough design, but not detailed
How does Tuxonice **save** snapshot to swap disk?
How does Tuxonice **restore** snapshot from swap disk?
Tuxonice (cont.)

- **Suspend** in Tuxonice
  - Freeze all user process
  - Sync and freeze file system
  - Freeze kernel threads
  - Copy non-essential memory to disk and free these memory pages for atomic-save
  - Suspend device
  - Save essential pages (including CPU and device states) to some preserved memory area (**atomic-save**)
  - Resume device and copy the above to disk
  - Power down device and machine
Snapshot Categorization in Tuxonice

- **Non-essential pages (PNE)**
  - Memory pages belong to user/kernel processes
  - Page cache
- **Essential pages (PE)**
  - All used memory pages (from memory allocator) except non-essential pages

Why Categorization?
- **atomic-save** requires free memory as large as snapshot
- PNE can be saved to disk earlier before atomic-save (after device suspend), and the memory can be freed.

Why atomic-restore doesn't care?
- When resume, the bootstrapped Linux Kernel is very small (no user-level applications running)
Tuxonice (cont.)

- Resume in Tuxonice
  - When system boot, kernel try to resume from swap disk just before user-level start scripts
  - Load resume image to temporarily allocated memory
  - Freeze all processes; suspend devices, and save CPU context (in case resume fails)
  - Atomic-restore
    - Copy essential and non-essential pages to original memory location
    - Restore CPU context and the old kernel starts
  - Resume devices; thaw processes to run
How does BOMLO **restore** snapshot from swap disk (on destination machine)?