Live Migration
KVM
Containers
Live Migration

- Move a VM from one machine to another without only a small loss of execution time
Naive Live Migration

- Pause execution
- Take snapshot, record
  - Registers
  - Memory
  - Device State, Disk
- Move snapshot
  - Memory
  - Disk
- Restore snapshot
- Restart execution
Naive Live Migration Speed

• **Size**
  
  Memory == 4 GB  
  Disk == 100 GBs

• **Network**
  
  1 Gb/s Dedicated Network

• **Downtime**
  
  \[(104\ \text{Gb})\ ÷ 1\ \text{Gb/s} ≈ 15\ \text{minutes}\]
Live Migration - Storage Architecture

Live Migration Requires Shared Storage
Live Migration - Memory

- Iterative Approach
  - Mark all memory invalid
  - Run VM on source while copying page to destination
  - VM will "fault-in" pages
    - These pages will need to be recopied
  - Once all memory has been copied
    - If remaining memory is small enough
      - Snapshot and copy remaining VM
    - else
      - GOTO "Mark"
Live Migration  – Memory

- Hopefully converges
  - Based on memory footprint
  - What is too many pages get modified during copy?
    - Tolerate longer downtime
    - Don't move VM

Other Solutions?

Only transfer diffs to pages already sent

Send cold pages first, then warm, finally hot
Storage VMotion
Deterministic Execution

**Deterministic execution**

- Same results each time
- Execution path is identical

- **Lock-Step**
  - Two or more runs
  - Follow same execution path
  - Stay close in execution timing
    - Synchronized
Sources Non-Determinism

Initial State
- Memory
- Disk

- Externally supplied data

- Interrupts
  - Timer
  - I/O

- Multiple-processor memory interleaving

- Exceptions?
Determinism in Single Threaded Process

Execution must take the same path on control flow

– Control flow is dependent on machine state
– Started in same state
– Run deterministically until this point
– Control flow will be deterministic

• So... Initial state must be the same
  – Either start from scratch or checkpoint

• Stop at precise point
  – Count instructions
How to Count Instructions

• **Binary Translation**
  – Instrument code to count each basic block
  – Counter per basic block
  – Multiply by block length
  – SLOW!

• **Hardware Performance Counters**
  – Count instruction, branches, etc
  – Need to interrupt at certain count
  – Counters must be precise
  – Often not available, complete, or precise
Determinism with System Calls

• System Calls are a form a External Data

• Some syscalls can be handled with Initial Data
  – Time, PID, etc can be checkpointed
  – Need to intercept calls are return checkpointed values
  – Files can be copied so process sees initial state

• Some syscalls may not be allowed
  – Ex: sockets are a form of multi-threaded execution

• What other factors can influence execution?
Determinism in System Virtualization

• **I/O Data must be same**
  – General approach
    • Record all data input into virtual machine in master run
    • Replay data
  – DMA must be handled specially
    • Generally made atomic

• **Interrupts must occur at precise points**
  – I/O interrupts must be based on master
  – Record instruction count for all interrupts
  – Replay interrupt at precise cycle counts
Lock Step and Fault Tolerance

• **Two VMs start off same initial state**
  – One is master
  – Other is slave

• **Master runs as normal**
  – All interrupt timings, data inputs are shipped to slave
  – Slave replays execution upto next event

• **When master dies**
  – Slave can take over without any downtime
  – Transactions continue
  – Just need to worry about network routing
Multi-Threaded Determinism

- Messages already handled interrupt delivery
- Memory is the Problem

```c
/* process 1 */
count = 0;
flag = 0;

while (!flag) {
   count++;
}
printf("%d\n", count);

/* process 2 */
while (!count) {
   
}
flag = 1;
```
Possible Solution

• Manage all memory sharing
  – Writeable pages are only given exclusively to one process
  – When other process touches page
    • Take fault
    • Steal page from owning process
    • Record time
    • Give page to faulting process
  – On replay force page steal at given time
    • Even if other process doesn't request it yet
KVM
KVM

KVM – the Kernel-based Virtual Machine – is a Linux kernel module that turns Linux into a hypervisor
Requires hardware virtualization extensions

Reuse Linux code as much as possible
Focus on virtualization, leave other things to respective developers
Integrate well into existing infrastructure, codebase, and mindset
VMware View (full vert)

- Console VM
- User VM
- User VM
- User VM

Hypervisor

Driver

Hardware
Xen

- Domain 0
- User VM
- User VM
- User VM

Driver

Hypervisor

Hardware

Driver
KVM View

Ordinary Linux
Process

User VM

User VM

User VM

Modules

Driver

Driver

Driver

Linux

KVM

Hardware
KVM Execution Modes

- 3 execution modes
  - User, Kernel, Guest
- Virtual CPU is just a linux thread
- Linux scheduler deals with it as normal
more details

- guest executes native + trap&emulate (h/w virt)
- Secure and fast actions always done in kernel
  - e.g. mode transitions, shadow MMU
  - pre-emption hooks, notifiers, …
- I/O emulation handled in userspace, e.g. qemu
- Guest physical just part of host virtual memory
Preemption Notifiers

- KVM runs with some guest state loaded while in kernel mode (KVM running in kernel on behalf of guest)
- Must swap state when switching back to user mode
- Linux notifies KVM whenever it preempts a process that has guest state loaded
MMU Notifiers

- Linux does not know about the KVM MMU, so it cannot:
  - Flush shadow page table entries when swaps out a page
- So, add a notifier
  - so KVM can track changes to the Linux view of the process page table map
VM, Containers, Docker

Type 1 Hypervisor

Hypervisor
Hardware

Virtual Machine

App
Bins / libs
Operating System

Linux Containers

Operating System
Hardware

Container

App
Bins / libs

Docker

REST API

Dockerfiles

CLI

docker

App
Bins / libs
Container

App
Bins / libs
Container

App
Bins / libs
Container