KVM: Virtual vs. Physical Machines

Karen Noel
Red Hat Enterprise Linux - Virtualization

DevConf.cz Feb 2014
Agenda

- Virtual vs. Physical Machines
- CPU and Memory Hot-plug
- Live Migration
- Timers and Clocks
- Performance Monitoring
- Testing your application in both worlds
Linux on Physical Systems

Userspace
- App 1
- App 2
- App 3
- Linux processes

Linux Kernel
- Virtual memory
- I/O stack
- Scheduler

Physical Hardware
KVM Architecture

**libvirt**

**Guest OS**
- App 1
- App 2
- Kernel

**QEMU**
- VM monitor
- I/O emulation
- CPU emulation
- Virtual hardware

**Linux Kernel**
- Virtual memory
- I/O stack
- KVM
- Scheduler

**Physical Hardware**
KVM: Virtual Machines

Linux Kernel

Linux sub-systems

KVM

Physical Hardware

VM

QEMU

vCPU

vCPU

I/O

Apps

Guest OS

VM

QEMU

vCPU

vCPU

I/O

Apps

Guest OS

Linux

Processes

Thread

Thread

5
QEMU running on a Linux host

```bash
# ps -ef | grep -i qemu
qemu   26705   1   1 Mar27 ?   00:45:11 /usr/bin/qemu-system-x86_64
       -machine accel=kvm -name knoel1 -S -machine pc-q35-1.6,accel=kvm,
       usb=off -cpu host -m 1024 -realtime mlock=off -smp 1,maxcpus=4,
       sockets=4,cores=1,threads=1 -uuid f8023c16-22b1-4163-a7b5-df2452241fed
       -no-user-config -nodefaults -chardev...

# ps -mo pid,tid,fname,user,psr -p 26705
     PID  TID COMMAND  USER  PSR
  26705  -  qemu-sys  qemu  -
      -  26705  -  qemu  2
      -  26737  -  qemu  2
      -  26756  -  qemu  0
      -  26802  -  qemu  0
      -  26808  -  qemu  3
```
Why do people use Virtual Machines?

- Consolidation of HW resources – save money/power
- Hardware abstraction
- Fast provisioning
- Flexibility
- Cloud – Infrastructure as a Service (IaaS)
- And others
- It's important to understand the differences...
KVM

Hot-plug:
CPUs and Memory
CPU Hot-plug – physical hardware

- ACPI hot-plug – supported by high-end systems

- In-plug:
  - Operator inserts new CPU
  - ACPI sends event to OS
  - OS onlines CPU

- Un-plug:
  - OS offlines CPU
  - Calls ACPI eject method
  - Operator removes CPU
CPU Hot-plug – OS commands

- To online a CPU
  - echo 1 > /sys/devices/system/cpu/cpu#/online
- To offline a CPU
  - echo 0 > /sys/devices/system/cpu/cpu#/online
vCPU Hot-plug – virtual machine

- Leverages ACPI hot-plug support
- In-plug: initiated on host
  - QEMU creates new vCPU thread and structures
  - ACPI sends event to OS
  - libvirt calls guest agent to online CPU
- Un-plug: initiated on host
  - libvirt calls guest agent on offline CPU
  - Calls ACPI eject method – coming soon
  - QEMU removes vCPU thread and structures – coming soon
vCPU Hot-plug: uses guest agent

<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/channel/target/knoell.org.qemu.guest_agent.0'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
  <address type='virtio-serial' controller='0' bus='0' port='1'/>
</channel>

# virsh qemu-agent-command knoell '{"execute":"guest-ping"}'
# virsh qemu-agent-command knoell '{"execute":"guest-info"}'

• Commands:
  • guest-get-vcpus
  • guest-set-vcpus
  • etc.
vCPU In-Plug using virsh commands

```bash
# virsh qemu-agent-command knoel1 '{"execute":"guest-get-vcpus"}'
{
    "return": [
    {
        "online": true,
        "can-offline": false,
        "logical-id": 0
    },
    {
        "online": true,
        "can-offline": true,
        "logical-id": 1
    }
    ]
}
```

```bash
# virsh setvcpus knoel1 3
# virsh setvcpus knoel1 3 --guest
```
vCPU Un-Plug using virsh commands

```
# virsh setvcpus knoel1 2 --guest
# virsh qemu-agent-command knoel1 '{"execute":"guest-get-vcpus"}'
{
  "return": [
    {
      "online": true,
      "can-offline": false,
      "logical-id": 0
    },
    {
      "online": false,
      "can-offline": true,
      "logical-id": 1
    },
    {
      "online": true,
      "can-offline": true,
      "logical-id": 2
    }
  ]
}
```
Watch out for vCPU hot-plug

- Number of CPUs can change at any time
- Examples:
  - Daemons with # of threads = # of CPUs
  - Licensing based on # of CPUs
Memory Ballooning

- Host over-commits memory for Virtual Machines
  - Without swapping in the host
- Balloon driver runs in guest kernel
  - Inflate the balloon
    - Allocate memory from guest kernel
    - Return memory to host kernel
  - Deflate the balloon - opposite
- Stay tuned for automatic ballooning
- Stay tuned for memory hot-plug
Processor Emulation
CPU info – on host (SandyBridge)

```
# grep "model name" /proc/cpuinfo
model name : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
model name : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
model name : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
model name : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz

# grep flags /proc/cpuinfo
flags: fpu vme de pse tse msr... constant_tsc
arch_perfmon pebs... vmx... xsave avx... tpr_shadow
vnmi flexpriority ept vpid
```
CPU info – in guest (host-passthrough)

libvirt XML (on the host):
  <cpu mode='host-passthrough'>
  </cpu>

In the guest:
# grep "model name" /proc/cpuinfo
model name : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
# cat /proc/cpuinfo
flags: fpu vme de pse tsc msr... constant_tsc
arch_perfmon... xsave avx hypervisor lafh_lm xsaveopt
tsc_adjust

• CPUID feature flags do not match host!
  • Note: arch_perfmon set in both
  • Missing: vmx, tpr_shadow, vnmi, ept, vpid, others
  • Added: hypervisor
CPU info – in guest (SandyBridge)

libvirt XML (on the host):
<cpu mode='custom' match='exact'>
  <model fallback='allow'>SandyBridge</model>
  ...

In the guest:
# grep "model name" /proc/cpuinfo
model name : Intel Xeon E312xx (Sandy Bridge)
# cat /proc/cpuinfo
flags: fpu vme de pse tsc msr... constant_tsc...
xsave avx hypervisor lafh_lm xsavempo

- CPUID feature flags do not match host!
  - Missing: arch_perfmon
  - Missing: vmx, tpr_shadow, vnmi, ept, vpid, others
  - Added: hypervisor
CPU info – in guest (Nehalem)

libvirt XML (on the host):
<cpu mode='custom' match='exact'>
  <model fallback='allow'>Nehalem</model>
</cpu>

In the guest:
# grep "model name" /proc/cpuinfo
model name: Intel Core i7 9xx (Nehalem Class Core i7)
# cat /proc/cpuinfo
flags: fpu vme de pse tsc msr... constant_tsc...
  hypervisor lafh_lm

- Do CPUID flags match Nehalem exactly?
  - Missing: arch_perfmon
  - Missing: vmx, ept, others
  - Added: hypervisor
Use /proc/cpuinfo flags

- Processor model - same on host and guest
  - With different features
- Processor model – different on host and guest
  - Hypervisor might emulate features, might not
- Example:
  - PMU features based on processor model
Live Migration
Live Migration: move running VM to another host
Live migration: downtime - internal

- Downtime = time that vCPUs are not running
- vCPUs freeze in between instructions
- No disruption of kernel or applications

(qemu) info migrate
capabilities: xbzrle: off x-rdma-pin-all: off auto-converge: on
zero-blocks: off
Migration status: completed
total time: 172293 milliseconds
downtime: 3008 milliseconds
setup: 688 milliseconds
transferred ram: 22699191 kbytes
throughput: 5365.96 mbps
remaining ram: 0 kbytes
total ram: 268444380 kbytes
duplicate: 64939287 pages
skipped: 0 pages
normal: 5521319 pages
Live migration downtime - external

- Downtime = time that guest is not on network
- Ping from external system

...  
64 bytes from 10.8.8.248: icmp_req=10 ttl=63 time=0.400 ms  
64 bytes from 10.8.8.248: icmp_req=11 ttl=63 time=0.366 ms  
64 bytes from 10.8.8.248: icmp_req=12 ttl=63 time=0.451 ms  
64 bytes from 10.8.8.248: icmp_req=13 ttl=63 time=0.484 ms  
64 bytes from 10.8.8.248: icmp_req=14 ttl=63 time=0.464 ms  
64 bytes from 10.8.8.248: icmp_req=15 ttl=63 time=0.399 ms 

... 
64 bytes from 10.8.8.248: icmp_req=69 ttl=63 time=3091 ms *  
64 bytes from 10.8.8.248: icmp_req=74 ttl=63 time=0.426 ms  
64 bytes from 10.8.8.248: icmp_req=75 ttl=63 time=0.440 ms  
64 bytes from 10.8.8.248: icmp_req=76 ttl=63 time=0.450 ms  
64 bytes from 10.8.8.248: icmp_req=77 ttl=63 time=0.425 ms  
...
Live migration – auto-convergence

- QEMU “throttles” the vCPUs
- Helps the live migration converge

64 bytes from 10.8.8.248: icmp_req=40 ttl=63 time=0.400 ms
64 bytes from 10.8.8.248: icmp_req=41 ttl=63 time=0.366 ms
64 bytes from 10.8.8.248: icmp_req=42 ttl=63 time=0.451 ms
64 bytes from 10.8.8.248: icmp_req=43 ttl=63 time=0.484 ms
64 bytes from 10.8.8.248: icmp_req=44 ttl=63 time=0.464 ms
64 bytes from 10.8.8.248: icmp_req=45 ttl=63 time=0.399 ms
...
64 bytes from 10.8.8.248: icmp_req=81 ttl=63 time=21.4 ms
64 bytes from 10.8.8.248: icmp_req=82 ttl=63 time=0.540 ms
64 bytes from 10.8.8.248: icmp_req=83 ttl=63 time=5.88 ms
64 bytes from 10.8.8.248: icmp_req=84 ttl=63 time=1.95 ms
64 bytes from 10.8.8.248: icmp_req=85 ttl=63 time=16.1 ms
64 bytes from 10.8.8.248: icmp_req=86 ttl=63 time=4.68 ms
64 bytes from 10.8.8.248: icmp_req=87 ttl=63 time=0.434 ms
64 bytes from 10.8.8.248: icmp_req=88 ttl=63 time=3.63 ms
64 bytes from 10.8.8.248: icmp_req=89 ttl=63 time=2983 ms *
Timers and Clocks
Hardware Timers/Clocks

- RTC - Real Time Clock
- LAPIC Timer - periodic, one-shot or TSC-deadline (recent)
- ACPI Timer, Power Management (PM) Timer - low resolution, not programmable
- TSC - Time Stamp Counter
- HPET - High Precision Event Timer
Virtual Machine Timers and Clocks

• RTC works – used by Windows
• LAPIC one-shot timer – fine in VM
• ACPI Timer, (PM) Timer – fine in VM, also used by Windows
• TSC – not reliable due to live migration even if iTSC
• HPET – no compensation for missed ticks (slow)

Timers and Clocks – para-virtualized

- kvm-clock – para-virtualized clock
- Hyper-V reference time counter – for Windows – MSR (emulated in host kernel)
- Hyper-V iTSC – for Windows
Timers and Clocks: Watch out for...

- Do not read TSC directly
  - TSC frequency may change on live migration!
  - Destination host may not have iTSC
  - TSC Frequency in MSR_PLATFORM_INFO – not emulated on KVM!
- Use standard clock functions for timing
  - Example: `clock_gettime(CLOCK_MONOTONIC, &ts)`
- Use NTP within the guest
  - Adjust for live migration downtime
  - Compensate for clock drift – even with kvm-clock!
- Leapseconds
Para-virtualized Clock – kvm-clock

```bash
# cat /sys/devices/system/clocksource/clocksource0/
current_clocksource
kvm-clock
```

- `struct pvclock_vcpu_time_info { ... }`
- Fields to calculate accurate system time (ns accuracy)
  - Offset from host TSC timestamp
  - Shift & multiply to adjust from host frequency
  - Migration count (version field)
    - To different host pCPUs
    - To different pCPUs on different hosts
Para-virtualized Clock – new vsyscall

- Very efficient vsyscall for gettimeofday() and clock_gettime()
  - Introduced in 3.7 kernel
- Page with pvclock_vcpu_time_info structure
  - Mapped in userspace for each vCPU
  - Written by KVM in the host
  - Read by guest userspace and kernel
Performance Monitoring
Performance Monitoring - PMU

- Traditionally...
  - Each processor model = different PMU design
- On recent Intel x86_64 processors:
  - Subset of performance events are “architectural”
  - Same on all - arch_perfmon feature
- QEMU sets arch_perfmon with “-cpu host”
- libvirt XML = <cpu mode='host-passthrough'>
Performance Monitoring – architectural events

# perf list

List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles          [Hardware event]
  instructions                 [Hardware event]
  cache-references              [Hardware event]
  cache-misses                  [Hardware event]
  branch-instructions OR branches [Hardware event]
  branch-misses                 [Hardware event]
  bus-cycles                    [Hardware event]
  idle-cycles-frontend          [Hardware event]
  idle-cycles-backend           [Hardware event]
  ref-cycles                    [Hardware event]
Performance Monitoring – non-arch events

- All other hardware events are non-architectural
  - Execution unit usage counts, ...
  - Events that rely on other MSRs to be set first
  - Uncore events – cannot be multiplexed by KVM

Performance Monitoring – live migration

• Issues with live migration and non-arch PMU
  • Events may not exist on another model
  • May be programmed differently on another model
  • Live migration to another model host --> bad results
Current QEMU and libvirt Support

- QEMU - arch_perfmon CPUID feature flag for -cpu host
- libvirt – disallow live migration with cpu host-passthrough unless same model on destination
- Future, for emulated processor models
  - QEMU – provide arch_perfmon?
  - KVM – disallow use of non-architectural PMU?
Conclusion: Testing!
Test on Physical Systems

- For testing hardware specific things
- Ensure application does not make Virtual Machine specific assumptions
Test on Virtual Machines

- Configuration flexibility!
- 1 to 160 vCPUs (recommend up to host CPUs)
  - Don't assume uni-processor systems are gone!
- 512M up to 4TiB (do not over-commit too much)
  - When over-committing, use ballooning or large swap file
- Test on many emulated processor models
- Test with live migration
  - Know the max_downtime for your application
Questions?