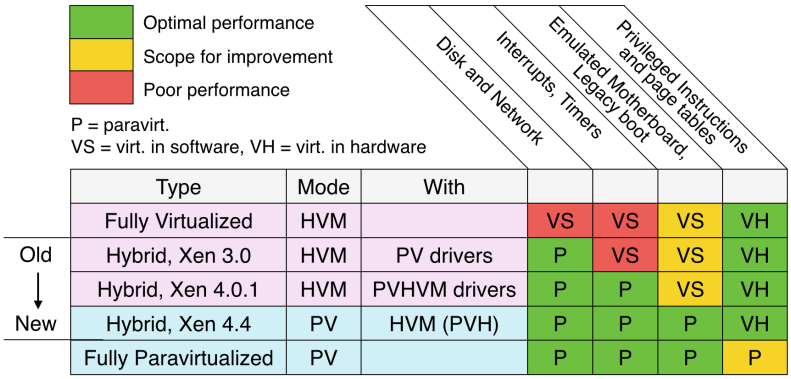
Xen Feature Detection

09 May 2014

In my previous post What Color Is Your Xen, I summarized the different Xen modes, and presented them in a slightly different manner to emphasize the evolution of versions. The current spectrum of modes has: HVM (full hardware virtual machine), PV (full paravirtualization), HVM with PV drivers, PVHVM, and PVH. So what mode is your Xen guest running with now?



Type	Mode	With				
Fully Virtualized	HVM		VS	VS	VS	VH
Hybrid, Xen 3.0	HVM	PV drivers	P	VS	VS	VH
Hybrid, Xen 4.0.1	HVM	PVHVM drivers	P	P	VS	VH
Hybrid, Xen 4.4	PV	HVM (PVH)	P	P	P	VH
Fully Paravirtualized	PV		P	P	P	P

There are various tools you can use check whether HVM is enabled and which PV features are active, which I've summarized for Linux in this post. These identify your guest type on the spectrum image above. Note that I'm not a Xen developer, so I'm not the source for Truth on this topic, although I hope I can help.

The following commands are intended for EC2 users, where you can't just read the AWS Xen config. For more info on verifying features, see Xen Linux PV on HVM drivers.

dmesg

I wish there was one tool that explained what mode/features a Xen guest was using. There are virtualization detection tools, like `virt-what`, `imvirt`, and `xen-detect`, but none of these provide much detail other than "Xen" and "HVM". The closest, right now, is running dmesg, and looking for lines containing "Xen".

Here is an EC2 PV instance, with a Linux 3.2.0-54-virtual kernel:

```
$ dmesg | grep -i xen
[    0.000000]  Xen: 0000000000000000 - 00000000000a0000 (usable)
[    0.000000]  Xen: 00000000000a0000 - 0000000000100000 (reserved)
[    0.000000]  Xen: 0000000000100000 - 00000001e0800000 (usable)
[    0.000000] xen: setting RW the range fda000 - 1000000
[    0.000000] xen: setting RW the range fffff000 - 100000000
[    0.000000] Booting paravirtualized kernel on Xen
[    0.000000] Xen version: 4.2.amazon (preserve-AD)
[    0.000000] Xen: using vcpuop timer interface
[    0.000000] installing Xen timer for CPU 0
[    0.044289] installing Xen timer for CPU 1
[    0.076374] installing Xen timer for CPU 2
[    0.077323] installing Xen timer for CPU 3
[    0.099492] PCI: setting up Xen PCI frontend stub
[    0.100172] xen/balloon: Initialising balloon driver.
[    0.100429] xen-balloon: Initialising balloon driver.
[    0.100429] Switching to clocksource xen
[    0.432488] Initialising Xen virtual ethernet driver.
[    1.483943] XENBUS: Device with no driver: device/console/0
```

I've highlighted a few interesting lines. The first two show that this is PV, on Xen 4.2. The "vcpuop timer interface" shows that this is using PV timers. The last shows the PV network driver is enabled.

Now an EC2 "HVM" (PVHVM) instance on Linux 3.2.0-54-virtual:

```
$ dmesg | grep -i xen
[    0.000000] DMI: Xen HVM domU, BIOS 4.2.amazon 01/24/2014
[    0.000000] Hypervisor detected: Xen HVM
[    0.000000] Xen version 4.2.
[    0.000000] Xen Platform PCI: I/O protocol version 1
[    0.000000] Netfront and the Xen platform PCI driver have been compiled for this kernel:
 unplug emulated NICs.
[    0.000000] Blkfront and the Xen platform PCI driver have been compiled for this kernel:
 unplug emulated disks.
[    0.000000] ACPI: RSDP 00000000000ea020 00024 (v02    Xen)
[    0.000000] ACPI: XSDT 00000000fc00f710 0005C (v01    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: FACP 00000000fc00f260 000F4 (v04    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: DSDT 00000000fc0035e0 0BBF6 (v02    Xen      HVM 00000000 INTL 20090123)
[    0.000000] ACPI: APIC 00000000fc00f360 000D8 (v02    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: SRAT 00000000fc00f4b0 00170 (v01    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: HPET 00000000fc00f620 00038 (v01    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: WAET 00000000fc00f660 00028 (v01    Xen      HVM 00000000 HVML 00000000)
[    0.000000] ACPI: SSDT 00000000fc00f690 00031 (v02    Xen      HVM 00000000 INTL 20090123)
[    0.000000] ACPI: SSDT 00000000fc00f6d0 00031 (v02    Xen      HVM 00000000 INTL 20090123)
[    0.000000] Booting paravirtualized kernel on Xen HVM
[    0.000000] Xen HVM callback vector for event delivery is enabled
[    0.127240] Xen: using vcpuop timer interface
[    0.127247] installing Xen timer for CPU 0
[    0.145766] installing Xen timer for CPU 1
[    0.244020] installing Xen timer for CPU 2
[    0.343945] installing Xen timer for CPU 3
[    0.703463] xen/balloon: Initialising balloon driver.
[    0.704063] xen-balloon: Initialising balloon driver.
[    0.746857] Switching to clocksource xen
[    0.766729] xen: --> pirq=16 -> irq=8 (gsi=8)
[    0.766777] xen: --> pirq=17 -> irq=12 (gsi=12)
[    0.766809] xen: --> pirq=18 -> irq=1 (gsi=1)
[    0.766840] xen: --> pirq=19 -> irq=6 (gsi=6)
[    0.766877] xen: --> pirq=20 -> irq=4 (gsi=4)
[    0.907476] xen: --> pirq=21 -> irq=47 (gsi=47)
[    0.907479] xen-platform-pci 0000:00:1f.0: PCI INT A -> GSI 47 (level, low) -> IRQ 47
[    1.424040] Initialising Xen virtual ethernet driver.
[    1.482442] XENBUS: Device with no driver: device/vfb/0
[    1.484535] XENBUS: Device with no driver: device/console/0
```

This time "Xen HVM" is detected, and we have "unplug emulated" lines for NICs and disks – this is Xen switching from the slower HVM network and disk drivers, to PV ones, after finding they are available.

The "HVM callback vector" line shows that PV interrupts are enabled (from PVHVM), which is a big difference. On full HVM mode, emulated PCI interrupts are used for device I/O delivery, along with emulating the PCI bus, local APIC, and IO APIC. If you doing a high rate of disk I/O or network packets – which is easy to do on today's networks – these emulation overheads add up. With vector callbacks instead of interrupts, the Xen hypervisor can call the destination guest driver directly, avoiding these overheads.

Another advantage of the vector callback interface is that Xen can perform CPU load balancing depending on the type of interrupt, instead of interrupts always directed to one CPU. Stefano Stabellini, the lead developer for this work, summarized these in his Linux PV on HVM talk (slides), which I recommend.

Note that newer kernels can add more lines; eg:

```
[    0.000000] xen: PV spinlocks enabled
```

This message was added recently (PV spinlocks have been available for a long time).

/sys/hypervisor

There are some useful details in this location, especially the feature bits:

```
# cat /sys/hypervisor/properties/features
00000705
```

Except I really need a tool to turn this into something human readable. They are defined in the Linux source, under include/xen/interface/features.h.

Ok, I just wrote a simple one, called xen-features.pl (GitHub):

```
$ ./xen-features.pl
Xen features: 00000705
enabled: writable_page_tables
enabled: auto_translated_physmap
enabled: hvm_callback_vector
enabled: hvm_safe_pvclock
enabled: hvm_pirqs
```

So far a little better than the raw hex number. This could easily be enhanced to say more.

I did write this in Perl, despite the hackernews community repeatedly saying that Perl is dead. It fits better with Xen this way, as the Xen hypervisor is also written in Perl, as are the Linux PV drivers – which actually stand for Perl-Virt, and not Para-Virt. (ok, I made that up. Here's the python version.)

Other Tools

Here are various other tools that might be helpful. Note that I'm not a Xen developer, and I'm not certain of these. The best reference (other than a Xen developer) is the Xen code, and the linux/arch/x86/xen code, which I've been reading.

Disks

Note the device names from `iostat`:

```
$ iostat
Linux 3.10.37 (bgregg-test-i-d98d7689)     04/29/2014

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.79    0.00    0.64    0.02    0.01   97.55

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
xvda1             1.06         2.51        22.45     853074    7629296
xvdb              0.12         0.08         0.44      25931     149584
xvdc              0.12         0.07         0.96      25154     325874
md0               0.17         0.14         1.40      48437     475458
```

The "xvd" devices are paravirt, which is good, as these are expected to be faster than HVM.

These devices can also be listed from /sys:

```
# echo /sys/bus/xen/devices/vbd*
/sys/bus/xen/devices/vbd-51728 /sys/bus/xen/devices/vbd-51744 /sys/bus/xen/devices/vbd-51760
/sys/bus/xen/devices/vbd-51776 /sys/bus/xen/devices/vbd-76
```

The virtual block devices (vbd) are paravirt.

Network

The following line shows the paravirt driver initializing:

```
# dmesg | grep xen_netfront
[    3.116464] xen_netfront: Initialising Xen virtual ethernet driver
```

This is good, as the paravirt driver is expected to be faster than HVM (which would not show this line). Faster still is SR-IOV (not shown here).

The driver type can also be checked using `ethtool`:

```
# ethtool -i eth0
driver: vif
version:
firmware-version:
bus-info: vif-0
```

The virtual interface (vif) driver is paravirt.

The vif driver can also be seen under /sys:

```
# echo /sys/bus/xen/devices/vif*
/sys/bus/xen/devices/vif-0
```

This shows a paravirt device is available.

Interrupts

Checking for paravirt interrupts:

```
$ dmesg | grep 'HVM callback vector'
[    0.000000] Xen HVM callback vector for event delivery is enabled
```

This shows the faster PVHVM version of interrupts, callback vectors, is used instead of emulating PCI interrupts the HVM way, for which this line would not be printed.

Timers

Checking for paravirt timers:

```
$ dmesg | grep 'Xen timer'
[    0.200843] installing Xen timer for CPU 0
[    0.220082] installing Xen timer for CPU 1
[    0.239894] installing Xen timer for CPU 2
[    0.254956] installing Xen timer for CPU 3
[...]
```

This shows the faster PV timers are in use.

Motherboard

Checking the type of motherboard virtualization:

```
# dmidecode
# dmidecode 2.11
# No SMBIOS nor DMI entry point found, sorry.
```

No real output from `dmidecode` happens for a paravirt guest.

The following shows `dmidecode` for HVM:

```
# dmidecode
# dmidecode 2.11
SMBIOS 2.4 present.
47 structures occupying 1371 bytes.
Table at 0x000EB01F.

Handle 0x0000, DMI type 0, 24 bytes
BIOS Information
        Vendor: Xen
        Version: 4.2.amazon
        Release Date: 01/24/2014
        Address: 0xE8000
        Runtime Size: 96 kB
        ROM Size: 64 kB
        Characteristics:
                PCI is supported
                EDD is supported
                Targeted content distribution is supported
        BIOS Revision: 4.2

Handle 0x0100, DMI type 1, 27 bytes
System Information
        Manufacturer: Xen
        Product Name: HVM domU
        Version: 4.2.amazon
[...]
```

HVM is expected to be slower, especially during boot.

HVM

Checking for HVM mode:

```
# dmesg | grep 'Xen HVM'
[    0.000000] DMI: Xen HVM domU, BIOS 4.2.amazon 01/24/2014
[    0.000000] Hypervisor detected: Xen HVM
[    0.000000] Booting paravirtualized kernel on Xen HVM
[    0.000000] xen:events: Xen HVM callback vector for event delivery is enabled
```

This shows that HVM is active, which will use hardware virtualization for privileged instructions and page tables should be fastest. This is expected to be faster than paravirt.

The capabilities of the hypervisor can also be checked:

```
# grep hvm /sys/hypervisor/properties/capabilities
xen-3.0-x86_64 xen-3.0-x86_32p hvm-3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-x86_64
```

This shows that the physical host and Xen hypervisor you are on support HVM, which is useful to know. It does *not* show that HVM is active for your current guest (in fact, this example came from a PV guest).

Red Herrings

Many sites recommend checking for the following flags, to see if HVM is supported:

```
# egrep 'vmx|smv' /proc/cpuinfo
#
```

Well, it showed nothing for me, despite Xen boot messages claiming this is HVM. These flags can only be inspected from the host, not the guest, where they may be masked if nested virtualization is not supported or disabled (eg, for security reasons).

Also, configured doesn't mean active:

```
# grep XEN.*PV /boot/config-3.15.0-rc2
CONFIG_XEN_PVHVM=y
CONFIG_XEN_PVH=y
CONFIG_XEN_HAVE_PVMMU=y
```

These mean the guest supports these modes, if the hypervisor does. I've turned on PVH here and booted the kernel, but this Xen version (4.2) doesn't have PVH.

More Info

In this post showed tools you can use from within a guest to see which HVM or PV features are enabled. For more information on this topic, see Xen Linux PV on HVM drivers from the Xen wiki, and also the tools in the Xen source (xen/tools).

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).