Paravirtualisation

COSC349—Cloud Computing Architecture
David Eyers

Learning objectives

- Define **paravirtualisation**
- Give a benefit and a downside of paravirtualisation
- Describe why **timekeeping** within a VM is difficult
- Give examples of different **paravirtualised device drivers** and their purpose

Paravirtualisation

- Complete isolation of VMs should imply VMs do not know that they are virtualised
  - However it can be ideal that VMs **actually know they're virtual**!
    - … *e.g.*, otherwise VMs may waste time managing fake devices
  - **Paravirtualisation** describes a VMM that runs **VM-aware OSs**

- Paravirtualization downside: VM **OS needs modification**
- Upside: guest requests privileged operations from VMM
  - Avoid frequent need to intercept guest OS kernel (inefficient)

The Xen project relied on paravirtualisation

- Using paravirtualisation means **Xen VMM is very small**
  - ... which in turn made it practical for University development
- Xen VMM was designed first, and then OSs ported to it
  - Paravirtualisation of Linux and Windows XP
- Microsoft did not release the Xen-compatible Windows
  - (It may well have not been a complete implementation.)
  - ... but better **CPU support for virtualisation** arrived soon after
  - Thus didn't need to try to get Microsoft to cooperate with Xen

Xen, dom0 and Linux kernels

- Recall that minimisation of Xen's VMM meant **a special VM** (dom0) was used to **manage the actual hardware**
  - dom0 Linux VM contains device drivers for real host hardware
  - dom0 Linux VM directly accesses these hardware devices
- Linux kernels could be patched—‘xenified’ for dom0
  - Many distributions provided convenient access to Xen kernels
- Soon came non-Linux dom0s: NetBSD, OpenSolaris, …

Mainline Linux is now paravirtualisable

- In 2006 Xen, IBM, Red Hat, and VMware met and agreed to collaborate on **paravirt-ops initiative**
  - Linux **paravirtualises itself** on VMMs or **boots normally** otherwise
  - Agnostic to the underlying VMM, and **supports many VMMs**:
    - Xen, VMware Workstation, VirtualBox, …
- Since Linux 2.6.37 (Jan 2011) mainline Linux kernels can be efficient Xen dom0 and domU without modification
  - However this is mostly about paravirtualising CPU features
  - Hardware device drivers we will discuss later

Potential pain point: timekeeping

- How can OS know what the time of day is?

- Time-of-day is maintained by **battery-backed clock**
  - Hardware clock access is really slow compared to CPU, so just:
    - read/write actual hardware clock once on OS startup/shutdown
    - maintain time of day using high frequency OS time source

- Further: time of day needs **resynchronisation**, since:
  - leap seconds are declared and must be added when necessary
  - timekeeping components will drift based on temperature, *etc*.

Potential pain point: timekeeping

- FYI—Timesource used by Linux? You can choose any of:
  - **HPET**—high precision event timer (hardware)
  - **PIT**—(older) programmable interval timer (hardware)
  - **TSC**—timestamp counter (built into CPU)
  - **ACPI_PM**—ACPI power management timer (hardware)
  - **Cyclone**—IBM EXA time source: some Itanium thing …
  - **SCX200_HRT**—… some high resolution timer …
    - (… and no doubt some other ones I don’t know anything about …)

- Haven’t even brought virtualisation into picture yet…

Virtualisation and clock sources?

- x86 hypervisors virtualise PIT, RTC, HPET, ACPI_PM, but the read speeds are too slow for a good clock source
- **TSC** is the most common non-VM clock source:
  - auto incrementing, **high precision counter** within the CPU
  - can be **read from user space in one instruction** (RDTSC)
  - … but counter can be reset while system is running
- Migrating a VM to a different physical host (+VMM)?
  - TSC offsets won't be equal: **VM's TSC might jump backwards!**
  - TSC frequencies need not be the same either

OK, so how do VMs measure time passing?

- Host OS can devote resources to timekeeping
  - but VM guest OSs can't sensibly do so: just get time from host
- Xen and KVM use the `pvclock` protocol
  - Shares a structure between host and guest
  - Allows guests to determine a reasonable TSC equivalent
- Intel VT-x added a control for hosts to add TSC offset
  - but TSC frequency needs control too… (Intel added in 2015)

Paravirtualised device drivers & virtue

- Previously discussed **paravirtualising OS kernel** functions
- Often hardware is accessed through device drivers
  - (Too many different types to build directly into OS effectively)
- Can use **paravirtualised dev. drivers** in unmodified OS
  - VirtualBox's guest extensions; VMware's Guest Tools; Xen's …

- **virtio** provides a set of common **emulated devices**
  - Specifically the front-end drivers within the guest OS
  - Back-end drivers map virtio API to real device drivers in host OS

The five typical front-end drivers in virtio

- vrtio-blk—*i.e.*, **block devices**: hard disks, DVD drives, …
- vrtio-net—*i.e.*, **network adapters**
- virtio-pci—*i.e.*, **PCI pass through**
  - PCI is for interconnecting peripherals with the CPU
  - *e.g.*, hot-pluggable storage devices
- virtio-console—*i.e.*, the **keyboard and screen**
  - Well, very basic versions of them, but useful for diagnostics
- virtio-balloon—for managing **guest memory size**
  - ... see next slide

Dynamically changing guest memory size

- When an OS starts up, it determines its **memory size**
  - This amount is usually then fixed until the point of reboot
  - (exception: some types of server hardware—$$$)
- Paging means host memory can be over-provisioned
  - VMs won't cause problems if they don't use all their memory
  - But guest OS may fill guest memory with unimportant caches
- **Balloon driver** is a process in the VM that allocates RAM
  - … but communicates with VMM to **give it back to the host OS**!
  - Analogy is inflating RAM balloon—guest OS minimises its use