Real-time in a virtual system?! Comparing real-time capabilities of various types of Linux hypervisors and containers, Basic lecture

Technical concepts of Linux hypervisors and containers

Alexander Bähr
Open Source Automation Development Lab (OSADL) eG

COMPACT
OSADL
ONLINE
LECTURES

Open Source Automation Development Lab eG

Main topics

- Virtualization: Definition and overview
  - Hardware virtualization
  - System virtualization
- Hardware virtualization example: kvm
- System virtualization examples: chroot, docker
- Conclusion

Virtualization: Definition and overview

Virtualization: Provision of resources **simulated** by software

- Hardware:
  - Particular **hardware**, *e.g.* interfaces (network, serial, ...)
  - **Platform** virtualization , incl. CPU, memory, devices

- Software:
  - Operating system emulation
  - Services, *e.g.* OpenGL
  - Application

Origin of virtualization: hardware time sharing

- **Shared usage** of computer resources among a large group of users
- **Increase** the **efficiency** of both users and expensive computer resources **by sharing**
- Costs for providing computing capability **dropped considerably** -> use a computer without actually owning one

Advantages of virtualization technologies

- Resource optimization: Assign each VM precisely the **amount of computing power** it needs for its jobs or tasks
- Increase the uptime of services over the network/Internet – 99.999 % (five nines) or better through **live migration**
- **Back up**, **copy**, and **clone** VMs
- Reduced hardware requirements also **reduces power consumption** -> minimizing the carbon footprint (**green IT**)

Reasons for the wide popularity of virtualization

- Hardware isolation
  - Running legacy OS in safe environment
  - Maintenance, manageability (i.e. installation of server)
  - Security and/or performance isolation
- Testing
  - Development of i.e. a driver without hardware
  - Wide range of test scenarios
- Power savings
  - Fewer machines = easier maintenance
  - Reduced energy costs

Virtualization: chronological overview

- 1968 CP/CMS from IBM with **Virtual Machines** (VM), community and growing interest provide a strong development
- Publication of **various hypervisors** and **virtualization technologies** over the years (i.e. chroot, XEN, VM-Ware, Hyper-V)
- 23 Oct 2006, **kvm patch set** for Linux 2.6.20 to run virtual machines on Linux **without** using a full hypervisor like XEN

Hypervisor definition (Popek and Goldberg)

- Equivalence / Fidelity

  A program running under the VMM should exhibit a behavior **essentially identical** to that demonstrated when running on an **equivalent machine** directly.

- Resource control / Safety

  The VMM must be in **complete control** of the virtualized resources.

- Efficiency / Performance

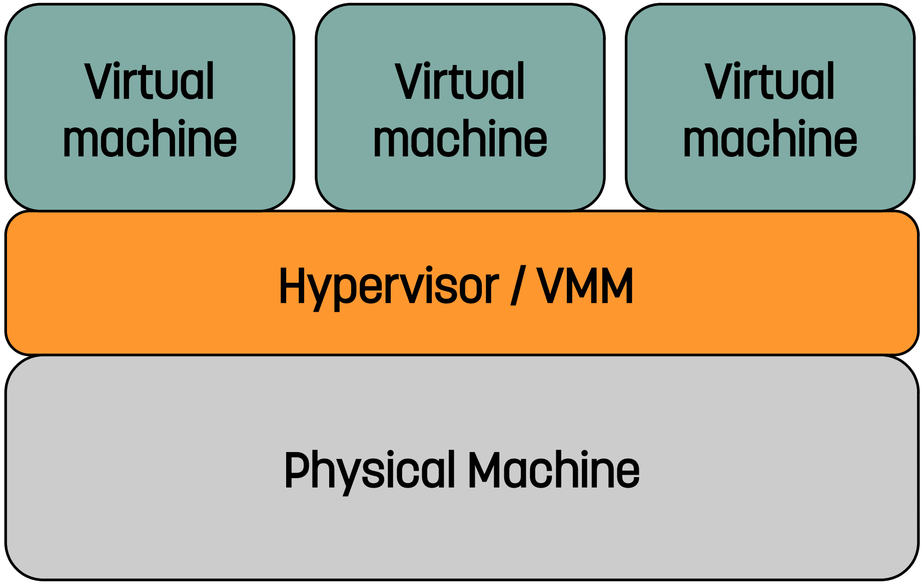
  A statistically dominant fraction of machine instructions must be executed **without VMM intervention**.

Hardware (full) virtualization

- Physical virtualization
- The **hypervisor** separates the guest systems from the host.
- It provides **simulated hardware** to the guest systems at instruction level.
- The guest systems are **completely independent** from each other and not aware that they are virtualized (except by inspecting the configuration).
- Examples: kvm

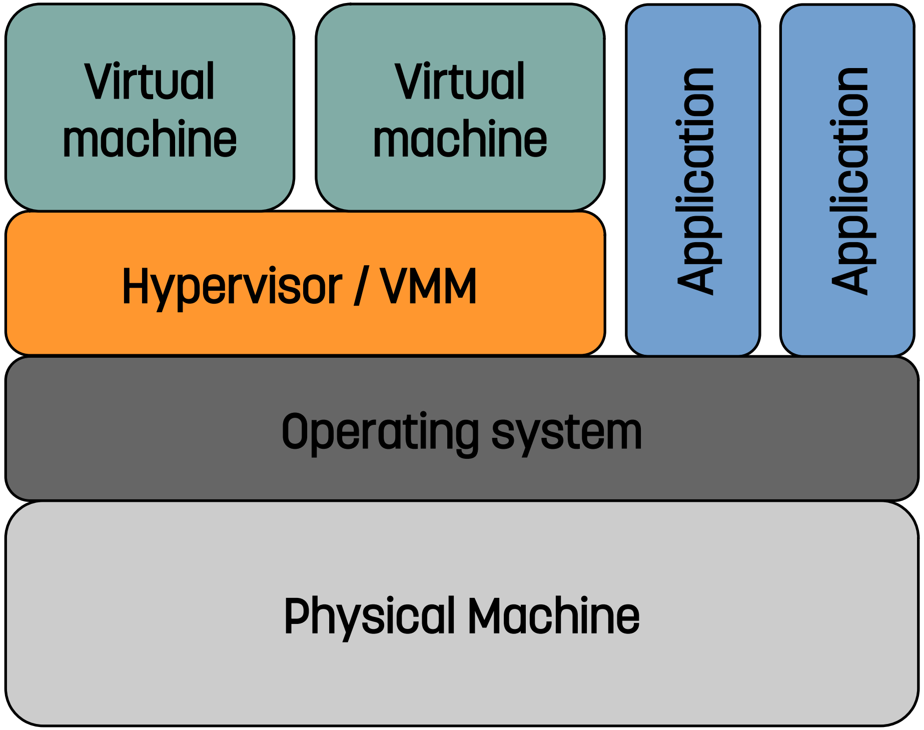
Hardware virtualization – Type-1 hypervisor

- runs directly on the underlying computer's physical hardware
- interacting directly with its CPU, memory, and physical storage
- takes the place of the host operating system
- Examples: kvm, XEN, jailhouse



Hardware virtualization – Type-2 hypervisor

- runs as an application in an OS
- accesses computing, memory, and network resources via the host OS
- can cause latency issues, affecting performance
- Security risk: compromised host OS could manipulate a guest OS
- Examples: QEMU



Hypervisor example: kvm

- Kernel module **kvm.ko** → add type-1 hypervisor to Linux
- **Open Source project** (GPL-2.0-only)
- Full functionality merged into Linux 2.6.20 on 5. Feb. 2007
- Initially provides CPUs from **Intel** with the **vmx** (**v**irtual **m**achine e**x**tension) and **AMD** with **svm** (**s**ecure **v**irtual **m**achine)
- Up to now full visualization on **x86, x86-64, s390, PowerPC, ARM**
- Use of the **same hardware,** so guest CPU same as host

Hypervisor example: kvm

Date Thu, 19 Oct 2006 15:45:49 +0200
From Avi Kivity <>
Subject [PATCH 0/7] KVM: Kernel-based Virtual Machine

share 0

```
The following patchset adds a driver for Intel's hardware virtualization
extensions to the x86 architecture.  The driver adds a character device
(/dev/kvm) that exposes the virtualization capabilities to userspace.  Using
this driver, a process can run a virtual machine (a "guest") in a fully
virtualized PC containing its own virtual hard disks, network adapters, and
display.
```

“It was the right project at the right time,” Kivity said. “Virtualization was very hot at the time, and the only open-source alternative at the time was Xen, which was outside of the Linux kernel.”

Prerequisits for virtualization on x86

- CPU-extensions (Intel, AMD)
  - Guest operating mode
  - Hardware state switch
  - Exit reason reporting
- Kvm architecture
  - Character device in `/dev/kvm`
  - Kvm API: `ioctl()` to control vm (type: system, vm, or vcpu)

Setup kernel and system for use of kvm

- Config and build kernel with:

```
-General setup
    [*]Control Group support
            [*]CPU controller
            [*]Cpuset controller
            [*]Freezer controller
            [*]Device controller

-Virtualization
    [*] Kernel Based Virtual Machine (KVM) support
            → (select the hardware dependent options)
```

- Install on system:

```
libvirt-bin
libvirt-daemon
```

Management of kvm-systems

Several tools available to manage kvm virtual machines, different emphasis for particular use cases, i.e.:

- QEMU/kvm (straight from the command line)
- virsh (cli)
- kvm-admin (python-scripts)
- Proxmox VE (server virtualization management, web,cli, API)
- kimchi (web, python)

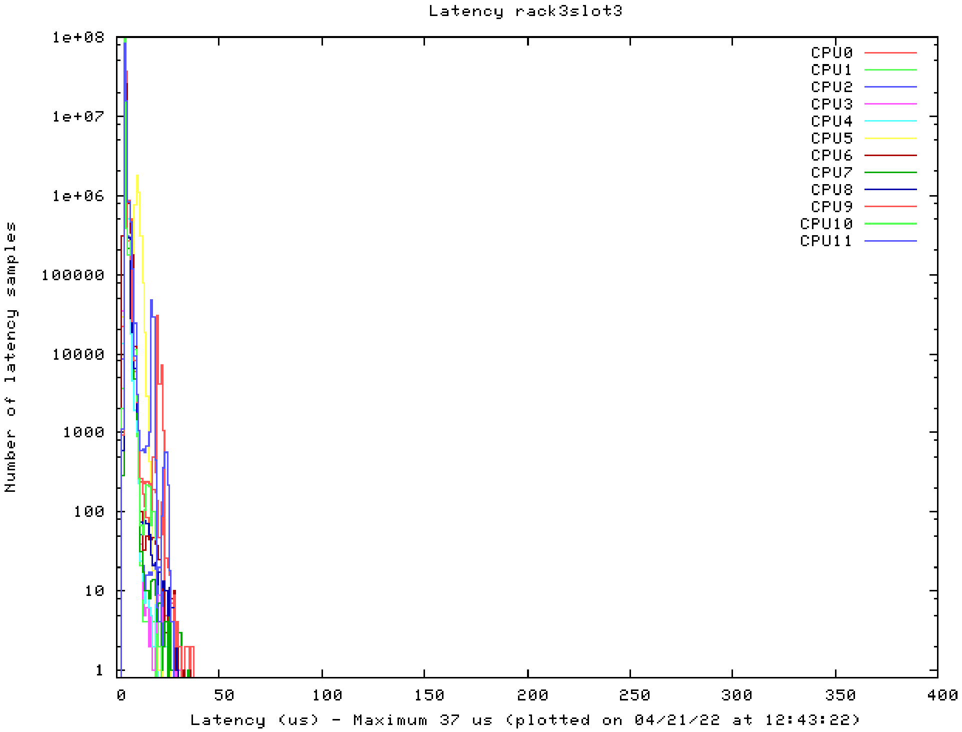
Management of kvm-systems - QEMU

QEMU (Quick Emulator) can work in two modes:

- **Full system emulation** mode, is to provide virtualization of an entire machine (CPU, memory, IO devices) to run a guest OS
  - CPU may be fully emulated using DBT (Dynamic Binary Translation)
  - Use a hypervisor such as KVM to gain better performance
- **User mode emulation**, where only a binary compiled for one CPU is executed on another CPU
  - To ease cross-compilation and cross-debugging
  - Running the Wine Windows API emulator

Scenario: Realtime and kvm (rack 3 slot 3)

Host



Guest

COMPACT OSADL ONLINE LECTURES

OSADL Open Source Automation Development Lab eG

Realtime with kvm?

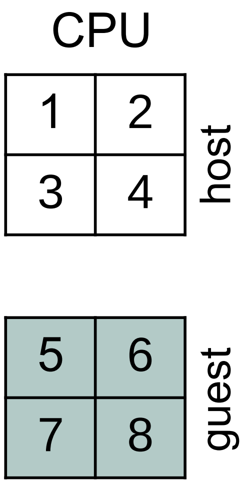
- Priority and lock holders are not visible for the host, therefore no priority inheritance.
- To achieve realtime in kvm:

CPU

1	2
3	4
5	6
7	8

Realtime with kvm?

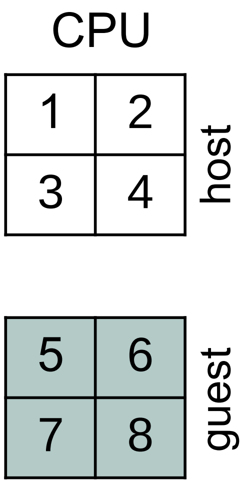
- Priority and lock holders are not visible for the host, therefore **no priority inheritance**.
- To achieve realtime in kvm:



- **Partition** between CPUs running system tasks and **isolated CPUs** running **realtime guests** (`isolcpu` and `nohz` in the kernel command line).

Realtime with kvm?

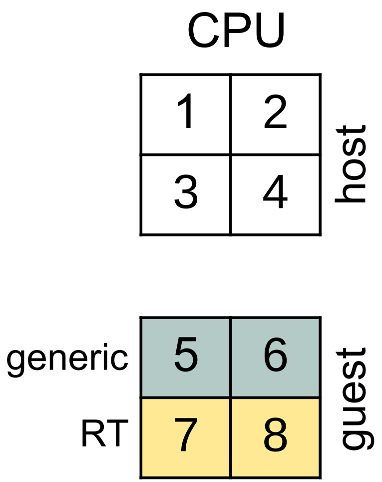
- Priority and lock holders are not visible for the host, therefore **no priority inheritance**.
- To achieve realtime in kvm:



  - **Partition** between CPUs running system tasks and **isolated CPUs** running **realtime guests** (`isolcpu` and `nohz` in the kernel command line).
  - Run virtual CPUs (VCPUs) of the guest with **very high priority**, only interfered by ksoftirqd.

Realtime with kvm?

- Priority and lock holders are not visible for the host, therefore **no priority inheritance**.
- To achieve realtime in kvm:



- **Partition** between CPUs running system tasks and **isolated CPUs** running **realtime guests** (`isolcpu` and `nohz` in the kernel command line).
- Run virtual CPUs (VCPUs) of the guest with **very high priority**, only interfered by ksoftirqd.
- On the guest **partition** between **realtime** VCPUs and **generic task** VCPUs.

Realtime with kvm?

Additional configurations to improve realtime capabilities of the guest systems:

- Disable **hyperthreading** (in BIOS) or via Kernel command line:

  `nr_cpus=<x>` (x=physical number of CPUs)

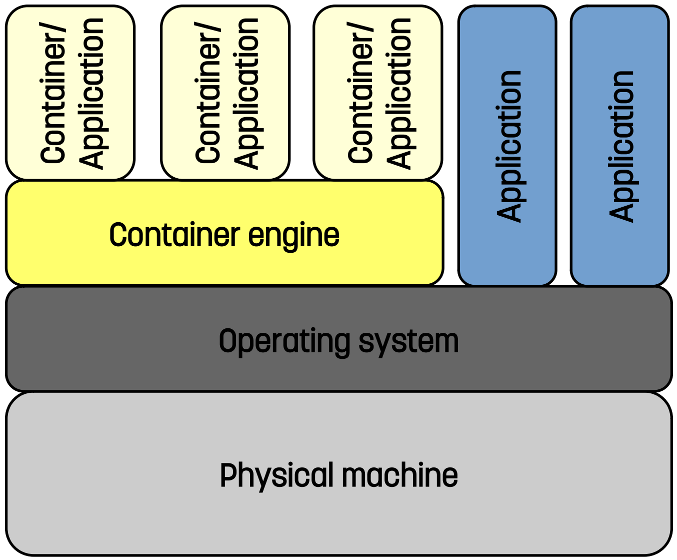
- Disable **interrupt balancing:**

```
# systemctl disable irqbalance
# reboot
```

System-level virtualization: containers

Main properties:

- multiple isolated user space instances, called containers
- includes all the code, its dependencies and even the operating system itself
- Ease of use
- Examples: Docker, proxmox, chroot



Container virtualization - docker

- "Docker" is the trademark of the company Docker®, Inc.
- Docker provides **isolation** between the workload and reproducible environment.
- Virtualization takes place at **operating system level.**
- The separation of guest systems is based on **partitioning** of resources.
- The guest systems use the host system's **kernel** and **virtual filesystems**.

Container virtualization - docker

- Docker is a tool to **distribute applications.**
- Docker uses a technology called **namespaces** to provide the isolated workspace called the container.
- **Docker Hub** provides the infrastructure for storing and sharing their code in containers.
- Personal use of **Docker Hub** is free of charge, incl. public repositories, other business plans available.
- Most containers include Free and Open Source software (FOSS) → **license obligations have to be fulfilled**

System level virtualization with chroot

- Introduced 1979 during development of Version 7 UNIX
- Create and host a **separated** virtualized copy of a root file system **(chroot jail)**
- Useful for
  - **Recovery** of damaged environment after bootstrapping from an alternate root file system.
  - **Test** of unstable/risky software in a safe environment
  - **Run and develop** software with the expected dependencies without the need to install it on the host

System level virtualization with chroot

- Change to the new root filesystem

```
# cd [NEW ROOT FS]
```

- Mount system fs:

```
# mount -t proc proc ./proc
# mount -t sysfs nodev ./sys
# mount -t devtmpfs nodev ./dev
# mount -t devpts devpts ./dev/pts
# mount -t debugfs nodev ./sys/kernel/debug
```

- Chroot to [NEW ROOT FS]

```
# chroot [NEW ROOT FS] /bin/bash
```

Summary

- In general, **virtual systems** do not inherit their host's deterministic behavior due to various separation methods.
- **Real-time capabilities** can be improved with additional configurations of host and guest systems.
- **System virtualization** (chroot, Docker, etc.) shows superior real-time performance compared to **physical virtualization**. However, it does not offer the same **degree of separation** between host and guest systems.