Virtualisation - A Spectrum of Technologies

Ross Alexander

July 20, 2020

Problems to solve

Historically computing resources have been very expensive. While these have come down in price dramatically the cost of maintaining these resources has increased to compensate. The earliest example of compute virtualisation is from 1967 with CP-67 on the IBM System/360. This was to allow users to have their own instance of CMS, which was a simple single uesr interactive OS.

Virtualisation allowed for the sharing of resources while protecting those resources from exploitation. It wasn't until the early 2000s and the rise of x86 servers that virtualisation become mainstream.

Isolation of Workloads

A workload can be defined as some sort of computation, from generating an HTML page to running payroll to computing a climate model. Isolation is a measure of surface a workload exposes to the outside environment.

Isolation improves security and reliability at the cost of convenience and potentially performance. All computing systems have resources, such as CPU cycles, memory, I/O performance, network access or storage space. Resources are almost never fully utilised, leading to wasted capital. The ability to pool resources can dramatically improve utilisation, but at the cost of reducing isolation. They can also lead to reduced performance of an single workload.

Spectrum of Isolation

Workloads are isolated, or pooled, for many reasons. In this spectrum virtualisation plays an essential role, covering a wide range of middle ground. Below is a rough spectrum.

1. Fully air gapped system.
2. Bare metal server on network.
3. Hardware partitioned system (LPAR).
4. Hardware assisted virtualisation with hypervisors.

5. Solaris Zones.
6. Containers.
7. Jailed environments.
8. Memory protected processes.
9. Multi-threaded processes.
10. Single MPP workload.

The biggest workloads

The biggest workloads in the world are either cryptoanalysis or scientific. In the realm of science it is either massive simulation, such as forecasting or climate modelling, or massive data, such as the LCR [Large Hadron Collider] or SKA [Squiare Kilometre Array]. In these cases a single workload can run on thousands of CPUs connected by low latency interconnects (such as Infiniband) running RDMA (Remote Direct Memory Access).

These servers often don't run full operating systems but a bare minimum supervisor running a single application.

Processes and threads

The unix model (and similar for other operating system) uses hardware assisted virtual memory to isolate a running process' memory from other processes. Virtual memory also protects the kernel from the running process. With multi-threading a single process can have multiple threads of execution. There is no isolation between threads.

Jailed environments

I put jailed environments ahead of countainers as chroot jails have been around a lot longer. In a chroot environment (in unix) a process no longer trees the original file system tree, instead its "root" file system is a directory and everything underneath that. For processes that allowed access to the whole file system to the world (such as anonymous FTP) this stopped it from seeing all world readable files.

Another jailing method is seccomp, where a process, even running as UID 0, cannot make certain system calls. This helps stop a process, running as UID 0 but in a jail, from breaking out if it compromised.

Containers

Linux containers are not a single technology. They consist of isolation via namespaces, resource limitations through control groups, packaging via docker or OCF (Open Container Format), container runtimes (docker, lxc, lxd, runc, jailer and others). On top of these are container orchestration platforms, such as dockerswarm and kubenetes.

Name spaces

In Linux namespaces allow a process to have a different view of resources from another process. The chroot environment is an example of a filesystem namespace. Other namespaces include PID, so a process no longer has a globally unique process ID, but instead is only locally (within the namespace) unique. This enables a process to have PID 1, which is significant within unix. It also stops processes from seeing what else is running at the same time.

For isolation the network namespace is critical, as it creates a completely new network stack. This allows any process with that network namespace to bind to a port, so that multiple httpd processes could all bind to port tcp/443 without their configuration behind changed. There are many mechanisms to connect a network namespace to the global network, either by moving a physical or VLAN interface into the namespace, or using a Virtual Ethernet (veth) pair to connect into a linux or OVS bridge, to using a software network overlay.

Control Groups

Cgroups are a form of resource control, such as CPU or memory. A process may create children cgroups but can never escape from its own cgroup, creating a possible hierachy of cgroups. Cgroups allow a collection of processes to be treated as a single entity. This is important because in unix a child process, if it forks again then the parent dies, will reparent to PID 1. This is often seen as a feature to allow a process to daemonize, but at the cost of it being no longer connected to the process that spawned it. With cgroups a reparented child still belongs to parent's cgroup so retains any resource allocations or contraints.

Running a "Full Fat" OS as a container

With PID namespaces it becomes possible to run a complete copy of Linux as a container. However because it sharing the kernel with the container runtime and requirement to allow any process within the container full access to the running kernel it not considered to be safe.

Solaris Zones

Isolation in the Extreme

A computer not connected to any network is known as air gapped, and is used for keeping ultra secure data (actually ultra secure should really only be the encryption keys, which are kept on tamper proof hardware and can never be read). In general air gapped computers are not very useful but can be considered to be the maximum in isolation.

OS running on hardware

This was the normal state of affairs before (x86) virtualisation, where the OS ran directly on the CPU and full control of all the hardware. This is known as bare metal running. Until the early 2000s this was the only way an OS could run on x86 hardware.

Emulation and Simulation

Emulation is where some functionality (in terms of performing some computation) is being done by not by the original hardware or software but another piece of hardware or software pretending to be that. Turing Completeness guarentees this is possible and critical to virtualisation, especially for older software.

Simulation is a form of emulation, but often seen as running at a much lower level. Emulation replicates the function (say of a SCSI controller) by replicating its behaviour while simulation would run the original SCSI controller logic at the logic gate or even transister level.

While it has always been possible to emulate a platform with perfect fidality in software the overhead involved limited this to niche cases where the original hardware was no longer viable.

Xen and Paravirtualisation

Starting in 2003 the Xen project worked on virtualising Linux on the x86 platform. At the time the x86 did not have the necessary support to run virtual workloads efficiently so the idea was not to emulate a fully virtual platform but to have guest operating systems aware that they were running in a virtual environment and offload hardware functions back to the hypervisor using hypercalls. This became Xen Paravirtualisation (PV) and required the guest OS to be PV aware. Ths worked well for Linux and gave good performance despite the lack of hardware support, but required custom linux kernels to work.

Because the guest kernel is aware that it is virtualised the hypervisor did not need to pretend to be an x86 server, instead it presented I/O devices via the XenBus protocol. Instead of trying to emulate say an LSI SCSI controller or an E1000 NIC over a PCI bus Xen presents a simple block device for disk and a simple virtual NIC for Ethernet. Because all of the hardware is paravirtualised there is no need for any emulation. To avoid issues with software expecting the hardware to be there, such as the BIOS or GRUB, Xen has a GRUB interpreter which then loads the kernel and calls a special Xen entry point. A fully PV environment has no BIOS, SMBIOS, ACPI, PCI bus or emulate any historical hardware such as the PIC, APIC, PIT or keyboard controller. All I/O is done using PV drivers over the Xen bus or with hypercalls.

The downside of PV is it cannot virtualise an OS that is not PV aware, and adds a considerable amount of code deep within the linux kernel (since critical

operations such as virtual memory and task scheduling need to be done by hypercalls).

Hardware assisted virtualisation on x86

In 2005 Intel released its first CPUs with explicit support for virtualisation with the Intel VT-x instructions. Until then virtualisation required either PV or full hardware emulation (with its performance penalties). with hardware assist full OS virtualisation was possible with limit performance penalties.

Most Operating Systems expected to be running on baremetal so hypervisors need to emulate various platform components. This includes legacy hardware such as the PIT, PIC, i8059 keyboard controller, IDE controllers. The emulation has to be good enough to fool a BIOS into thinking it running on bare metal.

QEMU and KVM / Xen HVM

The standard linux kernel supports virtualisation through the KVM [Kernel Virtual Machine] module. KVM has two sub modules, kvm-amd and kvm-intel, to perform the necessary architecture specific operations. A process (called a Virtual Machine Monitor or VMM) creates a new virtual machine via ioctls to /dev/kvm. KVM handles all virtual memory and CPU scheduling operations within the kernel itself, trapping any I/O operations (either PIO or MMIO) and exiting back up to the user space VMM to handle.

QEMU is a fully functioning emulator. It has two full x86 hardware models (one based on the i440fx chipset from 1996 and the Q35 chipset from 2007). In addition is can emulate a reason number of devices, and supports a wide range of file formats and storage and network backends.

In addition QEMU can emulate another CPU architecture entirely, so it possible to run an ARM or RISCV virtual machine on an x86 CPU (at a considerable performance cost).

When hardware virtualisation became available the Xen project created a new model called HVM [Hardware Virtual Machine]. This allowed it to run unmodified guests (such as standard Windows or other OSes that didn't support PV) at the cost of using QEMU to emulate a bare metal server. To improve performance Xen added PV device support to improve I/O and network performance (HVM+PV).

QEMU/KVM (QEMU on linux has support for using KVM to offload the virtualisation and just does any emulation required) is now seen as the defacto standard for virtualisation on linux.

PV Devices and OS Catchup

To avoid the high overhead costs to emulating physical devices such as NICs and SCSI HBAs hypervisors created paravirtualised devices. For ESX there is the

vmxnet3 NIC and the pvscsi HBA. For the NIC it easy to add to an existing OS but for the HBA it required the driver be included into the base OS installation before it was wide adopted (it is possible to include a non standard SCSI driver during OS installation but involved putting the driver onto a floppy and hoping the installer would pick it up correctly).

While linux has had PV devices since the early days of Xen for KVM it has only been (relatively) recently that the virtio standard has been adopted for PV devices. This is now widely supported (avaiable but not standard for Windows).

Beyond the basics

For full OS virtualisation all hypervisors (ESXi/HyperV/Xen/OVM/KVM) all offer basic services.

- Hardware virtualisation
- Virtual disks on local filesystem/datastore
- Networking via virtual bridges

With a single hypervisor the additional features are common.

- Suspend/restore an VM
- Snapshot VM state or disk
- Guest introspection
- Clone a VM
- Performance reporting
- Resource allocation

Some of the most important additional features of virtualisation come from the use of multiple hypervisors working in concert.

- Shared storage backend
- Live migration
- Distributed configuration
- Centralised management