Virtualisation

billy.trend@gmail.com

October 2014

Abstract

This essay examines the usefulness of virtual machines as a solution to computing problems. It explores the difficulties of their implementation and how these difficulties may be overcome. Finally, It looks forward to how virtualisation might develop in the future.

1 Introduction

In computer science, the term virtualisation describes the art of virtualising whole computers. That is to say developing an abstraction layer that gives a process or full-blown operating system the illusion that it has exclusive use of the resources - memory, CPU and I/O - that are presented to it, just as though it were running on its own dedicated hardware.

There are several software implementations that offer this layer of abstraction. They are known collectively and synonymously as hypervisors or virtual machine monitors. Broadly, there are two types of hypervisor. One type (show left in figure 1) runs as a basic operating system, directly on hardware and has administration facilities for managing its guest virtual machines. This type may be used in data centres that host 'virtual private servers' where the hardware's only required function is to manage and run these virtual processes. They are associated with good performance since the virtualised process is run 'close' to the hardware.

The second type of hypervisor (show right in figure 1) virtualises machines from within a 'host' operating system. This might be used in a development environment for simulating a device or server. These hypervisors are popular among users that need to use applications which require a different operating systems to their primary systems. 'Parallels' is a hypervisor that allows users to run Windows programmes on their Mac.

In general, the virtual machines are known as 'guests' and the OS that is running directly on the hardware is the 'host'.[11]

2 Advantages of virtualisation

Superficially, it might seem that virtualisation is a poor way of managing resources: every approach to virtualisation involves necessary overhead that adds no more functionality to an OS or process that it would have if it was running natively on its own hardware.[15]

However, the reverse is true. In many scenarios virtualisation makes managing resources easier and more efficient. Take, as an example a number of web services. They all have conflicting dependencies such that they cannot be deployed on the same system.

A concrete example of this might be a school that has some databases, timetabling software, an email service and some file stores. These might each depend upon a different java version to another; one might be tied to Windows XP while the others only function on a Unix OS. A solution that does not

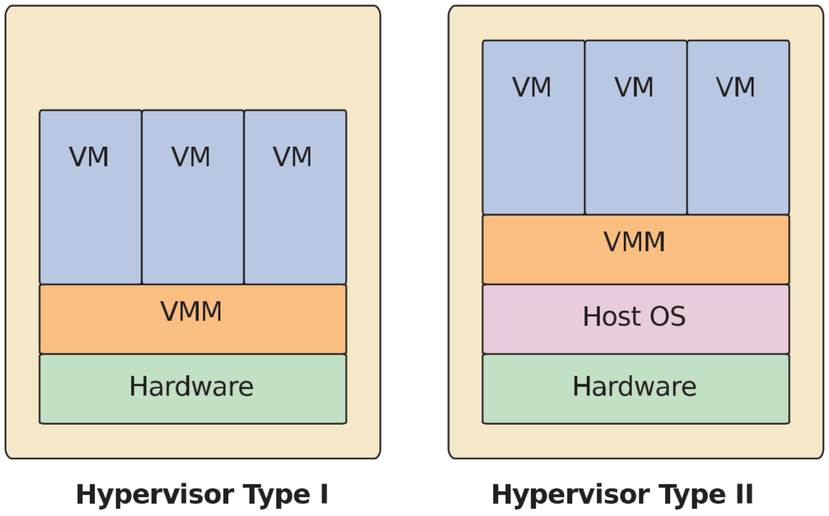

Figure 1: Types of hypervisor.[11, p. 270]

use virtualisation would require a server box for each service, each running a different configuration to suit.

This would have a number of disadvantages. Each piece of hardware would be under-utilised since these sort of web services do not require much processing time or memory especially in a small organisation. This would lead to unnecessary power use and high cost of purchase and maintenance of each machine. Scaling the services would be hard since each deployment is tied to its physical server box. If a service was in demand, a new physical server would have to be spun up manually - even though the other CPUs would likely still be running at nearly idle.

Virtualisation is an efficient solution to this. A hypervisor can manage all these systems virtually. This allows many services to exist on the same hardware without conflicting. Administrators can also move, copy, snapshot and backup virtual machine instances remotely. If a service becomes under increased demand, it can be duplicated to another server and placed behind a load balancer. This can all be done with out physical access to servers.

Since this sort of administration is so straightforward to do remotely, there is a growing trend towards hosting these virtual machines in the 'cloud'. Cloud is a buzzword for centralised IT services, including Software as a Service (SaaS), Platform as a Service (PaaS), Virtual Private Servers (VPS) and database as a service (DBaaS). All of these could be used in the scenario above. VPS services expose all the functionality required to run web services but without the complexity of managing the virtual servers. Many give simple interfaces which allow an administrator to scale a service to meet increased demands;

turn off a service but keep a snapshot should it need to be turned on again; and move services geographically between datacentres. These centralised solutions provide financial benefits since there is no initial cost of buying hardware or to scale it. They mean that organisations do not have to commit space, power and staff to datacentres.

Amazon Web Services (AWS) exemplify how virtualisation can be used to develop efficient and profitable usage of computer resources. Virtualisation allows Amazon to quantify its compute power and sell it. It is estimated that Amazon use the open source Xen hypervisor to manage virtual machines on nearly half a million physical servers.[7] These run the AWS databasing, processing and general purpose applications.

3 Disadvantages of virtualisation

The strategy of putting all your services on one piece of physical hardware can have very obvious benefits especially when your services have low resource utilisation and the hardware is expensive. However, it does not come without disadvantages. David Coyle, at the Gartner's Infrastructure, Operations and Management Summit, laid out several problems that are introduced when systems are virtualised. [1]

If a physical server running many virtual servers breaks down, all the virtual servers will go down with it. The benefit of having physical machines is that the chances of all your services going down are lower. A NEC white paper describes this problem as a 'single point of failure' and suggests that virtualisation makes a 'continuity solution' essential. [9]

It is less possible to predict the performance of a virtual machine. While it is more straightforward to ring fence a fixed quantity memory for each process, it is harder to give consistent CPU performance. This is corroborated by a 2006 paper on interference effects in virtual environments: performance interference may occur when multiple virtual machines are running CPU intensive processes.[6] Since virtual machines are isolated, it is impossible for them to predict when their CPU performance may drop. Hypervisors can move machines with high CPU utilisation to less utilised hardware but this is not instantaneous.

While hypervisors make deployment and management of virtual servers easier, the system as a whole is arguably more complex than a purely physical setup. Debugging can be made harder. Problems can no longer be isolated to one physical server box. This means that more knowledgeable and costly administrators are required. For economy, this cost must be balanced by the savings in computer hardware investment.

4 Implementation of virtual machine monitors

In 1974, Popek and Goldberg laid out the aims of their virtualisation project with the following three rules.

'As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.' [10]

This applies strictly to virtualisation where the guest OS matches the host OS. This was the only form of virtualisation under development at the time. More recently, virtualisation solutions are available to virtualise arbitrary guests on top of arbitrary hosts. However, the principals may still be roughly applied.

Security is an important feature of virtualisation, isolation of guest operating systems must be such that they cannot interfere with or access the data or namespaces of other guest systems. This is particularly important in a platform such as AWS where guest systems might be owned by mutually untrusting parties. Digital Ocean - a low cost VPS provider, similar to AWS' EC2 - came under fire when they neglected to scrub (zero over) a virtualised instance's storage space by default after it was destroyed. This meant that other instances that were created subsequently by any customer could read fragments of the storage of previously-destroyed instances. This made it possible to piece together private information and keys. [5]

4.1 Challenges

For clarity, I will initially limit my account of the technical challenges and implementation of virtualisation to that of processes and operating systems that share the common denominator of being built for the x86 instruction set. This includes the main desktop and server systems: Linux, Mac and Windows.

Conceptually, virtualisation is fairly easy to grasp at a high level; it can be diagrammed with neat, clear divisions between host systems, the hypervisor and guest systems. In practice, there are significant technical challenges to successfully virtualise a system.

It can be desirable for a hypervisors to support operating systems without modification. Most systems are designed and built primarily to run directly on hardware. For this reason, a hypervisor must ensure that a system has access - or appears to have access - to all the architectural features that it would if it were running directly on physical hardware. At the same time it must ensure that guest operating systems don't interfere with each other or the host operating system. There are some instances where these two constraints come under contention.

4.2 Protection rings

Hardware architectures implement 'protection rings'. These securely manage how processes use available computer resources such as memory, CPU and IO. Processes operate within different protection rings depending on how privileged their access to hardware needs to be. The kernel is generally an operating system's lowest level process. It handles base functionality such as memory management and system calls. This means it must operate in the most privileged ring - ring 0. Instructions permitted in ring 0 may access protected areas of memory and define procedures for interrupt handling.[14, p. 30-32]

A compromise must be made here. A hypervisor cannot allow guest processes to directly execute ring 0 instructions because they would break the isolation of each process, causing security and practical issues. However, since an unmodified process expects to be able to be able to use ring 0 features, a hypervisor must be able to emulate them.

4.3 Memory Virtualisation

Memory protection is covered by largely protection rings, loading and storing of memory needs be virtualised. Memory virtualisation is built into most systems already: processes are given the impression of a contiguous set of memory locations which have a constant size and are directly accessible even though some may be actually cached. These facts mean that memory virtualisation is less complex than implementing protection rings. All modern CPUs have silicon dedicated to memory access making it more efficient. [16, p. 6]

5 Solutions

5.1 Binary Translation and Full Virtualisation

Binary translation is a 'heavyweight' emulation technique. It isolates a guest OS by directly translating its instructions to those of the host architecture. Binary translation is an effective virtualisation technique but the translation process is done just in time like an interpreter for a language like python. This provides considerable overhead. For an x86 guest running on an x86 host, this overhead is redundant since most of the time, translation process simply converts x86 commands that operate at lower privilege than ring 0 to x86 commands with the same privilege.

In these cases, virtualisation can be made more efficient by using binary translation only on the subset of instructions that require privileged 'ring 0' execution. User level code is executed directly on the host CPU. This means that most of the operations of the guest d system have similar performance to those of the host OS. This approach is known as full virtualisation. [11]

Full virtualisation is sufficient to fully decouple Host and Guest systems. That is to say the guest process has no idea it is being virtualised. According to VMware it is the best virtualisation method for isolation and security. Since it supports unmodified operating systems, it also offers the best in migration and portability. A system may be moved from a virtualised environment to native hardware without any special adaptation.[16, p. 4]

5.2 OS Assisted Virtualisation or Paravirtualisation

Paravirtualisation requires that the guest operating system is modified to remove any functionality which requires privileged instructions and that cannot be virtualised. These are replaced by 'hypercalls' which communicate directly with the host hypervisor. Hypercall interfaces handle all the functionality that is usually only accessible in ring 0.

The argument for paravirtualisation is that it has lower overhead than the binary translation solution. It does however introduce maintainability issues since the hypercall implementation must be updated as the guest operating system is updated. It is easier to develop an effective hypercall interface than it is to perform binary translation reliably. CPU paravirtualisation cannot be implemented without vendor support by unmodifiable (closed source) operating systems such as Windows since the source code cannot be changed at a kernel level. Higher level features such as graphics card virtualisation can be supported by installing appropriate drivers to the host OS. This is a good compromise since it offers performance increase without needing to modify the kernel. [16, p. 5] Hypervisors such as virtual box can install these drivers by mounting a virtual disk to the virtual machine.

5.3 Hardware Assisted Virtualisation

Since protection rings are implemented at a hardware level, it makes sense that hardware vendors have taken steps to solve the problems that they create for virtual machines. A new root privilege ring below ring 0 allows hypervisors to run at higher privilege to the guest operating systems so that the guest systems may execute privileged instructions directly to the hardware.

The guest operating system's state is stored in the extra silicon implemented by hardware vendors. Depending on which process is running privileged instructions, this state is switched between the operating systems and the hypervisor thereby preventing conflict.[16, p. 6]

6 Virtualising devices and I/O

In principal, any system can be run as long as there is a way of storing data, such as memory, and a way of processing data, a CPU. Up until now, I have provided an account of how such processing of instructions and memory may be virtualised. However few operating systems work in this sort of isolation. Modern day applications require hardware services such as network interfaces, hard drives and graphics processors to name but a few.

Hypervisors can emulate these peripherals. In some cases this is simple: if the emulated hardware is roughly analogous to the available hardware. Virtual box for example, offers 6 types of virtual network cards. Some emulate physical AMD cards, some Intel and one is paravirtual card for improved performance. Virtual box translates the guest operating systems commands into network requests and offers various modes of connection to the host and outside world. Sometimes however, it is not so simple.

The Dolphin-Emu project attempts to fully emulate nintendo games consoles for PCs. The target architecture for the software on most games consoles is very different to many PCs, since games consoles tend to use optimised hardware which provides performance, power and cost benefits. The Wii is built for the PowerPC instruction set which is interpreted dynamically by dolphin through binary translation. There are, however, more significant challenges in emulating Wii architecture.

A particular area of focus for the Dolphin project was emulating the 'Flipper' and 'Hollywood' GPUS for the Nintendo Gamecube and Wii respectively. Both GPUs use integer maths in their pipelines which is unlike common PC GPUs which use floating point maths. Operations such as those involving integer overflow must be handled especially. For example, when you add 1 to an 8 bit integer of value 255, it should overflow to 0. To emulate this behaviour on a floating point GPU, the following code may work.

```
frac(value * (255.0/256.0)) * (256.0/255.0)
```

This line of code essentially maps the integer 'value' to a floating point number between 0 and 1. If the integer is lower than 0 or higher than 255, the result is an overflow, still between 0 and 1. However, this code has many edge cases and often intermediate steps compromise performance. An example on the Dolphin-Emu blog post compares two numbers 49.999 and 50.003 which equivocate on the wii integer math GPU but do not on floating point GPUs. This - it claims - is enough to make the difference between an object appearing on screen or not.

The area of GPU emulation is problematic since there are not standard instructions sets; vendors implement their own interfaces and provide drivers for different platforms. This has made things difficult for the dolphin project where a fix for one GPU architecture may cause a bug in another. [8]

7 Targeting other instruction sets

The solutions discussed up to now only work when the host and guest processes are both built to use the x86 instruction set. This is a fair constraint since most desktop OS use x86 because it is supported by the leading CPU manufacturers.

It is worth exploring however how it is possible to virtualise or 'emulate' processes with alternative instruction sets on an x86 platform. It cannot be simply a question of installing drivers to the guest OS since instruction sets underpin even drivers.

The problem is usually solved using an involved form of binary translation which translates every instruction to the target architecture. This technique is employed by the QEMU. QEMU provides full CPU emulation of guest instruction sets x86, x86-64, ARM, SPARC, PowerPC, MIPS, and m68k for host architectures x86, x86-64, and PowerPC. [11]

It is important to have this as an option since there may be legacy OS which require different architectures but nonetheless still have uses. A good example of where emulation is useful is in running mobile device emulators (often built for ARM chipsets) on PC hardware for development purposes.

8 The Future of Virtualisation

8.1 Industry development

Large companies necessarily have the biggest virtual server deployments and therefore are the strong indicators of future development of virtualisation.

Ravello, a company which helps other companies use cloud technologies to improve their businesses, has a blog post on its interests in the future of virtualisation. They describe a paradigm in which not just single machines are virtualised but whole hardware stacks. These would be portable across arbitrary hosts. For example, you could deploy whole virtual datacentres consisting of many virtual machines and networking appliances. This would allow easy management of complex services.[4]

In a presentation for VMware in 2010, Larry Rudolph shared his view of virtualisation's long-term future. He envisages an intelligent hypervisor that moves virtual machine instances to appropriate pieces of hardware based on their requirements and usage at any given moment. For example if a VM is using a significant amount of disk I/O, it may be moved to an instance which has access to fast solid state drives. If it needed more compute power, it might be migrated to a super computer or a server with a powerful graphics processing unit. These migrations would happen seamlessly with minimum downtime.

In the same presentation, Rudolph describes a more consumer-focussed usage of virtualisation. As imersive 3d technologies continue to develop, virtualisation may allow us to carry our technology between the real world and simulated worlds. Our mobile phone might appear exactly the same to us in a game such as second life as it does in our real life. [13]

8.2 Containerisation

Disruption is a well established business process in the computer industry. It is when small startups undercut whole industries using alternative solutions. It is therefore shrewd to identify up and coming alternatives to industry practices that could be described as dogmatic. Containerisation - though already over a decade old - is the best example of this in the virtualisation space.

Containerisation includes the two of the main advantages of virtualisation at the technical level. One is the isolation of namespaces such as files, network interfaces, inter-process communication pipes and process ids. The other is the allocation of physical resources such as memory, It brings about the benefits of easy dependancy management, reduced chance of conflicts between services and flexible limits on memory usage of services.

Containers provide this virtualised execution environment without the overhead associated with virtual machines. They rely on the unix kernel features, namespace and cgroups to enact resource and namespace isolation. The benefits of this are the reduced startup and shutdown times for containers since they are not full-blown operating systems. Containers are limited by the host OS: you cannot run a ubuntu container on OS X for example. [12]

Containerisation has been popularised recently by an open source project called Docker. Unlike a hypervisor which is executed as a software layer or interface with the host computer, throughout the existence of a virtual machine in its execution, Docker takes a back seat, managing only the creation and deployment of containers. The company that developed Docker has received 50 million dollars in funding and is in production in enterprise [2]; namely at Spotify, eBay and Yelp[3].

9 Conclusion

In this essay, I have explored the various solutions to virtualisation and the motivations for its deployment. Virtualisation is likely to continue to be a very relevant focus of research. While it is widely used, none of the implementations mentioned in this essay have won out as an undisputed 'best practice'. With technologies such as Docker gaining momentum, it is likely that there will continue to be diversity in the virtualisation space. This reflects other areas of technology; databasing for example has a similar diversity of roughly analogous services with many different implementations.

Despite the technical challenges presented by virtualisation, it is likely that continued improvement in hardware and software support will cause solutions to become more efficient, fault tolerant and easy to use.

Words: 3686

References

[1] David Coyle. *7 Side Effects of Sloppy Virtualization.* URL: `http://www.networkworld.com/article/2281117/data-center/7-side-effects-of-sloppy-virtualization.html?page=1` (visited on 10/09/2014).

[2] Crunchbase. *Docker.* URL: `http://www.crunchbase.com/organization/docker` (visited on 10/09/2014).

[3] Docker. *Use Cases.* URL: `https://www.docker.com/resources/usecases/` (visited on 10/09/2014).

[4] Geert Jansen and Alex Fishman. *The Future of Virtualization.* 2013. URL: `http://www.ravellosystems.com/blog/the-future-of-virtualization/` (visited on 10/09/2014).

[5] Sean Michael Kerner. "Scrubbing Data a Concern in the Digital Ocean Cloud." In: *eWeek* (2014), p. 14. ISSN: 15306283. URL: `http://search.ebscohost.com/login.aspx?direct=true\&db=buh\&AN=93513807\&site=eds-live`.

[6] Younggyun Koh et al. "An Analysis of Performance Interference Effects in Virtual Environments". In: (2006). URL: `http://www.neotextus.net/wp-content/uploads/2013/03/ispass07.pdf`.

[7] Huan Liu. "Amazon data center size". In: 2013. URL: `http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/`.

[8] MaJoR, JMC47, and Neobrain. *Pixel Processing Problems: On the Road to Pixel Perfection.* URL: `https://dolphin-emu.org/blog/2014/03/15/pixel-processing-problems/`.

[9] NEC. "Integrated System Continuity Solutions for Virtual System Consolidation". In: (2008). URL: `http://www.nec.com/en/global/prod/expresscluster/en/collaterals/pdf/WP\_virtual\_system\_continuity\_nec\_en.pdf`.

[10] Gerald J. Popek and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures". In: *Communications of the ACM* 17.7 (July 1974), pp. 412–421. ISSN: 00010782. DOI: `10.1145/361011.361073`. URL: `http://portal.acm.org/citation.cfm?doid=361011.361073`.

[11] Fernando Rodríguez-Haro et al. "A summary of virtualization techniques". In: *Procedia Technology* 3 (Jan. 2012), pp. 267–272. ISSN: 22120173. DOI: `10.1016/j.protcy.2012.03.029`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S2212017312002587`.

[12] RAMI ROSEN. "Linux Containers and the Future Cloud." In: *Linux Journal* 240 (2014), pp. 86–95. ISSN: 10753583. URL: `http://search.ebscohost.com/login.aspx?direct=true\&db=buh\&AN=95781759\&site=eds-live`.

[13] Larry Rudolph. *Future of Virtualisation.* Tech. rep. Ravello, 2010, http://www.ravellosystems.com/blog/t
future–of–v.

[14] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts essentials / Abraham Silberschatz, Yale University, Peter Baer Galvin, Corporate Technologies, Inc., Greg Gagne, Westminster College.* Hoboken, NJ : Wiley, 2014., 2014. ISBN: 9781118804926. URL: `http://search.ebscohost.com/login.aspx?direct=true\&db=cat00290a\&AN=sta.b2096124\&site=eds-live`.

[15] Amit Singh. *An Introduction to Virtualization.* 2004. URL: `http://www.kernelthread.com/publications/virtualization/` (visited on 10/09/2014).

[16] VMware. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist.* Tech. rep. URL: `http://www.vmware.com/files/pdf/VMware\_paravirtualization.pdf`.