Comparison of Virtualization Techniques in Distributed Cloud Environments

Lauri Suomalainen

Master's Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, December 19, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science

Tekijä — Författare — Author Lauri Suomalainen
Työn nimi — Arbetets titel — Title Comparison of Virtualization Techniques in Distributed Cloud Environments
Oppiaine — Läroämne — Subject Computer Science

Työn laji — Arbetets art — Level Master's Thesis	Aika — Datum — Month and year December 19, 2016	Sivumäärä — Sidoantal — Number of pages 8

Tiivistelmä — Referat — Abstract

Abstract

Avainsanat — Nyckelord — Keywords
Virtualization, Distributed Systems, Containerization

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

The usage of computational devices and services has seen radical new trends in the 21st century. The number of different devices keeps growing and the prevalence of smartphones is nothing short of ubiquitous [8]. With the emergence of Internet of Things devices, the amount of different computers at our disposal is a stark contrast to recent past when there was only one PC per household. In addition to quantity of devices, the way they are used has also changed. Devices are mobile and have sensing capabilities. Applications and storage space are no longer tied to personal hardware but served from cloud services.

In cloud computing, virtualisation is essential for delivering services to a large userbase. Virtualisation enables the cloud provider to share their limited hardware resources optimally among users, balance workloads and improve fault tolerance.

2 Background

In cloud computing, there are multiple recognized service models which dictate how the users can use the given system and what privileges they are given [16]. In its most limited form, a cloud service is offered to user as a predefined application or a set of applications. The users has some interface for interacting with the applications but is given no control over anything else such as other applications, the operating system the application is running on or network and hardware configurations. This is generally known as Software as a Service or SaaS for short. The most permissible service model is known as IaaS, Infrastructure as a Service. In its archetype the user gets access to all fundamental computing resources, possibly including some network components, and can run arbitrary software including operating systems. The user experience should be similar to that with their personal computers. The user is not allowed to access the underlying cloud infrastructure. Between the two falls Platform as a Service (PaaS). PaaS typically allows user to deploy their own applications along with their dependent libraries, tools, services etc. provided that they are supported by the cloud provider. The user has no control over underlying cloud infrastructure, operating system, storage or network but usually can configure certain settings and possibly choose different supporting services the cloud provider offers. There are also other "aaS" such as Data as a Service (DaaS) and Storage as a Service (SaaS) but they are based on one of the three aforementioned service models or are variations or subsets of them. Sometimes the numerous models are referenced with umbrella terms of XaaS and EaaS meaning Everything as a Service for both or Anything as a Service for the former [12].

Virtualisation in the context of distributed cloud environments usually refers to virtual machines. The core idea is analogous to computer hardware virtualisation. Operating systems offer an interface for the processes to utilize the computer hardware while giving them an illusion that they have all of the hardware for themselves [9]. In reality the resources are shared among many processes. Likewise in cloud environments resources are being share by processes but also by different users running different operating systems, configurations and programs. As with the processes, users are given an impression that they alone have access to the underlying hardware resources, whereas in reality there are multiple users using the same physical machines.

There are several reasons as to why would one prefer a virtualised environment to a non-virtualised one. Obviously in multitenant cloud services it is crucial for the service provider to maximise the use of their hardware resources. Thus it is imperative for the provider to try to share the limited hardware resources among as many users' virtualised environments as possible. Otherwise every user would need their own physical machine in the system which would both require more resources per user and leave resources underused. From the fault tolerance perspective, using virtual machines in a distributed environment decreases their dependency on the underlying physical hardware [10]. That is because in virtual machine architectures which support live migration of operating system instances can be seamlessly moved from one physical machine to another. This also helps the load-balancing in the distributed system and allows low-level and physical maintenance of the hardware without considerably disrupting the usage of the system. A end-user also has many reasons to use virtualised cloud services. User only needs a lightweight computer with an internet connection to perform computationally challenging tasks in the cloud back-end. Similarly devices with little storage capacity can leverage from a cloud service's vast storage space. Some users would like to use applications and programs not native to their operating system of choice making another virtualised OS a convenient option [9]. Virtualised environment allows software developers to test and debug their software with many different settings, as virtualised environments can have different operating systems and available hardware resources. Naturally this also allows emulating completely different devices [13].

2.1 Virtualisation Techniques

Traditionally virtualisation has referred to a software abstraction layer residing between the computer hardware and the operating system. [17] This layer has been called Virtual Machine Monitor (VMM) or more recently a hypervisor and it hides and abstracts the computing resources from the OS, allowing multiple OSs to run simultaneously on the same hardware. There are multiple ways to run hypervisor-based virtualisation. Lately a technology called container-based virtualisation has been gaining popularity. Instead of emulating whole hardware, containers make use of features provided by the host operating system to isolate processes from each other and other containers [13].

2.1.1 Full virtualisation

In full virtualisation, the hypervisor runs on top of the host OS. The guest OSs run on top of the hypervisor which in turn emulates the underlying real hardware to them. The hypervisors running on top of the host OS are generally referred as *Type 2 Hypervisors* [13]. The guest OSs can be arbitrary. Figure 1 shows the full virtualisation architecture with the hypervisor running on top of the Host OS and Guest OSs on top of the hypervisor using their emulated hardware.

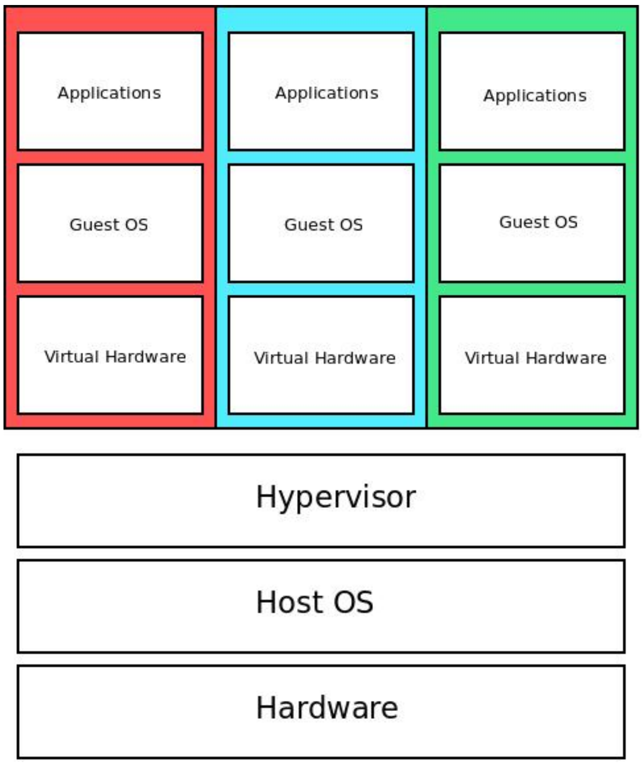

Figure 1: Full virtualisation architecture

The main advantage of full virtualisation is that it is easy to deploy and should not pose problems to an average user but the virtualisation overhead results in significantly reduced performance when compared to running directly on hardware. Popular examples of full virtualisation applications are Oracle's *VirtualBox*[4] and *VMware Workstation*[6].

2.1.2 Hardware-Layer virtualisation

Hardware-Layer virtualisation is also a type of full virtualisation, but unlike Type 2 hypervisors, the so called *Type 1 Hypervisors* (also *native* and *bare metal*) run directly on hardware. As seen on figure 2 there's no Host OS per se. Instead the Guest OSs access to hardware resources is controlled by the hypervisor.

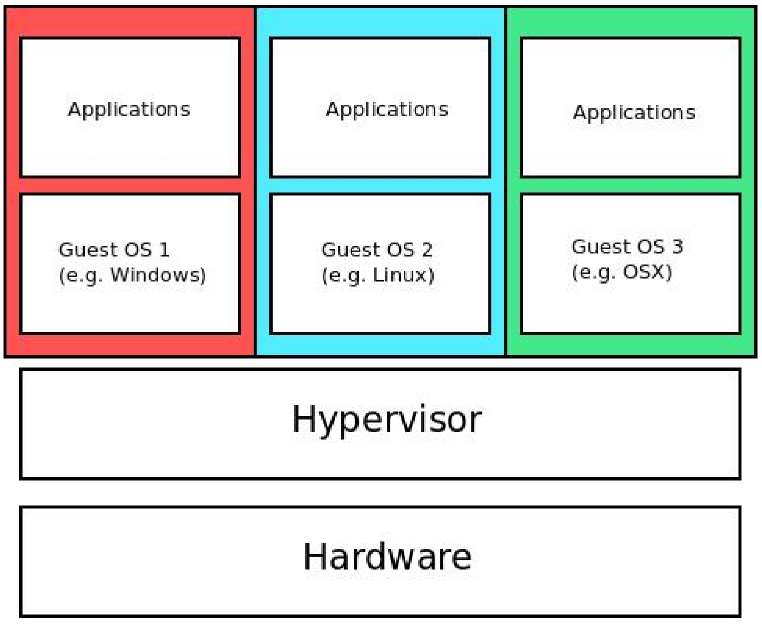

Figure 2: Hardware-Layer virtualisation architecture

Running directly on hardware, Hardware-Layer virtualisation techniques suffer less performance overhead than their OS-layer counterparts. On the other hand, Type 2 hypervisors being essentially applications themselves can be ran in parallel on the host OS whereas Type 1 hypervisors can not. For an average user, setting up a Type 1 hypervisor can be more difficult than Type 2. Commercial examples of Type 1 Hypervisors include Microsoft's *Hyper-V*[2] and VMware's *VSphere* [5].

2.1.3 Container-based virtualisation

Instead of virtualising the underlying hardware, container-based virtualisation also known as OS-Layer virtualisation [17] focuses on user space and allows running multiple operating systems in parallel as applications using the same kernel as the host operating system. A prime example of a popular container-based virtualisation platform is *Docker* [1] which leverages on native Linux kernel features to virtualise and isolate OS instances. Figure 3 shows a container-based virtualisation architecture in which containerised environments are running operating systems on host OS's kernel.

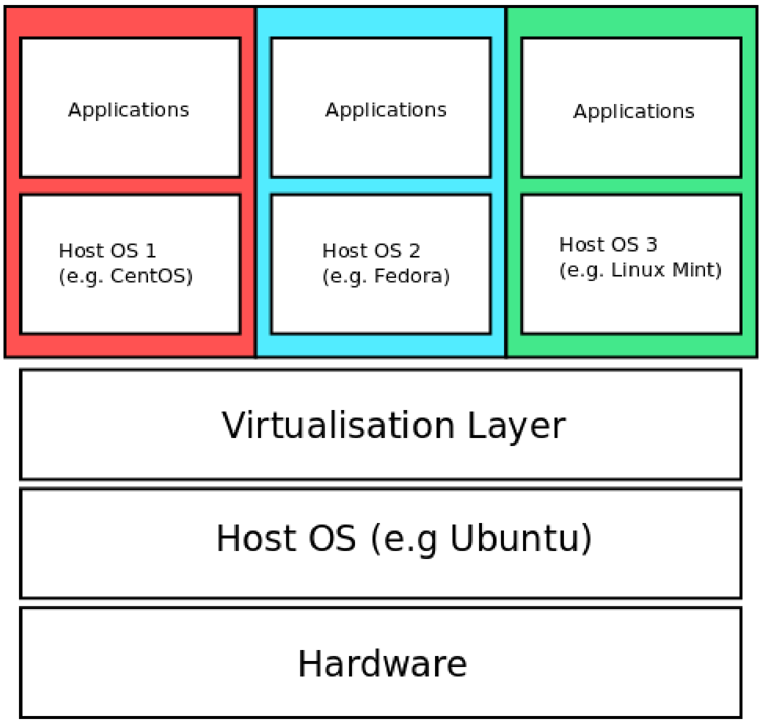

Figure 3: Container-based architecture

Container-based virtualisation does not need to emulate the hardware as containers communicate directly with the host kernel [13] and are thus very fast to start. They also do not require all of the components a fully virtualised environment would need to run and therefore their resource fingerprint is minimal when compared to hypervisor-based virtualisation techniques. The obvious drawback of the technique is that the kernel of the virtualised OS has to be the same as that of Host OS e.g. In a situation depicted in figure 3 operating systems based on Linux kernel could be ran on Ubuntu Host OS but OSs like Windows or OSX could not. On certain virtualisation platforms resource-intensive containers can also affect other containers detrimentally as the shared host OS's kernel is forced to spend its execution time on handling the instructions from the stressed container [18].

2.1.4 Paravirtualisation

Paravirtualisation differs from full virtualisation by requiring the Guest OS to be modified in order to accomodate the virtual environment in which it is ran. Otherwise the architecture is similar to that of full virtualisation, but with thinner hypervisor allowing performance close to that of a non-virtualised environment. A well-known example of a paravirtualisation hypervisor is *Xen*[7].

2.2 Unikernels

Unikernels are a relatively recent take on virtualising services. Building on the notion that in cloud environments each VM usually specialises to provide only one service even if each VM contains a full-fledged general computer [15]. Unikernels are essentially minimal single-purpose library operating system (*LibOS*)[11] VMs with a single address space. They contain only the minimal set of services, implemented as libraries, built and sealed against modification to run the one application. Unlike the earlier LibOSes unikernels do not require a host OS to run but run directly on a VM hypervisor, such as Xen.

Some benefits of unikernels are obvious. Constructing VMs with minimal set of service libraries results in small images and resource footprints as well as fast boot times. Other benefits include reduced attack surface due to smaller codebase and sealing preventing any code not compiled during the creation of the VM from running. Single-address space improves context switching and eliminates the need for privilege transitions making system calls as efficient as function calls [14]. Running directly on the hypervisor instead of a host OS eliminates superfluous levels of hardware abstraction.

Optimisation and simplification are not without drawbacks. By definition, unikernels are not intended for general-purpose multi-user computing but for microservice cloud environments. Running multiple applications on a single VM is risky due single-address space does not offer any inherent resource isolation. As unikernels are sealed during compiling, it is not possible to do changes to them afterwards. User is instead required to compile and launch a completely new modified VM.

Popular examples of unikernels are *MirageOS*[3] and *OSv*[14].

3 System design and implementation

4 Experiments and measurements

5 Related work

6 Future research

7 Conclusions

Sources

[1] *Docker*. `https://www.docker.com`. Accessed: 3.11.2016.

[2] *Microsoft Hyper-V.* https://technet.microsoft.com/en-us/library/mt169373(v=ws.11).aspx. Accessed: 7.11.2016.

[3] *MirageOS.* https://mirage.io/. Accessed: 19.12.2016.

[4] *Oracle VirtualBox.* https://www.virtualbox.org/. Accessed: 1.11.2016.

[5] *VMware VSphere Hypervisor.* http://www.vmware.com/products/vsphere-hypervisor.html. Accessed: 7.11.2016.

[6] *VMware workstation.* http://www.vmware.com/products/workstation.html. Accessed: 1.11.2016.

[7] *Xen project.* https://www.xenproject.org/l. Accessed: 7.11.2016.

[8] Anderson, Monica: *Technology Device Ownership, 2015*. 2015.

[9] Arpaci-Dusseau, Remzi H. and Arpaci-Dusseau, Andrea C.: *Operating Systems: Three Easy Pieces.* Arpaci-Dusseau Books, 0.91 edition, May 2015.

[10] Clark, Christopher, Fraser, Keir, H, Steven, Hansen, Jacob Gorm, Jul, Eric, Limpach, Christian, Pratt, Ian, and Warfield, Andrew: *Live migration of virtual machines.* In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI*, pages 273–286, 2005.

[11] Donald E. Porter, Galen Hunt, Jon Howell Reuben Olinsky Silas Boyd Wickizer: *Rethinking the library os from the top down.* In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* Association for Computing Machinery, Inc., March 2011. https://www.microsoft.com/en-us/research/publication/rethinking-the-library-os-from-the-top-down/.

[12] Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. C., and Hu, B.: *Everything as a service (xaas) on the cloud: Origins, current and future trends.* In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628, June 2015.

[13] Eder, Michael: *Hypervisor-vs. container-based virtualization.* Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 1, 2016.

[14] Kivity, Avi, Laor, Dor, Costa, Glauber, Enberg, Pekka, Har'El, Nadav, Marti, Don, and Zolotarov, Vlad: *OSv—optimizing the operating system for virtual machines.* In *2014 USENIX Annual Technical Conference*

(USENIX ATC 14), pages 61–72, Philadelphia, PA, June 2014. USENIX Association, ISBN 978-1-931971-10-2. `https://www.usenix.org/conference/atc14/technical-sessions/presentation/kivity`.

[15] Madhavapeddy, Anil, Mortier, Richard, Rotsos, Charalampos, Scott, David, Singh, Balraj, Gazagnaire, Thomas, Smith, Steven, Hand, Steven, and Crowcroft, Jon: *Unikernels: Library operating systems for the cloud.* SIGPLAN Not., 48(4):461–472, March 2013, ISSN 0362-1340. `http://doi.acm.org/10.1145/2499368.2451167`.

[16] Mell, Peter M. and Grance, Timothy: *Sp 800-145. the nist definition of cloud computing.* Technical report, Gaithersburg, MD, United States, 2011.

[17] Sahoo, J., Mohapatra, S., and Lath, R.: *Virtualization: A survey on concepts, taxonomy and associated security issues.* In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226, April 2010.

[18] Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and Rose, C. A. F. De: *Performance evaluation of container-based virtualization for high performance computing environments.* In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240, Feb 2013.