Virtualisation

Barry Denby

Griffith College Dublin

March 18, 2021

Virtualisation

- Virtualisation is the main reason why the cloud can scale applications rapidly
- All resources are virtualised: CPU, memory, storage, bandwidth etc
- Because they are virtualised it is easy to allocated and deallocate resources to applications
- In this lecture we will cover some strategies and technologies used to virtualise resources
- Lecture heavily based on Cloud Computing: Theory and Practice

Virtualisation

- Traditional datacentres relied on OSes to virtualise resources for their applications
- The problem here is that operating systems only virtualise resources belonging to a single node
- However in a cloud we need to virtualise resources for an entire network.
- For this we introduce a software layer beneath the OS called a hypervisor
  - piece of software or hardware that runs virtual machines
  - creates virtual versions of the node resources
  - will accept a virtual machine to run on top of those virtualised resources

Virtualisation: Advantages and Disadvantages

- Advantages
  - More than one virtual machine can be run on a node at any given time
  - Maximised use of resources
  - Virtual machine state can be saved and migrated to a different node and resumed there
- Disadvantages
  - Hypervisor consumes some CPU/memory in providing virtualisation
  - Virtual machines run at near native speeds if the instruction set of the virtual processor and real processor match (and processor supports virtualisation)
  - Provides an extra attack vector that must be secured

Virtualisation: Approaches

- We will discuss tow strategies for virtualising resources
- Full virtualisation: hardware abstraction provided by virtual machine monitor (VMM) is an exact replica of the physical hardware
  - no OS modifications are required
- Paravirtualisation: requires modifications of the guest OSes because the hardware abstration provided by the VMM does not support all the functions the hardware does
- As virtualisation has become popular CPUs have added hardware support for virtualisation

Virtualisation: Methods

- Virtualisation provides an abstraction or simulation of a resource
- Simulation is performed in one of four methods
- Multiplexing: for each resources to be virtualised, create multiple virtual objects. Grant access to the physical object fairly amongst the virtualised objects
  - e.g. one CPU being shared amongst many virtual machines
- Aggregation: The opposite of multiplexing. Multiple physical objects are combined together to have the appearance of one single object
  - e.g. a RAID array combining multiple disks into one single large area of storage

Virtualisation: Methods

- Emulation: Construct a virtual object from a different type of physical object
  - For example a virtual ARM processor running onto of a physical X86 one
  - This translation is expensive and slow
- Multiplexing and emulation combined: for example virtual memory in OSes

Virtualisation: Results of simulation

- Abstraction and simplification of resources
- Users are isolated from each other. i.e. one user cannot affect another
- Replication support: without virtualisation replication involves acquiring and initialising new hardware, OSes, and software stack
  - Replication in a VMM only requires a full copy of the VM
- System Security: Virtual Machines are not allowed to interfere with each other. They are sandboxed.
  - Restriction is strictly enforced by the VMM

Virtualisation: Results of simulation

- Performance and Reliability
  - should new faster hardware become available and the appropriate VMM is installed the VM can take immediate advantage of new hardware
  - Migration of a VM consists of suspending the VM, moving it to the new node, and restarting the VM
  - VM does not see changes in hardware

Virtualisation: Software stack

- Due to the introduction of the VMM the software stack is affected
- Software stack in a non-virtualised environment
  - Bottom layer lies the hardware, with an OS layer on top of this
  - libraries and APIs are the next layer followed by applications
- Software stack in a virtualised environment
  - Bottom layer lies the hardware, with the VMM layer on top of this
  - OS, libraries, and applications now go into moveable containers called VMs that sit on top of VMM

Virtualisation: VMs and virtualised hardware

- Normal CPU I/O operations that do not require privileges function in the same way
- Privileged operations are controlled by the VMM
- Partially because multiplexing for CPU, storage, and networking is needed
- Also by letting privileged operations run unchecked it could destroy the state of other VMs

Virtualisation: VM rights

- Virtual machines are assigned privilege access rights
- VMM is responsible for granting, revoking, and enforcing these rights
- Permits tight control on the amount of resources a VM can use at any time by modifying the privileges of the VM

Virtualisation: VMs and users

- Virtual machines provide user convenience
- Users can build new VMs by making copies of the current VM
- Permits the VM to run in a remote environment instead of user provided hardware
- Both of these points are what makes the cloud work

Virtualisation: Downsides

- The hypervisor and virtualisation require CPU time, and memory to work
  - Always slower than running the OS and applications on hardware directly
  - Usually because of the VMM trapping and validating privileged instructions
- Initial hardware costs are much higher
  - Nodes for running virtual machines are generally of a higher specification
  - But most of this cost is recouped by running many virtual machines on the same hardware

Virtualisation: Definitions

- Definition of a VMM
- "A VMM (or hypervisor) is a piece of software that securely partitions the resources of a computer system into one or more virtual machines"
- Definition of a Guest Operating System
- "A guest operating system is an operating system that runs under the control of a VMM rather than directly on hardware

Virtualisation: VMMs

- In normal applications there are two modes of execution
  - User mode: execution of non-privileged instructions (Ring 3 in x86)
  - Kernel mode: execution of privileged instructions (Ring 0 in x86)
- During execution an application will at times context switch between the two to run a privileged instruction
  - Application must context switch to kernel mode to execute instruction and then context switch back to usermode
  - Costly operation
  - Kernel enforces if applications are permitted to run privileged operations

Virtualisation: VMMs

- There is added complexity to this when running a VMM
  - VMM runs in kernel mode
  - VM runs in user mode
  - VM kernel mode must be simulated by the VMM if virtualisation is not supported by the hardware
    - produces slowdown due to translation of operation from VM to OS

Virtualisation: VMMs

- If hardware virtualisation is supported the cost of translation is reduced
  - A special set of instructions are added along with a new privilege level (Ring -1 on x86)
  - The hypervisor is in control of Ring -1
  - Can grant VMs access to Ring 0 from here (with simulation)
  - Ensures that current operating systems and applications are not affected by a hypervisor

Virtualisation: VMM forms

- A VM can take one of many forms
- Bare-Metal VMM: Thin software layer that runs on the hardware directly. VMs run on top of this layer
  - Provides the highest performance
  - Hypervisor has direct access to hardware
  - Reduces the delay in granting VMs access to privileged instructions
- Hybrid VMM: shares the hardware with a currently existing OS
  - VMM sits along side the OS (think VMWare player or VirtualBox)
  - Easy access to virtual machines
  - Slower than bare metal due to competition with OS

Virtualisation: VMM forms

- Hosted VMM: the VM shares many components of the host OS
  - Simplifies the construction of a VM
  - However there is a significant loss of performance and increased overhead
  - Some operations are enforced by the host OS and not the VMM
  - Not used anymore

Virtualisation: Performance Isolation

- A necessary condition needed to satisfy Quality of Service (QoS) in a cloud environment
  - The runtime performance of one VM should not be affected by other VMs
  - All VM's are competing for the same resources
  - Must ensure that all VM's get fair access to those resources
- In most cases the VMM will restrict the processing time each VM receives
  - To ensure quality of service

Virtualisation: Performance Isolation

- Isolation can be provided in two ways
- Processor virtualisation: each VMM gets access to a virtual version of the processor but all instructions are run in hardware
  - Virtual version is usually restricted in speed
  - Only supports OSes that use the same instruction set as the processor
- Processor emulation: each VMM gets access to a software version of the CPU but all instructions are emulated
  - Can support different processors on a single CPU
  - Must translate between instructions
  - Significantly slower than virtualisation
- In a traditional OS processes and threads are multiplexed

Virtualisation: Performance Isolation

- Whereas in a VMM entire OSes are multiplexed
  - OSes are much larger than a process or thread
  - Thus a VM context switch is more expensive than a process or thread context switch
- Each VM can only execute user mode instructions without interference from the VMM
  - Any kernel mode instructions will be trapped by the VMM and will be handled by the VMM
  - These context switches are expensive in time and instructions

Virtualisation: Security Isolation

- Because the VMM simulates dedicated hardware for each virtual machine the VMM must enforce security
- VMs are constrained and isolated from each other
- It appears to the VMs that they are each running on their own individual physical node
- Should a VM wish to communicate with another VM this will be marshalled by the VMM

Virtualisation: VMM implementations

- VMMs much simpler to construct and specify than an OS
- There are less privileged functions exposed
- Less chance of a security attack
- For example the Xen VMM consists of roughly 60,000 lines of code where a comparable OS would be in the millions

Virtualisation: efficient virtualisation

- Popek and Goldberg devised a set of conditions necessary for virtualisation to be efficient
  - A program running on a VMM should behave identically to a program running on an OS
  - The VMM should be in complete control of resources
  - A statistically significant fraction of machine instructions mus be executed without VMM intervention

Virtualisation: Instruction Virtualisation

- There are some instructions that are not virtualisable regardless of the VMM used.
- For these instructions one of two strategies may be used.
  - Binary Translation: non virtualisable instructions are replaced with other instructions
  - Paravirtualisation: The guest OS is modified such that all nonvirtualisable instructions are replaced by virtualisable instructions that achieve the same effect
  - Binary Translation is often used in full VMM environments whereas paravirtualisation is the only option for paravirtualised environments

Virtualisation: Full Vs Para

- Full virtualisation gives each VM an exact virtual copy of hardware
- Paravirtualisation gives each VM a slightly modified virtual version of the hardware
  - The CPU and other components must have modes or instructions to support virtualisation
  - If these are not present the only option is paravirtualisation

Virtualisation: Shadow copies

- VMMs must maintain shadow copies of standard OS control structures
- These include page tables for virtual memory
- Every time a VM wishes to modify one of these structures the VMM must trap that instruction and handle it
  - Introduces significant overhead
  - Ensures consistency among VMs
  - Prevents memory trashing and corruption
  - Introduces significant slowdown to a VM

Virtualisation: Shadow copies

- However, it is possible for an application to perform better in a VMM than in an OS environment
- Through a process of cache isolation, where cache is evenly divided between VMs
- Workloads that compete for cache when running concurrently can be run in two separate VMs and thus use two separate areas of cache
- Thus no cache trashing, i.e. less cache misses

Virtualisation Hardware Support

- Due to the rising popularity of virtualisation almost all CPU vendors have embedded virtualisation modes and instructions in their processors which bring the following benefits
  - Improved performance
  - Improved security
  - Simplified implementation of hypervisors
- However not all processors are designed to have virtualisation
- We will look at the example of x86 which is a 30+ year old design and the issues it presents when adding virtualisation

Virtualisation: x86 issues

- There are many problems faced by x86 with respect to virtualisation
- Ring deprivileging: applications and OSes are forced by the VMM to use a higher privilege level than 0
  - VMM should be only one with access to ring 0
  - With 64 bit processors the only option is to run the OS in ring 3 (same ring as userspace)
  - For this to work the hypervisor must trap and emulate all instructions intended for ring 0

Virtualisation: x86 issues

- Ring Aliasing: Issues arising from when a guest OS or application runs in a different privilege level under the VMM than it was designed for
- Address Space Compression: VMM stores system data structures such as interrupt tables in the guest address space
  - As a way of interfacing between the hypervisor and VM
  - The hypervisor must protect these structures but must permit guest access so the guest can function

Virtualisation: x86 issues

- Non-faulting access to privileged state
  - Some CPU instructions can only execute at privilege level 0
  - May load or modify special CPU registers that affect CPU operation
  - The VMM must trap and emulate these instructions in order to hide from the OS that it does not have the required privilege level
  - This will modify an emulated CPU structure that the guest will think is the actual CPU

Virtualisation: x86 issues

- Guest System Calls
  - SYSENTER and SYSEXIT permit software to transition in and out of ring 0
  - Used to reduce latency on system calls by grouping them together
  - VMM runs in ring zero
  - Therefore VMM is responsible for emulating all instructions that are contained in a SYSENTER and SYSEXIT

Virtualisation: x86 issues

- Interrupt virtualisation: interrupts generated by VMs must be caught by the VMM
- VMM must determine which virtual machine is receiving the interrupt and generate a virtual interrupt to pass to the VM
- Gets more complex if the OS has the ability to mask interrupts (pretty much any modern OS)
- Thus the VMM has to monitor and track this which adds complexity and performance overheads to the running of the VM

Virtualisation: x86 issues

- Access to hidden state
  - Some parts of system state are hidden
  - There is no method in x86 to save and restore this state
- Ring Compression: protects the VMM memory from trashing by VMs
- Frequent access to privileged resources will increase VMM overhead
  - guest OSes frequently use the task priority register
  - VMM must trap and emulate all accesses to this register and protect it from corruption
  - introduces yet more emulation and overheads

Virtualisation: Security

- Like all software systems VMMs have security concerns
- The VMM is responsible for determining access to virtual devices from VMs
- It is extremly difficult to get access to a VMM from a VM but it can be compromised
- In a layered software system like this control is enforced by the layer closest to the hardware (in this case the VMM)

Virtualisation: Security

- One way of circumventing a VMM is to insert a layer below the VMM itself
- This can be done by installing a VMBR (Virtual Machine Based Rootkit)
- Installs itself between the hardware and the VMM thus the VMM is now controlled by the VMBR
- VMM has no idea it is running on a rootkit. Therefore has no defence from it
- VMBR modifies boot order of the system to start itself first so it can run the VMM ontop of itself

Virtualisation: Security

- Software Fault Isolation is used to sandbox binary code
- As VMs can be insecure or tampered with
- VMs can be tampered in the same way as if it was running on hardware directly
- Assumes that we have a trusted runtime and an untrusted application running on top

Virtualisation: Security

- To enforce this rules are applied to the binary code
- It is set to be read only as malware is usually self modifying
- Code is divided into 32 byte boundaries which no instruction can cross
- Disassembly at boundary will reach all valid instructions
- All indirect flow control instructions are replaced by pseudo instructions