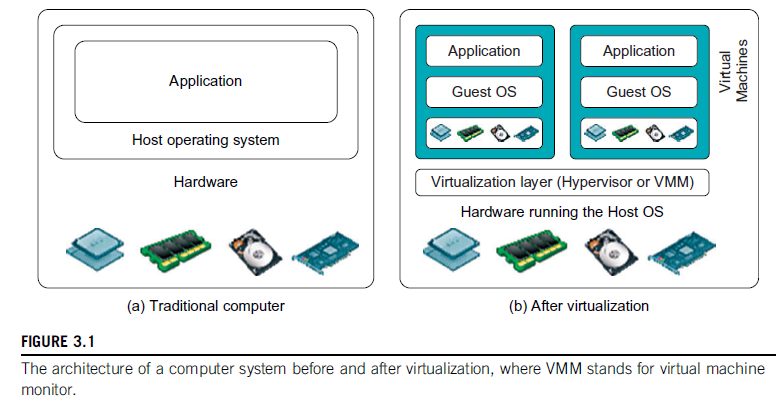
**UNIT – 2**

**Virtual Machines and Virtualization of Clusters and Data Centers**

1. The massive usage of virtual machines (VMs) opens up new opportunities for parallel, cluster grid, cloud and distributed computing. Virtualization enables the users to share expensive hardware resources by multiplexing (i.e., multiple analog/digital are combined into one signal over a shared medium [2]) VMs on the same set of hardware hosts like servers or data centers.
2. **Implementation Levels of Virtualization**: Virtualization is a concept by which several VMs are multiplexed into the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility. Hardware resources (CPU, memory, I/O devices etc.) or software resources (OS and apps) can be virtualized at various layers of functionality.

The main idea is to separate hardware from software to obtain greater efficiency from the system. Ex: Users can gain access to more memory by this concept of VMs. With sufficient storage, any computer platform can be installed in another host computer [1], even if processors’ usage and operating systems are different.

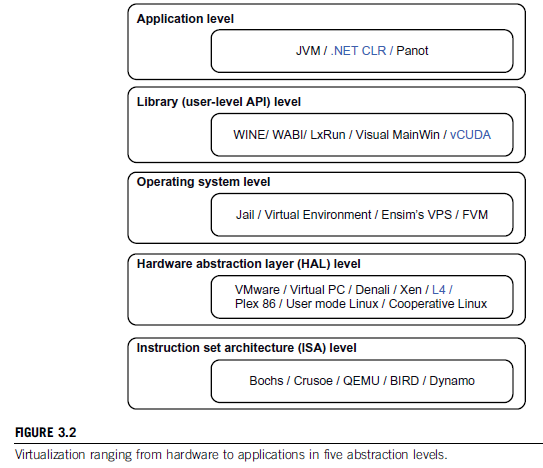
* 1. **Levels of Virtualization Implementation**: A traditional computer system runs with a host OS specially adjusted for its hardware architecture. This is depicted in Figure 3.1a [1].



After virtualization, different user apps managed by their own OS (i.e., guest OS) can run on the same hardware, independent of the host OS. This is often done by adding a virtualization layer as shown in Figure 3.1b [2].

This virtualization layer is called VM Monitor or hypervisor. The VMs can be seen in the upper boxes where apps run on their own guest OS over a virtualized CPU, memory and I/O devices.

* 1. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be saved by the VMs. The virtualization software creates the abstract of VMs by introducing a virtualization layer at various levels of a computer. General virtualization layers include the instruction set architecture (ISA) level, hardware level, OS level, library support level, and app level. This can be seen in Figure 3.2 [1]. The levels are discussed below.



* + 1. **Instruction Set Architecture Level**: At the ISA level, virtualization is performed by emulation (imitate) of the given ISA by the ISA of the host machine. Ex: MIPS binary code can run on an x86-based host machine with the help of ISA simulation. Instruction emulation leads to virtual ISAs created on any hardware machine.

Basic level of emulation can be traced at code interpretation. An interpreter (line-by-line compiler) program works on the instructions one-by-one and this process is slow. To speedup, *dynamic binary translation* can be used where it translates blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. This emulation requires binary translation and optimization. Hence, a Virtual-ISA requires a processor specific translation layer to the compiler.

* + 1. **Hardware Abstraction Level**: Hardware level virtualization is performed on the bare hardware. This approach generates a virtual hardware environment and processes the hardware in a virtual manner. The idea is to virtualize the resources of a computer by utilizing them concurrently. Ex: IBM Xen hypervisor (VMM) runs Linux or other guest OS applications. [Discussed later]
    2. **OS Level:** This refers to an abstraction layer between the OS and the user apps. The OS level virtualization creates isolated *containers* on a single physical server and OS instances to utilize software and hardware in data centers. The containers behave like real servers. OS level virtualization is used in creating virtual hosting environments to allocate hardware resources among a large number of ‘distrusting’ users. It can also be used to indirectly merge server hardware by moving resources on different hosts into different containers or VMs on one server.
    3. **NOTE: *Containers*** [3]use the host operating system as their base, and not the hypervisor. Rather than virtualizing the hardware (which requires full virtualized operating system images for each guest), containers virtualize the OS itself, sharing the host OS kernel and its resources with both the host and other containers.
    4. **Library Support Level**: Most applications use APIs exported by user-level libraries rather than lengthy system calls by the OS. Virtualization with library interfaces is possible by controlling the communication link between apps and the rest of the system through API hooks.

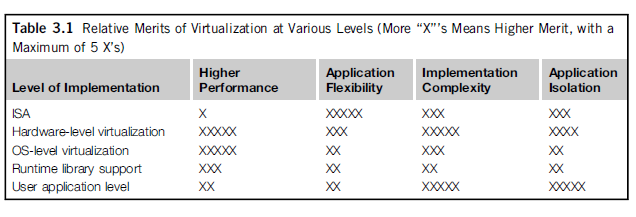
Ex: (a) **Wine** ([recursive acronym](https://en.wikipedia.org/wiki/Recursive_acronym) for *Wine Is Not an*[*Emulator*](https://en.wikipedia.org/wiki/Emulator)) is a [free and open source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [compatibility layer](https://en.wikipedia.org/wiki/Compatibility_layer) [software application](https://en.wikipedia.org/wiki/Application_software) that aims to allow applications designed for [MS-Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) to run on [Linux](https://en.wikipedia.org/wiki/Unix-like) [OS](https://en.wikipedia.org/wiki/Operating_system).

(b) vCUDA by NVIDIA. (CUDA – No acronym)

**NOTE: Library** [4]in computing is a collection of non-volatile (stable) resources used by computer programs to develop software. These include configuration data (organised data), documentation, help data, message templates, code subroutines classes and specifications.

* + 1. **User-App Level**: An app level virtualization brings out a real VM; this process is also known as process level virtualization. Generally HLL VMs are used where virtualization layer is an app above the OS; it can run programs written and compiled to an abstract machine definition. Ex: JVM and .NET CLR (Common Language Runtime).

Other forms of app level virtualization are app isolation, app sandboxing or app streaming. Here, the app is wrapped in a layer and is isolated from the host OS and other apps. This makes the app more much easier to distribute and remove from user workstations. Ex: LANDesk (an app virtualization platform) – this installs apps as self-contained, executable files in an isolated environment. No actual installation is required and no system modifications are needed.



Note from Table 3.1 [1] that hardware and OS support will yield the highest performance. At the same time, the hardware and app levels are most expensive to implement. User isolation is difficult to archive and ISA offers best flexibility.

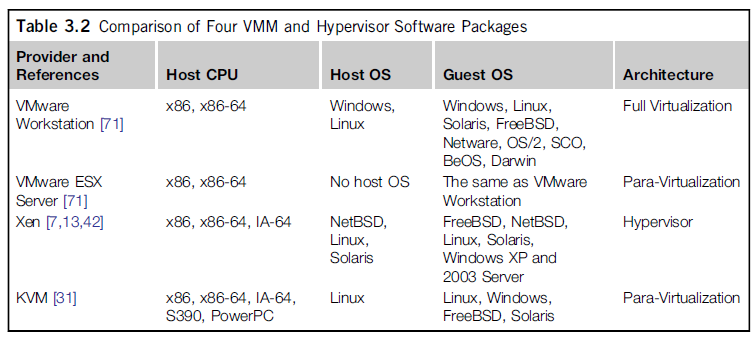
* 1. **VMM Design Requirement and Providers**: As seen before, hardware-level virtualization inserts a layer between real hardware and traditional OS. This layer (VMM/hypervisor) manages the hardware resources of the computer effectively. By the usage of VMM, different traditional operating systems can be used with the same set of hardware simultaneously.
  2. **Requirements for a VMM**:

1. For programs, a VMM should provide an identical environment, same as the original machine.
2. Programs running in this environment should show only minor decreases in speed.
3. A VMM should be in complete control of the system resources.

Some differences might still be caused due to availability of system resources (more than one VM is running on the same system) and differences caused by timing dependencies.

The hardware resource requirements (like memory) of each VM is reduced, but the total sum of them is greater that of the real machine. This is needed because of any other VMs that are concurrently running on the same hardware.

A VMM should demonstrate efficiency in using the VMs. To guarantee the efficiency of a VMM, a statistically dominant subset of the virtual processor’s instructions needs to be executed directly by the real processor with no intervention by the VMM. A comparison can be seen in Table 3.2 [1]:



The aspects to be considered here include (1) The VMM is responsible for allocating hardware resources for programs; (2) a program can’t access any resource that has not been allocated to it; (3) at a certain juncture, it is not possible for the VMM to regain control of the resources already allocated. Note that all processors might not satisfy these requirements of a VMM.

A VMM is tightly related to the architectures of the processors. It is difficult to implement a VMM on some types of processors like x86. If a processor is not designed to satisfy the requirements of a VMM, the hardware should be modified – this is known as hardware assisted virtualization.

* 1. **Virtualization Support at the OS Level**: CC is transforming the computing landscape by shifting the hardware and management costs of a data center to third parties, like banks. The challenges of CC are: (a) the ability to use a variable number of physical machines and VM instances depending on the needs of the problem. Ex: A work may need a single CPU at an instance but multi-CPUs at another instance (b) the slow operation of instantiating new VMs.

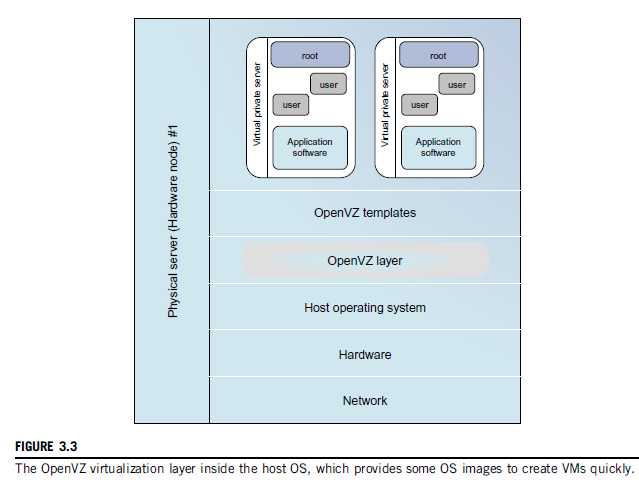
As of now, new VMs originate either as fresh boots or as replicates of a VM template – unaware of the current status of the application.

* 1. **Why OS Level Virtualization** (Disadvantages of hardware level virtualization):

1. It is slow to initiate a hardware level VM since each VM creates its own image from the beginning.
2. Redundancy content is high in these VMs.
3. Slow performance and low density
4. Hardware modifications maybe needed

To provide a solution to all these problems, OS level virtualization is needed. It inserts a virtualization layer inside the OS to partition the physical resources of a system. It enables multiple isolated VMs within a single OS kernel. This kind of VM is called a Virtual Execution Environment (VE) or Virtual Private System or simply a **container**. From the user’s point of view, a VE/container has its own set of processes, file system, user accounts, network interfaces (with IP addresses), routing tables, firewalls and other personal settings.

Note that though the containers can be customized for different people, they share the same OS kernel. Therefore this methodology is also called single-OS image virtualization. All this can be observed in Figure 3.3 [1].

****

* 1. **Advantages of OS Extensions**:

1. VMs at the OS level have minimal start-up shutdown costs, low resource requirements and high scalability.
2. For an OS level VM, the VM and its host environment can synchronise state changes

These can be achieved through two mechanisms of OS level virtualization:

1. All OS level VMs on the same physical machine share a single OS kernel
2. The virtualization layer can be designed in way that allows processes in VMs can access as many resources as possible from the host machine, but can never modify them.
   1. **Disadvantages of OS Extension:** The main disadvantageof OS extensions is that all VMs at OS level on a single container must have the same kind of guest OS. Though different OS level VMs may have different OS distributions (Win XP, 7, 10), they must be related to the same OS family (Win). A Windows distribution can’t run on a Linux based container.

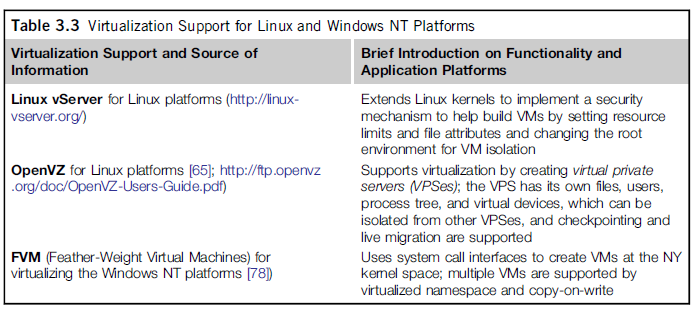
As we can observe in Figure 3.3, the virtualization layer is inserted inside the OS to partition the hardware resources for multiple VMs to run their applications in multiple virtual environments. To implement this OS level virtualization, isolated execution environments (VMs) should be created based on a single OS kernel. In addition, the access requests from a VM must be redirected to the VM’s local resource partition on the physical machine. For example, ‘*chroot*’ command in a UNIX system can create several virtual root directories within an OS that can be used for multiple VMs.

To implement the virtual root directories’ concept, there exist two ways: (a) duplicating common resources to each VM partition or (b) sharing most resources with the host environment but create private copies for the VMs on demand. It is to be noted that the first method incurs (brings up) resource costs and burden on a physical machine. Therefore, the second method is the apparent choice.

* 1. **Virtualization on Linux or Windows Platforms**: Generally, the OS-level virtualization systems are Linux-based. Windows based virtualization platforms are not much in use. The Linux kernel offers an abstraction layer to allow software processes to with and operate on resources without knowing the hardware details. Different Linux platforms use patched kernels to provide special support for extended functionality.

Note that most Linux platforms are not tied to a special kernel. In such a case, a host can run several VMs simultaneously on the same hardware. Examples can be seen in Table 3.3 [1].

* 1. **Middleware Support for Virtualization**: This is the other name for Library-level Virtualization and is also known as user-level Application Binary Interface or API emulation. This type of virtualization can create execution environments for running alien (new/unknown) programs on a platform rather than creating a VM to run the entire OS. The key functions performed here are API call interception and remapping (assign a function to a key).



1. **Virtualization Structures/Tools and Mechanisms**: It should be noted that there are three classes of VM architecture [Page 1]. Before virtualization, the OS manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the OS. Here, the virtualization layer is responsible for converting parts of real hardware into virtual hardware. Different operating systems like Windows and Linux can run simultaneously on the same machine in this manner. Depending on the position of the virtualization layer, several classes of VM architectures can be framed out: Hypervisor Architecture, para-virtualization and host-based virtualization.
   1. **Hypervisor and Xen Architecture**: The hypervisor (VMM) supports hardware level virtualization on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software exists between the hardware and its OS (platform). The hypervisor provides **hypercalls** for the guest operating systems and applications. Depending on the functionality, a hypervisor can assume **micro-kernel** architecture like MS Hyper-V or **monolithic** hypervisor architecture like the VMware ESX for server virtualization.

**Hypercall:** A hypercall is a software trap from a domain to the hypervisor, just as a syscall is a software trap from an application to the kernel. Domains will use hypercalls to request privileged operations like updating page tables.

**Software Trap**: A trap, also known as an exception or a fault, is typically a type of synchronous interrupt caused by an exceptional condition (e.g., breakpoint, division by zero, invalid memory access). A trap usually results in a switch to kernel mode, wherein the OS performs some action before returning control to the originating process. A trap in a system process is more serious than a trap in a user process and might be fatal. The term trap might also refer to an interrupt intended to initiate a context switch to a monitor program or debugger.

**Domain**: It is a group of computers/devices on a network that are administered as a unit with common rules and procedures. Ex: Within the Internet, all devices sharing a common part of the IP address are said to be in the same domain.

**Page Table**: A page table is the data structure used by a virtual memory system in an OS to store the mapping between virtual addresses and physical addresses.

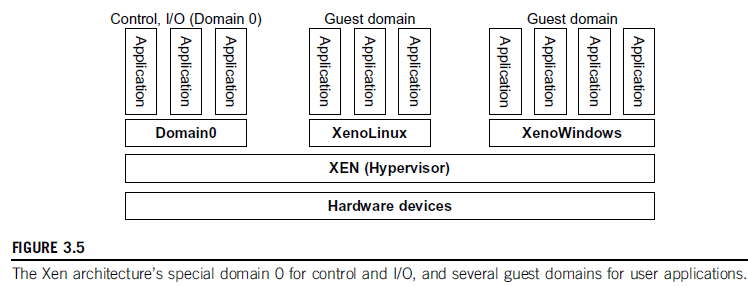
**Kernel**: A kernel is the central part of an OS and manages the tasks of the computer and hardware like memory and CPU time.

**Monolithic Kernel**: These are commonly used by the OS. When a device is needed, it is added as a part of the kernel and the kernel increases in size. This has disadvantages like faulty programs damaging the kernel and so on. Ex: Memory, processor, device drivers etc.

**Micro-kernel**: In micro-kernels, only the basic functions are dealt with – nothing else. Ex: Memory management and processor scheduling. It should also be noted that OS can’t run only on a micro-kernel, which slows down the OS.

[SIM – Micro SIM]

* 1. The size of the hypervisor code of a micro-kernel hypervisor is smaller than that of monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the VM usage.
  2. **Xen Architecture**: It is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, whose policy is implemented by Domain 0.



As can be seen in Figure 3.5 [1], Xen doesn’t include any device drivers; it provides a mechanism by which a guest-OS can have direct access to the physical devices. The size of Xen is kept small, and provides a virtual environment between the hardware and the OS. Commercial Xen hypervisors are provided by Citrix, Huawei and Oracle.

The core components of Xen are the hypervisor, kernel and applications. Many guest operating systems can run on the top of the hypervisor; but it should be noted that one of these guest OS controls the others. This guest OS with the control ability is called Domain 0 – the others are called Domain U. Domain 0 is first loaded when the system boots and can access the hardware directly and manage devices by allocating the hardware resources for the guest domains (Domain U).

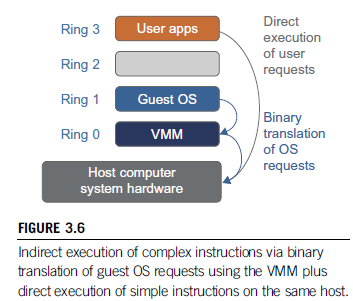
Say Xen is based on Linux and its security level is some C2. Its management VM is named as Domain 0, which can access and manage all other VMs on the same host. If a user has access to Domain 0 (VMM), he can create, copy, save, modify or share files and resources of all the VMs. This is a huge advantage for the user but concentrating all the resources in Domain 0 can also become a privilege for a hacker. If Domain 0 is hacked, through it, a hacker can control all the VMs and through them, the total host system or systems. Security problems are to be dealt with in a careful manner before handing over Xen to the user.

A machine’s lifetime can be thought of as a straight line that progresses monotonically (never decreases or increases) as the s/w executes. During this time, executions are made, configurations are changed, and s/w patches can be applied. VM is similar to tree in this environment; execution can go into N different branches where multiple instances of VM can be done in this tree at any time. VMs can also be allowed to rollback to a particular state and rerun from the same point.

* 1. **Binary Translation with Full Virtualization**: Hardware virtualization can be categorised into two categories: full virtualization and host-based virtualization.

**Full Virtualization** doesn’t need to modify the host OS; it relies upon binary translation to trap and to virtualize certain sensitive instructions. Normal instructions can run directly on the host OS. This is done to increase the performance overhead – normal instructions are carried out in the normal manner, but the difficult and precise executions are first discovered using a trap and executed in a virtual manner. This is done to improve the security of the system and also to increase the performance.

**Binary Translation of Guest OS Requests Using a VMM**:



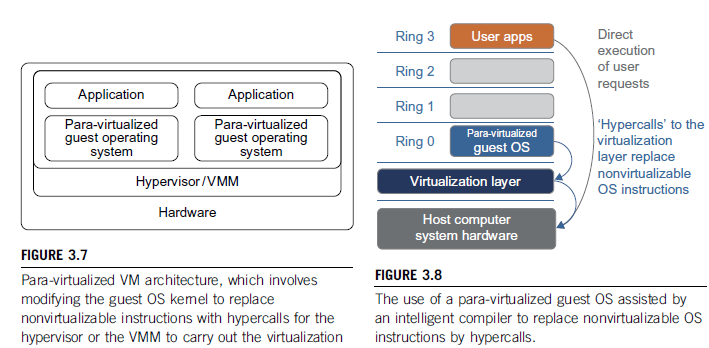
This approach is mainly used by VMware and others. As it can be seen in Figure 3.6 [1], the VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instructions to identify complex and privileged instructions and trap them into the VMM, which emulates the behaviour of these instructions. Binary translation is the method used for emulation (A => 97 => 01100001) [5]. Note that full virtualization combines both binary translation and direct execution. The guest OS is totally decoupled from the hardware and run virtually (like an emulator).

Full virtualization is ideal since it involves binary translation and is time consuming. Binary translation also is cost consuming but it increases the system performance. (Same as 90% of the host).

In a **host-based** virtualization system both host and guest OS are used and a virtualization layer is built between them. The host OS is still responsible for managing the hardware resources. Dedicated apps might run on the VMs and some others can run on the host OS directly. By using this methodology, the user can install the VM architecture without modifying the host OS. The virtualization software can rely upon the host OS to provide device drivers and other low level services. Hence the installation and maintenance of the VM becomes easier.

Another advantage is that many host machine configurations can be perfectly utilized; still four layers of mapping exist in between the guest and host operating systems. This may hinder the speed and performance, in particular when the ISA (Instruction Set Architecture) of a guest OS is different from that of the hardware – binary translation MUST be deployed. This increases in time and cost and slows the system.

* 1. **Para-Virtualization with Compiler Support**: Para-Virtualization modifies the guest operating systems; a para-virtualized VM provides special APIs which take up user apps needing those changes. Para-virtualization tries to reduce the virtualization burden/extra-work to improve the performance – this is done by modifying only the guest OS kernel. This can be seen in Figure 3.7 [1].



Ex: In a typical para-virtualization architecture, which considers an x86 processor, a virtualization layer is inserted between h/w and OS. According to the x86 ‘ring definition’ the virtualization layer should also be installed at Ring 0. In Figure 3.8 [1], we can notice that para-virtualization replaces instructions that cannot be virtualized with hypercalls (placing a trap) that communicate directly with the VMM. Notice that if a guest OS kernel is modified for virtualization, it can’t run the hardware directly – that should be done through the virtualization layer.

* 1. **Disadvantages of Para-Virtualization**: Although para-virtualization reduces the overhead, it has other problems. Its compatibility (suitability) and portability can be in doubt because it has to support both the modified guest OS and the host OS as per requirements. Also, the maintenance cost of para-virtualization is high since it may require deep kernel modifications. Finally, the performance advantage of para-virtualization is not stable – it varies as per the workload. But compared with full virtualization, para-virtualization is more easy and practical since binary translation is not much considered. Many products utiliza para-virtualization to overcome the less speed of binary translation. Ex: Xen, KVM, VMware ESX.
  2. **Note**: Kernel based VM (**KVM**): This is a Linux para-virtualization system – it is a part of the Linux kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. Other activities are taken care of by the KVM and this methodology makes it easier to handle than the hypervisor. Also note that KVM is hardware assisted para-virtualization tool, which improves performance and supports unmodified guest operating systems like Windows, Linux, Solaris and others.
  3. **Virtualization of CPU, Memory and I/O Devices**: Processors employ a special running mode and instructions, known as hardware-assisted virtualization. Through this, the VMM and guest OS run in different modes; all sensitive instructions of the guest OS and its apps are caught by the ‘trap’ in the VMM.
     1. **H/W Support for Virtualization**: Modern operating systems and processors permit multiple processes to run simultaneously. A protection mechanism should exist in the processor so that all instructions from different processes will not access the hardware directly – this will lead to a system crash.

All processors should have at least two modes – user and supervisor modes to control the access to the hardware directly. Instructions running in the supervisor mode are called privileged instructions and the others are unprivileged.

Ex: VMware Workstation

* + 1. **CPU Virtualization**: A VM is a duplicate of an existing system; majority of instructions are executed by the host processor. Unprivileged instructions run on the host machine directly; other instructions are to be handled carefully. These critical instructions are of three types: privileged, control-sensitive and behaviour-sensitive.

**Privileged**=> Executed in a special mode and are trapped if not done so.

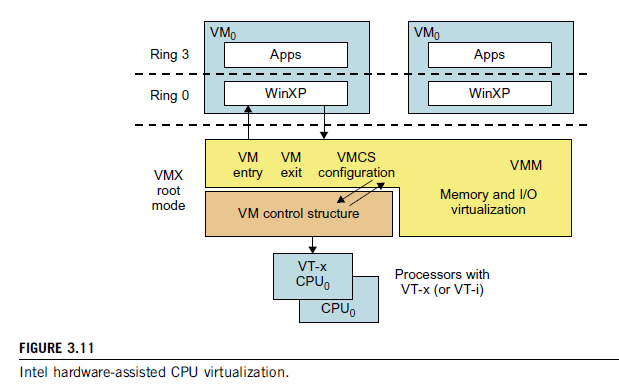
**Control-Sensitive=>** Attempt to change the configuration of the used resources

**Behaviour-Sensitive=>** They have different behaviours in different situations (high load or storage or capacity)

A CPU is VZ only if it supports the VM in the CPU’s user mode while the VMM runs in a supervisor’s mode. When the privileged instructions are executed, they are trapped in the VMM. In this case, the VMM acts as a mediator between the hardware resources and different VMs so that correctness and stability of the system are not disturbed. It should be noted that not all CPU architectures support VZ.

**Process**:

* System call triggers the 80h interrupt and passes control to the OS kernel.
* Kernel invokes the interrupt handler to process the system call
* In Xen, the 80h interrupt in the guest OS concurrently causes the 82h interrupt in the hypervisor; control is passed on to the hypervisor as well.
* After the task is completed, the control is transferred back to the guest OS kernel.
  + 1. **Hardware Assisted CPU VZ**: Since full VZ or para-VZ is complicated, this new methodology tries to simplify the situation. Intel and AMD add an additional mode called privilege mode level to the x86 processors. The OS can still run at Ring 0 and hypervisor at Ring 1. Note that all privileged instructions are trapped at the hypervisor. Hence, no modifications are required in the VMs at OS level.

****

VMCS=> VM Control System

VMX=> A virtual router

* + 1. **Memory Virtualization**: In the **traditional** methodology, the OS maintains mappings between virtual memory to machine memory (MM) using page tables, which is a one-stage mapping from virtual memory to MM.

**Virtual memory** is a feature of an operating system (OS) that allows a computer to compensate for shortages of physical memory by temporarily transferring pages of data from random access memory (**RAM**) to disk storage.

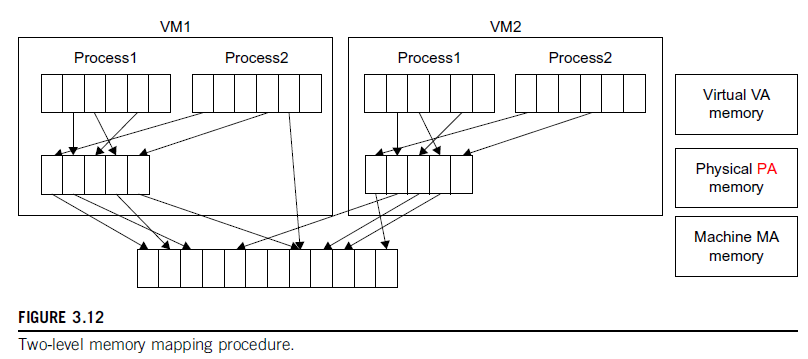
**Machine Memory [6]** is the upper bound (threshold) of the physical memory that a host can allocate to the VM. All modern x86 processors contain memory management unit (MMU) and a translation look-aside buffer (TLB) to optimize (use in the best way) the virtual memory performance.

In a **virtual execution environment**, virtual memory VZ involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.

Stages:

* Virtual memory to physical memory
* Physical memory to machine memory.

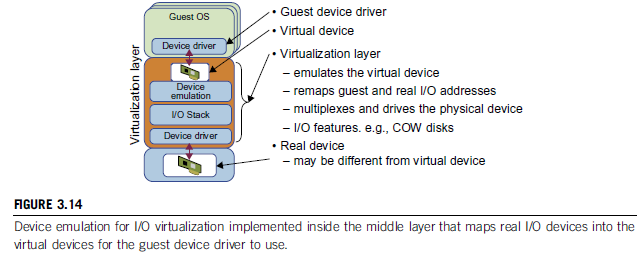
Other Points: MMU should be supported, guest OS controls to monitor mapping of virtual addresses to physical memory address of the VMs. All this is depicted in Figure 3.12 [1].



VA-Virtual Address; PA-Physical Address; MA-Machine Address

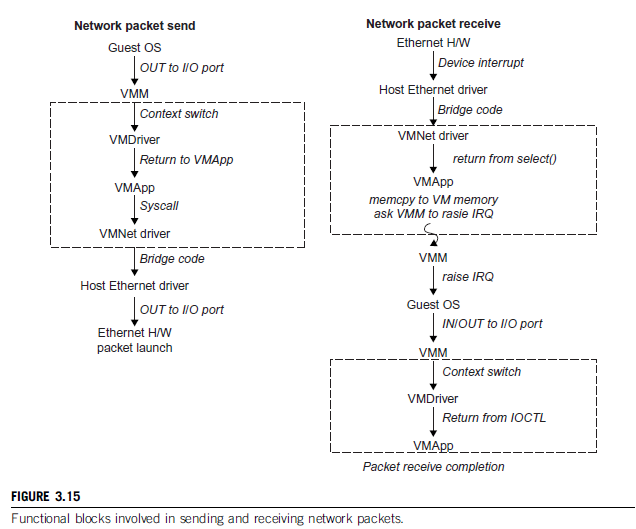
Each page table of a guest OS has a page table allocated for it in the VMM. The page table in the VMM which handles all these is called a **shadow page table**. As it can be seen all this process is nested and inter-connected at different levels through the concerned address. If any change occurs in the virtual memory page table or TLB, the shadow page table in the VMM is updated accordingly.

* + 1. **I/O Virtualization**: This involves managing of the routing of I/O requests between virtual devices and shared physical hardware. The there are three ways to implement this are full device emulation, para-VZ and direct I/O.
* **Full Device Emulation**: This process emulates well-known and real-world devices. All the functions of a device or bus infrastructure such as device enumeration, identification, interrupts etc. are replicated in the software, which itself is located in the VMM and acts as a virtual device. The I/O requests are trapped in the VMM accordingly. The emulation approach can be seen in Figure 3.14 [1].



* **Para-VZ**: This method of I/O VZ is taken up since software emulation runs slower than the hardware it emulates. In para-VZ, the frontend driver runs in Domain-U; it manages the requests of the guest OS. The backend driver runs in Domain-0 and is responsible for managing the real I/O devices. This methodology (para) gives more performance but has a higher CPU overhead.
* **Direct I/O VZ**: This lets the VM access devices directly; achieves high performance with lower costs. Currently, it is used only for the mainframes.

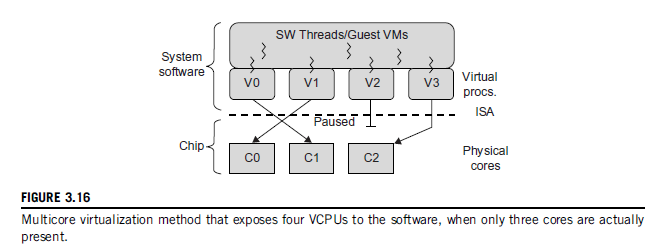
Ex: **VMware Workstation for I/O VZ: *NIC=> Network Interface Controller***



* 1. **Virtualization in Multi-Core Processors**: Virtualizing a multi-core processor is more complicated than that of a uni-core processor. Multi-core processors have high performance by integrating multiple cores in a chip, but their virtualization poses a new challenge. The main difficulties are that apps must be utilized in a parallelized way to use all the cores and this task must be accomplished by software, which is a much higher problem.

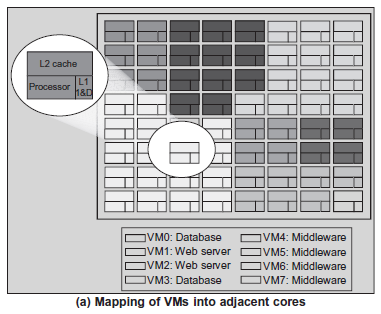
To reach these goals, new programming models, algorithms, languages and libraries are needed to increase the parallelism.

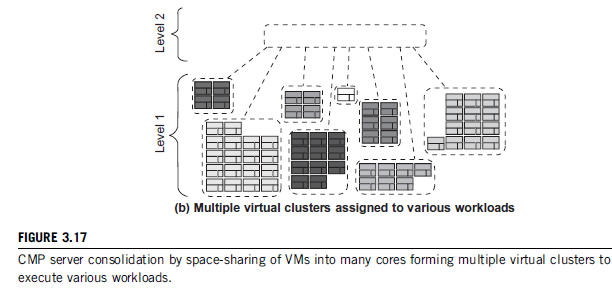
* 1. **Physical versus Virtual Processor Cores**: A multi-core virtualization method was proposed to allow hardware designers to obtain an abstraction of the lowest level details of all the cores. This technique alleviates (lessens) the burden of managing the hardware resources by software. It is located under the ISA (Instruction Set Architecture) and is unmodified by the OS or hypervisor. This can be seen in Figure 3.16 [1].



* 1. **Virtual Hierarchy**: The emerging concept of many-core chip multiprocessors (CMPs) is a new computing landscape (background). Instead of supporting time-sharing jobs on one or few cores, abundant cores can be used in a space-sharing – here single or multi-threaded jobs are simultaneously assigned to the cores. Thus, the cores are separated from each other and no interferences take place. Jobs go on in parallel, for long time intervals. To optimize (use effectively) the workloads, a virtual hierarchy has been proposed to overlay (place on top) a coherence (consistency) and caching hierarchy onto a physical processor. A virtual hierarchy can adapt by itself to fit how to carry out the works and share the workspace depending upon the workload and the availability of the cores.

The CMPs use a physical hierarchy of two or more cache levels that statically determine the cache (memory) allocation and mapping. A **virtual hierarchy is** a cache hierarchy that can adapt to fit the workloads. First level in the hierarchy locates data blocks close to the cores to increase the access speed; it then establishes a shared-cache domain, establishes a point of coherence, thus increasing communication speed between the levels. This idea can be seen in Figure 3.17(a) [1].





Space sharing is applied to assign three workloads to three clusters of virtual cores: VM0 and VM3 for DB workload, VM1 and VM2 for web server workload, and VM4-VM7 for middleware workload. Basic assumption here is that a workload runs in its own VM. But in a single OS, space sharing applies equally. To encounter this problem, Marty and Hill suggested a two-level virtual coherence and caching hierarchy. This can be seen in Figure 3.17(b) [1]. Each VM operates in its own **virtual cluster** in the first level which minimises both access time and performance interference. The second level maintains a globally shared memory.

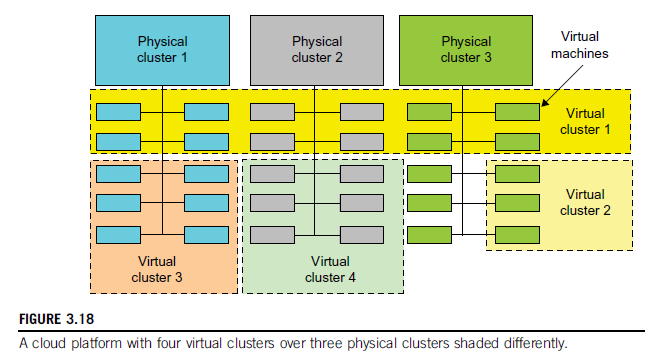
A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation.

1. **Virtual Clusters and Resource Management**: A physical cluster is a collection of physical servers that are interconnected. The issues that are to be dealt with here are: live migration of VMs, memory and file migrations and dynamic deployment of virtual clusters.

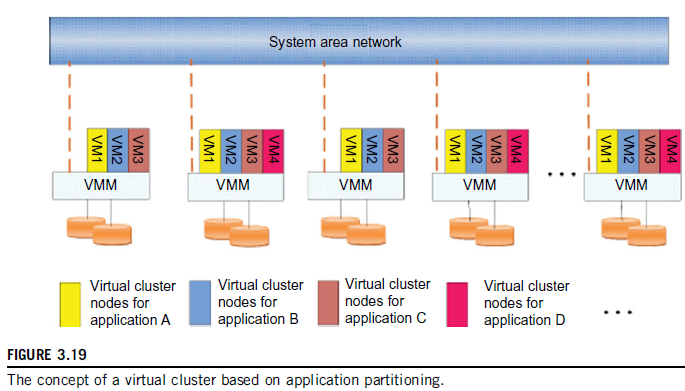
When a general VM is initialized, the administrator has to manually write configuration information; this increases his workload, particularly when more and more VMs join the clusters. As a solution to this, a service is needed that takes care of the configuration information (capacity, speed etc.) of the VMs. The best example is Amazon’s **Elastic Compute Cloud (EC2)**, which provides elastic computing power in a cloud.

Most VZ platforms like VMware ESX Server, and XenServer support a **bridging mode** which allows all *domains to appear* on the network *as individual hosts*. Through this mode, VMs can communicate with each other freely through the virtual network and configure automatically.

* 1. **Physical versus Virtual Clusters**: Virtual Clusters are built with VMs installed at one or more physical clusters. The VMs in a virtual cluster are interconnected by a virtual network across several physical networks. The concept can be observed in Figure 3.18 [1].



* 1. The provisioning of VMs to a virtual cluster is done dynamically to have the following properties:
* The virtual cluster nodes can be either physical or virtual (VMs) with different operating systems.
* A VM runs with a guest OS that manages the resources in the physical machine.
* The purpose of using VMs is to consolidate multiple functionalities on the same server.
* VMs can be replicated in multiple servers to promote parallelism, fault tolerance and disaster discovery.
* The no. of nodes in a virtual cluster can grow or shrink dynamically.
* The failure of some physical nodes will slow the work but the failure of VMs will cause no harm (fault tolerance is high).
  1. **NOTE**: Since system virtualization is widely used, the VMs on virtual clusters have to be effectively managed. The virtual computing environment should provide high performance in virtual cluster deployment, monitoring large clusters, scheduling of the resources, fault tolerance and so on.



* 1. Figure 3.19 [1] shows the concept of a virtual cluster based on app partitioning. The different colours represent nodes in different virtual clusters. The storage images (SSI) from different VMs from different clusters is the most important concept here. Software packages can be pre-installed as templates and the users can build their own software stacks. Note that the boundary of the virtual cluster might change since VM nodes are added, removed, or migrated dynamically.
  2. **Fast Deployment and Effective Scheduling**: The concerned system should be able to
* Construct and distribute software stacks (OS, libraries, apps) to a physical node inside the cluster as fast as possible
* Quickly switch runtime environments from one virtual cluster to another.

**NOTE**: **Green Computing**: It is a methodology that is environmentally responsible and an eco-friendly usage of computers and their resources. It is also defined as the study of designing, manufacturing, using and disposing of computing devices in a way that reduces their environmental impact.

* 1. Engineers must concentrate upon the point the available resources are utilized in a cost and energy-reducing manner to optimize the performance and throughput. Parallelism must be put in place wherever needed and virtual machines/clusters should be used for attaining this goal. Through this, we can reduce the overhead, attain load balancing and achieve scale-up and scale-down mechanisms on the virtual clusters. Finally, the virtual clusters must be clustered among themselves again by mapping methods in a dynamical manner.
  2. **High Performance Virtual Storage**: A template must be prepared for the VM construction and usage and distributed to the physical hosts. Software packages that reduce the time for customization (getting used to) and switching of environment. Users should be identified by their profiles that are stored in data blocks. All these methods increase the performance in virtual storage. Ex: Dropbox

Steps to deploy (arrange/install) a group of VMs onto a target cluster:

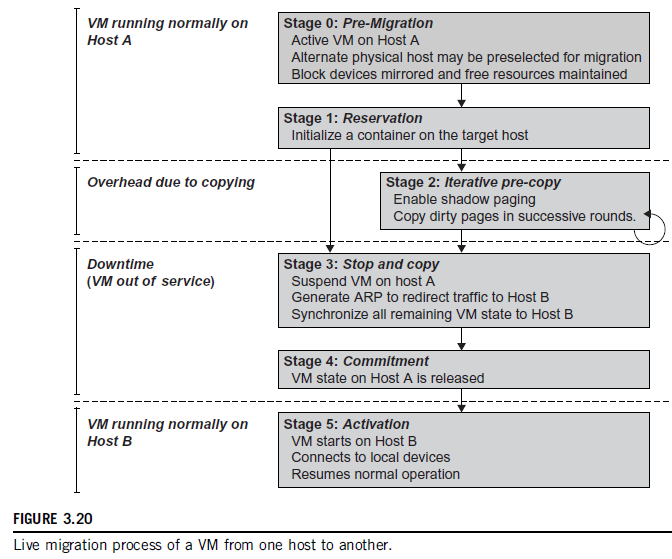
* Preparing the disk image (SSI)
* Configuring the virtual machines
* Choosing the destination nodes
* Executing the VM deployment commands at every host
  1. **NOTE**: A **template** is a disk image/SSI that hides the distributed environment from the user. It may consist of an OS and some apps. Templates are chosen by the users as per their requirements and can implement COW (Copy on Write) format. A new COW backup file is small and easy to create and transfer, thus reducing space consumption.

It should be noted that every VM is configured with a name, disk image, network settings, and is allocated a CPU and memory. But this might be cumbersome if the VMs are many in number. The process can be simplified by configuring similar VMs with pre-edited profiles. Finally, the deployment principle should be able to fulfil the VM requirement to balance the workloads.

* 1. **Live VM Migration Steps**: Normally in a cluster built with mixed modes of host and guest systems, the procedure is to run everything on the physical machine. When a VM fails, it can be replaced by another VM on a different node, as long as they both run the same guest OS. This is called a **failover** (a procedure by which a system automatically transfers control to a duplicate system when it detects a fault or failure) of a physical system to a VM. Compared to a physical-physical failover, this methodology has more flexibility. It also has a drawback – a VM must stop working if its host node fails. This can be lessened by migrating from one node to another for a similar VM. The live migration process is depicted in Figure 3.20 [1].
  2. **Managing a Virtual Cluster**: There exist four ways.

1. We can use a **guest-based manager**, by which the cluster manager resides inside a guest OS. Ex: A Linux cluster can run different guest operating systems on top of the Xen hypervisor.
2. We can bring out a **host-based manager** which itself is a cluster manager on the host systems. Ex: VMware HA (High Availability) system that can restart a guest system after failure.
3. An **independent cluster manager**, which can be used on both the host and the guest – making the infrastructure complex.
4. Finally, we might also use an **integrated cluster (manager)**, on the guest and host operating systems; here the manager must clearly distinguish between physical and virtual resources.

**NOTE**: The virtual cluster management schemes are greatly enhanced if the VM life migration is enabled with minimum overhead.



* 1. Virtual clusters are generally used where fault tolerance of VMs on the host plays an important role in the total cluster strategy. These clusters can be applied in grids, clouds and HPC platforms. The HPC is obtained by dynamical finding and usage of resources as per requirement, and less migration time & bandwidth that is used.
  2. A VM can be in one of the following states:

1. **Inactive State**: This is defined by the VZ platform, under which the VM is **not** enabled.
2. **Active State**: This refers to a VM that has been instantiated at the VZ platform to perform a task.
3. **Paused State**: A VM has been instantiated but disabled temporarily to process a task or is in a waiting state itself.
4. **Suspended State**: A VM enters this state if its machine file and virtual resources are stored back to the disk.
   1. **Live Migration Steps**: This consists of 6 steps.
5. **Steps 0 and 1**: Start migration automatically and checkout load balances and server consolidation.
6. **Step 2**: Transfer memory (transfer the memory data + recopy any data that is changed during the process). This goes on iteratively till changed memory is small enough to be handled directly.
7. **Step 3**: Suspend the VM and copy the last portion of the data.
8. **Steps 4 and 5**: Commit and activate the new host. Here, all the data is recovered, and the VM is started from exactly the place where it was suspended, but on the new host.
   1. Virtual Clusters are being widely used to use the computing resources effectively, generate HP, overcome the burden of interaction between different OSs and make different configurations to coexist.
   2. **Memory Migration**: This is done between the physical host and any other physical/virtual machine. The techniques used here depend upon the guest OS. MM can be in a range of megabytes to gigabytes. The Internet Suspend-Resume (ISR) technique exploits temporal locality since the memory states are may have overlaps in the suspended/resumed instances of a VM. **Temporal locality (TL)** refers to the fact that the memory states differ only by the amount of work done since a VM was last suspended.

To utilize the TL, each file is represented as a tree of small sub-files. A copy of this tree exists in both the running and suspended instances of the VM. The advantage here is usage of tree representation of a file and caching ensures that the changed files are only utilized for transmission.

* 1. **File System Migration**: To support VM migration from one cluster to another, a consistent and location-dependent view of the file system is available on all hosts. Each VM is provided with its own virtual disk to which the file system is mapped to. The contents of the VM can be transmitted across the cluster by inter-connections (mapping) between the hosts. But migration of an entire host (if required) is not advisable due to cost and security problems. We can also provide a global file system across all host machines where a VM can be located. This methodology removes the need of copying files from one machine to another – all files on all machines can be accessed through network.

It should be noted here that the actual files are not mapped or copied. The VMM accesses only the local file system of a machine and the original/modified files are stored at their respective systems only. This decoupling improves security and performance but increases the overhead of the VMM – every file has to be stored in virtual disks in its local files.

* 1. **Smart Copying** ensures that after being resumed from suspension state, a VM doesn’t get a whole file as a backup. It receives only the changes that were made. This technique reduces the amount of data that has to be moved between two locations.
  2. **Network Migration:** A migrating should maintain open network connections. It should not depend upon forwarding mechanisms (mediators) or mobile mechanisms. Each VM should be assigned a unique IP or MAC (Media Access Control) [7] addresses which is different from that of the host machine. The mapping of the IP and MAC addresses to their respective VMs is done by the VMM.

If the destination of the VM is also on the same LAN, special messages are sent using MAC address that the IP address of the VM has moved to a new location. If the destination is on another network, the migrating OS can keep its original Ethernet MAC address and depend on the network switch [9] to detect its move to a new port [8].

* 1. Notethat **live migration means** moving a VM from one physical node to another while keeping its OS environment and apps intact. All this process is carried out by a program called **migration daemon**. This capability provides efficient online system maintenance, reconfiguration, load balancing, and improved fault tolerance. The recently improved mechanisms are able to migrate without suspending the concerned VM.
  2. There are two **approaches in live migration**: pre copy and post copy.

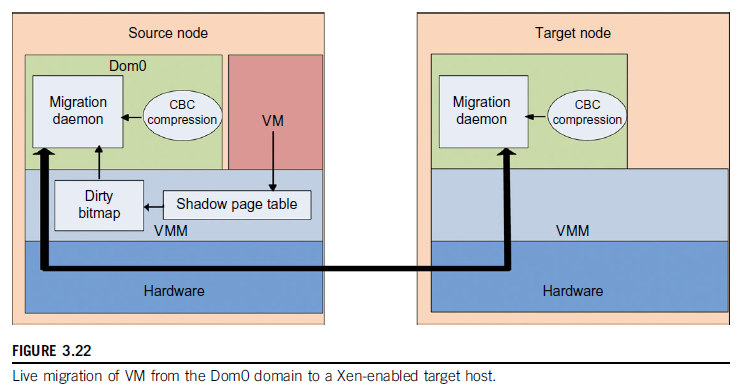
1. In **pre copy**, which is manly used in live migration, all memory pages are first transferred; it then copies the modified pages in the last round iteratively. Here, performance ‘degradation’ will occur because migration will be encountering dirty pages (pages that change during networking) [10] all around in the network before getting to the right destination. The iterations could also increase, causing another problem. To encounter these problems, check-pointing/recovery process is used at different positions to take care of the above problems and increase the performance.
2. In **post-copy**, all memory pages are transferred only once during the migration process. The threshold time allocated for migration is reduced. But the downtime is higher than that in pre-copy.

NOTE: **Downtime** means the time in which a system is out of action or can’t handle other works.

Ex: Live migration between two Xen-enabled hosts: Figure 3.22 [1]

CBC Compression=> Context Based Compression

RDMA=> Remote Direct memory Access



1. **VZ for Data Centre Automation:** Data Centres have been built and automated recently by different companies like Google, MS, IBM, Apple etc. By utilizing the data centres and the data in the same, VZ is moving towards mobility, reduced maintenance time, and increasing the number of virtual clients. Other factors that influence the deployment and usage of data centres are high availability (HA), backup services, and workload balancing.
   1. **Server Consolidation in Data Centres**: In data centers, heterogeneous workloads may run at different times. The two types here are
2. **Chatty (Interactive) Workloads**: These types may reach the peak at a particular time and may be silent at some other time. Ex: WhatsApp in the evening and the same at midday.
3. **Non-Interactive Workloads**: These don’t require any users’ efforts to make progress after they have been submitted. Ex: HPC

The data center should be able to handle the workload with satisfactory performance both at the peak and normal levels.

It is common that much of the resources of data centers like hardware, space, power and cost are under-utilized at various levels and times. To come out of this disadvantage, one approach is to use the methodology of **server consolidation**. This improves the server utility ratio of hardware devices by reducing the number of physical servers. There exist two types of server consolidation: (a) **Centralised and Physical Consolidation** (b) **VZ based server consolidation**. The second method is widely used these days, and it has some advantages.

* Consolidation increases hardware utilization
* It enables more agile provisioning of the available resources
* The total cost of owning and using data center is reduced (low maintenance, low cooling, low cabling etc.)
* It enables availability and business continuity – the crash of a guest OS has no effect upon a host OS.
  1. **NOTE**: To automate (VZ) data centers one must **consider several factors** like resource scheduling, power management, performance of analytical models and so on. This improves the utilization in data centers and gives high performance. Scheduling and reallocation can be done at different levels at VM level, server level and data center level, but generally **any one (or two) level is used** at a time.

The schemes that can be considered are:

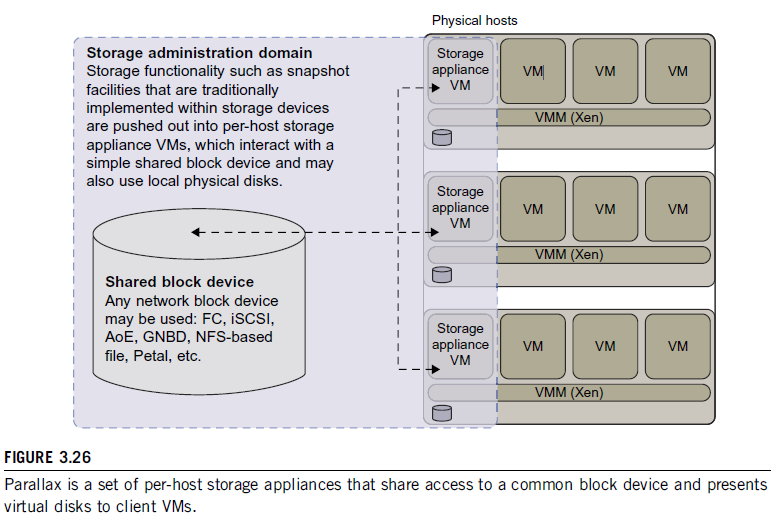
1. **Dynamic CPU allocation**: This is based on **VM utilization** and app level **QoS** [11] (Quality of Service) metrics. The CPU should adjust automatically according to the demands and workloads to deliver the best performance possible.
2. **Another scheme** uses two-level resource management system to handle the complexity of the requests and allocations. The resources are allocated automatically and autonomously to bring down the workload on each server of a data center.

Finally, we should efficiently balance the power saving and data center performance to achieve the HP and HT also at different situations as they demand.

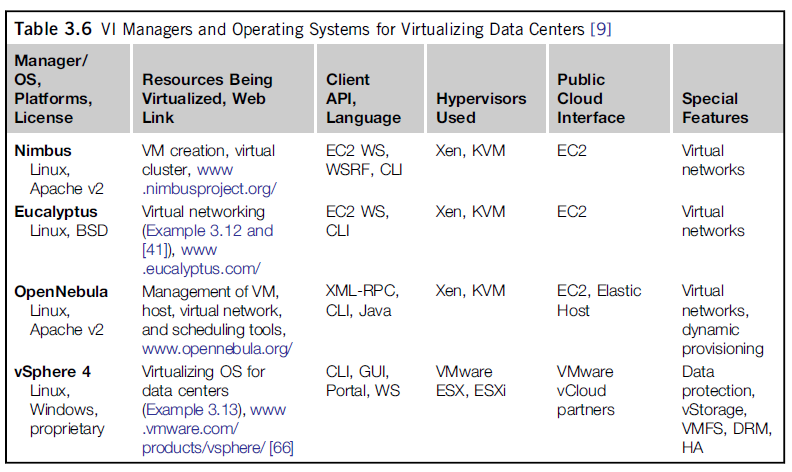
* 1. **Virtual Storage Management**: VZ is mainly lagging behind the modernisation of data centers and is the bottleneck of VM deployment. The CPUs are rarely updates, the chips are not replaced and the host/guest operating systems are not adjusted as per the demands of situation.

Also, the storage methodologies used by the VMs are not as fast as they are expected to be (nimble). Thousands of such VMs may flood the data center and their lakhs of images (SSI) may lead to data center collapse. Research has been conducted for this purpose to bring out an efficient storage and reduce the size of images by storing parts of them at different locations. The solution here is Content Addressable Storage (CAS). Ex: Parallax system architecture (A distributed storage system). This can be viewed at Figure 3.26 [1], P25.

* 1. Note that Parallax itself runs as a user-level application in the VM storage, providing Virtual Disk Images (VDIs). A **VDI** can accessed in a transparent manner from any host machine in the Parallax cluster. It is a core abstraction of the storage methodology used by Parallax.



* 1. **Cloud OS for VZ Data Centers**: VI => Virtual Infrastructure managers Types can be seen in Table 3.6 [1].



* 1. EC2 => Amazon Elastic Compute Cloud

WS => Web Service

CLI => Command Line Interface

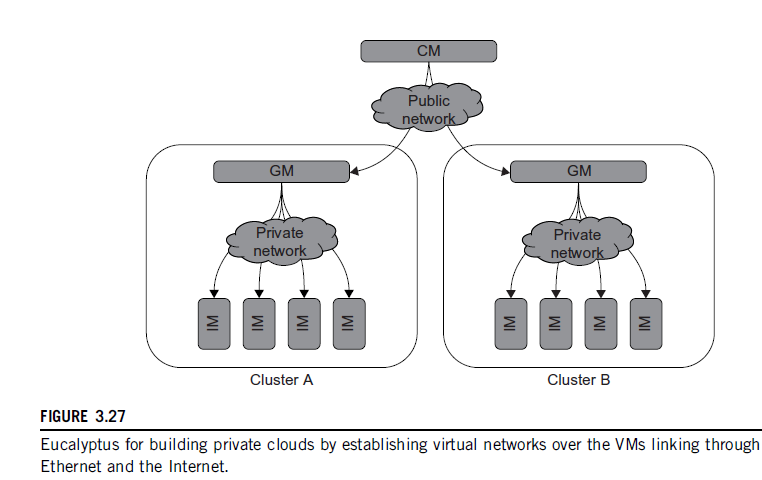
WSRF => Web Services Resource Framework

KVM => Kernel-based VM

VMFS => VM File System

HA => High Availability

* 1. Example of Eucalyptus for Virtual Networking of Private Cloud: It is an open-source software system intended for IaaS clouds. This is seen in Figure 3.27 [1].



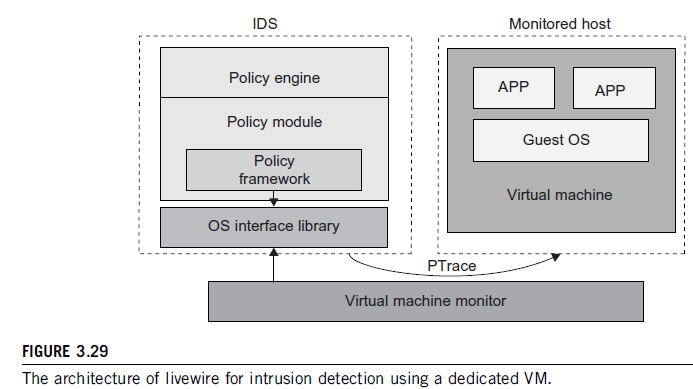
* 1. **Instance Manager** (IM): It controls execution, inspection and terminating of VM instances on the host machines where it runs.

**Group Manager** (GM): It gathers information about VM execution and schedules them on specific IMs; it also manages virtual instance network.

**Cloud Manager** (CM): It is an entry-point into the cloud for both users and administrators. It gathers information about the resources, allocates them by proper scheduling, and implements them through the GMs.

* 1. **Trust Management in VZ Data Centers**: As a recollect, a VMM (hypervisor) is a layer between the host OS and the hardware to create 1 or more VMs on a single platform. A VM encapsulates the guest OS and its current state and can transport it through the network as a SSI. At this juncture, in the network transportation, any intruders may get into the image or the concerned hypervisor itself and pose danger to both the image and the host system. Ex: A subtle problem lies in reusing a random number for cryptography.
  2. **VM-based Intrusion Detection**: **Intrusions** are unauthorized access to a computer from other network users. An **intrusion detection system** (IDS), which is built on the host OS can be divided into two types: host-based IDS (HIDS) and a network-based IDS (NIDS).

VZ based IDS can isolate each VM on the VMM and work upon the concerned systems without having contacts with the other. Any problem with a VM will not pose problems for other VMs. Also, a VMM audits the hardware allocation and usage for the VMs regularly so as to notice any abnormal changes. Still yet, the host and guest OS are fully isolated from each other. A methodology on these bases can be noticed in Figure 3.29 [1].



The above figure proposes the concept of granting IDS runs only on a highly-privileged VM. Notice that policies play an important role here. A **policy framework** can monitor the events in different guest operating systems of different VMs by using an OS interface library to determine which grant is secure and which is not.

It is difficult to determine which access is intrusion and which is not without some time delay. Systems also may use access ‘logs’ to analyze which is an intrusion and which is secure. The IDS log service is based on the OS kernel and the UNIX kernel is hard to break; so even if a host machine is taken over by the hackers, the IDS log book remains unaffected.

The security problems of the cloud mainly arise in the transport of the images through the network from one location to another. The VMM must be used more effectively and efficiently to deny any chances for the hackers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**References**

1. Kai Hwang et al, Distributed and Cloud Computing – From Parallel Processing to the Internet of Things, Morgan Kaufmann, Elsevier, 2012.
2. <https://en.wikipedia.org/wiki/Multiplexing>
3. <http://cloudacademy.com/blog/container-virtualization/>
4. <https://en.wikipedia.org/wiki/Library_(computing)#Shared_libraries>
5. <https://en.wikipedia.org/wiki/Binary_translation>
6. <https://pubs.vmware.com/vsphere-51/topic/com.vmware.vsphere.resmgmt.doc/GUID-C25A8823-F595-4322-BD0D-4FD5B081F877.html>
7. <http://searchnetworking.techtarget.com/definition/MAC-address>
8. <http://searchnetworking.techtarget.com/definition/port>
9. <https://en.wikipedia.org/wiki/Network_switch>
10. <https://en.wikipedia.org/wiki/Live_migration>
11. <http://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service>