Elective –V MCA 402 (2) Cloud Computing Technologies

UNIT IV Virtualization:

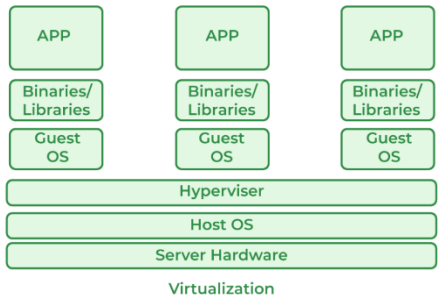
Contributed By: Om Shankar Mishra
0302CA221024

www.linkedin.com/in/omi92229

https://t.me/SolutionsAndTricksIT

Basics of Virtualization

Introduction to Virtualization:



- Virtualization is a technique that abstracts computing resources from the underlying physical hardware, enabling the creation of virtual instances of these resources.
- Initially developed during the mainframe era, virtualization has evolved to become a cornerstone technology in modern computing environments, especially in cloud computing.

Key Concepts:

Host Machine: The physical machine where virtual machines are created is known as the host machine.

Guest Machine: Virtual machines, also known as guest machines, run on the host machine.

Role of Virtualization in Cloud Computing:

- In cloud computing, virtualization allows for the efficient sharing of physical resources among multiple users or organizations.
- Cloud vendors leverage virtualization to offer Infrastructure-as-a-Service (IaaS) solutions, reducing costs for users while optimizing resource utilization.
- Virtualization facilitates the creation of virtual environments not only for executing applications but also for storage, memory, and networking.

Benefits of Virtualization:

Flexible Resource Allocation: Virtualization enables efficient allocation of resources, optimizing utilization and reducing waste.

Enhanced Development Productivity: Virtualized environments streamline development processes by providing easily reproducible environments.

Cost Reduction: By consolidating hardware resources and enabling resource sharing, virtualization lowers the cost of IT infrastructure.

Remote Access and Scalability: Virtualized environments support remote access and rapid scalability, allowing for dynamic adjustments to resource needs.

High Availability and Disaster Recovery: Virtualization enhances fault tolerance and disaster recovery capabilities through features like live migration and snapshotting.

Pay-per-Use Model: Users can pay for the IT infrastructure they use on-demand, leading to cost savings and flexibility.

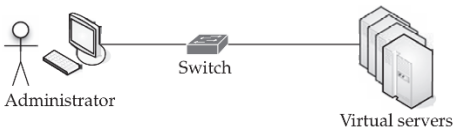
Support for Multiple Operating Systems: Virtualization allows for the simultaneous execution of multiple operating systems on a single physical machine.

Drawbacks of Virtualization:

High Initial Investment: Implementing virtualization infrastructure requires significant initial investment, although it leads to long-term cost savings.

Learning Curve: Transitioning to virtualized environments may require training or hiring staff with specialized skills, adding to operational costs.

Security Risks: Hosting data on third-party resources in virtualized environments can introduce security risks, requiring robust security measures to mitigate.



Characteristics of Virtualization:

Increased Security: Virtualization enables the creation of secure execution environments by controlling guest program execution transparently.

Managed Execution: Virtualization provides features such as sharing, aggregation, emulation, and isolation for efficient resource management.

Sharing: Virtualization allows the creation of separate computing environments within the same host, enabling resource sharing among multiple virtual machines.

Aggregation: Virtualization facilitates resource aggregation, enabling the efficient utilization of physical resources among multiple virtual machines.

Uses of Virtualization:

Data integration: Integrating data from disparate sources into a unified view.

Business integration: Supporting business processes and workflows through integrated virtualized environments.

Service-oriented architecture data services: Providing data services within a service-oriented architecture for efficient data access and utilization.

Organizational data search: Facilitating remote access and searchability of organizational data through virtualized data environments.

~ Om Shankar Mishra

Types of Virtualization

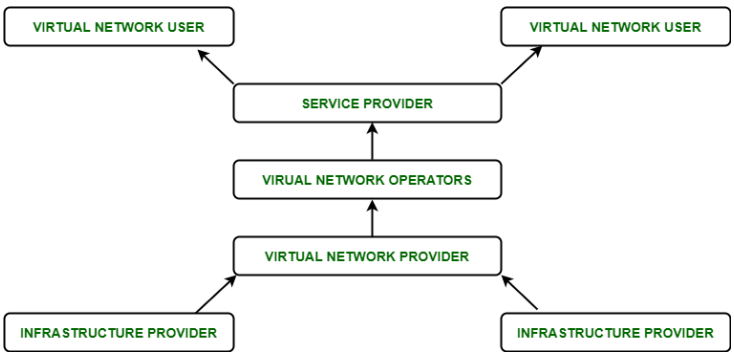
1. Application Virtualization:

Description: Allows remote access to applications hosted on servers while preserving personal settings and characteristics.

Example: Running multiple versions of the same software on a local workstation by accessing them from a server.

Technologies: Hosted applications, packaged applications.

2. Network Virtualization:



Description: Enables the operation of multiple virtual networks with separate control and data planes on a single physical network.

Benefits: Provides flexibility in network provisioning, logical segmentation, and workload security.

Components: Virtual networks, logical switches, routers, firewalls, load balancers, VPNs.

3. Desktop Virtualization:

Description: Stores users' operating systems on remote servers in data centers, allowing access to desktops from any location and device.

Benefits: Enables user mobility, portability, and centralized management of software installation and updates.

Use Cases: Users needing specific operating systems, remote access to desktop environments.

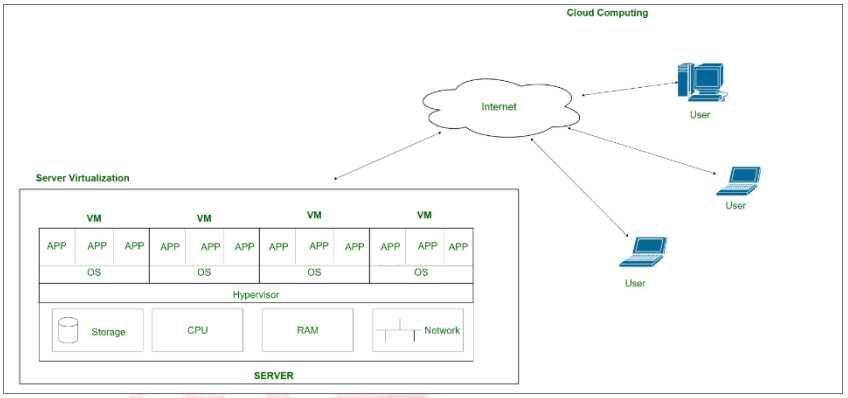
4. Storage Virtualization:

Description: Manages a pool of physical storage servers as a single virtual storage system, abstracting the underlying hardware.

Benefits: Simplifies storage management, improves utilization, and ensures consistent performance across heterogeneous storage environments.

Functionality: Smooth operations, consistent performance, advanced features.

5. Server Virtualization:



Description: Divides a physical server into multiple virtual servers, each running its own isolated operating system.

Benefits: Increases resource utilization, reduces costs, and enhances flexibility in resource allocation.

Features: Masking of server resources, isolation of operating systems, efficient resource utilization.

6. Data Virtualization:

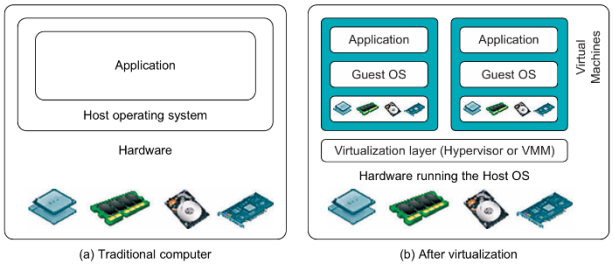
Description: Integrates data from diverse sources into a unified virtual view, accessible remotely via cloud services.

Benefits: Simplifies data integration, supports business integration, and enables service-oriented architecture data services.

Providers: Oracle, IBM, AtScale, CData, among others.

Use Cases: Data integration, business intelligence, service-oriented architecture.

Implementation Levels of Virtualization



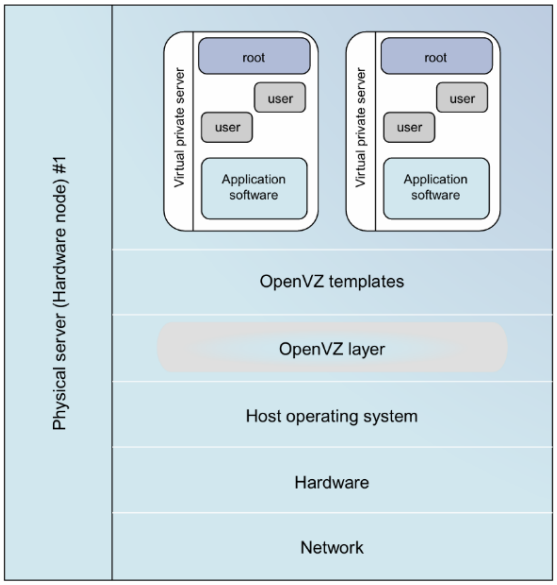
(a) Traditional computer (b) After virtualization

Virtualization is a technology that enables multiple virtual machines (VMs) to run on a single physical machine, allowing each VM to operate independently with its own operating system (guest OS). This is achieved by adding a software layer called a hypervisor or virtual machine monitor (VMM) on top of the host operating system (OS)

Instruction Set Architecture (ISA) Level:

- At this level, virtualization is achieved by emulating the instruction set architecture of the guest machine using the instruction set architecture of the host machine. This emulation can be done through techniques like code interpretation or dynamic binary translation.
- Virtualization at this level requires a software translation layer to translate source instructions to target instructions, ensuring compatibility between the guest and host architectures.

Hardware Abstraction Level:



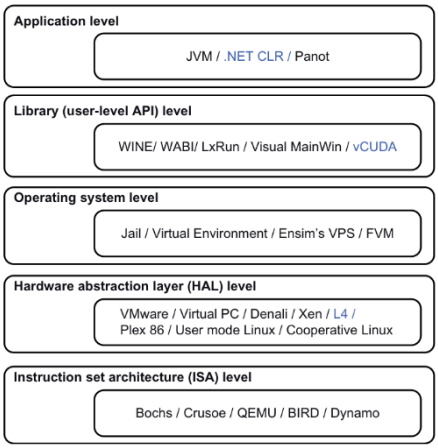
- Virtualization at the hardware abstraction level involves creating a virtual hardware environment for virtual machines (VMs) directly on top of the physical hardware.
- The goal is to abstract and virtualize physical resources such as processors, memory, and I/O devices to enable multiple VMs to share and utilize the hardware efficiently.
- This level of virtualization typically involves a hypervisor or virtual machine monitor (VMM) that manages the allocation of physical resources to VMs.

Operating System Level:

- Operating system-level virtualization, also known as containerization, creates isolated containers on a single physical server, each with its own operating system instance.
- Containers share the host operating system's kernel but have separate user spaces, allowing them to behave like independent servers.
- This approach is commonly used in cloud environments for creating lightweight and fast-starting instances, such as in Docker containers.

Library Support Level:

- Virtualization at the library support level involves controlling communication between applications and the rest of the system through library interfaces.
- Examples include API hooks used by software tools like WINE to support running Windows applications on UNIX hosts or libraries like vCUDA for GPU acceleration in virtualized environments.



User-Application Level:

- At the user-application level, virtualization focuses on virtualizing individual applications or processes rather than entire operating systems.
- This can be achieved through techniques like deploying high-level language virtual machines (HLL VMs) that run programs compiled to a specific abstract machine definition.
- Examples include the Microsoft .NET Common Language Runtime (CLR) and the Java Virtual Machine (JVM), which allow applications written in their respective languages to run on any platform that supports the VM.

Server-level virtualization: Also known as hardware virtualization, server-level virtualization involves partitioning a physical server into multiple virtual machines (VMs), each capable of running its own operating system and applications independently. This is typically achieved through a hypervisor, which abstracts the physical hardware and allocates resources to each VM.

Server-level virtualization allows for efficient resource utilization and isolation between VMs, making it suitable for consolidating multiple workloads onto a single physical server.

Block-level virtualization: Block-level virtualization operates at the storage level, abstracting physical storage devices (such as hard drives or SSDs) into virtualized blocks that can be dynamically allocated to different systems or applications. This is commonly implemented through technologies like storage area networks (SANs) or network-attached storage (NAS), where storage resources are pooled and presented as logical volumes to connected systems. Block-level virtualization provides flexibility and scalability in managing storage resources, enabling features like thin provisioning, snapshots, and replication.

File-level virtualization: File-level virtualization operates at the file system level, abstracting files and directories from underlying storage devices. It allows for logical grouping and management of files independent of physical storage locations. Network File System (NFS) and Common Internet File System (CIFS) are examples of file-level virtualization protocols that enable remote access to files over a network. File-level virtualization simplifies file management and sharing across heterogeneous environments, supporting features like access control, file locking, and file system abstraction.

Virtualization Structures

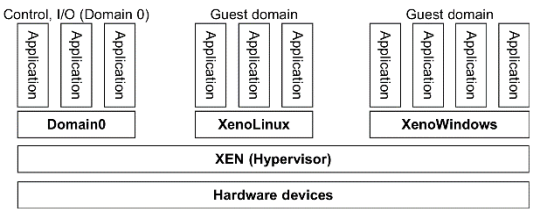
Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the OS. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Depending on the position of the virtualization layer, there are several classes of VM architectures.

1. Hypervisor Architecture: This architecture supports hardware-level virtualization by inserting a hypervisor or virtual machine monitor (VMM) directly between the physical hardware and the operating system (OS). The hypervisor facilitates the creation and management of VMs by converting portions of real hardware into virtual hardware. Two common types of hypervisor architectures are micro-kernel and monolithic hypervisors.

a. Micro-kernel Hypervisor: It includes only basic and unchanging functions, such as physical memory management and processor scheduling. Device drivers and other changeable components are outside the hypervisor, resulting in a smaller hypervisor code size.

b. Monolithic Hypervisor: It implements all functions, including device drivers, within the hypervisor itself. As a result, the code size of a monolithic hypervisor is larger than that of a micro-kernel hypervisor.

2. Xen Architecture: Xen is an open-source micro-kernel hypervisor that separates policy from mechanism. In the Xen architecture, the hypervisor implements all mechanisms, leaving policy to be handled by Domain 0. Key components of the Xen system include the hypervisor, kernel, and applications. Domain 0, also known as the privileged guest OS, manages hardware resources and controls other VMs (Domain U). Xen does not include device drivers natively; instead, it provides



a mechanism for guest OSes to have direct access to physical devices. Commercial Xen hypervisors are being developed by vendors such as Citrix XenServer and Oracle VM.

3. Host-Based Virtualization: In this architecture, the virtualization layer operates as a part of the host operating system. VMs are created and managed within the host OS environment. This approach is often used in desktop virtualization scenarios.

Tools and Mechanisms

~ Om Shankar Mishra

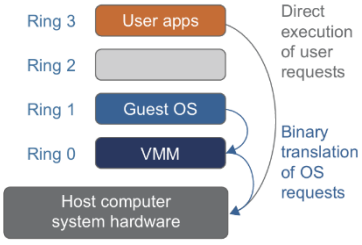
1. Full Virtualization:

- In full virtualization, the guest operating systems (OSes) and their applications run on virtual machines (VMs) without the need to modify the host OS. Instead, the virtualization layer employs binary translation to trap and virtualize the execution of certain sensitive, non-virtualizable instructions.
- Critical instructions that control hardware or threaten system security are replaced with traps into the virtual machine monitor (VMM) to be emulated by software. Non-critical instructions, on the other hand, run directly on the hardware.
- VMware is one of the software companies that have implemented full virtualization using binary translation. The VMM scans the instruction stream and identifies privileged, control-, and behavior-sensitive instructions, trapping them for emulation.

2. Host-Based Virtualization:

Definition: In host-based virtualization, a virtualization layer is installed on top of the host operating system. The host OS continues to manage the hardware, while guest OSes run on top of the virtualization layer.

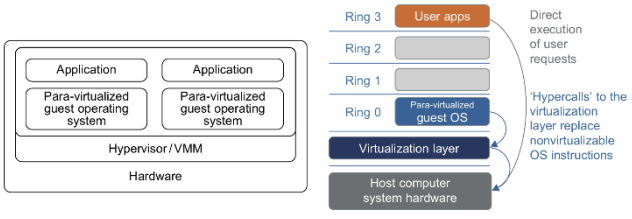
Flexibility and Performance: Host-based virtualization allows for greater flexibility as it doesn't require modifications to the host OS. However, the performance may be lower compared to full virtualization due to additional layers of mapping and potential ISA mismatches.



Implementation: The virtualization layer interacts with the host OS to provide device drivers and low-level services, simplifying VM design and deployment.

3. Para-Virtualization:

- Para-virtualization requires modification of the guest OS to provide special APIs and assistance for virtualization. It aims to reduce virtualization overhead and improve performance by modifying only the guest OS kernel.



- The virtualization layer is inserted between the hardware and the OS, typically at Ring 0 in the x86 processor architecture. Non-virtualizable instructions in the guest OS are replaced with hypercalls that communicate directly with the hypervisor or VMM.
- While para-virtualization reduces overhead, it may face compatibility and portability challenges and require deep OS kernel modifications.

- KVM (Kernel-Based VM) is a Linux para-virtualization system that is part of the Linux kernel version 2.6.20. It simplifies virtualization by letting the existing Linux kernel handle memory management and scheduling activities.

Virtualization of CPU

CPU virtualization involves enabling a virtual machine (VM) to execute most of its instructions directly on the host processor in native mode, enhancing efficiency. However, critical instructions must be handled carefully to ensure correctness and stability.

Critical Instructions Categories:

Privileged Instructions: These instructions execute in a privileged mode and must be trapped if executed outside this mode to prevent unauthorized access to critical resources.

Control-Sensitive Instructions: These instructions attempt to change resource configurations and require careful handling to maintain system stability.

Behavior-Sensitive Instructions: Instructions with different behaviors based on resource configurations, such as load and store operations over virtual memory, need to be managed appropriately.

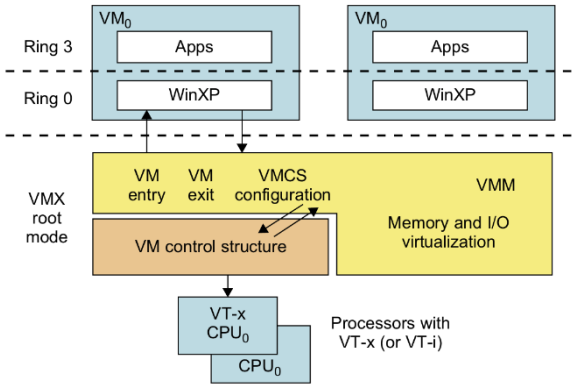
Virtualizability of CPU Architectures:

- A CPU architecture is considered virtualizable if it supports running both privileged and unprivileged instructions of VMs in the CPU's user mode while the Virtual Machine Monitor (VMM) operates in supervisor mode.
- RISC CPU architectures are naturally virtualizable because all control- and behavior-sensitive instructions are privileged instructions, simplifying virtualization.
- In contrast, x86 CPU architectures pose challenges for virtualization because some sensitive instructions are not privileged, complicating the trapping process in the VMM.

Paravirtualization:

- In a paravirtualization system like Xen, system calls in the guest OS trigger interrupts in both the guest OS and the hypervisor simultaneously.
- This allows the hypervisor to handle critical operations while still enabling unmodified applications to run in the VM, albeit with a slight performance penalty.

Hardware-Assisted CPU Virtualization:



- Hardware-assisted virtualization aims to simplify virtualization by introducing additional modes in the processor, such as privilege mode level (Ring-1), which allows the hypervisor to trap all privileged and sensitive instructions automatically.
- Intel's VT-x technology is an example of hardware-assisted virtualization, providing support for efficient trapping and handling of critical instructions by the hypervisor.

Efficiency Considerations:

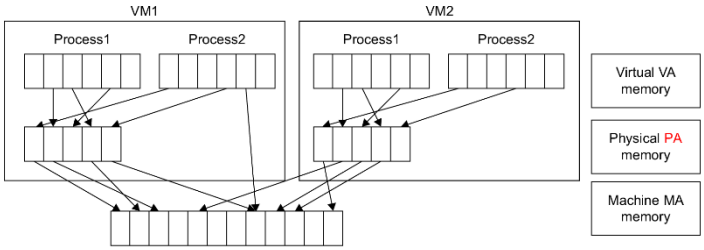
- While hardware-assisted virtualization is generally efficient, transitions between the hypervisor and the guest OS may incur overhead due to mode switches.

- Some virtualization systems, like VMware, adopt a hybrid approach combining hardware assistance with software-based techniques to optimize performance.
- Combining paravirtualization and hardware-assisted virtualization can further enhance performance by leveraging the strengths of both approaches.

Memory Virtualization

Memory virtualization involves managing physical system memory in a way that is transparent to the guest operating systems (OSes) running in virtual machines (VMs).

Traditional vs. Virtual Execution Environment:



- In a traditional execution environment, the operating system manages mappings from virtual memory to physical memory using page tables, typically involving a one-stage mapping.
- In a virtual execution environment, virtual memory virtualization involves dynamically allocating physical system memory to VMs, necessitating a two-stage mapping process: virtual memory to physical memory and physical memory to machine memory.

Memory Management in Virtualization:

- Guest OSes and the Virtual Machine Monitor (VMM) maintain separate mappings: the guest OS controls virtual-to-physical memory mappings, while the VMM maps guest physical memory to actual machine memory.
- This requires support for MMU virtualization, allowing the VMM to handle physical memory mapping transparently to the guest OS.

Shadow Page Tables and Nested Paging:

- To support memory virtualization, shadow page tables are used to perform virtual-memory-to-machine-memory address translation.
- Nested paging, as implemented in technologies like AMD Barcelona processors, adds another layer of indirection to virtual memory translation.

Hardware-Assisted Techniques:

- Intel developed Extended Page Tables (EPT) as a hardware-based technique to improve memory virtualization efficiency.
- EPT allows the CPU to efficiently translate virtual addresses to physical addresses by maintaining separate page tables for the guest OS and the VMM.
- Additionally, Intel offers Virtual Processor ID (VPID) to improve Translation Lookaside Buffer (TLB) efficiency, further enhancing memory virtualization performance.

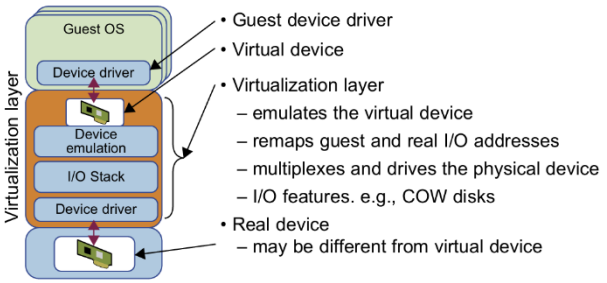
Memory Access Procedure:

- When a virtual address needs translation, the CPU first looks for the corresponding entry in the EPT TLB and then in the EPT.
- If the translation is not found, the CPU generates an exception, and the guest OS handles the interrupt.

- The CPU iteratively accesses the EPT to obtain translations, with Intel optimizing performance by increasing the EPT TLB size to minimize memory accesses.

I/O Virtualization

I/O device virtualization involves managing the routing of I/O requests between virtual devices within virtual machines (VMs) and the shared physical hardware. This process ensures that each VM can access and utilize I/O devices effectively without interfering with other VMs. There are several approaches to implementing I/O virtualization:



Full Device Emulation:

- This approach emulates real-world devices within the virtualization layer.
- All functions of a device or bus infrastructure, such as device enumeration, interrupts, and Direct Memory Access (DMA), are replicated in software.
- I/O access requests from guest OSes are trapped in the Virtual Machine Monitor (VMM), which interacts with the physical I/O devices.
- While versatile, full device emulation tends to be slower than native hardware due to software overhead.

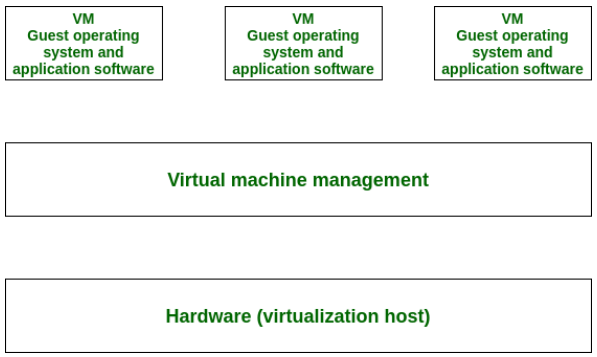
Para-virtualization:

- This method, used in systems like Xen, employs a split driver model with frontend and backend drivers.
- The frontend driver operates within guest VMs and manages I/O requests, while the backend driver, located in the host domain, interfaces with physical I/O devices.
- Both drivers communicate via shared memory to handle I/O operations efficiently.
- Para-virtualization typically offers better device performance compared to full device emulation but may incur higher CPU overhead.

Direct I/O Virtualization:

- This approach allows VMs to access I/O devices directly, achieving performance close to native levels without significant CPU overhead.
- Direct I/O virtualization is particularly useful for networking on mainframes but presents challenges for commodity hardware devices.
- One major challenge is managing the state of physical devices, especially during workload migration, to prevent system instability or crashes.

OS Virtualization



Operating system-based virtualization, also known as containerization, enables the existence of isolated user-space instances within an operating system environment. It involves the installation of virtualization software on a pre-existing operating system, which

serves as the host operating system. Here's a breakdown of key points and features of operating system-based virtualization:

Installation and Utilization:

- Users install virtualization software within their existing operating system.
- The host operating system provides support for hardware devices, potentially affecting hardware compatibility.
- Virtualization software allows users to create and manage multiple virtual machines directly.

Services Offered:

- Operating system-based virtualization offers various services, including backup and recovery, security management, and integration with directory services.
- These services leverage the capabilities and organizational tools available within the host operating system.

Operations:

- Hardware capabilities such as network connections and CPU can be utilized within virtual machines.
- Virtual machines can interact with connected peripherals such as webcams, printers, keyboards, or scanners.
- Data access, including reading/writing files, folders, and network shares, is facilitated by the host operating system.

Resource Isolation:

- Operating system-based virtualization provides high-level resource isolation, allowing each container to have its own set of resources, including CPU, memory, and I/O bandwidth.
- Containers share the same host operating system, resulting in lightweight and highly portable environments.
- Resource isolation ensures security by isolating containerized applications from each other and the host OS.

Performance Considerations:

- Performance overhead may occur due to hardware resource utilization by the host operating system.
- Hardware-related calls from guest operating systems may encounter multiple layers, impacting overall performance.
- Licensing requirements for both host and guest operating systems may add to operational costs.

Advantages:

- Resource efficiency is improved as containers do not need to emulate a complete hardware environment.
- Containers offer high scalability, easy management, reduced costs, faster deployment, and portability.

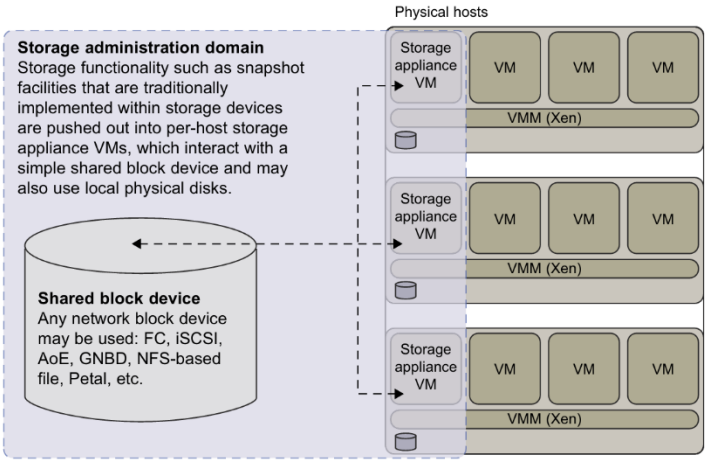
Disadvantages:

- Security risks may arise due to shared resources within containers and potential breaches affecting other containers.
- Limited isolation between applications may lead to performance degradation or resource contention.
- Complexity in setup and management, dependency issues, and limited hardware access are other challenges.

Virtualization for Data-center Automation

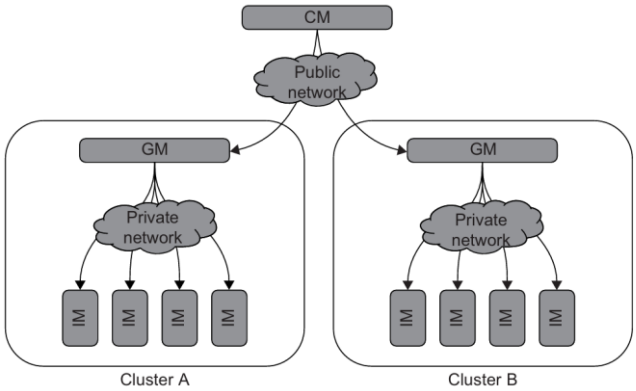
Data-center automation has become increasingly vital as data centers continue to grow in scale and complexity. Virtualization plays a key role in this automation process, enabling dynamic allocation of resources to meet the demands of millions of internet users while ensuring quality of service (QoS) and cost-effectiveness.

Server Consolidation: Data centers often run heterogeneous workloads, leading to underutilization of servers. Server consolidation, particularly through virtualization, is a powerful approach to improving hardware utilization. By consolidating multiple underutilized servers into fewer physical servers, resources can be more efficiently utilized. Virtualization allows for finer resource allocation compared to traditional server deployments.

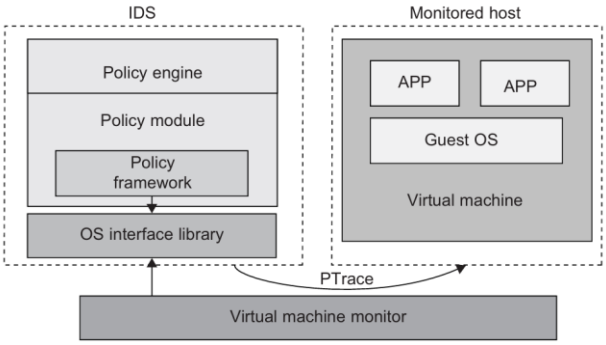


Virtual Storage Management: Storage virtualization is crucial in virtualized environments. Traditional storage management techniques may not fully address the needs of virtualized data centers. Virtual storage includes both VM images and application data, and managing storage in a virtualized environment requires specialized approaches to ensure performance, efficiency, and scalability.

Cloud OS for Virtualized Data Centers: Cloud operating systems (OS) are essential for managing compute, storage, and network resources in virtualized data centers. Various virtual infrastructure (VI) managers and OSes, such as Nimbus, Eucalyptus, OpenNebula, and vSphere 4, are tailored for virtualizing data centers. These systems provide features like VM creation, virtual networking, resource management, and scalability, catering to the needs of private and public clouds.



Trust Management in Virtualized Data Centers:
Security is a critical aspect of data-center automation, especially in virtualized environments where VMs share the same hardware platform. Trust management involves ensuring the integrity and security of VMs, preventing unauthorized access, and detecting intrusions. Techniques such as VM-based intrusion detection, honeypots, and trusted zones help mitigate security risks and protect virtual clusters from threats.



Introduction to MapReduce

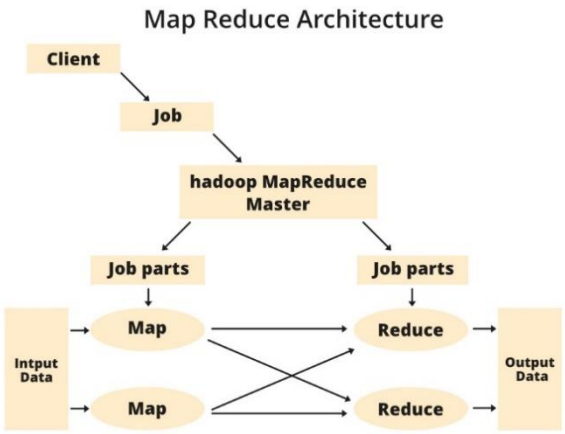
MapReduce is a powerful programming model used for efficient processing of large datasets in a distributed manner, typically within the Hadoop ecosystem.

Components of MapReduce Architecture:

a. Client: The client submits MapReduce jobs for processing to the Hadoop MapReduce manager. Multiple clients can submit jobs continuously.

b. Job: A MapReduce job represents the actual work that the client wants to perform. It consists of multiple smaller tasks that need to be processed or executed.

c. Hadoop MapReduce Master: This component divides the job into smaller, manageable job parts, also known as task or sub-jobs.



d. Job-Parts: These are the individual tasks obtained after dividing the main job. The results of all job parts are combined to produce the final output.

e. Input Data: The dataset provided to MapReduce for processing.

f. Output Data: The final result obtained after processing.

MapReduce Phases:

Map Phase: In this phase, the input data is mapped into key-value pairs. The Map function processes each input key-value pair and generates intermediate key-value pairs as output. The intermediate output is then fed into the Reduce phase.

Reduce Phase: In the Reduce phase, the intermediate key-value pairs generated by the Map phase are shuffled, sorted, and sent to the Reduce function. The Reduce function aggregates or groups the data based on the key-value pairs and performs the necessary computations to produce the final output.

Job Tracker and Task Tracker:

Job Tracker: The Job Tracker manages all the resources and jobs across the cluster. It schedules each map task on the Task Tracker running on the same data node. The Job Tracker is responsible for coordinating and monitoring the execution of MapReduce jobs.

Task Tracker: Task Trackers are deployed on each node in the cluster and execute Map and Reduce tasks as instructed by the Job Tracker. They are responsible for processing the individual tasks and reporting the progress back to the Job Tracker.

Job History Server: The Job History Server is a daemon process that stores historical information about MapReduce jobs, such as logs generated during or after job execution. It provides a centralized location for accessing job-related information and logs.

GFS

Google File System (GFS) was purpose-built to meet the unique requirements of Google's search engine, handling massive amounts of data on inexpensive, commodity hardware.

Purpose of GFS:

Storage for Web Data: GFS was designed to store the vast amounts of web data crawled and saved by Google's search engine.

Handling Component Failures: GFS addresses the high component failure rate typical in cloud computing environments by storing data redundantly across multiple cheap and unreliable computers.

Support for Large Files: GFS accommodates large files, with file sizes typically ranging from 100MB to several GB. To handle such large files efficiently, GFS uses a block size of 64MB, significantly larger than traditional file systems.

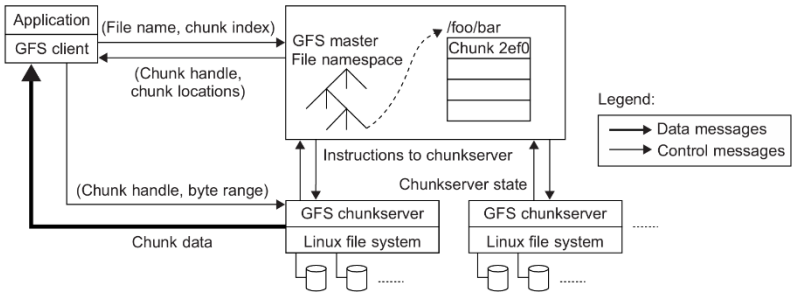
Design Decisions of GFS:

64MB Block Size: GFS uses a block size of 64MB to optimize performance for large file operations. Reliability through Replication: Data reliability is ensured through replication, with each chunk or data block of a file replicated across more than three chunk servers.

Single Master Architecture: GFS employs a single master to coordinate access and manage metadata. This simplifies the design and management of the system, as developers do not need to deal with complex distributed system issues like distributed consensus.

Customized API: GFS provides a customized API, offering additional operations such as snapshot and record append to facilitate Google's applications.

Architecture of GFS:

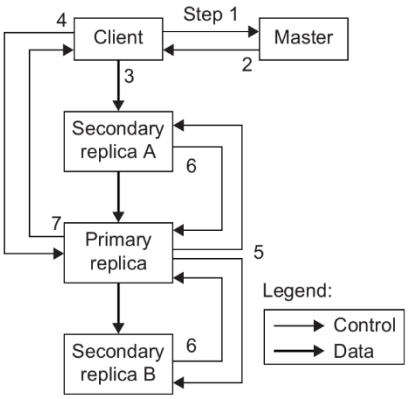


Single Master: The GFS architecture includes a single master node responsible for managing metadata and coordinating operations across the cluster.

Chunk Servers: Other nodes in the cluster act as chunk servers, storing data blocks. These chunk servers communicate with the master periodically to provide management information and receive instructions.

File Namespace Management: The master manages the file system namespace and locking facilities, ensuring consistency and coordination among client operations.

Data Mutation in GFS: Data mutation, including write and append operations, in GFS involves several steps:



- The client interacts with the master to determine the locations of replicas.
- Data is pushed to all replicas, and the primary assigns serial numbers to mutations.
- The primary forwards the mutation to secondary replicas, which apply mutations in the same serial order.
- The primary replies to the client, handling any errors encountered during the process.

Fault Tolerance in GFS:

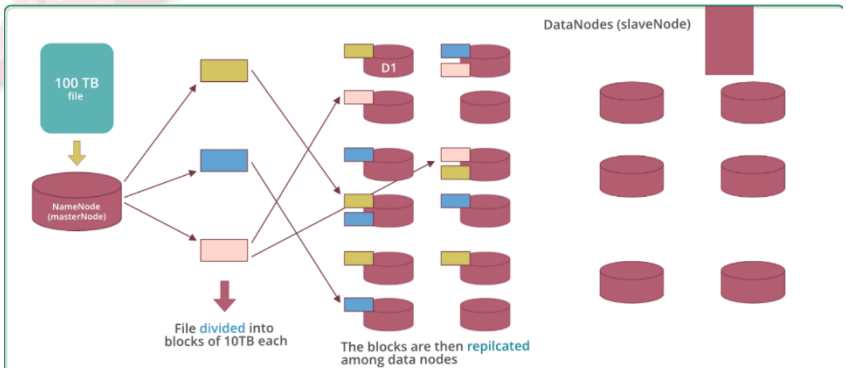
GFS achieves fault tolerance through fast recovery capabilities, replication of data across multiple servers, and checksums on data blocks to ensure data integrity.

HDFS

Hadoop Distributed File System (HDFS) is a key component of the Hadoop ecosystem, designed to handle the storage of extremely large files and datasets across a network of commodity hardware. Here's an overview of HDFS:

Extremely Large Files: HDFS is optimized for storing files in the range of petabytes (1000 TB) and beyond. It is capable of handling massive datasets that exceed the storage limits of a single machine.

Streaming Data Access Pattern: HDFS is built on the principle of write-once, read-many-times. Once data is written to HDFS, it can be accessed and processed multiple times without the need for repeated writes.



Commodity Hardware: HDFS is designed to run on inexpensive commodity hardware, making it cost-effective for organizations to store large volumes of data. This distinguishes it from other file systems that may require specialized hardware.

Nodes: HDFS clusters consist of master and slave nodes:

1. NameNode (Master Node): Manages the entire HDFS cluster and coordinates the storage and retrieval of data. It handles filesystem namespace operations and metadata management. The NameNode should be deployed on reliable hardware with high configuration.

2. DataNode (Slave Node): Act as worker nodes in the HDFS cluster, responsible for storing and retrieving data. They perform data storage, replication, and deletion operations as instructed by the NameNode. DataNodes can be deployed on commodity hardware.

~ Om Shankar Mishra

HDFS Daemons: These are background processes running on the nodes of the HDFS cluster:

NameNodes: Run on the master node and store metadata about the files stored in HDFS, such as file paths, block information, and block IDs. NameNodes require high amounts of RAM and store metadata in memory for fast retrieval, with a persistent copy kept on disk.

DataNodes: Run on slave nodes and are responsible for storing the actual data blocks. They require high memory resources as data is stored directly on them.

Data Storage in HDFS: HDFS stores data in a distributed manner by dividing large files into smaller blocks and distributing them across multiple DataNodes. Each block is replicated for fault tolerance, with the default replication factor being 3. The NameNode maintains information about the location and replication status of each block.

Key Terms Related to HDFS:

Heartbeat: Signal sent by DataNodes to the NameNode to indicate their status. If the NameNode doesn't receive a heartbeat from a DataNode, it considers it dead.

Balancing: Process of redistributing blocks across DataNodes to ensure even distribution and fault tolerance in case of DataNode failures.

Replication: Process of creating copies of data blocks across multiple DataNodes to achieve fault tolerance and high availability.

Features of HDFS:

- Distributed data storage.
- Reduced seek time through block storage.
- High availability and fault tolerance.
- Reliable data storage even in the event of multiple DataNode failures.

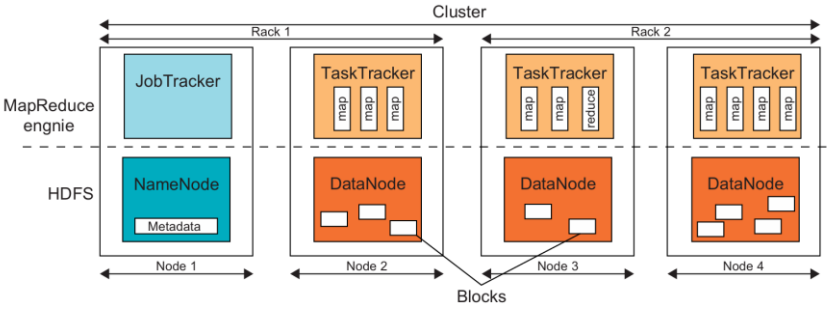
Limitations of HDFS:

- Not suitable for low-latency data access.
- Inefficient for handling large numbers of small files due to seek time and data movement overheads.

Hadoop Framework

- Hadoop is an open-source software framework used for storing and processing large datasets in a distributed computing environment.
- The MapReduce engine in Hadoop operates in conjunction with the HDFS to manage the execution of MapReduce jobs across distributed computing systems.
- Hadoop's journey began with the vision of Doug Cutting and Mike Cafarella, who co-founded the project under the umbrella of the Apache Software Foundation. Doug Cutting, one of the co-founders, named the project after his son's toy elephant, which he named Hadoop.

Architecture of MapReduce in Hadoop:



Master/Slave Architecture: The MapReduce engine follows a master/slave architecture with a single JobTracker as the master and multiple TaskTrackers as the slaves (workers).

- **JobTracker:** Manages MapReduce jobs over the cluster, monitors job progress, and assigns tasks to TaskTrackers.
- **TaskTracker:** Executes map and/or reduce tasks on computation nodes. Each TaskTracker node has multiple simultaneous execution slots, each supporting either a map or reduce task.

Running a Job in Hadoop:

Job Submission:

- A job is submitted from a user node to the JobTracker node within the cluster.
- The user node computes input file splits and requests a new job ID from the JobTracker.
- Resources such as the job's JAR file, configuration file, and computed input splits are copied to the JobTracker's file system.
- The job is submitted to the JobTracker using the submitJob() function.

Task Assignment:

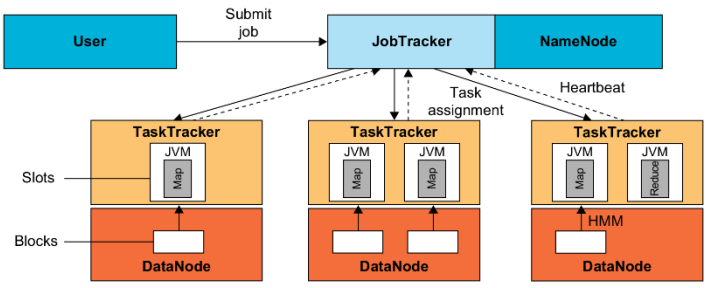
- The JobTracker creates map tasks for each input split and assigns them to TaskTrackers.
- Consideration is given to data locality when assigning map tasks.
- Reduce tasks are also created and assigned to TaskTrackers, with the number of reduce tasks predetermined by the user.

Task Execution:

- Inside each TaskTracker, the job JAR file is copied to the local file system, and instructions within the JAR file are executed.
- A Java Virtual Machine (JVM) is launched to run map or reduce tasks.

Task Running Check:

- TaskTrackers send periodic heartbeat messages to the JobTracker to indicate their status and readiness to run tasks.
- Heartbeat messages notify the JobTracker of the TaskTracker's availability and ability to accept new tasks.



~ Om Shankar Mishra