MODULE - 1

Of course, here is a detailed summary of your notes on Cloud Computing Module 1, structured to give you a comprehensive overview for your exam.

Module 1: Distributed System Models and Enabling Technologies

This summary covers the evolution of distributed computing, key technologies that enable modern systems, different system models, software environments, and crucial aspects like performance, security, and energy efficiency.

1. Scalable Computing Over the Internet

The foundation of modern cloud computing lies in the evolution from centralized computing to scalable, network-centric distributed systems. Instead of using a single large computer, distributed computing uses multiple computers over the Internet to solve large-scale problems.

1.1. Evolution of Computing Platforms:

- **Mainframes (1950s-1970s):** A few powerful, centralized computers like the IBM 360.
- **Minicomputers (1960s-1980s):** Lower-cost systems such as the DEC PDP 11.
- **Personal Computers (1970s-1990s):** Widespread use of PCs built with VLSI microprocessors.
- **Portable and Pervasive Devices (1980s-2000s):** Massive numbers of mobile devices in both wired and wireless applications.
- **Clusters, Grids, and Clouds (Since 1990):** Proliferation of High-Performance Computing (HPC) and High-Throughput Computing (HTC) systems.

1.2. Shift from High-Performance (HPC) to High-Throughput (HTC):

- **HPC** focuses on raw processing speed, measured in flops (floating-point operations per second), and is traditionally associated with supercomputers.
- **HTC**, in contrast, prioritizes the number of tasks completed over a period (throughput). HTC is vital for internet searches and web services that serve millions of users. This shift addresses challenges beyond speed, including **cost, energy efficiency, security, and reliability**.

1.3. Key Computing Paradigm Distinctions:

- **Centralized Computing:** All computer resources (processors, memory, storage) are located in a single physical system and are tightly integrated within one OS.
- **Parallel Computing:** Uses multiple processors that are either tightly coupled (with shared memory) or loosely coupled (with distributed memory) to execute parallel programs.
- **Distributed Computing:** A system of multiple autonomous computers, each with its own private memory, communicating over a network via message passing.

- **Cloud Computing:** An Internet-based system that can be centralized or distributed, using physical or virtualized resources from large data centers. It is often considered a form of utility or service computing.
- **Ubiquitous Computing & Internet of Things (IoT):** The concept of computing with pervasive devices anywhere, anytime. The IoT connects everyday objects, sensors, and computers, allowing them to be sensed and controlled remotely. It is supported by Internet clouds.

2. Technologies for Network-Based Systems

This section explores the hardware and software advancements that have enabled the growth of distributed and cloud computing.

2.1. Processor and Memory Technologies:

- **Multicore CPUs:** Modern processors integrate multiple cores (dual, quad, six, or more) to enhance parallelism at both the instruction (ILP) and task levels (TLP). Processor speeds have grown according to Moore's Law.
- **Many-Core GPUs:** GPUs have evolved from graphics accelerators to general-purpose computing powerhouses (GPGPU). With hundreds to thousands of lightweight cores, they excel at data-level parallelism (DLP) and are more power-efficient than CPUs.
- **Memory and Storage:** While DRAM capacity has grown exponentially, its speed has not kept pace, leading to the "memory wall problem". Storage has evolved from hard drives to faster Solid-State Drives (SSDs).
- **Networking:** Network speeds have increased dramatically from 10 Mbps Ethernet to 100 Gbps and beyond, which is crucial for efficient distributed systems.

2.2. Virtualization and Virtual Machines (VMs):

- **Virtualization** is a key enabling technology that abstracts hardware resources, allowing multiple operating systems (OS) to run on a single physical machine.
- **VM Architectures:**
    - **Native VM (Hypervisor-based):** A hypervisor runs directly on the hardware (e.g., VMware ESXi, Xen).
    - **Host VM (Software-based):** A VM runs as an application on top of a host OS (e.g., VirtualBox).
- **VM Operations:** VMs can be multiplexed (shared), suspended, resumed, and migrated across different physical servers, which enhances flexibility and resource utilization.

2.3. Data Center Virtualization:

- Cloud computing relies heavily on data center virtualization to provide scalable and cost-effective services.
- Modern data centers adopt a **low-cost design philosophy**, using commodity servers and letting software handle fault tolerance and load balancing.

- The convergence of technologies like **virtualization, multi-core processors, Service-Oriented Architecture (SOA), and autonomic computing** has enabled the rise of cloud computing.

3. System Models for Distributed and Cloud Computing

Distributed systems are built using different architectures, each suited for specific applications.

3.1. Clusters of Cooperative Computers:

- A cluster consists of interconnected stand-alone computers working cooperatively as a single resource. They are typically connected by high-bandwidth, low-latency networks like LANs.
- **Single-System Image (SSI):** The ideal for a cluster is to present itself as a single, powerful machine to the user. This is achieved through middleware or specialized OS support.

3.2. Grid Computing:

- Grids interconnect multiple, geographically distributed clusters via WANs to enable large-scale resource sharing. The term was coined in 1995.
- **Key difference from Clouds:** Grids enable access to *shared* computing power, while clouds provide access to *leased* computing power. Grids often use static resources, while clouds offer elastic, on-demand resources. Grids are open-source, whereas clouds are typically proprietary.

3.3. Peer-to-Peer (P2P) Networks:

- P2P systems are decentralized networks where client machines (peers) act as both clients and servers. They are self-organizing and use overlay networks for communication.
- They are well-suited for applications like file sharing and content delivery but face challenges in security, reliability, and data location.

3.4. Cloud Computing:

- Cloud computing delivers on-demand computing services over the Internet, leveraging massive, virtualized data centers.
- **Service Models:**
    - **Infrastructure as a Service (IaaS):** Provides virtualized computing resources like VMs and storage (e.g., Amazon EC2).
    - **Platform as a Service (PaaS):** Offers a platform for developers to build and deploy applications without managing the underlying infrastructure (e.g., Google App Engine).
    - **Software as a Service (SaaS):** Delivers software applications over the web on a subscription basis (e.g., Google Workspace).

- **Deployment Models:** Clouds can be private (for a single organization), public (available to anyone), managed, or a hybrid of both.

4. Software Environments for Distributed Systems and Clouds

This section describes the software architectures and programming models that underpin distributed applications.

4.1. Service-Oriented Architecture (SOA):

- SOA is an architectural style that enables the creation of applications built from modular, reusable software components called services. It is the foundation for web services, grids, and cloud computing.
- It is implemented using standards like SOAP (for web services) or lightweight alternatives like REST.

4.2. Distributed Operating Systems:

- While most distributed systems run independent OS instances on each node, a **truly distributed OS** aims to provide a Single-System Image (SSI) with full transparency, managing all resources coherently.
- Middleware solutions like **MOSIX2** for Linux clusters can enable features like seamless process migration across nodes, creating a more unified environment.

4.3. Parallel and Distributed Programming Models:

- **MPI (Message-Passing Interface):** A standard for writing parallel applications, especially in HPC, where processes communicate by explicitly sending and receiving messages.
- **MapReduce:** A programming model designed for processing and generating large datasets on clusters. It splits tasks into a "Map" phase (processing) and a "Reduce" phase (aggregation).
- **Hadoop:** An open-source framework that implements MapReduce for large-scale data processing. It provides distributed storage (HDFS) and is highly scalable and fault-tolerant.

5. Performance, Security, and Energy Efficiency

These are critical non-functional requirements for any large-scale distributed system.

5.1. Performance and Scalability:

- **Scalability** refers to a system's ability to handle an increasing workload by adding more resources. This can be measured through:
    - **Amdahl's Law:** States that speedup is limited by the sequential part of a program. It applies to fixed-workload scaling.

    - **Gustafson's Law:** A model for scaled-workload scenarios, where the problem size increases with the number of processors, often resulting in better speedup for large systems.

5.2. Fault Tolerance and Availability:

- **High Availability (HA)** is crucial and is measured by a system's uptime. It is improved by eliminating single points of failure through redundancy and fault isolation.
- Availability is calculated as: **Availability = MTTF / (MTTF + MTTR)**, where MTTF is Mean Time to Failure and MTTR is Mean Time to Repair.

5.3. Security and Data Integrity:

- Distributed systems face numerous threats, including loss of confidentiality, integrity, and availability (e.g., Denial of Service attacks).
- **Security is a shared responsibility** in the cloud. The responsibility shifts from the provider to the user as you move from SaaS to PaaS to IaaS.
- Defense technologies have evolved from prevention (access control) and detection (firewalls) to intelligent, AI-driven response systems.

5.4. Energy Efficiency:

- With massive data centers, energy consumption is a major concern due to both cost and environmental impact. A significant amount of energy is wasted by idle servers.
- Techniques to improve efficiency include:
    - Smart task scheduling and software optimization.
    - **Dynamic Voltage-Frequency Scaling (DVFS):** Reducing a CPU's voltage and clock frequency during idle periods to save power.

MODULE - 2

Of course. Here is a detailed summary of your notes on Cloud Computing Module 2, structured with headings and subheadings to provide a comprehensive overview for your exam.

Module 2: Virtual Machines and Virtualization of Clusters and Data Centers

This summary covers the fundamentals of virtualization, its various implementation levels, the mechanisms and tools used, how specific hardware components are virtualized, and its application in managing virtual clusters and automating data centers.

1. Introduction to Virtualization

Virtualization is the process of creating virtual instances of computing resources like servers, storage, networks, or operating systems on a single physical machine. This allows multiple independent virtual environments to share the same hardware, leading to significant improvements in resource utilization, scalability, and flexibility.

1.1. Traditional vs. Virtualized Computing: A traditional computer dedicates its hardware to a single OS and its applications, often leading to underutilized resources. In contrast, a virtualized computer allows multiple Operating Systems (e.g., Windows and Linux) to run simultaneously on the same physical machine, maximizing hardware efficiency. Virtualization provides key benefits such as:

- **Cost Savings:** Reduces hardware, power, and maintenance costs by running multiple virtual instances on fewer physical machines.
- **Better Resource Utilization:** Maximizes the use of shared CPU, RAM, and storage among various Virtual Machines (VMs).
- **Isolation and Security:** Each virtual instance is isolated, so a failure in one does not affect others.
- **Flexibility and Scalability:** Resources can be allocated dynamically without needing new hardware, and different OSes can run concurrently.
- **Efficient Deployment and Recovery:** VMs can be quickly deployed, cloned, and recovered using snapshots and templates.

1.2. The Virtualization Layer and VMM (Hypervisor):

- The **virtualization layer** is the software that abstracts physical hardware resources (CPU, memory, storage) and presents them as virtual resources to guest operating systems. It is responsible for abstraction, isolation, and dynamic resource allocation.
- A **Virtual Machine Monitor (VMM)**, also known as a **hypervisor**, is a specific component within the virtualization layer responsible for creating, managing, and running VMs. It directly manages VM access to hardware resources.
- While the VMM is focused on running VMs, the virtualization layer is a broader concept that includes the VMM along with other management tools, APIs, and virtual networking/storage components like Software-Defined Networking (SDN) and Software-Defined Storage (SDS).

2. Levels of Virtualization Implementation

Virtualization can be implemented at five different levels, from hardware instruction sets up to the application level.

2.1. Instruction Set Architecture (ISA) Level:

- This level focuses on **emulating** an instruction set of one architecture (source ISA) on a host machine with a different architecture (target ISA).

- This is achieved through techniques like **code interpretation** (translating instructions one-by-one, which is slow) or **dynamic binary translation** (translating and caching blocks of instructions, which is much faster).
- **Advantages:** Excellent for supporting legacy code and achieving cross-platform compatibility. It offers the best application flexibility.
- **Challenges:** High performance overhead and implementation complexity.

2.2. Hardware Abstraction Level (HAL):

- This involves a hypervisor (or VMM) running directly on the physical hardware (a "bare-metal" or Type 1 hypervisor) to create and manage VMs. The VMM is inserted between the hardware and the OS.
- It virtualizes resources like the CPU, memory, and I/O devices, allowing multiple, isolated OS instances to run concurrently on one machine.
- **Advantages:** High performance (near-native), strong fault isolation between VMs, and scalability.
- This level offers the highest performance but is also one of the most expensive and complex to implement.

2.3. Operating System (OS) Level:

- This method operates at the OS kernel layer, creating isolated user-space environments called **containers** or Virtual Environments (VEs).
- All containers share the same host OS kernel, which makes them very lightweight and efficient.
- **Advantages:** Very fast startup, low resource overhead, and near-native performance. Tools like OpenVZ for Linux support features like resource management and live migration.
- **Disadvantages:** All containers must use the same OS family (e.g., only Linux containers on a Linux host), and isolation is weaker than at the hardware level.

2.4. Library Support Level:

- This approach, also called middleware support, virtualizes the communication between an application and the OS by intercepting API calls.
- It allows "alien" programs to run on a host OS without virtualizing the entire system.
- **Examples:** WINE allows Windows applications to run on UNIX-based systems by translating Windows API calls. vCUDA enables applications in a VM to use a GPU on the host machine by virtualizing the CUDA library.

2.5. User-Application Level:

- This is also known as process-level virtualization and focuses on individual applications.
- A common implementation is the **High-Level Language (HLL) Virtual Machine**, like the Java Virtual Machine (JVM) or .NET CLR, which creates a platform-independent runtime environment.

- **Advantages:** Provides excellent cross-platform compatibility, simplified application deployment, and improved security through isolation (sandboxing).
- **Disadvantages:** Can introduce performance overhead and is limited to virtualizing single applications, not a full OS environment.

3. Virtualization Structures and Mechanisms

There are three main classes of VM architectures based on where the virtualization layer is placed.

3.1. Hypervisor Architecture (Type 1):

- The hypervisor runs directly on the hardware, managing physical resources and creating VMs. It can be **monolithic** (includes device drivers, e.g., VMware ESX) or **micro-kernel** (minimal core functions, e.g., Xen).
- **Xen** is a notable open-source micro-kernel hypervisor that uses a privileged VM called **Domain 0 (Dom0)** to manage hardware access for all other guest VMs (Domain U).

3.2. Full Virtualization with Binary Translation:

- This approach allows an unmodified guest OS to run in a VM. The VMM uses **binary translation** to trap and emulate privileged instructions that the guest OS tries to execute, while non-critical instructions run directly on the hardware for speed.
- It provides great compatibility but introduces performance overhead, especially for I/O-intensive tasks.

3.3. Para-Virtualization:

- This technique modifies the guest OS kernel to replace privileged instructions with **hypercalls** that communicate directly with the hypervisor.
- This avoids the overhead of binary translation and offers better performance, but it requires a modified OS, creating compatibility and maintenance challenges. KVM is a Linux-based system that supports para-virtualization.

3.4. Host-Based Architecture (Type 2):

- The virtualization layer runs as a standard application on top of a host OS. The host OS manages the hardware, and the virtualization software creates VMs.
- This is easier to deploy but is less efficient because I/O requests must pass through multiple software layers (guest OS, virtualization layer, host OS). VMware Workstation is a prime example.

4. Virtualization of CPU, Memory, and I/O Devices

4.1. CPU Virtualization:

- The goal is to run most instructions from the VM directly on the host CPU for performance while trapping and handling privileged instructions in the VMM.
- **Hardware-assisted virtualization** (Intel VT-x and AMD-V) simplifies this by adding a new processor mode (often called Ring -1) for the hypervisor, allowing the guest OS to run at Ring 0 without modification. All sensitive instructions are automatically trapped by the hardware, eliminating the need for binary translation or para-virtualization.

4.2. Memory Virtualization:

- In a virtualized system, a two-stage address translation is needed: from the guest's virtual address to the guest's physical address, and then from the guest's physical address to the machine's physical address.
- Initially, this was managed in software using **shadow page tables**, where the VMM maintained a separate page table for each VM. This caused significant overhead.
- Modern processors use **hardware-assisted memory virtualization** (Intel's EPT and AMD's NPT), which offloads this two-stage translation to the hardware's Memory Management Unit (MMU), greatly improving performance.

4.3. I/O Virtualization:

- This enables VMs to share physical I/O devices. Key methods include:
    - **Full Device Emulation:** The VMM emulates a real hardware device in software. This is compatible with any OS but is very slow due to high overhead.
    - **Para-Virtualization (Split Driver Model):** A "frontend" driver in the guest VM communicates with a "backend" driver in the hypervisor (or Dom0 in Xen) via shared memory. This offers better performance than emulation but has higher CPU overhead.
    - **Direct I/O Virtualization:** The VM is given direct access to a physical device, offering near-native performance. **Hardware support** like Intel VT-d enables this by handling DMA and interrupt remapping securely.

5. Virtual Clusters and Resource Management

A **virtual cluster** is a collection of VMs running across multiple physical machines, connected by a virtual network. They offer immense flexibility, allowing clusters to be deployed, resized, and migrated dynamically.

5.1. Live VM Migration:

- This is the process of moving a running VM from one physical host to another with minimal or no downtime.
- The process typically involves an **iterative pre-copy** of the VM's memory to the destination host, followed by a brief pause to transfer the final state (CPU registers and last-minute memory changes), and finally activating the VM on the new host.
- This is crucial for **load balancing**, **proactive maintenance**, and **fault tolerance** in cloud environments.

5.2. Resource Management:

- Managing virtual clusters requires efficient scheduling, fast deployment, and high-performance storage.
- Research projects like **Cluster-on-Demand (COD)** from Duke University and **VIOLIN** from Purdue University have demonstrated systems that can automatically provision, resize, and migrate virtual clusters to optimize resource usage and application performance.

6. Virtualization for Data Center Automation

Virtualization is the core technology for automating modern data centers, enabling the dynamic allocation of resources to meet fluctuating demand.

6.1. Server Consolidation:

- By running multiple VMs on fewer physical servers, data centers can significantly improve hardware utilization, which is often very low in traditional setups.
- This leads to major cost savings in hardware, power, cooling, and physical space.

6.2. Cloud OS and Management Platforms:

- Data centers use Virtual Infrastructure (VI) managers and Cloud OS platforms to manage virtualized resources at scale.
- **Open-source platforms** like Eucalyptus, OpenNebula, and Nimbus provide tools for creating and managing VMs, virtual networks, and private clouds. **Eucalyptus** is notable for its compatibility with Amazon Web Services (AWS) APIs.
- **Commercial platforms** like **VMware vSphere** offer a comprehensive suite of tools for managing virtual compute, storage, and networking, along with advanced features for high availability, security, and scalability.

6.3. Trust and Security Management:

- Security in a virtualized data center is critical. The **VMM (hypervisor) is the foundation of security**, as it enforces isolation between VMs. A compromised VMM puts the entire system at risk.
- **VM-based Intrusion Detection Systems (IDS)** can be placed within the VMM or a privileged VM to monitor guest VMs more effectively than traditional host-based or network-based systems.
- The concept of **Trusted Zones** helps establish secure, isolated environments for different tenants in a multi-tenant cloud, ensuring that VMs are protected while allowing controlled communication.