Abstract and Virtualization

Virtualization is a way of using resources (like servers, storage, or networks) efficiently by creating virtual versions of them. It works by assigning a logical name to a physical resource and connecting to it only when needed.

1. **Dynamic Mapping**: Virtualization dynamically assigns resources based on changing needs. For example, if more users join a system, it can quickly allocate extra servers.
2. **Efficient Management**: Changes to the resource mappings happen quickly, making operations smooth.

Examples in Real Life:

1. **Netflix Streaming**: When millions of people stream Netflix, virtual servers are used to deliver content. If a server gets overloaded, another server is instantly added to share the load.
2. **Google Drive**: When you upload files, the data is stored across multiple virtual storage systems for redundancy. You don't know the physical location—it's virtualized for efficiency.

How Virtualization Works:

- It maps virtual resources (like a virtual machine) to physical resources (like actual hardware).
- This mapping can change quickly to handle changing conditions.

Types of Virtualization:

1. **Access Virtualization**: Users can access services from anywhere, like logging into Gmail from any device.
2. **Application Virtualization**: Multiple users can access different instances of the same app, e.g., online gaming servers.
3. **CPU Virtualization**: A computer is divided into multiple virtual machines, each handling specific tasks. For example, a server hosting several websites assigns resources to each based on demand.
4. **Storage Virtualization**: Data is stored across multiple devices and often replicated for safety. For example, Google Drive replicates your data to ensure it's not lost.

Mobility Patterns in Virtualization

These patterns describe how resources are moved or configured in cloud systems:

- **P2V (Physical to Virtual)**: Turning a physical server into a virtual machine.
- **V2V (Virtual to Virtual)**: Moving one virtual machine to another.
- **V2P (Virtual to Physical)**: Turning a virtual machine into a physical one.
- **P2P (Physical to Physical)**: Transferring data between physical systems.
- **D2C (Datacenter to Cloud)**: Migrating resources from a physical datacenter to the cloud.

- **C2C (Cloud to Cloud)**: Moving data between cloud providers, like AWS to Google Cloud.
- **C2D (Cloud to Datacenter)**: Bringing cloud data back to a physical datacenter.
- **D2D (Datacenter to Datacenter)**: Moving resources between two physical data centers.

Key Characteristics Enabled by Virtualization (According to Gartner):

1. **Service-Based**: Clients access services through interfaces without worrying about the backend setup.
    - Example: Using Spotify to stream music without knowing where the data is stored.
2. **Scalable and Elastic**: Resources can grow or shrink as needed.
    - Example: During online sales like Black Friday, servers scale up to handle the traffic.
3. **Shared Services**: Resources are pooled for efficiency.
    - Example: Multiple apps using the same server infrastructure.
4. **Metered Usage**: You pay for only what you use.
    - Example: AWS charges based on how much storage or computing power you use.
5. **Internet Delivery**: All services are accessible over the internet using standard protocols.

Load Balancing and Virtualization

Load Balancing ensures that requests are evenly distributed across servers to:

1. Prevent overload.
2. Improve speed and response times.
3. Increase reliability.

How Load Balancing Works:

- A **load balancer** listens for incoming requests (e.g., from users) and assigns them to a server based on predefined rules.
- These rules can include:
    - **Round Robin**: Assign requests one by one to servers in a circular order.
    - **Fastest Response Time**: Send requests to the server that responds the quickest.
    - **Least Connections**: Assign requests to the server handling the fewest connections.

Network Resources That Can Be Load Balanced:

1. **Network Interfaces**: Distributing traffic for services like DNS, FTP, and HTTP.
2. **Connections**: Intelligent switches route connections to available servers.
3. **Processing**: Work is distributed across servers.
4. **Storage**: Data is spread across storage devices.
5. **Applications**: Requests are routed to application instances.

Advanced Load Balancing:

Some load balancers are **workload managers** that make smarter decisions by:

1. Polling resources to check their health.
2. Activating standby servers when needed (priority activation).
3. Using features like traffic compression, authentication, and content filtering.

Application Delivery Controller (ADC):

An **ADC** combines load balancing with application optimization. It:

- Routes traffic based on application-specific criteria.
- Handles tasks like security, caching, and compression.
- Ensures high availability for critical apps.

Benefits:

- **Fault Tolerance**: If one server fails, another takes over.
- **Efficiency**: Traffic is prioritized for better performance.
- **Reliability**: Data is stored redundantly to avoid loss.

Real-Life Example:

YouTube uses ADCs to:

- Cache video content for faster delivery.
- Distribute requests efficiently to avoid delays.

Google's Cloud and Load Balancing

Google's Cloud Infrastructure:

- Google uses **multi-layer load balancing** to handle billions of daily searches.
- Queries are routed to the **nearest datacenter** to reduce delay.
- Data Centers have clusters of servers, each performing tasks like caching, indexing, and web hosting.

How Google Manages Load Balancing:

1. A user's request is sent to the nearest datacenter using DNS-based load balancing.
2. Inside the datacenter:
    - A load balancer assigns requests to available clusters.
    - Another load balancer distributes work within the cluster.
3. Google's servers store and process data in memory for faster responses.

Benefits of Virtualization and Load Balancing

1. **Fault Tolerance**: If one server fails, another takes over.
2. **Redundancy**: Data is replicated to avoid loss.
3. **Performance Optimization**: Resources are always used efficiently.
4. **Scalability**: Systems grow or shrink based on demand.

Understanding Hypervisors

Hypervisors and virtualization technology help us create **virtual systems (virtual machines)** from physical systems. Let's break it down step by step in simple terms:

What Are Virtual Machines (VMs)?

- A **Virtual Machine** is like a computer inside a computer. It behaves like a physical machine but exists only in software.
- For users and applications, a VM looks and works just like a real computer, but it's actually a virtualized system running on top of physical hardware.

How Virtual Machines Work

1. **Resource Allocation**:
   - From a physical computer's resources (like CPU, RAM, and storage), we "slice" out portions to create virtual machines.
   - Each VM gets its own memory, CPU allocation, and virtual devices.
2. **Sandboxing**:
   - A VM is isolated (or "walled off") from the physical computer and other VMs. This makes it great for testing software, running old operating systems, or assigning workloads in cloud computing.
3. **Types of Virtual Machines**:
   - **System Virtual Machines**: These simulate an entire physical machine and can run a full operating system.
   - **Process Virtual Machines**: These are designed to run a single application or process.

Key Features of Virtual Machines

- **Independent Operation**: Each VM operates as its own system, with its own OS and applications.
- **Cloning and Replication**: VMs can be easily copied and duplicated for backup or testing.
- **Failover Support**: If one VM fails, another can take over to maintain service.

- **Downside**: Virtualization introduces some performance overhead because resources are accessed indirectly.

What Is a Hypervisor?

A **hypervisor**, or **Virtual Machine Monitor (VMM)**, is a program that allows virtual machines to use the resources of the physical hardware. Think of it as the "manager" that controls how VMs interact with the physical system.

Types of Hypervisors

1. **Type 1 Hypervisors (Bare-Metal Hypervisors)**:
   - Run directly on physical hardware without requiring a host operating system.
   - Offer **full virtualization** (a complete simulation of hardware).
   - Examples: VMware ESXi, Oracle VM, Microsoft Hyper-V, Xen.
2. **Key Points**:
   - There is no host operating system.
   - The operating system running in the VM is called the **guest operating system**.
   - Used in enterprise environments for high performance and reliability.

2. **Type 2 Hypervisors (Hosted Hypervisors)**:
   - Run on top of an existing operating system (host OS).
   - Provide a **paravirtualization** environment where the guest OS interacts with the host OS for I/O operations.
   - Examples: VMware Workstation, VirtualBox, Microsoft Virtual PC, Parallels Desktop.
3. **Key Points**:
   - Relies on the host OS to handle hardware-related tasks.
   - Commonly used in personal and development environments for easier setup.

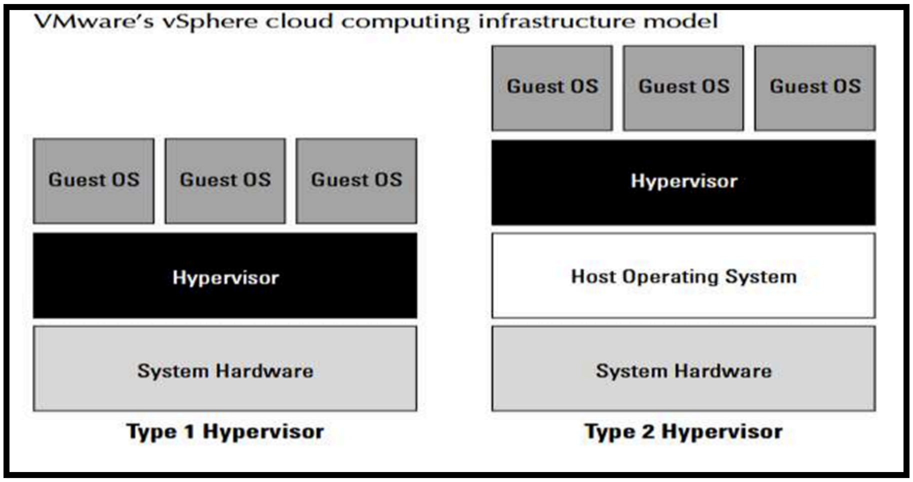
Paravirtualization

- In **paravirtualization**, the guest OS communicates with the host OS through a special interface called a **para-API**.
- This allows many I/O (input/output) tasks to be handled outside the VM, making operations more efficient.

  Examples of Hypervisors Using Paravirtualization:
   - Microsoft Hyper-V
   - Xen (used by Amazon Web Services for Amazon Machine Instances)

Examples for Real-Life Understanding

1. **Type 1 Hypervisor**:
   - Imagine a data center running dozens of virtual servers on physical hardware. VMware ESXi is used to directly manage these VMs for high efficiency and low overhead.
2. **Type 2 Hypervisor**:
   - A developer on a Mac uses Parallels Desktop to run a Windows environment for testing applications.



Types of Virtualization

1. **Emulation**:
   - The virtual machine (VM) simulates hardware, making it independent of the underlying physical system.
   - The guest operating system doesn't require modification.
   - **Example**: Running a game console emulator on your PC.
2. **Paravirtualization**:
   - The host OS provides a virtual machine interface (API) for the guest OS to communicate with the hardware.
   - The guest OS must be modified to use the host's interface.
   - Found in hypervisors like **Microsoft Hyper-V** and **Xen**.
3. **Full Virtualization**:
   - The hypervisor provides a complete hardware simulation.
   - Guest OS communicates directly with the hypervisor without modification.
   - Generally faster and more efficient.
   - **Example**: VMware ESXi for enterprise environments.

Application Virtual Machines

- **Features**:
  - Designed to run individual applications.
  - Slower but offer portability, rich programming languages, and platform independence.
- **Use in Cloud Computing**:
  - Not ideal for high-performance networks except for **parallel cluster computing**.
  - Examples: **Parallel Virtual Machine (PVM)** and **Message Passing Interface (MPI)** for high-performance systems.

Operating System Virtualization

- Some operating systems (e.g., Sun Solaris, IBM AIX) provide OS-level virtualization.
- Creates **virtual servers** called Virtual Private Servers (VPS), running in isolated virtual environments.
- VPSs must run the same OS version but offer low overhead for dense VM collections.
- **Examples**:
  - Sun Solaris Zones
  - IBM AIX System Workload Partitions (WPARs)

VMware vSphere

VMware vSphere is a popular framework for managing virtualized infrastructures (systems, storage, networks). It helps create cloud computing platforms by pooling resources.

Key Features of VMware vSphere:

1. **VMware vCompute**: Combines servers into a resource pool for easier management.
2. **VMware vStorage**: Manages storage resources as a pool.
3. **VMware vNetwork**: Creates and manages virtual network interfaces.
4. **High Availability (HA)**: Keeps applications running even during VM failures.
5. **Fault Tolerance**: Provides backup VMs that automatically take over if a primary VM fails.
6. **vCenter Server**: A management console for monitoring and provisioning VMs.

VMware Products and Tools

1. **ESXi (Type 1 Hypervisor)**:
   - Installed directly on physical hardware (bare-metal).
   - Uses a Linux kernel to boot and run the **vmkernel hypervisor**.
2. **Notable Tools**:
   - **Virtual Machine File System (VMFS)**: A high-performance cluster file system.

- **VMotion**: Migrates VMs between physical servers without downtime.
- **Storage VMotion**: Transfers VM storage across datastores while the VM remains active.
- **Distributed Resource Scheduler (DRS)**: Balances workloads dynamically.
- **vNetwork Distributed Switch (DVS)**: Maintains network runtime states for VMs across migrations.

Storage Virtualization

- **How It Works**:
  - Maps logical storage addresses to physical storage addresses.
  - Examples: Storage Area Networks (SANs) use Logical Unit Identifiers (LUNs) and Logical Block Addresses (LBAs) to manage virtual disks.
- **Types of Storage**:
  - **Direct Attached Storage (DAS)**: Connected directly to the server.
  - **Shared Storage**: Includes SANs, iSCSI arrays, and Network Attached Storage (NAS).

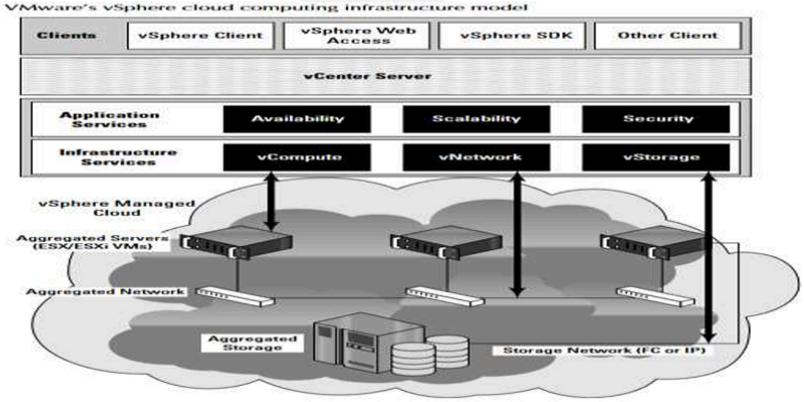
Network Virtualization

- Abstracts networking hardware and software into a virtualized environment.
- Creates **Virtual Network Interfaces (VNICs)** or **Virtual LANs (VLANs)**.
- Managed by hypervisors, operating systems, or external consoles.

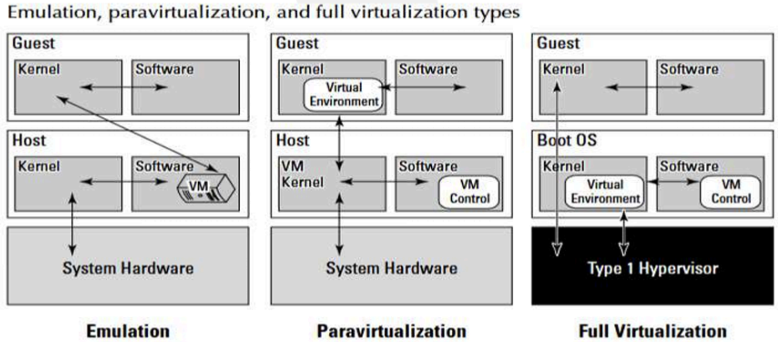
Real-Life Examples

1. **Emulation**:
   - Running an older video game designed for PlayStation on a PC emulator.
2. **VMware vSphere in Enterprises**:
   - A company uses **VMotion** to migrate virtual servers during peak hours, ensuring no downtime for customers.

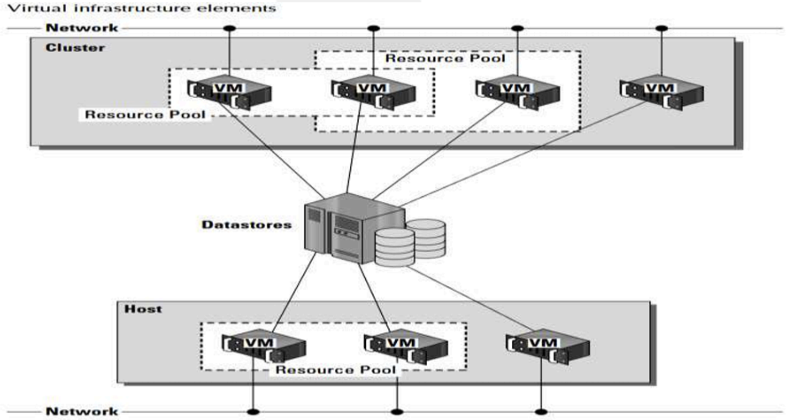
VMware's vSphere cloud computing infrastructure model



Emulation, paravirtualization, and full virtualization types



Virtual infrastructure elements



Understanding Machine Imaging

Machine Imaging is a key mechanism in cloud computing that enables **portability**, **application deployment**, and **system restoration** by creating a full copy (or clone) of an entire computer system in a single file.

What is a Machine Image?

- A **system image** captures the **state** of a system, including its operating system, applications, and configurations.
- It allows you to:
    1. **Clone or copy** a system for future use.
    2. **Restore** a system to its previous state if needed.
    3. **Snapshot systems** for backups or troubleshooting.

Example: Amazon Machine Images (AMIs)

- **Amazon Machine Images (AMIs)** are a common example of system images in cloud computing.
- Features:
    1. Pre-configured AMIs (e.g., with Windows or Linux) are available for public use.
    2. Users can create **custom AMIs** with specific configurations for their projects.
    3. AMIs can be:
        - Free (open-source).
        - Pay-per-use (e.g., for licensed software like Windows).
        - Shared privately with specific users.

Porting Applications

Porting applications refers to the process of moving software from one platform (cloud provider or environment) to another.

Challenges:

1. Many cloud vendors (e.g., AWS, Azure, Google Cloud) have **proprietary technologies**, which makes interoperability difficult.
2. Applications are often tightly coupled to specific hardware, operating systems, or frameworks.

The Simple Cloud API

- To address portability challenges, **Zend Technologies** introduced the **Simple API for Cloud Application Services**.

- **Goal**: Create **common interfaces** to simplify the movement of applications between platforms.
- Supported services include:
  - **File Storage Services**: Works with platforms like Amazon S3, Azure Blob Storage, and local storage.
  - **Document Storage Services**: Supports Amazon SimpleDB and Azure Table Storage.
  - **Simple Queue Services**: Works with Amazon SQS and Azure Queue Storage.

AppZero Virtual Application Appliance (VAA)

AppZero offers a solution for easily moving applications between platforms by isolating them in **virtual containers**.

How AppZero VAA Works:

1. **Encapsulation**:
   - The application and its dependencies (e.g., DLLs, registry entries, configurations) are packaged into a container.
   - This container acts as an **Application Image** for a specific operating system.
2. **Advantages**:
   - Applications run in an isolated environment without modifying the underlying OS.
   - No changes are made to the system registry or OS files.
3. **Stateless Cloud**:
   - AppZero envisions a **stateless cloud**, where application state information is stored on a network share. This allows applications to run seamlessly across different cloud systems.

Key Tools in AppZero:

- **AppZero Creator Wizard**: Creates the VAA.
- **AppZero Admin Tool**: Manages VAA operations.
- **AppZero Director**: Installs and runs the application.
- **AppZero Dissolve**: Removes the VAA layer and installs the application directly onto the OS if needed.

Real-Life Examples

1. **Machine Imaging**:
   - A company uses **AMIs** on AWS to deploy identical virtual machines for their web applications across multiple regions.
2. **AppZero VAA**:
   - A developer packages a Windows application into a VAA and deploys it on both Azure and AWS without compatibility issues.

Capacity Planning in Cloud Computing

Why Capacity Planning?

- At first glance, capacity planning might seem unnecessary for cloud computing, as the cloud is often perceived as **ubiquitous** (available everywhere) and **limitless**.
- **Reality**: Cloud resources are neither infinite nor always readily available. Planning is essential to ensure systems can handle demand efficiently while staying cost-effective.

Key Concepts of Capacity Planning

1. **Capacity Planning vs. System Optimization**:
   - **System Optimization**: Focuses on maximizing output from the resources you already have.
   - **Capacity Planning**: Identifies the system's maximum capacity and adds resources to meet future demand.
   - While optimization may happen during capacity planning, the focus remains on meeting demand even if inefficiencies exist.
2. **Iterative Process**: Capacity planning is a continuous process with the following steps:
   - **Step 1**: Assess the current system's characteristics.
   - **Step 2**: Measure workload across resources (CPU, RAM, disk, network, etc.).
   - **Step 3**: Test system overload to identify breaking points and required resources.
   - **Step 4**: Predict future demand using historical trends.
   - **Step 5**: Adjust resources (deploy or remove).
   - **Step 6**: Repeat the process regularly to accommodate changing demands.

Defining Baselines and Metrics

- The first step in capacity planning is to measure the system's current performance (baseline) and identify metrics for improvement.

LAMP Stack Example:

LAMP stands for:

- **Linux**: Operating system.
- **Apache**: Web server.

- **MySQL**: Database server.
- **PHP**: Scripting language.
- Example: A LAMP-based system on AWS might include:
  - A web server (Apache) that handles page views (hits per second).
  - A database server (MySQL) that processes transactions (queries per second).
- Metrics to monitor:
  - **Page Views**: Hits per second on the web server.
  - **Database Transactions**: Transactions per second or queries per second.

Traffic Analysis:

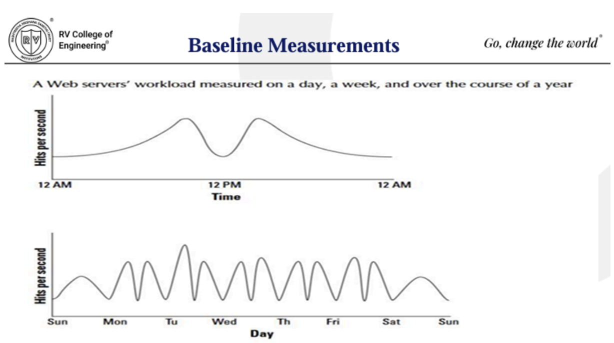
- Daily, weekly, and yearly logs are used to analyze traffic patterns and spikes.
  - Example: Daily spikes at **10 AM EST** (East Coast users) and **1 PM EST** (West Coast users).
- Capacity planners correlate these spikes with events and evaluate future traffic growth (e.g., yearly demand doubling).

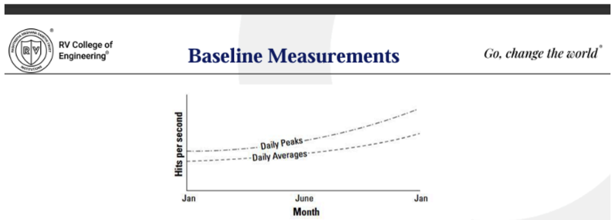
Baseline Metrics for Capacity Planning

Key metrics include:

1. **WT**: Total workload over a time period.
2. **WAVG**: Average workload across multiple periods.
3. **WMAX**: Peak workload observed.
4. **WTOT**: Total work done over a period.

- These metrics help planners evaluate demand patterns and correlate web server loads with database server activity.





System Metrics for Machine Instances

Machine instances in the cloud are typically measured by:

1. **CPU**: Processor utilization.
2. **Memory (RAM)**: Memory usage.
3. **Disk**: Input/output (I/O) operations.
4. **Network**: Bandwidth and packet transmission.

Tools for Monitoring Metrics:

- **Linux Tools**:
  - `sar` (from the `sysstat` package) to track CPU activity.
  - **RRDTool**: Captures time-dependent performance data (e.g., CPU, network usage).
- **Windows Tools**:
  - Task Manager or Performance Monitor (with logging and graphing options).
- **Cloud Monitoring Tools**:
  - **Amazon CloudWatch**: Monitors AWS resources and collects performance statistics.

Load Testing

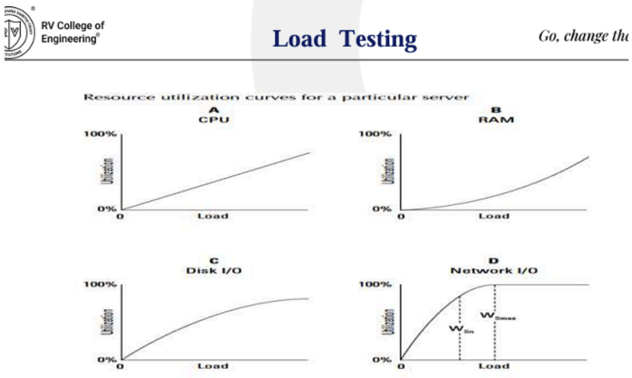
Load testing helps determine how a system performs under stress by simulating increasing workloads to identify bottlenecks.

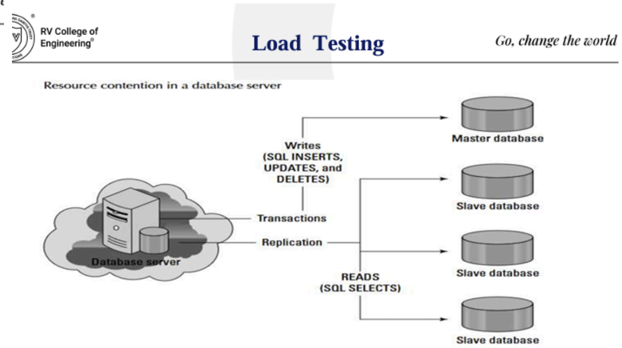
Questions Addressed by Load Testing:

1. What is the system's **maximum load**?
2. Which resource (CPU, RAM, disk, or network) is the **bottleneck** (resource ceiling)?
3. Can server configurations be adjusted to increase capacity?
4. How does this server's performance compare to others?

Common Load Testing Tools:

- **HTTPerf** and **Siege**: Simulate requests to web servers.
- **Autobench**: Runs HTTPerf from multiple clients for better testing.
- Advanced tools:
  - **JMeter**, **LoadRunner**, and **OpenSTA** for comprehensive load testing.





Resource Ceilings and Bottlenecks

A **resource ceiling** is the maximum capacity of a specific resource before performance starts degrading.

Example:

- **Web Server Bottleneck**: Network I/O reaches 100% utilization at 50% of the tested load.
  - Solutions:
    - Add more servers (scaling out).
    - Upgrade network connections (e.g., multi-homing or faster connections).

Database Resource Ceilings

1. **Disk I/O**: Improves with faster disk arrays or more spindles.
2. **Master/Slave Database Architecture**:
   - **Master**: Handles WRITE operations (e.g., INSERT, UPDATE, DELETE).
   - **Slaves**: Handle READ operations (e.g., SELECT queries).
   - **Replication Bottleneck**: WRITE traffic overwhelms the system's ability to replicate data to slaves.

Advanced Capacity Planning

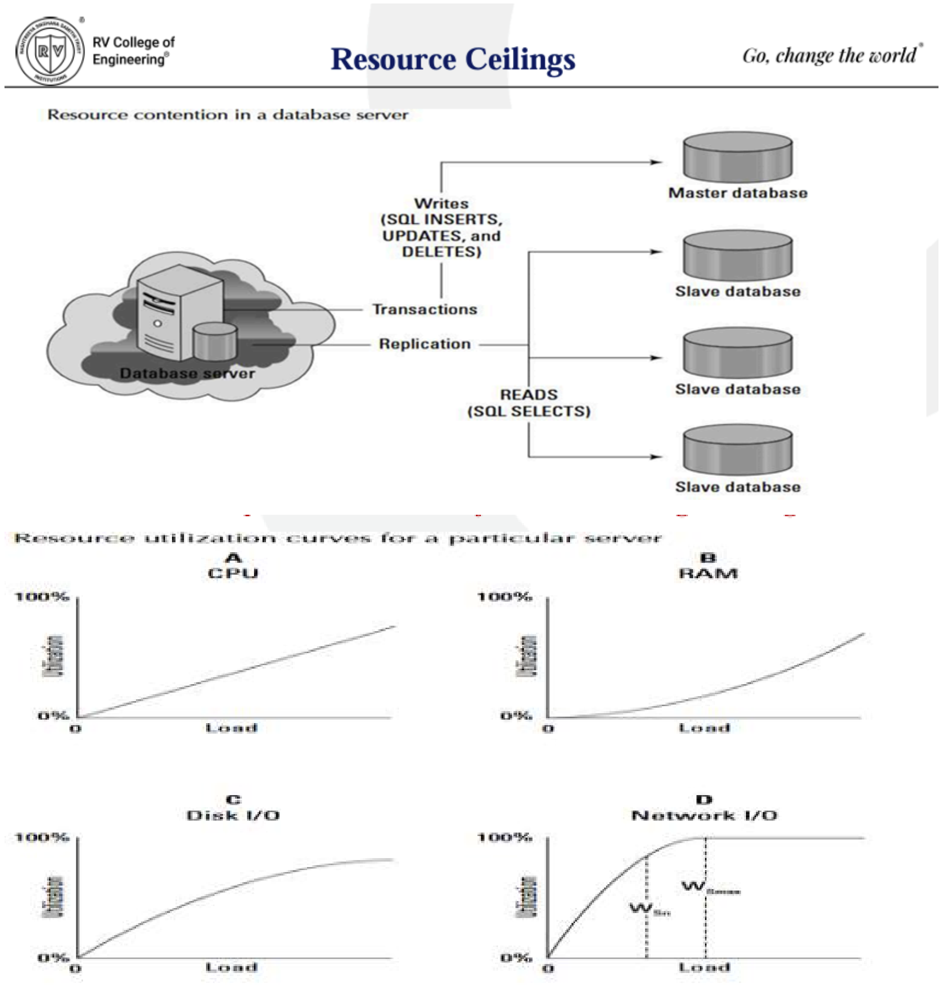
1. **Performance Consoles**:
   - Tools like **Microsoft Management Console (MMC)** and **Amazon CloudWatch** allow you to monitor system performance graphically.
   - They help evaluate **Key Performance Indicators (KPIs)** for better decision-making.
2. **Simulations vs. Real-World Data**:
   - Always prioritize real-world performance data over simulations, as real usage patterns provide more accurate insights.

Resource Ceilings

Resource ceilings refer to the maximum capacity of a system's individual resources (like CPU, RAM, disk I/O, or network) before performance begins to degrade.

Identifying Resource Ceilings

- Resource ceilings are identified by **load testing**, where the system is stressed at different levels to measure resource utilization rates.
- **Example (Figure 6.3)**: A graph might show:
    - **CPU (A)**, **RAM (B)**, and **Disk I/O (C)** utilization rise under load but do not reach their ceilings.
    - **Network I/O (D)**, however, hits 100% utilization at about 50% load, becoming the bottleneck.



Network I/O as a Bottleneck

- Network I/O is often the first bottleneck in web servers.
- **Solutions**:
  - **Scale out**: Add more low-powered servers to distribute the load.
  - **Upgrade network connections**: Use faster connections or multiple network interfaces (multi-homing).

Dealing with Overloaded Systems

- An overloaded system may fail, leading to degraded user experience or lost web hits.
- **Redline (WSn)**: The point at which the system must trigger alerts or initiate scripts to scale up capacity.
- **Example**: Set redline thresholds to ensure the system doesn't exceed maximum capacity:
  - For storage: Set the redline at 85% utilization, leaving a 15% safety margin.

Overall System Capacity (WT)

- **WT**: The total workload across all servers in the infrastructure.
  - **WT = Σ(WSnP + WSnV)**:
    - **WSnP**: Workload of physical servers.
    - **WSnV**: Workload of virtual servers.
- **Overhead considerations**:
  - Allow sufficient overhead in the system to handle unexpected spikes in demand.
  - This overhead is defined based on risk tolerance and reaction time to scale up.

\

Challenges in Load Testing

1. **Observer Effect**:
   - Load testing tools themselves introduce slight performance overhead.
   - Performance counters (e.g., queue length, file I/O rates) also impact system performance.
2. **Real-World Testing**:
   - Real-world performance data is more reliable than simulated results.
   - Example: Using **MySQL** master-slave setups for database replication.
     - **Master** handles WRITE operations.
     - **Slaves** handle READ operations.

Scaling in Capacity Planning

When resources reach their ceilings, scaling is the next step. Two approaches are available: **vertical scaling (scale-up)** and **horizontal scaling (scale-out)**.

Vertical Scaling (Scale-Up)

- **Definition**: Add more resources (e.g., CPU, memory, storage) to an existing system to make it more powerful.
- **Examples**:
  - Replace a dual-processor server with a quad-processor one.
  - Add more memory to support in-memory operations.
- **Best for**:
  - CPU- or memory-intensive applications like rendering or in-memory database queries.
- **Limitations**:
  - Scaling vertically indefinitely leads to single supercomputers, which may be expensive and difficult to manage.

Horizontal Scaling (Scale-Out)

- **Definition**: Add more servers or nodes to distribute the workload.
- **Examples**:
  - Add more dual-processor servers to a cluster.
  - Use server farms, as seen in web applications and grid computing.
- **Best for**:
  - Applications with I/O bottlenecks (e.g., web server connections).
- **Advantages**:
  - Better resource pooling and partitioning.
  - Supports distributed applications efficiently.

Trade-Offs Between Scale-Up and Scale-Out

1. **Cost**:
   - Scaling up (larger instances) may be more expensive than adding smaller instances in some cloud pricing models.
2. **Management**:
   - Scaling out increases system complexity (e.g., more nodes to manage and higher inter-system communication).
3. **Latency**:
   - Scale-out architectures introduce latency from communication between systems.

MySQL Database Resource Ceilings

1. **Master-Slave Setup**:
   - Master database handles all WRITE traffic.
   - Slave databases handle READ traffic.
   - **Problem**: High WRITE traffic can overwhelm the system's ability to replicate data to slaves.
   - **Solution**: Use more powerful disk arrays or faster interconnects for master-slave replication.
2. **Scaling Database Systems**:
   - For smaller applications, use master-slave architecture.
   - For larger applications, deploy federated databases to handle increased transactions.

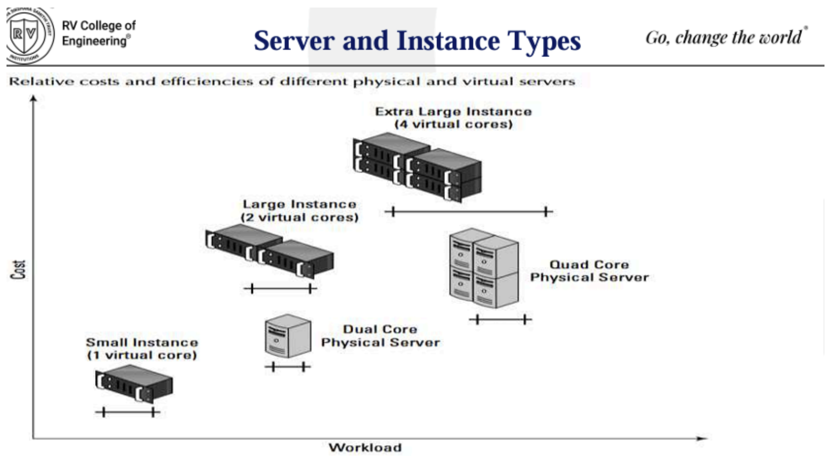
Performance Monitoring Tools

1. **System Monitoring**:
   - Tools like **RRDTool** (Linux) or Task Manager (Windows) monitor performance metrics like CPU, disk I/O, and network usage.
   - **Example**: Amazon CloudWatch provides a monitoring console for AWS resources.
2. **Network Monitoring**:
   - **Tools**: Wireshark, Kismet, TCPdump, and PathViewCloud.
   - PathViewCloud provides WAN performance metrics and evaluates network traffic between cloud and local systems.

Server and Instance Types in Cloud Computing

1. **Standardization**:
   - Standardizing hardware and software configurations makes capacity planning easier.
   - Identical servers with the same software perform predictably, simplifying troubleshooting and scaling.
2. **Amazon Machine Instances (AMIs)**:
   - Examples of instance types:
     - **Micro Instance**: Small memory and CPU bursts for lightweight tasks.
     - **Small Instance (m1.small)**: Moderate I/O for web applications.
     - **High-Memory Quadruple Extra Large (m2.4xlarge)**: Suitable for high-memory workloads.
     - **High-CPU Extra Large (c1.xlarge)**: Designed for CPU-intensive applications.
3. **Considerations for Cloud Instances**:

- **Variability**: Performance of cloud instances can vary due to underlying infrastructure changes.
- **Scaling Options**: Use features like Amazon Auto Scaling to adjust resources dynamically.



Network Capacity in Cloud Systems

1. **Key Factors**:
   - Network traffic to/from the server's interface.
   - WAN traffic within the cloud infrastructure.
   - Traffic between the cloud and local ISP.
2. **Challenges**:
   - Measuring WAN traffic (e.g., routing protocols, bandwidth) is difficult.
   - Network bottlenecks often occur in **last-mile connectivity** to homes or businesses.
3. **Solutions**:
   - Use monitoring tools like PathViewCloud to measure cloud network performance.
   - Advocate for better broadband infrastructure (e.g., Google Fiber initiatives.