DSCC 3 & 4

UNIT - 3

Here's a detailed breakdown with **10 points for each topic** in UNIT-III:

Distributed File Systems

1. **Definition**:
    - A **Distributed File System (DFS)** enables file storage and access across a network of interconnected nodes, providing users with a unified file system interface.
2. **Transparency**:

- **Location Transparency**: Users don't need to know file location.
- **Replication Transparency**: Multiple replicas of files appear as a single entity.

3. **Scalability**:
    - Designed to handle large datasets and an increasing number of users or nodes.
4. **Fault Tolerance**:
    - Utilizes replication and recovery mechanisms to ensure data availability even during failures.
5. **Concurrency**:
    - Multiple clients can access files simultaneously without interference.
6. **Performance**:
    - Employs caching, replication, and optimized access strategies to reduce latency.
7. **Consistency**:
    - Ensures all replicas of a file are updated uniformly after modifications.
8. **Access Control**:
    - Provides user authentication and file permission mechanisms to secure data.
9. **Interoperability**:
    - Supports multiple platforms and integrates with other file systems (e.g., NFS, SMB).
10. **Examples**:

- HDFS (Hadoop Distributed File System), Google File System (GFS), Amazon S3.

File Models

1. **Structured Model**:
    - Files are organized in a predefined structure, such as tables or databases.

2. **Unstructured Model**:
   - Files contain free-form data, such as text or multimedia, without predefined structure.
3. **Hierarchical Model**:
   - Files are arranged in a tree-like directory structure for organization.
4. **Flat Model**:
   - Files are organized in a single directory with no subfolders.
5. **Key Attributes**:
   - File metadata includes name, size, creation time, and permissions.
6. **Access Types**:
   - Read, write, append, and delete operations define how files are manipulated.
7. **Naming Conventions**:
   - Uses unique identifiers or paths to locate files.
8. **Storage Locations**:
   - Determines whether files are stored locally, on a server, or replicated across nodes.
9. **Access Granularity**:
   - Byte-level or record-level access to data within a file.
10. **Examples**:

- Text files, binary files, log files, and structured XML/JSON files.

File Accessing, Sharing, and Caching

File Accessing

1. **Sequential Access**:
   - Files are accessed in a linear sequence from start to end.
2. **Random Access**:
   - Files can be accessed non-linearly using offsets or indices.

3. **Access Protocols**:
    - Protocols like NFS, SMB ensure standardized file access in DFS.
4. **Remote Access**:
    - Uses RPC (Remote Procedure Call) or APIs for accessing files stored on remote nodes.
5. **Caching Mechanisms**:
    - Frequently accessed files are cached locally for quick access.

File Sharing

1. **Concurrent Access**:
    - Multiple users or processes can access the same file simultaneously.
2. **Synchronization**:
    - Locking mechanisms (e.g., read/write locks) to avoid conflicts.
3. **Version Control**:
    - Keeps track of changes to shared files to maintain consistency.
4. **Access Control**:
    - Implements permissions (read, write, execute) for shared files.
5. **Collaboration Tools**:

- DFS supports collaborative environments, e.g., Google Drive or Dropbox.

File Replication

1. **Definition**:
    - Copies of the same file are maintained on multiple nodes.
2. **Purpose**:
    - Improves fault tolerance and data availability.
3. **Replication Models**:
    - **Synchronous**: Updates all replicas simultaneously.
    - **Asynchronous**: Updates replicas at different times.
4. **Placement Policies**:

- Determines where replicas are stored to optimize performance and reliability.

5. **Consistency Models**:
   - **Strong Consistency**: All replicas show the same data at any time.
   - **Eventual Consistency**: Consistency achieved over time.
6. **Fault Tolerance**:
   - Replication ensures data recovery in case of node failures.
7. **Load Balancing**:
   - Distributes read and write requests across replicas to prevent overloading.
8. **Challenges**:
   - Ensuring consistency, avoiding stale data, and managing overhead.
9. **Performance**:
   - Balances storage cost with speed of access.
10. **Use Cases**:

- Data centers, HDFS, and RAID systems.

Atomic Transactions

1. **Definition**:
   - Ensures that file operations are completed entirely or not at all.
2. **ACID Properties**:
   - Atomicity, Consistency, Isolation, Durability.
3. **Role in DFS**:
   - Maintains system integrity during concurrent file operations.
4. **Transaction Logs**:
   - Maintains a log of operations to ensure rollback or recovery.
5. **Fault Tolerance**:
   - Recovers incomplete transactions after crashes.

6. **Concurrency Control**:
   - Prevents conflicts when multiple transactions modify the same data.
7. **Commit Protocols**:
   - Ensures all participants in a distributed transaction agree on changes.
8. **Two-Phase Commit**:
   - A protocol ensuring all nodes agree before committing a transaction.
9. **Rollback Mechanisms**:
   - Reverts changes in case of transaction failure.
10. **Applications**:

- Banking systems, file replication in DFS.

Case Study: HADOOP

1. **Introduction**:
   - Hadoop Distributed File System (HDFS) is a scalable, fault-tolerant DFS designed for big data.
2. **Architecture**:
   - **NameNode**: Manages metadata.
   - **DataNodes**: Stores file blocks.
3. **Fault Tolerance**:
   - Replicates data across multiple nodes for reliability.
4. **Scalability**:
   - Handles large datasets by distributing them over multiple nodes.
5. **Write-once-read-many**:
   - Files are immutable once written, optimizing for reads.
6. **Data Blocks**:
   - Files are divided into blocks (default: 128 MB) for distribution.
7. **Replication**:
   - Default replication factor is 3 for fault tolerance.

8. **YARN**:
    - Resource management and job scheduling framework in Hadoop.
9. **Use Cases**:
    - Data warehousing, machine learning, and log analysis.
10. **Performance**:
- Suitable for high-throughput applications but not low-latency tasks.

Resource and Process Management

1. **Resource Allocation**:
    - Ensures efficient usage of CPU, memory, and network bandwidth.
2. **Schedulers**:
    - Allocates resources using FIFO, priority, or fair-share scheduling.
3. **Load Monitoring**:
    - Tracks system resource usage to identify bottlenecks.
4. **Fault Recovery**:
    - Reallocates failed tasks to healthy nodes.
5. **Process Migration**:
    - Moves processes to underutilized nodes for better load distribution.
6. **Task Assignment**:
    - Matches tasks to nodes based on resource availability.
7. **Job Prioritization**:
    - Ensures high-priority jobs are executed first.
8. **Load Balancing**:
    - Evenly distributes workload across nodes.
9. **Dynamic Allocation**:
    - Adapts resource allocation in real-time based on workload changes.
10. **Tools**:
- Apache Mesos, Kubernetes for process and resource management.

Task Assignment and Load Management

Task Assignment

1. **Static Assignment**:
    - Fixed allocation of tasks to nodes at design time.
2. **Dynamic Assignment**:
    - Allocates tasks in real-time based on current system state.
3. **Metrics Considered**:
    - CPU load, memory usage, network bandwidth.
4. **Task Dependencies**:
    - Ensures dependent tasks are assigned to the same or nearby nodes.
5. **Load Prediction**:
    - Uses historical data to forecast and allocate resources.

Load Management

1. **Load Balancing**:
    - Redistributes workload to ensure no node is overburdened.
2. **Load Sharing**:
    - Distributes excess load from busy nodes to idle nodes.
3. **Algorithms**:
    - Round-robin, least connections, and weighted balancing.
4. **Overhead Minimization**:
    - Reduces communication and migration costs during load balancing.
5. **Real-time Adjustments**:

- Continuously monitors and reallocates tasks as needed.

Let me know if you'd like further elaboration on any subtopic!

Unit - 4

UNIT-IV: Cloud Computing

Here's a detailed breakdown with **10 points for each topic** in UNIT-IV:

Cloud Computing

1. **Definition**:
   - Delivery of computing resources (servers, storage, databases, networking, software) over the internet.
2. **On-Demand Services**:
   - Provides resources dynamically without requiring upfront provisioning.
3. **Resource Pooling**:
   - Combines resources for multiple users, enabling shared access.
4. **Elasticity**:
   - Resources scale up or down based on demand.
5. **Pay-as-You-Go**:
   - Users pay only for the resources they consume.
6. **Virtualization**:
   - Key enabler of cloud computing; creates virtual instances of hardware and software.
7. **Service Models**:
   - Includes IaaS, PaaS, and SaaS for varying levels of abstraction.
8. **Deployment Models**:
   - Public, private, hybrid, and community clouds.
9. **Security**:
   - Implements authentication, encryption, and compliance measures.
10. **Examples**:

- AWS, Microsoft Azure, Google Cloud Platform.

Roots of Cloud Computing

1. **Utility Computing**:

- Early model focused on providing computing as a metered utility.

2. **Grid Computing**:
    - Combines distributed resources to solve complex problems.
3. **Virtualization**:
    - Abstracts physical resources for efficient use in clouds.
4. **Web Services**:
    - Enabled application integration and service-oriented architecture.
5. **Data Centers**:
    - Centralized facilities hosting hardware and applications for cloud services.
6. **Advancements in Networking**:
    - High-speed internet made cloud computing feasible.
7. **Open Source Technologies**:
    - Projects like OpenStack and Kubernetes accelerated cloud adoption.
8. **Economy of Scale**:
    - Reduced costs by centralizing and optimizing resources.
9. **Internet Evolution**:
    - Transition from Web 1.0 to Web 2.0 supported cloud-hosted applications.
10. **First Movers**:

- Early pioneers like Salesforce (SaaS) and Amazon EC2 (IaaS).

Layers and Types of Clouds

1. **Layers**:
    - **Physical Layer**: Hardware infrastructure.
    - **Virtualization Layer**: Abstracts physical resources.
    - **Application Layer**: User-facing applications and interfaces.
2. **Types of Clouds**:

- **Public Cloud**: Open to the general public (e.g., AWS).
- **Private Cloud**: Exclusive to one organization.
- **Hybrid Cloud**: Combination of public and private.
- **Community Cloud**: Shared by organizations with similar needs.

3. **Infrastructure Layer**:
   - Includes servers, storage, and networking.
4. **Platform Layer**:
   - Provides tools and frameworks for application development.
5. **Application Layer**:
   - Hosts software and services for end-users.
6. **Service Models**:
   - IaaS, PaaS, SaaS correspond to infrastructure, platforms, and software layers.
7. **Deployment Differences**:
   - Public clouds are cost-effective; private clouds offer more control.
8. **Virtualization Technologies**:
   - Hypervisors like VMware, KVM, and Xen underpin cloud layers.
9. **Management Tools**:
   - Includes tools like Terraform and Kubernetes.
10. **Scalability and Flexibility**:

- Different cloud types support various business needs.

Desired Features of a Cloud

1. **Scalability**:
   - Dynamic scaling of resources based on demand.
2. **Reliability**:
   - Redundant systems ensure high availability.
3. **Security**:

- Data encryption, firewalls, and compliance measures.

4. **Cost Efficiency**:
   - Pay-as-you-go pricing reduces upfront costs.
5. **Accessibility**:
   - Accessible from any device with an internet connection.
6. **Automation**:
   - Provisioning and scaling handled automatically.
7. **Multi-tenancy**:
   - Serves multiple users securely on shared infrastructure.
8. **Interoperability**:
   - Supports integration with other systems and services.
9. **Performance**:
   - High-speed processing and low-latency networking.
10. **Self-Service**:

- Users can provision and manage resources independently.

Cloud Infrastructure Management

1. **Definition**:
   - Managing physical and virtual resources in a cloud environment.
2. **Monitoring**:
   - Tracks system performance and usage.
3. **Provisioning**:
   - Allocates resources for specific tasks or users.
4. **Configuration Management**:
   - Automates setup and maintenance of infrastructure.
5. **Resource Optimization**:
   - Ensures efficient usage of CPU, memory, and storage.
6. **Security Management**:

- Enforces access control, encryption, and threat monitoring.

7. **Backup and Recovery**:
    - Ensures data integrity and availability during failures.
8. **Compliance**:
    - Meets regulatory requirements for data handling.
9. **Automation Tools**:
    - Tools like Ansible, Puppet, and Chef simplify management.
10. **Cost Management**:

- Monitors resource usage to optimize expenses.

Infrastructure as a Service (IaaS)

1. **Definition**:
    - Provides virtualized computing resources over the internet.
2. **Features**:
    - Compute, storage, and networking on-demand.
3. **Use Cases**:
    - Hosting websites, running enterprise applications.
4. **Examples**:
    - AWS EC2, Google Compute Engine.
5. **Elasticity**:
    - Easily scale resources based on workload.
6. **Customization**:
    - Full control over operating systems and applications.
7. **Security**:
    - Responsibility shared between provider and user.
8. **Cost Model**:
    - Pay-per-use; reduces CapEx.
9. **Benefits**:

- Flexible, scalable, and fast deployment.

10. **Challenges**:

- Requires expertise to manage infrastructure effectively.

Platform as a Service (PaaS)

1. **Definition**:
    - Provides platforms for developers to build, test, and deploy applications.
2. **Features**:
    - Includes frameworks, databases, and development tools.
3. **Examples**:
    - Google App Engine, Microsoft Azure App Services.
4. **Ease of Use**:
    - Simplifies development by abstracting infrastructure management.
5. **Scalability**:
    - Automatically scales resources with application demand.
6. **Focus**:
    - Developers focus on code; providers handle infrastructure.
7. **Customization**:
    - Limited compared to IaaS but sufficient for most apps.
8. **Security**:
    - Applications inherit provider's robust security measures.
9. **Cost Model**:
    - Pay-as-you-use for services consumed.
10. **Challenges**:

- Vendor lock-in due to proprietary tools.

Software as a Service (SaaS)

1. **Definition**:

- Delivers software over the internet; users access via web browsers.

2. **Features**:
   - No installation or maintenance required by users.
3. **Examples**:
   - Google Workspace, Salesforce, Dropbox.
4. **Accessibility**:
   - Accessible from any device with a browser.
5. **Cost Model**:
   - Subscription-based pricing.
6. **Multi-tenancy**:
   - Single instance serves multiple users.
7. **Updates**:
   - Automatically updated by the provider.
8. **Security**:
   - Provider ensures compliance and threat protection.
9. **Use Cases**:
   - CRM, email, collaboration tools.
10. **Challenges**:

- Data security concerns and vendor dependency.

Challenges and Risks

1. **Security Concerns**:
   - Data breaches, compliance, and insider threats.
2. **Downtime**:
   - Service outages can disrupt business operations.
3. **Vendor Lock-In**:
   - Limited ability to switch providers without re-architecting.
4. **Cost Overruns**:

- Poor management of resources leads to unexpected expenses.

5. **Data Privacy**:
    - Risk of unauthorized access to sensitive data.
6. **Integration Issues**:
    - Difficulties in connecting with on-premise systems.
7. **Performance**:
    - Latency issues due to network dependency.
8. **Scalability Limits**:
    - Some providers may not meet peak demands effectively.
9. **Legal and Compliance**:
    - Challenges adhering to international data laws.
10. **Complexity**:

- Managing multi-cloud environments increases operational complexity.

Migrating into a Cloud

1. **Definition**:
    - Transitioning from traditional IT infrastructure to cloud-based services.
2. **Benefits**:
    - Reduces costs, improves scalability, and enables innovation.
3. **Planning**:
    - Requires careful assessment of workloads and dependencies.
4. **Broad Approaches**:
    - **Rehosting**: Lift-and-shift existing applications.
    - **Refactoring**: Optimize applications for cloud environments.
5. **Seven-Step Model**:
    - Assess, choose provider,

plan migration, secure data, execute migration, test, optimize.

1. **Security**:

- Encryption and access controls during migration.

2. **Tools**:
    - Use tools like AWS Migration Hub for streamlined transitions.
3. **Challenges**:
    - Downtime, data loss risks, and compatibility issues.
4. **Hybrid Approach**:
    - Retain critical workloads on-premise while migrating others.
5. **Testing**:

- Post-migration testing ensures everything functions as expected.