DISTRIBUTED FILE SYSTEMS (DFS)

INTRODUCTION TO DFS

- A Distributed File System (DFS) allows multiple users to access files across networked computers.
- It provides location transparency, scalability, and fault tolerance.
- DFS is widely used in cloud computing, big data, and enterprise environments.

CHARACTERISTICS OF DFS

- Transparency: Location, Access, Replication, and Concurrency transparency.
- Scalability: Handles increasing storage and users efficiently.
- Fault Tolerance: Replication ensures data availability.
- Performance Optimization: Caching, load balancing, and prefetching.
- Security: Authentication, encryption, and access control.

DFS ARCHITECTURE

- Client-Server Model: Centralized server manages file access.
- Peer-to-Peer Model: Peers share files without a central server.
- Hybrid Model: Combines client-server and P2P features.

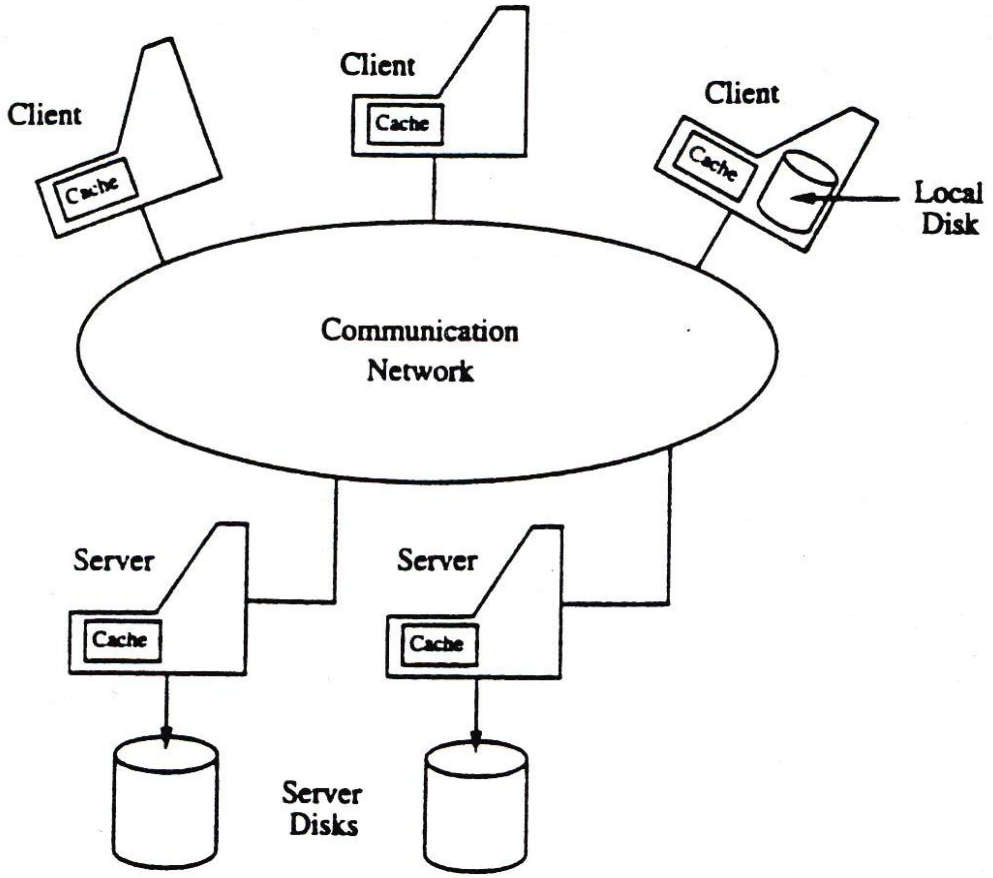
COMPONENTS OF DFS

- File Servers: Store and manage file data.
- Clients: Request and perform file operations.
- Metadata Server: Tracks file locations and access permissions.
- Storage Nodes: Store and replicate data.
- Cache Manager: Optimizes performance.

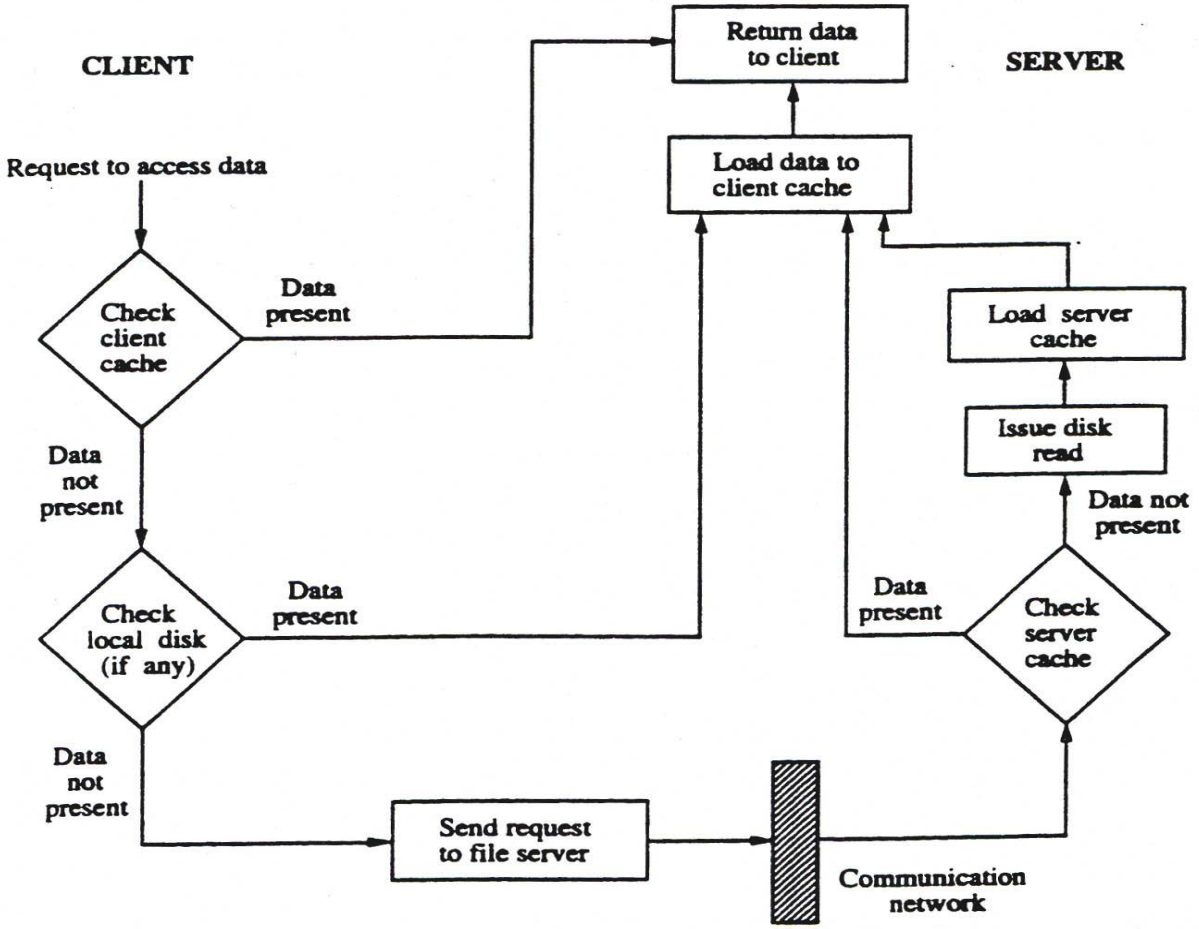
OPERATIONS IN DFS

- File Access: Read, write, and modify files.
- File Sharing: Multi-user collaboration.
- File Replication: Copies stored across multiple nodes.
- Locking & Synchronization: Ensures consistency.
- Security: Authentication and authorization.

1. CLIENT SERVER ARCHITECTURE



TYPICAL DATA ACCESS IN A CLIENT/FILE SERVER ARCHITECTURE



2. PEER-TO-PEER (P2P) MODEL

- The Peer-to-Peer (P2P) Model in Distributed File Systems (DFS) is a decentralized architecture where all participating nodes (or peers) have equal roles and capabilities.
- Unlike the traditional Client-Server model, where a central server is responsible for managing files and resources, the P2P model does not rely on a central server, making it more resilient and scalable.

P2P MODEL OVERVIEW

- **Decentralized Architecture**: In a P2P DFS, all nodes (peers) are equally responsible for storing and retrieving data.
- **File Sharing**: Files are distributed across multiple peers, and each peer can act as both a client and a server.
- **No Centralized Server**: There's no central repository; instead, each peer stores a portion of the data.

CHARACTERISTICS OF P2P IN DFS

- **Self-organizing**: Peers join and leave the network dynamically. There's no central entity to control these actions.
- **Resource Sharing**: Every peer can share its local resources (such as disk space, processing power, or bandwidth) with others in the network.
- **Fault Tolerance**: Because data is replicated across multiple peers, the system can handle failures without losing data.

P2P VS CLIENT-SERVER MODEL IN DFS

Aspect	P2P DFS	Client-Server DFS
Architecture	Decentralized	Centralized
Data Storage	Distributed among peers	Central server holds all data
Fault Tolerance	High (due to data replication)	Lower (depends on server availability)
Scalability	Highly scalable (as more peers join)	Limited by the capacity of the server
Data Access	Direct access between peers	Clients access data via the server

BitTorrent is a widely used example of P2P file sharing, where files are split into small pieces, and each peer shares pieces with others to help distribute the file efficiently.

3. HYBRID MODEL IN DISTRIBUTED FILE SYSTEMS (DFS)

- A Hybrid Model in Distributed File Systems (DFS) combines the strengths of both Client-Server and Peer-to-Peer (P2P) models to create a more flexible, scalable, and fault-tolerant distributed file system.
- This model attempts to balance the advantages of central control with the decentralized nature of peer participation.

HYBRID MODEL

- In a Hybrid DFS:
  - Some parts of the system may operate using a Client-Server model, where a central server manages metadata, file directories, and control functions.
  - Other parts may use a P2P approach, where file data is distributed across multiple peers for redundancy and scalability.
  - This model typically aims to combine the centralized control and efficiency of a client-server architecture with the fault tolerance and scalability of a P2P system.

CHARACTERISTICS OF THE HYBRID DFS MODEL

- **Centralized Metadata with Distributed Data:**
  - The metadata (file names, access control, and location information) is managed centrally, often by a master server or metadata server.
  - File data is stored and retrieved in a distributed manner across peer nodes.
- **Dynamic Data Replication:**
  - To ensure fault tolerance and high availability, data may be replicated on multiple peers.
  - The replication strategy can be adjusted dynamically based on load balancing or data access patterns.
- **Improved Fault Tolerance and Performance:**
  - By combining centralized control and distributed storage, a hybrid system benefits from both high availability and fault tolerance.

EXAMPLES OF DFS

- NFS (Network File System): UNIX/Linux file sharing.
- HDFS (Hadoop Distributed File System): Big Data applications.
- Google File System (GFS): Scalable distributed storage.
- SMB (Server Message Block): Windows file sharing.
- CephFS: Cloud-based storage system.

ADVANTAGES OF DFS

- High Availability: Redundant copies prevent data loss.
- Fault Tolerance: System recovers from failures.
- Scalability: Supports growing data storage needs.
- Load Balancing: Distributes file access efficiently.
- Data Sharing: Enables real-time collaboration.

DISADVANTAGES OF DFS

- Network Dependency: Affected by network speed.
- Complexity: Managing replication and security is challenging.
- Security Risks: Vulnerable to cyber threats.
- Storage Overhead: Replication increases storage costs.

ISSUES IN DISTRIBUTED FILE SYSTEMS (DFS)

- When implementing Distributed File Systems (DFS), several challenges must be addressed to ensure that the system is efficient, scalable, reliable, and easy to use.
- These challenges can arise from the nature of distributed storage and management of files across multiple machines or nodes. Let's break down these issues in detail:

ISSUES IN DISTRIBUTED FILE SYSTEMS…

The following are the key challenges in DFS that need to be addressed for an efficient system:

- Name Services: Ensuring proper naming, locating mobile entities, and removing unreferenced entities.
- Replication: Ensuring fault tolerance and availability through efficient replication and consistent updates.
- Update Protocols: Managing consistency and concurrency in updates, especially in large-scale systems.
- Scalability: Efficiently managing metadata, network traffic, and load balancing as the system grows.

By solving these issues, DFS can become more robust, scalable, and reliable, making it suitable for real-world applications.

1. NAME SERVICES IN DFS

- Name Services in DFS involve the management of file names and their corresponding metadata across distributed systems.
- The key challenges include:
  - **a) Naming Entities Problem**: In a distributed system, files and directories need unique identifiers or names. Ensuring that names are consistent and unique across distributed nodes can be challenging.
  - Solution: Systems must have a naming scheme that avoids conflicts and ensures uniqueness, such as hierarchical naming conventions (like in file systems) or global identifiers.

1. NAME SERVICES IN DFS…

b) Locating Mobile Entities Problem:

- In DFS, files may be moved between nodes for load balancing, fault tolerance, or performance reasons.
- When files or data are mobile, their location information must be kept up-to-date across all nodes.
- Solution: Techniques like distributed hash tables (DHTs), directory services, or centralized metadata servers are used to track the current locations of mobile entities efficiently.

1. NAME SERVICES IN DFS…

c) Removing Unreferenced Entities

- **Problem**: Over time, files or data may no longer be in use, but they are still stored in the system. These unreferenced files can waste space and resources.
- **Solution**: **Garbage collection** methods can be used to periodically identify and remove files that are no longer referenced, freeing up storage space.

2. REPLICATION IN DFS

Replication ensures that copies of files or data blocks are stored across multiple nodes to improve **fault tolerance** and **availability**.

- **Challenges in Replication:**
  - **Maintaining Consistency**: When a file is updated, all replicas must be synchronized to ensure consistency. If not handled properly, replication can lead to data inconsistency.
  - **Efficient Replication**: Deciding how many copies of a file should be stored, where the replicas should be located, and how to handle failures is a challenge in DFS.
  - **Replication Overhead**: Replication can introduce extra storage and network overheads, especially in large-scale systems.
  - **Solution:**
    - Systems often use strategies like **quorum-based** replication or **lazy replication** to minimize overhead while ensuring that replicas are kept consistent over time.

3.UPDATE PROTOCOLS IN DFS

In distributed systems, managing updates across multiple copies or replicas of files is a critical issue.

- Challenges in Update Protocols:
  - Consistency vs. Availability: Ensuring strong consistency (where all replicas are always updated immediately after a write) versus prioritizing availability (ensuring that the system is always operational, even when not all replicas are synchronized).
  - Latency: Synchronous updates (where updates are propagated immediately) can result in high latency, whereas asynchronous updates can lead to temporary inconsistency.
  - Concurrency Control: Managing updates when multiple users or systems attempt to update the same file at the same time.
  - **Solution:**
    - **Quorum-based Update Protocols**: Systems like **Cassandra** or **Riak** use a **quorum** approach where an update is considered successful only when it is acknowledged by a certain number of replicas.
    - **Versioning**: By maintaining multiple versions of files, systems allow for conflict resolution and rollback if an update fails.

4. SCALABILITY IN DFS

- Scalability refers to the system's ability to handle increasing amounts of data, users, and nodes without performance degradation.
  - **Challenges in Scalability:**
    - **Metadata Management**: As the number of files and nodes increases, managing the metadata (file names, locations, access control) can become a bottleneck.
    - **Network Traffic**: In large systems, network traffic to access and retrieve files increases. This can lead to congestion and delays.
    - **Load Balancing**: Ensuring that files and requests are evenly distributed across nodes to avoid overloading some nodes while others remain underutilized.
  - **Solution:**
    - **Distributed Hash Tables (DHT)**: DHTs provide an efficient way to scale metadata management by distributing the metadata across multiple servers.
    - **Caching**: Frequently accessed files are cached to reduce network load and improve response times.
    - **Sharding**: Files can be divided into smaller **chunks** or **shards**, and each chunk is stored across different nodes, helping the system scale more easily.