VIT Vidyalankar Institute of Technology Accredited A+ by NAAC	**DEPARTMENT OF COMPUTER ENGINEERING**

Assignment No. 11

Semester	B.E. Semester VIII – Computer Engineering
Subject	Distributed Computing Lab
Subject Professor In-charge	Dr. Umesh Kulkarni
Assisting Professor	Prof. Prakash Parmar
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Balancing Trade-offs in Distributed File Systems (DFS)

Distributed File Systems (DFS) aim to provide efficient, reliable, and scalable access to files over a network, but achieving this requires balancing **efficient file access, caching, replication, and fault tolerance**. Modern DFSs, such as **NFS (Network File System)** and **Google File System (GFS)**, implement various strategies to optimize performance while ensuring data consistency and availability.

---

1. Trade-offs in Distributed File Systems (DFS)

DFSs must balance the following key factors:

A. Efficient File Access

DFSs aim to minimize latency and maximize throughput when retrieving or writing files.

- **Challenges:**
    - Network delays can slow down file access.
    - Centralized metadata management may become a bottleneck.
- **Solutions:**

- **Chunk-based storage (GFS, HDFS):** Large files are split into chunks, allowing parallel access.
- **Distributed caching (NFS, AFS):** Frequently accessed files are cached at the client side.
- **Optimized read/write mechanisms:** Pre-fetching and asynchronous writes improve performance.

Example: Google File System (GFS) divides files into 64MB chunks stored on multiple chunk servers, reducing metadata overhead and improving parallel access.

B. Caching Mechanisms

Caching helps improve performance by reducing redundant data fetches from remote storage.

- **Challenges:**
  - **Cache consistency:** How to ensure updates from different clients remain consistent?
  - **Stale data:** Cached copies may become outdated if the original file changes.
- **Solutions:**
  - **Write-through caching (NFSv3):** Writes are immediately sent to the server, ensuring consistency.
  - **Client-side caching (AFS, NFSv4):** Local copies reduce access latency.
  - **Lease-based caching (NFSv4, GFS):** The server grants leases to clients, allowing temporary exclusive access.

Example: NFSv4 uses delegation, where the server temporarily gives clients the authority to cache files, reducing network load.

C. Replication for Fault Tolerance & Availability

Replication ensures data remains available even if some nodes fail.

- **Challenges:**
  - **Replication overhead:** Maintaining multiple copies requires extra storage and synchronization.
  - **Consistency issues:** Updates must be propagated to all replicas correctly.
- **Solutions:**
  - **Primary-backup replication (GFS, HDFS):** A primary copy manages writes, and updates are propagated to secondary replicas.
  - **Eventual consistency (Amazon S3, Cassandra):** Writes are asynchronously replicated to ensure availability.
  - **Quorum-based replication (Ceph, Google Spanner):** Ensures strong consistency by requiring a majority of replicas to acknowledge changes.

Example: Google File System (GFS) replicates chunks **three times** across different servers for fault tolerance.

D. Fault Tolerance & Recovery

DFSs must ensure availability despite node failures.

- **Challenges:**
  - **Detecting and recovering from failures efficiently.**
  - **Avoiding single points of failure.**
- **Solutions:**
  - **Metadata replication (GFS, Ceph):** Ensures the metadata server is not a single point of failure.
  - **Heartbeats & automatic failover (HDFS, GFS):** Regular health checks allow quick failover to backup nodes.
  - **Erasure coding (Ceph, HDFS):** Reduces storage overhead while providing fault tolerance.

Example: Google's GFS master node regularly checkpoints metadata and assigns a new master in case of failure.

---

2. How Modern DFSs Implement These Trade-offs

Feature	**NFS**	**Google File System (GFS)**
Efficient File Access	Client-server architecture, optimized for local networks	Chunk-based file storage with parallel access
Caching	Client-side caching with consistency mechanisms	Lease-based caching for efficient reads
Replication	External replication using RAID or backups	Three-way chunk replication for fault tolerance
Fault Tolerance	Stateless server (NFSv3), Stateful in (NFSv4)	Master node failure detection and automatic recovery
Scalability	Best for enterprise LANs	Optimized for massive-scale web applications

---

3. Conclusion: Achieving a Balanced DFS Design

A well-designed **Distributed File System** must optimize **efficiency, caching, replication, and fault tolerance** while adapting to application needs.

- **NFS is optimized for network file sharing**, ensuring consistency and security.
- **GFS prioritizes scalability and fault tolerance** with chunk-based storage and replication.
- **Modern DFSs (HDFS, Ceph, Amazon S3) continue evolving** by integrating cloud-based storage, machine learning-based optimizations, and erasure coding.