Assignment No. 10

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: Network File System (NFS): Ensuring Seamless Access to Remote Files

---

1. How NFS Provides Seamless Access to Remote Files

The **Network File System (NFS)** is a **distributed file system protocol** that allows clients to access files over a network as if they were stored locally. It achieves this by:

1. **Mounting Remote File Systems Locally**
   - Clients can mount remote directories, making them part of their local file hierarchy.
   - This allows applications and users to interact with files using standard system calls (open(), read(), write(), etc.), without knowing the files are remote.
2. **Stateless or Stateful Server Design (Depending on NFS Version)**
   - **NFSv3** is mostly stateless, meaning the server does not track client state, making it resilient to failures.
   - **NFSv4** introduces stateful operations, improving locking, security, and cache consistency.
3. **Remote Procedure Calls (RPCs)**
   - NFS clients communicate with the server using **RPCs** (Remote Procedure Calls), enabling transparent access to remote files.

- Examples of NFS operations include LOOKUP, READ, WRITE, and GETATTR.

4. **File Handle System**
    - Each file and directory is referenced by a unique **file handle** rather than a traditional path.
    - The file handle allows efficient file retrieval without relying on full pathnames.
5. **Caching for Performance Optimization**
    - Clients cache frequently accessed files to reduce network latency.
    - However, cache consistency mechanisms (e.g., attribute checks) ensure that updates from other clients are reflected.
6. **Locking Mechanisms for Concurrency**
    - **NFSv3** relies on an external **Network Lock Manager (NLM)** for file locking.
    - **NFSv4** integrates locking within the protocol, improving performance and reliability.

---

2. Major Differences Between NFS Versions (NFSv3 vs. NFSv4)

Feature	**NFSv3**	**NFSv4**
Statefulness	Stateless (except for locking via NLM)	Stateful (tracks client sessions)
Security	Uses UNIX-based permissions (weak security)	Uses **Kerberos**, ACLs, and stronger authentication
File Locking	External NLM daemon	Built-in locking support
Performance	Multiple requests per operation (higher overhead)	**Compound operations** reduce network overhead
Caching	Simple caching, risk of stale reads	Improved cache consistency
Protocol Complexity	Simpler implementation	More complex but robust

Feature	NFSv3	NFSv4
Firewall Friendliness	Uses multiple ports (hard to secure)	Uses a **single TCP port (2049)** for better firewall traversal
Cross-Platform Support	Linux, UNIX	Improved support for Windows & enterprise environments
Namespace Handling	No concept of a **global namespace**	Supports **pseudo-filesystems** for better organization

Why Use NFSv4 Over NFSv3?

- **Security**: Stronger authentication with **Kerberos 5**.
- **Performance**: Compound operations reduce **latency**.
- **Locking**: Built-in support prevents issues like stale locks.
- **Single-Port Communication**: Works better in firewall-restricted environments.

---

3. Designing an NFS Setup for Scalable Workloads & High Performance

To handle high workloads while maintaining performance, an **optimized NFS setup** should include:

A. Architecture Considerations

1. **Use Multiple NFS Servers for Load Distribution**
    - Deploy **multiple NFS servers** behind a **load balancer**.
    - Use **round-robin DNS** or **NFS high-availability clustering** to distribute requests.
2. **Enable NFSv4 for Better Performance & Security**
    - **NFSv4** improves throughput by **reducing network overhead** (compound operations).
    - **Kerberos authentication** ensures security without sacrificing speed.

3. **Use Parallel File Systems (e.g., pNFS)**
   - **pNFS (Parallel NFS)** in **NFSv4.1+** allows clients to access storage directly, bypassing the NFS server bottleneck.
4. **Optimize the Network Stack**
   - Use **Jumbo Frames (9000 bytes MTU)** to reduce packet overhead.
   - Deploy **10GbE or faster network interfaces** for high throughput.
   - Enable **TCP over UDP** for improved reliability in large-scale environments.

B. Storage Optimization

5. **Leverage SSDs or NVMe Storage for Low Latency**
   - If the NFS workload is **IOPS-sensitive**, use **SSD-backed storage** or NVMe drives.
6. **Implement RAID for Fault Tolerance**
   - Use **RAID-10** for a balance between **performance and redundancy**.
7. **Use a Distributed File System (Ceph, GlusterFS)**
   - For very large workloads, consider **GlusterFS** or **Ceph** with NFS gateways for horizontal scaling.

C. Caching & Performance Tuning

8. **Enable Client-Side Caching**
   - Increase actimeo (attribute cache timeout) to reduce metadata lookups.
   - Use noatime mount option to **reduce unnecessary disk writes**.
9. **Tune NFS Server Parameters**
   - Increase rsize and wsize values (65536 for high-speed networks).
   - Adjust threads in **rpc.nfsd** to handle more concurrent requests.
10. **Use Asynchronous Writes for Throughput**

- Enable async mode to **improve write speeds** (with some risk of data loss on failure).

D. High Availability & Reliability

11. **Enable NFS Failover with HA Mechanisms**

- Use **Keepalived** or **Pacemaker & Corosync** for automatic failover.
- Implement **NFS over DRBD (Distributed Replicated Block Device)** for data mirroring.

12. **Deploy NFS in a Kubernetes or Cloud Environment**

- Use **EFS (AWS)** or **Filestore (Google Cloud)** for managed NFS solutions.
- Containerized workloads can use **Persistent Volumes (PV) with NFS StorageClasses**.

---

4. Example: Optimized NFS Configuration

A high-performance NFS setup for a **large-scale web application** could include:

- **NFSv4.1+ with pNFS** for parallel storage access.
- **SSDs or NVMe-backed storage** with RAID-10.
- **Multiple NFS servers** behind a **load balancer**.
- **Kerberos authentication & firewall-optimized TCP communication**.
- **Jumbo frames & 10GbE networking**.
- **Client-side caching & read-ahead optimizations**.
- **Kubernetes Persistent Volumes (PV) with NFS StorageClass**.

Conclusion

- **NFS ensures seamless remote file access** by mounting remote directories as part of the local filesystem.
- **NFSv4 offers better performance, security, and scalability** over NFSv3.
- **A scalable NFS setup requires optimizations** in networking, caching, storage, and failover mechanisms.
- **For extreme scalability**, parallel NFS (pNFS) or cloud-based solutions like AWS EFS can be used.