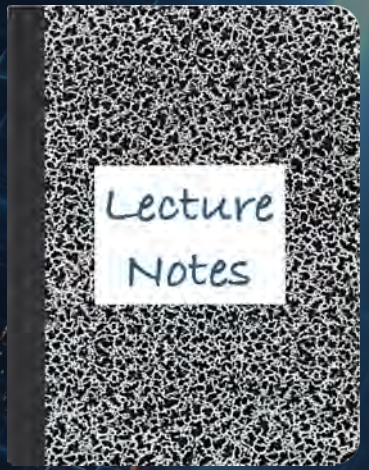
CS 417 – DISTRIBUTED SYSTEMS

Week 6: Part 2
Network-Attached Storage



Paul Krzyzanowski

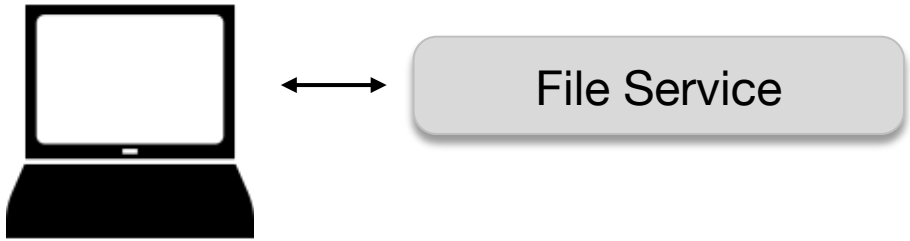
Accessing files

File sharing with socket-based services

- Make a local copy via: FTP/sftp, scp, HTTP, removable media
- Log into a remote system: ssh, virtual desktop, …

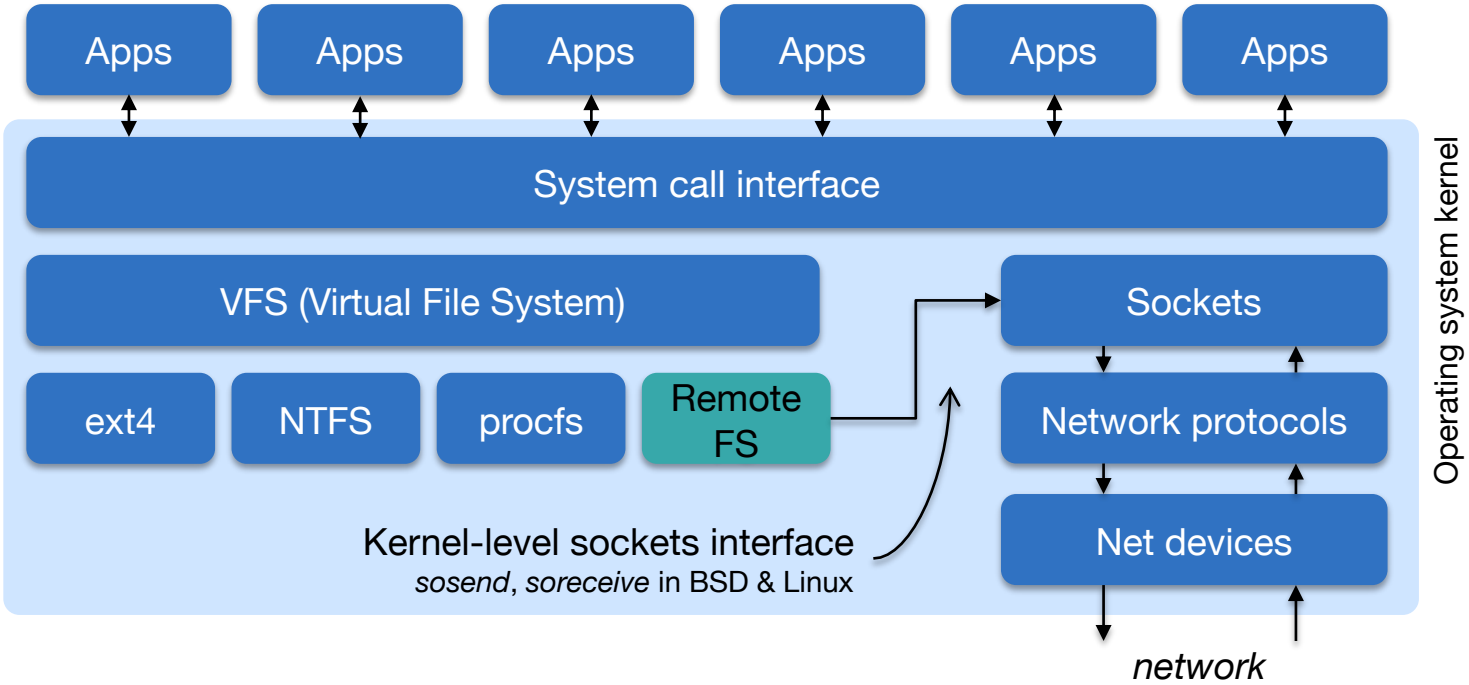
We want transparency: access remote resources just as local ones

⇒ **NAS: Network Attached Storage**



Access Transparency

Implement the client module as a file system type under VFS layer
(Installable File System driver, IFS, in Windows)



System Design Issues

- Caching (Consistency)
  - What happens if more than one user accesses the same file?
  - What if files are replicated across servers?
- Security
  - The local OS is no longer in charge
- State
  - Should the server keep track of clients between requests?
  - How is recovery handled?

File Service Access Models

Download/Upload model

- *Read file:* copy file from server to client
- *Write file:* copy file from client to server

Advantage

- Simple
- Local access speeds

Problems

- ***Wasteful***: what if client needs small piece?
- ***Problematic***: what if client doesn't have enough space?
- ***Consistency***: what if others need to modify the same file?

Remote access model

File service provides functional interface:

- *create, delete, read bytes, write bytes, etc…*

Advantages

- Client gets only what's needed
- Server can manage coherent view of file system

Problem

- Possible server and network **congestion**
  - Servers are accessed for duration of file access
  - Same data may be requested repeatedly

Semantics of File Sharing

Sequential Semantics

Read returns result of last write

Easily achieved *if*

- We use a remote access model
- Server data is not replicated
- Clients do not cache data

BUT

- Performance problems if no cache
  - Clients get obsolete data
- We can ***write-through***
  - Must notify all clients holding copies
  - Requires extra state, generates extra traffic

Session Semantics

Relax the rules

- Changes to an open file are visible only to the process (or machine) that modified it – made visible to others after the file is closed
- The last process to close the file overwrites any earlier changes
- To be safe – need to lock a file that is being modified to keep other clients from modifying it

File Access Semantics

Access semantics define consistency guarantees

- **Unix (POSIX) semantics** (sequential semantics) [original SMB]
  - A write is immediately visible to other processes reading the file
  - *Not practical for networked file systems because it forces contacting the server for every access – but you can get close with careful control of caching* [current SMB, NFS]
- **Close-to-open consistency** [original NFS]
  - Access data from the server when a file is opened but allow access to cached data during use. Cached data may be stale and differ across clients
- **Session semantics** [AFS, Coda]
  - A client's modifications are not visible until the client closes the file and the updated version is uploaded. The last client to close overwrites other changes.

Caching: The Key to Performance

- Caching = store data close to where it's needed
- Hide latency to improve performance for repeated accesses

File data can reside in several places

- Server's disk ← *original version of the file*
- Server's buffer cache ← *memory-resident copies of recent blocks of the file*
- Client's buffer cache
- Client's disk

Risk of cache consistency problems if multiple systems access the same file

Approaches to caching

Write-through

- What if another client reads its own (out-of-date) cached copy?
- All accesses will require checking with server
- Or … server maintains state and sends invalidations

Write-behind (delayed writes)

- Data can be buffered locally
  (watch out for consistency – others won't see updates!)
- Remote files updated periodically
- One bulk write is more efficient than lots of little writes
- Problem: semantics become ambiguous

Write-on-close

- Efficient, but doesn't support concurrent file modifications by multiple clients

Read-ahead (prefetch)

- Request chunks of data before it is needed
- Minimize wait times if that data is later needed

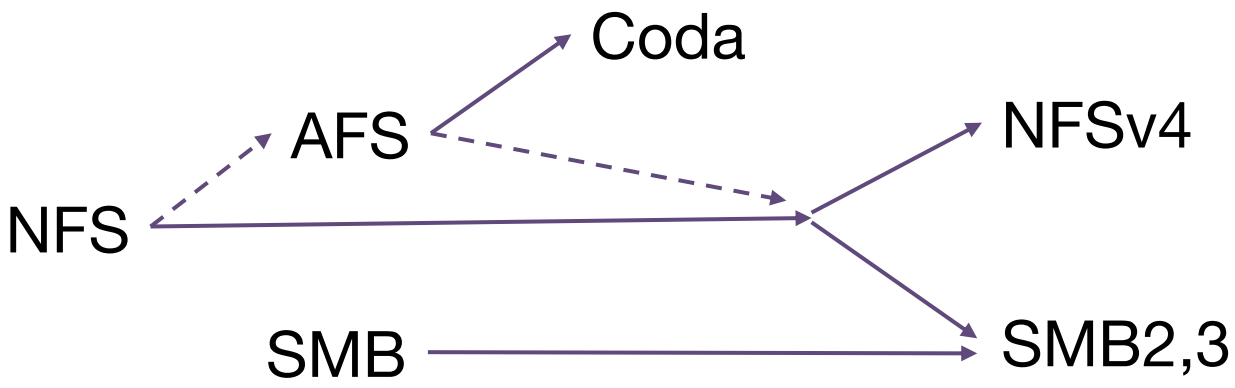
Callbacks:

- Server sends invalidations to clients with cached data when contents are modified

Centralized state:

- Supporting callbacks (and locking) requires the server to track who has cached data

Let's Look at Some Network Attached Storage



Represents inspiration

Represents evolution

NFS (Network File System)

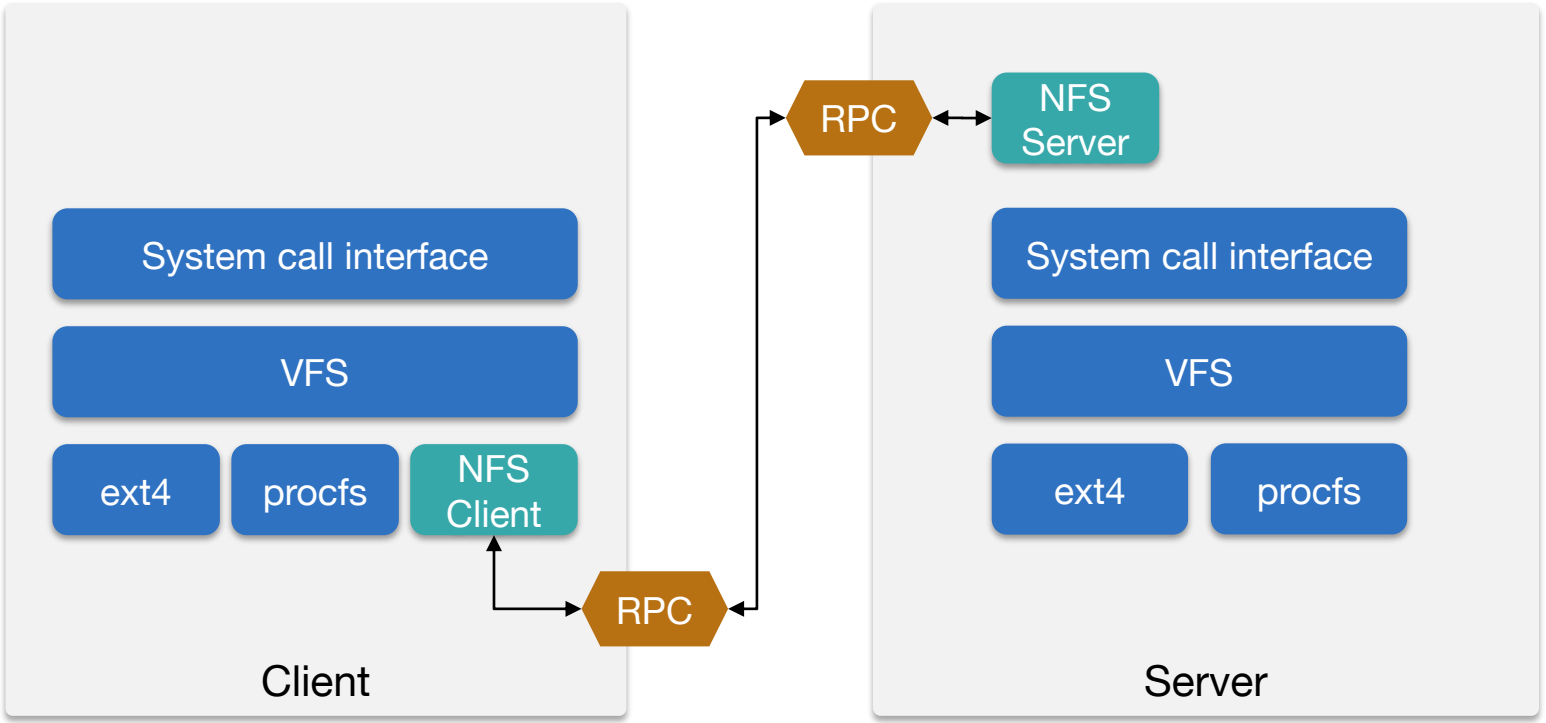
Sun Microsystems

Design Goals

- **Cross-platform interoperability**
  - Any system can be an NFS client or server
  - Interface built on Sun (ONC) RPC with XDR for encoding
- **Stateless design**
  - Clients should be able to continue if the server crashes and reboots
  - Timed-out requests are retransmitted
  - Initially used UDP
    - Easy to manage retransmissions
    - No need to set up connections – so no work for temporary loss of network or server
- **Simplicity**
  - Minimal RPC interface for basic file access

Result: not designed to support all Unix file system operations or semantics

VFS on client; Server accesses local file system



NFSv2 RPC Interface (Original Protocol - RFC 1094)

- 16 functions
- No *open, close, lock, seek*: those would require state

null
lookup

create
remove
rename

read
write

link
symlink
readlink

mkdir
rmdir
readdir

getattr
setattr

statfs

Directory and file access protocol

First, perform a ***lookup*** RPC

- returns **file handle** and attributes
- lookup is ***not*** like *open:* No information is stored on server

File handle passed as a parameter for other file access functions

- `read(handle, offset, count)`

NFS Caching

- Transfer data in **blocks** (8 KB by default)
  - Even if a process asked to read 1 byte
- **Read-ahead**
  - Prefetch additional blocks since most processes read an entire file
- No server state = no callbacks
  - Clients use **timestamp validation**
    - Before using cached data, check the modification time on the server
    - If the server's file modification time is newer then discard cached data for the file
- **Problem**: The timestamp is checked only when the file is open or after a cache timeout (~3 seconds for files; ~30 seconds for directories)
  - Modified blocks:
    - Scheduled to be sent but not sent immediately; unsent blocks flushed on file close

Problems with NFS

- **File locking**
  - Not possible because locking needs server state
  - Added through a separate service: **Network Lock Manager**
- **Appending to a file**
  - No APPEND RPC, so use GETATTR to get size + WRITE at that offset
  - But the file size could change between GETATTR and WRITE
- **Reference counting of open files**
  - Requires state – but that means a file on the server can disappear
  - **Silly renaming**: client renames the file to a hidden name and deletes on close
- **Security**
  - NFS sends user ID and group ID in each call. IDs can be impersonated and may not map to the right user on the server

AFS (Andrew File System)

Carnegie Mellon University

AFS

- Design Goal
  - Support information sharing on a *large* scale
    e.g., 10,000+ clients

- History
  - Developed at CMU
  - Became a commercial spin-off: Transarc
  - IBM acquired Transarc
  - Open source under IBM Public License
  - OpenAFS (openafs.org)

AFS Design Assumptions

- Most files are small
- Reads are more common than writes
- Most files are accessed by one user at a time
- Files are referenced in bursts (locality)
  - Once referenced, a file is likely to be referenced again

AFS Whole-File Caching

Upload/download model

AFS = Session Semantics

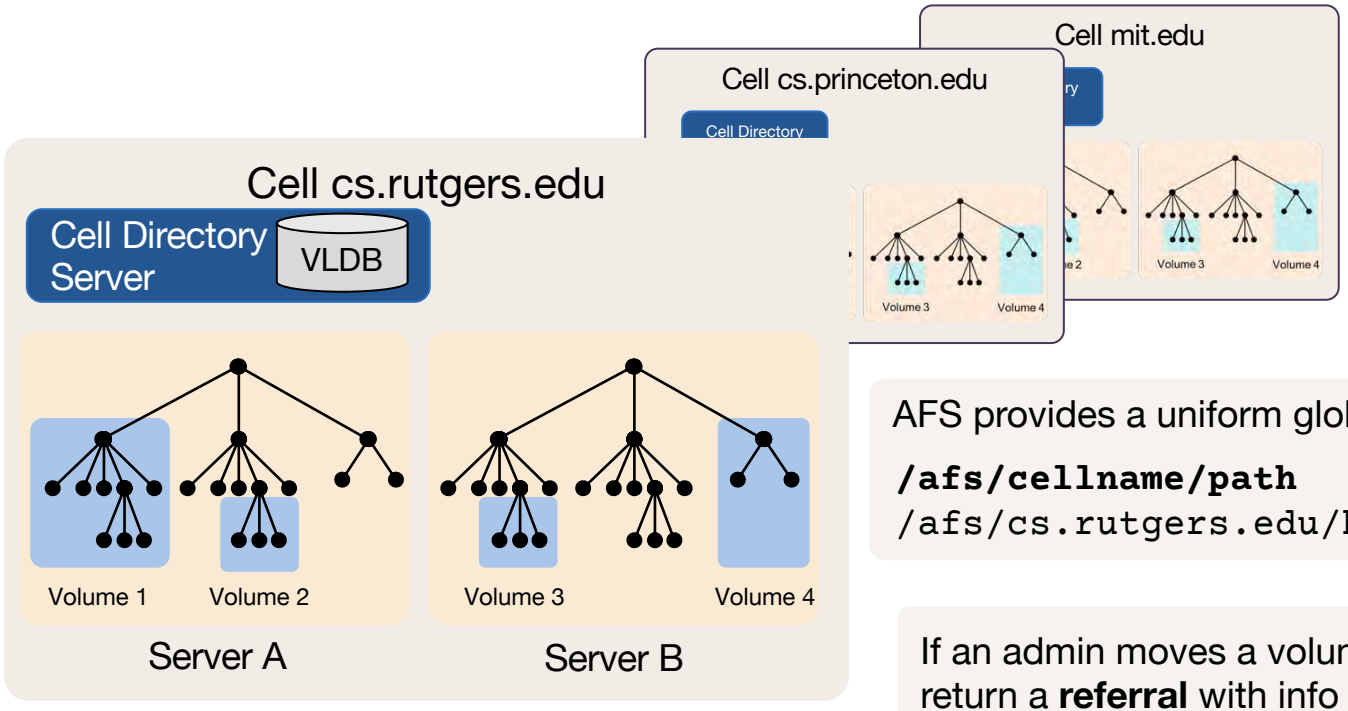
- When a process opens a file
  - Client fetches large chunks of the file on demand
  - Read/writes work on cached copies – no interaction with the server
- When a process closes a file
  - The client uploads the contents to the server if the file was modified

Callbacks

- Server tracks file downloads
- **Callback promise**: notify the client if the file changes at the server
- Clients can use cached data indefinitely until they get a **callback revocation**

AFS Uniform Namespace: Volumes & Cells

NFS (and Microsoft SMB) lets you mount resources anywhere in the directory tree



AFS provides a uniform global namespace

`/afs/cellname/path`

`/afs/cs.rutgers.edu/home/paul/notes.txt`

If an admin moves a volume, the old server can return a **referral** with info about the new location

Coda (COnstant Data Availability)

Carnegie-Mellon University

Design Goals

Originated from AFS

1. Provide better support for replication than AFS
   - Support shared read/write files

2. Support mobility of PCs
   - Provide constant data availability in **disconnected environments**
   - Use hoarding (user-directed caching)
   - Log updates on client
     - Reintegrate on connection to network (server)

Replicated Storage

Volumes can be replicated across multiple servers

- **Volume Storage Group** (**VSG**) = { set of replicated volumes }
- **Accessible Volume Storage Group** (**AVSG**)
  = { VSG members you can reach now }

- **Reads**: from any accessible server
- **Writes**: sent to ***all*** accessible servers
- On *open,* the client checks that all accessible servers have the same version of the file
  - If not, it initiates a **resolution process**

Disconnected Operation Mode: AVSG = Ø

- Client works directly from cache
  - Files not in the cache are not available
- File modifications are recorded in a **Client Modification Log** (**CML**)
  - Store, create, remove, rename, mkdir, rmdir
- On reconnection,
  - **Reintegration**: the client plays back the CML to upload
  - Optimized to send only the latest changes
- Detect conflicts between the server and client

Hoarding: User-Directed Caching

Keep important files up to date

- Ask server to send updates if necessary

Hoard database

- Automatically built by monitoring the user's activity
- And user-directed pre-fetch

Lessons

AFS and Coda aren't in wide use

But they pointed to some useful concepts

- Long term-caching with server callbacks
- Referrals to move locations

SMB (Server Message Block)

Microsoft

Microsoft SMB Protocol

- **Opposite philosophy to NFS**
- Connection-oriented (not UDP)
- Extensive server state
  - Open files, advisory/mandatory locks, byte range locking

Full support for all Microsoft file operations

- No goal of interoperability
- **No caching**: all operations would go to the server
- **Not resilient to network/server disruptions:**
  - All sessions, open files, locks lost if server reboots
  - Connections break if network glitches

SMB Commands

- Files
  - Get disk attributes
  - create/delete directories
  - search for file(s)
  - create/delete/rename file
  - lock/unlock file area
  - open/commit/close file
  - get/set file attributes

- Print-related
  - Open/close spool file
  - write to spool
  - Query print queue

- User-related
  - Discover home system for user
  - Send message to user
  - Broadcast to all users
  - Receive messages

Improving SMB Performance via Caching

Increase effective performance with

- Caching
  - Safe if multiple clients reading, nobody writing
- read-ahead
  - Safe if multiple clients reading, nobody writing
- write-behind
  - Safe if only one client is accessing file

Goal: minimize times client informs server of changes

Microsoft Oplocks

Server-granted capability to tell a client what kind of caching is safe

- **Exclusive (Level 1) oplock**: only one client has the file open
  - Client can cache reads & writes
- **Level 2 oplock**: multiple readers, no writer
  - Cache reads locally
  - Oplock broken if any client tries to write
- **Batch oplock**: designed for scripts that repeatedly open/close same file
  - Client keeps file open on the server even if a local process closes it
- **Filter oplock**: exclusive access but yields to any other process
  - Designed for file system/malware scanners that run in the background

Oplocks evolved to SMB Leases (SMB ≥ 2.1; Windows ≥ 7)

Update (cleanup) to oplocks — same purpose as oplock: control caching

- Lease types
  - Read-cache (R) lease: cache results of *read*; can be shared
  - Write-cache (W) lease: cache results of *write*; exclusive
  - Handle-cache (H) lease: cache file handles; can be shared
    - Optimizes re-opening files
- Leases can be combined: R, RW, RH, RWH
- Leases define oplocks:
  - *Read oplock* (R) – essentially same as Level 2
  - Read-handle (RH) – essentially same as Batch
  - Read-write (RW) – essentially the same as Level 1

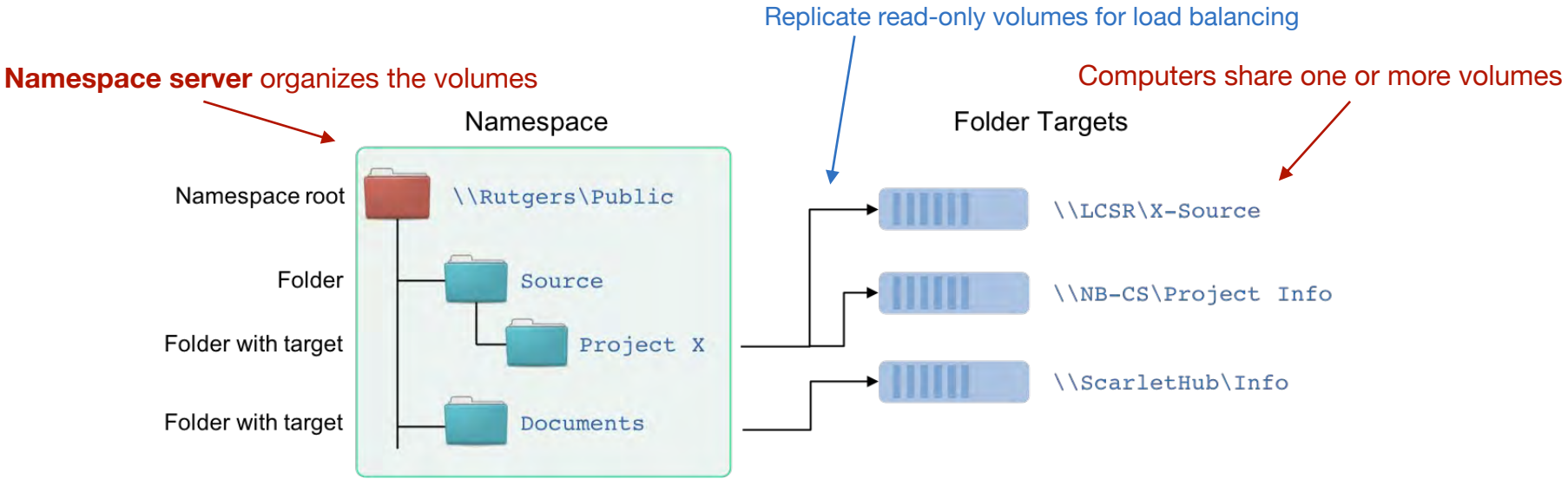
See https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/oplock-overview

https://blogs.msdn.microsoft.com/openspecification/2009/05/22/client-caching-features-oplock-vs-lease/

Microsoft DFS Namespaces

"Distributed File System": Windows Server Namespace Layer above SMB

- Shared folders from different servers can be organized into one file system view
- Provide location transparency



DFS = SMB + naming/ability to mount server shares on other server shares

Evolution of Network-Attached Storage

SMB 2+

Reduced complexity

- From >100 commands to 19

- **Pipelining**
  - Send additional commands before the response to a previous one is received

- **Compounding**
  - Avoid the need to have commands that combine operations
  - Send a set of commands in one request

- **Durable handles**
  - Allow reconnection to server if there was a temporary loss of connectivity

More SMB2 Additions

- Larger reads/writes
- Caching of folder & file properties
- "Durable handles"
  - Allow reconnection to server if there was a temporary loss of connectivity

Sample SMB2 vs. SMB comparison

Transfer 10.7 GB over 1 Gbps WAN link with 76 ms RTT

SMB: 5 hours 40 minutes: rate = 0.56 MB/s

SMB2: 7 minutes, 45 seconds: rate = 25 MB/s

SMB3

Key features

- Multichannel support for network scaling
- Transparent network failover
- “SMBDirect” – support for Remote DMA in clustered environments
  - Enables direct, low-latency copying of data blocks from remote memory without CPU intervention
- Direct support for virtual machine files
  - Volume Shadow Copy
  - Enables volume backups to be performed while apps continue to write to files.
- End-to-end encryption

NFS v4: Abandoning Statelessness

- Clients now open/close files; server tracks state
  - Clients can cache aggressively
- TCP protocol is mandatory
- Caching controlled with **delegations**
  - Similar to SMB oplocks/leases
  - Server grants rights to perform certain operations without contacting the server
  - **Callbacks** to notify clients of changes
- **Compound RPC**: multiple operations in one request
- **Referrals**: server can return a referral if a path has been moved

Convergence

NFS and SMB converged to adopt many similar mechanisms

Mechanism	NFSv4	SMB 2+
Stateful server	Yes	Yes
Compound/pipelined requests	Yes (compound RPC)	Yes (compounding + pipelining)
Client caching grants	Yes (delegations)	Yes (oplocks/leases)
Server-to-client notification	Yes (via delegation recall)	Yes (oplock break / lease break)
Referrals	Yes	Yes (via DFS)
Strong authentication	Mandatory	Kerberos / NTLMv2
Transport	TCP only	TCP

The End