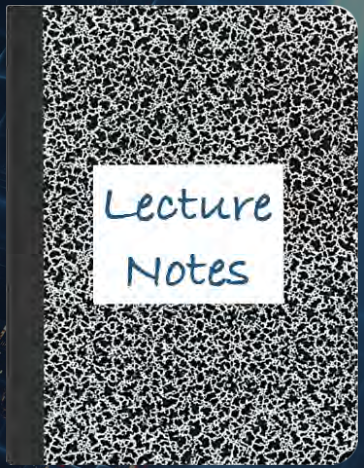


CS 417 – DISTRIBUTED SYSTEMS

Week 6: Distributed File Systems

Part 3: Other Remote File Systems

Paul Krzyzanowski

AFS

Andrew File System

Carnegie Mellon University

c. 1986(v2), 1989(v3)

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 Design Decisions

Whole file serving

- Send the entire file on *open*

Long-term whole file caching

- Client caches entire file on local disk
- Client writes the file back to server on *close*
  - if modified
  - Keeps cached copy for future accesses

AFS Server: cells

Servers are grouped into administrative entities called cells

- **Cell**: collection of
  - Servers
  - Administrators
  - Users
  - Clients
- Each cell is autonomous, but cells may cooperate and present users with one **uniform name space**

AFS Server: volumes

Disk partition contains

file and directories

Grouped into volumes

Volume

- Administrative unit of organization
  E.g., user's home directory, local source, etc.
- Each volume is a directory tree (one root)
- Assigned a name and ID number
- A server will often have 100s of volumes

Namespace management

Clients get information via cell directory server (Volume Location Server) that hosts the Volume Location Database (VLDB)

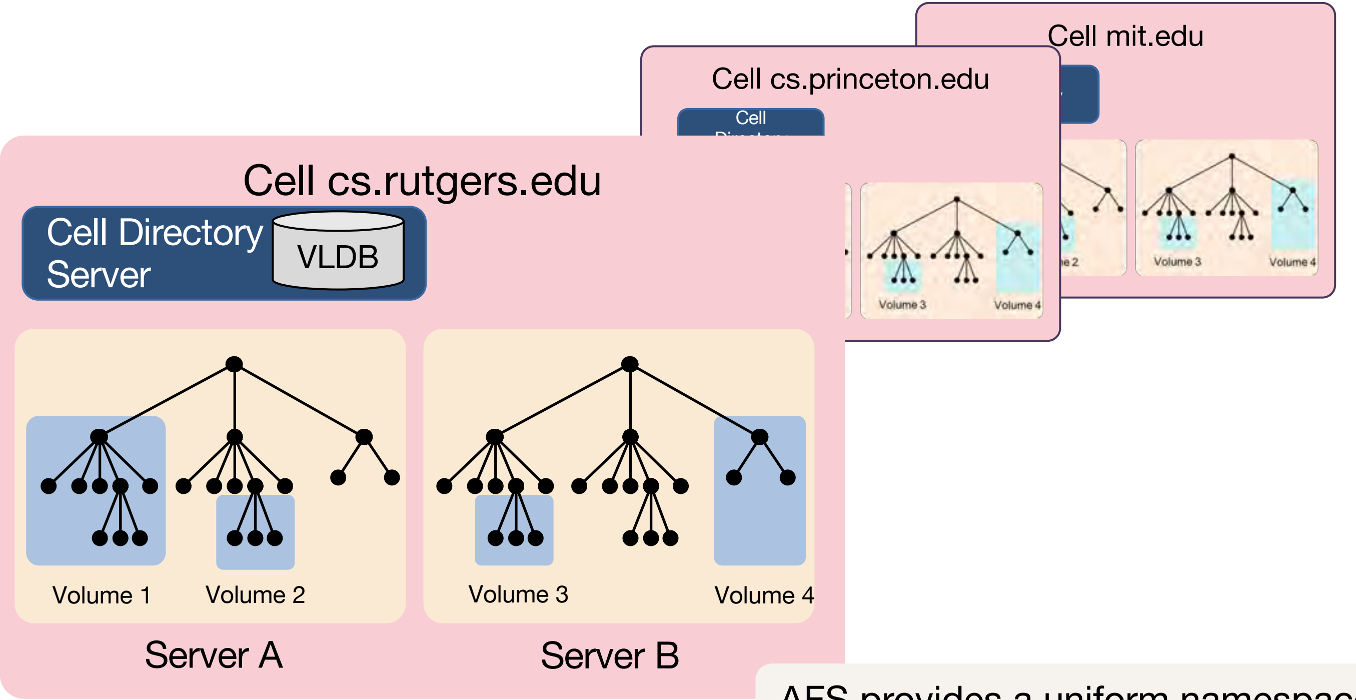
Goal:

everyone sees the same namespace

```
/afs/cellname/path

/afs/mit.edu/home/paul/src/try.c
```

Files, Directories, Volumes, Cells



AFS provides a uniform namespace from anywhere

`/afs/cellname/path`

`/afs/mit.edu/home/paul/src/try.c`

Communication with the server

- Communication is via RPC over UDP

- Access control lists used for protection
  - Directory granularity
  - UNIX permissions ignored (except execute)

AFS cache coherence

On open:

- Server sends entire file to client

  and provides a callback promise:

- *It will notify the client when any other process modifies the file*

If a client modified a file:

- Contents are written to server on *close*

Callbacks: when a server gets an update:

- it notifies *all* clients that have been issued the callback promise
- Clients invalidate cached files

AFS cache coherence

If a client was down

- On startup, contact server with timestamps of all cached files to decide whether to invalidate

If a process has a file open

- It continues accessing it even if it has been invalidated
- Upon close, contents will be propagated to server

AFS: Session Semantics

(vs. sequential semantics)

AFS replication and caching

- Limited replication
  - Read-only volumes may be replicated on multiple servers
- Advisory locking supported
  - Query server to see if there is a lock
- Referrals
  - An administrator may move a volume to another server
  - If a client accesses the old server, it gets a *referral* to the new one

AFS key concepts

- Single global namespace
  - Built from a collection of volumes across cells
  - Referrals for moved volumes
  - Replication of read-only volumes
- Whole-file caching
  - Offers dramatically reduced load on servers
- Callback promise
  - Keeps clients from having to poll the server to invalidate cache

AFS summary

AFS benefits

- AFS scales well
- Uniform name space
- Read-only replication
- Security model supports mutual authentication, data encryption

AFS drawbacks

- Session semantics
- Directory based permissions
- Uniform name space

DFS (based on AFS v3)
Distributed File System

DFS

AFS: scalable performance but session semantics were hard to live with

- Goal
  - Create a file system similar to AFS but with a **strong consistency** model
- History
  - Part of Open Group's Distributed Computing Environment (DCE)
  - Descendant of AFS - AFS version 3.x
- Assume (like AFS):
  - Most file accesses are sequential
  - Most file lifetimes are short
  - Majority of accesses are whole file transfers
  - Most accesses are to small files

Caching and Server Communication

- Increase effective performance with
  - Caching data that you read
    - Safe if multiple clients reading, nobody writing
  - read-ahead
    - Safe if multiple clients reading, nobody writing
  - write-behind (delaying writes to the server)
    - Safe if only one client is accessing file

Goal:

Minimize # of times client informs server of changes —
but do so in a way that clients all have valid data

DFS Tokens

Cache consistency maintained by **tokens**

Token

- Guarantee from server that a client can perform certain operations on a cached file
- Server grants & revokes tokens

- ***Open*** **tokens**
  - Allow token holder to open a file
  - Token specifies access (read, write, execute, exclusive-write)
- ***Data*** **tokens**
  - Applies to a byte range
  - *read* token - can use cached data
  - *write* token - write access, cached writes
- ***Status*** **tokens**
  - *read*: can cache file attributes
  - *write*: can cache modified attributes
- ***Lock*** **tokens**
  - Holder can lock a byte range of a file

Living with tokens

- Server grants and revokes tokens
  - Multiple *read* tokens OK
  - Multiple *read* and a *write* token or multiple *write* tokens
    - *Not OK if byte ranges overlap*
    - Revoke all other *read* and *write* tokens
    - Block new request and send revocation to other token holders

DFS key points

- Caching
  - Token granting mechanism
    - Allows for long term caching <u>and</u> strong consistency
  - Caching sizes: 8K – 256K bytes
  - Read-ahead (like NFS)
    - Don’t have to wait for entire file before using it as with AFS
- File protection via access control lists (ACLs)
- Communication via authenticated RPCs
- Essentially AFS v3 with server-based token granting
  - Server keeps track of who is reading and who is writing files
  - Server must be contacted on each open and close operation to request token

Coda
COnstant Data Availability
Carnegie-Mellon University

c. 1990-1992

Coda 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)

Modifications to AFS

Support replicated file volumes

- A volume can be replicated on a group of servers
  - **Volume Storage Group (VSG)**
- Replicated volumes
  - Volume ID used to identify files is a Replicated Volume ID
  - One-time lookup
    - Replicated volume ID → list of servers and *local* volume IDs
  - Read files from *any* server
  - Write to **all available servers**

Disconnected volume servers

AVSG: Accessible Volume Storage Group

- Subset of VSG

On first download, contact everyone you can and get a version timestamp of the file

If the client detects that some servers have old versions

- Client initiates a **resolution process**
  - Notifies server of stale data
  - Resolution handled entirely by servers
  - Administrative intervention may be required (if conflicts)

AVSG = Ø

- If no servers are accessible
  - Client goes to **disconnected operation mode**
- If file is not in cache
  - Nothing can be done… fail
- Do not report failure of update to server
  - Log update locally in **Client Modification Log** (CML)
  - User does not notice

Reintegration

Upon reconnection

- Commence **reintegration**

Bring server up to date with CML **log playback**

- Optimized to send latest changes

Try to resolve conflicts automatically

- Not always possible

Support for disconnection

Keep important files up to date

- Ask server to send updates if necessary

Hoard database

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

Coda summary

- Session semantics as with AFS
- Replication of read/write volumes
  - Clients do the work of writing replicas (extra bandwidth)
  - Client-detected reintegration
- Disconnected operation
  - Client modification log
  - Hoard database for needed files
    - User-directed pre-fetch
  - Log replay on reintegration

SMB
Server Message Block Protocol
Microsoft

c. 1987

SMB Goals

- File sharing protocol for Windows 9x - Windows 10, Window NT-20xx
- Protocol for sharing
  - Files, devices, communication abstractions (named pipes), mailboxes
- Servers: make file system and other resources available to clients
- Clients: access shared file systems, printers, etc. from servers

Design Priority: locking and consistency over client caching

SMB Design

- Request-response protocol – similar to RPC
  - Send and receive ***message blocks***
    - name from old DOS system call structure
  - Send *request* to server – the PC with the resource you want
  - Server sends response
- Connection-oriented protocol
  - Persistent connection – “session”
- Each message contains:
  - Fixed-size header
  - Command string (based on message) or reply string

Message Block

- Header: [fixed size]
  - Protocol ID
  - Command code (0..FF)
  - Error class, error code
  - Tree ID – unique ID for resource in use by client (handle)
  - Caller process ID
  - User ID
  - Multiplex ID (to route requests in a process)

- Command: [variable size]
  - Param count, params, #bytes data, data

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

Protocol Steps

- Establish connection

Protocol Steps

- Establish connection
- Negotiate protocol
  - ***negprot*** SMB
  - Responds with version number of protocol

Protocol Steps

- Establish connection
- Negotiate protocol
- Authenticate/set session parameters
  - Send ***sesssetupX*** SMB with username, password
  - Receive NACK or UID of logged-on user
  - UID must be submitted in future requests

Protocol Steps

- Establish connection
- Negotiate protocol - *negprot*
- Authenticate - *sesssetupX*
- Make a connection to a resource (similar to *mount*)
  - Send *tcon* (tree connect) SMB with name of shared resource
  - Server responds with a **tree ID** (TID) that the client will use in future requests for the resource

Protocol Steps

- Establish connection
- Negotiate protocol - *negprot*
- Authenticate - *sesssetupX*
- Make a connection to a resource – *tcon*
- Send open/read/write/close/… SMBs

SMB Evolves

Common Internet File System (1996)

SMB 2 (2006)

SMB 3 (2012)

SMB Evolves

- History
  - SMB was reverse-engineered for non-Microsoft platforms
    - samba.org
    - E.g., Linux & macOS use Samba to access file shares from Windows
  - Microsoft released SMB protocol to X/Open in 1992
  - Common Internet File System (CIFS)
    - SMB as implemented in 1996 for Windows NT 4.0
  - SMB 2.0: 2006
  - SMB 3.0: 2012
  - SMB 3.1: 2016

Caching and Server Communication

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

Oplocks

Server grants **opportunistic locks** (**oplocks**) to client

- Clients request oplocks from a server so they can cache data
- Oplock tells client how/if it may cache data
- Similar to DFS tokens (but more limited)

Client must request an oplock

- The oplock may be
  - Granted
  - Revoked by the server at some future time
  - Changed by server at some future time

Level 1 oplock (exclusive access)

- Client can open file for exclusive access
- Arbitrary caching
- Cache lock information
- Read-ahead
- Write-behind

If another client opens the file, the server has former client *break its oplock*:

- Client must send server any lock and write data and acknowledge that it does not have the lock
- Purge any read-aheads

Level 2 oplock (multiple readers, no writers)

- Level 1 oplock is replaced with a Level 2 oplock if another process tries to read the file
- Multiple clients may have the same file open as long as none are writing
- Cache reads, file attributes
  - Send other requests to server
- Level 2 oplock revoked if any client opens the file for writing

Batch oplock (remote open even if local closed)

- Client can keep file open on server even if a local process that was using it has closed the file
- Client requests batch oplock if it expects programs may behave in a way that generates a lot of traffic by opening & closing same files over and over
  - Designed for Windows batch files
- Batch oplock is exclusive: one client only
  - revoked if another client opens the file

Filter oplock (allow preemption)

- Allow apps to look through file data but be notified if someone else wants access

- Allow clients with *filter oplock* to be suspended while another process preempted file access
  - Indexing service can run and open files without causing programs to get an error when they need to open the file
    - Indexing service is notified that another process wants to access the file
    - It can abort its work on the file and close it or finish its indexing and then close the file

No oplock

- A server can ***break*** an oplock – tell a client it no longer has the oplock
- All requests must be sent to the server
- Can work from cache only if byte range was locked by client

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

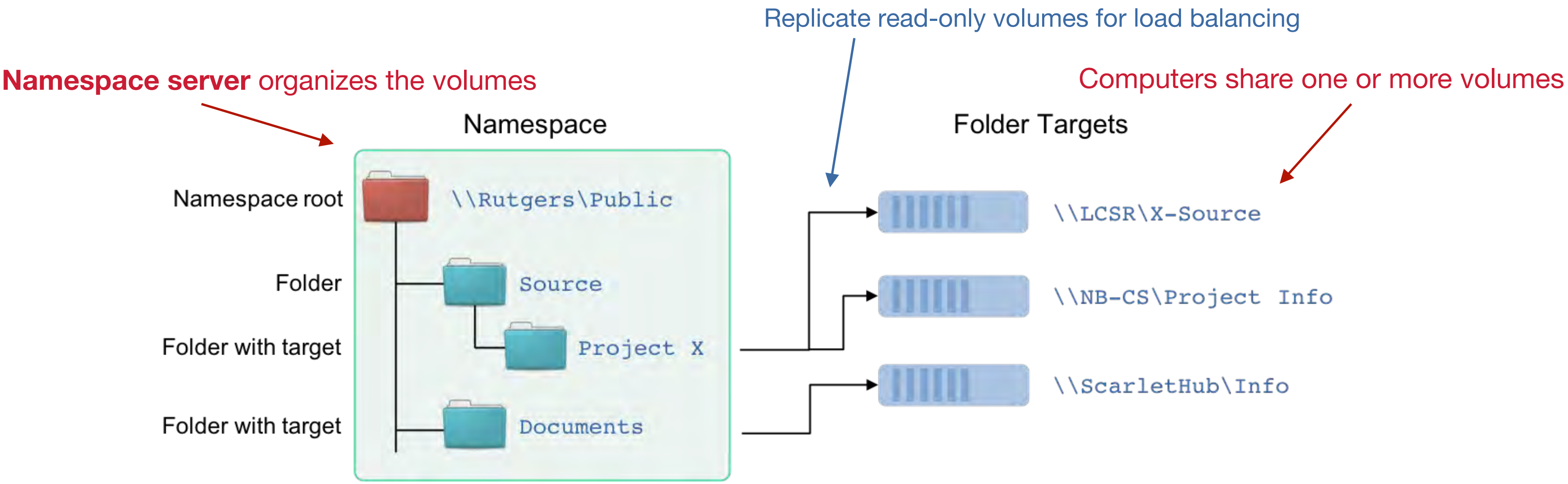
See https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/oplock-overview

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

Microsoft DFS Namespaces

"Distributed File System": Service in Windows Server

- 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

SMB Summary

- Stateful model with strong consistency
- Oplocks/leases offer flexible control for distributed consistency
- DFS adds namespace management to create a common hierarchy

SMB2 and SMB3

- Original SMB was…
  - Chatty: common tasks often required multiple round-trip messages
  - Not designed for WANs
- **SMB2** (2007)
  - Protocol dramatically cleaned up
  - New capabilities added
  - SMB2 became the default network file system in macOS Mavericks (10.9)
- **SMB3** (2012)
  - Added RDMA and multichannel support; end-to-end encryption
    - RDMA = Remote DMA (Direct Memory Access)
  - Windows 8 / Windows Server 2012: SMB 3.0
  - SMB3 became the default network file system in macOS Yosemite (10.10)

SMB2 Additions: Message Optimization

- Reduced complexity
  - From >100 commands to 19
- **Pipelining** support
  - Send additional commands before the response to a previous one is received
- **Compounding** support
  - Avoid the need to have commands that combine operations
  - Send an arbitrary set of commands in one request
  - E.g., instead of *RENAME*:
    - CREATE (create new file or open existing)
    - SET_INFO
    - CLOSE

SMB2 Additions: Credit-Based Flow Control

Credit-based flow control

Goal: keep more data in flight but avoid overloading servers

- Client session starts with a small # of “credits” and scales up as needed
- Each SMB request to the server costs one credit
  - Client decrements the credit count each time it sends a message
  - The server responds back with more credits
- If a server gets more loaded, it can issue fewer credits

Allows servers to control the amount of traffic from each client

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 benefits

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 version 4
Network File System
Sun Microsystems (now Oracle)

NFS version 4 enhancements

- Stateful server
- Compound RPC
  - Group operations together
  - Receive set of responses
  - Reduce round-trip latency
- Stateful open/close operations
  - Supports exclusive creates
  - Client can cache aggressively

NFS version 4 enhancements

- create, link, open, remove, rename
  - Inform client if the directory changed during the operation
- Strong security
  - Extensible authentication architecture
- File system read/write replication and migration
  - Mirror servers can be configured
    - If a client accesses a file on a replicated server, the server disables replication, and all requests go to that server until the client is done
  - Clients don't need to know where the data is: server will send **referrals**

NFS version 4 enhancements

- Stateful locking
  - Clients inform servers of lock requests
  - Locking is lease-based; clients must renew leases
- Improved caching
  - Server can delegate specific actions on a file to enable more aggressive client caching
  - Close-to-open consistency
    - File changes propagated to server when file is closed
    - Client checks timestamp on open to avoid accessing stale cached copy
  - Similar to Windows oplocks
    - Clients must disable caching to share files
- Callbacks
  - Notify client when file/directory contents change

Review: Core Concepts

- NFS
  - RPC-based access, stateless design (initially)
- AFS
  - Long-term caching
- DFS
  - AFS + tokens for consistency and efficient caching
- Coda
  - Read/write replication & disconnected operation
- SMB
  - RPC-like access with strong consistency
  - Oplocks to support caching
  - DFS Namespaces: add-on to provide a consistent view of volumes (AFS-style)

The End