Distributed File Systems

Chien-Min Wang
Institute of Information Science
Academia Sinica

Contents

- File System Overview
- Distributed File Systems: Issues
- Distributed File Systems: Case Studies
- Distributed File Systems for Clouds

Lecture 3
Distributed File Systems: Case Studies

Outline

- Network File System (NFS)
- Andrew File System (AFS)
- Server Message Blocks (SMB)
- Common Interface File System (CIFS)

Network File System

- NFS (Network File System)
  - Developed by Sun Microsystems (in 1985)
  - Most popular, open, and widely used.
  - NFS protocol standardised through IETF (RFC 1813)

NFS Design Goals

- Any machine can be a client or server
- Must support diskless workstations
- Heterogeneous systems must be supported
  - Different HW, OS, underlying file system
- Access transparency
  - Remote files accessed as local files through normal file system calls (via VFS in UNIX)
- Recovery from failure
  - Stateless, UDP, client retries
- High Performance
  - Use caching and read-ahead

NFS Design Goals

- No Migration Transparency
  - If resource moves to another server, client must remount resource.
- No support for UNIX file access semantics
  - Stateless design: file locking is a problem.
  - All UNIX file system controls may not be available.
- Devices
  - Must support diskless workstations where every file is remote.
  - Remote devices refer back to local devices.

NFS Design Goals

- Transport Protocol
  - Initially NFS ran over UDP using Sun RPC
- Why UDP?
  - Slightly faster than TCP
  - No connection to maintain (or lose)
  - NFS is designed for Ethernet LAN environment – relatively reliable
  - Error detection but no correction.
  - NFS retries requests

NFS Protocols

- Mounting protocol
  - Request access to exported directory tree
- Directory & File access protocol
  - Access files and directories
    (read, write, mkdir, readdir, …)

Mounting Protocol

- Send pathname to server
- Request permission to access contents

<u>client</u>: parses pathname contacts server for file handle

- Server returns **file handle**
  - File device #, inode #, instance #

<u>client</u>: create in-code vnode at mount point. (points to inode for local files) points to rnode for remote files - *stores state on client*

Mounting Protocol

static mounting

- mount request contacts server

Server: edit `/etc/exports`

Client: `mount fluffy:/users/paul /home/paul`

Directory and File Access Protocol

- First, perform a lookup RPC
  - returns file handle and attributes
- Not like open
  - No information is stored on server
- Handle passed as a parameter for other file access functions
  - e.g. read(handle, offset, count)

Directory and File Access Protocol

- NFS has 16 functions
  - (version 2; six more added in version 3)

null
lookup

create
remove
rename

read
write

link
symlink
readlink

mkdir
rmdir
readdir

getattr
setattr

statfs

NFS Performance

- Usually slower than local
- Improve by caching at client
  - Goal: reduce number of remote operations
  - Cache results of
    read, readlink, getattr, lookup, readdir
  - Cache file data at client (buffer cache)
  - Cache file attribute information at client
  - Cache pathname bindings for faster lookups
- Server side
  - Caching is “automatic” via buffer cache
  - All NFS writes are write-through to disk to avoid unexpected data loss if server dies

Inconsistencies may arise

- Try to resolve by validation
  - Save timestamp of file
  - When file opened or server contacted for new block
    - Compare last modification time
    - If remote is more recent, invalidate cached data

Validation

- Always invalidate data after some time
  - After 3 seconds for open files (data blocks)
  - After 30 seconds for directories
- If data block is modified, it is:
  - Marked *dirty*
  - Scheduled to be written
  - Flushed on file close

Improving Read Performance

- Transfer data in large chunks
  - 8K bytes default
- Read-ahead
  - Optimize for sequential file access
  - Send requests to read disk blocks before they are requested by the application

Problems with NFS

- File consistency
- Assumes clocks are synchronized
- Open with append cannot be guaranteed to work
- Locking cannot work
  - Separate lock manager added (stateful)
- No reference counting of open files
  - You can delete a file you (or others) have open!
- Global UID space assumed

Problems with NFS

- No reference counting of open files
  - You can delete a file you (or others) have open!
- Common practice
  - Create temp file, delete it, continue access
  - Sun's hack:
    - If same process with open file tries to delete it
    - Move to temp name
    - Delete on close

Problems with NFS

- File permissions may change
  - Invalidating access to file
- No encryption
  - Requests via unencrypted RPC
  - Authentication methods available
    - Diffie-Hellman, Kerberos, Unix-style
  - Rely on user-level software to encrypt

Improving NFS: version 2

- User-level lock manager
  - Monitored locks
    - status monitor: monitors clients with locks
    - Informs lock manager if host inaccessible
    - If server crashes: status monitor reinstates locks on recovery
    - If client crashes: all locks from client are freed
- NV RAM support
  - Improves write performance
  - Normally NFS must write to disk on server before responding to client write requests
  - Relax this rule through the use of non-volatile RAM

Improving NFS: version 2

- Adjust RPC retries dynamically
  - Reduce network congestion from excess RPC retransmissions under load
  - Based on performance
- Client-side disk caching
  - cacheFS
  - Extend buffer cache to disk for NFS
    - Cache in memory first
    - Cache on disk in 64KB chunks

The automounter

- Problem with mounts
  - If a client has many remote resources mounted, boot-time can be excessive
  - Each machine has to maintain its own name space
    - Painful to administer on a large scale
- Automounter
  - Allows administrators to create a global name space
  - Support on-demand mounting

Automounter

- Alternative to static mounting
- Mount and unmount in response to client demand
  - Set of directories are associated with a local directory
  - None are mounted initially
  - When local directory is referenced
    - OS sends a message to each server
    - First reply wins
  - Attempt to unmount every 5 minutes

Automounter

Example:

`automount /usr/src srcmap`

`srcmap` contains:

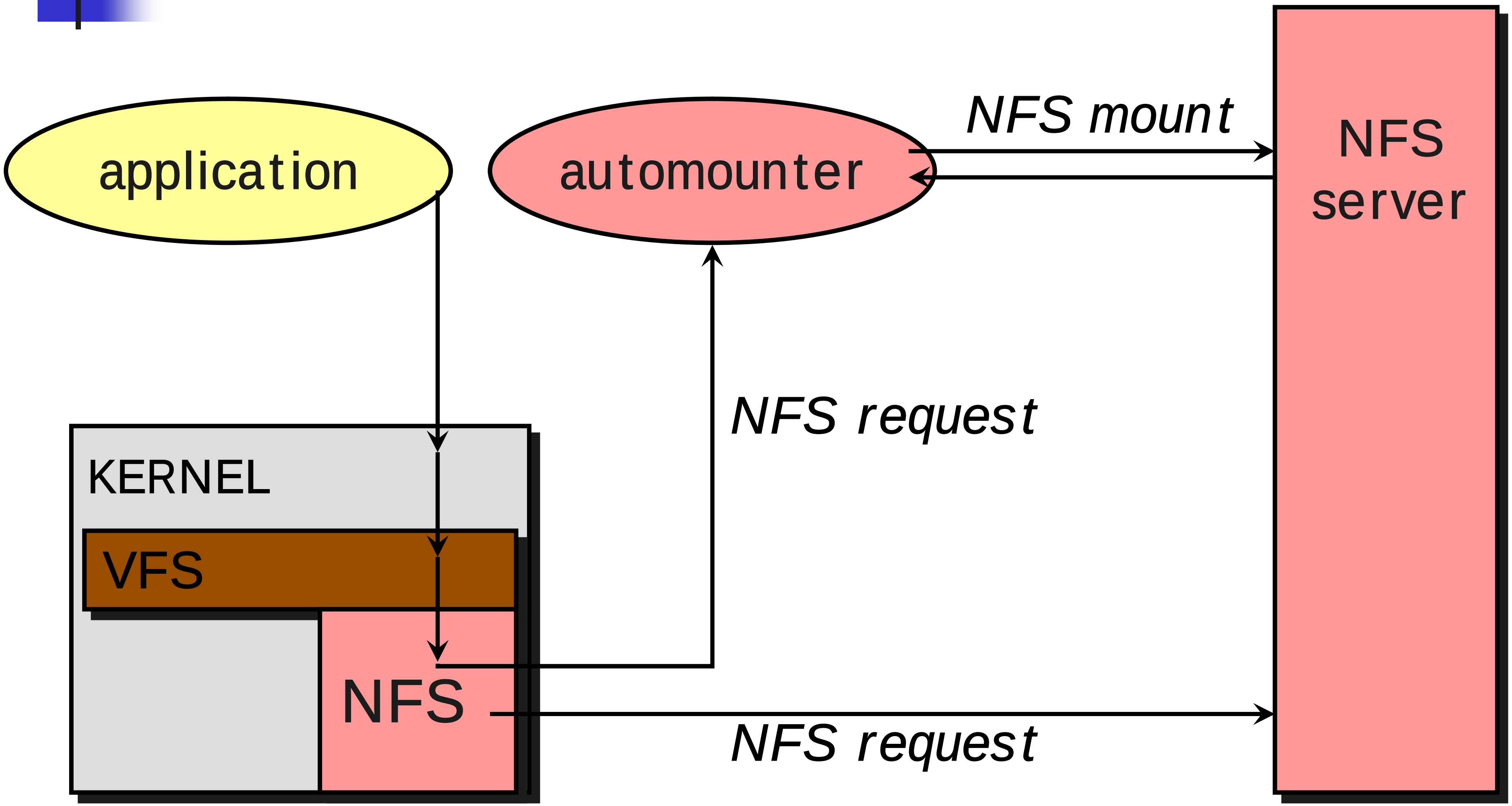
```
cmd         -ro   doc:/usr/src/cmd
kernel      -ro   frodo:/release/src \
                  bilbo:/library/source/kernel
lib         -rw   sneezy:/usr/local/lib
```

Access **`/usr/src/cmd`**: request goes to doc

Access **`/usr/src/kernel`**:

ping frodo and bilbo, mount first response

The automounter



More Improvements: NFS v3

- Updated version of NFS protocol
- Support 64-bit file sizes
- TCP support and large-block transfers
  - UDP caused more problems on WANs (errors)
  - All traffic can be multiplexed on one connection
    - Minimizes connection setup
  - No fixed limit on amount of data that can be transferred between client and server
- Negotiate for optimal transfer size
- Server checks access for entire path from client

More Improvements: NFS v3

- New commit operation
  - Check with server after a write operation to see if data is committed
  - If commit fails, client must resend data
  - Reduce number of write requests to server
  - Speeds up write requests
    - Don’t require server to write to disk immediately
- Return file attributes with each request
  - Saves extra RPCs

Outline

- Network File System (NFS)
- Andrew File System (AFS)
- Server Message Blocks (SMB)
- Common Interface File System (CIFS)

Andrew File System (AFS)

- Developed at CMU
- Commercial spin-off
  - Transarc
- IBM acquired Transarc
- Currently open source under IBM Public License
- Also: OpenAFS, Arla, and Linux version

AFS Design Goal

Support information sharing on a *large* scale

e.g., 10,000+ systems

AFS 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
- 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 Design

- Each client has an AFS disk cache
  - Part of disk devoted to AFS (e.g. 100 MB)
  - Client manages cache in LRU manner
- Clients communicate with set of trusted servers
- Each server presents one identical name space to clients
  - All clients access it in the same way
  - Location transparent

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

Accessing an AFS File

1. Traverse AFS mount point

   E.g., /afs/cs.rutgers.edu
2. AFS client contacts Volume Location DB on Volume Location Server to look up the volume
3. VLDB returns volume ID and list of machines (>1 for replicas on read-only file systems)
4. Request root directory from any machine in the list
5. Root directory contains files, subdirectories, and mount points
6. Continue parsing the file name until another mount point (from step 5) is encountered. Go to step 2 to resolve it.

Internally on the Server

- Communication is via RPC over UDP
- Access control lists used for protection
  - Directory granularity
  - UNIX permissions ignored (except execute)

Authentication and Access

- Kerberos authentication:
  - Trusted third party issues tickets
  - Mutual authentication
- Before a user can access files
  - Authenticate to AFS with `klog` command
    - "Kerberos login" – centralized authentication
  - Get a token (ticket) from Kerberos
  - Present it with each file access
- Unauthorized users have id of
  `system:anyuser`

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

AFS Cache Coherence

- If a client modified a file:
  - Contents are written to server on close
- 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

AFS: Replication and Caching

- Read-only volumes may be replicated on multiple servers
- Whole file caching not feasible for huge files
  - AFS caches in 64KB chunks (by default)
  - Entire directories are cached
- Advisory locking supported
  - Query server to see if there is a lock

AFS Summary

- Whole file caching
  - offers dramatically reduced load on servers
- Callback promise
  - keeps clients from having to check with 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

Sample Deployment (2008)

- Intel engineering (2007)
  - 95% NFS, 5% AFS
  - Approx 20 AFS cells managed by 10 regional organization
  - AFS used for:
    - CAD, applications, global data sharing, secure data
  - NFS used for:
    - Everything else
- Morgan Stanley (2004)
  - 25000+ hosts in 50+ sites on 6 continents
  - AFS is the primary distributed file system for all UNIX hosts
  - 24x7 system usage; near zero downtime
  - Bandwidth from LANs to 64 Kbps inter-continental WANs

Outline

- Network File System (NFS)
- Andrew File System (AFS)
- Server Message Blocks (SMB)
- Common Interface File System (CIFS)

SMB Goals

- File sharing protocol for Windows 95/98/NT/2000/ME/XP/Vista
- 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
  - Send and receive ***message blocks***
    - Name from old DOS system call structure
  - Send *request* to server (machine with resource)
  - 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 attr
  - create/delete directories
  - search for file(s)
  - create/delete/rename file
  - lock/unlock file area
  - open/commit/close file
  - get/set file attributes

SMB Commands

- 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
  - 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

Locating Services

- Clients can be configured to know about servers
- Each server broadcasts info about its presence
  - Clients listen for broadcast
  - Build list of servers
- Fine on a LAN environment
  - Does not scale to WANs
  - Microsoft introduced browse servers and the Windows Internet Name Service (WINS)
  - or … explicit pathname to server

Security

- Share level
  - Protection per “share” (resource)
  - Each share can have password
  - Client needs password to access all files in share
  - Only security model in early versions
  - Default in Windows 95/98
- User level
  - Protection applied to individual files in each share based on access rights
  - Client must login to server and be authenticated
  - Client gets a UID which must be presented for future accesses

Outline

- Network File System (NFS)
- Andrew File System (AFS)
- Server Message Blocks (SMB)
- Common Interface File System (CIFS)

SMB evolves

- SMB was reverse-engineered
  - samba under Linux
- Microsoft released protocol to X/Open in 1992
- Microsoft, Compaq, SCO, others joined to develop an enhanced public version of the SMB protocol:

Common Internet File System
(CIFS)

Original Goals

- Heterogeneous HW/OS to request file services over network
- Based on SMB protocol
- Support
  - Shared files
  - Byte-range locking
  - Coherent caching
  - Change notification
  - Replicated storage
  - Unicode file names

Original Goals

- Applications can register to be notified when file or directory contents are modified
- Replicated virtual volumes
  - For load sharing
  - Appear as one volume server to client
  - Components can be moved to different servers without name change
  - Use referrals
  - Similar to AFS

Original Goals

- Batch multiple requests to minimize round-trip latencies
  - Support wide-area networks
- Transport independent
  - But need reliable connection-oriented message stream transport
- DFS support (compatibility)

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
- Minimize times client informs server of changes

Oplocks

- Server grants opportunistic locks (oplocks) to client
  - Oplock tells client how/if it may cache data
  - Similar to DFS tokens (but more limited)
- Client must request an oplock
  - oplock may be
    - Granted
    - Revoked
    - Changed by server

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 (one writer)

- Level 1 oplock is replaced with a Level 2 lock if another process tries to read the file
- Request this if expect others to read
- 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 another client opens the file for writing

Batch oplock

- Client can keep file open on server even if a local process that was using it has closed the file
  - Exclusive R/W open lock + data lock + metadata lock
- Client requests batch oplock if it expects programs may behave in a way that generates a lot of traffic (e.g. accessing the same files over and over)
  - Designed for Windows batch files
- Batch oplock revoked if another client opens the file

Filter oplock

- Open file for read or write
- Allow clients with filter oplock to be suspended while another process preempted file access.
  - E.g., 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

- All requests must be sent to the server
- Can work from cache only if byte range was locked by client

Naming

- Multiple naming formats supported:
  - N:\junk.doc
  - \\myserver\users\paul\junk.doc
  - file://grumpy.pk.org/users/paul/junk.doc

Microsoft Dfs

- "Distributed File System"
  - Provides a logical view of files & directories
- Each computer hosts volumes
  - \\servername\dfsname
  - Each Dfs tree has one root volume and one level of leaf volumes.
- A volume can consist of multiple shares
  - Alternate path: load balancing (read-only)
  - Similar to Sun's automounter
- SMB + ability to mount server shares on other server shares

Redirection

- A share can be replicated (read-only) or moved through Microsoft's Dfs
- Client opens old location:
  - Receives STATUS_DFS_PATH_NOT_COVERED
  - Client requests referral:
    TRANS2_DFS_GET_REFERRAL
  - Server replies with new server

CIFS Summary

- Proposed standard has not yet fully materialized
  - Future direction uncertain
- Oplocks mechanism supported in base OS: Windows NT, 2000, XP
- Oplocks offer flexible control for distributed consistency