NFS & AFS

"Good judgment comes from experience…
Experience comes from bad judgment."
- attributed to many

Dave Eckhardt
de0u@andrew.cmu.edu

Bruce Maggs
user 1915@cs.cmu.edu

Synchronization

- Who runs shell? Passes test suite?
  - Who will be around Friday at midnight?
- Today
  - NFS, AFS
  - Partially covered by textbook: 11.9, 17.6
  - Chapter 17 is short, why not just read it?

Outline

- Why remote file systems?

- VFS interception

- NFS vs. AFS

  – Architectural assumptions & goals

  – Namespace

  – Authentication, access control

  – I/O flow

  – Rough edges

3

Why?

- Why remote file systems?

- Lots of "access data everywhere" technologies

  – Laptop

  – Multi-gigabyte flash-memory keychain USB devices

  – 4G Hitachi MicroDrive fits in a CompactFlash slot

  – iPod

- Are remote file systems dinosaurs?

Remote File System Benefits

- Reliability
  - Not many people carry multiple copies of data
  - Multiple copies **with you** aren't much protection
  - Backups are nice
    - Machine rooms are nice
      - Temperature-controlled, humidity-controlled
      - Fire-suppressed
    - Time travel is nice too
- Sharing
  - Allows multiple users to access data
  - May provide authentication mechanism

5

Remote File System Benefits

- Scalability
  - Large disks are cheaper
- Locality of reference
  - You don't use every file every day...
  - Why carry everything in expensive portable storage?
- Auditability
  - Easier to know who said what when with central storage...

What *Is* A Remote File System?

- OS-centric view
  - Something that supports file-system system calls "for us"
- Other possible views
  - RFS/DFS architect, for example
  - Mostly out of scope for this class
- Compared today
  - Sun Microsystems NFS
  - CMU/IBM/Transarc/IBM/open-source AFS

VFS interception

- VFS provides "pluggable" file systems

- Standard flow of remote access

  - User process calls read()

  - Kernel dispatches to VOP_READ() in some VFS

  - nfs_read()

    - check local cache
    - send RPC to remote NFS server
    - put process to sleep

VFS interception

- Standard flow of remote access (continued)
  - server interaction handled by kernel process
    - retransmit if necessary
    - convert RPC response to file system buffer
    - store in local cache
    - wake up user process
  - nfs_read()
    - copy bytes to user memory

NFS Assumptions, goals

- Workgroup file system

  – Small number of clients

  – Very small number of servers

- Single administrative domain

  – All machines agree on "set of users"

    - ...which users are in which groups

  – Client machines run mostly-trusted OS

    - "User #37 says read(...)"

NFS Assumptions, goals

- "Stateless" file server

  – Of course files are "state", but...

  – Server *exports* files without creating extra state

    - No list of "who has this file open"
    - No "pending transactions" across crash

  – Result: crash recovery "fast", protocol "simple"

- Some inherently "stateful" operations

  – File locking

  – Handled by separate service outside of NFS

AFS Assumptions, goals

- Global distributed file system
  - *Uncountable* clients, servers
  - "One AFS", like "one Internet"
    - Why would you want more than one?
- Multiple administrative domains
  - username*@cellname*
  - de0u@andrew.cmu.edu
  - davide@cs.cmu.edu

AFS Assumptions, goals

- Client machines are un-trusted
  - Must *prove* they act for a specific user
    - Secure RPC layer
  - Anonymous "system:anyuser"
- Client machines have disks (!!)
  - Can cache whole files over long periods
- Write/write and write/read sharing are rare
  - Most files updated by one user, on one machine

AFS Assumptions, goals

- Support *many* clients
    - 1000 machines could cache a single file
    - Some local, some (very) remote

NFS Namespace

- Constructed by client-side file system mounts

  - mount server1:/usr/local /usr/local

- Group of clients **_can achieve_** common namespace

  - Every machine executes same mount sequence at boot

  - If system administrators are diligent

NFS Namespace

- "Auto-mount" process based on maps
  - /home/dae means server1:/home/dae
  - /home/owens means server2:/home/owens

NFS Security

- Client machine presents credentials
  - user #, list of group #s – from Unix process
- Server accepts or rejects credentials
  - "root squashing"
    - map uid 0 to uid -1 unless client on special machine list
- Kernel process on server "adopts" credentials
  - Sets user #, group vector based on RPC
  - Makes system call (e.g., read()) with those credentials

AFS Namespace

- Assumed-global list of AFS cells

- Everybody sees same files in each cell

  – Multiple servers inside cell invisible to user

- Group of clients **_can achieve_** private namespace

  – Use custom cell database

AFS Security

- Client machine presents Kerberos ticket
  - Allows arbitrary binding of (machine,user) to (realm,principal)
    - davide on a cs.cmu.edu machine can be de0u@andrew.cmu.edu
    - iff the password is known!
- Server checks against *access control list*

AFS ACLs

- Apply to directory, not to file

- Format

  – de0u rlidwka

  – davide@cs.cmu.edu rl

  – de0u:friends rl

- Negative rights

  – Disallow "joe rl" even though joe is in de0u:friends

AFS ACLs

- AFS ACL semantics are not Unix semantics
  - Some parts obeyed in a vague way
    - Cache manager chesks for files being executable, writable
  - Many differences
    - Inherent/good: can name people in different administrative domains
    - "Just different"
      - ACLs are per-directory, not per-file
      - Different privileges: create, remove, lock
  - Not exactly Unix / not tied to Unix

21

NFS protocol architecture

- root@client executes mount RPC
  - returns "file handle" for root of remote file system
- RPC for each pathname component
  - /usr/local/lib/emacs/foo.el in /usr/local file system
    - h = lookup(root-handle, "lib")
    - h = lookup(h, "emacs")
    - h = lookup(h, "foo.el")
  - Allows disagreement over pathname syntax
    - Look, Ma, no "/"!

NFS protocol architecture

- I/O RPCs are *idempotent*

  - multiple repetitions have same effect as one

  - lookup(h, "emacs")

  - read(file-handle, offset, length)

  - write(file-handle, offset, buffer)

- RPCs do not create server-memory state

  - no open()/close() RPC

  - write() succeeds (to disk) or fails before RPC completes

NFS file handles

- Goals
  - Reasonable size
  - Quickly map to file on server
  - "Capability"
    - Hard to forge, so possession serves as "proof"
- Implementation (inode #, inode generation #)
  - inode # - small, fast for server to map onto data
  - "inode generation #" - must match value stored in inode
    - "unguessably random" number chosen in create()

NFS Directory Operations

- Primary goal
  - Insulate clients from server directory format
- Approach
  - readdir(dir-handle, cookie, nbytes) returns list
    - name, inode # (for display by ls -l), cookie

AFS protocol architecture

- *Volume* = miniature file system
    - One user's files, project source tree, ...
    - Unit of disk quota administration, backup
    - *Mount points* are pointers to other volumes
- Client machine has Cell-Server Database
    - /afs/andrew.cmu.edu is a *cell*
    - *protection server* handles authentication
    - *volume location server* maps volumes to *file servers*

AFS protocol architecture

- Volume location is *dynamic*

  – Moved between servers transparently to user

- Volumes may have multiple *replicas*

  – Increase throughput, reliability

  – Restricted to "read-only" volumes

    - /usr/local/bin

    - /afs/andrew.cmu.edu/usr

27

AFS Callbacks

- Observations
  - Client disks can cache files indefinitely
    - Even across reboots
  - Many files nearly read-only
    - Contacting server on each open() is wasteful
- Server issues *callback promise*
  - If this file changes in 15 minutes, I will tell you
    - *callback break* message
  - 15 minutes of free open(), read() for that client
    - More importantly, 15 minutes of silence for server

AFS file identifiers

- Volume number

  – Each file lives *in a volume*

  – Unlike NFS "server1's /usr0"

- File number

  – inode # (as NFS)

- "Uniquifier"

  – allows inodes to be re-used

  – Similar to NFS file handle inode generation #s

29

AFS Directory Operations

- Primary goal
  - Don't overload servers!

- Approach
  - Server stores directory as hash table on disk
  - Client fetches whole directory as if a file
  - *Client* parses hash table
    - Directory maps name to fid
  - Client caches directory (indefinitely, across reboots)
    - Server load reduced

AFS access pattern

`open("/afs/cs.cmu.edu/service/systypes")`

- VFS layer hands off "/afs" to AFS client module

- Client maps cs.cmu.edu to pt & vldb servers

- Client authenticates to pt server

- Client volume-locates root.cell volume

- Client fetches "/" directory

- Client fetches "service" directory

- Client fetches "systypes" file

AFS access pattern

`open("/afs/cs.cmu.edu/service/newCSDB")`

- VFS layer hands off "/afs" to AFS client module

- Client fetches "newCSDB" file

`open("/afs/cs.cmu.edu/service/systypes")`

- Assume
  - File is in cache
  - Server hasn't broken callback
  - Callback hasn't expired
- Client can read file with *no server interaction*

AFS access pattern

- Data transfer is by *chunks*

  – Minimally 64 KB

  – May be whole-file

- Write*back* cache

  – Opposite of NFS "every write is sacred"

  – Store chunk back to server

    - When cache overflows
    - On last user close()

AFS access pattern

- Is writeback crazy?
    - Write conflicts "assumed rare"
    - Who needs to see a half-written file?

NFS "rough edges"

- Locking
  - Inherently stateful
    - lock must *persist across client calls*
      - lock(), read(), write(), unlock()
  - "Separate service"
    - Handled by same server
    - Horrible things happen on server crash
    - Horrible things happen on client crash

NFS "rough edges"

- Some operations not really idempotent
  - unlink(file) returns "ok" *once*, then "no such file"
  - server caches "a few" client requests
- Cacheing
  - No real consistency guarantees
  - Clients typically cache attributes, data "for a while"
  - No way to know when they're wrong

NFS "rough edges"

- Large NFS installations are brittle
  - Everybody must agree on *many* mount points
  - Hard to load-balance files among servers
    - No volumes
    - No atomic moves
- Cross-realm NFS access basically nonexistent
  - No good way to map uid#47 from an unknown host

AFS "rough edges"

- Locking

  – Server refuses to keep a waiting-client list

  – Client cache manager refuses to poll server

  – User program must invent polling strategy

- Chunk-based I/O

  – No real consistency guarantees

  – close() failures surprising

AFS "rough edges"

- ACLs apply to *directories*
  - "Makes sense" if files will inherit from directories
    - Not always true
  - Confuses users
- Directories inherit ACLs
  - Easy to expose a whole tree accidentally
  - What else to do?
    - No good solution known
    - DFS horror

AFS "rough edges"

- Small AFS installations are punitive
  - Step 1: Install Kerberos
    - 2-3 servers
    - Inside locked boxes!
  - Step 2: Install ~4 AFS servers (2 data, 2 pt/vldb)
  - Step 3: Explain Kerberos to your users
    - Ticket expiration!
  - Step 4: Explain ACLs to your users

Summary - NFS

- Workgroup network file service

- Any Unix machine can be a server (easily)

- Machines can be both client & server

  – My files on my disk, your files on your disk

  – Everybody in group can access all files

- *Serious* trust, scaling problems

- "Stateless file server" model only partial success

41

Summary – AFS

- Worldwide file system

- Good security, scaling

- Global namespace

- "Professional" server infrastructure per cell

  – Don't try this at home

  – Only ~190 AFS cells (2005-11, also 2003-02)

    - 8 are cmu.edu, ~15 are in Pittsburgh

- "No write conflict" model only partial success

Further Reading

- NFS
  - RFC 1094 for v2 (3/1989)
  - RFC 1813 for v3 (6/1995)
  - RFC 3530 for v4 (4/2003)

Further Reading

- AFS

  - "The ITC Distributed File System: Principles and Design", Proceedings of the 10th ACM Symposium on Operating System Principles, Dec. 1985, pp. 35-50.

  - "Scale and Performance in a Distributed File System", ACM Transactions on Computer Systems, Vol. 6, No. 1, Feb. 1988, pp. 51-81.

  - IBM AFS User Guide, version 36

  - http://www.cs.cmu.edu/~help/afs/index.html