



Intelligent
Systems
Lab

Administration of Operating Systems

NFS
Chapter 22

November 24, 2011

Network File System



- First developed in 1980s
 - by Sun Microsystems
- Client-server architecture
 - export part of filesystem hierarchy
 - mount remote filesystems
- Uses *remote procedure calls*
 - inter-process communication mechanism
 - hiding network intricacies from applications
- Presents standard UNIX filesystem interface
- Transparent for both users and applications

Distributed Filesystems



Intelligent
Systems
Lab

- Access to remote files
 - list, read, write, delete, mkdir, ...
- Proper handling of concurrency
 - varying guarantees about locking
 - who “wins” with concurrent writes
 - read-write race conditions
- Handle dropped connections gracefully
 - most applications expect disks to be reliable
- Support for data replication and local caching
- Security
- ...
- Complexity vs features vs efficiency vs reliability

Service Types



- Upload/Download model
 - copy file from server to client
 - operate on file locally
 - copy file from client to server
 - simple but wasteful and inconsistent
- Remote access model
 - file service provides functional interface
 - create, delete, read bytes, write bytes, ...
 - clients only get necessary data
 - server has a coherent view of the filesystem
 - network problems can be tricky to handle
 - proper implementation can be difficult
 - excessive expectations on application side

NFS Design Goals



- Single machine can be both a client and server
 - including circular dependencies
- Support diskless workstations
 - devices in /dev hierarchy
- Support heterogeneous networks
 - hardware, operating system, filesystem, ...
- User and application transparency
 - remote data accessed as local files
 - support all file system calls
- Recovery from failure
 - stateless, client retries
 - UDP

Design Goals



- High performance
 - client-side caching
 - server-side read-ahead
- Limited consistency guarantees
- No data migration
 - resources must be re-mounted if moved
- No data replication
- No file access semantics
 - file locking
 - no open/close reference counting
 - very basic security model

NFS Architecture

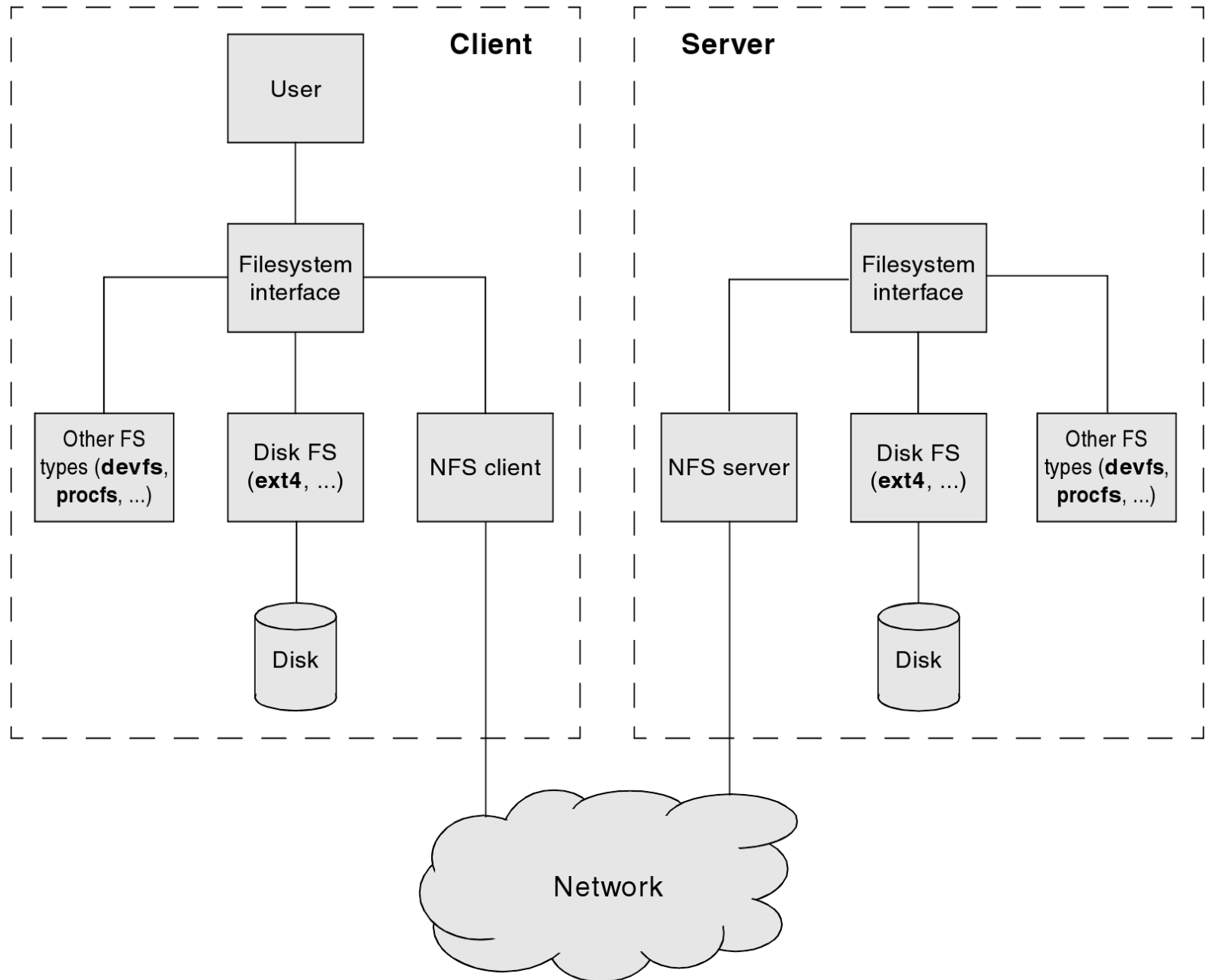


Figure 22-1 Flow of data in a typical NFS client/server setup

Installation



● Client

- `sudo apt-get install nfs-common`
- `sudo mount slawek:/home /home`
- `sudo mount -o ro,nosuid slawek:/apps /apps`
- `sudo vi /etc/fstab`

● Server

- `sudo apt-get install nfs-kernel-server`
- `sudo vi /etc/exports`
- `/home slawek(rw,sync,no_subtree_check)`
- `/home 1.2.3.4(ro,subtree_check,root_squash)`
- `sudo /etc/init.d/nfs-kernel-server restart`

NFS Protocol



- Operated over UDP
 - slightly faster than TCP
 - designed for Ethernet LAN
- Modern versions use TCP
 - better suited for in WAN setting
 - more efficient with heavy traffic
- Initial implementation was fully stateless
 - features such as file locking were implemented in higher-level protocols
 - separate, stateful lock manager
- Client and server crashes are relatively harmless
 - do not disturb the other side

NFS Server



- Defines a *virtual filesystem*
 - does not manage local disk layout
- Uses RPC
 - programmers can ignore network protocols
 - kernel sends requests to the server
 - awaits response
- Early versions required *write-through* semantics
 - later introduced commit operation
- Server returns file attributes with each request
 - saves extra requests
 - additional data transfer is usually negligible
- Trusted-host security paradigm

Mounting Protocol



- Client sends pathname to the server
 - requests permission to access contents
- Server returns file handle
 - device #
 - inode #
 - instance #
- File handle is persistent
 - calculated based on target inode
- Client creates in-memory *vnode* at mount point
 - points to *inodes* for local files
 - points to *rnode* for remote files
 - stores state on client

File Access Protocol



- First, perform a lookup RPC
 - returns file handle and attributes
 - no information is stored on server
 - a lot different from local open
- Handle is a parameter in other access functions
 - NFS version 3 has 22 different functions
- Cache validation
 - save timestamp of a file
 - compare with server on every data exchange
 - invalidate cached data if remote is more recent
 - *close-to-open* consistency model
 - good enough for most applications

File Locking



- Clients request lock from server
- Server informs clients about other lock requests
- Locking is lease-based
 - clients continually renew locks
- Loss of contact with server abandons locks
- User-level lock manager
- Monitored locks
 - status monitor monitors clients with locks
 - informs lock manager if host inaccessible
- If server crashes
 - status monitor recreates locks on recovery
- If client crashes: all locks from client are freed

Client Caching



- NFS Clients are allowed to cache copies of remote files for subsequent accesses
- Supports *close-to-open* cache consistency
- When client A closes a file, its contents are synchronised with the server
 - timestamp is changed
- When client B opens the file, it checks that local timestamp agrees with server timestamp
 - if not, it discards local copy
- Contacting server is always necessary when clients opens a remote file
 - even if data is in cache

System Calls



- A way for applications to requests a service
 - from the kernel of operating system
 - process control, file management, device access, information access, communication
- Privilege escalation
 - user mode programs have limited permissions
 - implemented on hardware level
 - for example using interrupts
 - processor mode and context switching
- Library function wrappers
 - C library

Typical Usages



Intelligent
Systems
Lab

- Roaming home directories
 - user convenience
- Data sharing
 - user cooperation
- Application export
 - saves space and administration overhead
- Diskless system
 - hardware savings
 - netboot
- Dataless system
 - simplifies configuration and upgrading

NFS Tradeoffs



Intelligent
Systems
Lab

- NFS volume is managed by a single server
 - can lead to high load
 - simplifies coherency protocols
- Full POSIX system
 - “drops in” very easily
 - isn’t “great” for any specific purpose
- Good enough for typical usage
- A lot of other remote filesystems available
 - Remote File System
 - Samba
 - Andrew File System
 - Google File System
 - WebDAV

Early NFS Problems



- Data consistency
- Clock synchronisation
- “Open with append”
- Locking
- No reference count
- Equal UIDs assumed
- File permissions
- Weak security model
- No traffic encryption

NFS Version 4



- Usernames instead of UIDs
- Kerberos authentication
- Encrypted network traffic
- Better performance on WAN
- Meta-data separation
- Parallel data access
- Sessions
- Notifications
- ACL

Windows Client



Intelligent
Systems
Lab

- Windows Services for UNIX
 - Subsystem for UNIX-based Applications
 - extension of “Microsoft POSIX subsystem”
- Implementation of user-mode POSIX subsystem
 - not kernel emulation
- Header files and libraries
- No binary compatibility
 - another UNIX-like platform
- Over 300 utilities
- NFS client and server
- NIS server linked with Active Directory

Automounter



- Classical mount is cumbersome
 - boot-time can be excessive
 - painful to administer on a large scale
- Automounter
 - on-demand mounting
 - only when local directory is referenced
 - OS sends a message to each server
 - first reply wins
- Load balancing
- Circular references become less problematic
- Automatically unmount when not used

Automounter



- `sudo apt-get install autofs`
- `sudo vi /etc/auto.master`
 - `/home /etc/auto.home –timeout=6000`
 - `/mnt /etc/auto.mnt –timeout=60`
- `sudo vi /etc/auto.home`
 - `slawek -fstype=nfs slawek:/home/slawek`
- `sudo vi /etc/auto.mnt`
 - `cdrom -fstype=iso9660 :/dev/sdb1`
- `sudo /etc/init.d/autofs restart`



Intelligent
Systems
Lab

Questions?

