

CS161: Operating Systems

Matt Welsh
mdw@eecs.harvard.edu



Lecture 17: NFS
April 17, 2007

Today: NFS and RAID

NFS allows a system to access files over a network

- One of many distributed file systems
- Extremely successful and widely used
 - *You use NFS on all of your shared files on the FAS and EECS machines*

RAID: Redundant Arrays of Inexpensive Disks

- Idea: Make file storage more robust by replicating data across multiple disks
- Developed at Berkeley in late 80's
- Now a multibillion-dollar industry

NFS Challenges

Development of LANs made it really attractive to provide shared file systems to all machines on a network

- Login to any machine and see the same set of files
- Install software on a single server that all machines can run
- Let users collaborate on shared set of files (before CVS!)

Why might this be hard to do???

NFS Challenges

Development of LANs made it really attractive to provide shared file systems to all machines on a network

- Login to any machine and see the same set of files
- Install software on a single server that all machines can run
- Let users collaborate on shared set of files (before CVS!)

Why might this be hard to do???

- Clients and servers might be running different OS
- Clients and servers might be using different CPU architecture with differing byte ordering (*endianess*)
- Client or server might crash independently of each other
 - *Must be easy to recover from crashes*
- Potentially very large number of client machines on a network
- Different users might be trying to modify a shared file at the same time
- Transparency: Allow user programs to access remote files just like local files
 - *No special libraries, recompilation, etc.*

NFS Overview

NFS was developed by Sun Microsystems in the mid-80s

- Networked machines at the time were predominantly UNIX-based workstations
- Various vendors: Sun, DEC, IBM, etc.
- Different CPU architectures and OS implementations
 - *But, all used UNIX filesystem structure and semantics*

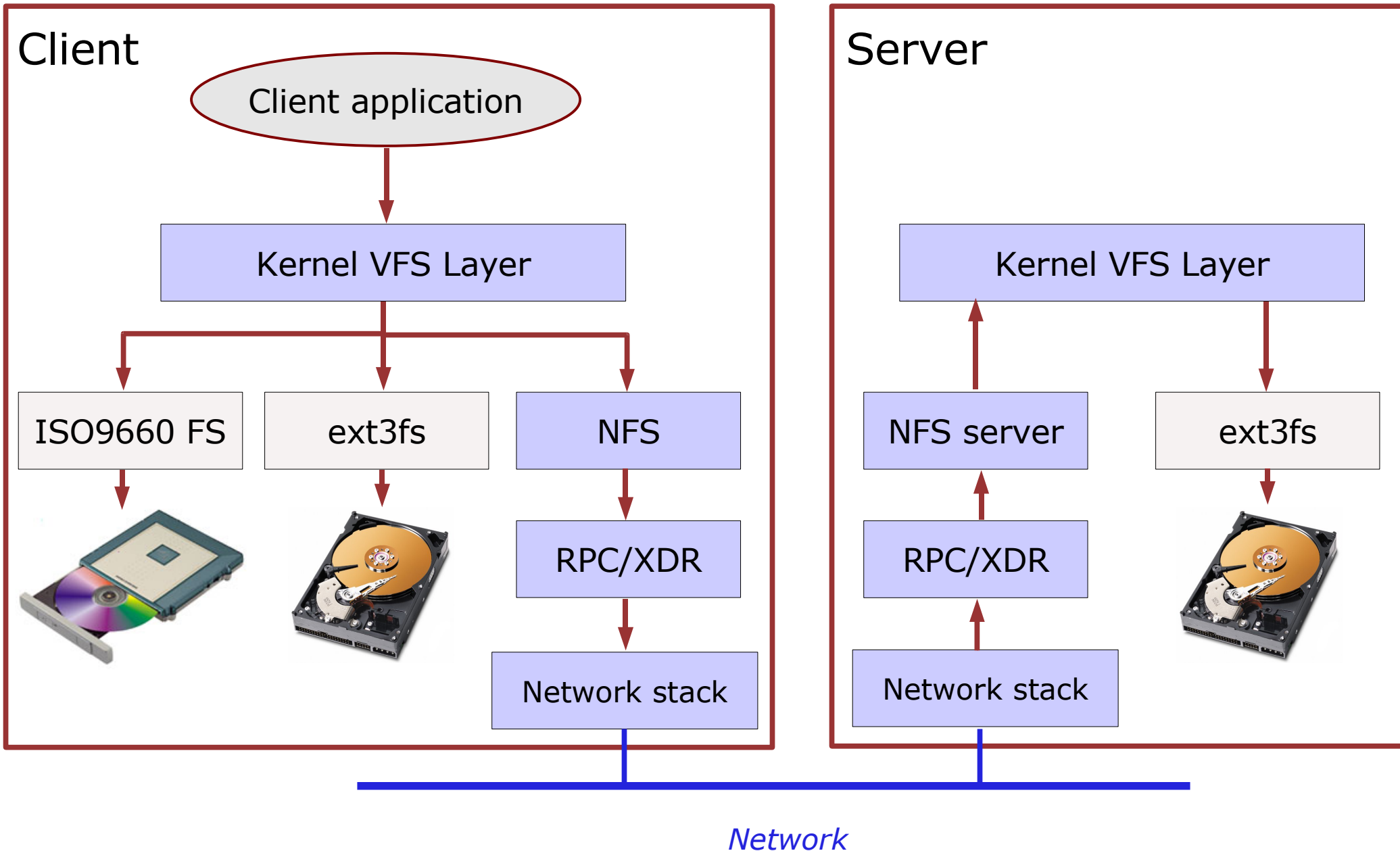
NFS is based on *Remote Procedure Call (RPC)*

- Allows a client machine to invoke a function on a server machine, over a network
- Client sends a message with the function arguments
- Server replies with a message with the return value.

External Data Representation (XDR) to represent data types

- Canonical network representation for ints, longs, byte arrays, etc.
- Clients and servers must translate parameters and return values of RPC calls into XDR before shipping on the network
 - *Otherwise, a little-endian machine and a big-endian machine would disagree on what is meant by the stream of bytes “fe 07 89 da” interpreted as an “int”*

NFS Design



Stateless Protocol

The NFS protocol is *stateless*

- The server maintains no information about individual clients!
- This means that NFS does not support any notion of “opening” or “closing” files
 - *Each client simply issues read and write requests specifying the file, offset in the file, and the requested size*

Advantages?

-
-
-

Disadvantages?

-
-
-

Stateless Protocol

The NFS protocol is *stateless*

- The server maintains no information about individual clients!
- This means that NFS does not support any notion of “opening” or “closing” files
 - *Each client simply issues read and write requests specifying the file, offset in the file, and the requested size*

Advantages:

- Server doesn't need to keep track of open/close status of files
- Server doesn't need to keep track of “file offset” for each client's open files
 - *Clients do this themselves*
- Server doesn't have to do anything to recover from a crash!
 - *Clients simply retry NFS operations until the server comes back up*

Disadvantages:

- Server doesn't keep track of concurrent access to same file
- Multiple clients might be modifying a file at the same time
 - *NFS does not provide any consistency guarantees!!!*
- However, there is a separate *locking protocol* – discussed later

NFS Protocol Overview

`mount()` returns filehandle for root of filesystem

- Actually a separate protocol from NFS...

`lookup(dir-handle, filename)` returns filehandle, attribs

- Returns unique file handle for a given file
- File handle used in subsequent read/write/etc. calls

`create(dir-handle, filename, attributes)` returns filehandle

`remove(dir-handle, filename)` returns status

`getattr(filehandle)` returns attribs

- Returns attributes of the file, e.g., permissions, owner, group ID, size, access time, last-modified time

`setattr(filehandle, attribs)` returns attribs

`read(filehandle, offset, size)` returns attribs, data

`write(filehandle, offset, count, data)` returns attribs

Example

What happens if we do “/bin/cat testfile” ?

```
140.247.60.36.031e 140.247.50.252.0801 U C3 cd9d450a 3
  lookup   fh 0013eef90000001b00000000 name "testfile" euid 813b egid 186c
  con = 146 len = 174

140.247.50.252.0801 140.247.60.36.031e U R3 cd9d450a 3
  lookup   OK fh 0013eef90000002a00000000 ftype 1 mode 180 nlink 1 uid 813b gid 186c
  size 1aca used 2000 rdev 0 rdev2 0 fsid 13eef9 fileid eef9002a
  atime 1082215906.000000 mtime 1082215906.000580 ctime 1082215906.000580 status=0
  pl = 192 con = 70 len = 262

140.247.60.36.031e 140.247.50.252.0801 U C3 ce9d450a 6
  read     fh 0013eef90000002a00000000 off 0 count 2000 euid 813b egid 186c

140.247.50.252.0801 140.247.60.36.031e U R3 ce9d450a 6
  read     OK ftype 1 mode 180 nlink 1 uid 813b gid 186c size 1aca
  used 2000 rdev 0 rdev2 0 fsid 13eef9 fileid eef9002a
  atime 1082215906.000000 mtime 1082215906.000580 ctime 1082215906.000580 count aca
```

NFS Caching

NFS clients are responsible for caching recently-accessed data

- Remember: the server is stateless!

The NFS protocol does not *require* that clients cache data ...

- But, it provides support allowing a range of client-side caching techniques

This is accomplished through the `getattr()` call

- Returns size, permissions, and last-modified time of file
- This can tell a client whether a file has changed since it last read it
 - *Read/write calls also return attributes so client can tell if object was modified since the last `getattr()` call*

How often does the client use `getattr()`?

NFS Caching

NFS clients are responsible for caching recently-accessed data

- Remember: the server is stateless!

The NFS protocol does not *require* that clients cache data ...

- But, it provides support allowing a range of client-side caching techniques

This is accomplished through the `getattr()` call

- Returns size, permissions, and last-modified time of file
- This can tell a client whether a file has changed since it last read it
 - *Read/write calls also return attributes so client can tell if object was modified since the last `getattr()` call*

How often does the client use `getattr()`?

- Whenever the file is accessed?
 - *Could lead to a lot of `getattr` calls!*
- Only if the file has not been accessed for some time?
 - *e.g., If the file has not been accessed in 30 sec?*
- Different OS's implement this differently!

NFS Locking

NFS does not prevent multiple clients from modifying a file simultaneously

- Clearly, this can be a Bad Thing for some applications...

Solution: Network Lock Manager (NLM) protocol

- Works alongside NFS to provide file locking
- NFS itself does not know anything about locks
 - *Clients have to use NLM “voluntarily” to avoid stomping on each other*
- NLM has to be *stateful*
 - *Why?*

NLM Protocol

NLM server has to keep track of locks held by clients

If the NLM server crashes...

- **All locks are released!**
- BUT ... clients can reestablish locks during a “grace period” after the server recovers
 - *No new locks are granted during the grace period*
 - *Server has to remember which locks were previously held by clients*

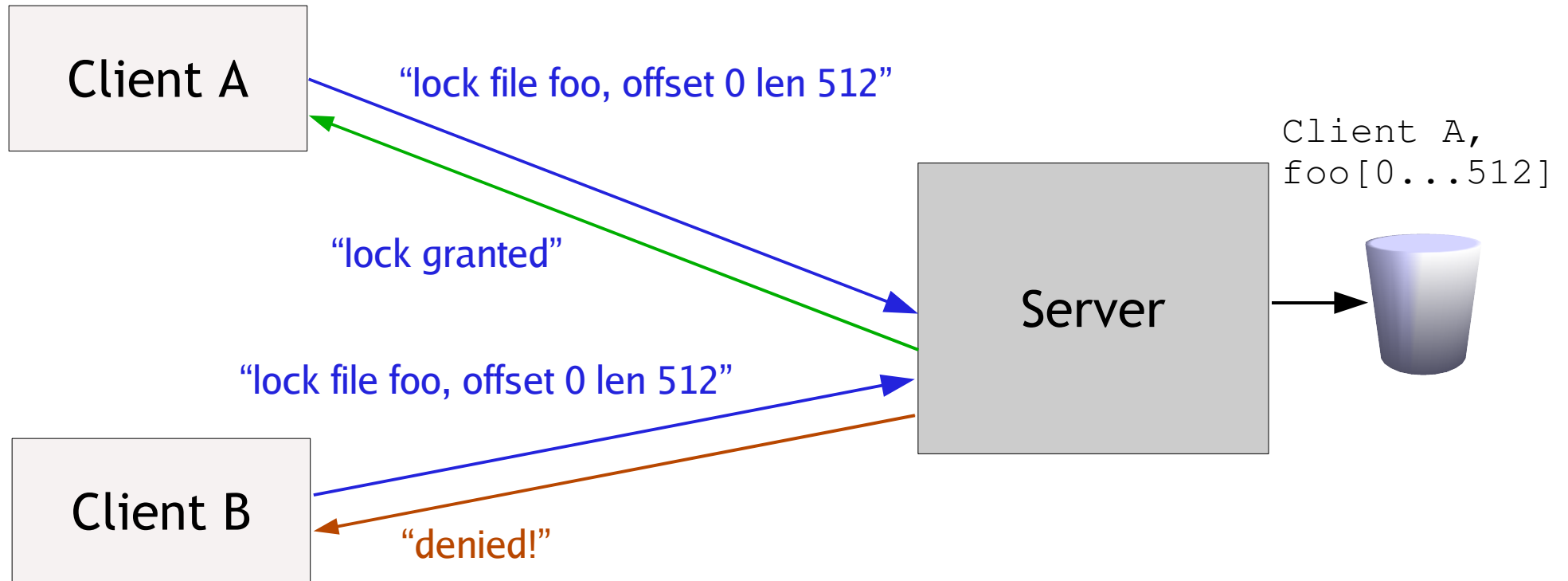
If an NLM client crashes...

- The server is notified when the client recovers and releases all of its locks
 - *What happens if a client crashes and does not come back up for a while?*

Servers and clients must be notified when they crash and recover

- This is done with the simple “Network Status Monitor” (NSM) protocol
- Essentially, send a notification to the other host when you reboot

NLM Example



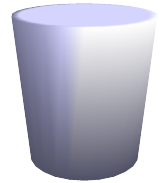
NLM Example

Client A

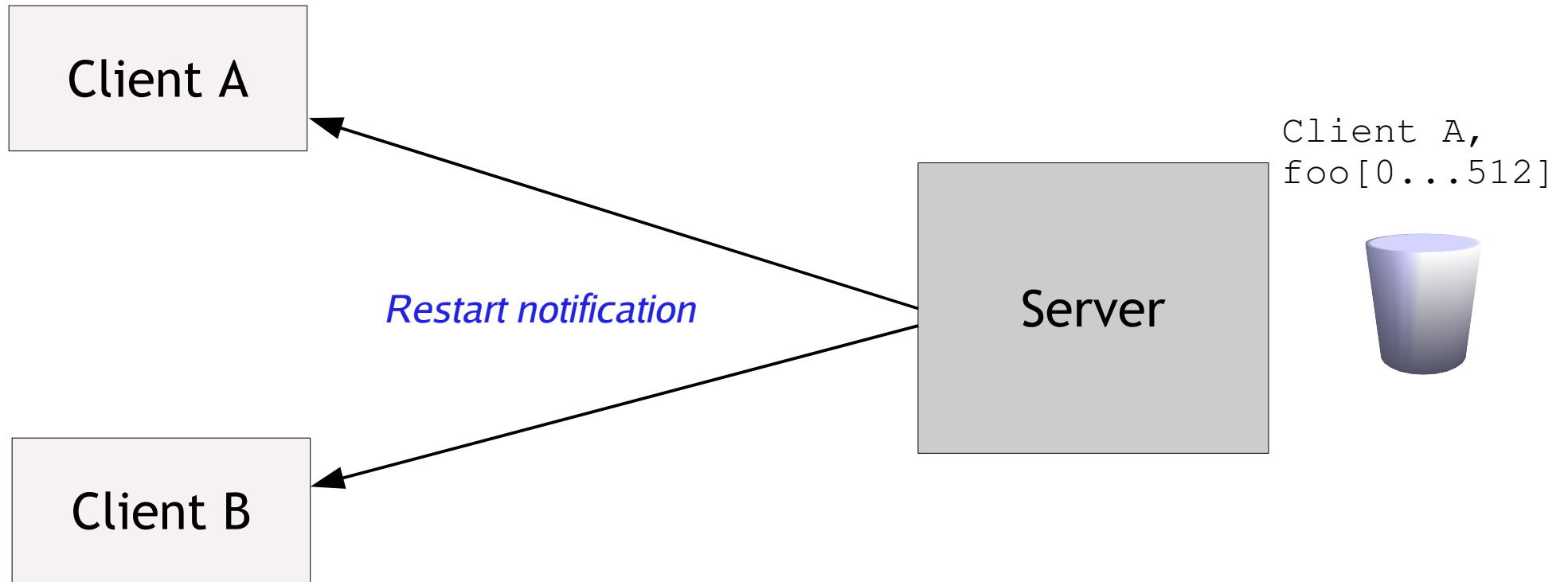
Client B



Client A,
foo[0...512]



NLM Example



NLM Example

