LECTURE

NFS

Dist FS : C C C S

(2) client App / client-side FS / net ⟷ net / server FS →

NFS : goal simple/fast server crash recovery
key: design of protocol (way that machines interact)
property (~~statelessness~~)

statelessness:
{ incorporate, into every request, all info needed to complete request }
(server shouldn't track things like which clients have open files, etc.)

Protocol:
key is file handle
(volume ID, inode #, gen #)
(why each?)

Examples:
#bytes read ← read (fh, offset, size)
#bytes written ← write ( )
0, -1 ← create (mkdir) (dfh, name)
fh of name ← lookup (dfh, name)
file attr ← getattr (fh)
(stat call, e.g., modified time)

Building FS from this

client calls:
open ("/a/b.txt")
(pretend this is root of NFS volume)

lookup(rootfh, "a") → (a's fh)
lookup (a's fh, "b.txt"); → (b.txt's fh)
access (b.txt fh, rwx) → ✓

client fs: map fd ⇒ this (b.txt's fh)
read (size);
read (b fh, 0, size);

How to handle failure?

key: idempotency
operation is idempotent if ...
use timeout/retry:
thus, client has unified response to lack of ack ⇒
RETRY

even when
C → write() → S →
S → X ack lost

Problem: Performance
server overloaded, client latency ↑
solution: client cache
⇒ new problem: cache consistency
for reads:
stale { read block X from file
keep in memory of client
for writes:
visibility { write new block Y to file
buffer in memory of client

to address visibility ⇒
flush on close ()
(client cache dirty writes)
to address stale cache ⇒
getattr : check if block has been changed before re use
but, still too many!
⇒ getattr cache w/ 3 second timeout

server side caching:
useful? buffering?
correct?