Funda : ✓

Techniques : → . Local Nodes

→ Distributed Systems.

→ NFS . (idempotency)

→ Leases

concurrency

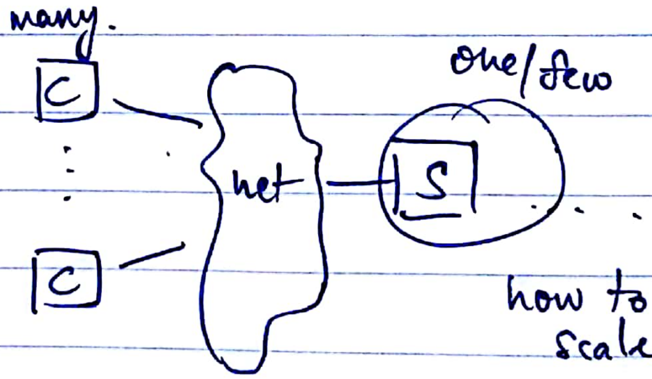
failure. ✓

- locks can't do the same
- includes time-lease.

NFS : "Distributed" in 1980's 1990's

⇓

Client, servers .



how to scale?

1) scale out (add servers)

happened later (challenging) → hard → increases complexity .

2) scale up (make server more powerfull).

↑CPU, ↑ memory.

easy → fewer interactions

Remzi's opinion.

**Main Goal :**

goals :-

=> transparency .

(performance ?

semantics ? ).

deleting open ~~and~~ files and writing to them .

=> simplify failures handling (server).

fault into a performance blip .

[client behaviour . -> just try again if anything bad happens]

(idempotency)

$$A = \frac{MTTF}{MTTF + \boxed{MTTR}}$$

↓

reduce to ↑ availability.

[unifies packet loss, server crash] .

based upon :

"statelessness".

why ? open files. handling

local -> no problem, entire system fails

c/s model -> partial failure

Protocol:

handle
( file ~~descriptor~~ )

↓

( vol #, inode #, gen # )

↑ which file sys?

↑ which file?

↑ useful why? (across file delete + reuse' inode #).

Ops:

→ read( fh, offset, size ).

⇒ data, error code.

→ write (fh, [offset], size, data)

local → OS takes care of offset of write.

→ lookup (parent fh, name).

⇒ fh of 'name' in parent dir.

→ getattr( fh )

→ stat() syscall about a file.

## from protocal to FS :

not a onetoone mapping .

=> fs API != DFS protocol.
↓
distributed

## APIs

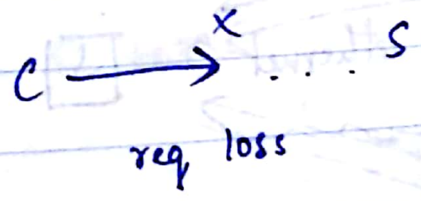
open ( ) ⟶ lookup

/x/y/z
fh1 .fh2..fh3.

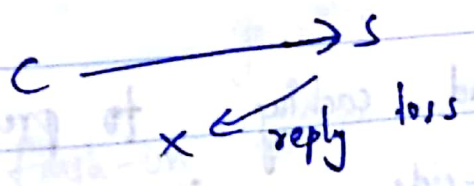
read ( ) ⟶ read (s).
write ( ) ⟶ write (s). } one or many .

close ( ) ⟶ . . . (N/A).
only client-side

## How to handle failure ?

1)


just retry

2)


retry should be okay .
(idempotency )

server got smarter . they (server) cached that they've done it and sends ACK .

3) C ⟶ S (crash)
×

Ans: <u>retry</u>.

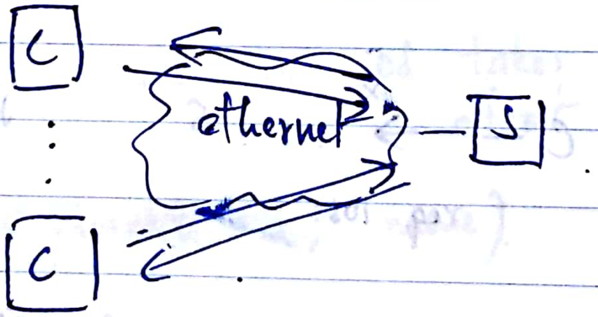
Be careful to retry:

```
→ while (1) {
      do:
        if success():
            break
  }.
```

too many retrys causing system to overload.

=) [<u>backoff</u> mechanisum needed required.]
↓
wait longer and longer after each retry.

**Performance problems:**



**NFS**

read caching to prevent server access everything.
[client-side caching].

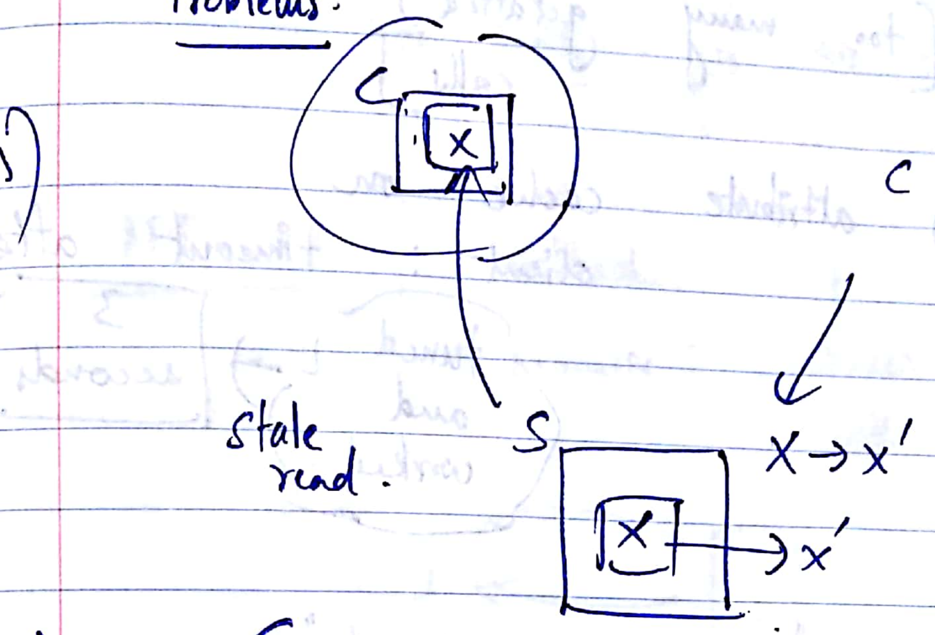
+ <u>write buffering</u>.

send bulk requests from client

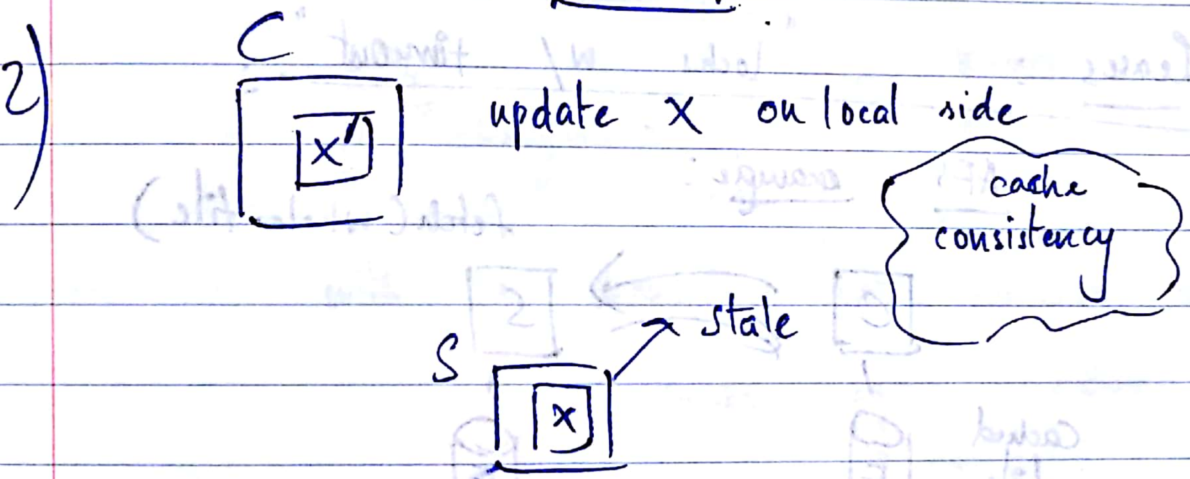
helpful? 1) one ~~[illegible]~~ chunk of requests.
2) redundant write or actians

**Problems:**

1)



2)



**Solutions:**

1) C before use of cached: ask if file changed? get attr ( ) → S. **Check before use.**

2) **flush-on-close:**

open
write
:
close → force update on close. to server

last problems :

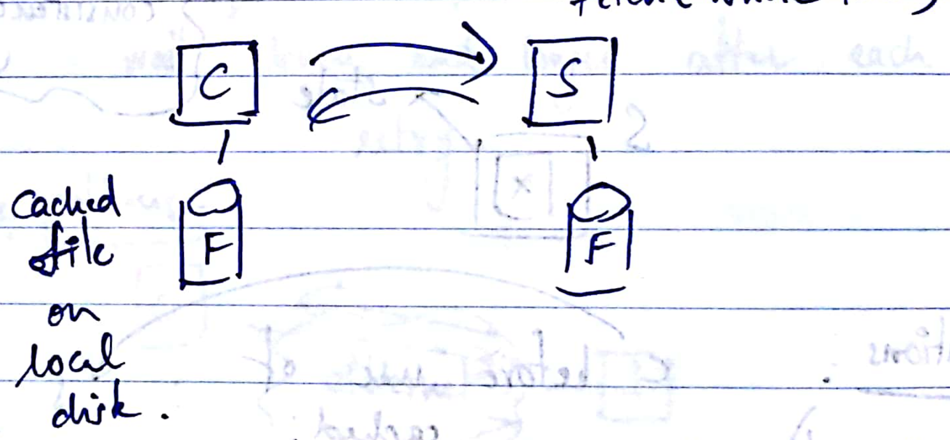
[too many getattr() calls].

=) attribute cache on client : timeout after

tuned and worked. → [3 seconds].

Leases : "locks w/ timeout" :

AFS example :



Reads & write on local.

on close :-, if file has changed flush to server

cache on mem or disk ?

↓ better

→ persistant (on crash)

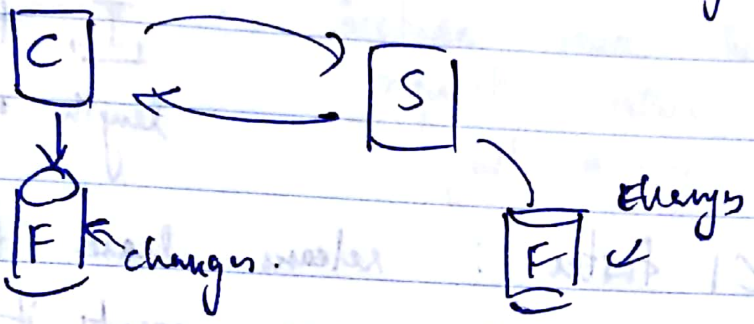
→ bigger.

AFS v1 : cache validation ?

Test Auth : is this file ~~cache~~ valid?

AFS v2 : callback.

promise : server notifies client with changes to file.



with ~~with~~ a cache : not stateless anymore.

server crash handling is different.

=> complex.

client crash => how to notify.

Solution :

Leases rights with timeout.

⇓

key to handling failure.

common :

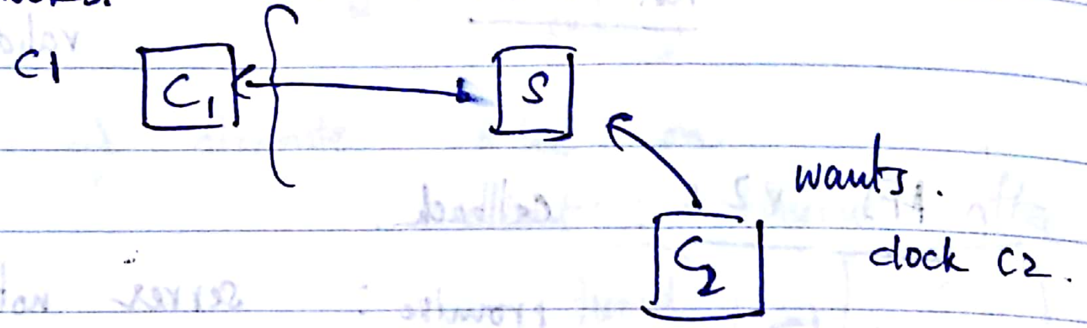
acquire → use it.

(maybe renew periodically).

release ( or just let timeout).

=) when lease safely grant to another client ?

clocks.

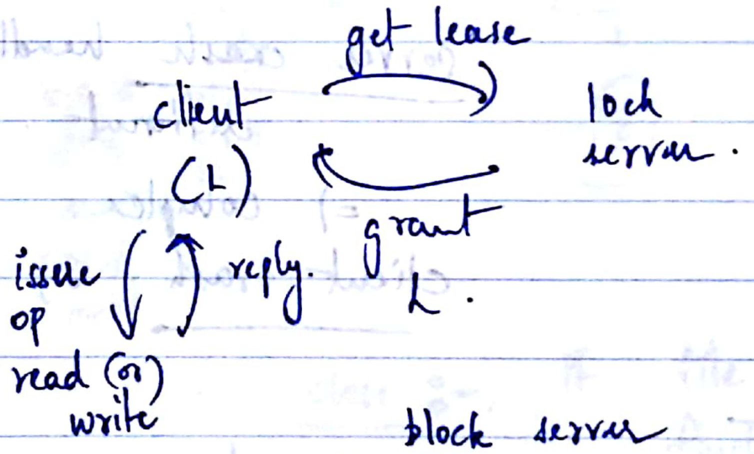


1) server waits, wait how long?

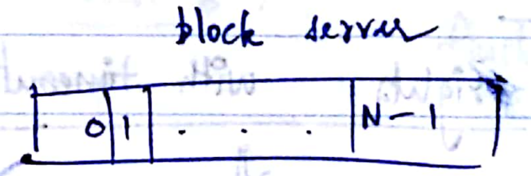
$\underbrace{T_1}_{\text{length of lease.}} + \underbrace{\Delta}_{\text{clock skew that we tolerate.}}$

Clock { C1 faster : releases lease faster

C1 slow : server grants it to C2 and C1 thinks it has lease.



lease : grants exclusive access ~~to~~ to block / range



leases to make sure ops ~~are~~ to block server is serial.

Problem : Race condition

client requests even after lease expires on server. and S grants lease to another client.

not serial anymore.

how to handle ?

2) fence : include lease info in request. rather than just relying on timing.