PERSISTENCE: NFS

Kai Mast
CS 537
Fall 2022

Agenda

- Today
  - Wrap up NFS
  - Code for P4
- Thursday: SSDs
- Next Week
  - No new material
  - Probably a review section

Can NFS Protocol Include Append?

```
fh = open(path);
read(fh, buf, size, offset);
write(fh, buf, size, offset);
append(fh, buf, size);
```

Problem with `append()`? RPC often has "at-least-once" semantics

-
- Implementing "exactly once" requires state on server, which we are trying to avoid

If RPC library replays messages, what happens when `append()` is retried on server?

-

IDEMPOTENT OPERATIONS

Solution: Design API so no that no harm is caused if we execute a function more than once

If f() is **idempotent**, then:

file		file		file		file
AAAA AAAA	`write`	**ABBA** AAAA	`write`	**ABBA** AAAA	`write`	**ABBA** AAAA

file		file		file		file
A	`append`	A**B**	`append`	A**BB**	`append`	A**BBB**

WHAT OPERATIONS ARE IDEMPOTENT?

Idempotent

- `write`
- any sort of read that doesn't change anything

Not idempotent

-

What about these?

- `mkdir`
- `create`

API STRATEGY 4: FILE HANDLES

Do not include append() in NFS protocol

```
fh = open(char *path);
read(fh, buf, size, offset);
write(fh, buf, size, offset);
append(fh, buf, size);
```

File Handle = <volume ID, inode #, generation #>

Can applications call append????

FINAL API STRATEGY 5: CLIENT LOGIC

Build normal UNIX API on client side **on top of** idempotent, RPC-based API

Clients maintain their own file descriptors

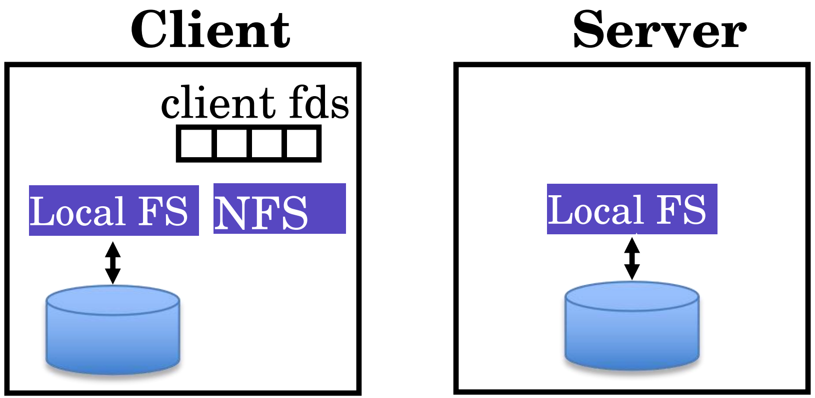
- Client `open()` creates a local fd object

Local fd object contains:

- file handle (returned by server)
- current offset (maintained by client)

On read/write:

- Client sends fh, offset, size to server
- Server extracts inode from fh



fd 5

```
fh=<fh42>
off=123
```

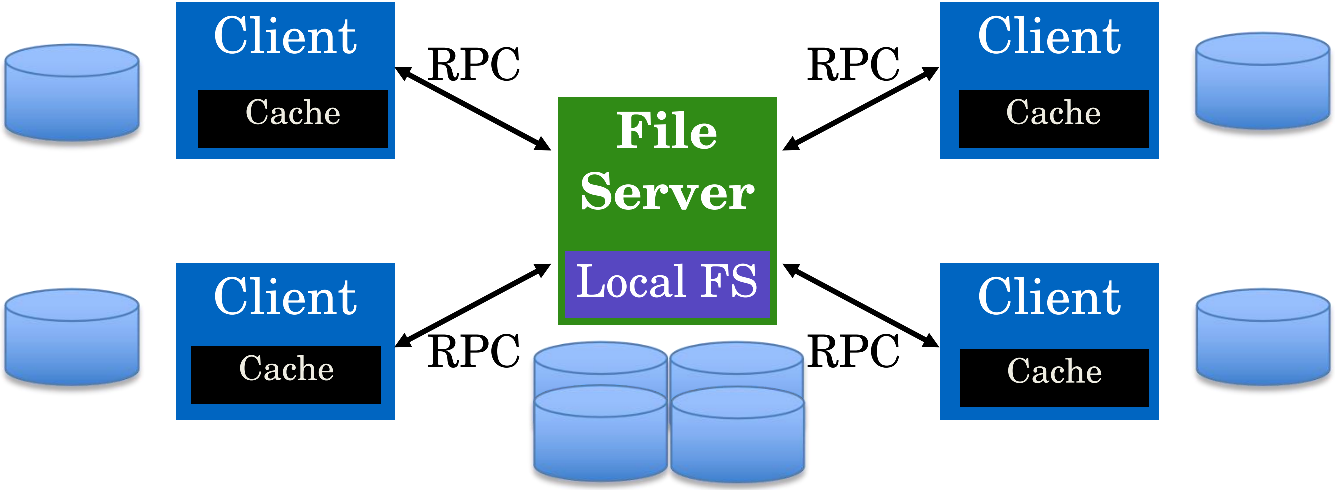
```
read(fd5, max)
↓
read(fh42, 123, max)
```

NFS OVERVIEW

1. ~~Architecture + Network API~~

2. Caching

NFS CACHING ARCHITECTURE

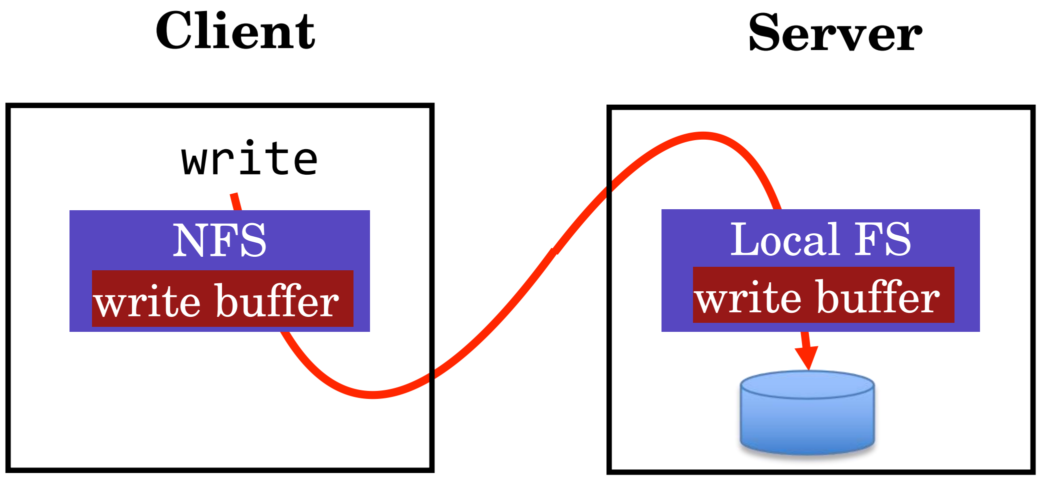


NFS can cache data in three places:

- server memory
- client disk or memory

How to make sure server and all client versions are in sync?

CACHE PROBLEM 1: SERVER MEMORY



NSF Server often buffers writes to improve performance
Server might acknowledge write before pushed to disk

What happens if server crashes?

SERVER MEMORY – LOST ON CRASH

client:

write A to 0

write B to 1

write C to 2

write X to 0

write Y to 1

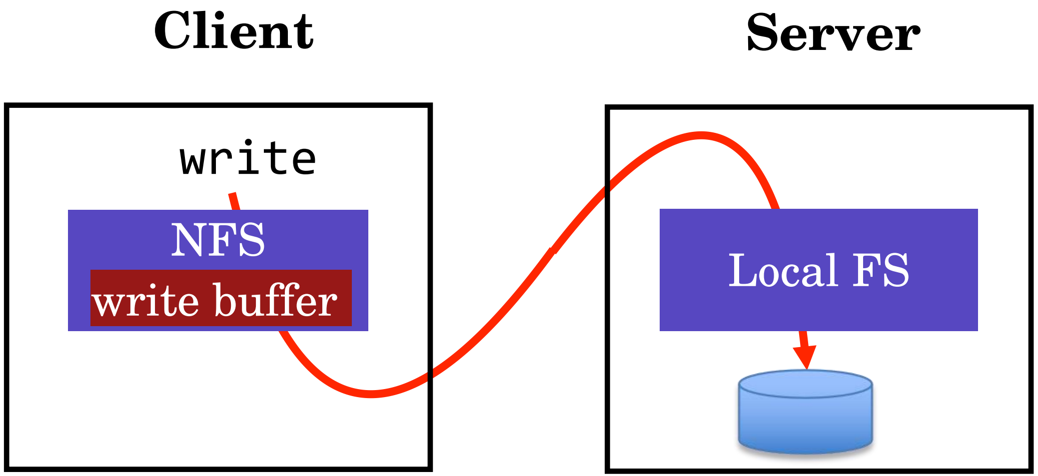
write Z to 2

	0	1	2
server memory:			
server disk:	X	B	Z

Problem: No write failed, but disk state doesn't match any point in time

What could have happened?

SERVER MEMORY: SOLUTIONS



Solution: Don't use server write buffer (persist data to disk before acknowledging write)

Problem: Slow!

CACHE PROBLEM 2 + 3: DISTRIBUTED CACHE

Clients must cache some data

- Too slow to always contact server; Server would become severe bottleneck

"Update Visibility" problem: Server doesn't have latest version

- Client 1 reads, Client 1 writes… What happens if process on Client 2 reads data?

When client buffers a write, how can server see update?

CACHE PROBLEM 2: UPDATE VISIBILITY

Possibilities

- After every write (too slow)
- Periodically after some interval (odd semantics)

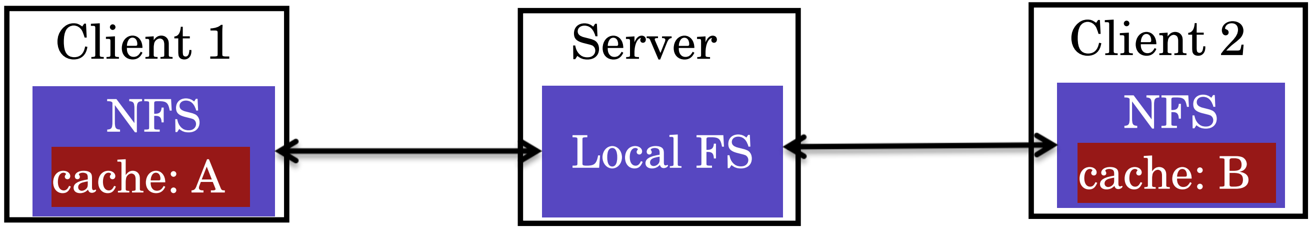
NFS solution

-
- Other times optionally too – e.g., when low on memory

Problems not solved by NFS:

- File flushes not atomic (one block of file at a time)
-

Cache Problem 3: Stale Cache



"Stale Cache" problem: Client 2 doesn't have latest version from server

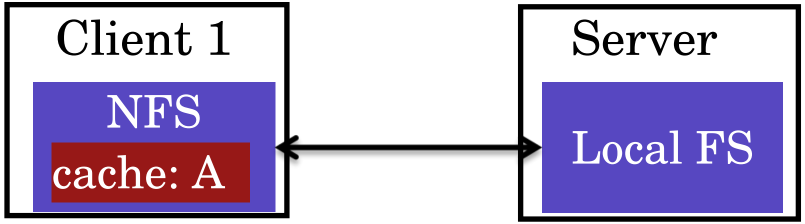
What happens if process on Client 2 reads data?

•

How can it get latest?

- One possible solution: If NFS server had **state**, could push update to relevant clients
- NFS stateless solution:

STAT CALL TO SERVER



Client cache records time when **data block** was fetched (t1)

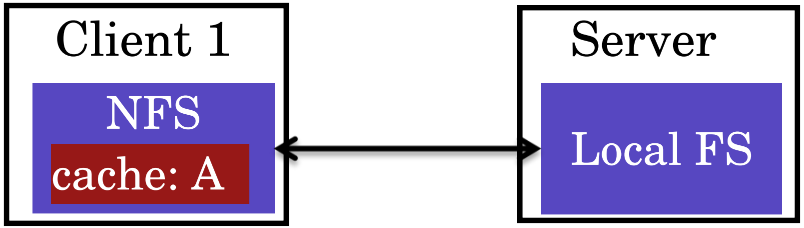
Before using data block, client sends file STAT request to server

- Gets last modified timestamp for this **file** (t2) (not block…)
- If file changed since block fetch timestamp (t2 > t1), then re-fetch data block

Measurements: NFS developers found server overloaded – limits number of clients

- Found stat accounted for 90% of server requests
- Why? Because clients frequently recheck cache

REDUCING STAT CALLS



Partial Solution: client caches result of stat (attribute cache)

What is the result?

Solution: Make stat cache entries expire after a given time (e.g., 3 seconds) (discard t2 at client 2)

What is the result?

NFS SUMMARY

NFS handles client and server crashes very well; robust APIs are often:

- **stateless**: servers don't remember clients or open files
- **idempotent**: repeating operations gives same results

Caching and write buffering is hard in distributed systems, especially with crashes

NFS Problems:

- Consistency model is odd; sees mix of updates within file
  (client may not see updates until 3 seconds after file is closed)
- Scalability limitations as more clients call stat() on server

PROJECT 4

(Demo)