NFS, REVIEW, SUMMARY

Shivaram Venkataraman

CS 537, Spring 2019

ADMINISTRIVIA

Project 4a, 4b grades out. Regrade requests by tomorrow

Final Exam:

- Everything up to NFS.
- May 8th at 2.45PM at Agr Hall 125

No discussion this week!

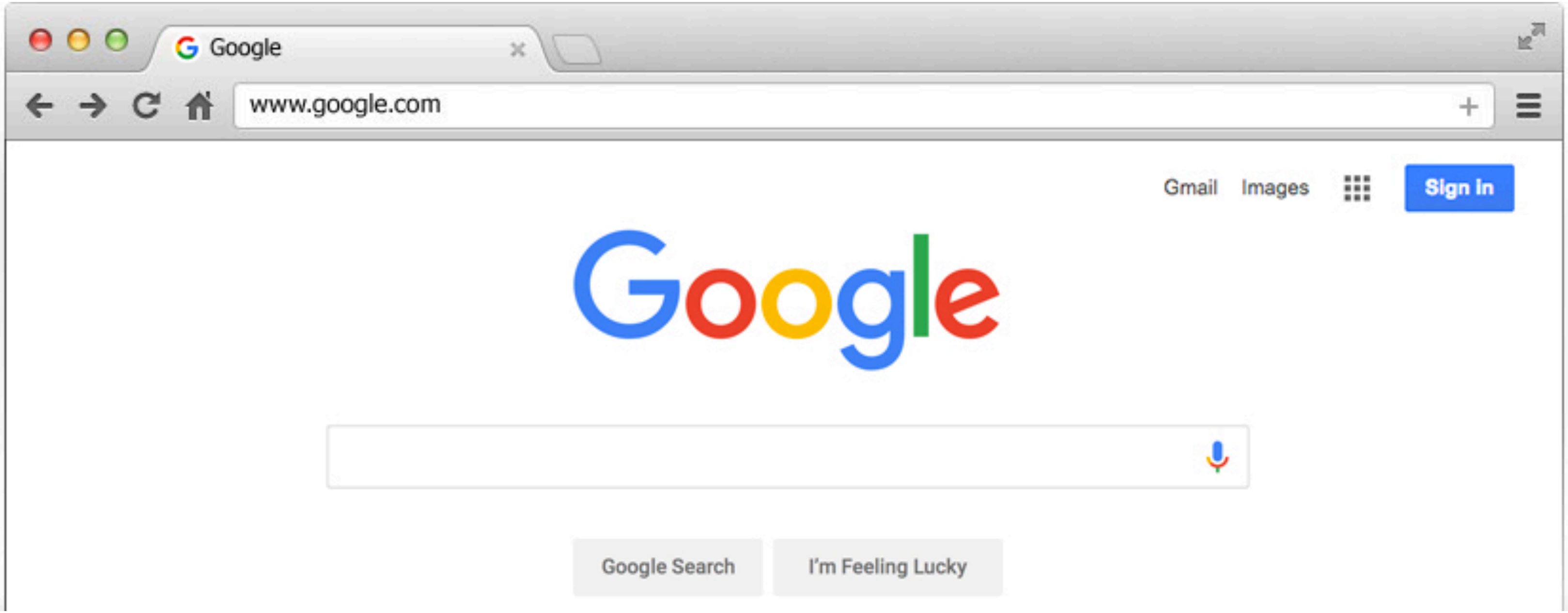
AGENDA / LEARNING OUTCOMES

What are consistency properties provided NFS?

What is the role of OS in context of cloud computing?

RECAP

DISTRIBUTED SYSTEMS



GOALS FOR DISTRIBUTED FILE SYSTEMS

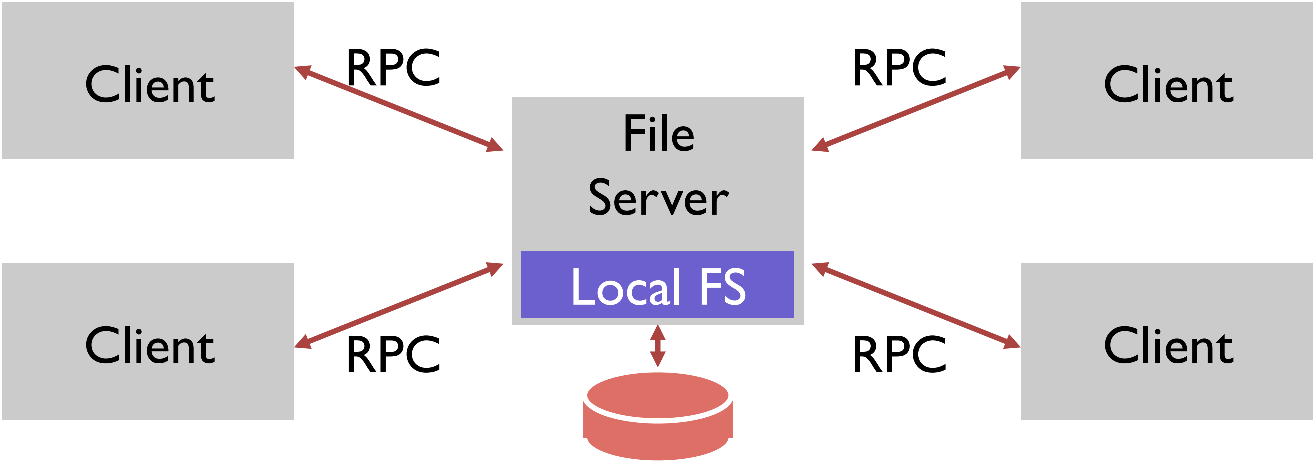
Transparent access

- can't tell accesses are over the network
- normal UNIX semantics

Fast + simple crash recovery: both clients and file server may crash

Reasonable performance?

NFS ARCHITECTURE



STRATEGY: FILE HANDLES

```
fh = open(char *path);
pread(fh, buf, size, offset);
pwrite(fh, buf, size, offset);
```

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

Opaque to client (client should not interpret internals)

```
NFSPROC_GETATTR
  expects: file handle
  returns: attributes
NFSPROC_SETATTR
  expects: file handle, attributes
  returns: nothing
NFSPROC_LOOKUP
  expects: directory file handle, name of file/directory to look up
  returns: file handle
NFSPROC_READ
  expects: file handle, offset, count
  returns: data, attributes
NFSPROC_WRITE
  expects: file handle, offset, count, data
  returns: attributes
NFSPROC_CREATE
  expects: directory file handle, name of file, attributes
  returns: nothing
NFSPROC_REMOVE
  expects: directory file handle, name of file to be removed
  returns: nothing
NFSPROC_MKDIR
  expects: directory file handle, name of directory, attributes
  returns: file handle
NFSPROC_RMDIR
  expects: directory file handle, name of directory to be removed
  returns: nothing
NFSPROC_READDIR
  expects: directory handle, count of bytes to read, cookie
  returns: directory entries, cookie (to get more entries)
```

CRASHES WITH IDEMPOTENT OPERATIONS

```
int fd = open("foo", O_RDONLY);
read(fd, buf, MAX);
write(fd, buf, MAX);
…
```

Server crash!

OVERVIEW

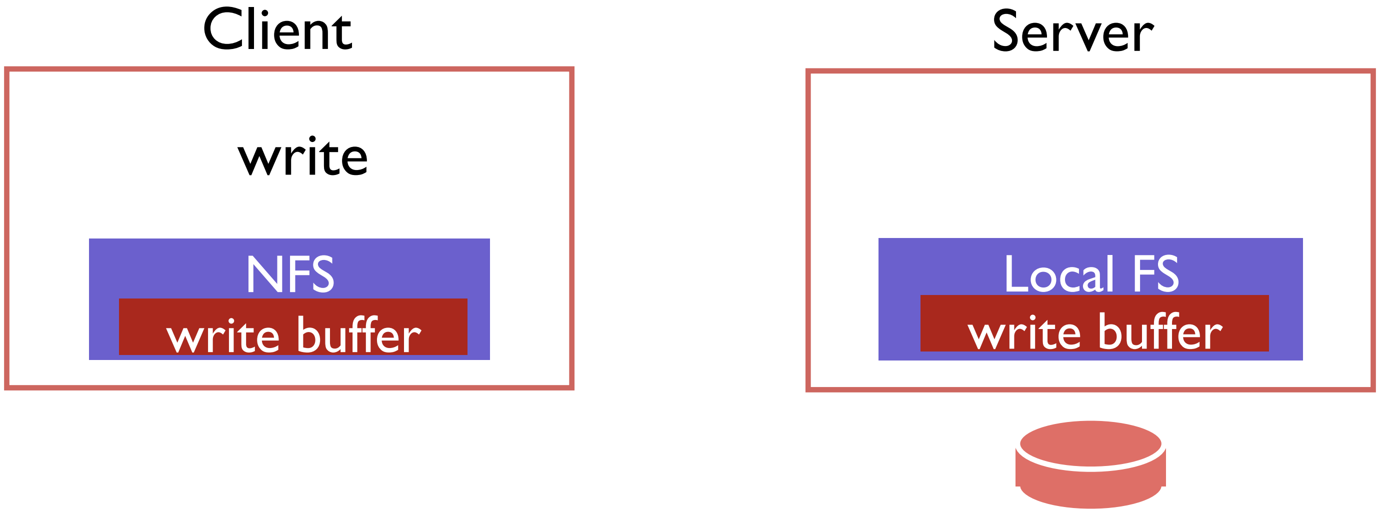
~~Architecture~~

~~Network API~~

Write Buffering

Cache

WRITE BUFFERS



Server acknowledges write before write is pushed to disk;
What happens if server crashes?

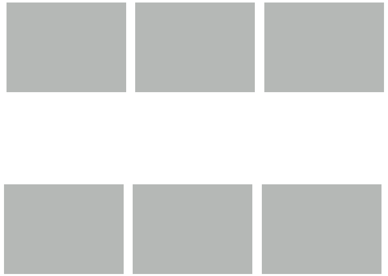
SERVER WRITE BUFFER LOST

```
client:

 write A to 0
 write B to 1
 write C to 2
```

server mem:

server disk:



server acknowledges write before write is pushed to disk

SERVER WRITE BUFFER LOST

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
```

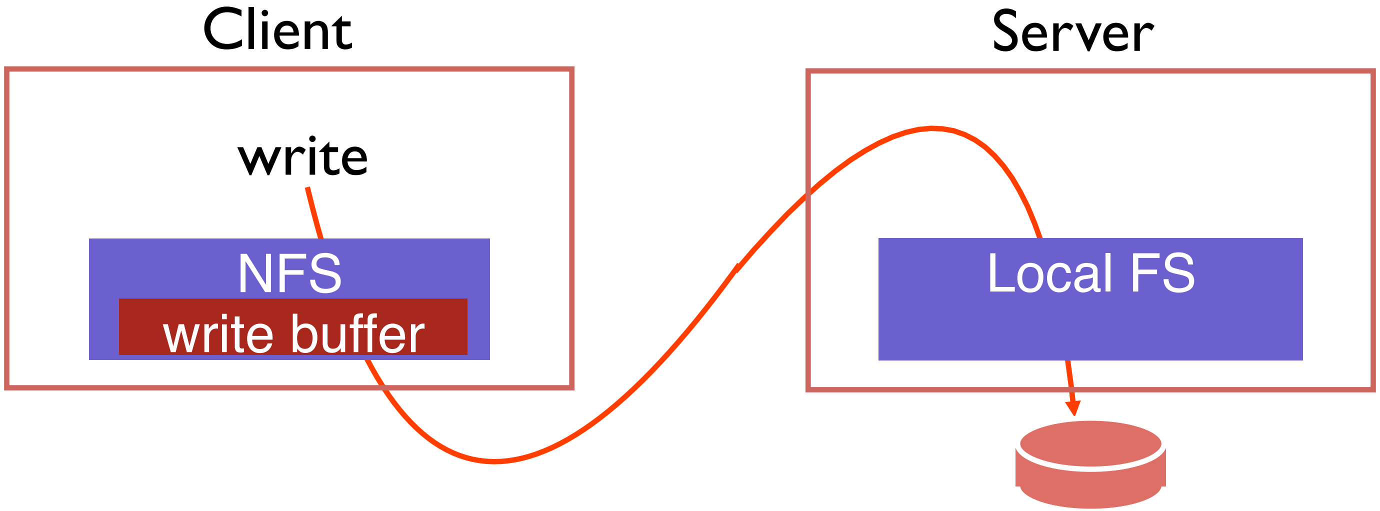
server mem: | | | Z |

server disk: | X | B | Z |

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

Solutions?

WRITE BUFFERS



Don't use server write buffer. Problem: Slow?

Use persistent write buffer (more expensive)

~~Architecture~~

~~Network API~~

~~Write Buffering~~

Cache

CACHE CONSISTENCY

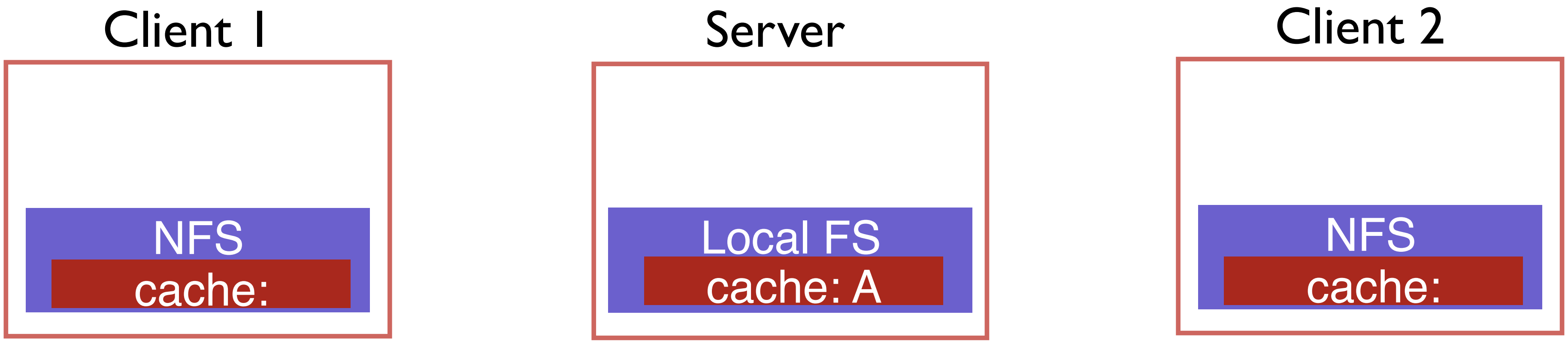
NFS can cache data in three places:

- server memory
- client disk
- client memory

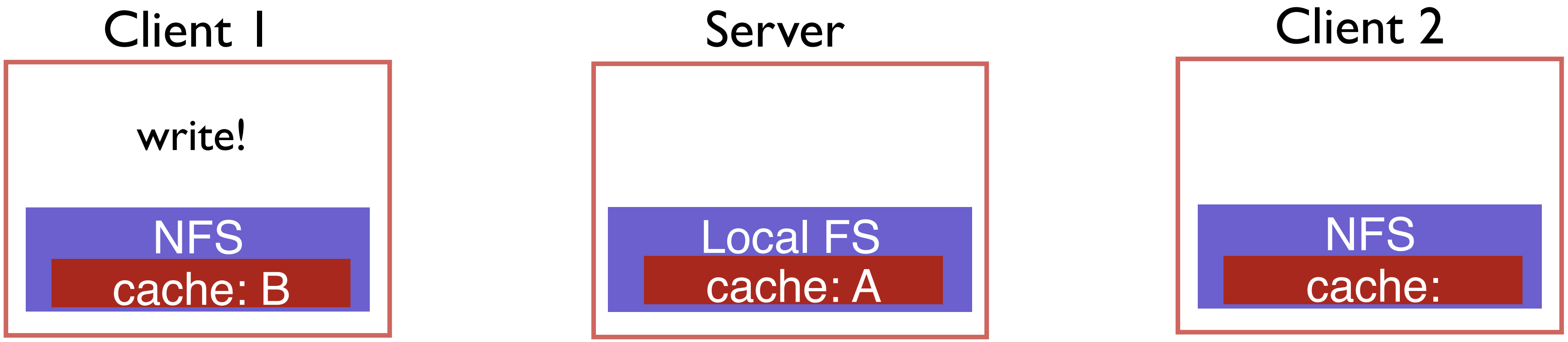
When is data cached?

How to make sure all versions are in sync?

DISTRIBUTED CACHE



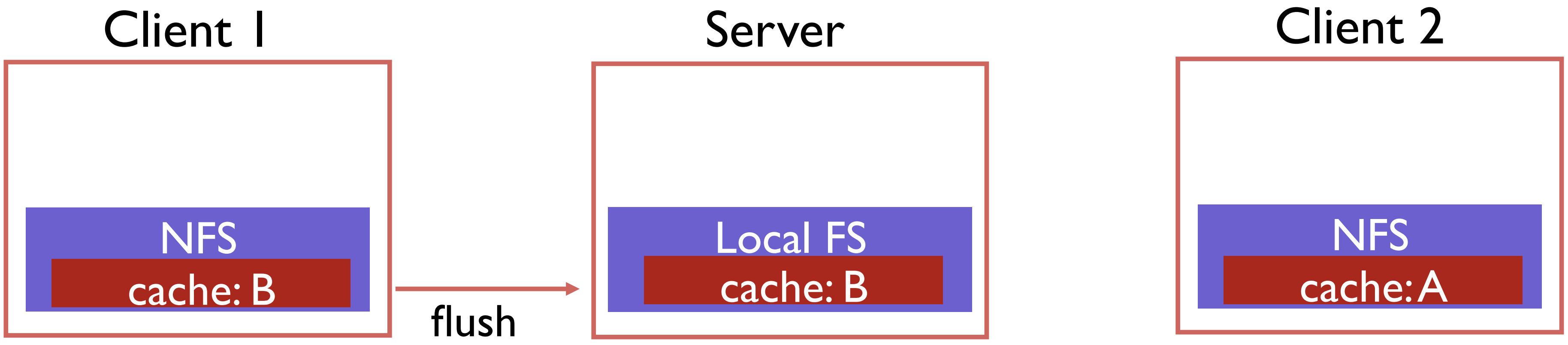
UPDATE VISIBILITY



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

What happens if Client 2 (or any other client) reads data?

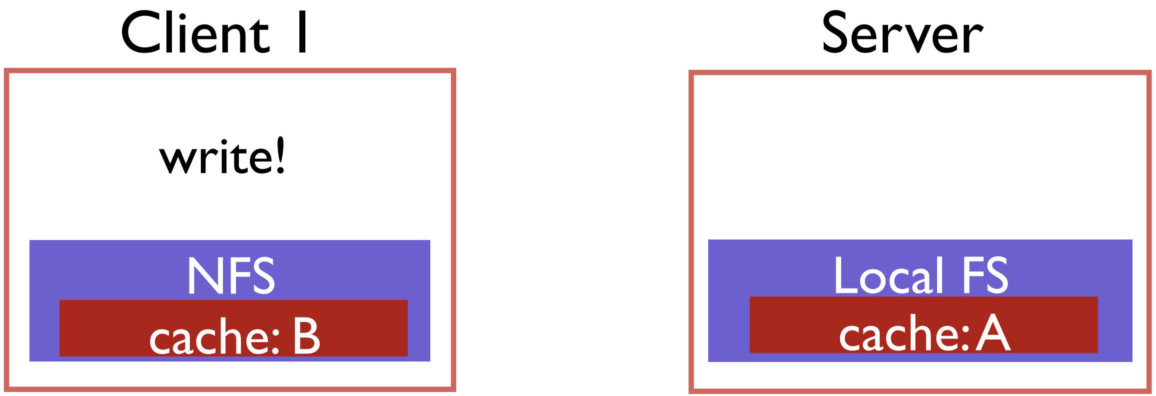
STALE CACHE



“Stale Cache” problem: client 2 doesn’t have latest version

What happens if Client 2 reads data?

SOLVING UPDATE VISIBILITY



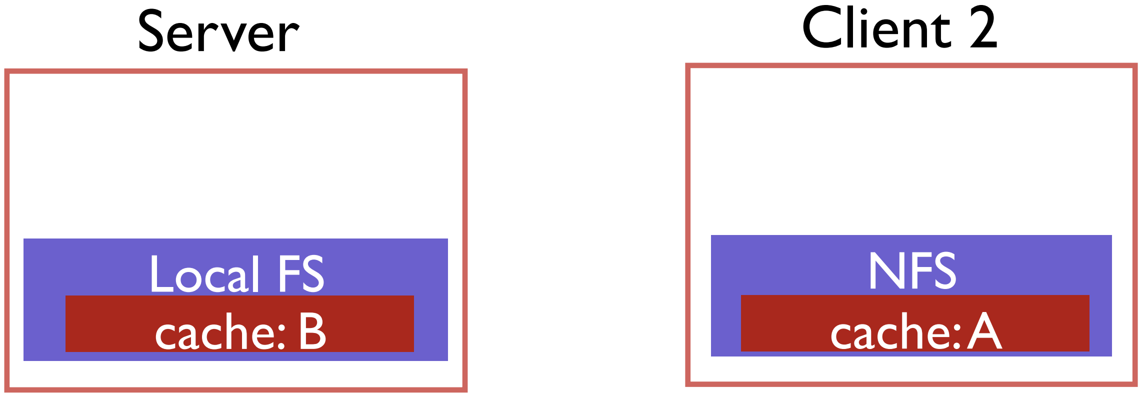
When client buffers a write, how can server (and other clients) see update?

Client flushes cache entry to server

When should client perform flush?

NFS solution: flush on fd close

SOLVING STALE CACHE

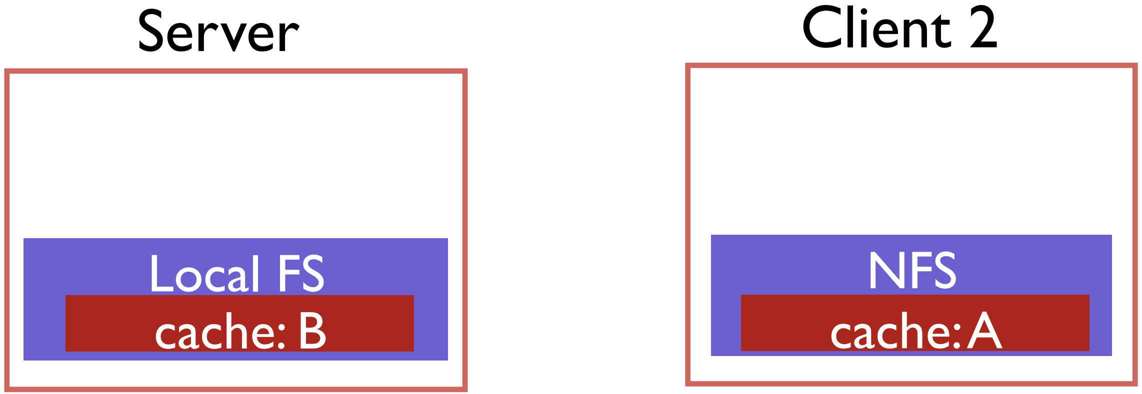


Problem: Client 2 has stale copy of data; how can it get the latest?

NFS solution:

- Clients recheck if cached copy is current before using data

STALE CACHE SOLUTION



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

Before using data block, client does a STAT request to server

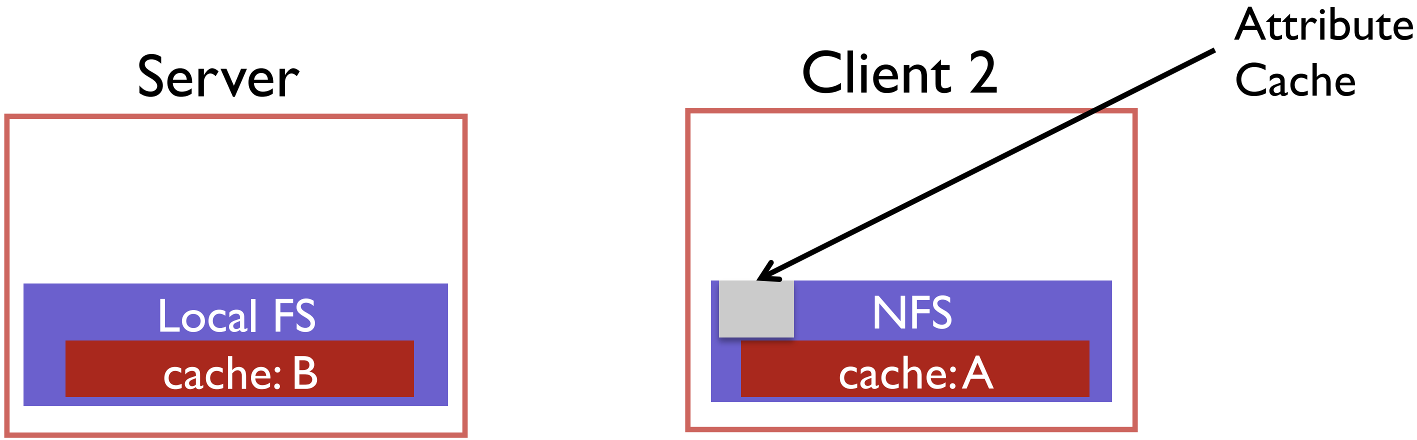
- get's last modified timestamp for this file (t2) (not block…)
- compare to cache timestamp
- refetch data block if changed since timestamp ($t2 > t1$)

Measure Then Build

NFS developers found `stat` accounted for 90% of server requests

Why? Because clients frequently recheck cache

REDUCING STAT CALLS



Solution: cache results of `stat` calls

Partial Solution:

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

What is the consequence?

NFS SUMMARY

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

- stateless: servers don't remember clients
- idempotent: doing things twice never hurts

Caching and write buffering is harder, especially with crashes

Problems:

- Consistency model is odd (client may not see updates until 3s after file closed)
- Scalability limitations as more clients call stat() on server

BUNNY 22!

https://tinyurl.com/cs537-sp19-bunny22

FEEDBACK?

https://aefis.wisc.edu/

BUNNY 22

https://tinyurl.com/cs537-sp19-bunny22

We'll now model the time of certain operations in NFS. The only costs to worry about are network costs. Assume any "small" message takes S units of time from one machine to another, whereas a "bigger" message (e.g., size of a disk block) takes B units. If a message is larger than 4KB, it should take proportionally longer (2B for 8KB). Assume we are using a file that is 100 blocks (400 KB) stored at /a/b/c.txt.

1. How long does it take to re-read a file immediately after it was read?

2. How long does it take to re-read the whole file after 10s assuming no edits to the file?

ALTERNATE DESIGN: ANDREW FILE SYSTEM (AFS)

WHOLE-FILE CACHING

Upon open, AFS client fetches whole file (even if huge), storing in local memory or disk

Upon close, client flushes file to server (if file was written)

Convenient and intuitive semantics:

- AFS needs to do work only for open/close
- Reads/writes are local
- Use same version of file entire time between open and close

UPDATE VISIBILITY

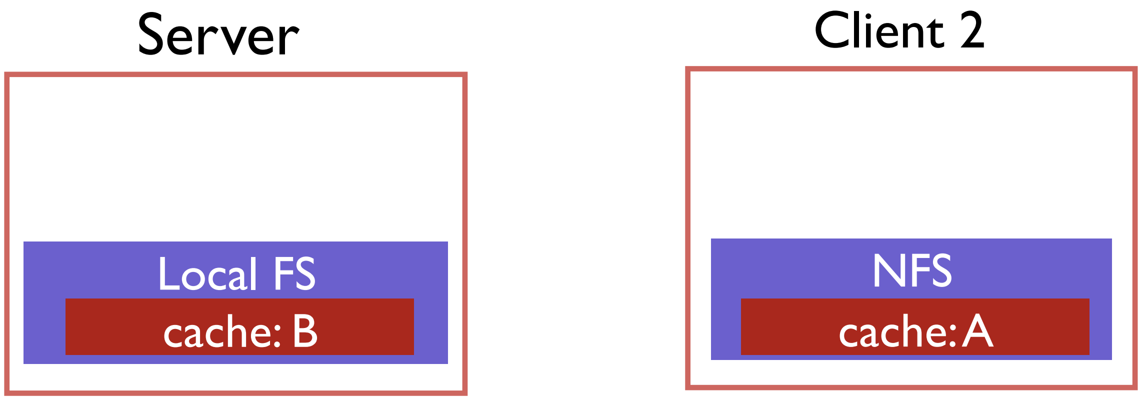
AFS solution:

- also flush on close
- buffer whole files on local disk; update file on server atomically

Concurrent writes?

- Last writer (i.e., last file closer) wins
- Never get mixed data on server

STALE CACHE



AFS solution: Tell clients when data is overwritten

- Server must remember which clients have this file open right now

When clients cache data, ask for "callback" from server if changes

- Clients can use data without checking all the time

Server no longer stateless!

SUMMARY

OPERATING SYSTEMS: THREE EASY PIECES

Three conceptual pieces

1. Virtualization

2. Concurrency

3. Persistence

VIRTUALIZATION

Make each application believe it has each resource to itself

CPU and Memory

Abstraction: Process API, Address spaces

Mechanism:

Limited direct execution, CPU scheduling

Address translation (segmentation, paging, TLB)

Policy: MLFQ, LRU etc.

CONCURRENCY

Events occur simultaneously and may interact with one another

Need to

- Hide concurrency from independent processes
- Manage concurrency with interacting processes

Provide abstractions (locks, semaphores, condition variables etc.)

Correctness: mutual exclusion, ordering

Performance: scaling data structures, fairness

Common Bugs!

PERSISTENCE

Managing devices: key role of OS!

Hard disk drives

- Rotational, Seek, Transfer time
- Disk scheduling: FIFO, SSTF, SCAN

Filesystems API

- File descriptors, Inodes
- Directories
- Hardlinks, softlinks

PERSISTENCE

Very simple FS

- Inodes, Bitmaps, Superblock, Data blocks

FFS

- Placement in groups, Allocation policy

LFS

- Write optimized, Garbage collection

Journaling, FSCK

NFS: Partial failures retry, cache consistency

OPERATING SYSTEMS FOR THE CLOUD?

The Datacenter Needs an Operating System

Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi,
Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

1 Introduction

Clusters of commodity servers have become a major computing platform, powering not only some of today's most popular consumer applications—Internet services such as search and social networks—but also a growing number of scientific and enterprise workloads [2]. This rise in cluster computing has even led some to declare that "the datacenter is the new computer" [16, 24]. However, the tools for managing and programming this new computer are still immature. This paper argues that, due to the growing diversity of cluster applications and users, the datacenter increasingly needs an operating system.[1]

and Pregel steps). However, this is currently difficult because applications are written independently, with no common interfaces for accessing resources and data.

In addition, clusters are serving increasing numbers of concurrent users, which require responsive time-sharing. For example, while MapReduce was initially used for a small set of batch jobs, organizations like Facebook are now using it to build data warehouses where hundreds of users run near-interactive ad-hoc queries [29].

Finally, programming and debugging cluster applications remains difficult even for experts, and is even more challenging for the growing number of non-expert users

DATACENTER OPERATING SYSTEMS

Resource sharing

Data sharing

Programming Abstractions

Debugging

THANK YOU!