NFS, SUMMARY

Shivaram Venkataraman

CS 537, Spring 2023

Administrivia

Project 7 grades

Project 8 deadline

Quiz grades

Midterm 3!

AGENDA / LEARNING OUTCOMES

How to design a distributed file system that can survive partial failures?

What are consistency properties for such designs?

RECAP

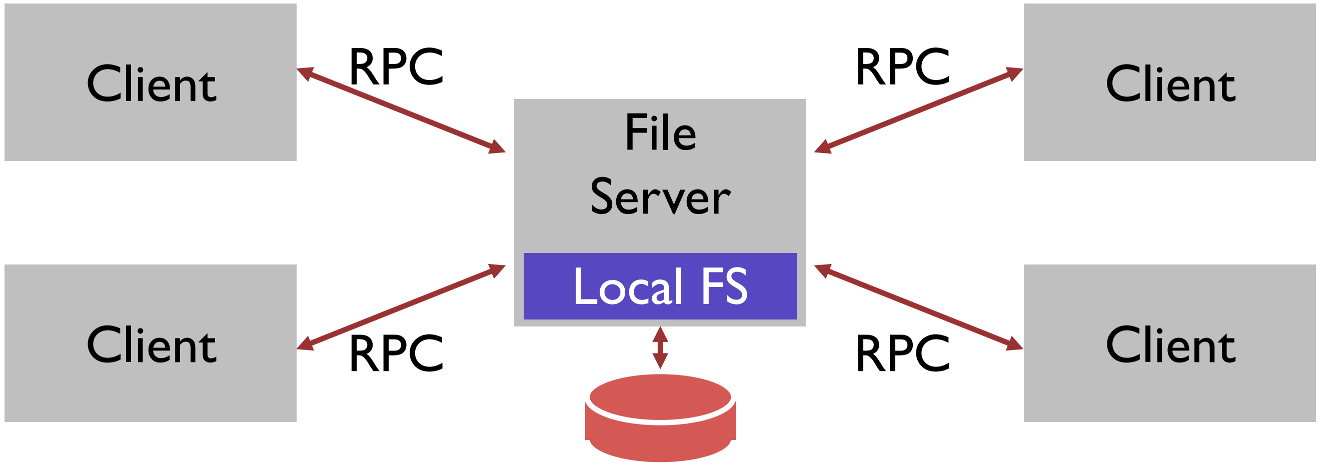
DISTRIBUTED FILE SYSTEMS

Local FS: processes on same machine access shared files

Network FS: processes on different machines access shared files in same way

Goals
- Transparent access
- Fast + simple crash recovery
- Reasonable performance?

NFS ARCHITECTURE



STRATEGY 3: FILE HANDLES

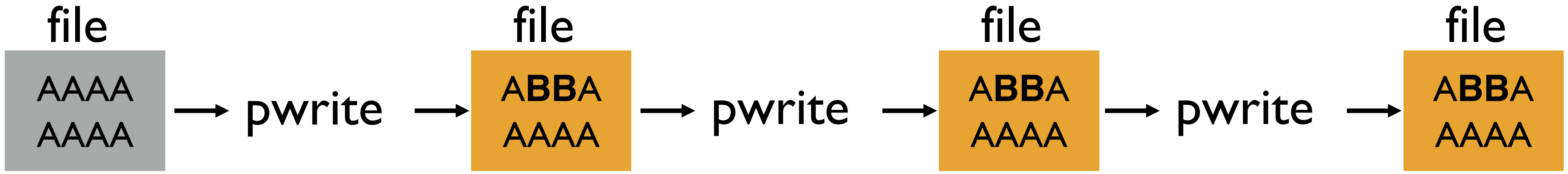
```
fh = open(char *path);
pread(fh, buf, size, offset);
pwrite(fh, buf, size, offset);
```

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

Opaque to client (client should not interpret internals)

PWRITE VS APPEND

pwrite(file, “BB”, 2, 2);



append(file, “BB”);

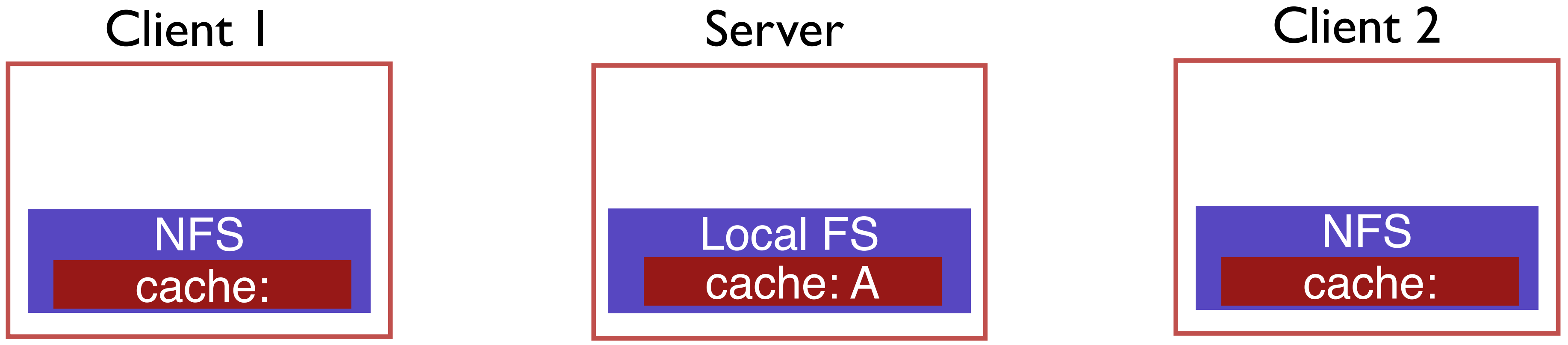
CACHE CONSISTENCY

NFS can cache data in three places:

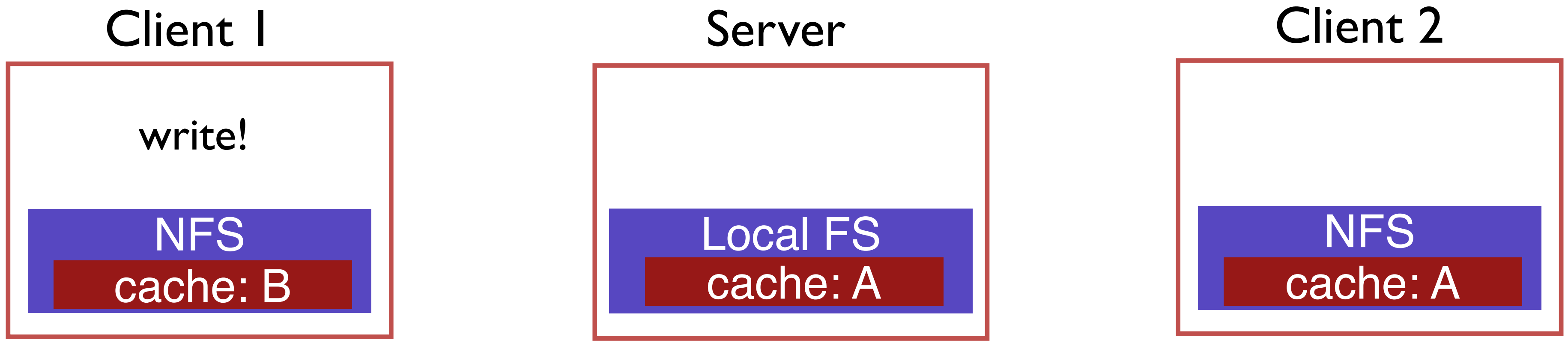
- server memory
- client disk
- client memory

How to make sure all versions are in sync?

DISTRIBUTED CACHE



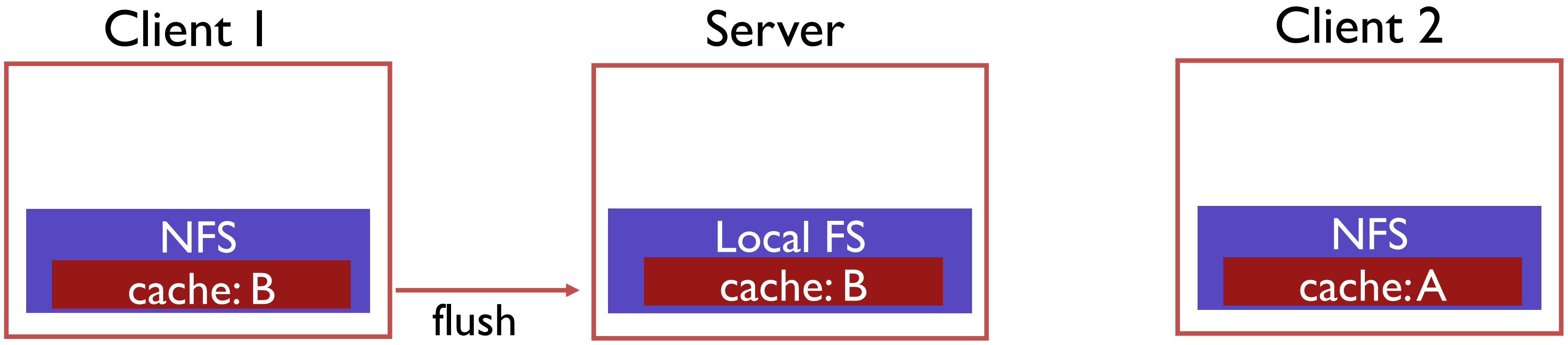
CACHE



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

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

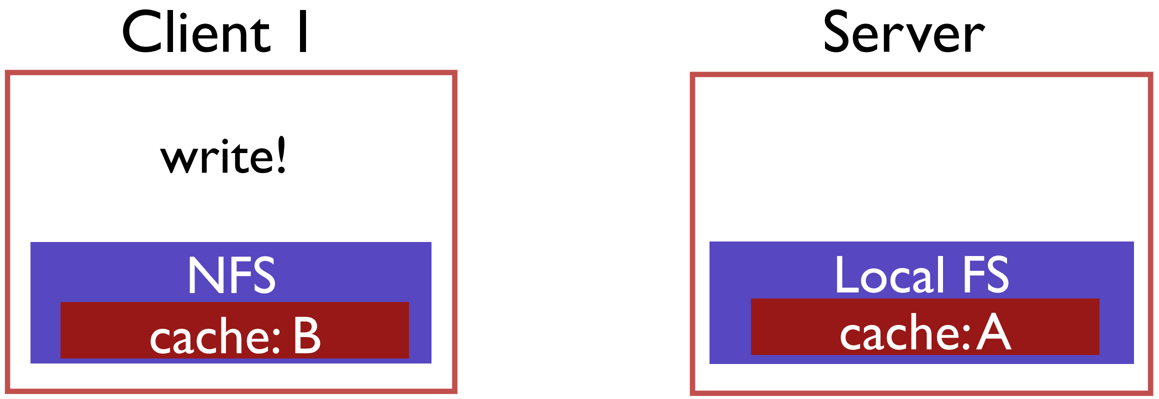
CACHE



"Stale Cache" problem: client 2 doesn't have latest version

What happens if Client 2 reads data?

PROBLEM 1: UPDATE VISIBILITY



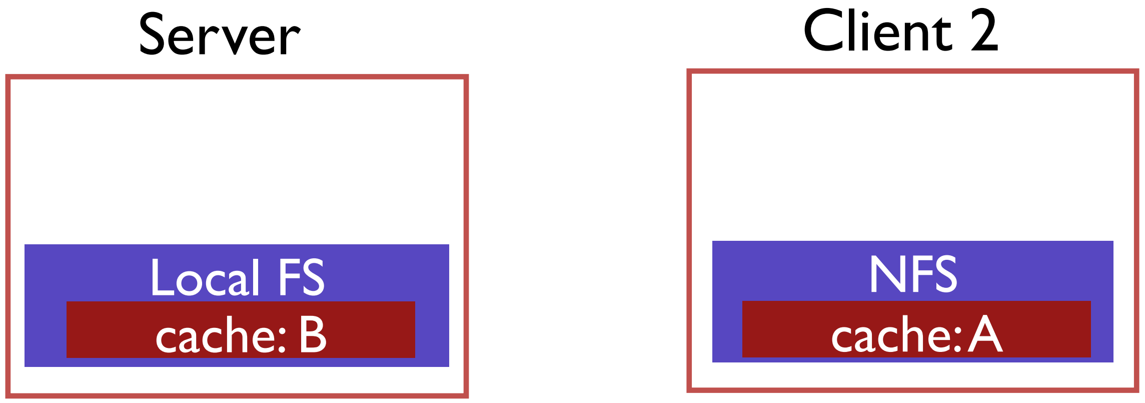
When client buffers a write, how can server (and other clients) see update?

Client flushes cache entry to server

When should client perform flush?

NFS solution: flush on fd close

PROBLEM 2: STALE CACHE

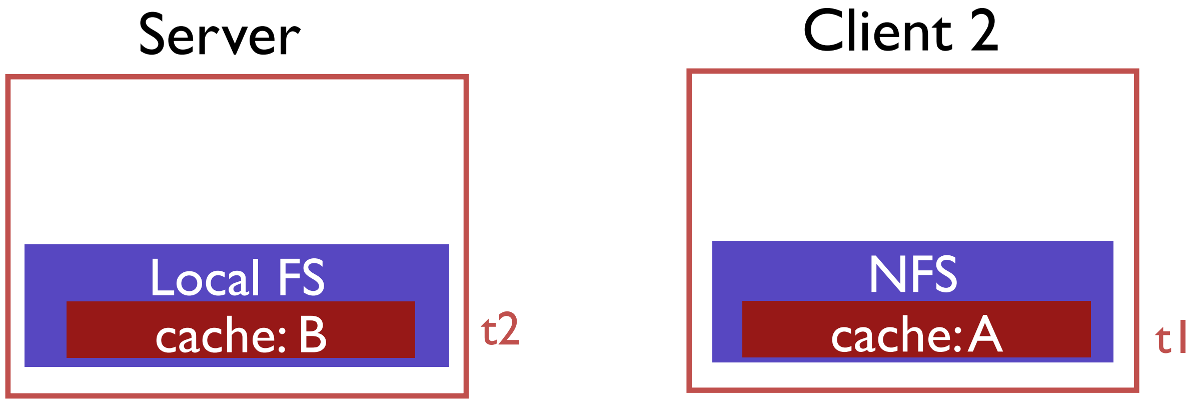


Problem: Client 2 has stale copy of data; how can it get the latest?

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)

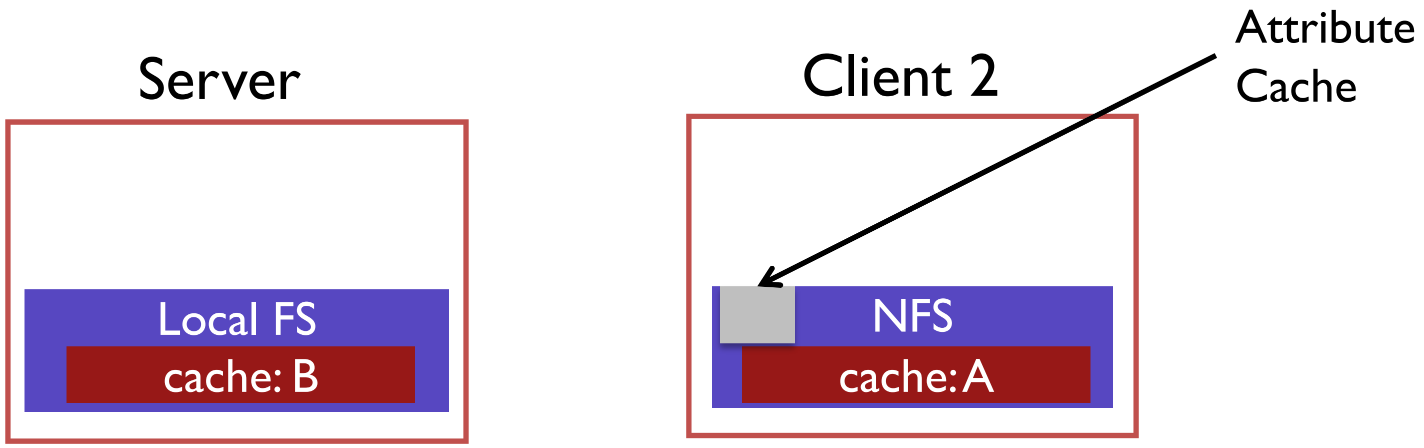
Measure Then Build

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

Why?

Because clients frequently recheck cache

REDUCING STAT CALLS



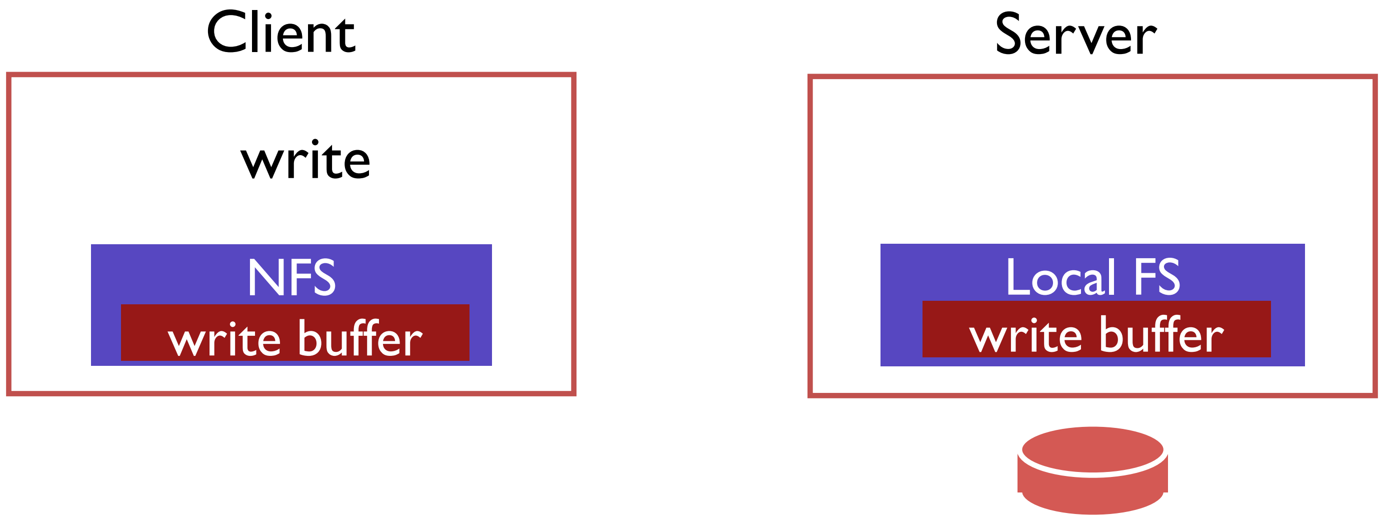
Solution: cache results of `stat` calls

Partial Solution:

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

What is the consequence?

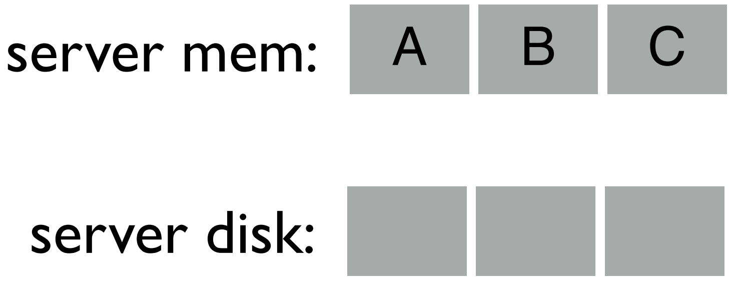
WRITE BUFFERS



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

SERVER WRITE BUFFER LOST

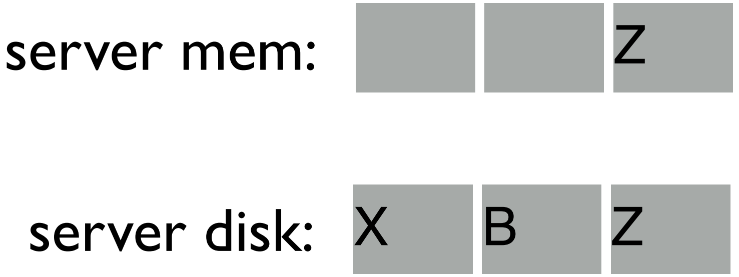
```
client:


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



server acknowledges write before write is pushed to disk

SERVER WRITE BUFFER LOST

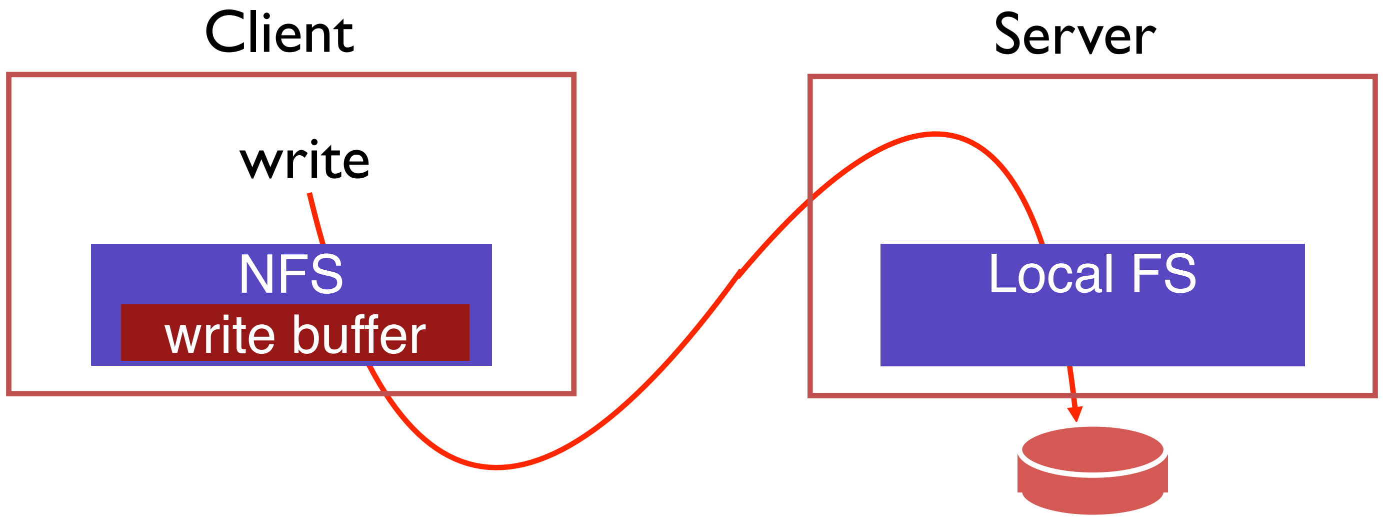
```
Client:

 write A to 0

 write B to 1

 write C to 2

 write X to 0

 write Y to 1

 write Z to 2
```



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)

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

FEEDBACK!

https://aefis.wisc.edu/

1. What was one idea or concept that you learnt in this course that you appreciated the most?

2. What are some future opportunities that you look forward to based on content from 537?

LOOKING FORWARD:
OS/FILESYSTEMS FOR THE CLOUD?

FROM MID 2006

Rent virtual computers in the "Cloud"

On-demand machines, spot pricing

amazon web services™

Microsoft Azure

Google Compute Engine

AMAZON EC2 (2018)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.nano	0.5	1	0	$0.0058
r5d.24xlarge	~~244~~ 768	~~104~~ 96	4x900 NVMe	$6.912
x1.32xlarge	2 TB	4 * Xeon E7	3.4 TB (SSD)	$13.338
p3.16xlarge	488 GB	8 Nvidia Tesla V100 GPUs	0	$24.48

DATACENTER EVOLUTION

Capacity:

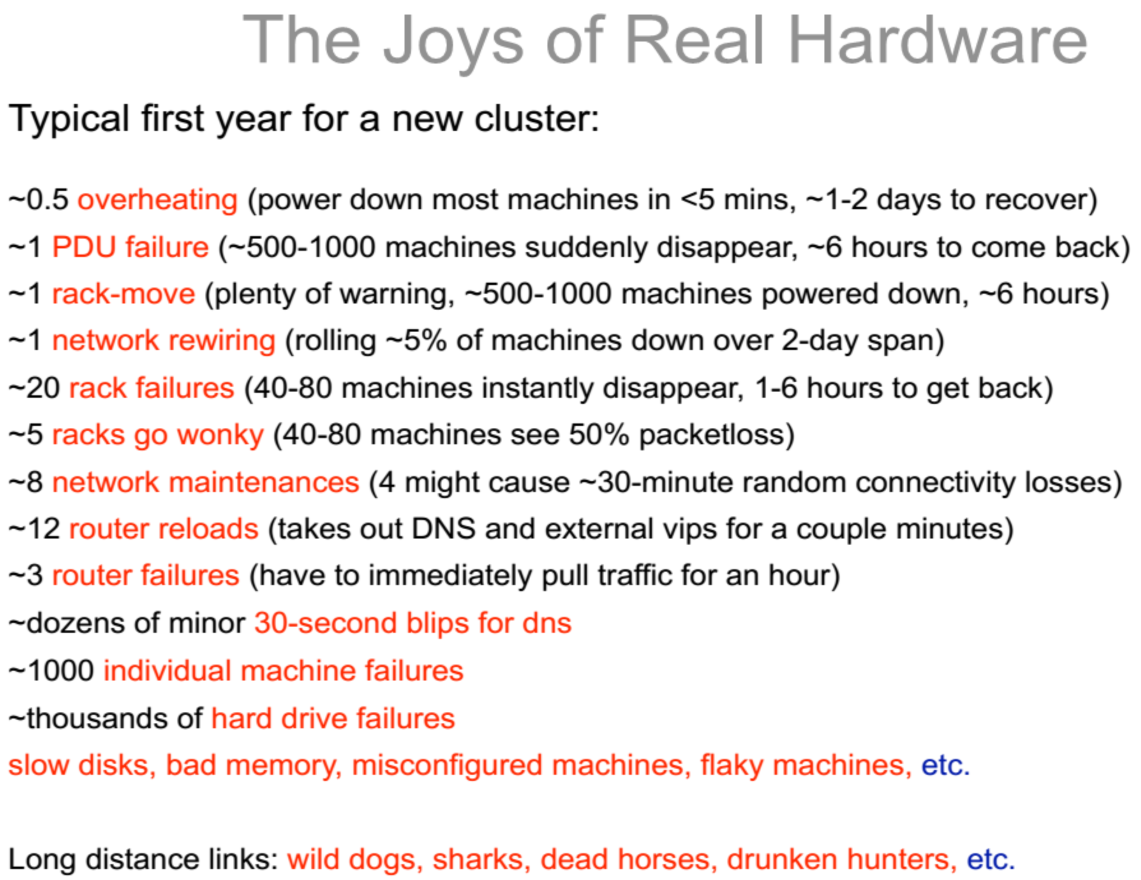
~10000 machines

Bandwidth:

12-24 disks per node

Latency:

256GB RAM cache



The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

JEFF DEAN @ GOOGLE

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 (e.g., scientists) starting to leverage cloud computing

DATACENTER OPERATING SYSTEMS

Resource sharing

Data sharing

Programming Abstractions

Debugging

COURSE 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

NEXT COURSES

CS 640: Computer Networks

CS 736: Advanced Operating Systems

CS 739: Advanced Distributed Systems

CS 744: Big Data Systems

THANK YOU!