Welcome back!

CS 744: GOOGLE FILE SYSTEM

Shivaram Venkataraman

Spring 2024

ANNOUNCEMENTS

- Assignment 1 out today → noon or so
- Group submission form → Today!!
- No class on Thursday! → Rest of the schedule is on the website

- Anybody on the waitlist?

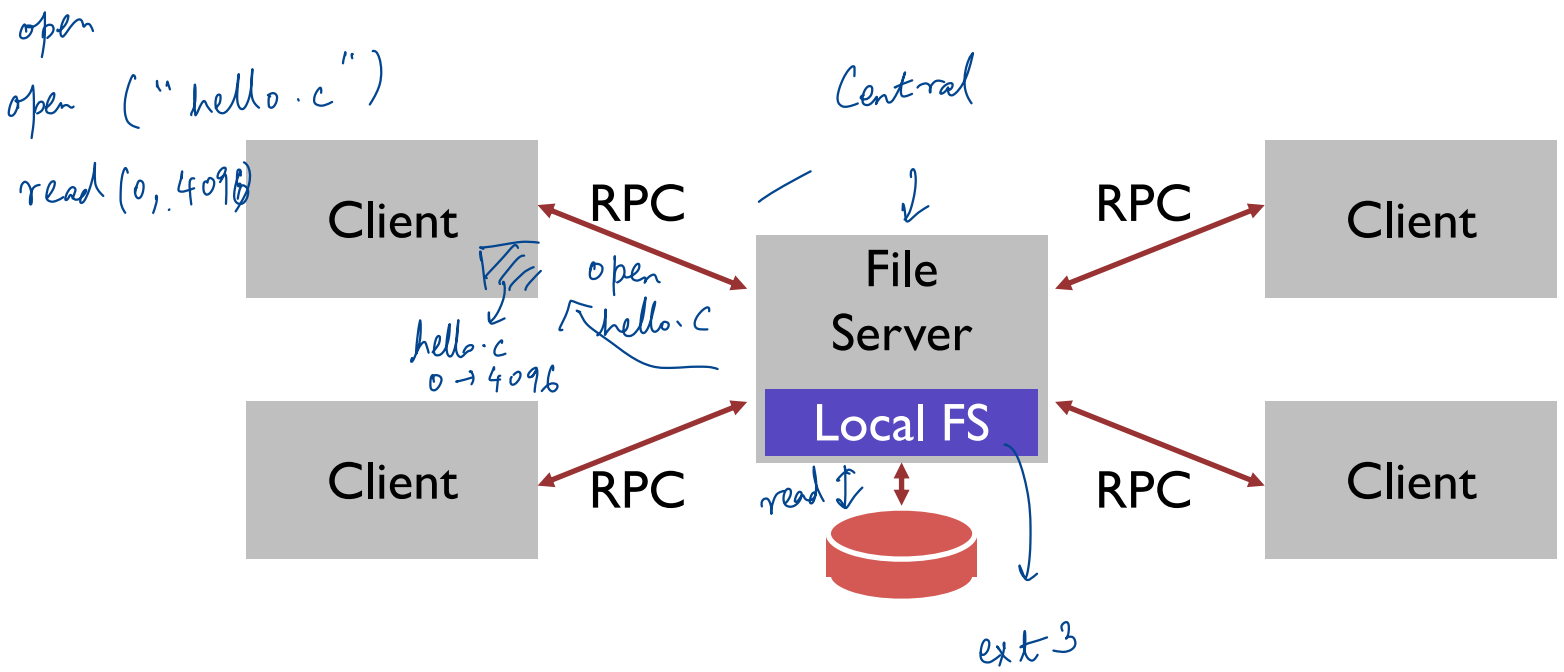
OUTLINE

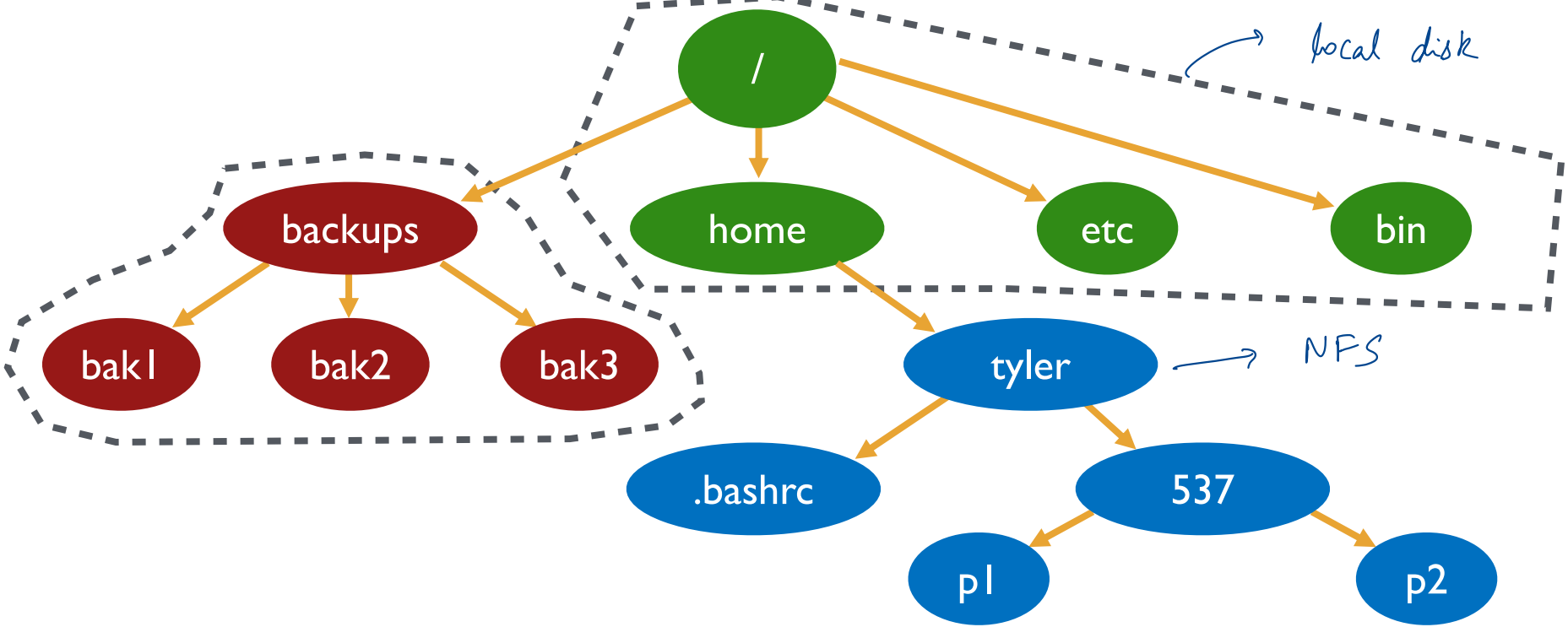
1. Brief history
2. GFS
3. Discussion
4. What happened next?

HISTORY OF DISTRIBUTED FILE SYSTEMS

SUN NFS

Office / Campus → ~1980s





/dev/sda1 **on** /

/dev/sdb1 **on** /backups

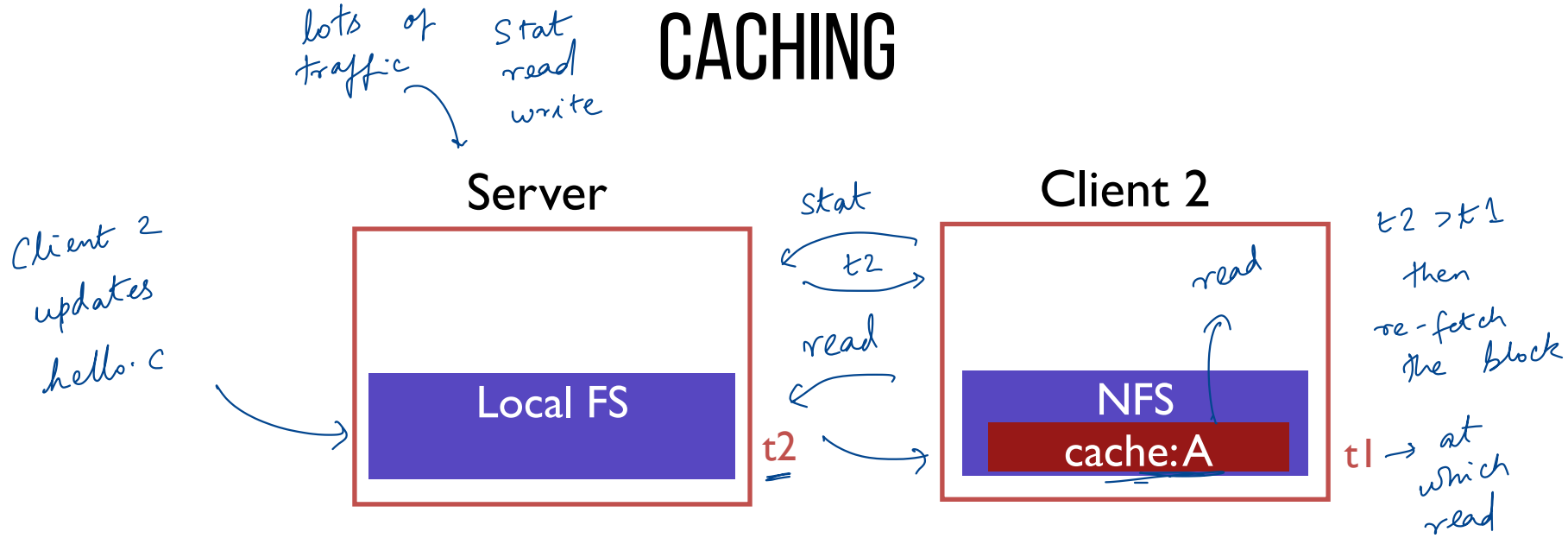
NFS **on** /home

Transparency

↳ unmodified application

vim / gcc / ...

CACHING



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)

ANDREW FILE SYSTEM

(AFS)

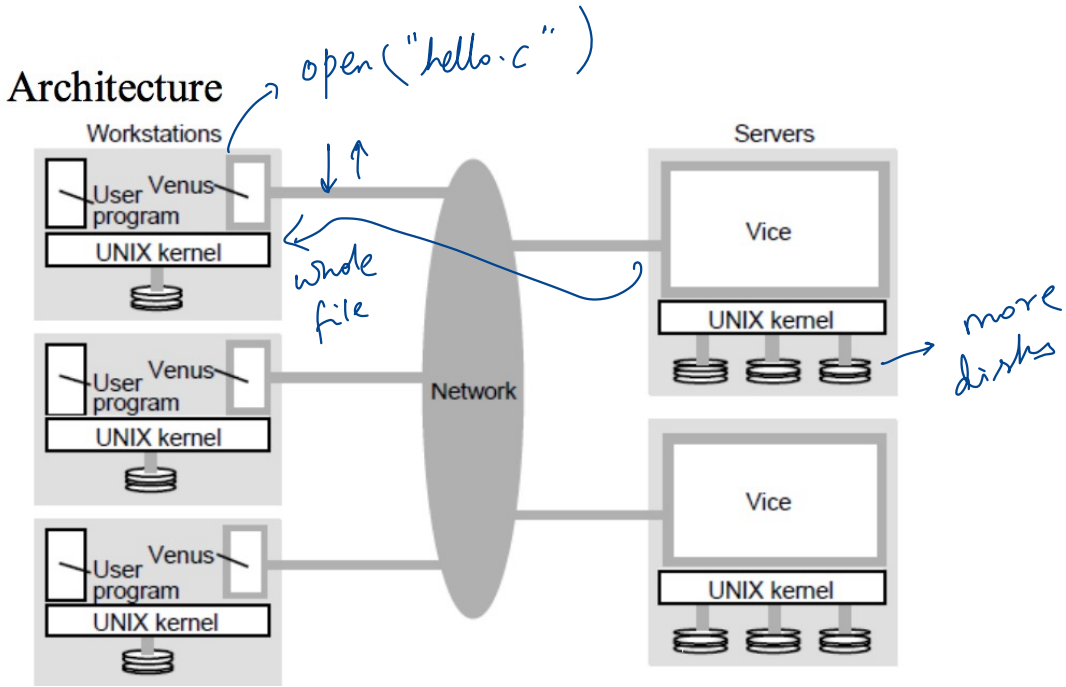
↳ CSL

late ~1980s

- Design for scale

- Whole-file caching

- Callbacks from server

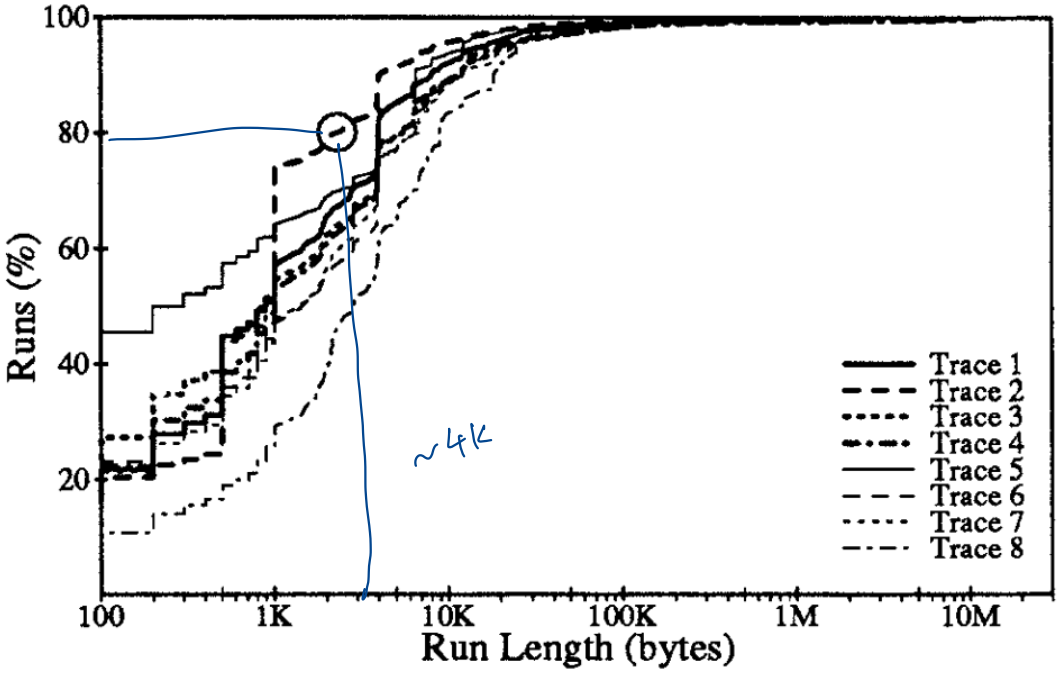


files were small. Mostly read/written by single client

WORKLOAD PATTERNS (1991)

most of file read/write lengths were small

→ machines in college campus



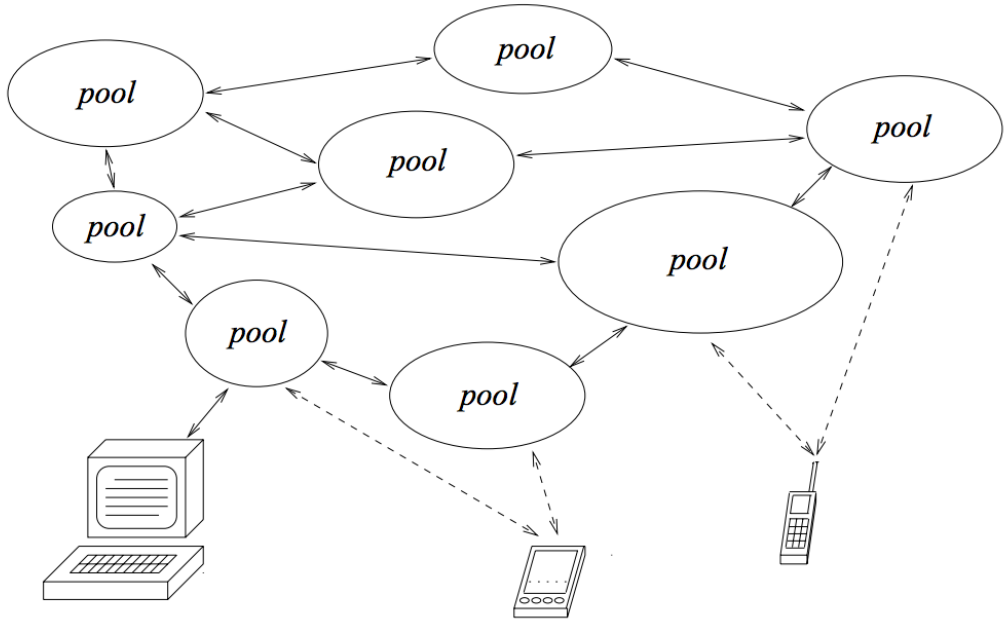
Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout

OCEANSTORE/PAST

Wide area storage systems (across the internet)

Fully decentralized (no single central server)

Built on distributed hash tables (DHT) → Bit Torrent



Failures are the norm

↳ commidity hardware → recover from failures quickly

↳ scale → 1000s of disks

↳ bottleneck → Single organization

GFS: WHY ?

Workload pattern

↳ Large reads → sequential

Large writes → appends → bandwidth rather than latency

→ Larger files, large number of clients

Components with failures

Files are huge !

GFS: WHY ?

Applications are different

GFS: WORKLOAD ASSUMPTIONS

"Modest" number of large files

Two kinds of reads: Large Streaming and small random

Writes: Many large, sequential writes. Few random

High bandwidth more important than low latency

GFS: DESIGN

- Single Master for metadata

- Chunkservers for storing data

- No POSIX API !
- No Caches!

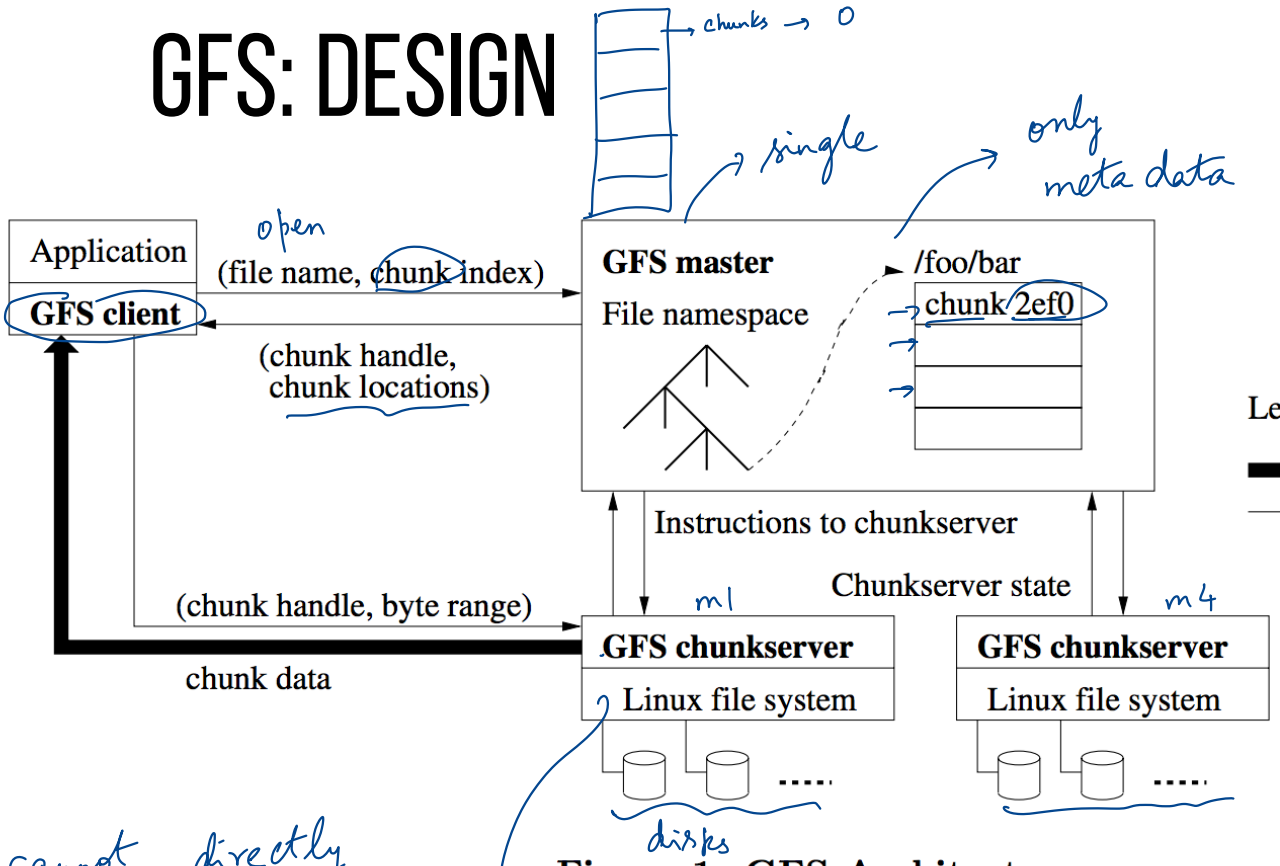


Figure 1: GFS Architecture

" gfs.h "

chunks → 0

single

only meta data

don't care abt latency

you cannot directly use this with vim / gcc

all the reads / writes

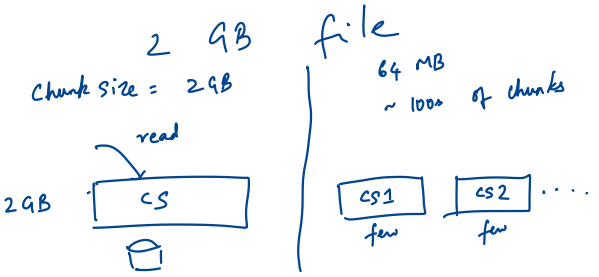
CHUNK SIZE TRADE-OFFS → 64 mb

Client → Master → every time you read a new chunk send RPC

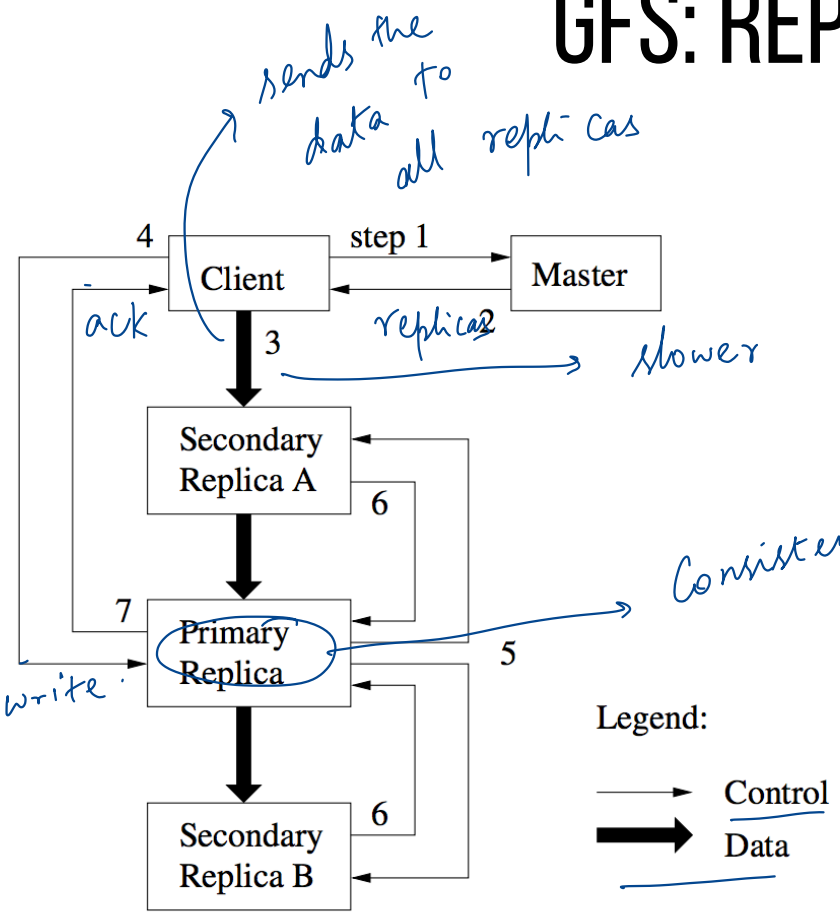
Client → Chunkserver

Metadata → more chunks → more metadata require much more memory

Fragmentation ?



GFS: REPLICATION



k-way

- 3-way replication to handle faults
- Primary replica for each chunk
- Chain replication (consistency)

- Decouple data, control flow
- Dataflow: Pipelining, network-aware → send to closest replica first.

RECORD APPENDS →

Write	Client specifies the offset
Record Append	GFS chooses offset

write (4096, "foo")

append ("foo") → offset

↳ structured data

[HTTP 200, "wisconsin", IP]

Consistency

- At-least once → record will appear at least once. Duplicates
- Atomic → entire record appears together

  ↳ Readers can filter out incomplete / duplicate

MASTER OPERATIONS

- No "directory" inode! Simplifies locking
- Replica placement considerations
- Implementing deletes

→ a Sym links are not support

/ gfs / a / b
↳ filename string

→ one where the client

→ one in same rack, diff rack

load balancing

→ Lazy · Metadata update

FAULT TOLERANCE

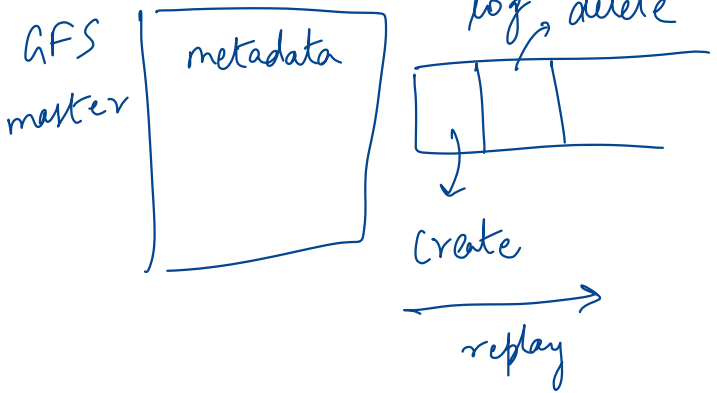
- Chunk replication with 3 replicas
- Master
  - Replication of log, checkpoint
  - Shadow master
- Data integrity using checksum blocks

Chunk servers can fail

Master kick off extra replica on failure

fast recovery

↳ replica. only serves read ops.



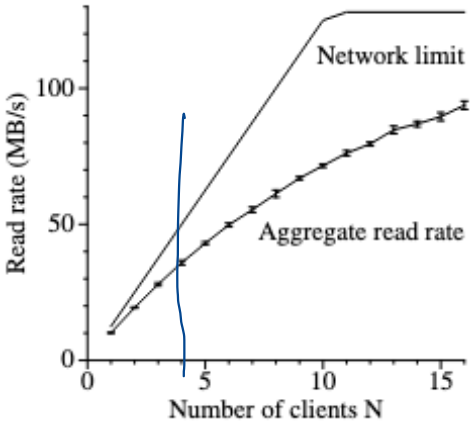
DISCUSSION

https://forms.gle/yPwbLvjjqKHevZ4k6

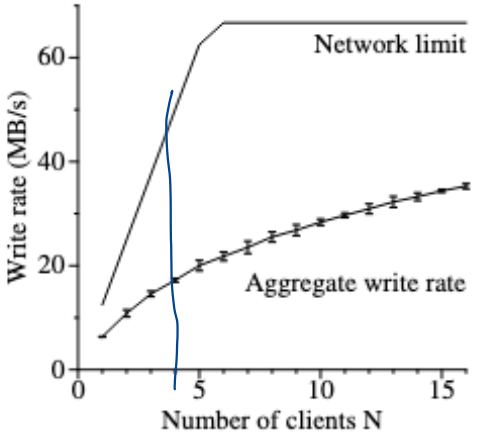
What happens with a faster network (125MB/s) but same disks (100 MB/s)?

1 master, 2 shadow
16 chunk servers, 16 clients
switch switch

80 GB 5400 rpm disks
100 Mbps full duplex → 1 Gbps
1 Gbps between switch
full duplex

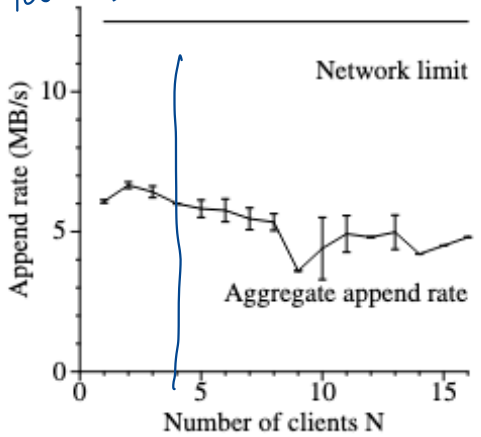


(a) Reads

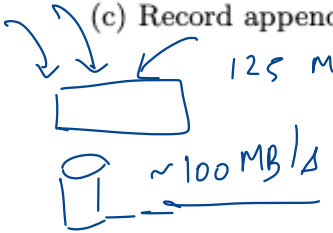


(b) Writes

100 MB/s



(c) Record appends



Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
0K	0.4	2.6	0	0	0	0
1B..1K	0.1	4.1	6.6	4.9	0.2	9.2
1K..8K	65.2	38.5	0.4	1.0	18.9	15.2
8K..64K	29.9	45.1	17.8	43.0	78.0	2.8
64K..128K	0.1	0.7	2.3	1.9	< .1	4.3
128K..256K	0.2	0.3	31.6	0.4	< .1	10.6
256K..512K	0.1	0.1	4.2	7.7	< .1	31.2
512K..1M	3.9	6.9	35.5	28.7	2.2	25.5
1M..inf	0.1	1.8	1.5	12.3	0.7	2.2

Table 4: Operations Breakdown by Size (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested.

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
1B..1K	< .1	< .1	< .1	< .1	< .1	< .1
1K..8K	13.8	3.9	< .1	< .1	< .1	0.1
8K..64K	11.4	9.3	2.4	5.9	2.3	0.3
64K..128K	0.3	0.7	0.3	0.3	22.7	1.2
128K..256K	0.8	0.6	16.5	0.2	< .1	5.8
256K..512K	1.4	0.3	3.4	7.7	< .1	38.4
512K..1M	65.9	55.1	74.1	58.0	.1	46.8
1M..inf	6.4	30.1	3.3	28.0	53.9	7.4

Table 5: Bytes Transferred Breakdown by Operation Size (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested. The two may differ if the read attempts to read beyond end of file, which by design is not uncommon in our workloads.

WHAT HAPPENED NEXT

Cluster-Level Storage @ Google

How we use *Colossus* to improve storage efficiency

Denis Serenyi
Senior Staff Software Engineer
dserenyi@google.com

Keynote at PDSW-DISCS 2017: 2nd Joint International Workshop On Parallel Data Storage & Data Intensive Scalable Computing Systems

GFS EVOLUTION

Motivation:

- GFS Master
  - One machine not large enough for large FS
  - Single bottleneck for metadata operations (data path offloaded)
  - Fault tolerant, but not HA

- Lack of predictable performance
  - No guarantees of latency
  - (GFS problems: one slow chunkserver -> slow writes)

GFS EVOLUTION

GFS master replaced by Colossus

Metadata stored in BigTable

Recursive structure ? If Metadata is ~1/10000 the size of data

100 PB data → 10 TB metadata

10TB metadata → 1GB metametadata

1GB metametadata → 100KB meta...

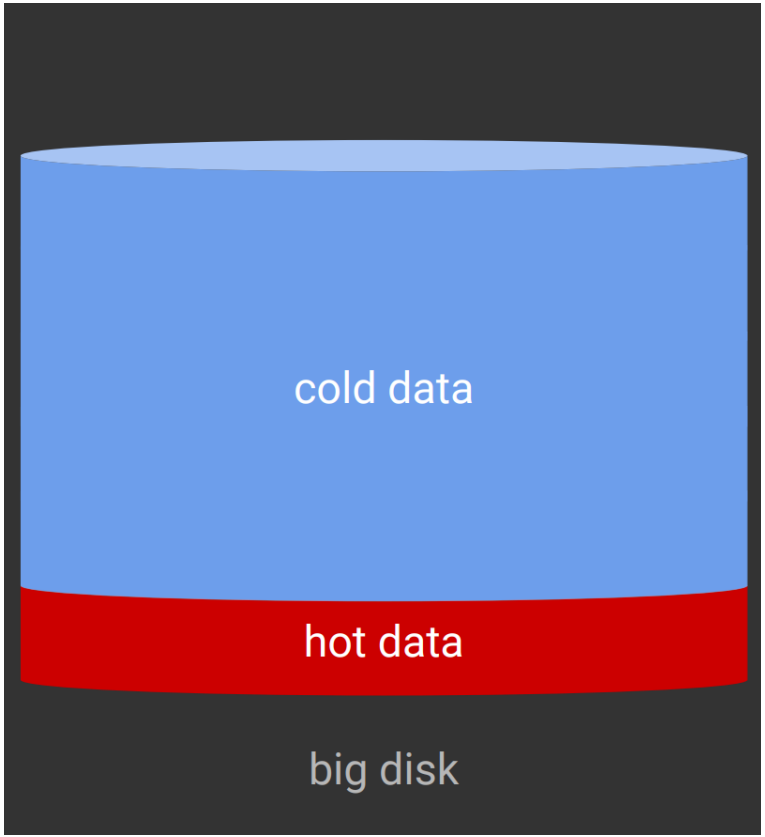
GFS EVOLUTION

Need for Efficient Storage

Rebalance old, cold data

Distributes newly written data evenly across disk

Manage both SSD and hard disks



NEXT STEPS

- Assignment 1 out tonight!
- No class on Thursday
- Next up: MapReduce, Spark