

Hello!

CS 744: GOOGLE FILE SYSTEM

Shivaram Venkataraman

Spring 2025

ANNOUNCEMENTS

- Assignment I out today
- Group submission form → form groups on CloudLab
- Office hours
 - Shivaram Venkataraman: Tue 3pm-4pm at CS 7367
 - Tareq Mahmood: Tue 4pm-5pm, Thu 4pm-5pm at CS 3250

↳ Tutorial on Apache Spark → Zoom

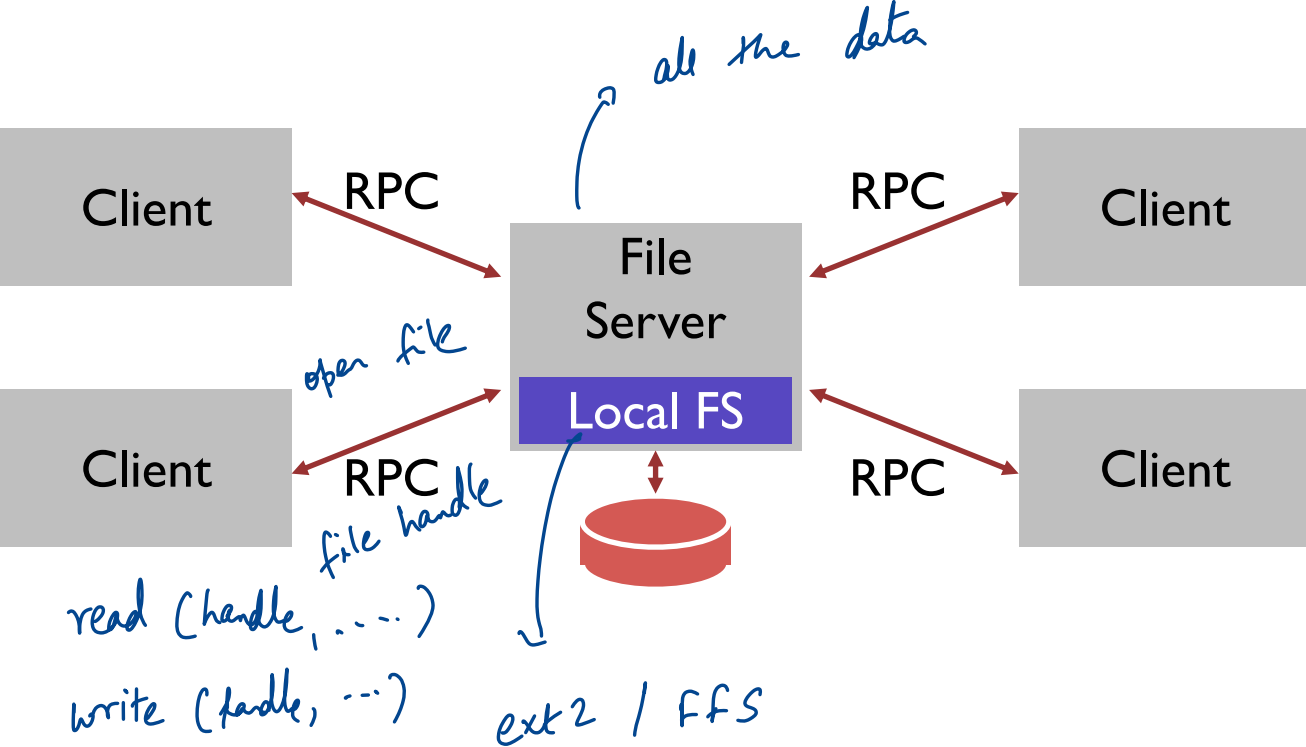
OUTLINE

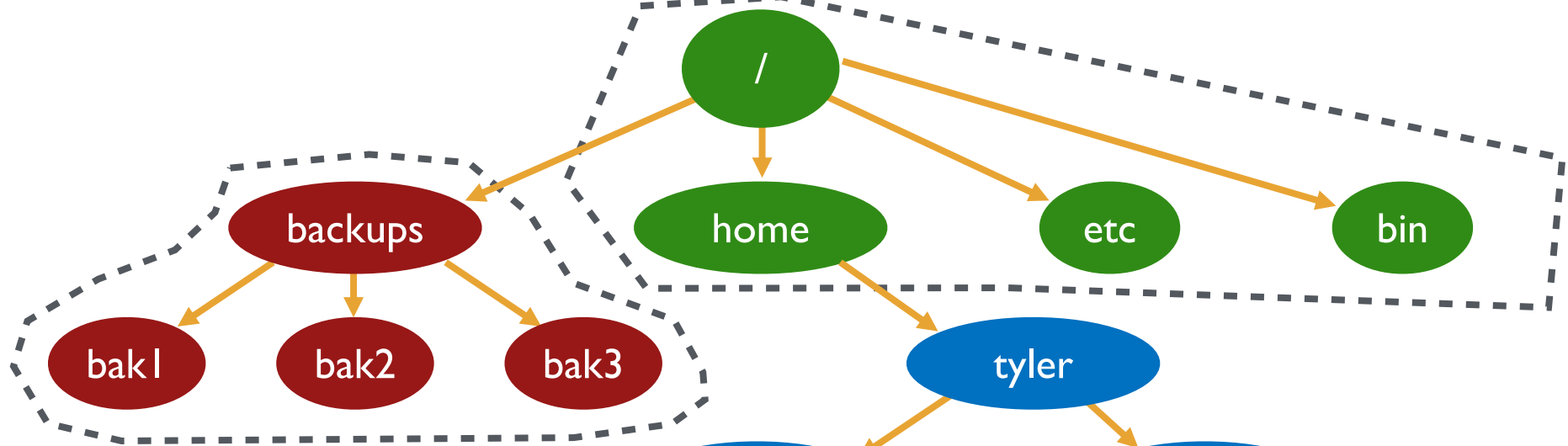
1. Brief history *before 2003*
2. GFS
3. Discussion
4. What happened next?

HISTORY OF DISTRIBUTED FILE SYSTEMS

SUN NFS

→ mid 1980s



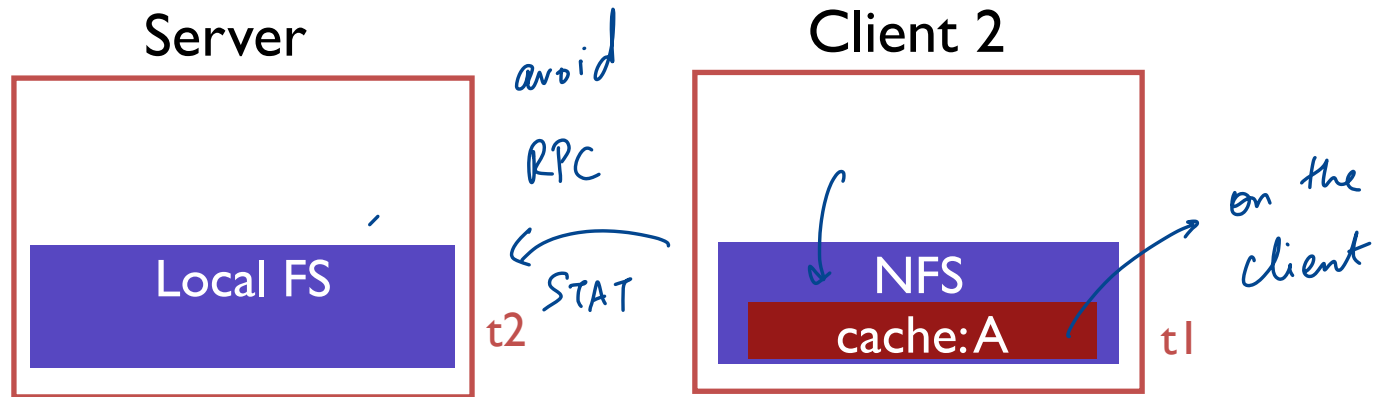


mount NFS

/dev/sda1 **on** /
 /dev/sdb1 **on** /backups
 NFS **on** /home

vim ; gcc
 file reside in NFS
 POSIX semantics

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

open a file

entire file cached at the client

→ read / write to cache

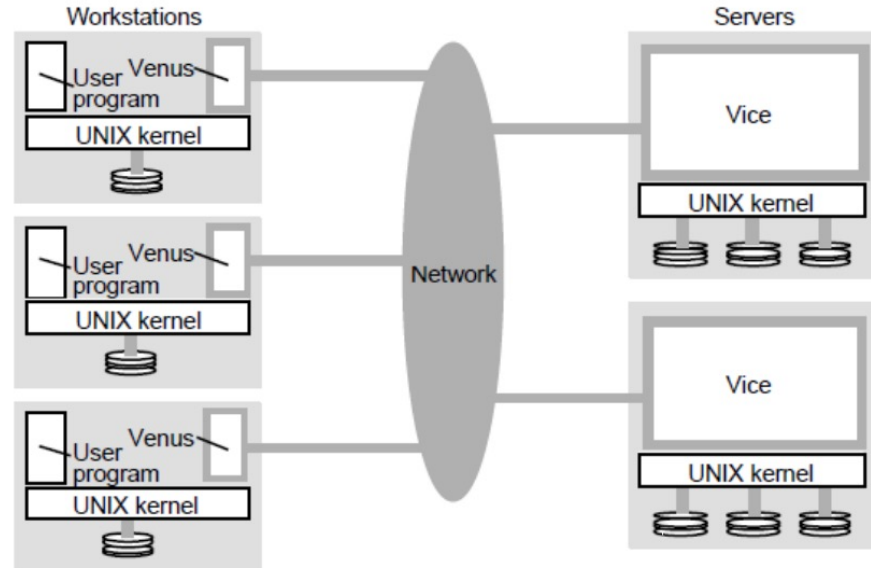
→ flushed file was closed.

- Design for scale

- Whole-file caching

- Callbacks from server

Architecture



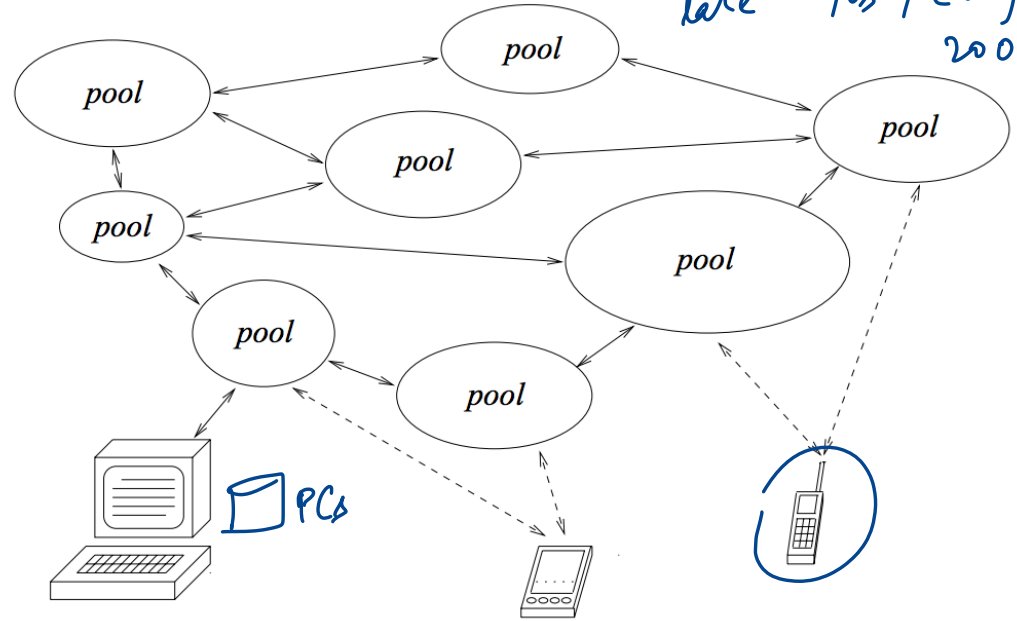
OCEANSTORE/PAST

→ examples of system designs
late 90s / early 2000s

Wide area storage systems

Fully decentralized

Built on distributed hash tables (DHT)



Large files

↳ ~ 100s of MBs

vs.

KBs etc. prior FSs.

→ Random writes
rare

append only operations

sequential reads

GFS: WHY ?

Commodity hardware

↳ Frequent failures

Latency was not
important

but throughput
was more
important

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!

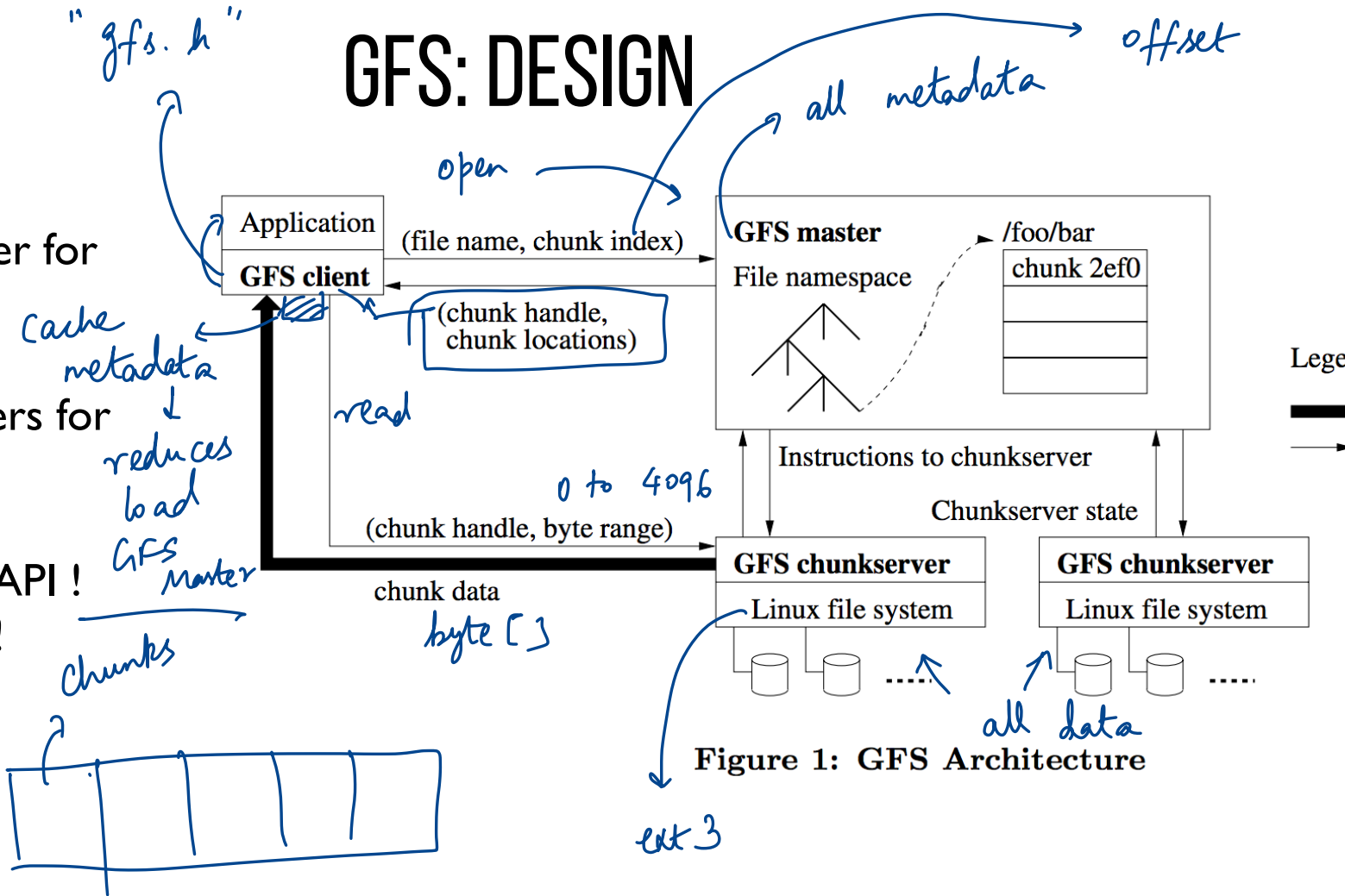


Figure 1: GFS Architecture

CHUNK SIZE TRADE-OFFS

Client → Master *→ too many requests*

Client → Chunkserver *→ lots of small files on chunk server
↳ avoid fragmentation?*

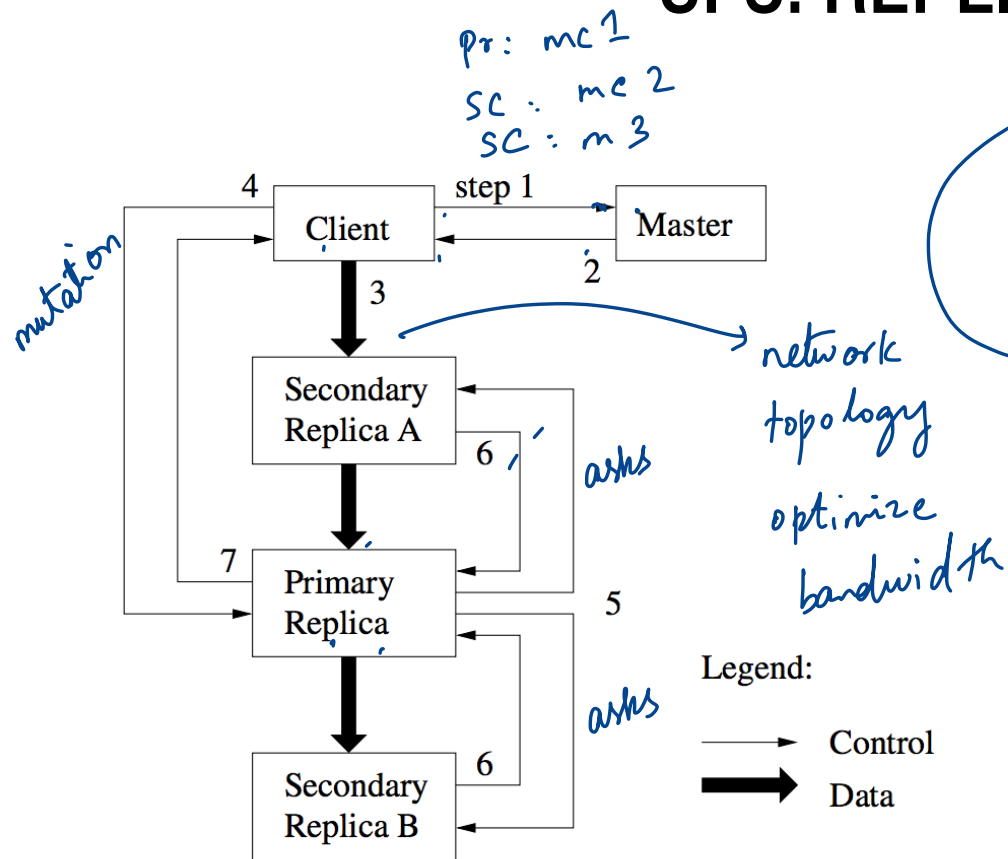
Metadata → *too big if chunk size is small
↳ GFS Master memory limits*

too large chunk size

↳ need padding? maybe not?

→ hotspot on the chunk server

GFS: REPLICATION



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

- Decouple data, control flow
- Dataflow: Pipelining, network-aware

RECORD APPENDS

Write

Record Append

Client specifies the offset

GFS chooses offset →

Workload

large number of
clients append at
the same time!

Consistency

At-least once

Atomic



entire record
appear together

```
struct {  
    int response code;  
    double ts;  
    string browser;  
} log record;
```

MASTER OPERATIONS

- No “directory” inode! Simplifies locking

Sym links

- Replica placement considerations

 - close to the client

 - same rack

 - diff rack

fault tolerance

1a / b / c

1a / b / d

- Implementing deletes

 - Metadata change mark it as deleted

 - lazy deletes

FAULT TOLERANCE

- Chunk replication with 3 replicas
- Master
 - Replication of log, **checkpoint**
 - Shadow master

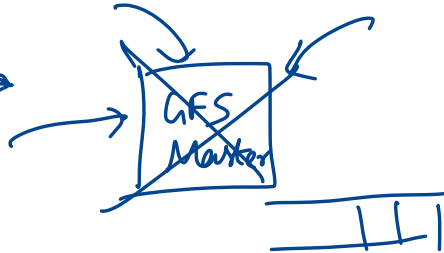
GFS Master
heart beat

Chunk server fails

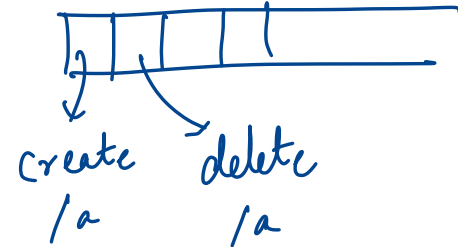
B-tree style data structure

Metadata memory

- Data integrity using checksum blocks



operation log



DISCUSSION



<https://forms.gle/Egik6VxfhHhBXsXM7>

Takeaways

1 client network server chunk

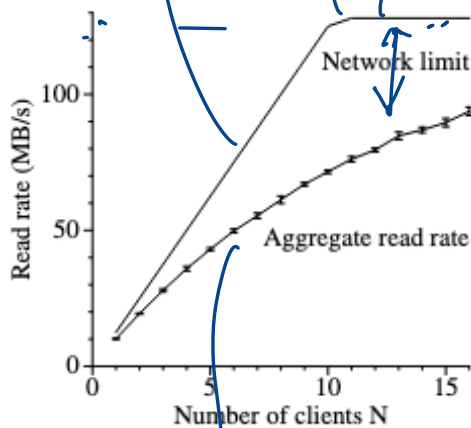
Switch

hit 3 replicas

short of

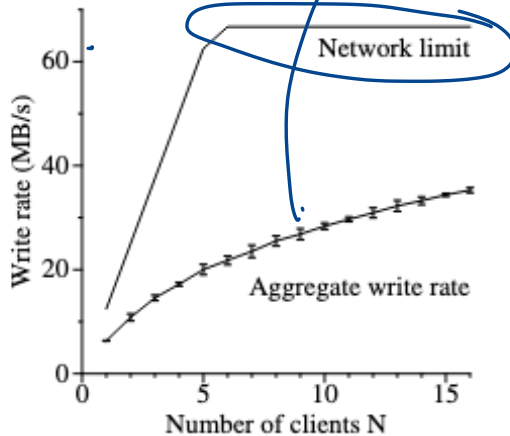
worse than reads

12.5 MB/s ~ 100 Mbps

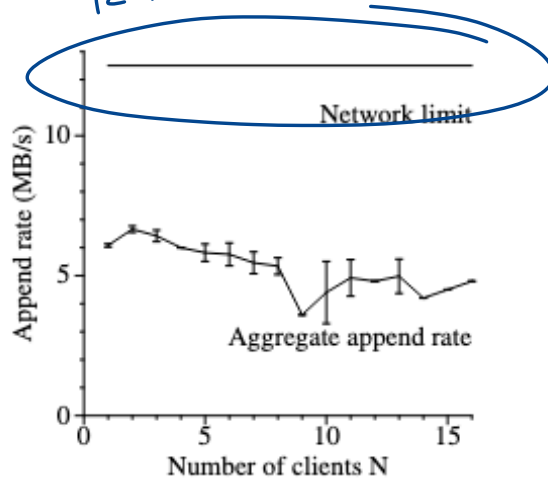


(a) Reads

scales linearly



(b) Writes



(c) Record appends

The evaluation (Table 2) shows clusters with up to 180 TB of data. What part of the design would need to change if we instead had 180 PB of data?

→ 64 MB chunksize

↳ lots of metadata →

Metadata

does not fit

in Master

memory

256 MB chunksize

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



*distributed
KV store style system
atop GFS*

Recursive structure ? If Metadata is $\sim 1/10000$ the size of data

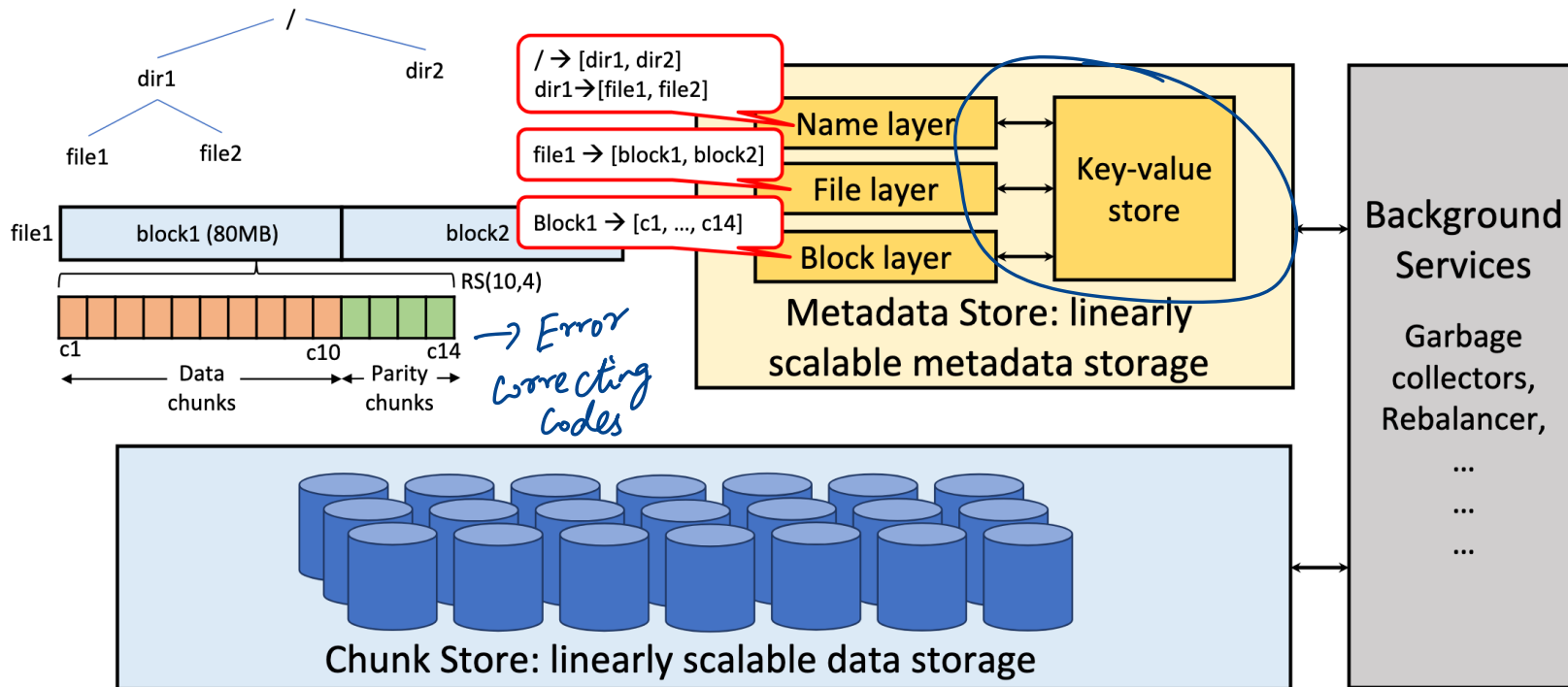
100 PB data \rightarrow 10 TB metadata

10TB metadata \rightarrow 1GB metametadata

1GB metametadata \rightarrow 100KB meta...

META: TECTONIC

Scalability: Support Exabyte Scale Clusters



NEXT STEPS

- Assignment 1 out tonight!
- Next up: MapReduce, Spark