Google FS (GFS)

Goal: support Google workloads
=> contrast to "general purpose" file system

differences?
- no latency concerns (scalable bandwidth)
- no POSIX legacy apps (they're also writing apps)
- only certain W/L charac.
  - e.g. large files, streaming

Assumptions

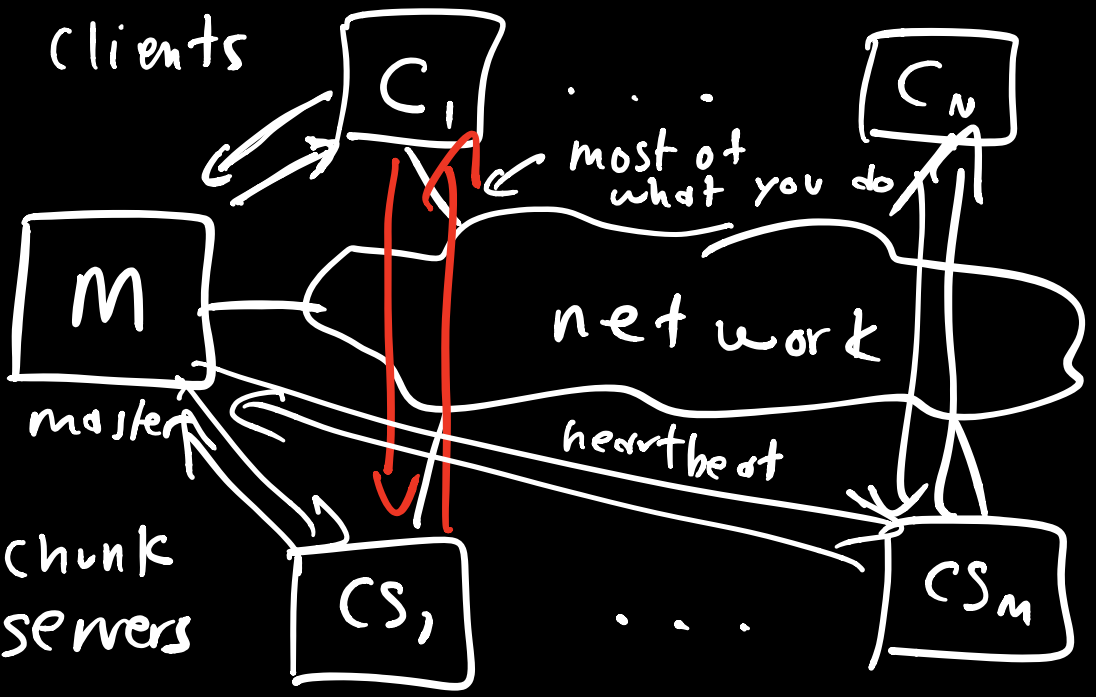
failure: common
"few" huge files (not many small)
large streaming reads
seq writes: esp. appends
bandwidth (not latency)
=> Design Decisions

<= e.g. no need for client caches

Interface : (to clients)
looks like normal "FS"
open / read / etc.
[append : commonly used]

General Arch

radical: single master
(single point of failure)
[follow-on: Colossus
adopted "multi master"]



roots: CMU NASD '98

File: ≥1 chunks

each: has ID (64-bit)
immutable
(64MB) (scale)

Chunk servers (3 replicas)

=) read/write of chunks (offset, range)

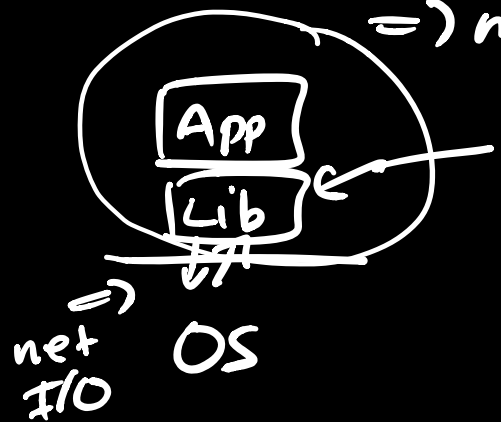
=) on top: Linux FS

advantages:
- easier (leverage)
- user-level service
- abstraction of local file
  chunk → file
  (64MB) supports "sparseness"

client: normal "fs" =) syscalls
GFS => client library

client lib => RPCs

=> master => CS's



=> no data caches

master: (centralized)

all metadata => all decisions

Chunk size: 64 MB

=> [no internal frag
=> local FS]

could have smaller block instead

but... problems:

- more metadata (@ master)
- more interaction w/ master
- more metadata (@ clients)

Problems w/ 64MB block:

=) hot spot : "small" file (< 64MB) is popular

Solution: replicate more: N ~ 20

Start @ 1:43/1:44

Admin:

=) Projects

=) one more meeting (Mon eve.)

=) Final pres.: ~20 minute 12/15 Fri or 12/20 wed

=) short writeup [5 pages]

=) Midterm Exam

take-home

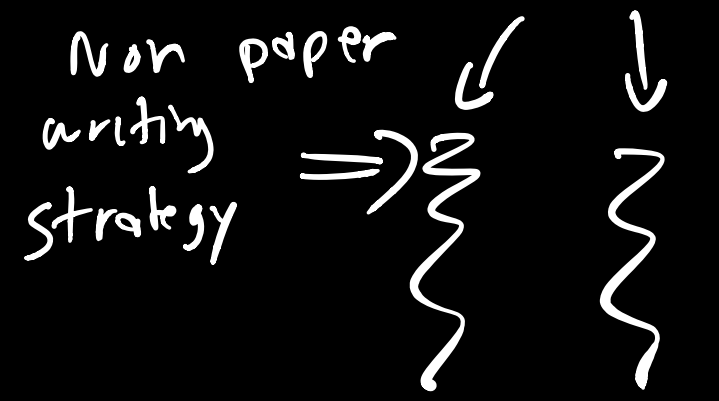
email:

[12 hr] [Sun 17th / Mon 18th]

(not collaborative)

Exam Goals: sleep

Non paper
quality
strategy



Master: Metadata => master memory

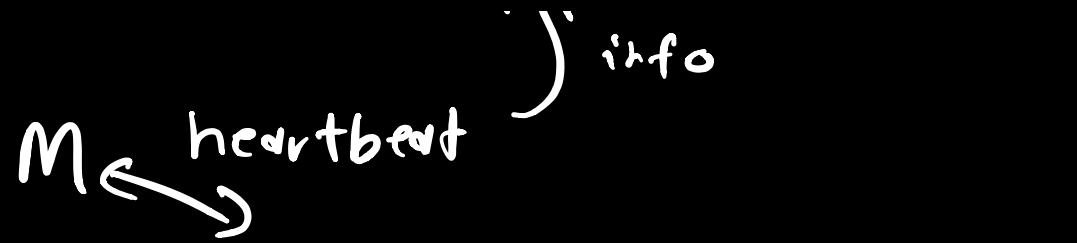
benefits: centralized, fast

Types of metadata:

- file/chunk namespaces } => oplog
- map: file -> chunk ID

- map: CID => replicas
  - stored at CS's

want: small => checkpoint periodically



Consistency mutations

Namespace: => master

mutate: easy (locks, do it)

=> atomic

File Data:

consistent: all clients see same data (regardless of replica read)

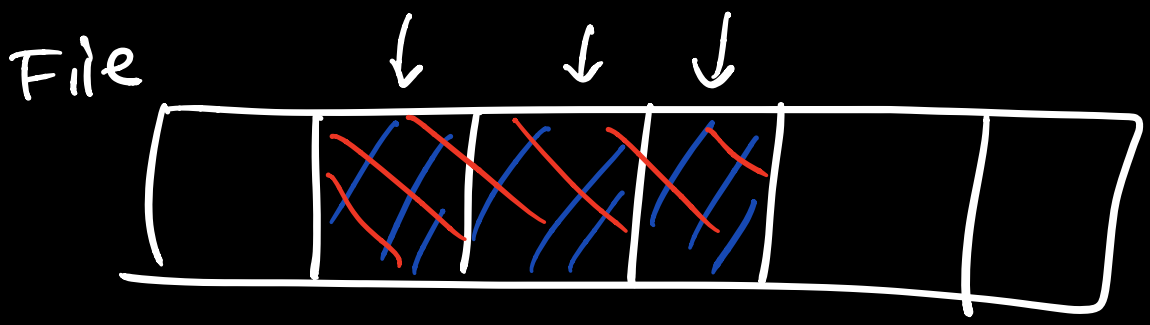
defined:

consistent and

see all results of entire client mutations

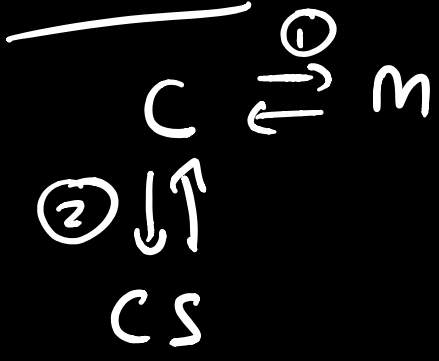
e.g. "defined"

P_1 write$_1$ ← either in entirety → P_2 write$_2$



	writes	appends
serial success	defined (cons.)	defined (but interspersed o/ inconsistent)
conc. success	cons. (undefined)	
failure	inconsistent	

Reads :



Mutations : writes / Appends

writes: use primary / backup
(serialize)
lease : master → CS

Flow : 3 phase

1) C ⇄ M metadata

2) C → CS, CS2, CS3 : data mumt

ack

pipeline not 1 → many

C → CS1 → CS2 → CS3

C ⇒ CS1, CS2, CS3

3) C → CS primary → CS sec (in order) → CS sec

C ← ack

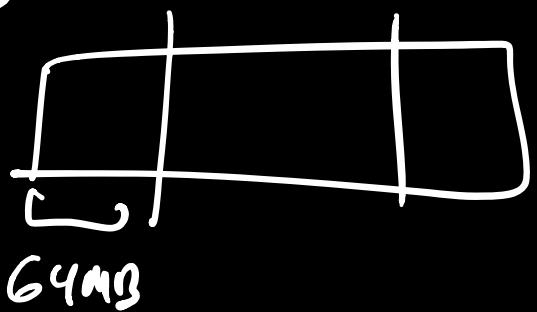
notes:

=> write can fail

=> retry

fail => inconsistent

=> big requests => undefined



Appends:

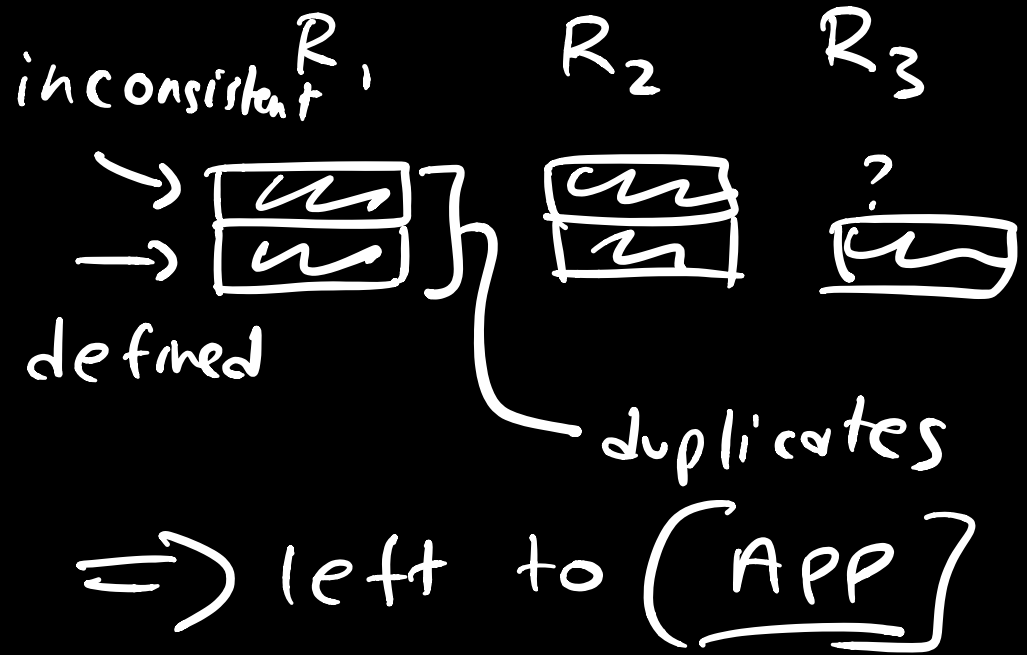
~ writes:

except:

primary: has to decide which offset to write

↓

secondaries



⇒ left to [APP]