NFS

- file access over LAN. in a single organization
- the namespace of files seen by different clients is different.

NFS was designed for:-

1. Not large no of users, machines — No replication F.T
2. Single admin domain/org — but no specific org
3. LAN (slow LAN) — caching is important
4. Most shared files only have concurrent read access — consistency is "not that much important"
5. Latency is important

GFS:-

Breaking into chunks — faster access

Replicate chunks — F.T

Master does not store any client state — Stateless

So, no caching as data access pattern does not have temporal locality.

So, no caching at all?

→ caches chunk server locations

→ Master not only sends the chunkserver asked for but also a few next ones (Prefetching)

Single Master

Pr. 1. Too many client requests
— reduced by caching/prefetching at client

2. Too much state info
— Do not store any client state
— Store reduced chunkserver states
— only locations
— Only store metadata (accessed less anyway)

3. Disk access
— Very little

Meta Data:-

File Namespace — Same tree as we know

Append - GFS Guarantee :-

— On success, will be there in its entirity atleast once.

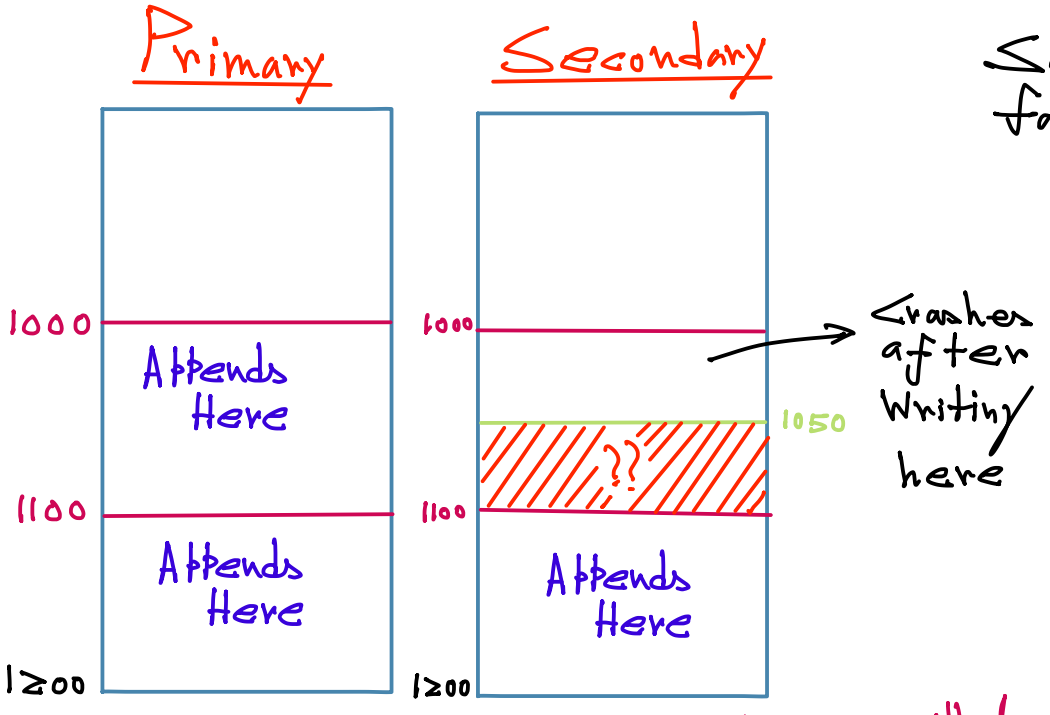
Append Op:-

Same as Write. But difference,

Primary doesn't just choose an order.

— Primary itself writes starting from current end of its file (say byte 1000 onwards)

— Asks everyone to write byte 1000 onwards



So, client gets failure, retries

Primary decides to write from 1100

Asks secondary to append at 1100 success

GFS does not gurantee replicas will be bytewise same

SOSP Hall of fame Papers

Spanner

What do we want & what do we know how to do them

1. A standard database with transactions

Providing Transaction Semantics

— Eases programming

— Non-distributed transactions

— Locking

— Distributed transactions

How does Spanner solve it?
- Two phase commit
- Pr. Blocking

2. "External" Consistency b/w R & R-W Transaction

How does Spanner do it then?
- "External" means linearizable
- Easy if we have a perfectly sync'ed global clock

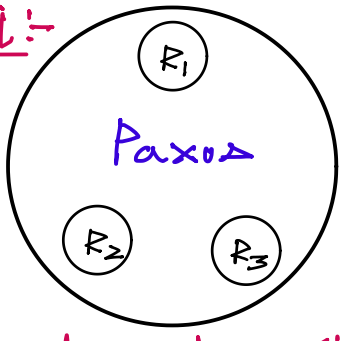
3. FT & Availability
- Replication
  - Paxos, Atomic Multicast

Universe — Created by Google

Apps create DB's in a universe

Directory / Tablets — Specified by Appl^n

Tablet i :-



"leader" implements TM

↓

Paxos Leader

↓ What if current leader is no longer leader

New Leader Elected

↓

The new Leader runs TM

So, how does the client do it?

Main Ideas

① How to use 2-PC while avoiding the bad effect of blocking?

② True Time

TT after(T)
= true if TT now.earliest > T

$TS(T_2) > TS(T_1)$

TT now.latest = 10 (say)

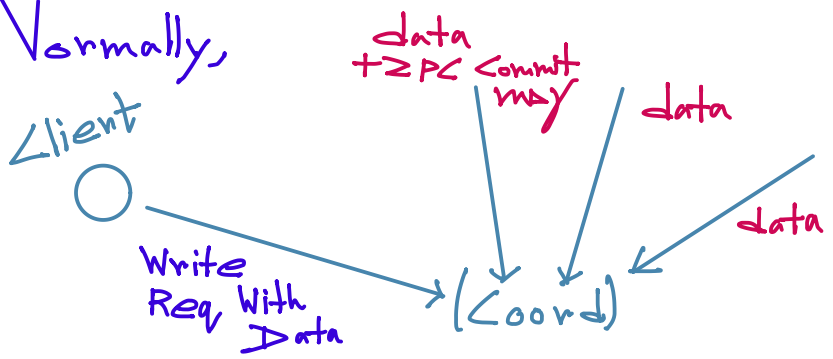
Assign S_i = 15 (say)

⟹ data is safe to read only if somebody reads at a time when 15 is guaranteed to have passed.

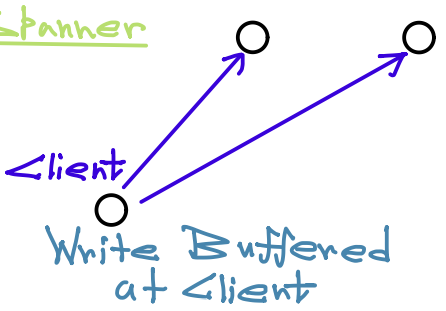
⟹ If the reader does a TT now.earliest > S_i

Safe time — Max time that guarantees no writes will happen before that.

Normally,



Spanner

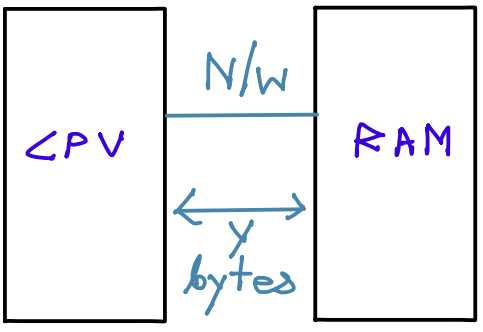


Environment:- Google's own appl^n. N/W owned by google.

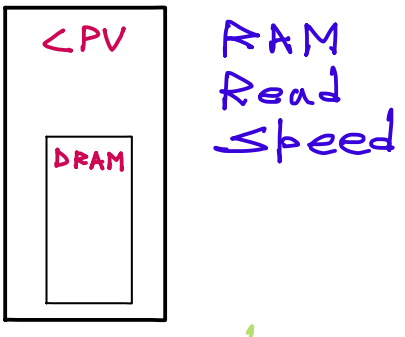
Requirements:-

- Global
- Strong Consistency
- User apps needs vary so user-configurable

Memcached



Y × RAM speed + N/W Access Servers



Y × bytes/sec

Q) Does Memcached Scale?

— Thousands of services accessing thousands of servers with large no of DBs.

— Hundreds of memcached servers

Challenges:-

① Each client may need hundreds of keys at a time. & some keys may be dependent on value of others.

② One large pool of memcached servers for all services. So, all services access all memcached servers potentially, causing heavy traffic. — all-to-all

Facebook Paper

1. Reducing Latency

(A) When Key is in Memcached

Three things

① DAG to optimize # of rounds

② Use UDP to Get, TCP for Set/Del
↳ can give very large no of TCP connections
— Very Memory Intensive

Sol^n
— Coalesce connections from one Webserver to Memcached Server

③ Large no of requests can still cause problem in switches or overload servers

Sol^n
— Do Sliding Window flow control.
Window — No of Requests sent.

(B) When Key is not memcached
(How to reduce the number of times you go to DB)

(B) How to reduce cache misses?

lease — Kind of a timestamp/version on the data given by the DB to the client when it writes
— Checked by memcached servers, invalid if Version lower

Thundering Herds :-

WRRRRWRRRR
Very Fast before the write is written to the cache.

Memcache pool :-