Cloud Storage Systems

CS 6410 - Fall 2019

Midhul Vuppalapati
(mvv25@cornell.edu)

History of Distributed Storage Systems

Networked File Systems

- Network File System (NFS), Sun, 1986
- Andrew File System (AFS), CMU, 1988

- **Goal**: Make files accessible over network
- Client/Server architecture
- **Q**: How to reduce network transfers?
  - **A**: Client-side caching
  - Consistency Model

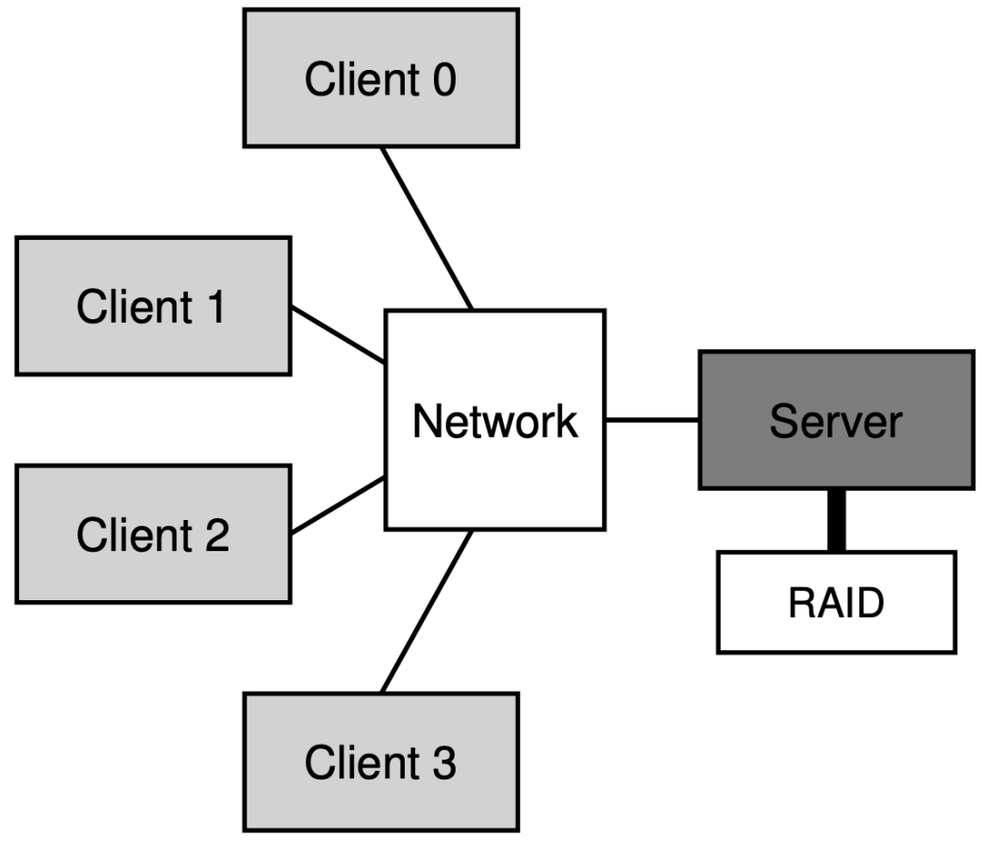


Figure taken from "Operating Systems: Three Easy Pieces"

“Distributed” File Systems

- NFS/AFS - Given sub-tree of namespace managed by single server
    - Can’t store files larger than the capacity of single server
    - Server down => all files owned by it unavailable
    - Capacity under-utilization
- **Ideally**: File should not be bound to single machine
- “Serverless” Distributed File Systems
    - **xFS** [SOSP ’95], **Petal** [ASPLOS ’96] + **Frangipani** [SOSP ’97]
        - File data + metadata split across multiple machines
        - Redundancy across machines
        - Recovery + load balancing transparently handled

P2P Storage Systems

- Designed for peers over WAN
- Napster [1999]
- Distributed Hash Tables (DHTs)
  - CAN, **Chord**, Pastry, Tapestry [2001]
- Distributed Storage based on DHTs
  - **Cooperative File System (CFS)** [SOSP'01]
  - **Pond** [FAST'03]
    - Untrusted peers (possibly malicious)
    - Join/Leave (churn)

Figures taken from the public Internet

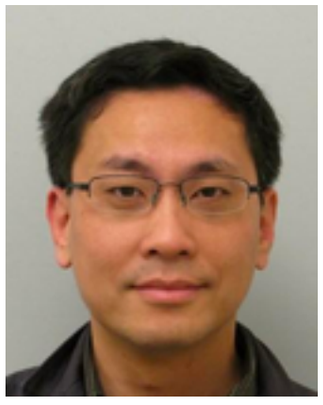
Google File System (GFS)

SOSP 2003

About the Authors

- Sanjay Ghemawat
  - ***Undergrad at Cornell!***

- Howard Gobioff



- Shun-Tak Leung

Why build GFS?

- Built by Google, for Google
- **Google's use-case**: Large-scale data processing workloads
    - Data: crawled documents, search logs etc..
    - Clusters: 100s-1000s of machines w/ commodity hardware
    - MapReduce jobs (distributed computing framework)
- **Why not use existing systems?**
    - Older DFS (xFS/Petal) don't scale
    - P2P Storage systems designed for un-trusted scenario

Assumptions

- Failures are common case
- Large (Multi-GB) files are the common case
- **Read Characteristics**
    - Large contiguous sequential reads (1MB+)
    - Small random reads (few KBs) at arbitrary offsets
- **Write Characteristics**
    - Large sequential appends (possibly concurrent) are common case
    - Random writes are rare
- Bandwidth more important than latency

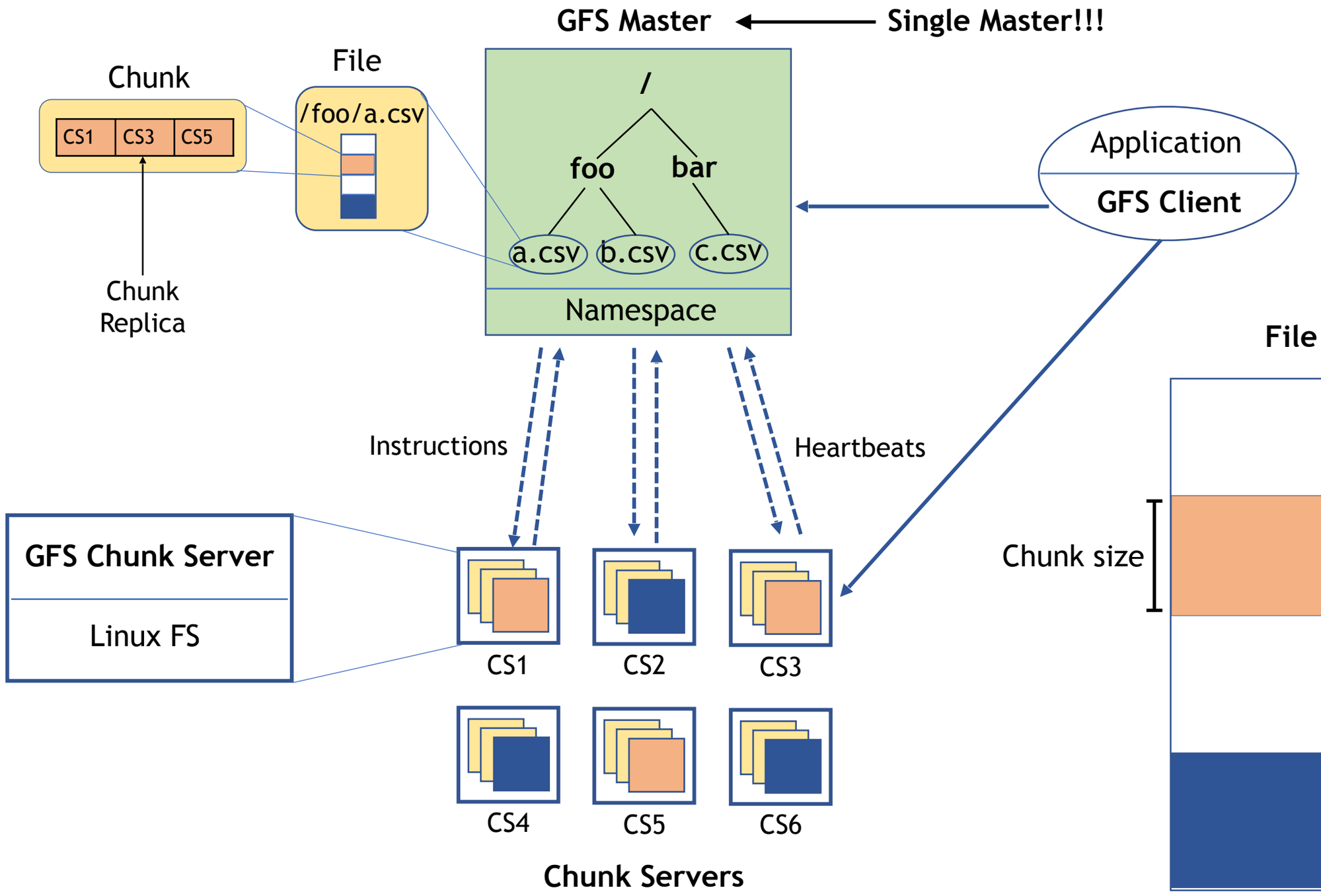
Interface

- Familiar file system interface
  - *create, delete*
  - *open, close*
  - *read, write*

- Additional operations
  - *Snapshot*
  - *Record Append*

- Not POSIX compliant

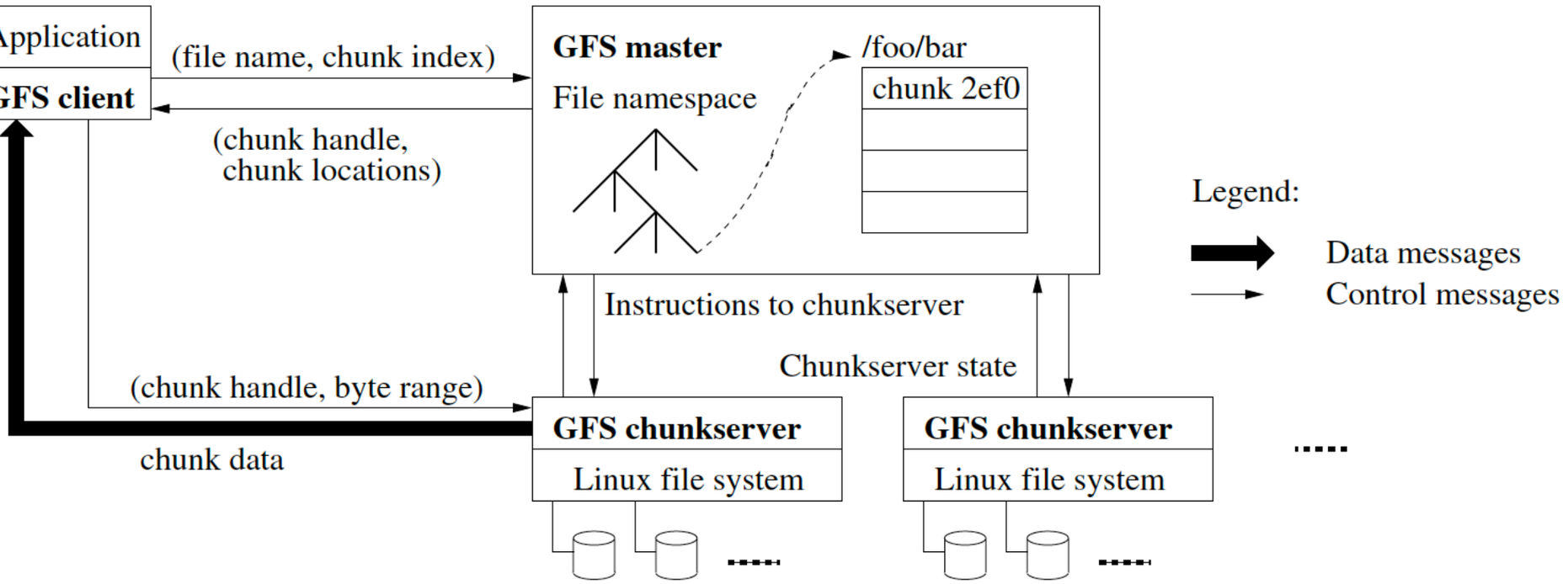
Architecture



Chunk size

- **Chunk-size = 64MB**
  - Significantly larger than other file systems
    - Linux FS block size: 4KB
    - xFS / Petal: 64KB
  - Based on assumption that large files are common case
- Reduces metadata footprint at Master
- Internal fragmentation ?
  - Lazy space allocation for files on chunkservers.

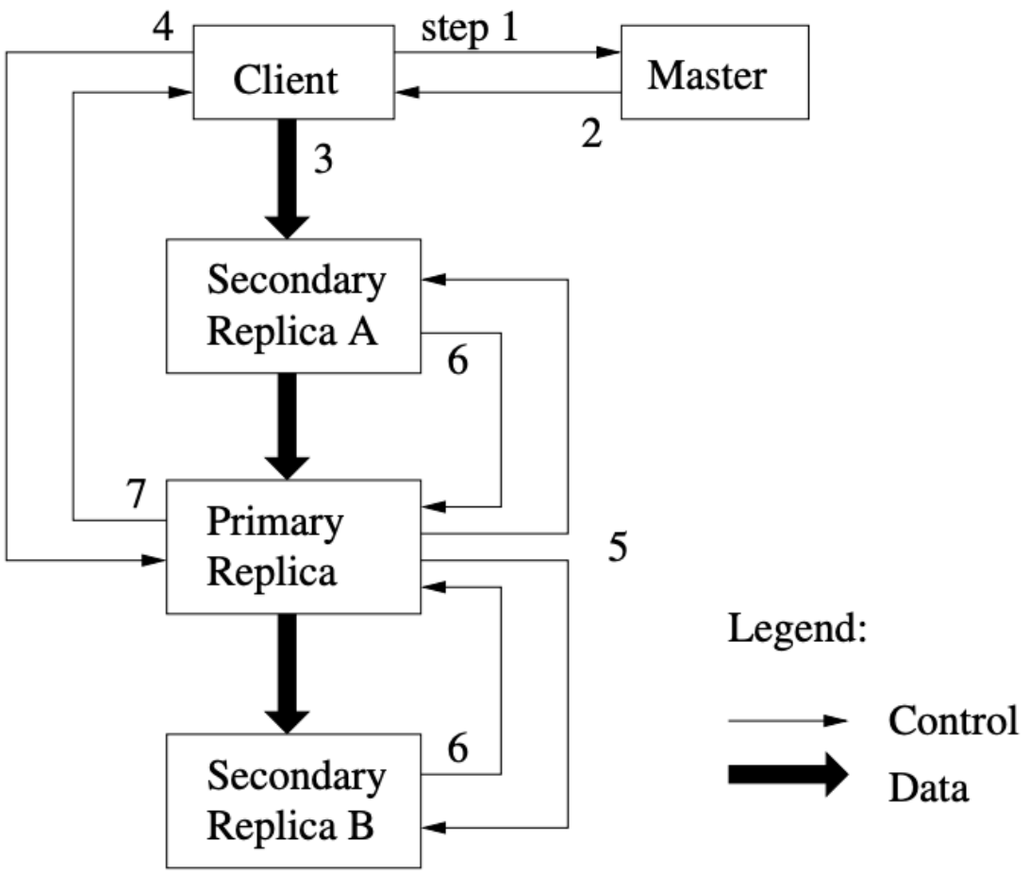
Read Path



- Clients cache chunk locations
- **No client-side data caching!**

Figure taken from GFS paper [SOSP'03]

Write Path

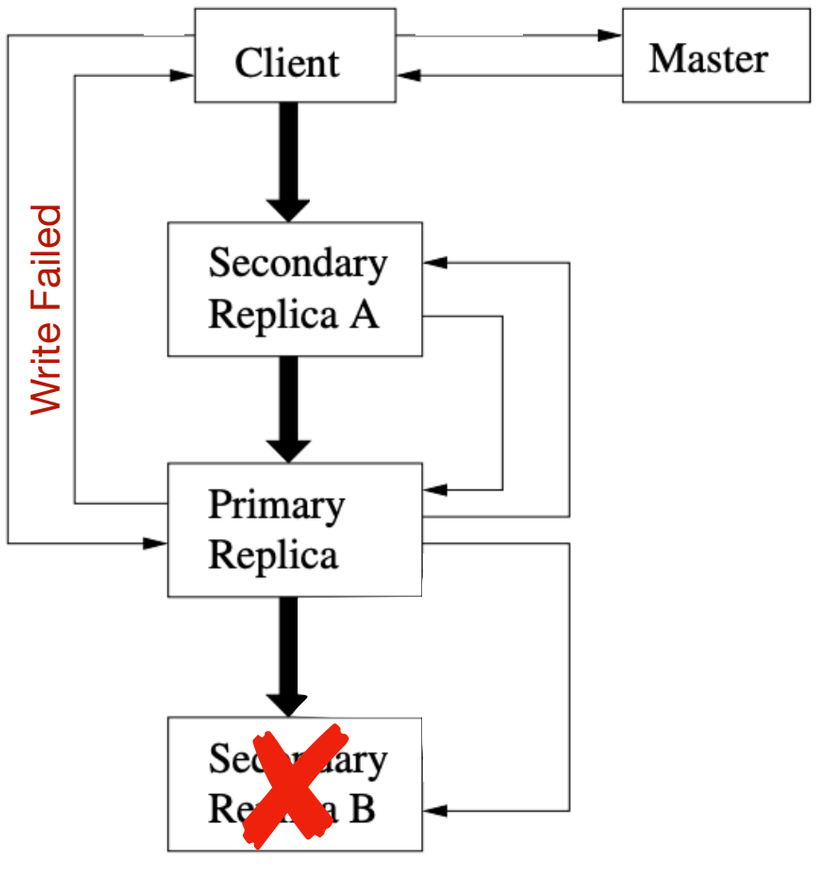


1. Client contacts Master
2. Master replies w/ location of primary and other replicas
3. Client pushes data to all replicas; replicas buffer received data
4. Client sends write request to primary; primary serializes requests
5. Primary applies request, forwards to secondaries in serial order
6. Secondaries send Ack to primary
7. Primary sends Ack to client

Figure taken from GFS paper [SOSP'03]

What can go wrong?

- Failures

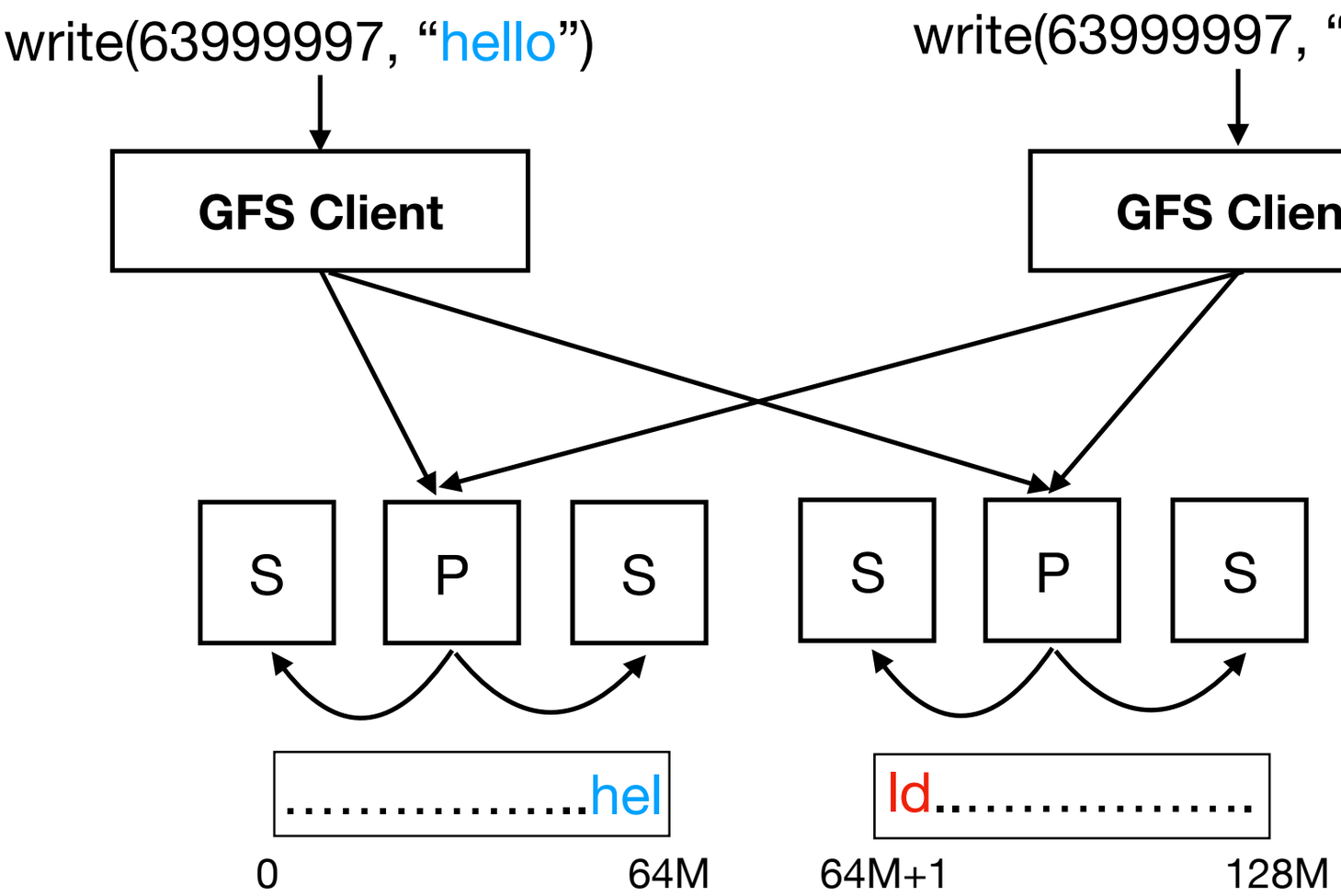


- Secondary B fails during write (transient failure)
- Write operation fails
- Primary & Secondary A have applied the write
- Region is ***inconsistent***

Figure taken from GFS paper [SOSP'03]

What can go wrong?

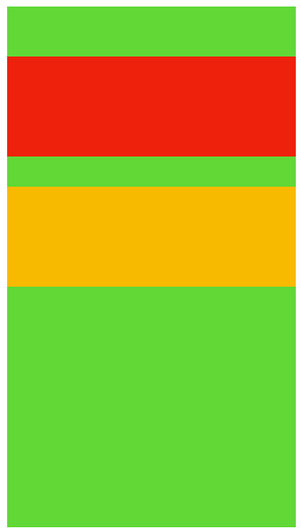
- Concurrent writes



- Data from concurrent writes can get intermingled
- *Consistent* but ***undefined***

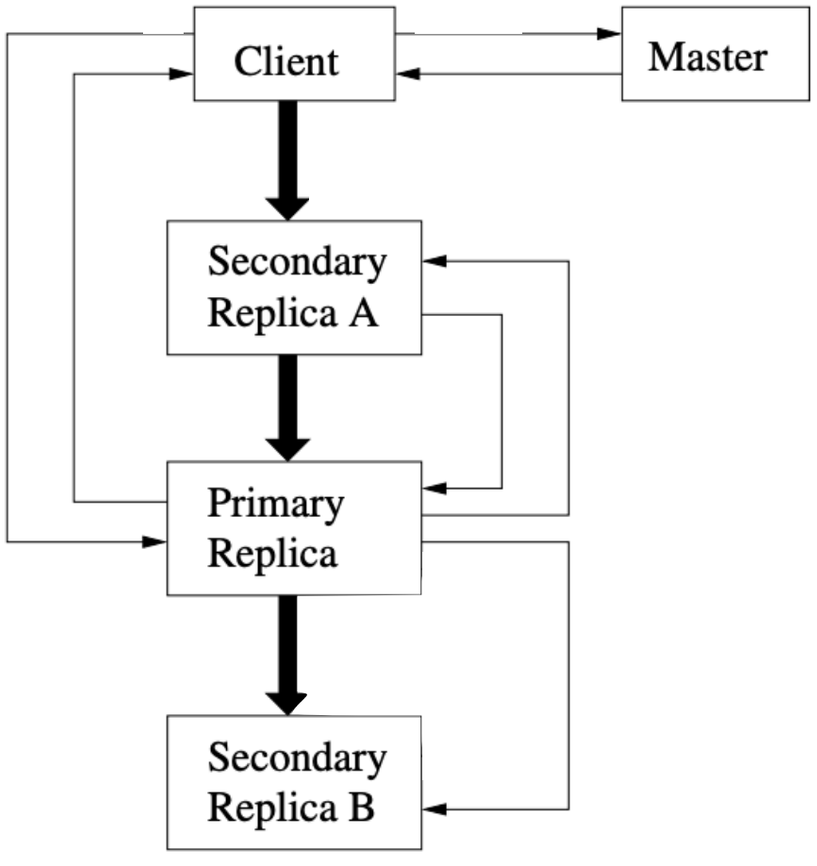
Writes - Consistency Model

- Failed write - region is ***inconsistent***
- Successful write - region is ***consistent***
  - Serial - region is ***defined***
  - Concurrent - region is ***undefined***
- Messy consistency model!



File

Record Append



- Atomic concurrent appends
- Offset is decided by GFS
- Primary checks if append can fit in chunk
  - No => pad and move to next chunk
  - Yes => write at current offset and tell secondaries to write at same offset
- Failures can lead to **duplicate records**

Figure taken from GFS paper [SOSP'03]

Master: Fault-tolerance

- Operation log + checkpoints replicated on multiple machines
- Failure => New Master reads checkpoint and replays operation log
- “shadow” masters provide read-only access
    - May lag behind Master

Master: Chunk Placement

- Place chunk replicas on different racks
    - Better fault-tolerance
    - Use aggregate b/w of multiple racks

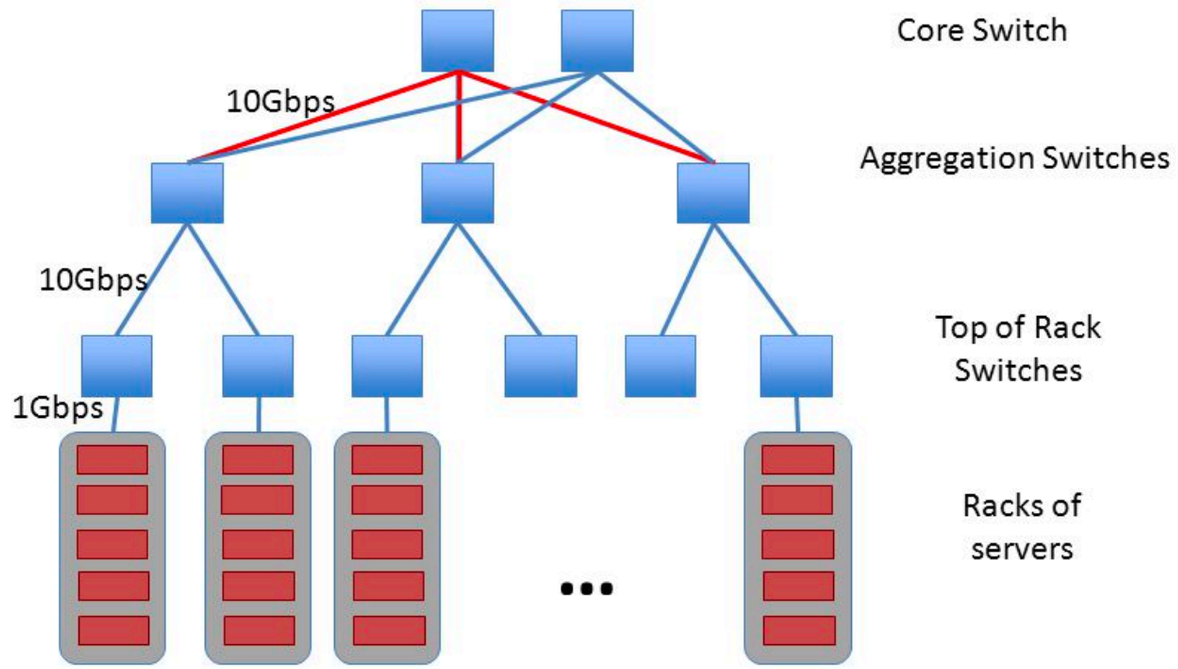


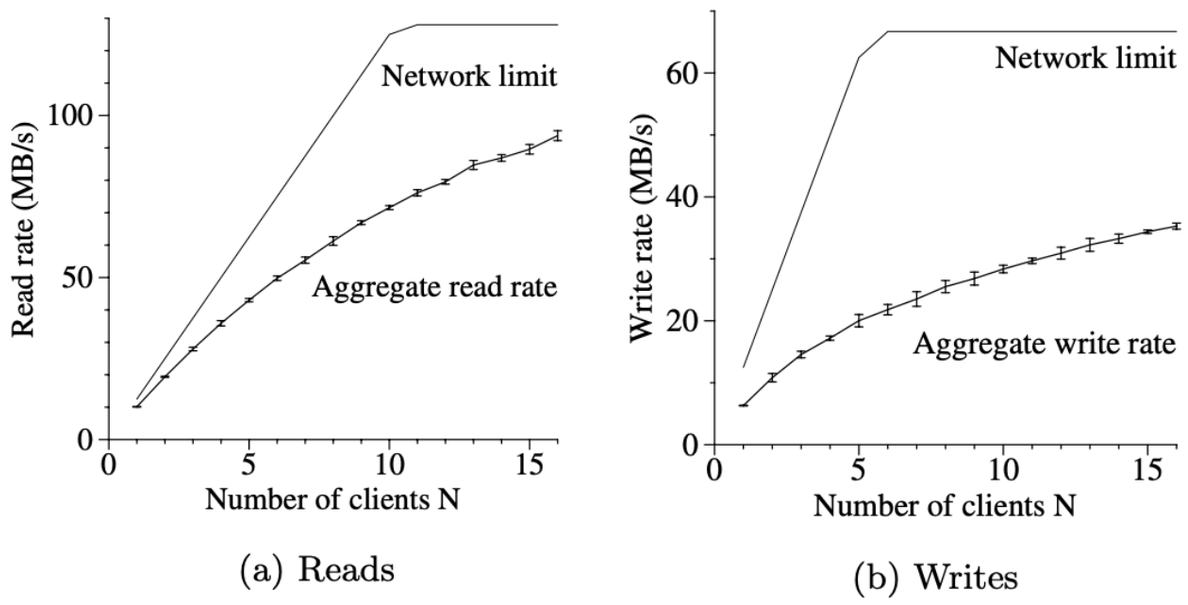
Figure taken from "Datacenter Network Topologies" by Costing Raiciu

Master: Other operations

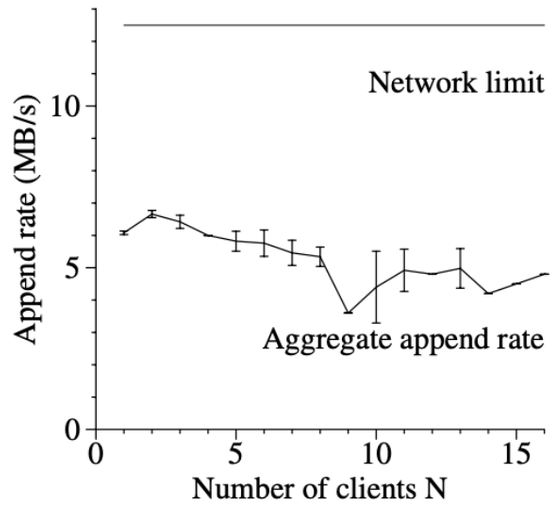
- Re-replication
  - Whenever # of available replicas < replication goal (3)
- Rebalancing
  - Periodically moves replicas for better load balance
- Garbage collection
  - Files & chunks are deleted asynchronously

Evaluation

- Microbenchmarks
- Real production clusters
  - Metadata overhead
  - Read/Write throughput
  - Master load
  - Recovery time
- Real workload characteristics



(a) Reads (b) Writes



(c) Record appends

Plots taken from GFS paper [SOSP'03]

Impact of GFS

- Inspired **Append-only** file systems
    - Ex: HDFS, Microsoft's COSMOS
    - No support for byte-offset writes, only appends (cleaner)
- SIGOPS Hall of Fame Award 2015
    - **Most influential systems paper in last 10 years**
- Scale of today's datacenter much larger
    - 10,000-100,000 machines
    - **Single master does not scale**
    - COSMOS: Partitions namespace across multiple Masters
- Tidbit: Recent GDPR policy invalidates append-only assumption

Thoughts?

Erasure Coding

- GFS uses **replication** for fault tolerance
    - Storage overhead of 3x
- **Erasure Coding**: Alternate technique for redundancy
    - Fault tolerance with lower storage overhead
- Simple example: XOR-based Parity

- In practice: (10,4) Reed-Solomon Codes

Recovery Cost

- **Key-challenge**: Recovery cost
  - Re-constructing one block => read multiple blocks over network
    - Disk I/O, Network I/O, Compute overhead
- Lot of work on reducing recovery cost
  - LRC Codes, Windows Azure Storage [ATC’12]
  - Hitchhiker [SIGCOMM’14]

Spanner [OSDI'12]

- Geo-distributed storage system
- Supports strong consistency guarantees
- Implements Paxos across replicas for strong consistency
- Implements Two phase commit across partitions to support transactions

TrueTime API

- In GFS: Lease management is time-based
  - *Clock skew across machines could cause issues!*
- Spanner [OSDI'12]
  - Uses ***TrueTime*** API for lease management
    - Implemented using GPS & Atomic clocks in Google clusters
    - Time returned not absolute, but *interval*
    - Truly Non-overlapping Paxos leaders

References

- Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." (2003).

- Sandberg, Russel, et al. "Design and implementation of the Sun network filesystem." Proceedings of the Summer USENIX conference. 1985.

- Howard, John H., et al. "Scale and performance in a distributed file system." ACM Transactions on Computer Systems (TOCS) 6.1 (1988): 51-81.

- Anderson, Thomas E., et al. "Serverless network file systems." ACM Transactions on Computer Systems (TOCS) 14.1 (1996): 41-79.

- Lee, Edward K., and Chandramohan A. Thekkath. "Petal: Distributed virtual disks." ACM SIGPLAN Notices. Vol. 31. No. 9. ACM, 1996.

- Thekkath, Chandramohan A., Timothy Mann, and Edward K. Lee. "Frangipani: A scalable distributed file system." SOSP. Vol. 97. 1997.

References

- Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160.

- Dabek, Frank, et al. "Wide-area cooperative storage with CFS." ACM SIGOPS Operating Systems Review. Vol. 35. No. 5. ACM, 2001.

- Rhea, Sean C., et al. "Pond: The OceanStore Prototype." FAST. Vol. 3. 2003.

- Huang, Cheng, et al. "Erasure coding in windows azure storage." Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12). 2012.

- Rashmi, K. V., et al. "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers." ACM SIGCOMM Computer Communication Review. Vol. 44. No. 4. ACM, 2014.

- Corbett, James C., et al. "Spanner: Google’s globally distributed database." ACM Transactions on Computer Systems (TOCS) 31.3 (2013): 8.