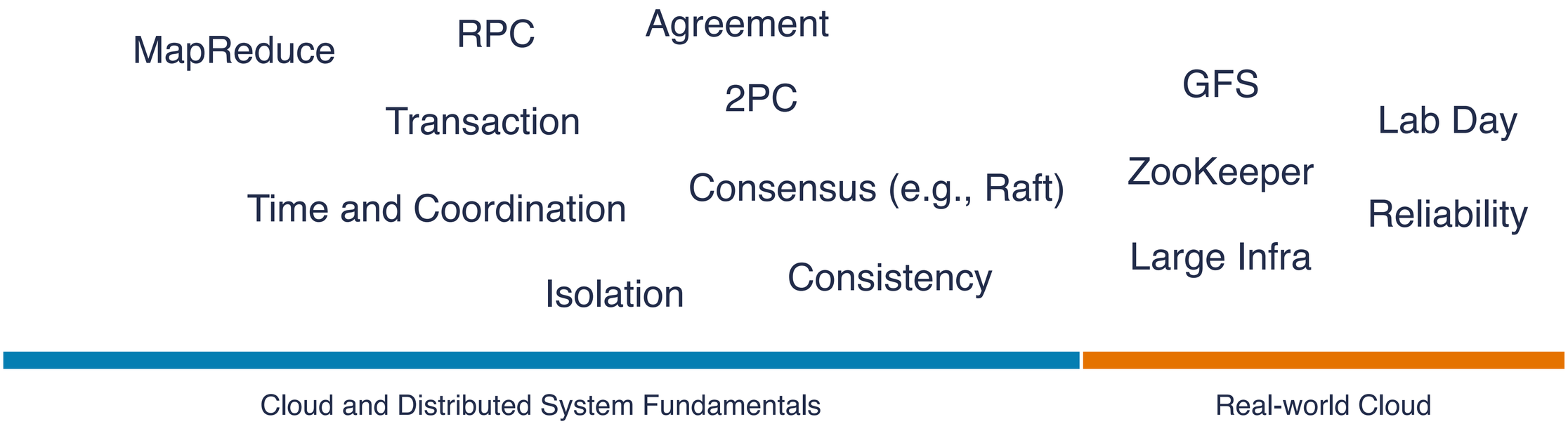
UNIVERSITY of VIRGINIA

CS4740
CLOUD COMPUTING

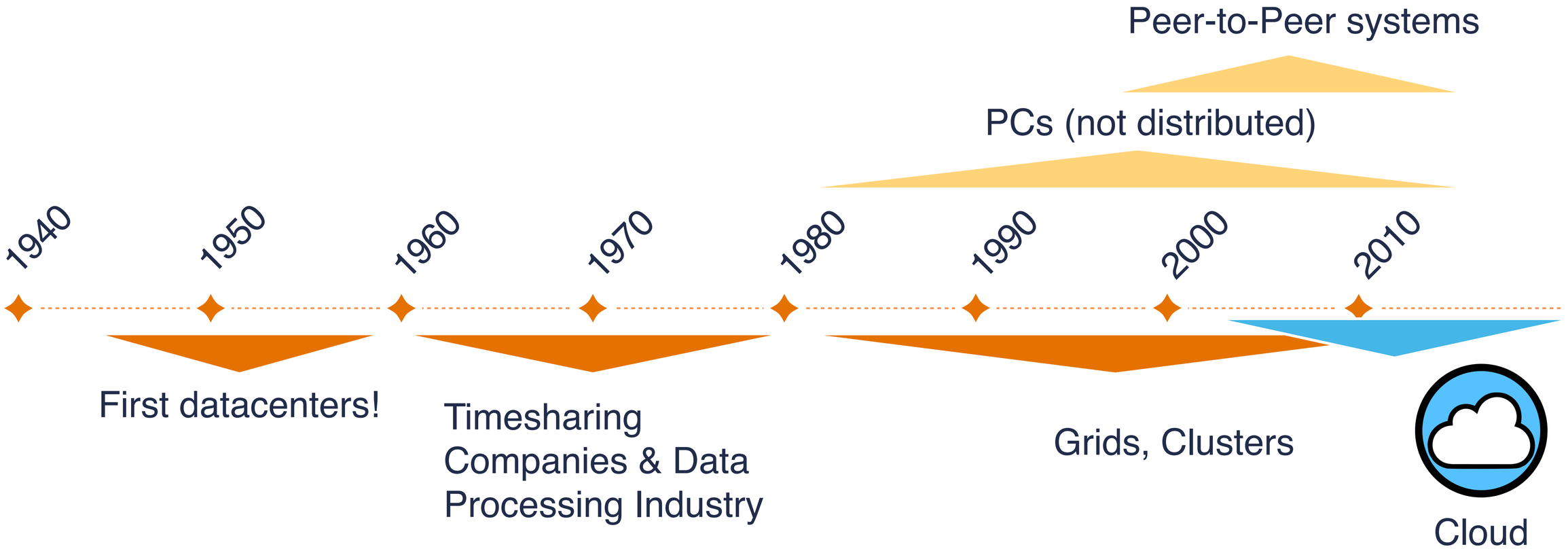
Case study: Google File System

Prof. Chang Lou, UVA CS, Fall 2026

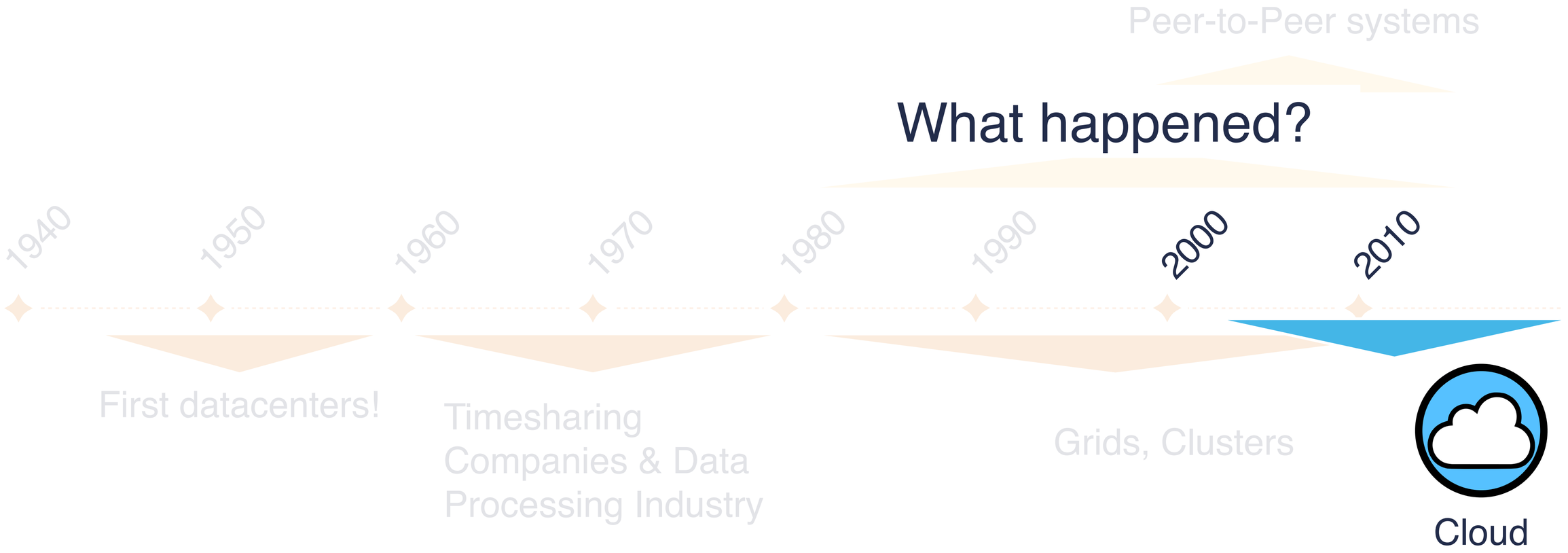
COURSE TOPICS



A CLOUDY HISTORY OF TIME

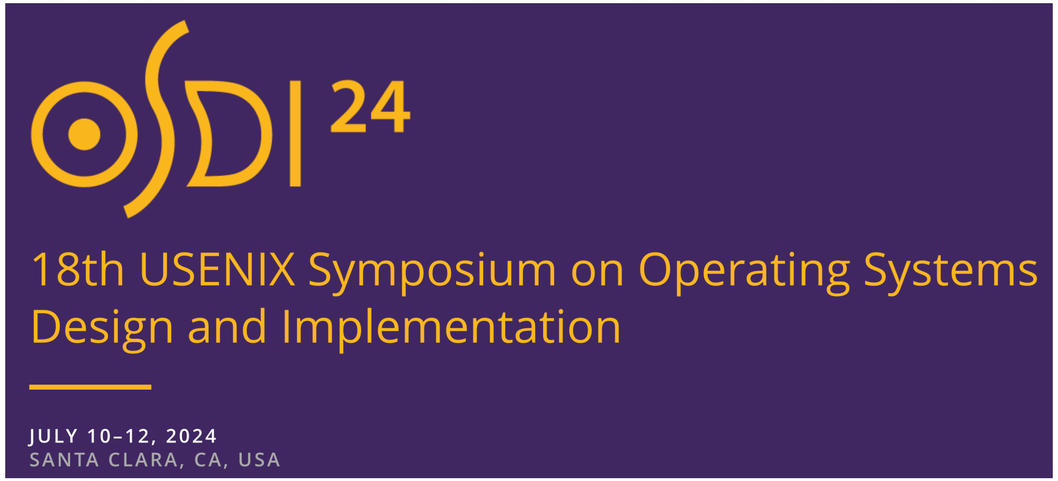


A CLOUDY HISTORY OF TIME



TOP SYSTEM CONFERENCES





THREE GOOGLE PAPERS THAT TRANSFORMED INDUSTRY PRACTICE

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

SOSP'03

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
Google, Inc.

OSDI'04

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

OSDI'06

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

SOSP 2003

WHY ARE WE READING THIS PAPER?

- GFS paper touches on many themes of this course
  - parallel performance, fault tolerance, replication, consistency
- good systems paper -- details from apps all the way to network
- successful real-world design

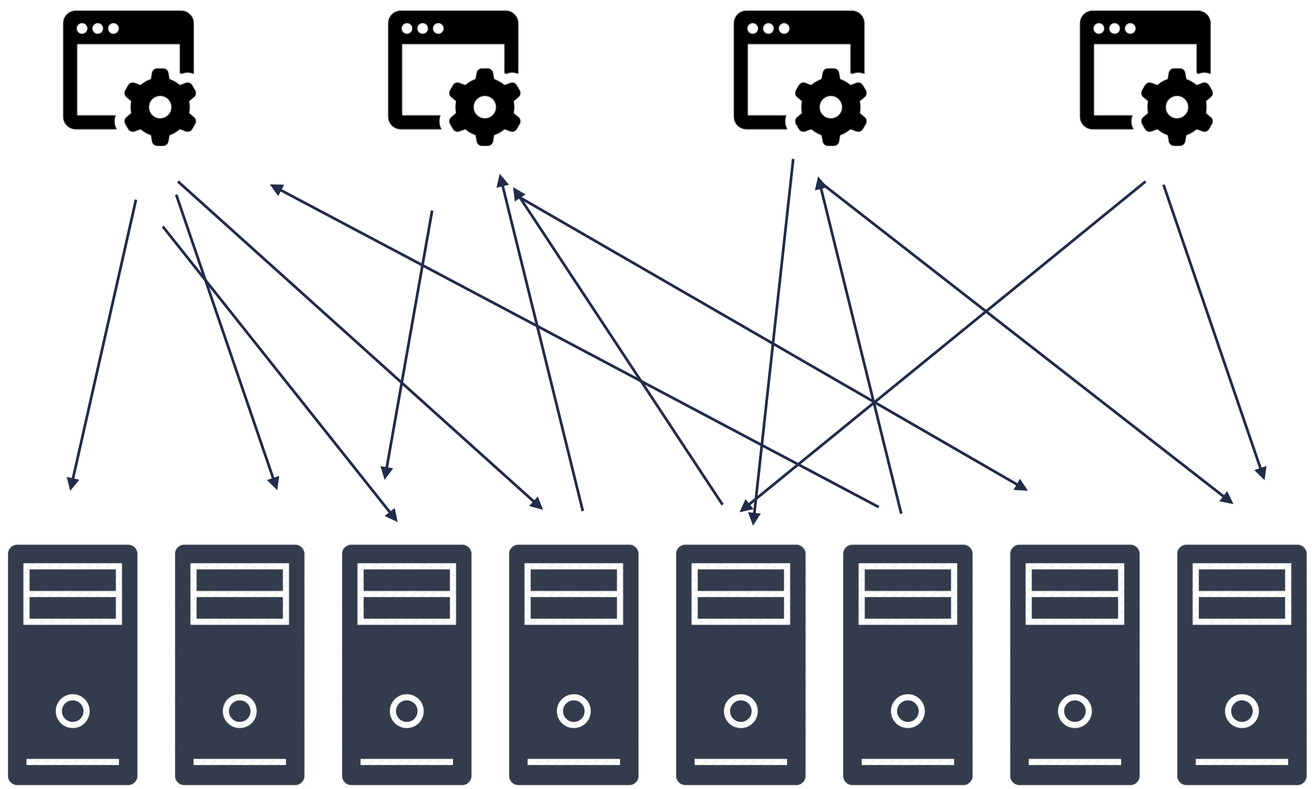
HOW TO READ AN ENGINEERING RESEARCH PAPER

by William G. Griswold, CSE, UC San Diego

- What are the **motivations** for this work?
- What is the proposed **solution**?
- What is the work's **evaluation** of the proposed solution?
- What is your **analysis** of the identified problem, idea and evaluation?
- What are the **contributions**?
- What are **future directions** for this research?
- What **questions** are you left with?
- What is your **take-away message** from this paper?

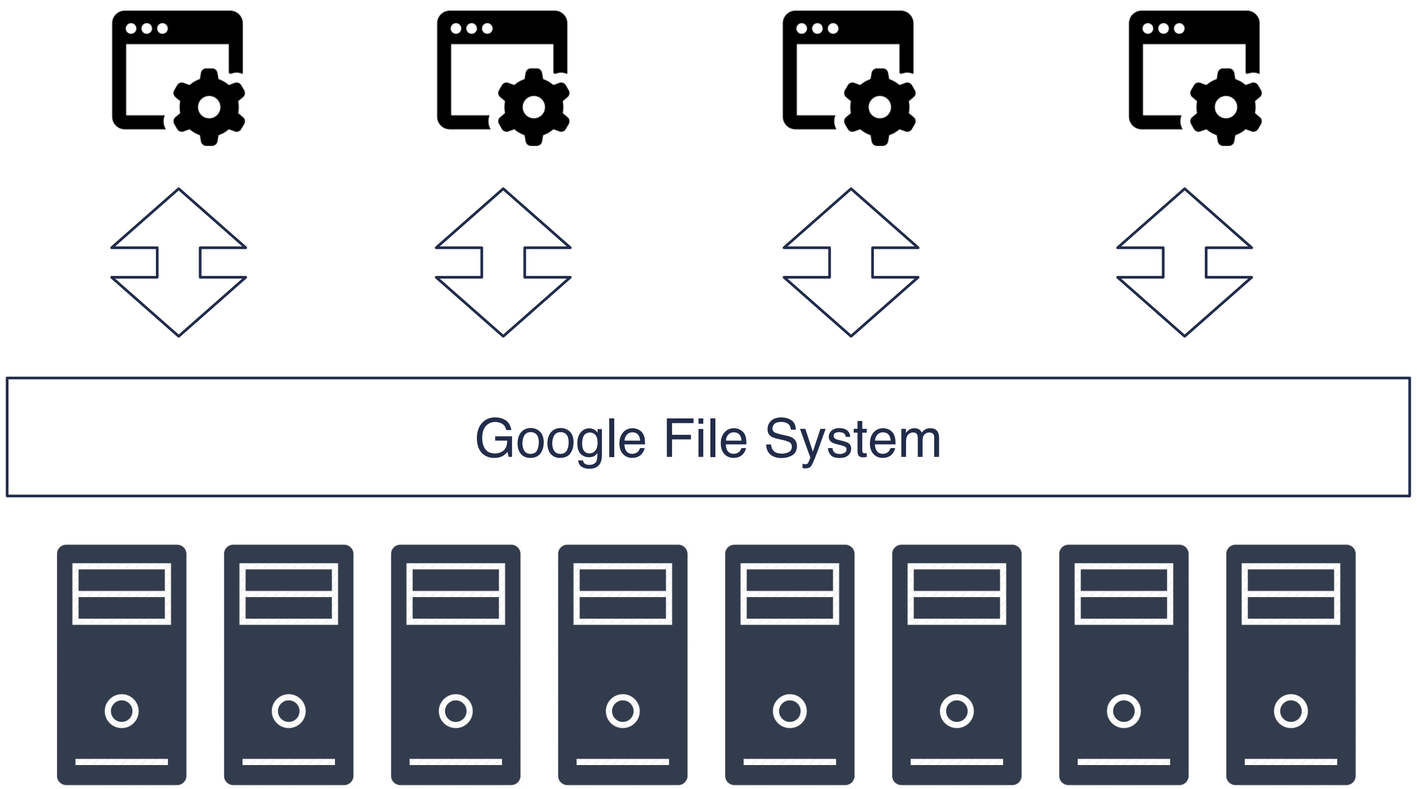
https://cseweb.ucsd.edu//~wgg/CSE210/howtoread.html

MOTIVATION



- Many Google services needed a big fast unified storage system
  - Mapreduce, crawler, indexer, log storage/analysis
- Shared among multiple applications e.g. crawl, index, analyze
- Capacity?
- Performance?
- Fault tolerance?

MOTIVATION



- Many Google services needed a big fast unified storage system
  - Mapreduce, crawler, indexer, log storage/analysis
- Shared among multiple applications e.g. crawl, index, analyze
- Capacity?
- Performance?
- Fault tolerance?

GFS OVERVIEW

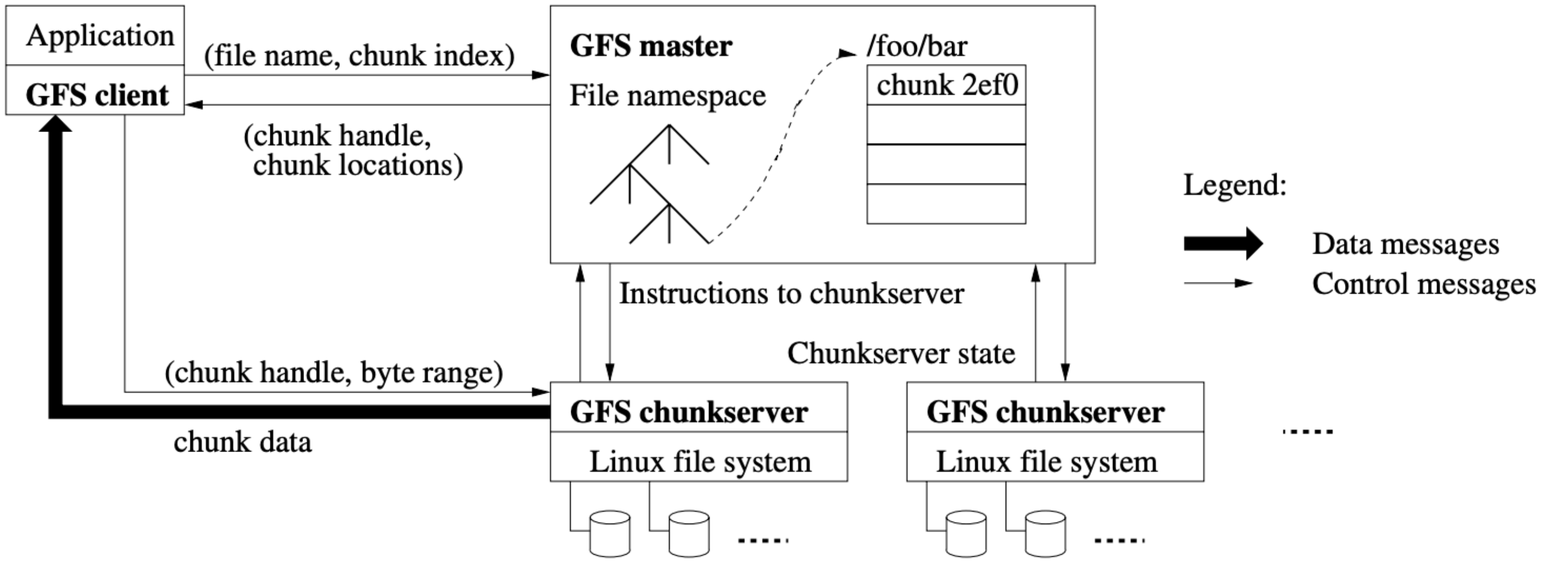


Figure 1: GFS Architecture

- 100s/1000s of clients (e.g. MapReduce worker machines)
- 100s of chunkservers, each with its own disk
- one coordinator

CAPACITY STORY?

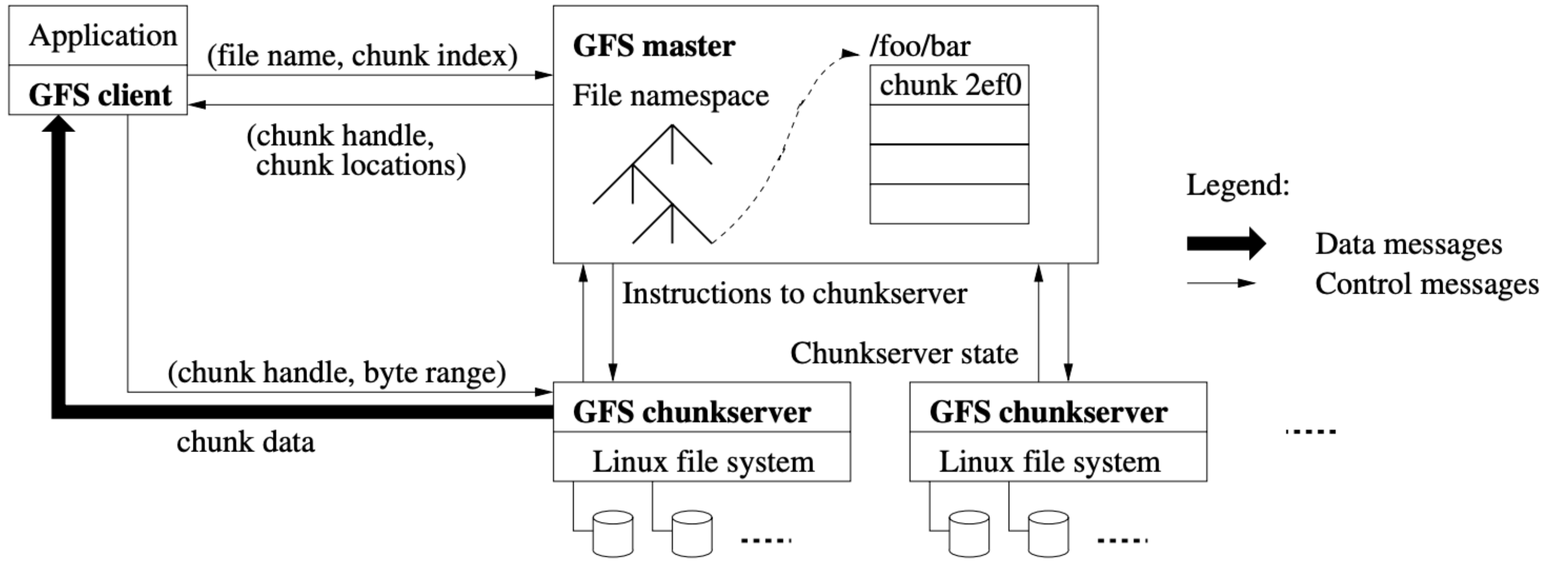


Figure 1: GFS Architecture

- big files split into 64 MB chunks
- each file's chunks striped/sharded over chunkservers
  - so a file can be much larger than any one disk
- each chunk in a Linux file

THROUGHPUT STORY?

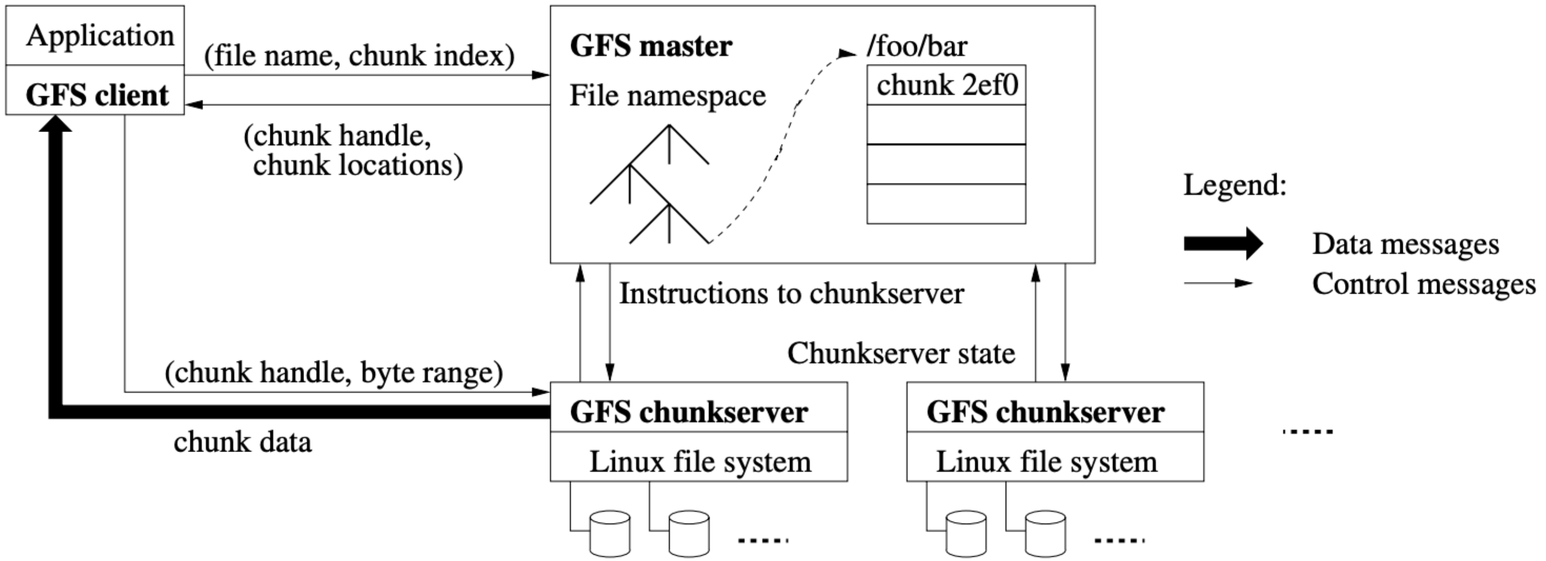


Figure 1: GFS Architecture

- clients talk directly to chunkservers to read/write data
- if lots of clients access different chunks, huge parallel throughput

FAULT TOLERANCE STORY?

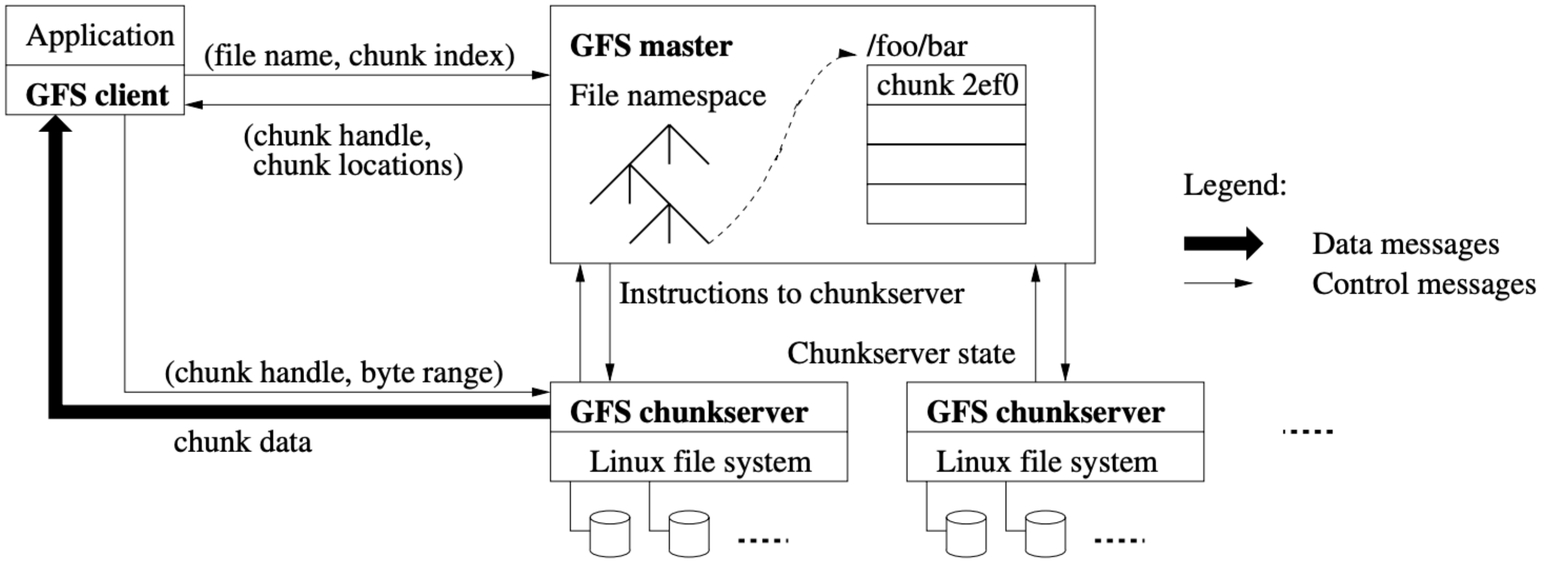


Figure 1: GFS Architecture

- each 64 MB chunk stored (replicated) on three chunkservers
- client writes are sent to all of a chunk's copies
- a read just needs to consult one copy

Basic ops (read, write..)

WHEN CLIENT C WANTS TO READ A FILE?

- 1. C sends filename and offset to coordinator (CO) (if not cached)
- CO has a filename -> array-of-chunkhandle table
- and a chunkhandle -> list-of-chunkservers table
- 2. CO finds chunk handle for that offset
- 3. CO replies with chunkhandle + list of chunkservers
- 4. C caches handle + chunkserver list
- 5. C sends request to nearest chunkserver
- chunk handle, offset
- 6. chunk server reads from chunk file on disk, returns to client

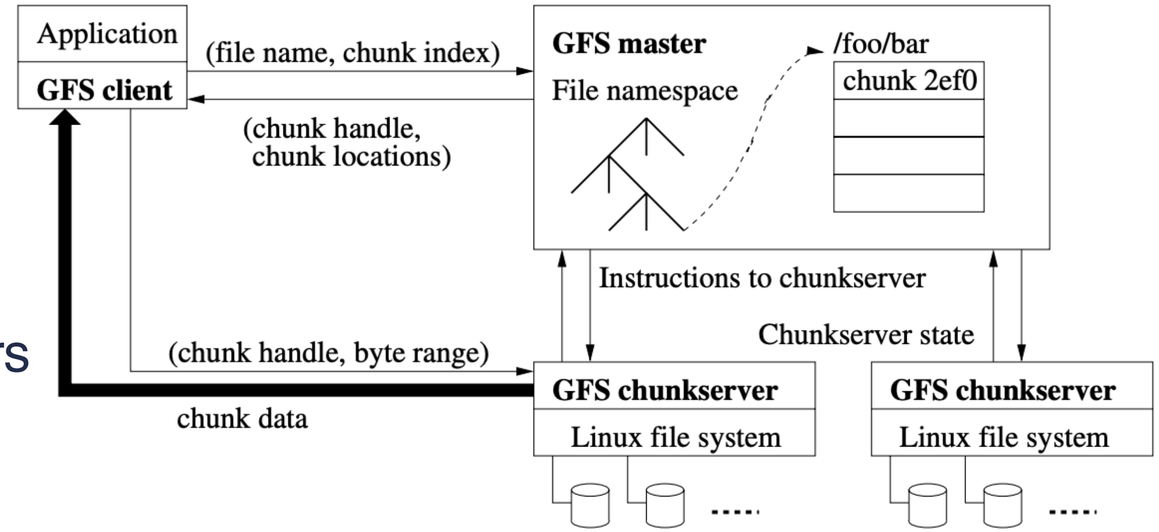


Figure 1: GFS Architecture

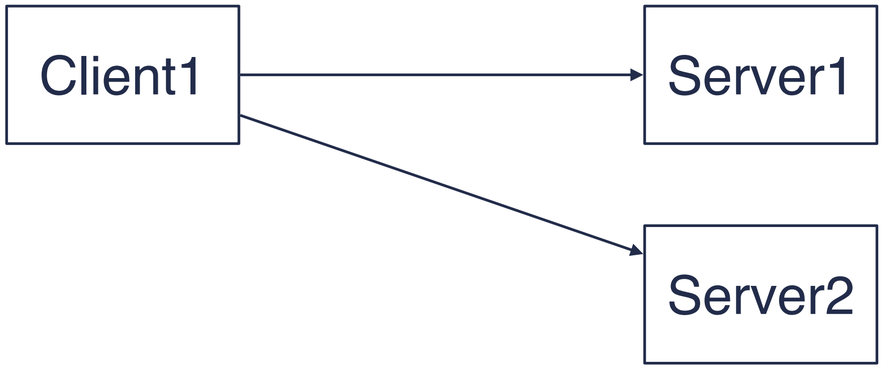
WHEN CLIENT C WANTS TO READ A FILE

- Clients only ask coordinator where to find a file's chunks
  - clients cache name -> chunkhandle info
  - coordinator does not handle data, so (hopefully) not heavily loaded

- What about writes?
  - Client knows which chunkservers hold replicas that must be updated.
  - How should we manage updating of replicas of a chunk?

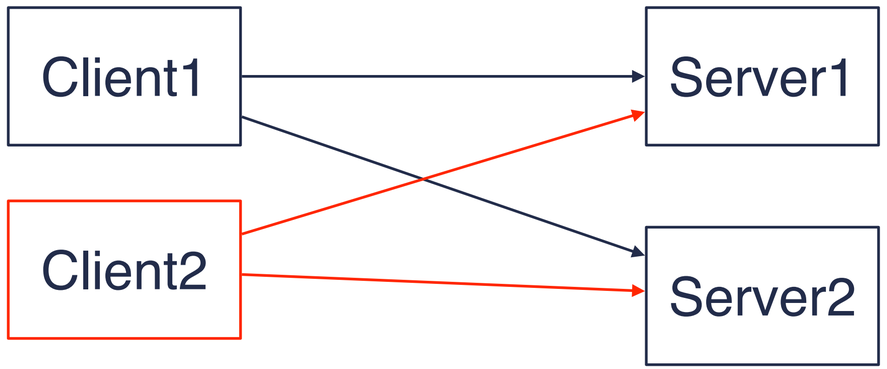
A BAD REPLICATION SCHEME

- Client sends update to each replica chunkserver
- Each chunkserver applies the update to its copy



WHAT CAN GO WRONG?

- *Two* clients write the same data at the same time
  - i.e. "concurrent writes"
  - Chunkservers may see the updates in different orders!
  - Again, the risk is that, later, two clients may read different content



Thoughts on how to fix it?

IDEA: PRIMARY/SECONDARY REPLICATION

- For each chunk, designate one server as "primary".
- Clients send write requests just to the primary.
  - The primary alone manages interactions with secondary servers.
  - (Some designs send reads just to primary, some also to secondaries)
- The primary chooses the order for all client writes.
  - Tells the secondaries -- with sequence numbers -- so all replicas
  - apply writes in the same order, even for concurrent client writes.

WHEN C WANTS TO WRITE A FILE AT SOME OFFSET?

- 1. C asks CO about file's chunk @ offset
- 2. CO tells C the primary and secondaries
- 3. C sends data to all (just temporary...), waits for all replies (?)
- 4. C asks P to write
- P checks that lease (?) hasn't expired
- P writes its own chunk file (a Linux file)
- 5. P tells each secondary to write
- (copy temporary into chunk file)
- 6. P waits for all secondaries to reply, or timeout
- secondary can reply "error" e.g. out of disk space
- 7. P tells C "ok" or "error"
- C retries from start if error

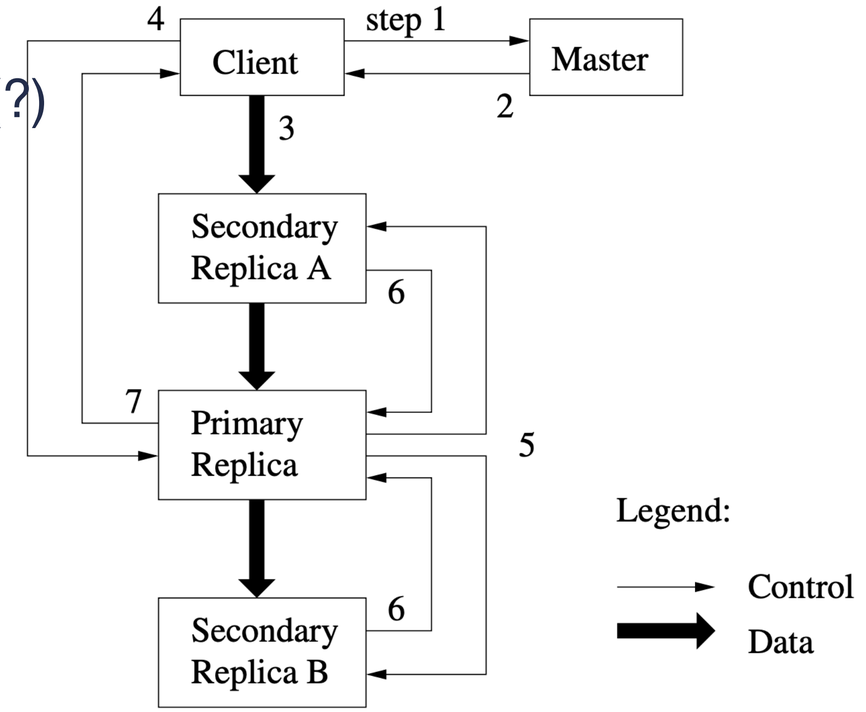


Figure 2: Write Control and Data Flow

Consistency

GFS CONSISTENCY GUARANTEES

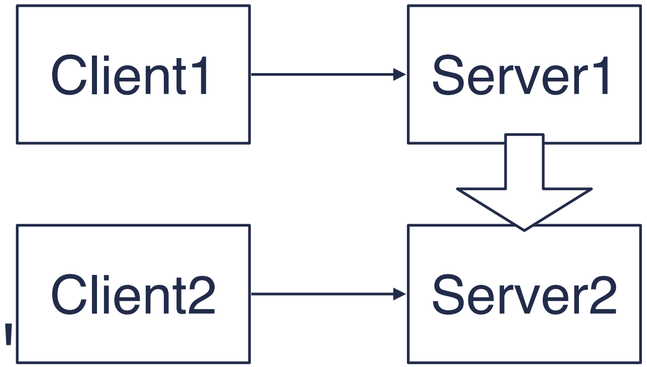
- somewhat complex!
- if primary tells client that a write succeeded,
  - and no other client is writing the same part of the file,
  - all readers will see the write.
  - "defined"
- if successful concurrent writes to the same part of a file,
  - and they all succeed, all readers will see the same content,
  - but maybe it will be a mix of the writes.
  - "consistent"
  - E.g. C1 writes "ab", C2 writes "xy", everyone might see "xb".
- if primary doesn't tell the client that the write succeeded,
  - different readers may see different content, or none.
  - "inconsistent"

	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

Table 1: File Region State After Mutation

HOW CAN INCONSISTENT CONTENT ARISE?

- Primary P updated its own state.
- But secondary S1 did not update (failed? slow? network problem?).
- Client C1 reads from P; Client C2 reads from S1.
  - they will see different results!
- Such a departure from ideal behavior is an "anomaly".
- But note that in this case the primary would have returned
  - an error to the writing client.



HOW CAN CONSISTENT BUT UNDEFINED ARISE?

- Clients break big writes into multiple small writes,
- e.g. at chunk boundaries, and GFS may interleave
  - them if concurrent client writes.

WHY ARE THESE ANOMALIES OK?

- They only intended to support a certain subset of their own applications.
  - Written with knowledge of GFS's behavior.
- Probably mostly single-writer and Record Append.
- Writers could include checksums and record IDs.
  - Readers could use them to filter out junk and duplicates.
- Later commentary by Google engineers suggests that it
  - might have been better to make GFS more consistent.
  - http://queue.acm.org/detail.cfm?id=1594206

WHAT MIGHT BETTER CONSISTENCY LOOK LIKE?

- There are many possible answers.
- Trade-off between easy-to-use for client application programmers,
  - and easy-to-implement for storage system designers.
- Maybe try to mimic local disk file behavior.
- Perhaps:
  - * atomic writes: either all replicas are updated, or none, even if failures.
  - * read sees latest write.
  - * all readers see the same content (assuming no writes).

Fault Tolerance

A CLIENT CRASHES WHILE WRITING?

- Either it got as far as asking primary to write, or not.

A SECONDARY CRASHES JUST AS THE PRIMARY ASKS IT TO WRITE?

- 1. Primary may retry a few times, if secondary revives quickly
- with disk intact, it may execute the primary's request
- and all is well.
- 2. Primary gives up, and returns an error to the client.
- Client can retry -- but why would the write work the second
- time around?
- 3. Coordinator notices that a chunkserver is down.
- Periodically pings all chunk servers.
- Removes the failed chunkserver from all chunkhandle lists.
- Perhaps re-replicates, to maintain 3 replicas.
- Tells primary the new secondary list.

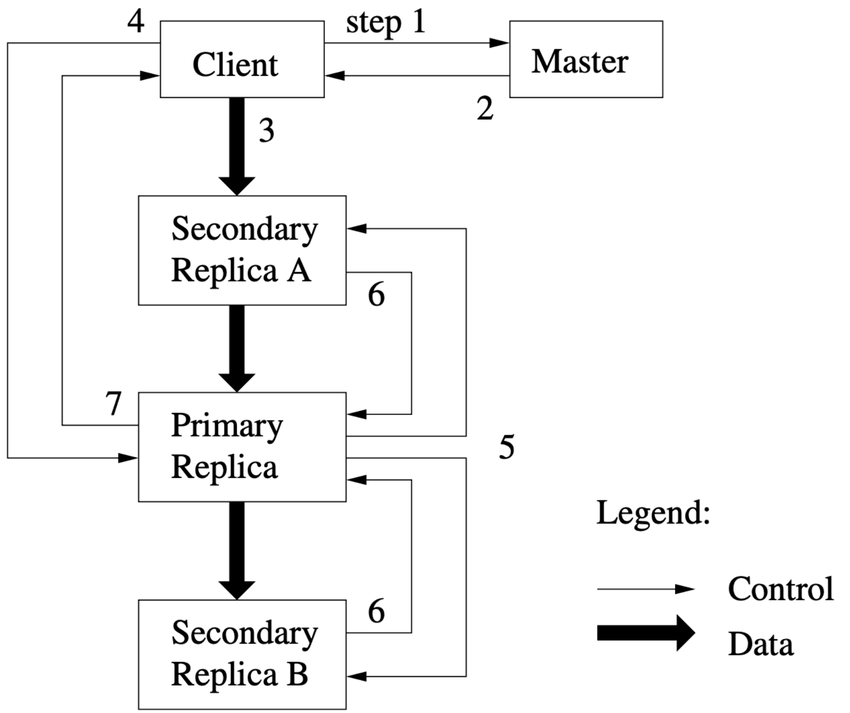


Figure 2: Write Control and Data Flow

A SECONDARY CRASHES JUST AS THE PRIMARY ASKS IT TO WRITE?

- Re-replication after a chunkserver failure may take a **long time**.
  - Since a chunkserver failure requires re-replication of all its chunks.
  - 80 GB disk, 10 MB/s network -> an hour or two for full copy.
  - So the primary probably re-tries for a while,
  - and the coordinator lets the system operate with a missing
  - chunk replica, before declaring the chunkserver permanently dead.
  - How long to wait before re-replicating?
  - Too short: wasted copying work if chunkserver comes back to life.
  - Too long: more failures might destroy all copies of data.

WHAT IF A PRIMARY CRASHES?

- The coordinator must be able to designate a
- new primary if the present primary fails.
- But the coordinator cannot distinguish "primary has failed"
- from "primary is still alive but the network has a problem."
- What if the coordinator designates a new primary
- while old one is active?
  - two active primaries!
  - C1 writes to P1, C2 reads from P2, doesn't seen C1's write!
  - called "split brain" -- a disaster

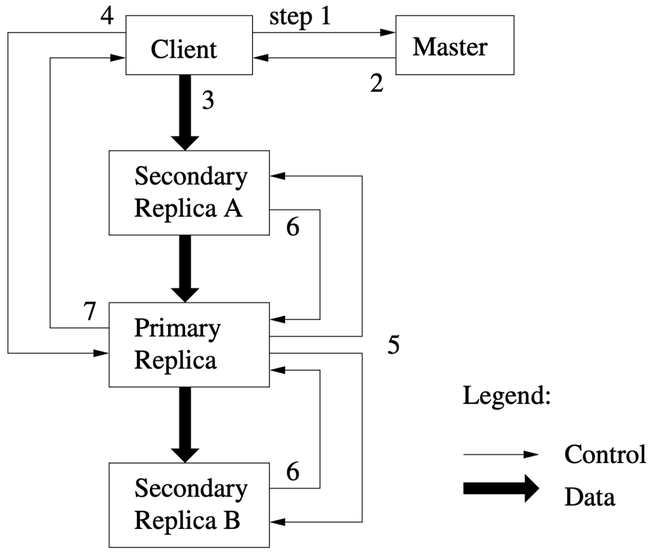


Figure 2: Write Control and Data Flow

Thoughts on how to fix it?

WHAT IF A PRIMARY CRASHES?

- Solution: Lease
  - Permission to act as primary for a given time (60 seconds).
  - Primary promises to stop acting as primary before lease expires.
  - Coordinator promises not to change primaries until after expiration.
  - Separate lease per actively written chunk.
- Leases help prevent split brain:
  - Coordinator won't designate new primary until the current one is
  - guaranteed to have stopped acting as primary.

WHAT IF A PRIMARY CRASHES?

- Remove that chunkserver from all chunkhandle lists.
- For each chunk for which it was primary,
  - wait for lease to expire,
  - grant lease to another chunkserver holding that chunk.

WHAT IF THE COORDINATOR CRASHES?

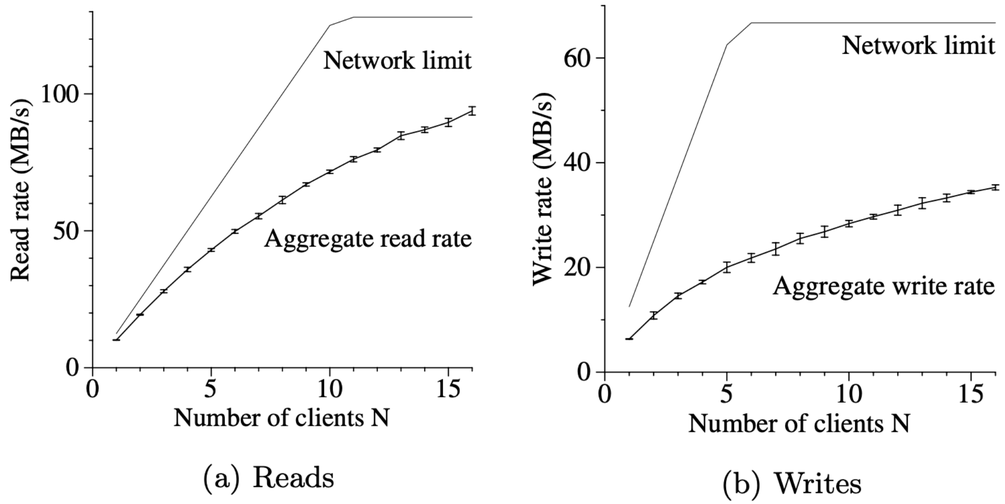
- Two strategies.
- 1. Coordinator writes critical state to its disk.
  - If it crashes and reboots with disk intact,
  - re-reads state, resumes operations.
- 2. Coordinator sends each state update to a "backup coordinator",
  - which also records it to disk; backup coordinator can take
  - over if main coordinator cannot be restarted.

INFORMATION FOR PERSISTENCE

- Table mapping file name -> array of chunk handles.
- Table mapping chunk handle -> current version #.
- What about the list of chunkservers for each chunk?
  - A rebooted coordinator asks all the chunkservers what they store.
- A rebooted coordinator must also wait one lease time before
  - designating any new primaries.

PERFORMANCE

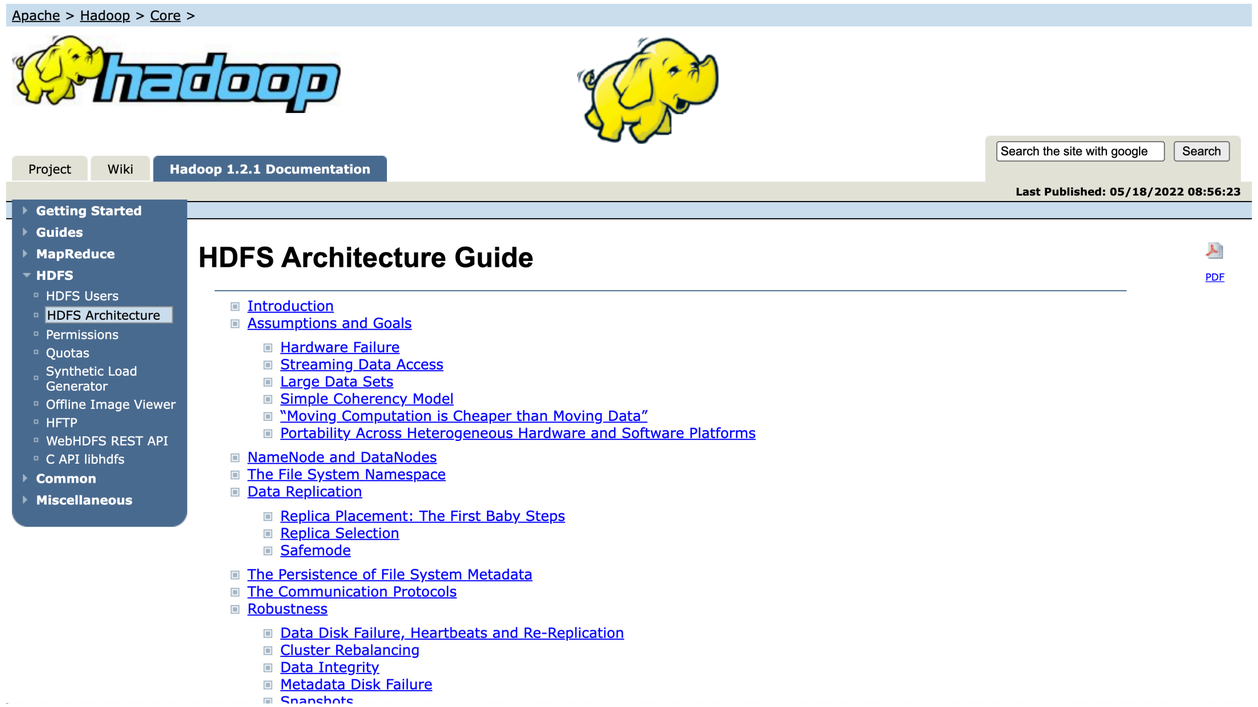
- large aggregate throughput for read
  - 94 MB/sec total for 16 clients + 16 chunkservers
  - or 6 MB/second per client
  - is that good?
  - one disk sequential throughput was about 30 MB/s
  - one NIC was about 10 MB/s
  - Close to saturating inter-switch link's 125 MB/sec (1 Gbit/sec)
  - So: multi-client scalability is good
  - Table 3 reports 500 MB/sec for production GFS, which was a lot
- writes to different files lower than possible maximum
  - authors blame their network stack (but no detail)
- hard to interpret after 15 years, e.g. how fast were the disks?



(a) Reads (b) Writes

WANNA PLAY WITH GFS YOURSELF?

– Try HDFS (an open-source clone inspired by GFS)!



UNIVERSITY of VIRGINIA

TAKEAWAYS

- Case study of performance, fault-tolerance, consistency, specialized for MapReduce
- Good ideas:
  - (1) global file system as universal infra (2) separation of naming (coordinator) from storage (chunkserver) (3) sharding for parallel throughput (4) huge chunks to reduce overheads (5) primary to choose order for concurrent writes (6) leases to prevent split-brain
- Not so great:
  - (1) single coordinator performance (ran out of RAM and CPU) (2) chunkservers not very efficient for small files (3) lack of automatic fail-over to coordinator replica (4) maybe consistency was too relaxed

---

- Next class: **ZooKeeper**



ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.