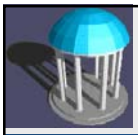


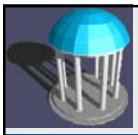
COMP 790-088 -- Distributed File Systems

Google File System

Google is Really Different….

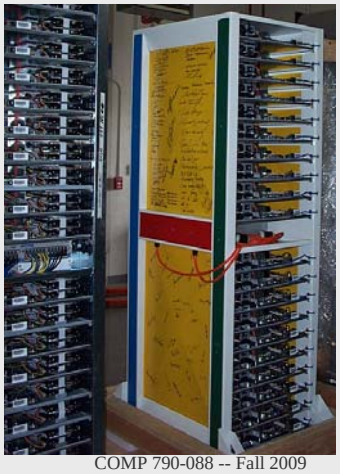
- Huge Datacenters in 25+ Worldwide Locations
- Datacenters house multiple server clusters
- Coming soon to Lenior, NC

The Dalles, OR (2006)

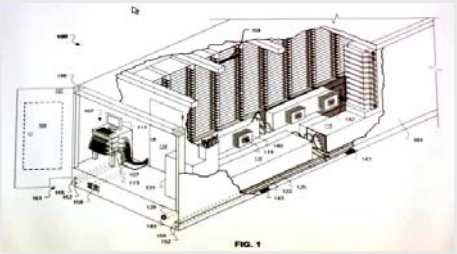


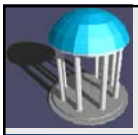
Google is Really Different….

- Each cluster has hundreds/thousands of Linux systems on Ethernet switches
- 500,000+ total servers



Google Hardware Today



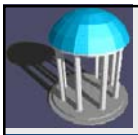


Google Environment

- Clusters of low-cost commodity hardware
  - Custom design using high-volume components
  - ATA disks, not SCSI (high capacity, low cost, somewhat less reliable)
  - No “server-class” machines
- Local switched network
  - Low end-to-end latency
  - more available bandwidth
  - low loss

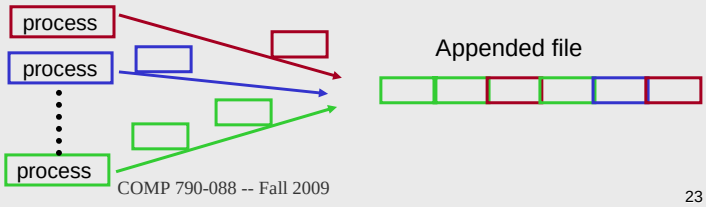
Google File System Design Goals

- Familiar operations but NOT Unix/Posix
  - Specialized operation for Google applications
    - record_append()
  - GFS client API code linked into each application
- Scalable -- O(1000s) of clients
- Performance optimized for throughput
  - No client caches (big files, little temporal locality)
- Highly available and fault tolerant
- Relaxed file consistency semantics
  - Applications written to deal with issues



File and Usage Characteristics

- Many files are 100s of MB or 10s of GB
  - Results from web crawls, query logs, archives, etc.
  - Relatively small number of files (millions/cluster)
- File operations:
  - Large sequential (streaming) reads/writes
  - Small random reads (rare random writes)
- Files are mostly "write-once, read-many."
- Mutations are dominated by appends, many from hundreds of concurrent writers



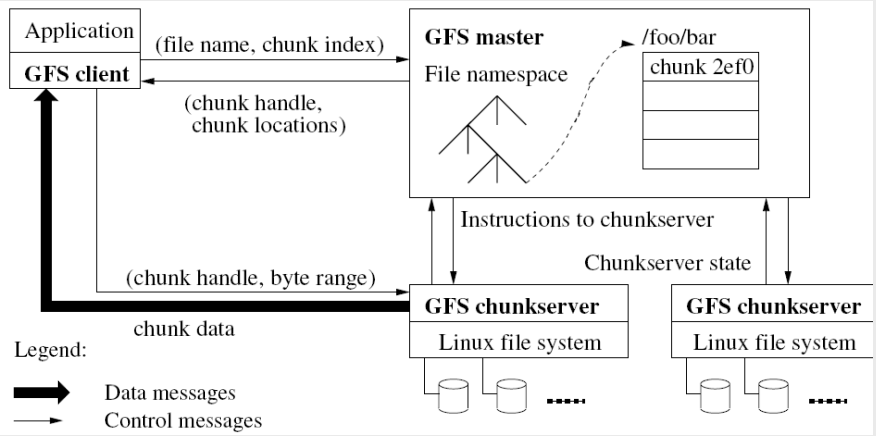
GFS Basics

- Files named with conventional pathname hierarchy
  - E.g., /dir1/dir2/dir3/foobar
- Files are composed of 64 MB "chunks"
- Each GFS cluster has servers (Linux processes):
  - One primary Master Server
  - Several "Shadow" Master Servers
  - Hundreds of Chunk Servers
- Each chunk is represented by a Linux file
  - Linux file system buffer provides caching and read-ahead
  - Linux file system extends file space as needed to chunk size
- Each chunk is replicated (3 replicas default)
  - Chunks are checksummed in 64KB blocks for data integrity

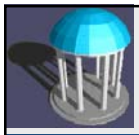
Master Server Functions

- Maintain file name space (atomic create, delete names)
- Maintain chunk metadata
  - Assign immutable globally-unique 64-bit identifier
  - Mapping from files name to chunk(s)
  - Current chunk replica locations
    - Refresh dynamically from chunk servers
- Maintain access control data
- Manage chunk-related actions
  - Assign primary replica and version number
  - Garbage collect deleted chunks and stale replicas
    - Stale replicas detected by old version numbers when chunk servers report
  - Migrate chunks for load balancing
  - Re-replicate chunks when servers fail
- Heartbeat and state-exchange messages with chunk servers

GFS Protocols for File Reads

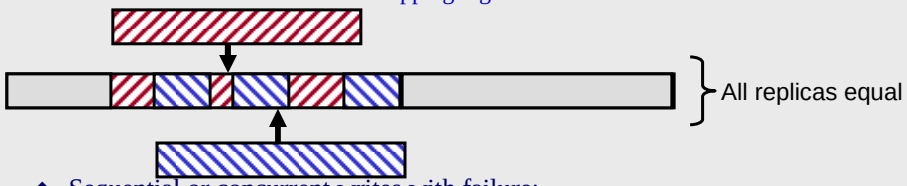


Minimizes client interaction with master:
- Data operations directly with chunk servers.
- Clients cache chunk metadata until new open or timeout



GFS Relaxed Consistency Model

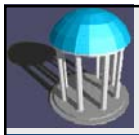
- Writes that are large or cross chunk boundaries may be broken into multiple smaller ones by GFS
- Sequential writes successful:
  - One copy semantics, writes serialized.
- Concurrent writes successful:
  - One copy semantics
  - Writes not serialized in overlapping regions



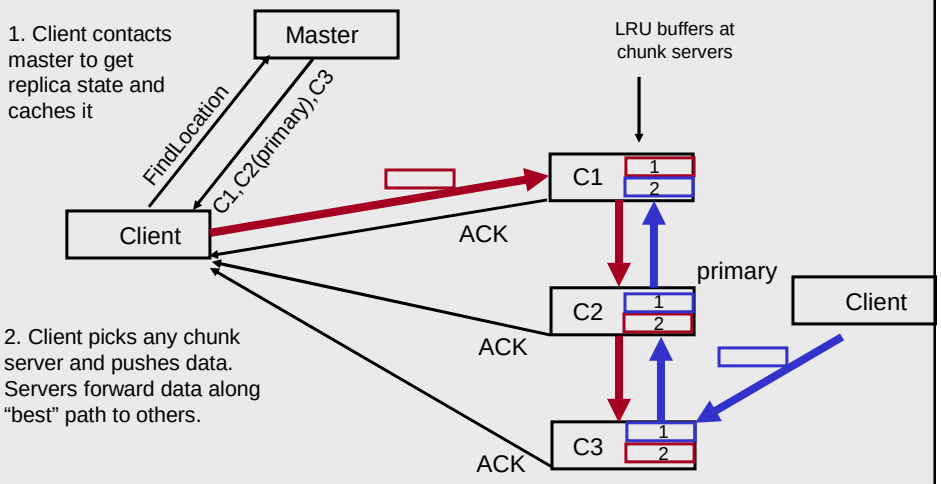
- Sequential or concurrent writes with failure:
  - Replicas may differ
  - Application should retry

GFS Applications Deal with Relaxed Consistency

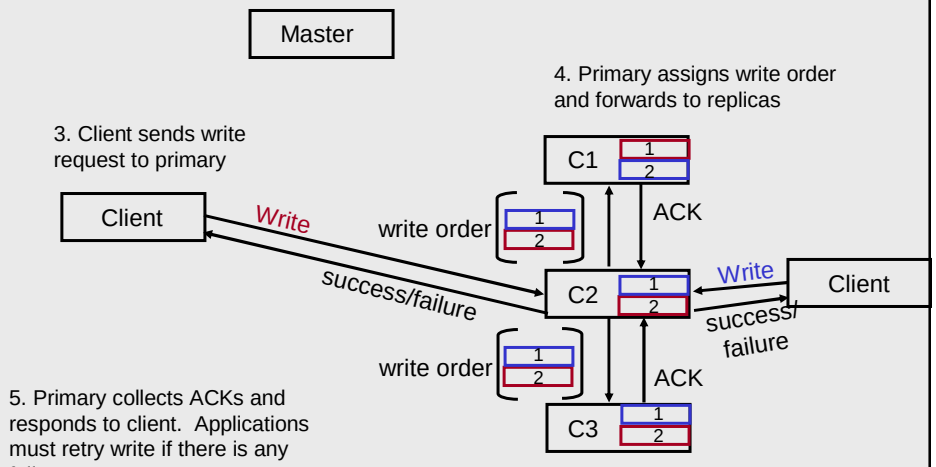
- Mutations
  - Retry in case of failure at any replica
  - Regular checkpoints after successful sequences
  - Include application-generated record identifiers and checksums
- Reading
  - Use checksum validation and record identifiers to discard padding and duplicates.

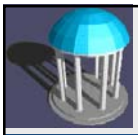


GFS Chunk Replication (1/2)



GFS Chunk Replication (2/2)

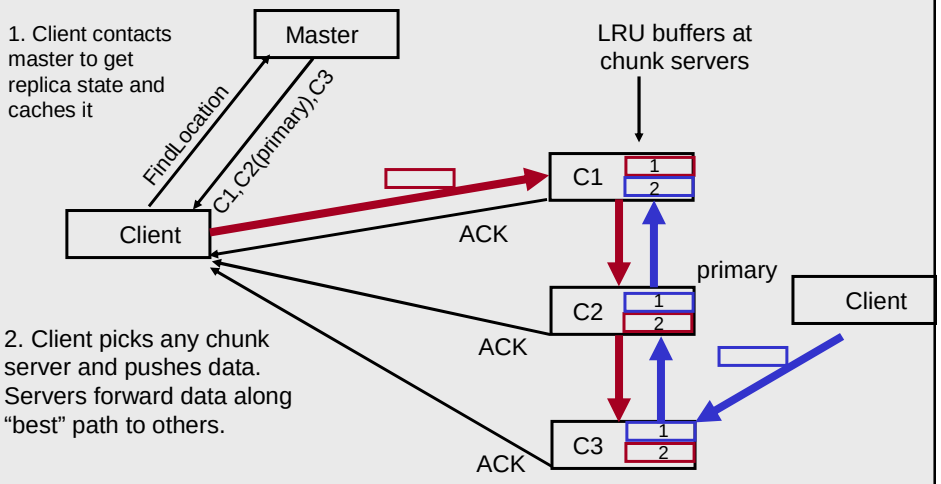


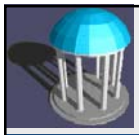


GFS record_append()

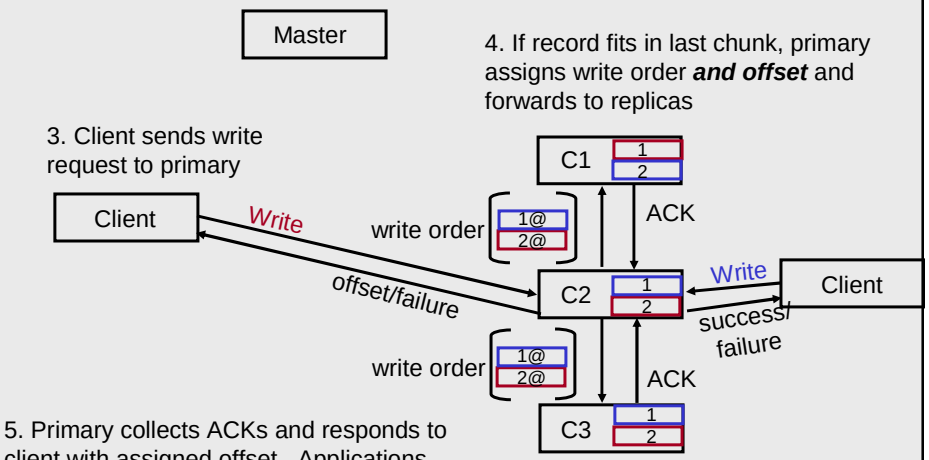
- Client specifies only data and region size; server returns actual offset to region
- Guaranteed to append at least once atomically
- File may contain padding and duplicates
  - Padding if region size won't fit in chunk
  - Duplicates if it fails at some replicas and client must retry *record_append()*
- If *record_append()* completes successfully, all replicas will contain at least one copy of the region at the same offset

GFS Record Append (1/3)

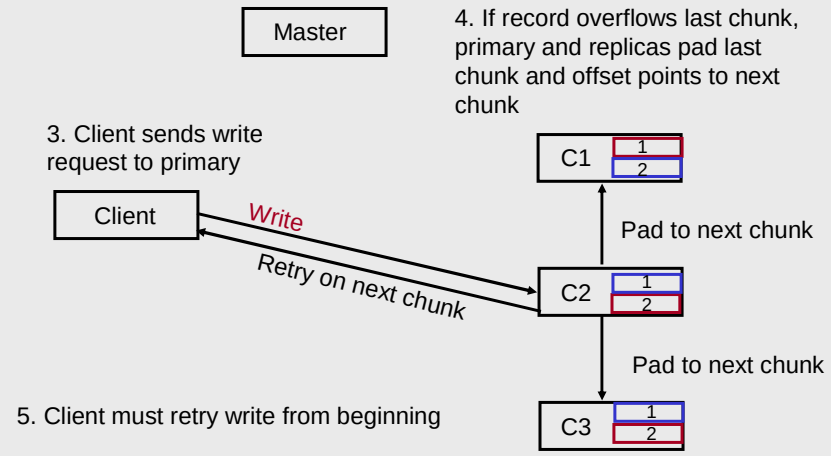


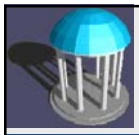


GFS Record Append (2/3)



GFS Record Append (3/3)





Metrics for 2 GFS Clusters (2003)

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

210 MB/file
(70 MB/replica)

75 MB/file
(10 MB/replica)

13.5 KB/chunk
(mostly checksums)

80 bytes/file

Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s

File Operation Statistics

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
0K	0.4	2.6	0	0	0	0
1B..1K	0.1	4.1	6.6	4.9	0.2	9.2
1K..8K	65.2	38.5	0.4	1.0	18.9	15.2
8K..64K	29.9	45.1	17.8	43.0	78.0	2.8
64K..128K	0.1	0.7	2.3	1.9	< .1	4.3
128K..256K	0.2	0.3	31.6	0.4	< .1	10.6
256K..512K	0.1	0.1	4.2	7.7	< .1	31.2
512K..1M	3.9	6.9	35.5	28.7	2.2	25.5
1M..inf	0.1	1.8	1.5	12.3	0.7	2.2

Table 4: Operations Breakdown by Size (%).

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
1B..1K	< .1	< .1	< .1	< .1	< .1	< .1
1K..8K	13.8	3.9	< .1	< .1	< .1	0.1
8K..64K	11.4	9.3	2.4	5.9	2.3	0.3
64K..128K	0.3	0.7	0.3	0.3	22.7	1.2
128K..256K	0.8	0.6	16.5	0.2	< .1	5.8
256K..512K	1.4	0.3	3.4	7.7	< .1	38.4
512K..1M	65.9	55.1	74.1	58.0	.1	46.8
1M..inf	6.4	30.1	3.3	28.0	53.9	7.4

(???)

Table 5: Bytes Transferred Breakdown by Opera