Google File System

File (System) Sizes in Linux

- Originally, Linux supported a maximum file size of 2 GB.

Table A.2. Maximum Sizes of File Systems (On-Disk Format)

File System	File Size [Byte]	File System Size [Byte]
Ext2 or Ext3 (1 kB block size)	2^{34} (16 GB)	2^{41} (2 TB)
Ext2 or Ext3 (2 kB block size)	2^{38} (256 GB)	2^{43} (8 TB)
Ext2 or Ext3 (4 kB block size)	2^{41} (2 TB)	2^{44} (16 TB)
Ext2 or Ext3 (8 kB block size) (systems with 8 kB pages (like Alpha))	2^{46} (64 TB)	2^{45} (32 TB)
ReiserFS 3.5	2^{32} (4 GB)	2^{44} (16 TB)
ReiserFS 3.6 (under Linux 2.4)	2^{60} (1 EB)	2^{44} (16 TB)
XFS	2^{63} (8 EB)	2^{63} (8 EB)
JFS (512 Bytes block size)	2^{63} (8 EB)	2^{49} (512 TB)
JFS (4 kB block size)	2^{63} (8 EB)	2^{52} (4 PB)
NFSv2 (client side)	2^{31} (2 GB)	2^{63} (8 EB)
NFSv3 (client side)	2^{63} (8 EB)	2^{63} (8 EB)

Assumptions

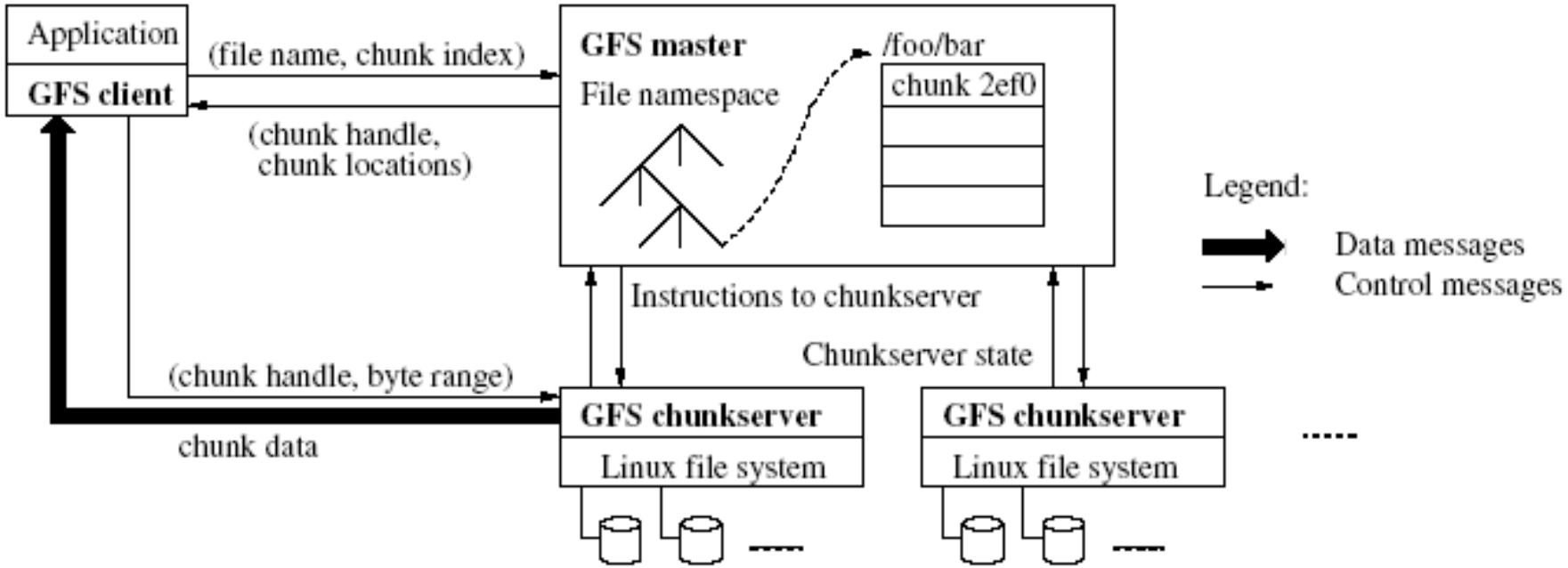
- GFS built with commodity hardware
- GFS stores a modest number of large files
  - A few million files, each typically 100MB or larger (Multi-GB files are common)
  - No need to optimize for small files
- Workloads
  - Large streaming reads (1MB or more) and small random reads (a few KBs)
  - Sequential appends to files by hundreds of data producers
- High sustained bandwidth is more important than latency
  - Response time for individual read and write is not critical

Architecture

- Single master, multiple chunk servers, multiple clients
  - User-level process running on commodity Linux machine
  - GFS client code linked into each client application to communicate
- File -> 64MB chunks -> Linux files
  - on local disks of chunk servers
  - replicated on multiple chunk servers (3r)
- Cache metadata but not chunk on clients

Single Master

- Why centralization? Simplicity!
- Global knowledge is needed for
  - Chunk placement
  - Replication decisions



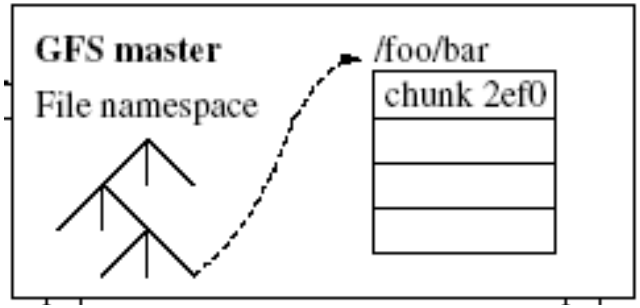
Chunk size

64MB – Much Larger than ordinary, why?

- Advantages
  - Reduce client-master interaction
  - Reduce network overhead
  - Reduce the size of the metadata
- Disadvantages
  - Internal fragmentation
    - Solution: lazy space allocation
  - Hot Spots – many clients accessing a 1-chunk file, e.g. executables
    - Solution:
    - Higher replication factor
    - Stagger application start times
    - Client-to-client communication

Metadata

- File & chunk namespaces
  - In master's memory and storage
- File-chunk mapping
  - In master's memory and storage
- Location of chunk replicas
  - In master's memory
  - Ask chunk servers when
    - Master starts
    - Chunk server joins the cluster
  - If persistent, master and chunk servers must be in sync



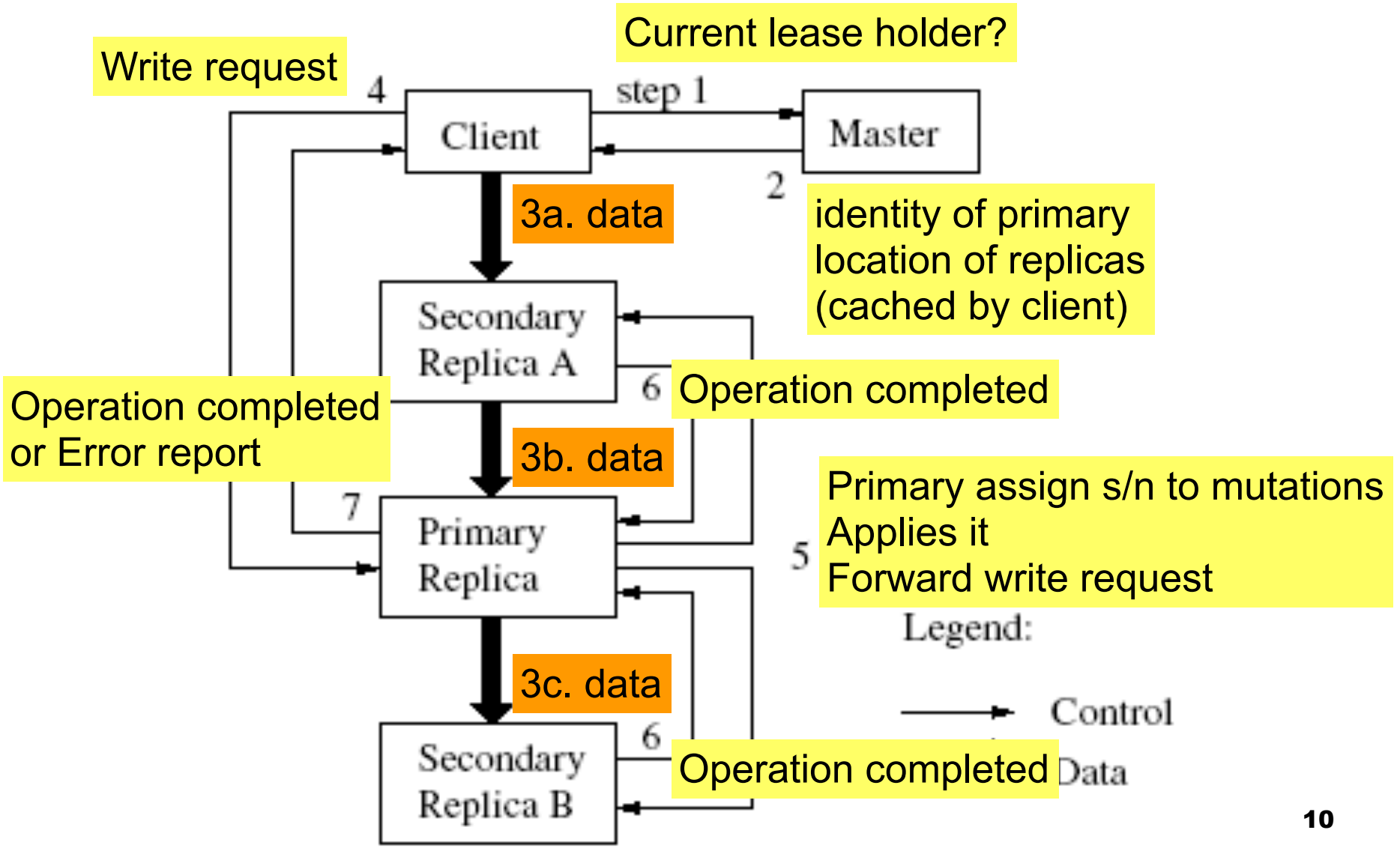
Metadata - log

- The persistent record of metadata
- Defines the order of concurrent operations
- Critical
  - Replicated on multiple remote machines
  - Respond to client only when log locally and remotely
- Fast recovery by using checkpoints
  - Use a compact B-tree like form directly mapping into memory
  - Switch to a new log, create new checkpoints in a separate threads

System Interactions - Lease

- Minimized management overhead
- Granted by the master to one of the replicas to become the primary
- Primary picks a serial order of mutation and all replicas follow
- 60 seconds timeout, can be extended
- Can be revoked

Write – Mutation order



System Interactions – Atomic Record Append

- Concurrent appends are serializable
  - Client specifies only data
  - GFS appends at least once atomically
  - Return the offset to the client
  - Step 6 extends to:
    - If record fits in current chunk: write record and tell replicas the offset
    - If record exceeds chunk: pad the chunk, reply to client to use next chunk
  - Heavily used by Google
  - On failures, the client retries the operation

Consistency Model

- File namespace mutations (e.g., file creation) are atomic
  - Namespace management and locking
  - The master's operation log
- File Regions
  - Consistent: all clients see same data, regardless which replicated chunk they use
  - Defined: after data mutation (*writes* or *record appends*) if it is consistent & all clients will see the entire mutation effect
- After a sequence of successful mutations, the mutated file is guaranteed to be defined and contain the data written by the last mutation. This is obtained by
  - Applying the same mutation order to all replicas
  - Using chunk version numbers to detect stale replica

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

System Interactions – Snapshot

- Master makes a copy of a file or a directory tree almost instantaneously
- Use copy-on-write
- Steps
  - Revokes lease
  - Logs operations to disk
  - Duplicates metadata, pointing to the same chunks
- Create real duplicate locally
  - Disks are 3 times as fast as 100 Mb Ethernet links

Master Operation – Namespace Management

- GFS has no directory (i-node) structure
  - Simply uses directory-like file names: /foo, /foo/bar
- Concurrent Access
  - Read lock on a parent path, write lock on the leaf file name
    - protect delete, rename and snapshot of in-use files
- Rebalancing
  - Places new replicas on chunk servers with below-average disk space utilizations
- Re-replication
  - When the number of replicas falls below 3 (or user-specified threshold)
    - The master assigns the highest priority to copy (clone) such chunks
  - Spread replicas of a chunk across racks

Master Operation – Garbage Collection

- File deletion
  - Rename the file to a hidden name (deferred deletion)
  - The master regularly scans and removes hidden files, existed more than three days
    - HeartBeat messages inform chunk servers of deleted chunks
- Stale Replica Detection
  - Version number is assigned for each chunk
    - increases when the master grants a new lease of the chunk

Fault Tolerance – High Availability

- Fast Recovery
  - The master and the chunk server are designed to restore their state in seconds no matter how they terminated.
  - Servers are routinely shut down just by killing the process
- Master Replications
  - Master has the maps from file names to chunks
  - Only one master manages chunk mutations
    - Several shadow masters exist for redundancy
      - Snoop operation logs and apply these operations exactly as the primary does
- Data Integrity
  - Checksums for each 64KB segment in a chunk
    - chunk servers verifies the checksum of data before sending it to the client or other chunk servers

Evaluation with 15 servers

- reading performance comparable to that of a single disk (80–100 MB/s)
- reduced write performance (30 MB/s)
- slow (5 MB/s) in appending data to existing files
- master node not directly involved in data reading, the read rate increases significantly with the number of chunk servers: 583 MB/s for 342 nodes

Conclusion

- GFS is radically different than traditional FS
  - Component failure is normal
  - Optimize for huge files
  - Append as much as possible
- Much attention paid to
  - Monitoring
  - Replication
  - Recovery
- Minimize master involvement with common operations to avoid bottleneck
- Design is a success and widely used in Google

GFS lessons (2010)

- Scaled to approximately 50M files, 10P
- Large files increased upstream app. complexity
- Not appropriate for latency sensitive applications
- Scaling limits added management overhead