Google File System

Google File System

- Google needed a good distributed file system
- Why not use an existing file system?
  - Different workload and design priorities
  - GFS is designed for Google apps
  - Google apps are designed for GFS!

- What are the applications and the workload considerations that drives the design of GFS?

Google Workload

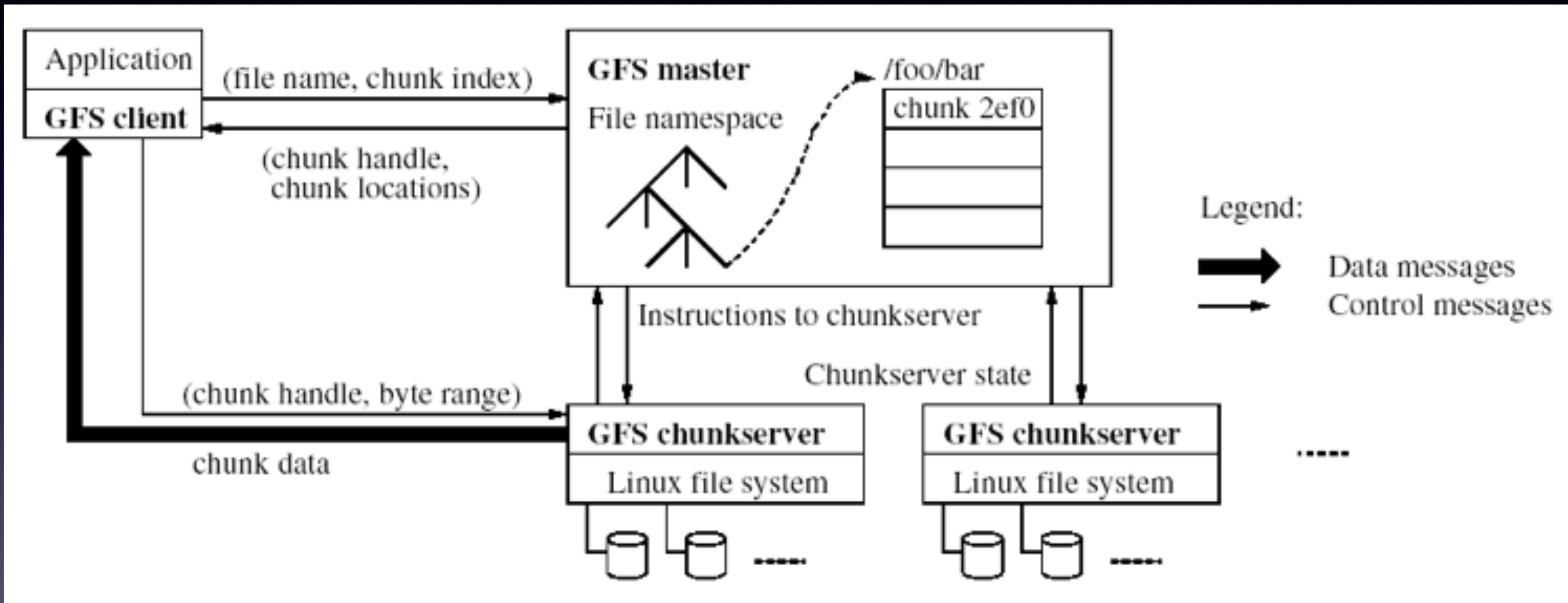
- Hundreds of web-crawling application
- Files: few million of 100MB+ files
- Reads: small random reads and large streaming reads
- Writes:
  - many files written once, read sequentially
  - random writes non-existent, mostly appends

- What are the design choices made by GFS?

GFS Design Decisions

- Files stored as chunks (fixed size: 64MB)
- Reliability through replication
  - each chunk replicated over 3+ chunkservers
- Simple master to coordinate access, keep metadata
- No data caching! Why?
- Familiar interface, but customize the API
  - focus on Google apps; add snapshot and record append operations

GFS Architecture



Key Design Choices

- Shadow masters
- Minimize master involvement
  - Never move data through it (only metadata)
  - Cache metadata at clients
  - Large chunk size
  - Master delegates authority to primary replicas in data mutations

Metadata

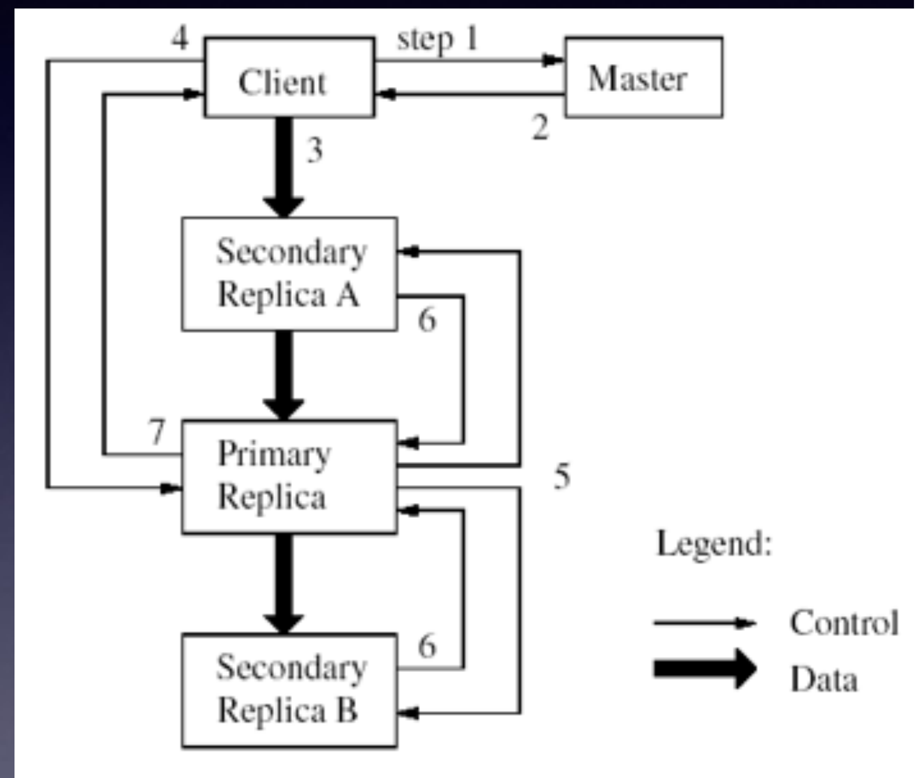
- Global metadata is stored on the master
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of each chunk's replicas
- All in memory (64B/chunk)
  - Few million files ==> can fit all in memory

Durability

- Master has an operation log for persistent logging of critical metadata updates
  - each log write is 2PC to multiple remote machines
  - replicated transactional redo log
  - group commit to reduce the overhead
  - checkpoint all (log) state periodically; essentially mmap file to avoid parsing
  - checkpoint: switch to new log and copy snapshot in background

Mutable Operations

- Mutation is write or append
- Goal: minimize master involvement
- Lease mechanism
  - Master picks one replica as primary; gives it a lease
  - Primary defines a serial order of mutations
- Data flow decoupled from control flow



Write Operations

- Application originates write request
- GFS client translates request from (fname, data) --> (fname, chunk-index) sends it to master
- Master responds with chunk handle and (primary +secondary) replica locations
- Client pushes write data to all locations; data is stored in chunkservers' internal buffers
- Client sends write command to primary

Write Operations (contd.)

- Primary determines serial order for data instances stored in its buffer and writes the instances in that order to the chunk
- Primary sends serial order to the secondaries and tells them to perform the write
- Secondaries respond to the primary
- Primary responds back to client
- Note: if write fails at one of the chunkservers, client is informed and retries the write

Life without random writes

- E.g., results of a previous search:

  www.page1.com -> www.my.blogspot.com
  www.page2.com -> www.my.blogspot.com

- Let’s say new results: page2 no longer has the link, but there is a new page, page3:

  www.page1.com -> www.my.blogspot.com
  www.page3.com -> www.my.blogspot.com

- Option: delete the old record (page2), and insert a new record (page3). This is cumbersome!
  - requires locking; just way too complex.
  - better: delete the old file, create a new file where this program (run on more than one machines) can append new records to the file “atomically”

Atomic Record Append

- GFS client contacts the primary
- Primary chooses and returns the offset
- Client appends the data to each replica at least once
- No need for a distributed lock manager; actual write can be an idempotent RPC (like in NFS)

Data Corruption

- Files stored on Linux and Linux has bugs
  - sometimes silent corruptions
- Files stored on disks and disks are not fail stop
  - stored blocks could be corrupted
  - rare events become common at scale
- Chunkserver maintains per-chunk CRC (64KB)

- Discussion: Identify one thing that you would improve about GFS and suggest an alternative design

~15 years later

- Scale is much bigger
  - now 10K servers instead of 1K, 100 PB instead of 100 TB
- Bigger upload change: updates to small files
- Around 2010: incremental updates of the Google search index

GFS -> Colossus

- Main scalability limit of GFS: single master
  - fixed by partitioning the metadata
  - ~100M files per master, smaller chunk sizes (1MB)
- Reduce storage overhead using erasure coding