Distributed Systems

- *48 - Communication Basics*
- *49 - NFS*
- *50 - AFS*
- GFS
- TRIO

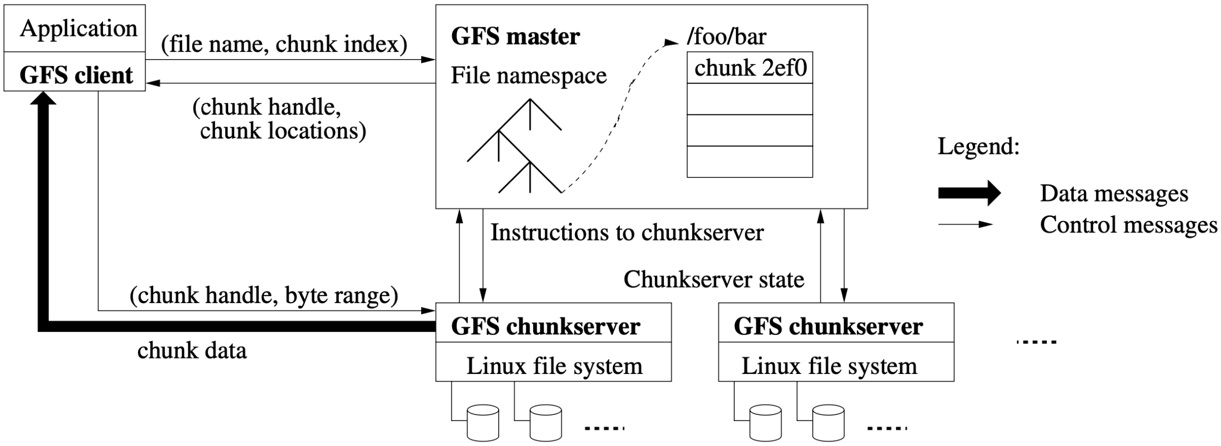
Google File System *v1*

- Needs
  - need to handle *massive* files
  - most mutations are appends
  - co-design w/ applications (*also an advantage*)
- Assumptions
  - built from hundreds, or thousands, of cheap machines
  - failures are the common case
- Features
  - relaxed consistency (*also an advantage*)
  - atomic record append (without locking)
  - no data caches
    - append-only model means re-use not common
    - host operating system does limited caching anyway

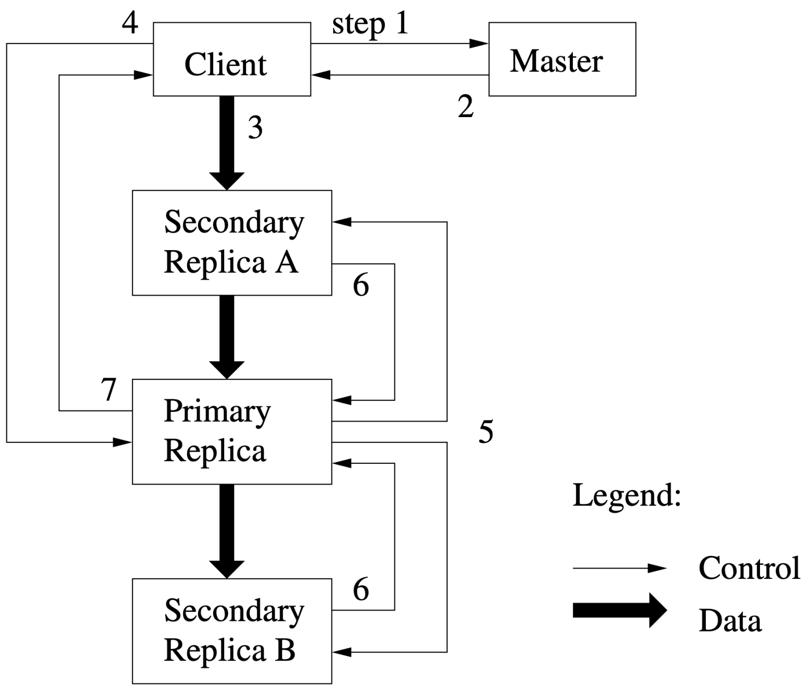
GFS *two types of nodes*

- multiple *chunk servers*
  - hold fixed size *chunks*
  - immutable once written
  - identified by a globally unique 64-bit ID
- coordinator (GFS *master*)
  - single machine holds all *metadata* in memory
    - persistent
      - file and chunk namespaces (think directories)
      - mappings from files to chunks
      - persistent by flushing *operations log* locally, remotely before visible
    - soft state
      - locations of chunk replicas
      - on startup or recovery restore by asking chunkservers
    - total state is 64 bytes for each 64MB chunk
    - background garbage collection, replica reassignment and balancing

GFS *architecture, and read*



GFS *pipelined writes*



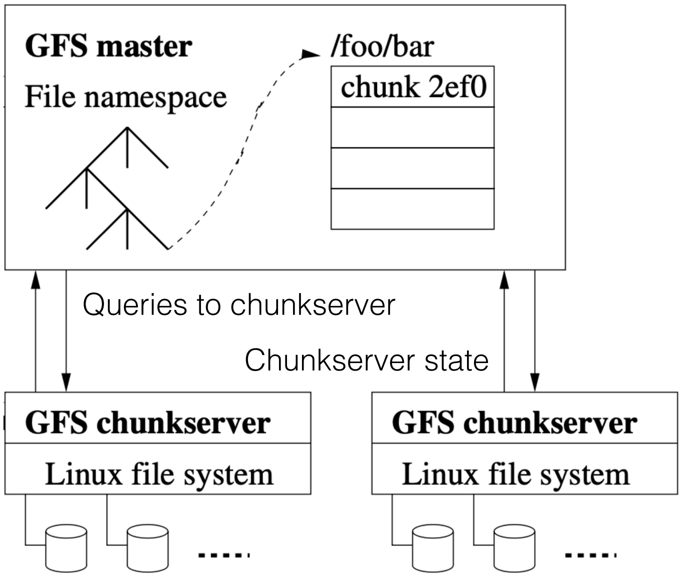
GFS *reliability*

- startup and recovery treated identically:
  - master polls all chunkservers for chunks they cache
  - read namespace info from locally persistent state
- other
  - master has *shadows* that are “almost” up to date
  - chunkservers can flush to disk asynchronously because of replication

GFS *consistency model*

- update consistency
  - file namespace mutations are atomic (handled by master)
  - state of a file region after append can be:
    - *consistent* if clients all guaranteed to see same data
    - *defined* if consistent and *last mutation correct not interleaved*
  - *concurrent updates* may leave system undefined, but consistent
    - all see same data, but may be mingled fragments of updates
    - usually when large writes broken into fragments
    - enough information for *application library* to fix
  - confusing
- cache consistency
  - no caches

During Recovery



GFS *summary*

- System for:
  - very large files (logs, like for web indexing)
  - very large writes
  - reads usually sequential through whole log
- Replication approach:
  - single master
  - multiple chunkservers
  - very simple consistency and recovery
  - single master only involved in lookups, not read or write
- Long-term view:
  - single master was a mistake

yes it's on the exam

Patterson's Law *for system building*

- Before building a new system:
  - measure old system
  - demonstrate a problem
- When you build the new system, two advantages:
  - you have evidence you are solving a real problem
  - you know exactly what to measure

This approach applies in many different contexts….

Distributed Systems

- *48 - Communication Basics*
- *49 - NFS*
- *50 - AFS*
- *GFS*
- TRIO