Assumptions

* component failures are the norm
* files are huge
* files mutated mostly by appending new data (not overwriting)
* introduced atomic append operation
* 2 kinds of reads
  + large streaming reads
    - client read through continuous region of file
  + small random read
    - few kbs at some arbitrary offset
* high sustained bandwidth is the goal

GFS cluster

* master
  + maintains all metadata
    - file and chunk namespaces
    - mapping from files to chunks
    - locations of each chunk's replicas
      * not stored persistently
      * master asks each chunkserver about its chunks at startup
    - first two types kept persistent in operation log
    - stored in memory
  + operation log
    - replicated on multiple machines
    - master can recover its file system state by replaying log
    - state is checkpointed whenever log grows beyond certain siz
  + communicates with each chunkserver in Heartbeats
  + rereplicates chunks when available replicas falls below user-specified goal
  + Master is replicated itself
    - if master fails, monitoring infrastructure *outside* gfs starts new master process elsewhere with replicated operation log
    - **shadow masters** provide read-only access to file system even when primary master is down section 5.1
* Clients
  + interact with master only for metadata operations
  + master tells it which chunkservers to contact and it caches info
* files divided into fixed-size chunks
  + Each chunk has a unique chunk handle
  + each chunk replicated on 3 chunkservers
  + chunk is 64 mb (large)
    - leads to less contact with master
    - can perform more operations on a single chunk
    - less metadata

Guarantees

* namespace mutations are atomic
* region is **consistent** if all clients will always see same data, regardless of replica
* region is **defined** if consistent and clients will see all of what mutation writes
* Concurrent successful mutations leave region undefined but consistent
* failed mutation makes region inconsistent

Mutations

* writes
  + data written at application-specified offset
* record append
  + data to be appended atomically at least once (but at offset of GFS's choosing)
  + offset returned to client and marks beginning of defined region that contains record
  + GFS may insert padding or duplicates in between
  + Steps
    - client pushes data to all replicas of last chunk of file
    - client sends request to primary
    - primary checks if appending exceeds chunk size
      * pad the chunk to max size and tell client to retry
    - otherwise primary appends data and tells secondaries to write data at same offset as where it has
    - replies to client
    - If fail at any replica, client retries operation
      * so replicas can contain different duplicates
      * but it will be written at same offset on all replicas if succeed
* Reads
  + client sends request to one of the replicas (closest one)
* Clients may read from stale replica before information is refreshed
  + window limited by cache entry's timeout and next open of file
* GFS identifies failed chunkservers by regular handshakes and checksumming
  + data is restored by valid replicas
* readers need to implement handlers for duplicates and padding

leases

* master grants a chunk lease to primary
* primary picks serial order for all mutations to chunk
  + all replicas follow this order to apply the mutations
* Write data flow
  + client asks master which chunkserver holds lease
  + client caches locations of primary and other replicas
  + client pushes data to all replicas
  + once all replicas ack, client sends write req to primary
  + primary assigns serial numbers to all mutations it receives and applies them
  + primary forwards write req to all secondaries, which apply mutations in order
  + secondaries all reply to primary once done
  + primary replies to client
  + retry if failed
* Data flows through chain of chunkservers in pipelined fashion
  + each machine forwards data to closest machine that has not received it

Snapshots

* makes copy of file or directory tree
* master revokes all leases on the chunks of files it is about to snapshot
* master duplicates metadata for source file or directory tree
* when a client wants to write to chunk C after snapshot
  + master sees chunk has ref count >1
  + picks new chunk handle C' and asks each chunkserver that has replica of C to create new chunk C'
  + Then master grants one of replicas with lease of new chunk C' and replies to client

Locks

* each node in namespace tree has lock (both file and directories)
  + file creation means master acquires read locks on all ancestor directories
    - also write lock on the file being created

Garbage collection

* once file deleted, GFS not immediately reclaim the memory
  + renamed to hidden name
  + master removes hidden files if existed for more than 3 days
    - in memory metadata erased
    - master finds orphaned chunks in the heartbeats
      * servers will tell master some subset of chunks it has, master replies with the chunks that are not in metadata
* lazy garbage collection

Stale replica detection

* replica becomes stale if chunkserver fails and misses mutations while down
* master maintains chunk version number for each chunk
* whenever master grants new lease on chunk, increases chunk version number and informs replicas
* When chunkserver restarts and reports its chunks and version numbers, master can see it is stale
* master removes stale replicas in garbage collection
* master also gives chunk version number to client so client can verify