Storage Systems

NPTEL Course

Jan 2012

(Lecture 40)

K. Gopinath

Indian Institute of Science

Google File System

- Non-Posix scalable distr file system for large distr data-intensive applications
  - performance, scalability, reliability and availability
  - high aggregate perf to a large number of clients
    - sustained bandwidth more important than low latency
- High fault tolerance to allow inexpensive commodity HW
  - component failures norm rather than exception
  - appl/OS bugs, human errors + failures of disks, memory, connectors, networking, and power supplies.
  - constant monitoring, error detection, fault tolerance, and automatic recovery integral in the design

GFS Consistency Mgmt

- Relaxed model
- File namespace ops (e.g., file creation) atomic
  - exclusively by master
  - namespace locking guarantees atomicity and correctness
  - master's op log defines a global total order of these ops
- Mutations: op that changes contents or metadata of a chunk
    - writes or record appends
  - write: data written at an appl-specified file offset.
  - record append: data appended atomically at least once even in spite of concurrent mutations, but at an offset of GFS's choosing
    - offset returned to client and marks beginning of a defined region that contains record.
    - GFS may insert padding or record duplicates in between

GFS Mutations

- state of a file region after a data mutation depends on the type of mutation
  - success/fail? concurrent?
- a file region *consistent* if all clients will always see same data, regardless of which replicas they read from
- a region *defined* after a file data mutation if it is consistent and clients will see what the mutation writes in its entirety
  - when a mutation succeeds without interference from concurrent writers, region defined and consistent
    - all clients will always see what the mutation has written
- concurrent successful mutations leave region undefined but consistent
  - all clients see the same data, but it may not reflect what any one mutation has written.
  - typically, fragments from multiple mutations.
- a failed mutation makes region inconsistent/ undefined
  - different clients may see different data at different times
- applications can distinguish defined regions from undefined

GFS File Consistency Model

	Write	Record Append
Serial Success	Defined	Defined interspersed with inconsistent
Concurrent Success	Consistent but undefined	Defined interspersed with inconsistent
Failure	Inconsistent	Inconsistent

Mutations

- After a sequence of successful mutations, mutated file region guaranteed to be defined and contain data written by last mutation
  - apply mutations to a chunk in same order on all its replicas
  - use chunk version numbers to detect any replica that has become stale because it has missed mutations while its chunkserver was down
    - Stale replicas never involved in a mutation or given to clients that ask master for chunk locs
    - Garbage collected asap
- As a client caches chunk locs, may read from a stale replica
  - window limited by cache entry's timeout and next open of file, which purges from the cache all chunk info for that file
  - most files append-only: a stale replica usually returns a premature end of chunk rather than outdated data.
    - when a reader retries and contacts the master, it will imm get current chunk locs
- component failures can corrupt or destroy data.
  - identify failed chunkservers by heartbeats; detect data corruption by checksumming
    - data restored from valid replicas asap (typically within minutes)
    - chunk lost irreversibly only if all its replicas are lost
      - data unavailable, not corrupted: appls receive clear errors, not corrupt data

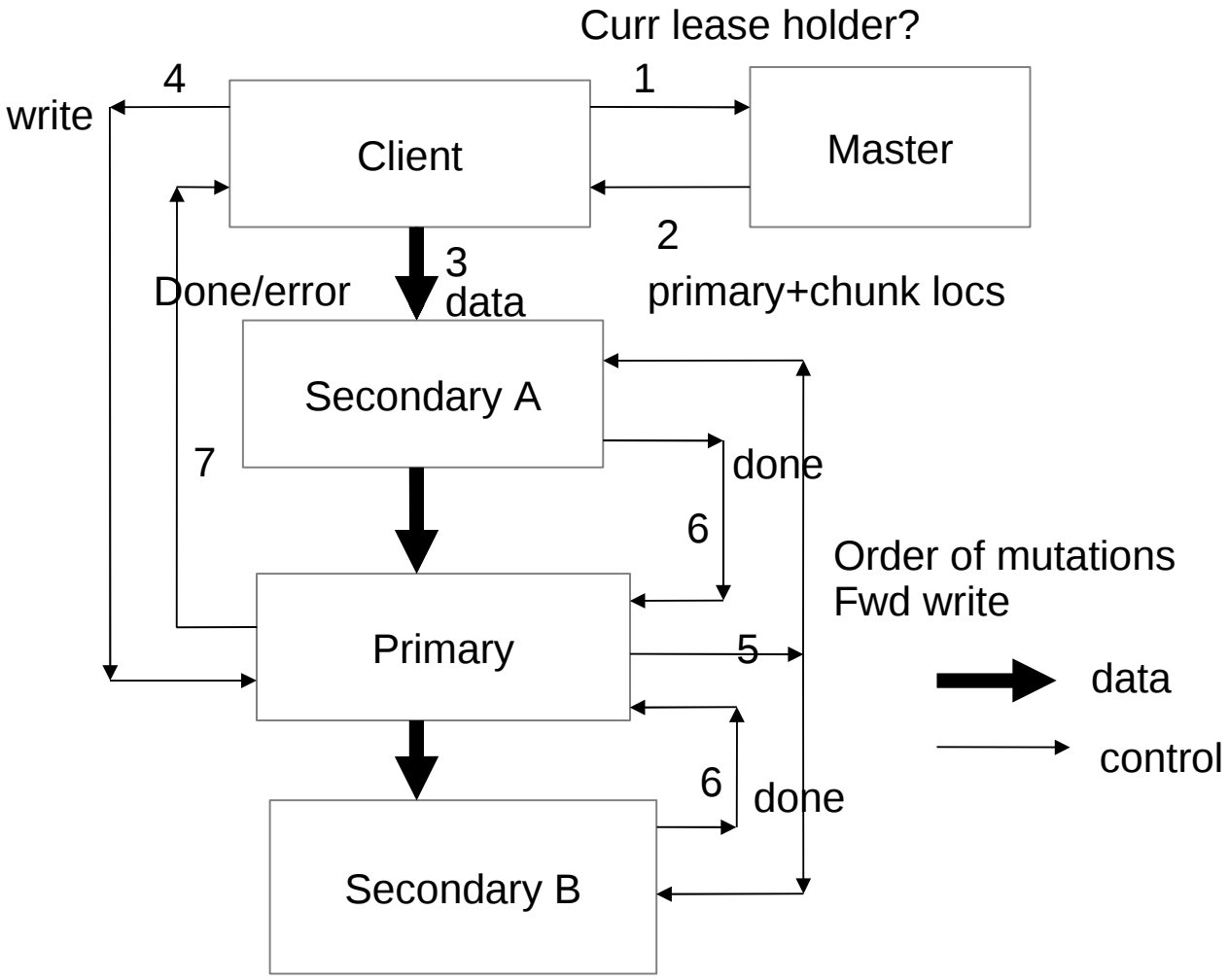
Appl Design with GFS

- rely on *appends* rather than overwrites, *checkpointing*, and *writing self-validating, self-identifying* records.
  - typical use: a writer generates a file from beginning to end and atomically renames file to a permanent name after writing all data
    - appending more efficient and resilient to appl failures than random writes
    - or, many writers concurrently append to a file for merged results or as a producer-consumer queue.
      - record append's "append-at-least-once" preserves each writer's output
      - readers identify/discard extra padding & record frags using checksums.
      - occasional duplicates and non-idempotent ops: readers ignore using unique ids in records, needed to name corresp appl entities such as web docs
  - or, periodically checkpoints how much has been successfully written
    - checkpointing allows writers to restart incrementally and keeps readers from processing successfully written file data that is still incomplete from appl's pov
- checkpoints can include appl level checksums
  - readers verify and process only file region up to last checkpoint, which is known to be in the defined state
- semantics of record I/O (except duplicate removal) in library code linked to appls
- simpler approach: avoids consistency and concurrency issues
  - no DLM needed with record append

Leases and Mutation Order

- each mutation performed at all chunk's replicas
  - large writes/straddling chunks broken up by client code
- use leases to maintain a consistent mutation order across replicas.
- master grants a chunk lease to one replica (primary) and chunk locs
  - cached by client
- client pushes data to all replicas (cached), acked by all
  - decouple data flow (pipelined) from control flow to improve perf by scheduling expensive data flow based on network topology regardless of which chunkserver is primary
- primary picks a serial order for all mutations to chunk
- all replicas follow this order when applying mutations
  - error: write may have succeeded at primary and some subset of secondary replicas
  - modified region left in an inconsistent state
  - client code handles such errors by retrying failed mutation: try resending data
    - no success: retry write from beginning
- file region may end up containing frags from different clients, but replicas identical
  - file region consistent but undefined state
- global mutation order defined by
  - lease grant order chosen by master
  - (within a lease) serial numbers assigned by primary

Write Control and Data Flow



Lease Mgmt

- designed to minimize mgmt overhead at master
- a lease initially times out at 60 secs.
  - primary can request and typ receive extensions indef
  - msgs piggybacked on heartbeats
- master may need to revoke a lease before expiry
  - eg. disable mutations on a file that is being renamed
- even if master loses comm with a primary, safe to grant a new lease to another replica after old lease expires.