Google File System

Main Structure

- Files are split into chunks of 64 MB that are distributed among chunkservers
    - Each chunk is replicated three times
- A single coordinator keeps track of the filesystem and redirects clients to tell them where the chunks they want are
- Reads are simple: coordinator tells client where the chunk is and then the chunk server goes and grabs it
    - This can read stale data if a chunkserver's location is cached by a client
    - Will be fixed when the chunkserver is garbage collected or the cache expires
    - This can go to any of the replicas

Writes

- When a write is wanted, a lease is granted to one of the replicas that makes it the primary
- Clients send data to all replicas and after they all confirm that they have received the data, the client issues a write to the primary
    - The primary chooses the order for all client writes and then sends the writes to the secondaries
    - Concurrent writes at the same time might come in and the replica just chooses some random order for them
- If any of the operations fail (i.e. a secondary does not respond / crashes), then the client is informed and tries again
    - Our consistency guarantees only hold if the write (after multiple tries) eventually succeeds

Consistency Guarantees

Two important definitions:

- Consistent - all clients will seem the same data regardless of which replica
- Defined - consistent **and** the mutation is seen in its entirety (not corrupted data)

Writes

- If only one client writes to a chunk at a time, we get consistency
- If we have multiple clients writing to a chunk, then we get consistency, but at chunk borders, we might get undefined regions
    - Writes are split up at chunk borders, so if the primary chooses a weird order, then we could get half of one write at one side of a chunk border and half of one write on the other side
    - Then we have some corrupted data where neither write has its full data put to the system
- Assuming our write operation succeeds, this will always result in one of these two and not inconsistent data

Atomic Appends

- For these appends, we just care that our data is added to the file **at least once**, we don't care if some replicas get more copies of the data than others
- With atomic appends, we always get defined and consistent regions interspersed with some inconsistent stuff
- If one replica fails the atomic append, then we just keep retrying and potentially add duplicates to the replicas that already have the data, but that's ok
- After the atomic append completes, then a whole bunch of space might be unusable but that's ok and GFS will handle that
- This may seem bad, but Google applications are built with this in mind and know that they have to filter duplicate data / do stuff

Handling Server Faults

Secondary Goes Down while Writing

- In the case that the server is revived quickly, then this can be fine
- If not, the primary will throw an error to the client
- The client can retry and just keep issuing the write
- If the server is permanently down, eventually the coordinator will notice (since lack of heartbeat)
    - Then the coordinator will remove the failed chunkserver, rereplicate, and then tells the primary a new list so the write will eventually succeed

Rereplicating Chunkserver

- There's a tradeoff between replicating a chunkserver and not because it's a lot of work (an hour or two) and so the coordinator may let the system operate without that chunk replica before declaring it dead
- But waiting too long might risk allowing all copies of the data to die

Primary Crashes

- Wait for the lease for that crashed primary to expire and then issue a new one
    - This is the whole point of the leases

Coordinator Crashes

- Coordinator saves critical state to disk to try to keep state intact in case of restart
- Or you can keep a backup coordinator that takes over when bad

Garbage Collection

- GFS does not immediately reclaim available storage, it instead does garbage collection
    - It renames a deleted file and after a few days it deletes this file (in the master file tree)
    - It periodic heartbeats with chunkservers, the master checks if any chunks don't have any files pointing to them
        - The chunkserver then deletes these
- If a chunkserver misses mutations, then it is considered stale and will be garbage collected as soon as the master notices its version number is out of date
    - This can occur if a write fails

Takeaways

- GFS sacrifices consistency for simplicity
    - In retrospect a bit too much
- Significantly simpler also because of single coordinator but this adds a bottleneck
- Optimized for very large files but when there were a lot of files / smaller files then the system became pretty bad when the coordinator ran out of RAM
- If clocks don't match though, this can be a problem with leases
    - Loss of performance if coordinator runs slower than primary
    - Loss of correctness if coordinator runs faster than primary