Graduate Operating Systems

Spring 2023

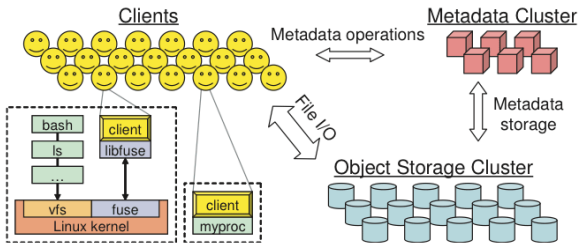
1

Paper “Ceph”

- *What are the goals of Ceph?*
- Object Storage Devices
- Metadata Servers
- *Why does OSD suffer from scalability problems?*

2

Paper “Ceph”



- **Clients:** near-POSIX system interface
- **OSD:** data + metadata
- **Metadata server cluster:** manages namespace, security, consistency, coherence

3

Paper “Ceph”

- **Decoupled data & metadata**
  - Metadata ops managed by metadata server cluster
  - No alloc lists, instead distribution function
- **Dyn. distr. metadata mgt**.
  - Dynamic subtree partitioning
- **Reliable autonomic distr. object storage**
  - Data migration, replication, failure detection & recovery done by OSD cluster (high-level: single logical object store)

4

Paper “Ceph”

- File: inode + capabilities + striping strategy for file
- Striping strategies:
  - Map file data onto sequence of objects
  - Object names combine file inode# and stripe number; mapping done by CRUSH algorithm
  - **Locate objects without need for alloc table**

5

Paper “Ceph”

- Client synchronization:
  - Multiple clients: revoke caching/buffering capabilities (synchronous I/O): SLOW
  - Ability to relax consistency in some situations
    - O_LAZY flag, lazyio_propagate, lazyio_synchronize
- Namespace operations:
  - Optimize certain scenarios (e.g., readdir followed by stat (ls -l operation)) using caching

6

Paper “Ceph”

- Metadata storage (load balancing)

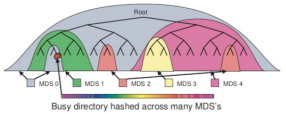
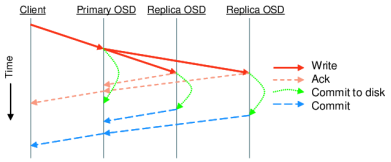


Figure 2: Ceph dynamically maps subtrees of the directory hierarchy to metadata servers based on the current workload. Individual directories are hashed across multiple nodes only when they become hot spots.

- Replication
  - Primary, replicas



7

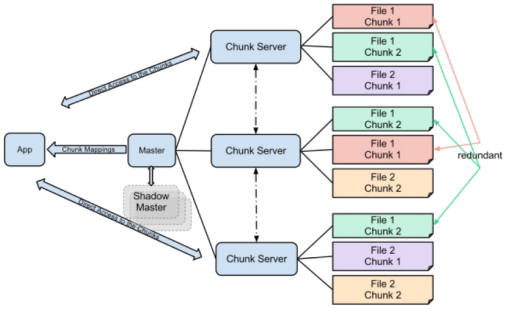
Paper “Google FS”

- Assumptions:
  - Component failures are the norm, not the exception
  - Files are huge by traditional standards
  - Most file updates are append-only
  - *How does this compare to previous papers?*
- System is built from many inexpensive commodity components
- System will store a modest number of large files

8

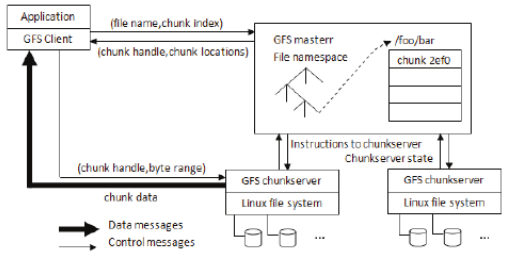
Paper "Google FS"

- Non-POSIX, "snapshot", "record append"



9

Paper "Google FS"



- Chunk size = 64MB
  - Lazy space allocation; throughput; persistent TCP connection; reduced metadata; hot spots

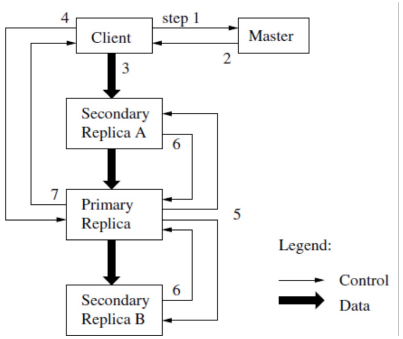
10

Paper "Google FS"

- Metadata:
  - File and chunk namespaces; mapping from files to chunks; locations of chunks (primary, replicas)
  - Cached in memory
  - Chunk locations requested at startup and periodically
- Keeping things consistent:
  - GFS applies mutations to a chunk in the same order on all replicas; chunk version# used to detect stale replicas
  - Leases to primary, which picks serial order

11

Paper "Google FS"



12

Paper “Google FS”

- Snapshot: copy of file
- Record Append: append data concurrently

- Snapshot */home/user -> /save/user*
- Create file */home/user/foo*

13

Paper “Google FS”

- Leases, heartbeat messages, namespace management
- Re-replication, rebalancing, garbage collection
- Results in Figure 3

- *How does it compare to Ceph?*
- *Why does Google FS avoid file caching?*
- *What are the pros/cons of large chunk sizes?*

- Google “Colossus” (2010)
- “Single point of failure”
- https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system

14