High performance Cloud native Distributed storage system

https://www.opencurve.io/

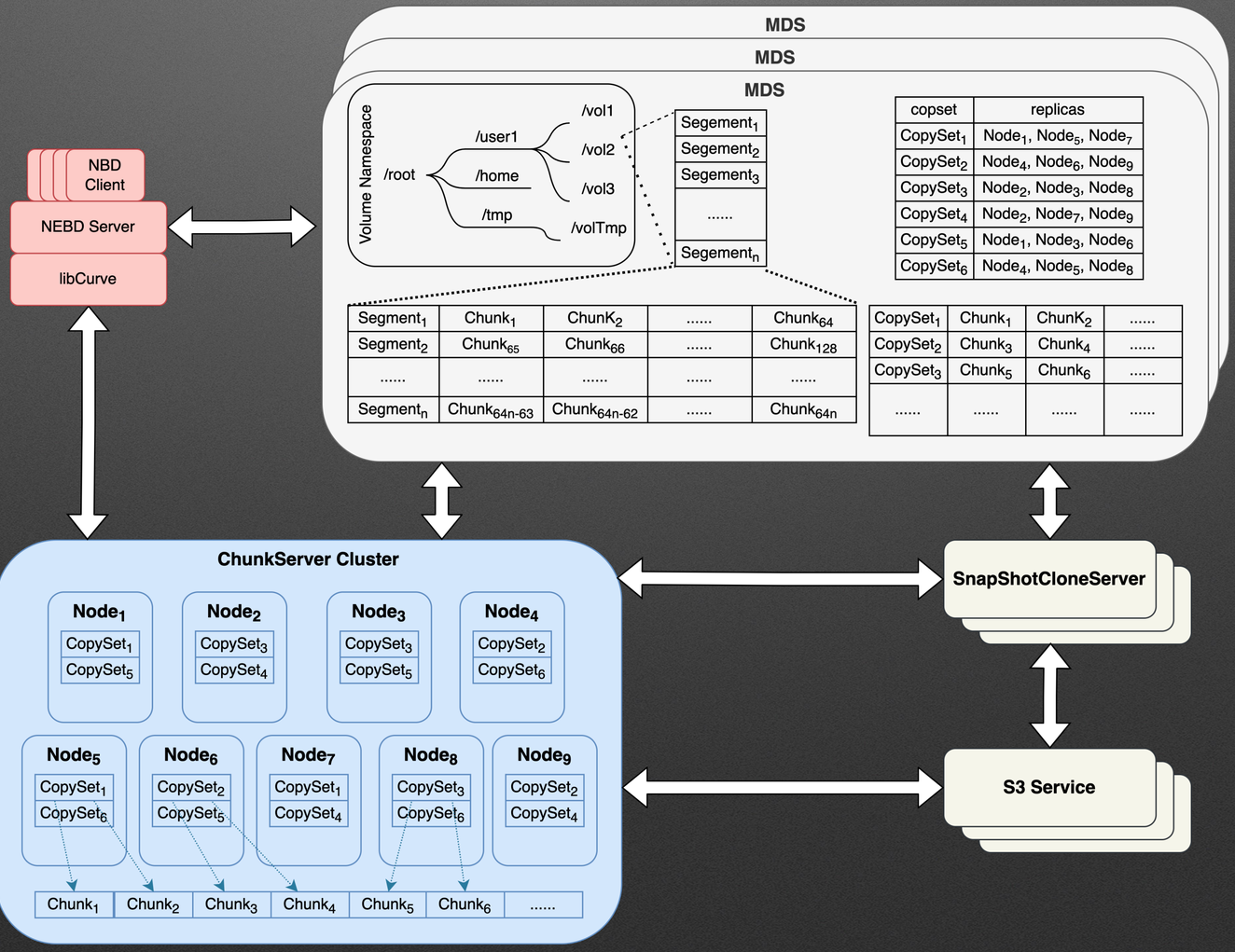
Agenda

- CurveBS Architecture
- CurveBS Topology
- CurveBS Data Organization
- MetaData Server (MDS)
- ChunkServer
- Client
- CurveBS IO processing flow
- CurveBS Performance considerations
- Cloud Native

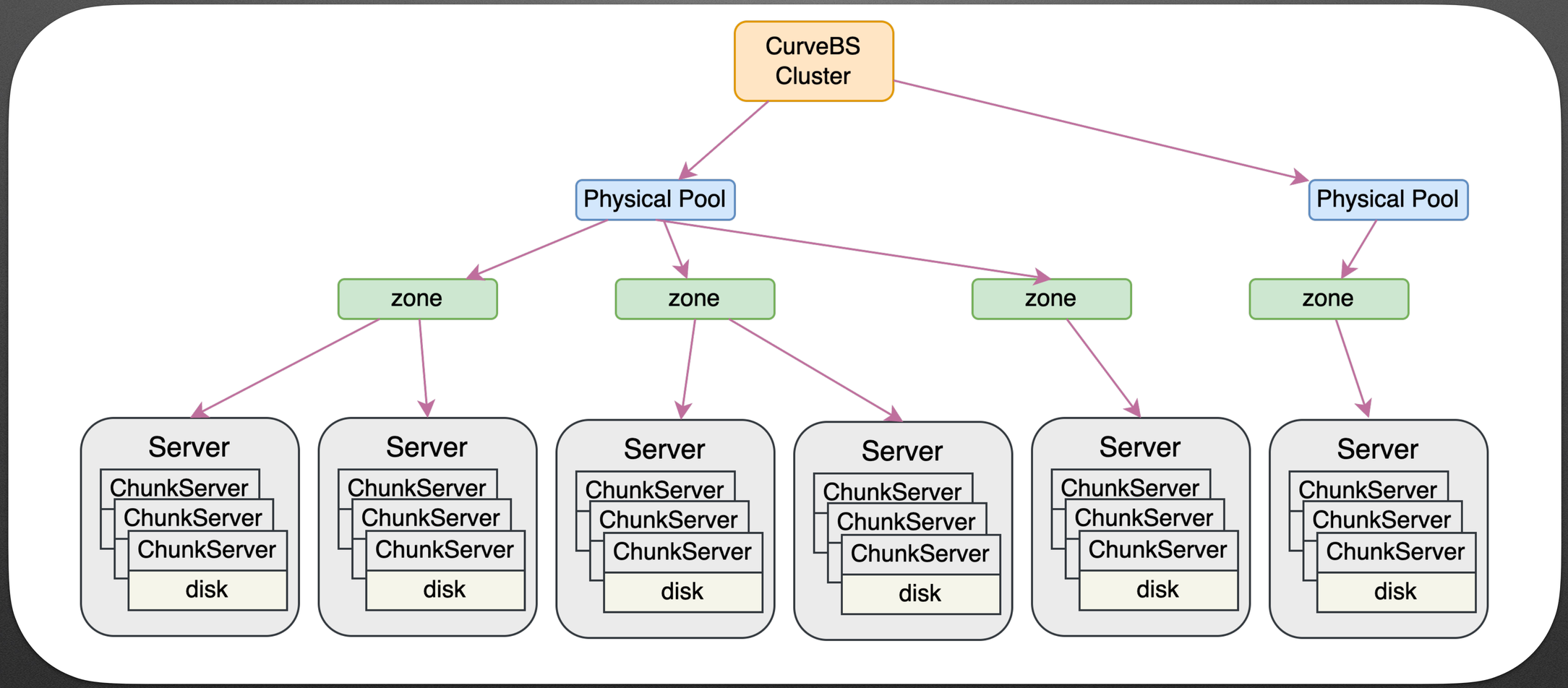
Agenda

- CurveFS Architecture
- CurveFS Data Organization
- CurveFS file Organization
- CurveFS MetaServer
- CurveFS Client
- CurveFS MKNode Flow
- CurveFS Write to S3 Flow

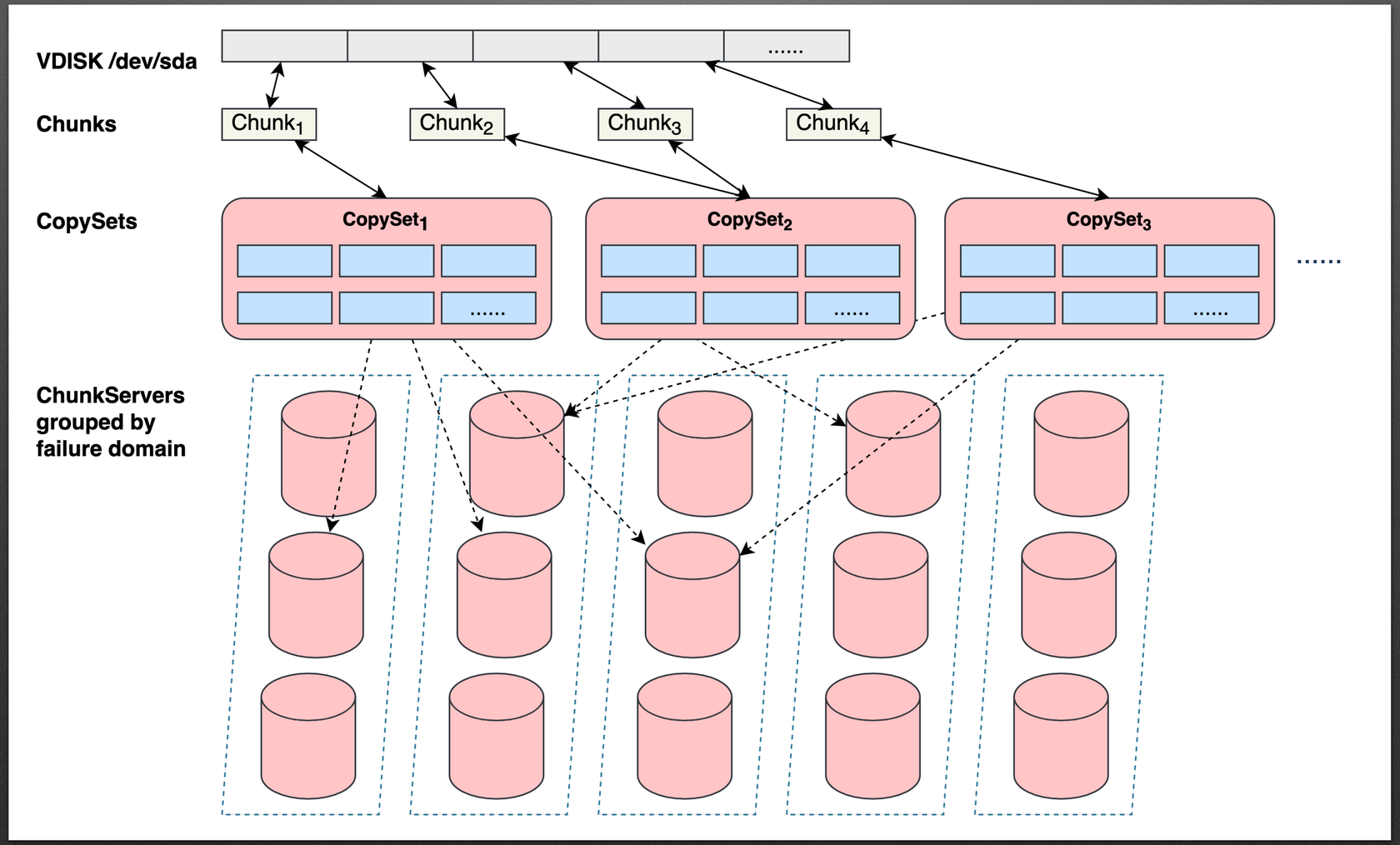
CurveBS Architecture



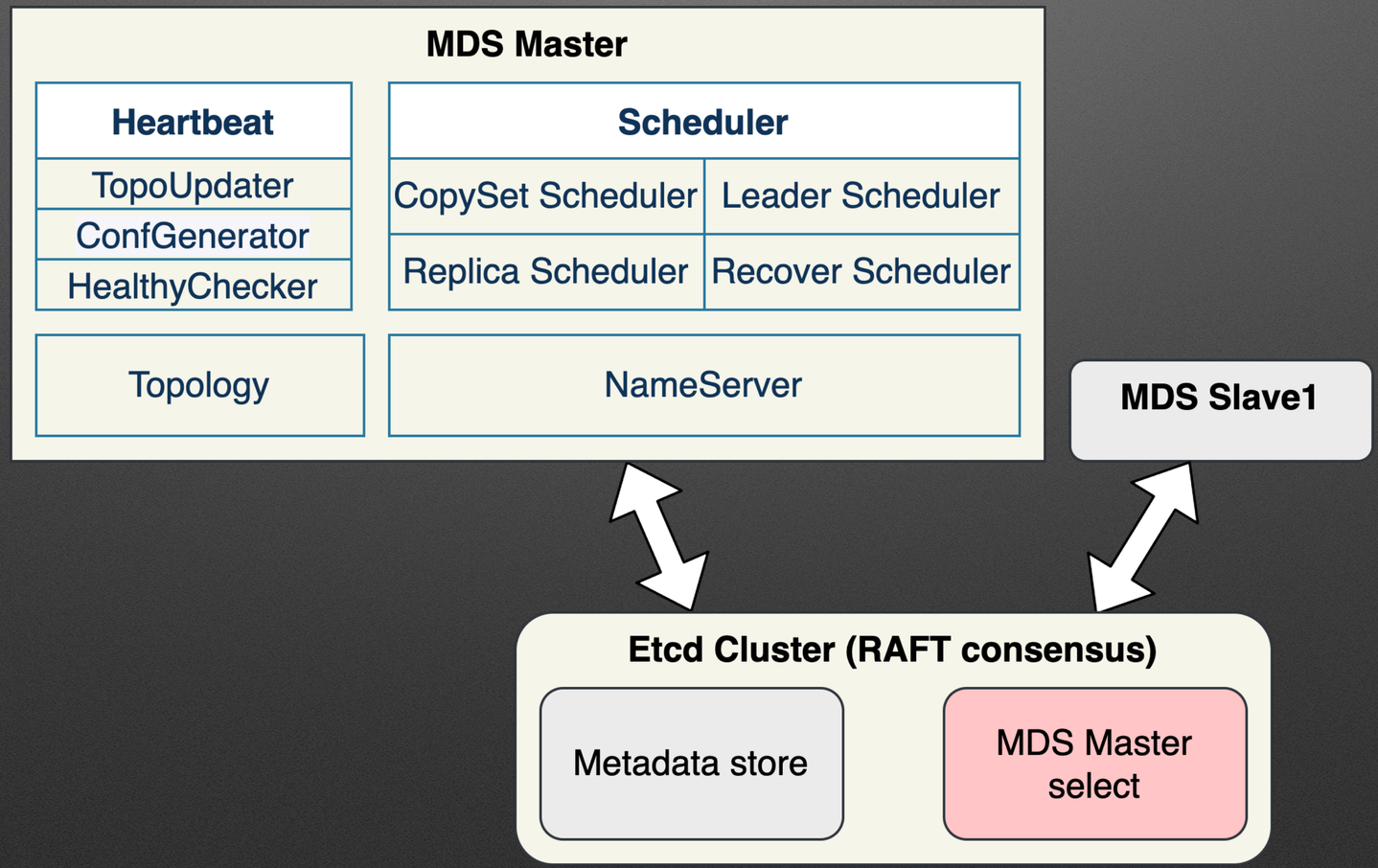
CurveBS Topology



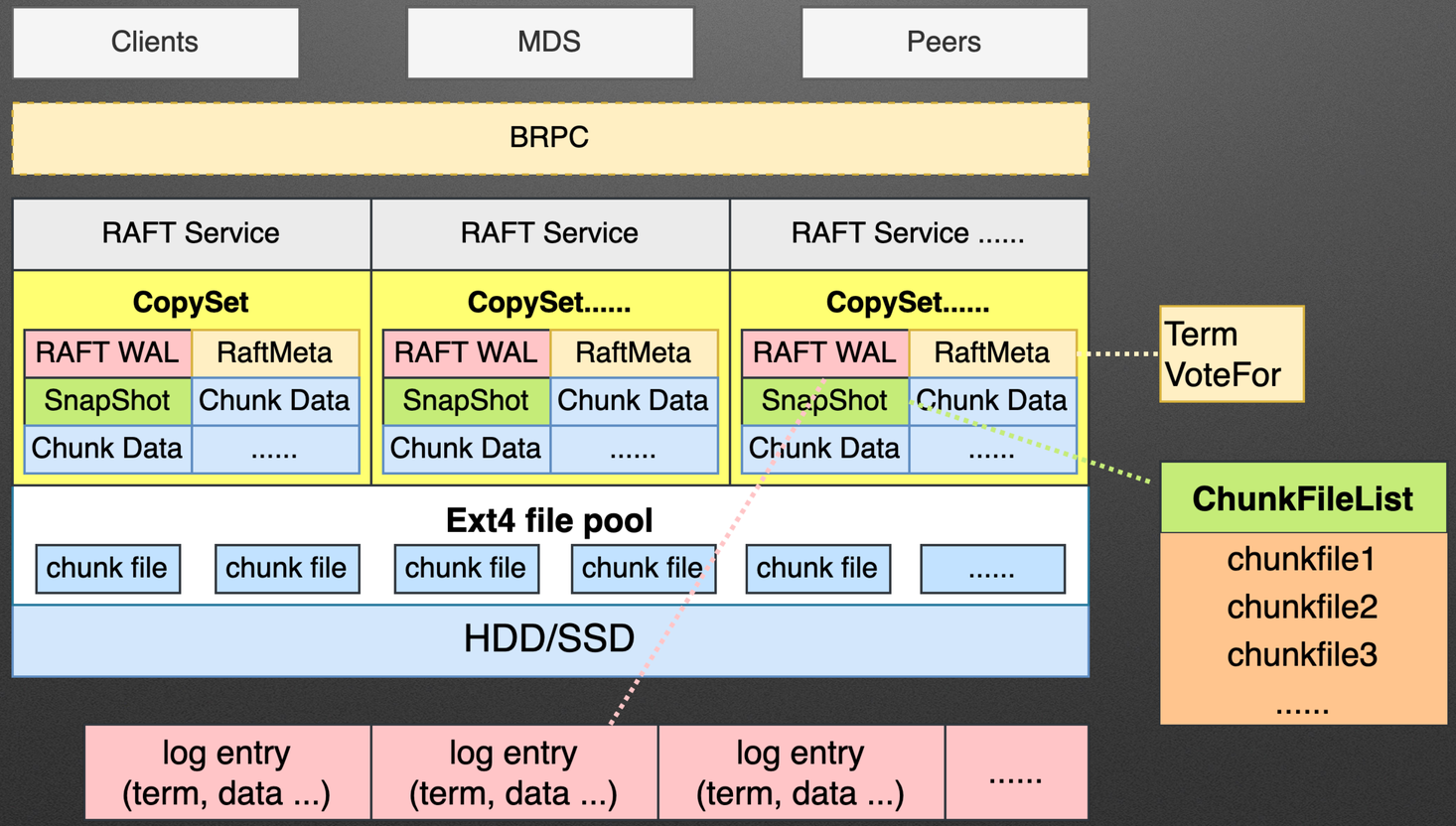
CurveBS Data Organization



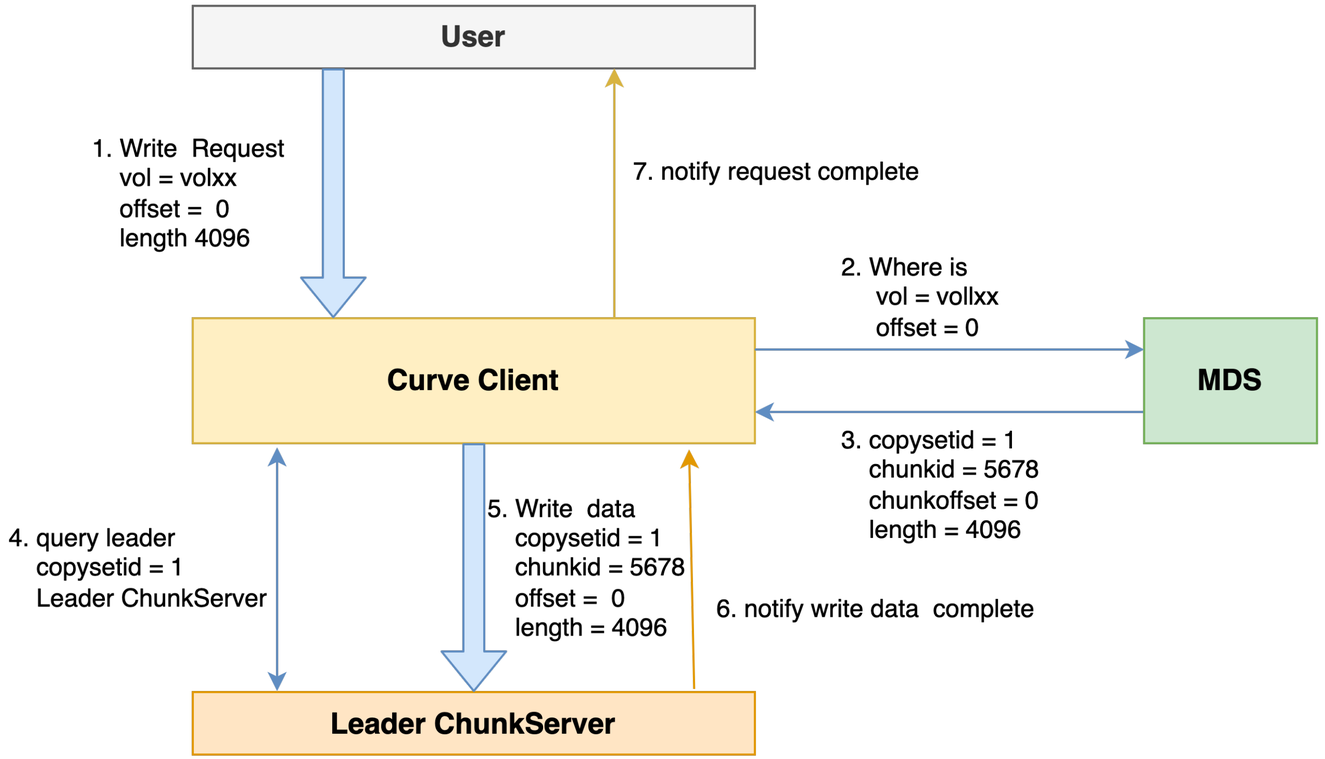
MetaData Server(MDS)

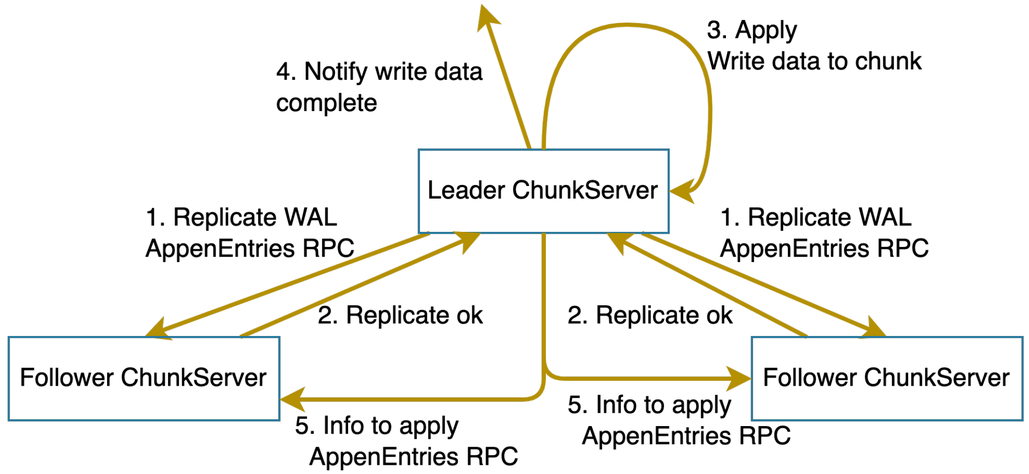


ChunkServer

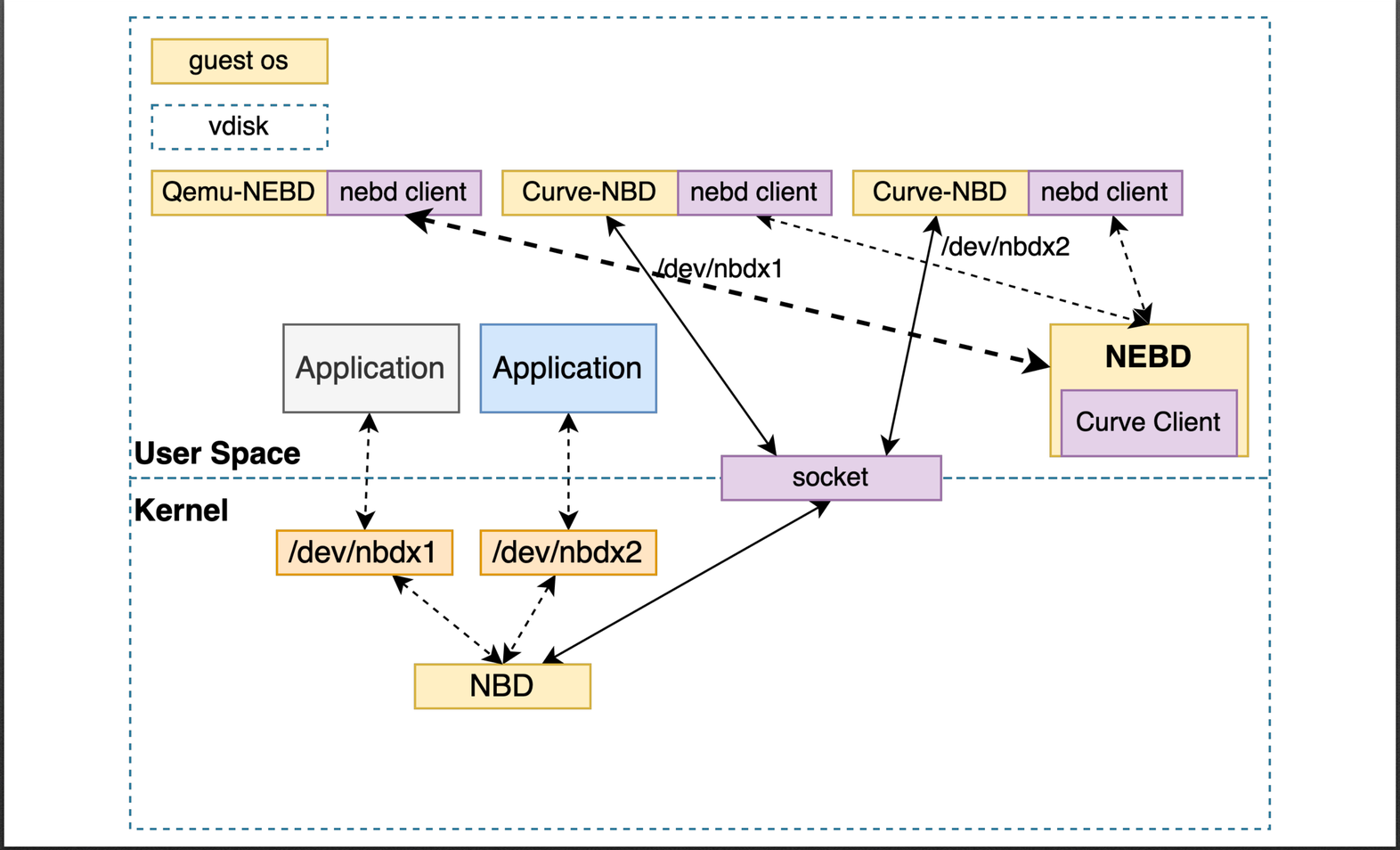


CurveBS I/O Flow





Client


guest os
vdisk
Qemu-NEBD
nebd client
Curve-NBD
nebd client
Curve-NBD
nebd client
/dev/nbdx1
/dev/nbdx2
Application
Application
NEBD
Curve Client
User Space
socket
Kernel
/dev/nbdx1
/dev/nbdx2
NBD

CurveBS Considerations

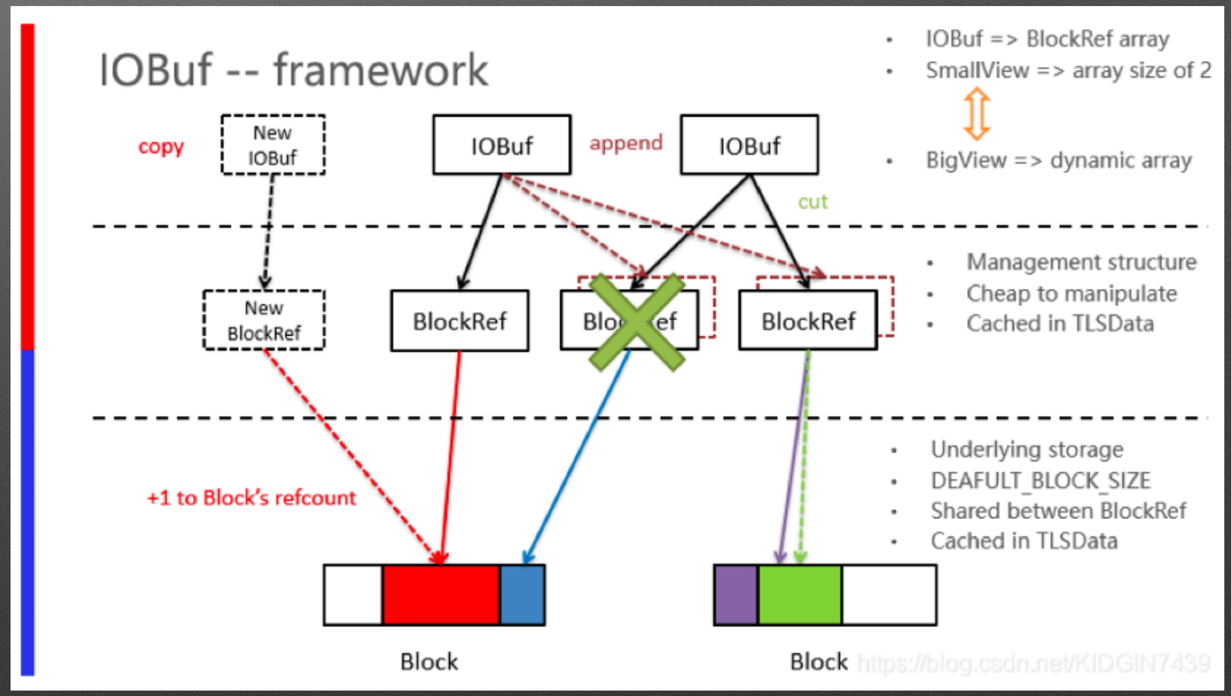
yanjiangxu commented on 10 May 2017 • edited

The behavior In original flow:
1) In the case of 3 copies, the client sends the write to the primary, and then the primary forwards to the other two replicas. The primary must wait for the commit until all the replicas from completion. The real write IO latency is the longest latency among three copies.
2) When the data distribution is not uniform, the osd node which owns hottest data will become the bottleneck of total latency.

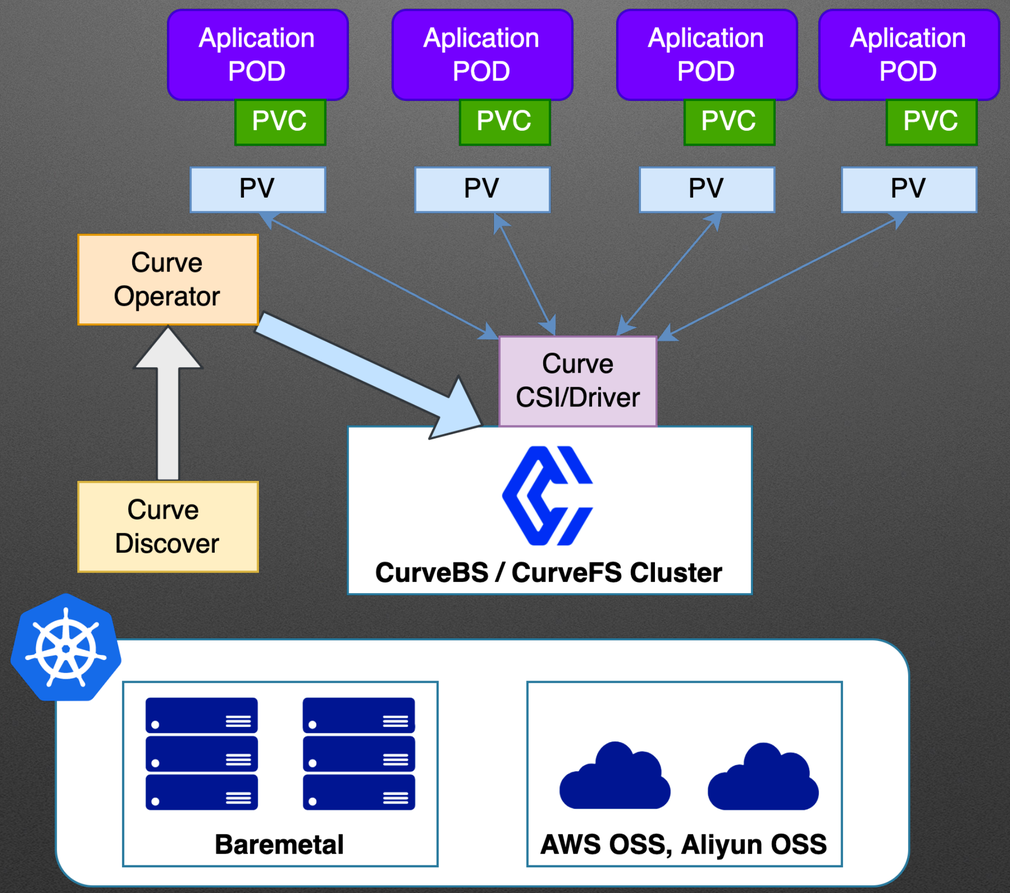
The behavior In my enhancement:
commit_majority function is not required to wait for all copies of commit to complete: as long as the majority of the commit is completed, the cluster would notify the client to complete.

As in my implementation:
We also follow the scenario: When the number of copies is less than min_size, ceph does not provide read and write capabilities. In order to ensure security, you must ensure that the number of commit to majority.
And I defined two important variables:
`allow_uncommitted_replicas = pool_min_size - 1,`
`majority = pool_default_size - allow_uncommitted_replicas.`

Test case:
1) 3 copies, pool_default_size = 3, pool_min_size = 2, majority = 2,allow_uncommitted_replicas=1
2) The average latency decreased by **11.5%**;
3) Long tail (99.9%) latency decreased by **71.59%**;
4) IOPS increased by **13%**.



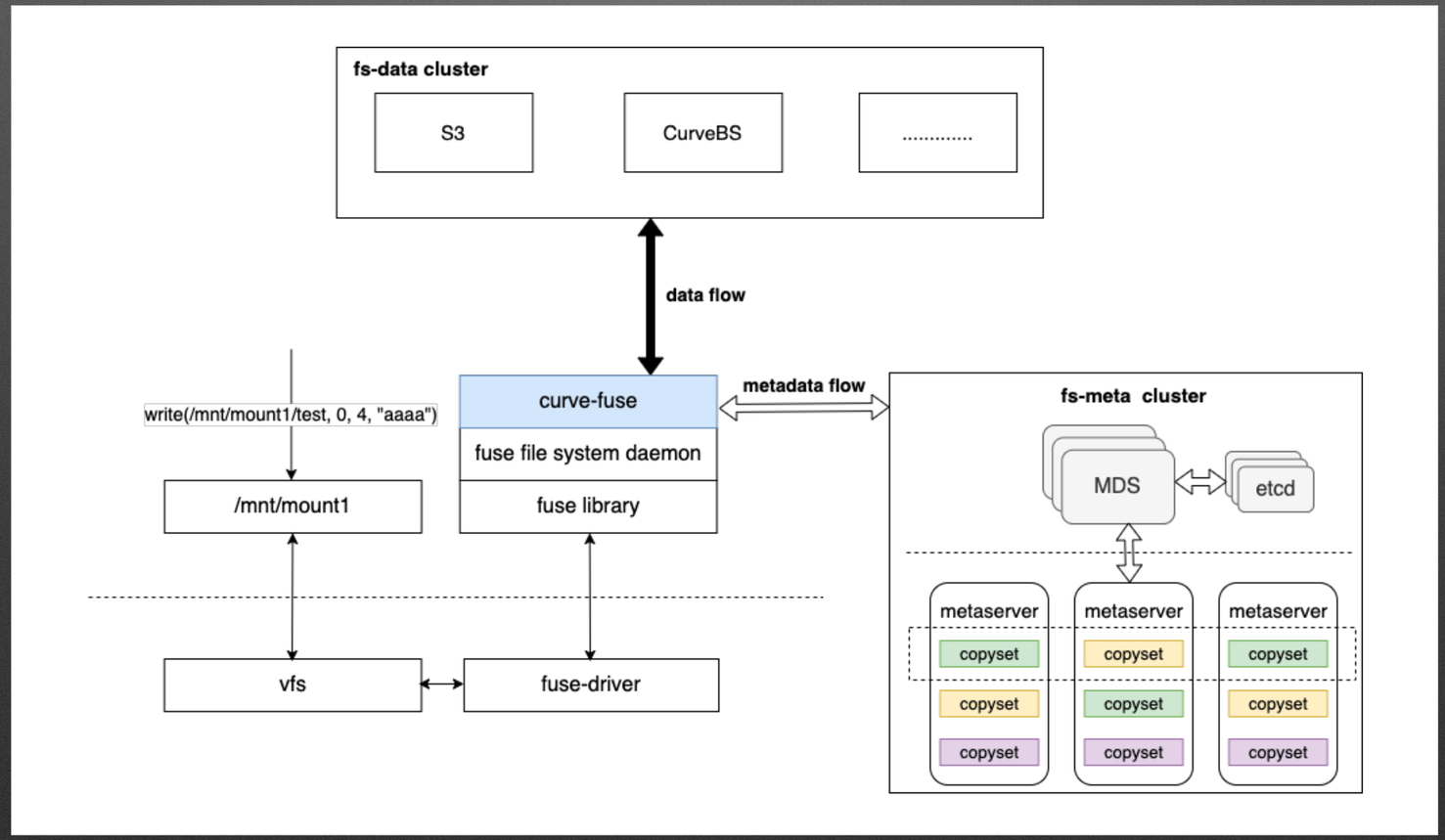
Cloud Native



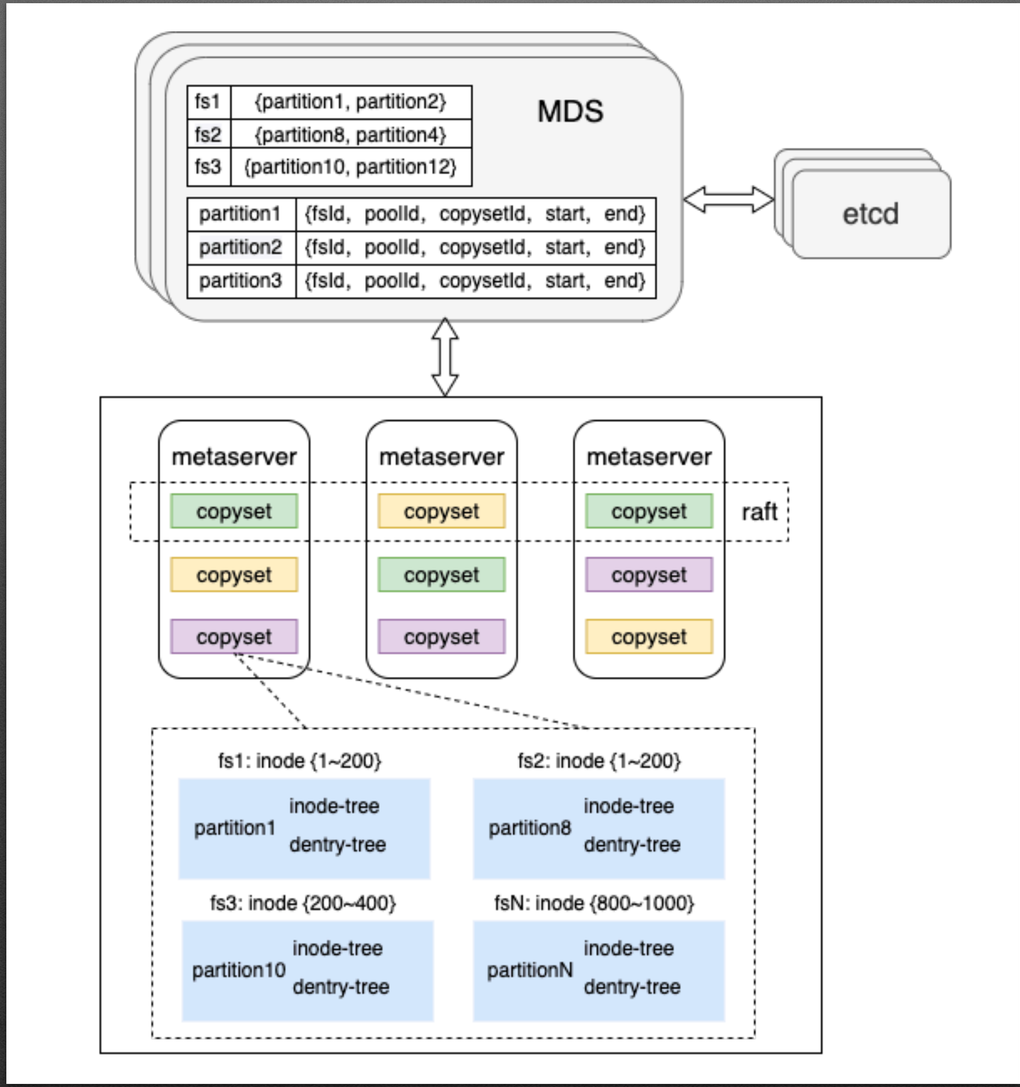
CurveBS Roadmap

- MultiRaft optimization
  - ParallelRaft for write
  - Reduce write magnification for file new write
- Cooperate with Alibaba to support high-performance polardb for postgresql using CurveBS
- Cloud-native support for CurveBS
  - CurveBS clusters and related monitoring services can be configured using CRDs in public cloud and on-premises environments
  - Use the Curve operator to install, upgrade, backup, and expand CurveBS clusters
  - Use Curve Discover to discover resource changes and to collect the system’s runtime status to operator

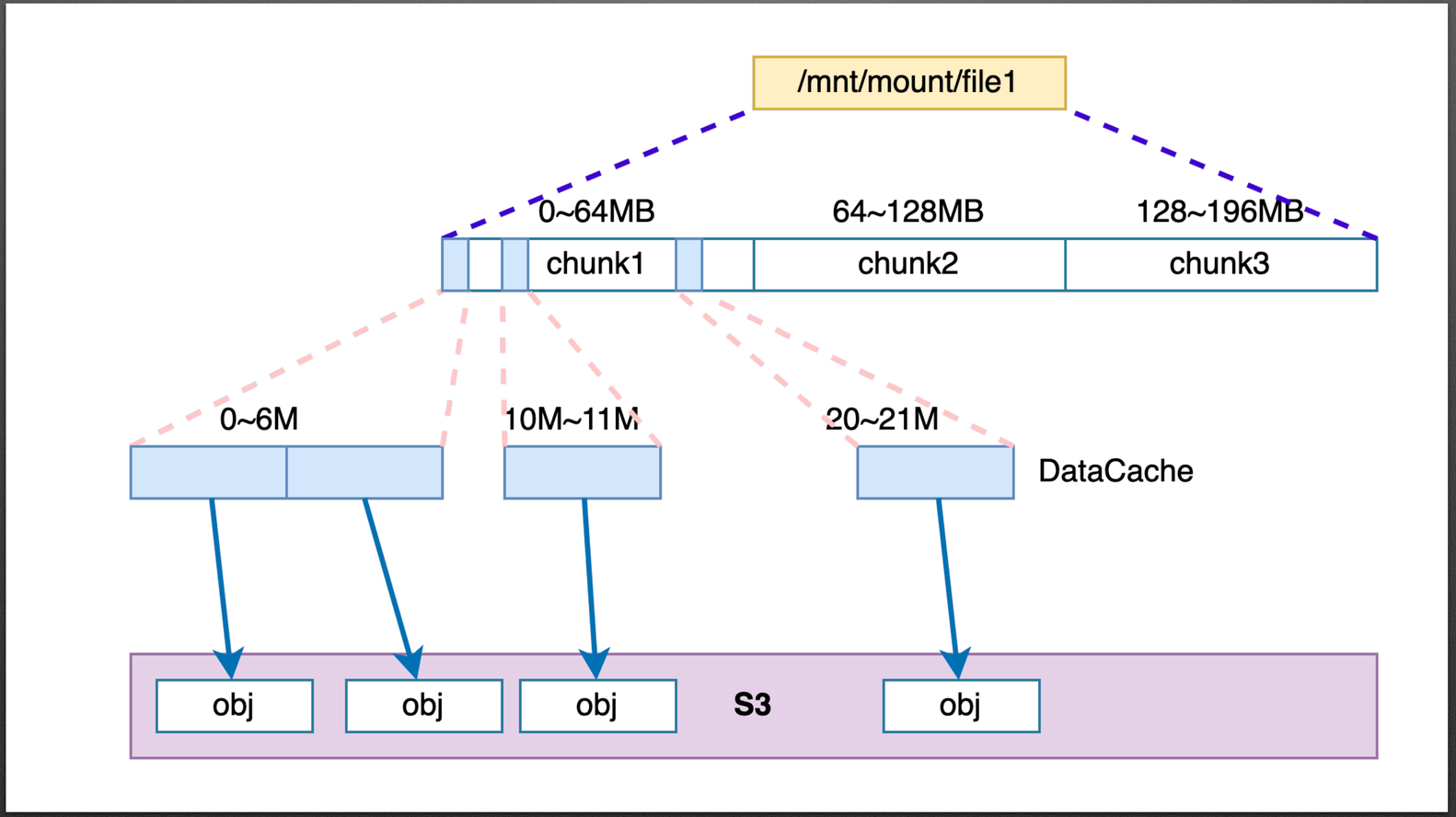
CurveFS Architecture



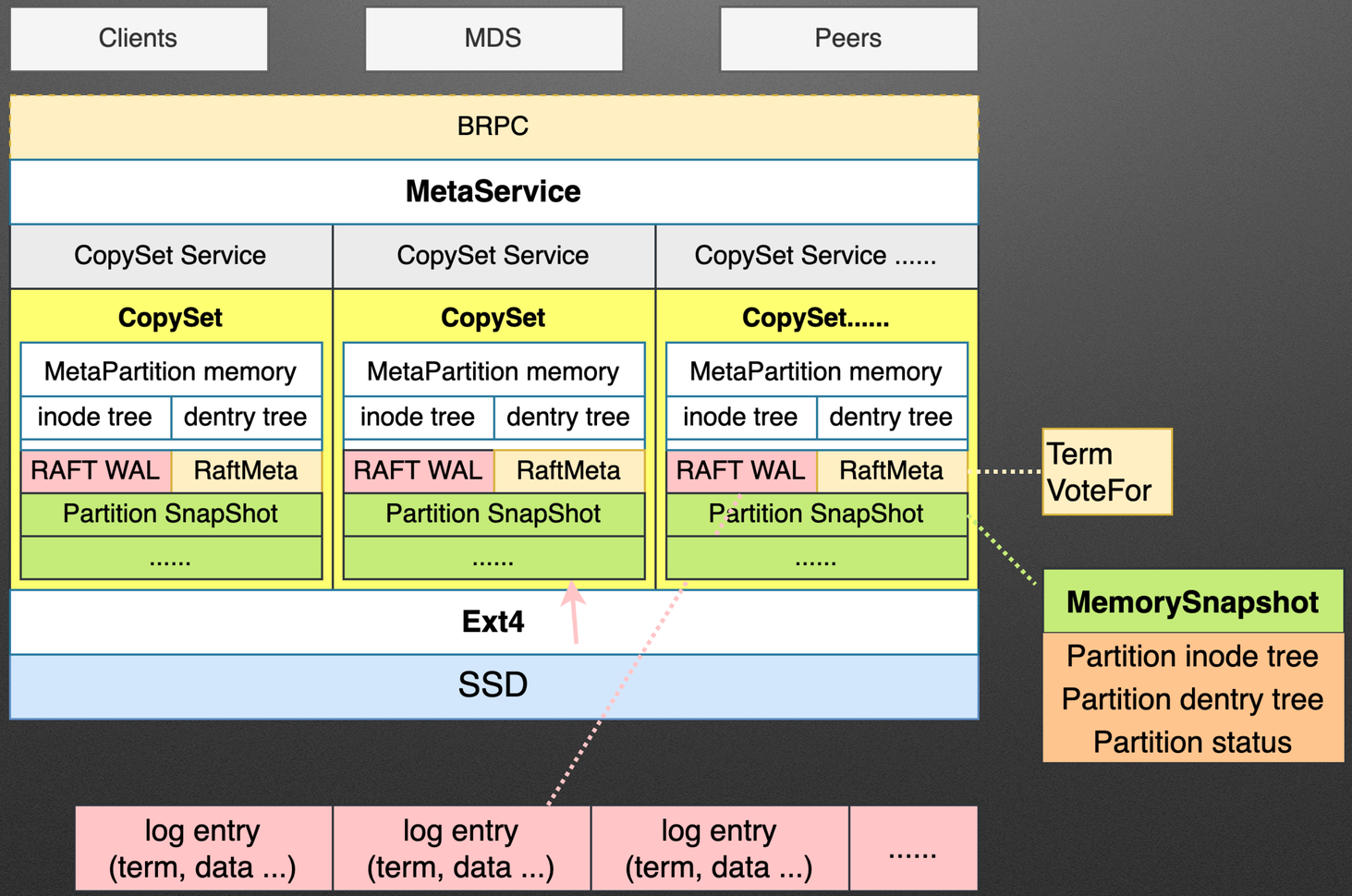
CurveFS Data Organization



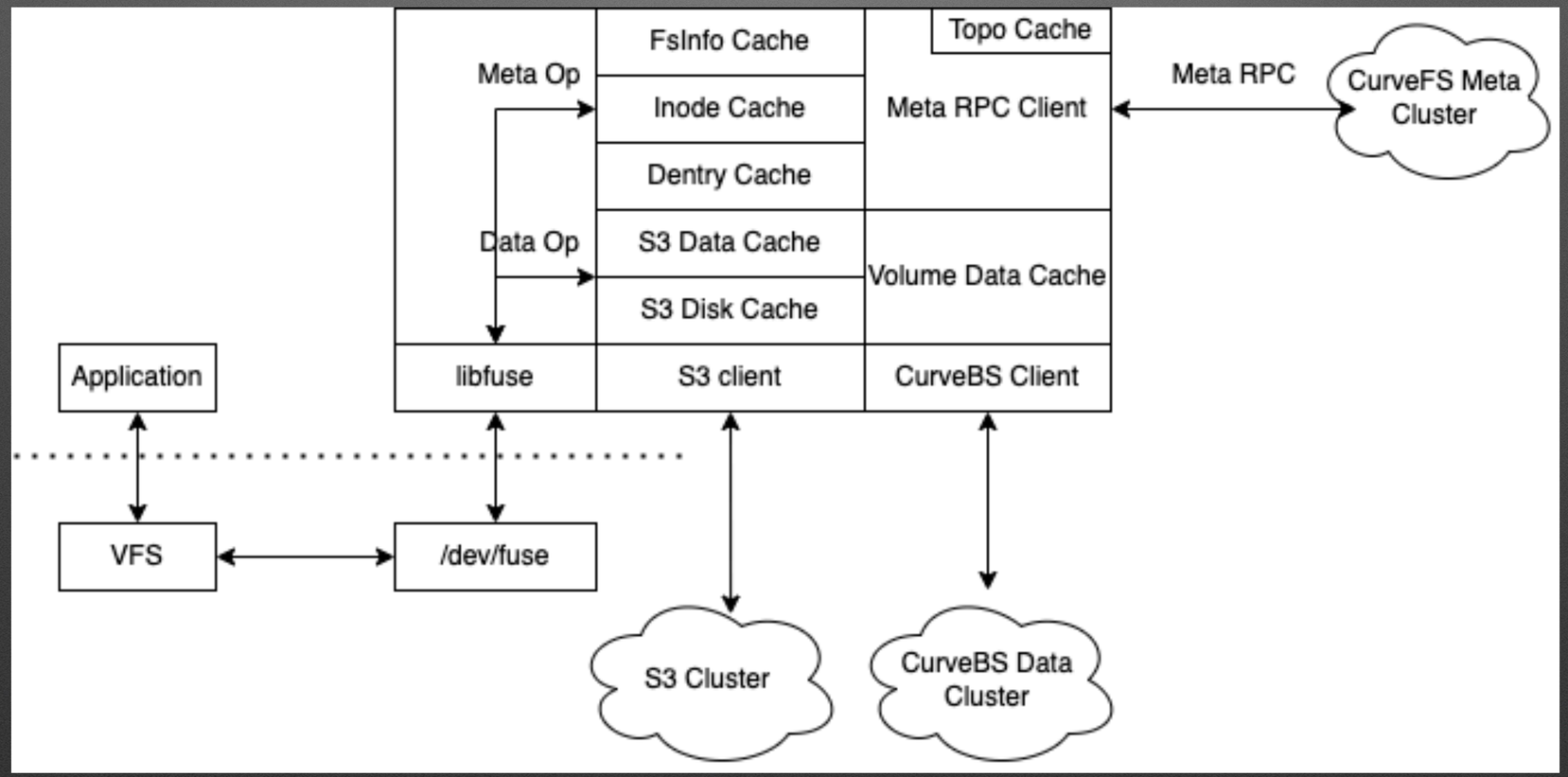
CurveFS File Organization



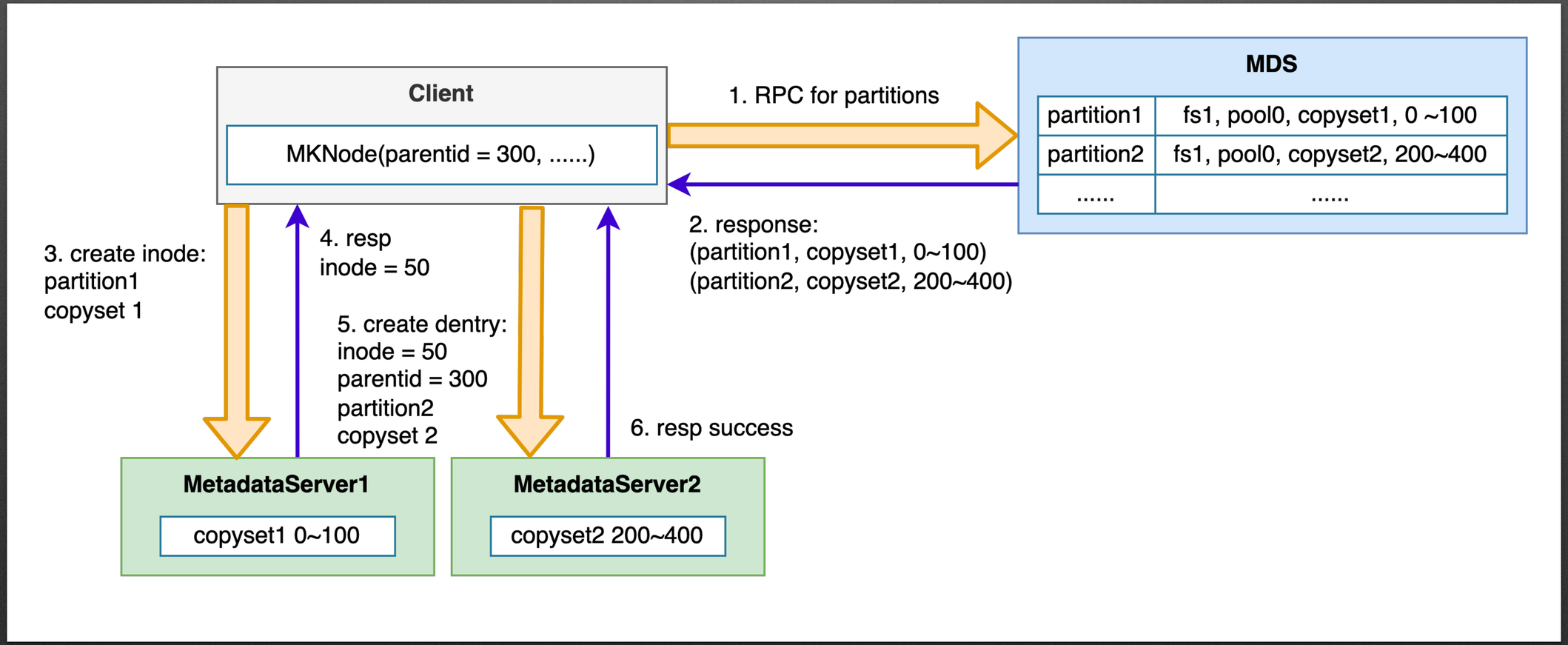
CurveFS Metadata Server



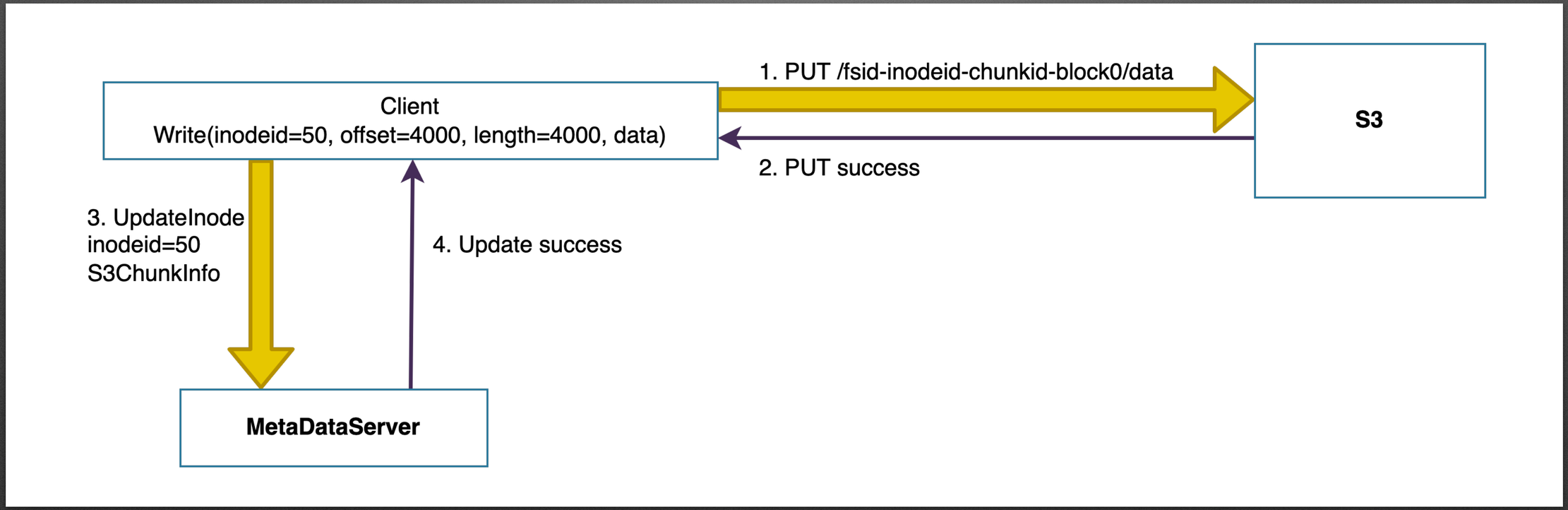
CurveFS Client



CurveFS Mknod Flow



CurveFS Write S3 Flow

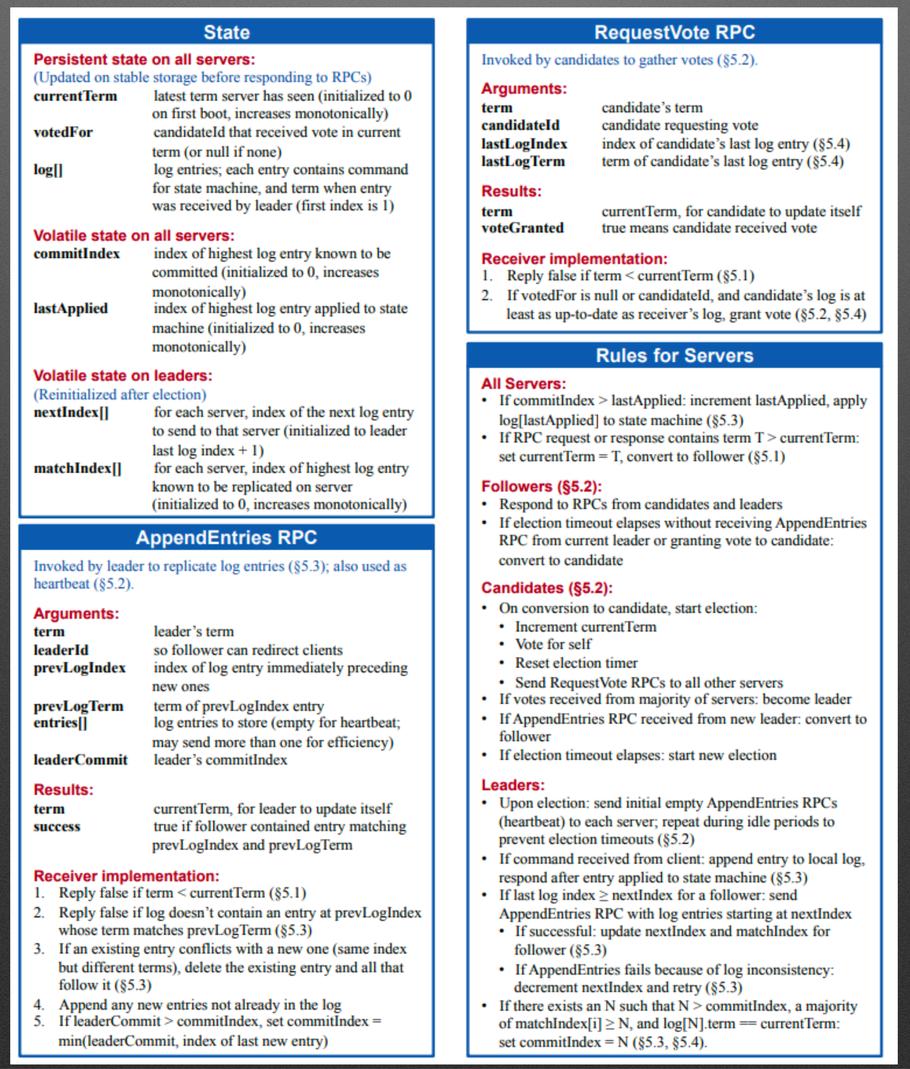


CurveFS Roadmap

- CurveFS based on CurveBS
- Cache support on CurveFS
- NFS support on CurveFS
- Cloud tiering support
- Cloud-native support for CurveFS
  - CurveFS clusters and related monitoring services can be configured using CRDs in public cloud and on-premises environments
  - Use the Curve operator to install, upgrade, backup, and expand CurveFS clusters
  - Use Curve Discover to discover resource changes and to collect the system’s runtime status to operator

thanks

About RAFT



State

Persistent state on all servers:
(Updated on stable storage before responding to RPCs)

currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)

Volatile state on all servers:

commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied	index of highest log entry applied to state machine (initialized to 0, increases monotonically)

Volatile state on leaders:
(Reinitialized after election)

nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

Rules for Servers

All Servers:

- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):

- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Leaders:

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).