Curve

# Curve核心组件之Client

DIGITALSAIL

吴汉卿

网易数帆存储团队

# CURVE

CURVE是高性能、高可用、高可靠的分布式存储系统

- 高性能、低延迟存储底座
- 可扩展存储场景：块存储、对象存储、云原生数据库、EC等
- 当前实现了高性能块存储，对接 OpenStack 和 k8s
  - 网易内部线上无故障稳定运行400+天
- 已开源
  - github主页： https://opencurve.github.io/
  - github代码仓库： https://github.com/opencurve/curve

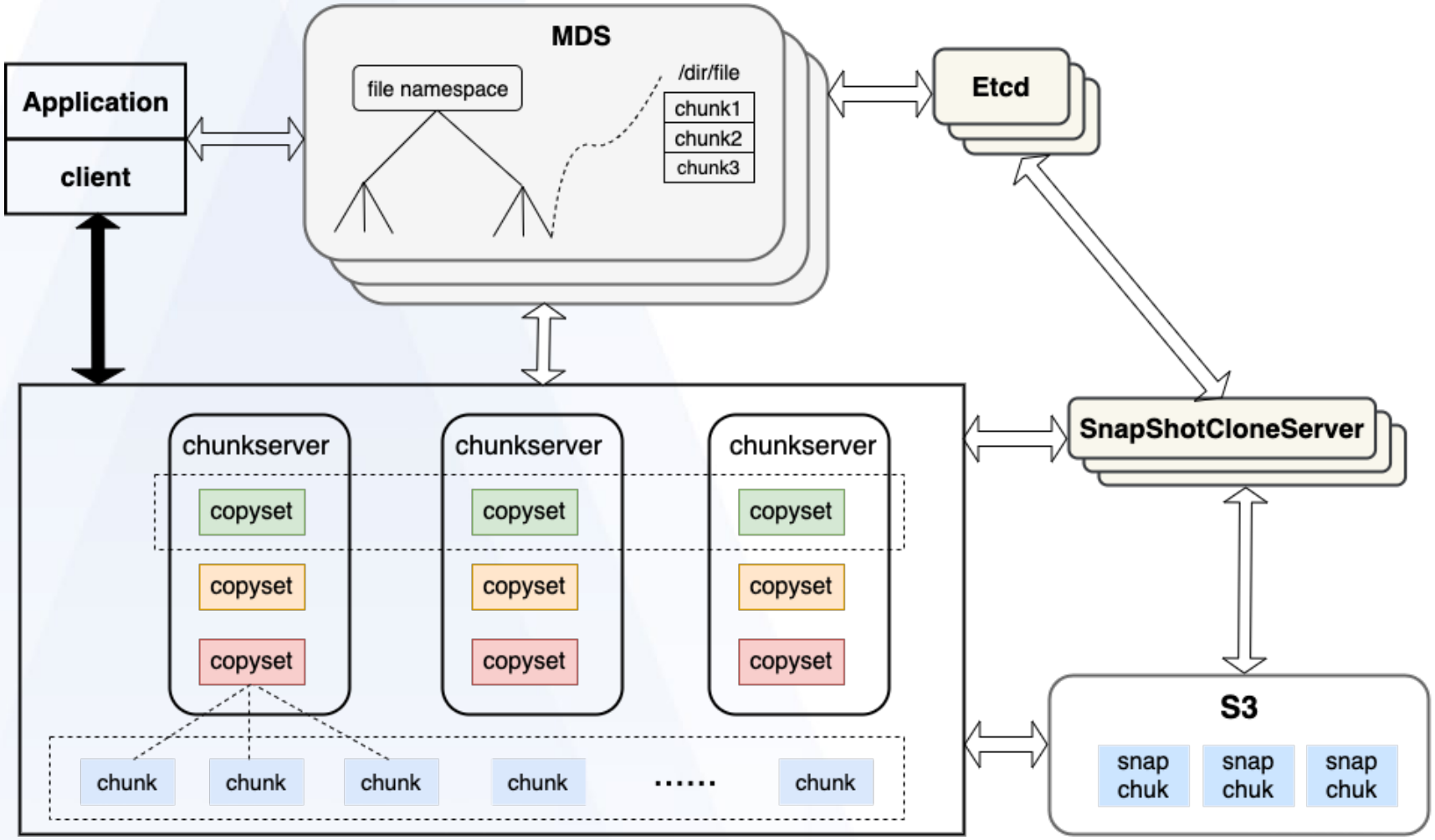
Curve

DIGITAL SAIL

# 目录

# CURVE基本架构

- 元数据节点 MDS
  - 管理和存储元数据信息
  - 感知集群状态，合理调度
- 数据节点 Chunkserver
  - 数据存储
  - 副本一致性，raft
- 客户端 Client
  - 对元数据增删改查
  - 对数据增删改查
- 快照克隆服务器

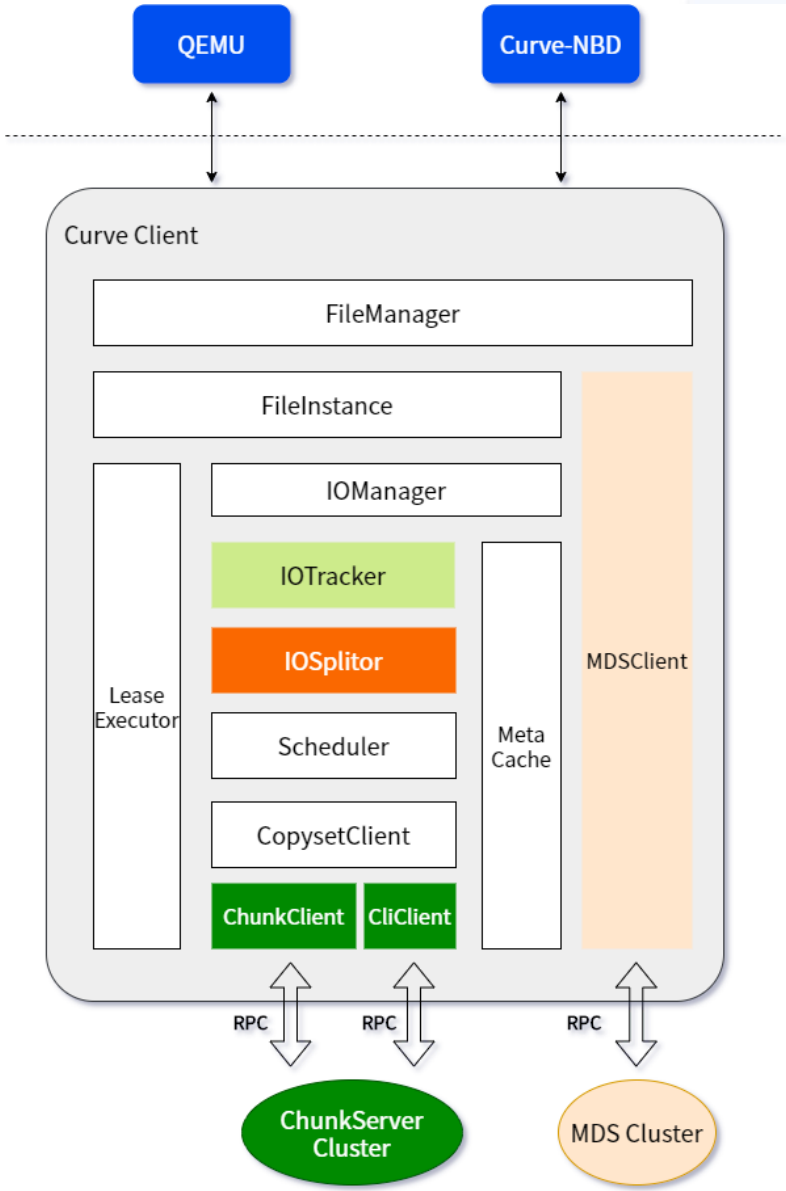


Curve

DIGITAL SAIL

# 目录

# CLIENT整体架构




- QEMU、Curve-NBD：上层应用
  - 通过链接curve-client使用curve提供的服务
- FileManager：提供接口，记录已挂载卷
- FileInstance：对应一个已挂载的卷
- LeaseExecutor：负责定期与MDS通信，获取卷的元数据信息
  - 元数据信息在打快照时会进行变化
- MetaCache：元数据缓存

- IOTracker：跟踪一个上层IO请求
- IOSplitor：IO转换拆分

- ChunkClient、CliClient：与Chunkserver进行通信
  - 前者负责IO请求
  - 后者负责获取复制组(copyset)的leader
- MDSClient：负责与MDS交互，挂卸载卷、获取元数据信息

# CLIENT上层应用

Curve

QEMU：

实现了QEMU block与Client的对接层

向cinder/glance提供了Python API

https://github.com/opencurve/curve-qemu-block-driver

NBD：

实现了Curve-NBD，与内核NBD模块进行交互

可以作为容器的数据存储

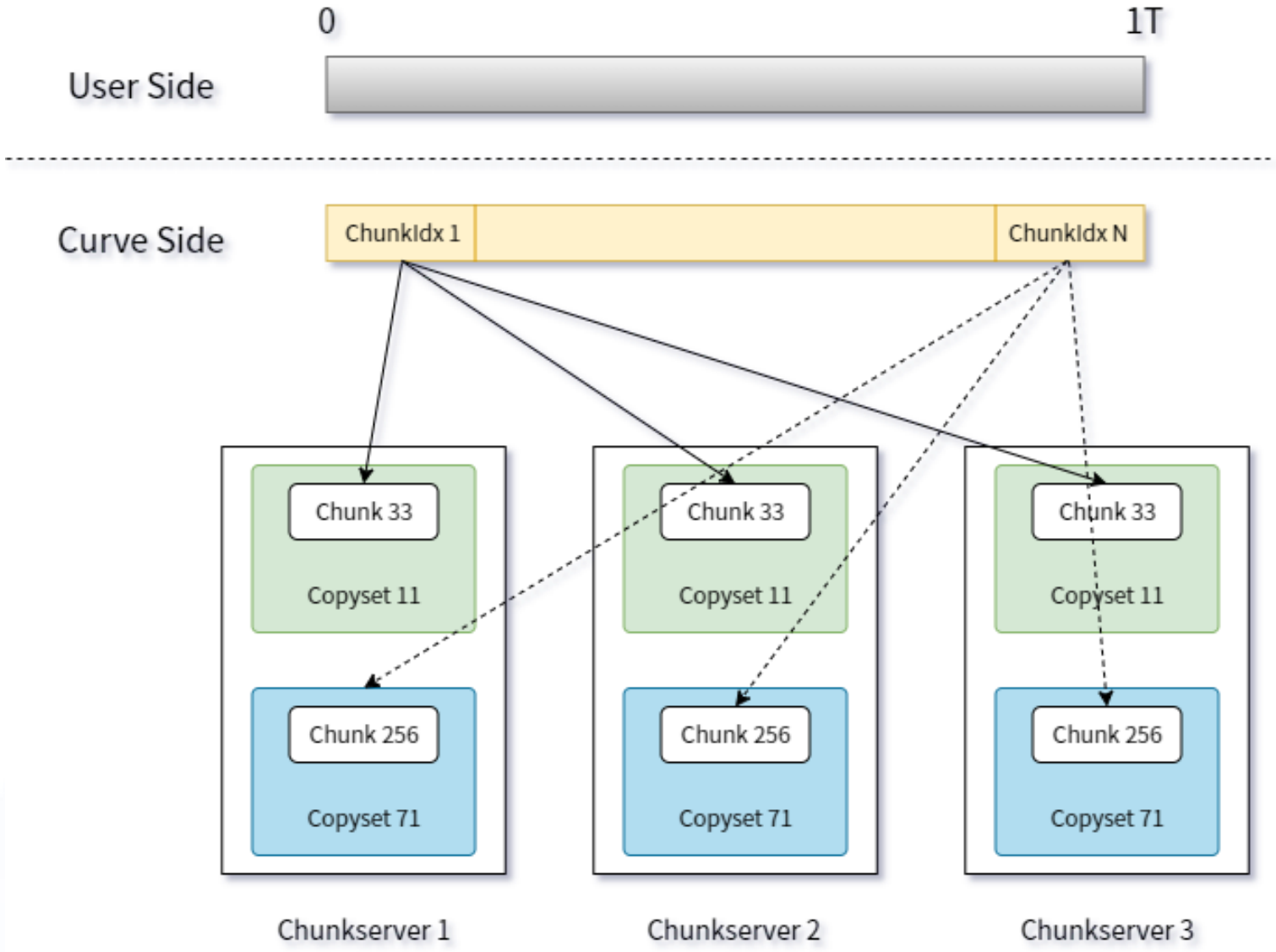
CSI插件也已经开源：

https://github.com/opencurve/curve-csi

```
username@hostname:~$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    254:0    0  20G  0 disk
└─vda1 254:1    0  20G  0 part /
vdb    254:16   0  64M  1 disk
vdc    254:32   0  10G  0 disk
```

```
username@hostname:~$ lsblk | grep nbd
nbd0    43:0     0  100G  0 disk
nbd1    43:256   0  100G  0 disk
nbd2    43:512   0  100G  0 disk
nbd3    43:768   0  100G  0 disk
```

# CLIENT虚拟块设备



# CLIENT主要功能

- 提供接口
  - 数据面：AioWrite/AioRead、Write/Read
  - 控制面：Create/Delete、Open/Close、Rename等
- IO处理：转换、拆分、合并
- 元数据获取及缓存
  - 逻辑chunk与物理chunk映射关系
  - 物理chunk所属的复制组(copyset)
  - 复制组所在的chunkserver列表
  - 复制组的leader信息
- Failover支持
  - MDS：只有主MDS才会监听端口
  - ChunkServer：通过raft维护复制组内的主-从关系

# CLIENT IO流程

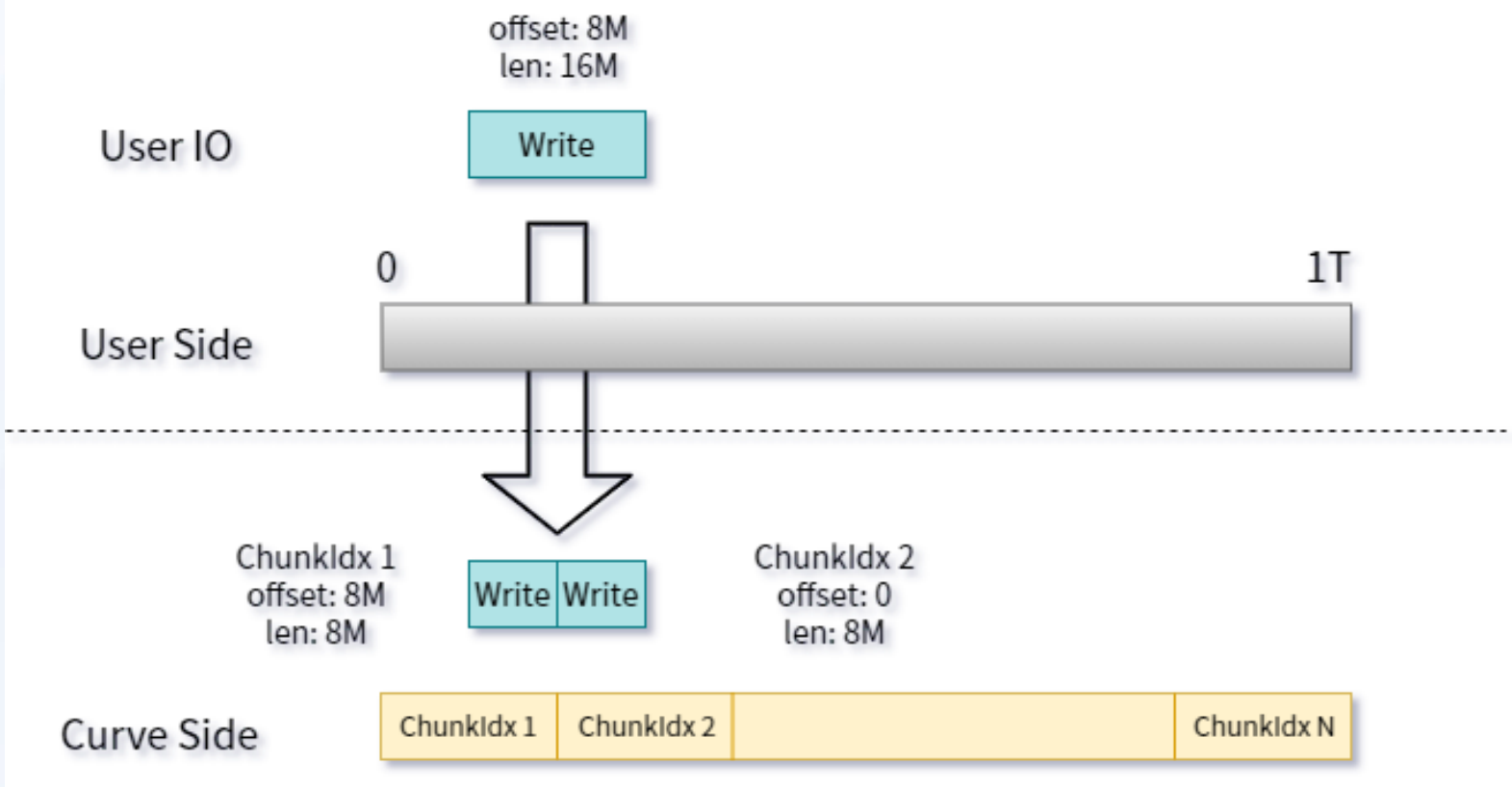
用户下发一个写请求

off: 8M

len: 16M

请求落在两个逻辑chunk上，所以请求会被拆分成两个子请求：

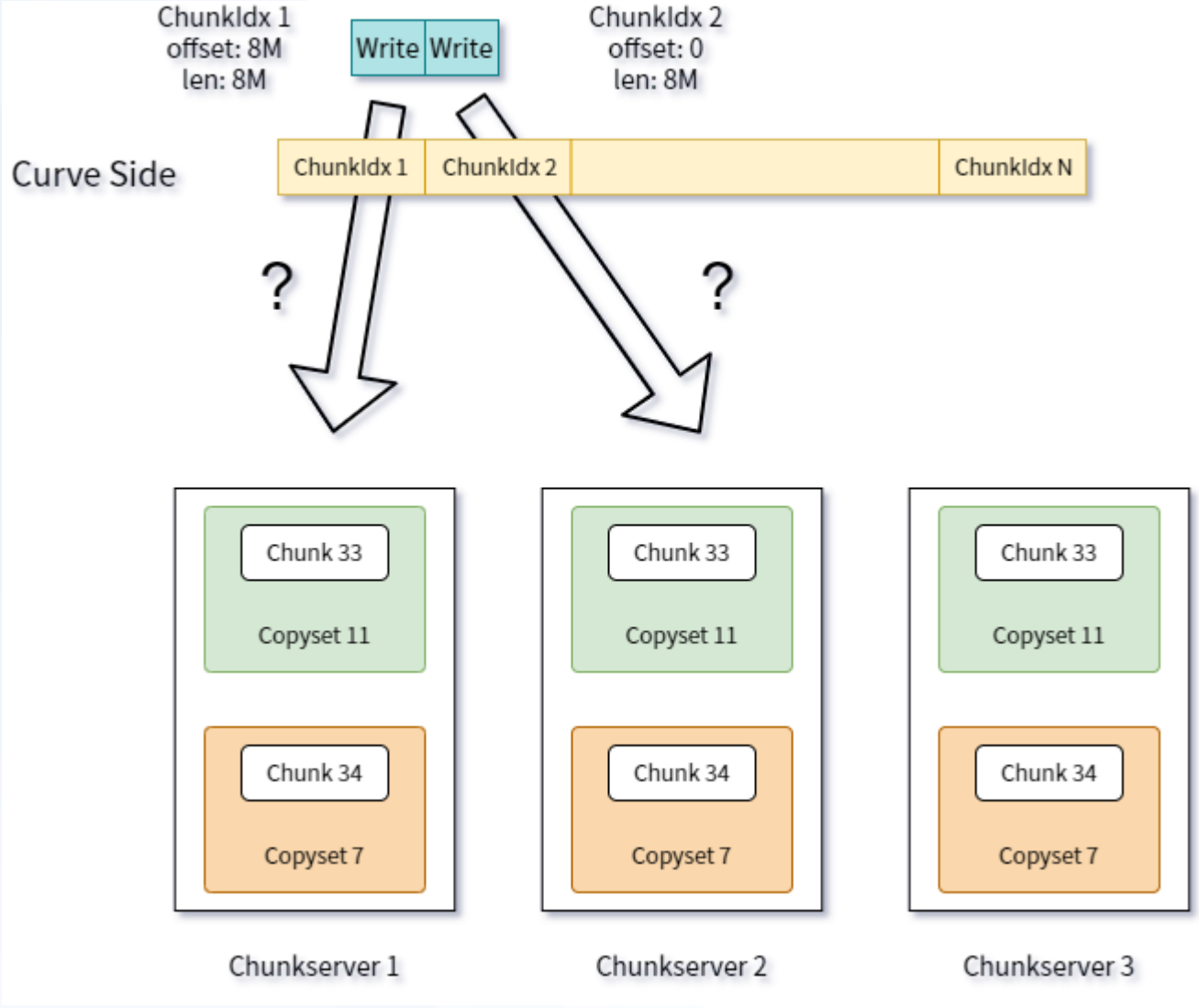
- ChunkIdx 1, off: 8M  len 8M
- ChunkIdx 2, off: 0  len 8M



# CLIENT IO流程

子请求由哪个chunkserver处理，依赖以下信息：

- 逻辑chunk与物理chunk映射关系
- 物理chunk所属的复制组(copyset)
- 复制组所在的chunkserver列表
- 复制组的leader信息

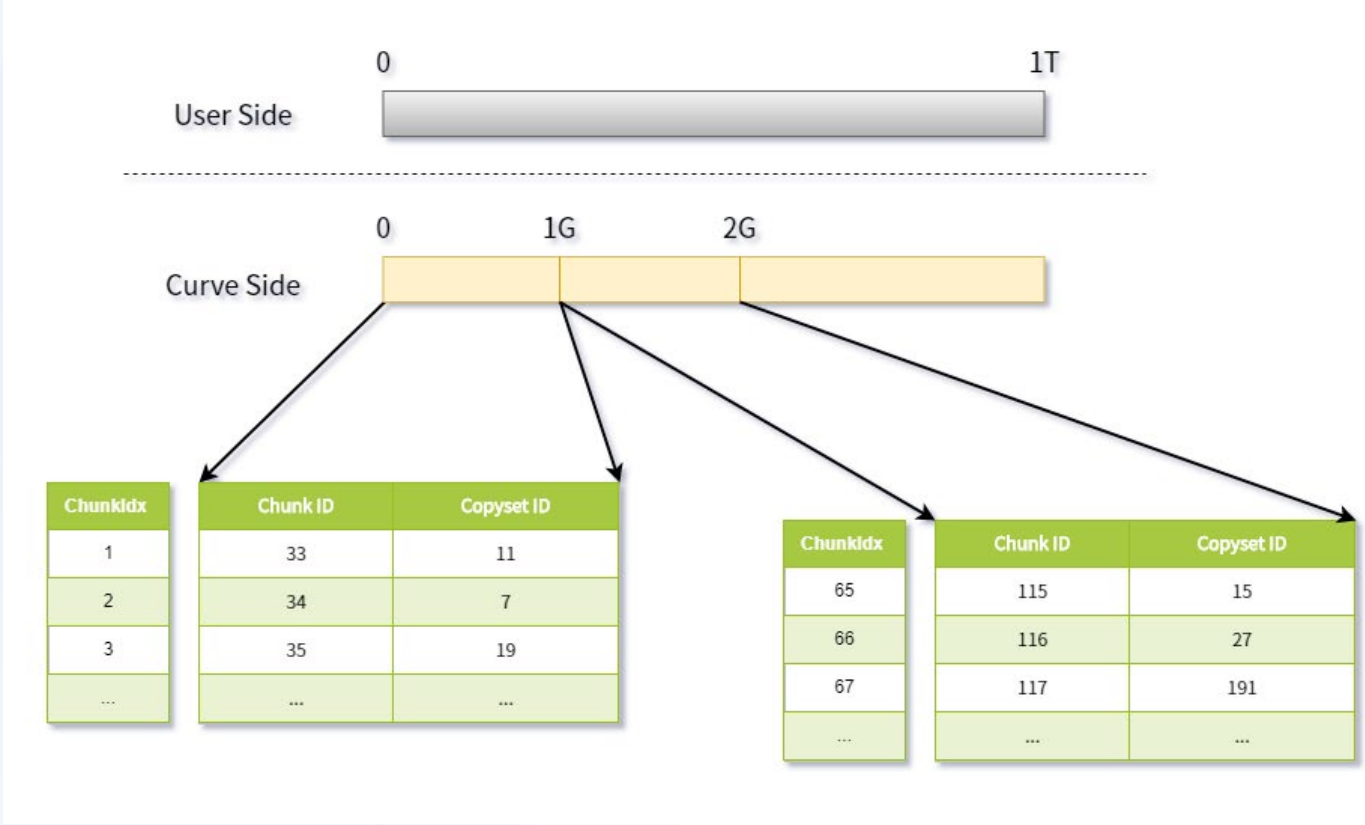


# CLIENT IO流程

逻辑chunk与物理chunk映射关系

物理chunk所属的复制组(copyset)

- 由MDS分配并持久化，client拆分用户请求时会获取并进行缓存
- 为了减少元数据量，MDS一次会连续分配1G范围内的映射关系，称为Segment



# CLIENT IO流程

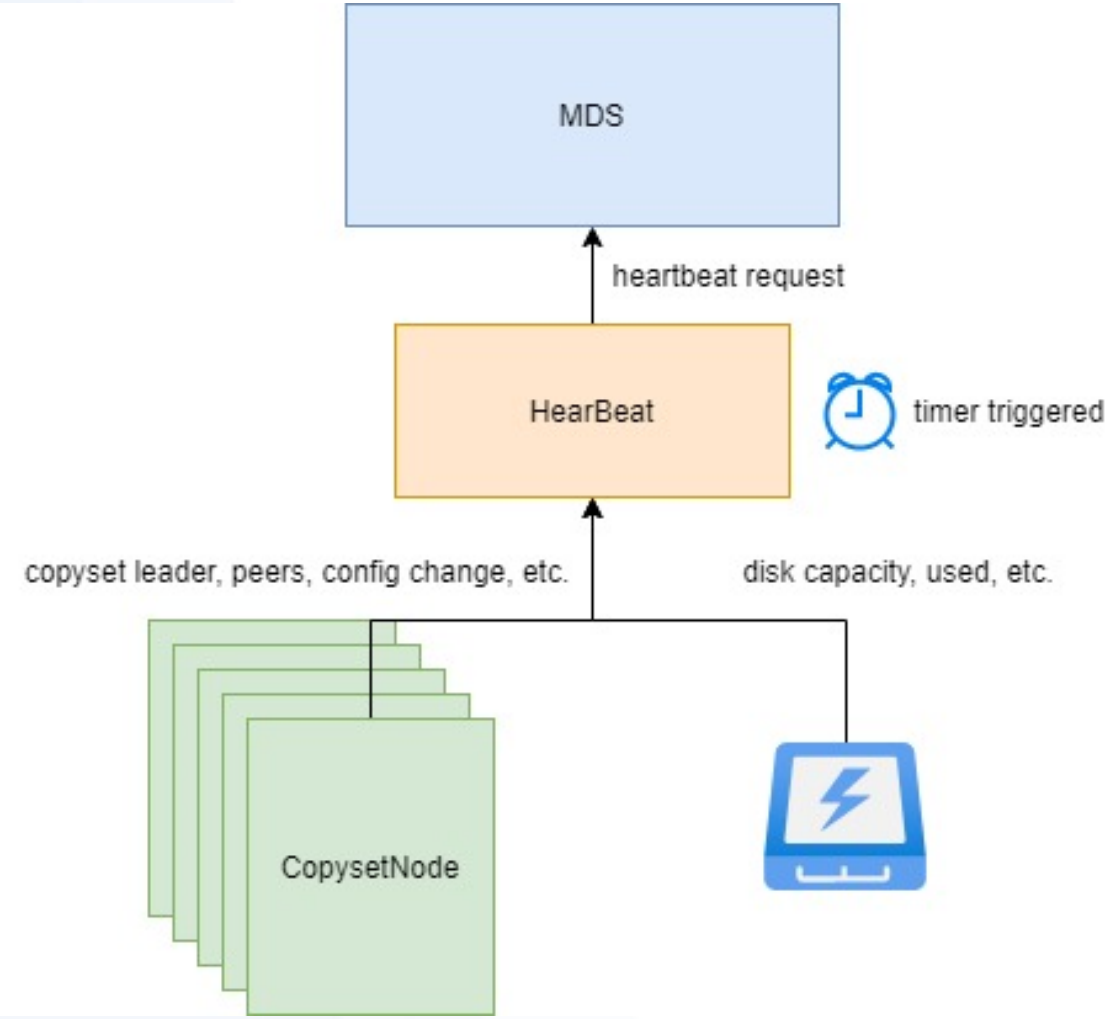
复制组所在的chunkserver列表

- chunkserver心跳定期上报给MDS
- 通过MDSClient向MDS获取

复制组的leader信息

- 复制组之间通过raft维护
- 通过CliClient向Chunkserver获取

这两种信息client也会进行缓存

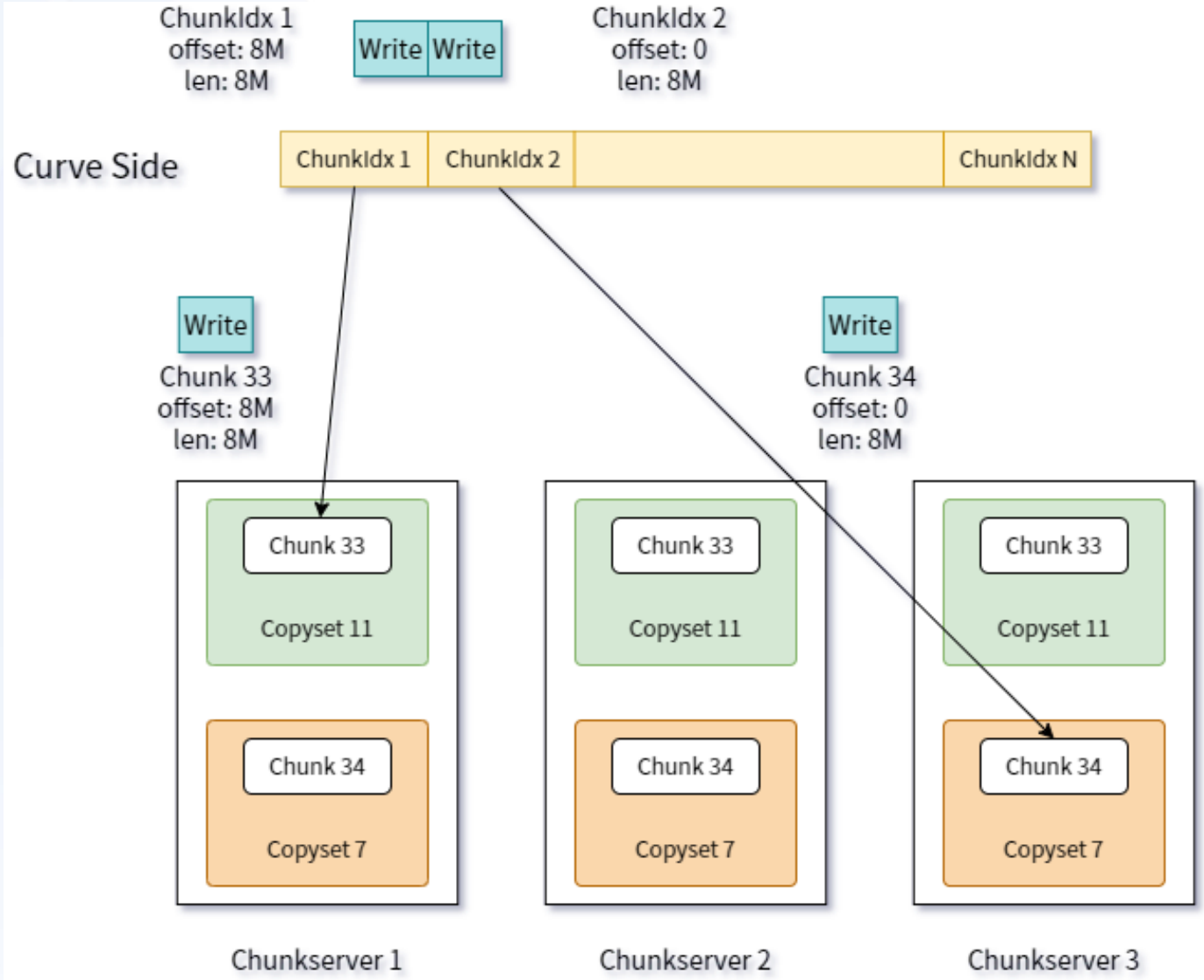



上报心跳

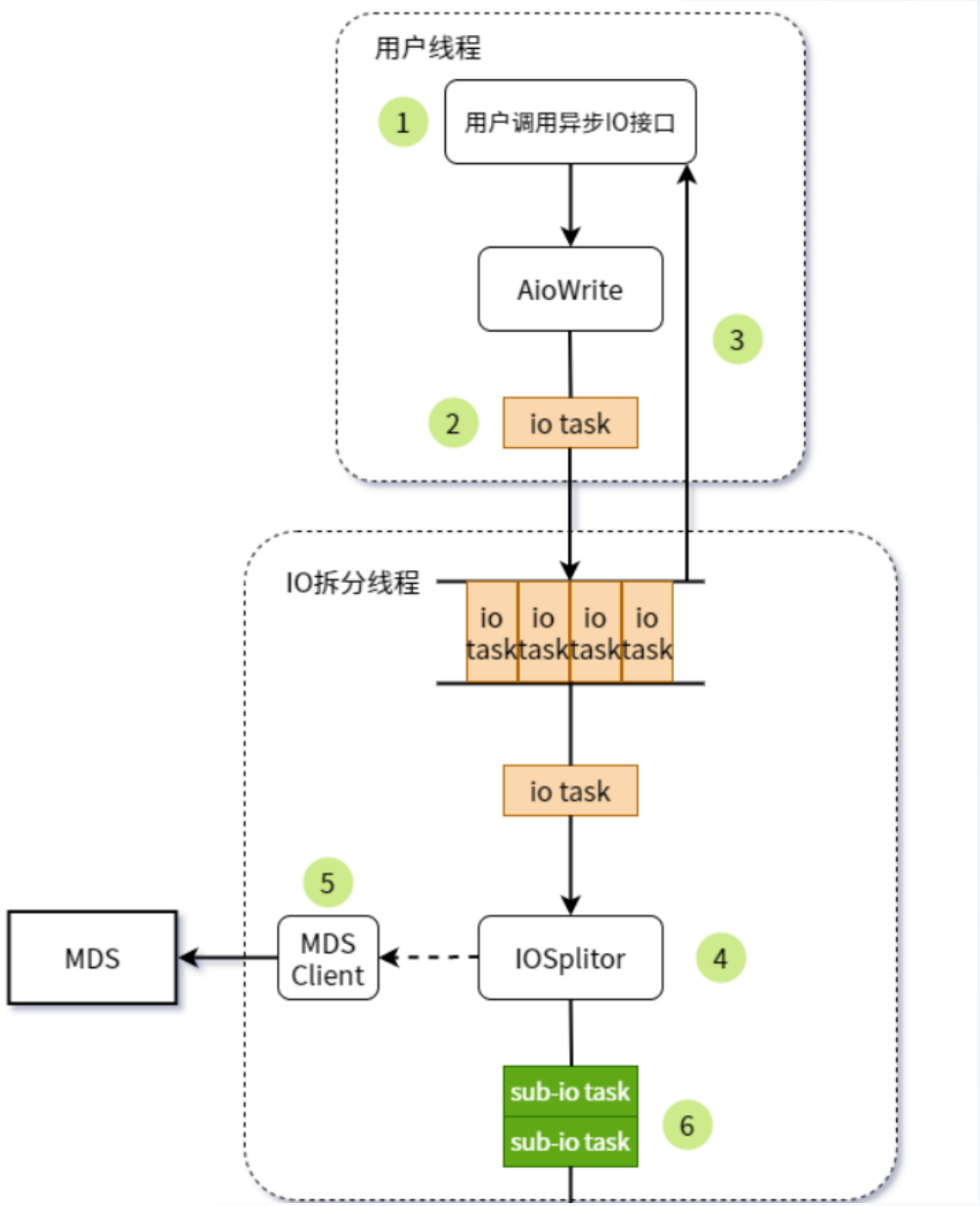
# CLIENT IO流程

子请求处理步骤：

1. 从MDS获取逻辑chunk与物理chunk的对应关系（包含逻辑池以及复制组信息）
2. 从MDS获取复制组所在的机器列表
3. 从Chunkserver获取复制组leader信息
4. 将请求发往leader节点



# CLIENT IO线程模型



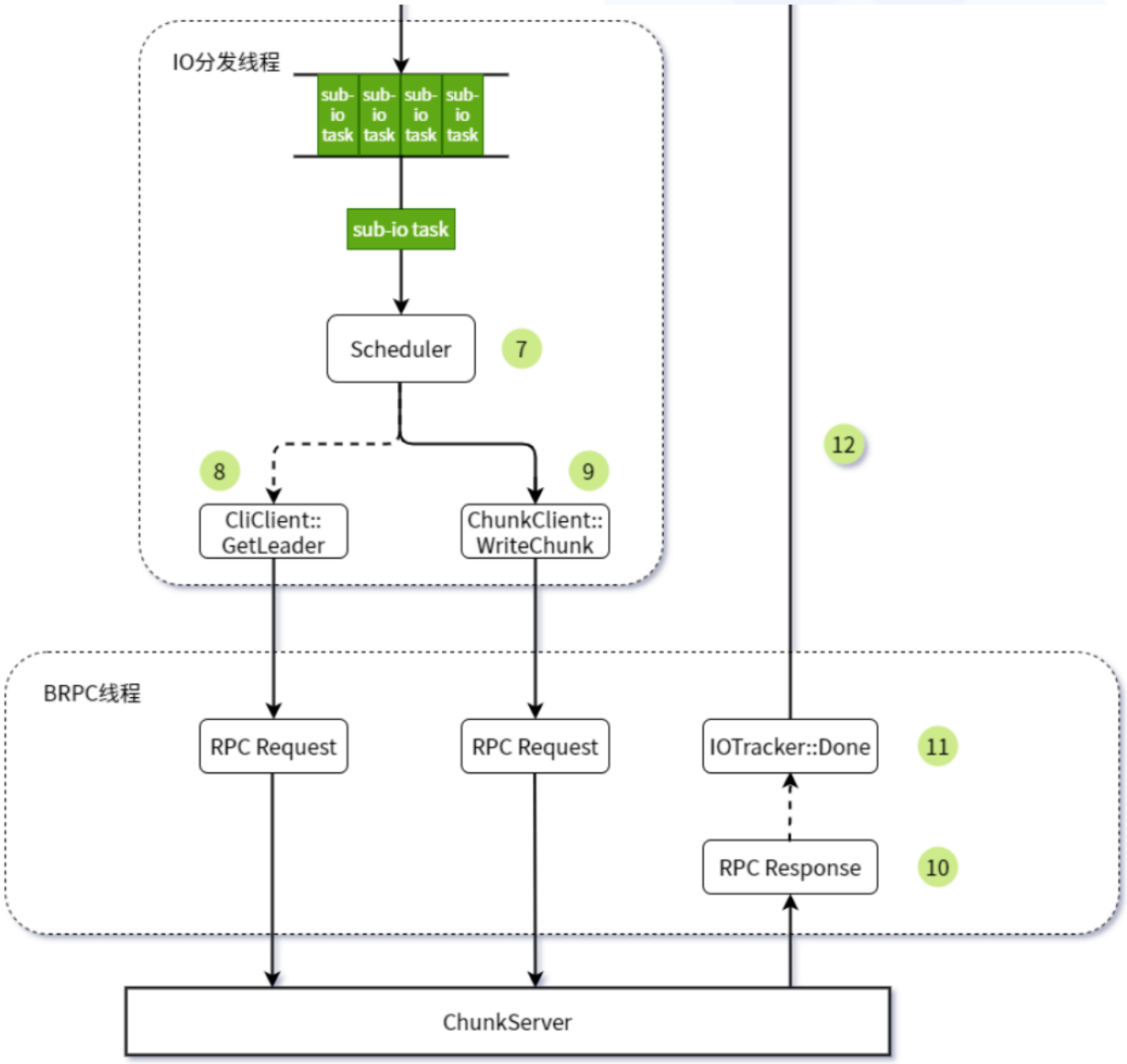

用户线程

1. 用户调用接口，发起IO请求
2. AioWrite将请求封装成io task并放入任务队列
3. 放入任务队列后，异步请求发起成功，返回用户

IO拆分线程

4. 从任务队列取出任务后进行拆分
5. 拆分过程依赖元数据，**可能**会通过MDSClient向MDS获取
6. 拆分成的子请求放入队列

# CLIENT IO线程模型




IO分发线程

7. 从队列中取出子请求准备发送
8. 发送依赖复制组leader，**可能**会向Chunkserver查询复制组leader
9. 发送写请求给Chunkserver

BRPC线程

10. Chunkserver处理完成后返回RPC Response
11. 用户请求的所有子请求完成后，调用IOTracker::Done
12. 调用异步请求回调，返回用户

# CLIENT IO请求重试

IO分发线程将拆分后的子请求通过RPC请求发往指定的Chunkserver上，RPC有可能会失败，一般情况下处理逻辑是sleep一个较短时间后重试，但是存在两种特殊的场景：

**Chunkserver Overload:**

这种情况下，对应的RPC Response中返回的错误码是OVERLOAD，说明底层Chunkserver正在处理的请求数量过多。按照一般重试逻辑，大概率情况下重试请求还是返回OVERLOAD，造成用户IO请求一直无法返回。

加入睡眠时间指数退避，并加入一个随机值，避免sleep后大量重试又碰撞到一起。

**RPC超时：**

请求在chunkserver端处理请求处理时间长，导致请求的返回时间超过了预期的RPC超时时间。

这种情况下，如果重试请求的RPC超时时间不发生变化，也有可能会重复上述流程，导致用户IO请求迟迟未能返回。所以，在这种情况下，重试请求会将RPC超时时间进行增加。

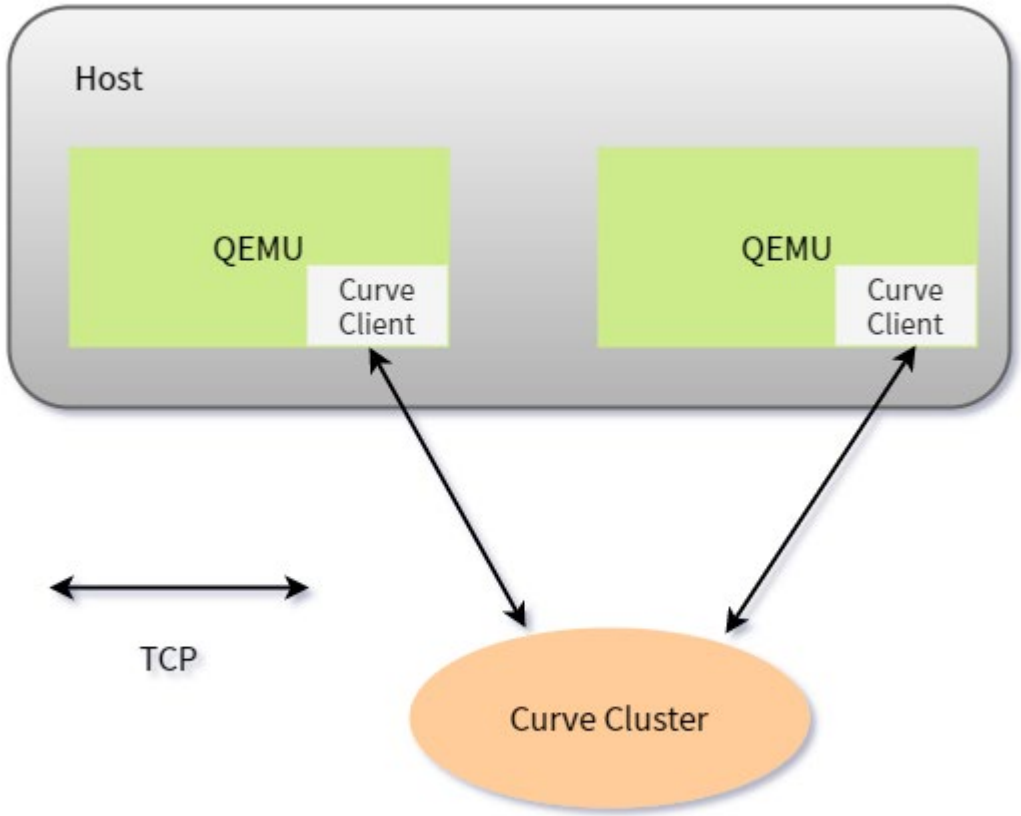
Curve

DIGITAL SAIL

# 目录

# NEBD 整体介绍

热升级之前，QEMU是直接链接curve-client，所以client版本需要升级时，需要对QEMU进程进行重启。



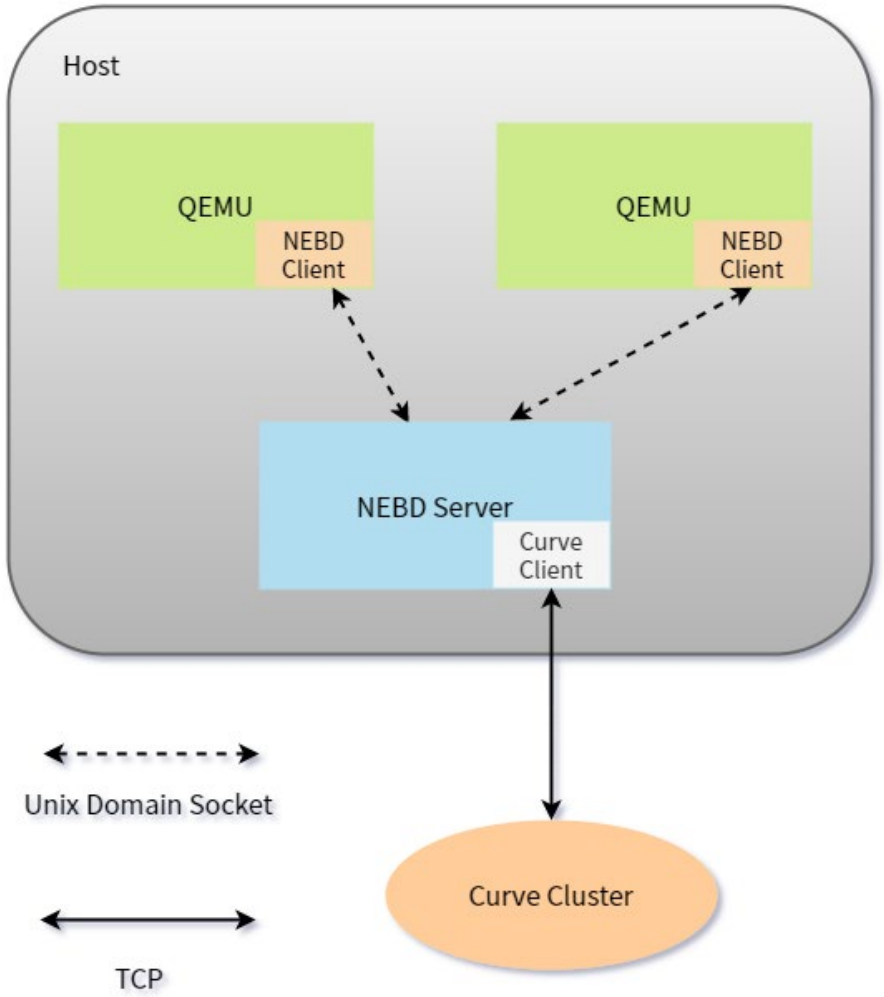
# NEBD 整体介绍

在QEMU和Curve Client中间加入热升级模块，避免直接依赖

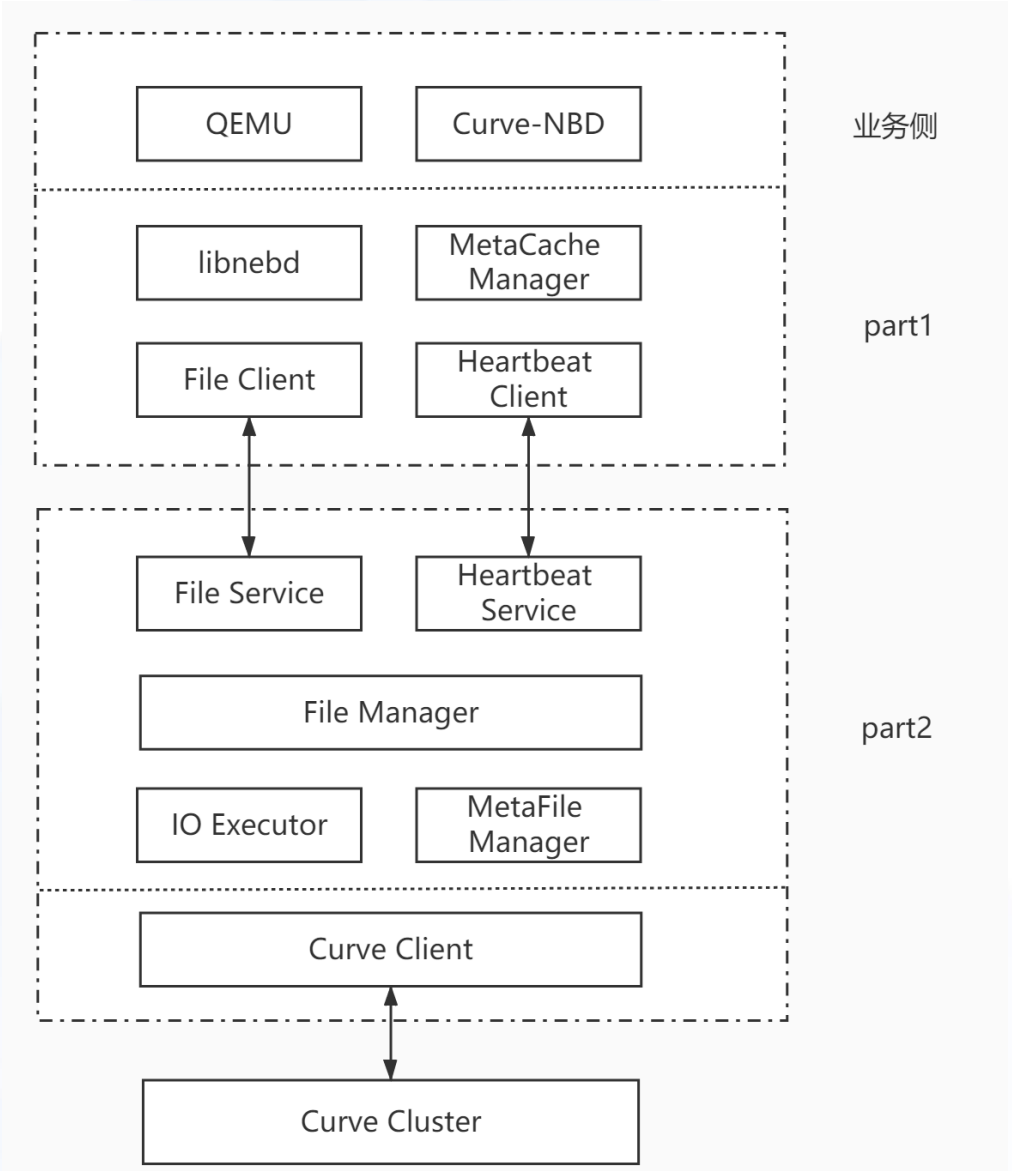
热升级模块是CS结构：

- NEBD Client(part1)：只包含轻量的业务逻辑，以链接库的形式提供给QEMU使用
- NEBD Server(part2)：将NEBD Client的请求转发到Curve Client

升级过程只需要重启NEBD Server即可，IO可在1~5s内恢复



# NEBD 整体介绍


QEMU
Curve-NBD
业务侧
libnebd
MetaCache Manager
part1
File Client
Heartbeat Client
File Service
Heartbeat Service
File Manager
part2
IO Executor
MetaFile Manager
Curve Client
Curve Cluster

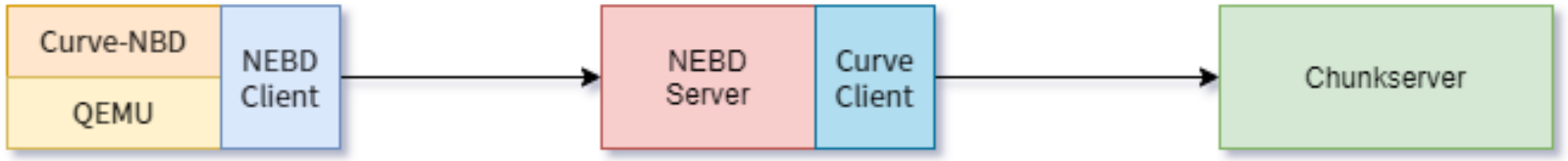
Curve

DIGITAL SAIL

# 目录

# NEBD性能优化

场景：fio 128深度、4K随机写




160K iops
872.55us

NEBD Client发送RPC处标记请求成功，并返回

86.4K iops
1404.15us

NEBD Server收到RPC处标记请求成功，并返回

80.7K iops
1584.72us

Curve Client发送RPC处标记请求成功，并返回

66.1K iops
1935.02us

Chunkserver收到RPC处标记请求成功，并返回

# NEBD性能优化

NEBD Client接收到IO请求后，直接发送异步RPC（在用户线程）

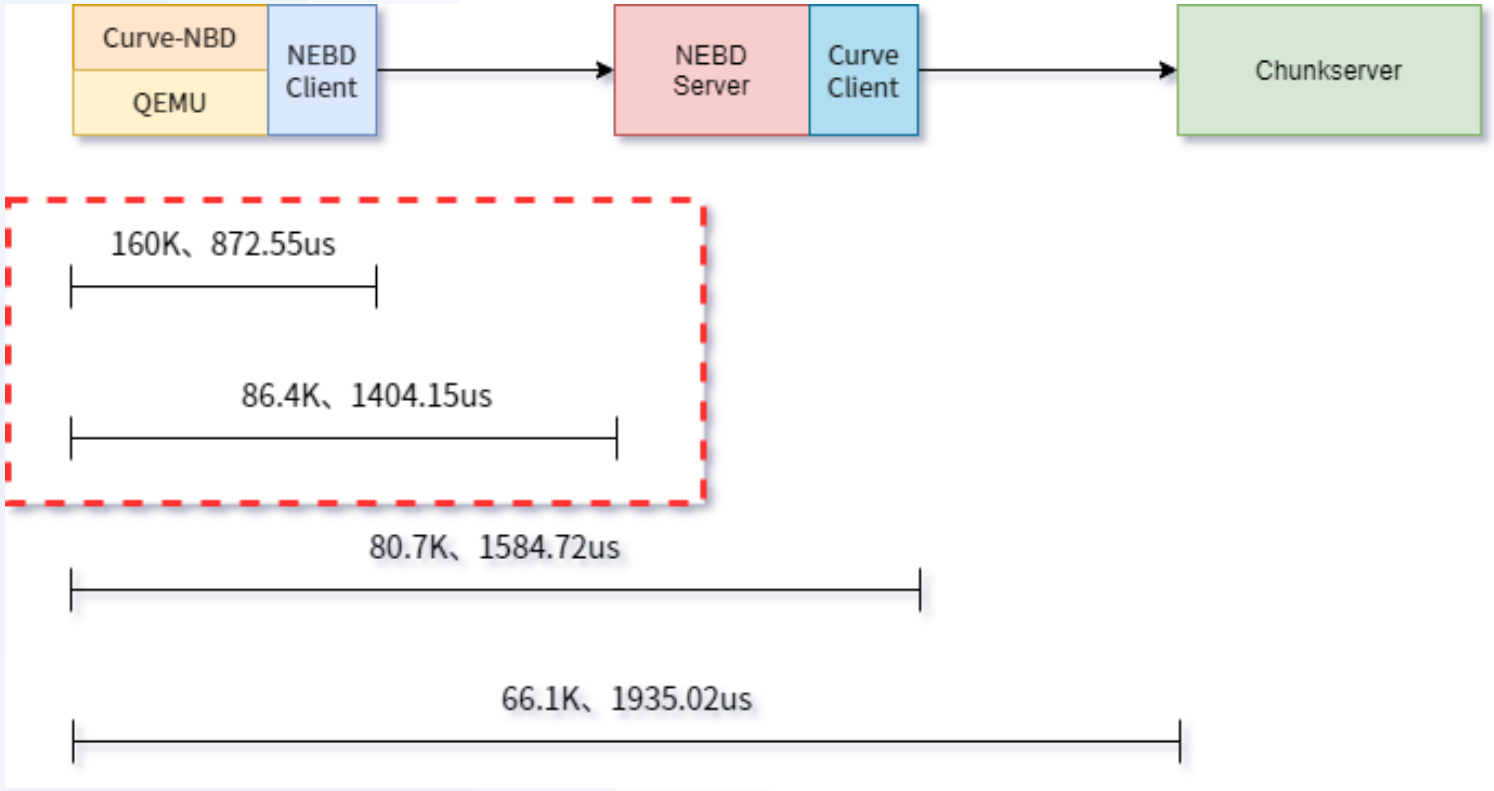
发送异步4K RPC的平均延迟在11.26us，这种情况下单线程只有 **89055** iops

发送RPC阻塞了用户线程，导致iops下降

优化点：

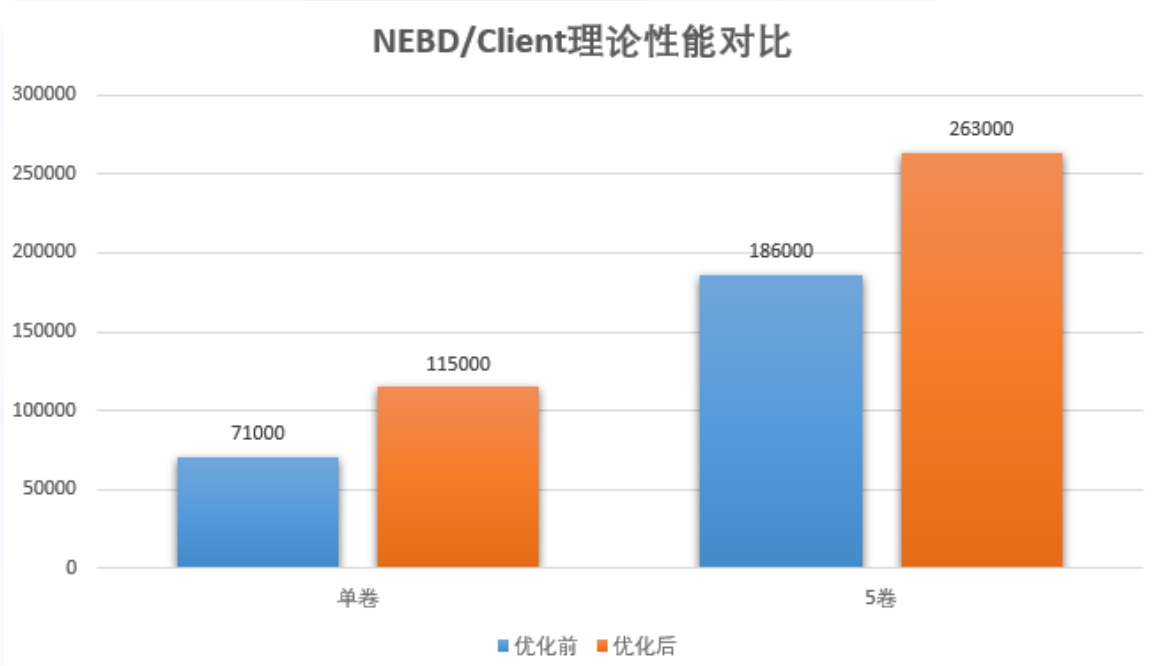
增加队列，用户请求放入队列，由后台线程负责发送

86.4K -> 130K



# CLIENT性能优化

- 发送RPC耗时较长
  - 增加发送线程个数
- 在bthread协程中使用std::mutex/spinlock，在大量并发的情况下，会阻塞worker线程，也存在瓶颈
  - std::mutex/spinlock 改成 bthread::Mutex
- ……




128深度、4K随机写

# 欢迎大家参与CURVE项目！


- github主页： https://opencurve.github.io/
- github代码仓库： https://github.com/opencurve/curve
- 系列讲座：https://space.bilibili.com/700847536/channel/detail?cid=153949

# THANK YOU

D I G I T A L S A I L


扫码即可关注