Curve

Curve核心组件之ChunkServer

D I G I T A L S A I L

查日苏

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500+天
- 已开源
  - github主页： https://opencurve.github.io/
  - github代码仓库： https://github.com/opencurve/curve

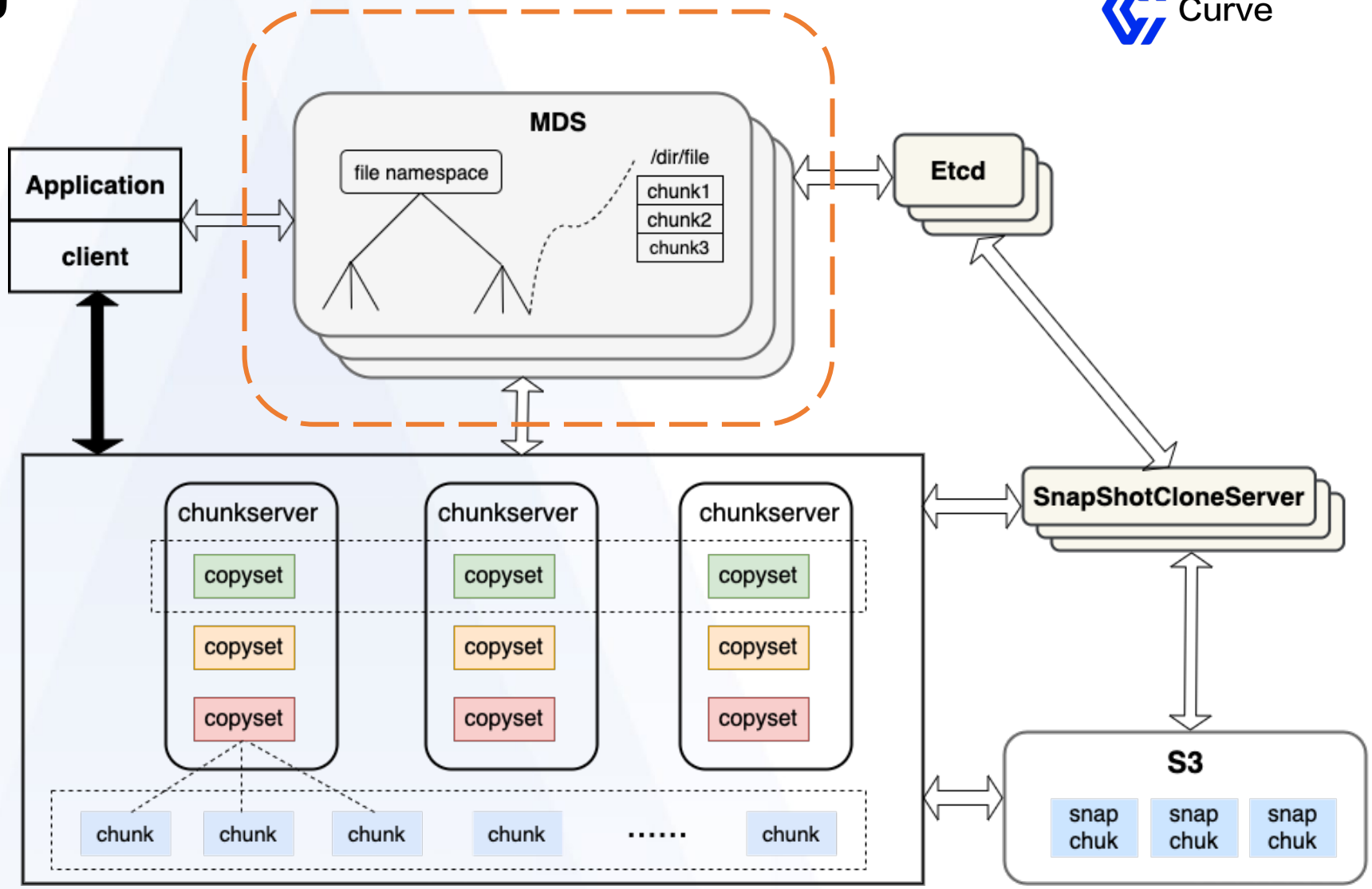
Curve

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



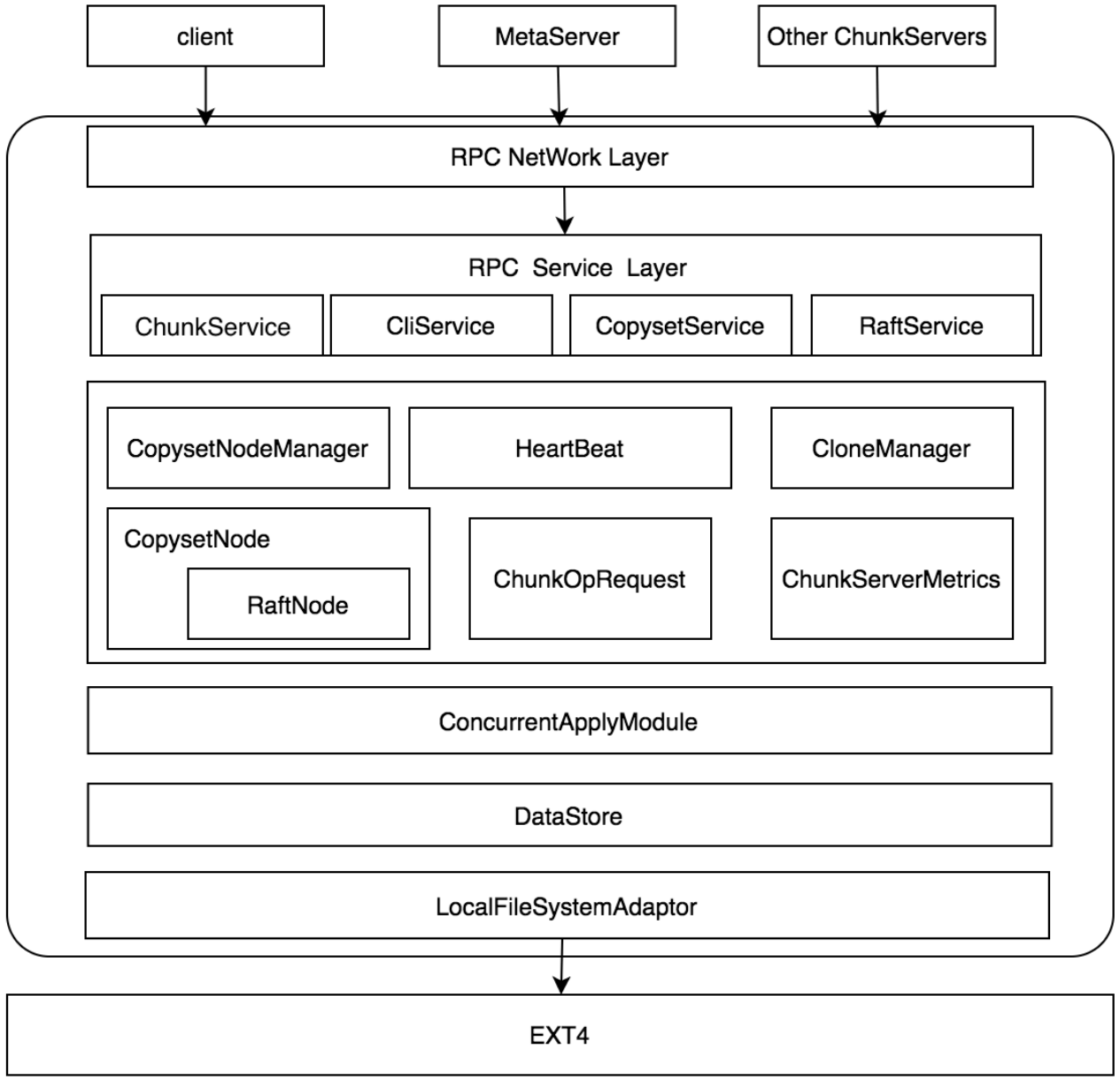
Curve

DIGITAL SAIL

目录

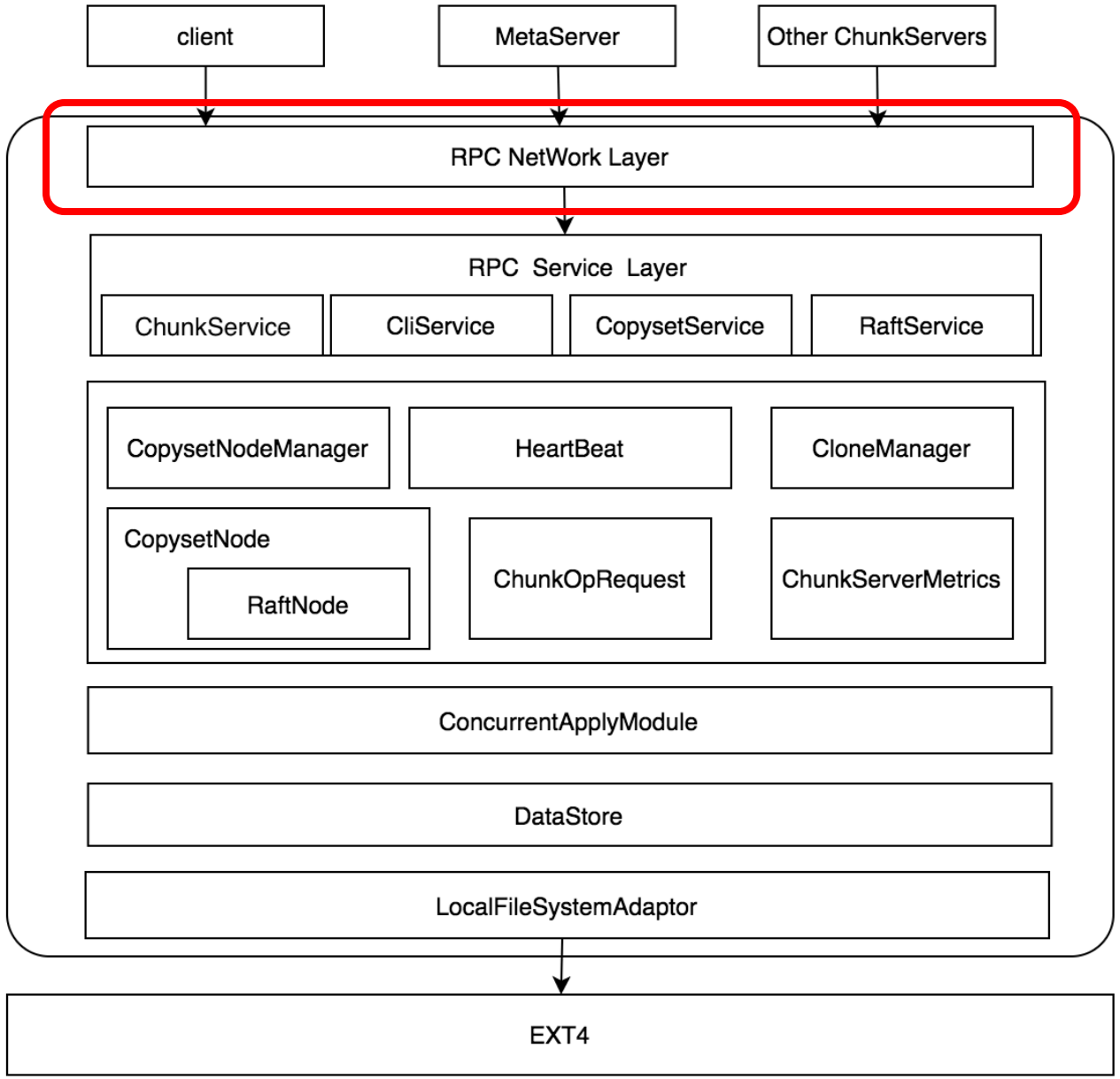
ChunkServer架构

Curve ChunkServer是数据节点，对外提供数据读写和节点管理功能，底层基于ext4文件系统，操作实际的磁盘。



ChunkServer架构

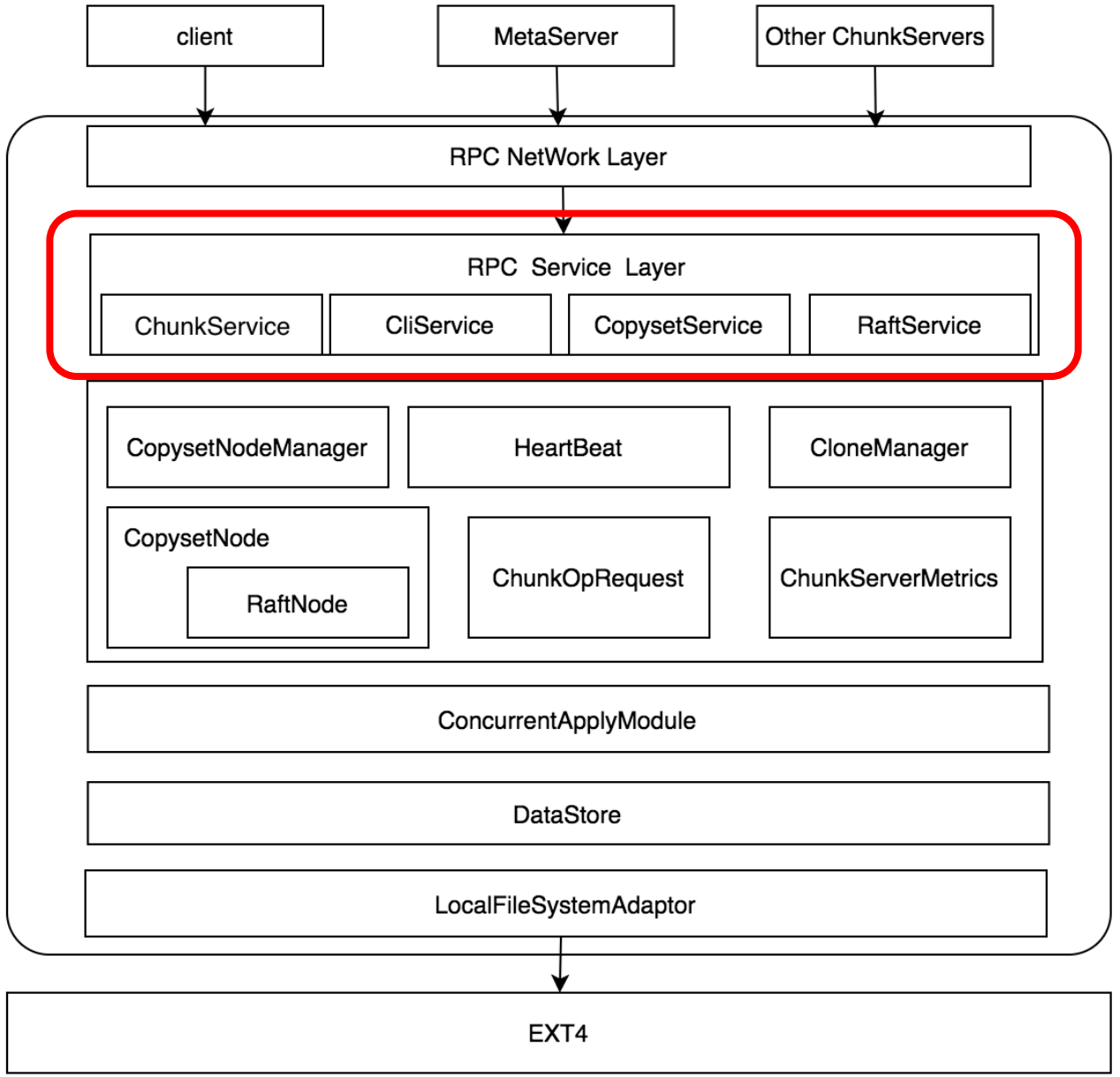
ChunkServer通过RPC网络层与client，MDS，其他ChunkServer通信。RPC网络层是由brpc框架去完成的。包括读写socket，rpc协议解析等。



ChunkServer架构

RPC Service层是对外提供的一些RPC服务的接口。包含的RPC服务有:

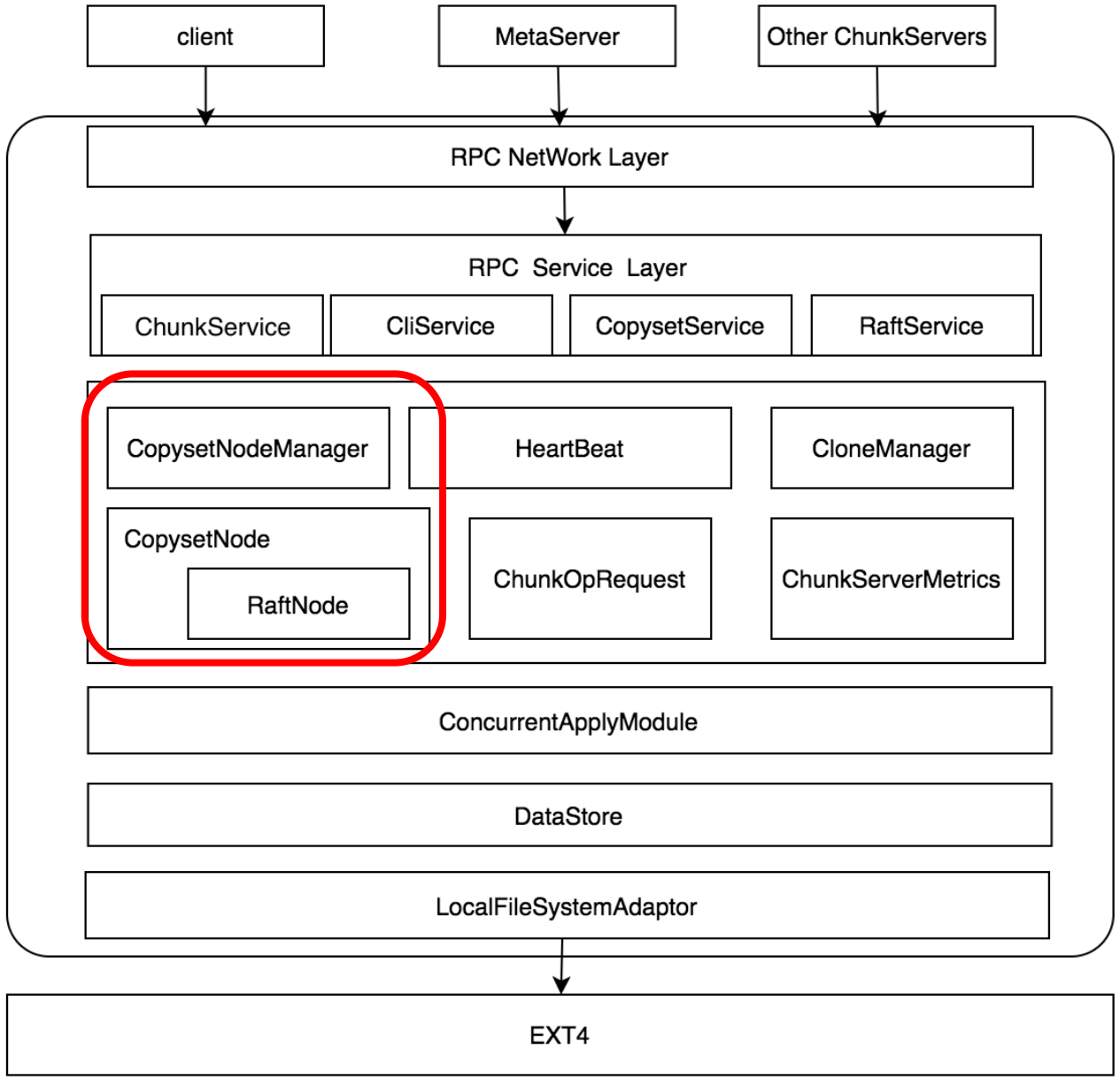
- ChunkService。IO相关操作
- CliService。成员变更相关操作
- CopySetService。创建copyset等操作
- RaftService。Braft内置的service，完成raft成员之间的选举，日志复制，安装快照等操作。



ChunkServer架构

CopysetNode封装了braft的Node，并实现了braft的状态机，完成与raft的交互。详细交互流程后面展开。

CopysetNodeManager负责管理CopysetNode的创建、初始化、删除等

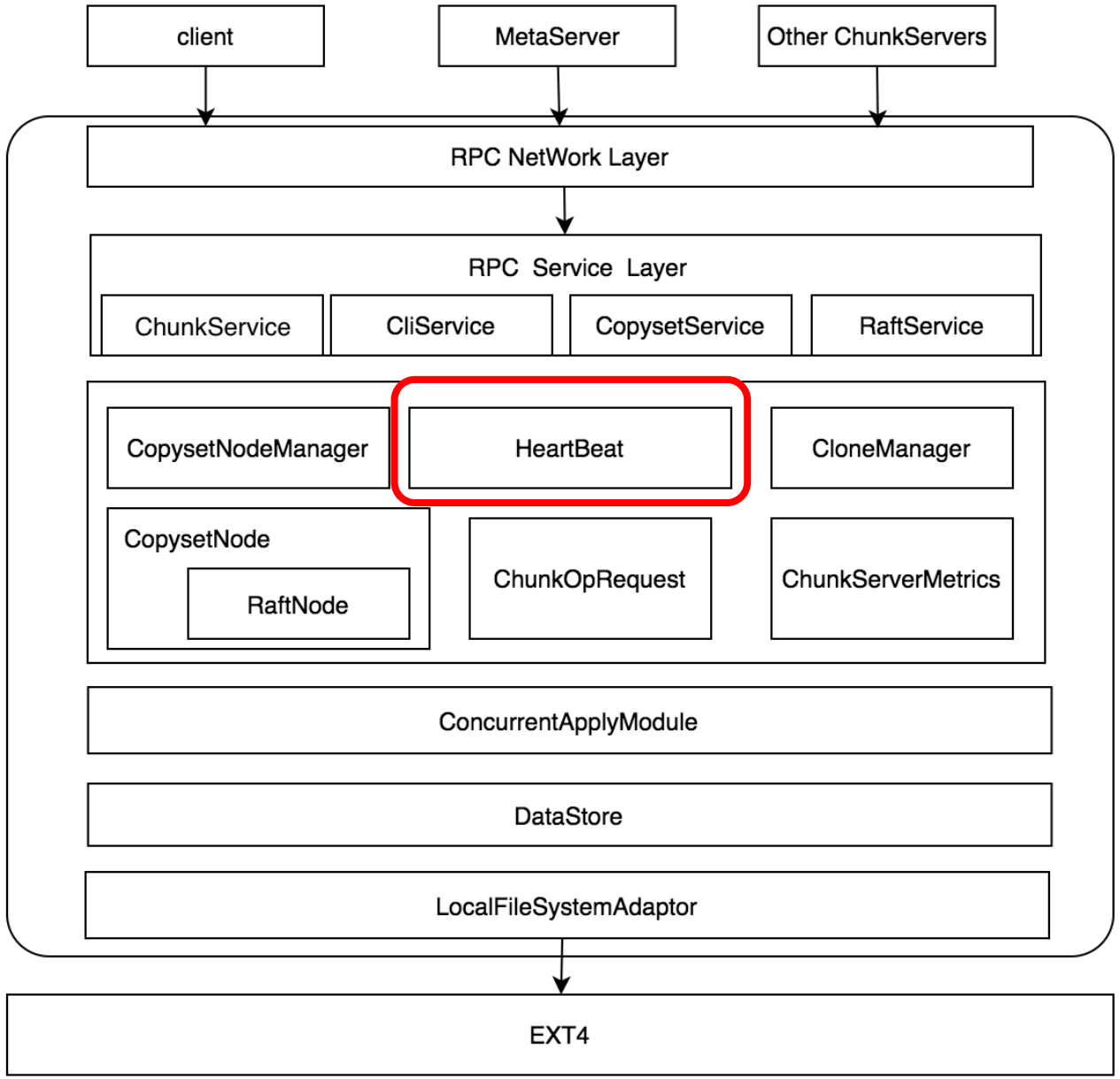


ChunkServer架构

心跳模块有两方面的职责：

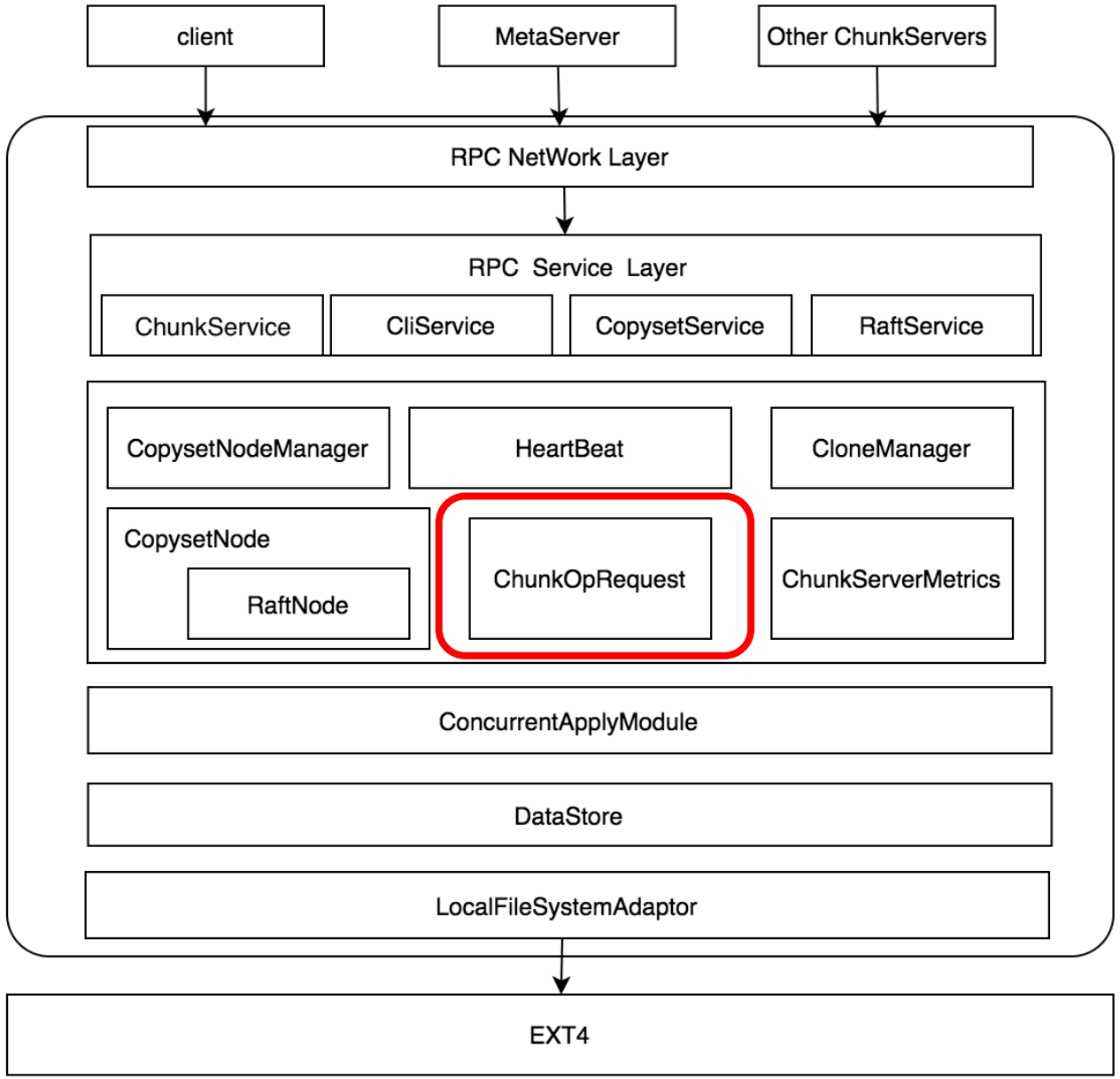
1、向MDS节点上报心跳，心跳中包括ChunkServer本身的一些统计信息

2、解析MDS的心跳response中的raft成员变更信息，向CopysetNode发起变更



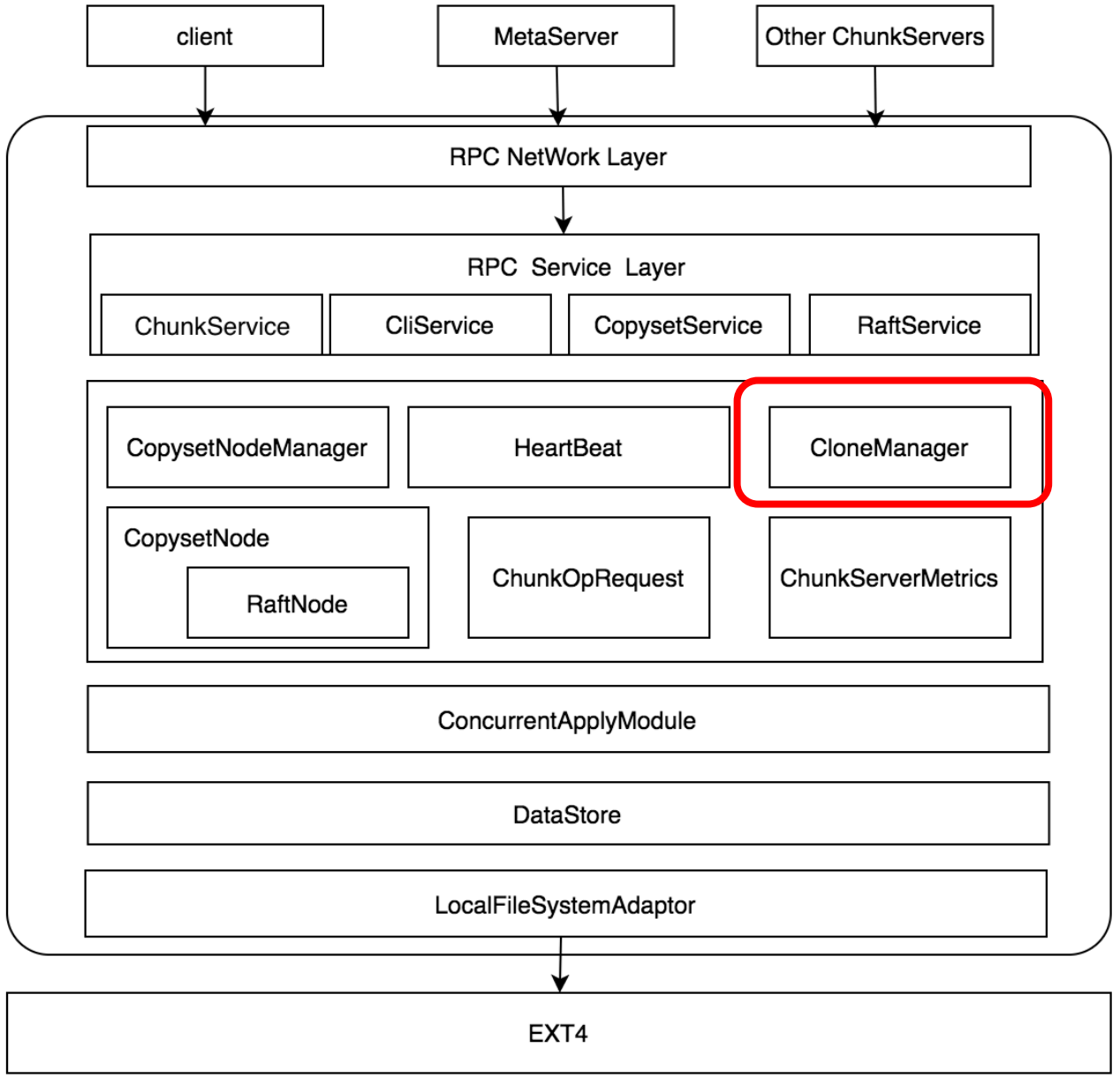
ChunkServer架构

ChunkOpRequest模块封装了对ChunkService到达的I/O请求的实际处理过程。请求到来时，封装一个OpRequest，将上下文保存在里面，然后发起Propose提交给raft，等raft apply后再执行后面的操作。



ChunkServer架构

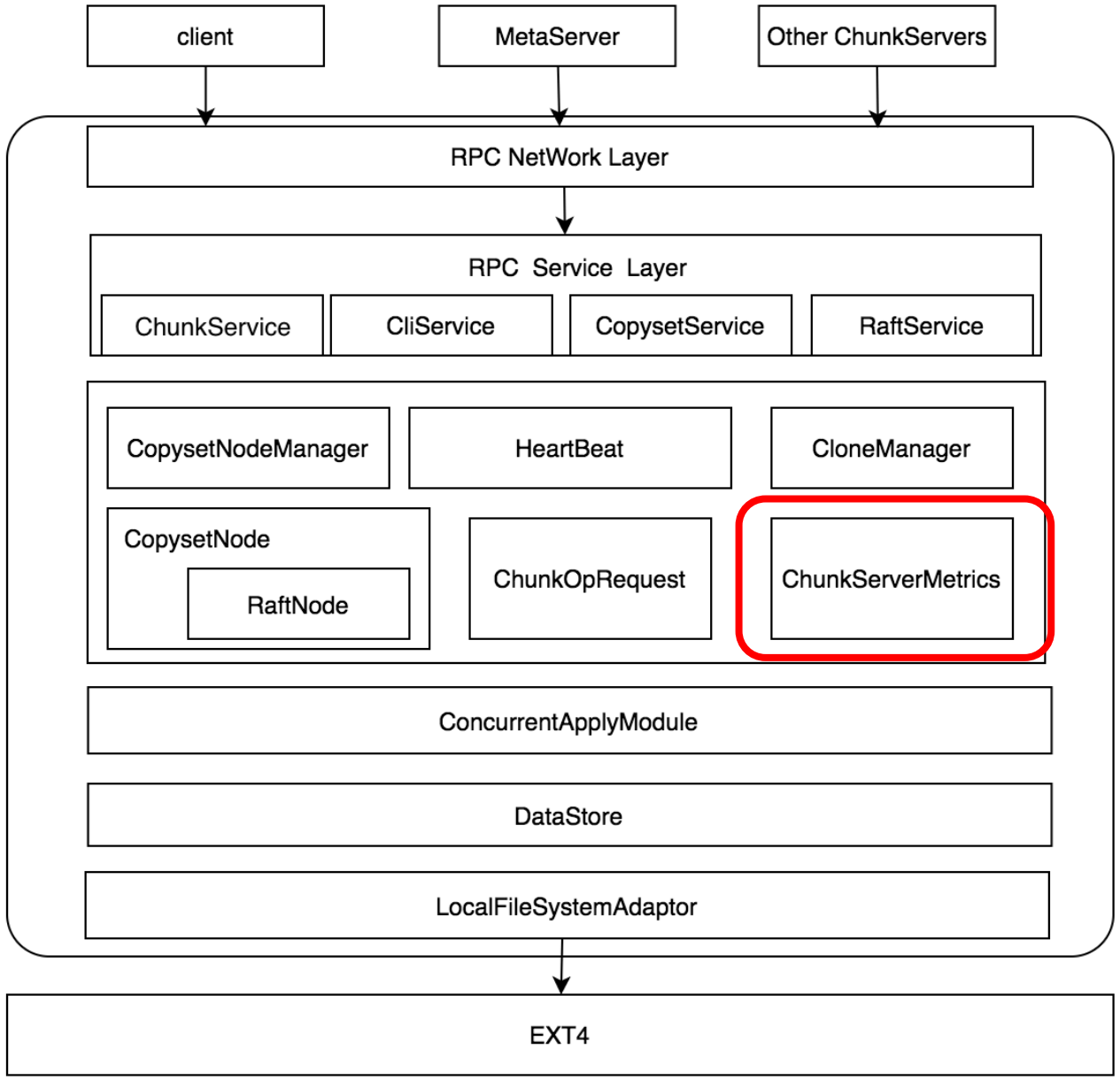
CloneManager主要负责克隆相关的功能，内部是一个线程池，主要负责异步完成克隆chunk的数据补全。关于克隆相关的内容将会在快照克隆相关介绍文档中详细介绍。



ChunkServer架构

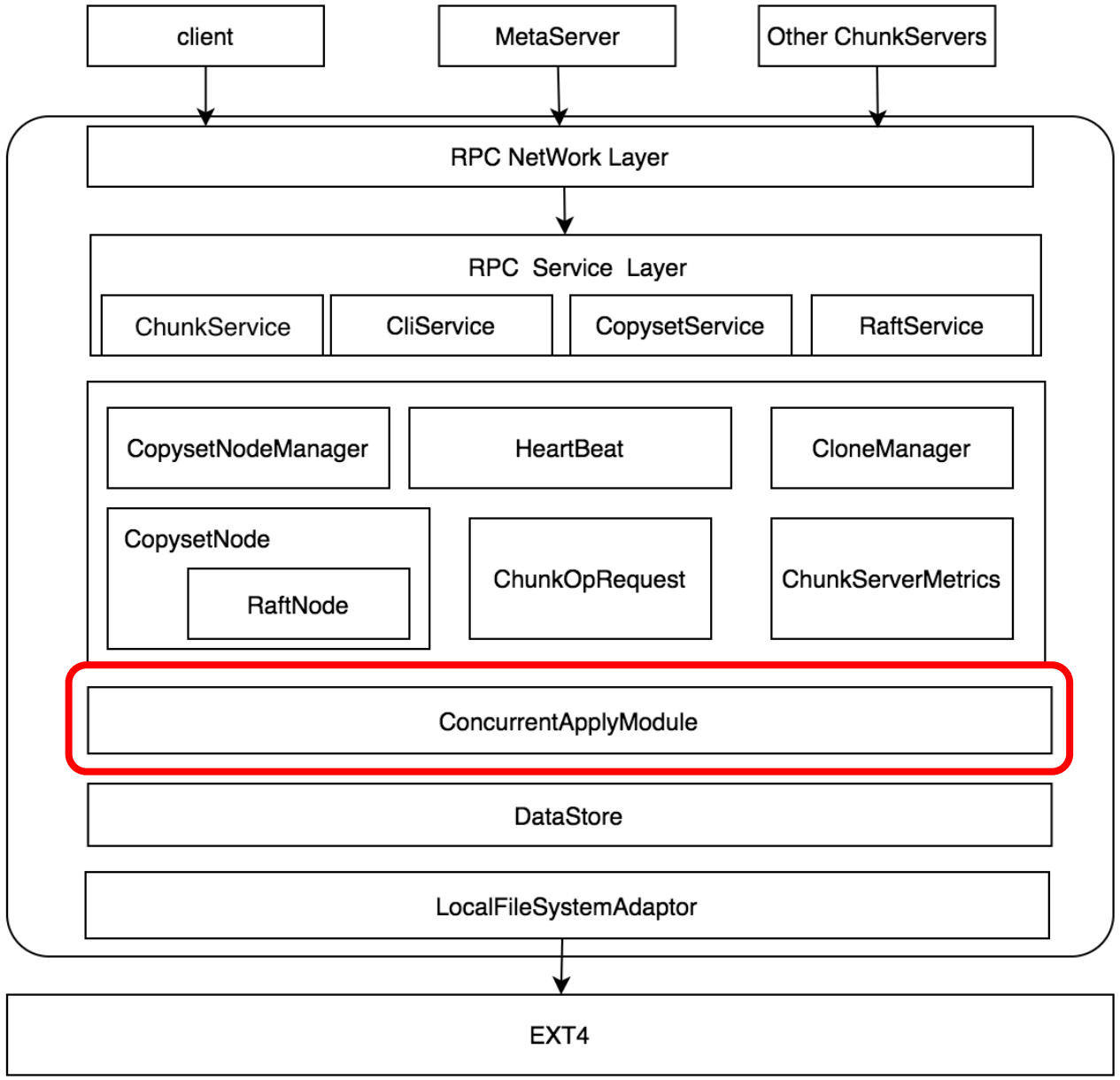
Curve

Metric统计模块使用brpc中的bvar计数器，统计一些IO层面和copyset层面的一些指标，方便监控和跟踪。



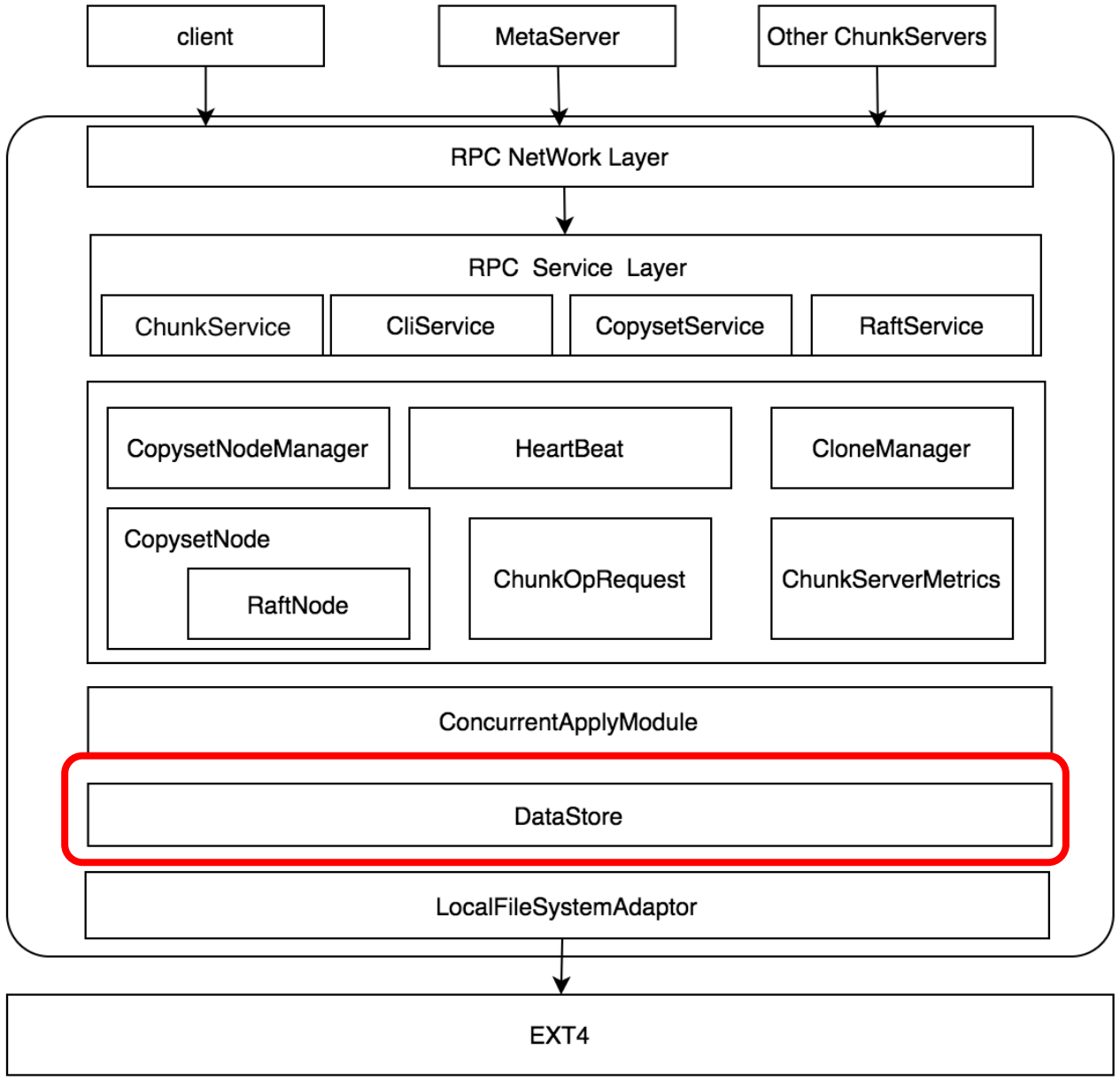
ChunkServer架构

并发控制层，负责对chunkserver的IO请求进行并发控制，对上层的读写请求安照chunk粒度进行Hash，使得不同chunk的请求可以并发执行。



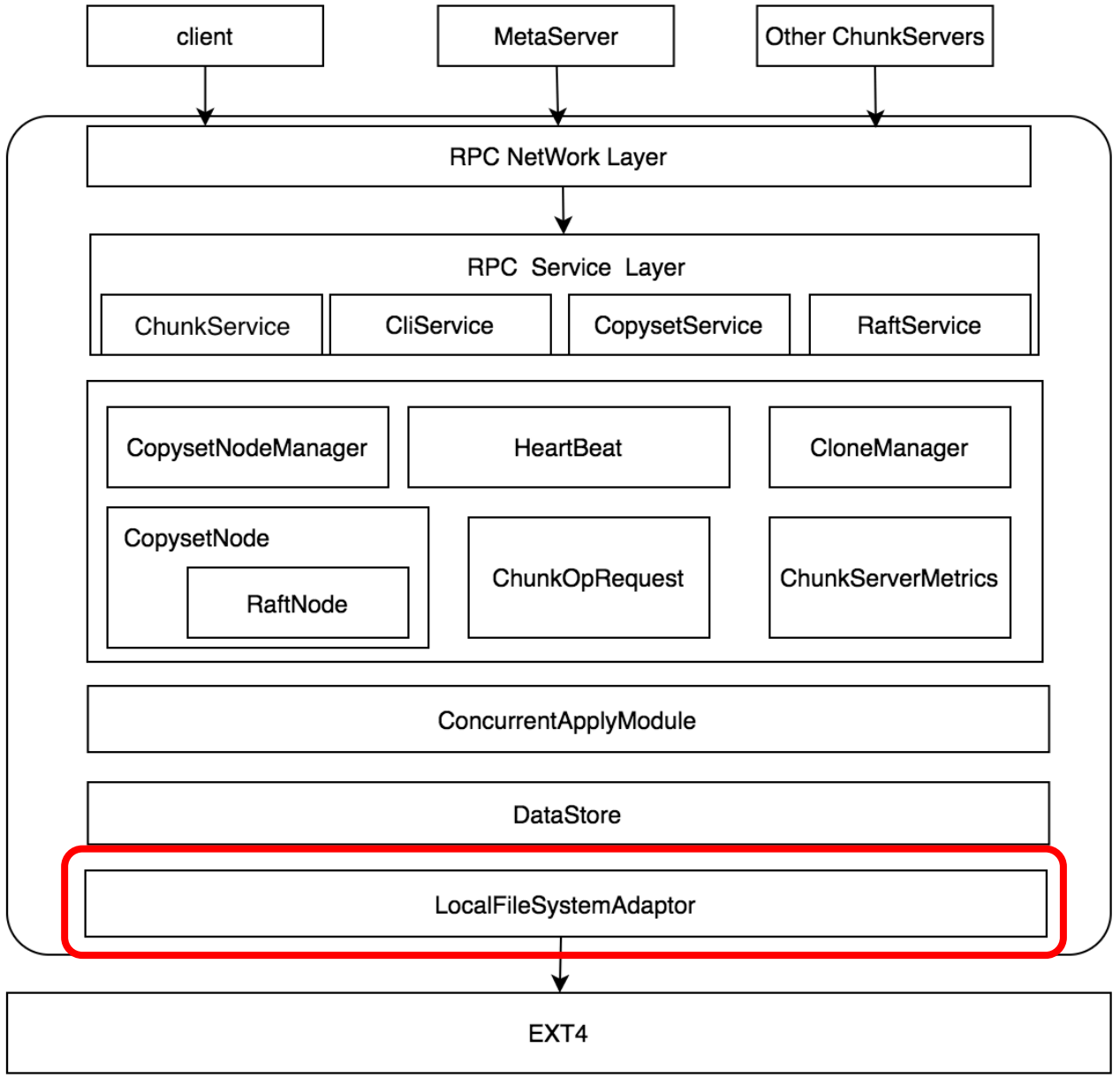
ChunkServer架构

DataStore是对chunk落盘逻辑的封装。包含chunkfile的创建、删除，以及实际对chunk的读写，chunk基本cow的快照，克隆chunk的管理等等。



ChunkServer架构

LocalFileSystermAdaptor是对底层文件系统的一层抽象，目前适配封装了ext4文件系统的接口。之所以要做这层抽象，目的是隔离了底层文件系统的实际读写请求，如果将来curve要适配裸盘或者采用其他文件系统，可以在这层进行适配。



Curve

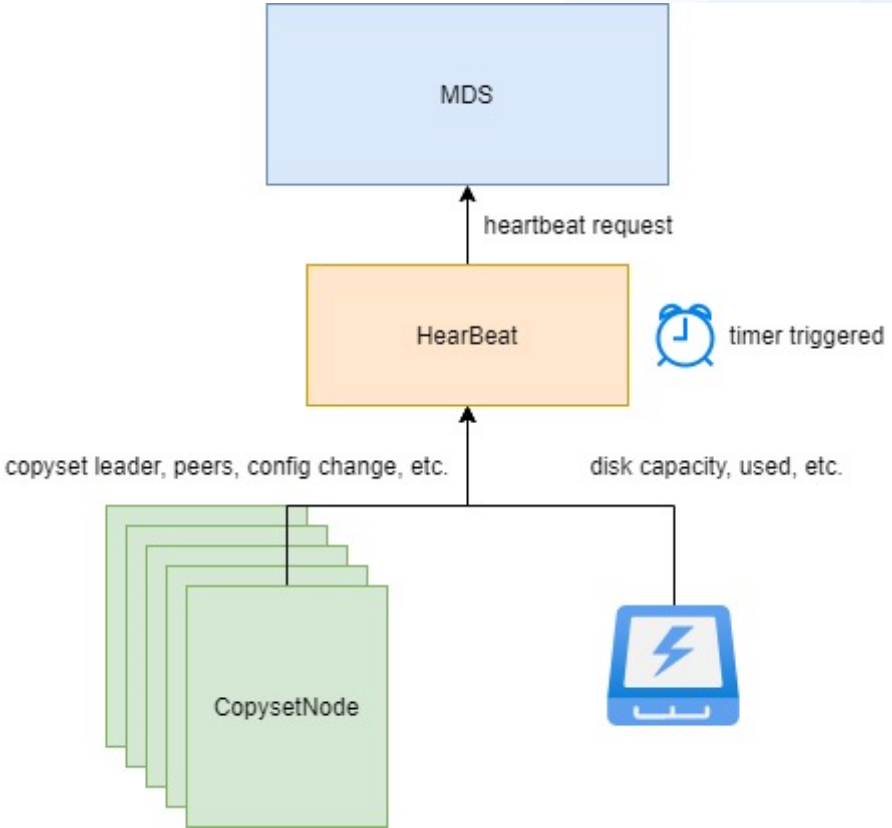
DIGITAL SAIL

目录

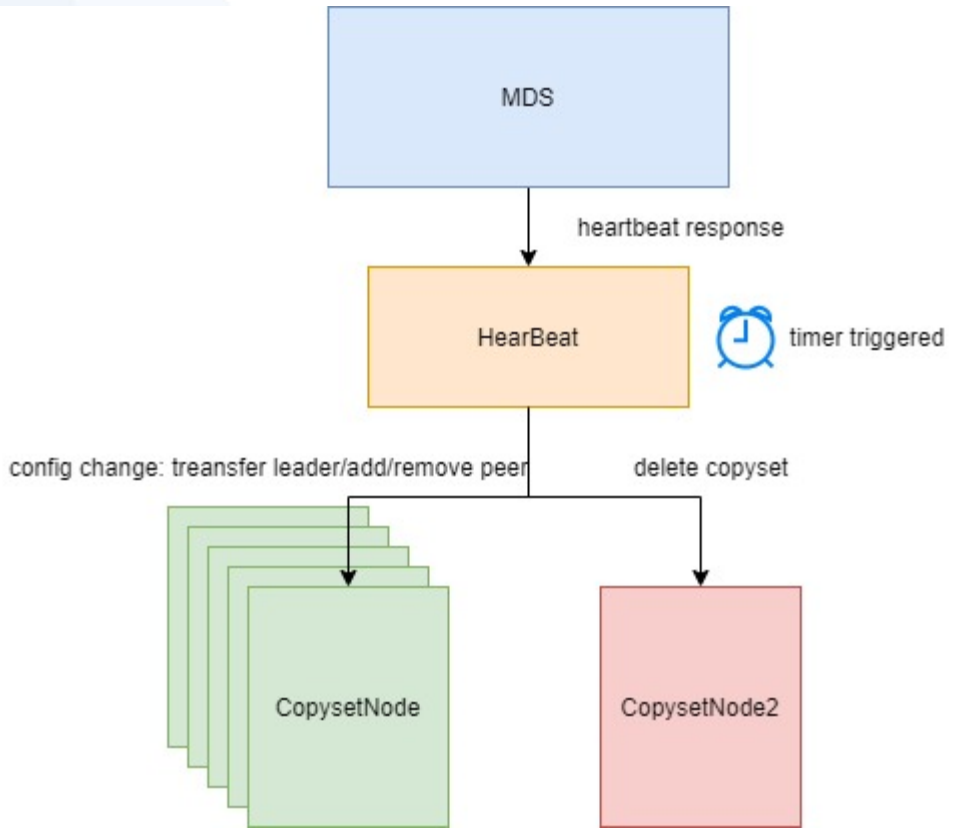
ChunkServer核心模块-注册和心跳

ChunkServer注册：chunkserver第一次启动时，需要向mds注册，mds分配并返回ChunkserverID，token，chunkserver持久化这些信息，并在后续心跳上报时携带这些信息。

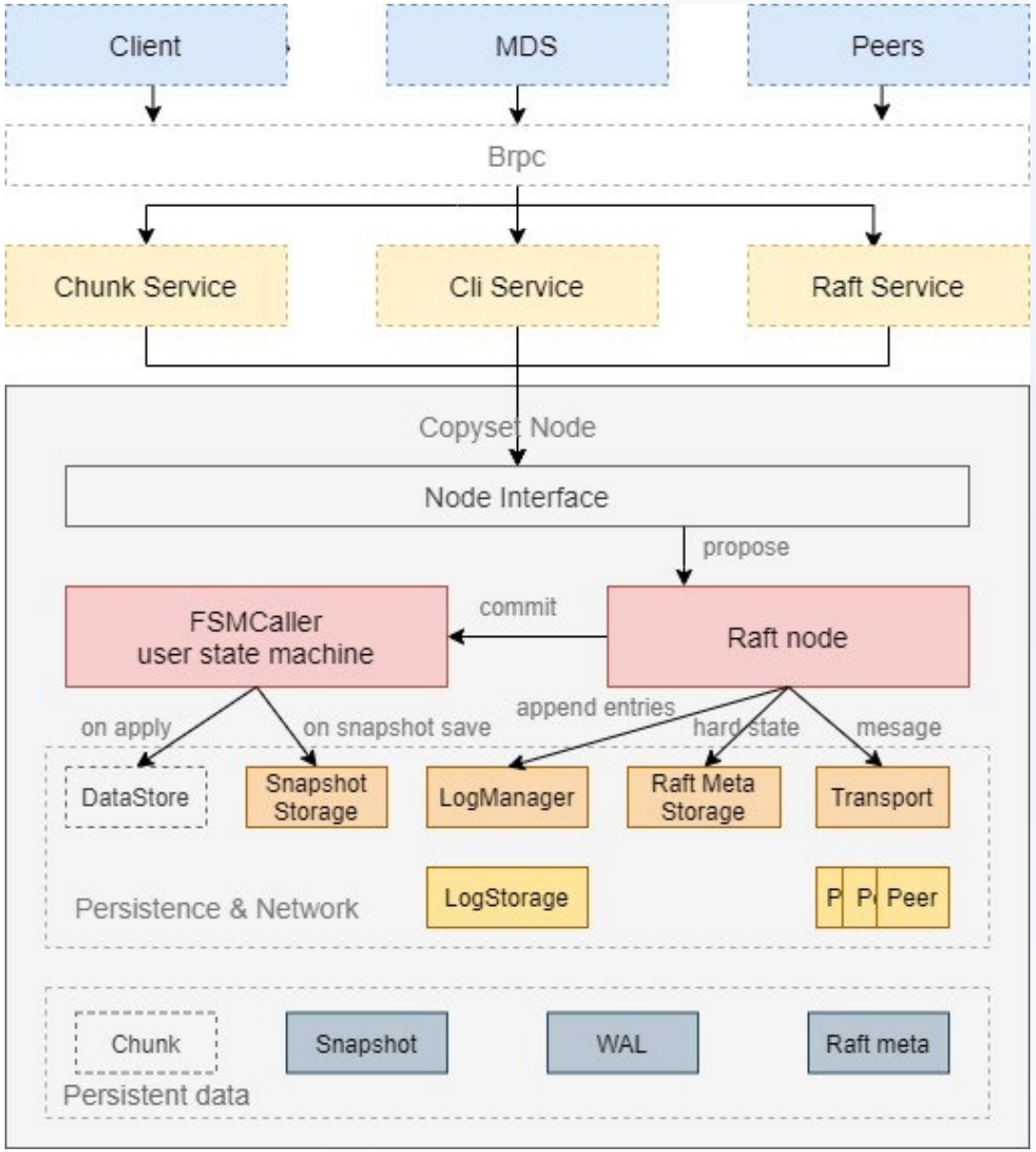
上报心跳



根据心跳下发raft成员变更



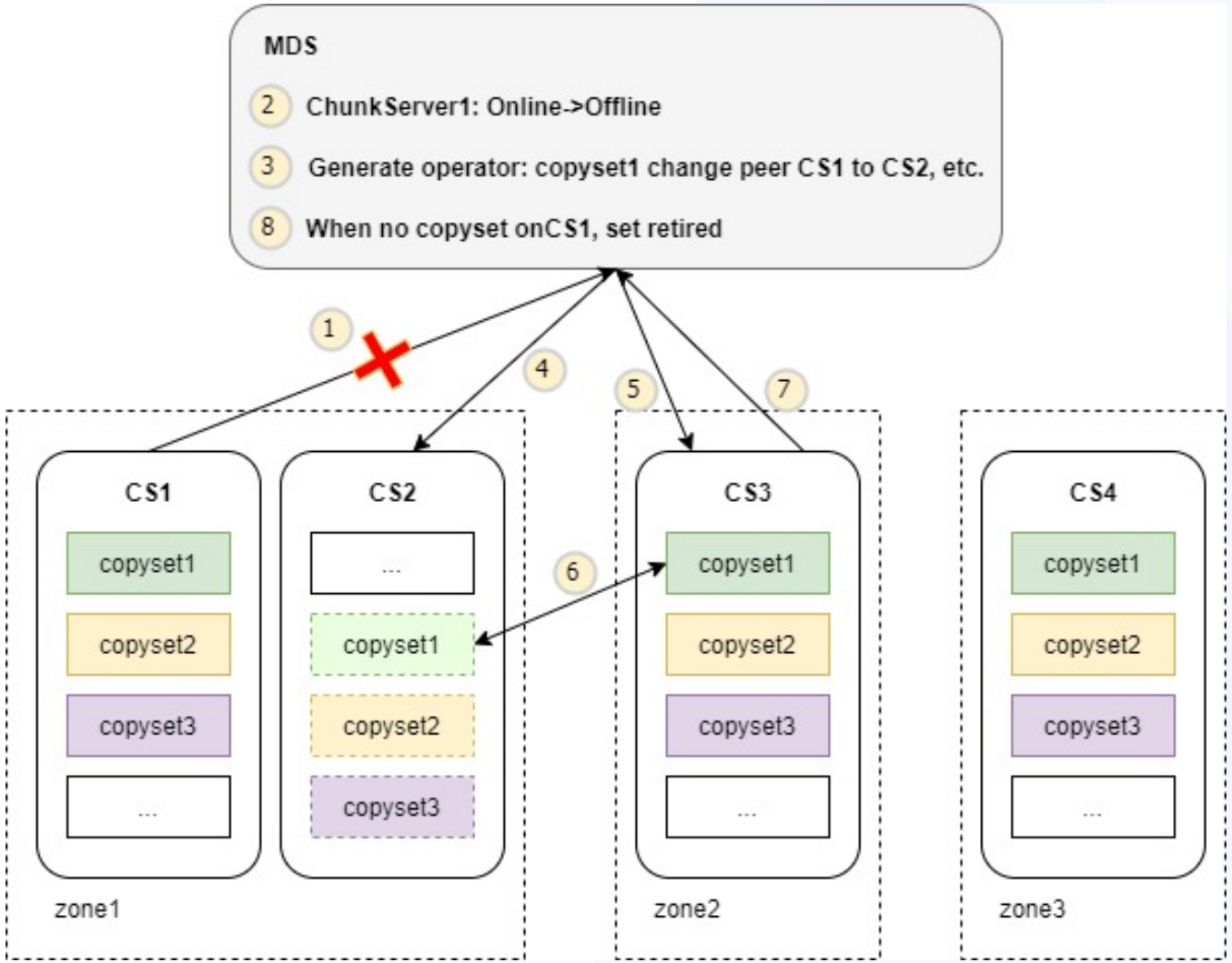
ChunkServer核心模块-CopysetNode



写请求：

1. Client发送写请求给Leader ChunkServer
2. 请求封装，提交给Raft node
3. 本地持久化entry的同时发送给其他peer
4. 本地持久化log entry成功，并且有一个peer也落盘成功，则commit
5. Commit后apply，此时把写请求写到chunk

ChunkServer核心模块-CopysetNode



坏盘（CS1对应的盘）后的迁移流程

初始状态，copyset1，copyset2，copyset3的三个副本分别在CS1,CS3,CS4上，完成迁移后，CS1上的副本迁移到CS2上

① CS1超时未向MDS上报心跳（默认半小时）

② MDS标记CS1状态为offline

③ MDS的recover scheduler发现copyset1, 2, 3的副本CS1 offline，生成change peer from CS1 to CS2的operator给这三个copyset

④ MDS通过RPC在CS2上创建copyset1,2,3这三个copyset

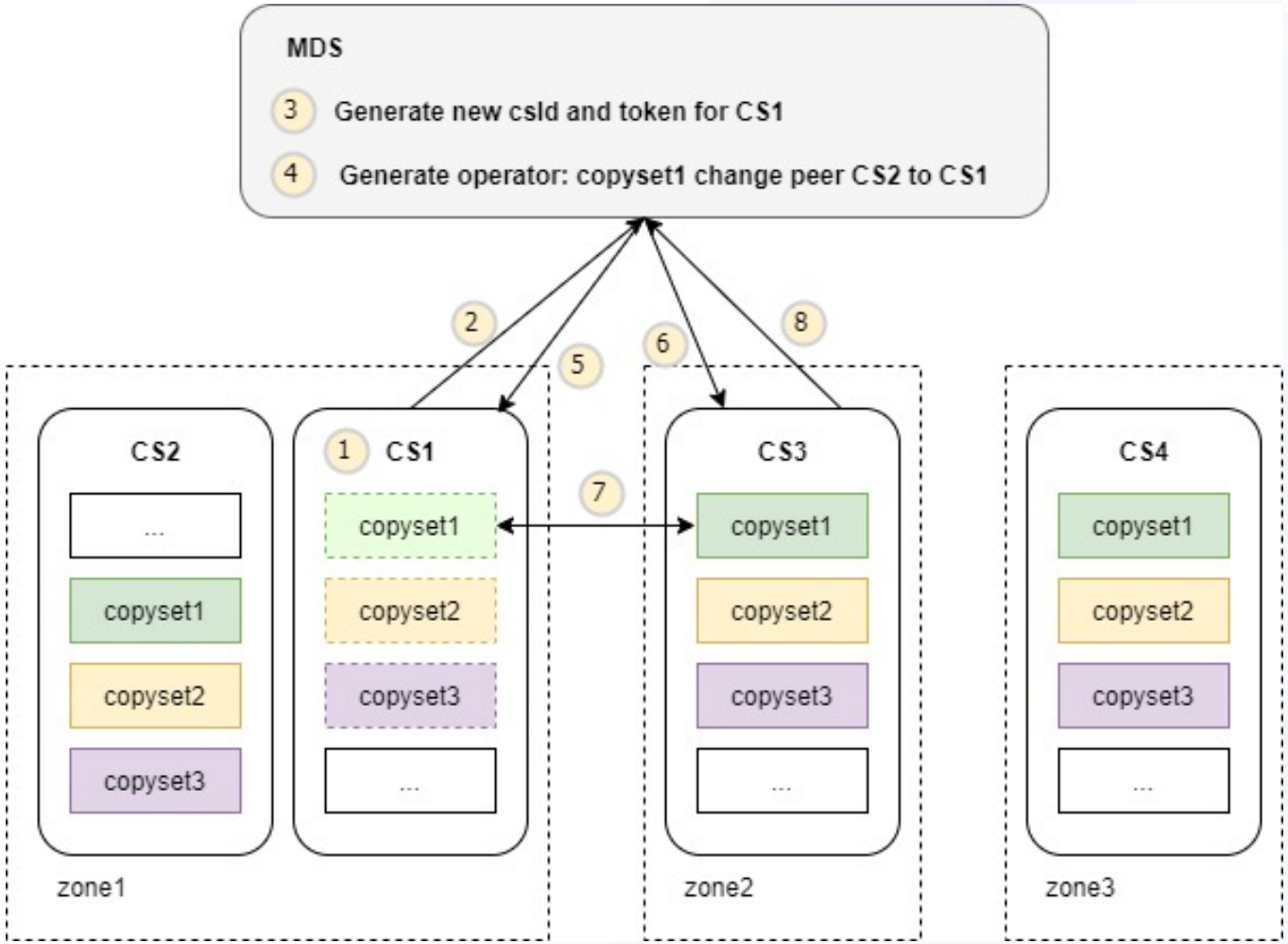
⑤ 假定三个copyset的leader都是CS3，在CS3的下一次心跳的response中，下发第三步生成的三个operator

⑥ CS3收到change peer from CS1 to CS2的operator，给CS2同步raft日志，当CS2成功赶上进度时，本次raft成员变更成功完成，CS2成为了复制组的一员，CS1不再属于这个复制组。

⑦ CS3在下一次心跳中向MDS报告本次raft成员变更已完成

⑧ MDS在得知CS1上的所有copyset都成功迁移后，把CS1设置为retired，CS1下线完毕。

ChunkServer核心模块-CopysetNode



换盘（CS1对应的盘）后重新上线的流程

初始状态，copyset1，copyset2，copyset3的三个副本分别在CS2,CS3,CS4上，完成恢复后，CS2上的copyset1,2,3迁移到CS1上

① CS1换了新盘，并重新格式化后启动chunkserver

② CS1重新向MDS注册

③ MDS生成新的chunkserver id和token给CS1

④ MDS的copyset scheduler发现CS1上的copyset数量为0，CS2上的copyset最多，生成change peer from CS2 to CS1的operator给部分copyset，比如copyset1,2,3

⑤ MDS通过RPC在CS1上创建copyset1,2,3这些copyset

⑥ 假定三个copyset的leader都是CS3，在CS3的下一次心跳的response中，下发第四步生成的三个operator

⑦ CS3收到change peer from CS2 to CS1的operator，给CS1同步raft日志，当CS1成功赶上进度时，本次raft成员变更成功完成，CS1成为了复制组的一员， CS2不再属于这个复制组。

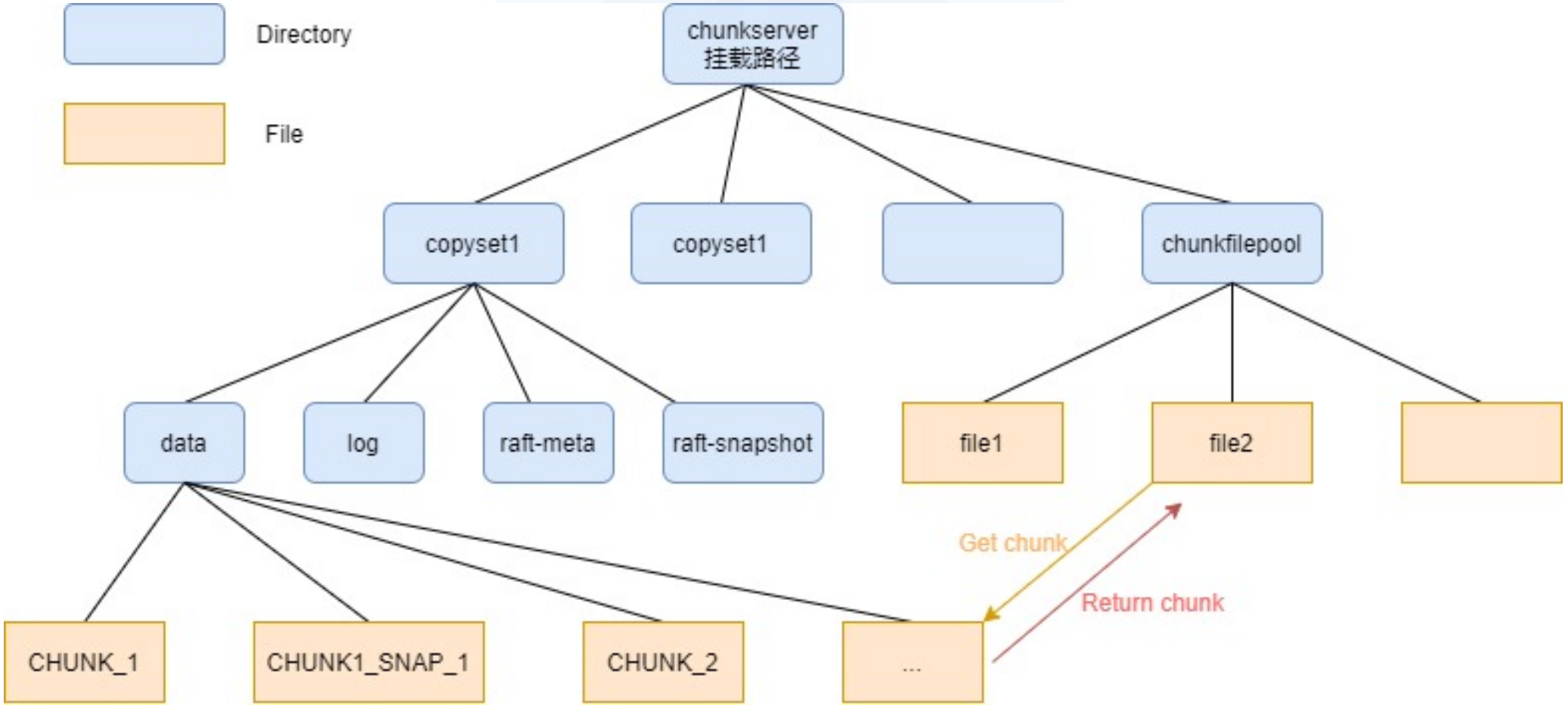
⑧ CS3在下一次心跳中向MDS报告本次raft成员变更已完成

⑨ 等CS1上的copyset数量恢复到和其它节点相差不大时，集群回到均衡状态，迁移结束

ChunkServer核心模块-DataStore

ChunkServer的目录结构：

- 每个copyset一个目录，后面三个目录由braft管理，data目录由DataStore管理
- Curve中的Chunk全部来自Chunkfilepool，是在系统初始化的时候预创建好并覆盖写过一遍的一些chunk，减少IO放大



Curve

DIGITAL SAIL

目录

新版本ChunkServer性能优化

Curve

Curve最新的release1.1中对client和chunkserver分别进行了优化，fio性能测试的结果如下表格：

Nbd单盘：

单盘性能对比	release1.1 iops/带宽	延时	99.00th	release 1.0	延时	99.00th	提升比例
4k randwrite (128 iodepth)	109k/s	1.1ms	2040us	62900/s	2ms	3ms	73%
4k randread (128 iodepth)	128k/s	1.0ms	1467us	76600/s	1.6ms	2ms	67%
512k write (128 iodepth)	204MB/s	314ms	393ms	147MB/s	435ms	609ms	38%
512k read (128 iodepth)	995MB/s	64ms	92ms	757MB/s	84ms	284ms	31%

Nbd 10盘：

多盘性能对比	release 1.1 iops/带宽	延时	99.00th (ms)	release 1.0 iops/带宽	延时	99.00th	提升比例
4k randwrite (128 iodepth)	262000/s	4.9ms	32ms	176000/s	7.2ms	16ms	48%
4k randread (128 iodepth)	497000/s	2.69ms	6ms	255000/s	5.2ms	22ms	94%
512k write (128 iodepth)	1122MB/s	569ms	1101ms	899MB/s	710ms	1502ms	24%
512k read (128 iodepth)	3241MB/s	200ms	361ms	1657MB/s	386ms	735ms	95%

新版本ChunkServer性能优化

How?

ChunkServer性能优化主要是braft日志落盘优化，包括三个方面：

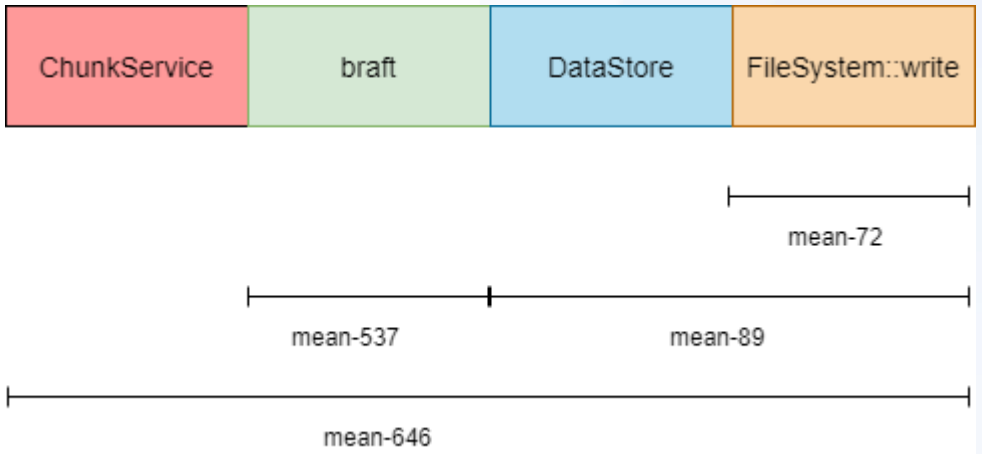
1、追加写改为覆盖写

2、写入时4KB对齐

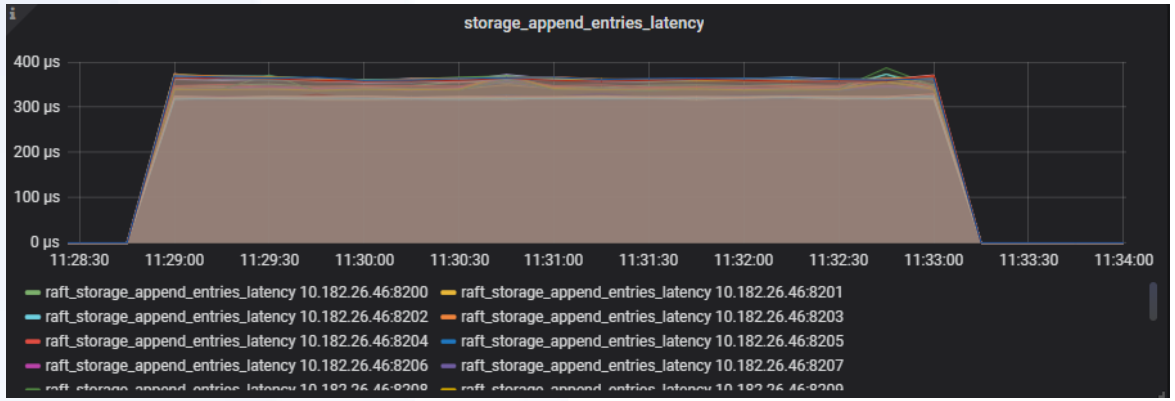
3、改为O_DIRECT模式

Why?

优化前性能分析：braft日志落盘时延占据了ChunkServer处理时延的50%以上，因此推断braft日志落盘存在瓶颈



单深度fio测试chunkserver内部时延统计



Raft 日志落盘延迟

新版本ChunkServer性能优化

Curve

1、模拟braft日志落盘方式的demo程序测试结果，latency为调用sync花费的时间（第一次新写，第二次覆盖写）；

```
charisu@pubbeta2-curve7:~$ python latency.py
None latency
mean: 311.152, std: 55.033943126, var: 3028.734896, min: 220.0, max: 1140.0
charisu@pubbeta2-curve7:~$ sudo ./test-sync | grep wal | awk -F " " '{print $4}' > latency.txt
charisu@pubbeta2-curve7:~$ python latency.py
None latency
mean: 132.1255, std: 254.333705493, var: 64685.6337498, min: 47.0, max: 11255.0
charisu@pubbeta2-curve7:~$ sudo ./test-sync | grep wal | awk -F " " '{print $4}' > latency.txt
charisu@pubbeta2-curve7:~$ python latency.py
None latency
mean: 124.3475, std: 52.8699985223, var: 2795.23674375, min: 50.0, max: 397.0
```

2、因为在请求4KB的情况下，写入的大小带上头部后是4120，是不对齐的，所以又测试了一下把写入大小改成4096的情况：

```
charisu@pubbeta2-curve7:~$ sudo ./test-sync | grep wal | awk -F " " '{print $4}' > latency.txt
charisu@pubbeta2-curve7:~$ python latency.py
None latency
mean: 42.917, std: 7.89196496444, var: 62.283111, min: 35.0, max: 144.0
charisu@pubbeta2-curve7:~$ sudo ./test-sync | grep wal | awk -F " " '{print $4}' > latency.txt
charisu@pubbeta2-curve7:~$ python latency.py
None latency
mean: 42.9795, std: 7.5927649608, var: 57.65007975, min: 35.0, max: 208.0
```

新版本ChunkServer性能优化

Curve

因此ChunkServer性能优化主要是braft日志落盘优化，包括三个方面：

1. 追加写改为覆盖写（避免每次写的时候改变元数据，减少IO放大）
2. 写入时4KB对齐（4KB不对齐的情况下，每次写入都会有读请求，从而影响效率）
3. 改为O_DIRECT模式（改为Direct模式可以避免显式调用sync）

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