curvefs copyset与fs对应关系

版本	时间	修改者	修改内容
1.0	2021/7/23	陈威	初稿
1.1	2021/8/4	陈威	根据评审意见修改
1.2	2021/8/9	陈威	增加详细设计

1、背景

curvefs使用raft作为元数据一致性的保证。为了提高元数据的可扩展性和并发处理能力，采用元数据分片的方式管理inode和dentry的元数据。inode的分片依据是fsid + inodeid，dentry的分片依据是fsid + parentinodeid。借鉴curve块设备的设计思路，（补充copyset的设计文档在这 ），curvefs的元数据分片仍然按照的copyset的方式去管理。

curve块存储的topo信息由PhysicalPool、LogicalPool、Zone、Server、ChunkServer、CopySetInfo组成。curvefs可以照搬curve块存储的topo设计，只是保存的内容从数据变成了元数据。

curvefs的topo信息设计可以由PhysicalPool、LogicalPool、Zone、Server、MetaServer、CopySetInfo组成。

curve块设备的copyset是在空间预分配的时候就确定了，每次预分配1GB的空间，然后这1GB的空间每个chunk对应的copyset在预分配的时候已经确定。后续的读写的操作直接去对应的copyset上去进行读写。这个分配copyset方式，并不适合curvefs的元数据。这种分配方式是提前分配了一批空间，即使用户只需要写4KB数据，也一次性分配1GB的空间。而curvefs的元数据，并不能一次申请一批在client端，而是每次都需要去metaserver上去进行分配。

这里需要重新考虑curvefs的copyset和fs的元数据分片的对应关系。

2、chubaofs的元数据管理

chubaofs（补充链接）的元数据也是采用的raft的方式进行管理，可以借鉴一下chubaofs的元数据的分片策略。

通过分析chubaofs的源代码。chubaofs的用volume管理一个文件系统，每个volume有若干meta partition和data partition。meta partition管理的元数据，data partition管理数据。meta partition管理inode和dentry信息。

创建一个文件系统时，如何初始化meta partition?

master\cluster.go， chubaofs的文件系统使用volume的来表示，在创建一个文件系统的时候，会创建3个meta partition和10个data partition。chubaofs的data partition的功能我们使用curve块设备替换。meta partition的创建，以及meta partition的管理的，下面会详细分析一下。

2.1、meta partition的创建

再创建文件系统的时候，会创建好指定大小的metapartition，meta partition的数量最小不低于3，最大不高于100。每个meta partition（除了最后一片）管理的inode id的个数为2^24，最后一片metapartition管理剩下的一直到2^63-1的Inode id。创建meta partition的时候，选择的3个meta node组成一个复制组。如何选择？论文上写的是按照存储节点的memory和disk usage来选的，通常选择内存和disk使用率最低的节点。

并去对应的meta node上去创建对应的meta partition。

```
76      defaultInitMetaPartitionCount                       = 3
77      defaultMaxInitMetaPartitionCount                    = 100
78      defaultMaxMetaPartitionInodeID          uint64 = 1<<63 - 1
79      defaultMetaPartitionInodeIDStep         uint64 = 1 << 24
80      defaultMetaNodeReservedMem              uint64 = 1 << 30
```

```
defaultIntervalToAlarmMissingMetaPartition          = 10 * 60 // interval of checking if a replica is missing
defaultMetaPartitionMemUsageThreshold      float32 = 0.75    // memory usage threshold on a meta partition
defaultMaxMetaPartitionCountOnEachNode              = 10000
defaultReplicaNum                                   = 3
defaultDiffSpaceUsage                               = 1024 * 1024 * 1024
```

如何选择partition的host，通过这个函数去选择。

```
func (c *Cluster) chooseTargetMetaHosts(excludeZone string, excludeNodeSets []uint64, excludeHosts []string, replicaNum int, crossZone bool, specifiedZone string) (hosts []string, peers []proto.Peer, err error)
```

metanode是否能够创建copyset，由这个函数判断。有这些判断条件：
1、metaNode的存活状态
2、metaNode的内存使用情况
3、metaNode的磁盘使用情况

4、metaNode上的partition的个数

```
func (metaNode *MetaNode) isWritable() (ok bool) {
    metaNode.RLock()
    defer metaNode.RUnlock()
    if metaNode.IsActive && metaNode.MaxMemAvailWeight > gConfig.metaNodeReservedMem &&
        !metaNode.reachesThreshold() && metaNode.MetaPartitionCount < defaultMaxMetaPartitionCountOnEachNode {
        ok = true
    }
    return
}
```

2.2、meta partition的管理

当这个partition inode用完了怎么办？当partition管理的分片的inode id分配完了。这个partition会变成readonly状态，不再接收新的inode的申请，但是dentry可以继续。而且meta partition还会自动的分裂，分裂是把volume的最后一个partition切出来。比如一个partition管理100个inode，最后一个partition是[100, max]，切完之后，变成了[100， 200]， [200, max]。怎么维持一定数目的meta partition数据，目前还没在代码里找到对应的逻辑。

Algorithm 1 Splitting Meta Partition

```
1:  procedure PARTITIONING
2:      mp ← current meta partition
3:      c ← current cluster
4:      v ← cluster.getVolume(mp.volName);
5:      maxPartitionID ← v.getMaxPartitionID()
6:      if metaPartition.ID < maxPartitionID then return
7:      if mp.end == math.MaxUint64 then
8:          end ← maxInodeID + Δ          ▷ curoff the inode range
9:          mp.end ← end
10:         task ← newSplitTask(c.Name, mp.partitionID, end)
11:
12:         c.addTask(task)               ▷ sync with the meta node
13:
14:         c.updateMetaPartition(mp.volName, mp)
15:         c.createMetaPartition(mp.volName, mp.end+1)
```

2.3、meta partition和inode以及dentry的对应关系？

怎么确定inode和dentry于partition的对应关系？

创建inode的时候，获取这个volume的所有的可用的（RW状态）meta partition，然后使用round robin的方式，遍历尝试去所有的partition中，直到找到一个partition可以创建inode。

```
rwPartitions = mw.getRWPartitions()
length := len(rwPartitions)
epoch := atomic.AddUint64(&mw.epoch, 1)
for i := 0; i < length; i++ {
    index := (int(epoch) + i) % length
    mp = rwPartitions[index]
    status, info, err = mw.icreate(mp, mode, uid, gid, target)
    if err == nil && status == statusOK {
        goto create_dentry
    }
}
return nil, syscall.ENOMEM
```

创建dentry，去parent inodeid所在的meta partition进行创建就好了。

```
create_dentry:
    status, err = mw.dcreate(parentMP, parentID, name, info.Inode, mode)
    if err != nil {
        return nil, statusToErrno(status)
    } else if status != statusOK {
        if status != statusExist {
            mw.iunlink(mp, info.Inode)
            mw.ievict(mp, info.Inode)
        }
        return nil, statusToErrno(status)
    }
```

查找inode和partition的时候，通过inodeid去查询应该由哪个partition进行处理。inode是拿着inodeid查询，dentry是拿着parent的inode id去查询。

```go
func (mw *MetaWrapper) getPartitionByInode(ino uint64) *MetaPartition {
    var mp *MetaPartition
    mw.RLock()
    defer mw.RUnlock()

    pivot := &MetaPartition{Start: ino}
    mw.ranges.DescendLessOrEqual(pivot, func(i btree.Item) bool {
        mp = i.(*MetaPartition)
        if ino > mp.End || ino < mp.Start {
            mp = nil
        }
        // Iterate one item is enough
        return false
    })

    return mp
}
```

一个fs的meta partition使用第一个叫做MetaWrapper的结构体组织起来

```
type MetaWrapper struct {
    sync.RWMutex
    cluster         string
    localIP         string
    volname         string

 ……

    // Partitions and ranges should be modified together. So do not
    // use partitions and ranges directly. Use the helper functions instead.

    // Partition map indexed by ID
    partitions map[uint64]*MetaPartition

    // Partition tree indexed by Start, in order to find a partition in which
    // a specific inode locate.
    ranges *btree.BTree

    rwPartitions []*MetaPartition


 ……
```

3、curvefs的copyset和fs的对应关系

curvefs的元数据的分片，需要考虑到在创建inode的时候，其实是不知道inodeid的，在创建完成之后，才有inodeid。inodeid的分配最好下放到各个分片去进行处理。否则整个集群的inode都去一个地方获取id会造成巨大的锁开销，这个是不能接受的。

curve块设备的元数据管理，在分配数据的时候，offset一开始就是知道的，这是和curvefs分配很大的一个不同点。

假设已经确定了一个分片规则，那么根据这个分片规则，一定可以找到两个函数

inodeid到copyset的映射：copysetid = getPartition(inodeid)

copyset管理的inode的范围：inoderange = getInodeRange(copyset)

3.1 如何获取inodeid

在create inode的时候，并不知道inode id，inode id是在创建完成之后返回的，这就没有办法利用分片规则去确定到底应该由哪个copyset去服务这个inode。有两种思路。

思路一：client在创建inode的时候，先去mds去获取一个inodeid，然后根据这个inode id找到服务这个inode的分片。出于性能上的考虑的，client可以一次从mds获取一批inode，这批inode用完了之后，再去mds去申请。

思路二：client在创建inode的时候，自己选择一个分片，然后由这个分片自己分配一个inode。采用这种思路，在create fs的时候，就为fs准备好的几个copyset，然后client把copyset缓存在本地。每个copyset管理一段inode。选定copyset，就选定了服务的3个metaserver。至于均衡上，创建inode的时候，轮流在这个fs的copyset上进行创建。这种方式肯定不如curve块设备的方案分配的那么均衡。

结论：采用思路二，由分片管理fs

3.2 copyset fs共用吗？

fs是否共用copyset，在实现上，只需要获取分片的时候把inodeid替换成fsid+inodeid的组合就行了。在获取inode和copyset的对应关系上，实现起来差别不大。fs是否共用copyset的影响比较大的方面在其他地方。一个是copyset的数目，如果的每个fs独占copyset，那么整个系统的copyset的个数一定会比非独占多。copyset对资源的占用开销大不大。大量的copyset会不会因为太吃资源导致性能反而下降。chubaofs的方案里面，每一个metanode上能够服务的copyset个数是有限制的，当内存和磁盘的到达一定的限度之后，这个metanode就变成readonly的，然后也不让分配新的copyset了。每个copyset的能力*copyset的个数 = 这个metanode的的处理能力。通过合理的配置copyset的能力的，应该的可以避免一个机器上，有太多的copyset。

结论：coypset由fs共用。具体的使用上，每一个copyset上，有一个可以由多少fs共用的限制。这个限制通过配置文件进行配置。用户挂载时可以通过参数配置是否独占copyset。原因是，为了避免fs独占copyset带来的copyset数量过多影响性能的问题。

3.3 copyset个数是否可以动态调整？

根据copyset个数是否可以动态调整，有两种实现。

一种是curve块存储方案，在集群初始化的时候，把所有的copyset创建好。采用这种方式，可以采用hash的方式去确定inode的分片。比如说，copysetid = （fsid + inodeid << shift) % totalCopysetNum，如果采用hash方案，扩容按照pool扩容，避免hash带来的数据迁移。用这种方式简化处理，改变hash映射方式带来的数据迁移，在技术实现上难度应该很大，暂时不考虑。

还有一种方式是chubaofs方案，在文件系统初始化的时候，初始化少数copyset，然后copyset的处理能力有限，当copyset的使用能力达到一定的限度的时候，这个copyset转化为readonly，继续创建新的copyset提供服务。

结论：copyset的个数动态调整，类似的chubaofs的方案，一开始的为fs分配少数copyset。随着fs使用和新建，动态的新建copyset。

4、curvefs的topo信息

curvefs的topo信息可以照搬curvef块设备的topo的实现，只需要把chunkserver改成metaserver。

curvefs的topo信息的层级最终是这样：

→pool ：存储池（curve的physical pool和logic pool这里合并，只保留一个pool）

→zone：可用域

→server：代表着一台服务器

→metaserver：代表着一块盘

每个copyset的由处于不同zone的metaserver组成复制组。

5、curvefs mds和metaserver的心跳

curvefs的mds和metaserver之间的心跳类似于curve块设备的心跳。metaserver需要定期通过心跳向mds上报自己的状态。mds一方面根据metaserver上报的状态，进行相应的调度；另一方面根据心跳确认metasever的存活状态。

这块内容参考的curve。metaserver定时向mds上报心跳，心跳内容参考curve。调度上，如果采用类似chubaofs的方案，那么copyset就会一直处于一个不均衡的状态中。如果copyset是可readwrite状态，新创建的文件和目录，会导致copyset管理的元数据越来越多；如果copyset管理的Inode分配完了，转为readonly状态，随着文件和目录的删除，copyset管理的元数据会越来越少。类chubaofs方案的均衡问题如何解决?

CopySetScheduler: copyset均衡调度器。根据集群中copyset的分布情况生成copyset迁移任务;
LeaderScheduler: leader均衡调度器。根据集群中leader的分布情况生成leader变更任务;
ReplicaScheduler: 副本数量调度器。根据当前copyset的副本数生成副本增删任务;
RecoverScheduler: 恢复调度器。根据当前copyset副本的存活状态生成迁移任务。

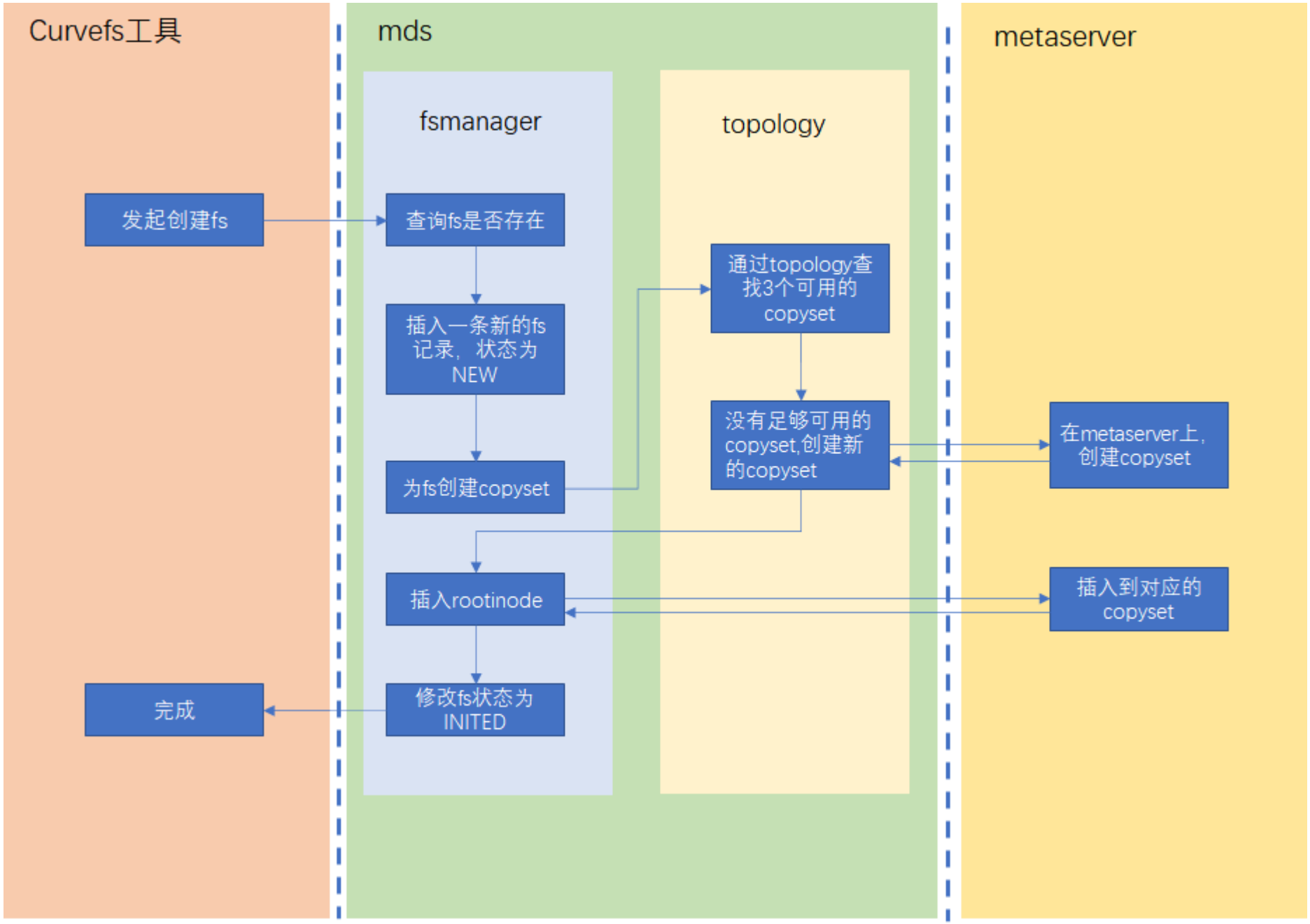
结论：心跳参考curve。目前这些调度器在curvefs第一阶段不用全部实现。所有和均衡相关的，暂时不做。只做和故障处理相关的副本补全恢复的调度。

6、详细设计

6.1 创建fs

curvefs管理工具发起创建fs命令，mds收到createfs命令之后，在mds插入的一条fs记录，状态为NEW。然后为fs创建copyset，默认为3个。mds调用topology的接口找到3个可用的copyset，如果没有足够可用的copyset，就创建新的copyset补齐。

然后生成一条rootInode的记录，根据copyset的分片规则，在对应的copyset上插入一条rootinode的记录。最后修改fs状态为INITED。


Curvefs工具
mds
metaserver
fsmanager
topology
发起创建fs
查询fs是否存在
插入一条新的fs记录，状态为NEW
为fs创建copyset
通过topology查找3个可用的copyset
没有足够可用的copyset,创建新的copyset
在metaserver上，创建copyset
插入rootinode
插入到对应的copyset
修改fs状态为INITED
完成

6.2、挂载fs

挂载fs的流程不变，client向mds发送mount rpc请求，mds对fs进行相应的检查，然后记录挂载点返回成功。

1、检查文件系统是否存在

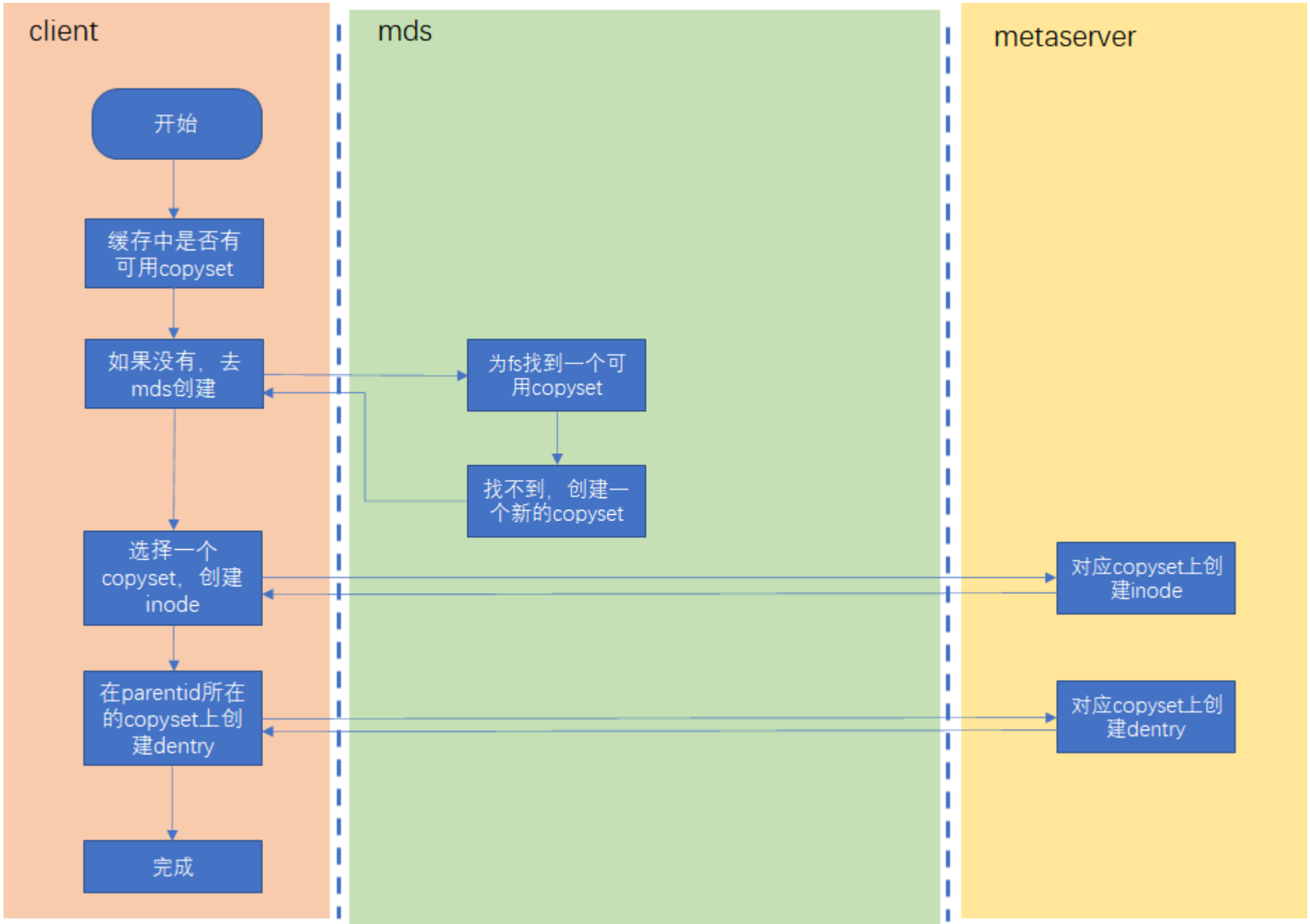
2、检查fs的状态，是否是INITED状态

3、检查挂载点是否已经存在

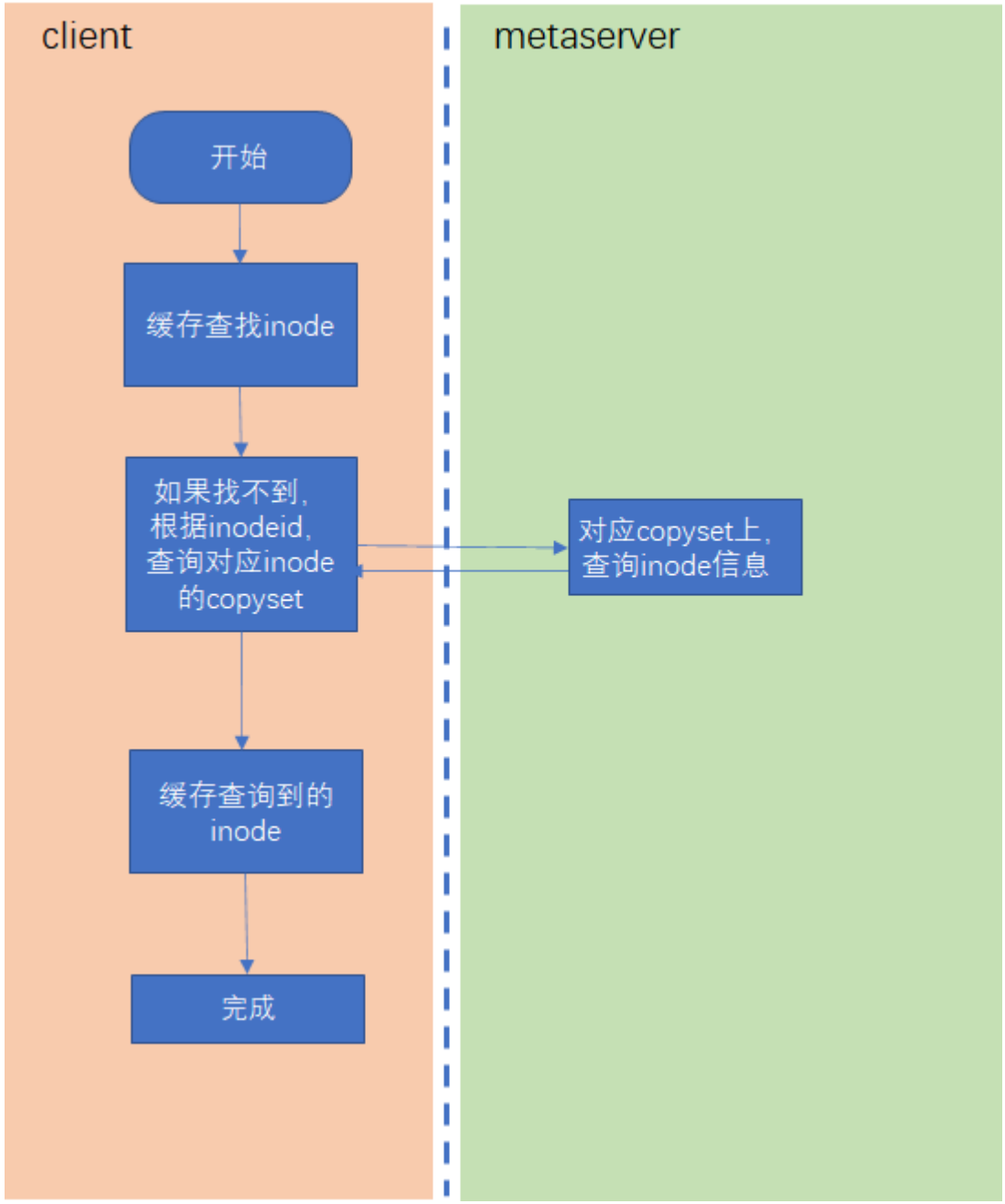
6.3、创建文件/目录

client在创建inode的时候，如何选择copyset。client在fs的所有可用的copyset中，轮询进行inode的分配。如果选择的copyset创建inode失败，比如说metaserver返回copyset上的资源已经满了，这时client需要把这个copyset的转为readonly模式，这个copyset不再承担inode的新建功能。client继续尝试下一个copyset，直到成功从一个copyset上创建到1个inode。

client在系统初始化的时候，还需要起来一个后台线程，定期的检查每一个fs的copyset的状态，如果某一个fs的可以提供分配inode能力的copyset的个数小于规定的值（来自配置文件，默认3个），就为这个fs创建一个新的copyset。


client
mds
metaserver
开始
缓存中是否有可用copyset
如果没有，去mds创建
为fs找到一个可用copyset
找不到，创建一个新的copyset
选择一个copyset，创建inode
对应copyset上创建inode
在parentid所在的copyset上创建dentry
对应copyset上创建dentry
完成

6.4、open流程


client
metaserver
开始
缓存查找inode
如果找不到，
根据inodeid，
查询对应inode
的copyset
对应copyset上，
查询inode信息
缓存查询到的
inode
完成

6.5、读写流程

读写流程和之前的读写流程大致保持不变。变化点在于之前inode修改是直接去metaserver上修改，现在变成了去copyset上修改。

client端缓存所有open的inode，读写的时候，根据inode的元数据，去对应的volume或者S3进行读写。如果涉及到inode的修改，根据inodeId查询对应的copyset，去对应的copyset进行inode的更新。

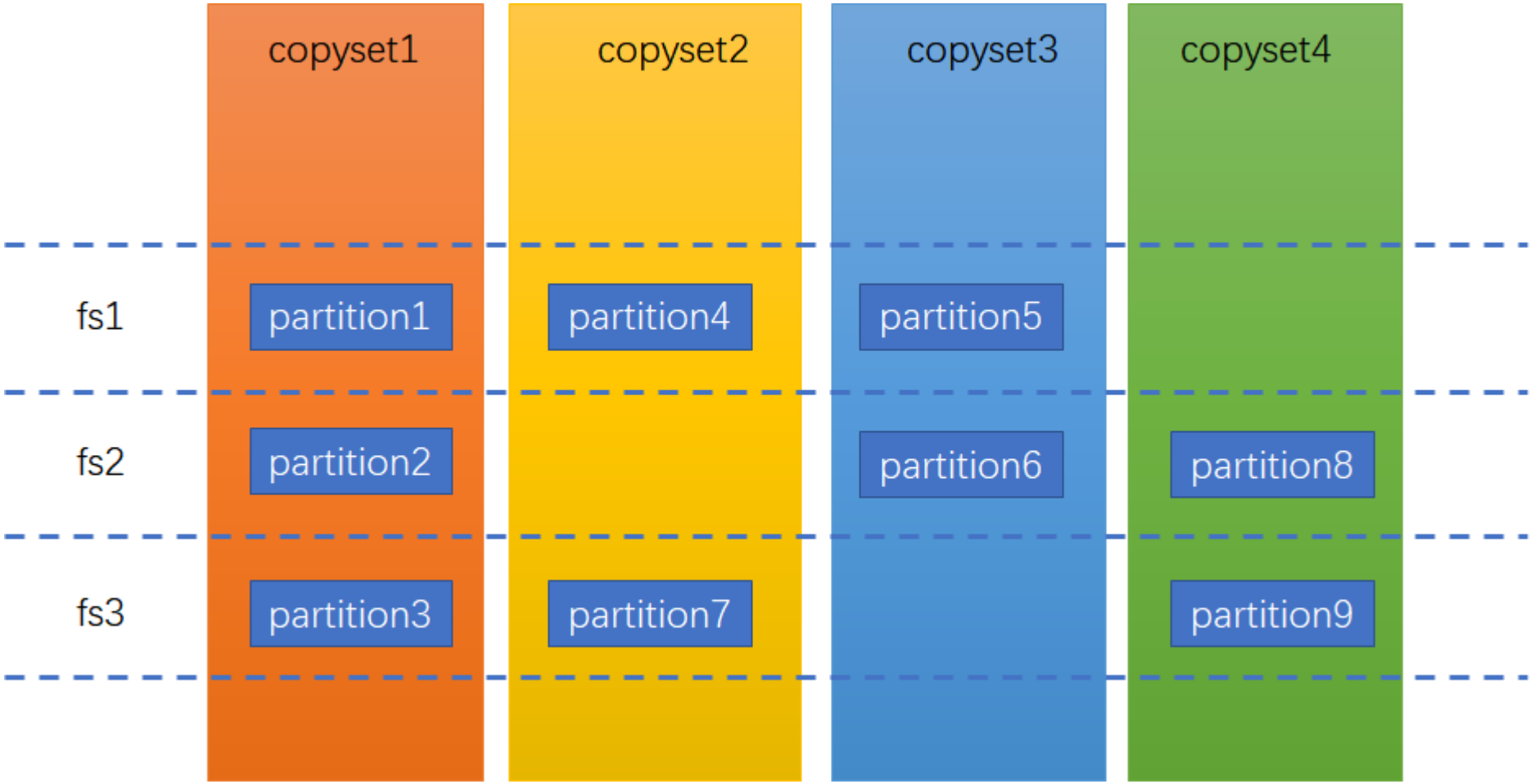
```
bool  CopysetManager::GenCopyset(const  ClusterInfo&  cluster,    int  numCopysets,    std::vector<Copyset>*  out)
```

6.6、topology

topology参考curve的topology的实现，由于curve的physical pool和logic pool在curvefs中合并成了一个，所以，并不能直接复用curve的topology的代码。

curve在创建logic
pool的时候去创建copyset。现在集群的topo信息在mds创建好了之后，topo中并没有coypset，而是提供接口，随用随创建。copyset选择哪些metaserver的作为3副本的过程，暂时先复用的原来curve块存储创建co
pyset的流程，将来再做优化。

文件系统的分片，用partition表示，每个partition由一个copyset管理。每个copyset管理的若干个partition。对应关系如下图所示。



fsinfo下面增加partition字段，用来维护fs的partition信息。

copyset在以前copyset的基础上，增加partiton信息。

partition id需要全局唯一。

7、工作评估

7.1 client端

1、mount的时候，获取这个fs的所有partition和copyset信息。分片信息的缓存。

2、paritition的选择。

3、和metaserver进行交互的时候，向对应的partition下发请求。包括get leader，重试。

4、和metaserver交互时，request请求需要带上copyset信息。

7.2 mds端

1、需要实现topo模块

2、实现mds和metaserver的心跳

3、实现fs和copyset的分片策略的实现

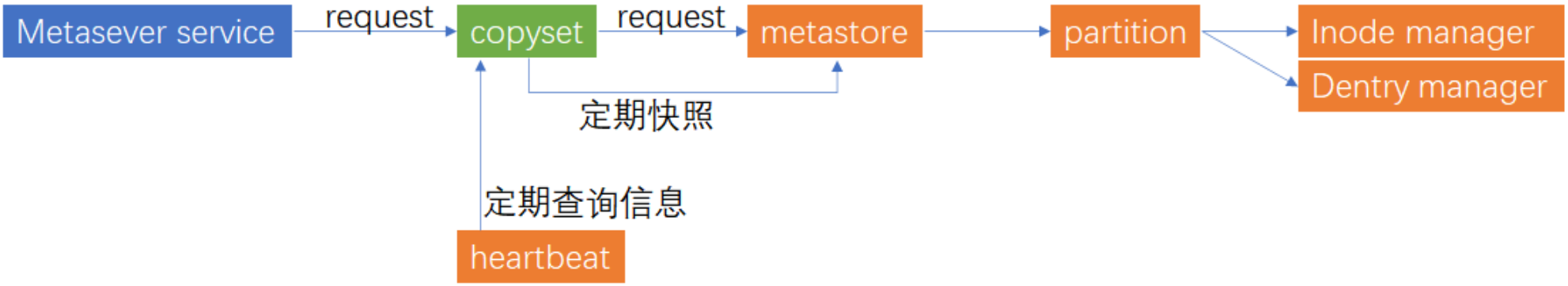
4、实现出现异常场景下的数据恢复，副本修复的调度。

7.3 metaserver端

1、需要提供copyset的创建功能

2、由copyset负责inode和dentry的管理

3、定期向mds上报心跳，并根据心跳结果执行配置变更



metaserver 子模块拆分

metaserver service：接受rpc请求

copyset：负责对元数据的持久化，主要是一致性协议raft的处理

metastore：负责元数据的内存部分的管理，负责选partition

partition：负责元数据的一段分片，每个元数据一定有对应的partition进行处理

inode manange/ dentry manager：负责管理元数据的内存结构

heartbeat：定期获取copyset的信息

模块	估算工作量（开发 + ci完成）
client	10d
mds	15d
metaserver	10d

考虑到partition和copyset的多对一关系会带来开发商的复杂性，是否考虑先只实现partition和copyset一对一的情况。等下一个版本，再实现的多对一的场景。

接口设计：https://github.com/opencurve/curve/pull/495

增加copyset.proto

增加heartbeat.proto

增加topology.proto

8、inode和dentry的内存估算

类型	byte
sizeof(dentry)	56
dentry的name字段，按照最大估算	256
sizeof(inode)	112
sizeof(volumeExtentList)	48
sizeof(S3ChunkInfoList)	48
sizeof(S3ChunkInfo)	64
sizeof(VolumeExtent)	56

dentry大小：

空的dentry占用56B；

最大的dentry占用字节56 + 256 = 312 B；

inode大小：

空的inode占用112 + 48 + 48 = 208B;

目录类型的inode占用208B;

link类型的inode最大占用208 + 256 = 464B;

volume file类型的inode，按照1w条extent估算，占用内存208 + 10000*56 = 560208B;

s3 类型的inode，按照1w条s3info估算，占用内存208 + 10000*64 = 640208B;

8.1 一台机器上能存放多少个inode和dentry

由于元数据全部缓存在本地，而且磁盘空间远大于内存空间，所以一台机器上能放多少个inode和dentry最大的限制在于内存。

按照最差的情况，文件里面全部都是碎片，那么metaserver上的空间碎片将会占用最多的空间。这里忽略掉其他次要的因素，考虑全是空间碎片信息，每条记录只保存4KB数据，可以保存多少元数据。

选取占用空间更多的S3ChunkInfo。按照一台metaserver 256GB内存容量全部用来保存空间分配计算。可以的保存chunkinfo 条数 = 256GB / 64B = 4G。可以保存的文件的大小为4G * 4KB = 64TB的空间。

inode和dentry按照1:1估算，dentry按照name使用最大字节，选择占用空间更多的s3来计算。

文件大小	dentry大小	inode大小	可以保存inode和dentry数
1MB	312B	208B + (1MB / 4KB) * 64B = 16,592B	16261116
10MB	312B	208B + (10MB / 4KB) * 64B = 164,048B	1672413
100MB	312B	208B + (100MB / 4KB) * 64B = 1,638,608B	167718
1000MB	312B	208B + (1000MB / 4KB) * 64B = 16,384,208B	16776

8.2 一台机器上建议的copyset数量

当前curve机器上的copyset的数量是100个。curvefs也可按照curve的规格，每个机器上管理100个copyset。实际上这个值通过配置文件控制，到时候可以根据测试结果确定合适的copyset的数量。

8.3 每个copyset建议管理存储容量的大小

如果有100个的copyset，每个copyset管理2GB大小的元数据。