
curve条带特性设计文档

- 修订记录
- 背景介绍
- 方案设计
 - 条带化特性
 - 数据结构
 - 条带特性需要修改的流程
- 详细流程描述
- 升级相关
 - cinder&&nova
 - storage&&rpc

修订记录

时间	内容	作者	Reviewer
2020. 12. 15	完成初稿	胡遥	
2020. 12. 17	修复评审意见	胡遥	curve全体人员

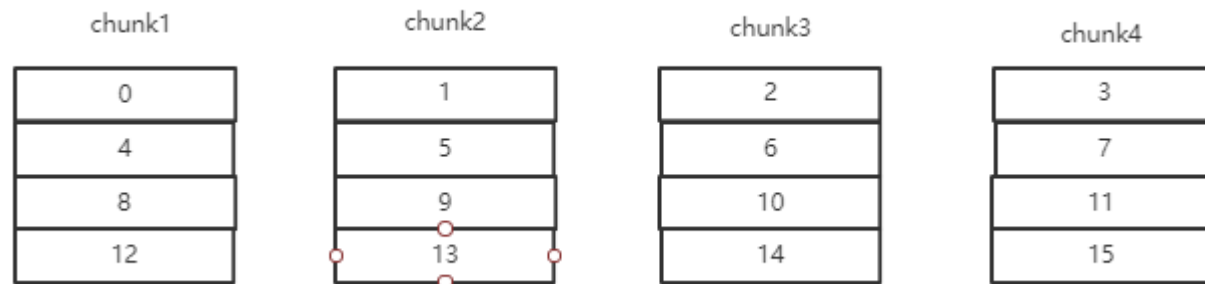
背景介绍

curve在大IO顺序写的场景下性能表现比较差，主要原因是顺序写一段时间内会集中发到某一个copyset，导致并发性不够，而某个copyset拥堵导致平均时延变高。在分析清楚愿意之后，考虑通过增加卷的条带特性来加大client到copyset的并发性，提高顺序写性能。

方案设计

条带化特性

条带化行为主要受3个参数的控制：
chunksize：默认文件的大小16MB
stripe_unit：条带切分的粒度，每个stripe_unit的连续字节会被保存在同一个文件中，client写满stripe_unit大小的数据后，去下一个chunk写同样的大小。
stripe_count：表示条带的宽度，client连续写多少个stripe_unit后，又重新写第一个文件。
一般来说stripe_count越大，stripe_unit越小，顺序写并发被打的越散，并发度越高。
举例说明一下client写入io的顺序，以及chunk数据的分布情况。假如chunksize为8MB，stripe_unit 2MB，stripe_count 4。如下图：每个编号代表了一个条带分片，即stripe_unit 2MB的大小，编号表示了io的顺序，client是先写了2MB到chunk1上，然后写2MB到chunk2上，依次循环。直到这4个chunk都写满后，开始写新的一组条带。
注意：stripe_unit * stripe_count不一定等于chunksize，比如这里如果chunksize是16MB的话，那么chunk1还要继续追加写16, 20, 24, 28。



数据结构

条带涉及的主要数据结构包括：
client端

```

typedef struct FInfo {
    uint64_t      id;
    uint64_t      parentid;
    FileType      filetype;
    uint32_t      chunksize;
    uint32_t      segmentsize;
    uint64_t      length;
    uint64_t      ctime;
    uint64_t      seqnum;
    // userinfo
    UserInfo_t     userinfo;
    // owner
    std::string    owner;
    std::string    filename;
    std::string    fullPathName;
    FileStatus     filestatus;
    std::string    cloneSource;
    uint64_t       cloneLength{0};
    uint32_t       stripeUnit;
    uint32_t       stripeCount;
} FInfo_t;

message CreateFileRequest {
    required string    fileName = 1;
    required FileType  fileType = 3;
    optional uint64     fileLength = 4;

    required string    owner = 2;
    optional string     signature = 5;
    required uint64     date = 6;
    optional uint32     stripeUnit = 7;
    optional uint32     stripeCount = 8;
};

```

```
message FileInfo {
    optional uint64 id = 1;
    optional string fileName = 2;
    optional uint64 parentId = 3;
    optional FileType fileType = 4;
    optional string owner = 5;
    optional uint32 chunkSize = 6;
    optional uint32 segmentSize = 7;
    optional uint64 length = 8;
    optional uint64 ctime = 9;
    optional uint64 seqNum = 10;
    optional FileStatus fileStatus = 11;
    //,
    //RecycleBin/
    optional string originalFullPathName = 12;

    // cloneSource (curvefs)
    // s3
    optional string cloneSource = 13;

    // cloneLength cloneextent
    optional uint64 cloneLength = 14;
    optional uint32 stripeUnit = 15;
    optional uint32 stripeCount = 16;
}
```

条带特性需要修改的流程

mds端

CreateFile流程：新增条带参数的合法性检查，如果没有设置条带参数，默认不开条带功能。fileInfo新增条带参数的set/get函数，将条带数据保存在fileinfo中。

DecodeFile流程：新增的stripeUnit和stripeCount采用optional类型，支持新老版本的数据结构兼容。

client端

CreateFile流程：新增create2接口，增加条带参数。

读/写流程：修改统一的io切分函数I02ChunkRequests，不影响原有逻辑。

工具

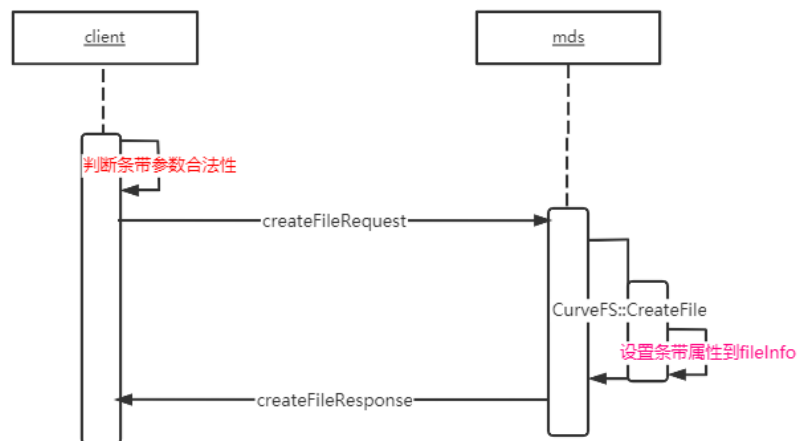
curve: create, stat

curve_ops_tool: 不需要改

curvefs_python: cbd_client: create接口; libcurvefs.h: create接口

详细流程描述

CreateFile流程，如下图。红色部分为需要添加的代码逻辑。



client读写流程:

这一块也是整个条带特性的核心修改流程，主要改动点是Splitor::IO2ChunkRequests函数。该函数主要是通过off和len找到读写对应chunklist以及chunk内的offset和len。由于之前poc已经完成相关代码，这里直接上代码分析

```
int Splitor::IO2ChunkRequests(IOTracker* iotracker, MetaCache* metaCache,
                             std::vector* targetlist,
                             butil::IOBuf* data, off_t offset, size_t length,
                             MDSClient* mdscclient, const FInfo_t* fileInfo) {
    ...
    targetlist->reserve(length / (iosplitopt_.fileIOSplitMaxSizeKB * 1024) + 1);

    const uint64_t chunkSize = fileInfo->chunksize;
    uint64_t stripeSize = fileInfo->stripeSize;
    const uint64_t stripeCount = fileInfo->stripeCount; //stripe count by one chunk
    if (stripeCount == 1) {
        LOG(INFO) << "stripe count is one, stripe size == chunk size";
        stripeSize = chunkSize;
    }
}
```

```

}
const uint64_t stripesPerChunk = chunkSize / stripeSize;

uint64_t cur = offset;
uint64_t left = length;
uint64_t curChunkIndex = 0;
while (left > 0) {
    uint64_t blockIndex = cur / stripeSize;
    uint64_t stripeIndex = blockIndex / stripeCount;
    uint64_t stripepos = blockIndex % stripeCount;
    uint64_t curChunkSetIndex = stripeIndex / stripesPerChunk;
    uint64_t curChunkIndex = curChunkSetIndex * stripeCount + stripepos;

    uint64_t blockInChunkStart = (stripeIndex % stripesPerChunk) * stripeSize;
    uint64_t blockOff = cur % stripeSize;
    uint64_t curChunkOffset = blockInChunkStart + blockOff;
    uint64_t requestLength = std::min((stripeSize - blockOff), left);

    /*LOG(INFO) << "request split curChunkIndex = " << curChunkIndex
        << ", curChunkOffset = " << curChunkOffset
        << ", requestLength = " << requestLength
        << ", cur = " << cur
        << ", left = " << left;*/

    if (!AssignInternal(iotracker, metaCache, targetlist, data,
        curChunkOffset, requestLength, mdsclient,
        fileInfo, curChunkIndex)) {
        LOG(ERROR) << "request split failed"
            << ", off = " << curChunkOffset
            << ", len = " << requestLength
            << ", seqnum = " << fileInfo->seqnum
            << ", chunksize = " << chunkSize
            << ", chunkindex = " << curChunkIndex;

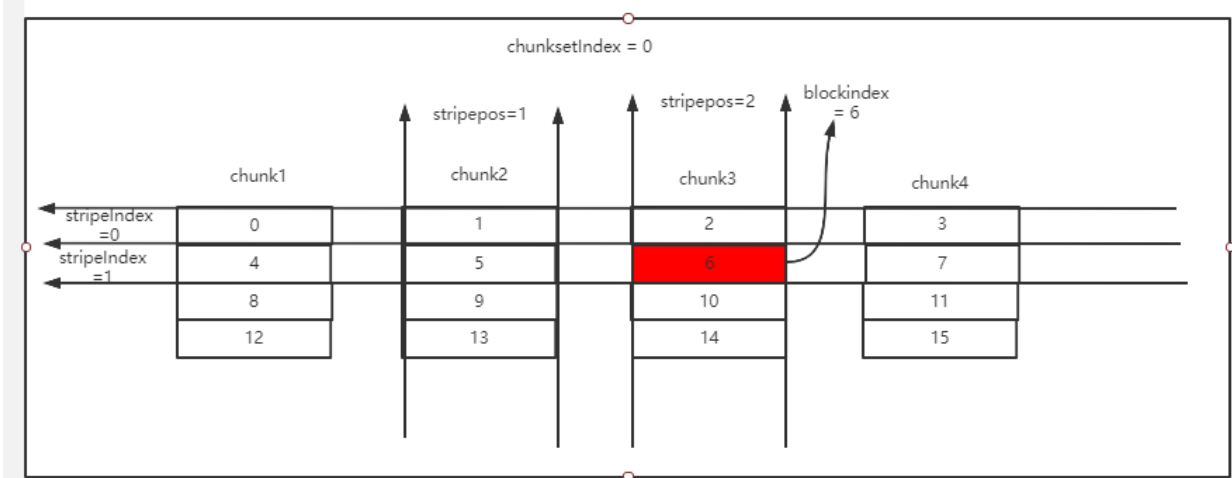
        return -1;
    }

    left -= requestLength;

```

```
        cur += requestLength;
    }
}
```

对代码中一些变量进行解释，如下图：



blockindex: 每个stripeSize大小可以认为是一个block，blockindex表示在整个文件中的block索引编号，如图红色部分的blockindex为6
stripeIndex: 如图每一行是一个stripe，stripeIndex表示是stripe的索引编号，如图红色部分的stripeIndex为1
chunksetIndex: 表示条带里包含的chunk组。如图，整个4个chunk组成一个chunkset，编号为0。
stripepos: 在chunkset中属于的chunk位置，如图，红色部分stripepos为2。
获取了以上几个数据后，我们可以计算真正需要获取的值。
curChunkIndex: chunk的索引，curChunkSetIndex * stripeCount （前面所有chunkset包含的chunk数量）+ stripepos
curChunkOffset: 在chunk中的偏移。先计算cur所在的block在当前chunk的的偏移位置即blockInChunkStart=(stripeIndex % stripesPerChunk) * stripeSize;
在计算cur在当前block的偏移位置: blockOff = cur % stripeSize;
最后curChunkOffset = blockInChunkStart + blockOff。
请求的长度为该block剩余的长度和整个请求剩余长度left的最小值: requestLength = std::min((stripeSize - blockOff), left);

升级相关

cinder&&nova

由于我们接口的改动，需要cinder那边也进行适配，并联动测试

storage&&rpc

存储和rpc涉及到的数据结构为message CreateFileRequest和message FileInfo，因为数据结构中添加的是optional参数，在序列化和反序列化都无报错。后续测试的时候也需要增对升级进行用例设计。