Ceph子树迁移

目的

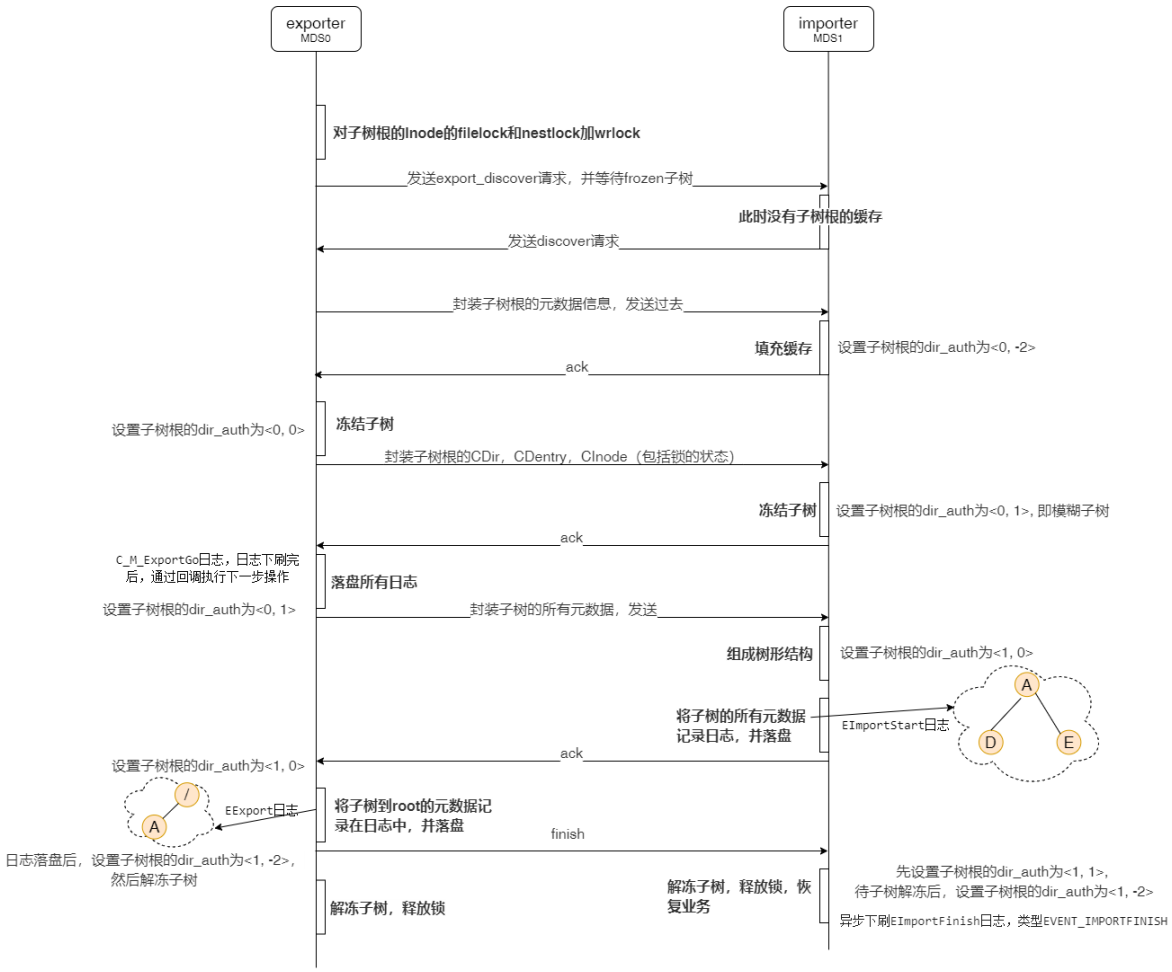
MDS集群间负载不平衡时，负载较重的节点通过将适当大小的子树迁移给负载较轻的节点，完成负载分担，最终达到整个集群负载的平衡。

影响

在子树迁移过程中，集群会阻塞来自客户端的请求，直到子树迁移完成，所以大量或频繁的子树迁移肯定会导致业务抖动，影响集群性能。

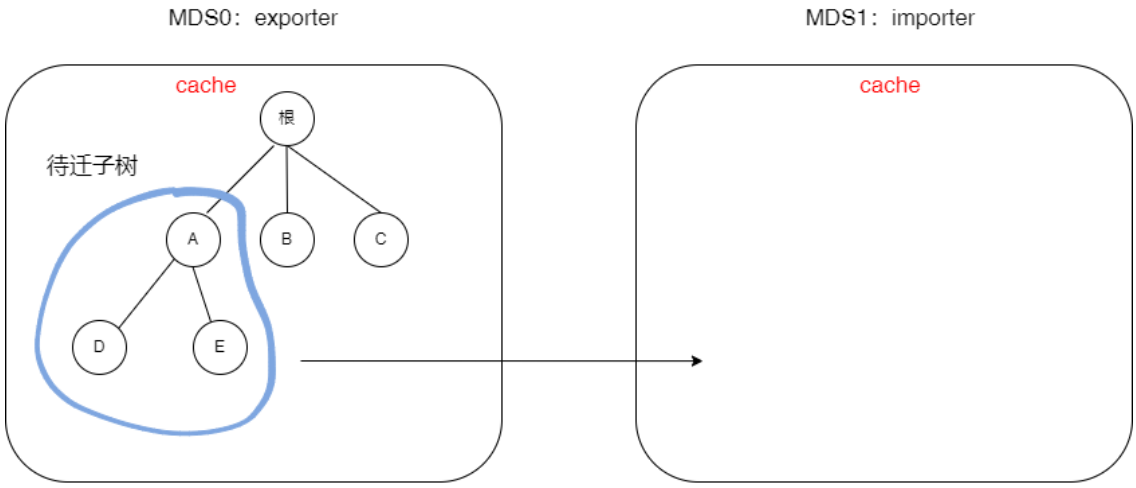
流程图

整个子树迁移的流程如下



简略过程

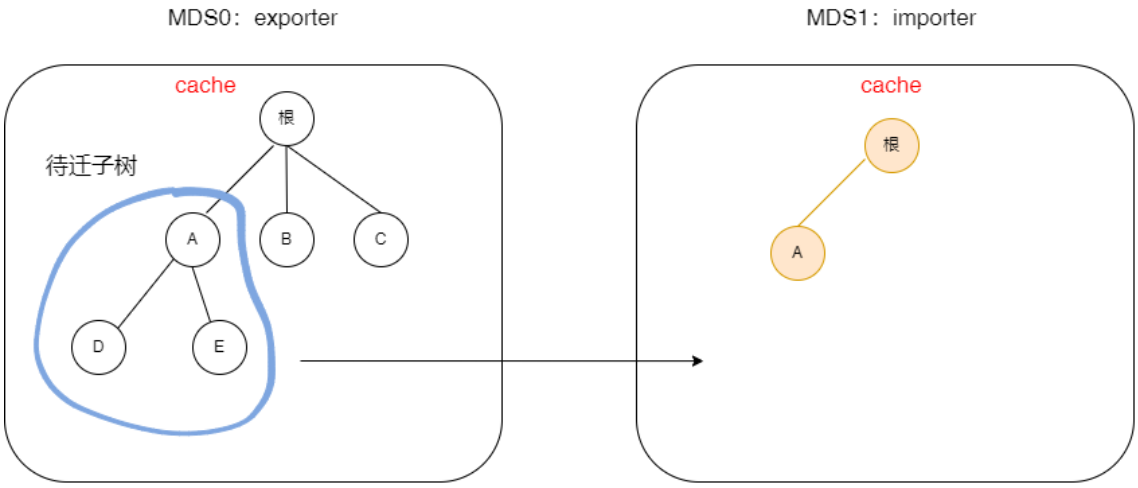
整个迁移过程分为4个阶段。例子：



discover阶段

进行一些初始化信息的交换来为正式迁移做准备。

exporter通知importer即将发动迁移，discover importer的cache中是否已存在当前迁移子树根（A目录）的inode和dentry信息。如果没有，则将exporter中子树根的信息发送过去。如下所示



exporter: dispatch_export_dir

加锁：子树根的Inode的filelock和nestlock加wrlock

发送MSG_MDS_EXPORTDIRDISCOVER消息给importer

等待frozen

importer：handle_export_discover

此时没有子树根的缓存，发送MSG_MDS_DISCOVER获取子树根的缓存。

exporter：handle_discover

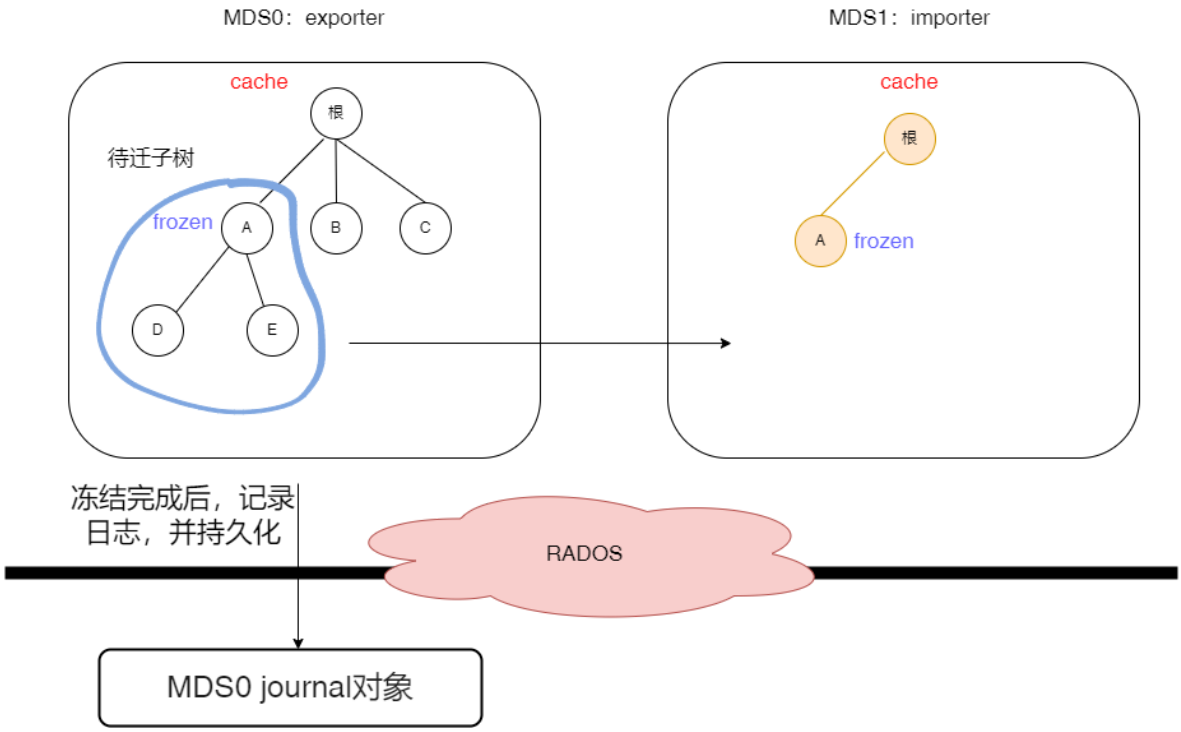
封装元数据信息(CDir的信息，主要是分片信息；CDentry信息，包括锁状态；CInode的信息，包括锁状态)，发送过去

importer：handle_discover_reply

填充缓存

prepare阶段

exporter和importer冻结子树。exporter冻结完成后，将此次事件记录在日志中，并持久化在后端日志对象中。



exporter：handle_export_discover_ack

freeze_dir: 设置STATE_FROZENDIR标志位，发送MSG_MDS_EXPORTDIRPREP消息给importer，如果有bounds，即不迁移整颗子树（子树之下有其他的分区）

importer：handle_export_prep

解析bound信息

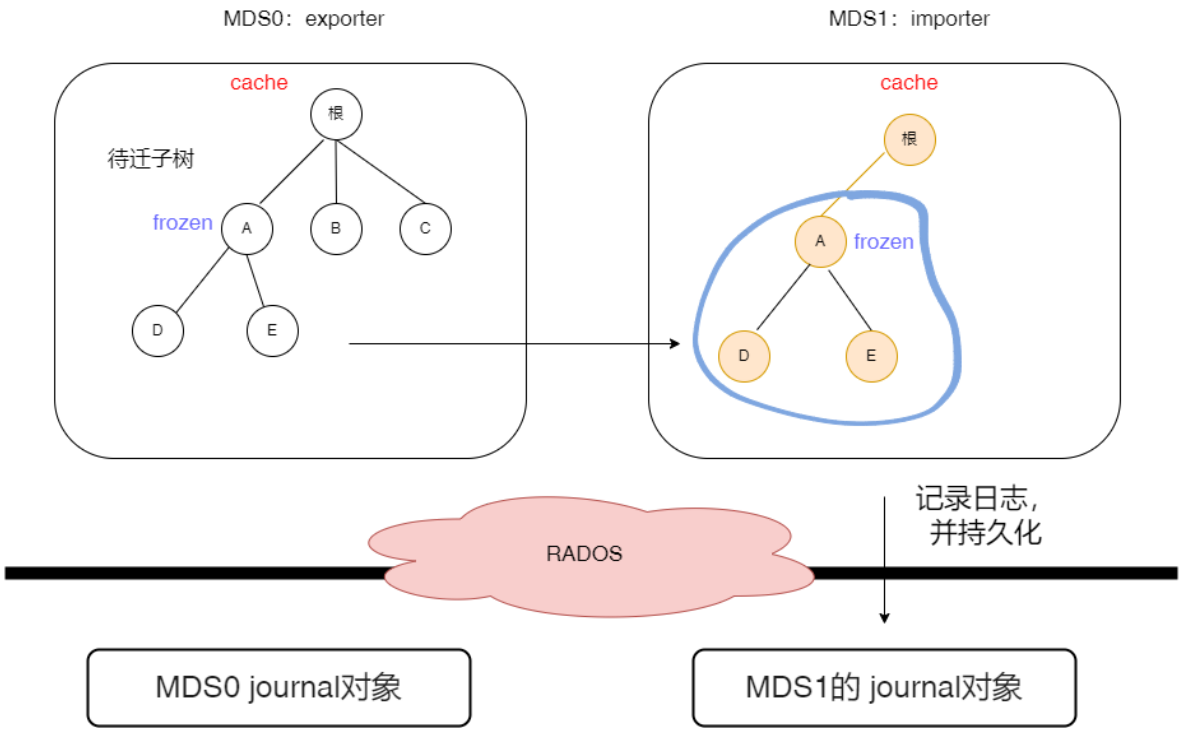
子树根的Inode的filelock和nestlock

冻结子树

发送MSG_MDS_EXPORTDIRPREPACK给exporter

export阶段

exporter节点将迁移子树所有的状态和元数据发送给importer，importer接收后将这些信息放入缓存，写入日志并持久化。



exporter：handle_export_prep_ack

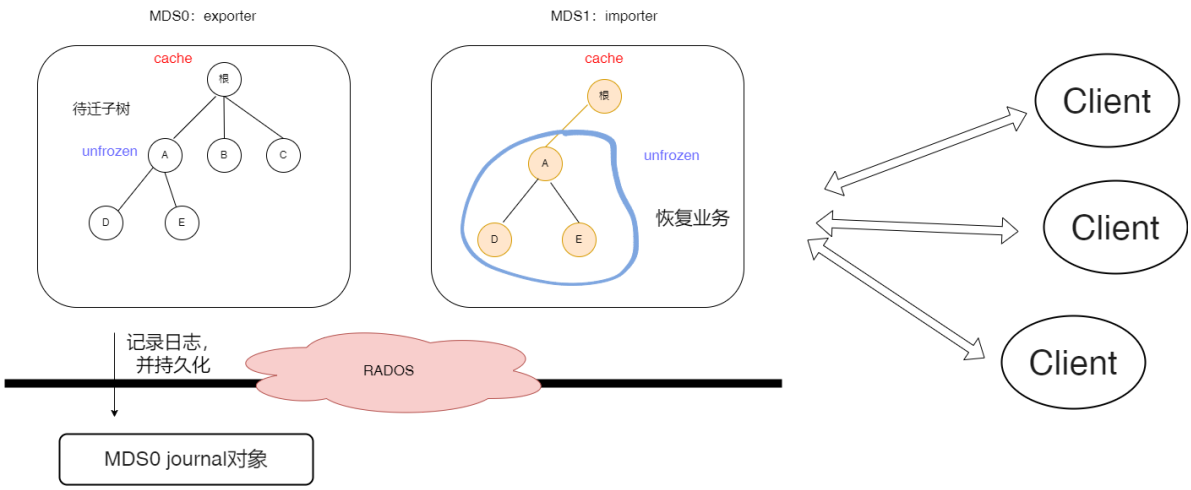
export_go，下刷所有日志之后，递归嵌套子树，将子树的所有元数据编码，包括锁的状态，客户端的caps。发送MSG_MDS_EXPORTDIR请求

importer：handle_export_dir

填充缓存，下刷日志，回复MSG_MDS_EXPORTDIRACK

finish阶段

exporter记录日志，并持久化后解冻子树，并通知importer解冻子树。importer接收后，将迁移成功事件写入日志，清理相关状态，解冻子树。



exporter：handle_export_ack

记录日志，并持久化，发送MSG_MDS_EXPORTDIRFINISH，释放锁

importer：handle_export_finish

恢复caps，unfreeze。

锁的迁移

锁迁移的内容

exporter迁移的子树的元数据包含了子树frozen之前锁的状态。

每个CInode中有10种类型的锁，如下

```
LocalLock versionlock; // 本地锁，锁的状态机在sm_locallock中定义
SimpleLock authlock; // 锁的状态机在sm_simplelock中定义
SimpleLock linklock;
SimpleLock xattrlock;
SimpleLock snaplock;
SimpleLock flocklock;
SimpleLock policylock;

ScatterLock dirfragtreelock; // 锁的状态机在sm_scatterlock中定义

ScatterLock nestlock; // 锁的状态机在sm_filelock
ScatterLock filelock;
```

在进行锁的迁移时，只需要在req中对锁的state字段编码

```
__s16 state;   // lock state
```

state的所有定义如下

```
// -- lock states --
enum {
LOCK_UNDEF = 0,
LOCK_SYNC,
LOCK_LOCK,
LOCK_PREXLOCK,
LOCK_XLOCK,
LOCK_SYNC_LOCK,
......
LOCK_XSYN,
......
LOCK_MAX,
......
};
```

每一个state分别代表了不同的读写控制状态，状态的定义为

```
struct sm_state_t {
int next; // 0 if stable
char loner;
int replica_state;
char can_read;
char can_read_projected;
char can_rdlock;
char can_wrlock;
char can_force_wrlock;
char can_lease;
char can_xlock;
int caps;
int loner_caps;
int xlocker_caps;
int replica_caps;
};
```

ceph中定义的状态机，简单示例如下

```
const struct sm_state_t filelock[LOCK_MAX] = {
// stable loner rep state r rp rd wr fwr l x caps(any,loner,xlocker,replica)
[LOCK_SYNC] = { 0, false, LOCK_SYNC, ANY, 0, ANY, 0, 0, ANY, 0, CEPH_CAP_GSHARED |
......
......
};
```

caps

在迁移过程中，会迁移当前MDS已分配给客户端的caps。

caps描述了文件系统客户端操作元数据/数据的能力，如是否可以缓存，是否可以读写。当文件系统客户端需要去操作文件/目录时，首先需要获取对应的caps。

迁移的caps内容如下：

```
struct Export {
int64_t cap_id;
int32_t wanted; // 客户端想要的权限
int32_t issued; // MDS已经分配的权限
int32_t pending; // MDS将要分配的权限
......
}
```

同步客户端的cap

1. importer端：与文件系统客户端建立session，并建立caps联系
2. exporter端：清理与文件系统客户端的session，并让客户端清理与exporter建立的caps。

锁记录的是自己节点对缓存中的CInode的读写控制，迁移之后，不需要做其他操作，当子树解冻之后，就可以继续对外提供业务。

多阶段的事务故障恢复

子树迁移中的日志

在子树迁移过程中，写了4次日志，

1. C_M_ExportGo日志，exporter为了确保将之前的未落盘的日志落盘，而注册的一个日志回调事件。该日志事件并未记录任何数据，并没有日志实体。
2. EImportStart日志，importer记录所有的元数据并落盘

EImportStart结构如下：

```
class EImportStart : public LogEvent {
protected:
dirfrag_t base; // 迁入的子树根
vector<dirfrag_t> bounds; // 最简单的场景下，即迁入整颗子树，此处为空
mds_rank_t from; // exporter mds
……
}
```

故障后，日志回放过程中，将日志中的元数据填充缓存，然后记录模糊import：

`my_ambiguous_imports[base] = bounds`

继而设置dir_auth

`dir->set_dir_auth(auth); // 设置dir_auth为<1, 1>, 即模糊子树`

3. EExport日志，exporter记录子树到根的元数据并落盘。

EExport结构如下：

```
class EExport : public LogEvent {
protected:
dirfrag_t base; // 迁出的子树根
set<dirfrag_t> bounds;
mds_rank_t target;
……
}
```

故障后，日志回放过程中，将日志中的元数据填充缓存，记录没有定义的子树

`mds->mdcache->adjust_bounded_subtree_auth(dir, realbounds, CDIR_AUTH_UNDEF)  // 即缓存中的dir是未定义的auth`

4. EImportFinish日志，importer仅仅记录import结束事件

EImportFinish结构如下

```
class EImportFinish : public LogEvent {
protected:
dirfrag_t base; // imported dir
bool success; //
......
}
```

故障后，日志回放过程中，解决EImportStart日志中的模糊import事件

`mds->mdcache->finish_ambiguous_import(base); //将dir_auth修改为<1,-2>`

迁移故障恢复

MDSRankDispatcher::handle_mds_map

MDSRank::resolve_start()

MDCache::resolve_start

mdcache->send_resolves();

send_slave_resolves();

send_subtree_resolves(); 发送MMDSResolve消息，类型为MSG_MDS_RESOLVE

MMDSResolve消息内容如下

```
class MMDSResolve : public Message {
public:
map<dirfrag_t, vector<dirfrag_t> > subtrees; // 自己的子树
map<dirfrag_t, vector<dirfrag_t> > ambiguous_imports; // 模糊import
......
}
```

故障的MDS往集群发送MMDSResolve消息，解决模糊子树的问题

1. 根据消息中的subtrees来解决自身的子树权威问题。

附录

Ceph锁的粒度控制

MDS锁类型	inode中的字段	补充
authlock	mode	
	uid	
	gid	
	ctime	基本所有锁都可以控制
	version	基本所有锁都可以控制
linklock	nlink	
dirfragtreelock	dirfragtree	分片
filelock	mtime	
	atime	
	size	
	layout，truncate_seq，truncate_size，client_ranges，inline_data	

	dirstat	struct frag_info_t
		{
		// this frag
		utime_t mtime;
		int64_t nfiles = 0;
		int64_t nsubdirs = 0;
		}
		记录该分片下的文件/目录数和mtime

	dirfrags	filelock和nestlock都可以控制

nestlock	rstat	struct nest_info_t
		{
		// this frag + children
		utime_t rctime;
		int64_t rbytes = 0;
		int64_t rfiles = 0;
		int64_t rsubdirs = 0;
		}
		记录该分片以及嵌套子树的统计信息。

xattr	xattrs	