Lustre 文件系统操作手册

译者: 李希

February 17, 2023

目录

第一章理解 Lustre 的架构 **31**

1.1 Lustre 文件系统是什么（又不是什么） 31

1.1.1. Lustre 的功能 32

1.2. Lustre 组件 35

1.2.1. 管理服务器 (MGS) 35

1.2.2 Lustre 文件系统的组件 35

1.2.3 Lustre 网络 (LNet) 37

1.2.4 Lustre 集群 37

1.3. Lustre 文件系统存储与 I/O 38

1.3.1. Lustre 文件系统条带化 40

第二章理解 Lustre 网络（LNet） **41**

2.1. LNet 简介 41

2.2. LNet 的主要功能 42

2.3.Lustre 网络 42

2.4. 支持的网络类型 42

第三章 Lustre 文件系统的故障切换 **43**

3.1. 什么是故障切换 43

3.1.1 故障切换的功能 43

3.1.2 故障切换配置类型 44

3.2. Lustre 文件系统中的故障切换功能 44

3.2.1 MDT 故障切换配置（主动/被动） 45

3.2.2 MDT 故障切换配置（主动/主动） 46

3.2.3 OST 故障切换配置（主动/主动） 47

第四章安装概述 **47**

4.1. 安装 Lustre 软件的步骤 47

第五章 Lustre 硬件配置要求和格式化选项 48

5.1. 硬件方面的考虑 48

5.1.1. 对 MGT 和 MDT 存储硬件的考量因素 49

5.1.2. 对 OST 存储硬件的考量因素 50

5.2. 确定空间需求 51

5.2.1. 确定 MGT 的空间需求 52

5.2.2. 确定 MDT 的空间需求 52

5.2.3. 确定 OST 的空间需求 53

5.3. 设置 ldiskfs 文件系统的格式化选项 53

5.3.1. 为基于 ldiskfs 的 MDT 设置格式化选项 54

5.3.2. 为基于 ldiskfs 的 OST 设置格式化选项 55

5.4. 文件和文件系统的极限值 56

5.5. 确定内存需求 59

5.5.1. 客户端内存需求 59

5.5.2. MDS 内存需求 59

5.5.3. OSS 内存需求 60

5.6. Lustre 文件系统网络的实施 61

第六章配置 Lustre 文件系统上的存储 63

6.1. 为 MDT 和 OST 选择存储设备 63

6.1.1. 元数据目标（MDT） 63

6.1.2. 对象存储服务器（OST） 63

6.2. 可靠性的最佳实践 64

6.3. 性能折衷 64

6.4. 为基于 ldiskfs 的 RAID 设备设置格式化选项 64

6.4.1. 为 mkfs 计算文件系统参数 65

6.4.2. 外部日志的参数设置 65

6.5. 将 SAN 连接至 Lustre 文件系统上 66

第七章网络端口绑定的设置 **66**

7.1. 绑定网络端口的概述 67

7.2. 相关要求 67

7.3. 网口绑定的模块参数 69

7.4. 设置绑定 69

7.4.1. 示例 71

7.5. 为绑定配置 Lustre 文件系统 73

7.6. 关于绑定的参考资料 73

第八章 Lustre 软件系统安装 **73**

8.1. 准备安装 Lustre 软件 73

8.1.1. 需要的软件 73

8.1.2. 环境要求 75

8.2. Lustre 软件安装流程 76

第九章配置 Lustre 网络（LNet） **77**

9.1. 通过`lnetctl`配置 LNet 77

9.1.1. 配置 LNet 78

9.1.2. 显示全局设置 78

9.1.3. 添加、删除、显示网络 78

9.1.4. 手动添加、删除、显示端点 80

9.1.5. 端点的动态发现（Lustre 2.11 中引入） 82

9.1.6. 添加、删除和显示路由 83

9.1.7. 启用和禁用路由 85

9.1.8. 显示路由信息 85

9.1.9. 配置路由缓冲 86

9.1.10. 非对称路由（Lustre 2.13 引入） 87

9.1.11. 引入 YAML 配置文件 87

9.1.12. 导出 YAML 配置文件 88
9.1.13. 显示 LNet 流量数据信息 88
9.1.14. YAML 的语法 88
9.2. LNet 模块参数概述 90
9.2.1. 使用 Lustre 网络标识符（NID）来标识节点 91
9.3. 设置 LNet 模块的networks参数 91
9.3.1. 多地址服务器的例子 92
9.4. 设置 LNet 模块的ip2nets参数 93
9.5. 设置 LNet 模块的routes参数 94
9.5.1. 路由配置示例 95
9.6. 测试 LNet 配置 95
9.7. 配置路由器检查器 95
9.8. LNet 选项的最佳实践 97
9.8.1. 用引号逗号 97
9.8.2. 增加注释 97

第十章 Lustre 文件系统配置 98

10.1. 配置简单的 Lustre 文件系统 98
10.1.1. 简单 Lustre 配置示例 101
10.2. 其他附加配置选项 107
10.2.1. 扩展 Lustre 文件系统 107
10.2.2. 更改条带化默认配置 107
10.2.3. 使用 Lustre 配置实用程序 108

第十一章 Lustre 故障切换配置 108

11.1. 故障切换环境设置 108
11.1.1 选择电源设备 109
11.1.2 选择电源管理软件 109

11.1.3 选择高可用性软件 109
11.2. 为故障切换而准备 Lustre 文件系统 110

第十二章 Lustre 文件系统监控配置 111

12.1. Lustre Changelogs 111
12.1.1 Changelogs 相关命令 112
12.1.2 Changelogs 命令示例 113
12.1.3 Changelogs 审计 116
12.1.3.1 启用审计功能 116
12.1.3.2 审计功能示例 117
12.2. Lustre Jobstats 118
12.2.1 Jobstats 如何工作 118
12.2.2 启用/禁用 Jobstats 119
12.2.3 查看 Jobstats 121
12.2.4 清除 Jobstats 123
12.2.5 配置自动清理 (Auto-cleanup) 时间间隔 123
12.3. Lustre 监控工具（LMT） 124
12.4. CollectL 124
12.5. 其他监控选项 124

第十三章 Lustre 操作详解 125

13.1. 通过标签挂载 125
13.2. 启动 Lustre 125
13.3. 挂载服务器 126
13.4. 关闭文件系统 126
13.5. 在服务器上卸载目标 128
13.6. 为 OSTs 指定故障切换模式 128
13.7. 处置降级 OST 磁盘阵列 129

13.8. 运行多个 Lustre 文件系统 129
13.9. 在特定 MDT 上创建子目录 131
13.10. 在多个 MDTs 上创建条带目录 132
13.10.1. 通过 space/inode 创建目录 132
13.11. 设置及查看 Lustre 参数 133
13.11.1. 用mkfs.lustre设置可调试参数 133
13.11.2. 用tunefs.lustre设置参数 133
13.11.3. 用 lctl设置参数 133
13.12. 指定 NIDs 和故障切换 136
13.13. 擦除文件系统 137
13.14. 回收预留磁盘空间 138
13.15. 替换当前 OST 或 MDT 138
13.16. 识别 OST 对象隶属于哪个 Lustre 文件 138

第十四章 Lustre 的日常维护 **140**

14.1. 非活动 OSTs 相关操作 140
14.2. 查看 Lustre 文件系统所有节点 140
14.3. 在无 Lustre 服务的情况下挂载服务器 141
14.4. 重新生成 Lustre 配置日志 141
14.5. 更改服务器 NID 143
14.6. 清除配置 144
14.7. 在 Lustre 文件系统中加入新的 MDT 144
14.8. 在 Lustre 文件系统中添加新的 OST 145
14.9. 移除及恢复 MDT 和 OST 146
14.9.1. 在文件系统中移除 MDT 147
14.9.2. 不活跃的 MDTs 147
14.9.3. 在文件系统中移除 OST 147
14.9.4. 备份 OST 配置文件 149

14.9.5. 恢复 OST 配置文件 150
14.9.6. 重新激活 OST 151
14.10. 终止恢复 151
14.11. 确定服务 OST 的机器 151
14.12. 更改故障节点地址 152
14.13. 分离组合的 MGS/MDT 152
14.14. 将 MDT 设置为只读 153
14.15. 调整 ldiskfs 的 Fallocate 154

第十五章管理 Lustre Networking (LNet) 154

15.1. 更新路由或端的健康状态 154
15.2. 启动和关闭 LNet 155
15.2.1. 启动 LNet 155
15.2.2. 关闭 LNet 155
15.3. 基于 LNet 多轨配置的硬件 156
15.4. 利用 InfiniBand* 网络实现负载平衡 156
15.4.1. 在`lustre.conf`中配置负载均衡 156
15.5. 动态配置 LNet 路由 159
15.5.1. `lustre_routes_config` 159
15.5.2. `lustre_routes_conversion` 159
15.5.3. 路由配置示例 160

第十六章 LNet 软件多轨 160

16.1. 概述 160
16.2. 多轨的配置 160
16.2.1. 在本地节点上配置多个接口 160
16.2.2. 删除网络接口 162
16.2.3. 增加具有多轨功能的远程对等节点 163

16.2.4. 删除远程对等节点 164
16.3. 多轨路由注意事项 165
16.3.1. 多轨集群示例 165
16.3.2. 路由器可恢复功能 167
16.3.3. 多轨及非多轨混合集群 168
16.4. LNet Health 下的多轨路由 168
16.4.1. 配置 168
16.4.2. 路由健康 169
16.4.3. 发现 169
16.4.4. 路由有效性标准 170
16.5. LNet Health 170
16.5.1. 健康值 170
16.5.2. 故障类型和行为 170
16.5.3. 用户接口 171
16.5.5. 推荐的初始设置 175

第十七章升级 Lustre 文件系统 175

17.1. 互操作性和升级要求 175
17.2. 升级至 Lustre Software Release 2.x (主版本) 176
17.3. 升级至 Lustre Software Release 2.x.y (次版本) 179

第十八章备份和恢复文件系统 181

18.1. 备份文件系统 181
18.1.1. Lustre_rsync 182
18.2. 备份和恢复 MDT 或 OST （ldiskfs 设备级） 184
18.3. 备份 OST 或 MDT （后端文件系统级） 185
18.3.1. 备份 OST 或 MDT（后端文件系统级） 186
18.3.2. 备份 OST 或 MDT 186

18.4. 恢复文件级备份 188
18.5. 使用 LVM 快照 190
18.5.1. 创建基于 LVM 的备份文件系统 191
18.5.2. 备份新的/更改后的文件 192
18.5.3. 创建快照卷 193
18.5.4. 从快照恢复文件系统 193
18.5.5. 删除旧的快照 195
18.5.6. 更改快照卷大小 196
18.6. ZFS 和 ldiskfs 目标文件系统间的迁移 196
18.6.1. 从 ZFS 迁移至 ldiskfs 文件系统 196
18.6.2. 从 ldiskfs 迁移至 ZFS 文件系统 196

第十九章管理文件布局（条带化）及剩余空间 196

19.1. Lustre 文件系统条带化如何工作 196
19.2. Lustre 文件布局（条带化）的一些考量 197
19.2.1. 选择条带大小 198
19.3. 配置 Lustre 文件布局（条带化模式）(`lfs setstripe`) 199
19.3.1. 为单个文件指定文件布局（条带化模式） 199
19.3.2. 为目录指定文件布局（条带化模式） 200
19.3.3. 为文件系统指定文件布局（条带化模式） 201
19.3.4. 在指定 OST 上创建文件 201
19.4. 检索文件布局/条带信息 (`getstripe`) 201
19.4.1. 显示当前条带大小 202
19.4.2. 搜索文件树 202
19.4.3. 为远程目录定位 MDT 202
19.5. 渐进式文件布局 (PFL) 203
19.5.1. `lfs setstripe` 204
19.5.2. `lfs migrate` 214

19.5.3. lfs getstripe 219
19.5.4. lfs find 225
19.6. 自扩展布局 226
19.6.1. lfs setstripe 227
19.6.2. lfs getstripe 230
19.6.3. lfs find 238
19.7. 外部布局 239
19.7.1. lfs set[dir]stripe 239
19.7.2. lfs get[dir]stripe 240
19.7.3. lfs find 240
19.8. 管理空闲空间 241
19.8.1. 查看文件系统可用空间 241
19.8.2. 条带分配方法 244
19.8.3. 调整可用空间和位置的权重 245
19.9. Lustre 条带化内部参数 245

第二十章在 MDT 上存储数据的功能 (DoM) 245

20.1. 简介 245
20.2. 用户命令 246
20.2.1. lfs setstripe 246
20.2.2. 为现有目录设置 DoM 布局 249
20.2.3.DoM 条带大小限制 251
20.2.4. lfs getstripe 251
20.2.5. lfs find 252
20.2.6. dom_stripesize 参数 253
20.2.7. 禁用 DoM 254

第二十一章 MDT 的 Lazy 大小功能 (LSoM) 255

21.1. 简介 255
21.2. 启动 LSoM 255
21.3. 用户命令 255
21.3.1 使用`lfs getsom`显示 LSoM 数据 256
21.3.2 lfs getsom 命令 256

第二十二章文件级冗余 (FLR) 256

22.1. 概述 256
22.2. 相关操作 257
22.2.1. 创建镜像文件或目录 257
22.2.2. 扩展镜像文件 263
22.2.3. 拆分镜像文件 268
22.2.4. 重新同步待同步镜像文件 275
22.2.5. 验证镜像文件 280
22.2.6. 查找镜像文件 284
22.3. 互操作性 284

第二十三章管理文件系统和 I/O 285

23.1. 处理满溢的 OSTs 285
23.1.1. 查看 OST 空间使用情况 285
23.1.2. 在满溢的 OST 上禁用创建功能 286
23.1.3. 在文件系统内迁移数据 287
23.1.4. 将禁用的 OST 重新上线 287
23.1.5. 在文件系统内迁移元数据 287
23.2. 创建和管理 OST 池 289
23.2.1. OST 池操作 289
23.2.2. OST 池使用建议 291
23.3. 在 Lustre 文件系统中添加 OST 292

23.4. 实施直接 I/O 292
23.4.1. 将文件系统对象设置为不可变 293
23.5. 其它 I/O 选项 293
23.5.1. Lustre 校验和 293
23.5.2. Ptlrpc 客户端线程池 294

第二十四章 Lustre 文件统故障切换和多挂载保护 295

24.1. 概览 295
24.2. 多挂载保护相关操作 295

第二十五章配额配置和管理 296

25.1. 配额相关操作 296
25.2. 启用磁盘配额 297
25.2.1. 配额验证 299
25.3. 配额管理 299
25.4 默认配额 302
25.4.1 用法 302
25.5. 配额分配 303
25.6. 配额和版本互操作性 304
25.7 在. 授权缓存和配额限制 305
25.8. Lustre 配额统计信息 305
25.8.1. 解析配额统计信息 307
25.9 池配额 307
25.9.1. DOM 池与 MDT 池 308
25.9.2. 用于设置配额池的 Lfs quota/setquota 选项 308
25.9.3. 配额池的交互性 308
25.9.4. 池配额设置硬限制示例 308
25.9.5. 池配额设置软限制示例 308

第二十六章分层存储管理 (HSM) **309**

26.1. 简介 . . . 309

26.2. 设置 . . . 310

26.2.1. 要求 . . . 310

26.2.2. 协调器（coordinator） . . . 310

26.2.3. 代理（agent） . . . 310

26.3. 代理（Agents）和复制工具（copytool） . . . 310

26.3.1. ARCHIVE ID 及多后端系统 . . . 311

26.3.2. 注册代理 . . . 311

26.3.3. 超时 . . . 312

26.4. 请求 . . . 312

26.4.1. 命令 . . . 312

26.4.2. 自动恢复 . . . 312

26.4.3. 请求监控 . . . 313

26.5. 文件状态 . . . 313

26.6. 调试 . . . 313

26.6.1. `hsm_controlpolicy` . . . 313

26.6.2. `max_requests` . . . 314

26.6.3. `policy` . . . 314

26.6.4. `grace_delay` . . . 314

26.7. 变更日志 . . . 314

26.8. 策略引擎 . . . 315

26.8.1. Robinhood . . . 315

第二十七章持久客户端缓存（PCC） **315**

27.1. 简介 . . . 315

27.2. 设计 . . . 316

27.2.1 Lustre 读 -写 PCC 缓存 . . . 316

27.2.2. 基于规则的持久性客户端缓存 317
27.3. PCC 命令行工具 317
27.3.1. 在客户端添加 PCC 后端 317
27.3.2. 从客户端删除一个 PCC 后端 319
27.3.3. 移除客户端上的所有 PCC 后端 319
27.3.4. 列出客户端上的所有 PCC 后端 319
27.3.5. 将指定文件附加到 PCC 上 320
27.3.6. 通过 FID 将指定文件附加到 PCC 上 320
27.3.7. 将指定文件从 PCC 分离 320
27.3.8. 通过 FID 从 PCC 分离指定文件 321
27.3.9. 显示指定文件的 PCC 状态 321
27.4. PCC 配置示例 321

第二十八章使用 Nodemap 映射 UIDs 和 GIDs 322

28.1. 设置映射 322
28.1.1. 定义 322
28.1.2. NID 范围 323
28.1.3. 示例：描述和部署映射 323
28.2. 属性变更 325
28.2.1. 管理属性 325
28.2.2. 混合属性 326
28.3. 启用 nodemap 326
28.4. `default` Nodemap 327
28.5. 校验设置 327
28.6. 确保一致性 328

第二十九章配置共享密钥 (SSK) 329

29.1. SSK 安全概述 329

29.1.1. 关键功能 . . . 329
29.2. SSK 安全特性 . . . 329
29.2.1. RPC 安全规则 . . . 330
29.3. SSK 密钥文件 . . . 332
29.3.1. 密钥文件管理 . . . 333
29.4. Lustre GSS 密钥环 . . . 336
29.4.1. 设置 . . . 336
29.4.2. 服务器设置 . . . 337
29.4.3. 调试 GSS 密钥环 . . . 339
29.4.4. 撤销密钥 . . . 340
29.5. Nodemap 在 SSK 中的作用 . . . 340
29.6. SSK 示例 . . . 341
29.6.1. 客户端到服务器的安全通信 . . . 341
29.6.2. MGS 安全通信 . . . 343
29.6.3. 服务器之间的安全通信 . . . 344
29.7. 查看 PtlRPC 安全环境 . . . 345

第三十章 Lustre 文件系统安全管理 347

30.1. 使用访问控制列表（ACL） . . . 347
30.1.1. ACL 如何工作 . . . 347
30.1.2. Lustre 软件上的 ACLs . . . 347
30.1.3. 示例 . . . 348
30.2. 使用 Root Squash . . . 349
30.2.1. 配置 Root Squash . . . 349
30.2.2. 启用和调试 Root Squash . . . 349
30.2.3. 使用 Root Squash 的技巧 . . . 351
30.3. 隔离客户端到子目录树上 . . . 352
30.3.1. 指定客户端 . . . 352

30.3.2. 配置 Isolation 352
30.3.3. 将 Isolation 持久化 353
30.4. 检查 Lustre 客户端执行的 SELinux 策略 353
30.4.1. 确定 SELinux 策略信息 354
30.4.2. 执行 SELinux 策略检查 354
30.4.3. 持久化 SELinux 策略检查 355
30.4.4. 客户端发送 SELinux 状态信息 355
30.5 加密文件和目录 355
30.5.1. 客户端加密访问语义 356
30.5.2. 客户端加密密钥的层次结构 357
30.5.3. 客户端加密模式和用法 357
30.5.4. 客户端加密威胁模式 357
30.6 配置 Kerberos（KRB）安全性 361
30.6.1. 什么是 Kerberos? 361
30.6.2. 安全类型（Security Flavor） 362
30.6.3. Kerberos 设置 362
30.6.4. 网络 364
30.6.5. 必要的软件包 364
30.6.6. 构建 Lustre 365
30.6.7. 运行 365
30.6.8. 安全的 MGS 连接 367

第三十一章 Lustre ZFS 快照 **367**

31.1. 概述 367
31.1.1. 需求 368
31.2. 配置 368
31.3. 快照操作 368
31.3.1. 创建快照 368

31.3.2. 删除快照 . . . 369
31.3.3. 挂载快照 . . . 369
31.3.4. 卸载快照 . . . 370
31.3.5. 列出快照 . . . 371
31.3.6. 修改快照属性 . . . 371
31.4. 全局写屏障 . . . 372
31.4.1. 添加屏障 . . . 372
31.4.2. 移除屏障 . . . 372
31.4.3. 查询屏障 . . . 373
31.4.4. 重新扫描屏障 . . . 374
31.5. 快照日志 . . . 374
31.6. Lustre 配置日志 . . . 374

第三十二章 Lustre 网络性能测试（LNet Self-Test） 375

32.1. LNet 自检概述 . . . 375
32.1.1. 前提条件 . . . 376
32.2. LNet 自检操作 . . . 376
32.2.1. 创建会话 . . . 376
32.2.2. 设置组 . . . 377
32.2.3. 定义及运行测试 . . . 377
32.2.4. 脚本样例 . . . 378
32.3. LNet 自检命令索引 . . . 379
32.3.1. 会话命令 . . . 379
32.3.2. 组命令 . . . 380
32.3.3. 批处理测试命令 . . . 383
32.3.4. 其他命令 . . . 387

第三十三章对 Lustre 文件系统进行基准测试 (LustreI/O 工具箱) 389

33.1. 使用 Lustre I/O 工具箱 389
33.1.1. Lustre I/O 工具箱内容 390
33.1.2. Lustre I/O 工具箱使用准备 390
33.2. 测试原始硬件 I/O 性能 (sgpdd-survey) 390
33.2.1. 调试 Linux 存储设备 391
33.2.2. 运行sgpdd-survey 392
33.3. OST 性能测试 (obdfilter-survey) 393
33.3.1. 本地磁盘性能测试 394
33.3.2. 网络性能测试 396
33.3.3. 远程磁盘性能测试 397
33.3.4. 输出文件 398
33.4. OST I/O 性能测试 (ost-survey) 399
33.5. MDS 性能测试 (mds-survey) 400
33.5.1. 输出文件 401
33.5.2. 脚本输出 402
33.6. 收集应用程序分析信息 (stats-collect) 403
33.6.1. stats-collect 403

第三十四章 Lustre 文件系统调试 404

34.1. 优化服务线程数量 404
34.1.1. 指定 OSS 服务线程数 405
34.1.2. 指定 MDS 服务线程数 405
34.2. 绑定 MDS 服务线程到 CPU 分区 406
34.3. LNet 参数调试 406
34.3.1. 发送和接收缓冲区大小 406
34.3.2. 硬件中断 (enable_irq_affinity) 406
34.3.3. 绑定针对 CPU 分区的网络接口 407
34.3.4. 网络接口信用 407

34.3.5. 路由器缓存区 408
34.3.6. 门户循环 408
34.3.7. LNet 对等节点健康状况 409
34.4. libcfs 调试 410
34.4.1. CPU 分区（字符串模式） 410
34.5. LND 调试 411
34.5.1. ko2iblnd 调试 411
34.6. 网络请求调度程序 (NRS) 调试 413
34.6.1. 先进先出 (FIFO) 策略 417
34.6.2. 基于 NID 的客户端循环 (CRR-N) 策略 417
34.6.3. 基于对象的循环（ORR）策略 418
34.6.4. 基于目标的循环 (TRR) 策略 421
34.6.5. 令牌桶过滤器 (TBF) 策略 421
34.6.6. 延迟策略 429
34.7. 无锁 I/O 可调参数 432
34.8. 服务器端建议和提示 433
34.8.1. 概述 433
34.8.2. 示例 434
34.9. 大批量 I/O (16MB RPC) 434
34.9.1. 概述 434
34.9.2. 示例 435
34.10. 提升 Lustre 小文件 I/O 性能 435
34.11. 写入性能与读取性能 436

第三十五章 Lustre 文件系统故障排除 436

35.1. Lustre 错误消息 436
35.1.1. 错误代码 436
35.1.2. 查看错误消息 437

35.2. 报告 Lustre 文件系统 Bug 438
35.2.1. 在 Jira* Tracker 中搜索重复故障单 438
35.3. Lustre 文件系统常见问题 439
35.3.1. OST 对象缺失或损坏 439
35.3.2. OSTs 变为只读 440
35.3.3. 识别丢失的 OST 440
35.3.4. 修复 OST 上错误的 LAST_ID 441
35.3.5. 处理"Bind: Address already in use" 错误 442
35.3.6. 处理错误"- 28" 442
35.3.7. 触发 PID NNN 看门狗定时器 444
35.3.8. 处理初始 Lustre 文件系统设置的超时 445
35.3.9. 处理"LustreError: xxx went back in time" 错误 445
35.3.10. Lustre 错误: "Slow Start_Page_Write" 446
35.3.11. 多客户端 O_APPEND 写入的劣势 446
35.3.12. Lustre 文件系统启动时的减速 446
35.3.13. OST 上的日志信息"Out of Memory" 447
35.3.14. 设置 SCSI I/O 大小 447

第三十六章故障恢复 **447**

36.1. 在备份 ldiskfs 文件系统上恢复错误或损坏 447
36.2. 在 Lustre 文件系统上恢复损坏 448
36.2.1. 处理孤立对象 449
36.3. 从不可用的 OST 中恢复 449
36.4. 使用 LFSCK 检查文件系统 449
36.4.1. LFSCK switch 接口 450
36.4.2. 查看 LFSCK 全局状态 453
36.4.3. LFSCK status 接口 453
36.4.4. LFSCK adjustment 接口 460

第三十七章 Lustre 文件系统调试 **461**

37.1. 诊断调试工具 461

37.1.1. Lustre 调试工具 462

37.1.2. 扩展调试工具 462

37.2. Lustre 调试过程 464

37.2.1. 了解 Lustre 调试消息格式 464

37.2.2. 使用 lctl 工具查看调试信息 466

37.2.3. 将缓冲区内容转储到文件 (debug_daemon) 467

37.2.4. 写入内核调试日志的控制信息 469

37.2.5. 使用 strace进行故障排除 469

37.2.6. 查看磁盘内容 470

37.2.7. 查找 OST 的 Lustre UUID 471

37.2.8. 打印调试消息至控制台 471

37.2.9. 锁流量跟踪 472

37.2.10. 控制台消息速率限制 472

37.3. Lustre 开发调试 472

37.3.1. 在 Lustre 源代码中添加调试功能 472

37.3.2. 访问ptlrpc请求历史 475

37.3.3. 使用 leak_finder.pl查找内存泄漏 477

第三十八章 Lustre 文件系统恢复 **477**

38.1. 概述 477

38.1.1. 客户端故障 478

38.1.2. 客户端驱逐 478

38.1.3. MDS 故障（切换） 478

38.1.4. OST 故障（切换） 479

38.1.5. 网络分区 480

38.1.6. 恢复失败 480

38.2. 元数据重放 480
38.2.1. XID 编号 481
38.2.2. 事务编号 481
38.2.3. 重放和重发 481
38.2.4. 客户端重放列表 482
38.2.5. 服务器恢复 482
38.2.6. 请求重放 482
38.2.7. 重放序列中的间隙 483
38.2.8. 锁恢复 483
38.2.9. 请求重发 484
38.3. 重建回复 484
38.3.1. 所需状态 484
38.3.2. 重建" 打开请求" 的回复 484
38.3.3. 客户端上的多个回复数据 485
38.4. 基于版本的恢复 485
38.4.1. VBR 消息 486
38.4.2. VBR 使用建议 486
38.5. 共享提交 486
38.5.1. COS 的工作原理 486
38.5.2. COS 调试 487
38.6. 强制恢复 487
38.6.1. MGS 的作用 488
38.6.2. IR 调试 488
38.6.3. IR 配置建议 491
38.7. Ping 抑制 491
38.7.1. 内核模块参数"suppress_pings" 492
38.7.2. 客户端死亡通知 492

第三十九章 Lustre 参数 **492**

39.1. 简介 492

39.1.1. 识别 Lustre 文件系统和服务器 494

39.2. 多块分配的调试 (mballoc) 496

39.3. Lustre 文件系统 I/O 监控 497

39.3.1. 客户端 RPC 流监控 498

39.3.2. 客户端活动监控 500

39.3.3. 客户端读写位移统计信息监控 502

39.3.4. 客户端读写范围统计信息监控 503

39.4. Lustre 文件系统 I/O 调试 508

39.4.1. 客户端 I/O RPC 流的调试 509

39.4.2. 文件 Readahead 和目录 Statahead 的调试 510

39.4.3. 服务器读缓存的调试 512

39.4.4. 启用 OSS 异步日志提交 515

39.4.5. 客户端元数据 RPC 流的调试 516

39.5. Lustre 文件系统超时配置 517

39.5.1. 配置自适应超时 518

39.5.2. 设置静态超时 520

39.6. LNet 监控 521

39.7. 在 OST 上分配空闲空间 523

39.8. 配置锁 524

39.9. 设置 MDS 和 OSS 线程计数 525

39.10. 调试日志 526

39.10.1. 解析 OST 统计数据 528

39.10.2. MDT 统计数据解析 530

第四十章用户实用程序 **530**

40.1. `lfs` 530

40.1.1. 梗概 . . . 530
40.1.2. 说明 . . . 532
40.1.3. 选项 . . . 532
40.1.4. 示例 . . . 537
40.2. lfs_migrate . . . 539
40.2.1. 梗概 . . . 539
40.2.2. 说明 . . . 539
40.2.3. 选项 . . . 540
40.2.4. 示例 . . . 541
40.3. filefrag . . . 541
40.3.1. 梗概 . . . 541
40.3.2. 说明 . . . 541
40.3.3. 选项 . . . 542
40.3.4. 示例 . . . 542
40.4. mount . . . 543
40.5. 处理超时 . . . 544

第四十一章程序接口 **544**

41.1. 用户/组的上行调用（upcall） . . . 544
41.1.1. 梗概 . . . 544
41.1.2. 说明 . . . 544
41.1.3. 数据结构 . . . 545

第四十二章在 C 程序中设置 Lustre 属性 (llapi) **546**

42.1. llapi_file_create . . . 546
42.1.1. 梗概 . . . 546
42.1.2. 说明 . . . 546
42.1.3. 示例 . . . 547

42.2. llapi_file_get_stripe 547
42.2.1. 梗概 548
42.2.2. 说明 548
42.2.3. 返回值 549
42.2.4. 错误 549
42.2.5. 示例 550
42.3. llapi_file_open 551
42.3.1. 梗概 551
42.3.2. 说明 552
42.3.3. 返回值 552
42.3.4. 错误 552
42.3.5. 示例 553
42.4. llapi_quotactl 553
42.4.1. 梗概 553
42.4.2. 说明 554
42.4.3. 返回值 555
42.4.4. 错误 555
42.5. llapi_path2fid 556
42.5.1. 梗概 556
42.5.2. 说明 556
42.5.3. 返回值 556
42.6. llapi_ladvise 556
42.6.1. 梗概 556
42.6.2. 说明 557
42.6.3. 返回值 558
42.6.4. 错误 558
42.7. llapi 库使用示例 559

第四十三章配置文件和模块参数 **565**

43.1. 简介 . . . 565

43.2. 模块选项 . . . 565

43.2.1. LNet 选项 . . . 566

43.2.2. SOCKLND 内核 TCP/IP LND . . . 571

第四十四章系统配置工具 **573**

44.1. e2scan . . . 573

44.1.1. 梗概 . . . 573

44.1.2. 说明 . . . 573

44.1.3. 选项 . . . 573

44.2. l_getidentity . . . 574

44.2.1. 梗概 . . . 574

44.2.2. 说明 . . . 574

44.2.3. 选项 . . . 574

44.2.4. 文件 . . . 574

44.3. lctl . . . 575

44.3.1. 梗概 . . . 575

44.3.2. 说明 . . . 575

44.3.3. 使用 lctl 设置参数 . . . 575

44.3.4. 选项 . . . 580

44.3.5. 示例 . . . 580

44.4. ll_decode_filter_fid . . . 581

44.4.1. 梗概 . . . 581

44.4.2. 说明 . . . 581

44.4.3. 示例 . . . 581

44.5. ll_recover_lost_found_objs . . . 582

44.5.1. 梗概 . . . 582

44.5.2. 说明 . . . 582
44.5.3. 示例 . . . 582
44.5.4. 文件 . . . 582
44.6. llog_reader . . . 583
44.6.1. 梗概 . . . 583
44.6.2. 说明 . . . 583
44.7. llstat . . . 583
44.7.1. 梗概 . . . 583
44.7.2. 说明 . . . 583
44.7.3. 选项 . . . 584
44.7.4. 示例 . . . 585
44.7.5. 文件 . . . 585
44.8. llverdev . . . 585
44.8.1. 梗概 . . . 585
44.8.2. 说明 . . . 586
44.8.3. 选项 . . . 586
44.8.4. 示例 . . . 586
44.9. lshowmount . . . 587
44.9.1. 梗概 . . . 587
44.9.2. 说明 . . . 587
44.9.3. 选项 . . . 587
44.10.4. 文件 . . . 588
44.10. lst . . . 588
44.10.1. 梗概 . . . 588
44.10.2. 说明 . . . 588
44.10.3. 模块 . . . 588
44.10.4. 功能 . . . 589

44.10.5. 脚本示例 589
44.11. lustre_rmmod.sh 589
44.12. lustre_rsync 590
44.12.1. 梗概 590
44.12.2. 说明 590
44.12.3. 选项 590
44.12.4. 示例 591
44.13. mkfs.lustre 593
44.13.1. 梗概 593
44.13.2. 说明 593
44.13.3. 示例 595
44.14. mount.lustre 595
44.14.1. 梗概 596
44.14.2. 说明 596
44.14.3. 选项 597
44.14.4. 示例 600
44.15. plot-llstat 601
44.15.1. 梗概 601
44.15.2. 说明 601
44.15.3. 选项 601
44.15.4. 示例 601
44.16. routerstat 602
44.16.1. 梗概 602
44.16.2. 说明 602
44.16.3. 输出 602
44.16.4. 示例 603
44.16.5. 文件 603

44.17. tunefs.lustre 603
44.17.1. 梗概 603
44.17.2. 说明 603
44.17.3. 选项 604
44.17.4. 示例 604
44.18. 附加系统配置程序 605
44.18.1. 应用程序分析工具 605
44.18.2. More/proc 统计信息 605
44.18.3. 测试和调试工具 606
44.18.4. Fileset（文件集）功能 606

第四十五章 LNet 配置的 C-API **608**

45.1. API 通用信息 608
45.1.1. API 返回代码 608
45.1.2. API 普通输入参数 608
45.1.3. API 普通输出参数 608
45.2. LNet 配置 C-API 610
45.2.1. 配置 LNet 610
45.2.2. 启用/禁用路由 611
45.2.3. 添加路由 612
45.2.4. 删除路由 613
45.2.5. 显示路由 613
45.2.6. 添加网络接口 614
45.2.7. 删除网络接口 615
45.2.8. 显示网络接口 616
45.2.9. 调整路由器缓冲池 617
45.2.10. 显示路由信息 618
45.2.11. 显示 LNet 流量统计数据 620

45.2.12. 添加/删除/显示参数 621

45.2.13. 添加路由的代码示例 623

本文档译自英文版 Lustre 操作手册（http://doc.lustre.org/lustre_manual.xhtml），并按原文相同的许可证（Creative CommonsAttribution-Share Alike 3.0 United States License http://creativecommons.org/licenses/by-sa/3.0/us）免费分享。根本该许可证，任何人均可复制、修改、分发本文档，以用于包含商业目的在内的任何用途，但需要遵循相同的许可证。

我们（译者）由于时间仓促，无法保证本文档完全正确、有效。因此，您自阅读本文档时起，即隐含着与我们达成了免责协议：本文作者和译者不为文档的任何错漏负责。因此，为避免因武断操作而遭受损失，您在根据本文档执行关键操作之前，应予以充分测试验证。如果您在生产系统上遇到棘手的 Lustre 技术问题，应向相关厂商寻求专业技术支持，而不应冒然尝试本文档中介绍的操作。

本文翻译工作由 China Open File System（COFS）委员会赞助。COFS 是一个非盈利的行业组织，支持包括 Lustre 在内的开放、开源的文件系统和存储技术在中国社区的使用和推广。COFS 以服务中国用户群体为宗旨，以实际应用需求为导向，以开源项目为基础，以相关厂商为依托，组织社区活动，促进用户交流，构建活跃、进取的中国用户社区，从而更好地促进 Lustre 等开源先进技术在中国的推广和应用，进而促使开源项目为中国用户的生产活动提供更好的支持。

第一章理解 Lustre 的架构

1.1 Lustre 文件系统是什么（又不是什么）

Lustre 架构是一种为集群设计的存储架构。Lustre 架构的核心组件是 Lustre 文件系统。该文件系统运行于 Linux 操作系统之上，提供符合 POSIX 标准的 UNIX 文件系统接口。

Lustre 存储架构可用于多种类型的集群。最广为人知的是，它为众多全球最大的高性能计算（HPC）集群赋能，为这些集群提供了数以万计的客户端系统、PB 级的存储容量和数百 GB 每秒的 I/O 吞吐率。许多 HPC 站点使用 Lustre 文件系统作为全站点级的全局文件系统，为内部数十个群集提供服务。

Lustre 文件系统具备按需扩展容量和性能的能力，弱化了部署众多独立文件系统的必要（如每个计算群集部署一个文件系统）。存储管理也得以简化，因为无需在计算集群间复制数据。Lustre 文件系统不仅聚合了众多服务器的存储容量，也聚合了其 I/O 吞吐率，且可通过增加服务器而扩展。通过动态添加服务器，可轻松增加吞吐率和容量。

Lustre 文件系统可在众多工作环境中运行，但并不一定是所有应用程序的最佳选择。虽然在一些使用场景下，因其强大的锁机制和数据一致性保障机制，Lustre 文件系统的

单服务器性能可能也比其他文件系统高，但使用 Lustre 最合适的场景仍然是：应用所需超过了单个服务器的能力。

目前，因为 Lustre 缺少软件级别的数据副本，Lustre 文件系统并不特别适合用这种于 ``点对点" 的使用模式：客户端和服务器运行在同一节点上，每个节点共享一小块存储。在该使用模式下，如果一个节点（连同其上的客户端和服务器）发生故障，存储在该节点上的数据将无法存取，直至该节点重启。

1.1.1. Lustre 的功能

Lustre 文件系统可运行多种发行版供应商提供的内核上。细节可查看本手册 8.1 节 ``准备安装 Lustre 软件"。

一个 Lustre 系统可在客户端节点数量、磁盘存储量、带宽各方面进行扩展或裁剪。其可扩展性和性能取决于系统中的可用磁盘、网络带宽以及服务器的处理能力。Lustre 文件系统可部署在多种配置下，这些配置可扩展到的容量和性能，超越了能在现有生产系统看到的情况。

下表中列出了一些 Lustre 文件系统的可扩展性和性能特征（更完整的 Lustre 文件系统极限列表，可参看 5.2 节）：

特征	当前实用范围	已知生产环境使用
客户端可扩展性	100-100000	50000+ 客户端, 许多或在 10000 ～ 20000 之间
客户端性能	单客户端：I/O 性能为 90% 的网络带宽。聚合：I/O 性能为 50TB/s，50M IOPS。	单客户端: 15 GB/s （HDR IB），50000 IOPS。聚合: I/O 性能为 10TB/s 10M IOPS。
OSS 可扩展性	单 OSS：支持 1 到 32 个 OST；单 OST：支持 5 亿对象, 1024TiB 容量。OSS 数目：1000 个 OSS，4000 个 OST。	单 OSS：连接 4 个 OST。单 OST：容量 1024TiB。OSS 数目：450 个 OSS，900 个基于 750 TiB HDD 的 OST，外加 450 个基于 25 TiB NVMe 的 OST。
OSS 性能	单 OSS：15GB/s，1.5M IOPS。聚合：50TB/s，50M IOPS。	单 OSS：10GB/s，1.5M IOPS。聚合：20TB/s，20M IOPS。

特征	当前实用范围	已知生产环境使用
MDS 扩展性	单 MDS：1 到 4 个 MDT。单 MDT：ldiskfs 情况下，单 MDT 40 亿个文件，16TiB 容量；ZFS 情况下，单 MDT 640 亿个文件，64TiB 容量。	单个 MDS：40 亿文件。MDS 数目：生产环境中 40 个 MDS，40 个 4TiB 的 MDT；测试环境中，256 个 MDS，256 个 64GiB 的 MDT。
MDS 性能	1M/s 的创建性能。2M/s 的 stat 性能。	100k/s 的创建性能。200k/s 的 stat 性能。
文件系统可扩展性	单文件：基于 ldiskfs 最大单文件大小 32PiB。聚合：512 PiB 容量，1 万亿文件。	单文件：最大单文件几个 TiB。聚合：700 PiB 空间，250 亿文件。

其他 Lustre 软件性能特征如下：

- **性能增强的 ext4 文件系统：**Lustre 文件系统使用改进版的 ext4 日志文件系统来存储数据和元数据。该版本名为 ldiskfs，不仅性能有所提升且提供了 Lustre 文件系统所需的附加功能。Lustre 也可使用 ZFS 作为 Lustre 的 MDT、OST 和 MGS 存储的后端文件系统。这使得 Lustre 能够在单个存储目标上利用 ZFS 的可扩展性和数据完整性功能。

- **符合 POSIX 标准：**POSIX 测试可在 Lustre 客户端上完整通过，就像能在本地的 ext4 文件系统通过一样，只有少量例外。在集群环境下，大多数 Lustre 操作都是原子操作，因此客户端不会看到过期的数据或元数据。Lustre 文件系统支持 mmap() 文件 I/O 操作。

- **高性能异构网络：**Lustre 软件支持各种高性能低延迟的网络，支持在 InfiniBand（利用 OpenFabrics Enterprise Distribution，OFED）、Intel OmniPath 及其他高级网上基于远程直接内存访问（Remote Direct Memory Access，RDMA）的方式获得快速、高效等网络传输。Lustre 路由可将多个 RDMA 网络桥接起来，以获得最佳性能。Lustre 软件也集成了网络诊断机制。

- **高可用：**Lustre 文件系统支持双活故障切换，实现方式是利用 OSTs（OSS 存储目标）和 MDT（MDS 存储目标）的共享存储分区。Lustre 文件系统可与各种高可用（High Availability，HA）管理程序一起工作，以实现自动故障切换并消除单点故障（No Single Point of Failure，NSPF）。这使得应用程序可以无感知地恢复。多重挂载

保护机制（Multiple Mount Protection，MMP）提供了内在集成的防护机制，防止因高可用系统的错误挂载而让文件系统遭受损坏。

- **安全：**默认情况下，Lustre 只允许特权端口建立 TCP 连接。UNIX 组成员身份会在 MDS 上进行验证。
- **访问控制列表（ACL）及扩展属性：**Lustre 遵循 UNIX 文件的安全模型，并使用 POSIX ACL 进行增强。值得推荐的是，它还有一些额外的安全功能，如 root squash。
- **互操作性：**Lustre 文件系统可运行在各类 CPU 架构和大小端混合的群集上。在依次发布的版本间，Lustre 具有互操作性。
- **基于对象的架构：**客户端与盘上文件结构相互隔离，可在不影响客户端的情况下升级存储架构。
- **字节粒度的文件锁和细粒度的元数据锁：**众多客户端可同时读取和修改同一文件或目录。Lustre 分布式锁管理器（Lustre Distributed Lock Manager，LDLM）确保了文件在文件系统中所有客户端和服务器间保持一致。MDT 锁管理器负责管理关于 inode 权限和路径名的锁。每个 OST 各自的锁管理器，用于管理存储于其上的文件条带的锁，其性能可随着文件系统的增大而扩展。
- **配额：**Lustre 文件系统支持用户和组配额。
- **容量增长：**通过在群集中新增 OST 和 MDT，可增加 Lustre 文件系统的容量和聚合带宽，可而无需中断服务。
- **控制文件布局：**跨 OST 的文件布局，可以以文件、目录或整个文件系统为单位进行配置。这使得我们可以在文件系统内，根据特定应用需求来优化文件 I/O。Lustre 文件系统使用 RAID-0 将数据条带化，并在 OST 间平衡空间用量。
- **网络数据完整性保护：**从客户端发送到 OSS 的所有数据具有校验码，和可防止数据在传输过程中遭到破坏。
- **MPI I/O：**Lustre 架构具有专用的 MPI ADIO 层，优化了并行 I/O 以匹配底层文件系统的架构特点。
- **NFS 和 CIFS 导出：**Lustre 文件系统可以通过 NFS（基于 Linux knfsd 或 Ganesha）或 CIFS（基于 Samba）重新导出，使其可以共享到非 Linux 客户端上（如 Microsoft Windows 和 Apple Mac OS X）。
- **灾难恢复工具：**Lustre 文件系统提供在线分布式文件系统检查工具（LFSCK）。在发生重大文件系统错误的情况下，该工具可恢复存储组件间的一致性。即使存在

不一致，Lustre 文件系统仍可运行，而在文件系统正在使用时，LFSCK 也可运行，因此即使 LFSCK 尚未完成，文件系统仍可恢复生产。

- **性能监视：**Lustre 文件系统提供了多种机制来进行性能检查和调优。
- **开放源代码：**Lustre 软件遵循 GPL 2.0 许可证，可在 Linux 操作系统上自由运行。

1.2. Lustre 组件

一个 Lustre 软件装置包含一个管理服务器（MGS）及一个或多个 Lustre 文件系统，这些部件通过 Lustre 网络（LNet）互连起来。下图给出了 Lustre 文件系统组件的一种基础配置：

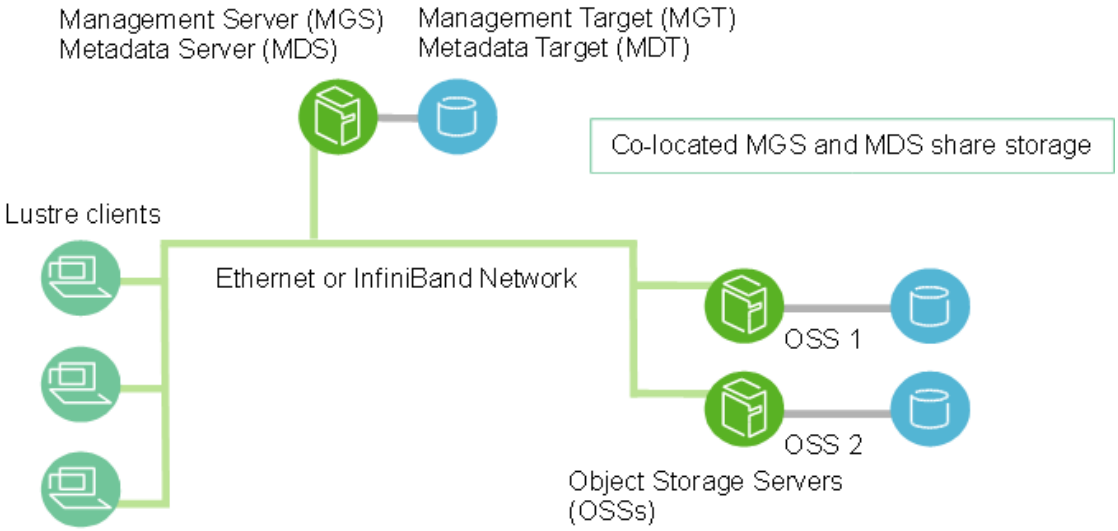


图 1: Lustre component

1.2.1. 管理服务器 (MGS)

MGS 存储了集群中所有 Lustre 文件系统的配置信息，并将此信息提供给其他 Lustre 组件。每个 Lustre 目标通过联系 MGS 来提供信息，而 Lustre 客户通过联系 MGS 获取信息。

MGS 最好能有自己的存储空间，以便独立管理。但同时，MGS 可以与 MDS 放在一起共享存储空间，如上图中所示。

1.2.2 Lustre 文件系统的组件

每个 Lustre 文件系统都由以下组件组成：

- **元数据服务器（Metadata Server，MDS）** - MDS 将存储在一个或多个 MDT 中的元数据提供给 Lustre 客户端使用。每个 MDS 管理 Lustre 文件系统中的名字和目

录，并为一个或多个本地 MDT 提供网络请求处理。

- **元数据目标（Metdata Targets，MDT）** - 每个文件系统都至少有一个 MDT，用于存放根目录。MDT 将元数据（例如文件名，目录，权限和文件布局）存储在 MDS 外挂的存储上。虽然共享存储目标上的 MDT 对多个 MDS 可见，但一次只能由一个 MDS 访问。如果一个活跃的 MDS 节点发生故障，另外一个 MDS 节点可以接管该 MDT，并将其服务提供给客户端。这被称为 MDS 故障切换。

分布式命名空间环境（Distributed Namespace Environment，DNE）可支持多个 MDT。除了存储文件系统根目录的主 MDT 之外，还可添加其他 MDS 节点，每个 MDS 节点可拥有自己的 MDT，用以存储文件系统的子目录树。

自 **Lustre 2.8 版本**起，DNE 还支持文件系统将单个目录下的子文件散布到多个 MDT 节点上。跨多个 MDT 分布的目录称为条带化目录（Striped Directory）。

- **对象存储服务器（OSS）：** OSS 为一个或多个本地 OST 提供文件 I/O 服务并处理网络请求。OSS 通常服务两个到八个 OST，每个 OST 容量多达 16TiB。一个典型的配置是：在一个专属节点上配备一个 MDT，在每个 OSS 节点上配置两个或更多 OST，在大量计算节点上安装客户端。

- **对象存储目标（OST）：** 用户的文件数据存储在一个或多个对象中，每个对象单独位于 Lustre 文件系统的 OST 中。每个文件的对象数目可由用户自行调配，对于特定的工作负载，可自行调试以获得最佳性能。

- **Lustre 客户端：** Lustre 客户端可以是计算节点、可视化节点或桌面节点，它们运行了 Lustre 客户端软件，从而可以挂载 Lustre 文件系统。

Lustre 客户端软件提供了一个接口，连接了 Linux 虚拟文件系统和 Lustre 服务器。客户端软件包含一个管理客户端（Management Client，MGC）、一个元数据客户端（Metadata Client，MDC）及多个对象存储客户端（Object Storage Clients，OSC）。每个 OSC 对应文件系统中的一个 OST。

逻辑对象卷（Logical Object Volume，LOV）将 OSC 集合起来，以提供一种跨越所有 OST 的透明访问。因此，挂载了 Lustre 文件系统的客户端看到的是一个单一的、一致的、同步的命名空间。多个客户端可以同时写入同一文件的不同部位，而与此同时，其他客户端也可以读取文件。

与 LOV 为文件访问所提供的功能类似，逻辑元数据卷（Logical Metadata Volume，LMV）将 MDC 集合起来，以提供一种跨越所有 MDT 的透明访问。这使得客户端可将分布在多个 MDT 上的目录树视为一个单一的、一致的命名空间，并将条带化目录合并到客户端形成一个单一目录以便用户和应用程序查看。

下表给出了每种 Lustre 文件系统组件所属存储的要求，以及使用硬件的合适特性。

	所属存储的要求	可取的硬件特性
MDS	1-2% 的文件系统容量	强大的 CPU 能力, 充足的内存, 快速的磁盘存储。
OSS	每个 OST 1 到 128 TiB，每个 OSS 1 到 8 个 OST	良好的总线带宽，建议在 OSS 间均衡分配存储，并使之与网络带宽匹配。
客户端	无需本地存储	低延迟，高带宽网络

1.2.3 Lustre 网络 (LNet)

Lustre Networking（LNet）是一种定制化网络 API，提供了通信基础设施，用以处理 Lustre 文件系统服务器和客户端间的元数据和文件 I/O 数据通信。更多关于 LNet 的介绍，请查看 ``理解 Lustre 网络（Lnet）" 一章。

1.2.4 Lustre 集群

在大规模系统上，一个 Lustre 文件系统集群可包含数百个 OSS 和数千个客户端（如下图所示）。Lustre 集群中可以使用多种类型的网络。OSS 间共享存储，因而可以启用故障切换功能。更多关于故障切换的介绍，请查看 ``理解 Lustre 文件系统种的故障切换" 一章。

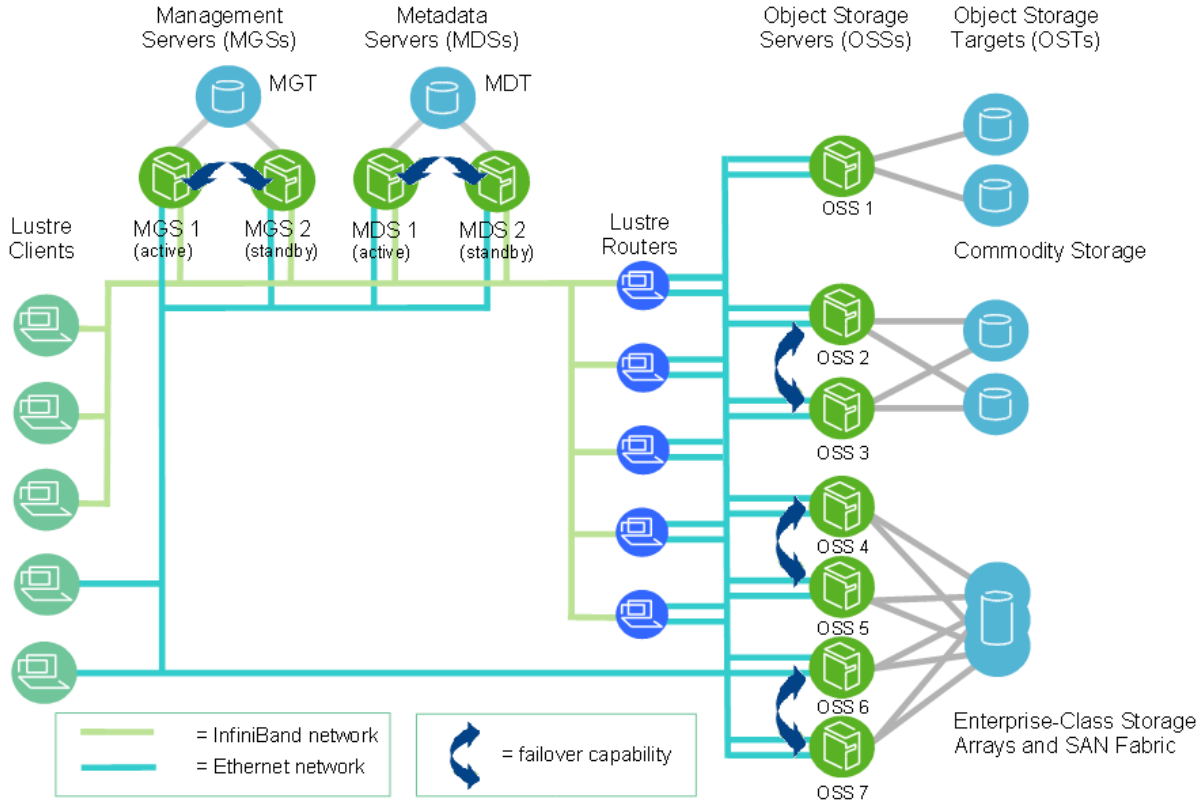


图 2: Lustre cluster at scale

1.3. Lustre 文件系统存储与 I/O

Lustre 文件标识符（File IDentifier，FID）在内部用于识别文件或对象，类似于本地文件系统的 inode 号。FID 是一个 128 位的标识符，包含一个唯一的 64 位序列号（Sequence Number，SEQ），一个 32 位对象 ID（Object ID，OID）和一个 32 位版本号（Version Number）。序列号在文件系统的所有 Lustre 目标（OST 和 MDT）中保持唯一，从而使得多个 MDT 和 OST 能够唯一地识别对象，而不用依赖底层文件系统的标识符（例如，inode 号）。这些底层文件系统的标识符则有可能在目标之间发生重叠和冲突。FID 的 SEQ 编号也使得我们可以将 FID 对应到相应的 MDT 或 OST 上。

LFSCK 是文件系统一致性检查工具，它提供了对现有文件启用 FID-in-dirent 的功能。其包含的功能如下：

- 验证每个目录条目上存储的 FID，如果 FID 无效或缺失，则从 inode 中重新生成。
- 验证每个 inode 的 linkEA 条目，如 linkEA 无效或丢失，则重新生成。linkEA 存储了自己的文件名和父目录的 FID。linkEA 以扩展属性的形式存储在每个 inode 中。因此，可利用 linkEA 来仅从 FID 重建文件的完整路径。

有关文件数据在 OST 上所处位置的信息也以一个扩展属性的形式存储，称为布局扩展属性（Layout Extended Atribute，Layout EA）。布局扩展属性存储在 MDT 对象中（具体如下图所示），MDT 对象则由 FID 标识。若该文件是普通文件（非目录或符号链接），则该 MDT 对象以 1-N 的方式指向多个位于 OST 上的 OST 对象，这些 OST 对象包含部分的文件数据。若该 MDT 文件指向一个 OST 对象，则所有的文件数据都存储在该 OST 对象中。若该 MDT 文件指向多个对象，则以 RAID 0 的方式将文件数据条带化（Stripe）到多个对象上，而每个 OST 对象分别存储在不同的 OST 上。关于 Lustre 文件系统如何进行条带化，请参看 1.3.1 节 ``Lustre 文件系统和条带化"。

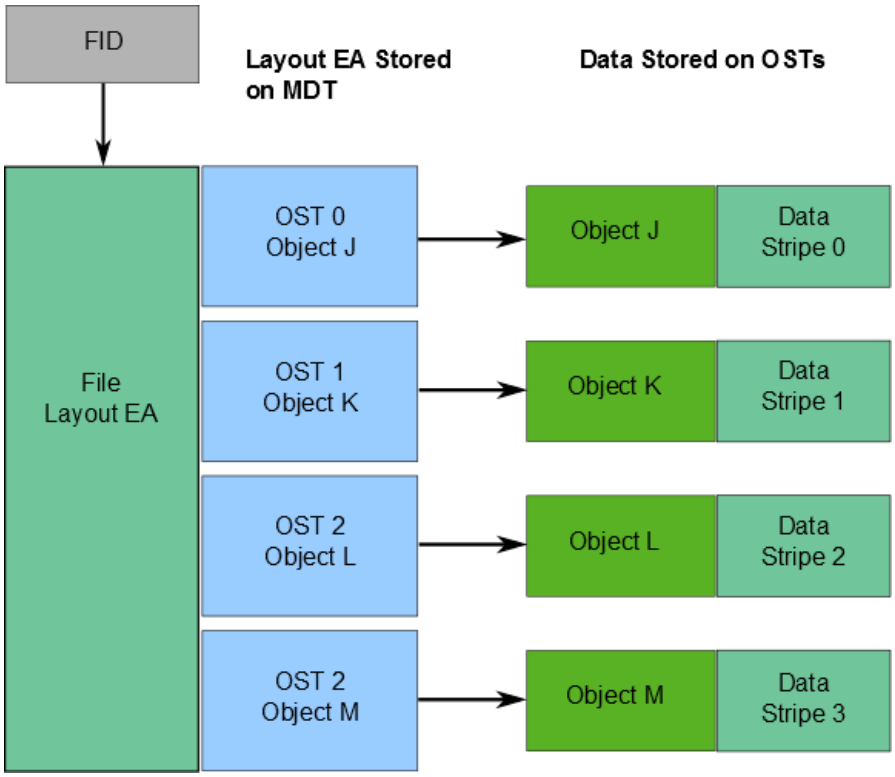


图 3: Lustre cluster at scale

当客户端想要读写文件时，首先从文件的 MDT 对象中获取布局 EA，然后使用这个信息对文件执行 I/O，此时会直接与存储了对象的 OSS 节点进行交互。具体过程如下图所示。

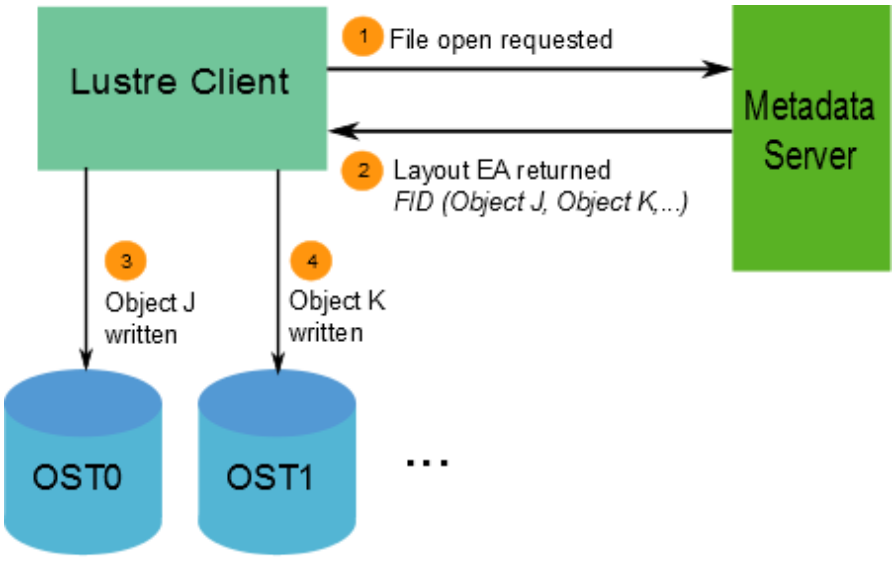


图 4: Lustre cluster at scale

Lustre 文件系统的可用带宽由以下因素决定：

- 网络带宽等于 OSS 到存储目标的带宽之和。
- 磁盘带宽等于存储目标（OST）的磁盘带宽之和，受网络带宽限制。

- 聚合带宽等于磁盘带宽和网络带宽的最小值。
- 可用的文件系统空间等于所有 OST 的可用空间之和。

1.3.1. Lustre 文件系统条带化

Lustre 文件系统之所以性能高，主要原因之一是能够以轮询方式将数据条带化到多个 OST 上。用户可根据需要为每个文件设置条带数量、条带大小和 OST。

使用条带，可以让单个文件的聚合带宽超过单个 OST 的带宽，从而提高性能。同时，当单个 OST 没有足够的可用空间来容纳整个文件时，条带化也可发挥作用。关于文件条带化的优点和缺点，请查看 19.2 节 ``Lustre 文件布局（条带化）的一些考量"。

如下图所示，条带将文件的数据区段或区块存储在不同的 OST 中。在 Lustre 文件系统中，数据以 RAID 0 的模式分条在若干数目的对象中。一个文件中的对象数目称为 stripe_count。

每个对象包含文件的一块数据，当写入特定对象的数据块超过 stripe_size 时，文件的下一数据块将存储在下一对象中。

stripe_count 和 stripe_size 的默认值由文件系统设置，其中，stripe_count 为 1，stripe_size 为 1MB。用户可以在每个目录或每个文件上更改这些值。

下图中，文件 C 的 stripe_size 大于文件 A 的 stripe_size，这让 C 能将更多的数据存储在文单个条带中。文件 A 的 stripe_count 为 3，导致 A 的数据分条在三个对象上。而文件 B 和文件 C 的 stripe_count 是 1。

OST 不会为未写入的数据预留空间。

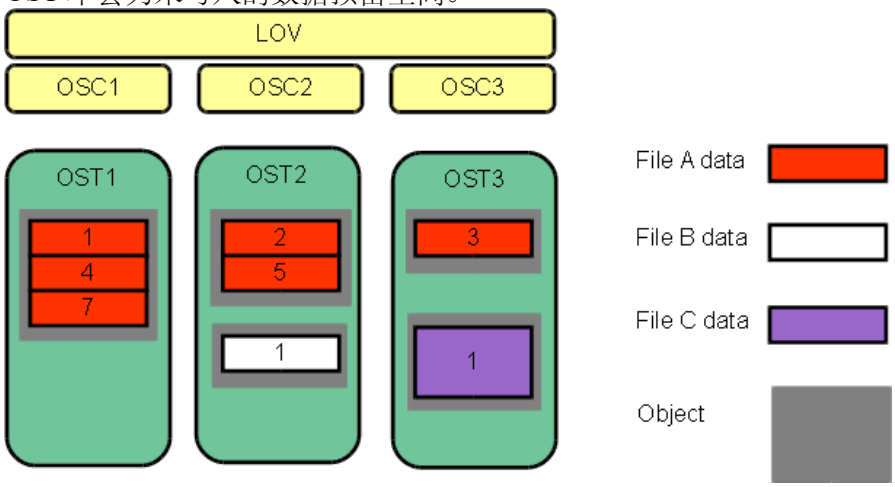


图 5: Lustre cluster at scale

文件的最大大小不受单个目标大小的限制。在 Lustre 文件系统中，文件可以跨越多个对象（最多 2000 个）条带化，在 ldiskfs 上，每个对象最大可达 16 TiB，在 ZFS 上，每个对象最大可达 256PiB。因而，ldiskfs 的最大文件大小为 31.25 PB，ZFS 的最大文件大小为 8EiB。Lustre 文件系统可支持最多 2^63 字节（8EB）的文件，只受限于 OST 上的可用空间。

注意

对于不支持ea_inode功能的 ldiskfs 文件系统，其单文件的最大条带数为 160 个 OST。

尽管一个文件只能条带化到 2000 个对象上，但是 Lustre 文件系统本身可拥有数千个 OST。访问单个文件的 I/O 带宽是文件所属所有对象的聚合 I/O 带宽，可至多获得高至 2000 个服务器的带宽。在具有 2000 多个 OST 的系统上，客户端通过同时执行多个文件读写来完美利用文件系统总带宽。

更多关于条带化的信息，请参阅第 19 章，管理文件布局（条带化）及剩余空间。

扩展属性 (xattrs)

Lustre 以 lov_user_md_v1/lov_user_md_v3 为数据结构，以扩展属性为存储形式，维护其文件条带信息。在文件和目录创建时，扩展属性被创建。Lustre 以一系列 trusted 扩展属性来存储一系列参数，这些参数只有 root 才能访问。这些参数包括：- **trusted.lov**: 保存普通文件的布局信息，或存储在目录上的默认文件布局信息（非 root 用户也可以通过lustre.lov访问）。- **trusted.lma**: 保存当前文件的 FID 和额外的状态标志。- **trusted.lmv**: 保存一个条带化目录的布局（DNE 2），如果目录没有条带化，则此参数不存在。- **trusted.link**: 对每个硬链接，保存其父目录 FID+ 文件名（用于lfs fid2path）。

在文件中存储且公开展示的 xattr 可以用以下方式验证：

```
1 # getfattr -d -m - /mnt/testfs/file>
```

第二章理解 Lustre 网络（LNet）

本章介绍 Lustre 网络（LNET）。

2.1. LNet 简介

在包含一个或多个 Lustre 文件系统的集群中，Lustre 文件系统所需的网络通信基础设施是通过 Lustre Networking（LNet）功能来实现的。

LNet 支持许多常见的网络类型，如 InfiniBand 和 IP 网络，并支持多种不同网络类型的同时可用及它们之间的路由。LNet 支持远程直接内存访问（Remote Direct Memory Access，RDMA），只要底层网络支持这一特性，且安装了合适的 Lustre 网络驱动程序（Lustre Network Driver，LND）。结合服务器的故障切换机制，LNET 的高可用和恢复功能，可以实现恢复对应用透明。

LND 是一种可插拔驱动程序，可为特定网络类型提供支持。例如，ksocklnd 实现了 TCP Socket LND，是支持 TCP 网络的驱动程序。LND 会被加载到驱动程序堆栈中，每种 LND 对应着正在使用中的一种网络类型。

如何配置 Lnet 网络，请查看第九章，``配置 Lustre 网络（LNet）"。

如何管理 Lnet 网络，请查看第三部分，``管理 Lustre"。

2.2. LNet 的主要功能

LNet 的主要功能包括：

- 远程直接内存访问，如果底层网络支持的话
- 支持多种常用的网络类型
- 高可用性和可恢复性
- 支持同时使用多种网络类型
- 多元网络间的路由

在各种不同网络互联间，LNet 可以让端到端的读写吞吐率达到或接近峰值带宽速率。

2.3.Lustre 网络

Lustre 网络由运行 Lustre 软件的客户端和服务器组成。它不局限于一个 LNet 子网，倘若网络之间可以进行路由，它可以跨越多个网络。类似地，一个单独的网络可以包含多个 LNet 子网。

Lustre 网络堆栈由两层组成：LNet 代码模块和 LND。LNet 层在 LND 层之上操作，其方式类似于网络层在数据链路层之上操作。LNet 层是无连接的、异步的，不进行传输数据验证。LND 层是面向连接，通常进行数据传输验证。

LNet 通过唯一的标签进行标识，该标签由两部分组成，前一部分为 LND 对应字符串，后一部分为一个数字，如 tcp0、o2ib0、o2ib1。LNet 上的每个节点至少有一个网络标识符（Network IDentifier，NID）。NID 由网络接口地址和 LNet 标签组成，形式为：`*address*@*LNet_label*`.

例如:

```
1 192.168.1.2@tcp0
2 10.13.24.90@o2ib1
```

在某些情况下，Lustre 文件系统流量可能需要在多个 LNets 之间传递，这就需要用到 LNet 路由。请注意，LNet 路由不同于网络路由。

2.4. 支持的网络类型

LNet 代码模块所包含的 LNDs 支持以下网络类型：

- InfiniBand: OpenFabrics OFED (o2ib)
- TCP（包括 GigE、10GigE、IPoIB 等在内的任何可传输 TCP 通信的网络）

- RapidArray: ra
- Quadrics: Elan

第三章 Lustre 文件系统的故障切换

3.1. 什么是故障切换

在高可用的（High-Availability,HA）系统中，通过使用冗余硬软件组件，可以让服务得以在故障发生时自动恢复，从而最大限度地减少计划外停机时间。当失效情况发生时，例如发生服务器宕机、存储设备宕机、网络故障或软件故障，在短暂中断后，系统服务将继续运行。通常，可用性的衡量方法是系统必须处在可工作状态的时间比例。

可用性通过硬件和（或）软件的冗余来实现，这样，当主服务器发生故障或不可用时，备用服务器将切换到就位状态，从而运行应用程序和相关资源。这个过程，称为故障切换（Failover），其在高可用性系统中是自动的，而且在大多数情况下，是对应用完全透明的。

支持故障切换的硬件装置需要一对服务器共享资源（通常共享的是物理存储设备，该设备可能基于 SAN、NAS、硬件 RAID、SCSI 或光纤通道技术）。在设备级别上，存储的共享方式必须基本上是透明的；相同的物理 LUN（Logical Unit Number）必须在两台服务器上可见。为了在物理存储级确保高可用，我们推荐使用 RAID 阵列来防护硬盘驱动器级别的故障。

注意

Lustre 软件暂不提供数据冗余，它完全依赖于备用存储设备的冗余。后端 OST 存储应该采用 RAID5，最好是 RAID6。MDT 存储应为 RAID1 或 RAID10。

3.1.1 故障切换的功能

为了创建高可用的 Lustre 文件系统，需要使用电源管理软件或硬件、高可用（High Availability，HA）软件，它们提供了故障切换的以下功能：

- **资源屏蔽：**防止两个节点同时访问物理存储。
- **资源管理：**作为故障切换的一部分，进行 Lustre 资源的启动和停止、维护集群状态、执行其他资源管理任务。
- **健康监控：**验证硬件和网络资源的可用性，并根据 Lustre 软件提供的健康指标进行响应。

这些功能可以由各种软件和（或）硬件解决方案提供。更多关于如何配合 Lustre 使用电源管理软硬件和高可用（High Availability，HA）软件的信息，请参考**第十一章，``在 Lustre 文件系统中配置 Lustre 文件系统''**。

HA 软件负责检测 Lustre 服务器节点的故障并控制故障切换。Lustre 软件可与任何含资源（I/O）屏蔽（fencing）功能的 HA 软件配合使用。为恰当地实现资源屏蔽，HA 软件必须能够完全关闭失效的服务器，或将其从共享存储设备上断开。若两个活动节点同时访问一个存储设备，则可能严重损坏数据。

3.1.2 故障切换配置类型

集群中的节点可以按照多种方式进行故障切换配置。它们通常成对配置（例如，两个 OST 连接到同一共享存储设备），但也存在其他故障切换配置方式。故障切换配置方式包括：

- **``主动/被动'' 对：**在此种配置中，主动节点供应资源并提供数据服务，而被动节点则通常闲置待命。如果主动节点发生故障，则被动节点将接管并成为主动节点。
- **``主动/主动'' 对：**两个节点都处于活动状态，每个节点都提供资源的部分子集。当发生故障时，故障节点上的资源由另一个节点接管。

如果文件系统仅包含单个 MDT，可将两个 MDS 配置为 ``主动/被动'' 对，而 OSS 可以用 ``主动/主动'' 的方式部署，从而既改进了 OST 的可用性又避免了额外开销。通常，备用 MDS 是另一个 Lustre 文件系统的活跃 MDS，或者是 MGS，因此集群中没有节点闲置。如果文件系统中包含多个 MDT，可以使用 ``主动/主动'' 故障切换配置来部署 MDS，从而为共享存储上的 MDT 提供服务。

3.2. Lustre 文件系统中的故障切换功能

Lustre 软件提供的故障切换功能可以在下面这种场景下发挥作用。当客户端尝试发出 I/O 时，但该 Lustre 目标发生故障，这个客户端将反复尝试，直到从该 Lustre 目标的任一已配置的故障切换节点收到回复。除了完成 I/O 操作所需的时间可能变长之外，用户空间应用程序检测不到任何异常。

Lustre 文件系统的故障切换功能要求将两个节点配置为一对故障切换对，这一对节点间共享一个或多个存储设备。Lustre 文件系统可通过不同的配置，提供 MDT 或 OST 的故障切换能力。

- **MDT 故障切换：**可为同一个 MDT 配置两个 MDS 节点。在任何时刻，任一 MDT 的服务只由一个 MDS 节点为提供。通过将两个或更多 MDT 分区放置在由两个 MDS 节点共享的存储上，当一个 MDS 发生故障时，另一个 MDS 可为无服务的 MDT 提供服务。这也就是 ``主动/主动 ``故障切换对（active/active failover pair）。
- **OST 故障切换：**可为一个 OST 配置多个 OSS 节点，但在任何时刻，只有一个 OSS 节点为 OST 提供服务。可使用`umount/mount`命令在可访问同一存储设备的 OSS 节点之间移动 OST 服务。

在创建 Lustre 文件系统时（用`mkfs.lustre`命令），可使用`--servicenode`选项来指定服务的故障切换节点。在激活 Lustre 文件系统后，也可以使用该选项来修改服务的故障切换节点（`tunefs.lustre`命令）。关于这些工具的介绍，请参看 ** 第 44.13 节，``tunefs.lustre``**。

Lustre 文件系统中的故障切换功能可用于在后续小版本间升级 Lustre 软件，而无须中断集群的运行。更多介绍，请参看**第十七章 ``升级 Lustre 文件系统''**。

更多关于故障切换功能的配置，请参看**第十一章 ``配置 Lustre 文件系统的故障切换''**。

注意：Lustre 软件仅在文件系统级别提供故障切换功能。在完整的故障切换解决方案中，系统级组件的故障切换功能（如节点故障检测或电源控制）必须由第三方工具提供。

注意：OST 故障切换功能无法防止磁盘故障造成的损坏。假如用作 OST 的存储介质（即物理磁盘）发生故障，则无法通过 Lustre 软件提供的功能来恢复。我们强烈建议在 OST 上使用某种形式的 RAID。Lustre 假设存储是可靠的，因此没有提供功能来提高存储的可靠性。

3.2.1 MDT 故障切换配置（主动/被动）

如下图所示，通常会配置两个 MDS 为 ``主动/被动'' 故障切换对。请注意，两个节点都必须能够访问 MDT 和 MGS 的共享存储。主（主动）MDS 管理 Lustre 系统元数据资源。当主 MDS 发生故障，则从（被动）MDS 将接管这些资源并为 MDT 和 MGS 提供服务。

注意：在具有多个文件系统的环境中，MDS 可配置为类似主动/主动的模式，每个 MDS 管理这些 Lustre 文件系统中元数据的一个子集。

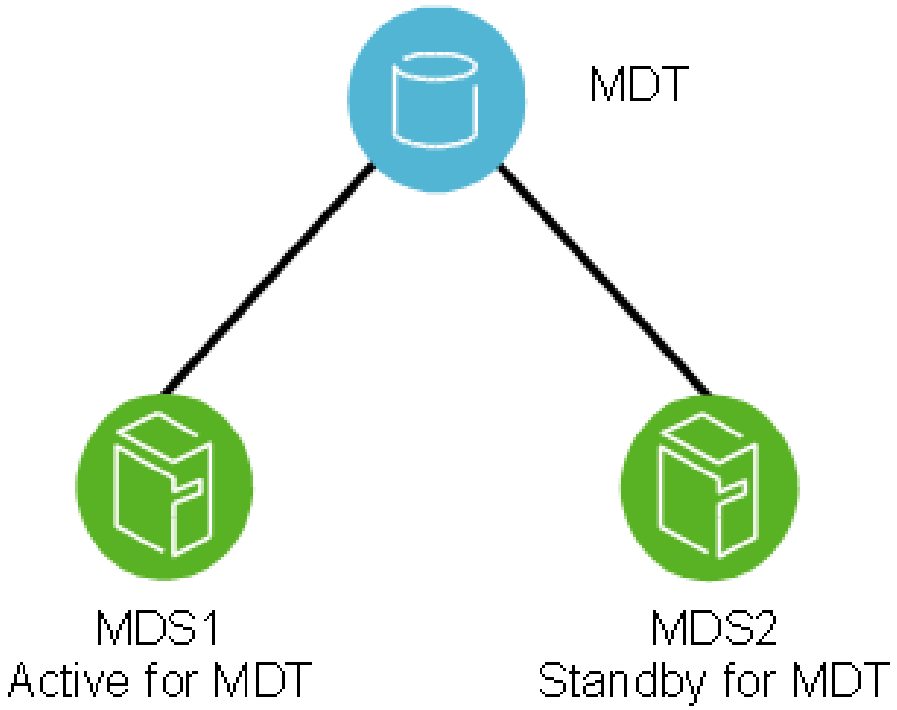


图 6: MDT_activepassive

3.2.2 MDT 故障切换配置（主动/主动）

MDT 设置可为 ``主动/主动" 故障切换模式。故障切换集群由两个 MDS 组成，如下图所示。

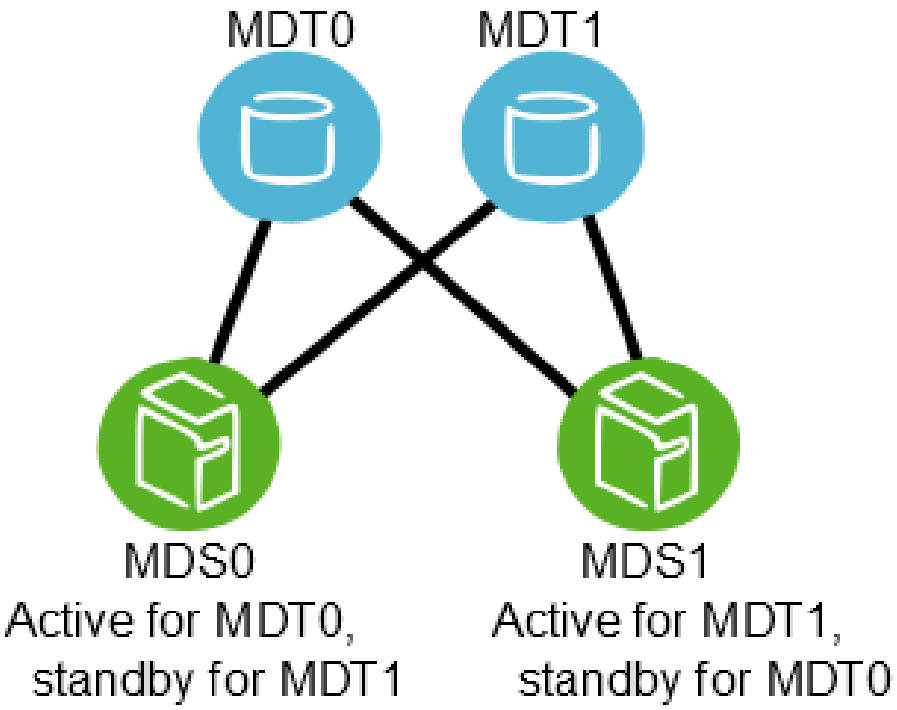


图 7: MDT_activeactive

3.2.3 OST 故障切换配置（主动/主动）

OST 通常配置为负载均衡的、``主动/主动" 式故障切换对。一个故障切换集群由两个 OSS 组成，如下图所示。

注意：配置为故障切换对的 OSS 必须共享磁盘或 RAID。

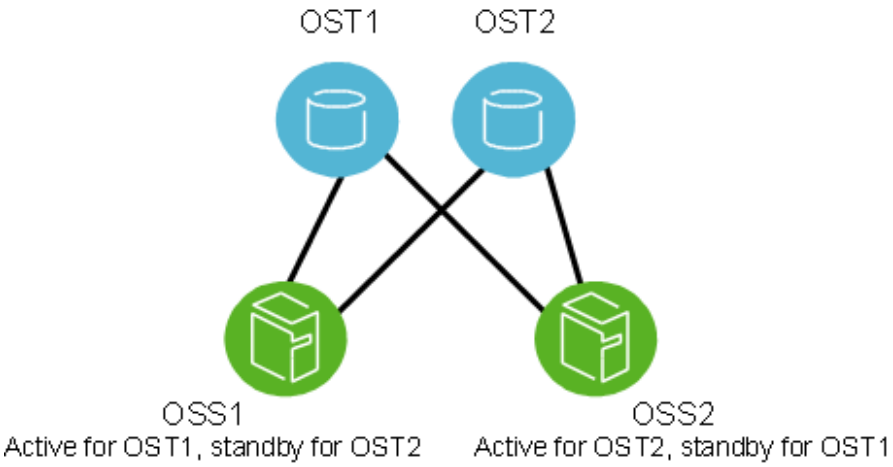


图 8: OST_activeactive

在 ``主动/主动" 式配置中，50％的可用 OST 分配给一个 OSS，其余的 OST 分配给另一个 OSS。每个 OSS 作为半数 OST 的主节点，又作为其余 OST 的故障切换节点。

在这种模式下，如果一个 OSS 故障，那么另一个 OSS 会接管所有剩余的 OST。客户端将尝试连接每个提供 OST 服务的 OSS，直到其中一个有响应。OST 上的数据是同步写入的，而故障切换完成后，客户端会重新发起在 OST 故障时正在进行但未提交到磁盘的事务。

更多关于故障切换功能的配置，请参看**第十一章，``配置 Lustre 文件系统的故障切换"**。

第四章安装概述

本章主要介绍了安排、安装和配置 Lustre 文件系统的全过程。

注意：如果 Lustre 文件系统对您来说是个新事物，那么在安装之前，建议您本文档的前面部分，以了解 Lustre 架构、文件系统组件和术语。

4.1. 安装 Lustre 软件的步骤

设置 Lustre 文件系统硬件、安装和配置 Lustre 软件，请按以下顺序进行，并参阅下列章节：

1. *(必要)* **设置 Lustre 文件系统硬件。**

请参阅 ``第五章 Lustre 硬件配置要求和格式化选项"------提供了为 Lustre 文件系统配置硬件的准则，包括对存储，内存和网络的要求。

2. *(可选 - 高度推荐)* **在 Lustre 设备上配置存储。**

请参阅 ``第六章 Lustre 文件系统的存储配置"------介绍了在 Lustre 存储设备上配备硬件 RAID 的说明。

3. *(可选)* **设置网络接口绑定。**

请参阅 ``第七章网络端口绑定设置"------讲解了如何设置网络接口绑定，从而同时使用多个网络接口，以实现网络冗余或提升带宽。

4. *(必要)* **安装 Lustre 软件。**

请参阅 ``第八章 Lustre 软件系统安装"------介绍了 Lustre 软件安装的准备步骤和实施步骤。

5. *(可选)* **配置 Lustre 网络 (LNet)。**

请参阅 ``第九章 Lustre 网络配置 (LNet)"------介绍了如果默认配置不够的情况下，如何配置 LNet。在默认配置下，LNet 使用在系统中找到的第一个 TCP/IP 网络。如果你在使用 InfiniBand 或多个以太网接口，则需要使用 LNet 配置。

6. *(必要)* **配置 Lustre 文件系统。**

请参阅 ``第十章 Lustre 文件系统配置"------提供了简单的 Lustre 配置过程示例，并介绍了用于完成更复杂配置的工具。

7. *(可选)* **配置 Lustre 故障切换。**

请参阅 ``第十一章 Lustre 故障切换配置"------介绍了如何配置 Lustre 故障切换。

第五章 Lustre 硬件配置要求和格式化选项

5.1. 硬件方面的考虑

Lustre 文件系统可以使用任何类型的块存储设备，如单个磁盘，软件 RAID，硬件 RAID，或者 LVM 逻辑卷。不同于其他一些网络文件系统，块设备只能连接到 Lustre 文件系统的 MDS 和 OSS 节点，而不能由客户端直接访问。

由于块设备仅由一个或两个服务器节点访问，因此无需采用所有服务器都能访问的 SAN（Storage Area Network，存储局域网）。在服务器和存储阵列之间，点对点连接通

常就能提供最简单、性能最好的连接，因此无需采用价格昂贵的交换机。(若需故障切换功能，那么存储必须连接到多个服务器。)

在生产环境下，最好给 MGS 配备单独的存储空间，以便将来扩展到多个文件系统。但是也可以在一台机器上同时运行 MDS 和 MGS，并让 MDS 和 MGS 共享同一个存储设备。

在生产环境中，为了获得最佳性能，非常有必要使用专门的客户端。在非生产环境或测试环境下，Lustre 客户端和服务器可以在同一台机器上运行。尽管如此，不使用专用客户端的配置是不受支持的。

警告：如果您将客户端放在 MDS 或 OSS 上，可能会出现以下性能和恢复的问题:

- 在同一台机器上运行 OSS 和客户端，可能导致低内存问题和内存压力。客户端尝试将数据写入文件系统，OSS 需要分配页面来从客户端接收数据。如果客户端消耗了所有内存，那么此操作会由于内存不足而无法执行。这会导致客户端挂起。
- 在同一台机器上运行 MDS 和客户端，可能导致恢复和死锁问题，并影响其他 Lustre 客户端性能。

Lustre 只支持服务器测试和运行在 64 比特 CPU 上。Lustre 客户端也一般在 64 比特的 CPU 上运行测试，以匹配预期的客户使用方式，同时避免 32 比特 CPU 存在的诸多限制，如 4GB 内存上限、1GB 的低端内存（low memory，内核可直接寻址）上限、16TB 文件大小上限。此外，由于内核 API 的限制，如果在 32 比特 CPU 上运行 Lustre 备份工具，而备份工具又必须依赖索引节点号码来正确运行，那么备份工具可能会因为把不同的文件的索引节点号错看成同一个而混淆了这些文件。

连接到服务器的存储设备通常使用 RAID 来提供容错能力，还可以选用 LVM（Logical Volume Management ）来管理它，随后格式化为 Lustre 文件系统。Lustre OSS 和 MDS 服务器按照文件系统限定的格式进行数据读取、写入和修改。

Lustre 文件系统在 MDT 和 OST 上使用了日志（Journal）文件系统技术。对于 MDT，可通过将其日志放在单独的设备上获得 20% 的性能提升。

MDS 可以有效地利用多 CPU 核，建议至少配备四个处理器核。如果文件系统有许多客户端，建议为 MDS 配备更多的处理器核心。

注意：Lustre 支持客户端运行在多种 CPU 架构上，但有一个限制：客户端上的内核宏 PAGE_SIZE 大小必须与服务器相同。特别地，使用 64kB 大页的 ARM 或 PPC 客户端可以连接使用 4kB 页的 x86 服务器。

5.1.1. 对 MGT 和 MDT 存储硬件的考量因素

MGT 存储需求很小（即使在最大 Lustre 文件系统中也少于 100MB），而且 MGT 上的数据访问仅发生在挂载服务器或客户端的时候，所以也不需要考虑磁盘性能。然而，

这些数据对于文件系统的访问至关重要，所以 MGT 应采用可靠的存储，最好配置为镜像 RAID1。

对 MDS 存储的访问模式类似于对数据库的访问模式，其中包含大量的寻道和对小块数据的读后改写。因此，强烈推荐采用寻道时间低得多的存储类型作为 MDT，例如 SSD 驱动器或 NVMe 驱动器，而高转速的 SAS 磁盘也可以接受。

为了获得最高性能，MDT 应该配置为由一个内部日志和不同控制器下的两个磁盘组成的 RAID1。

如果需要更大的 MDT，可以创建由一对磁盘组成的多个 RAID1 设备，然后使用这些 RAID1 设备构建 RAID0 阵列。对于 ZFS，可以在 MDT 中使用`mirror VDEV`。这种配置方法提供了最高的可靠性，因为在同一个 RAID1 设备中的两个磁盘上同时发生多个故障的概率非常低。

与此相反（构建一对 RAID0 设备组成的 RAID1），即使只有两个磁盘发生故障，也有 50% 的可能性出现可导致整个 MDT 数据丢失的情况。第一个故障使镜像的一半整个失效，第二个故障则有 50% 的概率使镜像的剩余一半失效。

如果系统中会存在多个 MDT，应根据每个 MDT 的预期使用方式和负载情况为之指定相应硬件配置。关于如何往文件系统中添加新的 MDT，请参考**第 14.7 节 ``在文件系统中添加新 MDT''**。

警告：MDT0000 含有 Lustre 文件系统的根目录。如果 MDT0000 因为任何原因而无法使用，那么文件系统就无法使用。

注意：通过使用 DNE（Distributed NamEspace）特性，`lfs mkdir -i mdt_index`命令可以把文件系统根目录下的子目录，或更下层的任意子目录，从 MDT0000 下分离出来，存储在新添加的 MDT 上。这种设置通常对顶层目录很有用，这样可以把不同的用户或项目指派给不同的 MDT 上，或者将几个大型文件工作集分布到多个 MDT 上。

在 Lustre 2.8 中引入：从 2.8 版本开始，就可以使用 DNE 条带目录特性，将单个大型文件目录分散到多个 MDT 上。在创建目录时，可通过`lfs mkdir -c stripe_count`命令，将目录分为多个条带（或分片），其中`stripe_count`一般是文件系统中 MDT 的数量。通常，不会在文件系统中的所有目录上都使用条带化目录，因为相较于非条带目录，条带目录将产生额外开销。而如果大型的目录（目录条目超过五万个）中有大量输出文件同时创建，条带化目录则会很有帮助。

5.1.2. 对 OST 存储硬件的考量因素

OSS 存储的数据访问模式是流式 I/O 模式，它依赖于正在使用的应用程序的访问模式。每个 OSS 都可以管理多个对象存储目标（Object Storage Target，OST），每个 OST 对应一个卷，在 OSS 和多个 OST 之间，还有 I/O 流量负载平衡机制。为了在网络带宽和外接的存储带宽之间保持平衡，应合理配置 OSS，以防止 I/O 瓶颈。根据服务器硬件

的不同，OSS 通常服务 2 到 8 个目标，每个目标的容量通常在 24 到 48TB 之间，但最高可达 256TB。

Lustre 文件系统容量是存储目标容量总和。例如，64 个 OSS，每个 OSS 含两个 8TB 的 OST，则可提供一个容量接近 1PB 的文件系统。如果每个 OST 使用 10 个 1TB 的 SATA 磁盘（在 RAID-6 配置中使用 8 个数据磁盘加 2 个校验磁盘），每个驱动器可达 50MB/秒的带宽，则每个 OST 则可达 400MB/秒的磁盘带宽。如果这个系统作为存储后端，连接到服务器上，而存储系统网络，例如 InfiniBand 网络也提供了相匹配的带宽，那么每个 OSS 则可以提供高达 800MB/秒的端到端 I/O 带宽。（这里描述的架构限制很简单，但实际上需要慎重地进行硬件选择、基准测试和集成才能获得该结果。）

5.2. 确定空间需求

MDT 和 OST 各自需要的后端文件系统性能特性是相互独立的。MDT 后端文件系统的大小取决于整个 Lustre 文件系统中所需的索引节点总数，而 OST 聚合空间大小取决于存储在文件系统上的数据总量。如果 MGS 数据须存储在 MDT 设备上（同时位于 MGT 和 MDT），则应增加 100MB 到 MDT 的预估容量上。

在 Lustre 文件系统上每创建一个文件，就会在 MDT 上消耗一个索引节点，还会在文件条带所在的每个 OST 上各消耗一个 OST 对象。通常，每个文件的条带数目继承于整个系统的默认条带数目，但单个文件的条带数可用`lfs setstripe`选项进行设置。更多细节，请查看**第十九章管理文件布局（条带化）及剩余空间**。

在 Lustre ldiskfs 文件系统中，所有 MDT 的索引节点和 OST 的对象都会在文件系统首次格式化时分配好。在文件系统使用过程中，创建一个文件，与该文件关联的元数据将被存储在预先分配的这些索引节点中，而不会占用任何用于存储文件数据的空闲空间。如果 MDT 或 OST 已经格式化好，它们 ldiskfs 上的索引节点总数是无法轻易更改的。因此，在格式化时应创建足够多的索引节点，并且要预见到短期内的使用情况，预留一部分增长空间，以避免添加额外存储的麻烦。

默认情况下，由 Lustre 服务器用作存储用户数据对象和系统数据的 ldiskfs 文件系统会预留 5% 的空间，该空间不会被 Lustre 文件系统使用。此外，Lustre ldiskfs 文件系统在每个 OST 上预留 400MB 空间，在每个 MDT 上预留 4GB 空间用来放置日志，此外还要预留少量空间，放置限额统计数据。这个预留空间不能用于存储普通数据，因此在保存任何文件对象数据之前，至少 OST 上的这些空间已被占用。

当 MDT 或 OST 使用 ZFS 作为后端文件系统时，索引节点和文件数据的空间分配是动态的，索引节点可按需分配。每个索引节点至少需要 4kB 的可用空间（如果没有镜像），除此之外，还有目录、内部日志文件、扩展属性、ACL 等其他开销。ZFS 也同样预留了全部存储空间的 3% 左右，用作内部的和冗余的元数据，这部分空间不可为 Lustre 所用。由于扩展属性和 ACL 的大小高度依赖于内核版本和站点策略，因此对于期望得到的索引节点数目，最好高估它所需消耗的空间大小。任何多余的空间都可用于存储更

多的索引节点。

5.2.1. 确定 MGT 的空间需求

MGT 所需空间通常小于 100MB，该大小是由 MGS 管理的 Lustre 文件系统服务器总数决定的。

5.2.2. 确定 MDT 的空间需求

在计算 MDT 大小时，需要考虑的一个重要因素是存储在文件系统中的文件数量，而 MDT 上每个索引节点至少需要 2KiB 的可用空间。由于 MDT 通常使用 RAID-1+0 镜像，所需的总存储量还须翻倍。

请注意，每个 MDT 实际使用的空间大小与诸多因素有关，如每个目录下的文件数量、每个文件的条带数、文件是否含 ACL 或用户扩展属性、每个文件的硬链接数目。Lustre 文件系统元数据所需的存储通常是文件系统容量的 1% 到 2%，具体取决于文件平均大小。如果在 Lustre 2.11 或更高版本上使用了**第二十章 MDT 数据存储功能 (DoM)** 所介绍的 DoM 功能，那么 MDT 空间通常应该占总空间的 5% 或更多，空间比率取决于小文件在文件系统中的占比，以及 MDT 上`lod.*.dom_stripesize`的限制，还有文件的布局方式。

对于基于 ZFS 的 MDT 文件系统，在 MDT 和 OST 上创建的索引节点的数量是动态的，因此不太需要预先确定索引节点的数量，但是仍然需要根据总文件系统的大小而考虑 MDT 的总空间大小。

例如，如果文件平均大小为 5MiB，而您有 100TiB 可用的 OST 空间，那么您可以计算出每个 MDT 和 OST 的索引节点最小总量：`(500 TB * 1000000 MB/TB)/ 5 MB/inode = 100M inodes`。

建议您将 MDT 空间至少设置为最小索引节点总量的两倍，从而方便未来扩展，或者预防文件平均大小小于预期。因此，ldiskfs MDT 的最小空间为：`2 KiB/inode x 100 million inodes x 2 = 400 GiB`。

注意：如果平均文件大小非常小，例如只有 4KB，那么每个文件在 MDT 上所占用的空间将会和在 OST 上一样多。因此在这种情况下，强烈建议 DoM。考虑到每个索引节点的额外数据空间使用情况，每个索引节点上的 MDT 空间也应做出相应的增加:

```
6 KiB/inode x 100 million inodes x 2 = 1200 GiB
```

注意：如果 MDT 的索引节点太少，则会因无法创建新文件而导致 OST 上的空间无法被使用。这种情况下，`lfs df -i`和`df -i`命令会将文件系统的空闲索引节点的数量限定为 OST 上可用对象的总数。在格式化文件系统之前，请一定确认文件系统所需 MDT 的合适大小。在文件系统格式化后，若存储允许的话，还是可以增加索引节点数量的。对于 ldiskfs MDT 文件系统，如果底层块设备为 LVM 逻辑卷，而且其大小可扩

展，则可使用 resize2fs 工具。对于 ZFS，可添加新的（镜像后的）VDEVs 到 MDT 池中，以增加作为索引节点存储的总空间。索引节点会根据增加的空间容量而按比例增加。

注意：对于基于 ZFS 的 MDT 和 OST，`lfs df -i`所报告的索引节点总数和空闲数是基于当前每个索引节点所使用的平均空间大小来估计的。当首次格式化 ZFS 文件系统时，估算出来的空闲索引节点数量将会很保守（低），这是因为，为存储内部 Lustre 元数据而创建的目录与文件，两者数目间的比值很高（译者注：因此每个索引节点占用的平均空间较大）。随着普通用户创建文件越来越多，但该估计值会逐渐提高，此时文件的平均大小将更好地反映实际的站点使用情况。

注意：基于 DNE 远程目录特性，通过在文件系统中配置额外的 MDT，可以增加 Lustre 文件系统的索引节点总数，同时提升元数据聚合性能。

5.2.3. 确定 OST 的空间需求

对于 OST，每个对象所占用的空间，取决于运行在系统上的用户或应用程序的使用模式。Lustre 软件对平均对象大小的估计较为保守（介于每个对象 64KiB 和 1MiB 之间，前者是对具备 10GiB 容量的 OST 的估计，后者为对具备 16TiB 容量的 OST 的估计）。如果确信应用程序的文件平均大小与此不同，您可以指定其他的平均文件大小（在规定的 OST 容量下索引节点的总数），从而减少文件系统开销，并且使得文件系统检查时间降到最短。

5.3. 设置 ldiskfs 文件系统的格式化选项

默认情况下，mkfs.lustre 工具将下面这些选项应用于存储数据和元数据的 Lustre 文件系统，以提高 Lustre 文件系统性能和可扩展性。这些选项包括:

- `flex_bg`：启用`flexible-block-groups`特性，属于多个块组（Block Group）的块位图及索引节点位图将聚集在一起，从而尽量减少读取或写入位图时的寻道操作，而且可以在典型的 RAID 存储（RAID 条带宽度为 1MiB ）上减少读/修改/写这类操作。OST 和 MDT 文件系统上都启用了该标志。在 MDT 文件系统中，`flex_bg`被设置为默认值 16。在 OST 中，`flex_bg`被设置为 256，这样单个`flex_bg`中，对所有的块位图或索引节点位图的读写可在一次 1MiB 的 I/O 中完成，而 1MiB 的 I/O 对于 RAID 存储具有典型性。
- `huge_file`： 设置此标志以允许 OST 上的文件大于 2TiB。
- `lazy_journal_init`： 如果没有打开这个选项，在格式化时，需要覆盖写入从而清零 Lustre 文件系统中默认分配的大块日志（OST 中高达 400 MiB，MDT 中高达 4GiB）。这个扩展选项可延迟这一覆盖写入，从而减少格式化时间。

我们可通过添加`mkfs.lustre`的参数来将格式化选项传递至后端文件系统，覆盖默认的格式化选项:

--mkfsoptions='backing fs options'

其他的mkfs.lustre的参数，请参看mke2fs(8)的 Linux 手册。

5.3.1. 为基于 ldiskfs 的 MDT 设置格式化选项

MDT 上的索引节点总数在格式化时确定，由要创建的文件系统总大小决定。在基于 ldiskfs 的 MDT 上，默认的每索引节点字节数比率（Bytes-per-Inode Ratio，Inode Ratio）被优化为每 2560 个字节的文件系统空间对应一个索引节点。

这个设置考虑到了 ldiskfs 文件系统级别的元数据所需要的额外空间，比如日志（最多 4GB）、位图和目录，同时还考虑到了 Lustre 用来保持集群内部一致性的文件。此外，还有每个文件的元数据，比如含多个条带的文件的布局信息、访问控制列表（ACL）、用户扩展属性。

在 Lustre 2.11 中引入：Lustre 2.11 引入了在 MDT 上存储数据（DoM）的特性，允许在 MDT 上存储小文件，从而利用闪存存储的高性能，并且减少空间和网络开销。如果您打算将 DoM 特性与 ldiskfs MDT 一起使用，建议增加每索引节点字节数的比率，从而在 MDT 上为小文件留出足够的空间，方法如下所述。

当首次格式化基于 ldiskfs 的 MDT 时，通过在mkfs.lustre添加--mkfsoptions="-i bytes-per-inode"选项，可把不同于默认的每索引节点 2560 字节的默认值修改为其他值。通过减小每索引节点字节数，可在大小给定的 MDT 上创建更多的索引节点，但为每个文件预留的额外元数据空间则变少，因此不推荐这么做。每索引节点字节数必须始终大于 MDT 上索引节点的的大小（默认为 1024 字节），建议采用的每索引节点字节数，至少比索引节点的大小还大 1536 字节，以确保不会耗尽 MDT 的空间。对于 DoM，建议增加每索引节点字节数，从而为最常见大小的文件提供足够的空间（例如文件普遍为 4KB 或 64KB，则应为每个索引节点保留 5632 字节或 66560 字节）。

通过添加--stripe-count-hint=N，可以让mkfs.lustre根据文件系统使用的默认条带数来自动计算合理的索引节点大小，也可以直接通过设置--mkfsoptions="-I inode-size"选项，来改变索引节点大小。增加索引节点大小意味着索引节点内部拥有更大的空间，以便存储更多的文件元数据，包括 Lustre 文件布局、ACL、用户和系统扩展属性、SELinux 和其他安全标签、其他内部元数据、DoM 数据等。但如果这些功能都不需要，也不需要在索引节点内部存储扩展属性，那么采用更大的索引节点大小可能会损害元数据性能，因为每次访问 MDT 索引节点都需要读取或写入 2 倍、4 倍甚至 8 倍的数据。

5.3.2. 为基于 ldiskfs 的 OST 设置格式化选项

在格式化一个 OST 文件系统时，应把本地文件系统的使用情况考虑进去，例如通过在当前文件系统上运行`df`和`df -i`来分别获取已用字节和已用索引节点，然后计算平均的每索引节点字节数。在给新系统指定每索引节点字节数时，在为将来更小的文件预留足够大回旋余地的同时，应尽量避免在每个 OST 上创建太多的索引节点。这有助于减少格式化时间和 e2fsck 时间，同时为数据保留更多可用空间。

下表列出了，在格式化时基于不同 OST 大小的默认每索引节点字节数。

LUN/OST 大小	默认的每索引节点字节数	索引节点总数
10GiB 以下	每索引节点 16KiB	640 至 655k
10GiB 至 1TiB	每索引节点 68KiB	153k 至 15.7M
1TiB 至 8TiB	每索引节点 256KiB	4.2M 至 33.6M
8TiB 以上	每索引节点 1MiB	8.4M 至 268M

在小文件很少的环境中，因其较大的平均文件大小，默认的每索引节点字节数将可能会导致索引节点数目过多。在这种情况下，可以通过增加每索引节点字节数来提高性能。为了指定每索引节点字节数，请给`mkfs.lustre`命令指定参数`--mkfsoptions="-i bytes-per-inode"`，从而指定 OST 对象的期望平均大小。例如，下面的命令用于创建一个平均对象大小预期为 8MiB 的 OST：

```
[oss #] mkfs.lustre --ost --mkfsoptions="-i $((8192 *
1024))" …
```

注意：使用 ldiskfs 格式化的 OST，每个 MDT 最好少于 3.2 亿个对象，且最多不超过 40 亿个索引节点。如果为一个超过此限制的大型 OST 指定了非常小的 inode ratio，可能导致过早地出现空间超限错误，OST 空间不能被全部使用，从而造成空间浪费，使 e2fsck 速度变慢。因此，请选择默认的 inode ratio，以确保索引节点的总数仍然低于这个限制。

注意：OST 文件系统检查时间受到包括索引节点数量在内等一系列变量的影响，如文件系统的大小、分配的块数量、分配块在磁盘上的分布、磁盘速度、CPU 速度、服务器上的内存数量。对于正常运行的文件系统，合理的文件系统检查时间大概在每 TiB 5-30 分钟左右，但如果检测到大量错误并需要修正，时间则会显著增加。

关于更多如何优化 MDT 和 OST 文件系统的细节，请参考**第 6.4 节，ldiskfs RAID 设备的格式化选项**。

5.4. 文件和文件系统的极限值

下表描述了当前 Lustre 的已知极限。这些极限值可能受限于 Lustre 体系结构、Linux 虚拟文件系统（Virtual File System，VFS）或虚拟内存子系统。其中少数极限值是在代码中基于测试结果定义的，可以通过修改和重新编译 Lustre 软件进行更改。在这种情况下，这些受 Lustre 代码限定的极限值，被用来测试 Lustre 软件。

名称	值	描述
最大 MDT 数量	256	单 MDS 可以承载一个或多个 MDT，可以分属不同的文件系统，也可以聚合为同一个命名空间。每个文件系统都要求使用一个独立的 MDT 来存储根目录。至多可添加 255 个 MDT 到文件系统，它们使用 DNE 远程目录或条带目录添加到文件系统命名空间中。
最大 OST 数量	8150	OST 的最大数量是一个可以在编译时改变的常量。曾有 Lustre 文件系统配置过多达 4000 个 OST。单个 OSS 节点上可配置多个 OST 目标。
最大 OST 大小	ldiskfs：1024TiB； ZFS：1024TiB	这不是一个硬性限制。配置更大的 OST 是可以的，但是大多数生产系统通常不会超过该限制，因为 Lustre 可以通过增加额外的 OST 来提升容量和性能，提高以及 I/O 总体性能，使拥塞降到最低，并且可进行并行恢复（e2fsck 或 scrub）。对于 32 比特内核，由于页面缓存限制，最大块设备大小被限制为 16TB，因此也限制了 OST 大小。**强烈**建议使用 64 比特内核运行 Lustre 客户端和服务器。
最大客户端数量	131072	客户端的最大数量是一个可以在编译时改变的常量。在生产环境中曾使用过高达 30000 个客户端。
最大单文件系统大小	2EiB 或更大	可将每个 OST 配置成最大 OST 大小，然后 将所允许的最大数量的 OST 组合成单个文件系统。

名称	值	描述
最大条带数	2000	该值受存储在磁盘上并以 RPC 请求形式发送的布局信息大小限制，但这不是协议中的硬性限制。文件系统中的 OST 数量可以超过条带数量，但本限制是**单个文件**所能条带化的 OST 数量的最大值。Lustre 2.13 引入：在 2.13 之前，基于 ldiskfs 的 MDT 在默认情况下单文件的最大条带数上限为 160 个 OST；在格式化 MDT 时使用 `mkfsoptions="-O ea_inode"`可增加该值；在格式化 MDT 后，可以使用 `tune2fs -O ea_inode`来启用这个功能。
最大条带大小	< 4GiB	在切换到下一个对象前写入到每个对象的数据量。
最小条带大小	64 KiB	由于在某些 64 位机器（如 ARM 和 POWER）上的 64KiB PAGE_SIZE 限制，条带大小的最小值被限定为 64 KiB。这样单个页面就不会被拆分到多个服务器上。这也是 Data-on-MDT 布局组件可以指定的最小值。
最大单个对象大小	ldiskfs：16TiB； ZFS：256TiB	即可以存储在单个对象中的数据量。一个对象对应一个条带。ldiskfs 的限制为 16TB，适用于单个对象。对于 ZFS，该限制来自于底层 OST 的大小。文件最多可以包含 2000 个条带，每个条带都可达到这个最大对象大小。
最大文件大小	32 比特系统：16 TiB； 64 比特 ldiskfs 系统：31.25PiB； 64 比特 ZFS 系统：8EiB	受内核内存子系统限制，在 32 位系统上的单个文件大小最大为 16TiB。在 64 位系统上，这个限制不存在。因此，如果后备文件系统可以支持足够大的对象，并且/或者是这个文件是空洞文件，则文件大小可以达到 2^63 位（8EiB）。

名称	值	描述
		单个文件最多可以有 2000 个条带，这使得 64 位 ldiskfs 系统的单个文件所消耗的容量能达到 31.25PiB。文件中可存储的实际数据量取决于文件条带所在的 OST 有多少可用空间。
单个目录下最大文件或子目录数量	ldiskfs：600M 到 3.8B 个文件； ZFS：16T 个文件	Lustre 软件使用了 ldiskfs 目录哈希代码，其目录项上限至少约为 6 亿，最终取决于，文件名长度。子目录数目的上限与常规文件相同。**在 Lustre 2.8 中引入，注意：**从 Lustre2.8 开始，可通过`lfs mkdir -c`命令把单个目录分条道多个 MDT 上，从而突破此限制，目录条带数为多少那么该目录下的最大文件或子目录数量就可以增加多少倍。 **在 Lustre 2.8 中引入，注意：**从 2.14 版本开始，ldiskfs 的`large_dir`功能默认为启用，支持目录拥有超过 10M 的条目；在 2.12 版本中，`large_dir`功能存在，但默认不启用。
文件系统上最大文件数量	ldiskfs：每 MDT 容纳 40 亿； ZFS：每 MDT 容纳 256 万亿	ldiskfs 文件系统的上限为 40 亿个索引节点。默认情况下，MDT 文件系统被格式化为每 2KB 对应一个索引节点，即每 1Ti 的 MDT 空间有 5.12 亿个索引节点。这个数值可以在 MDT 文件系统初始创建时增加。更多信息，参看**第五章，Lustre 硬件配置要求和格式化选项**。 ZFS 文件系统动态分配索引节点，因此在 MDT 上没有固定的每索引节点字节数比率，但每个索引节点消耗大约 4KiB 的镜像空间，具体数值取决于配置。

名称	值	描述
		每个新加的 MDT 可容纳的文件都可达到上述最大数量，最终取决于 MDT 的可用空间，以及文件系统中目录和文件分布在哪些 MDT 上。
最长文件名	255 字节	包括底层文件系统在内，单个文件名的最大限制为 255 字节。
最长路径名	4096 字节	受 Linux VFS 限制，最长路径名为 4096 字节。
Lustre 文件系统上当前打开的文件最大数量	无限制	Lustre 软件对打开的文件数量没有限制，但实际上，它还是受制于 MDS 上的内存大小。MDS 上没有所谓当前已打开文件的" 列表"，这些已打开文件只会链接到对应客户端的 Export。每个客户端进程最多能打开几千个文件，这取决于其`ulimit`。

5.5. 确定内存需求

本节介绍每种 Lustre 文件系统组件对内存的需求。

5.5.1. 客户端内存需求

建议客户端至少配备 2GB 内存。

5.5.2. MDS 内存需求

MDS 内存需求由以下因素决定:

- 客户端最大数量
- 目录大小
- 服务器的负载

MDS 使用的内存容量取决于系统中有多少客户端，以及它们使用的工作集中有多少文件。它主要是由在某一时刻客户端持有的 LDLM（Lustre Distributed Lock Manager）锁总数决定。客户端持有锁总数因服务器上的负载和可用内存而异。交互式客户端有时可能持有超过 1 万个锁。在 MDS 上，每个当前正在使用文件大约消耗 2KB 的内存，其

中包括 LDLM 锁和内核数据结构。与从磁盘读取数据相比，将文件数据放在缓存中可以将元数据性能提高 10 倍甚至更多。

MDS 内存需求包括:

- **文件系统元数据：**需要合理数量的 RAM 以支持文件系统元数据。文件系统的元数据量没有硬性限制，可用 RAM 越多，检索元数据就更少需要进行磁盘 I/O。
- **网络传输：**如果您使用的是 TCP 或其他使用系统内存来作为发送/接收缓冲的网络，那么也需要把这些内存需求考虑在内。
- **日志大小：**默认情况下，用于每个基于 ldiskfs 文件系统的 MDT 需要大小为 4096MB 的日志（Journal）。每个文件系统在 MDS 节点上占用相同数量的 RAM。
- **故障切换配置：**如果 MDS 节点会用于从另一个节点进行故障转移（Failover），那么每个文件系统日志所需的 RAM 应翻倍。这样当主服务器发生故障时，备份服务器才有能力处理额外的负载。

5.5.2.1. 计算 MDS 内存需求

默认情况下，文件系统日志使用 4096MB。额外的 RAM 会用于缓存更大工作集里的文件数据，通常这个工作集并不时时处于活跃状态，但应给它保温以提升访问速度。在没有锁的情况下，缓存每个文件大约需要 1.5KB 内存。

举个例子，在 MDS 上的单个 MDT，有 1024 个计算节点，12 个交互节点，工作集包含 2000 万文件的（其中同一时刻有 900 万个文件缓存在客户端上）:

操作系统开销 = 4096 MB（RHEL8）

文件系统日志 = 4096MB

1024 * 32 核客户端 * 256 个文件/核 * 2kB = 16384MB

12 个交互式客户端 * 100,000 个文件 * 2KB = 2400 MB

20,000,000 文件的工作集 * 1.5KB/文件 = 30720 MB

因此，这个 MDS 的合理配置是至少需要 60GB 内存。如果 MDS 同时还作为 DNE 的主动/主动故障转移对，那么应该至少为每个 MDS 配备 96GB 的内存。这部分额外的内存，可以在正常运行期间，缓存更多的元数据和锁，从而提高性能，这有赖于具体的工作负载。

对于包含 100 万或更多文件的目录，更多的内存大有裨益。例如，当一个客户端随机访问一个包含 1000 万个文件的单个目录时，在 MDS 上可能消耗多达 35GB 的内存。

5.5.3. OSS 内存需求

在为一个 OSS 节点规划硬件时，需要考虑到 Lustre 文件系统中几个组件的内存使用情况（如：日志、服务线程、文件系统元数据等）。另外，还需要考虑到 OSS 的读取缓存特性，这个特性在缓存数据时会消耗 OSS 节点上的内存。

除了包含上文中提到的 MDS 内存需求外，OSS 的内存要求还包括:

- **服务线程：**OSS 节点上的服务线程为每个ost_io服务线程预先分配 I/O 缓冲区，缓冲区大小等于 RPC 的最大大小，这样就不需要为每个 I/O 请求来分配又释放缓冲区。
- **OSS 读取缓存：**OSS 读取缓存提供对机械磁盘数据的只读缓存，它使用常规的 Linux 页面高速缓存来存储数据。与 Linux 操作系统中常规文件系统的缓存一样，OSS 读取缓存会尽力使用所有可用的物理内存。

由于文件访问而消耗的内存，其适用于 MDS 的计算方式也同样适用于从 OSS，但文件访问的负载一般分布在更多的 OSS 节点上，因此为 MDS 列出的锁、索引节点缓存等所需的内存容量也应该分散在这些 OSS 节点上。

由于上述这些内存需求，在确定 OSS 节点所需的最小 RAM 大小时，应采用下面的计算方式。

5.5.3.1. 计算 OSS 内存需求

一个 OSS 上，如果它连接了 8 个 OST，处理的对象数目为 MDS 上活动文件的 1/4，那么推荐的最小 RAM 容量计算如下:

Linux 内核与用户空间守护进程的内存 = 4096 MB

以太网/TCP 发送/接收缓冲区 (16 MB * 512 线程) = 8192 MB

1024 MB 日志大小 * 8 个 OST 设备 = 8192MB

每个 OST IO 线程的 16 MB 读/写操作缓存 * 512 个线程 = 8192 MB

2048 MB 文件系统读取缓存 * 8 OST = 16384 MB

1024 * 32 核客户端 * 64 个文件/核 * 2KB/文件 = 4096MB

12 个交互式客户端 * 25,000 个对象 * 2KB/对象 = 600MB

5,000,000 对象（工作集）* 1.5KB/对象 = 7500MB

因此，在没有故障切换的配置中，连接 8 个 OST 的 OSS 节点，它需要的 RAM 容量最少约为 60GB。在 OSS 上添加额外的内存，可以提高频繁访问的小文件读取性能。

在有故障切换的配置中，需要的 RAM 容量最少约为 90GB，因为在每个节点上都需要预留一部分内存。当 OSS 无需处理任何故障切换的 OST 时，额外的 RAM 将被用作读取缓存。

作为一个合理的经验法则，可以为 OSS 配备 24GB 的基础内存，加上每个 OST 额外 4GB 的内存。在故障切换配置中，每个 OST 需要配备额外 8GB 的内存。

5.6. Lustre 文件系统网络的实施

作为高性能文件系统，Lustre 文件系统会对网络产生大量的负载。因此，每个 Lustre 服务器和客户端通常都会有一个专门的网络接口，用作文件系统数据通信。通常情况下使用专用的 TCP/IP 子网，但也可使用其他网络硬件。

一个典型的 Lustre 文件系统实现通常包括:

- Lustre 服务器的高性能后端网络，通常是 InfiniBand（IB）网络。
- 一个更庞大的客户端网络。
- 连接两个网络的 Lustre 路由器。

通过在`/etc/modprobe.d/lustre.conf`配置文件中，给 Lustre 网络模块（Lustre Networking，Lnet）指定参数可以配置和管理 Lustre 网络和路由。

在准备配置 Lustre 网络时，请逐一完成以下步骤:

1. **确定有哪些机器会运行 Lustre 软件，而它们又使用哪些网络接口来进行 Lustre 文件系统通信。这些设备将形成 Lustre 网络。**

网络是一组相互直接通信的节点。Lustre 软件包含了 Lustre 网络驱动器（Lustre Network Driver，LND），以支持各类网络和硬件（完整的支持列表，请参看**第二章，理解 Lustre 网络（LNet）**）。配置网络的通用规则也适用于 Lustre 网络。例如，两个不同子网（tcp0 和 tcp1）上的两个 TCP 网络被认为是两个不同的 Lustre 网络。

2. **如果需要路由，请确定在不同网络间作为路由的节点。**如果使用多种网络类型，那么您将需要路由。任何具有合适接口的节点，都可以在不同的网络硬件类型或拓扑之间，作为 Lustre 网络（LNet）通信的路由。这些节点可以是 Lustre 服务器、客户端，或者只作为路由。LNet 可在不同的网络类型间（如从 TCP 到 InfiniBand）或跨越不同的拓扑（如桥接两个 InfiniBand 或 TCP/IP 网络）传输消息。关于如何配置路由，请参看**第九章，配置 Lustre 网络（LNet）**。

3. **确定有哪些网络接口要接入或排除在 LNet 网络内。**

如果没有明确指定，对每个特定的网络类型，LNet 要么使用第一个可用接口，要么使用预定义默认值接口。LNet 不应该使用的接口（如管理网络或 IP-over-IB）需要排除在外。

通过内核模块参数网络`networks`和`ip2nets`，可以指定使用或排除哪些网络接口。关于如何设置参数，请参看**第九章，配置 Lustre 网络（LNet）**。

4. **如果网络配置较为复杂，确定一个集群模版，以简化网络安装。**

对于大型集群，您可以通过在每个节点上的`lustre.conf`文件中配置一个统一的参数集，来配置所有节点的网络。关于网络配置的集群模版，请参看**第九章，配置 Lustre 网络（LNet）**。

注意：我们建议您使用 IP 地址而不是主机名，这样调试日志的可读性更强，而为多个网络接口调试配置也更容易。

第六章配置 Lustre 文件系统上的存储

本章介绍一些如何选择存储和文件系统选项，从而在 RAID 上获得最优性能的最佳实践。本章包含以下章节：

- 6.1 节，为 MDTs 和 OSTs 选择存储设备
- 6.2 节，可靠性的最佳实践
- 6.3 节，性能折衷
- 6.4 节，为基于 ldiskfs 的 RAID 设备设置格式化选项
- 6.5 节，将 SAN 连接至 Lustre 文件系统上

注意：强烈建议将 Lustre 文件系统的硬件存储配置为 RAID。Lustre 软件并不支持文件系统级别的冗余，因而需要 RAID 来防备磁盘故障。

6.1. 为 MDT 和 OST 选择存储设备

Lustre 体系架构允许采用任何类型的块设备作为后端存储。但这些设备的特性差别很大，尤其是在发生故障的情况下，也影响了系统配置的选择。

本节内容介绍关于后端存储的问题和建议。

6.1.1. 元数据目标（MDT）

在 MDT 上的 I/O 通常主要是数据的少量读写，因而我们建议您为 MDT 存储配置 RAID1。如果您需要的容量超过了单个磁盘，我们则建议您配置 RAID1+0 或 RAID10。

6.1.2. 对象存储服务器（OST）

通过下面的快速测算，我们可以知道如果没有其他冗余，大型集群应配置为 RAID6，而 RAID5 是不可接受的。

假设一个 2PB 文件系统 (2000 个容量为 1TB 的磁盘) 的磁盘平均故障时间（MTTF）为 1000 天。这意味着失败率的期望值是`2000/1000 = 2`个磁盘/天。如果修复带宽占磁盘带宽的 10%，那么磁盘修复时间则是`1000 GB /10 MB per sec = 100,000 seconds`，也就是大约 1 天。

而对于一个含 10 个磁盘的 RAID5，在重建的 1 天当中，相同阵列中的第二个磁盘失败的几率大约是 9/1000 或每天 1%。50 天之后，RAID5 阵列则有 50% 的几率出现双重故障，导致数据丢失。

因此，非常有必要采用 RAID6 或其他的双重奇偶校验算法来为 OST 存储提供足够的冗余。

为了获得更好的性能，我们建议您使用 4 个或 8 个数据磁盘和一个或两个奇偶磁盘来创建 RAID 阵列。相比于采用多个独立的 RAID 阵列，使用更大的 RAID 阵列将会对性能造成负面影响。

为了获得细粒度 I/O 请求的最高性能，可将存储配置为 RAID1+0，但这将增加成本、降低容量。

6.2. 可靠性的最佳实践

推荐使用 RAID 控制软件以快速检测出发生故障的磁盘，并及时将其替换，从而避免双重故障和数据丢失。推荐使用热备份磁盘，这样 RAID 重建不会有延迟。

推荐及时备份文件系统的元数据。关于备份文件系统，请参看**第十八章，备份和恢复文件系统**。

6.3. 性能折衷

在写操作的粒度不是整个条带宽度的情况下，RAID 存储控制器中的回写缓存可以极大地提高多种 RAID 阵列的写性能。不幸的是，除非 RAID 阵列配备的缓存是电池供电缓存 (这个功能只有一些价格较高的硬件 RAID 阵列才支持)，否则阵列电源中断可能会导致写入失序或者写丢失、奇偶校验损坏或/和文件系统元数据损坏，这些问题会导致数据丢失。

在支持高可用（High-Availability，HA）故障切换的配置中，在 MDS 或 OSS 上的 PCI 适配卡上配备读取或回写缓存是**不安全**的，因为这会导致节点之间的不一致，进而直接或最终导致文件系统损坏。应该避免使用这种设备，或者禁用其板载缓存。

如果回写缓存已启用，那么在阵列断电后需要对文件系统进行检查。这种情况下也可能发生数据丢失。

因此，当数据完整性极其重要时，我们建议避免使用回写缓存。请您审慎地考虑启用回写缓存是否利大于弊。

6.4. 为基于 ldiskfs 的 RAID 设备设置格式化选项

当在 RAID 设备上格式化 ldiskfs 文件系统时，确保 I/O 请求与底层 RAID 对齐大有裨益。这避免了 Lustre RPC 产生不必要的磁盘操作，从而大大降低性能。在格式化 OST 或 MDT 时，可使用`--mkfsoptions`参数以指定额外的参数项。

对于 RAID5、RAID6 或 RAID1+0 存储，在`--mkfsoptions`下指定以下参数可改进文件系统元数据的布局，确保没有把所有的分配位图都存储在单一的磁盘上：

```
-E stride = chunk_blocks
```

`chunk_blocks`变量以 4096 字节大小的块为单位，含义是在移动到下一个磁盘前，连续写入到单个磁盘的数据量。它同时也被叫做 RAID 条带大小。它对于 MDT 和 OST

两者都适用。

更多关于在格式化 MDT 和 OST 文件系统的时候，如何覆盖默认参数的信息，请参考 **5.3 节，设置 ldiskfs 文件系统的格式化选项**。

6.4.1. 为 mkfs 计算文件系统参数

为了获得最好的结果，建议采用含 5 个或 9 个磁盘的 RAID5，或含 6 个或 10 个磁盘的 RAID6。RAID 的条带宽度就是最小 I/O 的最佳粒度。理想情况下，RAID 配置应使得 1MB 的 Lustre RPC 正巧匹配单个 RAID 条带，而无需任何昂贵的" 读 -修改 -写" 流程。以下为计算`stripe_width`的公式：

```
stripe_width_blocks = chunk_blocks * number_of_data_disk = 1 MB
```

其中`number_of_data_disk`不包括 RAID 奇偶校验磁盘（RAID5 有一个奇偶校验磁盘，RAID6 有两个)。

如果 RAID 配置无法让`chunk_blocks`恰好能装进 1MB 里，那么就让`stripe_width_blocks`接近但不大于 1MB。

`stripe_width_blocks`的值必须等于`chunk_blocks * number_of_data_disks`的值。仅在使用 RAID5 或 RAID6 时，需指定*`stripe_width_blocks`*参数，RAID1+0 则不需要。

下面的参数，用于在文件系统设备（`/dev/sdc`）上运行`--reformat`时，向底层 ldiskfs 文件系统指定 RAID 配置：

```
--mkfsoptions "other_options -E stride=chunk_blocks, stripe_width=stripe_width_block"
```

例如，如果一个含 6 个磁盘的 RAID6，配置有 4 个数据和 2 个奇偶校验磁盘，那么`chunk_blocks <= 1024KB/4 = 256KB`。由于数据磁盘的数量为 2 的指数，条带宽度恰好为 1MB。

6.4.2. 外部日志的参数设置

如果您配置了 RAID 阵列并直接使用它作为 OST，那么其中就同时存储了数据和元数据。为了获得更好的性能，我们建议将 OST 日志（Journal）放在一个单独的设备上，也就是创建一个小型 RAID1 阵列，并将其作为 OST 的外部日志。

在一个典型的 Lustre 文件系统中，默认的 OST 日志大小高达 1GB，默认的 MDT 日志大小高达 4GB，这样可以高速率处理事务（Transaction），而无需阻塞在日志刷新上。此外，此外，日志在 RAM 中有副本。因此，请务必确保服务器有足够的 RAM 来保存所有日志的副本。

文件系统日志选项可以通过使用`--mkfsoptions`参数指定给`mkfs.lustre`，例

如:

```
--mkfsoptions "other_options -j -J device=/dev/mdJ"
```

想要创建一个外部日志，请在 OSS 上的每个 OST 执行以下步骤:

1. 创建一个 400MB (或更大) 的日志分区（建议使用 RAID1，在本例中，`/dev/sdb`是 RAID1 设备)。

2. 在分区上创建一个日志设备。运行:

```
[oss#] mke2fs -b 4096 -O journal_dev /dev/sdb journal_size
```

其中`journal_size`以 4096 字节块为单位。如，1GB 的日志大小为 262144。

3. 格式化 OST。

在本例中，被用作 OST 的`/dev/sdc`是 RAID6 设备，运行:

```
[oss #] mkfs.lustre --ost... \
--mkfsoptions ="-J device=/dev/sdb1" /dev/sdc
```

4. 正常挂载 OST。

6.5. 将 SAN 连接至 Lustre 文件系统上

根据您的集群规模和工作负载情况，您可能希望通过 SAN 连接至 Lustre 文件系统。在连接之前，请考虑以下因素:

- 在许多 SAN 文件系统中，客户端在更新数据块或索引节点时，会单独对数据块或索引节点进行分配和上锁。而 Lustre 文件系统的设计，则避免了这种在块和 inode 上的高度竞争。
- Lustre 文件系统具有高度可扩展性，可能拥有非常多的客户端。SAN 交换机无法扩展到大量节点，而 SAN 的单位端口成本通常高于其他网络。
- 允许客户端以 direct-to-SAN 方式接入的文件系统存在安全风险，因为存在客户端能够读取 SAN 磁盘上任何数据的潜在可能，有问题的客户端可能因多种原因破坏文件系统，如不当的文件系统软件、网络软件或其他内核软件，破损的电缆，坏掉的内存等等。随着直接访问存储的客户端数量的增加，这种风险会成倍增加。

第七章网络端口绑定的设置

本章介绍如何并发使用多个网络接口，以增加性能或/和冗余度。其中话题包括:

- 7.1. 绑定网络端口的概述

- 7.2. 相关要求
- 7.3. 网口绑定的模块参数
- 7.4. 设置绑定

注意：绑定网络端口为可选功能。

7.1. 绑定网络端口的概述

绑定，也称为链路聚合（Link Aggregation），聚合（Trunking）和端口聚合（Trunking），是一种为了增加带宽而将多个物理网络链路聚合为单个逻辑链路的方法。

Linux 发行版中提供了几种不同类型的绑定。所有这些类型的绑定都使用绑定内核模块，被称为模式（Mode）。

模式 0 到模式 3 通过使用多个接口完成负载平衡和容错。模式 4 将一组接口聚合成单个虚拟接口，而组里的所有成员共享相同的速度和双工设定。该模式的相关描述可在 IEEE spec 802.3ad 里找到，被称为模式 4 或者 802.3ad。

7.2. 相关要求

成功绑定的最基本的需求是，连接的两个端点都支持绑定。在正常情况下，非服务器的那个端点是交换机（通过交叉线连接的两个系统也可以使用绑定）。而所使用的任何交换机都必须明确处理 802.3ad 动态链路聚合（Dynamic Link Aggregation）。

同时，内核也必须配置为具备绑定支持。所有具备支持的 Lustre 内核都包含绑定功能。需要绑定的接口，其网络驱动程序必须具有`ethtool`功能，以确定从属端口的速度及其双工设置。最新的网络驱动器都具备该功能

下面的命令用于验证您的接口是否适用`ethtool`:

```
# which ethtool
/sbin/ethtool

# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
```

```
            Duplex: Full
            Port: MII
            PHYAD: 1
            Transceiver: internal
            Auto-negotiation: on
            Supports Wake-on: pumbg
            Wake-on: d
            Current message level: 0x00000001 (1)
            Link detected: yes

# ethtool eth1

Settings for eth1:
    Supported ports: [ TP MII ]
    Supported link modes:   10baseT/Half 10baseT/Full
                            100baseT/Half 100baseT/Full
    Supports auto-negotiation: Yes
    Advertised link modes:  10baseT/Half 10baseT/Full
    100baseT/Half 100baseT/Full
    Advertised auto-negotiation: Yes
    Speed: 100Mb/s
    Duplex: Full
    Port: MII
    PHYAD: 32
    Transceiver: internal
    Auto-negotiation: on
    Supports Wake-on: pumbg
    Wake-on: d
    Current message level: 0x00000007 (7)
    Link detected: yes
    To quickly check whether your kernel supports bonding, run:
    # grep ifenslave /sbin/ifup
    # which ifenslave
    /sbin/ifenslave
```

7.3. 网口绑定的模块参数

绑定模块参数控制了绑定的各个方面。

基于传输哈希策略，流出流量会被映射到不同的从属接口。建议您将xmit_hash_policy 选项设置为适合绑定的layer3+4。该策略会获得上层协议信息，如果能获得的话，然后基于这些信息生成哈希。这样流向特定网络端口的流量会跨越多个从属端口，虽然单个连接不会跨越到多个从属端口。

```
$ xmit_hash_policy=layer3+4
```

miimon 选项使得用户可以监控链路状态（该参数是以毫秒为单位的时间间隔)。它使得单接口故障变得透明，从而在单链路发生故时，避免网络发生严重的退化。一个合理的默认设定是 100 毫秒:

```
$ miimon=100
```

对于较忙碌的网络，可适当增加延时。

7.4. 设置绑定

设置绑定的步骤如下:

1. 通过创建一个配置文件来创建一个虚拟的绑定端口:

```
# vi /etc/sysconfig/network-scripts/ifcfg-bond0
```

2. 将以下内容写入文件末尾:

```
DEVICE=bond0 IPADDR=192.168.10.79 # Use the free IP Address
of your network NETWORK=192.168.10.0 NETMASK=255.255.255.0
USERCTL=no BOOTPROTO=none ONBOOT=yes
```

3. 将一个或多个从属端口加入到绑定端口里。更新 eth0 和 eth1 的配置文件（使用 VI 文本编辑器)。

a. 用 VI 文本编辑器打开eth0配置文件。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

b. 修改eth0文件，或追加以下内容:

```
DEVICE=eth0 USERCTL=no ONBOOT=yes MASTER=bond0 SLAVE=yes
BOOTPROTO=none
```

c. 用 VI 文本编辑器打开eth1配置文件。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth1
```

d. 修改 eth1 文件，或追加以下内容：

```
DEVICE=eth1 USERCTL=no ONBOOT=yes MASTER=bond0 SLAVE=yes
BOOTPROTO=none
```

4. 在配置文件/etc/modprobe.d/bond.conf 中设置绑定端口和其选项。照常启动从属端口。

```
# vi /etc/modprobe.d/bond.conf
```

a. 在文件末尾写入：

```
alias bond0 bonding options bond0 mode=balance-alb
miimon=100
```

b. 加载绑定模块：

```
# modprobe bonding # ifconfig bond0 up # ifenslave bond0
eth0 eth1
```

5. 启动或重启从属端口。

注意：必须用 modprobe 命令为每个被绑定的端口加载绑定模块。如果需要创建bond0和bond1，那么两者都必须在bond.conf中进行配置。

以下的例子来自于运行 Red Hat Enterprise Linux 的系统。可以通过允许/etc/sysconfig/networking-scripts/ifcfg-*脚本进行配置。下面的网站详细介绍了其他的配置方法，如何用 DHCP 进行绑定，以及其他的设置细节。我们强烈建议参考这个网站。

https://wiki.linuxfoundation.org/networking/bonding

6. 查看/proc/net/bonding文件，以确定绑定状态。每个绑定端口都应该对应该目录下的一个文件：

``` # cat /proc/net/bonding/bond0 Ethernet Channel Bonding Driver: v3.0.3 (March 23, 2006)

Bonding Mode: load balancing (round-robin) MII Status: up MII Polling Interval (ms): 0 Up Delay (ms): 0 Down Delay (ms): 0

Slave Interface: eth0 MII Status: up Link Failure Count: 0 Permanent HW addr: 4c: 00:10:ac:61:e0

Slave Interface: eth1 MII Status: up Link Failure Count: 0 Permanent HW addr: 00:14:2a: 7c:40:1d ```
```

7. 通过ethtool或ifconfig查看端口状态，第一个绑定的端口为bond0。

``` # ifconfig bond0 Link encap:Ethernet HWaddr 4C: 00:10:AC: 61:E0 inet addr: 192.168.10.79 Bcast: 192.168.10.255 Mask: 255.255.255.0 inet6 addr: fe80::4e00:10ff:feac: 61e0/64 Scope:Link UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1 RX packets:3091 errors:0 dropped:0 overruns:0 frame:0 TX packets:880 errors:0 dropped: 0 overruns:0 carrier:0 collisions:0 txqueuelen:0 RX bytes:314203 (306.8 KiB) TX bytes:129834 (126.7 KiB)

eth0 Link encap:Ethernet HWaddr 4C:00:10:AC:61:E0 inet6 addr: fe80::4e00:10ff:feac: 61e0/64 Scope:Link UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric: 1 RX packets:1581 errors:0 dropped:0 overruns:0 frame:0 TX packets:448 errors:0 dropped: 0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:162084 (158.2 KiB) TX bytes: 67245 (65.6 KiB) Interrupt:193 Base address:0x8c00

eth1 Link encap:Ethernet HWaddr 4C:00:10:AC:61:E0 inet6 addr: fe80::4e00:10ff:feac: 61e0/64 Scope:Link UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric: 1 RX packets:1513 errors:0 dropped:0 overruns:0 frame:0 TX packets:444 errors:0 dropped: 0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:152299 (148.7 KiB) TX bytes: 64517 (63.0 KiB) Interrupt:185 Base address:0x6000 ```

#### 7.4.1. 示例

以下例子显示了bond.conf中绑定以太网端口eth1和eth2到bond0的条目:

```
cat /etc/modprobe.d/bond.conf
alias eth0 8139too
alias eth1 via-rhine
alias bond0 bonding
options bond0 mode=balance-alb miimon=100

# cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BOOTPROTO=none
NETMASK=255.255.255.0
IPADDR=192.168.10.79 # (Assign here the IP of the bonded interface.)
ONBOOT=yes
USERCTL=no

ifcfg-ethx
```
```

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
DEVICE=eth0
HWADDR=4c:00:10:ac:61:e0
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
IPV6INIT=no
PEERDNS=yes
MASTER=bond0
SLAVE=yes
```

以下例子中，bond0为主端口（MASTER），eth0和eth1为从属端口（SLAVE）。

注意：bond0的所有从属端口的 MAC 地址（Hwaddr）相同，TLB 和 ALB 下的每个从属端口的 MAC 地址必须是唯一的，除此之外的其他所有模式共享此 MAC 地址。

```
$ /sbin/ifconfig

bond0     Link encap:EthernetHwaddr 00:C0:F0:1F:37:B4
inet addr:XXX.XXX.XXX.YYY Bcast:XXX.XXX.XXX.255 Mask:255.255.252.0
UP BROADCAST RUNNING MASTER MULTICAST MTU:1500  Metric:1
RX packets:7224794 errors:0 dropped:0 overruns:0 frame:0
TX packets:3286647 errors:1 dropped:0 overruns:1 carrier:0
collisions:0 txqueuelen:0

eth0     Link encap:EthernetHwaddr 00:C0:F0:1F:37:B4
inet addr:XXX.XXX.XXX.YYY Bcast:XXX.XXX.XXX.255 Mask:255.255.252.0
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500  Metric:1
RX packets:3573025 errors:0 dropped:0 overruns:0 frame:0
TX packets:1643167 errors:1 dropped:0 overruns:1 carrier:0
collisions:0 txqueuelen:100
Interrupt:10 Base address:0x1080

eth1     Link encap:EthernetHwaddr 00:C0:F0:1F:37:B4
inet addr:XXX.XXX.XXX.YYY Bcast:XXX.XXX.XXX.255 Mask:255.255.252.0
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500  Metric:1
RX packets:3651769 errors:0 dropped:0 overruns:0 frame:0
```

```
22 TX packets:1643480 errors:0 dropped:0 overruns:0 carrier:0
23 collisions:0 txqueuelen:100
24 Interrupt:9 Base address:0x1400
```

7.5. 为绑定配置 Lustre 文件系统

Lustre 软件使用绑定端口的 IP 地址，不需要其他额外配置。绑定端口被看做普通的 TCP/IP 端口。必要时，可通过/etc/modprobe中的 Lustrenetworks参数指定bond0：

```
options lnet networks=tcp(bond0)
```

7.6. 关于绑定的参考资料

我们推荐参考下面的参考资料：

- Linux 内核源码树中的documentation/networking/bonding.txt
- http://linux-ip.net/html/ether-bonding.html。
- http://linux-ip.net/html/ether-bonding.html。
- Linux 基金会绑定网站：https://www.linuxfoundation.org/networking/bonding。该网站包含大量文档，强烈推荐。该网站包含关于很多更复杂设置的解释，包括在绑定中使用 DHCP。

第八章 Lustre 软件系统安装

8.1. 准备安装 Lustre 软件

安装 Lustre 软件，您可以使用下载的软件包（RPM），或直接从源代码安装。本章主要介绍如何安装 Lustre RPM 软件包。从源码安装 Lustre 的介绍不在本文档的范围内，可以在其他地方找到。

可供下载的 Lustre RPM 软件包，都在当时的 Linux 商用发行版上通过了测试。关于每个版本的具体细节，请参看发布说明。

8.1.1. 需要的软件

使用 RPM 安装 Lustre 软件，需要以下安装包：

- **Lustre 服务器软件包。** 下表中列出了 Lustre2.9 EL7 服务器所需的软件包，其中，*ver* 指 Lustre 和 kernel 发行版本（例如2.9.0-1.el7），*arch* 指处理器架构（例如x86_64）。这些安装包可在Lustre Releases目录中获得。

软件包

kernel-*verlustre*.arch | *带 Lustre 补丁的 Linux 内核* | | | *（被称为 patched kernel）* | | *lustre*-ver.arch | *Lustre 软件*

- **Lustre 客户端软件包。**下表中列出了 Lustre2.9 EL7 客户端所需软件包。其中，*ver* 指 Linux 发行版本 (如`3.6.18-348.1.1.el5`) 。这些安装包可在Lustre Releases目录中获得。

软件包	说明
kmod-lustre-client-*ver.arch*	与不带 Lustre 补丁的 Linux 内核匹配的内核模块
lustre-client-*ver.arch*	客户端命令行工具
lustre-client-dkms-*ver.arch*	`kmod-lustre-client`的替代者，客户端 RPM，含动态内核模块支持 (DKMS) 。避免了每次内核更新都安装新的 RPM，但需要客户端的完整构建环境。

注意：除非安装了 DKMS 软件包，否则在 Lustre 客户端上运行的内核版本必须与正在安装的 kmod-lustre-client-*ver* 软件包版本相同。如果在客户端上运行的内核与 Lustre 不兼容，则在使用 Lustre 文件系统软件之前，必须在客户端上安装兼容的内核。

- **Lustre LNet 网络驱动器 (LND)。** 下表列出了 Lustre 软件提供的 LND。

支持的网络类型	说明
TCP	任何支持 TCP 通信的网络，包括 GigE、10GigE 和 IPoIB。
InfiniBand 网络	OpenFabrics OFED（o2ib）
gni	Gemini（Cray 公司的网络）

注意：在发行周期中，InfiniBand 和 TCP 的 Lustre LND 会经常性地被测试，其他 LND 则由各自的所有者维护。

- **高可用软件**。如必要的话，可安装第三方高可用软件。更多信息，请参考 **11.2 节 为故障切换而准备 Lustre 文件系统**。
- **可选软件包**。Lustre Releases目录中所提供的可选软件包有（不同的操作系统和平台各有不同）：
- `kernel-debuginfo`，`kernel-debuginfo-common`，`lustre-debuginfo`，`lustre-osd-ldiskfs-debuginfo`：所需软件包的调试符号和选项，用作故障发现和解决。
- `kernel-devel`：编译第三方模块（如网络驱动程序）所需的部分内核源码树。
- `kernel-firmware`：针对 Lustre 内核重新编译的标准 Red Hat Enterprise Linux 发行版。
- `kernel-headers`：安装在`/user/include`下的头文件，用于编译用户空间的与内核相关的代码。
- `lustre-source`：Lustre 软件源代码。
- `perf`、`perf-debuginfo`、`python-perf`、`python-perf-debuginfo`（推荐）：配合 Lustre 内核版本编译过的 Linux 性能分析工具。

8.1.2. 环境要求

在安装 Lustre 软件之前，请确保软件环境符合以下要求：

- （必要）**在所有客户端上使用相同的用户 ID（UID）和组 ID（GID）。**如果需要使用补充组，请参看 **41.1 节，用户/组的上行调用（upcall）**，了解有关补充用户和组缓存 upcall 的内容（`identity_upcall`）。
- （推荐）**为客户提供远程 shell 访问。**建议赋予所有集群节点通过远程 shell 访问客户端的权限，以更好地利用 Lustre 配置和监视脚本。推荐使用 pdsh（Parallel Distributed SHell），也可使用 SSH（Secure SHell）。
- （推荐）**确保客户端时钟同步。** Lustre 文件系统使用客户端时钟作为时间戳。如果客户端之间的时钟不同步，则不同客户端访问时，文件将显示不同的时间戳。时钟漂移也可能导致问题，例如，难以调试多节点问题及关联日志等依赖于时间戳的事件。我们建议您使用网络时间协议（NTP）保持客户端和服务器时钟同步。有关 NTP 的更多信息，请参阅：http://www.ntp.org。
- （推荐）**确保安全扩展**（如 Novell AppArmor 安全系统）和**网络包过滤工具**不会干扰 Lustre 正常运行。

8.2. Lustre 软件安装流程

注意：在安装 Lustre 软件前，请备份**所有**的数据。Lustre 软件包含了对内核的修改，它需要同存储设备进行交互，如果软件未正确安装、配置或管理，可能会导致安全问题和数据丢失。

安装 Lustre 软件，请参照以下步骤：

1. 核实是否满足 Lustre 安装需求。

- 硬件需求，请查看**第五章，Lustre 硬件配置要求和格式化选项**。
- 软件需求，请查看上面 **8.1 节，准备安装 Lustre 软件**。

2. 从Lustre Releases目录下载适用于您平台的`e2fsprogs`的 RPM。

3. 从Lustre Releases目录下载适用于您平台的 Lustre 服务器 RPM。

4. 在所有 Lustre 服务器（MGS、MDS、OSS）上安装服务软件包及`e2fsprogs`软件包。

a. 用`root`用户登录 Lustre 服务器。

b. 用`yum`命令安装软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 核查软件包是否正确安装：

```
# rpm -qa|egrep "lustre|wc"|sort
```

d. 重启服务器。

e. 在每个 Lustre 服务器上重复以上步骤。

5. 从Lustre Releases目录下载适用于您平台的 Lustre 客户端 RPMs。

6. Lustre 客户端上安装其客户端软件包。

注意：客户端上运行的内核版本必须与安装的 lustre-client-modules-*ver* 版本相同。否则，在安装 Lustre 客户端软件包前必须安装兼容内核。

a. 用`root`用户登录 Lustre 客户端。

b. 用`yum`命令安装软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 核查软件包是否正确安装：

```
# rpm -qa|egrep "lustre|kernel"|sort
```

d. 重启客户端。

e. 在每个 Lustre 客户端上重复以上步骤。

第九章配置 Lustre 网络（LNet）

本章介绍如何配置 Lustre 网络（Lustre Networking，LNet），包含以下章节：

- 9.1. 通过lnetctl配置 LNet
- 9.2. LNet 模块参数概述
- 9.3. 设置 Net 模块的networks参数
- 9.4. 设置 LNet 模块的ip2nets参数
- 9.5. 设置 LNet 模块的routes参数
- 9.6. 测试 LNet 配置
- 9.7. 配置路由检查器
- 9.8. LNet 选项的最佳实践

注意：配置 LNet 是可选的。如果使用lctl network up命令加载 LNet，它会使用在系统上发现的首个 TCP/IP 端口（etho0）。如果这种网络配置就足够了，那么您不需要配置 LNet。如果需要使用 Infiniband 或者多个以太网端口，您就需要配置 LNet 了。

在 Lustre 2.7 中引入：lnetctl命令可以用来在没有启动任何网络之前初始化好 LNet。通过lnetctl命令，可以在配置好 LNet 之后，追加网络端口。然而，如果 LNet 不是由lnetctl初始化的，那么需要在使用lnetctl命令管理 LNet 之前，运行lnetctl lnet configure命令。

在 Lustre 2.7 中引入：动态 LNET 配置（Dynamic LNet Configuration，DLC）还提供了 C-API，可以用程序来配置 LNet。请参看**第四十五章，LNet 配置的 C-API**。

9.1. 通过lnetctl配置 LNet

通过modprobe加载 LNet 内核模块后，可通过lnetctl工具进行初始化和配置。一般来而言，lnetctl格式如下：

```
lnetctl cmd subcmd [options]
```

该工具可管理以下配置：

- 配置/清除配置 LNet

- 增加/移除/显示网络
- 增加/移除/显示路由
- 启用/禁用路由
- 配置路由缓冲池

9.1.1. 配置 LNet

通过modprobe加载 LNet 后，可使用lnetctl工具进行配置，而无需启动模块参数中指定的网络。通过指定-all选项，lnetctl工具可用来配置模块参数中指定的网络端口。

```
lnetctl lnet configure [--all]
# --all: load NI configuration from module parameters
```

lnetctl工具也可用来清除 LNet 的配置。

```
lnetctl lnet unconfigure
```

9.1.2. 显示全局设置

使用如下lnetctl命令，显示活动的 LNet 全局设置：

```
lnetctl global show
```

如：

```
# lnetctl global show
        global:
        numa_range: 0
        max_intf: 200
        discovery: 1
        drop_asym_route: 0
```

9.1.3. 添加、删除、显示网络

在加载 LNet 内核模块后，可添加、删除、显示网络。

lnetctl net add 命令用来添加网络：

```
lnetctl net add: add a network
        --net: net name (ex tcp0)
        --if: physical interface (ex eth0)
        --peer_timeout: time to wait before declaring a peer dead
```

```
        --peer_credits: defines the max number of inflight messages
        --peer_buffer_credits: the number of buffer credits per peer
        --credits: Network Interface credits
        --cpts: CPU Partitions configured net uses
        --help: display this help text

Example:
lnetctl net add --net tcp2 --if eth0
                --peer_timeout 180 --peer_credits 8
```

注意：Lustre 2.10 中新增了基于软件的多轨功能（Software based Multi-Rail），应注意：

- `--net`：不再是独一无二的，因为可以将多个网络端口添加到同一个网络中；
- `--if`：每个端口只能向同一个网络添加一次，但可以为每个节点指定多个端口（逗号分隔），如：`eth0, eth1, eth2`。

关于通过`lnetctl net add`或/和 YAML 添加多个网络端口的方法，请参看 **16.2 节，多轨的配置**。

lnetctl net del 命令用来删除网络：

```
net del: delete a network
        --net: net name (ex tcp0)
        --if:  physical inerface (e.g. eth0)

Example:
lnetctl net del --net tcp2
```

注意：在软件多轨配置中，单独使用`--net`参数会删除整个网络及其包含的所有端口。如果要删除特定的端口，可使用`--if`配合`--net`。

lnetctl net show 命令用来显示所有或者部分配置好的网络，`--verbose`用于指定是否显示详细信息。

```
net show: show networks
        --net: net name (ex tcp0) to filter on
        --verbose: display detailed output per network

Examples:
lnetctl net show
lnetctl net show --verbose
```

```
lnetctl net show --net tcp2 --verbose
```

以下分别显示了网络配置的简略版和详细版。

```
# non-detailed show
> lnetctl net show --net tcp2
net:
    - nid: 192.168.205.130@tcp2
      status: up
      interfaces:
          0: eth3

# detailed show
> lnetctl net show --net tcp2 --verbose
net:
    - nid: 192.168.205.130@tcp2
      status: up
      interfaces:
          0: eth3
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
```

9.1.4. 手动添加、删除、显示端点

lnetctl peer add 命令用于向软件多轨配置中手动添加远程端点。配置端点时，--prim_nid选项用于指定端点的主 NID，--nid选项用于指定由逗号分割的一串 NID。在 Lustre 2.11.0 版本中加入了动态端点发现（Dynamic Peer Discovery）功能，关于这个功能，请参看 **9.1.5，端点的动态发现**。

```
peer add: add a peer
            --prim_nid: primary NID of the peer
            --nid: comma separated list of peer nids (e.g. 10.1.1.2@tcp0)
            --non_mr: if specified this interface is created as a non
                mulit-rail
            capable peer. Only one NID can be specified in this case.
```

如：

```
lnetctl peer add
    --prim_nid 10.10.10.2@tcp
    --nid 10.10.3.3@tcp1,10.4.4.5@tcp2
```

也可不指定--prim-nid的值（端点的主 NID）。这时，--nid选项中列出的第一个 NID 即为该端点的主 NID。如：

```
lnetctl peer_add --nid 10.10.10.2@tcp,10.10.3.3@tcp1,10.4.4.5@tcp2
```

YAML 也可用于配置端点：

```
peer:
            - primary nid: <key or primary nid>
            Multi-Rail: True
            peer ni:
            - nid: <nid 1>
            - nid: <nid 2>
            - nid: <nid n>
```

配合使用其他命令，lnetctl peer show命令的结果可用来收集信息，从而用于协助端点的配置或删除。

```
lnetctl peer show -v
```

输出结果的一个例子如下：

```
peer:
            - primary nid: 192.168.122.218@tcp
            Multi-Rail: True
            peer ni:
            - nid: 192.168.122.218@tcp
            state: NA
            max_ni_tx_credits: 8
            available_tx_credits: 8
            available_rtr_credits: 8
            min_rtr_credits: -1
            tx_q_num_of_buf: 0
            send_count: 6819
            recv_count: 6264
            drop_count: 0
            refcount: 1
            - nid: 192.168.122.78@tcp
```

```
            state: NA
            max_ni_tx_credits: 8
            available_tx_credits: 8
            available_rtr_credits: 8
            min_rtr_credits: -1
            tx_q_num_of_buf: 0
            send_count: 7061
            recv_count: 6273
            drop_count: 0
            refcount: 1
            - nid: 192.168.122.96@tcp
            state: NA
            max_ni_tx_credits: 8
            available_tx_credits: 8
            available_rtr_credits: 8
            min_rtr_credits: -1
            tx_q_num_of_buf: 0
            send_count: 6939
            recv_count: 6286
            drop_count: 0
            refcount: 1
```

使用lnetctl命令删除端点:

```
peer del: delete a peer
            --prim_nid: Primary NID of the peer
            --nid: comma separated list of peer nids (e.g. 10.1.1.2@tcp0)
```

prim_nid参数总是需要指定。它用来确定要删除的端点。如果只有prim_nid参数被指定，那么整个端点都会被删除。

下面是删除端点10.10.10.3@tcp的单个 NID 的例子:

```
lnetctl peer del --prim_nid 10.10.10.2@tcp --nid
10.10.10.3@tcp
```

下面是删除整个端点的例子:

```
lnetctl peer del --prim_nid 10.10.10.2@tcp
```

9.1.5. 端点的动态发现（Lustre 2.11 中引入）

9.1.5.1. 概述 动态发现（Dynamic Discovery，DD）是一种让节点能动态发现端点，而无需显式配置的功能。这对于配置多轨（Multi-Rail，MR）非常有用。在大型集群中，可能有数百个节点，在每个节点上配置一遍 MR 端点容易出错。动态发现默认为启用状态，它使用一种新的基于 LNet ping 的协议，在第一条消息中就能发现远程端点的端口。

9.1.5.2. 协议 当节点上的 LNet 被要求向某端点发送消息时，它将首先尝试 ping 该端点。对 ping 的应答包含端点的 NID，以及描述该端点支持哪些功能的比特位。动态发现功能为多轨功能增加了一个比特位。如果端点支持多轨功能，它会在对 ping 的应答设置 MR 比特位。节点收到应答后，会检查 MR 比特位，如果 MR 比特位被设置，则使用新的 PUT 消息将自己的 NID 列表推送到端点，这个消息被称为 Push Ping。在这个简短的协议完成之后，本节点和端点都将知晓彼此的端口列表。MR 算法在随后就使用相应端点的端口列表了。

如果端点不支持 MR 功能，就不会在 ping 应答中设置 MR 功能比特位。通过查看 ping 应答，节点将获悉端点不具备 MR 功能，随后就仅会使用上层协议所提供的端口发送消息。

9.1.5.3. 动态发现和用户空间配置 即使动态发现运行，仍可手动配置端点。手动的端点配置始终优先于动态发现，如果手动配置和动态发现的信息之间存在差异，则会打印一条警告信息。

9.1.5.4. 配置 动态发现在配置方面非常轻巧。只能打开或关闭，命令如下：

```
lnetctl set discovery [0 | 1]
```

查看当前discovery设置，请使用lnetctl global show命令，具体请参看**9.1.2 节，显示全局设置**。

9.1.5.5. 根据需要初始化动态发现协议 可以根据需要初始化动态发现协议，而无需等待与端点的通信：

```
lnetctl discover <peer_nid> [<peer_nid> ...]
```

9.1.6. 添加、删除和显示路由

可通过添加一组路由来指示 LNet 消息如何选择路由：

```
lnetctl route add: add a route
        --net: net name (ex tcp0) LNet message is destined to.
               The can not be a local network.
        --gateway: gateway node nid (ex 10.1.1.2@tcp) to route
```

```
                    all LNet messaged destined for the identified
                    network
        --hop: number of hops to final destination
               (1 < hops < 255)
        --priority: priority of route (0 - highest prio)

Example:
lnetctl route add --net tcp2 --gateway 192.168.205.130@tcp1 --hop 2 --prio 1
```

lnetctl命令用于删除路由：

```
lnetctl route del: delete a route
        --net: net name (ex tcp0)
        --gateway: gateway nid (ex 10.1.1.2@tcp)
```

例如：

```
lnetctl route del --net tcp2 --gateway 192.168.205.130@tcp1
```

lnetctl命令用于显示配置好的路由：

```
lnetctl route show: show routes
        --net: net name (ex tcp0) to filter on
        --gateway: gateway nid (ex 10.1.1.2@tcp) to filter on
        --hop: number of hops to final destination
               (1 < hops < 255) to filter on
        --priority: priority of route (0 - highest prio)
                    to filter on
        --verbose: display detailed output per route
```

例如：

```
# non-detailed show
lnetctl route show

# detailed show
lnetctl route show --verbose
```

使用--verbose选项可显示更详细的信息。所有的信息显示和错误提示皆为YAML 格式。以下为简略版和详细版：

```
#Non-detailed output
> lnetctl route show
```

```
route:
    - net: tcp2
      gateway: 192.168.205.130@tcp1

#detailed output
> lnetctl route show --verbose
route:
    - net: tcp2
      gateway: 192.168.205.130@tcp1
      hop: 2
      priority: 1
      state: down
```

9.1.7. 启用和禁用路由

当一个 LNet 节点被配置为路由器时，LNet 转发那些不指向自己的消息。该功能可通过以下命令启用或禁用：

```
lnetctl set routing [0 | 1]
# 0 - disable routing feature
# 1 - enable routing feature
```

9.1.8. 显示路由信息

在节点上启用路由时，会分配微型、小型和大型路由缓冲区。请参看 **34.3 节，LNet 参数调试**。相关信息可通过如下命令进行查看：

```
lnetctl routing show: show routing information

Example:
lnetctl routing show
```

输出如下：

```
> lnetctl routing show
routing:
    - cpt[0]:
          tiny:
              npages: 0
              nbuffers: 2048
```

```
            credits: 2048
            mincredits: 2048
        small:
            npages: 1
            nbuffers: 16384
            credits: 16384
            mincredits: 16384
        large:
            npages: 256
            nbuffers: 1024
            credits: 1024
            mincredits: 1024
    - enable: 1
```

9.1.9. 配置路由缓冲

配置的路由缓冲区数值指定了微型、小型和大型路由缓冲区各组中的缓冲区数量。

很多时候，需要将微型、小型和大型路由缓冲区的数量配置为默认值之外的某些值。这些值是全局值，设置后，它们将用于所有已配置的 CPU 分区。如果路由已启用，那么设置的值将立即生效。如果指定了更多的缓冲区，那么会分配缓冲区以满足配置更改；如果指定了较少的缓冲区，那么就会在缓冲区停止使用时将其释放。如果路由尚未配置，则无法更改该值。缓冲区数量在在路由关闭或打开时会重置为默认值。

`lnetctl set`命令用于设置缓冲区数目。当设置的数值比0大时，会相应地设置缓冲区数量。当该值为0时，则会将缓冲区数量重置为默认值。

```
set tiny_buffers:
      set tiny routing buffers
               VALUE must be greater than or equal to 0

set small_buffers: set small routing buffers
        VALUE must be greater than or equal to 0

set large_buffers: set large routing buffers
        VALUE must be greater than or equal to 0
```

例如：

```
> lnetctl set tiny_buffers 4096
> lnetctl set small_buffers 8192
```

```
3 > lnetctl set large_buffers 2048
```

缓冲区数量可以按下面这样重置为默认值：

```
1 > lnetctl set tiny_buffers 0
2 > lnetctl set small_buffers 0
3 > lnetctl set large_buffers 0
```

9.1.10. 非对称路由（Lustre 2.13 引入）

9.1.10.1. 概述 非对称路由（Asymmetrical Route ）是指来，自远程端点的一个消息，它通过某个路由器传输到本节点，但是本节点在向该远程端点传输消息时，却不会使用该路由器。

在调试网络时，非对称路由可能会引发问题。而允许非对称路由也会给攻击打开大门，恶意的客户端可能会通过非对称路由向服务器注入数据。

因此，可以打开 LNet 的一个检查，它能检测到所有来自非对称路由的消息，并将其丢弃。

9.1.10.2. 配置 打开或关闭非对称路由检测，可以使用如下命令：

```
1 lnetctl set drop_asym_route [0 | 1]
```

该命令以节点为单位工作。意思就是，Lustre 群集中的每个节点都可以自行决定是否接受非对称路由消息。

要想检查当前的drop_asym_route设置，可使用lnetctl global show命令。请参看 **9.1.2 节，显示全局设置**。

默认情况下，非对称路由检测处于关闭状态。

9.1.11. 引入 YAML 配置文件

对 LNet 的配置可以使用 YAML 格式描述，输入给lnetctl工具。lnetctl工具会解析 YAML 文件，然后对其中描述的各个条目，执行指定的操作。如果命令像下面那样，未定义任何操作，则执行默认操作add 。YAML 的语法将在后面章节介绍。

```
1 lnetctl import FILE.yaml
2 lnetctl import < FILE.yaml
```

lnetctl import命令提供了三个可选参数来指定要在 YAML 文件中描述的条目上执行哪个操作。

```
1 # 如果命令没有指定选项，默认执行“”操作add
2 lnetctl import --add FILE.yaml
```

```
lnetctl import --add < FILE.yaml

# 删除文件中描述的所有条目YAML
lnetctl import --del FILE.yaml
lnetctl import --del < FILE.yaml

# 显示文件中描述的所有条目YAML
lnetctl import --show FILE.yaml
lnetctl import --show < FILE.yaml
```

9.1.12. 导出 YAML 配置文件

`lnetctl export`命令用于将配置以 YAML 格式导出。

```
lnetctl export FILE.yaml
lnetctl export > FILE.yaml
```

9.1.13. 显示 LNet 流量数据信息

`lnetctl`工具可通过以下命令打印出 LNet 通信的统计信息:

```
lnetctl stats show
```

9.1.14. YAML 的语法

`lnetctl`实用程序可导入 YAML 文件，并在其中描述的条目上执行添加、删除或显示三种操作之一:。

如下述章节所述，网络、路由和路由表的 YAML 块都可定义为 YAML 序列。每个序列带一个`seq_no`字段。在返回的错误块中也会有`seq_no`字段。这样调用者可以把错误对应到导致错误发生的条目。`lnetctl`工具会尽最大努力配置 YAML 文件中定义的条目，在遇到首个错误时不会停止处理文件。

以下就是 YAML 的语法，描述了可以通过 DLC 操作的各种配置元素。不是所有 YAML 元素对每种操作（添加/删除/显示）都是必须的。系统将忽略与请求的操作无关的元素。

```
net:
   - net: <network.  Ex: tcp or o2ib>
     interfaces:
         0: <physical interface>
```

```
    detail: <This is only applicable for show command.  1 - output
        detailed info.  0 - basic output>
    tunables:
      peer_timeout: <Integer. Timeout before consider a peer dead>
      peer_credits: <Integer. Transmit credits for a peer>
      peer_buffer_credits: <Integer. Credits available for receiving
          messages>
      credits: <Integer.  Network Interface credits>
  SMP: <An array of integers of the form: "[x,y,...]", where each
  integer represents the CPT to associate the network interface
  with> seq_no: <integer.  Optional.  User generated, and is
  passed back in the YAML error block>
```

seq_no字段和详细信息字段都没有在输出中显示。

```
routing:
  - tiny: <Integer. Tiny buffers>
    small: <Integer. Small buffers>
    large: <Integer. Large buffers>
    enable: <0 - disable routing.  1 - enable routing>
    seq_no: <Integer.  Optional.  User generated, and is passed back in
        the YAML error block>
```

seq_no字段没有在输出中显示。

```
statistics:
  seq_no: <Integer. Optional.  User generated, and is passed back in the
      YAML error block>
```

seq_no字段没有在输出中显示。

```
route:
  - net: <network. Ex: tcp or o2ib>
    gateway: <nid of the gateway in the form <ip>@<net>: Ex:
        192.168.29.1@tcp>
```

```
4     hop: <an integer between 1 and 255. Optional>
5     detail: <This is only applicable for show commands.  1 - output
          detailed info.  0. basic output>
6     seq_no: <integer. Optional. User generated, and is passed back in the
          YAML error block>
```

seq_no字段和详细信息字段都没有在输出中显示。

9.2. LNet 模块参数概述

LNet 内核模块参数指定了如何配置 LNet 以配合 Lustre 运行，其中包括了应配置哪些 NIC 给 Lustre 使用，Lustre 使用哪个路由等。

LNet 的参数一般保存在/etc/modprobe.d/lustre.conf文件中。有些情况下，这些参数可能存储在/etc/modprobe.conf文件中，但自 RHEL5 和 SLES10 后，这种用法就弃置了，而使用单独的/etc/modprobe.d/lustre.conf文件，简化了 Lustre 网络配置的管理和分发。该文件包含一个或多个条目，每个条目语法如下：

```
1 options lnet parameter=value
```

要想指定 Lustre 应该使用哪个网络端口，那就要么设置networks参数，要么设置ip2nets参数（一次只能使用其中之一）：

- networks：指定 Lustre 使用的网络。
- ip2nets：列出所有全局可用的网络，每个网络对应一个 IP 地址范围。LNet 通过地址列表的匹配查找来识别本机可用的网络。

更多细节，请查看 **9.3 节，设置 LNet 模块的networks参数**和 **9.4 节，设置 LNet 模块的ip2nets参数**。

要想设置网络间的路由，那就使用：

- routes：列出网络，以及通过哪些 NID 转发到这些网络。

更多细节，请查看 **9.5 节，设置 LNet 模块的routes参数**

可通过配置路由器检查程序，启用 Lustre 节点上的路由器运行状况检测功能，从而绕过看上去死机的路由节点，然后在故障修复后重新恢复使用这些路由节点。

完整的路由模块参数，请参看**第四十三章，配置文件和模块参数**。

注意

建议您使用 IP 地址而不是主机名，以便更轻松地读取调试日志并使用多个端口调试配置。

9.2.1. 使用 Lustre 网络标识符（NID）来标识节点

Lustre 网络标识符（NID）可以唯一地标识 Lustre 网络端点，它包含节点 ID 和网络类型。NID 的格式为：

```
network_id@network_type
```

例如：

```
10.67.73.200@tcp0
10.67.75.100@o2ib
```

第一项标识了一个 TCP/IP 节点，第二项标识了一个 InfiniBand 节点。

当在客户端上运行`mount`命令时，客户端通过 MDS 的 NID 来检索配置信息。如果该 MDS 具有多个 NID，则客户端应为其本地网络选用适当的 NID。

想要确认哪个 NID 适合用在`mount`命令中，可以使用`lctl`命令。想要列出 MDS 的所有 NID，请在 MDS 上运行：

```
lctl list_nids
```

想要确认客户端是否能通过给定的 NID 连接到 MDS，可以在客户端上运行：

```
lctl which_nid MDS_NID
```

9.3. 设置 LNet 模块的`networks`参数

如果一个节点有多个网络端口，那么您通常需要为 Lustre 指定一个专用端口。可在`lustre.conf` 文件中添加一条设置 LNet 模块的`networks`参数的条目。

```
options lnet networks=comma-separated list of
    networks
```

下面的例子指定让 Lustre 节点使用一个 TCP/IP 端口和一个 InfiniBand 端口：

```
options lnet networks=tcp0(eth0),o2ib(ib0)
```

下面的例子指定让 Lustre 节点使用 TCP/IP 端口`eth1`：

```
options lnet networks=tcp0(eth1)
```

根据不同的网络设计，可能需要为 Lustre 明确指定网络的端口。例如，以下命令中，明确指定了网络`tcp0`使用端口`eth2` 、网络`tcp1`使用端口`eth3`：

```
options lnet networks=tcp0(eth2),tcp1(eth3)
```

当网络启动期间有多个端口可用时，Lustre 会根据跳数选择最佳路由。一旦网络连接建立，Lustre 将期望网络保持连接。在 Lustre 网络中，即使同一节点上有多个端口可用，网络连接在发生故障时，也不会将路由切换至另一端口。

注意：`lustre.conf` 中的 LNet 条目仅决定了本机节点用哪个名字称呼它的端口，而不影响路由决策。

9.3.1. 多地址服务器的例子

当一个具有多个 IP 地址的服务器（多地址服务器，Multihome Server）连入 Lustre 网络时，需要设置一些配置。下面的例子中，网络包含了以下节点：

- 服务器 svr1，三个 TCP 网卡（`eth0`、`eth1`和`eth2`）和一个 InfiniBand 网卡。
- 服务器 svr2，三个 TCP 网卡（`eth0`、`eth1`和`eth2`）和一个 InfiniBand 网卡。其中，端口`eth2`不用于 Lustre 网络。
- TCP 客户端，每个客户端有一个单独的 TCP 端口。
- InfiniBand 客户端，每个客户端有一个单独的 Infiniband 端口以及一个用于管理的 TCP/IP 端口。

在这个例子中，设置 `networks` 选项：

- 在每个服务器，（即`svr1`和 `svr2`）的`lustre.conf`文件中添加：

```
1 options lnet networks=tcp0(eth0),tcp1(eth1),o2ib
```

- 对于 TCP 客户端来说，第一个非回送地址的 IP 端口自动被用于`tcp0`。因此，只有一个端口的 TCP 客户端无需在`lustre.conf`文件中定义选项。
- 在 InfiniBand 客户端的`lustre.conf`文件中添加：

```
1 options lnet networks=o2ib
```

注意：在默认情况下，Lustre 将忽略回送端口（loopback，`lo0`）。然而，Lustre 不会忽略回送端口的别名 IP 地址。因此，如果您为回送端口设置了别名 IP 地址，您必须使用 LNet 的`networks`参数指定所使用的 Lustre 网络。

注意：如果服务器在同一子网上有多个网络端口，那么 Linux 内核将使用配置第一个的端口发送所有流量（受限于 Linux 而不是 Lustre）。在这种情况下，应使用网络端口绑定。更多关于网络端口绑定的信息，请参看**第七章，绑定网络端口的设置**。

9.4. 设置 LNet 模块的ip2nets参数

使用 ip2nets 选项的场景通常是，在所有服务器和客户端上使用单个全局通用的lustre.conf文件。每个节点会用其中的 IP 地址模式列表匹配本节点的本地 IP 地址，从而找到本节点的可用网络。

请注意，ip2nets选项中列出的 IP 地址模式**仅仅**用于标识每个节点应该使用哪些网络。LNet **不会**将其用于任何其他的通信目的。

例如，网络中的节点具有以下 IP 地址：

- 服务器 svr1：eth0的 IP 地址为192.168.0.2，Infiniband（o2ib）上的 IP 地址为132.6.1.2。
- 服务器 svr2：eth0的 IP 地址为192.168.0.4 ，Infiniband（o2ib）上的 IP 地址为132.6.1.4。
- TCP 客户端的 IP 地址为192.168.0.5-255。
- Infiniband 客户端 Infiniband（o2ib）上的 IP 地址为132.6.[2-3].2, .4, .6, .8。

在每个客户端和服务器的lustre.conf文件中添加：

```
1 options lnet 'ip2nets="tcp0(eth0) 192.168.0.[2,4]; \
2 tcp0 192.168.0.*; o2ib0 132.6.[1-3].[2-8/2]"'
```

ip2nets中的每一条命令相当于一条规则。

配置服务器时，LNet 条目的顺序很重要。如果一个服务器可通过多个网络访问，将使用在lustre.conf文件中指定的第一个网络。

如果svr1和svr2匹配第一条规则，则 LNet 在这些机器上将使用eth0作为tcp0。(即使svr1和svr2也匹配第二条规则，仍使用匹配第一条规则的网络)。

[2-8/2]格式表示从 2 到 8，以 2 为间隔逐步增加，即 2、4、6、8。因此，客户端132.6.3.5将找不到匹配的 o2ib 网络。

在 Lustre 2.10 中引入，注意：多轨模式弃用了内核模块中对ip2nets的解析，转而在用户空间中对ip2nets进行 IP 模式匹配，将之转换为网络端口，然后添加到系统中。

首个匹配该 IP 模式的网络端口将被添加到网络中。

如果端口和 IP 模式两者都显式指定了，那么通过匹配 IP 模式得到的端口，将同显式定义的端口进行比照。

例如，tcp(eth0)为192.168.*.3，而在系统中同时存在eth0 == 192.158.19.3和 eth1 == 192.168.3.3，则该配置将会因模式匹配与指派的端口相冲突而失败。

当出现配置不一致时，会清晰地打印警告信息。您可使用以下命令配置ip2nets：

lnetctl import < ip2nets.yaml

如：

```
ip2nets:
  - net-spec: tcp1
    interfaces:
         0: eth0
         1: eth1
    ip-range:
         0: 192.168.*.19
         1: 192.168.100.105
  - net-spec: tcp2
    interfaces:
         0: eth2
    ip-range:
         0: 192.168.*.*
```

9.5. 设置 LNet 模块的routes参数

LNet 模块的routes参数用于标识 Lustre 配置中的路由器，它们在每个 Lustre 节点的 modprobe.conf文件中进行配置。

设置routes的目的通常是用来连接相互隔离的子网，或者交叉连接两种不同类型的网络，如 tcp 和 o2ib。

LNet 的routes参数指定一个路由节点列表，以冒号分隔。每项路由包含一个网络号码，以及后面跟随的路由节点列表：

```
routes=net_type router_NID(s)
```

下面的例子指定了双向路由，TCP 客户端可以访问 IB 网络上的 Lustre 资源，IB 服务器也可以访问 TCP 网络：

```
options lnet 'ip2nets="tcp0 192.168.0.*; \
  o2ib0(ib0) 132.6.1.[1-128]"' 'routes="tcp0   132.6.1.[1-8]@o2ib0; \
  o2ib0 192.16.8.0.[1-8]@tcp0"'
```

桥接两个网络的所有 LNet 路由节点都是等效的。它们中不存在主路由节点或备用路由节点，流量负载在所有可用路由节点上均衡分布。

LNet 路由节点的数量没有上限。应该配备足够数目的路由节点，以处理所需的文件服务带宽，外加 25%的动态余量。

9.5.1. 路由配置示例

在客户端上的lustre.conf文件中添加:

```
1 lnet networks="tcp" routes="o2ib0 192.168.0.[1-8]@tcp0"
```

在服务器节点上使用:

```
1 lnet networks="tcp o2ib" forwarding=enabled
```

在 MDS 节点上使用相反的路由:

```
1 lnet networks="o2ib0" routes="tcp0 132.6.1.[1-8]@o2ib0"
```

启动路由服务运行:

```
1 modprobe lnet
2 lctl network configure
```

9.6. 测试 LNet 配置

在完成 Lustre 网络配置后，强烈建议您使用 Lustre 软件提供的 LNet Self-Test 来测试您的 LNet 配置。关于使用 LNet Self-Test 的更多信息，请参看**第三十二章，Lustre 网络性能测试（LNet Self-Test）**。

9.7. 配置路由器检查器

如果在 Lustre 配置中，不同类型的网络，如 TCP/IP 网络和 Infiniband 网络，通过路由节点进行连接，那么可以在客户端和服务器上运行路由节点检查器以监视路由器的状态。在多跳路由配置中，可以在路由节点上配置路由节点检查器，从而监视其下一跳路由节点的运行状况。

路由节点检查器的配置可通过在lustre.conf中设置 LNet 参数进行。添加的条目格式如下:

```
1 options lnet
2     router_checker_parameter=value
```

路由节点检查器的参数包括:

- live_router_check_interval：指定路由节点检查程序 ping 在线路由节点的时间间隔（以秒为单位）。默认值为 0，即不进行检查。将该值设置为 60，请输入：

```
1 options lnet live_router_check_interval=60
```

- dead_router_check_interval：指定路由节点检查程序检查死亡路由节点的时间间隔（以秒为单位）。默认值为 0，即不进行检查。将该值设置为 60，请输入：

```
1 options lnet dead_router_check_interval=60
```

- auto_down - 启用/禁用（1/0）路由节点状态的自动标记。默认值为 1。要想禁用路由器标记，请输入：

```
1 options lnet auto_down=0
```

- router_ping_timeout：指定路由节点检查器在检查在线路由节点或死亡路由节点时的超时时间。路由节点检查器分别在时间间隔为dead_router_check_interval和live_router_check_interval内给在线路由节点或死亡路由节点发送一次 ping 消息。默认值为 50。要将值设置为 60，请输入：

```
1 options lnet router_ping_timeout=60
```

注意：router_ping_timeout 与默认的 LND 超时一致。在大集群上，如果 LND 超时增加，那么router_ping_timeout也必须增加。对于较大的群集，我们建议使用较大的时间间隔。

- check_routers_before_use：指定是否在使用前检查路由节点。默认为off。如果此参数设置为on，则dead_router_check_interval的参数必须设置为正整数。

```
1 options lnet check_routers_before_use=on
```

路由节点检查器从每个路由器获得以下信息：

- 路由节点被禁用的时间点
- 路由节点被禁用的时间长度

如果router_ping_timeout时间内路由节点检查器未收到路由节点返回的消息，则我们认为该路由节点为down状态。

如果一个路由节点被标记为up状态并能够响应ping，超时将被重置。

如果路由节点成功发送了 100 个数据包，则该路由节点的发送数据包计数器的值为 100。

9.8. LNet 选项的最佳实践

对于networks、ip2nets和routes选项，请参照以下实例以避免配置错误。

9.8.1. 用引号逗号

在一些 Linux 发行版中，逗号可能需要使用单引号或双引号进行转义。在某些特殊情况下，options可能如下所示：

```
options
      lnet'networks="tcp0,elan0"'
      'routes="tcp [2,10]@elan0"'
```

添加的引号可能会造成一些 Linux 发行版的困惑。以下的消息可能提示问题与添加的引号相关：

```
lnet: Unknown parameter 'networks'
```

'Refusing connection - no matching NID'消息通常指向 LNet 模块配置错误。

9.8.2. 增加注释

在注释末尾使用分号来标记注释的结束。LNet 将会自动忽略#字符和下一个分号之间的内容。

下面是一个错误的例子。LNet 将跳过pt11 192.168.0.[92,96]语句，导致这些节点的初始化不正常，但不会打印任何错误消息。

```
options lnet ip2nets="pt10 192.168.0.[89,93]; # comment
      with semicolon BEFORE comment \ pt11 192.168.0.[92,96];
```

正确的语法应为：

```
options lnet ip2nets="pt10 192.168.0.[89,93] \
# comment with semicolon AFTER comment; \
pt11 192.168.0.[92,96] # comment
```

请不要添加过多的注释。 Linux 内核限制了模块选项中使用的字符串的长度（通常为 1KB，但不同供应商的内核可能会有所不同）。超过此限制则会导致错误，而指定的配置也可能无法被正确处理。

第十章 Lustre 文件系统配置

10.1. 配置简单的 Lustre 文件系统

通过使用 Lustre 软件提供的管理实用程序，Lustre 文件系统可被设置为各种不同的配置。以下是配置一个简单的 Lustre 文件系统（由一个 MGS/MDS 组合、一个 OSS 带两个 OST、一个客户端组成）的流程。

此配置过程假定您已完成以下任务：

- **设置并配置您的硬件**
- **下载并安装 Lustre 软件**

以下的可选步骤如必要，则应在配置 Lustre 软件前完成：

- 在用作 OSTs 或 MDTs 的块设备上设置硬件或软件 RAID。
- 在以太网接口上设置网络接口绑定。
- 设置 lnet 模块参数用于指定 Lustre Networking（LNet）的配置以配合 Lustre 文件系统工作，并测试 LNet 配置。默认情况下，LNet 将使用它在系统上发现的第一个 TCP / IP 接口。如果此网络配置足够，则不需要配置 LNet。如果您使用 InfiniBand 或多个以太网接口，则需要 LNet 配置。
- 运行基准测试脚本 `sgpdd-survey`以获得您的硬件基准性能。对您的硬件进行基准测试可以简化与 Lustre 软件无关的调试性能问题，并确保您在安装时获得最佳性能。

注意

sgpdd-survey脚本会覆盖正在测试的设备，因此必须在配置 OST 之前运行。

配置简单的 Lustre 文件系统，请完成以下步骤：

1. 在块设备上创建一个 MGS / MDT 组合文件系统。在 MDS 节点上运行：

```
mkfs.lustre --fsname= fsname --mgs --mdt --index=0
/dev/block_device
```

默认文件名 (fsname) 为 lustre。

注意

若您希望建立多个文件系统，则 MGS 应在其专用的块设备上分别进行创建。运行脚本为：

```
```
mkfs.lustre --fsname=
fsname --mgs
/dev/block_device
```
```

2. 可选增加附加的 MDTs:

```
mkfs.lustre --fsname=
fsname --mgsnode=
nid --mdt --index=1
/dev/block_device
```

注意

最多可附加 4095 个 MDTs。

3. 在块设备上装入 MGS/MDT 组合文件系统。在 MDS 节点上运行：

```
mount -t lustre /dev/block_device /mount_point
```

注意

如您已在不同块设备上创建 MGS 和 MDT，则须同时装入它们。

4. 创建 OST，在 OSS 节点上运行：

```
mkfs.lustre --fsname=
fsname --mgsnode=
```

```
3 MGS_NID --ost --index=
4 OST_index
5 /dev/block_device
```

当您创建 OST 时，块存储设备上的 ldiskfs 或 ZFS 文件系统将被格式化（正如使用本地文件系统一样）。

只要硬件或驱动程序允许，每个 OSS 可以有足够多的 OST。

每个块设备只能配置一个 OST。创建 OST 时，应使用未分区的原始块设备。

您应在格式化时指定 OST 索引编号，以便于简化从错误消息或文件条带化中的 OST 编号转换为 OSS 节点和块设备的过程。

如您使用了可从多个 OSS 节点访问的块设备，请确保一次只从一个 OSS 节点载入 OST。我们强烈建议您为这些设备启用多载保护，以避免严重的数据损坏。

注意

Lustre 软件在 Red hat 企业版的 Linux 5 和 6 上目前支持高达 128TB 大小的块设备（在其他发行版上刻支持高达 8 TB 大小的块设备）。如果设备大小仅略大于 16 TB，建议您在格式化时将文件系统大小限制为 16 TB。我们建议您不要将 DOS 分区置于 RAID 5/6 块设备之上，因为它会对性能产生负面影响。因此，更保险的做法是格式化整个磁盘。

5. 装入 OST。在创建 OST 的 OSS 节点上运行：

```
1 mount -t lustre
2 /dev/block_device
3 /mount_point
```

注意

创建附加的 OSTs，请重复步骤 4 及步骤 5 并指定下个 OST 索引编号。

6. 在客户端上装入 Lustre 文件系统，在客户端上运行：

```
1 mount -t lustre
2 MGS_node:/
3 fsname
4 /mount_point
```

注意

在附加的客户端上装入文件系统，请重复步骤 6。

如您在装入文件系统时出错，请查看客户端和所有服务器上的系统日志并检查网络配置。一个新安装系统的常见错误是 hosts.deny 或防火墙可能禁止了端口 988 的连接。

7. 通过在客户端上运行 lfs df，dd，ls 命令，确认文件系统是否成功启动并在正常工作中。

8. （可选）运行基准测试组件来验证集群中硬件层和软件层的性能。可用的工具包括：

obdfilter-survey：指向 Lustre 文件系统的存储性能。
ost-survey：对 OST 执行 I/O 操作以检测其他相同磁盘子系统之间的异常情况。

10.1.1. 简单 Lustre 配置示例

请按照此示例的步骤来完成简单的 Lustre 文件系统配置。其中，我们创建了 MGS/MDT 组合和两个 OST 以构成名为 temp 的文件系统；使用了三个块设备，一个用于 MGS/MDT 的组合节点，另两个用于 OSS 节点。以下列出了本示例中使用的通用参数以及各个节点参数：

通用参数	值	说明
MGS node	`10.2.0.1@tcp0`	MGS/MDS 组合节点
file system	`temp`	Lustre 文件系统名
network type	`TCP/IP`	Lustre 文件系统 temp 的网络类型

节点参数	值	说明
MGS/MDS 节点		
MGS/MDS node	`mdt0`	Lustre 文件系统 temp 中的 MDS
block device	`/dev/sdb`	MGS/MDS 组合节点的块设备
mount point	`/mnt/mdt`	MGS/MDS 节点块设备 mdt0 (`/dev/sdb`) 上的载入点
首个 OSS 节点		
OSS node	`oss0`	Lustre 文件系统 temp 中的首个 OSS 节点
OST	`ost0`	Lustre 文件系统 temp 中的首个 OST 节点

节点参数	值	说明
block device	`/dev/sdc`	首个 OSS 节点 (`oss0`) 的块设备
mount point	`/mnt/ost0`	oss0 节点块设备 ost0 (`/dev/sdc`) 上的载入点
第二个 OSS 节点		
OSS node	`oss1`	Lustre 文件系统 temp 中的第二个 OSS 节点
OST	`ost1`	Lustre 文件系统 temp 中的第二个 OST 节点
block device	`/dev/sdd`	第二个 OSS 节点 (`oss1`) 的块设备
mount point	`/mnt/ost1`	oss1 节点块设备 ost1 (`/dev/sdc`) 上的载入点
客户端节点		
client node	`client1`	Lustre 文件系统 temp 中的客户端
mount point	`/lustre`	客户端节点上 Lustre 文件系统 temp 的载入点

注意

为增加调试日志的可读性并更方便为多个接口调试配置，我们建议您使用 IP 地址而不是主机名。

在本例中，请完成以下步骤:

1. 在块设备上创建一个 MGS / MDT 组合文件系统。在 MDS 节点上运行:

```
[root@mds /]# mkfs.lustre --fsname=temp --mgs --mdt --index=0 /dev/sdb
```

该命令的输出为:

```
Permanent disk data:
Target:             temp-MDT0000
Index:              0
Lustre FS: temp
Mount type:         ldiskfs
Flags:              0x75
    (MDT MGS first_time update )
Persistent mount opts: errors=remount-ro,iopen_nopriv,user_xattr
Parameters: mdt.identity_upcall=/usr/sbin/l_getidentity

```

```
checking for existing Lustre data: not found
device size = 16MB
2 6 18
formatting backing filesystem ldiskfs on /dev/sdb
    target name             temp-MDTffff
    4k blocks               0
    options                 -i 4096 -I 512 -q -O dir_index,uninit_groups -F
mkfs_cmd = mkfs.ext2 -j -b 4096 -L temp-MDTffff  -i 4096 -I 512 -q -O
dir_index,uninit_groups -F /dev/sdb
Writing CONFIGS/mountdata
```

2. 在块设备上载入 MGS/MDT 组合文件系统。在 MDS 节点上运行：

```
[root@mds /]# mount -t lustre /dev/sdb  mnt/mdt
```

该命令的输出为：

```
Lustre: temp-MDT0000: new disk, initializing
Lustre: 3009:0:(lproc_mds.c:262:lprocfs_wr_identity_upcall()) temp-MDT0000:
group upcall set to /usr/sbin/l_getidentity
Lustre: temp-MDT0000.mdt: set parameter
    identity_upcall=/usr/sbin/l_getidentity
Lustre: Server temp-MDT0000 on device /dev/sdb has started
```

3. 创建并载入 ost0。

在本示例中，OSTs (`ost0` and `ost1`) 在不同 OSS (`oss0` and `oss1`) 节点上创建。

a. 在 oss0 上创建 ost0:

```
[root@oss0 /]# mkfs.lustre --fsname=temp --mgsnode=10.2.0.1@tcp0 --ost
--index=0 /dev/sdc
```

该命令的输出为：

```
 Permanent disk data:
Target:             temp-OST0000
Index:              0
```

```
Lustre FS: temp
Mount type:        ldiskfs
Flags:             0x72
 (OST first_time update)
Persistent mount opts: errors=remount-ro,extents,mballoc
Parameters: mgsnode=10.2.0.1@tcp

checking for existing Lustre data: not found
device size = 16MB
2 6 18
formatting backing filesystem ldiskfs on /dev/sdc
    target name             temp-OST0000
    4k blocks               0
    options                 -I 256 -q -O dir_index,uninit_groups -F
mkfs_cmd = mkfs.ext2 -j -b 4096 -L temp-OST0000  -I 256 -q -O
dir_index,uninit_groups -F /dev/sdc
Writing CONFIGS/mountdata
```

b. 在 OSS 上载入 ost0，在 oss0 上运行:

```
root@oss0 /] mount -t lustre /dev/sdc /mnt/ost0
```

该命令的输出为:

```
LDISKFS-fs: file extents enabled
LDISKFS-fs: mballoc enabled
Lustre: temp-OST0000: new disk, initializing
Lustre: Server temp-OST0000 on device /dev/sdb has started
```

等候一小段时间后，显示如下:

```
Lustre: temp-OST0000: received MDS connection from 10.2.0.1@tcp0
Lustre: MDS temp-MDT0000: temp-OST0000_UUID now active, resetting orphans
```

4. 创建并载入 ost1。

a. 在 oss1 上创建 ost1:

```
[root@oss1 /]# mkfs.lustre --fsname=temp --mgsnode=10.2.0.1@tcp0 \
            --ost --index=1 /dev/sdd
```

该命令的输出为:

```
 Permanent disk data:
Target:             temp-OST0001
Index:              1
Lustre FS: temp
Mount type:         ldiskfs
Flags:              0x72
 (OST first_time update)
Persistent mount opts: errors=remount-ro,extents,mballoc
Parameters: mgsnode=10.2.0.1@tcp

checking for existing Lustre data: not found
device size = 16MB
2 6 18
formatting backing filesystem ldiskfs on /dev/sdd
    target name             temp-OST0001
    4k blocks               0
    options                 -I 256 -q -O dir_index,uninit_groups -F
mkfs_cmd = mkfs.ext2 -j -b 4096 -L temp-OST0001  -I 256 -q -O
dir_index,uninit_groups -F /dev/sdc
Writing CONFIGS/mountdata
```

b. 在 OSS 上载入 ost1，在 oss1 上运行:

```
root@oss1 /] mount -t lustre /dev/sdd /mnt/ost1
```

该命令的输出为:

```
LDISKFS-fs: file extents enabled
LDISKFS-fs: mballoc enabled
Lustre: temp-OST0000: new disk, initializing
Lustre: Server temp-OST0000 on device /dev/sdb has started
```

等候一小段时间后，显示如下:

```
Lustre: temp-OST0001: received MDS connection from 10.2.0.1@tcp0
```

```
2 Lustre: MDS temp-MDT0000: temp-OST0001_UUID now active, resetting orphans
```

5. 在客户端上挂载 Lustre 文件系统。在客户端节点上运行：

```
1 root@client1 /] mount -t lustre 10.2.0.1@tcp0:/temp /lustre
```

该命令的输出为：

```
1 Lustre: Client temp-client has started
```

6. 确认文件系统已成功启动并正常工作，在客户端上运行 df，dd，ls 命令。

a. 运行lfs df -h命令：

```
1 [root@client1 /] lfs df -h
```

lfs df -h命令列出了每个 OST 和 MDT 的空间使用情况，如下所示：

```
1  UUID               bytes      Used      Available   Use%    Mounted on
2 temp-MDT0000_UUID  8.0G      400.0M       7.6G        0%      /lustre[MDT:0]
3 temp-OST0000_UUID  800.0G    400.0M     799.6G        0%      /lustre[OST:0]
4 temp-OST0001_UUID  800.0G    400.0M     799.6G        0%      /lustre[OST:1]
5 filesystem summary:  1.6T    800.0M       1.6T        0%      /lustre
```

b. 运行lfs df -ih 命令：

```
1 [root@client1 /] lfs df -ih
```

lfs df -ih命令列出了每个 OST 和 MDT 的节点使用情况，如下所示：

```
1 UUID              Inodes      IUsed       IFree   IUse%     Mounted on
2 temp-MDT0000_UUID   2.5M        32         2.5M      0%       /lustre[MDT:0]
3 temp-OST0000_UUID   5.5M        54         5.5M      0%       /lustre[OST:0]
4 temp-OST0001_UUID   5.5M        54         5.5M      0%       /lustre[OST:1]
5 filesystem summary: 2.5M        32         2.5M      0%       /lustre
```

c. 运行 dd命令：

```
[root@client1 /] cd /lustre
[root@client1 /lustre] dd if=/dev/zero of=/lustre/zero.dat bs=4M count=2
```

dd命令通过创建一个全为字符 0 的文件来验证写入功能。在此命令中，创建了一个 8MB 的文件。输出如下：

```
2+0 records in
2+0 records out
8388608 bytes (8.4 MB) copied, 0.159628 seconds, 52.6 MB/s
```

d. 运行 ls 命令：

```
[root@client1 /lustre] ls -lsah
```

ls -lsah命令列出了当前工作路徑下的所有文件及目录，如下所示：

```
total 8.0M
4.0K drwxr-xr-x  2 root root 4.0K Oct 16 15:27 .
8.0K drwxr-xr-x 25 root root 4.0K Oct 16 15:27 ..
8.0M -rw-r--r--  1 root root 8.0M Oct 16 15:27 zero.dat
```

当 Lustre 文件系统配置完成，则可投入使用。

10.2. 其他附加配置选项

这一部分我们将介绍如何扩展 Lustre 文件系统并利用 Lustre 配置实用程序更改配置。

10.2.1. 扩展 Lustre 文件系统

Lustre 文件系统可以通过添加 OST 或客户端来进行扩展。如须创建附加 OST，请参照上述步骤 3 和步骤 5 的说明。如须安装更多客户端，请为每个客户端重复执行步骤 6。

10.2.2. 更改条带化默认配置

文件布局条带类型的默认配置如下表所示：

文件布局参数	默认值	说明
stripe_size	1 MB	在移到下一个 OST 之前写入一个 OST 的数据量。
stripe_count	1	单个文件所使用的 OSTs 个数。
start_ost	-1	每个文件用于创建对象的首个 OST。默认值为 -1，允许 MDS 根据可用空间和负载平衡来选择起始索引。强烈建议不要将此参数的默认值更改为 -1 以外的值。

使用`lfs setstripe`来更改文件布局配置。

10.2.3. 使用 Lustre 配置实用程序

如须进行其他附加配置，Lustre 提供了一些实用的配置工具：

- mkfs.lustre：用于为 Lustre 服务器格式化磁盘。
- tunefs.lustre：用于在 Lustre 目标磁盘上修改配置信息。
- lct1：用于通过 ioctl 接口直接控制 Lustre 功能，允许访问各种配置、维护和调试功能。
- mount.lustre：用于启动 Lustre 客户端或目标服务器。

实用程序 lfs 可用来配置和查询有关文件的一些不同选项功能。

注意

一些示例脚本可在 Lustre 软件安装目录中找到。如您安装了 Lustre 源代码，则脚本位于 luster / tests 子目录中。利用这些脚本您可以快速设置一些简单的标准 Lustre 配置。

第十一章 Lustre 故障切换配置

11.1. 故障切换环境设置

Lustre 软件提供了在 Lustre 文件系统层面的故障切换机制，但没有提供完整的故障切换解决方案。一般来说，完整的故障切换解决方案会为失效的系统级别组件提供故障

切换功能，例如切换失效的硬件或应用，甚至切换失效的整个节点。但是 Lustre 没有提供这部分功能。诸如节点监视、故障检测和资源保护等故障切换功能必须由外部 HA 软件提供，例如 PowerMan，或由 Linux 操作系统供应商提供的开源 Corosync 和 Pacemaker 软件包。其中，Corosync 提供了检测故障的支持，Pacemaker 则在检测到故障后采取行动。

11.1.1 选择电源设备

Lustre 文件系统中的故障切换需要使用远程电源控制（Remote Power Control, RPC）机制，它具有多种配置。例如，Lustre 服务器节点可能配备了支持远程电源控制的 IPMI/BMC 设备。我们不推荐使用过去一度常见的相关软件。有关推荐的设备，请参阅 PowerMan 集群电源管理工具网站上的RPC 支持设备列表。

11.1.2 选择电源管理软件

在将 I/O 重定向到故障转移节点之前，需要验证故障节点已经关闭，Lustre 故障切换机制需要 RPC 和管理功能软件来验证这一点。这样可以避免重复在两个节点上挂载同一个服务，产生不可逆的数据损坏风险。Lustre 可使用很多不同的电源管理工具，但最常见的两个软件包是 PowerMan 和 Linux-HA（又名 STONITH）。

PowerMan 集群电源管理工具可用于集中控制 RPC 设备。它为多种 RPC 提供了原生支持，其专家级的配置简化了新设备添加操作。（最新版本的 PowerMan）

STONITH (Shoot The Other Node In The Head) 是一套电源管理工具，早在 Red Hat Enterprise Linux 6 之前就已经包含在 Linux-HA 包中。Linux-HA 对许多电源控制设备具备原生支持，具备可扩展性（使用 Expect 脚本来进行自动化控制），提供了相关软件来检测和处置故障。Red Hat Enterprise Linux 6 之后，Linux-HA 在开源社区被 Corosync 和 Pacemaker 的组合所取代。Red Hat Enterprise Linux 用户可以从 Red Hat 获得使用 CMAN 的集群管理功能。

11.1.3 选择高可用性软件

Lustre 文件系统必须设置高可用性（HA）软件以启用完整的 Lustre 故障切换解决方案。上述 HA 软件包，除了 PowerMan 之外，都同时提供了电源管理和集群管理。使用 Pacemaker 来设置故障转移，请参阅：

- Pacemaker 项目网站

- 在 Lustre 文件系统中使用 Pacemaker 详解

11.2. 为故障切换而准备 Lustre 文件系统

为使 Lustre 文件系统其具备高可用性，我们通过第三方 HA 应用程序对其进行配置和管理。每个存储目标（MGT, MGS, OST）都必须与另一个备用节点相关联，以创建故障切换对。当客户端挂载文件系统时，此配置信息由 MGS 传送给客户端。

在挂载存储目标时，其配置信息会转发 MGS。与此相关的一些规则是：

- 初次挂载目标时，MGS 从目标读取配置信息（诸如 mgt vs. ost, failnode, fsname），并将该存储目标配置到 Lustre 文件系统上。如果 MGS 是首次读取到这一挂载配置，则该节点将成为该存储目标的" 主" 节点。
- 再次挂载目标时，MGS 从目标读取当前配置，并根据需要重新配置 MGS 数据库里的目标信息

使用`mkfs.lustre`命令格式化目标时，通过`--servicenode`选项来指定目标的故障切换服务节点。在下面的示例中，文件系统 testfs 中编号为 0 的 OST 被格式化，两个服务节点被指定成该 OST 的故障切换对：

```
mkfs.lustre --reformat --ost --fsname testfs --mgsnode=192.168.10.1@o3ib \
            --index=0 --servicenode=192.168.10.7@o2ib \
            --servicenode=192.168.10.8@o2ib \
            /dev/sdb
```

可为目标指定两个以上的潜在服务节点。可在任何指定的服务节点上载入目标。

在存储目标上配置 HA 时，Lustre 软件会启用该存储目标上的重复挂载保护（Multiple Mount Protection, MMP）。MMP 可防止多个节点同时挂载，从而造成目标上的数据损坏。

如果 MGT 在格式化时被指定了多个服务节点，那么这个信息必须通过文件系统的 mount 命令传递给 Lustre 客户端。在下面的示例中，我们在客户端上执行的 mount 命令，指定了两个可为 MGT 提供服务的 MGSs 节点的 NIDs：

```
mount -t lustre 10.10.120.1@tcp1:10.10.120.2@tcp1:/testfs /lustre/testfs
```

当客户端挂载文件系统时，MGS 向客户端提供文件系统的配置信息（MDT 和 OST 的相关信息，每个目标关联的所有服务节点的 NID，以及当前载入目标的服务节点）。随后，当客户端发起目标上的数据访问时，它会尝试每个指定的服务节点的 NID，直到成功连接到目标。

第十二章 Lustre 文件系统监控配置

12.1. Lustre Changelogs

Changelogs（更新日志）功能负责记录文件系统名称空间或文件元数据变更事件。诸如文件创建，删除，重命名，属性变更等，这些修改将与目标文件标识符（FID)、父目录文件标识符、目标名称、时间戳及用户信息一起被记录下来。这些记录可用于多种用途：- 捕获最近的更改，存入归档系统。- 利用 Changelogs 条目完成文件系统镜像的变更精确复制。- 设置监测脚本，作用于某些事件或目录。- 维护一个粗略的审计跟踪（文件/目录随时间戳变化，不含用户信息)。

Changelogs 的记录类型有：

值	说明
MARK	内部记录保存
CREAT	常规文件创建
MKDIR	目录创建
HLINK	硬链接
SLINK	软链接
MKNOD	其他文件创建
UNLNK	常规文件移除
RMDIR	目录移除
RNMFM	重命名，原名称
RNMTO	重命名，目标名称
OPEN *	打开
CLOSE	关闭
LYOUT	Layout 变更
TRUNC	常规文件截短
SATTR	属性变更
XATTR	附加属性变更
HSM	HSM 相关事件

值	说明
MTIME	MTIME 变更
CTIME	CTIME 变更
ATIME *	ATIME 变更
MIGRT	文件迁移事件
FLRW	文件级别副本：文件初始写入
RESYNC	文件级别副本：文件重新同步
GXATR *	扩展属性访问（getxattr）
NOPEN *	被拒绝的文件打开

其中带 * 号的类型在默认情况下不会被记录。

Lustre 还提供了从文件标识符（FID）到文件路径，以及从文件路径到文件标识符的操作，以方便将目标文件和父目录的文件标识符映射到文件系统命名空间。

12.1.1 Changelogs 相关命令

以下是一些与 Changelogs 相关的命令：

12.1.1.1 `lctl changelog_register` 由于 Changelog 记录需要在 MDT 上占用一些空间，系统管理员必须注册 Changelogs 用户。一旦 Changlog 用户注册完成，Changelogs 功能则被打开。注册用户需指定哪些记录已经" 处理完成"，系统则将清除已处理完成的记录，直到遇到那些未被所有用户处理完成的记录。

要注册新的日志用户，请运行：

```
mds# lctl --device fsname-MDTnumber changelog_register
```

超出注册用户的设置点的 Changelogs 条目不会被清除（请参阅 lfs changelog_clear 相关信息）。

12.1.1.2 `lfs changelog` 在 MDT 上显示元数据变更（changelog 记录），请运行：

```
lfs changelog fsname-MDTnumber [startrec [endrec]]
```

可选择是否指定开始和结束记录。以下是 changelog 记录示例：

```
1 1 02MKDIR 15:15:21.977666834 2018.01.09 0x0 t=[0x200000402:0x1:0x0]
  j=mkdir.500 ef=0xf \
```

```
2 u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x1:0x0] pics
3 2 01CREAT 15:15:36.687592024 2018.01.09 0x0 t=[0x200000402:0x2:0x0]
    j=cp.500 ef=0xf \
4 u=500:500 nid=10.128.11.159@tcp p=[0x200000402:0x1:0x0] chloe.jpg
5 3 06UNLNK 15:15:41.305116815 2018.01.09 0x1 t=[0x200000402:0x2:0x0]
    j=rm.500 ef=0xf \
6 u=500:500 nid=10.128.11.159@tcp p=[0x200000402:0x1:0x0] chloe.jpg
7 4 07RMDIR 15:15:46.468790091 2018.01.09 0x1 t=[0x200000402:0x1:0x0]
    j=rmdir.500 ef=0xf \
8 u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x1:0x0] pics
```

12.1.1.3 `lfs changelog_clear` 为某个特定用户清除老的 changelog 记录（该用户不再需要的记录），请运行：

```
lfs changelog_clear mdt_name userid endrec
```

changelog_clear命令说明该用户对 endrec 之前的 Changelog 记录已经不再感兴趣，这也使得 MDT 能够释放一部分磁盘空间。当 endrec 值为 0 时，表明清除到当前最后一条记录。要运行 changelog_clear，changelog 用户必须已经通过 lctl 命令在 MDT 节点上注册。

当所有 changelog 用户处理完成了某个节点之前的记录时，记录被完全删除。

12.1.1.4 `lctl changelog_deregister` 注销 changelog 用户，请运行：

```
lctl --device mdt_device changelog_deregister userid
```

changelog_deregister cl1 在完成注销操作时，相当于快速执行了 lfs changelog_clear cl1 0 命令。

12.1.2 Changelogs 命令示例

以下是一些不同的 Changelogs 命令的示例

- 注册 Changelog 用户

为某个设备 (lustre-MDT0000) 注册一个新的 Changelog 用户：

```
mds# lctl --device lustre-MDT0000 changelog_register
lustre-MDT0000: Registered changelog userid 'cl1'
```

- 显示 Changelog 记录

在 MDT 上显示 Changelog 记录：

```
$ lfs changelog lustre-MDT0000
1 02MKDIR 15:15:21.977666834 2018.01.09 0x0 t=[0x200000402:0x1:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x1:0x0] pics
2 01CREAT 15:15:36.687592024 2018.01.09 0x0 t=[0x200000402:0x2:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000402:0x1:0x0] chloe.jpg
3 06UNLNK 15:15:41.305116815 2018.01.09 0x1 t=[0x200000402:0x2:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000402:0x1:0x0] chloe.jpg
4 07RMDIR 15:15:46.468790091 2018.01.09 0x1 t=[0x200000402:0x1:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x1:0x0] pics
```

Changelog 记录包含了如下信息：

```
rec#
operation_type(numerical/text)
timestamp
datestamp
flags
t=target_FID
ef=extended_flags
u=uid:gid
nid=client_NID
p=parent_FID
target_name
```

显示格式为：

```
rec# operation_type(numerical/text) timestamp datestamp flags t=target_FID \
ef=extended_flags u=uid:gid nid=client_NID p=parent_FID target_name
```

如：

```
2 01CREAT 15:15:36.687592024 2018.01.09 0x0 t=[0x200000402:0x2:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000402:0x1:0x0] chloe.jpg
```

- 清除 Changelog 记录

通知设备某个特定用户 (c11) 已经不需要相关记录 (3 及 3 之前的)：

```
$ lfs changelog_clear lustre-MDT0000 cl1 3
```

确认 changelog_clear 操作成功，运行 lfs changelog。我们看到只显示了 id-3 以后的条目：

```
 $ lfs changelog lustre-MDT0000
4 07RMDIR 15:15:46.468790091 2018.01.09 0x1 t=[0x200000402:0x1:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x1:0x0] pics
```

- 注销 Changelog 用户

在某个设备上 (lustre-MDR0000) 注销某个 Changelog 用户 (c11):

```
 mds# lctl --device lustre-MDT0000 changelog_deregister cl1
 lustre-MDT0000: Deregistered changelog user 'cl1'
```

注销操作清除了该用户所有 Changelog 记录。

```
$ lfs changelog lustre-MDT0000
5 00MARK  15:56:39.603643887 2018.01.09 0x0 t=[0x20001:0x0:0x0] ef=0xf \
u=500:500 nid=0@<0:0> p=[0:0x50:0xb] mdd_obd-lustre-MDT0000-0
```

注意

MARK 记录表明了 Changelog 记录状态变化。

- 显示 Changelog 索引及注册用户

显示某个设备 (lustre-MDR0000) 上的当前最大 Changelog 索引及已注册的 Changelog 用户:

```
mds# lctl get_param  mdd.lustre-MDT0000.changelog_users
mdd.lustre-MDT0000.changelog_users=current index: 8
ID    index (idle seconds)
cl2   8 (180)
```

- 显示 Changelog 掩码

在某个设备上 (lustre-MDR0000) 显示当前 Changelog 掩码:

```
mds# lctl get_param  mdd.lustre-MDT0000.changelog_mask

mdd.lustre-MDT0000.changelog_mask=
MARK CREAT MKDIR HLINK SLINK MKNOD UNLNK RMDIR RENME RNMTO CLOSE LYOUT \
TRUNC SATTR XATTR HSM MTIME CTIME MIGRT
```

- 设置 Changelog 掩码

在某个设备上 (lustre-MDR0000) 设置 Changelog 掩码:

```
mds# lctl set_param mdd.lustre-MDT0000.changelog_mask=HLINK
mdd.lustre-MDT0000.changelog_mask=HLINK
$ lfs changelog_clear lustre-MDT0000 cl1 0
$ mkdir /mnt/lustre/mydir/foo
$ cp /etc/hosts /mnt/lustre/mydir/foo/file
$ ln /mnt/lustre/mydir/foo/file /mnt/lustre/mydir/myhardlink
```

只有掩码中有的条目类型才在 Changelog 中显示：

```
$ lfs changelog lustre-MDT0000
9 03HLINK 16:06:35.291636498 2018.01.09 0x0 t=[0x200000402:0x4:0x0] ef=0xf \
u=500:500 nid=10.128.11.159@tcp p=[0x200000007:0x3:0x0] myhardlink
```

12.1.3 Changelogs 审计

Lustre Changelogs 的一个特殊用例是审计。根据其在维基百科上的定义，信息技术审计被用来评估机构的信息资产保护及合理分发信息至授权机构的能力。基本上，它根据当前访问控制策略对所有数据访问进行控制，而这一操作通常通过分析访问日志来实现。

审计可以用作考量当前安全情况的证据，但同时也是应参照遵循的准则。

Lustre Changelogs 提供了很好的审计机制，他是一个集中化的工具，易于交互。Changelogs 包含了用于审计的所有必要信息:

- 文件标识符 (FIDs) 和目标名称可用于识别活动对象。
- UID/GID 和 NID 信息可用于识别活动主体。
- 时间戳可用于读取活动时间。

12.1.3.1 启用审计功能

如果需要一个功能齐全的基于 Changelogs 的审计工具，必须启用一些额外的 Changelog 记录类型，以便能够记录诸如 OPEN，ATIME，GETXATTR 和 DENIED OPEN 等事件。请注意，启用这些记录类型可能会对性能造成一些影响。比如从文件系统的角度来看，进行读操作时，记录 OPEN 和 GETXATTR 事件将生成 Changelog 记录写入。

从审计角度来看，能够记录 OPEN 或 DENIED OPEN 等事件很重要。例如，如果使用 Lustre 文件系统在致力于生命科学的系统上存储医疗记录，则数据隐私至关重要。管理员可能需要知道某个病历有哪些医生访问或尝试访问，何时进行的访问；以及某个医生访问了哪些医疗记录。

要启用所有更改日志条目类型，请执行：

```
mds# lctl set_param mdd.lustre-MDT0000.changelog_mask=ALL
```

```
2 mdd.seb-MDT0000.changelog_mask=ALL
```

一旦所有必需的记录类型都被启用了，只需注册一个 Changelogs 用户，审计工具即可运行。

注意，通过 nodemap 条目上的 audit_mode 标志，可以控制哪些 Lustre 客户端节点可以触发文件系统访问事件在 Changelogs 生产记录。为防止某些节点（如备份，HSM 代理节点）使审计日志溢出，我们在 per-nodemap 基础上禁用审计。当 nodemap 条目中的 audit_mode 标志为 1，且 Changelogs 已被激活时，与此 nodemap 有关的客户端将能够完成文件系统访问事件的 Changelogs 记录。当设置为 0 时，无论 Changelogs 是否激活，事件都不会记录到 Changelogs 中。默认情况下，在新创建的 nodemap 条目中，audit_mode 标志被设置为 1。同时，它在' 默认'nodemap 中也被设置为 1。

为防止与 nodemap 有关的节点生成 Changelogs 条目，请执行以下操作：

```
1 mgs# lctl nodemap_modify --name nm1 --property audit_mode --value 0
```

12.1.3.2 审计功能示例

- **OPEN**

OPEN changelog 条目的格式如下：

```
1  7 10OPEN  13:38:51.510728296 2017.07.25 0x242 t=[0x200000401:0x2:0x0] \
2  ef=0x7 u=500:500 nid=10.128.11.159@tcp m=-w-
```

它包含了有关打开模式的信息，格式为m = rwx。

在某种打开模式下，针对每个 UID / GID，只要该文件没有被关闭，OPEN 条目只记录一次。这样的话，即使有一个 MPI 作业从不同的线程打开同一文件几千次，changelog 也不会溢出。它在不影响审计信息的情况下显着降低了 ChangeLog 负载。同样，对于 CLOSE 条目，也只记录每个 UID / GID 的最后一条 CLOSE。

- **GETXATTR**

GETXATTR changelog 条目的格式如下：

```
1  8 23GXATR 09:22:55.886793012 2017.07.27 0x0 t=[0x200000402:0x1:0x0] \
2  ef=0xf u=500:500 nid=10.128.11.159@tcp x=user.name0
```

它包含了被评估的附加属性名，格式为x=<xattr name>。

- **SETXATTR**

SETXATTR changelog 条目格式如下：

```
4 15XATTR 09:41:36.157333594 2018.01.10 0x0 t=[0x200000402:0x1:0x0] \
ef=0xf u=500:500 nid=10.128.11.159@tcp x=user.name0
```

它包含了被更改的附加属性名，格式为x=<xattr name>。

- **DENIED OPEN**

DENIED OPEN changelog 条目格式如下:

```
4 24NOPEN 15:45:44.947406626 2017.08.31 0x2 t=[0x200000402:0x1:0x0] \
ef=0xf u=500:500 nid=10.128.11.158@tcp m=-w-
```

它具有和常规 OPEN 条目相同的信息。为避免 changelog 溢出，DENIED OPEN 条目受速率限制: 即每个时间间隔每个用户每个文件不得超过一个条目，此时间间隔（以秒为单位，默认值为 60 秒）可通过mdd.<mdtname>.changelog_deniednextx=<xattr name>设置。

```
mds# lctl set_param mdd.lustre-MDT0000.changelog_deniednext=120
mdd.seb-MDT0000.changelog_deniednext=120
mds# lctl get_param mdd.lustre-MDT0000.changelog_deniednext
mdd.seb-MDT0000.changelog_deniednext=120
```

12.2. Lustre Jobstats

Lustre jobstats 功能为运行在 Lustre 客户端上的用户进程收集文件系统操作统计数据，并使用作业调度器为每个作业提供的唯一作业标识符（JobID），然后在服务器上进行展示。已知的能够与 jobstats 合作的作业调度程序包括：SLURM，SGE，LSF，Loadleveler，PBS，以及 Maui / MOAB。

由于 jobstats 是以不依赖于调度程序的方式实现的，因此它能够与其他调度程序一起工作，也可以通过在 *jobid_name* 中存储自定义格式的字符串，从而在不同使用作业调度程序的环境中。

12.2.1 Jobstats 如何工作

客户端上的 Lustre jobstats 代码从用户进程的环境变量中提取唯一的 JobID，并通过 I / O 操作将此 JobID 发送到服务器。服务器则负责跟踪给定 JobID 的相关操作统计信息，可通过该 ID 进行索引。

客户端上的 Lustre 设置 jobid_var，用来指定哪个环境变量来保存该进程的 JobID，任何环境变量都可以被指定。例如，当作业首次在节点上启动时，SLURM 在每个客户端上设置 SLURM_JOB_ID 环境变量，为其分配唯一的 job ID。SLURM_JOB_ID 将被该进程下启动的所有子进程继承。

通过设置jobid_var=procname_uid，Lustre 可配置生成客户端进程名称和数值的 ID 合成的 JobID。在多个客户端节点上运行同一个二进制文件时，将产生一个统一的 JobID，但无法区分该二进制文件时单个分布式进程的其中一部分，还是多个独立的进程。

(Lustre 2.8 引入)

在 Lustre 2.8 及以后的版本中，可以设置jobid_var=nodelocal，也可以设置jobid_name=name，则该客户端节点上的所有进程都将使用这个名字。如果一个客户端上一次只运行一个作业，这是很有用的，但是如果一个客户端上同时运行多个作业，则应该对每个会话使用不同的 JobID。

(Lustre 2.12 引入) 在 Lustre 2.12 及以后的版本中，通过使用一个包含格式代码的字符串，可以为jobid_name指定更复杂的 JobID 值。这些格式代码会为每个进程产生一个特定于站点或节点的 JobID 字符串。

- %e 打印可执行文件的名称
- %g 打印 group ID
- %h 打印全限定的主机名
- %H 打印简短的主机名
- %j 打印由参数jobid_var命名的进程环境变量的 JobID。
- %p 打印数值化的进程 ID
- %u 打印用户 ID

(Lustre 2.13 引入) 在 Lustre 2.13 及以后的版本中，可以通过设置jobid_this_session参数来为每个会话设置一个 JobID。该 JobID 将由这个登录会话中启动的所有进程所继承，但是每个登录会话可以有一个不同的 JobID。

所有客户端上的 jobid_var 设置不必相同。可在由 SLURM 管理的所有客户端上使用 SLURM_JOB_ID，而在未由 SLURM 管理的客户端上使用 procname_uid，如交互式登录节点。

在单个节点上不可能有不同的 jobid_var 设置，因为多个作业调度程序在一个客户端上不可能被同时激活。但对于每个进程环境，JobID 是本地变量，可以一次在单个客户端上激活具有不同 JobID 的多个作业。

12.2.2 启用/禁用 Jobstats

Jobstats 在默认下是禁用的。jobstats 的当前状态可以通过客户端上的lctl get_param jobid_var命令来查看：

```
1 $ lctl get_param jobid_var
2 jobid_var=disable
```

在 testfs 文件系统上启用 jobstats，配置为 SLURM：

```
#
lctl conf_param testfs.sys.jobid_var = SLURM_JOB_ID
```

用于启用或禁用 jobstats 的`lctl conf_param`命令应以 root 身份在 MGS 上运行。此更改具有持续性，并且会自动传播到 MDS，OSS 和客户端节点（包括每次挂载的新客户端）。

如须在客户端上临时启用 jobstats，或在节点子集上使用不同的 jobid_var（如使用不同作业调度程序的远程集群节点，以及不使用作业调度程序的交互式登录节点），请在文件系统挂载后，直接在客户端节点上执行`lctl set_param`命令。例如，在登录节点上启用 procname_uid 合成 JobID：

```
#
lctl set_param jobid_var = procname_uid
```

`lctl set_param`的设置不是永久性的，如果在 MGS 上设置全局 jobid_var 或卸载文件系统，该设置将被重置。

下表显示了由各种作业调度程序设置的环境变量。将 jobid_var 设置为相应的作业调度程序值以完成每个作业的统计信息收集。

Job Scheduler	**Environment Variable**
Simple Linux Utility for Resource Management (SLURM)	SLURM_JOB_ID
Sun Grid Engine (SGE)	JOB_ID
Load Sharing Facility (LSF)	LSB_JOBID
Loadleveler	LOADL_STEP_ID
Portable Batch Scheduler (PBS)/MAUI	PBS_JOBID
Cray Application Level Placement Scheduler (ALPS)	ALPS_APP_ID

jobid_var 有两个特殊值：disable 和 procname_uid。要禁用 jobstats，请将 jobid_var 指定为 disable：

```
#
lctl conf_param testfs.sys.jobid_var=disable
```

跟踪每个进程名称和用户标识的作业统计信息（用于调试，或当某些节点（如登录节点）上没有使用作业调度程序），请将 jobid_var 指定为 procname_uid：

```
#
```

```
lctl conf_param testfs.sys.jobid_var=procname_uid
```

12.2.3 查看 Jobstats

MDTs 采集元数据操作的统计信息，并通过 lctl get_param mdt.*.job_stats 命令对所有文件系统和任务进行评估。例如，在客户端上运行jobid_var=procname_uid:

```
# lctl get_param mdt.*.job_stats
job_stats:
- job_id:          bash.0
  snapshot_time:   1352084992
  open:            { samples:     2, unit:  reqs }
  close:           { samples:     2, unit:  reqs }
  mknod:           { samples:     0, unit:  reqs }
  link:            { samples:     0, unit:  reqs }
  unlink:          { samples:     0, unit:  reqs }
  mkdir:           { samples:     0, unit:  reqs }
  rmdir:           { samples:     0, unit:  reqs }
  rename:          { samples:     0, unit:  reqs }
  getattr:         { samples:     3, unit:  reqs }
  setattr:         { samples:     0, unit:  reqs }
  getxattr:        { samples:     0, unit:  reqs }
  setxattr:        { samples:     0, unit:  reqs }
  statfs:          { samples:     0, unit:  reqs }
  sync:            { samples:     0, unit:  reqs }
  samedir_rename:  { samples:     0, unit:  reqs }
  crossdir_rename: { samples:     0, unit:  reqs }
- job_id:          mythbackend.0
  snapshot_time:   1352084996
  open:            { samples:    72, unit:  reqs }
  close:           { samples:    73, unit:  reqs }
  mknod:           { samples:     0, unit:  reqs }
  link:            { samples:     0, unit:  reqs }
  unlink:          { samples:    22, unit:  reqs }
  mkdir:           { samples:     0, unit:  reqs }
  rmdir:           { samples:     0, unit:  reqs }
```

```
  rename:          { samples:     0, unit:  reqs }
  getattr:         { samples:   778, unit:  reqs }
  setattr:         { samples:    22, unit:  reqs }
  getxattr:        { samples:     0, unit:  reqs }
  setxattr:        { samples:     0, unit:  reqs }
  statfs:          { samples: 19840, unit:  reqs }
  sync:            { samples: 33190, unit:  reqs }
  samedir_rename:  { samples:     0, unit:  reqs }
  crossdir_rename: { samples:     0, unit:  reqs }
```

OSTs 采集数据操作的统计信息，可通过lctl get_param obdfilter.*.job_stats命令进行评估，如:

```
$ lctl get_param obdfilter.*.job_stats
obdfilter.myth-OST0000.job_stats=
job_stats:
- job_id:          mythcommflag.0
  snapshot_time:   1429714922
  read:    { samples: 974, unit: bytes, min: 4096, max: 1048576, sum:
      91530035 }
  write:   { samples:   0, unit: bytes, min:    0, max:       0, sum:
           0 }
  setattr: { samples:   0, unit:  reqs }
  punch:   { samples:   0, unit:  reqs }
  sync:    { samples:   0, unit:  reqs }
obdfilter.myth-OST0001.job_stats=
job_stats:
- job_id:          mythbackend.0
  snapshot_time:   1429715270
  read:    { samples:   0, unit: bytes, min:     0, max:      0, sum:
           0 }
  write:   { samples:   1, unit: bytes, min: 96899, max:  96899, sum:
      96899 }
  setattr: { samples:   0, unit:  reqs }
  punch:   { samples:   1, unit:  reqs }
  sync:    { samples:   0, unit:  reqs }
obdfilter.myth-OST0002.job_stats=job_stats:
```

```
obdfilter.myth-OST0003.job_stats=job_stats:
obdfilter.myth-OST0004.job_stats=
job_stats:
- job_id:          mythfrontend.500
  snapshot_time:   1429692083
  read:    { samples:   9, unit: bytes, min: 16384, max: 1048576, sum:
      4444160 }
  write:   { samples:   0, unit: bytes, min:     0, max:       0, sum:
           0 }
  setattr: { samples:   0, unit:  reqs }
  punch:   { samples:   0, unit:  reqs }
  sync:    { samples:   0, unit:  reqs }
- job_id:          mythbackend.500
  snapshot_time:   1429692129
  read:    { samples:   0, unit: bytes, min:     0, max:       0, sum:
           0 }
  write:   { samples:   1, unit: bytes, min: 56231, max:   56231, sum:
      56231 }
  setattr: { samples:   0, unit:  reqs }
  punch:   { samples:   1, unit:  reqs }
  sync:    { samples:   0, unit:  reqs }
```

12.2.4 清除 Jobstats

已收集的作业统计信息可通过写入 proc file `job_stats`进行重置。

在本地节点上清除所有作业的统计信息:

```
# lctl set_param obdfilter.*.job_stats=clear
```

清除设备 lustre-MDT0000 上的作业'bash.0' 相关统计信息:

```
# lctl set_param mdt.lustre-MDT0000.job_stats=bash.0
```

12.2.5 配置自动清理 (Auto-cleanup) 时间间隔

默认情况下，一个作业持续未激活状态超过 600 秒，这个作业的统计信息将被丢弃。可通过以下命令临时更改该时间值:

```
# lctl set_param *.*.job_cleanup_interval={max_age}
```

或永久性更改，如将其更改为 700 秒:

```
# lctl conf_param testfs.mdt.job_cleanup_interval=700
```

可将 job_cleanup_interval 设置为 0 以禁用自动清理功能。请注意，如果禁用了 Jobstats 的自动清理功能，则所有统计信息将永久保存在内存中，这可能会导致最终服务器上的所有内存都被占用。在这种情况下，任何监控工具都应该在处理各个工作统计数据时明确相关清理设置，如上所示。

12.3. Lustre 监控工具（LMT）

Lustre 监控工具（LMT）是一个基于 Python 的分布式系统，可在一个或多个 Lustre 文件系统上的服务器端节点（MDS，OSS 和门户路由器）上提供活动的顶层视图。但它不支持监视客户端。有关 LMT 的设置程序以及更多信息，请参阅：

https://github.com/chaos/lmt/wiki

12.4. CollectL

CollectL 是另一个可用于监视 Lustre 文件系统的工具。您可以在具有 MDS，OST 和客户端组合的 Lustre 系统上运行 CollectL。它所收集的数据可以连续写入记录，并在稍后显示，或转换成适合绘图的格式。

有关 CollectL 的更多信息，请参阅：

http://collectl.sourceforge.net

针对 Lustre 的相关文档，请参阅:

http://collectl.sourceforge.net/Tutorial-Lustre.html

12.5. 其他监控选项

更多可公开获得的标准工具如下：

- lltop - 集成了批量调度程序的 Lustre 负载监视器。https://github.com/jhammond/lltop
- tacc_stats -能够探析 Lustre 接口及收集统计信息的作业导向的系统监控器、分析器、可视化工具。https://github.com/jhammond/tacc_stats
- xltop - 集成了批量调度程序的连续性 Lustre 监视器 https://github.com/jhammond/xltop

您也可以自行编写一个简单的监控解决方案，用于查看分析 ipconfig 的各种报告和 Lustre 软件生成的 procfs 文件。

第十三章 Lustre 操作详解

13.1. 通过标签挂载

Lustre 文件系统名称限于 8 个字符。Lustre 已将文件系统和目标的相关信息编码到磁盘标签中，以方便通过标签进行挂载。这使得系统管理员可随意移动磁盘，而不用担心出现 SCSI 磁盘重新排序，使用错误的/dev/device 作为共享设备等问题。文件系统命名很快将尽可能做到故障安全。目前，Linux 磁盘标签限于 16 个字符。为识别文件系统中的目标，预留了 8 个字符，其余 8 个字符则为文件系统名称预留：

```
fsname-MDT0000 或者
fsname-OST0a19
```

运行以下命令，通过标签进行挂载：

```
mount -t lustre -L
file_system_label
/mount_point
```

下面是通过标签挂载的一个例子：

```
mds# mount -t lustre -L testfs-MDT0000 /mnt/mdt
```

注意

用标签进行挂载，不应使用在多路径环境中，也不应该使用在设备再创建快照时，因为在这些情况下，多个块设备具有相同的标签。

尽管文件系统名称被内部限制为 8 个字符，但实际上您可以在任何挂载点挂载客户端，因此文件系统用户并不受限于短名称。例如：

```
client# mount -t lustre mds0@tcp0:/short
/dev/long_mountpoint_name
```

13.2. 启动 Lustre

第一次启动 Lustre 文件系统时，各组件必须按照以下顺序启动：

1. 挂载 MGT。

注意

如果出现组合的 MGT/MDT，Lustre 将自动地正确完成 MGT 和 MDT 的挂载。

2. 挂载 MDT。

注意

如果出现多个 MDTs，则将它们全部挂载（Lustre 2.4 版本中引入）。

3. 挂载 OST(s).

4. 挂载客户端.

13.3. 挂载服务器

启动 Lustre 服务器操作较简单，只需运行挂载命令。Lustre 服务可以加入到 /etc/fstab中:

```
mount -t lustre
```

得到类似如下输出:

```
/dev/sda1 on /mnt/test/mdt type lustre (rw)
/dev/sda2 on /mnt/test/ost0 type lustre (rw)
192.168.0.21@tcp:/testfs on /mnt/testfs type lustre (rw)
```

在这个例子中，MDT, OST (ost0) 和文件系统 (testfs) 挂载成功。

```
LABEL=testfs-MDT0000 /mnt/test/mdt lustre defaults,_netdev,noauto 0 0
LABEL=testfs-OST0000 /mnt/test/ost0 lustre defaults,_netdev,noauto 0 0
```

通常，指定 noauto 并让高可用性（HA）程序包管理何时装载设备是比较明智的做法。如果您未使用故障转移机制，请确保在挂载 Lustre 服务器之前已启动网络连接。如果您运行的是 Red Hat Enterprise Linux，SUSE Linux Enterprise Server，Debian 等操作系统（或其他），请使用_netdev标志来确保在安装这些磁盘前网络连接已正常启动。

我们在这里通过磁盘标签进行挂载。设备的标签可以用e2label读取。如果mkfs.lustre未指定--index 选项，则刚刚格式化的 Lustre 服务器的标签可能以 FFFF 结尾，这意味着它尚未被赋值。赋值将发生在服务器首次启动时，磁盘标签随之被更新。建议您始终使用--index 选项以确保在格式化时就完成标签设置。

注意

当客户端和 OSS 位于同一节点时，客户端和 OSS 之间的内存压力可能导致死锁。

注意

在多路徑环境中请不要使用按标签装载。

13.4. 关闭文件系统

若按照以下顺序卸载所有客户端和服务器，Lustre 文件系统则将完全关闭。注意，卸载一个块设备只会让 Lustre 软件在该节点上关闭。

注意

请注意在以下命令中 -a -t lustre 不是文件系统名, 它指代的是卸载 /etc/mtab 所有条目中的 lustre 类型。

1. 卸载客户端

在每个客户端节点上，运行 umount 命令卸载该节点上的文件系统：

```
umount -a -t lustre
```

以下是在客户端节点上卸载 testfs 文件系统的例子：

```
[root@client1 ~]# mount |grep testfs
XXX.XXX.0.11@tcp:/testfs on /mnt/testfs type lustre (rw,lazystatfs)

[root@client1 ~]# umount -a -t lustre
[154523.177714] Lustre: Unmounted testfs-client
```

2. 卸载 MDT 和 MGT

在 MGS 和 MDS 节点上，运行 umount 命令：

```
umount -a -t lustre
```

以下是在组合的 MGS/MDS 上卸载 testfs 文件系统的例子：

```
[root@mds1 ~]# mount |grep lustre
/dev/sda on /mnt/mgt type lustre (ro)
/dev/sdb on /mnt/mdt type lustre (ro)

[root@mds1 ~]# umount -a -t lustre
[155263.566230] Lustre: Failing over testfs-MDT0000
[155263.775355] Lustre: server umount testfs-MDT0000 complete
[155269.843862] Lustre: server umount MGS complete
```

对于独立的 MGS 和 MDS，命令不变，但需要先在 MDS 上运行，随后在 MGS 上运行。

3. 卸载所有 OSTs

在每个 OSS 节点上，运行 umount 命令：

```
umount -a -t lustre
```

以下是卸载OSS1服务器上所有 OSTs 的 testfs 文件系统的例子：

```
[root@oss1 ~]# mount |grep lustre
/dev/sda on /mnt/ost0 type lustre (ro)
/dev/sdb on /mnt/ost1 type lustre (ro)
/dev/sdc on /mnt/ost2 type lustre (ro)

```

```
[root@oss1 ~]# umount -a -t lustre
[155336.491445] Lustre: Failing over testfs-OST0002
[155336.556752] Lustre: server umount testfs-OST0002 complete
```

13.5. 在服务器上卸载目标

关闭 lustre OST, MDT 或 MGT, 请运行 `umount /mount_point` 命令。

以下是在挂载点 `/mnt/ost0` 关闭 OST(`ost0`) `testfs` 文件系统的例子：

```
[root@oss1 ~]# umount /mnt/ost0
[  385.142264] Lustre: Failing over testfs-OST0000
[  385.210810] Lustre: server umount testfs-OST0000 complete
```

使用 `umount` 命令是一种优雅地停止服务器的方式，因为它保留了客户端的连接状态。下次启动时，服务器将重新连接客户端，然后执行恢复过程。

如果使用了强制标志 (`-f`) ，服务器则会中断所有客户端连接并停止恢复。重新启动后，服务器不会进行恢复。任何当前连接的客户端在重新连接之前都会收到 I/O 错误。

注意

如果您使用了 loop 设备，请加上 `-d` 标志，以安全地清除 loop 设备。

13.6. 为 OSTs 指定故障切换模式

在 Lustre 文件系统中，由于 OST 故障、网络故障、OST 未挂在等原因而无法访问的 OST 可以通过以下两种方式之一进行处置：

- `failout` 模式：Lustre 客户端在超时后将立即接收到错误消息，而不是一直等待 OST 恢复。
- `failover` 模式：Lustre 将等待 OST 恢复。

默认情况下，Lustre 文件系统在 OSTs 上采用`failover` 模式. 若您想采用 `failout` 模式，请通过 `--param="failover.mode=failout"` 选项进行指定：

```
oss# mkfs.lustre --fsname=
fsname --mgsnode=
mgs_NID --param=failover.mode=failout
      --ost --index=
ost_index
/dev/ost_block_device
```

在下面的例子中，在 MGS (`mds0`) `testfs`文件系统上为 OSTs 指定了 `failout` 模式。

```
oss# mkfs.lustre --fsname=testfs --mgsnode=mds0
   --param=failover.mode=failout
     --ost --index=3 /dev/sdb
```

在首次文件系统配置后，请使用 tunefs.lustre 工具进行模式更改。在下面的例子中，模式被设置为 failout：

```
$ tunefs.lustre --param failover.mode=failout
/dev/ost_device
```

注意

在运行该命令前，请卸载所有会被 failover/failout 切换所影响的 OSTs。

13.7. 处置降级 OST 磁盘阵列

Lustre 具备告知功能，可以在当外部 RAID 阵列出现性能下降（以致整体文件系统性能下降）时，及时告知 Lustre 系统。该性能下降通常是由于磁盘发生故障而未被更换，或更换了新磁盘正在重建所造成的。当 OST 处于降级状态时，MDS 将不会为其分配新对象，从而避免因 OST 降级引起全局性能下降。

每个 OST 都有一个 degraded 参数，用于指定 OST 是否在降级模式下运行。将 OST 标记为降级，请运行：

```
lctl set_param obdfilter.{OST_name}.degraded=1
```

将 OST 恢复正常模式，请运行：

```
lctl set_param obdfilter.{OST_name}.degraded=0
```

确认是否 OSTs 当前处于降级模式，请运行：

```
lctl get_param obdfilter.*.degraded
```

若 OST 因重启或其它状况被重新挂载，该标志将被重置为 0。

我们建议通过一个自动脚本来实现各个 RAID 设备状态的监控，如通过 MD-RAID 的mdadm(8) 命令以及--monitor 来标记受影响的设备处于降级状态还是已恢复状态。

13.8. 运行多个 Lustre 文件系统

在确保 NID:fsname 唯一性的情况下，Lustre 可支持多文件系统。每个文件系统在创建时都必须使用 --fsname 参数分配一个唯一的名称。如果只存在单个 MGS，则强制执行文件系统名称唯一性。如果存在多个 MGS（如每个 MDS 上都有一个 MGS），由管理员负责确保文件系统名称是唯一的。单个 MGS 和唯一的文件系统名称提供了单一的管理点，即使该文件系统尚未挂载，也可对该文件系统发出命令。

Lustre 在单个 MGS 上支持多个文件系统。由于只有一个 MGS，fsname 保证是唯一的。

Lustre 也允许多个 MGS 共存。例如，不同的 Lustre 软件版本上同时使用了多个文件系统，需要多个 MGS。在这种情况下必须格外小心，以确保文件系统名称是唯一的。在未来可能互操作的所有系统中，每个文件系统都应该有一个唯一的 fsname。

默认情况下，`mkfs.lustre` 命令将创建一个名为 `lustre`的文件系统。如须在格式化时指定不同的文件系统名称（限制为 8 个字符），请使用 `--fsname` 选项：

```
mkfs.lustre --fsname=
file_system_name
```

注意

新文件系统的 MDT、OSTs 必须使用相同的文件名 (替代设备名)。例如对于新文件系统`foo`，MDT 和两个 OSTs 将被命名为 `foo-MDT0000` ，`foo-OST0000` 和 `foo-OST0001`。

在文件系统上挂载客户端，运行：

```
client# mount -t lustre
mgsnode:
/new_fsname
/mount_point
```

在文件系统`foo` 的载入点 /mnt/foo 上挂载一个客户端，运行：

```
client# mount -t lustre mgsnode:/foo /mnt/foo
```

注意

如果客户端要挂载多个文件系统，为避免文件在不同文件系统间移动时出现问题，请在`/etc/xattr.conf` 文件中增加：`lustre.* skip`

注意

为确保新的 MDT 已被添加至现有 MGS 上，创建 MDT 时请指定：`--mdt --mgsnode=mgs_NID`.

含有两个文件系统 (`foo` and `bar`) 的 Lustre 安装如下所示，其中 MGS 节点为 `mgsnode@tcp0` ，挂载点为 `/mnt/foo` 和 `/mnt/bar`：

```
mgsnode# mkfs.lustre --mgs /dev/sda
mdtfoonode# mkfs.lustre --fsname=foo --mgsnode=mgsnode@tcp0 --mdt --index=0
/dev/sdb
ossfoonode# mkfs.lustre --fsname=foo --mgsnode=mgsnode@tcp0 --ost --index=0
/dev/sda
ossfoonode# mkfs.lustre --fsname=foo --mgsnode=mgsnode@tcp0 --ost --index=1
```

```
/dev/sdb
mdtbarnode# mkfs.lustre --fsname=bar --mgsnode=mgsnode@tcp0 --mdt --index=0
/dev/sda
ossbarnode# mkfs.lustre --fsname=bar --mgsnode=mgsnode@tcp0 --ost --index=0
/dev/sdc
ossbarnode# mkfs.lustre --fsname=bar --mgsnode=mgsnode@tcp0 --ost --index=1
/dev/sdd
```

在文件系统 foo 的挂载点/mnt/foo上挂载客户端，运行:

```
client# mount -t lustre mgsnode@tcp0:/foo /mnt/foo
```

在文件系统 bar 的挂载点 /mnt/bar 上挂载客户端，运行:

```
client# mount -t lustre mgsnode@tcp0:/bar /mnt/bar
```

13.9. 在特定 MDT 上创建子目录

Lustre 可以创建单独的目录，以及文件和子目录，以存储在特定的 MDT 上。要在一个给定的 MDT 上创建一个子目录，请使用以下命令:

```
client# lfs mkdir -i
mdt_index
/mount_point/remote_dir
```

该命令将分配子目录 remote_dir 至 MDT 索引mdt_index.

注意

管理员可以分配远程子目录来隔离 MDT。由于父级 MDT 的失败将使命名空间不可访问，建议不要在非 MDT0000 上的父目录中创建远程子目录。出于这个原因，在默认情况下，只能从 MDT0000 创建远程子目录。为放宽该限制并允许在任何 MDT 上创建远程子目录，管理员必须在 MGS 上执行以下命令:

```
mgs# lctl conf_param *fsname*.mdt.enable_remote_dir=1
```

Lustre 文件系统'scratch' 的相应命令为:

```
mgs# lctl conf_param scratch.mdt.enable_remote_dir=1
```

如须确认配置情况，请在任一 MDS 上运行:

```
mds# lctl get_param mdt.*.enable_remote_dir
```

Lustre 2.8 中，有一个新的可调参数可用于允许某个 group ID 的用户创建和删除远程目录和条带目录，即 enable_remote_dir_gid 。如果将此参数设置为'wheel' 或'admin'，则具有这两个 group ID 的用户可创建和删除远程目录和条带目录。在 MDT0000 上将此参数设置为 -1，则可永久地允许任何非 root 用户创建和删除远程和条带目录。在 MGS 上执行以下命令:

```
mgs# lctl conf_param *fsname*.mdt.enable_remote_dir_gid=-1
```

对于 Lustre 文件系统'scratch'，则须将命令扩展为：

```
mgs# lctl conf_param scratch.mdt.enable_remote_dir_gid=-1
```

确认更改，在 MDS 上运行：

```
mds# lctl get_param mdt.***.enable_remote_dir_gid
```

13.10. 在多个 MDTs 上创建条带目录

Lustre 2.8 中的 DNE 功能允许指定目录（条带目录）下的文件将它们的元数据存在不同的 MDTs 上（附加 MDTs 被添加到文件系统中时）。这样做的结果是，条带目录中的文件的元数据请求由多个 MDT 提供服务，并且元数据服务负载分布在服务给定目录的所有 MDT 上。通过在多个 MDT 上分发元数据服务负载，可超越单个 MDT 性能的限制，提高了整体性能。在该功能出现前，目录中的所有文件只能在单个 MDT 上记录它们的元数据。在 *mdt_count* MDTs 上分割目录，运行：

```
client# lfs mkdir -c
mdt_count
/mount_point/new_directory
```

鉴于条带目录比非条带化目录开销更大，该功能对于跨越多个 MDT 分发单个大目录（50k 条目以上）最有效。

（Lustre2.14 引入）

13.10.1. 通过 space/inode 创建目录

如果在创建一个新的目录时没有指定起始 MDT，则这个目录及其条带将按空间使用情况分布在 MDT 上。例如，下面将在 MDT 上创建一个目录和它的条带，以平衡空间使用：

```
lfs mkdir -c 2 <dir1>
```

另外，如果在目录上设置了默认的目录条带数，随后在<dir1>下的系统调用mkdir也会有同样的效果:

```
lfs setdirstripe -D -c 2 <dir1>
```

该策略是:

- 如果所有 MDT 上的空闲 inodes/blocks 几乎相同，即 max_inodes_avail * 84% < min_inodes_avail 和 max_blocks_avail * 84% < min_blocks_avail，那么选择 MDT roundrobin。
- 否则，在有更多空闲 inodes/blocks 的 MDT 上创建更多的子目录。

13.11. 设置及查看 Lustre 参数

以下选项可用于在 Lustre 中设置参数:

- 创建文件系统，请使用 mkfs.lustre。
- 当服务器停止运行时，请使用 tunefs.lustre。
- 当文件系统正在运行时，可用 lctl 来设置或查看 Lustre 参数。

13.11.1. 用mkfs.lustre设置可调试参数

当文件系统第一次进行格式化时，参数可通过在mkfs.lustre 命令中添加 --param 选项进行设置，如:

```
1 mds# mkfs.lustre --mdt --param="sys.timeout=50" /dev/sda
```

13.11.2. 用tunefs.lustre设置参数

当服务器 (OSS 或 MDS) 停止运行时，可通过 tunefs.lustre 命令及 --param 选项添加参数至现有文件系统，如:

```
1 oss# tunefs.lustre --param=failover.node=192.168.0.13@tcp0 /dev/sda
```

tunefs.lustre 命令添加的为附加参数，即在已有参数的基础上添加新的参数，而不是替代它们。擦除所有的已有参数并使用新的参数，运行:

```
1 mds# tunefs.lustre --erase-params --param=
2 new_parameters
```

tunefs.lustre可用于设置任何在 /proc/fs/lustre 文件中可设置的具有 OBD 设备的参数，可指定为 obdname|fsname. obdtype.proc_file_name= value。如:

```
1 mds# tunefs.lustre --param mdt.identity_upcall=NONE /dev/sda1
```

13.11.3. 用 lctl设置参数

当文件系统运行时，lctl 可用于设置参数 (临时或永久) 或报告当前参数值。临时参数在服务器或客户端未关闭时处于激活状态，永久参数在服务器和客户端重启后仍不变。

注意

lctl list_param 可列出所有可设置参数。

13.11.3.1. 设置临时参数 lctl set_param 用于设置在当前运行节点上的临时参数。这些参数将映射至/proc/{fs,sys}/{lnet,lustre}。语法如下:

```
lctl set_param [-n] [-P]
obdtype.
obdname.
proc_file_name=
value
```

如：

```
# lctl set_param osc.*.max_dirty_mb=1024
osc.myth-OST0000-osc.max_dirty_mb=32
osc.myth-OST0001-osc.max_dirty_mb=32
osc.myth-OST0002-osc.max_dirty_mb=32
osc.myth-OST0003-osc.max_dirty_mb=32
osc.myth-OST0004-osc.max_dirty_mb=32
```

13.11.3.2. 设置永久参数 `lctl conf_param` 用于设置永久参数。一般来说，`lctl conf_param` 可用于设置 `/proc/fs/lustre` 文件中所有可设置参数，语法如下：

```
obdname|fsname.
obdtype.
proc_file_name=
value)
```

以下是 `lctl conf_param` 命令的一些示例：

```
mgs# lctl conf_param testfs-MDT0000.sys.timeout=40
$ lctl conf_param testfs-MDT0000.mdt.identity_upcall=NONE
$ lctl conf_param testfs.llite.max_read_ahead_mb=16
$ lctl conf_param testfs-MDT0000.lov.stripesize=2M
$ lctl conf_param testfs-OST0000.osc.max_dirty_mb=29.15
$ lctl conf_param testfs-OST0000.ost.client_cache_seconds=15
$ lctl conf_param testfs.sys.timeout=40
```

注意

通过`lctl conf_param` 命令设置的参数是永久性的，它们被写入了位于 MGS 的文件系统配置文件中。

13.11.3.3. 用 lctl set_param -P 设置永久参数 该命令必须在 MGS 上执行。通过`lctl upcall`在每个主机上设置给定参数。这些参数将映射至`/proc/{fs,sys}/{lnet,lustre}`中的条目。lctl set_param 命令使用以下语法：

```
lctl set_param -P
obdtype.
obdname.
proc_file_name=
value
```

如：

```
# lctl set_param -P osc.*.max_dirty_mb=1024
osc.myth-OST0000-osc.max_dirty_mb=32
osc.myth-OST0001-osc.max_dirty_mb=32
osc.myth-OST0002-osc.max_dirty_mb=32
osc.myth-OST0003-osc.max_dirty_mb=32
osc.myth-OST0004-osc.max_dirty_mb=32
```

用 -d (只带 -P) 删除永久参数，语法为：

```
lctl set_param -P -d
obdtype.
obdname.
proc_file_name
```

如：

```
# lctl set_param -P -d osc.*.max_dirty_mb
```

13.11.3.4. 列出当前参数 列出所有 Lustre 或 LNet 可设置参数，运行 `lctl list_param` 命令：

```
lctl list_param [-FR]
obdtype.
obdname
```

以下参数可用于 `lctl list_param` 命令：

`-F`，可加上'/','@','=' 分别用于表示目录，符号链接，可写文件。

`-R`，递归方式列出某路径下的所有文件。

如：

```
oss# lctl list_param obdfilter.lustre-OST0000
```

13.11.3.5. 报告当前参数值 用 `lctl get_param` 命令报告当前 Lustre 参数值的语法为：

```
lctl get_param [-n]
obdtype.
obdname.
proc_file_name
```

以下示例显示了 RPC 持续服务时间：

```
oss# lctl get_param -n ost.*.ost_io.timeouts
service : cur 1 worst 30 (at 1257150393, 85d23h58m54s ago) 1 1 1 1
```

以下示例报告了在该客户端上每个 OST 用于写回缓存的预留空间：

```
client# lctl get_param osc.*.cur_grant_bytes
osc.myth-OST0000-osc-ffff8800376bdc00.cur_grant_bytes=2097152
osc.myth-OST0001-osc-ffff8800376bdc00.cur_grant_bytes=33890304
osc.myth-OST0002-osc-ffff8800376bdc00.cur_grant_bytes=35418112
osc.myth-OST0003-osc-ffff8800376bdc00.cur_grant_bytes=2097152
osc.myth-OST0004-osc-ffff8800376bdc00.cur_grant_bytes=33808384
```

13.12. 指定 NIDs 和故障切换

如果一个节点具有多个网络接口，则它可能具有多个 NIDs（网络标识符）。其他节点通过对它们进行识别，从而选择适合它们的网络接口的相应 NID。通常，NID 由一个列表指定，不同的 NID 由逗号（,）分隔。但指定故障切换节点时，NID 由冒号（:) 进行分隔，或通过重复关键字进行指定（如：`--mgsnode=` 或 `--servicenode=`）。

显示网络中所有服务器的 Lustre 文件系统配置的 NIDs ，请运行（LNet 正在运行时）：

```
lctl list_nids
```

在下面的示例中，`mds0` 和 `mds1` 被配置为组合的 MGS/MDT 故障切换对，`oss0` 和`oss1` 被配置为 OST 故障切换对. `mds0` 的以太网地址为 192.168.10.1，`mds1` 为 192.168.10.2，`oss0` 和 `oss1` 分别为 192.168.10.20，192.168.10.21。

```
mds0# mkfs.lustre --fsname=testfs --mdt --mgs \
        --servicenode=192.168.10.2@tcp0 \-
        -servicenode=192.168.10.1@tcp0 /dev/sda1
mds0# mount -t lustre /dev/sda1 /mnt/test/mdt
oss0# mkfs.lustre --fsname=testfs --servicenode=192.168.10.20@tcp0 \
        --servicenode=192.168.10.21 --ost --index=0 \
        --mgsnode=192.168.10.1@tcp0 --mgsnode=192.168.10.2@tcp0 \
```

```
8         /dev/sdb
9 oss0# mount -t lustre /dev/sdb /mnt/test/ost0
10 client# mount -t lustre 192.168.10.1@tcp0:192.168.10.2@tcp0:/testfs \
11         /mnt/testfs
12 mds0# umount /mnt/mdt
13 mds1# mount -t lustre /dev/sda1 /mnt/test/mdt
14 mds1# lctl get_param mdt.testfs-MDT0000.recovery_status
```

当多个 NIDs 被逗号分隔开时，例如：`10.67.73.200@tcp,192.168.10.1@tcp`，这两个 NIDs 指向同一个主机，Lustre 则将选择" 最好" 的那个进行交互。当一对 NIDs 被冒号分隔开时，例如：`10.67.73.200@tcp:10.67.73.201@tcp`，这两个 NIDs 指向不同的主机并被看作故障切换对（Lustre 将先尝试第一个，失败后尝试第二个）。

`mkfs.lustre` 命令下有两个选项可用来指定故障切换对。其中，`--servicenode` 选项用来指定所有服务 NIDs（包括主节点和故障切换节点）。当使用 `--servicenode` 选项时，第一个载入目标设备的服务节点将作为主服务节点，对应其它 NIDs 的节点将作为该目标设备的故障切换点。另外一个选项`--failnode`, 用于指定故障切换节点的 NIDs.

13.13. 擦除文件系统

擦除文件系统并永久性删除文件系统中的所有数据，请在目标上运行以下命令：

```
1 $ "mkfs.lustre --reformat"
```

如果您使用的是独立的 MGS，并希望在 MGS 上保留其它文件系统，请在 MDT 上为该文件系统设置`writeconf` 标志。`writeconf` 标志将导致配置日志被擦除，它们将在服务器重启时重新生成。

在 MDT 上设置 `writeconf` 标志：

1. 卸载所有使用 Lustre 文件系统的服务器及客户端，运行：

   ```
   $ umount /mnt/lustre
   ```

2. 永久性地擦除文件系统，并用另一个文件系统进行替代：

   ```
   $ mkfs.lustre --reformat --fsname spfs --mgs --mdt
   --index=0 /dev/ {mdsdev}
   ```

3. 如果您有一个独立的 MGS (并且不希望它被格式化)，则在 MDT 上运行 `mkfs.lustre`命令，并使用`--writeconf` 标志：

```
$ mkfs.lustre --reformat --writeconf --fsname spfs --mgsnode=
mgs_nid --mdt --index=0
/dev/mds_device
```

注意

如果您使用的是组合的 MGS/MDT，重新格式化 MDT 也将使 MGS 被重新格式化，所有配置信息随之丢失。您则可重新启动新的文件系统。无须对不再是新文件系统一部分的老磁盘做任何处理，只是要确保不再挂载这些老磁盘。

13.14. 回收预留磁盘空间

当前的 Lustre 系统在服务节点上内部运行 ldiskfs 文件系统。默认情况下，ldiskfs 将预留 5%的磁盘空间以避免文件系统碎片。要回收这个空间，请在 OSS 上为文件系统中的每个 OST 运行以下命令：

```
tune2fs [-m reserved_blocks_percent] /dev/
{ostdev}
```

在运行该命令前无须关闭 Lustre，运行后也无须重启 Lustre。

注意

减少空间预留会导致严重的性能下降，这是因为当 OST 文件系统被占用了 95%以上的空间时，就很难找到大面积的连续可用空间。即使空间使用率再次下降到 95%以下，性能下降仍可能持续。因此，建议您不要将预留磁盘空间设置为 5%以下。

13.15. 替换当前 OST 或 MDT

我们将在随后的系列中介绍如何将当前 OST 内容复制到新的 OST 中，以及如何移除 MDT.

13.16. 识别 OST 对象隶属于哪个 Lustre 文件

识别包含指定 OST 上指定对象的文件，请参照以下程序：

1. 在 OST 上 (根用户)，运行 `debugfs` 显示与该目标相关文件的 FID（文件标识符)。

例如，如果目标为 `/dev/lustre/ost_test2`上的 `34976` ，调试命令为：

```
# debugfs -c -R "stat /O/0/d$((34976 % 32))/34976"
/dev/lustre/ost_test2
```

输出为：

```
debugfs 1.45.6.wc1 (20-Mar-2020)
```

```
2  /dev/lustre/ost_test2: catastrophic mode - not reading inode or group
       bitmaps
3  Inode: 352365   Type: regular    Mode:  0666   Flags: 0x80000
4  Generation: 2393149953    Version: 0x0000002a:00005f81
5  User:  1000   Group:  1000   Size: 260096
6  File ACL: 0    Directory ACL: 0
7  Links: 1   Blockcount: 512
8  Fragment:  Address: 0    Number: 0    Size: 0
9  ctime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009
10 atime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009
11 mtime: 0x4a216b48:00000000 -- Sat May 30 13:22:16 2009
12 crtime: 0x4a216b3c:975870dc -- Sat May 30 13:22:04 2009
13 Size of extra inode fields: 24
14 Extended attributes stored in inode body:
15   fid = "b9 da 24 00 00 00 00 00 6a fa 0d 3f 01 00 00 00 eb 5b 0b 00 00
         00 0000
16 00 00 00 00 00 00 00 00 " (32)
17   fid: objid=34976 seq=0 parent=[0x200000400:0x122:0x0] stripe=1
18 EXTENTS:
19 (0-64):4620544-4620607
```

2. 父类 FID 的格式为 [0x200000400:0x122:0x0]，可通过在任意 Lustre 客户端上运行 `lfs fid2path [0x200000404:0x122:0x0] /mnt/lustre` 进行直接解析。

3. 在升级 1.x inode 的示例中 (如果 FID 第一部分小于 0x200000400)，MDT 节点序号为 0x24dab9 ，生成序号为0x3f0dfa6a，路徑名通过 debugfs进行解析。

4. 在 MDS 上 (根用户)，运行 debugfs 来查找与该索引节点相关的文件：

```
# debugfs -c -R "ncheck 0x24dab9" /dev/lustre/mdt_test
```

输出为：

```
1  debugfs 1.42.3.wc3 (15-Aug-2012)
2  /dev/lustre/mdt_test: catastrophic mode - not reading inode or group
       bitmap\
3  s
4  Inode      Pathname
5  2415289
       /ROOT/brian-laptop-guest/clients/client11/~dmtmp/PWRPNT/ZD16.BMP
```

该命令列出了与给定目标相关的索引节点及路经名。

注意

Debugfs "ncheck" 是一种暴力搜索，可能需要花很长时间。

第十四章 Lustre 的日常维护

这一章主要介绍了 Lustre 文件系统完成设置和运行后的基础维护任务。

14.1. 非活动 OSTs 相关操作

在客户端或 MDT 上挂载一个或多个非活动 OSTs，请运行类似下面的命令：

```
client# mount -o exclude=testfs-OST0000 -t lustre \
          uml1:/testfs /mnt/testfs
           client# lctl get_param lov.testfs-clilov-*.target_obd
```

想要在一个活跃客户端或 MDT 上激活 OST ，请在 OSC 设备上运行 lctl activate 命令，如：

```
lctl --device 7 activate
```

注意

也可使用用冒号分隔的列表进行指定，如：exclude=testfs-OST0000:testfs-OST0001。

14.2. 查看 Lustre 文件系统所有节点

有时，您可能希望查找 Lustre 文件系统所有节点并获取所有 OST 的名称。

想要查看所有 Lustre 节点，请在 MGS 上运行：

```
# lctl get_param mgs.MGS.live.*
```

注意

该命令必须在 MGS 上运行。

在下面的例子中，testfs 文件系统有三个节点：testfs-MDT0000, testfs-OST0000, testfs-OST0001。

```
mgs:/root# lctl get_param mgs.MGS.live.*
               fsname: testfs
               flags: 0x0     gen: 26
               testfs-MDT0000
               testfs-OST0000
               testfs-OST0001
```

想要获取所有 OSTs 名称，请在 MDS 上运行：

```
1 mds:/root# lctl get_param lov.*-mdtlov.target_obd
```

注意

该命令必须在 MGS 上运行。

在下面的例子中，共有两个 OSTs：testfs-OST0000 和 testfs-OST0001，二者皆为活动状态。

```
1 mgs:/root# lctl get_param lov.testfs-mdtlov.target_obd
2 0: testfs-OST0000_UUID ACTIVE
3 1: testfs-OST0001_UUID ACTIVE
```

14.3. 在无 Lustre 服务的情况下挂载服务器

如果您使用的为组合的 MGS/MDT，但却只希望启动 MGS，而不启动 MDT，请运行：

```
1 mount -t lustre /dev/mdt_partition -o nosvc /mount_point
```

变量*mdt_partition*为组合 MGS/MDT 块设备。

在这个例子汇中，组合的 MGS/MDT 为 `testfs-MDT0000`，挂载点为 `/mnt/test/mdt`。

```
1 $ mount -t lustre -L testfs-MDT0000 -o nosvc /mnt/test/mdt
```

14.4. 重新生成 Lustre 配置日志

如果 Lustre 文件系统配置日志所处状态使得文件系统无法启动，那么可以使用`tunefs.lustre --writeconf` 命令重新生成日志。该命令运行后，服务器重启，配置日志也将在 MGS (在新的文件系统中) 重新生成和保存。

`writeconf` 只在如下情况下使用：

- 配置日志所处状态使文件系统无法启动
- 需要更新服务器 NID

`writeconf` 对一些配置项具有毁灭性（如通过`conf_param`设置的 OST 池信息条目），因此须谨慎使用。

注意

使用 OST 池功能，可以命名一组 OST，以进行文件条带化。请注意，运行 `writeconf` 命令将擦除所有池信息（包括通过`lctl conf_param`设置的参数）。我们

推荐使用脚本运行池定义（conf_param 设置），以便于在 writeconf 命令运行后快速进行重定义。但是在这种情况下使用lctl set_param -P 设置的参数不会被擦除。

注意

如果 MGS 仍保留任何配置日志，则可以通过在 MGS 导出配置日志并保存输出，获取保存在lctl conf_param中的所有参数:

```
mgs# lctl --device MGS llog_print fsname-client
mgs# lctl --device MGS llog_print fsname-MDT0000
mgs# lctl --device MGS llog_print fsname-OST0000
```

重新生成 Lustre 文件系统配置日志:

1. 在运行tunefs.lustre --writeconf命令前，按照以下顺序关闭文件系统:

a. 卸载客户端

b. 卸载 MDT

c. 卸载所有 OSTs

d. 如果 MGS 与 MDT 独立，则可以在这个过程中挂载 MDT

2. 确保 MDT 和 OST 设备可用。

3. 在所有目标设备上运行 tunefs.lustre --writeconf 命令。

请先在 MDT 上运行writeconf，随后在所有 OSTs 上运行:

a. 在每个 MDS 上运行:

```
mdt# tunefs.lustre --writeconf /dev/mdt_device
```

b. 在每个 OST 上运行:

```
ost# tunefs.lustre --writeconf /dev/ost_device
```

4. 按照以下顺序重启文件系统:

a. 挂载独立的 MGS

b. 按顺序挂载 MDT，从 MDT0000 开始

c. 安顺讯挂载 OSTs，从 OST0000 开始

d. 挂载客户端

tunefs.lustre --writeconf 运行完成后，配置日志重新生成，服务器重启。

14.5. 更改服务器 NID

为了完全重写 Lustre 配置，可以使用`tunefs.lustre --writeconf`命令来重写所有的配置文件。

如果只需要改变 MDT 或 OST 的 NID，`replace_nids`命令可以简化这个过程。与`tunefs.lustre --writeconf`不同，`replace_nids` 命令并不擦除所有配置日志，从而免去了在所有服务器上运行 `writeconf` 时必须并重新指定所有参数设置的麻烦（必要情况下仍可使用 `writeconf`）。

更改服务器 NID 操作适用于以下情况：

- 新服务器硬件加入文件系统，而 MDS, OSS 服务迁入这些新服务器
- 服务器装入新网卡
- 重新分配 IP 地址

请参照以下步骤更改服务器 NID:

1. 更新 `/etc/modprobe.conf` 文件中的 LNet 配置以确保服务器 NIDs 列表无误。使用 `lctl list_nids` 查看服务器 NIDs 列表、Lustre 文件系统配置的网络。
2. 按照以下顺序关闭文件系统：

a. 卸载所有客户端

b. 卸载 MDT

c. 卸载所有 OST

3. 只启动 MGS（MGS 和 MDS 共享同一个分区）：

```
mount -t lustre MDT partition -o nosvc mount_point
```

4. 在 MGS 上运行 `replace_nids` 命令：

```
lctl replace_nids devicename nid1[,nid2,nid3 ...]
```

其中，*devicename* 为 Lustre 目标名称，如：`testfs-OST0013`。

5. 关闭 MGS（MGS 和 MDS 分享同一个分区）：

```
umount mount_point
```

注意

`replace_nids` 命令可清除配置日志中所有旧的、无效的记录，但保留当前记录。

注意

原先的配置日志将被备份在后缀为`'.bak'`的文件中，并保存在 MGS 磁盘上（Lustre 2.4 中引入）。

14.6. 清除配置

清除配置的命令运行在使用-o nosvc挂载的 MGS 设备的 MGS 节点上。它会清除所有标记了 ``SKIP" 的记录中的CONFIS/目录中存储的配置文件。如果给定了设备名称，则应清除该文件系统指定的日志 (例如，testfs-MDT0000)。如果给定了文件系统名称，则应清除所有配置文件。之前的配置日志以 ``config.timestamp.bak" 为后缀备份在 MGS 磁盘上。如:Lustre-MDT0000-1476454535.bak

请参照以下步骤清除配置:

1. 按照以下顺序关闭文件系统:

a. 卸载所有客户端

b. 卸载 MDT

c. 卸载所有 OST

2. 使用nosvc选项，只启动 MGS（MGS 和 MDS 共享同一个分区）:

```
mount -t lustre MDT partition -o nosvc mount_point
```

3. 在 MGS 上运行clear_conf命令

```
lctl clear_conf config
```

如：在文件系统testfs上清除 MDT0000 的配置，请运行:

```
mgs# lctl clear_conf testfs-MDT0000
```

14.7. 在 Lustre 文件系统中加入新的 MDT

通过 DNE 功能添加额外的 MDT，为文件系统中的一个或多个远程子目录提供服务，可以用来增加文件系统中可创建的文件总数，提高元数据总体性能，隔离用户或来自其他用户的应用程序工作负载。多个远程子目录可以使用相同的 MDT，但根目录将始终位于 MDT0000 上。想要添加新的 MDT 到文件系统中，请执行以下操作:

1. 查看最大 MDT 索引。每个 MDT 必须有一个唯一的索引。

```
client$ lctl dl | grep mdc
36 UP mdc testfs-MDT0000-mdc-ffff88004edf3c00
    4c8be054-144f-9359-b063-8477566eb84e 5
37 UP mdc testfs-MDT0001-mdc-ffff88004edf3c00
    4c8be054-144f-9359-b063-8477566eb84e 5
```

```
4   38 UP mdc testfs-MDT0002-mdc-ffff88004edf3c00
        4c8be054-144f-9359-b063-8477566eb84e 5
5   39 UP mdc testfs-MDT0003-mdc-ffff88004edf3c00
        4c8be054-144f-9359-b063-8477566eb84e 5
```

2. 在下一个可用的索引处添加新的块设备作为 MDT。在下面的例子中，下一个可用索引为 4。

```
mds# mkfs.lustre --reformat --fsname=testfs --mdt
--mgsnode=mgsnode --index 4 /dev/mdt4_device
```

3. 挂载 MDT。

```
mds# mount -t lustre /dev/mdt4_blockdevice /mnt/mdt4
```

4. 在新的 MDT 上创建新的文件或目录，须通过 `lfs mkdir` 命令将它们附加在命名空间的一个或多个子目录上。除非另外指定，否则通过 `lfs mkdir`创建的所有从属的文件和目录也将在同一个 MDT 上被创建。

```
1   client# lfs mkdir -i 3 /mnt/testfs/new_dir_on_mdt3
2   client# lfs mkdir -i 4 /mnt/testfs/new_dir_on_mdt4
3   client# lfs mkdir -c 4 /mnt/testfs/new_directory_striped_across_4_mdts
```

14.8. 在 Lustre 文件系统中添加新的 OST

可在 Lustre 文件系统中将新的 OST 添加至现有的 OSS 节点或新的 OSS 节点上。为维持客户端在多个 OSS 节点上的 IO 负载均衡，实现最大的总体性能，建议不要为每个 OSS 节点配置不同数量的 OST。

1. 当文件系统第一次进行格式化时，使用 `mkfs.lustre` 命令添加新的 OST。每个新的 OST 必须有一个唯一的索引，可使用 `lctl dl` 查看所有 OST 的列表。以下示例为添加一个新的 OST 至 `testfs` 文件系统，索引为 12：

```
oss# mkfs.lustre --fsname=testfs --mgsnode=mds16@tcp0 --ost
--index=12 /dev/sda oss# mkdir -p /mnt/testfs/ost12 oss# mount
-t lustre /dev/sda /mnt/testfs/ost12
```

2. 平衡 OST 空间使用。

当新的空白 OST 添加到相对拥挤的文件系统时，可能导致该文件系统的不平衡。但由于正在创建的新文件将优先放置在新的空白 OST 或不那么满的 OST 上，以自动平衡文件系统的使用量，如果这是一个暂存的或定期进行文件修剪的文件系统，则可能不需要进一步的操作来平衡 OST 空间使用率。当旧文件被删除时，原 OST 上的相应空间被释放。

可使用`lfs_migrate` 有选择性地重新平衡扩展前就存在的旧文件，从而使得所有 OST 上的文件数据被重新分配。

例如，重新平衡 `/mnt/lustre/dir`目录下的所有文件，请输入：

```
client# lfs_migrate /mnt/lustre/dir
```

将`OST0004` 上 `/test`文件系统中所有大于 4GB 的文件迁移至其他 OSTs，请输入：

```
client# lfs find /test --ost test-OST0004 -size +4G |
lfs_migrate -y
```

14.9. 移除及恢复 MDT 和 OST

可从 Lustre 文件系统中将 OST 和 DNE MDT 移除并恢复。将 OST 设置为不活跃状态意味着它将暂时或永久地被标记为不可用。将 MDS 上将 OST 设置为不活跃状态，意味着它将不再尝试在 MDS 上分配新对象或执行 OST 恢复；而在客户端上将 OST 设置为非活动状态则意味着：在无法联系上 OST 的情况下，它不会等待 OST 恢复，而是在 OST 文件被访问时立即将 IO 错误返回给应用。在特定的情况下或运行特定的命令，OST 可能会永久地在文件系统中停用。

注意

永久停用的 MDT 或 OST 仍会出现在文件系统配置中，直到使用 `writeconf` 重新生成配置或新 MDT 或 OST 在同一索引位置替代原设备并永久激活。`lfs df`不会列出已停用的 OST。

在以下情况中，您可能希望在 MDS 上暂时地停用 OST 以防止新文件写入：

- 硬盘驱动器出现故障并正在进行 RAID 重新同步或重建。（OST 在此时也可能被 RAID 系统标记为 *degraded* ，以避免在慢速 OST 上分配新文件，从而降低性能。）
- OST 接近其空间容量。（尽管 MDS 在这种情况下会尽可能尝试避免在过度拥挤的 OST 上分配新文件。）
- MDT/OST 存储或 MDS/OSS 节点故障并持续（或永久）不可用，但文件系统在修复前仍须继续工作。

（Lustre 2.4 中引入）

14.9.1. 在文件系统中移除 MDT

如果 MDT 永久不可用，可使用 `lfs rm_entry {directory}` 删除该 MDT 的目录条目，由于 MDT 处于不活跃状态，使用 `rmdir` 将导致 IO 错误。请注意，如果 MDT 可用，则应使用标准的 `rm -r` 命令来删除远程目录。该删除操作完成后，管理员应使用以下命令将 MDT 标记为永久停用状态：

```
lctl conf_param {MDT name}.mdc.active=0
```

用户可使用 `lfs` 工具确认含有远程子目录的 MDT，如：

```
client$ lfs getstripe --mdt-index /mnt/lustre/remote_dir1
1
client$ mkdir /mnt/lustre/local_dir0
client$ lfs getstripe --mdt-index /mnt/lustre/local_dir0
0
```

`lfs getstripe --mdt-index`命令返回服务于当前给定目录的 MDT 索引。

14.9.2. 不活跃的 MDTs

位于不活跃 MDT 上的文件在该 MDT 被重新激活前不可用。尝试访问不活跃 MDT 的客户端将收到 EIO 错误。

14.9.3. 在文件系统中移除 OST

当将 OST 设置为不活跃状态时，客户端和 MDS 都各有一个 OSC 设备用于处理和响应与该 OST 的交互。从文件系统中移除 OST：

1. 如果 OST 仍然可用，并且有文件落在这个 OST 上，而文件必须迁移出这个 OST，那么应在 MDS 上暂时停用在该 OST 上的文件创建（如果有多个 MDS 节点在 DNE 模式下运行，则应在每个 MDS 执行该操作）。

a. 在 Lustre2.9 或更高版本中，通过在 MDS 上将`max_create_count` 设置为 0，从而禁止该 OST 的文件创建：

```
mds# lctl set_param osp.*osc_name*.max_create_count=0
```

这可以确保，一旦文件从 OST 中删除或迁移出去，那么它对应的 OST 对象将被被销毁，相应空间将被释放。例如，在文件系统 testfs 中停用 OST0000，在 testfs 文件系统上的每个 MDS 上运行：

```
mds# lctl set_param osp.testfs-OST0000-osc-MDT*.max_create_count=0
```

b. 在更老的 Lustre 版本中，将 MDS 节点上的 OST 设置为不活跃状态，请运行：

```
mds# lctl set_param osp.osc_name.active=0
```

这将阻止 MDS 尝试与该 OST 进行通信，MDS 也不会连接 OST 以删除位于 OST 上的对象。如果 OST 被永久删除，或者因 OST 在操作中不稳定或处于只读状态而这么做，那么就没什么问题。否则，删除文件之后，OST 上的空闲空间和对象不会减少，对象也不会被销毁，直到 MDS 重新连接到 OST。

例如，在文件系统 testfs中，将 OST0000 设置为不活跃状态：

```
mds# lctl set_param osp.testfs-OST0000-osc-MDT*.active=0
```

在 MDS 上将 OST 设置为不活跃状态不会影响客户端对当前对象进行读取/写入。

注意

如果从正在工作的 OST 中迁移文件，请不要停用客户端上的 OST。这会导致访问位于该 OST 上文件时产生 IO 错误，从而使 OST 迁移文件失败。

如果 OST 在工作中，请不要使用 lctl conf_param 将其设置为不活跃状态，因为这会使其在 MDS 和所有客户端上的文件系统配置中立刻并永久停用。

2. 查找所有含驻留在不活跃 OST 上对象的文件。如果该 OST 可访问，则需要将来自该 OST 的数据迁移到其他 OST 上，不然将需要从备份恢复数据。

a. 如果该 OST 在线或可访问，查找所有含驻留其上对象的文件并将其数据复制到文件系统的其他 OST 上：

```
client# lfs find --ost ost_name /mount/point | lfs_migrate
-y
```

注意如果多个 OST 在同一时间被停用，lfs find 命令可带多个--ost参数，返回所有位于指定 OST 上的文件。

b. 如果该 OST 不可访问，则删除在该 OST 上的所有文件并从备份恢复数据：

```
client# lfs find --ost ost_uuid -print0 /mount/point | tee
/tmp/files_to_restore | xargs -0 -n 1 unlink
```

需要从备份恢复的文件列表存储在 /tmp/files_to_restore中。

3. 将 OST 设置为不活跃状态。

a. 如果预计在短时间（几天）内有可替代的 OST，可使用以下方式临时停用 OST：

```
client# lctl set_param osc.fsname-OSTnumber-*.active=0
```

注意

该设置为暂时的，当客户端重新挂载或重启时将被重置。该命令需要在所有客户端上运行。

b. 如果预计近期内无可替代的 OST，在 MDS 上运行以下命令以在所有客户端 MDS 上永久停用 OST：

```
mgs# lctl conf_param ost_name.osc.active=0
```

注意

停用的 OST 仍然出现在文件系统配置中，不过可以使用`mkfs.lustre --replace`选项创建一个替代的 OST，如需重新使用相同的 OST 索引，可参考第 14.9.5 章节，``恢复 OST 配置文件"。

如需从文件系统配置中完全删除 OST，应在 MGS 上运行 ``lctl --device MGS llog_print fsname-client" 命令，以在启动日志中找到 OST 配置记录（对于所有 MDT 也是 ``...$fsame-MDTxxxx"），然后列出所有与被删除的 OST 相关的`attach`、`setup`、`add_osc`、`add_pool`以及其他记录。一旦知道每个配置记录的索引值，则运行命令 ``lctl --device MGS llog_cancel llog_name -i index" 将从每个`fsame-client`和`fsame-MDTxxxx`配置日志的 llog_name 中删除该记录，这样新的挂载就不会再处理它。如果移除了整个 OSS，则 OSS 的 add_uuid 记录也应同样被删除。

```
mgs# lctl --device MGS llog_print testfs-client |
egrep "192.168.10.99@tcp|OST0003" - { index: 135, event:
add_uuid, nid: 192.168.10.99@tcp(0x20000c0a80a63), node:
192.168.10.99@tcp } - { index: 136, event: attach, device:
testfs-OST0003-osc, type: osc, UUID: testfs-clilov_UUID }
- { index: 137, event: setup, device: testfs-OST0003-osc,
UUID: testfs-OST0003_UUID, node: 192.168.10.99@tcp } - {
index: 138, event: add_osc, device: testfs-clilov, ost:
testfs-OST0003_UUID, index: 3, gen: 1 } mgs# lctl --device
MGS llog_cancel testfs-client -i 138 mgs# lctl --device
MGS llog_cancel testfs-client -i 137 mgs# lctl --device MGS
llog_cancel testfs-client -i 136
```

14.9.4. 备份 OST 配置文件

如果 OST 设备仍可访问，则 OST 上的 Lustre 配置文件应及时备份并保存以供将来使用，从而避免更换 OST 恢复服务时出现问题。这些文件很少发生变化，所以它们应在 OST 正常工作且可访问的情况下进行备份。如果停用的 OST 仍可成功挂载（即未因严重损坏而永久失效或无法挂载），则应努力保留这些文件。

1. 挂载 OST 文件系统。

```
oss# mkdir -p /mnt/ost
oss# mount -t ldiskfs /dev/ost_device /mnt/ost
```

2. 备份 OST 配置文件。

```
oss# tar cvf ost_name.tar -C /mnt/ost last_rcvd \
CONFIGS/ O/0/LAST_ID
```

3. 卸载 OST 文件系统。

```
oss# umount /mnt/ost
```

14.9.5. 恢复 OST 配置文件

替换因损坏或硬件故障而从服务中被删除的 OST，请首先使用 `mkfs.lustre`将新的 OST 格式化，并恢复 Lustre 文件系统配置（如果可用）。存储在 OST 上的所有对象都将永久丢失，使用 OST 的文件应该从备份中删除和（或）恢复。

Lustre 2.5 及更高版本中，可在不恢复配置文件的情况下替换 OST 至原索引处。请在格式化时使用 `--replace` 选项：

```
oss# mkfs.lustre --ost --reformat --replace --index=old_ost_index \
        other_options /dev/new_ost_dev
```

MDS 和 OSS 负责协商替换 OST 的 `LAST_ID` 值。

当 OST 文件系统完全无法访问时，OST 配置文件未备份时，即使 OST 文件系统完全无法访问，仍可在相同索引处用新的 OST 替换故障 OST。

1. 更早的版本中的 OST 文件系统格式化和配置恢复（不使用 `--replace` 选项）。

```
oss# mkfs.lustre --ost --reformat --index=old_ost_index \
            other_options /dev/new_ost_dev
```

2. 挂载 OST 文件系统。

```
oss# mkdir /mnt/ost
oss# mount -t ldiskfs /dev/new_ost_dev /mnt/ost
```

3. 恢复 OST 配置文件（如果可用）。

```
oss# tar xvf ost_name.tar -C /mnt/ost
```

4. 重新创建 OST 配置文件（如果恢复不可用）。

当使用默认参数（一般情况下适用于所有文件系统）第一次挂载 OST 时，last_rcvd 文件将会被重建。CONFIGS/mountdata 文件由 mkfs.lustre 在格式化时创建，并含有标志设置以向 MGS 发出注册请求。可从另一个工作中的 OST 复制标志。

```
oss1# debugfs -c -R "dump CONFIGS/mountdata /tmp" /dev/other_osdev
oss1# scp /tmp/mountdata oss0:/tmp/mountdata
oss0# dd if=/tmp/mountdata of=/mnt/ost/CONFIGS/mountdata bs=4 count=1
    seek=5 skip=5 conv=notrunc
```

5. 卸载 OST 文件系统。

```
oss# umount /mnt/ost
```

14.9.6. 重新激活 OST

如果 OST 永久不可用，须在 MGS 配置中重新激活它。

```
mgs# lctl conf_param ost_name.osc.active=1
```

如果 OST 暂时不可用，须在 MGS 和客户端上重新激活它。

```
mds# lctl set_param osp.fsname-OSTnumber-*.active=1
client# lctl set_param osc.fsname-OSTnumber-*.active=1
```

14.10. 终止恢复

可使用 lctl 工具或通过abort_recov选项（mount -o abort_recov）终止恢复。启动一个目标，请运行:

```
mds# mount -t lustre -L mdt_name -o abort_recov /mount_point
```

注意

恢复过程将被阻塞，直到所有 OST 都可用时。

14.11. 确定服务 OST 的机器

在管理 Lustre 文件系统的过程中，您可能需要确定哪台机器正在为特定的 OST 提供服务。这不像识别机器 IP 地址那么简单，IP 只是 Lustre 软件使用的几种网络协议之一，因此 LNet 使用 NID 而不是 IP 地址作为节点标识符。要识别服务 OST 的机器 NID，请在客户端上运行以下命令之一（不必是 root 用户）:

```
client$ lctl get_param osc.fsname-OSTnumber*.ost_conn_uuid
```

```
client$ lctl get_param osc.*-OST0000*.ost_conn_uuid
osc.testfs-OST0000-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
```

```
client$ lctl get_param osc.*.ost_conn_uuid
osc.testfs-OST0000-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
osc.testfs-OST0001-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
osc.testfs-OST0002-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
osc.testfs-OST0003-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
osc.testfs-OST0004-osc-f1579000.ost_conn_uuid=192.168.20.1@tcp
```

14.12. 更改故障节点地址

更改故障节点的地址（如使用节点 X 替换节点 Y），在 OSS/OST 分区上运行（取决于定义 NID 时使用的选项）：

```
oss# tunefs.lustre --erase-params --servicenode=NID /dev/ost_device
```

或

```
oss# tunefs.lustre --erase-params --failnode=NID /dev/ost_device
```

14.13. 分离组合的 MGS/MDT

以下操作在服务器和客户端开机状态下进行，并假设 MGS 节点与 MDS 节点相同。

1. 暂停 MDS 服务。卸载 MDT。

   ```
   umount -f /dev/mdt_device
   ```

2. 创建 MGS。

   ```
   mds# mkfs.lustre --mgs --device-size=size /dev/mgs_device
   ```

3. 从 MDT 磁盘拷贝配置信息至新的 MGS 磁盘。

   ```
   mds# mount -t ldiskfs -o ro /dev/mdt_device /mdt_mount_point
   mds# mount -t ldiskfs -o rw /dev/mgs_device /mgs_mount_point
   ```

```
```

mds# cp -r /mdt_mount_point/CONFIGS/filesystem_name-* /mgs_mount_point/CONFIGS/. ```

```
mds# umount /mgs_mount_point
mds# umount /mdt_mount_point
```

4. 启动 MGS。

```
mgs# mount -t lustre /dev/mgs_device /mgs_mount_point
```

查看其是否获知所有文件系统。

```
mgs:/root# lctl get_param mgs.MGS.filesystems
```

5. 从 MDT 上移除 MG 选项，设置新的 MGS NID。

```
mds# tunefs.lustre --nomgs --mgsnode=new_mgs_nid
/dev/mdt-device
```

6. 启动 MDT。

```
mds# mount -t lustre /dev/mdt_device /mdt_mount_point
```

查看 MGS 配置是否正确。

```
mgs# lctl get_param mgs.MGS.live.filesystem_name
```

(Lustre 2.13 引入)

14.14. 将 MDT 设置为只读

有时，我们希望能够在服务器上直接标记文件系统为只读，而不需要在重新挂载客户端时设置该选项。如果有一个流氓客户正在删除文件，或者在停用一个系统时，防止已经挂载的客户再修改它，这就很有用。

将 mdt.*.readonly 参数设置为 1，可以立即将 MDT 设置为只读。之后所有对 MDT 访问将立即返回一个 ``只读文件系统" 的错误（EROFS），直到该参数再次被设置为 0。

以下示例为将只读参数设置为 1，验证当前设置，从客户端访问，并将参数设置回 0：

```
1 mds# lctl set_param mdt.fs-MDT0000.readonly=1
2 mdt.fs-MDT0000.readonly=1
3 mds# lctl get_param mdt.fs-MDT0000.readonly
4 mdt.fs-MDT0000.readonly=1
5 client$ touch test_file
6 touch: cannot touch ‘’ test_file: Read-only file system
7 mds# lctl set_param mdt.fs-MDT0000.readonly=0
8 mdt.fs-MDT0000.readonly=0
```

(Lustre 2.14 引入)

14.15. 调整 ldiskfs 的 Fallocate

本节说明了如何调整/启用/禁用 ldiskfs OSTs 的 fallocate。

默认的`mode=0`表示 ext4 使用标准的 ``分配未写入的 extents'' 行为。这是迄今为止最快的空间分配方式，但是当未写入的 extents 被覆盖时，需要对其进行分割或清零。

`mode=1`即 OST fallocate ，表示也可以设置为使用 ``归零的 extents''，这可以由 ``WRITE SAME''、``TRIM zeroes data'' 或其他底层块设备的低级功能来处理。

`mode=-1`表示完全禁用 fallocate。

示例：完全禁用 fallocate

```
1 lctl set_param osd-ldiskfs.*.fallocate_zero_blocks=-1
```

示例: 启用 fallocate 以使用 ``零扩展（zeroed extents）''

```
1 lctl set_param osd-ldiskfs.*.fallocate_zero_blocks=1
```

第十五章管理 Lustre Networking (LNet)

15.1. 更新路由或端的健康状态

LNET 路由或端（peer）的健康状态更新机制，有两种：

- LNet 可以主动检查所有路由的健康状况，并自动将其标记为'dead' 或'alive'。默认情况下，该功能为关闭状态，可通过设置`auto_down`启用，并根据需要设置`check_routers_before_use`。如果系统中存在已死亡的路由，系统启动时进行的初始检查可能导致`router_ping_timeout`时间的暂停。
- 当出现通信错误时，所有 LND 都会通知 LNet 端（不一定是路由）已下线。该功能呈始终开启，并且没有参数可以关闭它。但如果将 LNet 模块参数`auto_down`设置为 0，则 LNet 将忽略所有这种端下线的通知。

这两种机制的关键不同点在于：

- 路由 Pinger 只检查路由的健康状态，而 LND 则会注意到所有死掉的端，无论这些端是否为路由。
- 路由通过发送 ping 命令来主动检查路由的健康状态，而 LND 只会在网络上有通信时才会注意到一个死掉的端。
- 路由 Pinger 可以将路由从活动状态变为死亡状态，反之亦然，但 LND 只能标记端为下线状态。

15.2. 启动和关闭 LNet

Lustre 软件可自动启动和关闭 LNet，但 LNET 也可以以独立方式手动启动。这个方法可用于在尝试启动 Lustre 文件系统之前验证网络设置是否正常。

15.2.1. 启动 LNet

启动 LNet，运行：

```
$ modprobe lnet
$ lctl network up
```

查看本地 NID 列表，运行：

```
$ lctl list_nids
```

该命令显示了 Lustre 文件系统的网络配置。

如果网络未正确设置，查看`modules.conf`文件中`networks=`行，并确保已正确安装和配置网络层模块。

想要获取最佳远程 NID，运行：

```
$ lctl which_nid NIDs
```

其中，*`NIDs`*为可用 NID 列表。

该命令将从远程主机列表中选取" 最佳" 的 NID，即本地节点与远程节点通信时会使用的 NID。

15.2.1.1. 启动客户端

启动 TCP 客户端，运行：

```
mount -t lustre mdsnode:/mdsA/client /mnt/lustre/
```

启动 Elan 客户端，运行：

```
mount -t lustre 2@elan0:/mdsA/client /mnt/lustre
```

15.2.2. 关闭 LNet

在移除 LNet 模块之前，必须移除 LNet 引用。通常，关闭 Lustre 文件系统时会自动删除这些引用。但对于独立路由，关闭 LNet 需要明确的步骤。运行：

```
lctl network unconfigure
```

注意

试图在停止网络之前删除 Lustre 模块可能会导致系统崩溃或 LNet 挂起。如果发生这种情况，必须重新启动节点（在大多数情况下）。请确保在卸载模块之前 Lustre 网络和 Lustre 文件系统已关闭，并谨慎使用 `rmmod -f`.

取消 LNet 网络配置，请运行：

```
1 modprobe -r lnd_and_lnet_modules
```

注意

卸载所有 Lustre 模块，请运行：

```
$ lustre_rmmod
```

15.3. 基于 LNet 多轨配置的硬件

使用 LNet 在双轨（dual-rail）IB 群集（o2iblnd）的两个轨道上聚合带宽，请考虑以下几点问题：

- LNet 可以使用多轨（multi-rail）配置，但并不会在它们之间进行负载均衡。在通信中实际使用的轨道由端的 NID 决定。
- 硬件多轨 LNet 配置不会增加一级额外的网络容错。下面章节中描述的配置仅用于增加聚合带宽。
- 对一给定端 NID，Lustre 节点总是使用某一相同本地 NID 进行通信。如何确定本地 NID，请参照：
- 最低的优先值（优先值越低，优先级越高，在 **Lustre 2.5** 中引入）；
- 最少的跳数，以减少路由；
- 在"networks" 或"ip2nets"LNet 配置字符串中位于首位。

15.4. 利用 InfiniBand* 网络实现负载平衡

若 Lustre 文件系统中的 OSS 有两个 InfiniBand HCAs，客户端有一个 InfiniBand HCA（使用 OFED-based Infiniband "o2ib" 驱动器）。OSS 上 HCA 间的负载均衡可通过 LNet 实现。

15.4.1. 在lustre.conf中配置负载均衡

在 LNet 中为客户端和服务器配置负载均衡：

1. 设置lustre.conf选项。

   根据您的配置，可将lustre.conf选项配置为：

   - 双 HCA OSS 服务器

     ```
     options lnet networks="o2ib0(ib0),o2ib1(ib1)"
     ```

   - IP 地址为奇数的客户端

```
options lnet ip2nets="o2ib0(ib0)192.168.10.[103-253/2]"
```

- IP 地址为偶数的客户端

```
options lnet ip2nets="o2ib1(ib0)192.168.10.[102-254/2]"
```

2. 运行 modprobe lnet 命令，创建组合的 MGS/MDT 文件系统。

以下命令将创建一个组合的 MGS/MDT 或 OST 文件系统并在服务器上挂载目标。

```
1 modprobe lnet
2 # mkfs.lustre --fsname lustre --mgs --mdt /dev/mdt_device
3 # mkdir -p /mount_point
4 # mount -t lustre /dev/mdt_device /mount_point
```

如：

```
1 modprobe lnet
2 mds# mkfs.lustre --fsname lustre --mdt --mgs /dev/sda
3 mds# mkdir -p /mnt/test/mdt
4 mds# mount -t lustre /dev/sda /mnt/test/mdt
5 mds# mount -t lustre mgs@o2ib0:/lustre /mnt/mdt
6 oss# mkfs.lustre --fsname lustre --mgsnode=mds@o2ib0 --ost --index=0
    /dev/sda
7 oss# mkdir -p /mnt/test/mdt
8 oss# mount -t lustre /dev/sda /mnt/test/ost
9 oss# mount -t lustre mgs@o2ib0:/lustre /mnt/ost0
```

3. 挂载客户端。

```
client# mount -t lustre mgs_node:/fsname /mount_point
```

以下为挂载 IB 客户端的例子：

```
1 client# mount -t lustre
2 192.168.10.101@o2ib0,192.168.10.102@o2ib1:/mds/client /mnt/lustre
```

假设，两轨的 IB 集群在 OFED 栈运行，而被分配的 IP 地址如下所示。

```
1             ib0                          ib1
2 Servers           192.168.0.*                  192.168.1.*
3 Clients           192.168.[2-127].*            192.168.[128-253].*
```

您可创建以下配置：

- 客户端比服务器更多的群集。单个客户端无法获得两轨带宽，但由于服务器带宽通常才是实际的瓶颈，这一问题并不重要。

```
ip2nets="o2ib0(ib0),    o2ib1(ib1)      192.168.[0-1].*
    \
                                          #all servers;\
                  o2ib0(ib0)      192.168.[2-253].[0-252/2]       #even cl\
ients;\
                  o2ib1(ib1)      192.168.[2-253].[1-253/2]       #odd cli\
ents"
```

该配置给每个服务器分配两个 NIDs，每个网络一个 NID，对客户端在两轨间使用静态负载平衡。

- 获得两轨带宽的客户端。单个客户端必须获得两轨带宽，即使最大总带宽仅为 (# servers) * (1 rail)。

```
ip2nets="  o2ib0(ib0)                      192.168.[0-1].[0-252/2]     \
#even servers;\
           o2ib1(ib1)                      192.168.[0-1].[1-253/2]        \
#odd servers;\
           o2ib0(ib0),o2ib1(ib1)           192.168.[2-253].*              \
#clients"
```

该配置给每个服务器的每一轨分配一个 NID，客户端获得的两个轨道各有一个 NID。

- 所有客户端和服务器都获得两轨带宽。

```
ip2nets="   o2ib0(ib0),o2ib2(ib1)           192.168.[0-1].[0-252/2]       \
#even servers;\
           o2ib1(ib0),o2ib3(ib1)           192.168.[0-1].[1-253/2]        \
#odd servers;\
           o2ib0(ib0),o2ib3(ib1)           192.168.[2-253].[0-252/2)      \
#even clients;\
           o2ib1(ib0),o2ib2(ib1)           192.168.[2-253].[1-253/2)      \
#odd clients"
```

此配置包含两个额外的 o2ib 代理网络，用来绕过 Lustre 软件中简单的 NID 选择算法。" 偶数" 客户端通过 o2ib0 网络在 rail0 上连接" 偶数" 服务器，通过 o2ib3 网络在 rail1

上连接" 奇数" 服务器。同样地，" 奇数" 客户端通过 o2ib1 网络在 rail0 上连接" 奇数" 服务器，通过 o2ib2 网络在 rail1 上连接" 偶数" 服务器。

Lustre 2.4 中引入

15.5. 动态配置 LNet 路由

我们提供了两个脚本：`lustre/scripts/lustre_routes_config`，`lustre/scripts/lustre_routes_conversion`。

`lustre_routes_config` 通过指定的配置文件设置或清除 LNet 路由。`/etc/sysconfig/lnet_routes.conf`文件用于在 LNet 启动时自动配置路由。

`lustre_routes_conversion`将传统的路由配置文件转换为新的语法，并通过`lustre_routes_config`进行解析。

15.5.1. `lustre_routes_config`

`lustre_routes_config` 的用法如下：

```
lustre_routes_config [--setup|--cleanup|--dry-run|--verbose] config_file
        --setup: configure routes listed in config_file
        --cleanup: unconfigure routes listed in config_file
        --dry-run: echo commands to be run, but do not execute them
        --verbose: echo commands before they are executed
```

导入脚本的文件格式为：

network: { gateway: *gateway@exit_network* [hop: *hop*] [priority: *priority*] }

当 LNet 路由的本地 NID 出现在路由列表中时，该路由将被识别。脚本只能在路由被识别后才能添加额外的路由。因此，为使路由被正确识别，请确保在 modprobe luster 配置文件的 routes 参数中添加其本地 NID。

15.5.2. `lustre_routes_conversion`

`lustre_routes_conversion` 用法如下：

```
lustre_routes_conversion legacy_file new_file
```

`lustre_routes_conversion` 的第一个参数为一个包含如下路由配置的文件：

network [*hop*] *gateway@exit network*[:*priority*];

该脚本将文件中的每条路由转换为：

network: { gateway: *gateway@exit network* [hop: *hop*] [priority: *priority*] }

并将新的转换后的路由条目附加到输出文件（脚本的第二个参数）中。

15.5.3. 路由配置示例

下面是一个传统的 LNet 路由配置的例子，含有多个条目。

```
tcp1 10.1.1.2@tcp0:1;
tcp2 10.1.1.3@tcp0:2;
tcp3 10.1.1.4@tcp0;
```

以下是lustre_routes_conversion脚本对以上传统路由配置实施转换后的 LNet 路由配置示例：

```
tcp1: { gateway: 10.1.1.2@tcp0 priority: 1 }
tcp2: { gateway: 10.1.1.2@tcp0 priority: 2 }
tcp1: { gateway: 10.1.1.4@tcp0 }
```

第十六章 LNet 软件多轨

16.1. 概述

在计算机网络中，多轨（Multi-rail）指的是在计算机节点上使用两个或更多的网络接口，以达到提高吞吐量的目的。多轨也可能采用在单一节点有一个或更多的网络接口连接多个不同网络的情形，这些网络甚至可能包含不同的类型（如：Ethernet, Infiniband, and Intel® Omni-Path）。通过多轨配置，Lustre 客户端通常将多个网络的能力组合当作单个 LNet 网络。具备多轨功能的端节点，将同用户定义的接口策略一起，在配置期间创建。

该功能更详细的高级配置及设计请参阅：Multi-Rail High-Level Design

16.2. 多轨的配置

每个使用多轨网络的节点都需要进行适当的配置。多轨机制使用 `lnetctl` 和 LNet 配置库来进行配置。配置多轨牵涉到两个任务：

1. 配置本地节点上的多个网络接口。
2. 添加具有多轨功能的远程端（通过至少两个接口连接到一个或多个网络）。

16.2.1. 在本地节点上配置多个接口

运行lnetctl add命令在多轨配置中添加多个接口：

```
lnetctl net add --net tcp --if eth0,eth1
```

以 YAML 方式显示网络信息：

```
lnetctl net show -v
net:
    - net type: lo
      local NI(s):
        - nid: 0@lo
          status: up
          statistics:
              send_count: 0
              recv_count: 0
              drop_count: 0
          tunables:
              peer_timeout: 0
              peer_credits: 0
              peer_buffer_credits: 0
              credits: 0
          lnd tunables:
          tcp bonding: 0
          dev cpt: 0
          CPT: "[0]"
    - net type: tcp
      local NI(s):
        - nid: 192.168.122.10@tcp
          status: up
          interfaces:
              0: eth0
          statistics:
              send_count: 0
              recv_count: 0
              drop_count: 0
          tunables:
              peer_timeout: 180
              peer_credits: 8
              peer_buffer_credits: 0
              credits: 256
          lnd tunables:
          tcp bonding: 0
```

```
          dev cpt: -1
          CPT: "[0]"
        - nid: 192.168.122.11@tcp
          status: up
          interfaces:
              0: eth1
          statistics:
              send_count: 0
              recv_count: 0
              drop_count: 0
          tunables:
              peer_timeout: 180
              peer_credits: 8
              peer_buffer_credits: 0
              credits: 256
          lnd tunables:
          tcp bonding: 0
          dev cpt: -1
          CPT: "[0]"
```

16.2.2. 删除网络接口

`lnetctl net del`命令用于删除网络接口。假设当前网络配置如上所示(`lnetctl net show -v`命令显示了当前网络信息)，运行以下命令删除指定的网络接口：

```
lnetctl net del --net tcp --if eth0
```

删除后网络信息如下：

```
lnetctl net show -v
net:
    - net type: lo
      local NI(s):
        - nid: 0@lo
          status: up
          statistics:
              send_count: 0
              recv_count: 0
```

```
            drop_count: 0
        tunables:
            peer_timeout: 0
            peer_credits: 0
            peer_buffer_credits: 0
            credits: 0
        lnd tunables:
        tcp bonding: 0
        dev cpt: 0
        CPT: "[0,1,2,3]"
```

如使用 YAML 方式进行删除操作，语法如下：

```
- net type: tcp
  local NI(s):
    - nid: 192.168.122.10@tcp
      interfaces:
          0: eth0
```

16.2.3. 增加具有多轨功能的远程对等节点

在下面的例子中，`lnetctl peer add`命令增加了一个含两个 NID 的端节点，其中，主 NID 为 `192.168.122.30@tcp`：

```
lnetctl peer add --prim_nid 192.168.122.30@tcp --nid
   192.168.122.30@tcp,192.168.122.31@tcp
```

运行`lnetctl peer show`查看相关信息：

```
lnetctl peer show -v
peer:
    - primary nid: 192.168.122.30@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.30@tcp
          state: NA
          max_ni_tx_credits: 8
          available_tx_credits: 8
          min_tx_credits: 7
          tx_q_num_of_buf: 0
```

```
          available_rtr_credits: 8
          min_rtr_credits: 8
          refcount: 1
          statistics:
              send_count: 2
              recv_count: 2
              drop_count: 0
        - nid: 192.168.122.31@tcp
          state: NA
          max_ni_tx_credits: 8
          available_tx_credits: 8
          min_tx_credits: 7
          tx_q_num_of_buf: 0
          available_rtr_credits: 8
          min_rtr_credits: 8
          refcount: 1
          statistics:
              send_count: 1
              recv_count: 1
              drop_count: 0
```

使用 YAML 方式进行该操作：

```
addPeer.yaml
peer:
    - primary nid: 192.168.122.30@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.31@tcp
```

16.2.4. 删除远程对等节点

删除对等节点的单个 NID(192.168.122.31@tcp)：

```
lnetctl peer del --prim_nid 192.168.122.30@tcp --nid 192.168.122.31@tcp
```

删除整个对等节点：

```
lnetctl peer del --prim_nid 192.168.122.30@tcp
```

通过 YAML 方式删除对等节点：

```
Assuming the following peer configuration:
peer:
    - primary nid: 192.168.122.30@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.30@tcp
          state: NA
        - nid: 192.168.122.31@tcp
          state: NA
        - nid: 192.168.122.32@tcp
          state: NA

You can delete 192.168.122.32@tcp as follows:

delPeer.yaml
peer:
    - primary nid: 192.168.122.30@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.32@tcp

% lnetctl import --del < delPeer.yaml
```

16.3. 多轨路由注意事项

本节详细介绍了在 Lustre 2.13 中落地的第 16.4 节" 使用 LNet 健康的多轨制路由 ``功能之前，如何用路由功能配置多轨。路由代码会一直监控路由的状态，以避免使用了不可用的路由。

本节介绍了如何在同一个网关节点上配置多个接口作为不同的路由。这里使用了现有的路由监控算法来防范接口宕机。通过 Lustre 2.13 中引入的第 16.4 节 ``LNet Health 下的多轨路由" 功能，新的算法使用第 16.5 节 ``LNet Health" 功能来监控网关的不同接口，并始终确保使用最健康的接口。因此，本节中描述的配置适用于 Lustre 2.13 之前的版本。这些配置在 2.13 中也仍然可以工作，但是由于上面提到的原因，因此不是必需的。

16.3.1. 多轨集群示例

如图所示，在该集群中所有的 Lustre 节点都具有多轨功能，并配置有两个接口。

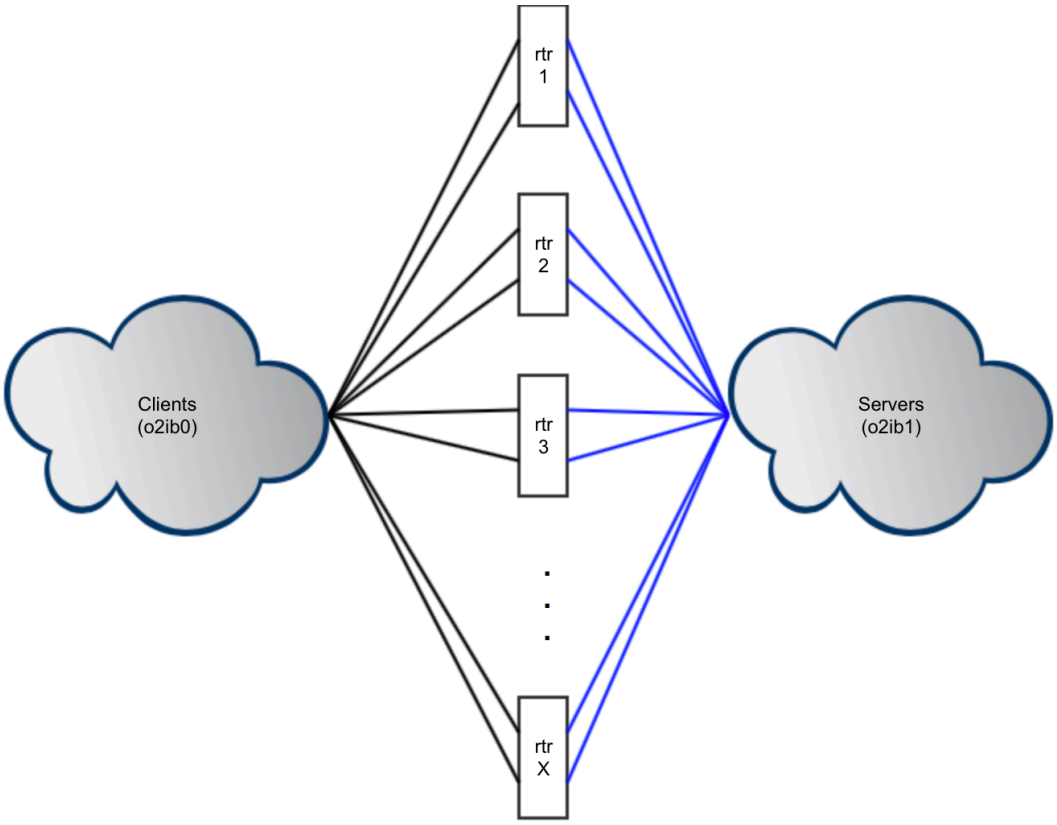


图 9: Lustre component

路由器通过在适当的网络上配置两侧接口来聚合接口性能。配置示例如下：

```
Routers
lnetctl net add --net o2ib0 --if ib0,ib1
lnetctl net add --net o2ib1 --if ib2,ib3
lnetctl peer add --nid <peer1-nidA>@o2ib,<peer1-nidB>@o2ib,...
lnetctl peer add --nid <peer2-nidA>@o2ib1,<peer2-nidB>>@o2ib1,...
lnetctl set routing 1

Clients
lnetctl net add --net o2ib0 --if ib0,ib1
lnetctl route add --net o2ib1 --gateway <rtrX-nidA>@o2ib
lnetctl peer add --nid <rtrX-nidA>@o2ib,<rtrX-nidB>@o2ib

Servers
lnetctl net add --net o2ib1 --if ib0,ib1
lnetctl route add --net o2ib0 --gateway <rtrX-nidA>@o2ib1
lnetctl peer add --nid <rtrX-nidA>@o2ib1,<rtrX-nidB>@o2ib1
```

在上述配置中，客户端和服务器为每个路由只配置了一行路由条目。由于路由具有多轨（MR）功能，这是可行的。路由作为含多个网络接口的端节点加入客户端和服务器中。当发送数据到路由时，MR 算法将确保路由的两个接口都会使用到。

到 **Lustre 2.10** 版本为止，LNet 可恢复（LNet Resiliency）功能仍在开发中，单接口故障仍会导致整个路由器停机。

16.3.2. 路由器可恢复功能

目前，LNet 提供了一种机制来监视每条路由。LNet 以固定的、可配置的时间间隔来 ping 路由条目中标识的每个网关，以确保其处于活跃状态。如果特定路由通信失败，或路由器 pinger 确定网关已掉线，则该路由被标记为'down' 状态且不可用。下一个固定的时间间隔后，确定它是否再次生效。

该机制可与 Lustre2.10 中的多轨功能结合使用，以增加路由器在使用过程中的可恢复性。

```
Routers
lnetctl net add --net o2ib0 --if ib0,ib1
lnetctl net add --net o2ib1 --if ib2,ib3
lnetctl peer add --nid <peer1-nidA>@o2ib,<peer1-nidB>@o2ib,...
lnetctl peer add --nid <peer2-nidA>@o2ib1,<peer2-nidB>@o2ib1,...
lnetctl set routing 1

Clients
lnetctl net add --net o2ib0 --if ib0,ib1
lnetctl route add --net o2ib1 --gateway <rtrX-nidA>@o2ib
lnetctl route add --net o2ib1 --gateway <rtrX-nidB>@o2ib

Servers
lnetctl net add --net o2ib1 --if ib0,ib1
lnetctl route add --net o2ib0 --gateway <rtrX-nidA>@o2ib1
lnetctl route add --net o2ib0 --gateway <rtrX-nidB>@o2ib1
```

上述配置中有以下几点注意事项：

1. 客户端和服务器现配置有两条路由，每条路由的网关是路由的接口之一。客户端和服务器将各自把同一路由器的各个接口看作单独的网关，并按照上述方式进行监视。
2. 客户端和服务器并未配置为视路由器为多轨模式。这很重要，因为我们希望将每个接口作为单独的端进行处理，而不是同一端的不同接口。

3. 路由则配置为视这些端为多轨模式。这看起来有些奇怪，但目前需要这样做才能使路由器实现负载均衡，即流量在所有接口上平均分配。

16.3.3. 多轨及非多轨混合集群

上述原则可应用于多轨及非多轨混合集群。例如，客户端和服务器不具备多轨功能而路由器具备多轨功能的情况下，可使用以上相同的配置。这是一种常见的集群升级方案。

(以下 Lustre 2.13 引入)

16.4. LNet Health 下的多轨路由

本节详细介绍了从 Lustre 2.13 开始如何配置路由和相关的模块参数。

动态发现的多轨功能允许 LNet 发现并使用一个节点的所有配置接口。通过一个节点的主要 NID 来引用该节点。多轨路由将这一概念延续到了路由基础架构中。Lustre 2.13 版本所带来的变化：

1. 不再需要为每个网关接口配置不同的路由了。应该为每个网关配置一条路由，并根据多轨的选择标准使用网关接口。
2. 路由现在根据第 16.5 节 ``LNet Health" 状况来跟踪路由的有效性。
3. 路由器接口通过 LNet Health 进行监控。如果一个接口发生故障，将使用其他接口。
4. 路由使用 LNet 发现来定期发现网关。
5. 网关在发现其接口状态的任何变化时，会推送其接口列表。

16.4.1. 配置

16.4.1.1. 配置路由 一个网关可以在相同或不同的网络上有多个接口。使用该网关的对等节点可以通过网关的一个或多个接口到达网关。多轨路由负责管理使用哪个接口。

```
lnetctl route add --net <remote network> --gateway <NID for the gateway>
 --hops <number of hops> --priority <route priority>
```

模块参数	用法
check_routers_before_use	默认为 0。如果设置为 1，在系统进行之前，所有路由器都必须启动。
avoid_asym_router_failure	默认为 1。如果设置为 1，当且仅当网关的本地

16.4.1.2. 配置模块参数

模块参数	用法
alive_router_check_interval	默认为 60 秒。每隔 alive_router_check_interval 时间段将触发一次发现网关。如果该网关可以通过多个网络到达，则每个网络的间隔时间为 alive_router_check_interval/网络数。
router_ping_timeout	默认为 50 秒。如果网关在 router_ping_timeout + alive_router_check_interval 期间没有收到任何流量，则将关闭接口。
router_sensitivity_percentage	默认为 100。这个参数定义了网关接口对故障的敏感程度。如果设置为 100，那么任何一个网关接口的故障都会导致使用它的所有路由瘫痪。这个值越小，系统对故障的容忍度就越高。

16.4.2. 路由健康

现在，路由基础架构依靠 LNet Health 来跟踪接口的健康状况。每个网关接口都有一个与之相关的健康值。如果向其中一个接口发送失败，那么该接口的健康值就会被递减，并被放在一个恢复队列中。然后每隔lnet_recovery_interval对不健康的接口进行 ping。这个值默认为 1 秒。

如果对等体收到来自网关的消息，那么它立即假定网关的接口是正常的，并将其健康值重置为最大值。这样是必要的，确保我们能立即开始使用网关，而不是拖到接口恢复到完全健康状态。

16.4.3. 发现

LNet Discovery 用来代替 Ping 对等体。这样做两个目的：

1. 发现的通信基础设施不需要重复用于路由功能。
2. 允许将网关的接口状态变化传播给使用该网关的对等体。

对于第二点，如果一个接口的状态从 UP 变为 DOWN，或者相反，那么就会向所有可以到达的对等体发送一个发现 PUSH。这使得对等体能够更快地适应变化。

发现协议设计为向后兼容，由一个 GET 和一个 PUT 组成。GET 请求对等体的接口信息，这里使用了基本的 lnet ping。对方返回其接口信息和一个特征位。如果对等体具有多轨功能，并且发现功能启用，那么该节点将推送其接口信息。因此，两个对等体都会知道对方的接口。

对等体根据网关提供的接口状态，使用接口信息决定路由是否处于活动状态。

16.4.4. 路由有效性标准

如果以下条件成立，则认为路由处于活动状态:

1. 在本地网络上，可以通过至少一条路径到达网关。
2. 如果启用了 avoid_asym_router_failure，那么路由中定义的远程网络必须在网关上有至少一个健康的接口。

（以下由 Lustre2.12 引入）

16.5. LNet Health

多轨 LNet 实现了在同一 LNet 网络上或跨多个 LNet 网络使用多个接口的能力。LNet Health 特性为每个本地和远程接口增加了维护健康值的功能。这使得多轨算法在选择通信的接口之前可以考虑接口的健康状况。在检测到接口或网络故障时，该功能提供了跨不同接口重新发送消息的能力。这样 LNet 可以在将故障传递给上层以进行进一步的错误处理之前，缓和通信故障。为实现这个功能，LNet Health 监视发送和接收操作的状态，并根据该状态是成功还是失败，决定增加或减少接口的健康值。

16.5.1. 健康值

本地或远程接口的初始健康值设置为`LNET_MAX_HEALTH_VALUE`，该值目前为 1000。考虑到健康粒度，这个值本身可以是任意的，而不是简单的布尔状态。粒度使得多轨算法可以选择最有可能发送或接收消息的接口。

16.5.2. 故障类型和行为

LNet Health 行为取决于检测到的故障类型:

故障类型	行为
`localresend`	发生了本地故障，如找不到路由或地址解析错误。这些故障可能是暂时的，因此 LN

故障类型	行为
localno-resend	系统中出现本地不可恢复错误，如内存不足错误。在这些情况下，LNet 不会尝试重
remoteno-resend	如果 LNet 成功发送了一条消息，但该消息未完成或未收到预期的回复，则该消息会
remoteresend	还有一组故障，我们可以合理地确定消息在到达远端之前就被丢弃了。在这种情况

16.5.3. 用户接口

LNet Health 默认处于关闭状态。可用于控制 LNet Health 功能模块参数有多个。

所有模块参数都在 sysfs 中实现，位于/sys/module/lnet/parameters/。可以通过向它们回显一个值来直接设置，或在 lnetctl 中设置。

| 故障类型 | 描述 | | ------------------------- | -- | | lnet_health_sensitivity | 当 LNet 检测到特定接口上的故障时，它将按照健康灵敏度lnet_health_sensitivity来降低其健康值。lnet_health_sensitivity的值越大，界面恢复健康所需的时间就越长。其默认值设置为 0，这意味着健康值不会减少，同时表示健康功能是关闭的。灵敏度的值可以设置为大于 0。lnet_health_sensitivity为 100 意味着，连续 10 次消息失败，或稳态故障率超过 1% 会降低接口的健康值，直到该接口被禁用，而较低的故障率会引导流量绕过该接口，但它仍继续可用。当接口发生故障时，其健康值会递减，并标记该接口进行恢复。lnetctl set health_sensitivity: sensitivity to failure 0 - turn off health evaluation >0 - sensitivity value not more than 1000 | | lnet_recovery_interval | 当 LNet 在本地或远程接口上检测到故障时，它会将该接口放在恢复队列中。本地接口和远程接口各有一个恢复队列。恢复队列上的接口将在每一个lnet_recovery_interval间隔被 PING 检测一次。该值默认为 1 秒。每次成功 PING 通时，该接口的健康值将增加 1。通过配置该值，系统管理员可以控制网络上的流量。lnetctl set recovery_interval: interval to ping unhealthy interfaces >0 - timeout in seconds | | lnet_transaction_timeout | 这个超时值一定程度上是一个过载值。它具有以下功能: - 当超过lnet_transaction_timeout时间且未达到retry_count充数次数时，如果消息发送不成功，则该消息将被放弃。- 如果在lnet_transaction_timeout时间内没有收到 REPLY 或 ACK，则该 GET 或 PUT 请求会超时。该值默认为 30 秒。lnetctl set transaction_timeout: Message/Response timeout >0 - timeout in seconds**注意**下一节中描述的 LND 超时也包含在lnet_transaction_timeout内。这意味着，在预计会有很大延迟的网络中，有必要相应地增加该值。| | lnet_retry_count | 当 LNet

检测到它认为可以重新发送消息的故障时，接下来它会检查消息是否已超过指定的最大重试次数retry_count。之后，如果消息没有成功发送，故障事件才会被传递到发起消息发送的层处理。由于消息重试间隔 (lnet_lnd_timeout) 是根据lnet_transaction_timeout/lnet_retry_count计算的，因此lnet_retry_count应保持足够低，以使重试间隔不短于网络中的往返消息延迟。对于 50 秒的默认lnet_transaction_timeout，将lnet_retry_count设置为为 5 是合理的。lnetctl set retry_count: number of retries 0 - turn off retries >0 - number of retries, cannot be more than lnet_transaction_timeout | | lnet_lnd_timeout | 这不是一个可配置的参数，但它是从两个可配置的参数派生出来的: lnet_transaction_timeout和retry_count。lnet_lnd_timeout = lnet_transaction_timeout / retry_count因此，此处存在一个限制lnet_transaction_timeout >= retry_count这里假设在一个健康的网络中，发送和接收 LNet 消息不应有大的延迟。RPC 消息及其响应可能会有很大的延迟，但这是在 PtlRPC 层处理的。| ### 16.5.4. 显示信息

16.5.4.1 显示 LNet Health 配置设置 通过lnetctl global show命令，可以显示所有 LNet Health 的配置信息。

```
#> lnetctl global show
    global:
    numa_range: 0
    max_intf: 200
    discovery: 1
    retry_count: 3
    transaction_timeout: 10
    health_sensitivity: 100
    recovery_interval: 1
```

16.5.4.1 显示 LNet Health 统计信息 指定显示详细信息时，会显示 LNet Health 统计信息。要显示本地界面运行状况统计信息，请运行:

```
lnetctl net show -v 3
```

要显示远程界面运行状况统计信息，请运行:

```
lnetctl peer show -v 3
```

输出如下:

```
#> lnetctl net show -v 3
    net:
    - net type: tcp
      local NI(s):
         - nid: 192.168.122.108@tcp
           status: up
           interfaces:
               0: eth2
           statistics:
               send_count: 304
               recv_count: 284
               drop_count: 0
           sent_stats:
               put: 176
               get: 138
               reply: 0
               ack: 0
               hello: 0
           received_stats:
               put: 145
               get: 137
               reply: 0
               ack: 2
               hello: 0
           dropped_stats:
               put: 10
               get: 0
               reply: 0
               ack: 0
               hello: 0
           health stats:
               health value: 1000
               interrupts: 0
               dropped: 10
               aborted: 0
               no route: 0
```

```
                timeouts: 0
                error: 0
            tunables:
                peer_timeout: 180
                peer_credits: 8
                peer_buffer_credits: 0
                credits: 256
            dev cpt: -1
            tcp bonding: 0
            CPT: "[0]"
    CPT: "[0]"
```

有一个名为 ``health stats" 的新的 YAML 块，可以显示每个本地或远程网络接口的健康统计信息。

全局统计信息中也包含全局健康统计信息，如下所示:

```
#> lnetctl stats show
      statistics:
          msgs_alloc: 0
          msgs_max: 33
          rst_alloc: 0
          errors: 0
          send_count: 901
          resend_count: 4
          response_timeout_count: 0
          local_interrupt_count: 0
          local_dropped_count: 10
          local_aborted_count: 0
          local_no_route_count: 0
          local_timeout_count: 0
          local_error_count: 0
          remote_dropped_count: 0
          remote_error_count: 0
          remote_timeout_count: 0
          network_timeout_count: 0
          recv_count: 851
          route_count: 0
```

```
            drop_count: 10
            send_length: 425791628
            recv_length: 69852
            route_length: 0
            drop_length: 0
```

16.5.5. 推荐的初始设置

``LNet Health" 默认处于关闭状态。这意味着lnet_health_sensitivity和lnet_retry_count都设置为 0。

将lnet_health_sensitivity设置为 0 不会在出现故障时降低接口的健康值，也不会影响接口的选择行为。此外，出现故障的接口不会被放在恢复队列中。这本质上是关闭了网络健康功能。

LNet Health 设置应根据具体集群进行配置调整。但是，基本配置如下:

```
#> lnetctl global show
    global:
        numa_range: 0
        max_intf: 200
        discovery: 1
        retry_count: 3
        transaction_timeout: 10
        health_sensitivity: 100
        recovery_interval: 1
```

此设置将允许在 5 秒的事务超时时间内，最多重新发送两次失败的消息。

如果接口出现故障，健康值将减 1，接口将每 1 秒进行 PING 检测一次。

第十七章升级 Lustre 文件系统

本章介绍了 Lustre 软件版本之间的互操作性，还提供了从旧的 Lustre 2.x 软件版本升级到较新的 2.y Lustre 版本 a（主版本升级），以及从 Lustre 软件版本 2.x.y 升级到较新的 Lustre 软件版本 2.x.z（次版本升级）的步骤。

17.1. 互操作性和升级要求

Lustre software release 2.x (主要) 升级:

- 所有服务器必须在同一时间进行升级，客户端可独立于服务器进行升级。

- 所有服务器必须升级为 Lustre 软件支持的 Linux 内核。请查看 Lustre 发行日志以获取支持的 Linux 版本列表。
- 待升级的客户端必须运行兼容的 Linux 版本（查看 Lustre 发行日志）。

Lustre software release 2.x.y (次要) 升级：

- 所有服务器必须在同一时间进行升级，一些或所有客户端可随之进行升级
- 次要版本支持滚动升级，即允许在不暂停 Lustre 文件系统的情况下升级单个服务器和客户端。

17.2. 升级至 Lustre Software Release 2.x (主版本)

本节介绍如何从 Lustre 软件 2.x 版本升级到 Lustre 软件最新的 2.y 主版本。如需将 Lustre 2.x 升级至更新的主版本，请完成以下步骤：

1. 创建一个完整的、可恢复的文件系统备份。

注意

在安装 Lustre 软件之前，请备份所有数据。Lustre 软件所包含的内核更新将作用在存储设备上，如果未正确安装、配置或管理，可能会导致安全问题和数据丢失。如果无法实现文件系统的完整备份，建议您使用 MDT 文件系统的设备级备份。

2. 关闭整个文件系统。
3. 将所有服务器上的 Linux 操作系统升级至兼容版本，并重启。
4. 将所有客户端上的 Linux 操作系统升级至可兼容（测试过）的版本，并重启。
5. 从Lustre Releases目录中下载适用于您平台的 Lustre 服务器 RPMs。
6. 在所有 Lustre 服务器（MGS、MDS、OSS）上安装 Lustre 服务器软件包。

a. 使用`root`用户登录 Lustre 服务器。

b. 使用`yum`命令安装所有软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 确认所有软件包是否正确安装：

```
rpm -qa|egrep "lustre|wc"
```

d. 在每个 Lustre 服务器上重复以上步骤。

7. 从Lustre Releases目录中下载适用于您平台的 Lustre 客户端 RPM。

注意

客户端运行的内核版本必须与所安装的lustre-client-modules-ver软件包版本一致。否则，在 Lustre 客户端软件包安装前，必须安装兼容的内核版本。

8. 在每个待升级的 Lustre 客户端上安装 Lustre 客户端软件包。

a. 使用root用户登录 Lustre 客户端。

b. 使用yum命令安装所有软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 确认所有软件包是否正确安装：

```
# rpm -qa|egrep "lustre|kernel"
```

d. 在每个 Lustre 客户端上重复以上步骤。

9. DNE 功能允许在一个文件系统命名空间内使用多个 MDT，每个 MDT 可以分别服务于文件系统中的一个或多个远程子目录。根目录总是位于 MDT0 上。

请注意，运行 Lustre 软件 2.4 版本之前的客户端只能看到 MDT0 托管的命名空间，如果试图访问其他 MDT 上的目录，将返回 IO 错误。

（可选）如需格式化附加的 MDT，请完成以下步骤：

a. 确定首个 MDT 所用索引（每个 MDT 有一个唯一的索引），输入：

```
client$ lctl dl | grep mdc 36 UP mdc lustre-MDT0000-mdc-ffff88004edf3c00
4c8be054-144f-9359-b063-8477566eb84e 5
```

在这个例子中，下一个可用索引为 1。

b. 在下一个可用索引处，格式化新的块设备作为新的 MDT，输入：

```
mds# mkfs.lustre --reformat --fsname=filesystem_name --mdt
\ --mgsnode=mgsnode --index new_mdt_index /dev/mdt1_device
```

10. （可选）如您从 Lustre 2.10 之前的版本进行升级，并希望启用 project 配额功能，可在未加载时在每个 ldiskfs 后端目标上输入：

```
1 tune2fs -O project /dev/dev
```

注意

启用`project`功能将阻止文件系统使用旧版本的 ldiskfs，因此请在确实需要项目配额功能或文件系统不需要再降级的情况下启用该功能。

11. 配置文件系统，请输入：

```
conf_param $FSNAME.quota.mdt=$QUOTA_TYPE
conf_param $FSNAME.quota.ost=$QUOTA_TYPE
```

12. (可选，Lustre2.13 引入) 如果升级到 Lustre 2.13 之前格式化的 ldiskfs MDT，则默认不启用 ``宽条带" 功能。该功能允许文件有超过 160 个条带和存储其他大型 xattrs，可以通过在所有 MDT 设备上运行以下命令现有 MDT 上启用：

```
mds# tune2fs -O ea_inode /dev/mdtdev
```

更多关于宽条带的内容，请参考第 19.9 章节 ``Lustre 条带化内部参数"。

13. 按照以下顺序启动 Lustre 文件系统的各组件：

    a. 挂载 MGT，在 MGS 运行：

```
mgs# mount -a -t lustre
```

    b. 挂载 MDT，在每个 MDT 运行：

```
mds# mount -a -t lustre
```

    c. 挂载所有 OSTs，在每个 OSS 节点运行：

```
oss# mount -a -t lustre
```

注意

该命令假设`/etc/fstab`文件列出了所有的 OST。没有在`/etc/fstab`文件中列出的 OST 必须另外使用以下命令进行挂载：

```
mount -t lustre /dev/block_device/mount_point
```

    d. 在客户端上加载文件系统，请在每个客户端上运行：

```
client# mount -a -t lustre
```

14. (可选）如果你是从 Lustre 2.7 之前的版本升级的，为了使 OST 的 FID 也能存储 OST 的索引（以提高 LFSCK 和调试信息的可靠性），在挂载 OST 后，在每个 OSS 上运行一次:

```
oss# lctl set_param osd-ldiskfs.*.osd_index_in_idif=1
```

注意

启用index_in_idif功能会阻止旧版本的 Lustre 使用 OST，所以只有在知道 OST 不需要降级到较早的版本时才应该启用它。

15. 如果将一个新的 MDT 添加到文件系统中，新的 MDT 必须通过使用lfs mkdir命令创建一个或多个使用新 MDT 的新 DNE 子目录来连接到命名空间。

```
client# lfs mkdir -i new_mdt_index /testfs/new_dir
```

(Lustre 2.8 引入)

在 Lustre 2.8 及以后的版本中，可以通过创建多个条带来将一个新目录分割到多个 MDT 中:

```
client# lfs mkdir -c 2 /testfs/new_striped_dir
```

(Lustre 2.13 引入)

在 Lustre 2.13 和更高版本中，可以设置现有目录的默认条带，以便在使用率不太高的 MDT 上创建新的远程子目录:

```
client# lfs setdirstripe -c 1 -i -1 /testfs/some_dir
```

注意

文件系统进行首次加载和升级后的首次注册时必须遵循上述步骤中的所描述的挂载顺序。对于 Lustre 文件系统的普通启动，挂载顺序为 MGT、OST、MDT、客户端。

17.3. 升级至 Lustre Software Release 2.x.y (次版本)

从任一 Lustre 2.x.y 升级到更新的 Lustre 2.x.y，可使用滚动升级，即可在 Lustre 文件系统运行时，挨个升级每个服务器（或其故障切换节点）和客户端。

要将 Lustre2.x.y 升级到更新的次要版本，请完成以下步骤:

1. 创建一个完整的、可恢复的文件系统备份。

注意

在安装 Lustre 软件之前，请备份所有数据。Lustre 软件所包含的内核更新将作用在存储设备上，如果未正确安装、配置或管理，可能会导致安全问题和数据丢失。如果无法实现文件系统的完整备份，建议您使用 MDT 文件系统的设备级备份。

2. 从 Lustre Releases目录中下载适用于您平台的 Lustre 服务器 RPMs。

3. 在滚动升级中，服务器进行脱机升级，保持 Lustre 文件系统运行，并完成所需的操作，如将主服务器故障转移至备用服务器上。

4. 卸载待升级的 Lustre 服务器（MGS, MDS, OSS）。

5. 在 Lustre 服务器上安装 Lustre 服务器软件包。

a. 使用 root用户登录 Lustre 服务器。

b. 使用 yum 命令安装所有软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 确认所有软件包是否正确安装：

```
rpm -qa|egrep "lustre|wc"
```

d. 挂载 Lustre 服务器，在服务器上重启 Lustre 软件：

```
server# mount -a -t lustre
```

e. 在每个 Lustre 服务器上重复以上步骤：

6. 从 Lustre Releases目录中下载适用于您平台的 Lustre 客户端 RPMs。

7. 在每个待升级的 Lustre 客户端上安装 Lustre 客户端软件包。

a. 使用 root用户登录 Lustre 客户端。

b. 使用 yum 命令安装所有软件包：

```
# yum --nogpgcheck install pkg1.rpm pkg2.rpm ...
```

c. 确认所有软件包是否正确安装：

```
# rpm -qa|egrep "lustre|kernel"
```

d. 挂载 Lustre 服务器，在服务器上重启 Lustre 软件：

```
client# mount -a -t lustre
```

e. 在每个 Lustre 客户端上重复以上步骤。

第十八章备份和恢复文件系统

强烈建议在各站点定期执行 MDTs 设备级备份（如没有足够的容量来完成文件系统所有数据的完全备份，可每周备份两次，轮替地将不同数据备份到单独的设备上）。即使文件系统中有部分文件或所有文件的单独的文件级备份，MDT 设备级备份在 MDT 故障或损坏时非常必要。从 MDT 设备级备份进行恢复比从备份中恢复整个文件系统所需时间短得多。由于访问所有文件都需要 MDT，即使 OST 状态正常，在丢失 MDT 时，仍需要进行完整的文件系统恢复（如果可能的话）。

执行定期的 MDT 设备级备份的代价相对较小，存储只须连接到主 MDS（在需要的情况下可以手动连接到备份 MDS），并且只需要良好的线性读写性能。尽管 MDT 设备级备份不能恢复单个文件，但对于处理 MDT 故障或损坏，它是最有效的方式。

18.1. 备份文件系统

备份完整的文件系统，让您能完全控制备份哪些文件，而且可以根据需要恢复单个的文件。同时，文件系统级的备份也最容易集成到现有的备份方案中。

文件系统备份是在 Lustre 客户端（或在不同目录中并行工作的客户端）执行的，而不是在单独的服务器节点上执行的；该备份操作与其他文件系统备份没有区别。

然而，由于大多数 Lustre 文件系统体积庞大，进行完整的备份并不总是可行的。我们建议您在这种情况下备份文件系统的子集，包括整个文件系统的子目录、单个用户的文件集、按日期排序的文件等，以便更有效地完成恢复。

注意

Lustre 为所有文件内置了 128 位的文件标识符 (FID)。在与用户应用程序进行交互时，通过系统调用`stat()`、`fstat()`、`readdir()` 返回 inode 号，64 位应用程序返回 64 位 inode 号，32 位应用程序返回 32 位 inode 号。

在一些情况下，有些 32 位应用程序在用`stat()`、`fstat()`、`readdir()` 访问 Lustre 文件系统时（32 或 64 位 CPUs）可能会出现问题（尽管 Lustre 客户端应返回应用程序 32 位 inode 号）。

特别是，如果 Lustre 文件系统通过 NFS 从 64 位客户端导至 32 位客户端，那么 Linux NFS 服务器会将 64 位 inode 号导出至在 NFS 客户端上运行的应用程序。如果 32 位应用程序未使用 LFS（大文件支持）进行编译，则在访问 Lustre 文件时将返回 `EOVERFLOW` 错误。为避免此问题，Linux NFS 客户端可使用内核命令行选项"`nfs.enable_ino64=0`"强制 NFS 客户端将 32 位 inode 号导出到客户端。

解决办法：我们强烈建议使用 `tar(1)` 和其他依赖于 inode 编号的实用程序来唯一地标识在 64 位客户端上运行的 inode。128 位 Lustre 文件标识符不能一一映射到 32 位的 inode 编号，因此这些实用程序可能无法在 32 位客户端上正确运行。FID 分配模式的设计尽量避免了长时间使用中出现 inode 号码碰撞，发生 64 位 inode 号码碰撞的可能性极

小。

18.1.1. Lustre_rsync

lustre_rsync 功能通过将文件系统的更改复制到另一个文件系统（该文件系统不需要是 Lustre 文件系统，但必须足够大）来保持整个文件系统备份的同步。lustre_rsync 使用 Lustre 更新日志来高效地同步文件系统，而无需扫描（directory walk）Lustre 文件系统。这种高效率对于大型文件系统至关重要，也将lustre_rsync 功能与其他复制/备份解决方案区分开来。

18.1.1.1. Lustre_rsync 用法 lustre_rsync 功能通过定期运行 lustre_rsync （一种用于将 Lustre 文件系统中的更改同步到目标文件系统中的用户空间程序）来实现。lustre_rsync 实用程序将保留一个状态文件，使其可以在不影响文件系统之间的同步的情况下安全地中断或重启。

在使用 lustre_rsync 前:

- 注册更改日志用户
- 在注册更改日志用户之前，验证 Lustre 文件系统（源）和副本文件系统（目标）是否相同。如果文件系统不一致，请使用实用程序如常规 rsync （不是 lustre_rsync)。

第一次运行 lustre_rsync 时，用户必须指定一组参数供程序使用（下表对这些参数进行了说明)。在后续操作中，这些参数被存储在状态文件中，只须将状态文件的名称传给 lustre_rsync。

参数	说明
--source=src	将被同步的 Lustre 文件系统（源）根目录的路径。如果未指定上次同步时创建的有效状态日志 (--statuslog)，则这是一个必需选项。
--target=tgt	将被同步的 Lustre 文件系统（目标）根目录的路径。如果未指定上次同步时创建的有效状态日志 (--statuslog)，则这是一个必需选项。如果有多个同步目标，可重复指定该选项。
--mdt= mdt	将被同步的元数据设备。必须为该设备注册变更日志用户。如果未指定上次同步时创建的有效状态日志 (--statuslog)，这是一个必需选项。

参数	说明
--user=userid	指定 MDT 的变更日至用户 ID。要使用 lustre_rsync，必须注册变更日至用户。如果未指定上次同步时创建的有效状态日志（--statuslog），则这是一个必需选项。
--statuslog=log	保存同步状态的日志文件。当lustre_rsync实用程序启动时，如指定了来自先前同步操作的状态日志，则从日志中读取状态，否则将强制使用--source、--target 和--mdt。在--statuslog 选项的基础上指定--source、--target 、--mdt 将覆盖状态日志中指定的参数。命令行选项优先于状态日志中的选项。
--xattr yes|no	指定扩展属性 (xattrs) 是否进行同步。默认情况下，扩展属性进行同步。请注意，禁用 xattrs 时，Lustre 条带化信息将不会被同步。
--verbose	输出详细的信息。
--dry-run	显示lustre_rsync命令（copy, mkdir等）的输出，但并不真的在目标文件系统上执行它们。
--abort-on-err	当 lustre_rsync 操作报错时，立即停止该操作。默认情况下该操作将继续进行。

18.1.1.2. lustre_rsync 示例 lustre_rsync 命令的示例如下所示：

为某 MDT （如 testfs-MDT0000）注册更新日志用户。

```
1 # lctl --device testfs-MDT0000 changelog_register testfs-MDT0000
2 Registered changelog userid 'cl1'
```

同步 Lustre 文件系统（/mnt/lustre) 至目标文件系统（/mnt/target)。

```
1 $ lustre_rsync --source=/mnt/lustre --target=/mnt/target \
2            --mdt=testfs-MDT0000 --user=cl1 --statuslog sync.log  --verbose
3 Lustre filesystem: testfs
4 MDT device: testfs-MDT0000
5 Source: /mnt/lustre
```

```
Target: /mnt/target
Statuslog: sync.log
Changelog registration: cl1
Starting changelog record: 0
Errors: 0
lustre_rsync took 1 seconds
Changelog records consumed: 22
```

文件系统发生更改后，将更改同步到目标文件系统仅需指定statuslog 名称。该状态日志中包含了先前传递的所有参数。

```
$ lustre_rsync --statuslog sync.log --verbose
Replicating Lustre filesystem: testfs
MDT device: testfs-MDT0000
Source: /mnt/lustre
Target: /mnt/target
Statuslog: sync.log
Changelog registration: cl1
Starting changelog record: 22
Errors: 0
lustre_rsync took 2 seconds
Changelog records consumed: 42
```

同步 Lustre 文件系统（/mnt/lustre) 至两个目标文件系统（/mnt/target 和 /mnt/target2)。

```
$ lustre_rsync --source=/mnt/lustre --target=/mnt/target1 \
          --target=/mnt/target2 --mdt=testfs-MDT0000 --user=cl1  \
          --statuslog sync.log
```

18.2. 备份和恢复 MDT 或 OST （ldiskfs 设备级）

在某些情况下，在更换硬件或执行维护等操作之前，对单个设备（MDT 或 OST）进行完整的设备级备份是非常有必要的。执行完整的设备级备份可确保所有数据和配置文件均保存在原始状态，是做备份的最简单的方法。对于 MDT 文件系统，它也可能是执行备份和恢复的最快方式，因为它可以在基础设备上以最大的带宽执行大数据流读写操作。

注意

保持 MDT 的更新完整备份尤其重要，因为 MDT 文件系统的永久性故障或损坏会

使大部分存储在 OSTs 上数据（比 MDT 大得多的数据量）也无法访问和无法使用。一个或两个完整的 MDT 设备备份所需的存储比完整的文件系统备份小得多。同时，因为备份所需存储只需要具有良好的流读/写速度而不用满足高随机 IOPS，它可使用比实际的 MDT 设备更便宜的存储。

如果之所以备份是为了进行硬件替换，或者有可用的备用存储设备，则可将 MDT 或 OST 的原始副本从一个块设备复制到另一个块设备（只要新设备至少和原始设备一样大）。请运行：

```
dd if=/dev/{original} of=/dev/{newdev} bs=4M
```

如果因为硬件故障导致原始设备上出现读取问题，请运行以下命令以便从原始设备读取尽可能多的数据并跳过故障的磁盘分区：

```
dd if=/dev/{original} of=/dev/{newdev} bs=4k conv=sync,noerror /
      count={original size in 4kB blocks}
```

ldiskfs 文件系统非常强大，尽管面临硬件故障，仍可能通过在新设备上运行 e2fsck -fy /dev/{newdev}来恢复文件系统数据。

在 **Lustre 2.6** 之后，LFSCK 扫描会在目录损坏后自动将对象从 lost+found 移回 OST 的正确位置上。

为确保备份完全一致，必须卸载 MDT 或 OST，以避免在传输数据时设备上的任何更改。如果备份的原因是预防性的（即在正在运行的 MDS 上进行 MDT 备份以预防将来可能出现的故障），则可从 LVM 快照执行一致的备份。如果 LVM 快照不可用，且 MDS 脱机备份不可接受，则可从原始的 MDT 块设备执行备份。尽管从原始设备的备份可能因为正在进行的更改而不会完全一致，但绝大多数的 ldiskfs 元数据都是静态分配的。备份中的不一致部分可通过在备份设备上运行 e2fsck 来解决。不管如何，这仍然比不进行任何备份好得多。

18.3. 备份 OST 或 MDT （后端文件系统级）

此过程提供了文件级的 OST 或 MDT 数据备份或迁移的替代方法。在文件级别，未使用的空间被忽略，且可使用较小的总备份大小以便更快地完成该过程。备份单个 OST 设备不一定是执行 Lustre 文件系统备份的最佳方式，因为如果没有在 MDT 上存储元数据以及可能存在于其他 OST 上的额外的文件条带，存储在备份中的文件将无法使用。但它却是 OST 设备迁移的首选方法，尤其是在需要使用不同配置选项重新格式化底层文件系统或减少碎片时。

注意

由于 Lustre 存储了内部元数据，用于通过对象索引文件将 FIDs 映射到本地的 inode 编号，因此在检测到恢复后的第一次挂载时需要重建这些元数据，以便支持文件级 MDT 备份和恢复。在检测到恢复后的第一次挂载时，OI Scrub 会自动重建这些文件（可能影响

挂载后的MDT性能)，直到重建完成。通过`lctl get_param osd-*.*.oi_scrub`可以在恢复目标文件系统的 MDS 或 OSS 节点上监控进度。

18.3.1. 备份 OST 或 MDT（后端文件系统级）

在 **Lustre 2.11.0** 之前，我们只能为基于 ldiskfs 的系统执行后端文件系统级别的备份和恢复过程。Lustre-2.11.0 引入了基于 ZFS 的 MDT/OST 文件系统级备份和恢复的功能。与基于 ldiskfs 的系统不同，索引对象必须在卸载目标（MDT 或 OST）之前进行备份以便能够成功恢复文件系统。要在目标上启用索引备份（`index_backup`)，请在目标服务器上执行以下命令：`# lctl set_param osd-zfs.${fsname}-${target}.index_backup=1`。其中，`${target}`由目标类型（MDT 或 OST）加上目标索引，如 MDT0000、OST0001 等。**注意**

`index_backup`对于基于 ldiskfs 的系统也是有效的，用于基于 ldiskfs 和 ZFS 的系统之间的数据迁移。

18.3.2. 备份 OST 或 MDT

下面的示例展示了如何备份一个 OST 文件系统。如需备份 MDT，请在以下的命令中将 `ost` 替换成 `mdt` 。

1. **卸载目标。**

2. **为文件系统上创建挂载点。**

```
[oss]# mkdir -p /mnt/ost
```

3. **挂载文件系统。**

基于 ldiskfs 的系统：

```
[oss]# mount -t ldiskfs /dev/{ostdev} /mnt/ost
```

基于 zfs 的系统：

a. 如果目标已导出，请导入目标池。例如：

```
[oss]# zpool import lustre-ost [-d ${ostdev_dir}]
```

b. 在目标文件系统上启用 `canmount` 属性。例如：

```
[oss]# zfs set canmount=on ${fsname}-ost/ost
```

也可指定挂载点属性，默认情况下为：`/${fsname}-ost/ost`

c. 将目标挂载为'zfs'，如：

```
[oss]# zfs mount ${fsname}-ost/ost
```

4. **切换至将被备份的挂载点：**

```
[oss]# cd /mnt/ost
```

5. **备份附加属性。**

```
[oss]# getfattr -R -d -m '.*' -e hex -P . > ea-$(date
+%Y%m%d).bak
```

注意

如果 tar(1) 命令支持 --xattr 选项，只要 tar 正确备份了 trusted.* 属性，则 getfattr 步骤就不是必需的。然而，这一步骤不仅不会有任何害处，还能作为附加的安全措施。

在大部分发行版本中，getfattr 命令是 attr 包的一部分。如果 getfattr 命令返回如 Operation not supported 的错误，则内核没有正确地支持 EAs 运行。请使用另一种不同的备份方法。

6. **确认 ea-$date.bak 文件已经成功备份了 OST 上的 EA 数据。**

如果没有此属性数据，MDT 还原过程将失败并导致文件系统不可用，OST 恢复过程可能会丢失在日后文件系统损坏时大有用处的额外数据。可使用 more 或文本编辑器查看此文件。每个对象文件有一个对应的类似于如下内容的项目：

```
1  [oss]# file: O/0/d0/100992
2  trusted.fid= \
3  0x0d822200000000004a8a73e500000000808a0100000000000000000000000000
```

7. **备份所有文件系统数据。**

```
1  [oss]# tar czvf {backup file}.tgz [--xattrs]
       [--xattrs-include="trusted.*" --sparse .
```

注意

对于备份 MDT 来说，tar 的 --sparse 选项至关重要。在非常老的版本中，tar 可能不能很好地支持 --sparse 选项，这将使得 MDT 备份需要很长的时间。已知版本包括了 Red Hat Enterprise Linux (RHEL 6.3) tar 或更新版本，GNU tar 1.25 或更新版本。

tar --xattrs 选项只在 GNU tar 1.27 或 RHEL 6.3 以及更新版本中可用，为完成正确的恢复，请使用 --xattrs-include="trusted.*" 。

8. **将目录切换到文件系统之外。**

```
[oss]# cd -
```

9. **卸载文件系统。**

```
[oss]# umount /mnt/ost
```

注意

作为 OST 迁移的一部分，在其他节点上恢复 OST 备份时，还必须更改服务器 NID 并使用 `--writeconf` 命令重新生成配置日志。

18.4. 恢复文件级备份

要从文件级备份还原数据，您需要格式化设备，然后还原文件数据和 EA 数据。

1. 格式化新设备。

```
[oss]# mkfs.lustre --ost --index {OST index} --replace
--fstype=${fstype} {other options} /dev/{newdev}
```

2. 设置文件系统标签 (**仅针对基于 ldiskfs 的系统**)。

```
[oss]# e2label {fsname}-OST{index in hex} /mnt/ost
```

3. 挂载文件系统。

基于 ldiskfs 的系统：

```
[oss]# mount -t ldiskfs /dev/{newdev} /mnt/ost
```

基于 zfs 的系统：

a. 如果目标已导出，请导入目标池。例如：

```
[oss]# zpool import lustre-ost [-d ${ostdev_dir}]
```

b. 在目标文件系统上启用 `canmount` 属性。例如：

```
[oss]# zfs set canmount=on ${fsname}-ost/ost
```

也可指定挂载点属性，默认情况下为：`/${fsname}-ost/ost`

c. 将目标挂载为'zfs'，如：

```
[oss]# zfs mount ${fsname}-ost/ost
```

4. 切换至将被备份的挂载点。

```
[oss]# cd /mnt/ost
```

5. 恢复文件系统备份。

```
[oss]# tar xzvpf {backup file} [--xattrs]
[--xattrs-include="trusted.*"] --sparse
```

注意

tar `--xattrs` 选项只在 GNU tar 1.27 或 RHEL 6.3 以及更新版本中可用，为完成正确的恢复，请使用 `--xattrs-include="trusted.*"` 。在其它情况下，请执行 `setfattr` 步骤。

6. 如果未使用支持 xattr 备份的 tar 版本，请恢复文件系统扩展属性。

```
[oss]# setfattr --restore=ea-${date}.bak
```

注意

如果 tar 支持`--xattrs`选项，且在之前的步骤中指定了该选项，那么此步骤是多余的。

7. 确认扩展属性已完成恢复。

```
1 [oss]# getfattr -d -m ".*" -e hex O/0/d0/100992 trusted.fid= \
2 0x0d822200000000004a8a73e500000000808a0100000000000000000000000000
```

8. 移除旧的 OI 和 LFSCK 文件。

```
[oss]# rm -rf oi.16* lfsck_* LFSCK
```

9. 移除旧的 CATALOGS 文件。

```
[oss]# rm -f CATALOGS
```

注意

在 MDT 端，为可选步骤。CATALOGS 记录用于恢复跨服务器更新的 llog 文件处理程序。在 OI scrub 为 llog 文件重建 OI 映射前，如果相关恢复的运行速度比 OI scrub 速度快，则恢复失败。这将导致整个挂载过程失败。OI scrub 是一个在线工具，因此安装失败意味着 OI scrub 停止。删除旧的 CATALOGS 将避免这种潜在的问题。副作用是，跨服务器相关更新的恢复将被中止。但好在，系统挂载完成后可通过 LFSCK 进行处理。

10. 将目录切换到文件系统之外。

```
1 [oss]# cd -
```

11. 卸载新的文件系统。

```
[oss]# umount /mnt/ost
```

注意

如果恢复的系统有不同于备份系统的 NID，请更改其 NID，例如：

```
[oss]# mount -t lustre -o nosvc ${fsname}-ost/ost /mnt/ost
[oss]# lctl replace_nids ${fsname}-OSTxxxx $new_nids
[oss]# umount /mnt/ost
```

12. 挂载目标为 `lustre`。

通常我们会使用 `-o abort_recov` 选项来跳过不必要的恢复操作，如：

```
[oss]# mount -t lustre -o abort_recov #{fsname}-ost/ost /mnt/ost
```

Luste 可以在挂载目标时自动检测还原，然后触发 OI scrub 以在后台异步重建 OI 和索引对象。您可以使用以下命令检查 OI scrub 状态：

```
[oss]# lctl get_param -n osd-${fstype}.${fsname}-${target}.oi_scrub
```

如果您在备份和恢复之间的时间段使用文件系统，则会自动运行在线的 LFSCK 工具（版本 2.3 之后的 Lustre 代码的一部分）以确保文件系统的一致性。如果在整个 Lustre 文件系统停止后同时备份所有设备文件系统，则没有必要进行此步骤。无论哪种情况，文件系统都会立即生效。尽管可能会存在读取 MDT 文件的 I/O 错误，但不会存在读取 OST 文件的 I/O 错误。在 MDT 备份后创建的文件将无法被访问或不可见。

18.5. 使用 LVM 快照

如果要执行基于磁盘的备份（例如，当需要访问备份系统速度与主要 Lustre 文件系统一样快时），则可以使用 Linux LVM 快照工具来维护多个增量文件系统备份。

由于 LVM 快照需要花费数个 CPU 周期来处理新文件的写入，使用 Lustre 文件系统的快照可能会导致难以忍受的性能损失。建议您创建一个新的备份的 Lustre 文件系统，并定期（如每晚）备份新的或更改后的文件。使用此备份文件系统的定期快照来创建一系列" 完整" 备份。

注意

创建 LVM 快照并不像创建单独的备份那样可靠，因为 LVM 快照与主 MDT 设备共享同一磁盘，并且依赖主 MDT 设备来获取其大部分数据。如果主 MDT 设备故障，快照也可能随之损坏。

18.5.1. 创建基于 LVM 的备份文件系统

用 LVM 快照机制创建备份的 Lustre 文件系统，程序如下：

1. 为 MDT 和 OSTs 创建 LVM 卷。

为您的 MDT 和 OST 目标创建 LVM 设备。请勿将整个磁盘作为目标并为快照节省一些空间。快照起始大小为 0，但随着对当前文件系统的更改而增加。如果您希望在不同备份之间更改 20% 的文件系统，则最新快照将为目标大小的 20%，第二新的快照将为目标大小的 40%，以此类推。下面是一个示例：

```
cfs21:~# pvcreate /dev/sda1
   Physical volume "/dev/sda1" successfully created
cfs21:~# vgcreate vgmain /dev/sda1
   Volume group "vgmain" successfully created
cfs21:~# lvcreate -L200G -nMDT0 vgmain
   Logical volume "MDT0" created
cfs21:~# lvcreate -L200G -nOST0 vgmain
   Logical volume "OST0" created
cfs21:~# lvscan
   ACTIVE                   '/dev/vgmain/MDT0' [200.00 GB] inherit
   ACTIVE                   '/dev/vgmain/OST0' [200.00 GB] inherit
```

2. 格式化 LVM 卷为目标 Lustre。

在这个例子中，备份文件系统为 `main` ，指代当前的最新的备份。

```
cfs21:~# mkfs.lustre --fsname=main --mdt --index=0 /dev/vgmain/MDT0
 No management node specified, adding MGS to this MDT.
    Permanent disk data:
 Target:     main-MDT0000
 Index:      0
 Lustre FS:  main
 Mount type: ldiskfs
 Flags:      0x75
               (MDT MGS first_time update )
 Persistent mount opts: errors=remount-ro,iopen_nopriv,user_xattr
 Parameters:
checking for existing Lustre data
```

```
   device size = 200GB
   formatting backing filesystem ldiskfs on /dev/vgmain/MDT0
           target name  main-MDT0000
           4k blocks     0
           options        -i 4096 -I 512 -q -O dir_index -F
   mkfs_cmd = mkfs.ext2 -j -b 4096 -L main-MDT0000  -i 4096 -I 512 -q
    -O dir_index -F /dev/vgmain/MDT0
   Writing CONFIGS/mountdata
cfs21:~# mkfs.lustre --mgsnode=cfs21 --fsname=main --ost --index=0
/dev/vgmain/OST0
      Permanent disk data:
   Target:     main-OST0000
   Index:      0
   Lustre FS:  main
   Mount type: ldiskfs
   Flags:      0x72
                 (OST first_time update )
   Persistent mount opts: errors=remount-ro,extents,mballoc
   Parameters: mgsnode=192.168.0.21@tcp
checking for existing Lustre data
   device size = 200GB
   formatting backing filesystem ldiskfs on /dev/vgmain/OST0
           target name  main-OST0000
           4k blocks     0
           options        -I 256 -q -O dir_index -F
   mkfs_cmd = mkfs.ext2 -j -b 4096 -L lustre-OST0000 -J size=400 -I 256
    -i 262144 -O extents,uninit_bg,dir_nlink,huge_file,flex_bg -G 256
    -E resize=4290772992,lazy_journal_init, -F /dev/vgmain/OST0
   Writing CONFIGS/mountdata
cfs21:~# mount -t lustre /dev/vgmain/MDT0 /mnt/mdt
cfs21:~# mount -t lustre /dev/vgmain/OST0 /mnt/ost
cfs21:~# mount -t lustre cfs21:/main /mnt/main
```

18.5.2. 备份新的/更改后的文件

定期（如每晚）将新文件和更改后的文件备份到基于 LVM 的备份文件系统。

```
cfs21:~# cp /etc/passwd /mnt/main

cfs21:~# cp /etc/fstab /mnt/main

cfs21:~# ls /mnt/main
fstab  passwd
```

18.5.3. 创建快照卷

无论何时您想要创建主要 Lustre 文件系统的" 检查点"，都可以在基于 LVM 的备份文件系统中创建所有目标 MDT 和 OST 的 LVM 快照。您必须事先决定快照的最大大小，但可以稍后进行动态更改。每日快照的大小取决于主要 Lustre 文件系统中每日发生变更的数据量。两天的快照很可能会是一天快照的两倍。

如卷组中有空间，可创建尽可能多的快照。如有必要，可动态地将磁盘添加到卷组。

目标 MDT 和 OST 的快照应在同一时间点生成。更新备份文件系统的 cronjob 是唯一写入磁盘的东西，请确保它没有运行。示例如下：

```
cfs21:~# modprobe dm-snapshot
cfs21:~# lvcreate -L50M -s -n MDT0.b1 /dev/vgmain/MDT0
  Rounding up size to full physical extent 52.00 MB
  Logical volume "MDT0.b1" created
cfs21:~# lvcreate -L50M -s -n OST0.b1 /dev/vgmain/OST0
  Rounding up size to full physical extent 52.00 MB
  Logical volume "OST0.b1" created
```

快照生成后，您可以继续备份新的或更改后的文件至"main"。快照不会包含这些新文件。

```
cfs21:~# cp /etc/termcap /mnt/main
cfs21:~# ls /mnt/main
fstab  passwd  termcap
```

18.5.4. 从快照恢复文件系统

请参照以下程序从 LVM 快照恢复文件系统。

1. 重命名 LVM 快照。

将文件系统快照从"main" 重命名为"back"，以便在不卸载"main" 的情况下挂载"back"。该操作不是必需的（虽然我们推荐这么做），可通过设置 `tunefs.lustre` 的 `--reformat` 标志来强制名称更改。例如：

```
cfs21:~# tunefs.lustre --reformat --fsname=back --writeconf
    /dev/vgmain/MDT0.b1
 checking for existing Lustre data
 found Lustre data
 Reading CONFIGS/mountdata
Read previous values:
 Target:     main-MDT0000
 Index:      0
 Lustre FS:  main
 Mount type: ldiskfs
 Flags:      0x5
              (MDT MGS )
 Persistent mount opts: errors=remount-ro,iopen_nopriv,user_xattr
 Parameters:
Permanent disk data:
 Target:     back-MDT0000
 Index:      0
 Lustre FS:  back
 Mount type: ldiskfs
 Flags:      0x105
              (MDT MGS writeconf )
 Persistent mount opts: errors=remount-ro,iopen_nopriv,user_xattr
 Parameters:
Writing CONFIGS/mountdata
cfs21:~# tunefs.lustre --reformat --fsname=back --writeconf
    /dev/vgmain/OST0.b1
 checking for existing Lustre data
 found Lustre data
 Reading CONFIGS/mountdata
Read previous values:
 Target:     main-OST0000
 Index:      0
 Lustre FS:  main
 Mount type: ldiskfs
 Flags:      0x2
              (OST )
```

```
35     Persistent mount opts: errors=remount-ro,extents,mballoc
36     Parameters: mgsnode=192.168.0.21@tcp
37   Permanent disk data:
38     Target:     back-OST0000
39     Index:      0
40     Lustre FS:  back
41     Mount type: ldiskfs
42     Flags:      0x102
43                   (OST writeconf )
44     Persistent mount opts: errors=remount-ro,extents,mballoc
45     Parameters: mgsnode=192.168.0.21@tcp
46   Writing CONFIGS/mountdata
```

重命名文件系统时，必须从快照中擦除 last_rcvd 文件。

```
1   cfs21:~# mount -t ldiskfs /dev/vgmain/MDT0.b1 /mnt/mdtback
2   cfs21:~# rm /mnt/mdtback/last_rcvd
3   cfs21:~# umount /mnt/mdtback
4   cfs21:~# mount -t ldiskfs /dev/vgmain/OST0.b1 /mnt/ostback
5   cfs21:~# rm /mnt/ostback/last_rcvd
6   cfs21:~# umount /mnt/ostback
```

2. 从 LVM 快照挂载文件系统，如：

```
1   cfs21:~# mount -t lustre /dev/vgmain/MDT0.b1 /mnt/mdtback
2   cfs21:~# mount -t lustre /dev/vgmain/OST0.b1 /mnt/ostback
3   cfs21:~# mount -t lustre cfs21:/back /mnt/back
```

3. 注意截至快照时间的原目录内容。例如：

```
1   cfs21:~/cfs/b1_5/lustre/utils# ls /mnt/back
2   fstab  passwds
```

18.5.5. 删除旧的快照

要回收磁盘空间，请按照备份策略的要求删除旧快照，运行；

```
1 lvremove /dev/vgmain/MDT0.b1
```

18.5.6. 更改快照卷大小

如果您发现每日增量小于或大于预期，您还可以扩展或收缩快照卷，运行：

```
lvextend -L10G /dev/vgmain/MDT0.b1
```

注意

在更老的 LVM 版本中，扩展快照卷可能不可用。该功能在 LVM v2.02.01 正常。

18.6. ZFS 和 ldiskfs 目标文件系统间的迁移

从 **Lustre 2.11.0** 开始，可以在 ZFS 和 ldiskfs 后端之间进行迁移。要迁移 OST，最好使用 `lfs find/lfs_migrate` 在文件系统正在使用时清空 OST，然后使用新的 fstype 重新格式化 OST。

18.6.1. 从 ZFS 迁移至 ldiskfs 文件系统

第一步，请按照本章第 3 节" 备份 OST 或 MDT（后端文件系统级别）" 中介绍的方法使用 tar 进行 ZFS 后端备份。第二步，请将备份恢复到基于 ldiskfs 的系统，参照第 4 节" 恢复文件级备份"。

18.6.2. 从 ldiskfs 迁移至 ZFS 文件系统

第一步，请按照本章第 3 节" 备份 OST 或 MDT（后端文件系统级别）" 中介绍的方法使用 tar 进行 ldiskfs 后端备份。第二步，请将备份恢复到基于 ZFS 的系统，参照本章第 4 节" 恢复文件级备份"。

注意

对于从 ldiskfs 到 zfs 的迁移，需要在卸载目标之前启用 index_backup。这是基于 ldiskfs 常规备份/恢复的一个附加步骤，很容易被忽略。

第十九章管理文件布局（条带化）及剩余空间

19.1. Lustre 文件系统条带化如何工作

在 Lustre 文件系统中，MDS 使用循环算法或加权算法将对象分配给 OST。当可用空间大小平衡良好时（默认情况下，不同 OST 之间的空闲空间相差不到 17%即算平衡良好），循环算法用于选择要写入条带的下一个 OST。MDS 定期调整条带布局以消除一些算法退化的情况，如创建非常规律的、总是偏好序列中某个特定 OST 的文件布局（条带化类型）。

OST 的使用通常非常均衡。但是，如果用户创建一些特大文件或指定错误的条带参数，将可能会导致 OST 的用量不均衡。当 OST 之间的可用空间相差超过特定数量（默

认为 17%）时，MDS 将使用加权随机分配，从而优先在拥有更多可用空间的 OST 上分配对象。（这会影响 I/O 性能，直到空间使用再次平衡。）有关如何分配条带的更详细说明，请参见本章第 6 节" 管理可用空间"。

受限于存储在 MDT 上的属性所允许的最大大小，文件只能在有限数量的 OST 上进行条带化。如果 MDT 基于 ldiskfs，而又不具备 `ea_inode` 功能，文件最多可以分为 160 个 OSTs。如果是基于 ZFS 的 MDT，或者如果基于 ldiskfs 的 MDT 启用了 `ea_inode` 功能（从 Lustre 2.13.0 开始默认启用），文件做多可以条带化到 2000 个 OST 上。有关更多信息，请参见本章第 7 节"Lustre 条带化内部参数"。

19.2. Lustre 文件布局（条带化）的一些考量

是否设置文件条带、选择什么样的参数值取决于您的需求。原则上您应该在满足需求的前提下尽可能地在更少的对象上进行条带化。

进行文件条带化的一些动机包括：

- **提供高带宽访问。** 许多应用程序都需要对某个文件进行高带宽访问，其对带宽的需求可能比单个 OSS 能提供的带宽要高。比如一些应用程序可能需要将来自数百个节点的数据写入单个文件，或者在启动时需要从多个节点加载二进制可执行文件。

在这些情况下，可将文件分割到尽可能多的 OSS 上，以达到该文件所需的峰值聚合带宽。请注意，只有当文件大小很大或文件一次被许多节点访问时，才建议使用大量 OSS 进行分条。目前，Lustre 文件可以在多达 2000 个 OST 上进行条带化。

- **超出 OSS 带宽时用于提升性能。** 如果客户端总带宽超过服务器带宽，且应用程序数据读写速率足够快而能够充分利用额外的 OSS 带宽，则跨越多个 OSS 将文件条带化可以提高性能。最大有效条带数的限制为：客户端/作业的 I/O 率除以每个 OSS 性能。

(Lustre 2.13 引入)

将条带与 IO 模式相匹配。当多个节点对一个文件同时进行写入时，即有一个以上的客户端写入一个条带，即使需要操作的 IO 部分没有重合，也会导致锁交换的问题，即客户端争夺对该条带的写入。如果 IO 也能相应条带化，使每个条带只由一个客户端访问，那么就能避免这个争夺情况。从 Lustre 2.13 开始，'overstripe' 功能使可用的，允许每个 OST 包含多个条带。这对于线程数量超过 OST 数的情况特别有用，可以使得条带数量与线程数量相匹配.

- **为大文件提供空间。**当单个 OST 没有足够多的空闲空间来存放整个文件时，可将文件分条。

减少或避免使用条带化的原因：

- **增加开销。** 在常规操作（如 `stat` 和 `unlink` ）期间，条带化会导致更多的锁定和额外的网络操作。即使这些操作并行执行，一次网络操作所花的时间也少于 100 次操作。

同时，服务器竞争情况也会随之增加。考虑一个拥有 100 个客户端和 100 个 OSS 的集群，每个 OSS 含一个 OST。如果每个文件只有一个对象并且负载均匀分布，每台服务器上的磁盘都可以管理线性的 I/O，则不存在竞争。如果每个文件都有 100 个对象，那么客户端就会彼此竞争以获得服务器的注意，并且每个节点上的磁盘将在 100 个不同的方向上寻找，导致不必要的竞争。

- **增加风险。** 当文件在所有服务器上进行条带化，而其中一台服务器出现故障，这些文件的一小部分将丢失。相反，如果每个文件只有一个条带，丢失的文件会更少，但它们将完全丢失。许多用户更能接受丢失部分文件（即使是全部内容），而不是所有文件都丢失部分内容。

19.2.1. 选择条带大小

选择条带大小是一种权衡行为。下面将介绍较为合理的默认值。条带大小对于单条带文件没有影响。

- **条带大小必须是页大小的整数倍。**Lustre 软件工具将强制执行 64KB 的整数倍（ia64 和 PPC64 节点的最大页大小），避免页规格较小的平台上的用户创建可能会导致 ia64 客户端出现问题的文件。
- **推荐的最小条带大小是 512KB。** 虽然可以创建条带大小为 64KB 的文件，但最小的实际条带大小为 512KB，因为 Lustre 文件系统通过网络发送数据块大小为 1MB。选择更小的条带大小可能会导致磁盘 I/O 效率低下，性能下降。
- **适用于高速网络线性 I/O 的条带大小在 1MB 到 4MB 之间。**在大多数情况下，大于 4MB 的条带大小可能导致更长的锁定保持时间，增加共享文件访问期间的争用情况。
- **最大条带大小为 4GB。** 在访问非常大的文件时，使用较大的条带大小可以提高性能。它允许每个客户端独占访问文件的一部分。但如果条带大小与 I/O 模式不匹配，较大的条带大小可能会适得其反。
- **选择一个考虑到应用程序的写入模式的条带化模式。** 跨越对象边界的写入效率要比在单个服务器上完整写入的效率略低。如果文件以一致且对齐的方式写入，请将条带大小设置为 `write()` 大小的整数倍。

19.3. 配置 Lustre 文件布局（条带化模式）(lfs setstripe)

使用 lfs setstripe 命令创建指定文件布局（条带化模式）配置的新文件。

```
lfs setstripe [--size|-s stripe_size] [--stripe-count|-c stripe_count]
    [--overstripe-count|-C stripe_count] \
[--index|-i start_ost] [--pool|-p pool_name] filename|dirname
```

stripe_size

stripe_size 表示移动到下一个 OST 前向现有 OST 写入的数据量。默认的 stripe_size 是 1 MB。将该参数设置为 0，则会使用默认的条带大小。stripe_size 值必须是 64 KB 的整数倍。

stripe_count (--stripe-count, --overstripe-count)

stripe_count 表示要使用的条带数量。默认stripe_count值为 1。将其设置为 0，则会使用该默认条带数量。将 stripe_count 设置为 -1 意味着对所有可用的 OST(跳过满 OST) 进行条带化。如果使用了 --overstripe-count 参数，则必要时会对每个 OST 条带化。

start_ost

start_ost 是文件写入的第一个 OST。start_ost 的默认值是 -1，它允许 MDS 选择起始索引。强烈建议使用此默认设置，因为它可根据需要通过 MDS 完成空间和负载均衡。如果将 start_ost 的值设置为非 -1，则该文件将从指定的 OST 索引开始。OST 索引编号从 0 开始。

注意

如果指定的 OST 处于非活动状态或处于降级模式，则 MDS 将自动选择另一个目标。

如果 start_ost 值为 0，stripe_count 值为 1，则所有文件都将写入 OST0，直到空间耗尽。这很可能不是你想要的。如果您只希望调整 stripe_count ，而保持其他参数为默认设置，请不要指定任何其他参数：

```
client# lfs setstripe -c stripe_count filename
```

pool_name

pool_name 指定文件将写入的 OST 池。这可以将使用的 OST 数量限制为文件系统中所有 OST 的子集。有关使用 OST 池的更多详细信息，请参阅创建和管理 OST 池。

19.3.1. 为单个文件指定文件布局（条带化模式）

使用lfs setstripe 创建新文件，可以指定其文件布局。用户可以覆盖文件系统的默认参数，从而更好地调整文件布局以适应其应用程序。如果文件已经存在，执行 lfs setstripe 是无效的。

19.3.1.1. 设置条带大小 创建一个指定条带大小的新文件的命令类似于：

```
[client]# lfs setstripe -s 4M /mnt/lustre/new_file
```

该示例创建了新文件：`/mnt/lustre/new_file`，条带大小为 4MB。

当文件创建时，新的条带设置生效，条带大小为 4MB 的新文件将在单个 OST 上被创建。

```
 [client]# lfs getstripe /mnt/lustre/new_file
/mnt/lustre/4mb_file
lmm_stripe_count:   1
lmm_stripe_size:    4194304
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  1
obdidx     objid       objid           group
1          690550       0xa8976         0
```

19.3.1.2. 设置条带数 下面的命令创建了一个条带数为 -1 的新文件，表明条带数为所有可用 OST 的数量：

```
[client]# lfs setstripe -c -1 /mnt/lustre/full_stripe
```

下面的例子表明文件 `full_stripe` 在配置中的所有的六个活动 OST 上被条带化：

```
[client]# lfs getstripe /mnt/lustre/full_stripe
/mnt/lustre/full_stripe
  obdidx   objid   objid   group
  0        8       0x8     0
  1        4       0x4     0
  2        5       0x5     0
  3        5       0x5     0
  4        4       0x4     0
  5        2       0x2     0
```

与 3.1.1 " 设置条带大小" 中的输出不同，这里显示的是每个文件的单个对象。

19.3.2. 为目录指定文件布局（条带化模式）

在目录中，`lfs setstripe` 命令为目录下文件设置默认的条带化配置。其用法与常规文件的 `lfs setstripe` 相同，但在设置默认条带配置之前该目录必须存在。如

果在默认条带配置的目录中创建新文件（没有另外指定条带配置），则 Lustre 文件系统将使用该配置中的参数而不是文件系统默认值。

要更改子目录的条带化模式，请按上述方法为新目录设置您希望的文件布局。子目录继承根/父目录的文件布局。

19.3.3. 为文件系统指定文件布局（条带化模式）

在根目录的条带化配置决定了文件系统中创建的所有新文件的条带化配置，除非有优先级更高的条带化配置进行重载（例如应用程序、父目录，或 `lfs setstripe`命令指定的条带化布局）。

注意

除非为子目录指定了条带设置，否则根目录的条带设置默认应用于在根目录中创建的任何新的子目录。

19.3.4. 在指定 OST 上创建文件

您可以使用 `lfs setstripe` 在特定的 OST 上创建文件。在以下示例中，文件`file1`在第一个 OST（OST 索引为 0）上创建。

```
$ lfs setstripe --stripe-count 1 --index 0 file1
$ dd if=/dev/zero of=file1 count=1 bs=100M
1+0 records in
1+0 records out

$ lfs getstripe file1
/mnt/testfs/file1
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  0
      obdidx    objid   objid    group
      0         37364   0x91f4   0
```

19.4. 检索文件布局/条带信息 (`getstripe`)

`lfs getstripe`命令用于显示文件被分发到哪些 OST 上、每个 OST 的索引和 UUID，以及文件中每个条带的 OST 索引和对象 ID。在目录下运行该命令将显示在该目录中创建的文件的默认设置。

19.4.1. 显示当前条带大小

想要看 Lustre 文件或目录的当前条带大小，请使用 `lfs getstripe` 命令。例如，查看某一目录的相关信息，请输入：

```
1 [client]# lfs getstripe /mnt/Lustre
```

输出为：

```
1 /mnt/Lustre
2 (Default) stripe_count: 1 stripe_size: 1M stripe_offset: -1
```

在这个例子中，默认的条带数是 1 （数据块在单个 OST 上分条），默认条带大小为 1MB，对象在所有可用 OSTs 上创建。

查看某一目录的相关信息，请输入：

```
1 $ lfs getstripe /mnt/lustre/foo
2 /mnt/lustre/foo
3 lmm_stripe_count:   1
4 lmm_stripe_size:    1048576
5 lmm_pattern:        1
6 lmm_layout_gen:     0
7 lmm_stripe_offset:  0
8   obdidx   objid    objid      group
9   2        835487   m0xcbf9f   0
```

在这个例子中，该文件位于`obdidx 2`，对应于 OST`lustre-OST0002`。查看服务该 OST 的节点，运行：

```
1 $ lctl get_param osc.lustre-OST0002-osc.ost_conn_uuid
2 osc.lustre-OST0002-osc.ost_conn_uuid=192.168.20.1@tcp
```

19.4.2. 搜索文件树

要搜索整个文件树，请使用 `lfs find` 命令：

```
1 lfs find [--recursive | -r] file|directory ...
```

19.4.3. 为远程目录定位 MDT

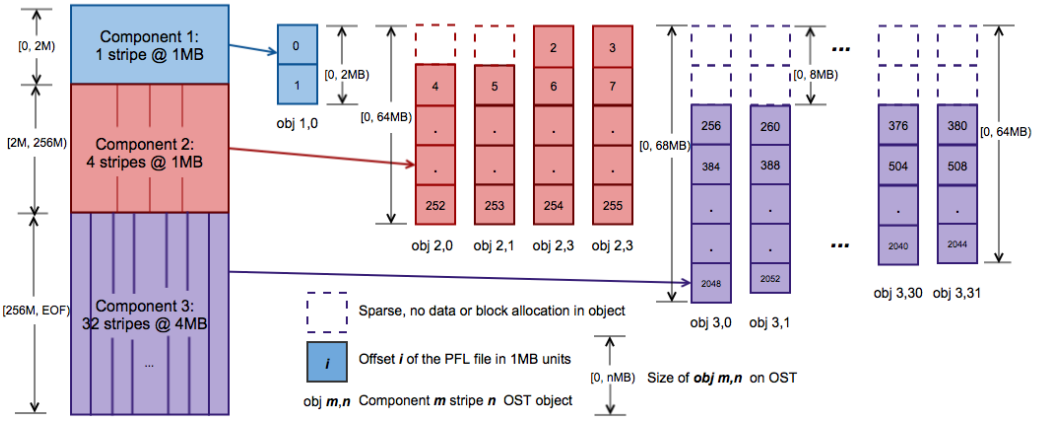
Lustre 可以在同一个文件系统中配置多个 MDT。每个目录和文件可以位于不同的 MDT 上。要确定给定子目录位于哪个 MDT 上，请将 `getstripe [--mdt-index|-M]` 的参数传递给 `lfs`。第 14.9.1 节 ``从文件系统中删除 MDT'' 中给出了一个该命令的示例。

19.5. 渐进式文件布局 (PFL)

Lustre 渐进式文件布局（Lustre Progressive File Layout，PFL）功能简化了 Lustre 的使用，使得用户无需事先明确了解其 IO 模型或 Lustre 使用细节就可以预期各种常规文件 IO 模式的性能。特别是，用户不一定需要在创建输出文件之前就知道其大小或并行性，也不需要为了实现并行共享单个大文件 IO 和更小的每进程文件 IO 的高性能而为每个文件明确地指定最佳布局。

PFL 文件的布局以复合布局的方式存储在磁盘上。PFL 文件基本上是一个子布局组件的数组，每个子布局组件都是一个覆盖不同的不重叠的文件部分的普通布局。对于 PFL 文件，文件布局由一系列组件组成，因此可能有某些文件部分未由任何组件描述。

以下的 PFL 对象映射图显示了 PFL 文件的数据块映射到 OST 对象组件的示例：



Mapping from 2055MB PFL file data blocks to OST objects of three components

图 10: Lustrecluster at scale

图中的 PFL 文件包含 3 个组件，显示了一个大小为 2055MB 的文件中不同块的映射。前两个组件的条带大小为 1MB，第三个组件的条带大小为 4MB。三个组件的条带数在不断增加。第一个组件只有两个 1MB 的块，一个对象的大小为 2MB。第二个组件将文件接下来的 254MB 保存在 RAID-0 的 4 个独立的 OST 对象上，每个对象的大小为 256MB/4= 64MB。请注意，前两个对象 `obj 2,0` 和 `obj 2,1` 在存储时起始位置处有一个 1MB 大小的空洞。最后的组件存有文件接下来的 1800MB，覆盖了 32 个 OST 对象。每个对象在开始处有 256MB/32 = 8MB 的空洞。每个对象的大小为 2048MB/32= 64MB，不同之处在于 obj 3,0 包含额外的 4MB 块，而 obj 3,1 包含额外的 3MB 块。如果将更多数据写入文件，只有第三个组件中的对象的大小会增加。

当访问具有已定义但未实例化组件的文件范围时，客户端向 MDT 发送一个布局意图 RPC，MDT 将实例化覆盖该范围的组件的对象。

接下来我们将介绍用于操作 PFL 文件的一些命令，并给出一些合成布局的例子。Lustre 提供命令`lfs setstripe` 和 `lfs migrate` 以供用户对 PFL 文件进行操作。其中，`lfs setstripe` 用于创建 PFL 文件，将组件添加到现有组合文件或从现有组合文

件中删除组件；`lfs migrate` 命令将当前 OST 中的数据复制到新 OST 中，使用新布局参数重新布局现有文件中的数据。另外，`lfs getstripe` 命令用于列出给定 PFL 文件的条带化/组件信息，`lfs find` 命令可用于搜索以给定的目录或文件为根的目录树，以查找与 PFL 组件参数相匹配的文件。

注意

使用 PFL 文件需要客户端和服务器都能解析 PFL 文件布局，**Lustre 2.9** 或更早版本中没有该功能。但这不影响更早版本的客户端访问文件系统中的非 PFL 文件。

19.5.1. `lfs setstripe`

`lfs setstripe` 命令用于创建 PFL 文件，将组件添加到现有组合文件或从现有组合文件中删除组件。（在下面的例子中，我们假设有 8 个 OST，默认条带大小为 1MB。）

19.5.1.1. 创建一个 PFL 文件 命令

```
1 lfs setstripe
2 [--component-end|-E end1] [STRIPE_OPTIONS]
3 [--component-end|-E end2] [STRIPE_OPTIONS] ... filename
```

`-E` 选项用于指定每个组件的结束偏移量（以字节为单位或使用后缀"kMGTP"，如 256M），同时也指示了 `STRIPE_OPTIONS`用于此组件。每个组件在 [start，end）范围内定义文件的条带化模式。第一个组件必须从偏移量 0 开始，所有组件必须彼此相邻，不允许有空洞，因此每个范围都将从上一个范围的末尾开始。`-1`为结束偏移，或用 `eof`表示这是一直延伸到文件结尾的最后一个组件。

示例

```
1 $ lfs setstripe -E 4M -c 1 -E 64M -c 4 -E -1 -c -1 -i 4 \
2 /mnt/testfs/create_comp
```

该命令创建一个具有如下图所示复合布局的文件。第一个组件有 1 个条带，覆盖 [0,4M]，第二个组件有 4 个条带，覆盖 [4M，64M]，最后一个组件从 OST4 开始，跨越所有可用的 OST 并覆盖 [64M，EOF]。

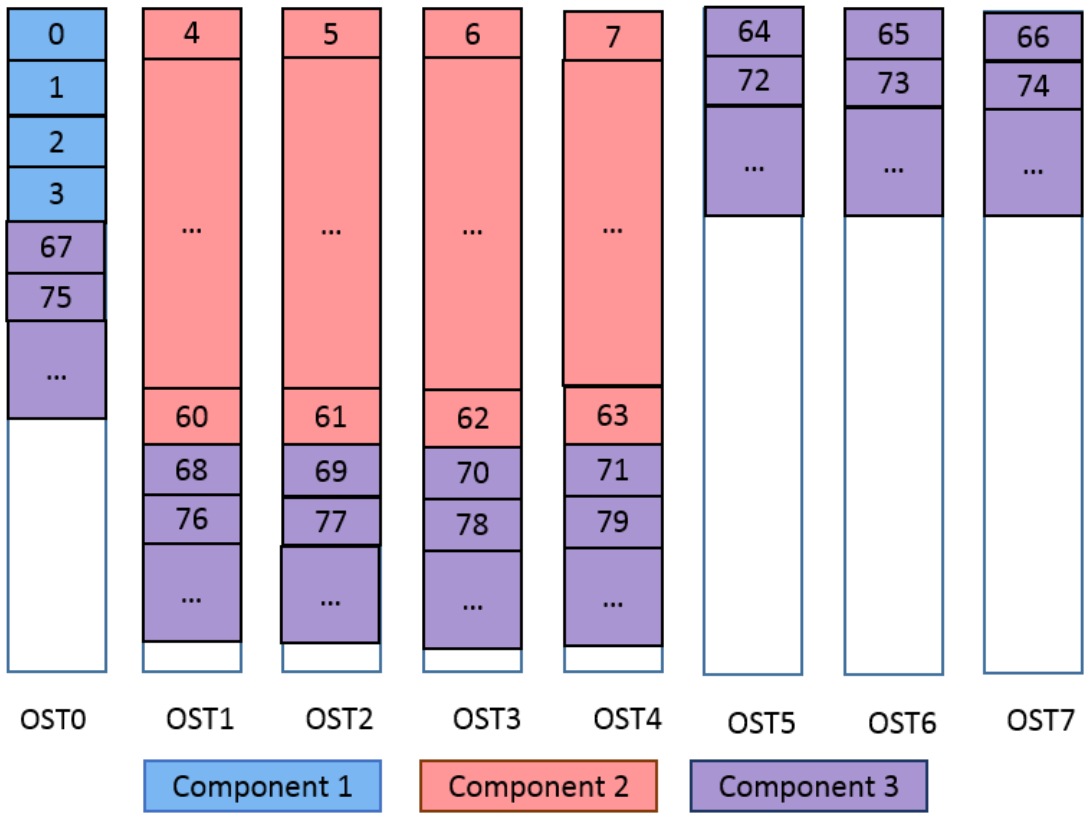


图 11: Lustrecluster at scale

该组合布局可通过以下命令显示：

```
$ lfs getstripe /mnt/testfs/create_comp
/mnt/testfs/create_comp
  lcm_layout_gen:  3
  lcm_entry_count: 3
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }

    lcme_id:             2
```

```
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   67108864
      lmm_stripe_count:  4
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
    lcme_id:             3
    lcme_flags:          0
    lcme_extent.e_start: 67108864
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  -1
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
```

注意

当设置文件布局时，只有 PFL 文件的第一个组件的 OST 对象被实例化。其他对象的实例化操作将延迟到稍后的写入或截断操作。

如果我们向这个 PFL 文件写入 128M 数据，第二个和第三个组件将被实例化：

```
$ dd if=/dev/zero of=/mnt/testfs/create_comp bs=1M count=128
$ lfs getstripe /mnt/testfs/create_comp
/mnt/testfs/create_comp
  lcm_layout_gen:  5
  lcm_entry_count: 3
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
```

```
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }

  lcme_id:             2
  lcme_flags:          init
  lcme_extent.e_start: 4194304
  lcme_extent.e_end:   67108864
    lmm_stripe_count:  4
    lmm_stripe_size:   1048576
    lmm_pattern:       1
    lmm_layout_gen:    0
    lmm_stripe_offset: 1
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x2:0x0] }
    - 1: { l_ost_idx: 2, l_fid: [0x100020000:0x2:0x0] }
    - 2: { l_ost_idx: 3, l_fid: [0x100030000:0x2:0x0] }
    - 3: { l_ost_idx: 4, l_fid: [0x100040000:0x2:0x0] }

  lcme_id:             3
  lcme_flags:          init
  lcme_extent.e_start: 67108864
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  8
    lmm_stripe_size:   1048576
    lmm_pattern:       1
    lmm_layout_gen:    0
    lmm_stripe_offset: 4
    lmm_objects:
    - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x3:0x0] }
    - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }
    - 2: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
    - 3: { l_ost_idx: 7, l_fid: [0x100070000:0x2:0x0] }
    - 4: { l_ost_idx: 0, l_fid: [0x100000000:0x3:0x0] }
    - 5: { l_ost_idx: 1, l_fid: [0x100010000:0x3:0x0] }
    - 6: { l_ost_idx: 2, l_fid: [0x100020000:0x3:0x0] }
    - 7: { l_ost_idx: 3, l_fid: [0x100030000:0x3:0x0] }
```

19.5.1.2. 在现有组合布局文件中增加组件 命令

```
lfs setstripe --component-add
[--component-end|-E end1] [STRIPE_OPTIONS]
[--component-end|-E end2] [STRIPE_OPTIONS] ... filename
```

--component-add 选项用于将组件添加到现有的组合布局文件中。要添加的第一个组件范围的开始点等于当前文件中最后一个组件范围结束点，所有要添加的组件必须相邻。

注意

如果最后一个现有组建被指定为 -E -1 或 -E eof，表明它覆盖了文件的结尾。添加一个新组件之前必须删除该组件。

示例

```
$ lfs setstripe -E 4M -c 1 -E 64M -c 4 /mnt/testfs/add_comp
$ lfs setstripe --component-add -E -1 -c 4 -o 6-7,0,5 \
/mnt/testfs/add_comp
```

该命令添加了一个新组件，该组件从最后一个现有组件的末尾开始，到文件末尾结束。下图说明了此示例的布局。最后一个组件的条带跨越了 4 个 OST，顺序为 OST6、OST7、OST0、OST5，覆盖 [64M，EOF)。

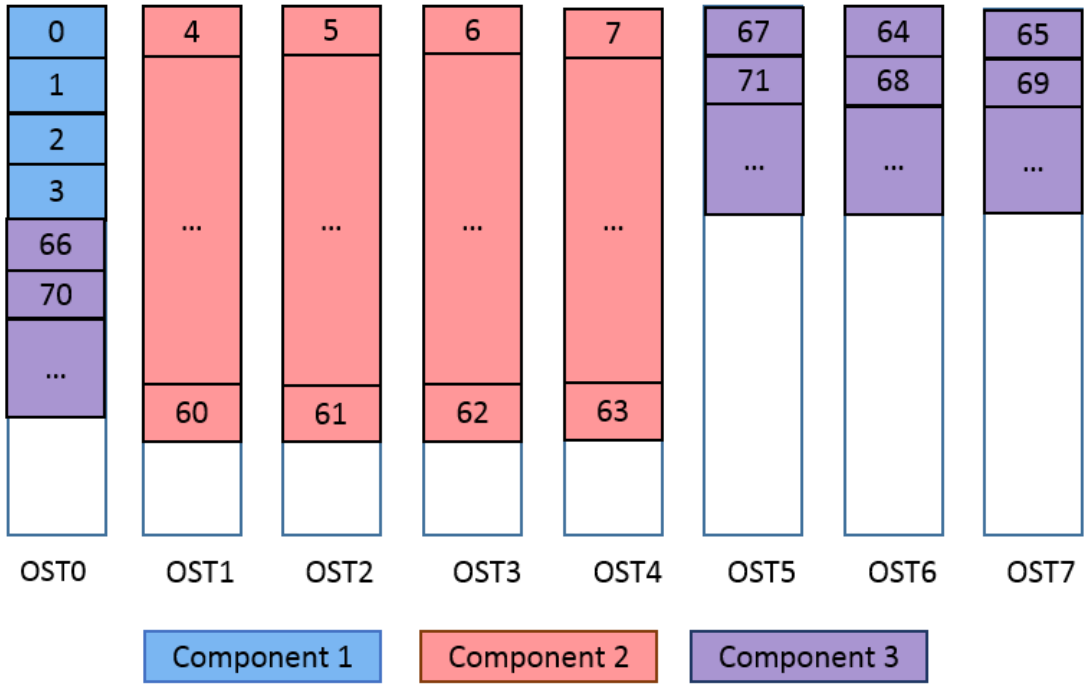


图 12: Lustrecluster at scale

该布局可通过以下命令显示：

```
$ lfs getstripe /mnt/testfs/add_comp
/mnt/testfs/add_comp
```

```
lcm_layout_gen:  5
lcm_entry_count: 3
  lcme_id:             1
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   4194304
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       1
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }

  lcme_id:             2
  lcme_flags:          init
  lcme_extent.e_start: 4194304
  lcme_extent.e_end:   67108864
    lmm_stripe_count:  4
    lmm_stripe_size:   1048576
    lmm_pattern:       1
    lmm_layout_gen:    0
    lmm_stripe_offset: 1
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x2:0x0] }
    - 1: { l_ost_idx: 2, l_fid: [0x100020000:0x2:0x0] }
    - 2: { l_ost_idx: 3, l_fid: [0x100030000:0x2:0x0] }
    - 3: { l_ost_idx: 4, l_fid: [0x100040000:0x2:0x0] }

  lcme_id:             5
  lcme_flags:          0
  lcme_extent.e_start: 67108864
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  4
    lmm_stripe_size:   1048576
    lmm_pattern:       1
```

```
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
```

组件 ID "lcme_id" 随着布局更改而变化。组件 ID 不一定是线性的，也就是说它与组件的顺序无关。

注意

与在文件创建时指定全文件组合布局类似，`--component-add` 不会实例化 OST 对象，实例化将被延迟到稍后的写入或截断操作。例如，写入文件最后一个组件的 64MB 后，新组件的对象完成了分配：

```
$ lfs getstripe -I5 /mnt/testfs/add_comp
/mnt/testfs/add_comp
  lcm_layout_gen:  6
  lcm_entry_count: 3
    lcme_id:             5
    lcme_flags:          init
    lcme_extent.e_start: 67108864
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  4
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 6
      lmm_objects:
      - 0: { l_ost_idx: 6, l_fid: [0x100060000:0x4:0x0] }
      - 1: { l_ost_idx: 7, l_fid: [0x100070000:0x4:0x0] }
      - 2: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
      - 3: { l_ost_idx: 5, l_fid: [0x100050000:0x4:0x0] }
```

19.5.1.3. 从现有文件中删除组件　命令

```
lfs setstripe --component-del
[--component-id|-I comp_id | --component-flags comp_flags]
filename
```

`--component-del` 选项用于从现有文件中删除指定组件 ID 或标志的组件。此操作将导致存储在已删除组件中的所有数据都将丢失。

由 `-I` 选项指定的 ID 是唯一的组件数字 ID，可通过命令`lfs getstripe -I` 命令获取。由 `--component-flags`选项指定的标志是某种类型的组件，可通过 `lfs`

getstripe --component-flags获得。目前只有两个标志 init 和 ^init ，分别用于实例化的组件和未实例化的组件。

注意

由于不允许空洞的存在。删除必须从最后一个组件开始。

示例

```
$ lfs getstripe -I /mnt/testfs/del_comp
1

2

5

$ lfs setstripe --component-del -I 5 /mnt/testfs/del_comp
```

该示例删除了文件 /mnt/testfs/del_comp 的 component 5。

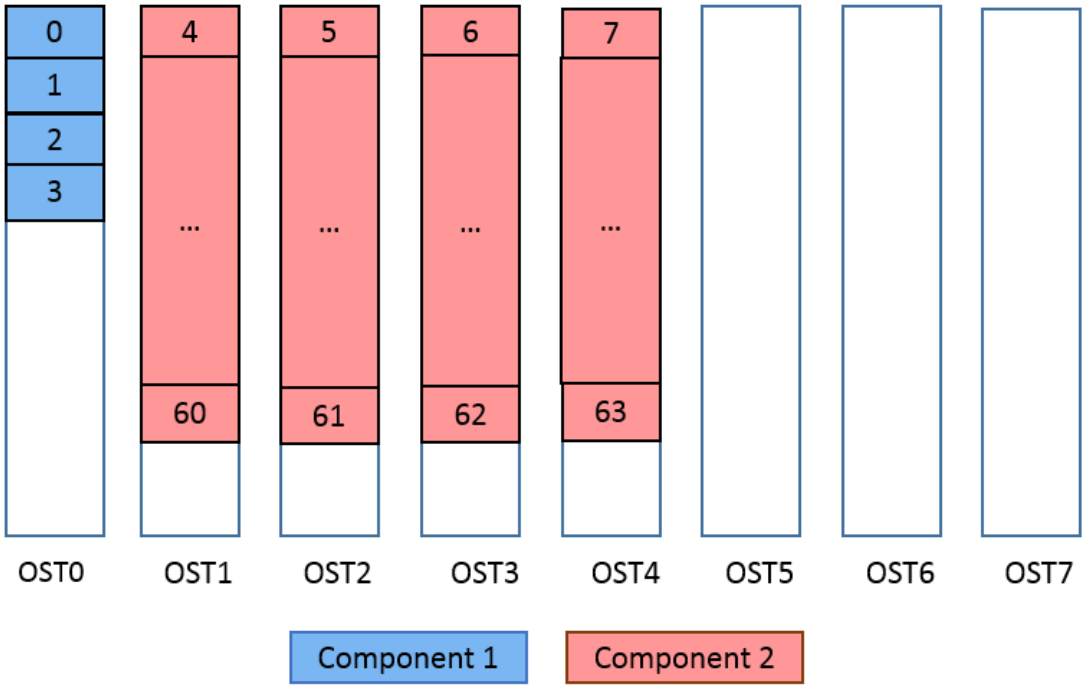


图 13: Lustrecluster at scale

如果你尝试删除不是最末尾的组件，会产生如下错误：

```
$ lfs setstripe -component-del -I 2 /mnt/testfs/del_comp
Delete component 0x2 from /mnt/testfs/del_comp failed. Invalid argument
error: setstripe: delete component of file '/mnt/testfs/del_comp' failed:
    Invalid argument
```

19.5.1.4. 为现有目录设置默认 PFL 布局 与创建 PF L 文件类似，您可以为现有目录设置默认 PFL 布局。目录下创建的所有文件将默认继承此布局。

命令

```
lfs setstripe
[--component-end|-E end1] [STRIPE_OPTIONS]
[--component-end|-E end2] [STRIPE_OPTIONS] ... dirname
```

示例

```
$ mkdir /mnt/testfs/pfldir
$ lfs setstripe -E 64M -c 2 -i 0 -E -1 -c 4 -i 0 /mnt/testfs/pfldir
```

运行 `lfs getstripe`:

```
$ lfs getstripe /mnt/testfs/pfldir
/mnt/testfs/pfldir
  lcm_layout_gen:  0
  lcm_entry_count: 3
    lcme_id:             N/A
    lcme_flags:          0
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      stripe_count:  1       stripe_size:   1048576       stripe_offset: -1
    lcme_id:             N/A
    lcme_flags:          0
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   17179869184
      stripe_count:  4       stripe_size:   1048576       stripe_offset: -1
    lcme_id:             N/A
    lcme_flags:          0
    lcme_extent.e_start: 17179869184
    lcme_extent.e_end:   EOF
      stripe_count:  -1       stripe_size:   4194304       stripe_offset: -1
```

如果你在 `/mnt/testfs/pfldir`目录下创建新的文件，则该文件的布局将从父目录那继承两个组件:

```
$ touch /mnt/testfs/pfldir/pflfile
$ lfs getstripe /mnt/testfs/pfldir/pflfile
/mnt/testfs/pfldir/pflfile
  lcm_layout_gen:  2
  lcm_entry_count: 3
    lcme_id:             1
```

```
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   268435456
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 1
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0xa:0x0] }

  lcme_id:             2
  lcme_flags:          0
  lcme_extent.e_start: 268435456
  lcme_extent.e_end:   17179869184
    lmm_stripe_count:  4
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1

  lcme_id:             3
  lcme_flags:          0
  lcme_extent.e_start: 17179869184
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  -1
    lmm_stripe_size:   4194304
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

注意

`lfs setstripe --component-add/del` 无法在目录上运行。目录中的默认布局类似于配置，可通过 `lfs setstripe` 对其进行任意更改，而文件中的布局可能会附加数据（OST 对象）。如果您想要删除目录中的默认布局，请运行 `lfs setstripe -d dirname`：

```
$ lfs setstripe -d /mnt/testfs/pfldir
$ lfs getstripe -d /mnt/testfs/pfldir
/mnt/testfs/pfldir
stripe_count:  1 stripe_size:   1048576 stripe_offset: -1
/mnt/testfs/pfldir/commonfile
lmm_stripe_count:  1
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 0
    obdidx       objid       objid       group
         0               9           0x9                0
```

19.5.2. `lfs migrate`

`lfs migrate` 命令用于将现有 OST 中的数据复制到新 OST 中设置新的布局参数，从而重新布局现有文件中的数据。

命令

```
lfs migrate [--component-end|-E comp_end] [STRIPE_OPTIONS] ...
filename
```

`migrate` 和 `setstripe` 的区别在于 `migrate` 用于重新布局现有文件的数据，而 `setstripe` 用于按照指定的布局创建新的文件。

示例

例 1. 普通布局到组合布局的迁移

```
$ lfs setstripe -c 1 -S 128K /mnt/testfs/norm_to_2comp
$ dd if=/dev/urandom of=/mnt/testfs/norm_to_2comp bs=1M count=5
$ lfs getstripe /mnt/testfs/norm_to_2comp --yaml
/mnt/testfs/norm_to_comp
lmm_stripe_count:  1
lmm_stripe_size:   131072
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 7
lmm_objects:
      - l_ost_idx: 7
        l_fid:     0x100070000:0x2:0x0
```

```
$ lfs migrate -E 1M -S 512K -c 1 -E -1 -S 1M -c 2 \
/mnt/testfs/norm_to_2comp
```

在这个例子中，一个只有一个条带、条带大小为 128K 的 5MB 大小的文件被迁移至有两个组件的组合布局文件中，如下图所示：

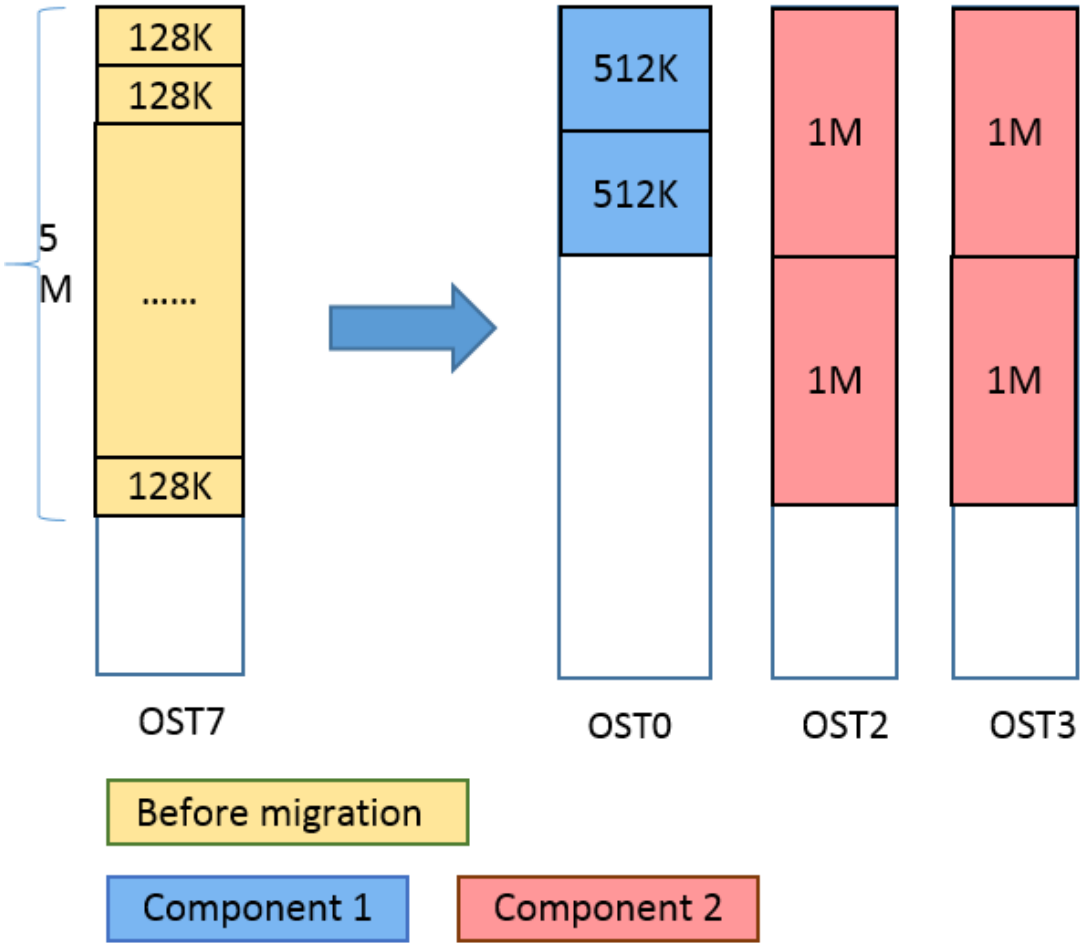


图 14: Lustrecluster at scale

迁移后的条带信息如下：

```
$ lfs getstripe /mnt/testfs/norm_to_2comp
/mnt/testfs/norm_to_2comp
  lcm_layout_gen:  4
  lcm_entry_count: 2
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  1
      lmm_stripe_size:   524288
```

```
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }

    lcme_id:             2
    lcme_flags:          init
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 2
      lmm_objects:
      - 0: { l_ost_idx: 2, l_fid: [0x100020000:0x2:0x0] }
      - 1: { l_ost_idx: 3, l_fid: [0x100030000:0x2:0x0] }
```

例 2. 一个组合布局到另一个组合布局的迁移

```
$ lfs setstripe -E 1M -S 512K -c 1 -E -1 -S 1M -c 2 \
/mnt/testfs/2comp_to_3comp
$ dd if=/dev/urandom of=/mnt/testfs/norm_to_2comp bs=1M count=5
$ lfs migrate -E 1M -S 1M -c 2 -E 4M -S 1M -c 2 -E -1 -S 3M -c 3 \
/mnt/testfs/2comp_to_3comp
```

下图显示了两个组件的组合布局文件到三个组件布局文件的迁移。

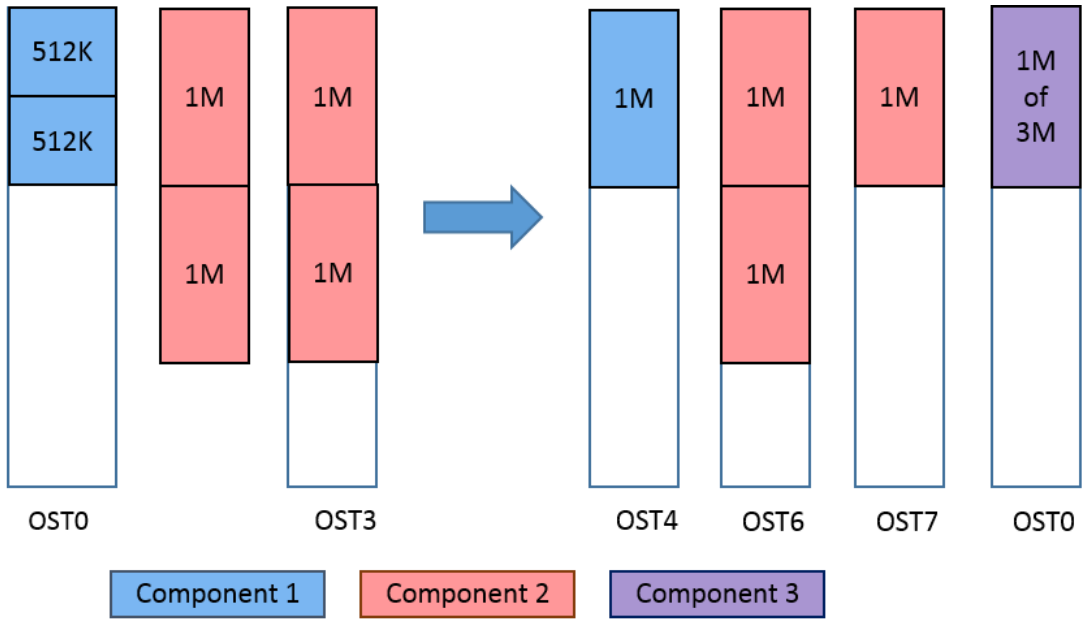


图 15: Lustrecluster at scale

条带信息如下：

```
$ lfs getstripe /mnt/testfs/2comp_to_3comp
/mnt/testfs/2comp_to_3comp
  lcm_layout_gen:  6
  lcm_entry_count: 3
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x2:0x0] }
      - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }

    lcme_id:             2
    lcme_flags:          init
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   4194304
```

```
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 6
      lmm_objects:
      - 0: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
      - 1: { l_ost_idx: 7, l_fid: [0x100070000:0x3:0x0] }

    lcme_id:             3
    lcme_flags:          init
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  3
      lmm_stripe_size:   3145728
      lmm_pattern:       1
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x3:0x0] }
      - 1: { l_ost_idx: 1, l_fid: [0x100010000:0x2:0x0] }
      - 2: { l_ost_idx: 2, l_fid: [0x100020000:0x3:0x0] }
```

例 3. 组合布局到普通布局的迁移

```
$ lfs migrate -E 1M -S 1M -c 2 -E 4M -S 1M -c 2 -E -1 -S 3M -c 3 \
/mnt/testfs/3comp_to_norm
$ dd if=/dev/urandom of=/mnt/testfs/norm_to_2comp bs=1M count=5
$ lfs migrate -c 2 -S 2M /mnt/testfs/3comp_to_normal
```

下图显示了 3 个组件的组合布局文件到普通布局文件（2 个条带，条带大小为 2M）的迁移：

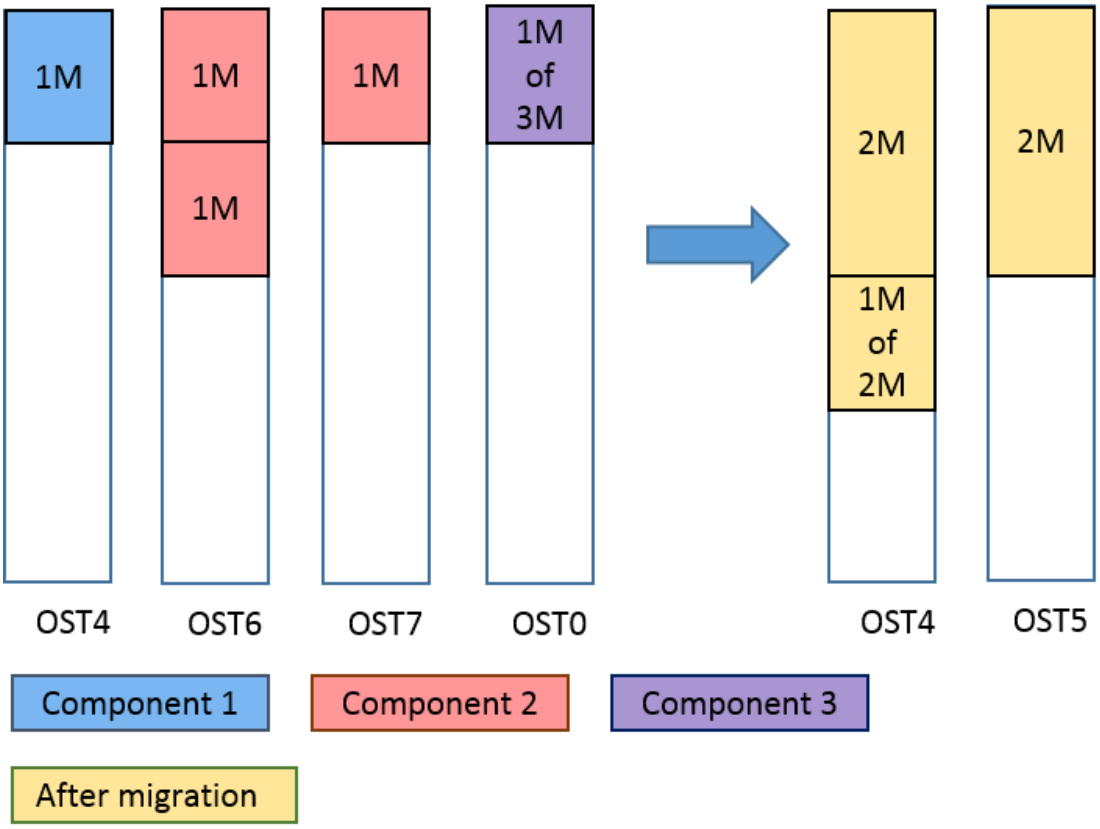


图 16: Lustrecluster at scale

条带信息如下：

```
$ lfs getstripe /mnt/testfs/3comp_to_norm --yaml
/mnt/testfs/3comp_to_norm
lmm_stripe_count:  2
lmm_stripe_size:   2097152
lmm_pattern:       1
lmm_layout_gen:    7
lmm_stripe_offset: 4
lmm_objects:
      - l_ost_idx: 4
        l_fid:     0x100040000:0x3:0x0
      - l_ost_idx: 5
        l_fid:     0x100050000:0x3:0x0
```

19.5.3. `lfs getstripe`

`lfs getstripe` 命令可用于列出给定 PFL 文件的条带化/组件信息。这里只显示 PFL 文件的新参数。

命令

```
lfs getstripe
[--component-id|-I [comp_id]]
[--component-flags [comp_flags]]
[--component-count]
[--component-start [+-][N][kMGTPE]]
[--component-end|-E [+-][N][kMGTPE]]
dirname|filename
```

示例

假设我们的组合文件/mnt/testfs/3comp 是由以下命令创建的：

```
$ lfs setstripe -E 4M -c 1 -E 64M -c 4 -E -1 -c -1 -i 4 \
/mnt/testfs/3comp
```

并写入数据：

```
$ dd if=/dev/zero of=/mnt/testfs/3comp bs=1M count=5
```

例 1. 列出组件 ID 及相关信息

- 列出所有组件 ID

```
$ lfs getstripe -I /mnt/testfs/3comp
1
2
3
```

- 列出组件 ID 为 2 的所有详细条带信息

```
$ lfs getstripe -I2 /mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             2
lcme_flags:          init
lcme_extent.e_start: 4194304
lcme_extent.e_end:   67108864
lmm_stripe_count:  4
lmm_stripe_size:   1048576
```

```
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 5
lmm_objects:
- 0: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }
- 1: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
- 2: { l_ost_idx: 7, l_fid: [0x100070000:0x2:0x0] }
- 3: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }
```

- 列出组件 ID 为 2 的条带大小和位移

```
$ lfs getstripe -I2 -i -c /mnt/testfs/3comp
lmm_stripe_count:  4
lmm_stripe_offset: 5
```

例 2. 列出含指定标志的组件

- 列出每个组件的标志

```
$ lfs getstripe -component-flag -I /mnt/testfs/3comp
lcme_id:             1
lcme_flags:          init
lcme_id:             2
lcme_flags:          init
lcme_id:             3
lcme_flags:          0
```

- 列出没有实例化的组件

```
$ lfs getstripe --component-flags=^init /mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             3
lcme_flags:          0
lcme_extent.e_start: 67108864
lcme_extent.e_end:   EOF
lmm_stripe_count:  -1
```

```
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    4
lmm_stripe_offset: 4
```

例 3. 列出所有组件数量

- 列出所有组件数量

```
$ lfs getstripe --component-count /mnt/testfs/3comp
3
```

例 4. 列出指定范围起始点和结束点的组件

- 列出每个组件起始位置字节数

```
$ lfs getstripe --component-start /mnt/testfs/3comp
0
4194304
67108864
```

- 列出组件 ID 为 3 的起始位置字节数

```
$ lfs getstripe --component-start -I3 /mnt/testfs/3comp
67108864
```

- 列出起始位置为 64M 的组件

```
$ lfs getstripe --component-start=64M /mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             3
lcme_flags:          0
lcme_extent.e_start: 67108864
lcme_extent.e_end:   EOF
lmm_stripe_count:  -1
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    4
lmm_stripe_offset: 4
```

- 列出起始位置大于 5M 的组件

```
$ lfs getstripe --component-start=+5M /mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             3
lcme_flags:          0
lcme_extent.e_start: 67108864
lcme_extent.e_end:   EOF
lmm_stripe_count:  -1
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    4
lmm_stripe_offset: 4
```

- 列出起始位置小于 5M 的组件

```
$ lfs getstripe --component-start=-5M /mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             1
lcme_flags:          init
lcme_extent.e_start: 0
lcme_extent.e_end:   4194304
lmm_stripe_count:  1
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 4
lmm_objects:
- 0: { l_ost_idx: 4, l_fid: [0x100040000:0x2:0x0] }

lcme_id:             2
lcme_flags:          init
lcme_extent.e_start: 4194304
```

```
lcme_extent.e_end:   67108864
lmm_stripe_count:  4
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 5
lmm_objects:
- 0: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }
- 1: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
- 2: { l_ost_idx: 7, l_fid: [0x100070000:0x2:0x0] }
- 3: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }
```

- 列出起始位置在 3M 和 70M 之间的组件

```
$ lfs getstripe --component-start=+3M --component-end=-70M \
/mnt/testfs/3comp
/mnt/testfs/3comp
lcm_layout_gen:  4
lcm_entry_count: 3
lcme_id:             2
lcme_flags:          init
lcme_extent.e_start: 4194304
lcme_extent.e_end:   67108864
lmm_stripe_count:  4
lmm_stripe_size:   1048576
lmm_pattern:       1
lmm_layout_gen:    0
lmm_stripe_offset: 5
lmm_objects:
- 0: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }
- 1: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
- 2: { l_ost_idx: 7, l_fid: [0x100070000:0x2:0x0] }
- 3: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }
```

19.5.4. `lfs find`

`lfs find` 命令可用于搜索以给定的目录或文件为根的目录树，以查找与 PFL 组件参数相匹配的文件。这里只显示 PFL 文件的新参数。其用法与 `lfs getstripe` 命令类似。

命令

```
lfs find directory|filename
[[!] --component-count [+-=]comp_cnt]
[[!] --component-start [+-=]N[kMGTPE]]
[[!] --component-end|-E [+-=]N[kMGTPE]]
[[!] --component-flags=comp_flags]
```

注意

使用 `--component-xxx` 选项，只搜索组合文件。使用 `! --component-xxx` 选项，搜索所有文件。

示例

以下面的目录和组合文件为例显示 `lfs find` 如何工作。

```
$ mkdir /mnt/testfs/testdir
$ lfs setstripe -E 1M -E 10M -E eof /mnt/testfs/testdir/3comp
$ lfs setstripe -E 4M -E 20M -E 30M -E eof /mnt/testfs/testdir/4comp
$ mkdir -p /mnt/testfs/testdir/dir_3comp
$ lfs setstripe -E 6M -E 30M -E eof /mnt/testfs/testdir/dir_3comp
$ lfs setstripe -E 8M -E eof /mnt/testfs/testdir/dir_3comp/2comp
$ lfs setstripe -c 1 /mnt/testfs/testdir/dir_3comp/commnfile
```

例 1. 查找与指定组件计数情况相符的文件

查找目录 /mnt/testfs/testdir 下组件个数不为 3 的文件。

```
$ lfs find /mnt/testfs/testdir ! --component-count=3
/mnt/testfs/testdir
/mnt/testfs/testdir/4comp
/mnt/testfs/testdir/dir_3comp/2comp
/mnt/testfs/testdir/dir_3comp/commonfile
```

例 2. 查找与指定组件起始/结束点情况相符的文件或目录

查找目录 /mnt/testfs/testdir 下组件起始点在 4M 和 70M 之间的文件和目录

```
$ lfs find /mnt/testfs/testdir --component-start=4M -E -30M
/mnt/testfs/testdir/4comp
```

例 3. 查找与指定组件标志情况相符的文件或目录

查找目录 /mnt/testfs/testdir 下组件标志含 `init`的文件和目录。

```
$ lfs find /mnt/testfs/testdir --component-flag=init
/mnt/testfs/testdir/3comp
/mnt/testfs/testdir/4comp
/mnt/testfs/testdir/dir_3comp/2comp
```

注意

由于 `lfs find` 使用"!" 来做反向搜索，这里不支持标志 `^init` 。

(Lustre 2.13 引入)

19.6. 自扩展布局

Lustre 自扩展布局（Self-Extending Layout, SEL）功能是第 19.5 节 ``渐进式文件布局（PFL）" 功能的扩展，它允许 MDS 动态地改变定义的 PFL 布局。有了这个功能，MDS 会监控 OST 的使用空间，当 OST 空间不足时则为当前文件交换 OST，从而避免了 SEL 文件在应用程序写入时出现 ENOSPC 问题。

尽管 PFL 已经将一些组件的实例化推迟到对应区域发生 IO 操作时进行，SEL 仍允许将这种非实例化的组件分成两部分：一个 ``可扩展（extendable）" 组件和一个 ``扩展（extension）" 组件。可扩展组件是一个常规的 PFL 组件，只覆盖本来就很小的一块区域的一部分。扩展（或 SEL）组件是一个新的组件类型，一般是非实例化和未分配的空间，覆盖区域的另一部分。当需要对这个未分配的空间进行写入时，客户端调用 MDS 对其实例化，MDS 决定是否授予可扩展组件额外的空间。允许授予的区域从扩展组件的头部移动到可扩展组件的尾部，因此可扩展组件的空间增加，而 SEL 组件减少了。因此，文件可以在相同的 OST 上继续运行，或者在当前的一个 OST 上空间不足的情况下，修改布局以切换到新 OST 上的新组件。特别地，一旦小的 SSD OST 池的空间越来越少，则允许 IO 自动溢出到一个大的 HDD OST 池。

默认的扩展策略以下列方式修改布局：1. 扩展（Extension）：继续在相同的 OST 上 ------ 在当前组件的任何 OST 上的空间不低时使用该策略；允许在特定的范围授予可扩展组件。

2. 溢出（Spill over）：切换到下一个组件的 OST 上 ------仅用于不是最后一个组件，且在当前的 OST 中至少有一个空间不足时；SEL 组件的整个区域移动到下一个组件，SEL 组件被依次移除。

3. 重复（Repeating）：在空闲的 OST 上创建一个具有相同布局的新组件 ------ 仅用于最后一个组件，且在当前至少有一个 OST 空间不足时；新组件具有与之前相同的布局，但实例化在有足够空间的（同一个池的）不同 OST 上。

4. 强制扩展（Forced extension）：尽管空间不足，但继续使用当前组件的 OST ------仅用于最后一个组件，且重复尝试检测到空间不足的情况，此时不可能使用溢出策略，重复策略也没有意义。

注意 SEL 功能不需要客户端理解已经创建的文件的 SEL 格式，只需要 Lustre 2.13 中引入的 MDS 支持功能。然而，由于 Lustre 工具不支持，旧的客户端会有一些限制。

19.6.1. `lfs setstripe`

`lfs setstripe`命令用于创建具有复合布局的文件，也用于在现有文件中添加或删除组件，并支持 SEL 组件。

19.6.1.1. 创建 SEL 文件 命令

```
lfs setstripe
[--component-end|-E end1] [STRIPE_OPTIONS] ... filename
STRIPE OPTIONS:
--extension-size, --ext-size, -z <ext_size>
```

添加-z选项是为了指定每次迭代时授予的可扩展组件区域大小。在声明任何组件时，该选项将声明的组件变成一对组件：可扩展组件和扩展组件。

示例下面的命令创建了两对可扩展和扩展组件：

```
# lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/file
```

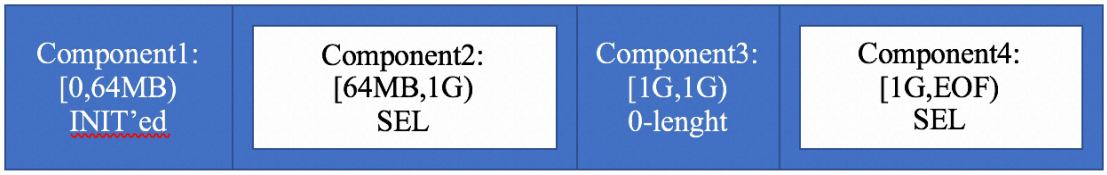


图 17: 示例：创建 SEL 文件

注意和之前一样，在创建时只有第一个 PFL 组件被实例化，因此它立即扩展到扩展大小（第一个组件为 64M），而第三个组件则长度为零。

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
 lcm_layout_gen: 4
 lcm_mirror_count: 1
 lcm_entry_count: 4
 lcme_id: 1
 lcme_mirror_id: 0
 lcme_flags: init
```

```
lcme_extent.e_start: 0
lcme_extent.e_end: 67108864
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
 lmm_stripe_offset: 0
lmm_objects:
- 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
lcme_id: 2
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 67108864
lcme_extent.e_end: 1073741824
lmm_stripe_count: 0
lmm_extension_size: 67108864
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
lcme_id: 3
lcme_mirror_id: 0
lcme_flags: 0
lcme_extent.e_start: 1073741824
lcme_extent.e_end: 1073741824
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
lcme_id: 4
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 1073741824
lcme_extent.e_end: EOF
lmm_stripe_count: 0
lmm_extension_size: 268435456
```

```
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
```

19.6.1.2. 创建 SEL 布局模版 与 PFL 类似，可以将 SEL 布局模板设置为一个目录。此后，所有在其下创建的文件默认继承这个布局。

```
# lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/dir
# ./lustre/utils/lfs getstripe /mnt/lustre/dir
/mnt/lustre/dir
lcm_layout_gen: 0
lcm_mirror_count: 1
lcm_entry_count: 4
lcme_id: N/A
lcme_mirror_id: N/A
lcme_flags: 0
lcme_extent.e_start: 0
lcme_extent.e_end: 67108864
stripe_count: 1 stripe_size: 1048576 pattern: raid0
lcme_id: N/A
lcme_mirror_id: N/A
lcme_flags: extension
lcme_extent.e_start: 67108864
lcme_extent.e_end: 1073741824
stripe_count: 1 extension_size: 67108864 pattern: raid0 stripe_offset: -1
lcme_id: N/A
lcme_mirror_id: N/A
lcme_flags: 0
lcme_extent.e_start: 1073741824
lcme_extent.e_end: 1073741824
stripe_count: 1 stripe_size: 1048576 pattern: raid0 stripe_offset: -1
lcme_id: N/A
lcme_mirror_id: N/A
lcme_flags: extension
lcme_extent.e_start: 1073741824
lcme_extent.e_end: EOF
```

```
30 stripe_count: 1 extension_size: 268435456 pattern: raid0
```

19.6.2. `lfs getstripe`

`lfs getstripe`命令可以用来列出一个给定的 SEL 文件的条带/组件信息。这里只显示了那些对 SEL 文件来说新的参数。

命令

```
1 lfs getstripe
2 [--extension-size|--ext-size|-z] filename
```

增加`-z`选项来打印扩展大小，单位是字节。对于复合文件来说，这是第一个扩展组件的扩展大小。如果其他选项（--componentid, --component-start, etc…）识别出了特定的组件，则打印出该组件的扩展大小。

示例 1: 列出 SEL 组件的信息假设我们已经有一个由以下命令创建的复合文件`/mnt/lustre/file`:

```
1 # lfs setstripe -E 1G -z 64M -E -1 -z 256M /mnt/lustre/file
```

第 2 个组件可以用下面的命令列出:

```
1 # lfs getstripe -I2 /mnt/lustre/file
2 /mnt/lustre/file
3  lcm_layout_gen: 4
4  lcm_mirror_count: 1
5  lcm_entry_count: 4
6  lcme_id: 2
7  lcme_mirror_id: 0
8  lcme_flags: extension
9  lcme_extent.e_start: 67108864
10 lcme_extent.e_end: 1073741824
11   lmm_stripe_count: 0
12 lmm_extension_size: 67108864
13 lmm_pattern: raid0
14 lmm_layout_gen: 0
15 lmm_stripe_offset: -1
```

注意

正如所见，SEL 组件被标记为`extension`标志，`lmm_extension_size`字段保持指定的扩展大小。

示例 2: 列出扩展名的大小如上例中的文件，第二个组件的扩展大小可以用以下方法列出：

```
# lfs getstripe -z -I2 /mnt/lustre/file
67108864
```

示例 3: 扩展如上例中的文件，假设有一个写入越过了第一个组件的末端 (64M)，然后又有一个写入越过了第一个组件的末端 (128M)，则布局变化如下：

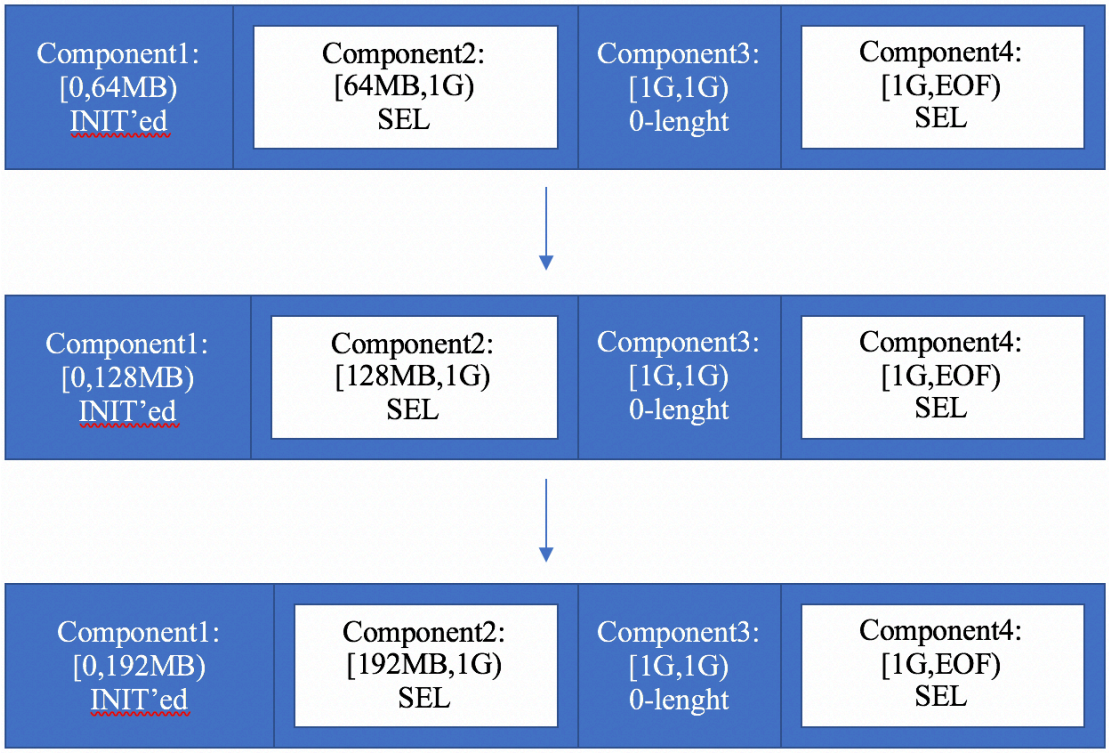


图 18: 示例：SEL 文件的扩展

该布局可由以下命令打印出：

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
 lcm_layout_gen: 6
 lcm_mirror_count: 1
 lcm_entry_count: 4
 lcme_id: 1
 lcme_mirror_id: 0
 lcme_flags: init
 lcme_extent.e_start: 0
 lcme_extent.e_end: 201326592
 lmm_stripe_count: 1
 lmm_stripe_size: 1048576
```

```
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 0
lmm_objects:
- 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
lcme_id: 2
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 201326592
lcme_extent.e_end: 1073741824
lmm_stripe_count: 0
lmm_extension_size: 67108864
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
lcme_id: 3
lcme_mirror_id: 0
lcme_flags: 0
lcme_extent.e_start: 1073741824
lcme_extent.e_end: 1073741824
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
lcme_id: 4
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 1073741824
lcme_extent.e_end: EOF
lmm_stripe_count: 0
lmm_extension_size: 268435456
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
```

示例 4: 溢出

如果 OST0 的空间不足，而一个 SEL 组件发生了 IO，就会发生溢出：SEL 组件的全部区域被添加到下一个组件中，例如，在上面的例子中，下一个布局修改将看起来类似下图：

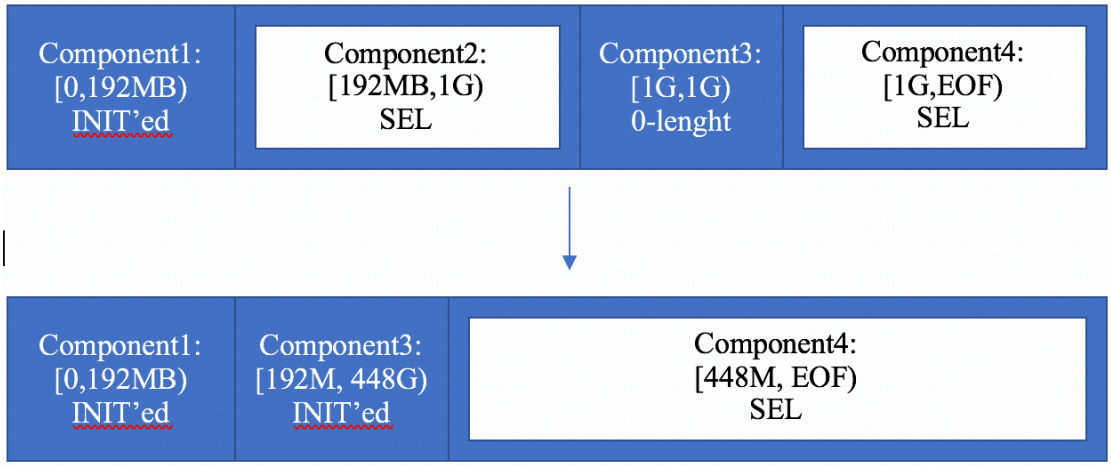


图 19: 示例：SEL 文件的溢出

注意

尽管第三个分量最初是 [1G, 1G]，但当它没有被实例化时，它不是向后扩展，而是向后移动到前一个 SEL 分量的起点（192M），并从该位置开始按其扩展大小（256M）扩展，因此它变成了 [192M,448M]。

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
 lcm_layout_gen: 7
 lcm_mirror_count: 1
 lcm_entry_count: 3
 lcme_id: 1
 lcme_mirror_id: 0
 lcme_flags: init
 lcme_extent.e_start: 0
 lcme_extent.e_end: 201326592
 lmm_stripe_count: 1
 lmm_stripe_size: 1048576
 lmm_pattern: raid0
 lmm_layout_gen: 0
 lmm_stripe_offset: 0
 lmm_objects:
 - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
```

```
lcme_id: 3
lcme_mirror_id: 0
lcme_flags: init
lcme_extent.e_start: 201326592
lcme_extent.e_end: 469762048
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 1
lmm_objects:
- 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }
lcme_id: 4
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 469762048
lcme_extent.e_end: EOF
lmm_stripe_count: 0
lmm_extension_size: 268435456
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
```

示例 5: 重复

假设在上面的例子中，OST0 得到了足够的空闲空间，但是 OST1 的空间不足，接下来的对最后一个 SEL 组件的写入导致了在 SEL 组件之前分配新组件，它重复之前的组件布局，但在空闲的 OST 上实例化。

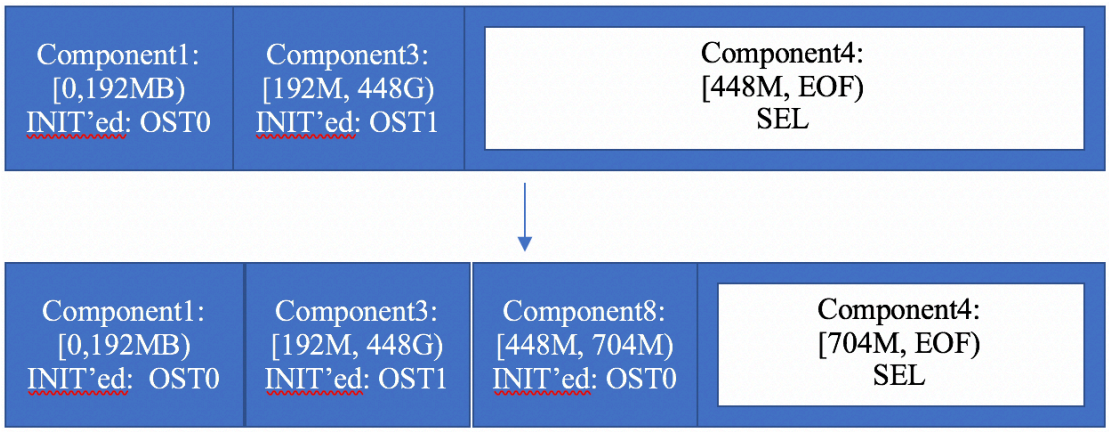


图 20: 示例：重复 SEL 组件

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
 lcm_layout_gen: 9
 lcm_mirror_count: 1
 lcm_entry_count: 4
 lcme_id: 1
 lcme_mirror_id: 0
 lcme_flags: init
 lcme_extent.e_start: 0
 lcme_extent.e_end: 201326592
 lmm_stripe_count: 1
 lmm_stripe_size: 1048576
 lmm_pattern: raid0
 lmm_layout_gen: 0
 lmm_stripe_offset: 0
 lmm_objects:
 - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
 lcme_id: 3
 lcme_mirror_id: 0
 lcme_flags: init
  lcme_extent.e_start: 201326592
 lcme_extent.e_end: 469762048
 lmm_stripe_count: 1
 lmm_stripe_size: 1048576
 lmm_pattern: raid0
 lmm_layout_gen: 0
 lmm_stripe_offset: 1
 lmm_objects:
 - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }
 lcme_id: 8
 lcme_mirror_id: 0
 lcme_flags: init
 lcme_extent.e_start: 469762048
 lcme_extent.e_end: 738197504
 lmm_stripe_count: 1
 lmm_stripe_size: 1048576
```

```
lmm_pattern: raid0
lmm_layout_gen: 65535
lmm_stripe_offset: 0
lmm_objects:
- 0: { l_ost_idx: 0, l_fid: [0x100000000:0x6:0x0] }
lcme_id: 4
lcme_mirror_id: 0
lcme_flags: extension
lcme_extent.e_start: 738197504
lcme_extent.e_end: EOF
lmm_stripe_count: 0
lmm_extension_size: 268435456
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
```

示例 6: 强制扩展

假设在上面的例子中，OST0 和 OST1 的空间都很低，则接下来对最后一个 SEL 组件的写入将作为一个扩展，因为无法重复。

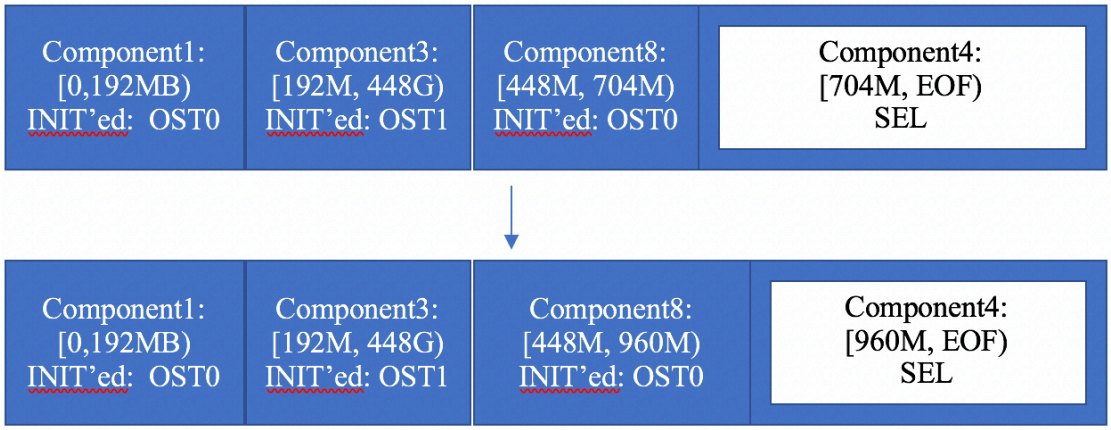


图 21: 示例：SEL 文件强制扩展

```
# lfs getstripe /mnt/lustre/file
/mnt/lustre/file
 lcm_layout_gen: 11
 lcm_mirror_count: 1
 lcm_entry_count: 4
 lcme_id: 1
 lcme_mirror_id: 0
```

```
lcme_flags: init
lcme_extent.e_start: 0
lcme_extent.e_end: 201326592
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 0
lmm_objects:
- 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }
lcme_id: 3
lcme_mirror_id: 0
lcme_flags: init
lcme_extent.e_start: 201326592
lcme_extent.e_end: 469762048
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 1
lmm_objects:
- 0: { l_ost_idx: 1, l_fid: [0x100010000:0x8:0x0] }
lcme_id: 8
lcme_mirror_id: 0
lcme_flags: init
lcme_extent.e_start: 469762048
lcme_extent.e_end: 1006632960
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 65535
lmm_stripe_offset: 0
lmm_objects:
- 0: { l_ost_idx: 0, l_fid: [0x100000000:0x6:0x0] }
lcme_id: 4
lcme_mirror_id: 0
```

```
lcme_flags: extension
lcme_extent.e_start: 1006632960
lcme_extent.e_end: EOF
lmm_stripe_count: 0
lmm_extension_size: 268435456
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: -1
```

19.6.3. `lfs find`

`lfs find`命令可以用来搜索符合给定 SEL 组件参数的文件。这里只显示了那些新的 SEL 文件的参数。

```
lfs find
[[!] --extension-size|-ext-size|-z [+-]ext-size[KMG]
[[!] --component-flags=extension]
```

添加`-z`选项是为了指定搜索的扩展大小。任何具有符合给定条件的扩展大小的组件的文件都将打印出来。如往常一样，``+" 和 ``-" 符号用来指定最小和最大的大小。

添加了一个新的 extension 组件标志。只打印至少有一个 SEL 组件的文件。

注意负搜索标志可以搜索包含非 SEL 组件的文件（而不是不包含任何 SEL 组件的文件）。

示例

```
# lfs setstripe --extension-size 64M -c 1 -E -1 /mnt/lustre/file
# lfs find --comp-flags extension /mnt/lustre/*
/mnt/lustre/file
# lfs find ! --comp-flags extension /mnt/lustre/*
/mnt/lustre/file
# lfs find -z 64M /mnt/lustre/*
/mnt/lustre/file
# lfs find -z +64M /mnt/lustre/*
# lfs find -z -64M /mnt/lustre/*
# lfs find -z +63M /mnt/lustre/*
/mnt/lustre/file
# lfs find -z -65M /mnt/lustre/*
/mnt/lustre/file
# lfs find -z 65M /mnt/lustre/*
```

```
# lfs find ! -z 64M /mnt/lustre/*
# lfs find ! -z +64M /mnt/lustre/*
/mnt/lustre/file
# lfs find ! -z -64M /mnt/lustre/*
/mnt/lustre/file
# lfs find ! -z +63M /mnt/lustre/*
# lfs find ! -z -65M /mnt/lustre/*
# lfs find ! -z 65M /mnt/lustre/*
/mnt/lustre/file
```

（Lustre 2.13 引入）

19.7. 外部布局

Lustre 外部布局功能是 LOV 和 LMV 格式的扩展，允许创建具有必要规格的空文件和目录，以指向 Lustre 命名空间以外的相应对象。

新的 LOV/LMV 外部格式可以表示为：

图 22: 示例：外部格式

19.7.1. `lfs set[dir]stripe`

`lfs set[dir]stripe`命令通过调用相应的 API 来创建具有外部布局的文件或目录，API 本身调用相应的 ioctl()。

19.7.1.1. 创建外部文件/目录 命令

```
lfs set[dir]stripe \
--foreign[=<foreign_type>] --xattr|-x <layout_string> \
[--flags <hex_bitmask>] [--mode <mode_bits>] \
{file,dir}name
```

`--foreign`和`--xattr|-x`选项都是强制性的。`<foreign_type>`（默认为``none"，意味着没有特殊行为），以及`--flags`和`--mode`（默认为 0666）两个选项都是可选的。

示例

下面的命令创建了一个类型为 ``none''，内容为 ``foo@bar'' 的 LOV 文件，以及特定的模式和标志的外部文件：

```
# lfs setstripe --foreign=none --flags=0xda08 --mode=0640 \
--xattr=foo@bar /mnt/lustre/file
```

图 23: 示例：创建外部文件

19.7.2. `lfs get[dir]stripe`

`lfs get[dir]stripe`命令可以用来检索外部 LOV/LMV 信息和内容。

命令

```
lfs get[dir]stripe [-v] filename
```

列出外部布局信息

假设我们已经有一个由以下命令创建的外部文件`/mnt/lustre/file`：

```
# lfs setstripe --foreign=none --flags=0xda08 --mode=0640 \
--xattr=foo@bar /mnt/lustre/file
```

完整的外部布局信息可以通过以下命令列出：

```
# lfs getstripe -v /mnt/lustre/file
/mnt/lustre/file
 lfm_magic: 0x0BD70BD0
 lfm_length: 7
 lfm_type: none
 lfm_flags: 0x0000DA08
 lfm_value: foo@bar
```

正如所见，`lfm_length`字段的值是可变长度`lfm_value`字段的字符数。

19.7.3. `lfs find`

`lfs find`命令可以用来搜索所有的外部文件/目录或符合给定选择参数的文件/目录。

```
lfs find
[[!] --foreign[=<foreign_type>]
```

增加了--foreign[=<foreign_type>]选项，以指定所有 [!,but not] 文件和/或检索具有外部布局的目录 [and [!,but not] of] 。

示例

```
# lfs setstripe --foreign=none --xattr=foo@bar /mnt/lustre/file
# touch /mnt/lustre/file2
# lfs find --foreign /mnt/lustre/*
/mnt/lustre/file
# lfs find ! --foreign /mnt/lustre/*
/mnt/lustre/file2
# lfs find --foreign=none /mnt/lustre/*
/mnt/lustre/file
```

19.8. 管理空闲空间

为了优化文件系统性能，MDT 根据两种分配算法将文件分配给 OST。循环分配法优先考虑位置（将条带分配到多个 OSS 以提高网络带宽利用率），加权分配法优先考虑可用空间（平衡 OST 间的负载）。用户可以调整这两种算法的阈值和加权因子。MDT 为每个 OST 预留总 OST 空间的 0.1%和 32 个 inode。如果可用空间小于该预留空间或空闲 inode 少于 32 个，MDT 会停止为该 OST 分配对象。当 OST 可用空间是预留空间的两倍且有 64 个以上的空闲 inode 时，MDT 开始对象分配。请注意，无论对象分配状态如何，客户端都可以附加现有文件。

Lustre 2.9 中，用户可以使用lctl set_param命令来调整每个 OST 的预留空间。如以下命令为所有 OST 都设置 1GB 的预留空间：

```
lctl set_param -P osp.*.reserved_mb_low=1024
```

本节将介绍如何查看磁盘上的可用空间、如何分配可用空间，以及如何设置分配算法的阈值和加权因子。

19.8.1. 查看文件系统可用空间

可用空间是分配文件条带需要考虑的重要因素。lfs df 命令可用于显示已安装的 Lustre 文件系统上的可用磁盘空间以及每个 OST 的空间消耗情况。如果安装了多个 Lustre 文件系统，可指定安装路径（不是必需的）。lfs df 命令的选项如下表所示：

选项	说明
-h, --human-readable	以可读的形式显示大小 (例如: 1K, 234M, 5G)，使用基数为 2(二进制）的值，如 1G=1024M).

选项	说明
-H, --si	与-h 一样，以可读的格式显示大小，但使用基数 10（十进制）的值（即 1G=1000M）。
-i, --inodes	列出索引节点的使用情况，而不是块使用情况。
-l, --lazy	不要试图连接当前没有连接到客户端的任何 OST 或 MDT，避免在目标离线或无法到达时阻塞 lfs df输出，并且只返回当前可以访问的 OST 上的空间。
-p, --pool	限制报告的使用，只报告指定池中的 OST。如果挂载了多个 Lustre 文件系统，则为每个文件系统列出池中的 OST，或者如果给出fsname.pool，则限制显示指定文件系统的池中的 OST。同时指定fsname和pool相当于指定了一个挂载点。
-v, --verbose	显示 MDT 和 OST 的详细状态，可能包括每行末尾的一个或多个可选的标志。

如果目标有问题，lfs df也可能报告其额外的状态，作为最后一列显示。目标状态包括：- D: 表示 OST/MDT 是降级的（Degraded）。该目标在 RAID 设备中有一个故障驱动器，或者正在进行 RAID 重建。这个状态在服务器上通过zed自动标记为 ZFS 目标，或者通过（用户提供的）脚本监视目标设备并在 OST 上设置``lctl set_param obdfilter.target.degraded=1"。避免在新的分配中使用这个目标，但仍可读取位于其上的现有文件，或者在没有足够的非降级的 OST 来组成一个宽条带化文件时也可使用。

- R: 表示 OST/MDT 是只读的 (Read-only)。ldiskfs 或 ZFS 检测到的文件系统损坏，因此将目标文件系统标记为只读。在这个 OST 上不允许有任何修改，需要卸载并运行e2fsck或zpool scrub来修复基础文件系统。
- N: 表示 OST/MDT 不能被预创建（No-precreate)。将目标配置表示拒绝通过``lctl set_param obdfilter.target.no_precreate=1" 参数或``-o no_precreate" 挂载选项设定的对象预创建。这样做可能是为了将一个 OST 添加到文件系统中，而不允许在其上分配对象，或出于某些其他原因。

- S: 表示 OST/MDT 没有空间 (Space) 了。目标文件系统的空闲空间小于最低要求，在它拥有更多的空闲空间之前，不会被用于新的对象分配。
- I: 表示 OST/MDT 没有Inodes了。目标文件系统的空闲节点少于最低要求，在它拥有更多的空闲节点之前，不会被用于新的对象分配。
- f: 表示 OST/MDT 在闪存flash上。目标文件系统正在使用一个闪存（非旋转）存储设备。这通常是由底层的 Linux 块设备检测出来的，但也可以在相应的 OST 上用 ``lctl set_param osd-*.*.nonrotational=1" 手动设置。这个小写的状态只在使用了-v选项时显示，因为它不算是一个错误。

注意 df -i 和 lfs df -i 命令显示当前可以在文件系统中创建的最小 inode 数目。如果所有 OST 中可用对象的总数小于 MDT 上可用对象的总数，考虑到默认的文件条带化，则 df -i 将报告比实际可创建更少数量的 inode。lfs df -i 将报告每个目标上实际上空闲的 inode 数量。

对于 ZFS 文件系统来说，可创建的 inode 数量是动态的，取决于文件系统的可用空间。所报告的 ZFS 文件系统的空闲 inode 数和 inode 总数只是基于每个目标的当前使用情况的估计值。已使用的 inode 数是文件系统实际使用的 inode 数量。

示例

```
client$ lfs df
UUID 1K-blocks Used Available Use% Mounted on
testfs-OST0000_UUID 9174328 1020024 8154304 11% /mnt/lustre[MDT:0]
testfs-OST0000_UUID 94181368 56330708 37850660 59% /mnt/lustre[OST:0]
testfs-OST0001_UUID 94181368 56385748 37795620 59% /mnt/lustre[OST:1]
testfs-OST0002_UUID 94181368 54352012 39829356 57% /mnt/lustre[OST:2]
filesystem summary: 282544104 167068468 39829356 57% /mnt/lustre
[client1] $ lfs df -hv
UUID bytes Used Available Use% Mounted on
testfs-MDT0000_UUID 8.7G 996.1M 7.8G 11% /mnt/lustre[MDT:0]
testfs-OST0000_UUID 89.8G 53.7G 36.1G 59% /mnt/lustre[OST:0] f
testfs-OST0001_UUID 89.8G 53.8G 36.0G 59% /mnt/lustre[OST:1] f
testfs-OST0002_UUID 89.8G 51.8G 38.0G 57% /mnt/lustre[OST:2] f
filesystem summary: 269.5G 159.3G 110.1G 59% /mnt/lustre
[client1] $ lfs df -iH
UUID Inodes IUsed IFree IUse% Mounted on
testfs-MDT0000_UUID 2.21M 41.9k 2.17M 1% /mnt/lustre[MDT:0]
testfs-OST0000_UUID 737.3k 12.1k 725.1k 1% /mnt/lustre[OST:0]
```

```
19 testfs-OST0001_UUID 737.3k 12.2k 725.0k 1% /mnt/lustre[OST:1]
20 testfs-OST0002_UUID 737.3k 12.2k 725.0k 1% /mnt/lustre[OST:2]
21 filesystem summary: 2.21M 41.9k 2.17M 1% /mnt/lustre[OST:2]
```

19.8.2. 条带分配方法

Lustre 文件系统提供了两种条带分配方法。

- **循环分配法** - 当 OST 的可用空间大小大致相同时，循环分配法将轮流在 OSS 的不同 OST 上进行条带化，所以用于每个文件 stripe 0 的 OST 将均匀分布在不同 OST 之间，无论条带计数是多少。下面的例子有八个 OST，编号为 0-7，对象将按如下方式分配：

```
1 File 1:OST1,OST2,OST3,OST4
2 File 2:OST5,OST6,OST7
3 File 3:OST0,OST1,OST2,OST3,OST4,OST5
4 File 4:OST6,OST7,OST0
```

以下是几个循环条带分配顺序的示例（每个字母表示 OSS 上的不同 OST）：

3: AAA	一个 3-OST 的 OSS
3x3: ABABAB	两个 3-OST 的 OSSs
3x4: BBABABA	一个 3-OST 的 OSS (A) 和一个 4-OST 的 OSS (B)
3x5: BBABBABA	一个 3-OST 的 OSS (A) 和一个 5-OST 的 OSS (B)
3x3x3: ABCABCABC	三个 3-OST 的 OSSs

- **加权分配法** - 当 OST 之间的空闲空间大小差异显著时，用加权算法根据大小（每个 OST 上的可用空间大小）和位置（均匀分布在 OST 上的条带）来决定 OST 排序。加权分配法能更快地填满空的 OST，但由于其使用的是随机算法，每次选择的不一定是有最多空闲空间的 OST。

该分配方法适用于 OST 上的空闲空间不平衡的情况。当 OST 的空闲空间相对平衡时，请使用更快的循环分配法，从而最大限度地实现网络平衡。而当任何两个 OST 的空闲空间大小差超过指定阈值（默认为 17%）时，使用加权分配法。两种分配方式的阈值由`qos_threshold_rr`参数定义。

若要将 qos_threshold_rr 暂时设置为 25，可在每个 MDS 上运行：

```
1 mds# lctl set_param lod.fsname*.qos_threshold_rr=25
```

19.8.3. 调整可用空间和位置的权重

通过qos_prio_free参数可以设置加权分配器中使用的加权优先级。增加 qos_prio_free 的值，基于各 OST 空闲空间而进行分配的权重会增加，基于 OST 上的条带分布方式的权重。默认值是 91（百分比）。当空闲空间优先级设置为 100（百分比）时，条带算法完全基于空闲空间，而不考虑位置。

若要将分配器权重永久地更改为 100，请在 MGS 上输入此命令：

```
1 lctl conf_param fsname-MDT0000-*.lod.qos_prio_free=100
```

注意

当 qos_prio_free设置为 100 时，仍然使用加权随机算法来分配条。如果 OST2 的可用空间是 OST1 的两倍，则使用 OST2 的可能性是 OST1 的两倍，但不能保证就一定使用 OST2。

19.9. Lustre 条带化内部参数

根据能够存储在 MDT 上的属性的最大大小，单个文件可在有限数量的 OST 上进行分条。如果是基于 ldiskfs 的 MDT 且没有启用 ea_inode 功能，则文件最多可以在 160 个 OST 上分条。如果是基于 ZFS 的 MDT 或是基于 ldiskfs 的 MDT 启用了 ea_inode 功能，则文件可以在多达 2000 个 OST 进行分条。

Lustre inode 使用扩展属性来记录每个对象所在的 OST 以及每个对象在该 OST 上的标识符。扩展属性的大小可以表示为条带数量的函数。

如果使用基于 ldiskfs 的 MDT，可以通过启用 MDT 上的 ea_inode 功能将文件分割在更多的 OST 上，最大数量为 2000：

```
1 tune2fs -O ea_inode /dev/mdtdev
```

（**Lustre 2.13** 中引入）

注意从 Lustre 2.13 开始，所有新格式化的 ldiskfs MDT 文件系统都默认启用 ea_inode 功能。

注意

单个文件的最大条带数不会限制整个文件系统中 OST 的最大数量，只会限制文件的最大大小和最大聚合带宽。

第二十章在 MDT 上存储数据的功能 (DoM)

20.1. 简介

LustreMDT 数据功能（DoM）通过将小文件直接放置 MDT 上来改进小文件 IO，通过避免使用容易被随机小 IO 事件（将导致设备搜索）影响流 IO 性能的 OST 来改进大

文件 IO。因此，用户在小文件 IO 模式和混合 IO 模式上都获得更好的一致性性能。

DoM 文件的布局作为组合布局存储在磁盘上，是渐进式文件布局（PFL）的特例。DoM 文件的布局由文件的组件组成，放在 MDT 上，其余的组件放在 OST 上（如果需要）。第一个组件放置在 MDT 上的对象数据块中。该组件只有一个条带，大小等于组件大小。这种具有 MDT 布局的组件只能是组合布局中的第一个组件。其余组件像往常一样通过 RAID0 布局放置在 OST 上。在超出 MDT 组件大小的文件之后，客户端进行数据写入或截断，OST 组件才被实例化。

20.2. 用户命令

Lustre 提供 `lfs setstripe` 命令以方便用户创建 DoM 文件。此外，像往常一样，`lfs getstripe` 命令可用于列出给定文件的分条/组件信息。而`lfs find` 命令可用于搜索以给定目录或文件名为根的目录树，以查找与给定 DoM 组件参数（如布局类型）匹配的文件。

20.2.1. `lfs setstripe`

`lfs setstrip`命令用于创建 DoM 文件。

```
lfs setstripe --component-end|-E end1 --layout|-L \
mdt [--component-end|-E end2 [STRIPE_OPTIONS] ...] <filename>
```

上面的命令创建了一个具有特殊组合布局的文件，它将第一个组件定义为 MDT 组件。MDT 组件必须从偏移 0 开始并在`end1`结束。`end1`也是该组件的条带大小，并受 MDT 的`lod.*.dom_stripesize`限制。无需其他选项。其余组件使用正常的语法来创建组合文件。

注意

如果下个组件未指定条带信息，如：

```
lfs setstripe -E 1M -L mdt -E EOF <filename>
```

则该组件将使用文件系统默认条带配置。

20.2.1.2. 示例 下面的命令将创建一个带有 DoM 布局的文件。第一个组件为 MDT 布局，被放置在 MDT 上，覆盖 [0，1M)。第二个组件覆盖 [1M，EOF)，并在所有可用的 OST 上进行分条。

```
client$ lfs setstripe -E 1M -L mdt -E -1 -S 4M -c -1 \
          /mnt/lustre/domfile
```

其布局如下图所示：

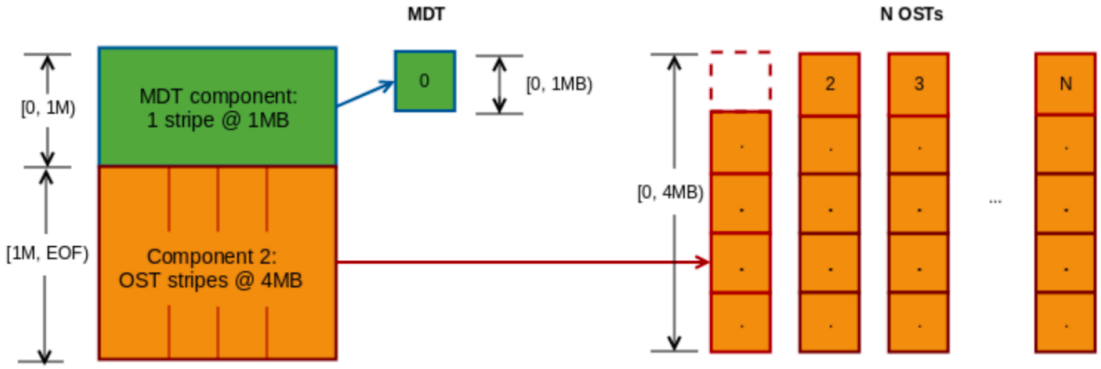


图 24: Lustre component

相关布局信息也可通过 `lfs getstripe` 命令显示：

```
client$ lfs getstripe /mnt/lustre/domfile
/mnt/lustre/domfile
  lcm_layout_gen:    2
  lcm_mirror_count: 1
  lcm_entry_count:  2
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  0
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:

    lcme_id:             2
    lcme_flags:          0
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  -1
      lmm_stripe_size:   4194304
      lmm_pattern:       raid0
      lmm_layout_gen:    65535
      lmm_stripe_offset: -1
```

上面的输出表明：第一个组件大小为 1MB，类型为'mdt'。第二个组件还未被示例化，见标志 `lcme_flags: 0`。

如果有超过 1MB 的数据被写入文件，`lfs getstripe` 的输出也将相应地发生变化。

```
client$ lfs getstripe /mnt/lustre/domfile
/mnt/lustre/domfile
  lcm_layout_gen:   3
  lcm_mirror_count: 1
  lcm_entry_count:  2
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  0
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
      lmm_layout_gen:    0
      lmm_stripe_offset: 2
      lmm_objects:

    lcme_id:             2
    lcme_flags:          init
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   4194304
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }
      - 1: { l_ost_idx: 1, l_fid: [0x100010000:0x2:0x0] }
```

如上所示，第二个组件有对象布置在 OSTs，条带大小为 4MB。

20.2.2. 为现有目录设置 DoM 布局

也可在现有目录上设置 DoM 布局。设置后，所有在此目录下创建的文件将默认继承此布局。

```
lfs setstripe --component-end|-E end1 --layout|-L mdt \
[--component-end|-E end2 [STRIPE_OPTIONS] ...] <dirname>
```

```
client$ mkdir /mnt/lustre/domdir
client$ touch /mnt/lustre/domdir/normfile
client$ lfs setstripe -E 1M -L mdt -E -1 /mnt/lustre/domdir/
client$ lfs getstripe -d /mnt/lustre/domdir
  lcm_layout_gen:   0
  lcm_mirror_count: 1
  lcm_entry_count:  2
    lcme_id:             N/A
    lcme_flags:          0
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      stripe_count:  0    stripe_size:   1048576    \
      pattern:  mdt    stripe_offset:  -1

    lcme_id:             N/A
    lcme_flags:          0
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   EOF
      stripe_count:  1    stripe_size:   1048576    \
      pattern:  raid0    stripe_offset:  -1
```

在上面的输出中，可以看到该目录具有含 DoM 组件的默认布局。

查看该目录的文件布局：

```
client$ touch /mnt/lustre/domdir/domfile
client$ lfs getstripe /mnt/lustre/domdir/normfile
/mnt/lustre/domdir/normfile
lmm_stripe_count:  2
lmm_stripe_size:   1048576
```

```
lmm_pattern:       raid0
lmm_layout_gen:    0
lmm_stripe_offset: 1
  obdidx   objid   objid   group
       1              3          0x3              0
       0              3          0x3              0

client$ lfs getstripe /mnt/lustre/domdir/domfile
/mnt/lustre/domdir/domfile
  lcm_layout_gen:   2
  lcm_mirror_count: 1
  lcm_entry_count:  2
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  0
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
      lmm_layout_gen:    0
      lmm_stripe_offset: 2
      lmm_objects:

    lcme_id:             2
    lcme_flags:          0
    lcme_extent.e_start: 1048576
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    65535
      lmm_stripe_offset: -1
```

我们可以看到该目录中的第一个文件 **normfile** 具有普通布局，而文件 **domfile** 继承了目录的默认布局，为 DoM 文件。

注意

尽管服务器的 DoM 大小限制会被设置成一个较低的值，该目录的默认布局设置仍会被新文件继承。

20.2.3.DoM 条带大小限制

DoM 组件的最大大小受到几种限制，以预防 MDT 最终被大文件填满。

20.2.3.1. Lustre 文件系统 (LFS) 限制 `lfs setstripe` 允许将 MDT 布局的组件大小设置为 1GB，但由于受 Lustre 中的最小条带大小所限（见表 5.2" 文件和文件系统限制"），其组件最大大小也只能为 64KB。同时，`lfs setstripe -E end`可以对每个文件有一个限制，如果对某一特定用途来说，这个限制可能小于 MDT 规定的限制。

20.2.3.2.MDT 服务器限制 LOD 参数`lod.$fsname-MDTxxxx.dom_stripesize` 用于控制 DoM 组件的每个 MDT 的最大大小。如果用户指定的 DoM 组件较大，将被截断到 MDT 指定的限制。因此，如果需要的话，每个 MDT 上的 DoM 空间使用量可能不同，以获取平衡。它默认为 1MB，可通过 lctl 工具进行更改。有关设置`dom_stripesize` 的更多信息，请参见本章第 2.6 节"dom_stripesize 参数"。

20.2.4. `lfs getstripe`

`lfs getstripe` 命令用于列出给定文件的分条/组件信息。对于 DoM 文件，它可以用来检查其布局和大小。

```
lfs getstripe [--component-id|-I [comp_id]] [--layout|-L] \
              [--stripe-size|-S] <dirname|filename>
```

```
client$ lfs getstripe -I1 /mnt/lustre/domfile
/mnt/lustre/domfile
  lcm_layout_gen:   3
  lcm_mirror_count: 1
  lcm_entry_count:  2
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
```

```
    lcme_extent.e_end:   1048576
      lmm_stripe_count:  0
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
      lmm_layout_gen:    0
      lmm_stripe_offset: 2
      lmm_objects:
```

DoM 组件布局和大小的简略信息课通过 -L选项配合-S 或 -E 选项来获取：

```
client$ lfs getstripe -I1 -L -S /mnt/lustre/domfile
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
client$ lfs getstripe -I1 -L -E /mnt/lustre/domfile
    lcme_extent.e_end:   1048576
      lmm_pattern:       mdt
```

这两个命令都将返回布局类型及其大小。条带大小等于 DoM 文件中组件的范围大小，因此两者都可用于获取 MDT 上的范围大小。

20.2.5. `lfs find`

`lfs find` 命令可用于搜索以给定目录或文件名为根的目录树，以查找与指定参数相匹配的文件。下面的命令输出了 DoM 文件的新参数，用法类似于 `lfs getstripe` 命令.

```
lfs find <directory|filename> [--layout|-L] [...]
```

20.2.5.2. 示例 在目录 `/mnt/lustre` 下搜索所有 DoM 布局的文件：

```
client$ lfs find -L mdt /mnt/lustre
/mnt/lustre/domfile
/mnt/lustre/domdir
/mnt/lustre/domdir/domfile

client$ lfs find -L mdt -type f /mnt/lustre
```

```
/mnt/lustre/domfile
/mnt/lustre/domdir/domfile

client$ lfs find -L mdt -type d /mnt/lustre
/mnt/lustre/domdir
```

通过该命令可查找所有 DoM 对象，DoM 文件或具有默认 DoM 布局的目录。

搜索指定条带大小的 DoM 文件/目录：

```
client$ lfs find -L mdt -S -1200K -type f /mnt/lustre
/mnt/lustre/domfile
/mnt/lustre/domdir/domfile

client$ lfs find -L mdt -S +200K -type f /mnt/lustre
/mnt/lustre/domfile
/mnt/lustre/domdir/domfile
```

第一个命令查找条带大小小于 1200KB 的所有 DoM 文件。第二个命令查找条带大小大于 200KB 的所有 DoM 文件。这两种情况下都能返回所有 DoM 文件，因为这里的 DoM 大小为 1MB。

20.2.6. dom_stripesize 参数

MDT 通过 LOD 设备上的参数 `dom_stripesize` 控制服务器上默认 DoM 最大大小。必要时，可以为每个 MDT 设置不同的 `dom_stripesize` 。该参数的默认值为 1MB，可以使用 `lclt` 工具进行更改。

```
lctl get_param lod.*MDT<index>*.dom_stripesize
```

20.2.6.2. Get 示例 运行下面的命令可获取服务器允许的最大 DoM 大小。之后，我们尝试创建了一个比参数值还大的文件，和预期一样，该操作失败并报错。

```
mds# lctl get_param lod.*MDT0000*.dom_stripesize
lod.lustre-MDT0000-mdtlov.dom_stripesize=1048576

mds# lctl get_param -n lod.*MDT0000*.dom_stripesize
1048576

```

```
client$ lfs setstripe -E 2M -L mdt /mnt/lustre/dom2mb
Create composite file /mnt/lustre/dom2mb failed. Invalid argument
error: setstripe: create composite file '/mnt/lustre/dom2mb' failed:
Invalid argument
```

20.2.6.3. Set （暂时）命令 暂时性地设置参数值，请运行 `lctl set_param`:

```
lctl set_param lod.*MDT<index>*.dom_stripesize=<value>
```

20.2.6.4. Set （暂时）示例 在下面的例子中，服务器上的默认 DoM 限制被更改为 64KB，并尝试创建大小为 1MB 的 DoM 文件。

```
mds# lctl set_param -n lod.*MDT0000*.dom_stripesize=64K
mds# lctl get_param -n lod.*MDT0000*.dom_stripesize
65536

client$ lfs setstripe -E 1M -L mdt /mnt/lustre/dom
Create composite file /mnt/lustre/dom failed. Invalid argument
error: setstripe: create composite file '/mnt/lustre/dom' failed:
Invalid argument
```

20.2.6.5. Set (永久）命令 永久性地设置参数值，请运行 `lctl conf_param`:

```
lctl conf_param <fsname>-MDT<index>.lod.dom_stripesize=<value>
```

20.2.6.6. Set (永久）示例 参数的新值被永久地存在配置日志中:

```
mgs# lctl conf_param lustre-MDT0000.lod.dom_stripesize=512K
mds# lctl get_param -n lod.*MDT0000*.dom_stripesize
524288
```

新设置将在几秒之内被应用，并永久保存到服务器配置中。

20.2.7. 禁用 DoM

当 `lclt set_param` 或 `lctl conf_param` 将 `dom_stripesize` 设置为 0 时，所选服务器将禁止 DoM 文件创建。

注意

DoM 文件仍可以使用默认的 DoM 布局在现有目录中创建。（**Lustre 2.11** 中引入）

第二十一章 MDT 的 Lazy 大小功能 (LSoM)

21.1. 简介

在 Lustre 文件系统中，MDS 上存储着 ctime、mtime、所有者和其他文件属性。OSS 上则存储着每个文件使用的块的大小和数量。要获得正确的文件大小，客户端必须访问存储文件的每个 OST，这意味着当一个文件在多个 OST 上分条时，需要使用多个 RPC 来获取文件的大小和块。MDT 上的 Lazy 大小 (LSoM) 功能将文件的大小存储在 MDS 上，如果应用程序能接受获取的文件大小不精准，则可以避免访问多个 OST 以获取文件大小。Lazy 意味着不能保证存储在 MDS 上的属性的准确性。

由于许多 Lustre 安装环境都使用固态硬盘作为 MDT，因此 LSoM 的目标是通过将数据存储在 MDT 上来加快从 Lustre 文件系统获取文件大小所需的时间。我们希望 Lustre 策略引擎初始使用这一功能，以扫描后端 MDT 存储，或根据不同的大小做出决策，且不依赖于完全准确的文件大小。类似的例子还包括 Lester, Robinhood, Zester 和各个供应商的产品。未来将改进为允许通过`lfs find`等工具访问 LSoM 数据。

21.2. 启动 LSoM

当使用策略引擎扫描 MDT 信息节点时，LSoM 始终处于启用状态，不需要做任何操作来启用获取 LSoM 数据的功能。通过`lfs getsom`命令也可以访问客户端上的 LSoM 数据。因为当前在客户端上通过 xattr 接口访问 LSoM 数据，所以只要缓存了索引节点，xattr_cache 就会在客户端上缓存文件大小和块计数。在大多数情况下，这是可行的，因为它改善了对 LSoM 数据的访问频率。但是，这也意味着，如果在首次访问 xattr 后文件大小发生了变化，或者在首次创建文件后不久访问 xattr，LSoM 数据可能会过时。

如果需要访问过时的最近 LSoM 数据，可以在客户端通过`lctl set_param ldlm.namespaces.*mdc*.lru_size=clear`取消 MDC 锁定，刷新 xattr 缓存。否则，如果在 LDLM 锁定超时前未访问文件，则将从客户端缓存中删除文件属性。通过`lctl get_param ldlm.namespaces.*mdc*.lru_max_age`储存锁定超时时长。

如果从特定客户端 (如 HSM 代理节点) 重复访问最近创建或频繁修改的文件的 LSoM 属性，则可以使用`lctl set_param llite.*.xattr_cache=0`来禁用客户端上的 xattr 缓存。但这可能会导致在访问文件时的额外开销，一般不建议使用。

21.3. 用户命令

Lustre 提供了`lfs getsom`命令以显示存储在 MDT 上的文件属性。

`llsom_sync`命令允许用户将 MDT 上的文件属性与 OSTs 上的有效或最新数据同步。可以在具有 Lustre 文件系统载入点的客户端上调用`llsom_sync`命令。该命令使用

Lustre MDS 变更日志，因此必须注册变更日志用户才能使用此命令工具。

21.3.1 使用lfs getsom显示 LSoM 数据

lfs getsom命令列出了存储在 MDT 上的文件属性。调用该命令需使用 Lustre 文件系统上文件的完整路径和文件名。如果没有使用选项，则存储在 MDS 上的所有文件属性都将显示出来。

21.3.2 lfs getsom 命令

```
lfs getsom [-s] [-b] [-f] <filename>
```

下面列出了各种 lfs getsom 选项。

选项	说明
-s	仅显示给定文件的 LSoM 数据的大小值。这是一个可选标志
-b	仅显示给定文件的 LSoM 数据的块值。这是一个可选标志
-f	仅显示给定文件的 LSoM 数据的标志值。这是一个可选标志。有效的标志值有: SOM_FL_UNKNOWN = 0x0000 ，表示未知或没有 SoM 数据，必须从 OSTs 获取大小；SOM_FL_STRICT = 0x0001，表示已知且严格正确，FLR 文件 (SOM 保证) ；SOM_FL_DEISE = 0x0002，表示已知但已过时，即在过去的某个时间点是正确的，但现在已知 (或可能) 不正确 (例如，打开进行写入)；SOM_FL_LAZY = 0x0004，表示近似值，可能从未严格正确过，需要同步 SOM 数据以实现最终的一致性。

第二十二章文件级冗余 (FLR)

22.1. 概述

Lustre 文件系统最初就是为 HPC 而设计的，它一直在具备内部冗余性和容错性的高端存储上运行良好。然而，尽管这些存储系统的成本昂贵、结构复杂，存储故障仍然时有发生。事实上，在 Lustre 2.11 发布之前，Lustre 文件系统并不比其底层的单个存储和服务器组件更可靠。Lustre 文件系统并没有机制能够缓解硬件存储故障。当服务器无法访问或终止服务时，将无法访问文件。

Lustre 2.11 中引入了 Lustre 文件级冗余（FLR）功能，任何 Lustre 文件都可将相同的数据存储在多台 OST 上，以提升系统在存储故障或其它故障发生时的稳健性。在存

在多个镜像的情况下，可选择最合适的镜像来响应单个请求，这对 IO 可用性有直接影响。此外，对于许多客户端同时读取的文件（如输入版，共享库或可执行文件），可以通过创建文件数据的多个镜像来提高单个文件的并行聚合读取性能。

第一阶段的 FLR 功能通过延迟写入实现（如" 图 21.1 FLR 延迟写入" 所示）。在写入镜像文件时，只有一个主镜像或首选镜像在写入过程中直接更新，而其他镜像将被标记为`stale`。通过使用命令行工具（由用户或管理员直接运行或通过自动监控工具运行）同步各镜像之间同步，该文件可在随后再次写入其它镜像。

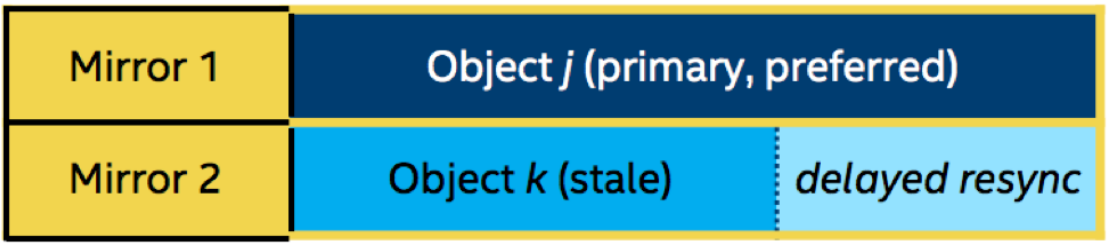


图 25: FLR delay writting

图 21.1 FLR 延迟写入

22.2. 相关操作

Lustre 为用户提供了 `lfs mirror` 命令行工具来操作镜像目录或文件。

22.2.1. 创建镜像文件或目录

命令：

```
lfs mirror create <--mirror-count|-N[mirror_count]
[setstripe_options|[--flags<=flags>]]> ... <filename|directory>
```

上述命令将创建由 *filename* 或 *directory* 指定的镜像文件或目录。

选项	说明
--mirror-count|-N[mirror_count]	用于指定使用`setstripe_options`所创建的镜像的数量。它可以重复多次使用，以分离具有不同布局的镜像。该参数是可选的，如果未指定，则默认为 1；如果指定，所指定的值必须紧挨着选项，不留空格。
setstripe_options	用于指定镜像的特定布局，可以是具有特定条带模式的简单布局，也可以是复合布局，如渐进式文件布局（PFL）。这些选项与`lfs setstrip`命令的选项相同。如果未指定该选项，则将从前一个组件继承条带

选项	说明
	选项。如果这是第一个组件，则stripe_count和stripe_size将继承文件系统范围的默认值，OST pool_name将继承父目录设定值。
--flags<=flags>	用于为创建的镜像设置标志，当前仅支持prefer标志。prefer标志能够给 Lustre 提示：哪些镜像将用于为 I / O 提供服务。当读取镜像文件时，具有prefer标志的组件可能会被选中，为该读操作提供服务；当写入镜像文件时，MDT 也将倾向于选择具有prefer标志的组件，并将与之重叠的其他组件标记为stale。由于该标志只是为 Lustre 提供提示，这意味着 Lustre 仍然可以选择没有此标志的镜像（例如，当 I/O 操作发生时所有首选镜像都不可用）。该标志可以在多个组件上设置。**注意**：设置该标志时，将对相应镜像的所有组件生效。如果需要在创建镜像时为其单个组件设置标志，请使用--comp-flags 选项。

注意: 考虑到冗余和容错，用户需要确保不同的镜像必须位于不同的 OSTs 上，甚至是不同的 OSSs 和机架上。实现这种架构，需要对集群拓扑的深刻理解。在最初的实现中，使用现有的 OST 池机制将允许通过任意的标准（即故障域）来分组 OST。在实际操作中，用户可以根据拓扑信息来对 OSTs 分组，从而充分利用 OST 池。因此，在创建镜像文件时，用户可以指出哪些 OST 池可以被镜像使用。

示例:

以下命令创建了有两个简单布局镜像的镜像文件。

```
client# lfs mirror create -N -S 4M -c 2 -p flash \
                          -N -c -1 -p archive /mnt/testfs/file1
```

显示镜像文件 /mnt/testfs/file1的布局信息，请运行:

```
client# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    2
```

```
lcm_mirror_count:  2
lcm_entry_count:   2
  lcme_id:             65537
  lcme_mirror_id:      1
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  2
    lmm_stripe_size:   4194304
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 1
    lmm_pool:          flash
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x2:0x0] }
    - 1: { l_ost_idx: 0, l_fid: [0x100000000:0x2:0x0] }

  lcme_id:             131074
  lcme_mirror_id:      2
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  6
    lmm_stripe_size:   4194304
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 3
    lmm_pool:          archive
    lmm_objects:
    - 0: { l_ost_idx: 3, l_fid: [0x100030000:0x2:0x0] }
    - 1: { l_ost_idx: 4, l_fid: [0x100040000:0x2:0x0] }
    - 2: { l_ost_idx: 5, l_fid: [0x100050000:0x2:0x0] }
    - 3: { l_ost_idx: 6, l_fid: [0x100060000:0x2:0x0] }
    - 4: { l_ost_idx: 7, l_fid: [0x100070000:0x2:0x0] }
    - 5: { l_ost_idx: 2, l_fid: [0x100020000:0x2:0x0] }
```

第一个镜像有和两个条带在 OST 池中的不同 OSTs 上，条带大小为 4MB。第二个镜像从第一个镜像继承了 4MB 的条带大小，在"archive"OST 池中的所有可用 OSTs 上进行条带化。

如上所述，建议在 OST 池中配置独立故障域，并使用`--pool|-p`选项（`lfs setstripe`选项之一）以确保不同的镜像放置在不同的 OST、服务器或机架上，从而提高可用性和性能。如果未指定`setstripe`选项，则可以在同一 OST 上创建带有对象的镜像，但这将消除使用备份的大多数好处。

使用`lfs getstripe`输出的布局信息中，`lcme_mirror_id`为镜像 ID，它是镜像的唯一数字标识符。`lcme_flags`为镜像组件标志。有效的标志名称有：

- `init` - 表示镜像组件已完成初始化（即已经分配了 OST 对象).
- `stale` - 表示镜像组件没有最新的数据。陈旧的组件不会用于读取或写入操作，在再次访问它们之前需要通过运行`lfs mirror resync`命令与最新数据进行同步。
- `prefer` - 表示镜像组件在读写时优先。例如，该镜像位于基于 SSD 的 OST 上，或者在网络上与客户端距离更近（跳数更少）。该标志可由用户在创建镜像时设置。

以下命令创建了有 3 个 PFL 镜像的镜像文件：

```
client# lfs mirror create -N -E 4M -p flash --flags=prefer -E eof -c 2 \
-N -E 16M -S 8M -c 4 -p archive --comp-flags=prefer -E eof -c -1 \
-N -E 32M -c 1 -p none -E eof -c -1 /mnt/testfs/file2
```

以下命令镜像文件`/mnt/testfs/file2`的布局信息：

```
client# lfs getstripe /mnt/testfs/file2
/mnt/testfs/file2
  lcm_layout_gen:    6
  lcm_mirror_count:  3
  lcm_entry_count:   6
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
```

```
    lmm_pool:          flash
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x3:0x0] }

  lcme_id:             65538
  lcme_mirror_id:      1
  lcme_flags:          prefer
  lcme_extent.e_start: 4194304
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  2
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
    lmm_pool:          flash

  lcme_id:             131075
  lcme_mirror_id:      2
  lcme_flags:          init,prefer
  lcme_extent.e_start: 0
  lcme_extent.e_end:   16777216
    lmm_stripe_count:  4
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 4
    lmm_pool:          archive
    lmm_objects:
    - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x3:0x0] }
    - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x3:0x0] }
    - 2: { l_ost_idx: 6, l_fid: [0x100060000:0x3:0x0] }
    - 3: { l_ost_idx: 7, l_fid: [0x100070000:0x3:0x0] }

  lcme_id:             131076
  lcme_mirror_id:      2
  lcme_flags:          0
```

```
  lcme_extent.e_start: 16777216
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  6
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
    lmm_pool:          archive

  lcme_id:             196613
  lcme_mirror_id:      3
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   33554432
    lmm_stripe_count:  1
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x3:0x0] }

  lcme_id:             196614
  lcme_mirror_id:      3
  lcme_flags:          0
  lcme_extent.e_start: 33554432
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  -1
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

第一个镜像中，第一个组件继承文件系统的默认条带数和条带大小。第二个组件继承第一个组件的条带大小和 OST 池，有两个条带。这两个组件都是从"flash" OST 池中分配的。此外，`prefer`标志应用于第一个镜像的所有组件，指示客户端在这些组件可

用时优先从他们读取数据。

第二个镜像中，第一个组件在"archive"OST 池中具有 8MB 条带大小和 4 个条带。第二个组件继承第一个组件的条带大小和 OST 池，并在"archive"OST 池中的所有可用 OST 上进行分条。prefer标志只应用于第一个组件。

第三个镜像中，第一个组件继承第二个镜像最后一个组件的 8MB 条带大小，只有一个单独的条带。OST 池名称被清除并从父目录继承（如果父目录设置了 OST 池名称）。第二个组件继承第一个组件的条带大小，并在所有可用的 OST 上进行分条。

22.2.2. 扩展镜像文件

命令:

```
lfs mirror extend [--no-verify] <--mirror-count|-N[mirror_count]
[setstripe_options|-f <victim_file>]> ... <filename>
```

上述命令将追加由 setstripe options 选项指定的镜像，或者将由filename指定的文件的布局替换为现有文件victim_file的布局。该filename必须是现有文件，但可以是镜像文件也可以是常规的非镜像文件。如果它是非镜像文件，则该命令会将其转换为镜像文件。

选项	说明
--mirror-count|-N[mirror_count]	用于指定使用setstrip_options所创建的镜像的数量。它可以重复多次使用，以分离具有不同布局的镜像。该参数是可选的，如果未指定，则默认为 1;如果指定，所指定的值必须紧挨着选项，不留空格。
setstripe_options	用于指定镜像的特定布局，可以是具有特定条带模式的简单布局，也可以是复合布局，如渐进式文件布局（PFL)。这些选项与lfs setstrip命令的选项相同。如果未指定该选项，则将从前一个组件继承条带选项。如果这是第一个组件，则stripe_count和stripe_size将继承文件系统范围的默认值，OST pool_name将继承父目录设定值。
-f	如果该存在vicitim_file选项，该命令将从该文件拆分布局并将其作为镜像添加到镜像文件。命令

选项	说明
	完成后，相应的victim_file将被删除。**注意**：在命令行中，不能使用-f <victim_file> 选项指定setstripe_options。
--no-verify	如果指定了victim_file，该命令将验证victim_file中的文件内容与文件名是否相同。否则，命令将返回失败信息。可以使用--no-verify选项来覆盖此验证操作。如果文件很大，此选项可以节省大量的文件内容比较时间，但只有在明确了文件内容相同的情况下才使用此选项。

注意: lfs mirror extend 选项不会应用到整个目录上。

示例:

以下命令创建了一个非镜像文件，然后将其转化为镜像文件，并为其扩展一个简单布局镜像。

```
# lfs setstripe -p flash /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
lmm_stripe_count:  1
lmm_stripe_size:   1048576
lmm_pattern:       raid0
lmm_layout_gen:    0
lmm_stripe_offset: 0
lmm_pool:          flash
        obdidx           objid           objid           group
             0               4            0x4               0

# lfs mirror extend -N -S 8M -c -1 -p archive /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    2
  lcm_mirror_count:  2
```

```
lcm_entry_count:   2
  lcme_id:             65537
  lcme_mirror_id:      1
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_pool:          flash
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x4:0x0] }

  lcme_id:             131073
  lcme_mirror_id:      2
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  6
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 3
    lmm_pool:          archive
    lmm_objects:
    - 0: { l_ost_idx: 3, l_fid: [0x100030000:0x3:0x0] }
    - 1: { l_ost_idx: 4, l_fid: [0x100040000:0x4:0x0] }
    - 2: { l_ost_idx: 5, l_fid: [0x100050000:0x4:0x0] }
    - 3: { l_ost_idx: 6, l_fid: [0x100060000:0x4:0x0] }
    - 4: { l_ost_idx: 7, l_fid: [0x100070000:0x4:0x0] }
    - 5: { l_ost_idx: 2, l_fid: [0x100020000:0x3:0x0] }
```

以下命令将从 victim_file中分离 PFL 布局，并将其作为镜像文件添加到在之前示例中创建的镜像文件 /mnt/testfs/file1 中（无数据验证）：

```
# lfs setstripe -E 16M -c 2 -p none \
               -E eof -c -1 /mnt/testfs/victim_file
# lfs getstripe /mnt/testfs/victim_file
/mnt/testfs/victim_file
  lcm_layout_gen:    2
  lcm_mirror_count:  1
  lcm_entry_count:   2
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   16777216
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 5
      lmm_objects:
      - 0: { l_ost_idx: 5, l_fid: [0x100050000:0x5:0x0] }
      - 1: { l_ost_idx: 6, l_fid: [0x100060000:0x5:0x0] }

    lcme_id:             2
    lcme_mirror_id:      0
    lcme_flags:          0
    lcme_extent.e_start: 16777216
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  -1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1

# lfs mirror extend --no-verify -N -f /mnt/testfs/victim_file \
                    /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
```

```
lcm_layout_gen:    3
lcm_mirror_count:  3
lcm_entry_count:   4
  lcme_id:             65537
  lcme_mirror_id:      1
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_pool:          flash
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x4:0x0] }

  lcme_id:             131073
  lcme_mirror_id:      2
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  6
    lmm_stripe_size:   8388608
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 3
    lmm_pool:          archive
    lmm_objects:
    - 0: { l_ost_idx: 3, l_fid: [0x100030000:0x3:0x0] }
    - 1: { l_ost_idx: 4, l_fid: [0x100040000:0x4:0x0] }
    - 2: { l_ost_idx: 5, l_fid: [0x100050000:0x4:0x0] }
    - 3: { l_ost_idx: 6, l_fid: [0x100060000:0x4:0x0] }
    - 4: { l_ost_idx: 7, l_fid: [0x100070000:0x4:0x0] }
    - 5: { l_ost_idx: 2, l_fid: [0x100020000:0x3:0x0] }

```

```
  lcme_id:             196609
  lcme_mirror_id:      3
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   16777216
    lmm_stripe_count:  2
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 5
    lmm_objects:
    - 0: { l_ost_idx: 5, l_fid: [0x100050000:0x5:0x0] }
    - 1: { l_ost_idx: 6, l_fid: [0x100060000:0x5:0x0] }

  lcme_id:             196610
  lcme_mirror_id:      3
  lcme_flags:          0
  lcme_extent.e_start: 16777216
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  -1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
```

镜像扩展完成后，victim_file被移除：

```
# ls /mnt/testfs/victim_file
ls: cannot access /mnt/testfs/victim_file: No such file or directory
```

22.2.3. 拆分镜像文件

命令：

```
lfs mirror split <--mirror-id <mirror_id>>
[--destroy|-d] [-f <new_file>] <mirrored_file>
```

上述命令将从mirrored_file指定的现有镜像文件中拆分出<mirror_id>指定 ID 的镜像。默认情况下，将使用该分离镜像的布

局创建名为<mirrored_file>. mirror<mirror_id>的新文件。如果指定了--destroy|-d选项，则该分离镜像将被销毁。如果指定了-f <new_file>选项，则将使用该分离镜像的布局创建一个名为new_file的文件。如果mirrored_file在被拆分后只有一个镜像，它将转为常规的非镜像文件。如果原 mirrored_file不是镜像文件，则命令将返回错误。

选项	说明
--mirror-id	镜像的唯一标识符，即该 ID 在镜像文件中是唯一的。将在镜像文件创建或扩展时进行自动分配。可通过 lfs getstripe 命令返回。
--destroy|-d	用于销毁分离镜像。
-f	使用分离镜像的布局创建一个文件名为 new_file 的新文件。

示例:

以下命令将创建有四个镜像的镜像文件，然后从镜像文件中分离三个镜像。

创建有四个镜像的镜像文件:

```
# lfs mirror create -N2 -E 4M -p flash -E eof -c -1 \
                    -N2 -S 8M -c 2 -p archive /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    6
  lcm_mirror_count:  4
  lcm_entry_count:   6
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
      lmm_pool:          flash
      lmm_objects:
```

```
      - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x4:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          flash

    lcme_id:             131075
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_pool:          flash
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x100000000:0x5:0x0] }

    lcme_id:             131076
    lcme_mirror_id:      2
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
```

```
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          flash

    lcme_id:             196613
    lcme_mirror_id:      3
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x5:0x0] }
      - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x6:0x0] }

    lcme_id:             262150
    lcme_mirror_id:      4
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 7
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 7, l_fid: [0x100070000:0x5:0x0] }
      - 1: { l_ost_idx: 2, l_fid: [0x100020000:0x4:0x0] }
```

从 /mnt/testfs/file1 中拆分出 ID 为 1 的镜像，并使用该分离镜像的布局创建文件 /mnt/testfs/file1.mirror~1：

```
# lfs mirror split --mirror-id 1 /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1.mirror~1
/mnt/testfs/file1.mirror~1
  lcm_layout_gen:    1
  lcm_mirror_count:  1
  lcm_entry_count:   2
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 1
      lmm_pool:          flash
      lmm_objects:
      - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x4:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          flash
```

从 /mnt/testfs/file1 中拆分出 ID 为 2 的镜像，并摧毁该分离镜像：

```
# lfs mirror split --mirror-id 2 -d /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
```

```
  lcm_layout_gen:    8
  lcm_mirror_count:  2
  lcm_entry_count:   2
    lcme_id:             196613
    lcme_mirror_id:      3
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x5:0x0] }
      - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x6:0x0] }

    lcme_id:             262150
    lcme_mirror_id:      4
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 7
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 7, l_fid: [0x100070000:0x5:0x0] }
      - 1: { l_ost_idx: 2, l_fid: [0x100020000:0x4:0x0] }
```

从 /mnt/testfs/file1 中拆分出 ID 为 1 的镜像，并使用该分离镜像的布局创建文件 /mnt/testfs/file2：

```
# lfs mirror split --mirror-id 3 -f /mnt/testfs/file2 \
```

```
                  /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file2
/mnt/testfs/file2
  lcm_layout_gen:    1
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             196613
    lcme_mirror_id:      3
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x5:0x0] }
      - 1: { l_ost_idx: 5, l_fid: [0x100050000:0x6:0x0] }

# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    9
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             262150
    lcme_mirror_id:      4
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   8388608
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 7
```

```
38      lmm_pool:          archive
39      lmm_objects:
40      - 0: { l_ost_idx: 7, l_fid: [0x100070000:0x5:0x0] }
41      - 1: { l_ost_idx: 2, l_fid: [0x100020000:0x4:0x0] }
```

以上布局信息显示了 ID 为 1, 2, 3 的镜像已全部从镜像文件 /mnt/testfs/file1 中分离。

22.2.4. 重新同步待同步镜像文件

命令:

```
1 lfs mirror resync [--only <mirror_id[,...]>]
2 <mirrored_file> [<mirrored_file2>...]
```

上述命令将重新同步由mirrored_file指定的待同步镜像文件。它支持在一行命令中指定多个镜像文件。

如果指定的镜像文件没有过时的镜像，那么该命令什么也不做。否则，它会将数据从已同步到最新数据的镜像复制到旧镜像，并将所有成功完成复制的镜像标记为 SYNC。如果指定了--only <mirror_id ,…[]>选项，那么命令将只重新同步由mirror_id(s)指定的镜像，此选项不能指定多个镜像文件。

选项	说明
--only	由 mirror_id 指定的某个或某些镜像需要重新同步。mirror_id 是镜像的唯一标识符。多个mirror_id由逗号分隔。指定多个镜像文件时，不能使用此选项。

注意: 由于 FLR 第一阶段的延迟写入操作，在完成镜像文件的数据写入后，用户需要运行lfs mirror resync 命令来同步所有镜像。

示例:

以下命令创建了一个有三个镜像的镜像文件，然后写入了数据病重新同步了过时的镜像。

创建有三个镜像的镜像文件:

```
1 # lfs mirror create -N -E 4M -p flash -E eof \
2                     -N2 -p archive /mnt/testfs/file1
3 # lfs getstripe /mnt/testfs/file1
4 /mnt/testfs/file1
```

```
lcm_layout_gen:    4
lcm_mirror_count:  3
lcm_entry_count:   4
  lcme_id:             65537
  lcme_mirror_id:      1
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   4194304
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 1
    lmm_pool:          flash
    lmm_objects:
    - 0: { l_ost_idx: 1, l_fid: [0x100010000:0x5:0x0] }

  lcme_id:             65538
  lcme_mirror_id:      1
  lcme_flags:          0
  lcme_extent.e_start: 4194304
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: -1
    lmm_pool:          flash

  lcme_id:             131075
  lcme_mirror_id:      2
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
```

```
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 3
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 3, l_fid: [0x100030000:0x4:0x0] }

    lcme_id:             196612
    lcme_mirror_id:      3
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_pool:          archive
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0x100040000:0x6:0x0] }
```

在镜像文件 /mnt/testfs/file1 中写入数据：

```
# yes | dd of=/mnt/testfs/file1 bs=1M count=2
2+0 records in
2+0 records out
2097152 bytes (2.1 MB) copied, 0.0320613 s, 65.4 MB/s

# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    5
  lcm_mirror_count:  3
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init
    lcme_extent.e_start: 0
```

```
    lcme_extent.e_end:   4194304
    ......

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             131075
    lcme_mirror_id:      2
    lcme_flags:          init,stale
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             196612
    lcme_mirror_id:      3
    lcme_flags:          init,stale
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......
```

以上布局信息显示，数据被写入了 ID 为 1的镜像的第一个组件，ID 为 2 和 3的镜像被标记为"stale"（过时的）。

重新同步镜像文件 /mnt/testfs/file1的 ID 为 2 的过时镜像。

```
# lfs mirror resync --only 2 /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    7
  lcm_mirror_count:  3
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init
```

```
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
    ......

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             131075
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             196612
    lcme_mirror_id:      3
    lcme_flags:          init,stale
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......
```

以上为同步化完成后的布局信息，ID 为 2的镜像的"stale" 标记被移除。

重新同步镜像文件 /mnt/testfs/file1的所有过时镜像。

```
# lfs mirror resync /mnt/testfs/file1
# lfs getstripe /mnt/testfs/file1
/mnt/testfs/file1
  lcm_layout_gen:    9
  lcm_mirror_count:  3
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init
```

```
    lcme_extent.e_start: 0
    lcme_extent.e_end:   4194304
    ......

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 4194304
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             131075
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......

    lcme_id:             196612
    lcme_mirror_id:      3
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
    ......
```

以上为同步化完成后的布局信息，没有镜像被标记为"stale"。

22.2.5. 验证镜像文件

命令:

```
lfs mirror verify [--only <mirror_id,mirror_id2[,...]>]
[--verbose|-v] <mirrored_file> [<mirrored_file2> ...]
```

上述命令将验证由`mirrored_file`指定的镜像文件的所有 SYNC 镜像（包含最新数据）是否具有完全相同的数据。它支持在一行命令中指定多个镜像文件。

这是一个应定期运行的清理工具，以确保镜像文件没有损坏。当镜像文件损坏时，该命令将不会修复文件。通常，管理员应该检查每个镜像中的文件内容并决定哪一个是正确的，随后调用`lfs mirror resync`进行手动修复。

选项	说明
--only	用于指示需要验证的镜像。mirror_id 是镜像的唯一数字标识符。多个mirror_id 以逗号分隔。**注意：**至少需要两个mirror_id。此选项不能指定多个镜像文件。
--verbose|-v	使用该选项时，将在数据不匹配时显示哪些地方出现了差异。未使用该选项只会在数据不匹配时返回错误。此选项可以重复多次，以输出更多信息。

注意:

带有"stale" 或"offline" 标志的镜像组件将被略过，不会被验证。

示例:

以下命令用于验证镜像文件的每个镜像包含了完全一样的数据。

```
# lfs mirror verify /mnt/testfs/file1
```

以下命令使用了 -v 选项以显示具体哪些地方出现了数据不匹配的情况。

```
# lfs mirror verify -vvv /mnt/testfs/file2
Chunks to be verified in /mnt/testfs/file2:
[0, 0x200000)   [1, 2, 3, 4]    4
[0x200000, 0x400000)    [1, 2, 3, 4]    4
[0x400000, 0x600000)    [1, 2, 3, 4]    4
[0x600000, 0x800000)    [1, 2, 3, 4]    4
[0x800000, 0xa00000)    [1, 2, 3, 4]    4
[0xa00000, 0x1000000)   [1, 2, 3, 4]    4
[0x1000000, 0xffffffffffffffff) [1, 2, 3, 4]    4

Verifying chunk [0, 0x200000) on mirror: 1 2 3 4
CRC-32 checksum value for chunk [0, 0x200000):
Mirror 1:       0x207b02f1
Mirror 2:       0x207b02f1
Mirror 3:       0x207b02f1
Mirror 4:       0x207b02f1

```

```
Verifying chunk [0, 0x200000) on mirror: 1 2 3 4 PASS

Verifying chunk [0x200000, 0x400000) on mirror: 1 2 3 4
CRC-32 checksum value for chunk [0x200000, 0x400000):
Mirror 1:       0x207b02f1
Mirror 2:       0x207b02f1
Mirror 3:       0x207b02f1
Mirror 4:       0x207b02f1

Verifying chunk [0x200000, 0x400000) on mirror: 1 2 3 4 PASS

Verifying chunk [0x400000, 0x600000) on mirror: 1 2 3 4
CRC-32 checksum value for chunk [0x400000, 0x600000):
Mirror 1:       0x42571b66
Mirror 2:       0x42571b66
Mirror 3:       0x42571b66
Mirror 4:       0xabdaf92

lfs mirror verify: chunk [0x400000, 0x600000) has different
checksum value on mirror 1 and mirror 4.
Verifying chunk [0x600000, 0x800000) on mirror: 1 2 3 4
CRC-32 checksum value for chunk [0x600000, 0x800000):
Mirror 1:       0x1f8ad0d8
Mirror 2:       0x1f8ad0d8
Mirror 3:       0x1f8ad0d8
Mirror 4:       0x18975bf9

lfs mirror verify: chunk [0x600000, 0x800000) has different
checksum value on mirror 1 and mirror 4.
Verifying chunk [0x800000, 0xa00000) on mirror: 1 2 3 4
CRC-32 checksum value for chunk [0x800000, 0xa00000):
Mirror 1:       0x69c17478
Mirror 2:       0x69c17478
Mirror 3:       0x69c17478
Mirror 4:       0x69c17478

```

```
Verifying chunk [0x800000, 0xa00000) on mirror: 1 2 3 4 PASS

lfs mirror verify: '/mnt/testfs/file2' chunk [0xa00000, 0x1000000]
exceeds file size 0xa00000: skipped
```

以下命令使用 --only 选项来指定需要验证的镜像。

```
# lfs mirror verify -v --only 1,4 /mnt/testfs/file2
CRC-32 checksum value for chunk [0, 0x200000):
Mirror 1:       0x207b02f1
Mirror 4:       0x207b02f1

CRC-32 checksum value for chunk [0x200000, 0x400000):
Mirror 1:       0x207b02f1
Mirror 4:       0x207b02f1

CRC-32 checksum value for chunk [0x400000, 0x600000):
Mirror 1:       0x42571b66
Mirror 4:       0xabdaf92

lfs mirror verify: chunk [0x400000, 0x600000) has different
checksum value on mirror 1 and mirror 4.
CRC-32 checksum value for chunk [0x600000, 0x800000):
Mirror 1:       0x1f8ad0d8
Mirror 4:       0x18975bf9

lfs mirror verify: chunk [0x600000, 0x800000) has different
checksum value on mirror 1 and mirror 4.
CRC-32 checksum value for chunk [0x800000, 0xa00000):
Mirror 1:       0x69c17478
Mirror 4:       0x69c17478

lfs mirror verify: '/mnt/testfs/file2' chunk [0xa00000, 0x1000000]
exceeds file size 0xa00000: skipped
```

22.2.6. 查找镜像文件

`lfs find` 命令用于列出含指定属性的文件和目录。下面为该镜像文件或目录指定了两个属性参数：

```
lfs find <directory|filename ...>
    [[!] --mirror-count|-N [+-]n]
    [[!] --mirror-state <[^]state>]
```

选项	说明
--mirror-count|-N [+-]n	用于指示镜像数量。
--mirror-state <[^]state>	用于指示镜像文件的状态。只能指定一个状态。如果使用 ^ state，则仅输出不匹配状态的文件。 有效状态名称有： ro，只读状态，表示所有镜像都包含了最新的数据。 wp，写入状态。sp，重新同步状态。

注意:

在选项之前指定!表示否定（即不符合该参数的文件）。在数值之前使用+意思为" 多于 n"，而在数值之前使用-意思为" 小于 n"。如果两者均未使用，则表示在指定单位（如果有的话）上" 等于 n"。

示例:

以下命令将递归地列出在 /mnt/testfs目录下所有有两个镜像以上的镜像文件：

```
# lfs find --mirror-count +2 --type f /mnt/testfs
```

以下命令将递归地列出在 /mnt/testfs目录下待同步的镜像文件：

```
# lfs find --mirror-state=^ro --type f /mnt/testfs
```

22.3. 互操作性

Lustre 2.11.0 中，我们引入了 FLR 功能。

对于 Lustre 2.9 或更老版本的客户端来说，由于它们不理解 PFL 布局，将无法访问和打开在 Lustre 2.11 文件系统中创建的镜像文件。

以下例子显示了在 **Lustre 2.9** 客户端上访问和打开（在 Lustre 2.11 文件系统中创建的）镜像文件返回的错误：

```
# ls /mnt/testfs/mirrored_file
ls: cannot access /mnt/testfs/mirrored_file: Invalid argument

# cat /mnt/testfs/mirrored_file
cat: /mnt/testfs/mirrored_file: Operation not supported
```

Lustre 2.10 客户端能够理解 PFL 布局，但不能理解镜像文件布局。它们能访问但不能打开在 **Lustre 2.11** 文件系统中创建的镜像文件。这是因为 Lustre 2.10 客户端不会验证重叠的组件，它们会将镜像文件视为普通的 PFL 文件一样进行读写，这将导致已同步了的镜像实际上包含了不同数据。

以下例子显示了在 Lustre 2.10 客户端上访问和打开（在 Lustre 2.11 文件系统中创建的）镜像文件返回的结果：

```
# ls /mnt/testfs/mirrored_file
/mnt/testfs/mirrored_file

# cat /mnt/testfs/mirrored_file
cat: /mnt/testfs/mirrored_file: Operation not supported
```

第二十三章管理文件系统和 I/O

23.1. 处理满溢的 OSTs

有时 Lustre 文件系统会变得不平衡，这通常是由于错误的条带设置，或者非常大的文件在创建时未被分条到所有 OST 上。如果 OST 已满，试图写入更多信息到文件系统将会发生错误。以下程序描述了如何处理满溢的 OST。

MDS 一般会在文件创建时自动平衡空间，因此通常不需要此程序，但在某些情况下（如创建须占用超过所有 OST 的总可用空间的大文件时）可能需要此程序。

23.1.1. 查看 OST 空间使用情况

下面的例子显示了一个不平衡的文件系统。

```
client# lfs df -h
UUID                       bytes          Used           Available       \
Use%            Mounted on
testfs-MDT0000_UUID        4.4G           214.5M         3.9G            \
4%              /mnt/testfs[MDT:0]
testfs-OST0000_UUID        2.0G           751.3M         1.1G            \
```

```
37%             /mnt/testfs[OST:0]
testfs-OST0001_UUID       2.0G            755.3M          1.1G            \
37%             /mnt/testfs[OST:1]
testfs-OST0002_UUID       2.0G            1.7G            155.1M          \
86%             /mnt/testfs[OST:2] ****
testfs-OST0003_UUID       2.0G            751.3M          1.1G            \
37%             /mnt/testfs[OST:3]
testfs-OST0004_UUID       2.0G            747.3M          1.1G            \
37%             /mnt/testfs[OST:4]
testfs-OST0005_UUID       2.0G            743.3M          1.1G            \
36%             /mnt/testfs[OST:5]

filesystem summary:       11.8G           5.4G            5.8G            \
45%             /mnt/testfs
```

在这种情况下，OST0002 几乎已经全满了，任何往文件系统写入更多信息的尝试（即使在所有 OSTs 上平均地分条）都将失败，如下所示：

```
client# lfs setstripe /mnt/testfs 4M 0 -1
client# dd if=/dev/zero of=/mnt/testfs/test_3 bs=10M count=100
dd: writing '/mnt/testfs/test_3': No space left on device
98+0 records in
97+0 records out
1017192448 bytes (1.0 GB) copied, 23.2411 seconds, 43.8 MB/s
```

23.1.2. 在满溢的 OST 上禁用创建功能

为避免文件系统空间不足，如果 OST 空间使用不平衡，甚至一个或多个 OSTs 接近满溢而其他 OSTs 有很多空间，则可以在 MDS 上有选择性地停用满溢的 OSTs 以防止 MDS 在这些 OSTs 上分配新的对象。

1. 登陆 MDS 服务器并使用`lctl`命令禁止在满溢的 OSTs 上创建新对象：

```
mds# lctl set_param osp.fsname-OSTnnnn*.max_create_count=0
```

在文件系统中创建新文件时，将只使用剩余的 OST。可以通过将数据迁移到其他 OST 来手动平衡空间（将在下一节介绍），同时，可以通过删除和创建文件来被动地平衡空间。

23.1.3. 在文件系统内迁移数据

如果需要将文件数据从当前的 OST 迁移到新的 OST，则必须将数据迁移（复制）到新的位置。最简单的方法是使用`lfs_migrate`命令。

23.1.4. 将禁用的 OST 重新上线

一旦停用的 OST 经过主动或被动数据重新分配后不再严重不平衡，它们应该重新被激活，以便再次分配新文件到这些 OSTs 上。

```
[mds]# lctl set_param osp.testfs-OST0002.max_create_count=20000
```

23.1.5. 在文件系统内迁移元数据

23.1.5.1. 整体目录迁移 Lustre 2.8 引入了在 MDTs 之间直接迁移元数据（目录和索引节点）的功能。此迁移只能在整个目录上执行。Lustre 2.12 引入了条带化目录的功能。例如，要将`/testfs/remotedir`目录的内容从当前所在的 MDT 迁移到 MDT0000, 以允许删除该 MDT，使用的命令如下:

```
$ cd /testfs
$ lfs getdirstripe -m ./remotedir which MDT is dir on?
1
$ touch ./remotedir/file.{1,2,3}.txtcreate test files
$ lfs getstripe -m ./remotedir/file.*.txtcheck files are on MDT0001
1
1
1
$ lfs migrate -m 0 ./remotedir migrate testremote to MDT0000
$ lfs getdirstripe -m ./remotedir which MDT is dir on now?
0
$ lfs getstripe -m ./remotedir/file.*.txtcheck files are on MDT0000
0
0
0
```

更多信息见`man lfs-migrate`。

注意

迁移期间，每个文件都会被分配一个新的标识符（FID）。因此，该文件也会将新的 inode 编号通知给用户空间应用。即使内容未更改，一些系统工具（例如备份、归档工具，NFS，Samba）可能仍会将迁移文件视为新文件。如果 Lustre 系统通知了新的 FID

给 NFS，但客户端或服务器仍使用旧的 FID 缓存过时的文件句柄，则在迁移期间和之后，迁移的文件可能变得不可访问。重新启动 NFS 服务将刷新本地文件句柄缓存，但客户端也可能需要重新启动，因为它们可能会缓存了过时的文件句柄。

23.1.5.2. 条带化目录迁移 Lustre 2.8 引入了在 MDTs 之间迁移元数据 (其中的目录和索引节点) 的功能，但是它不支持条带化目录的迁移，也不支持更改现有目录的条带数。Lustre 2.12 增加了对在迁移时重新分条目录的功能。`lfs migrate -m`命令只能对整个目录执行，它会递归地迁移指定的目录及其子条目。例如，要将大型目录`/testfs/largedir`的内容从其在 MDT0000 上的当前位置迁移到 MDT0001 和 MDT0003，请运行以下命令:

```
1 $ lfs migrate -m 1,3 /testfs/largedir
```

元数据迁移会将文件和索引节点直接迁移到其他 MDT，但不涉及文件数据的迁移。在迁移过程中，目录及其子文件可以像普通文件一样被访问，这些同样适用于依赖于文件索引节点编号的工具。迁移可能会由于多种原因而失败，如 MDS 重启或磁盘已满。在这些情况下，可能出现一些子文件可能已经迁移到新的 MDT，而其他子文件仍然在原始 MDT 上，但这些文件仍可正常访问的问题。解决这些问题后，应该再次执行与之前相同的`lfs migrate -m`命令来完成此迁移。但是，您不能中止失败的迁移，也不能从以前的迁移命令迁移到不同的 MDTs。

(Lustre 2.12 引入)

23.1.5.3. 目录重条带化 Lustre 2.14 包括一个改变现有目录的条带数的功能。`lfs setdirstripe -c`命令可以在一个现有的目录上执行，以改变其条带数。例如，一个目录`/testfs/testdir`正在变大，运行下面的命令，将其条带数增加到 2:

```
1 $ lfs setdirstripe -c 2 /testfs/testdir
```

默认情况下，目录重条带化将只迁移子文件目录，但不会移动节点。如需同时移动目录和节点，请在所有的 MDS 上运行以下命令:

```
1 mds$ lctl set_param mdt.*.dir_restripe_nsonly=0
```

在目录中重条带化不允许指定 MDT，相反，服务器会根据空间和节点的使用情况为增加的条带选择 MDT。在重条带化过程中，目录和其子文件可以像普通文件一样访问，这和目录迁移是一样的。同样地，你无法中止失败的重条带化，当服务器注意到未完成的重条带化时，将自动恢复失败的操作。

(Lustre 2.12 引入)

23.1.5.4. 目录自动拆分 Lustre 2.14 包含一个功能，当一个目录变得很大时会自动增加条带数。这可以通过以下命令启用:

```
mds$ lctl set_param mdt.*.enable_dir_auto_split=1
```

触发目录自动拆分的子文件数是 50k，该值可以通过以下命令改变：

```
mds$ lctl set_param mdt.*.dir_split_count=value
```

如果是一个普通的目录，目录的条带数将从 0 增加到 4，在第二次拆分时从 4 增加到 8，以此类推。然而，最终的条带数不会超过总 MDT 数，当一个目录在所有的 MDT 中分配完毕后，将停止分裂。这个 delta 值可以通过以下命令改变：

```
mds$ lctl set_param mdt.*.dir_split_delta=value
```

23.2. 创建和管理 OST 池

有了 OST 池（OST Pool）功能，用户就能将 OSTs 分组，从而更加灵活地放置对象。``池"（Pool）拥有一个名字，这个名字同 Lustre 集群中全体 OST 的一个子集相关联。

OST 池遵循以下规则：

- 一个 OST 可以是多个池的成员。
- OSTs 在池内没有顺序。
- 池内的条带分配遵循普通条带分配规则。
- OST 作为池的成员是灵活的，可以随时更改。

定义 OST 池时，可以进行文件分配。当为池设置文件或目录条带配置时，只可以使用池中的 OST 进行条带化。如果为`stripe_index`指定了一个不是池成员的 OST，则会返回错误。

OST 池仅用于创建文件。如果池的定义发生更改（添加或删除 OST 或池被销毁），已创建的文件不受影响。

注意

如果用空池创建文件，将返回错误 (`EINVAL`)。

如果某个目录使用池条带设置而该池随后被删除，则在该目录中创建的新文件将使用该目录的默认条带化模式（非池条带模式），不会返回错误。

23.2.1. OST 池操作

OST 池在 MGS 上的配置日志中定义。使用`lctl`命令：

- 创建/销毁池
- 在池中增加/移除 OSTs
- 列出所有池及某个池中的 OSTs

lctl命令必须在 MGS 上运行。同时，要么将 MDT 和 MGS 放在同一个节点上，要么在 MGS 节点上挂载 Lustre 客户端（如果与 MDS 分离）。这是必须的，以验证正在运行的池命令是否正确。

注意

在 MDS 上运行writeconf命令将擦除所有池信息（以及使用lctl conf_param设置的任何其他参数）。我们建议使用脚本执行池定义（和conf_param设置），以便在执行writeconf后可以轻松地再现它们。

要创建新池，请运行：

```
1 mgs# lctl pool_new fsname.poolname
```

注意

池名称是长达 15 个字符的 ASCII 串。

将已命名的 OST 加入池，运行：

```
1 mgs# lctl pool_add fsname.poolname ost_list
```

其中：

- ost_list 为 fsname-OST index_range
- index_range为 ost_index_start - ost_index_end 或 ost_index_start- ost_index_end/step

如果开头的 fsname和（或）结尾的 _UUID 被省略了，他们将自动被添加。

例如，增加偶数号的 OSTs 在文件系统testfs 的 pool1中，轻运行 pool_add（一次性添加多个 OSTs）：

```
1 lctl pool_add testfs.pool1 OST[0-10/2]
```

注意

每次有新的 OST 添加到池中，将创建新的llog 配置记录。为方便起见，您可以运行单个命令添加多个 OSTs。

从池中移除 OST，请运行：

```
1 mgs# lctl pool_remove
2 fsname.
3 poolname
4 ost_list
```

销毁池，请运行：

```
1 mgs# lctl pool_destroy
2 fsname.
3 poolname
```

注意

在该池被销毁之前，所有此池中的 OSTs 都必须被移除。

列出所指定文件系统中的所有池，运行：

```
1 mgs# lctl pool_list
2 fsname|pathname
```

列出指定池中所有的 OSTs，运行：

```
1 lctl pool_list
2 fsname.
3 poolname
```

23.2.1.1. 使用lfs命令操作 OST 池 一些lfs命令可以配合 OST 池进行操作。使用lfs setstripe可将目录与 OST 池相关联，即目录中的所有新常规文件和新目录也将在池中创建。lfs命令可用于列出文件系统中的池和池中的 OST。

将目录与池相关联，以使新文件和新目录都将在池中创建，请运行：

```
1 client# lfs setstripe --pool|-p pool_name
2 filename|dirname
```

设置条带模式，运行：

```
1 client# lfs setstripe [--size|-s stripe_size] [--offset|-o start_ost]
2             [--stripe-count|-c stripe_count] [--overstripe-count|-C
                 stripe_count]
3             [--pool|-p pool_name]
4
5 dir|filename
```

注意

使用无效的池名称（该池不存在或池名称错误）指定条带，lfs setstripe将返回错误。运行lfs pool_list以确保该池存在且名称输入正确。

lfs setstripe的-pool选项与其他修饰符兼容。例如，您可以在目录上为条带设置明确的起始索引。

23.2.2. OST 池使用建议

以下是使用 OST 池的一些建议：

- 目录和文件可以附加扩展属性 (EA)，使条带设置局限于池内。

- 可以使用池将相同技术或性能（更慢或更快）的 OSTs 分为一组，或者将某些作业偏好的 OSTs 分为一组。例如，可分为 SATA OST 和 SAS OST，或者远程 OST 与本地 OST。
- 在 OST 池中创建的文件通过将池名称保留在文件 LOV EA 中来跟踪池。

23.3. 在 Lustre 文件系统中添加 OST

在现存的 Lustre 文件系统中添加一个 OST：

1. 通过命令添加一个 OST：

```
oss# mkfs.lustre --fsname=testfs --mgsnode=mds16@tcp0 --ost --index=12
    /dev/sda
oss# mkdir -p /mnt/testfs/ost12
oss# mount -t lustre /dev/sda /mnt/testfs/ost12
```

2. 迁移数据。

新增空的 OST 将使文件系统非常不平衡。新文件创建将自动进行平衡。如果这是一个新的文件系统或者文件被定期修剪，那么可能不需要进一步的工作。在扩展之前存在的文件可以通过就地拷贝来重新平衡，只需简单的脚本来实现。

其基本方法为：复制现有文件到临时文件，然后用临时文件替换旧文件。注意不要在用户或应用程序正在写入的文件上进行操作。该操作将在整个 OST 集上重新进行分条。

一个明智的迁移脚本将执行以下操作：

- 检查当前数据分配。
- 计算有多少数据需要从每个满溢的 OST 迁移至空的 OST。
- 在给定的满溢的 OST 搜索文件（使用 `lfs getstripe`）。
- 限定目标 OST （使用 `lfs getstripe`）。
- 只复制刚好能够解决不平衡状态的数据。

如果 Lustre 文件系统管理员希望进一步探索此方法，可以在 `/proc/fs/lustre/osc/*/rpc_stats`下找到每个 OST 磁盘使用统计信息。

23.4. 实施直接 I/O

Lustre 软件支持打开 O_DIRECT 标志。

使用 `read()` 和 `write()` 调用的应用程序必须提供在页边界上对齐的缓冲区（通常为 4K）。如果对齐方式不正确，则将返回`-EINVAL`。在客户端执行大量 I/O 且受 CPU 性能限制（当 CPU 利用率达到 100%）的情况下，直接 I/O 可能有助于提高性能。

23.4.1. 将文件系统对象设置为不可变

不可变的文件或目录是指无法修改、重命名或删除的文件或目录。运行以下命令进行设置：

```
chattr +i
file
```

使用 `chattr -i`移除该标志。

23.5. 其它 I/O 选项

本节介绍其他 I/O 选项，包括校验和和 ptlrpcd 线程池。

23.5.1. Lustre 校验和

为了防止网络数据损坏，Lustre 客户端可以执行两种类型的数据校验和：内存（用于客户端内存中的数据）和线路（用于网络中传输的数据)。对于每种校验和类型，计算在客户端和服务器上读写数据的 32 位校验和，以确保数据在网络传输中未被破坏。`ldiskfs`备份文件系统不执行任何持久性校验和，因此它不检测 OST 文件系统中的数据损坏。

在默认情况下，校验和功能在客户端节点上启用。如果客户端或 OST 检测到校验和不匹配，则在系统日志中记录类似以下的错误：

```
LustreError: BAD WRITE CHECKSUM: changed in transit before arrival at OST: \
from 192.168.1.1@tcp inum 8991479/2386814769 object 1127239/0 extent [10240\
0-106495]
```

如果发生这种情况，客户端将重读/写受影响的数据五次，以便通过网络获取数据的完整副本。如果仍然不可行，则将返回 I/O 错误至应用程序。

启用两种类型的校验和（内存和线路），请运行：

```
lctl set_param llite.*.checksum_pages=1
```

禁用两种类型的校验和（内存和线路），请运行：

```
lctl set_param llite.*.checksum_pages=0
```

查看线路校验和的状态，请运行：

```
lctl get_param osc.*.checksums
```

23.5.1.1. 更改校验和算法 默认情况下，Lustre 软件使用 adler32 校验和算法，因其鲁棒性强且对性能影响相较 crc32 更低。Lustre 文件系统管理员可以通过`lctl get_param`更改校验和算法（具体取决于内核支持情况)。

要查看 Lustre 软件正在使用的校验和算法，请运行：

```
$ lctl get_param osc.*.checksum_type
```

更改线路校验和算法，请运行：

```
$ lctl set_param osc.*.checksum_type=
algorithm
```

注意

内存校验和总是使用 adler32 算法（如果可用），只有在 adler32 不能使用时才会回退到 crc32 算法。

在以下示例中，`lctl get_param`命令确定了 Lustre 软件正在使用 adler32 校验和算法，随后使用`lctl set_param`命令将校验和算法更改为 crc32，并运行第二个`lctl get_param`命令确认现在正在使用 crc32 校验和算法。

```
$ lctl get_param osc.*.checksum_type
osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum_type=crc32 [adler]
$ lctl set_param osc.*.checksum_type=crc32
osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum_type=crc32
$ lctl get_param osc.*.checksum_type
osc.testfs-OST0000-osc-ffff81012b2c48e0.checksum_type=[crc32] adler
```

23.5.2. Ptlrpc 客户端线程池

使用大型 SMP 节点的 Lustre 客户端需要在内核内有显著的并行性，以避免某个 CPU 达到 100％的利用率而其他 CPU 却相对空闲的情况。当一个单线程遍历一个大目录时，这个问题将尤其明显。

Lustre 客户端实现了一个 PtlRPC 守护程序线程池，以创建多个线程来提供异步 RPC 请求服务，即使只有一个用户空间的线程在运行。ptlrpcd 线程的数量是在模块加载时通过模块选项控制的。默认情况下，每个 CPU 槽会产生两个服务线程。

线程操作存在将线程上下文从一个 CPU 移动到另一个 CPU 的成本问题，这将导致 CPU 缓存的损耗。为了降低成本，可以将 ptlrpc 线程绑定到 CPU。但是，绑定的线程在 CPU 忙（可能忙于执行其他任务）时可能无法快速响应，且线程则必须等待。

考虑到这些因素，ptlrpc 线程池可以是绑定线程和非绑定线程的混合。系统操作员可以根据系统大小和工作量来进行平衡。

23.5.2.1. ptlrpcd 参数 这些参数作为 ptlrpc 模块的选项，应在`/etc/modprobe.conf`或`etc/modprobe.d`目录中设置.

```
options ptlrpcd ptlrpcd_per_cpt_max=XXX
```

设置每 CPU 槽创建 ptlrpcd 线程的数量。如果未指定，则默认为每槽一个线程，包括超线程 CPU。最多为每槽 2 线程。

```
1 options ptlrpcd ptlrpcd_bind_policy=[1-4]
```

有关线程与 CPU 的绑定，有以下四种策略：

- PDB_POLICY_NONE(ptlrpcd_bind_policy=1) 所有线程都不绑定。
- PDB_POLICY_FULL(ptlrpcd_bind_policy=2) 所有线程尝试绑定 CPU。
- PDB_POLICY_PAIR(ptlrpcd_bind_policy=3) 这是默认策略。线程被分配为绑定/非绑定对。每个线程（绑定或空闲）都有一个伙伴线程。ptlrpcd 加载策略时使用伙伴形式，决定线程在 CPU 上如何分配。
- PDB_POLICY_NEIGHBOR(ptlrpcd_bind_policy=4) 线程被分配为绑定/非绑定对。每个线程（绑定或空闲）都有两个伙伴线程。

第二十四章 Lustre 文件统故障切换和多挂载保护

24.1. 概览

多挂载保护（MMP）功能用于防止 Lustre 文件系统挂载到多个节点。此功能在共享存储环境中非常重要（如 OSS 故障转移对共享同一个 LUN 时）。

后端文件系统ldiskfs支持 MMP 机制。文件系统块由kmmpd守护进程每隔 1 秒更新一次，在块中写入序列号。如果文件系统被完全卸载，则在该块中写入一个特殊的"clean" 序列。挂载文件系统时，ldiskfs将会检查 MMP 块是否包含"clean" 序列。

即使 MMP 块包含该"clean" 序列，为防止出现以下情况，ldiskfs也会等待一段时间：

- 如果 I/O 流量很大，MMP 块更新可能需要更长的时间。
- 如果另一个节点试图挂载相同的文件系统，则可能发生" 竞争" 情况。

启用 MMP 后，挂载一个干净的文件系统至少需要 10 秒。如果文件系统未被完全卸载，则文件系统挂载可能需要更多时间。

注意

MMP 功能仅在 Linux 2.6.9 及更新的内核版本上得到支持。

24.2. 多挂载保护相关操作

在新的 Lustre 文件系统上，如果使用了故障转移且内核和e2fsprogs版本支持，mkfs.lustre会在格式化时自动启用 MMP。在现有文件系统上，Lustre 文件系统管理员可以在卸载文件系统时手动启用 MMP。

使用以下命令确定 MMP 是否正在该 Lustre 文件系统中运行，从而启用或禁用 MMP 功能。

确定 MMP 是否已启用，请运行：

```
dumpe2fs -h /dev/block_device | grep mmp
```

输出示例如下：

```
dumpe2fs -h /dev/sdc | grep mmp
Filesystem features: has_journal ext_attr resize_inode dir_index
filetype extent mmp sparse_super large_file uninit_bg
```

手动禁用 MMP，请运行：

```
tune2fs -O ^mmp /dev/block_device
```

手动启用 MMP，请运行：

```
tune2fs -O mmp /dev/block_device
```

启用 MMP 后，如果ldiskfs在文件系统挂载后检测到多次挂载尝试，将会阻止这些挂载尝试，并报告上次更新 MMP 块的时间、节点名称和当前文件系统所挂载的设备名。

第二十五章配额配置和管理

25.1. 配额相关操作

设置配额将允许系统管理员对用户、组或项目可使用的磁盘空间量进行限制。配额由 root 用户设置，可针对个人用户、组或项目进行指定。文件写入配置了配额的分区之前，将检查创建者所在组的配额情况。如果存在配额，则文件大小将计入组的配额。如果不存在配额，则在写入文件之前检查所有者的用户配额。同样地，如果用户过度使用分配的空间，也可对特定功能的 inode 使用情况进行控制。

Lustre 配额执行时与标准的 Linux 配额在以下几个方面有所不同：

- 配额通过lfs和lctl命令进行管理（挂载后）。
- Lustre 软件中的配额功能分布在整个系统中（因为 Lustre 文件系统是分布式文件系统）。因此，Lustre 上的配额设置和行为与本地磁盘配额在以下几个方面有些不同：
- 没有单一的管理节点：一些命令必须在 MGS 上执行，一些必须在 MDSs 和 OSSs 上执行，另一些必须要在客户端上执行。
- 粒度：本地配额通常指定千字节分辨率，而 Lustre 使用的最小配额分辨率为 1 兆字节。

- 准确度：配额信息分布在整个文件系统中，只有对静止的文件系统才能进行精确计算，以尽可能减少正常使用时的性能开销。
- 配额以量化方式分配和使用。
- 客户端挂载时不设置usrquota或grpquota选项。空间计算功能在默认情况下始终处于启用状态，同时可以使用lctl conf_param在每个文件系统基础上启用或禁用配额功能。

(Lustre 2.8 引入)

值得注意的是，lfs quotaon、lfs quotaoff、lfs quotacheck和quota_type子命令从 Lustre 2.4.0 开始废弃，并在 Lustre 2.8.0 中完全删除。

注意

尽管 Lustre 软件中提供了配额功能，但不会强制执行 root 配额。

lfs setquota -u root －不强制执行配额。

lfs quota -u root －显示 Lustre 内部数据（大小动态变化且不能准确反映挂载点可见块）和 inode 使用情况。

25.2. 启用磁盘配额

Lustre 的配额设计将管理和执行与资源使用和计算分开。Lustre 软件负责管理和执行，后端文件系统负责资源使用和计算。因此，必须先在后端磁盘系统上启用配额。

注意

配额设置由 MGS 编排，本节中的所有设置命令必须在 MGS 上运行。项目配额设置则需要 Lustre 2.10 或更高版本。根据不同的内核版本和后端系统类型，决定是否需要补丁服务器。

配置	是否需要补丁服务器
ldiskfs 且内核版本低于 4.5	配额从设备发送获取配额的请求并等待回复。
ldiskfs 且内核版本为 4.5 及以上	不需要
zfs 版本为 0.8 及以上，内核版本低于 4.5	需要
zfs 版本为 0.8 及以上，内核版本为 4.5 及以上	需要

* 注意：低于 0.8 的 zfs 版本不支持项目配额。

设置完成后，必须在 MDT 上执行配额状态的验证。尽管配额执行由 Lustre 软件管理，但每个 OSD 的实施依赖于后端文件系统来维护每个用户/组/项目的 block/inode 使用。因此，使用 ldiskfs 和 ZFS 后端设置配额存在差异。

- ldiskfs 后端。`mkfs.lustre`创建空配额文件并启用超级块中的 QUOTA 功能标志，该标志会在挂载时自动打开配额功能。当 QUOTA 功能标志存在时，通过修改`e2fsck`修复配额文件。项目配额功能在默认情况下处于禁用状态，需要运行`tune2fs`手动启用每个目标。如果用户、组和项目配额使用不一致，则在所有未挂载的 MDT 和 OST 上运行`e2fsck -f`。
- ZFS 后端。ZFS 低于 0.8.0 的版本尚不支持项目配额功能。计算 ZAP 由 ZFS 文件系统创建和维护。虽然 ZFS 可以跟踪每个用户和组块的使用情况，但它并不处理 zfs-0.7.0 之前的 ZFS 版本的 inode 计算。ZFS OSD 本身提供了 inode 跟踪支持。有两个选项可用：

1. ZFS OSD 根据给定用户或组使用的块的数量来估计正在使用的 inode 的数量。启用该模式，请在目标的服务器上运行以下命令：`lctl set_param osd-zfs.${FSNAME}-${TARGETNAME}.quota_iused_estimate=1`。
2. 与块计算类似，专用 ZAP 也创建了 ZFS OSD 来维护每个用户和组 inode 的使用。默认模式下，`quota_iused_estimate`被设置为 0 的默认模式。

注意

如需在 ldiskfs 文件系统上（重新）启用空间使用配额，则需对所有目标运行`tune2fs -O quota`。该命令将设置超级块中的 QUOTA 功能标志，并在内部运行`e2fsck`（故目标必须处于脱机状态）以针对每个 UID/GID 的磁盘使用情况构建数据库。

被格式化为 Lustre 2.10 版本之前的 Lustre 文件系统仍然可以安全地升级到版本 2.10，但只有对所有 ldiskfs 后端目标运行`tune2fs -O project`或者在 2.15.0 版本上才会有项目配额使用情况报告功能。该命令将设置超级块中的 PROJECT 功能标志，并运行 e2fsck（故目标必须处于脱机状态）。

注意

在使用 ldiskfs 后端的服务器节点上，Lustre 须安装支持配额的 e2fsprog 版本（ZFS 后端不需 e2fsprogs）。通常，我们建议使用https://downloads.hpdd.intel.com/public/e2fsprogs/上提供的最新 e2fsprogs 版本。

ldiskfs OSD 依赖标准的 Linux 配额功能来维护磁盘上的配额计算信息。因此，如果使用了 ldiskfs 后端，在 Lustre 服务器上运行的的 Linux 内核必须启用`CONFIG_QUOTA`，`CONFIG_QUOTACTL`和`CONFIG_QFMT_V2`。

配额执行功能可独立地打开或关闭，而与始终启用的空间计算功能无关。存在一个单一的每文件系统的配额参数来控制 inode/block 配额执行。像所有永久参数一样，此配额参数可以通过 MGS 上的`lctl conf_param`按照以下语法来设置：

```
1 lctl conf_param fsname.quota.ost|mdt=u|g|p|ugp|none
```

- `ost`：配置 OSTs 管理的块配额
- `mdt`：配置 MDTs 管理的 inode 配额
- `u`：为用户启用配额执行功能
- `g`：为组启用配额执行功能
- `p` -- 为项目启用配额执行功能
- `ugp` -- 为用户、组及项目启用配额执行功能
- `none` --禁用用户、组及项目的配额执行功能

示例：

在文件系统`testfs1`上打开针对块的用户、组、项目配额功能，在 MGS 上运行：

```
$ lctl conf_param testfs1.quota.ost=ugp
```

在文件系统`testfs2`上打开针对 inode 的组配额功能，在 MGS 上运行：

```
mgs# lctl conf_param testfs2.quota.mdt=g
```

在文件系统`testfs3`上关闭针对块和 inode 的用户、组、项目配额功能，在 MGS 上运行：

```
mgs# lctl conf_param testfs3.quota.ost=none
mgs# lctl conf_param testfs3.quota.mdt=none
```

25.2.1. 配额验证

配额参数完成配置后，属于文件系统一部分的所有目标将自动收到新配额设置，并根据需要启用/禁用配额。通过在 MDS 上运行以下命令，可验证每个目标的执行状态：

```
$ lctl get_param osd-..quota_slave.info
osd-zfs.testfs-MDT0000.quota_slave.info=
target name:    testfs-MDT0000
pool ID:        0
type:           md
quota enabled:  ug
conn to master: setup
user uptodate:  glb[1],slv[1],reint[0]
group uptodate: glb[1],slv[1],reint[0]
```

25.3. 配额管理

文件系统启动并运行后，可以为用户、组和项目设置块和 inode 的配额限制。这完全由客户端通过三个配额参数进行控制：

Grace period（宽限期） - 允许用户超过软限制的时间段（以秒为单位）。有以下六种类型:

- 用户块软限制
- 用户 inode 软限制
- 组块软限制
- 组 inode 软限制
- 项目块软限制
- 项目 inode 软限制

宽限期适用于所有用户。例如，用户块软限制针对的是所有使用块配额的用户。

Soft limit（软限制） -- 宽限定时器在超过软限制时启动。此时，用户、组、项目仍然可以分配 block 或 inode。当宽限时间到期并且用户仍然高于软限制时，软限制将变为硬限制，用户、组、项目不能再分配任何新的 block/inode。随后，用户、组、项目应该删除在软限制下的文件。软限制必须小于硬限制。如果不需要软限制，则应将其设置为 0。

Hard limit（硬限制） -- 当达到硬限制时，block 或 inode 分配将失败，并伴随 EDQUOT 标志（即超出配额）。硬限制是绝对限制。如果设置了宽限期，则在硬限制以下可以在宽限期内超过软限制。

由于 Lustre 文件系统的分布式特性以及保证负载下性能的需求，这些配额参数可能不是百分之百准确。配额设置可以通过在客户端上执行`lfs`命令进行操作，包含以下选项:

- `quota` -- 显示通用配额信息（磁盘使用情况和限制）
- `setquota` -- 指定配额限制，调试宽限期。默认情况下，宽限期为一周。

用法:

```
lfs quota [-q] [-v] [-h] [-o obd_uuid] [-u|-g|-p
   uname|uid|gname|gid|projid] /mount_point
lfs quota -t {-u|-g|-p} /mount_point
lfs setquota {-u|--user|-g|--group|-p|--project} username|groupname [-b
   block-softlimit] \
            [-B block_hardlimit] [-i inode_softlimit] \
            [-I inode_hardlimit] /mount_point
```

显示当前运行命令用户及其主要组的通用配额信息（磁盘使用情况和限制），请运行:

```
$ lfs quota /mnt/testfs
```

显示指定用户（如下例中的bob）的通用配额信息，请运行：

```
$ lfs quota -u bob /mnt/testfs
```

显示指定用户（如下例中的bob）的通用配额信息以及针对每个 MDT 和 OST 的详细配额统计信息，请运行：

```
$ lfs quota -u bob -v /mnt/testfs
```

显示指定项目（如下例中的 ``1"）的通用配额信息，请运行：

```
$ lfs quota -p 1 /mnt/testfs
```

显示指定组（如下例中的 ``eng"）的通用配额信息，请运行：

```
$ lfs quota -g eng /mnt/testfs
```

指定目录下的某一项目设置配额限制（如下例中的"/mnt/testfs/dir"），请运行：

```
$ lfs project -s -p 1 -r /mnt/testfs/dir
$ lfs setquota -p 1 -b 307200 -B 309200 -i 10000 -I 11000 /mnt/testfs
```

递归地列出目录中（本例中为 ``/mnt/testfs/dir"）的所有子项目属性，请运行：

```
$ lfs project -r /mnt/testfs/dir
```

请注意，如果要使用 lfs quota -p 正确显示目录下的 space/inode 使用率（比du快得多），则用户或管理员需要为不同的目录使用不同的项目 ID。

显示用户配额的 block/inode 宽限期：

```
$ lfs quota -t -u /mnt/testfs
```

为指定 ID 设置用户或组配额（如下例中的bob），运行：

```
$ lfs setquota -u bob -b 307200 -B 309200 -i 10000 -I 11000 /mnt/testfs
```

在这个例子汇总，"bob" 的配置被设置为 300MB (309200*1024)，硬限制被设置为 11000 个文件。因此，inode 硬限制应为 11000。

lfs quota命令显示了每个 Lustre 目标分配和使用的配额：

```
$ lfs quota -u bob -v /mnt/testfs
```

输出内容为：

```
Disk quotas for user bob (uid 6000):
Filesystem          kbytes quota limit grace files quota limit grace
/mnt/testfs         0      30720 30920 -     0     10000 11000 -
testfs-MDT0000_UUID 0      -      8192 -     0     -     2560  -
```

```
5 testfs-OST0000_UUID 0      -      8192 -     0     -     0     -
6 testfs-OST0001_UUID 0      -      8192 -     0     -     0     -
7 Total allocated inode limit: 2560, total allocated block limit: 24576
```

全局配额限制被存在配额主目标 (QMT) 上的专用索引文件中（每种配额类型都有一个索引)。QMT 在 MDT0000 上运行并通过*lctl get_param*输出全局索引。全局索引可以通过以下命令转出：

```
1 # lctl get_param qmt.testfs-QMT0000.*.glb-*
```

全局索引的格式取决于 OSD 类型。ldiskfs OSD 使用 IAM 文件，而专用 ZAP 由 ZFS OSD 创建。

每个从机也在本地存储这个全局索引的副本。当全局索引在主机上修改时，会在全局配额锁上发出"glimpse callback"，以通知所有从机全局索引已被修改。这个"glimpse callback" 包括受更改影响的标识符的信息。如果 QMT 上的全局索引在从机断开连接时被修改，则索引版本用于确定全局索引的从属副本是否不再最新。如果是这样，从机将再次获取整个索引并更新本地副本。全局索引的从副本也通过/proc导出，并可通过以下命令访问：

```
1 lctl get_param osd-*.*.quota_slave.limit*
```

(Lustre 2.12 引入)

25.4 默认配额

默认配额强制对所有管理员没有设置配额的用户、组或项目执行配额限制。默认配额可以通过将限额设置为 0 来禁用。

25.4.1 用法

```
1 lfs quota [-U|--default-usr|-G|--default-grp|-P|--default-prj] /mount_point
2 lfs setquota {-U|--default-usr|-G|--default-grp|-P|--default-prj} [-b
     block-softlimit] \
3  [-B block_hardlimit] [-i inode_softlimit] [-I inode_hardlimit] /mount_point
4 lfs setquota {-u|-g|-p} username|groupname -d /mount_point
```

设置默认用户配额

```
1 # lfs setquota -U -b 10G -B 11G -i 100K -I 105K /mnt/testfs
```

设置默认组配额

```
1 # lfs setquota -G -b 10G -B 11G -i 100K -I 105K /mnt/testfs
```

设置默认项目配额

```
1 # lfs setquota -P -b 10G -B 11G -i 100K -I 105K /mnt/testfs
```

禁用默认用户配额

```
1 # lfs setquota -U -b 0 -B 0 -i 0 -I 0 /mnt/testfs
```

禁用默认组配额

```
1 # lfs setquota -G -b 0 -B 0 -i 0 -I 0 /mnt/testfs
```

禁用默认项目配额

```
1 # lfs setquota -P -b 0 -B 0 -i 0 -I 0 /mnt/testfs
```

注意

如果为某些用户、组或项目设置了配额限制，则将使用这些特定的配额限制而不是默认配额。如果配额限制设置为 0，则所有用户、组或项目的配额限制将使用默认配额。

25.5. 配额分配

在 Lustre 文件系统中，配额必须正确分配，否则用户将可能遇到不必要的故障。文件系统块配额在文件系统内的 OSTs 之间分配。每个 OST 请求分配的额度都被将被添加到配额限制里。Lustre 通过量化配额分配减少配额请求相关流量。

Lustre 配额系统中，配额主目标（QMT）负责分配配额。目前，Lustre 仅支持一个 QMT 实例，且只能在类似 MDT0000 的节点上运行。但所有的 OST 和 MDT 都建立了配额从设备（QSD），它们通过连接到 QMT 来分配和释放配额空间。QSD 直接在 OSD 层进行设置。

为了减少配额请求，最初配额空间以非常大的块分配给 QSDs。一个目标可以容纳多少未使用的配额空间由 qunit 大小控制。当给定 ID 的配额空间在 QMT 上快要耗尽时，qunit 大小将会减少，QSD 将通过"glimpse callback" 获悉新的 qunit 大小值。随后，从设备需要释放比新的 qunit 值更大的配额空间。qunit 大小不会无限缩小，对于块来说，其最小值为 1MB，对于 inodes 来说，其最小值为 1024。这意味着达到此最小值时配额空间重新平衡过程将停止。因此，即使许多从设备还有 1MB 块或 1024 个 inode 的剩余配额空间，仍会返回配额超标的消息。

如果我们再次查看`setquota`示例，运行以下`lfs quota`命令:

```
1 # lfs quota -u bob -v /mnt/testfs
```

输出为:

```
1 Disk quotas for user bob (uid 500):
2 Filesystem          kbytes quota limit grace       files  quota limit grace
3 /mnt/testfs         30720* 30720 30920 6d23h56m44s 10101* 10000 11000
```

```
6d23h59m50s
testfs-MDT0000_UUID 0      -     0     -           10101  -     10240
testfs-OST0000_UUID 0      -     1024  -           -      -     -
testfs-OST0001_UUID 30720* -     29896 -           -      -     -
Total allocated inode limit: 10240, total allocated block limit: 30920
```

总共 30920 的配额限制被分配给了用户`bob`，又进一步分配给了两个 OSTs。

如上所示，值后面如果跟着 `*` ，表明已超过配额限制，尝试写入或创建文件将返回以下错误:

```
$ cp: writing `/mnt/testfs/foo`: Disk quota exceeded.
```

注意

值得请注意的是，每个 OST 上的块配额以及每个 MDS 上的 inode 配额都会被消耗。因此，如果其中一个 OST（或 MDT）上配额已用尽，客户端将可能无法创建文件，尽管其他 OSTs（或 MDTs）上还有可用配额。

将配额限制设置得比最小 qunit 更低可能会使用户或组无法创建所有文件。因此建议使用软/硬限制（OST 数量和最小 qunit 大小的乘积）。

请使用`lfs df -i`（以及`lctl get_param *.*.filestotal`）确定 inode 的总数。

`statfs`接口不直接报告空闲 inode 计数，而是报告总 inode 数和已使用的 inode 数。空闲 inode 计数是由`df`（总 inodes - 使用的 inode）计算得到。尽管知晓文件系统的总 inode 数并不重要，但您应该知道（准确的）空闲 inode 数和已使用的 inode 数。Lustre 软件通过操纵 inode 总计数，以准确报告其他两个值。

25.6. 配额和版本互操作性

（Lustre 2.10 引入）

要使用 Lustre 2.10 中引入的项目配额功能，必须将所有 Lustre 服务器和客户端升级到 Lustre 版本 2.10 或更高版本，项目配额才能正常工作。否则，客户端将无法访问项目配额，也无法在 OSTs 上进行核算。另外，**服务器可能还需要使用补丁内核**，更多信息见第 25.2 节*启用磁盘配额*。

（Lustre 2.14 引入）

如果项目配额限制较小，运行命令`df`与`lfs df`将返回该项目的可用空间大小，而不是文件系统的总空间大小。**只有客户端需要升级到 Lustre 2.14 版本或更高版本时，这两个命令才会返回以上结果。**

25.7 在. 授权缓存和配额限制

在 Lustre 文件系统中，授权缓存并不受配额限制影响。为加速 I/O，OSTs 会向 Lustre 客户端授权缓存。该缓存使数据即使超过 OSTs 配额，仍能成功写入，并重写配额限制。

顺序是：

1. 用户将文件写入 Lustre 文件系统。2. 如果 Lustre 客户端拥有足够的授权缓存，则会向用户返回" 成功" 并安排在 OSTs 上的写入操作。3. 因为 Lustre 客户已经向用户返回" 成功"，OST 不能使这些写入失败。

由于授权缓存，写入操作将始终重新配额限制。例如，如果您为用户 A 设置 400GB 的配额并使用 IOR 从一批客户端为用户 A 写入数据，则您将写入比 400GB 多得多的数据，最终导致超出配额的错误（EDQUOT）。

注意

授权缓存对配额限制的作用可以得到缓解，但无法消除。运行以下命令减少客户端上脏数据最大值（最小值为 1MB）：

- `lctl set_param osc.*.max_dirty_mb=8`

25.8. Lustre 配额统计信息

Lustre 软件可以收集监控配额活动的统计信息，如特定期间发送的配额 RPC 类型、完成 RPC 的平均时间等。这些统计信息对于衡量 Lustre 文件系统的性能很有用。

每个配额统计信息由配额事件和`min_time`，`max_time`和`sum_time`值组成。

配额事件	说明
sync_acq_req	配额从设备发送获取配额的请求并等待回复。
sync_rel_req	配额从设备发送释放配额的请求并等待回复。
async_acq_req	配额从设备发送获取配额的请求但不等待回复。
async_rel_req	配额从设备发送释放配额的请求但不等待回复。
wait_for_blk_quota (lquota_chkquota)	在数据写入 OSTs 之前，OSTs 将检查剩余块配额是否足够。这将在 lquota_chkquota 函数中完成的。
wait_for_ino_quota (lquota_chkquota)	在 MDS 上创建文件之前，MDS 检查剩余的 inode 配额是否足够。这将在 lquota_chkquota 函数中完成的。
wait_for_blk_quota	将块写入 OST 后，会更新相关配额信息。这是在

配额事件	说明
(lquota_pending_commit)	lquota_pending_commit 函数中完成的。
wait_for_ino_quota	文件完成创建后，会更新相关配额信息。这是在
(lquota_pending_commit)	lquota_pending_commit 函数中完成的。
wait_for_pending_blk_quota_req	在 MDS 或 OSTs 上，有一个线程随时为特定
(qctxt_wait_pending_dqacq)	UID/GID 发送块配额请求。其他线程发送配额请求则需要等待。这是在 qctxt_wait_pending_dqacq 函数中完成的。
wait_for_pending_ino_quota_req	在 MDS 上，有一个线程随时为特定 UID/GID
(qctxt_wait_pending_dqacq)	发送 inode 配额请求。其他线程发送配额请求则需要等待。这是在 qctxt_wait_pending_dqacq 函数中完成的。
nowait_for_pending_blk_quota_req	在 MDS 或 OSTs 上，有一个线程随时为特定
(qctxt_wait_pending_dqacq)	UID/GID 发送块配额请求。当线程进入 qctxt_wait_pending_dqacq 时，无需再等待。这是在 qctxt_wait_pending_dqacq 函数中完成的。
nowait_for_pending_ino_quota_req	在 MDS 上，有一个线程随时为特定 UID/GID
(qctxt_wait_pending_dqacq)	发送 inode 配额请求。当线程进入 qctxt_wait_pending_dqacq 时，无需再等待。这是在 qctxt_wait_pending_dqacq 函数中完成的。
quota_ctl	使用 lfs `setquota`，`lfs quota` 等将生成 quota_ctl 统计信息。
adjust_qunit	每当 qunit 发生调整时，都将被记录。

25.8.1. 解析配额统计信息

配额统计是衡量 Lustre 文件系统性能的重要指标。正确解析这些统计信息可以帮助您诊断配额问题，并做出一些调整，以提高系统性能。

例如，如果您在 OST 上运行此命令：

```
lctl get_param lquota.testfs-OST0000.stats
```

您将得到类似以下的结果：

```
snapshot_time                                 1219908615.506895 secs.usecs
async_acq_req                               1 samples [us]  32 32 32
async_rel_req                               1 samples [us]  5 5 5
nowait_for_pending_blk_quota_req(qctxt_wait_pending_dqacq) 1 samples [us] 2\
 2 2
quota_ctl                             4 samples [us]  80 3470 4293
adjust_qunit                                1 samples [us]  70 70 70
....
```

在第一行中，`snapshot_time` 表明获得这些数据的时间。其余行列出了配额事件及其相关数据。

在第二行中，`async_acq_req`事件发生一次。此事件的`min_time`，`max_time`和`sum_time`分别为 32、32 和 32。单位是微秒（µs）。

在第五行中，`quota_ctl`事件发生四次。此事件的`min_time`，`max_time`和`sum_time`分别为 80、3470 和 4293。单位是微秒（µs）。

（在 **Lustre 2.14** 中引入）

25.9 池配额

OST 池配额功能提供了一种在 OST 池级别上限制用户（组/项目）对磁盘使用的能力。每个 OST 池配额（Pool Quotas, PQ）直接映射到同名的 OST 池。因此，PQ 可以用标准的`lctl pool_new/add/remove/erase`命令调整大小。所有的 PQ 都是全局池的一个子集，全局池包括所有的 OST 和 MDT（DOM 实例）。由于使用不同的配额设置，而限制无法使用 ``全部" 配额，该功能听上去可能会令人感到困惑。在 Lustre 中，配额的实质是限制，而非使用一定额度的权利。客户并不总是能使用其配额，如 OST 可能已经没有空间了，或者存在其他配额限制。例如，如果同时存在节点配额和空间配额，由于受到节点配额限制，即使仍然有足够的空间也无法使用。此外，配额可能很容易被过度分配：在一个 15PB 的系统中，每个人都分配 10PB 的配额。这不是意味着他们有使用 10PB 的权利，而是表示使用的空间不能超过 10PB。他们很可能在那之前就获取了 ENOSPC - 但不会获得 EDQUOT。Lustre 中现在已经包含这种行为，但是池配额增加

了起作用的限制的数量：用户、组或项目全局空间配额，以及现在还可以为每个池单独定义所有这些配额限制。如果多种配额同时存在，实际效果是可以使用的实际空间量限制在所有适用的最小（min）配额中。参见 OST 池配额 HLD 中的更多细节 [http://wiki.lustrefs.cn/index.php?title=OST 池配额概要设计]。

25.9.1. DOM 池与 MDT 池

从 Quota Master 的角度来看，``数据"MDT 和 OST 都是普通成员。然而，池配额只支持 OST，因为目前没有机制将 MDT 分组到池中。

25.9.2. 用于设置配额池的 Lfs quota/setquota 选项

同样的长选项`--pool`用于设置和报告`lfs setquota`和`lfs` 的`setquota`池配额。

`lfs setquota --pool_name`用于设置用户、组或项目的块与使用软限制，用于指定池的名称。

`lfs quota --pool_name`显示指定池名的用户、组或项目的使用情况。

25.9.3. 配额池的交互性

客户端和服务器都至少需要 Lustre 2.14 来支持池配额。**注意**如果服务器支持池配额的话，那么池配额可以在低版本的客户端上运行，但无法查看或修改池配额。由于配额的执行是在服务器上完成的，所以只需要一个客户端来配置配额。如果需要的话，可以通过在 MDS 上直接挂载一个客户端来完成。

25.9.4. 池配额设置硬限制示例

让我们设想一下，你需要为已经存在的 OST 池的 flash_pool 设置配额限制：

```
1 # it is a limit for global pool. PQ don't work properly without that
2 lfs setquota -u ivan -B100T /mnt/testfs
3 # set 1TiB block hard limit for ivan in a flash_pool
4 lfs setquota -u ivan --pool flash_pool -B1T /mnt/testfs
```

在设置池配额限制之前，首先需要有系统侧的硬限制。如果你不需要在系统的所有 OST 和 MDT 上限制用户，只需要在每个池上限制，建议将硬设置设置为一些合理范围外的超大的值。如果没有全局限制，则不会强制执行配额池限制。所以至少应该设置一个全局限制，不管是硬限制还是软限制。

25.9.5. 池配额设置软限制示例

```
# notify OSTs to enforce quota for ivan
lfs setquota -u ivan -B10T /mnt/testfs
# soft limit 10MiB for ivan in a pool flash_pool
lfs setquota -u ivan --pool flash_pool -b1T /mnt/testfs
# set block grace 600 s for all users at flash_pool
lfs setquota -t -u --block-grace 600 --pool flash_pool /mnt/testfs
```

第二十六章分层存储管理 (HSM)

26.1. 简介

Lustre 文件系统可以使用一组特定的功能绑定到分层存储管理（HSM）解决方案。这些功能可将 Lustre 文件系统连接到一个或多个外部存储系统（通常是 HSM）。通过绑定到 HSM 解决方案，Lustre 文件系统可以作为高速缓存在这些速度较慢的 HSM 存储系统的前端工作。

Lustre 文件系统与 HSM 的集成提供了一种机制，使文件同时存在于 HSM 解决方案中，并在 Lustre 文件系统中存有元数据条目可供检查。读取，写入或截断文件将触发文件数据从 HSM 存储中取回到 Lustre 文件系统中。

将文件复制到 HSM 存储器的过程称为存档。存档完成后，便可删除 Lustre 文件数据（即释放）。将数据从 HSM 存储取回到 Lustre 文件系统的过程称为恢复。存档和恢复操作需要用到名为"Agent"（代理）的 Lustre 文件系统组件。

代理是为装载处理中的 Lustre 文件系统而专门设计的 Lustre 客户端节点。在代理上，运行有一个名为"copytool"（复制工具）的用户空间程序，以协调 Lustre 文件系统和 HSM 解决方案之间文件的存档和恢复。

恢复给定文件的请求由 MDT 上的"coordinator"（协调器）进行注册和分派。

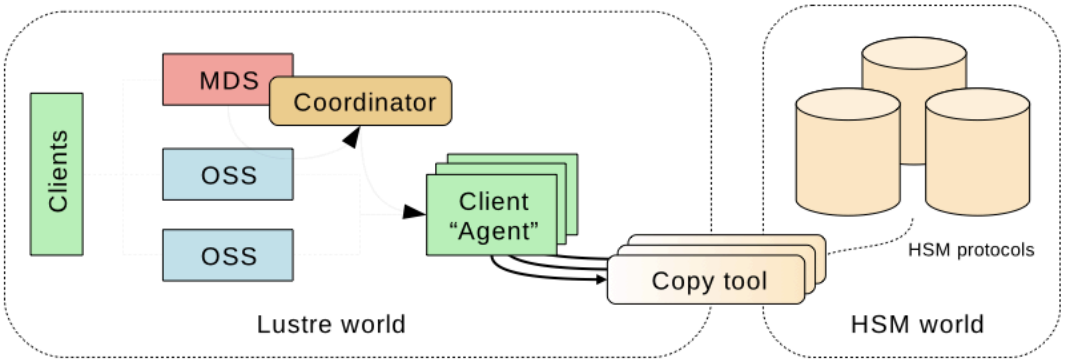


图 26: Overview of the Lustre file system HSM

图 25.1 Lustre 文件系统 HSM 总览

26.2. 设置

26.2.1. 要求

设置 Lustre/HSM 配置，您需要：

- 标准 Lustre 文件系统（2.5.0 及以上版本）
- 最少两个客户端，一个用于生成有效数据的计算任务，一个作为代理。

可以使用多种代理。所有代理都需要共享对后端存储的访问。对于 POSIX copytool 来说，像 NFS 或其他 Lustre 文件系统这样的 POSIX 名称空间是合适的。

26.2.2. 协调器（coordinator）

将 Lustre 文件系统绑定到 HSM 系统上，必须在每个文件系统 MDT 上激活协调器，请运行：

```
$ lctl set_param mdt.$FSNAME-MDT0000.hsm_control=enabled
mdt.lustre-MDT0000.hsm_control=enabled
```

确认协调器已被正常启用：

```
$ lctl get_param mdt.$FSNAME-MDT0000.hsm_control
mdt.lustre-MDT0000.hsm_control=enabled
```

26.2.3. 代理（agent）

协调器启动后，在每个代理节点载入复制工具（copytool）以连接到你的 HSM 存储。如果你的 HSM 存储可以进行 POSIX 访问，则该命令为：

```
lhsmtool_posix --daemon --hsm-root $HSMPATH --archive=1 $LUSTREPATH
```

POSIX copytool 只能通过发送 TERM 信号来关闭。

26.3. 代理（Agents）和复制工具（copytool）

代理是运行 copytool 的 Lustre 文件系统客户端，而 copytool 是一个在 Lustre 和 HSM 解决方案之间传输数据的用户空间守护程序。由于不同的 HSM 解决方案使用不同的 API，copytools 通常只能与特定的 HSM 一起使用。代理节点只能运行一个 copytool。

以下规则适用于 copytool 实例：Lustre 文件系统每个客户端节点，每个 ARCHIVE ID（请参见下文）仅支持一个 copytool 进程。这是受制于 Lustre 软件，与代理挂载的 Lustre 文件系统的数量无关。

与 Lustre 工具捆绑在一起，POSIX copytool 可以与任何导出 POSIX API 的 HSM 或外部存储一起使用。

26.3.1. ARCHIVE ID 及多后端系统

Lustre 文件系统可以绑定到几种不同的 HSM 解决方案。每个绑定的 HSM 解决方案由 ARCHIVE ID 标识。必须为每个绑定的 HSM 解决方案选择唯一的 ARCHIVE ID 值，且其值必须介于 1 到 32 之间。

Lustre 文件系统支持无限数量的 copytool 实例。每个 ARCHIVE ID 至少需要一个 copytool。当使用 POSIX copytool 时，通过`--archives`开关定义 ID。

例如，如果单个 Lustre 文件系统绑定到 2 个不同的 HSMs (A 和 B)，则可以选择 ARCHIVE ID ``1" 作为 HSM A 的标识，ARCHIVE ID ``2" 作为 HSM B 的标识。如果为 ARCHIVE ID 1 启动 3 个 copytool 实例，则这三个实例都将使用 Archive ID ``1" 标识。同样的规则也适用于处理使用 Archive ID ``2" 为标识的 HSM B 的 copytool 实例。

发出 HSM 请求时，您可以使用`--archive`开关来选择要使用的后端。在本例中，文件`foo`将被存档到后端 ARCHIVE ID ``5" 中:

```
$ lfs hsm_archive --archive=5 /mnt/lustre/foo
```

当未指定`--archive`开关时，可使用默认 ARCHIVE ID。定义默认 ARCHIVE ID:

```
$ lctl set_param -P mdt.lustre-MDT0000.hsm.default_archive_id=5
```

运行`lfs hsm_state`命令查看已归档文件的 ARCHIVE ID:

```
$ lfs hsm_state /mnt/lustre/foo
/mnt/lustre/foo: (0x00000009) exists archived, archive_id:5
```

26.3.2. 注册代理

Lustre 文件系统为每个文件系统的每个客户端挂载点分配唯一 UUID。每个 Luster 挂载点只能注册一个 copytool。因此，在每个文件系统中，UUID 也是 copytool 的唯一标识。

通过在 MDS 节点上（每个 MDT）运行以下命令，可以检索当前注册的 copytool 实例（代理 UUID）:

```
$ lctl get_param -n mdt.$FSNAME-MDT0000.hsm.agents
uuid=a19b2416-0930-fc1f-8c58-c985ba5127ad archive_id=1 requests=[current:0
    ok:0 errors:0]
```

返回的值域为:

- `uuid`： 此 copytool 使用的客户端挂载点。
- `archive_id`： 此 copytool 可访问的 ARCHIVE ID 列表（ID 之间由逗号隔开）。
- `requests`： 有关此 copytool 处理的请求的各种统计信息。

26.3.3. 超时

一个或多个 copytool 实例可能会遇到导致它们无法响应的情况。为避免系统阻塞对相关文件的访问，我们为请求处理定义了一个超时值。copytool 必须在这段时间内完全完成请求，其默认值为 3600 秒。

```
1 $ lctl set_param -n mdt.lustre-MDT0000.hsm.active_request_timeout
```

26.4. 请求

文件系统和 HSM 解决方案之间的数据管理是由请求驱动的。有以下五种类型：

- `ARCHIVE`： 从 Lustre 文件系统拷贝数据至 HSM 解决方案。
- `RELEASE`： 从 Lustre 文件系统移除数据。
- `RESTORE`： 从 HSM 解决方案拷回数据至相应的 Lustre 文件系统。
- `REMOVE`： 从 HSM 解决方案中删除拷贝数据。
- `CANCEL`： 取消进行中或等待中的请求。

只有 `RELEASE` 是同步进行且不需要协调器配合的操作。其他请求由协调器处理，每个 MDT 协调器对它们进行弹性的管理。

26.4.1. 命令

请求通过 `lfs` 命令提交：

```
1 $ lfs hsm_archive [--archive=ID] FILE1 [FILE2...]
2 $ lfs hsm_release FILE1 [FILE2...]
3 $ lfs hsm_restore FILE1 [FILE2...]
4 $ lfs hsm_remove  FILE1 [FILE2...]
```

如果没有通过 `--archive` 指定 ARCHIVE ID，请求将被发送到默认 ARCHIVE ID。

26.4.2. 自动恢复

当一个进程试图读取或修改已释放的文件时，它们将被被自动恢复。相关 I/O 将被阻塞直到文件恢复完成。这些操作对进程来说是透明的。例如，以下命令将自动恢复该文件（如果它已被释放）：

```
1 $ cat /mnt/lustre/released_file
```

26.4.3. 请求监控

可以监控每个 MDT 上的已注册请求列表和它们的状况，运行：

```
1 $ lctl get_param -n mdt.lustre-MDT0000.hsm.actions
```

当前复制工具正在处理的请求列表可通过以下命令获取：

```
1 $ lctl get_param -n mdt.lustre-MDT0000.hsm.active_requests
```

26.5. 文件状态

当文件被存档（释放），它们在 Lustre 文件系统上的状态发生改变。使用以下`lfs`命令查看文件状态：

```
1 $ lfs hsm_state FILE1 [FILE2...]
```

可以为每个文件设置以下的特定策略标志：

- `NOARCHIVE`： 该文件永远不会被存档。
- `NORELEASE`： 该文件永远不会被释放。如果已经设置了`RELEASED`标志，则不能再设置此标志。
- `DIRTY`：文件在复制到 HSM 解决方案后发生了更改。`DIRTY` 文件需要再次存档。`DIRTY` 标志只能在已有 `EXIST`标志的情况下设置。

以下选项只能由 root 用户设置：

- `LOST`：该文件已存档，但其在 HSM 解决方案上的副本由于某种原因（如磁盘损坏）丢失，并且不能进行恢复。如果该文件处于 `RELEASE` 状态，则文件丢失；如果不处于 `RELEASE` 状态，则该文件需要再次存档。

有些标志可通过以下命令手动设置或清除：

```
1 $ lfs hsm_set [FLAGS] FILE1 [FILE2...]
2 $ lfs hsm_clear [FLAGS] FILE1 [FILE2...]
```

26.6. 调试

26.6.1. `hsm_controlpolicy`

`hsm_control` 负责控制协调器活动并可以清除动作列表。

```
1 $ lctl set_param mdt.$FSNAME-MDT0000.hsm_control=purge
```

可能的值有：

- enabled：启动协调器线程。在可用复制工具实例上分发请求。
- disabled：暂停协调器活动，将不进行新请求分发，不处理超时。新的请求会被注册，但只有协调器重新启动后才会进行处理。
- shutdown：关闭协调器线程。将无法提交请求。
- purge：清除所有记录的请求。不改变协调器状态。

26.6.2. max_requests

max_requests 是同一时间最大的活动请求数（每个协调器）。该值与代理数量无关。

例如，如果有 2 个 MDT 和 4 个代理，代理不需要处理 2 倍的 max_requests。

```
1 $ lctl set_param mdt.$FSNAME-MDT0000.hsm.max_requests=10
```

26.6.3. policy

更改系统行为，其值可以通过将'+' 或'-' 作为前缀来添加或移除。

```
1 $ lctl set_param mdt.$FSNAME-MDT0000.hsm.policy=+NRA
```

可以是以下情况组合的值:

- NRA：不进行重试。如果恢复失败，不自动重调度请求。
- NBR：不阻塞 I/O 来等待恢复。即触发自动恢复但不阻塞客户端，访问已释放的文件返回 ENODATA。

26.6.4. grace_delay

grace_delay 指的从整个请求列表中清除请求（成功或失败）的延迟，单位为秒。

```
1 $ lctl set_param mdt.$FSNAME-MDT0000.hsm.grace_delay=10
```

26.7. 变更日志

Lustre 文件系统添加了记录 HSM 相关事件的类型为 HSM 的变更日志。

```
1 16HSM   13:49:47.469433938 2013.10.01 0x280 t=[0x200000400:0x1:0x0]
```

有两种可用信息可以写入每条 HSM 记录：变更文件的 FID 和位掩码。位掩码对以下信息进行编码（最低位在前）

- 错误代码（如果存在）(7 bits)
- HSM 事件 (3 bits)

- HE_ARCHIVE = 0：文件已被存档。
- HE_RESTORE = 1：文件已恢复。
- HE_CANCEL = 2: 关于此文件的请求已被取消。
- HE_RELEASE = 3：文件已被释放。
- HE_REMOVE = 4：已删除的请求被自动执行。
- HE_STATE = 5：文件标志已更改。
- HSM 标志 (3 bits)
- CLF_HSM_DIRTY=0x1

在上面的例子中，0x280 标示错误代码为 0，事件为 HE_STATE。

使用 liblustreapi时，可以借助一些辅助函数轻松地从位掩码中提取不同的值，如：hsm_get_cl_event()、hsm_get_cl_flags()、hsm_get_cl_error()。

26.8. 策略引擎

Lustre 文件系统在任何情况下（如空间不足时）都没有内部组件负责自动调度存档请求和发布请求。自动调度存档操作由策略引擎完成。

策略引擎是一个使用 Lustre 文件系统的特定 HSM API 来监视文件系统和调度请求的用户空间程序。

我们建议您在专用客户端上运行策略引擎（类似于代理节点），并使用 Lustre 2.5 以上版本。推荐使用 Robinhood 策略引擎。

26.8.1. Robinhood

Robinhood 是大型文件系统的策略引擎和报告工具。它负责维护数据库中文件系统元数据的副本，以供任意查询。Robinhood 通过定义基于属性的策略，实现了调度文件系统条目的批量行为；通过 Web 界面和命令行工具，为管理员提供了文件系统内容的全面视图。同时，它也为快速的 find 和 du操作提供了增强版的克隆。Robinhood 是一个外部项目，可以用于各种配置。更多信息请参阅：https://sourceforge.net/apps/trac/robinhood/wiki/Doc。（在 **Lustre 2.9** 引入）

第二十七章持久客户端缓存（PCC）

27.1. 简介

基于闪存的固态硬盘有助于（部分地）缩小磁性磁盘和 CPU 之间不断扩大的性能差距。固态硬盘在价格和性能方面都在存储层次中建立了一个新的层次。在 Lustre 中存储的数据集规模很大，在最大的中心中可达到数百个 PiB，这使得将大部分数据存储在 HDD 上，而只将活跃的数据子集存储在 SSD 上更具性价比。

PCC 机制允许配备了内部 SSD 的客户端为具有节点本地 I/O 模式的读和写密集型应用提供额外的性能，而不会失去全球 Lustre 命名空间的优势。PCC 与 Lustre HSM 和布局锁机制相结合，利用本地 SSD 存储提供持久的缓存服务，同时允许在本地和共享存储之间迁移单个文件。这使得 I/O 密集型应用能够在客户端节点上读写数据，而不会失去全局 Lustre 命名空间的优势。

在 Lustre 客户端上使用这种缓存的主要优点是，I/O 堆栈处理缓存的数据来说要简单得多，因为没有来自其他客户端的 I/O 的干扰，从而优化性能。对客户端节点的硬件也没有特殊要求。任何 Linux 文件系统，如 NVMe 设备上的 ext4，都可以作为 PCC 缓存使用。本地文件缓存减少了对象存储目标（OST）的压力，因为小的或随机的 I/O 可以聚集成大的顺序 I/O，临时文件甚至不需要刷新到 OST 上。

27.2. 设计

27.2.1 Lustre 读 -写 PCC 缓存

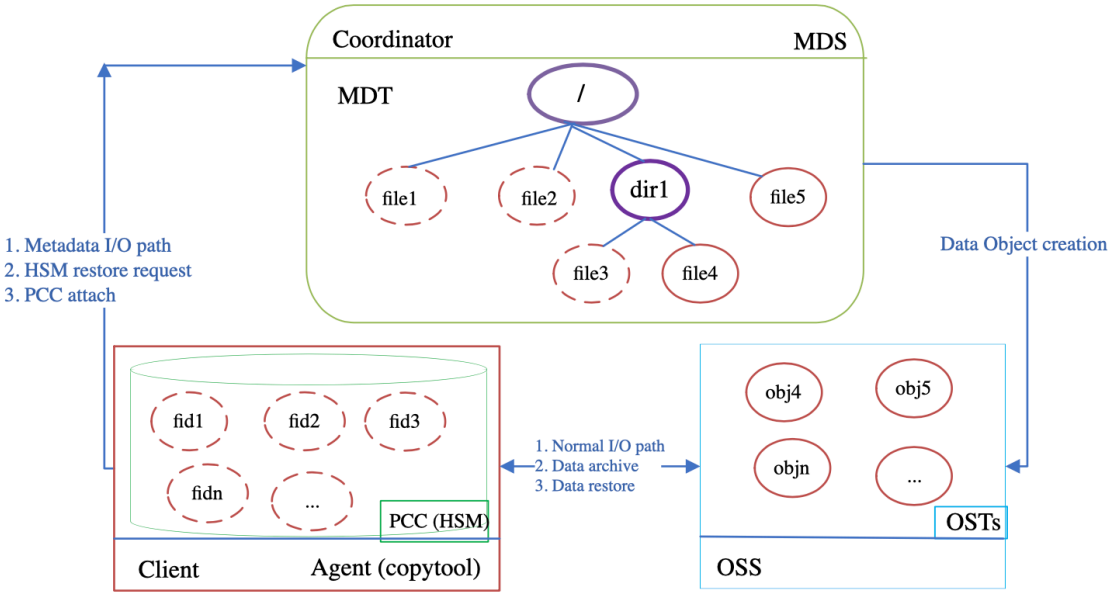


图 27: Overview of PCC-RW Architecture

图 27.1 PCC-RW 架构图

Lustre 通常使用其集成的 HSM 机制，与使用磁带或其他媒体的较大和较慢的归档存储对接。相反，PCC-RW 实际上是一个 HSM 后端存储系统，在 Lustre 客户端提供了一组高速本地缓存。图 27.1，``PCC-RW 架构图" 显示了 PCC-RW 的架构。每个客户端使用自己的本地存储，通常是 NVMe 的形式，格式化为本地缓存的本地文件系统。缓存的 I/O 指向本地文件系统中的文件，而正常的 I/O 则指向 OST。

PCC-RW 使用 Lustre 的 HSM 机制进行数据同步。每个 PCC 节点实际上是一个 HSM 代理，其上运行了 copytool 实例。Lustre HSM copytool 用于将文件从本地缓存恢复到 Lustre OSTs。任何从其他 Lustre 客户端对 PCC 缓存文件的远程访问都会触发这种数据

同步。如果一个 PCC 客户端离线，则其缓存的数据将暂时无法被其他客户端访问。在 PCC 客户端重新启动，然后挂载 Lustre 文件系统，并重新启动 copytool 之后，这些数据才能再次被访问。

目前，PCC 客户端在其本地文件系统上缓存整个文件。在将 I/O 引导到客户端缓存之前，文件必须被附加到 PCC 上。Lustre 的布局锁功能用于确保缓存服务与全局文件系统状态一致。在附加操作成功后，文件数据可以直接写入/读出本地 PCC 缓存。如果附加操作没有成功，客户端将直接退回到正常的 I/O 路径，并将 I/O 直接发送到 OST 上。当其他客户端的进程试图读取或修改 PCC-RW 缓存的文件时，这些文件会自动恢复至全局文件系统。相应的 I/O 将阻塞，等待释放的文件恢复。这对应用程序是透明的。

撤销布局锁可以在任何时候自动将文件从 PCC 缓存中分离出来。PCC-RW 缓存的文件可以通过`lfs pcc detach`命令手动分离。缓存文件从缓存中分离并恢复到 OST 后，它将从 PCC 文件系统中删除。

失败的 PCC-RW 操作通常会返回相应的错误代码。但存在一种特殊情况，即本地 PCC 文件系统的空间已经用完。这种情况下，PCC-RW 可以自动回到正常的 I/O 路径，因为 Lustre 文件系统的容量远远大于 PCC 设备的容量。

27.2.2. 基于规则的持久性客户端缓存

PCC 包括一个基于规则的、可配置的缓存基础架构，使其能够实现各种目标，比如定制 I/O 缓存，提供性能隔离和 QoS 保证。

对于 PCC-RW 来说，当一个文件被创建时，将根据基于规则的策略决定是否会缓存该文件。其支持针对不同用户、组、项目或文件名扩展的规则。

基于规则的 PCC-RW 对新创建的文件的缓存可以直接决定哪个文件可以使用 PCC 上的缓存，而不需要管理员的干预。

27.3. PCC 命令行工具

Lustre 为用户提供了 lfs 和 lctl 命令行工具来与 PCC 功能进行交互。

27.3.1. 在客户端添加 PCC 后端

命令：

```
1 client# lctl pcc add mountpoint pccpath [--param|-p cfgparam]
```

上面的命令将为 Lustre 客户端添加一个 PCC 后端。

参数	说明
mountpoint | Lust	re 客户端的挂载点。

参数	说明
pccpath| 本地文件系	统中用于 PCC 缓存的目录路径。整个文件系统不需要专门用于 PCC 缓存，但该目录不应该被普通用户访问。
cfgparam	一个键 -值对形式的字符串，用于配置 PCC 后端，如读写 attach ID（存档 ID）和自动缓存规则等。

注意：当客户节点有一个以上的 Lustre 挂载点或 Lustre 文件系统实例时，参数`mountpoint`确保只会配置指定的 Lustre 文件系统实例或 Lustre 挂载点的 PCC 后端。如果 PCC 后端被用作 PCC-RW 缓存，这个 Lustre 挂载点必须与 HSM（lhsmtool_posix）配置相同。另外，参数`pccpath`应该与 POSIX copytool（lhsmtool_posix）的 HSM 根参数相同。

PCC-RW 使用 Lustre 的 HSM 机制进行数据同步。在客户端上使用 PCC-RW 之前，仍然需要在 MDT 和 PCC 客户端节点上设置 HSM。

首先，必须在每个文件系统 MDT 上激活一个协调器。这可以通过以下命令实现:

```
1 mds# lctl set_param mdt.$FSNAME-MDT0000.hsm_control=enabled
2 mdt.lustre-MDT0000.hsm_control=enabled
```

接下来，在每个代理节点（PCC 客户节点）上启动 copytool，连接到你的 HSM 存储。这个命令将是以下形式:

```
1 client# lhsmtool_posix --daemon --HSM-root $PCCPATH --archive=$ARCHIVE_ID
    $LUSTREPATH
```

示例:

下面的命令为客户端添加了一个 PCC 后端:

```
1 client# lctl pcc add /mnt/lustre /mnt/pcc --param
    "projid={500,1000}&fname={*.h5}, uid=1001 rwid=2"
```

配置参数的第一个子串是自动缓存规则，其中 ``&" 代表逻辑 AND 运算符，而 ``," 代表逻辑 OR 运算符。这个例子的规则是新文件只有在满足以下任一条件的情况下才会被自动缓存: - 项目 ID 是 500 或 1000，且文件名的后缀是 ``h5"; - 用户 ID 是 1001。

目前支持的 PCC 后端配置的键 -值对列举如下: - `rwid` PCC-RW attach ID，与该 PCC 节点上运行的 copytool 代理的存档 ID 相同。- `auto_attach "auto_attach=1 "`在下一次打开或 I/O 时启用自动附加。启用这个选项将导致自动附加有效的 PCC 缓存文

件，这些文件将会由于手动lfs pcc detach命令或撤销布局锁（即 LRU 锁收缩）而分离。"auto_attach=0 "意味着自动附加文件被禁用，是默认模式。

27.3.2. 从客户端删除一个 PCC 后端

命令：

```
lctl pcc del <mountpoint> <pccpath>
```

上述命令将从 Lustre 客户端删除一个 PCC 后端。

+----------------+----------------------------+

+----------------+----------------------------+ | 参 数 | 说 明 | +==================+==========================+ |mountpoint | Lustre 客户端的挂载点。| +------------+----------------------------+ |pccpath| 一个 PCC 后端是由这个路径指定的。| | | 详情请参考 lctl pcc add。| **示例：** 下面的命令将删除挂载点为 ``/mnt/lustre" 的客户端上由 ``/mnt/pcc" 引用的 PCC 后端。

```
client# lctl pcc del /mnt/lustre /mnt/pcc
```

27.3.3. 移除客户端上的所有 PCC 后端

命令：

```
lctl pcc clear <mountpoint>
```

上述命令将从 Lustre 客户端删除所有 PCC 后端。

+----------------+----------------------------+ | 参 数 | 说 明 | +==================+==========================+ |mountpoint | Lustre 客户端的挂载点。| +------------+----------------------------+ **示例：** 下面的命令将删除挂载点为 ``/mnt/lustre" 的客户端上所有的 PCC 后端。

```
client# lctl pcc clear /mnt/lustre
```

27.3.4. 列出客户端上的所有 PCC 后端

命令：

```
lctl pcc list <mountpoint>
```

上述命令将列出 Lustre 客户端上所有 PCC 后端。

+----------------+----------------------------+ | 参 数 | 说 明 | +==================+==========================+ |mountpoint | Lustre 客户端的挂载点。| +------------+----------------------------+ **示例：** 下面的命令将列出挂载点为 ``/mnt/lustre" 的客户端上所有的 PCC 后端。

```
client# lctl pcc list /mnt/lustre
```

27.3.5. 将指定文件附加到 PCC 上

命令：

```
lfs pcc attach --id|-i <NUM> <file...>
```

上述命令将把给定的文件附加到 PCC 上。

+----------------+----------------------------+ | 参 数 | 说 明 | +================ +==============================+ |--id|-i <NUM> | Attach ID 用于选择使用哪个 PCC 后端 | +----------------+----------------------------+ **示例：** 下面的命令将把/mnt/lustre/ test 引用的文件附加到 PCC-RW attachID 为 2 的 PCC 后端上。

```
client# lfs pcc attach -i 2 /mnt/lustre/test
```

27.3.6. 通过 FID 将指定文件附加到 PCC 上

命令：

```
lfs pcc attach_fid --id|-i <NUM> --mnt|-m <mountpoint> <fid...>
```

上述命令将把由 FID 指定的文件附加到 PCC 上。

+----------------+----------------------------+ | 参 数 | 说 明 | +================ +==============================+ |--id|-i <NUM> | Attach ID 用于选择使用哪个 PCC 后端 | +----------------+----------------------------+ |--mnt|-m <mountpoint> | Lustre 客户端的挂载点。| +----------------+----------------------------+ **示例：** 以下命令将把 FID 为 0x200000401:0x1:0x0 的文件附加到 PCC-RW attach ID 为 2 的 PCC 后端上。

```
client# lfs pcc attach_fid -i 2 -m /mnt/lustre 0x200000401:0x1:0x0
```

27.3.7. 将指定文件从 PCC 分离

命令：

```
lfs pcc detach [--keep|-k] <file...>
```

上述命令将从 PCC 中分离指定的文件。

+----------------+----------------------------+ | 参 数 | 说 明 | +================ +==============================+ |--keep|-k | 默认情况下，detach 命令会将文件 ||| 从 PCC 中永久分离出来，并在分离 ||| 后删除 PCC 副本。||| 这个选项只分离文件，||| 但在缓存中保留 PCC 副本。||| 如果缓存中的文件副本仍然有效，||| 则允许在下次打开时自附加 ||| 被分离的文件。| +----------------+----------------------------+ **示例：** 下面的命令将删除挂载点为 ``/mnt/lustre" 的客户端上所有的 PCC 后端。

```
client# lctl pcc clear /mnt/lustre
```

27.3.8. 通过 FID 从 PCC 分离指定文件

命令：

```
lfs pcc detach_fid [--keep|-k] <mountpoint> <fid...>
```

上述命令将通过 FID 从 PCC 分离指定文件。

+----------------+----------------------------+ | 参 数 | 说 明 | +================
+==============================+ |--keep|-k | 详情请参考 lfs pcc detach 命令 | +----
------------+----------------------------+ **示例：** 下面的命令将把 FID 为 0x200000401:0x1:0x0 的文件从 PCC 上永久分离，并删除 PCC 上相应的缓存文件。

```
client# lfs pcc detach_fid /mnt/lustre 0x200000401:0x1:0x0
```

以下命令将把 FID 为 0x200000401:0x1:0x0 的文件从 PCC 中分离出来，但允许在下次打开时自动连接该文件。

```
client# lfs pcc detach_fid -k /mnt/lustre 0x200000401:0x1:0x0
```

27.3.9. 显示指定文件的 PCC 状态

命令：

```
lfs pcc state <file...>
```

上述命令将显示指定文件的 PCC 状态。

示例： 以下命令将显示/mnt/lustre/test 所引用的文件的 PCC 状态。

```
client# lfs pcc state /mnt/lustre/test
file: /mnt/lustre/test, type: readwrite, PCC file: /mnt/pcc/0004/0000/0bd1
```

如果文件 ``/mnt/lustre/test" 没有缓存在 PCC 上，其 PCC 状态的输出如下。

```
client# lfs pcc state /mnt/lustre/test
file: /mnt/lustre/test, type: none
```

27.4. PCC 配置示例

1. 设置 MDT 上的 HSM

```
mds# lctl set_param mdt.lustre-MDT0000.hsm_control=enabled
```

2. 设置客户端上的 PCC

```
client1# lhsmtool_posix --daemon --hsm-root /mnt/pcc --archive=1
    /mnt/lustre < /dev/null > /tmp/copytool_log 2>&1
client1# lctl pcc add /mnt/lustre /mnt/pcc "projid={1000},uid={500} rwid=1"
client2# lhsmtool_posix --daemon --hsm-root /mnt/pcc --archive=2
    /mnt/lustre < /dev/null > /tmp/copytool_log 2>&1
client2# lctl pcc add /mnt/lustre /mnt/pcc "projid={1000}&gid={500} rwid
```

3. 在客户端上执行 PCC 命令

```
client1# echo "QQQQQ" > /mnt/lustre/test
client2# lfs pcc attach -i 2 /mnt/lustre/test
client2# lfs pcc state /mnt/lustre/test
file: /mnt/lustre/test, type: readwrite, PCC file:
    /mnt/pcc/0004/0000/0bd1/0000/0002/0000/0x200000bd1:0x4:0x0, user
    number: 1, flags: 6
client2# lfs pcc detach /mnt/lustre/test
```

第二十八章使用 Nodemap 映射 UIDs 和 GIDs

28.1. 设置映射

Lustre 2.9 很好地支持了 nodemap 功能，此前，该技术作为预热在 Lustre 2.7 中被首次引入。它允许来自远程系统的 UID 和 GID 映射到本地 UID 和 GID 集，并同时保留 POSIX 的所有权、其他权限和配额信息。因此，即使来自多个站点的用户和组标识符相互冲突，它们仍可以在单个 Lustre 文件系统上运行，而不用担心在 UID 或 GID 空间中产生冲突。

28.1.1. 定义

启用 nodemap 功能后，客户端文件系统对 Lustre 系统的访问将通过 nodemap 标识映射策略引擎进行过滤。Lustre 连接由网络标识符（NID）管理，例如`192.168.7.121@tcp`。当通过 NID 进行操作时，Lustre 将决定该 NID 是否为 nodemap（由一个或多个 NID 范围组成的策略组）的一部分。如果该 NID 不存在于任何策略组中，则默认情况下此访问会被压缩到用户`nobody`。策略组具有一些属性，如 `trusted` 和 `admin`，这些属性决定了访问条件。每个策略组有一组标识映射（idmaps），这些 idmaps 决定了客户端上的 UID 和 GID 如何转换到本地 Lustre 文件系统的规范用户空间上。

为了使 nodemap 正常运行，MGS、MDS 和 OSS 系统必须都运行支持 nodemap 的 Lustre 版本。nodemap 对客户端来说是透明的，不需要特别的配置或相关设置。

28.1.2. NID 范围

NID 可以被描述为单个地址或一个地址范围。单个地址为标准 Lustre NID 格式（如`10.10.6.120@tcp`），地址范围由一个短横来分隔来描述（如`192.168.20.[0-255]@tcp`）。

范围必须是连续的。一个 nidlist 的完整 LNet 定义如下：

```
<nidlist>       :== <nidrange> [ ' ' <nidrange> ]
<nidrange>      :== <addrrange> '@' <net>
<addrrange>     :== '*' |
                        <ipaddr_range> |
                        <numaddr_range>
<ipaddr_range>  :==
        <numaddr_range>.<numaddr_range>.<numaddr_range>.<numaddr_range>
<numaddr_range> :== <number> |
                        <expr_list>
<expr_list>     :== '[' <range_expr> [ ',' <range_expr>] ']'
<range_expr>    :== <number> |
                        <number> '-' <number> |
                        <number> '-' <number> '/' <number>
<net>           :== <netname> | <netname><number>
<netname>       :== "lo" | "tcp" | "o2ib" | "gni"
<number>        :== <nonnegative decimal> | <hexadecimal>
```

28.1.3. 示例：描述和部署映射

部署 nodemap 时，首先应考虑哪些用户需要映射，以及涉及的网络地址或地址范围。必须检查用户之间的可见性问题。

例如，假设研究人员正在研究鸟类相关数据，他们使用了一个计算系统，该系统从单个 IPv4 地址`192.168.0.100`挂载 Lustre 并将此策略组命名为`BirdResearchSite`。该 IP 地址组成了其 NID，即`192.168.0.100@tcp`。运行以下`lctl`命令创建策略组并将此 NID 添加到 MGS 上的相应组：

```
mgs# lctl nodemap_add BirdResearchSite
mgs# lctl nodemap_add_range --name BirdResearchSite --range
   192.168.0.100@tcp
```

注意

一个 NID 不能位于多个策略组中。如果要把 NID 分配给新的策略组，请先将其从现有组中删除。

研究人员在其主机系统上使用以下标识符：

- `swan` (UID 530) — `wetlands` (GID 600) 组成员
- `duck` (UID 531) —`wetlands` (GID 600) 组成员
- `hawk` (UID 532) —`raptor` (GID 601) 组成员
- `merlin` (UID 533) — `raptor` (GID 601) 组成员

为此策略组分配六个 idmaps，其中四个用于 UID，两个用于 GID。选择一个起点，例如 UID 11000，预留空间以便添加额外的 UID 和 GID。使用`lctl`命令设置 idmaps：

```
1 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype uid --idmap
    530:11000
2 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype uid --idmap
    531:11001
3 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype uid --idmap
    532:11002
4 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype uid --idmap
    533:11003
5 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype gid --idmap
    600:11000
6 mgs# lctl nodemap_add_idmap --name BirdResearchSite --idtype gid --idmap
    601:11001
```

参数`530:11000`将客户端 UID（530）映射到单个的规范 UID（11000）。每个映射都是单独进行的，且没有方法可以指定范围`530-533:11000-11003`。UID 和 GID 标识分开进行映射，两者之间没有暗含的关系。

在 NID `192.168.0.100@tcp` 的 Lustre 文件系统上使用 UID `duck` 和 GID `wetlands` 创建的文件将使用规范标识符存储在 Lustre 文件系统中（在本例中为 UID 11001 和 GID 11000）。不同的 NID，如果不同属一个策略组，将看到同一文件空间的各自的视图。

假设先前创建的项目目录由 UID 11002/GID 11001 所有，模式为 770。当位于 192.168.0.100 处的用户`hawk`和`merlin`将名为`hawk-file`和`merlin-file`的文件放入该目录时，来自 192.168.0.100 客户端的内容显示为：

```
1 [merlin@192.168.0.100 projectsite]$ ls -la
2 total 34520
```

```
3 drwxrwx--- 2 hawk   raptor     4096 Jul 23 09:06 .
4 drwxr-xr-x 3 nobody nobody     4096 Jul 23 09:02 ..
5 -rw-r--r-- 1 hawk   raptor 10240000 Jul 23 09:05 hawk-file
6 -rw-r--r-- 1 merlin raptor 25100288 Jul 23 09:06 merlin-file
```

在特权视图中，显示了规范标识符：

```
1 [root@trustedSite projectsite]# ls -la
2 total 34520
3 drwxrwx--- 2 11002 11001     4096 Jul 23 09:06 .
4 drwxr-xr-x 3 root root     4096 Jul 23 09:02 ..
5 -rw-r--r-- 1 11002 11001 10240000 Jul 23 09:05 hawk-file
6 -rw-r--r-- 1 11003 11001 25100288 Jul 23 09:06 merlin-file
```

如果在 Lustre MDS 或 MGS 上不存在 UID 11002 或 GID 11001，请在 LDAP 或其他数据源上创建它们，或通过把将`identity_upcall` 设置为 `NONE`使客户端可信。

通过遍历上面的`lctl`命令可以构建更大、更复杂的配置。简单来说，步骤如下：

1. 命名策略组。
2. 创建策略组的一组 NID 范围。
3. 定义哪些 UID 和 GID 转换需要发生。

28.2. 属性变更

特权用户访问映射系统的权限取决于特定属性。默认情况下，root 访问将被压缩至`nobody`用户，将对大多数管理操作造成影响。

28.2.1. 管理属性

这些属性可以改变客户端行为，默认情况下为关闭状态：`admin`、`trusted`、`squash_uid`、`squash_gid` 和 `deny_unknown`。

- `trusted`属性允许策略组的成员查看文件系统的规范标识符。在上面的例子中，UID 11002 和 GID 11001 在不进行转换的的情况下仍然可见。当本地 UID 和 GID 集已经直接映射到指定用户时，可以使用此功能。
- `admin` 属性定义 root 是否在策略组中被压缩。默认情况下 root 被压缩，除非启用此属性。结合 `trusted`属性将允许备份节点、传输节点或其他管理挂载节点的非映射访问。
- `deny_unknown` 属性拒绝访问所有未映射到特定 nodemap 的用户。如果您希望拒绝未映射的用户访问文件系统以满足安全要求，请使用该属性。

- `squash_uid` 和 `squash_gid` 定义了未映射用户被默认压缩到的 UID 和 GID。如果使用了`deny_unknown`标志，所有访问都将被拒绝。

在 MGS 更改其值，1 代表" 真"，2 代表" 假"。

```
mgs# lctl nodemap_modify --name BirdAdminSite --property trusted --value 1
mgs# lctl nodemap_modify --name BirdAdminSite --property admin --value 1
mgs# lctl nodemap_modify --name BirdAdminSite --property deny_unknown
    --value 1
```

如果策略组处于活动状态，请在系统停机期间更改值，从而尽量减少出现所有权或权限问题的可能性。虽然可以进行实时更改，但由于更改发布前有几秒的前置时间，客户端进行数据缓存时可能会影响更改。

28.2.2. 混合属性

同时设置 `admin`和 `trusted`时，策略组具有 Lustre 文件系统的完全访问权限（就像关闭了 nodemap 一样）。Lustre 文件系统的管理站点至少需要一个具有两个属性的组来执行维护或管理任务。

注意

MDS 系统**必须**位于同时具有这两个属性的策略组中。建议将 MDS 放入标记为"TrustedSystems" 的策略组或在标识中明确此关联。

如果策略组设置了 `admin` 属性但没有设置 `trusted` 属性，则 root 将直接映射到 root，任何显式指定的 UID 和 GID idmaps 将被允许，而其他访问会被压缩。如果 root 用户将所有权更改为本地主机已知但不属于 idmap 的 UID 或 GID，则 root 会将这些文件的所有权有效地更改为默认的压缩 UID 和 GID。

如果设置了 `trusted` 属性但没有设置 `admin` 属性，则策略组可以完全访问 Lustre 文件系统的规范 UID 和 GID 集，且 root 被压缩。

一旦启用`deny_unknown`属性，未映射的用户访问文件系统将被拒绝。如果未设置 `admin` 属性且 root 不是任何映射的一部分，root 访问也会被拒绝。

修改 nodemap 时，更改事件将排队并在整个集群中分布。在正常情况下，这些更改大约需要十秒的传播时间。在此期间，文件访问可能使用旧的 nodemap 设置，也可能使用新的 nodemap 设置。因此，建议为此维护窗口保存更改或在映射节点没有在文件系统中进行写操作时部署变更。

28.3. 启用 nodemap

启用 nodemap 功能非常简单：

```
mgs# lctl nodemap_activate 1
```

相反，传递参数 0 将再次禁用该功能。在部署该功能之后，请在允许客户端挂载文件系统之前验证映射是否完整.

(在 **Lustre 2.8** 中引入)

至此，变更已在 MGS 上生效。在 Lustre 2.9 之前，还必须在 MDS 系统上手动更改设置。另外，如果执行了配额，则必须使用`lctl set_param`（不是`lctl`）将更改手动部署到 OSS 服务器，在 2.9 之前，该配置并非永久生效，需要在每次 Lustre 重启后都使用脚本生成映射。请参照以下示例在 OSS 上部署设置:

```
oss# lctl set_param nodemap.add_nodemap=SiteName
oss# lctl set_param nodemap.add_nodemap_range='SiteName 192.168.0.15@tcp'
oss# lctl set_param nodemap.add_nodemap_idmap='SiteName uid 510:1700'
oss# lctl set_param nodemap.add_nodemap_idmap='SiteName gid 612:1702'
```

在 Lustre 2.9 及更高版本中，nodemap 配置保存在 MGS 中，并自动分发到 MGS、MDS 和 OSS 节点，正常情况下这一过程在大约需要 10 秒钟。

28.4. `default` Nodemap

有一个特殊的 nodemap 叫`default`。顾名思义，它是默认创建的，且不能删除。它就像一个后备 nodemap，为 Lustre 客户端设置与任何其他 nodemap 都不匹配的行为。

由于其特殊的角色，只能在`default` nodemap 上设置一些参数:

- `admin`
- `trusted`
- `squash_uid`
- `squash_gid`
- `fileset`
- `audit_mode`

在默认节点映射上不能定义任何 UID/GID 映射。

注意

更改`default`nodemap 的`admin`和`trusted`属性时要小心，尤其是当您的 Lustre 服务器属于此 nodemap 时。

28.5. 校验设置

使用`lctl nodemap_info all`可列出现有的 nodemap 配置，并可导出。该命令相当于通过`/proc`接口访问 nodemap 的快捷方式。在 Lustre MGS 的`/proc/fs/luster/nodemap/`中，如果 nodemap 在系统上处于活动状态，则`active`包含"1"。在每个策略组中创建一个包含以下参数的目录:

- admin 和 trusted 在设置了值的情况下包含"1"，否则为"0"。
- idmap 包含策略组的 idmaps 列表。ranges 包含策略组的 NID 列表。
- squash_uid 和 squash_gid 哪些 UID 和 GID 用户被压缩（必要的话）。

在 BirdResearchSite 的例子中，预计结果为：

```
mgs# lctl get_param nodemap.BirdResearchSite.idmap

 [
  { idtype: uid, client_id: 530, fs_id: 11000 },
  { idtype: uid, client_id: 531, fs_id: 11001 },
  { idtype: uid, client_id: 532, fs_id: 11002 },
  { idtype: uid, client_id: 533, fs_id: 11003 },
  { idtype: gid, client_id: 600, fs_id: 11000 },
  { idtype: gid, client_id: 601, fs_id: 11001 }
 ]

 mgs# lctl get_param nodemap.BirdResearchSite.ranges
 [
  { id: 11, start_nid: 192.168.0.100@tcp, end_nid: 192.168.0.100@tcp }
 ]
```

28.6. 确保一致性

当 Luster 客户端从未知的 NID 范围进行挂载、添加了不属于已知映射的新的 UID 和 GID、规则中存在错误配置时，启用 nodemap 可能会出现一致性问题。在生产系统上激活 nodemap 时请注意以下事项：

- 可在生产系统上创建新的策略组或 idmaps，但为避免元数据问题，请保留一个维护窗口来更改 trusted 属性。
- 执行管理任务，请使用设置了 trusted 和 admin 属性的策略组访问 Lustre 文件系统。这可以防止创建孤立文件和压缩文件。在没有同时授予 trusted 属性的情况下授予admin属性很危险，客户端上的 root 用户可能知道没有在任何 idmap 中出现的 UID 和 GID。如果 root 用户将所有权更改为这些标识符，那么所有权将被压缩。例如，提取 tar 文件可能从预期的 UID（如 UID 500）变为nobody（通常为 UID 99）。
- 要将两个或更多站点上的不同 UID 映射到 Lustre 文件系统上的单个 UID 或 GID，请创建相互有重叠的 idmaps 并将每个站点放置在其自己的策略组中。每个不同的

UID 到目标 UID 或 GID 可能有不同的映射。

- 在 Lustre 2.8 中，必须手动将更改保存在脚本文件中以便 Lustre 重载后再重新应用。由于在节点之间没有自动同步机制，这些更改必须在每个 OSS，MDS 和 MGS 节点上都部署。
- 如果`deny_unknown`有效，未映射的用户可能会看到映射用户才能看到的条目（由于客户端进行了缓存），但无法查看任何文件内容。
- 使用`lctl nodemap_info`可查看 nodemap 激活状态。如果您希望进行额外的验证并确保生产系统上的有效部署，一种方法是创建已知文件的指纹，将特定 UIDs 和 GIDs 映射到测试客户端。完成 Lustre 系统维护后联机，则测试客户端可以在挂载到用户空间之前验证 UID 和 GID 是否正确地映射。（在 **Lustre 2.9** 中引入）

第二十九章配置共享密钥 (SSK)

29.1. SSK 安全概述

SSK 功能保护了 Lustre PtlRPC 流量数据，确保了其数据完整性。共享属性和会话特定属性集成到 SSK 密钥文件中，分发给 Lustre 主机。Lustre 主机依据授权挂载文件系统，并根据不同的安全特性配置而启用不同的安全模式进行数据传输。管理员负责 SSK 密钥文件的生成，分发和安装，请参见本章第 3.1 节 ``密钥文件管理"。

29.1.1. 关键功能

SSK 提供了以下关键功能：

- 基于主机的认证
- 数据传输隐私性
- Lustre RPCs 加密
- 防窃听
- 数据传输完整性：密钥哈希消息认证码 (HMAC)
- 防止中间人攻击
- 确保 RPCs 不会遭到未检测的更改

29.2. SSK 安全特性

SSK 以一种 General Security Services（GSS）机制的形式，通过 Lustre 支持的 GSS 应用程序接口（GSSAPI）来实现。SSK GSS 机制支持五种不同级别的保护：

- `skn` - SSK Null （仅身份认证）
- `ska` - 用于非批量 RPC 的 SSK 身份和完整性验证

- `ski` - SSK 身份和完整性验证
- `skpi` - SSK 身份、隐私和完整性验证
- `gssnull` -无保护，仅作测试用

下表描述了每种特性的安全性质。

Table 1. SSK 安全保护

	skn	ska	ski	skpi
挂载文件系统时需验证	是	是	是	是
提供 RPC 完整性	否	是	是	是
提供 RPC 隐私性	否	否	否	是
提供批量 RPC 完整性	否	否	是	是
提供批量 RPC 隐私性	否	否	否	是

有效的非 GSS 特性包括：

`null` - 无保护，为默认值。

`plain` - 在每个 RPC 上使用哈希列表的明文。

29.2.1. RPC 安全规则

使用`lctl`命令将 RPC 安全配置规则写入 Lustre 日志（llog)。规则通过 llog 进行处理，它规定了用于特定 Lustre 网络或方向的安全特性。

注意

规则只需几秒钟即可生效，将影响现有连接和新建连接。

规则格式：*target*.srpc.flavor.*network*[.*direction*]=*flavor*

- *target* - 可为文件系统名或特定 MDT/OST 设备名。
- *network* - RPC 启动程序的 LNet 网络名。如：`tcp1` 或 `o2ib0`。如没有指定特定网络，该值也可为关键字`default`，以指代所有网络。
- *direction* - 可选。可为 `mdt2mdt`、`mdt2ost`、`cli2mdt`或 `cli2ost`中的一个。

注意

要确保与 MGS 的安全连接，请使用`mgssec = flavor` 的挂载选项。这是必需的，因为发起方在 MGS 连接建立之前不知道安全规则。

以下示例适用于名为`testfs`的测试用 Lustre 文件系统。

29.2.1.1. 定义规则 规则可以按任何顺序定义和删除。对于给定连接，采用描述最具体的规则。`fsname.srpc.flavor.default`规则限定的范围最广，因为它适用于文件系统内所有非 MGS 的连接。您可以根据您的需求定制 SSK 安全特性，进一步指定特定目标、网络或方向。

以下示例给出了为三个 LNet 网络组成的环境配置 SSK 安全性的方法。需求为:

- 所有非 MGS 连接都必须经过认证。
- LNet 网络`tcp0`上的 PtlRPC 流量必须加密。
- LNet 网络`tcp1`和`o2ib0`是高性能的本地物理安全网络，位于其上的 PtlRPC 流量不需要加密。

1. 确保所有非 MGS 连接在默认情况下都经过身份验证和加密。

```
mgs# lctl conf_param testfs.srpc.flavor.default=skpi
```

2. 在 LNet 网络`tcp1`和`o2ib0`上使用安全特性`ska`覆盖文件系统默认的安全特性。`ska` 提供了身份验证，但没有提供加密和批量 RPC 完整性。

```
1 mgs# lctl conf_param testfs.srpc.flavor.tcp1=ska
2 mgs# lctl conf_param testfs.srpc.flavor.o2ib0=ska
```

注意

目前，"`lctl set_param -P`" 格式和 sptlrpc 不兼容。

29.2.1.2. 列出规则 查看 RPC 安全配置规则，请输入:

```
1 mgs# lctl get_param mgs.*.live.testfs
2 ...
3 Secure RPC Config Rules:
4 testfs.srpc.flavor.tcp.cli2mdt=skpi
5 testfs.srpc.flavor.tcp.cli2ost=skpi
6 testfs.srpc.flavor.o2ib=ski
7 ...
```

29.2.1.3. 删除规则 使用`conf_param -d` 命令删除某 LNet 网络的安全特性:

例如，删除 `testfs.srpc.flavor.o2ib1=ski`规则，输入:

```
1 mgs# lctl conf_param -d testfs.srpc.flavor.o2ib1
```

29.3. SSK 密钥文件

SSK 密钥文件是一组属性的集合，由管理员分发给各客户端和服务器节点。这些属性被格式化为固定长度值并存储在文件中，它们包括：

- **Version** - 密钥文件模式版本号。非用户定义。
- **Type** - 表示密钥文件使用者的 Lustre 角色，为强制属性。有效的密钥类型有：
- **mgs** - MGS，当使用 `mgssec` 和`mount.lustre` 选项时。
- **server** - MDS 或 OSS 服务器。
- **client** - 客户端及在客户端环境中与其他服务器进行通信的服务器（如与 OST 通信的 MDS）。
- **HMAC algorithm** - 用于完整性的密钥哈希消息认证代码算法。有效的算法有（默认为 SHA256）：
- SHA256
- SHA512
- **Cryptographic algorithm** -用于加密的密码算法。有效的算法有（默认为 AES-256-CTR）。
- AES-256-CTR
- **Session security context expiration** - 由密钥生成的会话环境到密钥过期须重新生成的秒数（默认值：604800 秒，即 7 天）。
- **Shared key length** - 共享密钥长度（以位为单位，默认值：256）。
- **Prime length** - 用于 Diffie-Hellman 密钥交换（DHKE）的素数（p）长度（以位为单位，默认值：2048）。仅用于生成客户端密钥，并可能需要一段时间。此值同时也是服务器和 MGS 从客户端接受的最小素数长度。尝试用长度小于此最小值进行连接的客户端将被拒绝。通过这种方式，服务器可以保证最低加密级别。
- **File system name** - Lustre 文件系统名。
- **MGS NIDs** - 由逗号间隔的 MGS NID 列表。只有当使用了`mgssec`时才是必要的（默认值：""）。
- **Nodemap name** - Nodemap 名称（默认值：default）。
- **Shared key** - 被所有 SSK 特性共享的共享密钥，提供身份认证。
- **Prime (p)** - 用于 Diffie-Hellman 密钥交换（DHKE）的素数。仅用于类型为`Type=client`的密钥。

注意

密钥文件提供了验证 Lustre 连接的方法，请安全地存储和传输密钥文件。密钥文件不能全局写入，否则将导致无法加载。

29.3.1. 密钥文件管理

`lgss_sk` 功能用于读、写、更改 SSK 密钥文件。`lgss_sk`可以用来将密钥文件单独加载到内核密钥环中。`lgss_sk`选项包括：

Table 2. lgss_sk 参数

参数	值	说明
`-l|--load`	*filename*	将文件中的密钥安装到用户的会话密钥环中。必须由 root 执行。
`-m|--modify`	*filename*	更改文件的密钥属性
`-r|--read`	*filename*	显示文件的密钥属性
`-w|--write`	*filename*	生成文件密钥
`-c|--crypt`	*cipher*	加密算法（默认：AES 计数器模式）；AES-256-CTR。
`-i|--hmac`	*hash*	用于完整性的哈希算法（默认：SHA256）；SHA256 或 SHA512。
`-e|--expire`	*seconds*	由密钥登入的会话过期的秒数（默认：604800，即 7 天）
`-f|--fsname`	*name*	文件系统名
`-g|--mgsnids`	*NID(s)*	由逗号间隔的 MGS NID 列表。只有当使用了`mgssec`时才是必要的（默认值：""）。
`-n|--nodemap`	*map*	Nodemap 名称（默认值：default）。
`-p|--prime-bits`	*length*	用于 DHKE 的素数 (p) 长度（以位为单位，默认值：2048）。
`-t|--type`	*type*	密钥类型（mgs, server, client）
`-k|--key-bits`	*length*	共享密钥长度（以位为单位，默认值：256）。
`-d|--data`	*file*	共享密钥随机数据源（默认：/dev/random）
`-v|--verbose`		包含错误信息的详细版信息

28.3.1.1. 写入密钥文件 密钥文件由`lgss_sk`工具生成，通过在命令行后附加`--write`参数和要写入的文件名来指定参数。`lgss_sk`工具不会覆盖文件，因此文件名必须是唯一的。`--type`是生成密钥文件的强制参数，`--fsname`，`--mgsnids`和`--write`等其他参数都是可选的。

lgss_sk使用/dev/random作为默认的熵数据源，可使用--data参数覆盖它。当执行lgss_sk的系统上没有硬件随机数生成器时，您可能需要按键盘上的键或移动鼠标（如果直接连接到系统），以便为共享密钥生成熵；如果系统是远程的，可能将导致磁盘 IO。可以使用/dev/urandom进行测试，但这可能会在某些情况下降低安全性。

例如，要在 *biology* nodemap 中为客户端的 testfs Lustre 文件系统创建 *server* 类型密钥文件，请输入：

```
server# lgss_sk -t server -f testfs -n biology \
-w testfs.server.biology.key
```

28.3.1.2. 修改密钥文件 像写入密钥文件一样，您可以通过在命令行上指定要更改的参数来修改它们。只有与指定的参数相关联的密钥文件属性会发生更改，所有其他属性保持不变。

修改客户端密钥文件的 *type* 属性，并填充 *Prime (p)* 密钥属性（如果缺失的话），请输入：

```
client# lgss_sk -t client -m testfs.client.biology.key
```

在服务器密钥文件testfs.server.biology.key和客户端密钥文件testfs.client.biology.key中添加 MGS NIDs 192.168.1.101@tcp,10.10.0.101@o2ib：

```
server# lgss_sk -g 192.168.1.101@tcp,10.10.0.101@o2ib \
-m testfs.server.biology.key

client# lgss_sk -g 192.168.1.101@tcp,10.10.0.101@o2ib \
-m testfs.client.biology.key
```

在 MGS 上修改testfs.server.biology.key以支持 *biology* 客户端到 MGS 的连接，更改密钥文件的 *Type* 属性，在 *server* 的基础上添加 *mgs*：

```
mgs# lgss_sk -t mgs,server -m testfs.server.biology.key
```

28.3.1.3. 读取密钥文件 使用 lgss_sk 工具和 --read 参数读取密钥文件。下面的例子中，我们读取了上面修改过的密钥文件：

```
mgs# lgss_sk -r testfs.server.biology.key
Version:        1
Type:           mgs server
HMAC alg:       SHA256
Crypt alg:      AES-256-CTR
Ctx Expiration: 604800 seconds
```

```
    Shared keylen:  256 bits
    Prime length:   2048 bits
    File system:    testfs
    MGS NIDs:       192.168.1.101@tcp 10.10.0.101@o2ib
    Nodemap name:   biology
    Shared key:
      0000: 84d2 561f 37b0 4a58 de62 8387 217d c30a  ..V.7.JX.b..!}..
      0010: 1caa d39c b89f ee6c 2885 92e7 0765 c917  .......l(....e..

    client# lgss_sk -r testfs.client.biology.key
    Version:        1
    Type:           client
    HMAC alg:       SHA256
    Crypt alg:      AES-256-CTR
    Ctx Expiration: 604800 seconds
    Shared keylen:  256 bits
    Prime length:   2048 bits
    File system:    testfs
    MGS NIDs:       192.168.1.101@tcp 10.10.0.101@o2ib
    Nodemap name:   biology
    Shared key:
      0000: 84d2 561f 37b0 4a58 de62 8387 217d c30a  ..V.7.JX.b..!}..
      0010: 1caa d39c b89f ee6c 2885 92e7 0765 c917  .......l(....e..
    Prime (p) :
      0000: 8870 c3e3 09a5 7091 ae03 f877 f064 c7b5  .p....p....w.d..
      0010: 14d9 bc54 75f8 80d3 22f9 2640 0215 6404  ...Tu...".&@..d.
      0020: 1c53 ba84 1267 bea2 fb05 37a4 ed2d 5d90  .S...g....7..-].
      0030: 84e3 1a67 67f0 47c7 0c68 5635 f50e 9cf0  ...gg.G..hV5....
      0040: e622 6f53 2627 6af6 9598 eeed 6290 9b1e  ."oS&'j.....b...
      0050: 2ec5 df04 884a ea12 9f24 cadc e4b6 e91d  .....J...$......
      0060: 362f a239 0a6d 0141 b5e0 5c56 9145 6237  6/.9.m.A..\V.Eb7
      0070: 59ed 3463 90d7 1cbe 28d5 a15d 30f7 528b  Y.4c....(..]0.R.
      0080: 76a3 2557 e585 a1be c741 2a81 0af0 2181  v.%W.....A*...!.
      0090: 93cc a17a 7e27 6128 5ebd e0a4 3335 db63  ...z~'a(^...35.c
      00a0: c086 8d0d 89c1 c203 3298 2336 59d8 d7e7  ........2.#6Y...
      00b0: e52a b00c 088f 71c3 5109 ef14 3910 fcf6  .*....q.Q...9...
```

```
00c0: 0fa0 7db7 4637 bb95 75f4 eb59 b0cd 4077  ..}.F7..u..Y..@w
00d0: 8f6a 2ebd f815 a9eb 1b77 c197 5100 84c0  .j.......w..Q...
00e0: 3dc0 d75d 40b3 6be5 a843 751a b09c 1b20  =..]@.k..Cu....
00f0: 8126 4817 e657 b004 06b6 86fb 0e08 6a53  .&H..W........jS
```

28.3.1.4. 载入密钥文件 将密钥文件加载到内核密钥环中，可使用lgss_sk工具，也可在挂载时使用skpath挂载选项。skpath方法的优点是它将接受一个目录路径并将目录中的所有密钥文件都加载到密钥环中。而lgss_sk工具在每次调用时将单个密钥文件加载到密钥环中。密钥文件不能全局写入，否则将无法加载。

如果必要的话，也可以使用第三方工具加载密钥。唯一需要注意的是，当request_key向用户空间回调时，密钥必须可用并使用正确的密钥描述，以便在回调期间找到它（请参阅密钥描述）。

例如，使用 lgss_sk载入 testfs.server.biology.key 密钥文件：

```
server# lgss_sk -l testfs.server.biology.key
```

在挂载存储目标时，使用 skpath 挂载选项载入在 /secure_directory 目录下的所有密钥文件，请输入：

```
server# mount -t lustre -o skpath=/secure_directory \
/storage/target /mount/point
```

在客户端上使用 skpath 挂载选项将密钥文件载入密钥环：

```
client# mount -t lustre -o skpath=/secure_directory \
mgsnode:/testfs /mnt/testfs
```

29.4. Lustre GSS 密钥环

Lustre GSS 密钥环二进制文件lgss_keyring被 SSK 用来处理 request-key从内核空间向用户空间回调的操作。lgss_keyring的目的是创建一个令牌，作为安全环境初始化 RPC（SEC_CTX_INIT）的一部分进行传递。

29.4.1. 设置

Lustre GSS 密钥环类特性利用 Linux 内核密钥环基础结构来维护密钥、执行从内核空间到用户空间的回调以完成密钥的协商或建立。当加载 Lustre ptlrpc_gss内核模块时，GSS 密钥环将创建一个名为lgssc的密钥类型。当必须建立安全环境时，它会创建一个密钥并使用回调中的 request-key二进制文件来建立密钥。该密钥将在/etc/request-key.d中查找名称为 *keytype.conf* 形式的配置文件，对于 Lustre 来说该配置文件为lgssc.conf。

SSK 安全涉及的每个节点都必须有/etc/request-key.d/lgssc.conf文件，且文件中包含以下语句：

```
create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u
%g %T %P %S
```

request-key 二进制将调用 lgss_keyring ，请使用相应的值代入随后的参数。

29.4.2. 服务器设置

Lustre 服务器不像客户端那样使用 Linux request-key 机制，而是运行守护进程。该守护进程使用 pipefs 来触发基于文件描述符读写操作的事件。服务器端的二进制文件是lsvcgssd ，它可以在前台或作为守护进程执行。以下是lsvcgssd的参数，它需要明确启用各种安全特性（gssnull，krb5，sk）。这将确保仅启用所需的功能。

Table 3. lsvcgssd 参数

参数	说明
-f	在前台运行
-n	不建立 Kerberos 凭证
-v	详细版
-m	MDS 服务器
-o	OSS 服务器
-g	MGS 服务器
-k	启用 Kerberos
-s	启用共享密钥
-z	启用 gssnull

安装 SysV 样式的初始化脚本来启动和停止lsvcgssd守护进程。初始化脚本将检查/etc/sysconfig/lsvcgss配置文件中的LSVCGSSARGS变量用作启动参数。

通过内核密钥环中的每个密钥的特定描述查找客户端的回调期间以及服务器处理 RPC 期间的密钥。

每个 MGS NID 必须加载一个单独的密钥。密钥描述的格式如下表所示：

Table 4. 密钥描述

类型	密钥描述	示例
MGC	lustre:MGC*NID*	lustre:MGC192.168.1.10@tcp

类型	密钥描述	示例
MDC/OSC/OSP/LWP	lustre:*fsname*	`lustre:testfs`
MDT	lustre:*fsname*:*NodemapName*	`lustre:testfs:biology`
OST	lustre:*fsname*:*NodemapName*	`lustre:testfs:biology`
MGS	lustre:MGS	`lustre:MGS`

Lustre 的所有密钥都使用 `user` 的密钥类型，并被附加到用户的密钥环中。这是不可配置的。以下示例显示了如何列出用户的密钥环、加载密钥文件、读取密钥，以及从内核密钥环中清除密钥。

```
client# keyctl show
Session Keyring
  17053352 --alswrv      0     0  keyring: _ses
 773000099 --alswrv      0 65534   \_ keyring: _uid.0

client# lgss_sk -l /secure_directory/testfs.client.key

client# keyctl show
Session Keyring
  17053352 --alswrv      0     0  keyring: _ses
 773000099 --alswrv      0 65534   \_ keyring: _uid.0
1028795127 --alswrv      0     0       \_ user: lustre:testfs

client# keyctl pipe 1028795127 | lgss_sk -r -
Version:        1
Type:           client
HMAC alg:       SHA256
Crypt alg:      AES-256-CTR
Ctx Expiration: 604800 seconds
Shared keylen:  256 bits
Prime length:   2048 bits
File system:    testfs
MGS NIDs:
Nodemap name:   default
```

```
Shared key:
  0000: faaf 85da 93d0 6ffc f38c a5c6 f3a6 0408  ......o.........
  0010: 1e94 9b69 cf82 d0b9 880b f173 c3ea 787a  ...i.......s..xz
Prime (p) :
  0000: 9c12 ed95 7b9d 275a 229e 8083 9280 94a0  ....{.'Z".......
  0010: 8593 16b2 a537 aa6f 8b16 5210 3dd5 4c0c  .....7.o..R.=.L.
  0020: 6fae 2729 fcea 4979 9435 f989 5b6e 1b8a  o.')..Iy.5..[n..
  0030: 5039 8db2 3a23 31f0 540c 33cb 3b8e 6136  P9..:#1.T.3.;.a6
  0040: ac18 1eba f79f c8dd 883d b4d2 056c 0501  .........=...l..
  0050: ac17 a4ab 9027 4930 1d19 7850 2401 7ac4  .....'I0..xP$.z.
  0060: 92b4 2151 8837 ba23 94cf 22af 72b3 e567  ..!Q.7.#..".r..g
  0070: 30eb 0cd4 3525 8128 b0ff 935d 0ba3 0fc0  0...5%.(...]....
  0080: 9afa 5da7 0329 3ce9 e636 8a7d c782 6203  ..]..)<..6.}..b.
  0090: bb88 012e 61e7 5594 4512 4e37 e01d bdfc  ....a.U.E.N7....
  00a0: cb1d 6bd2 6159 4c3a 1f4f 1167 0e26 9e5e  ..k.aYL:.O.g.&.^
  00b0: 3cdc 4a93 63f6 24b1 e0f1 ed77 930b 9490  <.J.c.$....w....
  00c0: 25ef 4718 bff5 033e 11ba e769 4969 8a73  %.G....>...iIi.s
  00d0: 9f5f b7bb 9fa0 7671 79a4 0d28 8a80 1ea1  ._....vqy..(....
  00e0: a4df 98d6 e20e fe10 8190 5680 0d95 7c83  ..........V...|.
  00f0: 6e21 abb3 a303 ff55 0aa8 ad89 b8bf 7723  n!.....U......w#

client# keyctl clear @u

client# keyctl show
Session Keyring
  17053352 --alswrv      0     0  keyring: _ses
 773000099 --alswrv      0 65534   \_ keyring: _uid.0
```

29.4.3. 调试 GSS 密钥环

Lustre 客户端和服务器支持不同的调试级别，如下所示：

- 0 - 显示错误
- 1 - 显示警告
- 2 - 更多信息
- 3 - 调试
- 4 - 追踪

在客户端上设置调试级别，设置如下 Lustre 参数：

`sptlrpc.gss.lgss_keyring.debug_level`

将调试级别设为追踪：

```
client# lctl set_param sptlrpc.gss.lgss_keyring.debug_level=4
```

通过向守护程序的命令行参数添加" 详细" 标志（-v）来增加服务器端调试显示信息的详细程度。在前台运行`lsvcgssd`守护进程，并设置 (-v) 以支持详细版的 gssnull 和 SSK：

```
server# lsvcgssd -f -vvv -z -s
```

lgss_keyring 是作为 request-key upcall 的一部分被调用的，它没有标准的输出，因此日志记录是通过 syslog 记录的。使用`lsvcgssd`记录服务器端日志，在前台执行时将写入标准输出，在守护进程模式下将写入系统日志 syslog。

29.4.4. 撤销密钥

上面讨论的使用`lgss_sk`和`skpath`挂载选项的密钥并不会被撤销。它们仅用于为客户端连接创建有效的环境，以下两种方式的任一种将使它们失效。

- 从服务器上的用户密钥环中卸载密钥会导致新的客户端连接失败。如果不再需要它可以删除。
- 更改服务器上客户端的 nodemap 名称。由于 nodemap 是共享密钥环境实例的一个组成部分，重命名一组 NID 所属的 nodemap 会阻止创建新的密钥环境。

目前还没有从 Lustre 上清除密钥环境的机制。可以从服务器上卸载目标来进行清除；也可以在创建密钥时使用更短的密钥环境时限，以便环境（比默认）更频繁地刷新。具体设置的超时时间取决于用例，3600 秒是一个合理值，即每隔一小时必须重新协商密钥环境。

29.5. Nodemap 在 SSK 中的作用

SSK 使用 Nodemap（请查看第 28 章，通过 Nodemap 映射 UID 和 GID）策略组名称及其关联的 NID 范围，作为防止密钥文件伪造的一种机制，从而控制某密钥文件所能使用的 NID 范围。

客户端假定自己处在正在使用的密钥文件中指定的 nodemap 中。当客户端初始化安全上下文时，会触发一个回调，回调中包含了上下文信息。利用这些上下文信息，`request-key` 将调用`lgss_keyring`，并反过来为 MGC 查找密钥，查找的描述符为 lustre:fsname 或 lustre:target_name。使用在用户密钥环找到的与描述相匹配的密钥，从密钥读取 nodemap 名称，用 SHA256 算法计算哈希值并发送到服务器。

服务器查找客户端的 NID 以确定它与哪个 nodemap 关联，将 nodemap 名称发送给`lsvcgssd`。`lsvcgssd`守护程序将验证 HMAC 是否等于客户端发送的 nodemap 值。当客户端的 NID 与服务器上定义的 nodemap 名称没有关联时，密钥无效，从而防止了密钥文件伪造。

SSK 执行客户端 NID 到 nodemap 的名称查找无需激活 nodemap 功能。

29.6. SSK 示例

这一小节的例子中，我们用了 1 个 MGS/MDS (NID 172.16.0.1@tcp)，1 个 OSS (NID 172.16.0.3@tcp) 和 2 个客户端。Lustre 文件系统名为 *testfs*。

29.6.1. 客户端到服务器的安全通信

本示例阐述了如何配置 SSK，将隐私和完整性保护应用于 tcp 网络上的客户端到服务器的 PtlRPC 流量。我们使用规则指定了方向，在这里为`cli2mdtand`和`cli2ost`。该配置不提供服务器到服务器的保护，请参阅第"6.3. 确保服务器到服务器的通信安全"。服务器到服务器通信的安全特性在这里为 null（所有 Lustre 连接的默认特征）。

1. 创建存储 SSK 密钥文件的安全目录。

```
mds# mkdir /secure_directory
mds# chmod 600 /secure_directory
oss# mkdir /secure_directory
oss# chmod 600 /secure_directory
cli1# mkdir /secure_directory
cli1# chmod 600 /secure_directory
cli2# mkdir /secure_directory
cli2# chmod 600 /secure_directory
```

2. 为 MDS 和 OSS 服务器生产密钥文件，运行：

```
mds# lgss_sk -t server -f testfs -w \
/secure_directory/testfs.server.key
```

3. 将密钥文件`/secure_directory/testfs.server.key`可靠地拷贝到 OSS 上。

```
mds# scp /secure_directory/testfs.server.key \
    oss:/secure_directory/
```

4. 将密钥文件`/secure_directory/testfs.server.key`可靠地拷贝到 *client1* 上的`/secure_directory/testfs.client.key`中。

```
mds# scp /secure_directory/testfs.server.key \
    client1:/secure_directory/testfs.client.key
```

5. 在 *client1* 上将密钥文件类型改为client 。此操作同时也生成了长度为 Prime length 的素数以填充 Prime (p) 属性，运行:

```
client1# lgss_sk -t client \
    -m /secure_directory/testfs.client.key
```

6. 在 所 有 含'create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u %g %T %P %S' (没 有 单 引 号) 的 节 点 上 创 建 /etc/request-key.d/lgssc.conf 文件:

```
mds# echo create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u %g %T
    %P %S > /etc/request-key.d/lgssc.conf
    oss# echo create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u %g
        %T %P %S > /etc/request-key.d/lgssc.conf
    client1# echo create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u
        %g %T %P %S > /etc/request-key.d/lgssc.conf
    client2# echo create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u
        %g %T %P %S > /etc/request-key.d/lgssc.conf
```

7. 在 MDS 和 OSS 上 配 置 lsvcgss 守 护 程 序。 在 MDS 上 的 /etc/sysconfig/lsvcgss 中，将LSVCGSSDARGS 变量的值设置为'-s -m'。在 OSS 上的/etc/sysconfig/lsvcgss中，将 LSVCGSSDARGS 变量的值设为 '-s -o'。

8. 在 MDS 和 OSS 上启动 lsvcgssd 守护程序，运行:

```
mds# systemctl start lsvcgss.service oss# systemctl start
lsvcgss.service
```

9. 使 用-o skpath=/ secure_directory挂 载 选 项 挂 载 MDT 和 OST。skpath选项将目录中找到的所有 SSK 密钥文件加载到内核密钥环中。

10. 将客户端到 MDT 连接、客户端到 OST 连接的安全特性设置为保障 SSK 隐私性和完整性，即skpi:

```
mds# lctl conf_param testfs.srpc.flavor.tcp.cli2mdt=skpi
mds# lctl conf_param testfs.srpc.flavor.tcp.cli2ost=skpi
```

11. 在 client1 和 client2 上挂载文件系统 *testfs*。

```
client1# mount -t lustre -o skpath=/secure_directory 172.16.0.1@tcp:/testfs
    /mnt/testfs
client2# mount -t lustre -o skpath=/secure_directory 172.16.0.1@tcp:/testfs
    /mnt/testfs
mount.lustre: mount 172.16.0.1@tcp:/testfs at /mnt/testfs failed:
    Connection refused
```

12. client2 未能通过身份验证，因为它没有有效的密钥文件。请重复步骤 4 和 5，将 client1 替换为 client2，然后再将文件系统 *testfs* 安装到 client2 上：

```
client2# mount -t lustre -o skpath=/secure_directory 172.16.0.1@tcp:/testfs
    /mnt/testfs
```

13. 确认 mdc 和osc连接使用了 SSK 机制，rpc和bulk 安全特性为 skpi。请参阅"7. 查看 PtlRPC 安全环境"。

    请注意，mgc到 MGS 的连接没有可靠的 PtlRPC 安全环境。这是因为我们在步骤 10 中仅为客户端到 MDT 和客户端到 OST 连接指定了skpi安全特性。以下示例详细说明了保护客户端到 MGS 的连接所需步骤。

29.6.2. MGS 安全通信

此示例建立在之前示例的基础上。

1. 在 MGS 上启用 lsvcgss MGS 服务。在 MGS 上编辑/etc/sysconfig/lsvcgss，将参数 (-g) 添加到变量 LSVCGSSDARGS上。重启 lsvcgss 服务。

2. 在 MDS 上将 mgs 密钥类型和 MGS NIDs 添加到密钥文件 /secure_directory/testfs.server.key 中。

```
```

mgs# lgss_sk -t mgs,server -g 172.16.0.1@tcp,172.16.0.2@tcp -m /secure_directory/testfs.server.key ```

3. 在 MGS 载入更改后的密钥文件，运行：

```
mgs# lgss_sk -l /secure_directory/testfs.server.key
```

4. 在客户端 client1 上将 MGS NIDs 添加至密钥文件 /secure_directory/testfs.client.key中。

```
client1# lgss_sk -g 172.16.0.1@tcp,172.16.0.2@tcp -m
    /secure_directory/testfs.client.key
```

5. 在 client1 上卸载文件系统 *testfs*，用挂载选项 `mgssec=skpi` 重新挂载文件系统：

```
cli1# mount -t lustre -o mgssec=skpi,skpath=/secure_directory
    172.16.0.1@tcp:/testfs /mnt/testfs
```

6. 确认 client1 的 MGC 连接使用了 SSK 机制及`skpi` 安全特性。请参阅"7. 查看 PtlRPC 安全环境"。

29.6.3. 服务器之间的安全通信

此示例阐述了如何配置 SSK，实现在 tcp 网络上 MDT 到 OST 的 PtlRPC 流量的完整性保护和 `ski` 安全特性。

此示例建立在之前示例的基础上。

1. 为 Lustre 文件系统在 MGS 上创建名为 `LustreServers` 的 nodemap 策略组，输入：

```
mgs# lctl nodemap_add LustreServers
```

2. 将 MDS 和 OSS NIDs 添加到 nodemap `LustreServers`中：

```
mgs# lctl nodemap_add_range --name LustreServers --range
172.16.0.[1-3]@tcp
```

3. 为`LustreServers` nodemap 中所有节点创建类型为 `mgs,server` 的密钥文件。

```
```
mds# lgss_sk -t mgs,server -f testfs -g \
172.16.0.1@tcp,172.16.0.2@tcp -n LustreServers -w \
/secure_directory/testfs.LustreServers.key
```
```

4. 将密钥文件 `/secure_directory/testfs.LustreServers.key` 可靠地拷贝至 OSS。

```
```
mds# scp /secure_directory/testfs.LustreServers.key oss:/secure_directory/
```
```

5. 在MDS和OSS上，将密钥文件 /secure_directory/testfs.LustreServers.key可靠地拷贝至/secure_directory/testfs.LustreServers.client.key中。

6. 在每个服务器上将/secure_directory/testfs.LustreServers.client.key的密钥类型更改为client。此操作同时也生成了长度为 Prime length 的素数以填充 Prime (p) 属性，运行：

```
```
mds# lgss_sk -t client -m \
/secure_directory/testfs.LustreServers.client.key
oss# lgss_sk -t client -m \
/secure_directory/testfs.LustreServers.client.key
```
```

7. 将密钥文件/secure_directory/testfs.LustreServers.key和/secure_directory/testfs.LustreServers.client.key 加载至 MDS 和 OSS 上的密钥环上。

```
mds# lgss_sk -l /secure_directory/testfs.LustreServers.key
mds# lgss_sk -l /secure_directory/testfs.LustreServers.client.key
oss# lgss_sk -l /secure_directory/testfs.LustreServers.key
oss# lgss_sk -l /secure_directory/testfs.LustreServers.client.key
```

8. 将 MDT 至 OST 连接的安全特性特质为ski以保证数据完整性：

```
mds# lctl conf_param testfs.srpc.flavor.tcp.mdt2ost=ski
```

9. 确认 osc 和 osp 到 OST 的连接拥有可靠的ski 安全环境。. 请参阅 ``29.7. 查看 PtlRPC 安全环境"。

29.7. 查看 PtlRPC 安全环境

从客户端（或具有 mgc，osc，mdc 环境的服务器）上，您可以导入文件查看所有用户的密钥环境及使用中的安全特性。对于用户环境（srpc_context），SSK 和 gssnull 仅支持单个 root UID，因此只能有一个环境。导入的另一个文件（srpc_info）具有 *sptlrpc* 其它详细信息。rpc和 bulk允许您确认哪些安全特性正在使用中。

```
client1# lctl get_param *.*.srpc_*
mdc.testfs-MDT0000-mdc-ffff8800da9f0800.srpc_contexts=
ffff8800da9600c0: uid 0, ref 2, expire 1478531769(+604695), fl
    uptodate,cached,, seq 7, win 2048, key 27a24430(ref 1), hdl
    0xf2020f47cbffa93d:0xc23f4df4bcfb7be7, mech: sk
```

```
mdc.testfs-MDT0000-mdc-ffff8800da9f0800.srpc_info=
rpc flavor:     skpi
bulk flavor:    skpi
flags:          rootonly,udesc,
id:             3
refcount:       3
nctx:   1
gc internal     3600
gc next 3505
mgc.MGC172.16.0.1@tcp.srpc_contexts=
ffff8800dbb09b40: uid 0, ref 2, expire 1478531769(+604695), fl
    uptodate,cached,, seq 18, win 2048, key 3e3f709f(ref 1), hdl
    0xf2020f47cbffa93b:0xc23f4df4bcfb7be6, mech: sk
mgc.MGC172.16.0.1@tcp.srpc_info=
rpc flavor:     skpi
bulk flavor:    skpi
flags:          -,
id:             2
refcount:       3
nctx:   1
gc internal     3600
gc next 3505
osc.testfs-OST0000-osc-ffff8800da9f0800.srpc_contexts=
ffff8800db9e5600: uid 0, ref 2, expire 1478531770(+604696), fl
    uptodate,cached,, seq 3, win 2048, key 3f7c1d70(ref 1), hdl
    0xf93e61c64b6b415d:0xc23f4df4bcfb7bea, mech: sk
osc.testfs-OST0000-osc-ffff8800da9f0800.srpc_info=
rpc flavor:     skpi
bulk flavor:    skpi
flags:          rootonly,bulk,
id:             6
refcount:       3
nctx:   1
gc internal     3600
gc next 3505`
```

第三十章 Lustre 文件系统安全管理

30.1. 使用访问控制列表（ACL）

访问控制列表（ACL）是用于向操作系统通知每个用户或组对特定系统对象（如目录或文件）的权限（或访问权限）的一组数据。每个对象都有一个唯一的安全属性，用于标识有权访问它的用户。ACL 列出了每个对象和用户的访问权限，如读、写或执行。

30.1.1. ACL 如何工作

在不同的操作系统实现 ACL 有所不同。支持可移植操作系统接口（POSIX）系列标准的系统共享一个简单但功能强大的文件系统权限模型，Linux/UNIX 管理员应该对这个模型非常熟悉。ACL 将为此模型添加更细化的权限，形成更复杂的权限方案。有关 Linux 操作系统上 ACL 的详细说明，请参阅 SUSE Labs 的文章Linux 上 Posix 的访问控制列表。

我们根据这个模型来实施 ACL。Lustre 软件可以与标准的 Linux ACL 工具，setfacl，getfacl 以及历史 chacl（通常随 ACL 软件包一起安装）配合工作。

注意

ACL 是系统范围内的功能，这意味着所有客户端都将启用或不启用 ACL。您无法指定其中一部分客户端启用或不启用 ACL。

30.1.2. Lustre 软件上的 ACLs

POSIX 访问控制列表（ACL）可以与 Lustre 软件一起使用。一个 ACL 由代表权限（基于标准 POSIX 文件系统对象权限）的文件条目组成，这些权限定义了三类用户（所有者，组和其他）。每个类都与一组权限关联：读（r），写（w）和执行（x）。

- 所有者类定义文件所有者的访问权限。
- 组类定义组用户的访问权限。
- 其他类定义不在所有者类或组类的其它用户访问权限。

`ls -l`命令输出的第一列将显示所有者、组和其他类的权限（对于常规文件来说，-rw-r- -表示所有者拥有读取和写入权限，组用户拥有读取权限，其它用户无访问权限）。

最小 ACL 有三个条目，扩展 ACL 有三个以上的条目。扩展 ACL 还包含一个掩码条目，并可能包含任意数量的指定用户和命名组条目。

配置 MDS 来启用 ACL，请在创建配置时使用`--mountfsoptions`:

```
1 $ mkfs.lustre --fsname spfs --mountfsoptions=acl --mdt -mgs /dev/sda
```

您也可以在运行时使用`mkfs.lustre`命令和 `--acl` 选项来启用 ACL:

```
1 $ mount -t lustre -o acl /dev/sda /mnt/mdt
```

在 MDS 上查看 ACL：

```
$ lctl get_param -n mdc.home-MDT0000-mdc-*.connect_flags | grep acl acl
```

挂载不带 ACL 的客户端：

```
$ mount -t lustre -o noacl ibmds2@o2ib:/home /home
```

在 Lustre 文件系统中，ACL 功能在系统范围内启用；要么所有客户端都启用 ACL，要么都不启动。由 MDS 挂载选项acl /noacl（启用/禁用 ACL）控制激活 ACL。客户端挂载时，将忽略acl/noacl选项。您不需要更改客户端配置，且'acl' 字符串不会出现在客户端/etc/mtab 中。如果使用了该选项装载客户端，则在 MDS 系统日志中将出现以下消息提示：

```
...MDS requires ACL support but client does not
```

该消息是无害的，但指示了应予以更正的配置问题。

如果 MDS 上未启用 ACL，则在客户端上进行任何引用 ACL 的尝试都会返回" 不支持操作" 的错误。

30.1.3. 示例

这些示例来自上面引用的 POSIX 文件。Lustre 文件系统上 ACL 的工作机制与任何 Linux 文件系统上的 ACL 完全相同。它们通过标准工具以标准方式进行操作。我们在下面创建了一个目录并允许特定的用户访问。

```
[root@client lustre]# umask 027
[root@client lustre]# mkdir rain
[root@client lustre]# ls -ld rain
drwxr-x---  2 root root 4096 Feb 20 06:50 rain
[root@client lustre]# getfacl rain
# file: rain
# owner: root
# group: root
user::rwx
group::r-x
other::---

[root@client lustre]# setfacl -m user:chirag:rwx rain
[root@client lustre]# ls -ld rain
drwxrwx---+ 2 root root 4096 Feb 20 06:50 rain
[root@client lustre]# getfacl --omit-header rain
```

```
17 user::rwx
18 user:chirag:rwx
19 group::r-x
20 mask::rwx
21 other::---
```

30.2. 使用 Root Squash

Root Squash 是一种安全功能，它限制了超级用户访问 Lustre 文件系统的权限。如果未启用 Root Squash 功能，则未授信任客户端上的 Lustre 文件系统用户可以访问、修改，甚至删除系统 root 用户的文件。使用 Root Squash 功能可以限定能够访问或修改 root 用户文件的客户端。注意，这不会阻止未授信客户端上的用户访问其他用户的文件。

Root Squash 功能通过 Lustre 配置管理服务器（MGS）将 root 用户的用户标识（UID）和组标识（GID）重新映射到由系统管理员指定的 UID 和 GID 来工作。Root Squash 功能同时也允许 Lustre 文件系统管理员指定不适用于 UID/GID 重映射的一组客户端。

注意

Nodemaps（用 Nodemap 映射 UID 和 GID）是 root squash 的一种替代方案，因为它也允许在每个客户端上进行 root squash。通过 UID 映射，客户端甚至可以拥有一个本地的 root UID，而不需要实际拥有对文件系统本身的 root 访问权限。

30.2.1. 配置 Root Squash

Root Squash 由两种配置参数进行管理：`root_squash`，`nosquash_nids`。

- `root_squash` 参数用于指定 root 用户访问 Lustre 文件系统使用的 UID 和 GID。
- `nosquash_nids` 参数用于指定不适用 Root Squash 的一组客户端，使用 LNet NID 范围的语法，如：

```
1 nosquash_nids=172.16.245.[0-255/2]@tcp
```

在此示例中，Root Squash 不适用于子网 172.16.245.0 且 IP 地址最后一部分为偶数的 TCP 客户端。

30.2.2. 启用和调试 Root Squash

`nosquash_nids` 的默认值为 NULL，表明默认情况下 Root Squash 适用于所有客户端。关闭 Root Squash，请将 root squash UID 和 GID 设为 0。

创建 MDT (`mkfs.lustre --mdt`) 时可设置 Root Squash 参数，如：

```
mds# mkfs.lustre --reformat --fsname=testfs --mdt --mgs \
       --param "mdt.root_squash=500:501" \
       --param "mdt.nosquash_nids='0@elan1 192.168.1.[10,11]'" /dev/sda1
```

Root Squash 参数可在未挂载的设备上通过tunefs.lustre更改：

```
tunefs.lustre --param "mdt.root_squash=65534:65534"  \
--param "mdt.nosquash_nids=192.168.0.13@tcp0" /dev/sda1
```

Root Squash 参数也可通过 lctl conf_param 命令更改，如：

```
mgs# lctl conf_param testfs.mdt.root_squash="1000:101"
mgs# lctl conf_param testfs.mdt.nosquash_nids="*@tcp"
```

要检索当前的 root squash 参数设置，可以使用如下lctl get_param命令：

```
mgs# lctl get_param mdt.*.root_squash
mgs# lctl get_param mdt.*.nosquash_nids
```

注意

使用lctl conf_param命令时，请谨记：

- lctl conf_param 必须在活动 MGS 上运行。
- lctl conf_param 将导致所有 MDSs 上的参数发生改变。
- 运行一次lctl conf_param只能更改一个参数。

Root Squash 设置也可以通过 lctl set_param暂时改变，或者通过 lctl set_param -P 永久改变。例如：

```
mgs# lctl set_param mdt.testfs-MDT0000.root_squash="1:0"
mgs# lctl set_param -P mdt.testfs-MDT0000.root_squash="1:0"
```

清除 nosquash_nids 列表：

```
mgs# lctl conf_param testfs.mdt.nosquash_nids="NONE"
```

或：

```
mgs# lctl conf_param testfs.mdt.nosquash_nids="clear"
```

nosquash_nids 包含了一些 NID 范围（如：0@elan, 1@elan1），NID 范围列表必须用单引号 (`) 或双引号 (") 进行引用，每个值用空格分开，如：

```
mds# mkfs.lustre ... --param "mdt.nosquash_nids='0@elan1 1@elan2'" /dev/sda1
lctl conf_param testfs.mdt.nosquash_nids="24@elan 15@elan1"
```

以下是一些语法错误的例子：

```
mds# mkfs.lustre ... --param "mdt.nosquash_nids=0@elan1 1@elan2" /dev/sda1
lctl conf_param testfs.mdt.nosquash_nids=24@elan 15@elan1
```

使用lctl get_param 命令查看 Root Squash 参数：

```
mds# lctl get_param mdt.testfs-MDT0000.root_squash
lctl get_param mdt.*.nosquash_nids
```

注意

nosquash_nids列表为空，将返回 NONE。

30.2.3. 使用 Root Squash 的技巧

在 Lustre 配置管理中，Root Squash 功能在以下几个方面有所限制：

- lctl conf_param 指定的值将覆盖参数先前的值。如果新值使用不正确的语法，那么系统将继续使用旧的参数，但在重新挂载时之前正确的值将丢失。请谨慎调试 Root Squash。
- mkfs.lustre 和 tunefs.lustre 不进行参数语法检查。如果 root squash 参数错误，它们将在挂载时被忽略，系统将使用默认值。
- Root Squash 参数将通过严格的语法检查。root_squash 参数应由 <decnum>:<decnum>指定。nosquash_nids 参数应遵循 LNet NID 范围的语法。

LNet NID 范围的语法：

```
<nidlist>     :== <nidrange> [ ' ' <nidrange> ]
<nidrange>    :== <addrrange> '@' <net>
<addrrange>   :== '*' |
           <ipaddr_range> |
           <numaddr_range>
<ipaddr_range>       :==
<numaddr_range>.<numaddr_range>.<numaddr_range>.<numaddr_range>
<numaddr_range>      :== <number> |
                   <expr_list>
<expr_list>  :== '[' <range_expr> [ ',' <range_expr>] ']'
<range_expr> :== <number> |
           <number> '-' <number> |
           <number> '-' <number> '/' <number>
<net>          :== <netname> | <netname><number>
```

```
<netname>    :== "lo" | "tcp" | "o2ib"
           | "ra" | "elan"
<number>     :== <nonnegative decimal> | <hexadecimal>
```

注意

对于使用数字地址的网络（如 elan），地址范围必须由<numaddr_range> 语法指定。对于使用 IP 地址的网络，地址范围必须由<ipaddr_range>语法指定。例如，如果 elan 使用数字地址，则 1.2.3.4@elan 是错误的。

30.3. 隔离客户端到子目录树上

Isolation（隔离）是通过 Lustre 多租户这一通用概念的实现，其目的在于从一个文件系统中提供分离的命名空间。Lustre Isolation 使同一文件系统上的不同用户群体能够超越正常的 Unix 权限/ACL，即使客户端上的用户可能有 root 访问权限。这些租户共享同一个文件系统，但他们相互之间是隔离的：他们不能访问甚至看不到对方的文件，也不知道他们正在共享共同的文件系统资源。

Lustre Isolation 使用了 Fileset 特性，只挂载文件系统的一个子目录，而不是根目录。为了实现隔离，必须让客户端挂载子目录（只向租户展示自己的 fileset）。为此，我们使用了 nodemap 功能（用 nodemap 映射 UID 和 GID）。我们将一个租户使用的所有客户端归类到一个共同的 nodemap 条目下，并将该租户被限制的 fileset 分配给这个 nodemap 条目。

30.3.1. 指定客户端

在 Lustre 上强制执行多租户，依赖于能正确识别租户使用的客户端节点，并信任这些节点的能力。这可以通过物理硬件和/或网络安全来实现，从而使客户端节点拥有众所周知的 NID。还可以使用 Kerberos 或共享密钥，使用强认证。Kerberos 可以防止 NID 欺骗，因为每个客户端都需要基于其 NID 来连接到服务器。公私密钥还可以防止租户冒充，因为密钥可以链接到特定的 nodemap。

30.3.2. 配置 Isolation

Lustre 上的 Isolation 可以通过在 nodemap 条目上设置 fileset 参数来实现。所有属于这个 nodemap 条目的客户端将自动挂载这个 fileset，而不是挂载 root 目录。例如：

```
mgs# lctl nodemap_set_fileset --name tenant1 --fileset '/dir1'
```

因此，所有匹配tenant1 nodemap 的客户端在挂载时都会自动显示/dir1的文件集合（fileset），表示这些客户端正在对子目录/dir1进行隐式子目录挂载。

注意如果文件系统中不存在定义为 fileset 的子目录，则会阻止任何属于 nodemap 的客户端挂载 Lustre。

要删除 fileset 参数，只需将其设置为空字符串即可：

```
1 mgs# lctl nodemap_set_fileset --name tenant1 --fileset ''
```

30.3.3. 将 Isolation 持久化

为了使 Isolation 持久化，必须使用带选项 `-P`的`lctl set_param` 来设置 nodemap 上的 fileset 参数。

```
1 mgs# lctl set_param nodemap.tenant1.fileset=/dir1
2 mgs# lctl set_param -P nodemap.tenant1.fileset=/dir1
```

这样，fileset 参数将被存储在 Lustre 配置的日志中，供服务器重启后获取该信息。

30.4. 检查 Lustre 客户端执行的 SELinux 策略

SELinux 在 Linux 中提供了一种支持强制访问控制（MAC）策略的机制。当 MAC 策略被强制执行时，操作系统的内核就会定义应用的权限，使应用不会危及整个系统。普通用户没有能力使该策略失效。

SELinux 的一个目的是保护**操作系统**不受权限升级的影响。为此，SELinux 为进程和用户定义了受限域和非受限域。每个进程、用户、文件都被分配了一个安全环境，规则定义了进程和用户对文件允许执行的操作。

SELinux 的另一个目的是保护**数据**的敏感性，这要归功于多级安全（MLS）功能。MLS 是在 SELinux 的基础上，通过定义域之外的安全级别概念发挥作用。每个进程、用户和文件都被分配了一个安全级别，且该模型规定，进程和用户可以读取与自己相同或更低的安全级别的数据，但只能写入与自己相同或更高的安全级别的数据。

从文件系统的角度来看，文件的安全环境必须持久存储。Lustre 利用文件上的`security.selinux`扩展属性来存储这些信息。Lustre 在客户端支持 SELinux。要在 Lustre 上实现 MAC 和 MLS，需要做的就是在所有 Lustre 客户端上执行适当的 SELinux 策略（由 Linux 发行版提供）。Lustre 服务器上不需要 SELinux 策略。

因为 Lustre 是一个分布式文件系统，所以使用 MLS 的特殊性在于，Lustre 确实需要确保数据总是被节点访问，并正确执行 SELinux MLS 策略。否则，数据就无法得到保护。这意味着 Lustre 必须检查 SELinux 是否在客户端正确执行了 SELinux 策略，并且是正确的、未被修改的策略。而如果 SELinux 在客户端没有按预期执行该策略，服务器会拒绝其访问 Lustre。

30.4.1. 确定 SELinux 策略信息

服务器使用一个代表 SELinux 状态信息的字符串作参考，以检查客户端是否正确地执行 SELinux 策略。这个参考字符串可以通过在已知执行正确的 SELinux 策略的客户端节点上调用`l_getsepol`命令行工具获得。

```
client# l_getsepol
SELinux status info:
    1:mls:31:40afb76d077c441b69af58cccaaa2ca63641ed6e21b0a887dc21a684f508b78f
```

描述 SELinux 策略的字符串的语法如下。

```
mode:name:version:hash
```

其中:

- mode 表示一个数字，告诉 SELinux 是在 Permissive 模式 (0) 还是强制模式 (1) 下执行。
- name 表示 SELinux 策略的名称。
- version 表示 SELinux 策略的版本。
- hash 表示计算出的策略的二进制表示的哈希值，从`/etc/selinux/name/policy/policy/policy. version`中导出。

30.4.2. 执行 SELinux 策略检查

可以通过在 nodemap 条目上设置 `sepol` 参数来执行 SELinux 策略检查。所有属于这个 nodemap 条目的客户端必须执行该参数描述的 SELinux 策略，否则将被拒绝访问 Lustre 文件系统。例如:

```
mgs# lctl nodemap_set_sepol --name restricted
     --sepol
         '1:mls:31:40afb76d077c441b69af58cccaaa2ca63641ed6e21b0a887dc21a684f508b78f'
```

因此，所有匹配`restricted` nodemap 的客户端必须执行 SELinux 策略，该策略的描述匹配`1:mls:31:40afb76d077c441b69af58ccca2ca63641ed6e21b0a887dc21a684f508b78f`。如果不匹配，当试图挂载或访问 Lustre 文件系统上的文件时，会得到`Permission Denied`的提示。

要删除`sepol`参数，只需将其设置为空字符串即可。

```
mgs# lctl nodemap_set_sepol --name restricted --sepol ''
```

30.4.3. 持久化 SELinux 策略检查

为了持久化 SELinux 策略检查，必须使用lctl set_param的-P选项来设置 nodemap 上的sepol参数。

```
mgs# lctl set_param
   nodemap.restricted.sepol=1:mls:31:40afb76d077c441b69af58cccaaa2ca63641ed6e21b0a887dc21a68
mgs# lctl set_param -P
   nodemap.restricted.sepol=1:mls:31:40afb76d077c441b69af58cccaaa2ca63641ed6e21b0a887dc21a68
```

这样，sepol参数将被存储在 Lustre 配置日志中，供服务器在重启后获取该信息。

30.4.4. 客户端发送 SELinux 状态信息

为了让 Lustre 客户端能够发送 SELinux 状态信息，在本地启用 SELinux，send_sepol ptlrpc 内核模块的参数必须设置为非零。send_sepol可以设置为以下值：

- 0: 不发送 SELinux 策略信息。
- -1: 每次请求都会获取 SELinux 策略信息。
- N > 0: 每隔 N 秒只获取 SELinux 策略信息。设置 N = 2^31-1 则只在挂载时获取 SELinux 策略信息。

在定义了sepol 的 nodemap 中的客户端必须发送 SELinux 状态信息。而且他们执行的 SELinux 策略必须与存储在 nodemap 中的策略相匹配。否则它们将被拒绝访问 Lustre 文件系统。

30.5 加密文件和目录

对客户端加密是希望能够为每个用户提供一个特殊的目录，安全地存储敏感文件。其目的是保护在客户端和服务器之间传输的数据，并保护静止的数据。

该功能直接在 Lustre 客户端级实现。Lustre 客户端加密依赖于内核fscrypt。fscrypt是一个库，文件系统可通过它支持文件和目录的透明加密。因此，下面描述的关键内容是从fscrypt文档中摘出的。

对于更多完整的细节，请参考 Lustre 源代码中的Documentation/client_side_encryption目录下的文档。

注意 客户端加密功能在运行着 5.4 以上版本的内核的 Linux 发行版的 Lustre 客户端上是可用的。由于 Lustre 提供了一个另外色内核库，因此它也可以在运行支持基本加密的 Linux 发行版的客户端上使用，包括：

- CentOS/RHEL 8.1 及以上版本；
- Ubuntu 18.04 及以上版本；
- SLES 15 SP2 及以上版本。

30.5.1. 客户端加密访问语义

仅有 Lustre 客户端需要访问加密主密钥。密钥添加到 Lustre 客户端的文件系统级加密密钥圈中。

- 有密钥有了加密密钥，加密的普通文件、目录和符号链接的行为与未加密的对应文件非常相似 --毕竟，加密预期是透明的。然而，细心的用户可能会注意到行为上的一些差异。
- 未加密的文件，或用不同的加密策略（即不同的密钥、模式或标志）加密的文件，不能进行重命名或链接到一个加密的目录。但加密文件可以在加密目录中重命名，或链接到未加密的目录中。**注意**

  将一个未加密的文件 ``移动" 到一个加密的目录中，如通过`mv`命令，在用户空间是通过复制和删除来实现的。请注意，原始的未加密数据可能仍然可以从磁盘上的空闲空间中恢复；最好从一开始就保持所有文件加密。
- 在 Lustre 上，加密的文件支持直接 I/O。
- `fallocate()`操作`FALLOC_FL_COLLAPSE_RANGE`，`FALLOC_FL_INSERT_RANGE`和`FALLOC_FL_ZERO_RANGE`不支持加密文件，并且会以 EOPNOTSUPP 失败。
- 加密文件不支持 DAX（直接访问）。
- 支持`mmap`。这是有可能的，因为加密文件的分页缓存包含明文，而不是加密文本。
- 无密钥无密钥时（包括添加加密密钥之前，或删除加密密钥被之后），一些文件系统的操作仍可以在加密的普通文件、目录和符号链接上进行：
- 可读取文件元数据，如使用 stat()。
- 可列出目录，包括整个命名空间树。
- 可删除文件。非目录的文件可以像往常一样使用 unlink() 删除，而空目录可以像往常一样用 rmdir() 删除。因此，rm 和 rm -r 将按预期工作。
- 可读取和跟踪 Symlink 目标，但它们将以加密的形式呈现，类似于目录中的文件名。因此，它们不太可能呈现出所有有用的目标。

无密钥时无法打开或截断普通文件，试图这样做将以 ENOKEY 失败。这意味着任何需要文件描述符的常规文件操作，如 read()、write()、mmap()、fallocate() 和 ioctl()，都被禁止了。

同样，如果没有密钥，则无法在加密目录中创建或链接任何类型的文件（包括目录），加密目录中的名字也不能成为重命名的来源或目标，也不能在加密目录中创建 O_TMPFILE 临时文件。所有这些操作都会因为 ENOKEY 而失败。

目前不可能在没有加密密钥的情况下备份和恢复加密的文件。这需要特殊的 API，目前还没有实现。

- 加密策略的执行在一个目录上设置了加密策略后，在该目录（递归子目录）中创建的所有普通文件、目录和符号链接都将继承该加密策略。但不加密特殊文件------即命名的管道、设备节点和 UNIX 域套接字。

除了这些特殊文件，禁止在加密的目录树中有未加密的文件或使用不同加密策略加密的文件。

30.5.2. 客户端加密密钥的层次结构

每个加密目录树都受到一个主密钥的保护。

要 ``解锁" 一个加密目录树，用户空间必须提供恰当的主密钥。可以有任意数量的主密钥，每个主密钥保护任意数量文件系统上的任意数量的目录树。

30.5.3. 客户端加密模式和用法

`fscrypt`允许为文件内容指定一种加密模式，为文件名指定另一种加密模式。不同的目录树允许使用不同的加密模式。

目前，支持以下几对加密模式：

- 内容为 AES-256-XTS，文件名为 AES-256-CTS-CBC
- 内容为 AES-128-CBC，文件名为 AES-128-CTS-CBC

如果不确定使用哪种模式，则应该使用（AES-256-XTS，AES-256-CTS-CBC）对。

警告 在 Lustre 2.14 中，客户端加密只支持内容加密，而不是文件名加密。因此，只有内容加密模式会被识别并执行，而文件名加密模式会被忽略，从而使文件名成为明文。

30.5.4. 客户端加密威胁模式

- 离线攻击对于 Lustre，块设备是连接到 Lustre 服务器的 Lustre 目标。离线操纵文件系统意味着在 Lustre 离线时访问这些目标上的文件系统。当块设备内容在某一

时间点永久离线的情况下，只要选择了一个强加密密钥，则fscrypt可以护文件内容的机密性。Lustre 客户端加密并不能保护元数据的机密性，如文件名、文件大小、文件权限、文件时间戳和扩展属性。另外，也无法保护文件中的洞（逻辑上包含所有零的未分配块）的存在及位置。

- 在线攻击
- 在 Lustre 客户端

  添加了加密密钥后，fscrypt并不对同一节点上的其他用户隐藏明文文件内容或文件名，而现有的访问控制机制（如文件模式位、POSIX ACLs、LSMs 或命名空间）应提供隐藏功能。

  对于 Lustre，这意味着明文文件内容或文件名不会对同一 Lustre 客户端的其他用户隐藏。

  攻击者如果破坏了系统（如利用内核的安全漏洞），足以从任意内存中读取，则可以破坏当前使用的所有加密密钥。然而，fscrypt允许将加密密钥从内核中移除，从而保护它们不被破坏。非 root 用户也可以进行密钥移除操作。更确切地说，密钥移除将从内核内存中擦除主加密密钥。另外，它还将尝试驱逐所有使用该密钥 ``解锁" 的缓存节点，从而擦除它们的每个文件的密钥，并使它们再次被 ``锁定"，即以密码文本或加密的形式出现。

- 在 Lustre 服务器上

  如果是 Lustre 服务器上的攻击者破坏了系统（如利用内核安全漏洞），虽然足以读取任意内存，但无法获取 Lustre 文件内容。事实上，加密密钥不会被转发给 Lustre 服务器，服务器也不会进行解密或加密。此外, 服务器收到的批量 RPC 包含加密的数据, 被原封不动地写入底层文件系统中。### 30.5.5. 管理目录上的加密默认情况下，Lustre 客户端加密是启用的，让用户在每个目录的基础上定义加密策略。

注意 管理员可以通过指定noencrypt客户端挂载选项来阻止 Lustre 客户端挂载点使用加密功能，且服务端可以通过 nodemap 上的 forbid_encryption属性强制执行这一功能。请参考第 28.2 章节 ``修改属性"，了解如何管理 nodemap。

用户空间工具fscrypt可以用来管理加密策略。参见 https://github.com/google/fscrypt，了解全面的说明。下面是关于如何在 Lustre 中使用这个工具的例子。如果没有特别说明，则必须在 Lustre 客户端运行命令。

- 在实际决定哪些目录要加密之前，需要两个准备步骤，这是唯一需要 root 权限的功能。管理员必须运行：

```
1 # fscrypt setup
```

```
2 Customizing passphrase hashing difficulty for this system...
3 Created global config file at "/etc/fscrypt.conf".
4 Metadata directories created at "/.fscrypt".
```

这第一条命令必须在所有想要使用加密的客户端上运行，因为它在 Lustre 之外设置了全局 fscrypt 参数。

```
1 fscrypt setup /mnt/lustre
2 Metadata directories created at "/mnt/lustre/.fscrypt"
```

这第二条命令必须只在一个 Lustre 客户端上运行。

注意 可以编辑文件/etc/fscrypt.conf。强烈建议将policy_version设置为 2，这样当删除加密密钥时，fscrypt就会从内存中清除文件。

- 现在一个普通用户能够选择一个目录进行加密:

```
$ fscrypt encrypt /mnt/lustre/vault The following
protector sources are available: 1 - Your login passphrase
(pam_passphrase)2 - A custom passphrase (custom_passphrase)3
- A raw 256-bit key (raw_key)Enter the source number for
the new protector [2 - custom_passphrase]: 2 Enter a name
for the new protector: shield Enter custom passphrase for
protector "shield": Confirm passphrase: "/mnt/lustre/vault"
is now encrypted, unlocked, and ready for use.
```

从这条命令之后，在/mnt/lustre/vault下创建的所有文件和目录都将（根据前面步骤中定义的策略）被加密。

注意 加密策略会被所有子目录继承。无法改变一个子目录的策略。

- 另一个用户可以决定用自己的保护器对不同的目录进行加密:

```
$ fscrypt encrypt /mnt/lustre/private Should we create a
new protector? [y/N] Y The following protector sources are
available: 1 - Your login passphrase (pam_passphrase)2 - A
custom passphrase (custom_passphrase)3 - A raw 256-bit key
(raw_key)Enter the source number for the new protector
[2 - custom_passphrase]: 2 Enter a name for the new
protector: armor Enter custom passphrase for protector
"armor": Confirm passphrase: "/mnt/lustre/private" is now
encrypted, unlocked, and ready for use.
```

- 用户可以在任何时候决定锁定一个加密的目录: `$ fscrypt lock /mnt/lustre/vault "/mnt/lustre/vault" is now locked.` 这个动作可以防止对加密内容的访问，或通过从内存中删除密钥。如果该文件还没有打开，也会从内存中擦除。
- 用户可以通过以下命令重新获得对加密目录的访问: `$ fscrypt unlock /mnt/lustre/vault Enter custom passphrase for protector "shield": "/mnt/lustre/vault" is now unlocked and ready for use.`
- 实际上，fscrypt并不直接访问主密钥，而是访问用于加密的保护器。这种机制提供了改变口令的能力: `$ fscrypt status /mnt/lustre lustre filesystem "/mnt/lustre" has 2 protectors and 2 policies PROTECTOR LINKED DESCRIPTION deacab807bf0e788 No custom protector "shield" e691ae7a1990fc2a No custom protector "armor" POLICY UNLOCKED PROTECTORS 52b2b5aff0e59d8e0d58f962e715862e No deacab807bf0e788 374e8944e4294b527e50363d86fc9411 No e691ae7a1990fc2a $ fscrypt metadata change-passphrase --protector=/mnt/lustre:deacab807bf0e788 Enter old custom passphrase for protector "shield": Enter new custom passphrase for protector "shield": Confirm passphrase: Passphrase for protector deacab807bf0e788 successfully changed.` 这可以为同一策略设置多个保护器。当几个用户共享一个加密目录时，这非常有用，因为它避免了在他们之间共享任何秘密。`$ fscrypt status /mnt/lustre/vault "/mnt/lustre/vault" is encrypted with fscrypt. Policy: 52b2b5aff0e59d8e0d58f962e715862e Options: padding:32 contents:AES_256_XTS filenames:AES_256_CTS policy_version:2 Unlocked: No Protected with 1 protector: PROTECTOR LINKED DESCRIPTION deacab807bf0e788 No custom protector "shield" $ fscrypt metadata create protector /mnt/lustre Create new protector on "/mnt/lustre" [Y/n] Y The following protector sources are available: 1 - Your login passphrase (pam_passphrase)2 - A custom passphrase (custom_passphrase)3 - A raw 256-bit key (raw_key)Enter the source number for the new protector [2 - custom_passphrase]: 2 Enter a name for the new protector: bunker Enter custom`

```
passphrase for protector "bunker": Confirm passphrase:
Protector f3cc1b5cf9b8f41c created on filesystem
"/mnt/lustre". $ fscrypt metadata add-protector-to-policy
--protector=/mnt/lustre:f3cc1b5cf9b8f41c
--policy=/mnt/lustre:52b2b5aff0e59d8e0d58f962e715862e
WARNING: All files using this policy will be
accessible with this protector Protect policy
52b2b5aff0e59d8e0d58f962e715862e with protector
f3cc1b5cf9b8f41c? [Y/n] Y Enter custom passphrase for
protector "bunker": Enter custom passphrase for protector
"shield": Protector f3cc1b5cf9b8f41c now protecting
policy 52b2b5aff0e59d8e0d58f962e715862e. $ fscrypt status
/mnt/lustre/vault "/mnt/lustre/vault" is encrypted with
fscrypt. Policy: 52b2b5aff0e59d8e0d58f962e715862e Options:
padding:32 contents:AES_256_XTS filenames:AES_256_CTS
policy_version:2 Unlocked: No Protected with 2 protectors:
PROTECTOR LINKED DESCRIPTION deacab807bf0e788 No custom
protector "shield" f3cc1b5cf9b8f41c No custom protector
"bunker"
```

30.6 配置 Kerberos（KRB）安全性

本章介绍如果在 Lustre 中使用 Kerberos。

30.6.1. 什么是 Kerberos?

Kerberos 是一种验证 ``不安全" 网络上所有实体（如用户和服务器）的机制。每一个实体（也称为 ``委托人"）与 Kerberos 服务器协商一个运行时密钥。这个密钥使委托人能够验证来自 Kerberos 服务器的信息是真实的。通过信任的 Kerberos 服务器，用户和服务可以互相认证。

使用 Kerberos 设置 Lustre 可以为 Lustre 网络提供高级安全保护。广义上讲，Kerberos 提供了三种类型的保护：

- 允许 Lustre 连接对等体（MDS、OSS 和客户端）相互认证。
- 保护 PTLRPC 消息的完整性，防止在网络传输过程中被修改。
- 保护 PTLRPC 消息的隐私，防止在网络传输期间被窃听。

Kerberos 使用 ``内核钥匙圈" 客户端向上调用机制。

30.6.2. 安全类型（Security Flavor）

安全类型是一个字符串，用于描述在 PTLRPC 连接上执行什么样的认证和数据转换，包括 RPC 消息和批量数据。下表描述了支持的类型：

基础类型	认证	RPC 消息保护	批量数据保护	说明
null	N/A	N/A	N/A	
krb5n	GSS/Kerberos5	null	校验和	不对 RPC 消息进行保护，对批量数据进行校验保护，性能开销较小。
krb5a	GSS/Kerberos5	部分完整性保护（krb5）	校验和	对 RPC 消息头部进行完整性保护，对批量数据进行校验保护，与 krb5n 相比，性能开销更大。
krb5i	GSS/Kerberos5	完整性保护（krb5）	完整性保护（krb5）	转化算法由 KDC 和委托人强制执行的实际 Kerberos 算法决定；性能损失很大。
krb5p	GSS/Kerberos5	隐私保护（krb5）	隐私保护（krb5）	转换隐私保护算法是由 KDC 和委托人执行实际 Kerberos 算法决定的；最大的性能损失

30.6.3. Kerberos 设置

30.6.3.1. 版本 Lustre（从 1.3 版本开始）只支持 MIT Kerberos 5。

关于一般的环境要求，尤其是时钟同步，请参考第 8.1.2 章节 ``环境要求"。

30.6.3.2. 原理配置

- 配置客户端节点：
- 对于每个客户节点，创建一个 lustre_root principal 并生成 keytab。

```
1  kadmin> addprinc -randkey lustre_root/client_host.domain@REALM
2 kadmin> ktadd lustre_root/client_host.domain@REALM
```

- 在客户端节点上安装 keytab。
- 配置 MGS 节点：
- 对于每个 MGS 节点，创建一个 lustre_mgs principal 并生成 keytab。

```
kadmin> addprinc -randkey lustre_mgs/mgs_host.domain@REALM
kadmin> ktadd lustre_mds/mgs_host.domain@REALM
```

- 在 MGS 节点上安装 keytab。
- 配置 MDS 节点:
- 对于每个 MDS 节点，创建一个 lustre_mds principal 并生成 keytab。

```
kadmin> addprinc -randkey lustre_mds/mds_host.domain@REALM
kadmin> ktadd lustre_mds/mds_host.domain@REALM
```

- 在 MDS 节点上安装 keytab。
- 配置 OSS 节点:
- 对于每个 OSS 节点，创建一个 lustre_oss principal 并生成 keytab。

```
kadmin> addprinc -randkey lustre_oss/oss_host.domain@REALM
kadmin> ktadd lustre_oss/oss_host.domain@REALM
```

- 在客户机节点上安装 keytab。

  注意

  - `host.domain`应该是您网络中的 FQDN，否则服务器可能无法识别任何 GSS 请求。
  - 作为客户端 keytab 的替代方案，如果你想省去为每个客户端节点分配唯一 keytab 的麻烦，则可以创建一个通用的 lustre_root 委托人和它的 keytab，并在你想要的许多客户端节点上安装相同的 keytab。**请注意，在这种方式下，一个客户端被攻击则意味着所有客户端都不安全**。

  ```
  kadmin> addprinc -randkey lustre_root@REALM kadmin> ktadd
  lustre_root@REALM
  ```

- Lustre 支持 MIT Kerberos 5 1.3 或更高版本的以下 enctypes:
  - aes128-cts
  - aes256-cts

30.6.4. 网络

在一个如 TCP 或 InfiniBand 等可以将名称解析为 IP 地址的网络上，在委托人中使用的名称必须可以解析为 Lustre NID 使用的 IP 地址。

如果您使用的网络不是 TCP 或 InfiniBand（例如 PTL4LND），则每个节点上都需要有一个/ etc/lustre/nid2hostname脚本，其目的是将 NID 翻译成主机名。以下是 PTL4LND 可能会用到的一个示例：

```
#!/bin/bash
set -x
# 用三个参数调用：lnd netid ，可将的转换为主机名，nidLNDNID
# 是字符串$lnd"PTL4LND等等"
# 是十六进制字符串格式的网络标识符$netid
# 是十六进制格式的$nidNID
# 输出相应的主机名，或由'@开头的错误信息，用于错误记录。'
lnd=$1
netid=$2
# 将十六进制号码转换为十进制NID
nid=$((0x$3))
case $lnd in
 PTL4LND) # 只需在开头添加'node即可'
 echo "node$nid"
 ;;
 *)
 echo "@unknown LND: $lnd"
 ;;
esac
```

30.6.5. 必要的软件包

每个节点都应该安装以下软件包：

- krb5-workstation
- krb5-libs
- keyutils
- keyutils-libs

在用于构建支持 GSS 的 Lustre 的节点上，应该安装以下软件包：

- krb5-devel
- keyutils-libs-devel

30.6.6. 构建 Lustre

在配置时启用 GSS：

```
./configure --enable-gss --other-options
```

30.6.7. 运行

30.6.7.1. GSS 守护进程 在启动 Lustre 之前，请确保在每个服务器节点（MGS、MDS 和 OSS）上启动守护进程`lsvcgssd`。命令的语法是:

```
lsvcgssd [-f] [-v] [-g] [-m] [-o] -k
```

- -f: 在前台运行，而非作为守护程序运行
- -v：增加 1 个日志的详细程度。例如，要将详细程度设置为 3，则运行'lsvcgssd -vvv'。详细的日志记录可以帮助你确定 Kerberos 的设置是否正确。
- -g: 服务 MGS
- -m: 服务 MDS
- -o: 服务 OSS
- -k: 启用 Kerberos 支持

30.6.7.2. 设置安全类型 安全类型可以通过在 MGS 上定义 sptlrpc 规则来设置。这些规则是持久性的，其形式为：`<spec>=<flavor>`。- 添加一个规则:

`mgs> lctl conf_param <spec>=<flavor>`

如果`<spec>`上已经存在一个现有的规则，则将被新规则覆盖。

- 删除一个规则：

`- mgs> lctl conf_param -d <spec>`

- 列出现有的规则:

`msg> lctl get_param mgs.MGS.live.<fs-name> | grep "srpc.flavour"`

注意

- 如果没有指定任何东西，默认情况下所有 RPC 连接将使用`null`类型，这意味着没有安全性。

- 在你改变规则之前，确保受影响的节点已经准备好接受新的安全风味。例如，如果你把规则从 null 改为 krb5p，但 GSS/Kerberos 环境没有正确地但在受影响的节点上没有正确配置 GSS/Kerberos 环境，这些节点可能会被驱逐，因为它们不能互相通信。彼此之间的通信。

- 在改变规则后，通常需要几分钟时间将新规则应用到所有节点，时间长短取决于全局系统的负载。

- 在改变规则之前，确保受影响的节点已经准备好接受新的安全类型。例如，如果你把类型从null改为krb5p，但在需要改变的节点上没有正确配置 GSS/Kerberos 环境，那么这些节点可能会被驱逐，因为它们之间无法相互通信。#### 30.6.7.3. 规则的语法和示例一般的语法是 <target>.srpc.flavor.<network>[.<direction>]=flavor

- <target>可以是文件系统名称，或特定的 MDT/OST 设备名称。例如，testfs、testfs-MDT0000、testfs-OST0001。

- <network>是 LNet 网络名称，例如 tcp0、o2ib0，或者为在 LNet 网络默认为default。

- <direction>可以是 cli2mdt, cli2ost, mdt2mdt, mdt2ost 之一。方向是可选的。

示例:

- 在文件系统testfs的所有连接上应用krb5i:

```
1 mgs> lctl conf_param testfs.srpc.flavor.default=krb5i
```

- 网络 tcp0 中的节点使用 krb5p；所有其他节点使用 null。mgs> lctl conf_param testfs.srpc.flavor.tcp0=krb5p mgs> lctl conf_param testfs.srpc.flavor.default=null

- 网络 tcp0 中的节点使用 krb5p；o2ib0 中的节点使用 krb5n；在其他节点中，客户端连接到 MDT/ OST 使用 krb5i，MDT 到其他 MDT 使用 null，MDT 到 OST 使用 krb5a。mgs> lctl conf_param testfs.srpc.flavor.tcp0=krb5p mgs> lctl conf_param testfs.srpc.flavor.o2ib0=krb5n mgs> lctl conf_param testfs.srpc.flavor.default.cli2mdt=krb5i mgs> lctl conf_param testfs.srpc.flavor.default.cli2ost=krb5i mgs> lctl conf_param testfs.srpc.flavor.default.mdt2mdt=null mgs> lctl conf_param testfs.srpc.flavor.default.mdt2ost=krb5a

30.6.7.4. 普通用户认证 在客户端节点上，非 root 用户在访问 Lustre 之前需要发布`kinit`，就像其他 Kerberized 应用一样。

- 按照 Kerberos 的要求，用户的委托人 (username@REALM) 应该添加到 KDC 中。
- 客户端和 MDT 节点应该有相同的用户数据库，用于名称和 uid/gid 转换。

普通用户可以在注销前销毁已建立的安全上下文，方法是执行以下命令：

```
1 lfs flushctx -k -r <mount point>
```

这里`-k`是销毁磁盘上的 Kerberos 凭证缓存，类似于`kdestroy`，而`-r`可在刷新 GSS 上下文时，从密钥环中获取被撤销的密钥。否则，它只销毁了内核内存中已建立的上下文。

30.6.8. 安全的 MGS 连接

每个节点可以通过使用`mgssec=flavor` 挂载选项指定连接到 MGS 的安全类型。一旦选择了一种类型，在重新挂载之前不能改变。

因为一个 Lustre 节点只有一个到 MGS 的连接，如果节点上有一个以上的目标或客户端，则它们必然使用与 MGS 相同的安全模式，即在建立与 MGS 的第一个连接时执行的模式。

默认情况下，MGS 接受任何类型的 RPC。但也可以将 MGS 配置为只接受一个给定类型。其语法与第 30.6.7.3 章节 ``规则语法和示例" 中的说明相同，但使用特殊的目标`_mgs`:

```
1 mgs> lctl conf_param _mgs.srpc.favour.<network>=<flavor>
```

第三十一章 Lustre ZFS 快照

31.1. 概述

快照能够快速从先前创建的检查点恢复文件，而无需借助脱机备份或远程副本。快照还提供了存储的版本控制，用于恢复丢失的文件或之前不同版本的文件。

文件系统快照应挂载在用户可访问的节点上（如登录节点），以便用户在无需管理员干预的情况下恢复文件（在意外删除或覆盖之后）。用户访问时，可以自动挂载快照文件系统而不是挂载所有快照，从而降低登录节点的开销（当快照不在使用中）。

从快照恢复丢失的文件通常比从任何脱机备份或远程副本进行恢复要快得多。请注意，快照并不会提高存储可靠性。与其他任何存储阵列一样，快照无法防御硬件故障。

31.1.1. 需求

所有 Lustre 服务器目标必须是运行 Lustre 2.10 或更高版本的 ZFS 文件系统。此外，MGS 必须能够通过 ssh 或其他远程访问协议与所有服务器进行通信，无需密码验证。

该功能默认为启用状态，且不能禁用。快照的管理通过 MGS 上的`lctl`命令完成。

Lustre 快照基于 Copy-On-Write，快照和文件系统在文件系统上的文件发生更改前可能共享数据的同一副本。直到引用这些文件的快照被删除，存放已删除或已覆盖文件的空间才会被释放。文件系统管理员需要根据系统的实际大小和使用情况建立快照的创建、备份、删除策略。

31.2. 配置

快照工具从 MGS 上的`/etc/ldev.conf`文件载入系统配置，调用相关 ZFS 命令来维护所有目标 (MGS/MDT/OST) 的 Lustre 快照。请注意，`/etc/ldev.conf`文件还有其他用途。

文件的格式为:

```
1 <host> foreign/- <label> <device> [journal-path]/- [raidtab]
```

的格式为:

```
1 fsname-<role><index> or <role><index>
```

的格式为:

```
1 [md|zfs:][pool_dir/]<pool>/<filesystem>
```

快照只使用域、和。

示例如下:

```
1 mgs# cat /etc/ldev.conf
2 host-mdt1 - myfs-MDT0000 zfs:/tmp/myfs-mdt1/mdt1
3 host-mdt2 - myfs-MDT0001 zfs:myfs-mdt2/mdt2
4 host-ost1 - OST0000 zfs:/tmp/myfs-ost1/ost1
5 host-ost2 - OST0001 zfs:myfs-ost2/ost2
```

配置文件是手动编辑的。当配置文件更新为当前最新的文件系统设置，您可以开始创建文件系统快照。

31.3. 快照操作

31.3.1. 创建快照

创建现有 Lustre 文件系统的快照，在 MGS 上运行以下 `lctl` 命令:

```
lctl snapshot_create [-b | --barrier [on | off]] [-c | --comment
comment] -F | --fsname fsname> [-h | --help] -n | --name ssname>
[-r | --rsh remote_shell][-t | --timeout timeout]
```

选项	说明
-b	在创建快照之前设置写屏障。默认值是'on'。
-c	快照目的说明
-F	文件系统名
-h	帮助信息
-n	快照名
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。
-t	写屏障的时间周期。默认值是 30 秒。

31.3.2. 删除快照

删除现有 snapshot，在 MGS 上运行 `lctl` 命令：

```
lctl snapshot_destroy [-f | --force] <-F | --fsname fsname>
<-n | --name ssname> [-r | --rsh remote_shell]
```

选项	**说明**
-f	暴力销毁快照
-F	文件系统名
-h	帮助信息
-n	快照名
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。

31.3.3. 挂载快照

快照被视为单独的文件系统，可以在 Lustre 客户端上挂载，但必须使用`-o ro`选项将快照文件系统挂载为只读文件系统。如果`mount`选项不包含该只读选项，将导致挂载失败。

注意

在客户端上挂载快照之前，必须使用 `lctl` 工具将先在服务器上挂载快照。

在服务器上挂载快照，请在 MGS 上运行 `lctl` 命令：

```
lctl snapshot_mount <-F | --fsname fsname> [-h | --help]
<-n | --name ssname> [-r | --rsh remote_shell]
```

选项	说明
-F	文件系统名
-h	帮助信息
-n	快照名
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。

成功在服务器上挂载快照后，客户端便可以将快照挂载为只读文件系统。例如，要为名为 *myfs* 的文件系统装入名为 *snapshot_20170602* 的快照，使用以下挂载命令：

```
mgs# lctl snapshot_mount -F myfs -n snapshot_20170602
```

服务器上的快照挂载完成后，使用 `lctl snapshot_list`返回快照的文件系统名：

```
ss_fsname=$(lctl snapshot_list -F myfs -n snapshot_20170602 |
          awk '/^snapshot_fsname/ { print $2 }')
```

最后，在客户端上挂载快照：

```
mount -t lustre -o ro $MGS_nid:/$ss_fsname $local_mount_point
```

31.3.4. 卸载快照

要从服务器卸载快照，首先在每个客户端上使用标准的`umount`命令从所有客户端卸载快照文件系统。例如，要卸载名为 *snapshot_20170602* 的快照文件系统，请在所有挂载了它的客户端上运行以下命令：

```
client# umount $local_mount_point
```

所有客户端完成快照文件系统卸载后，在挂载快照的服务器节点上运行以下`lctl`命令：

```
lctl snapshot_umount [-F | --fsname fsname] [-h | --help]
<-n | -- name ssname> [-r | --rsh remote_shell]
```

选项	说明
-F	文件系统名
-h	帮助信息
-n	快照名
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。

例如：

```
1 lctl snapshot_umount -F myfs -n snapshot_20170602
```

31.3.5. 列出快照

列出给定文件系统的可用快照，请在 MGS 上运行以下lctl命令：

```
1 lctl snapshot_list [-d | --detail] <-F | --fsname fsname>
2 [-h | -- help] [-n | --name ssname] [-r | --rsh remote_shell]
```

选项	说明
-d	列出指定快照的各部分
-F	文件系统名
-h	帮助信息
-n	快照名。如果没有提供快照名，将显示此文件系统的所有快照。
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。

31.3.6. 修改快照属性

Lustre 快照目前有五个用户可见属性：快照名称、快照注释、创建时间、修改时间和快照文件系统名称。其中，前两个属性可以修改。重命名遵循通用的 ZFS 快照名称规则，如最大长度为 256 字节、不能与预留名称冲突等等。

要修改快照的属性，请在 MGS 上运行以下lctl命令：

```
1 lctl snapshot_modify [-c | --comment comment]
2 <-F | --fsname fsname> [-h | --help] <-n | --name ssname>
3 [-N | --new new_ssname] [-r | --rsh remote_shell]
```

选项	说明
-c	更新快照注释
-F	文件系统名
-h	帮助信息
-n	快照名
-N	重新命名快照为 *new_ssname*
-r	用于与远程目标进行通信的远程外壳。默认值是'ssh'。

31.4. 全局写屏障

快照在多个 MDT 和 OST 上是非原子型的，这意味着如果创建快照时文件系统上存在活动，则在 MDT 快照和 OST 快照之间的时间间隔中创建或销毁的文件可能存在用户可见的名称空间不一致问题。为保证文件系统快照的一致性，我们可以设置全局写屏障或将系统" 冻结"。完成该设置后，所有元数据修改在写屏障被主动移除（" 解冻"）或过期前都将被阻止。用户可以为该全局屏障设置超时参数，或明确地删除屏障。超时时间默认为 30 秒。

请注意，即使没有设置全局屏障，快照仍可用。如果不使用屏障，当前客户端正在修改的文件（写入、创建、取消链接）可能存在如上所述的不一致情况，其他未修改的文件可以正常使用。

使用`lctl snapshot_create`及`-b`选项请求创建快照，将在内部调用写屏障。因此，使用快照时不需要明确使用屏障，但在创建快照之前请包含该选项。

31.4.1. 添加屏障

要添加全局写屏障，请在 MGS 上运行`lctl barrier_freeze`命令：

```
1 lctl barrier_freeze <fsname> [timeout (in seconds)]
2 where timeout default is 30.
```

将文件系统 *testfs* 冻结 15 秒：

```
1 mgs# lctl barrier_freeze testfs 15
```

命令成功运行则无输出信息，否则将输出错误消息。

31.4.2. 移除屏障

移除全局写屏障，请在 MGS 上运行`lctl barrier_thaw`命令：

```
lctl barrier_thaw <fsname>
```

为文件系统 *testfs* 解冻：

```
mgs# lctl barrier_thaw testfs
```

命令成功运行则无输出信息，否则将输出错误消息。

31.4.3. 查询屏障

查看全局写障碍剩余时间，请在 MGS 上运行`lctl barrier_stat`命令：

```
# lctl barrier_stat <fsname>
```

查询文件系统 *testfs* 的写屏障统计信息：

```
mgs# lctl barrier_stat testfs
The barrier for testfs is in 'frozen'
The barrier will be expired after 7 seconds
```

命令成功运行则将输出如下表所示类别的统计信息，否则将输出错误消息。
写屏障可能存在的状态和相关含义如下表所示：

状态	含义
`init`	该系统上未曾设置屏障
`freezing_p1`	设置写屏障的第一阶段
`freezing_p2`	设置写屏障的第二阶段
`frozen`	已成功设置写屏障
`thawing`	写屏障" 解冻"
`thawed`	写屏障已" 解冻"
`failed`	设置写屏障失败
`expired`	写屏障超时
`rescan`	MDTs 状态扫描，见 `barrier_rescan`
`unknown`	其他情况

如果屏障处于'freezing_p1'、'freezing_p2' 或'frozen' 状态，将返回写屏障剩余的时间。

31.4.4. 重新扫描屏障

要重新扫描全局写屏障以检查哪些 MDT 处于活动状态，请在 MGS 上运行`lctl barrier_rescan`命令：

```
lctl barrier_rescan <fsname> [timeout (in seconds)],
where the default timeout is 31. seconds.
```

重新扫描文件系统 *testfs* 的写屏障：

```
mgs# lctl barrier_rescan testfs
1 of 4 MDT(s) in the filesystem testfs are inactive
```

如果该命令成功，将输出总 MDT 数量及不可用的 MDT 数量。否则，将输出错误消息。

31.5. 快照日志

所有快照活动的日志可以在文件 `/var/log/lsnapshot.log`中找到。该文件包含了快照创建和挂载、属性更改的时间信息，以及其他快照相关信息。

以下是 `/var/log/lsnapshot` 文件的样本：

```
Mon Mar 21 19:43:06 2016
(15826:jt_snapshot_create:1138:scratch:ssh): Create snapshot lss_0_0
successfully with comment <(null)>, barrier <enable>, timeout <31.>
Mon Mar 21 19:43:11 2016(131.31.:jt_snapshot_create:1138:scratch:ssh):
Create snapshot lss_0_1 successfully with comment <(null)>, barrier
<disable>, timeout <-1>
Mon Mar 21 19:44:38 2016 (17161:jt_snapshot_mount:2013:scratch:ssh):
The snapshot lss_1a_0 is mounted
Mon Mar 21 19:44:46 2016
(17662:jt_snapshot_umount:2167:scratch:ssh): the snapshot lss_1a_0
have been umounted
Mon Mar 21 19:47:12 2016
(20897:jt_snapshot_destroy:1312:scratch:ssh): Destroy snapshot
lss_2_0 successfully with force <disable>
```

31.6. Lustre 配置日志

快照独立于其原始文件系统，被视为可由 Lustre 客户端节点挂载的新文件系统名。文件系统名是配置日志名的一部分，存在于配置日志条目中。有两个用于操作配置日志的命令：`lctl fork_lcfg`和`lctl erase_lcfg`。

快照命令将在需要时内部调用配置日志功能。因此，使用快照时，屏障不是必需的，而是作为一个选项包含在这里。以下配置日志命令独立于快照，可单独使用。

分配配置日志，请在 MGS 上运行以下`lctl`命令：

```
lctl fork_lcfg
```

用例：fork_lcfg

擦除配置日志，请在 MGS 上运行以下`lctl`命令：

```
lctl erase_lcfg
```

用例：erase_lcfg

第三十二章 Lustre 网络性能测试（LNet Self-Test）

32.1. LNet 自检概述

LNet self-test（自检）是在 LNet 和 Lustre 网络驱动程序（LND）上运行的内核模块。它旨在：

- 测试 Lustre 网络的连接能力
- 运行 Lustre 网络的回归测试
- 测试 Lustre 网络的性能

获得 Lustre 网络的性能结果后，可调整 LNet 的参数以获得最佳性能。

注意

除了性能影响之外，LNet 自检对 Lustre 文件系统不可见。

LNet 自检集群包括以下两种类型的节点：

- **控制台节点** - 用于控制和监视 LNet 自检集群的节点。控制台节点可以看作 LNet 自检的用户界面，可以是测试集群中的任何节点。所有自检命令都从控制台节点输入。通过控制台节点，用户可以控制和监视整个 LNet 自检集群（会话）的状态。控制台节点是独占的，用户不能通过一个控制台节点控制两个不同的会话。
- **测试节点** - 运行测试的节点。测试节点由用户通过控制台节点进行控制，用户不需要直接登录它们。

LNet 自检有两个用户实用程序：

- **lst** - 自检控制台的用户界面（在控制台节点上运行）。它提供了用于控制整个测试系统的命令列表，如创建会话、创建测试组等等。
- **lstclient** - 用户空间 LNet 自检程序（运行在测试节点上）。`lstclient`实用程序与用户空间 LND 和 LNet 链接。如果仅使用了内核空间 LNet 和 LND，则不需要此实用程序。

注意

测试节点可以位于内核或用户空间中。控制台节点可以通过运行`lst add_group NID`邀请内核测试节点加入会话，但控制台节点不能主动向会话中添加用户空间测试节点。当测试节点运行`lstclient`连接控制台节点时，控制台节点可以被动地接受加入会话的测试节点。

32.1.1. 前提条件

要运行 LNet 自检，以下模块必需同时在控制台节点和测试节点上加载：

- `libcfs`
- `net`
- `lnet_selftest`
- `klnds`：您网络配置所需的 Lustre 内核网络驱动 (LND)（如 `ksocklnd`、`ko2iblnd`.....）。

加载所需模块，运行：

```
1 modprobe lnet_selftest
```

该命令将递归地加载 LNet 自检所依赖的模块。

注意

尽管控制台节点和测试节点需要加载所有必备模块，用户空间测试节点不需要这些模块。

32.2. LNet 自检操作

本节主要介绍如何创建和运行 LNet 自检。以下示例模拟了在 InfiniBand 网络上的客户端访问 TCP 网络上的一组 Lustre 服务器的流量模式（通过 LNet 路由器连接）。在这个例子中，一半的客户端正在读，一半的客户端正在写。

32.2.1. 创建会话

会话是在测试节点上运行的一组进程。为确保会话独占该测试节点，一个节点上一次只能运行一个会话。使用控制台节点创建、更改或销毁会话（`new_session`, `end_session`, `show_session`）。有关会话参数的更多信息，请参见本章第 3.1 节" 会话命令"。

几乎所有的操作都应该在会话环境中执行。用户只能在自己的会话中通过控制台节点操作节点。如果会话结束，所有测试节点的会话环境中止。

在控制台节点上设置`LST_SESSION`环境变量以标识会话，并创建一个名为`read_write`的会话：

```
export LST_SESSION=$$
lst new_session read_write
```

32.2.2. 设置组

组是被命名的节点集合。一个 LNet 自检会话中可以有任意数量的组。由于测试节点可以包含在任意数量的组中，组成员不受限制。

组中的每个节点都有一个等级，由节点被添加到组中的顺序决定。该排名用于建立流量测试模式。

用户只能控制自己会话中的节点。用户需要将节点添加到（会话的）组中来把节点分配给会话。组中的所有节点都可以通过组名来引用。一个节点可以分配给多个会话组。

在以下示例中，我们在控制台节点上建立了三个组：

```
lst add_group servers 192.168.10.[8,10,12-16]@tcp
lst add_group readers 192.168.1.[1-253/2]@o2ib
lst add_group writers 192.168.1.[2-254/2]@o2ib
```

这三个组包括：

- LNet 自检会话期间作为" 服务器" 被" 客户端" 访问的节点
- 作为" 客户端" 节点，模拟从" 服务器" 读取数据。
- 作为" 客户端" 节点，模拟将数据写入" 服务器"。

注意

通过运行`lst add_group NID`，控制台节点可以将内核空间测试节点与会话相关联，但不能主动添加用户空间测试节点到会话中。当测试节点运行`lstclient`连接控制台节点时，控制台节点可以被动地接受加入会话的测试节点。

32.2.3. 定义及运行测试

测试将在两组节点之间生成网络负载。其中，源组用`--from`参数标识，目标组用`--to`参数标识。当测试运行时，`--from group`模拟客户端，`--to group`模拟一组服务器，客户端向服务器节点发送请求并接收返回的响应。此活动旨在模拟 Lustre 文件系统 RPC 流量。

批处理测试（*batch*）是同时开始、同时停止，且并行运行的一组测试。测试必须始终作为 *batch* 的一部分运行（即使它只包含单个测试）。用户只能运行或停止这一组测试，而不是单个测试。

批处理测试对文件系统没有破坏性，可以在普通的 Lustre 文件系统环境中运行（假设其性能影响是可接受的）。

假设一个简单的批处理测试只包含了一个测试，即确定网络带宽是否导致了 I/O 瓶颈。在本例中，`--to group`由 Lustre OSS 组成，`--from group`由计算节点组成。可以在此批处理测试中添加第二个测试来执行登录节点到 MDS 的 ping，以查看检查点是如何影响`ls -l`进程的。

有两种类型的测试可用：

- **ping -** `ping` 生成一个简短的请求消息，以触发一个同样简短的响应。Ping 有助于确定延迟和小消息开销并模拟 Lustre 元数据流量。
- **brw -** 在 `brw` （批量读写）测试中，数据从目标传输到源（`brwread`）或数据从源传输到目标（`brwwrite`）。批量传输的大小使用 size 参数设置。`brw`测试有助于确定网络带宽和模拟 Lustre /O 流量。

下面的例子创建了一个名为`bulk_rw`的批处理。，然后添加了两个`brw`测试。第一个测试模拟读操作，1M 数据从服务器发送到客户端，进行简单数据验证检查。第二个测试模拟写操作，4K 数据从客户端发送到服务器，进行完整的数据验证检查。

```
1 lst add_batch bulk_rw
2 lst add_test --batch bulk_rw --from readers --to servers \
3   brw read check=simple size=1M
4 lst add_test --batch bulk_rw --from writers --to servers \
5   brw write check=full size=4K
```

流量模式和测试强度由测试类型、测试节点分布、测试并发性和 RDMA 操作类型等属性决定。有关更多详细信息，请参见本章第 3.3 节" 批处理和测试命令"。

32.2.4. 脚本样例

此 LNet 自检脚本样例模拟了在 InfiniBand 网络上的客户端访问 TCP 网络上的一组服务器的流量模式（通过 LNet 路由器连接）。在这个例子中，一半的客户端正在读，一半的客户端正在写。

在控制台节点上运行此脚本：

```
1 #!/bin/bash
2 export LST_SESSION=$$
3 lst new_session read/write
4 lst add_group servers 192.168.10.[8,10,12-16]@tcp
5 lst add_group readers 192.168.1.[1-253/2]@o2ib
6 lst add_group writers 192.168.1.[2-254/2]@o2ib
7 lst add_batch bulk_rw
8 lst add_test --batch bulk_rw --from readers --to servers \
```

```
brw read check=simple size=1M
lst add_test --batch bulk_rw --from writers --to servers \
brw write check=full size=4K
# start running
lst run bulk_rw
# display server stats for 30 seconds
lst stat servers & sleep 30; kill $!
# tear down
lst end_session
```

注意

该脚本可简单地利用 shell 变量或命令行参数传递组 NID 参数（适用于通用目的）。

32.3. LNet 自检命令索引

LNet self-test (`lst`) 实用程序用于发出 LNet 自检命令。`lst`需要一些命令行参数。第一个参数为命令名称，后面的参数由命令指定。

32.3.1. 会话命令

这一小节介绍 `lst` 会话命令。

LST_FEATURES

`lst`使用`LST_FEATURES`环境变量来确定启用哪些可选功能。所有功能都默认为禁用。`LST_FEATURES`支持的值有：

- **1 -** 为 LNet 自检启用可变的页面大小。

示例：

```
export LST_FEATURES=1
```

LST_SESSION

`lst` 使用 `LST_SESSION` 环境变量在本地自检控制台节点上标识会话。该变量应为标识节点上每个会话进程的唯一数字值。在 shell 脚本内将其设置为 shell 进程 ID 非常方便，这样也有益于交互式使用。几乎所有的`lst`命令都需要设置`LST_SESSION`。

示例：

```
export LST_SESSION=$$
```

`new_session [--timeout SECONDS] [--force] SESSNAME`

创建一个名为 *SESSNAME* 的新会话。

参数	说明
--timeout seconds	会话的控制台超时值。如果在此期间它一直保持空闲状态（即没有命令发出），会话将自动结束。
--force	结束冲突的会话。这决定了一个会话与另一个会话产生冲突时谁" 胜出"。当此节点上已有活动会话时，除非指定了 --force 标志，否则尝试创建新会话将失败。如果指定了 --force 标志，则活动会话结束。同样地，如果会话尝试添加已被另一个会话" 占有" 的节点，则--force 标志将允许此会话" 窃取" 该节点。
name	列出会话或报告会话冲突时打印的可读字符串。

示例：

```
1 $ lst new_session --force read_write
```

end_session

停止当前会话中的所有操作和测试，并清除会话的状态。

```
1 $ lst end_session
```

show_session

显示会话信息。该命令输出有关当前会话的信息。它不需要在过程环境中定义 LST_SESSION。

```
1 $ lst show_session
```

32.3.2. 组命令

这一小节介绍 lst 组命令。

add_group name NIDS [NIDs...]

创建组并向该组添加测试节点的列表。

参数	**说明**
name	组的名称
NIDs	一个字符串，可以扩展为包含一个或多个 LNet NID。

示例：

```
$ lst add_group servers 192.168.10.[35,40-45]@tcp
$ lst add_group clients 192.168.1.[10-100]@tcp 192.168.[2,4].\
  [10-20]@tcp
```

update_group name [--refresh] [--clean status] [--remove NIDs]

更新组中节点的状态或调整组成员。可用于从组中排除某些已崩溃的节点。

参数	说明
--refresh	刷新组中所有不活动节点的状态。
--clean status	从组中删除具有指定状态的节点。状态可能是： active －该节点正处于当前会话； busy －该节点被其他会话占有； down －该节点被标记为下线状态； unknown －该节点状态尚不明； invalid －除活动状态外的任一状态。
--remove NIDs	从组中移除指定节点。

示例：

```
$ lst update_group clients --refresh
$ lst update_group clients --clean busy
$ lst update_group clients --clean invalid // \
  invalid == busy || down || unknown
$ lst update_group clients --remove \192.168.1.[10-20]@tcp
```

list_group [name] [--active] [--busy] [--down] [--unknown] [--all]

打印组的有关信息；如果未指定组，则列出当前会话中的所有组。

参数	说明
name	组名称
--active	列出状态为 active 的节点
--busy	列出状态为 busy 的节点

参数	说明
--down	列出状态为 down 的节点
--unknown	列出状态为 unknown 的节点
--all	列出所有节点

示例:

```
$ lst list_group
1) clients
2) servers
Total 2 groups
$ lst list_group clients
ACTIVE BUSY DOWN UNKNOWN TOTAL
3 1 2 0 6
$ lst list_group clients --all
192.168.1.10@tcp Active
192.168.1.11@tcp Active
192.168.1.12@tcp Busy
192.168.1.13@tcp Active
192.168.1.14@tcp DOWN
192.168.1.15@tcp DOWN
Total 6 nodes
$ lst list_group clients --busy
192.168.1.12@tcp Busy
Total 1 node
```

del_group name

从会话中删除一个组。如果该组被任何测试引用，则操作失败。如果组中的节点仅由该组引用，则将其从当前会话中踢出; 否则，他们将仍在当前会话中。

```
$ lst del_group clients
```

lstclient --sesid NID --group name [--server_mode]

使用lstclient来运行 userland 自检客户端。应先在控制台上创建会话，然后再执行lstclient命令。lstclient只有两个强制选项:

参数	说明
--sesid NID	第一个控制台的 NID。
--group name	要加入的测试组。
--server_mode	可选选项。包含此选项将强制 LNet 以服务器的方式工作，如启动接受器（如果底层 NID 需要）或使用特权端口。只允许 root 用户使用--server_mode 选项。

示例：

```
1 Console $ lst new_session testsession
2 Client1 $ lstclient --sesid 192.168.1.52@tcp --group clients
```

示例：

```
1 Client1 $ lstclient --sesid 192.168.1.52@tcp |--group clients --server_mode
```

32.3.3. 批处理测试命令

这一小节介绍 lst 批量测试命令。

add_batch name

会话启动时会创建一个名为 *batch* 的默认批处理测试集。您可以使用add_batch指定批处理测试名称：

```
1 $ lst add_batch bulkperf
```

创建一个名为 *bulkperf* 的批处理测试。

```
1 add_test --batch batchname [--loop loop_count] [--concurrency active_count]
    [--distribute source_count:sink_count] \
2         --from group --to group brw|ping test_options
```

在批处理中添加一个测试，其参数如下所示：

参数	说明
--batch batchname	命名一组测试以便随后执行。
--loop loop_count	运行测试的次数。
--concurrency active_count	一次激活的请求数。
--distribute source_count:	为指定测试确定客户端节点与服务器节点

参数	说明
sink_count	的比率。这将允许您指定各种拓扑，如一对一、多对多；将源组和目标组划分成子集，只有来自源组子集的节点与来自目标组子集的节点匹配时才能进行通信。
--from group	源组（测试客户端）。
--to group	目标组（测试服务器）。
ping	发送一个简短的请求信息，收获一个简短的应答消息。更多内容请参见 2.3 节" 定义和运行测试"。ping没有任何附加选项。
brw	收获一个简短的应答消息。更多内容请参见 2.3 节" 定义和运行测试"。选项有： read | write－读或写，默认为读。 size=bytes[KM]－以字节，千字节或兆字节为单位的 I/O 大小（即 size= 1024，size=4K，size=1M）；默认值是 4 千字节。 check=full|simple－数据校验检查（数据校验和）。默认不进行此项数据校验检查。

有关参数 distribute 使用的示例

```
1 Clients: (C1, C2, C3, C4, C5, C6)
2 Server: (S1, S2, S3)
3 --distribute 1:1 (C1->S1), (C2->S2), (C3->S3), (C4->S1), (C5->S2),
4 \(C6->S3) /* -> means test conversation */ --distribute 2:1 (C1,C2->S1),
      (C3,C4->S2), (C5,C6->S3)
5 --distribute 3:1 (C1,C2,C3->S1), (C4,C5,C6->S2), (NULL->S3)
6 --distribute 3:2 (C1,C2,C3->S1,S2), (C4,C5,C6->S3,S1)
7 --distribute 4:1 (C1,C2,C3,C4->S1), (C5,C6->S2), (NULL->S3)
8 --distribute 4:2 (C1,C2,C3,C4->S1,S2), (C5, C6->S3, S1)
9 --distribute 6:3 (C1,C2,C3,C4,C5,C6->S1,S2,S3)
```

--distribute 1:1 为默认设置，即一个源节点与一个目标节点进行通信。

使用 --distribute 1: n （*n* 为目标组的大小）时，一个源节点与目标组的所有节点进行通信。

注意，如果源节点比目标节点多，则某些源节点可能共享相同的目标节点。如果目标节点数多于源节点数，则排名较高的目标节点将处于空闲状态。

有关 brw 测试的示例

```
1 $ lst add_group clients 192.168.1.[10-17]@tcp
2 $ lst add_group servers 192.168.10.[100-103]@tcp
3 $ lst add_batch bulkperf
4 $ lst add_test --batch bulkperf --loop 100 --concurrency 4 \
5   --distribute 4:2 --from clients brw WRITE size=16K
```

在上面的例子中，一个名为 *bulkperf* 的批量测试将执行 16k 字节的批量写入请求。在此测试中，两组的四个客户端（源）分别写入四台服务器（目标），如下所示：

- 192.168.1.[10-13]将写入 192.168.10.[100,101]
- 192.168.1.[14-17]将写入 192.168.10.[102,103]

list_batch [name] [--test index] [--active] [--invalid] [--server|client]

列出当前会话中的批量测试或列出批量测试中的客户端和服务器节点。

参数	说明
--test index	列出批处理中的测试。如果未使用任何选项，则会列出该批次中的所有测试。如果使用下列选项之一，则只列出指定的测试： active －只列出活动的测试； invalid －只列出无效的测试； server/client －列出此批量测试的服务器和客户端节点。

示例：

```
1 $ lst list_batchbulkperf
2 $ lst list_batch bulkperf
3 Batch: bulkperf Tests: 1 State: Idle
4 ACTIVE BUSY DOWN UNKNOWN TOTAL
5 client 8 0 0 0 8
```

```
server 4 0 0 0 4
Test 1(brw) (loop: 100, concurrency: 4)
ACTIVE BUSY DOWN UNKNOWN TOTAL
client 8 0 0 0 8
server 4 0 0 0 4
$ lst list_batch bulkperf --server --active
192.168.10.100@tcp Active
192.168.10.101@tcp Active
192.168.10.102@tcp Active
192.168.10.103@tcp Active
```

`run name`

运行此批量测试：

```
$ lst run bulkperf
```

`stop name`

停止此批量测试：

```
$ lst stop bulkperf
```

`query name [--test index] [--timeout seconds] [--loop loopcount] [--delay seconds] [--all]`

查询批量测试状态：

参数	说明
`--test index`	只查询指定测试。测试的起始索引为 1。
`--timeout seconds`	等待 RPC 的超时时间。默认值是 5 秒。
`--loop #`	查询的循环次数。
`--delay seconds`	每次查询的时间间隔。默认值是 5 秒。
`--all`	批处理或测试中所有节点的状态列表。

示例：

```
$ lst run bulkperf
$ lst query bulkperf --loop 5 --delay 3
Batch is running
Batch is running
```

```
Batch is running
Batch is running
Batch is running
$ lst query bulkperf --all
192.168.1.10@tcp Running
192.168.1.11@tcp Running
192.168.1.12@tcp Running
192.168.1.13@tcp Running
192.168.1.14@tcp Running
192.168.1.15@tcp Running
192.168.1.16@tcp Running
192.168.1.17@tcp Running
$ lst stop bulkperf
$ lst query bulkperf
Batch is idle
```

32.3.4. 其他命令

这一小节介绍 lst 命令。

ping [-session] [--group name] [--nodes NIDs] [--batch name] [--server] [--timeout seconds]

向节点发送'hello' 查询。

参数	说明
--session	向当前会话的所有节点发送 Ping。
--group name	向指定组的节点发送 Ping。
--nodes NIDs	向指定节点发送 Ping。
--batch name	向批处理的所有客户端发送 Ping。
--server	将 RPC 发送到所有服务器节点而不是客户端节点。该选项仅和--batch name 一起使用。
--timeout seconds	RPC 超时时间。

示例：

```
# lst ping 192.168.10.[15-20]@tcp
```

```
192.168.1.15@tcp Active [session: liang id: 192.168.1.3@tcp]
192.168.1.16@tcp Active [session: liang id: 192.168.1.3@tcp]
192.168.1.17@tcp Active [session: liang id: 192.168.1.3@tcp]
192.168.1.18@tcp Busy [session: Isaac id: 192.168.10.10@tcp]
192.168.1.19@tcp Down [session: <NULL> id: LNET_NID_ANY]
192.168.1.20@tcp Down [session: <NULL> id: LNET_NID_ANY]
```

stat [--bw] [--rate] [--read] [--write] [--max] [--min] [--avg] " " [--timeout seconds] [--delay seconds] group|NIDs [group|NIDs]

一个或多个节点的总体性能和 RPC 统计信息。

参数	说明
--bw	显示指定组/节点的带宽
--rate	显示指定组/节点的 RPC 率
--read	显示指定组/节点的读操作统计信息
--write	显示指定组/节点的写操作统计信息
--max	显示统计信息中的最大值
--min	显示统计信息中的最小值
--avg	显示统计信息中的平均值
--timeout seconds	统计数字中的 RPC 超时时间。默认为 5 秒。
--delay seconds	统计数字中的时间间隔（以秒为单位）。

示例：

```
$ lst run bulkperf
$ lst stat clients
[LNet Rates of clients]
[W] Avg: 1108 RPC/s Min: 1060 RPC/s Max: 1155 RPC/s
[R] Avg: 2215 RPC/s Min: 2121 RPC/s Max: 2310 RPC/s
[LNet Bandwidth of clients]
[W] Avg: 16.60 MB/s Min: 16.10 MB/s Max: 17.1 MB/s
[R] Avg: 40.49 MB/s Min: 40.30 MB/s Max: 40.68 MB/s
```

指定组名称（*group*）将为该测试组中的所有节点收集统计信息。例如：

```
$ lst stat servers
```

servers 是由 `lst add_group`创建的组的名称。

指定*NID* 范围 (*NIDs*) 将为选定节点收集统计信息。例如：

```
$ lst stat 192.168.0.[1-100/2]@tcp
```

只有 LNet 性能统计信息可用。默认情况下，所有统计信息都会显示。用户可以使用这些选项指定附加信息。

`show_error [--session] [group|NIDs]...`

列出测试节点上故障 RPC 数量：

参数	说明
`--session`	列出当前测试会话中的错误。此选项不会列出历史 RPC 错误。

示例：

```
$ lst show_error client
sclients
12345-192.168.1.15@tcp: [Session: 1 brw errors, 0 ping errors] \
  [RPC: 20 errors, 0 dropped,
12345-192.168.1.16@tcp: [Session: 0 brw errors, 0 ping errors] \
  [RPC: 1 errors, 0 dropped, Total 2 error nodes in clients
$ lst show_error --session clients
clients
12345-192.168.1.15@tcp: [Session: 1 brw errors, 0 ping errors]
Total 1 error nodes in clients
```

第三十三章对 Lustre 文件系统进行基准测试 (LustreI/O 工具箱)

33.1. 使用 Lustre I/O 工具箱

Lustre I/O 工具箱中的工具用于对 Lustre 文件系统硬件进行基准测试，并在安装 Lustre 软件之前确认硬件是否正常工作。它也可以用来验证集群中各种硬件和软件层的性能，查找和排除 I/O 问题。

通常，性能测试从单个原始设备开始，然后再到设备组。一旦建立了原始性能，其他软件层就会逐渐增加并进行测试。

33.1.1. Lustre I/O 工具箱内容

I/O 工具箱包含三个测试，其中每个测试都会测试 Lustre 软件堆栈中的一个更高层次的软件层：

- `sgpdd-survey` - 只测量设备的基础" 裸机" 性能，绕过内核块设备层，缓冲区缓存和文件系统。
- `obdfilter-survey` - 直接在 OSS 节点上或通过网络在 Lustre 客户端上测量一个或多个 OST 的性能。
- `ost-survey` - 单独对 OST 执行 I/O 操作来进行性能比较，以检测 OST 是否由于硬件问题性能受损。

通常在这些测试中，Lustre 文件系统应该提供 85-90％的原始设备性能。

`stats-collect` 实用程序负责收集来自 Lustre 客户端和服务器的应用程序分析信息。更多内容见第 6 节" 收集应用程序分析信息（`stats-collect`）"。

33.1.2. Lustre I/O 工具箱使用准备

必须满足以下条件才能使用 Lustre I/O 套件中的测试：

- 可在系统中对节点进行远程免密访问（`ssh` 或 `rsh`）。
- LNet 自检已完成，Lustre 网络已正常安装及配置。
- Lustre 文件系统软件已安装。
- `sg3_utils` 软件包提供了 `sgp_dd` 工具（`sg3_utils` 为独立的 RPM 软件包，可通过 YUM 在网上获取）。

从 https://downloads.whamcloud.com/下载 LustreI/O 工具箱 (`lustre-iokit`)。

33.2. 测试原始硬件 I/O 性能 (`sgpdd-survey`)

`sgpdd-survey`工具用于测试原始硬件的裸机 I/O 性能，同时绕过尽可能多的内核。本调查通过模拟 OST 服务多个条带文件来表征 SCSI 设备的性能。此调查收集的数据可以帮助您了解此设备的 Lustre OST 的性能预期。

该脚本使用`sgp_dd`执行原始磁盘 I/O。它使用数量可变的`sgp_dd`线程运行，以显示性能如何随着请求队列深度的变化而变化。

该脚本生成数量可变的`sgp_dd`实例，每个实例读取或写入磁盘的不同区域，从而演示多个并发的条带文件的性能差异。

下面介绍了有关磁盘性能测量的一些技巧和见解。其中一些信息针对的是 RAID 阵列或 Linux RAID 的实施。

- **性能受限于最慢的磁盘。** 在创建 RAID 阵列之前，分别对所有磁盘进行基准测试。我们经常会遇到阵列中所有设备的驱动器性能不一致的情况，请更换比其他磁盘慢得多的磁盘。
- **磁盘和阵列对请求大小非常敏感。** 要确定给定磁盘的最佳请求大小，请使用不同请求大小（从 4 KB、1MB 到 2 MB）对磁盘进行基准测试。

注意

`sgpdd-survey` 脚本会覆盖正在测试的设备，从而导致该设备上**所有数据丢失**。因此，选择要测试的设备时请务必小心。

加载所有 LUN 的阵列性能并不总是与单个 LUN 单独测试时的性能相匹配。

必要条件:

- `sg3_utils` 软件包的`sgp_dd` 工具。
- Lustre 软件不少必需的。

被测试的设备必须满足以下两个要求之一:

- 如果设备是 SCSI 设备，则它必须出现在`sg_map`的输出中（确保内核模块`sg`已加载）。
- 如果设备是原始设备，则它必须出现在`raw -qa`的输出中。

原始设备和 SCSI 设备在测试规范中不能混合使用。

注意

如果您需要创建原始设备以使用`sgpdd-survey`工具，请注意：由于某些版本的" 原始" 实用程序（包括 Red Hat Enterprise Linux 4U4 提供的版本）的某些版本中的错误，原始设备 0 无法使用。

33.2.1. 调试 Linux 存储设备

传输大量 I/O 数据（1 MB）到磁盘，可能需要如是调整以下几个内核参数:

```
1 /sys/block/sdN/queue/max_sectors_kb = 4096
2 /sys/block/sdN/queue/max_phys_segments = 256
3 /proc/scsi/sg/allow_dio = 1
4 /sys/module/ib_srp/parameters/srp_sg_tablesize = 255
5 /sys/block/sdN/queue/scheduler
```

注意

推荐使用的调度程序为 **deadline** 和 **noop**。默认为 **deadline**，也可将其设置为 **noop**。

33.2.2. 运行sgpdd-survey

sgpdd-survey脚本必须要根据测试的特定设备以及脚本保存其工作和结果文件的位置（通过指定${rslt}变量）来进行自定义。自定义变量在脚本开始处描述。

当sgpdd-survey脚本运行时，它会创建大量工作文件和两个结果文件。所有创建的文件的名称都以变量${rslt}中定义的内容为前缀。（默认值是/tmp。）这些文件包括:

- 包含标准输出数据的文件（如 stdout）

  rslt_date_time.summary

- 临时（tmp）文件

  rslt_date_time_*

- 收集 tmp 文件进行再次进行详细验证

  rslt_date_time.detail

  stdout 和 .summary 文件将含以下类似内容:

```
1 total_size 8388608K rsz 1024 thr 1 crg 1 180.45 MB/s 1 x 180.50 \
2         = 180.50 MB/s
```

每一行对应于一个测试的运行。每个运行的测试将具有不同数量的线程、记录大小或区域数量。

- total_size - 被测试文件的大小（以 KB 为单位，以上示例中为 8 GB）。
- rsz - 记录大小（以 KB 为单位，以上示例为 1 MB）。
- thr - 生成 I/O 的线程数量（以上示例为 1 个线程）。
- crg - 当前区域，I/O 发送到的磁盘上的不相交的区域的数量（以上示例为 1 个区域，表示不需要进行搜索）。
- MB/s - 总带宽，即总数据大小除以所需时间（以上示例为 180.45 MB/s）。
- MB/s - 剩下的数字显示：区域数量 X 最慢磁盘的性能，作为对总带宽的完整性检查。

如果线程太多，sgp_dd脚本不太可能分配 I/O 缓冲区，则会输出 ENOME 而不是总带宽结果。

如果一个或多个sgp_dd实例未成功报告带宽，则会输出 FAILED 而不是总带宽结果。

33.3. OST 性能测试 (obdfilter-survey)

obdfilter-survey脚本通过不同数量的线程和对象（文件）生成顺序 I/O，以模拟 Lustre 客户端的 I/O 模式。

obdfilter-survey脚本可以无需任何干预网络直接在 OSS 节点上运行，来测量 OST 存储性能；也可以在 Lustre 客户端上远程运行，来测量包括网络开销在内的 OST 性能。

obdfilter-survey用于描绘以下性能情况：

- **本地文件系统**－在这种模式下，obdfilter-survey脚本直接执行一个或多个obdfilter实例。该脚本可以在一个或多个 OSS 节点上运行（如所有 OSS 都连接到相同的多端口磁盘子系统时）。

使用case = disk参数，运行脚本对所有的本地 OST 运行测试。该脚本会自动检测所有本地 OST 并将其包括在调查结果中。

要仅针对特定的 OST 运行测试，请使用 targets=parameter运行脚本，明确列出要测试的 OST。如果某些 OST 位于远程节点上，除了指定 OST 名称（例如，oss2:luster-OST0004）外，还需指定它们的主机名。

所有obdfilter实例都是直接驱动的。该脚本自动加载obdecho模块（如果需要的话）并为每个obdfilter实例创建一个echo_client实例，以便直接生成对 OST 的 I/O 请求。

更多详细信息，请参见本章第 3.1 节" 本地磁盘性能测试"。

- **网络**－在此模式下，Lustre 客户端通过网络生成 I/O 请求，但这些请求不会发送到 OST 文件系统。OSS 节点运行obdecho服务器来接收请求，但在将它们发送到磁盘之前丢弃它们。

将参数 case=network 和 targets=hostname|IP_of_server 传递给脚本。对于每个网络测试案例，脚本都会执行所需的设置。

更多详细信息，请参见本章第 3.2 节" 网络性能测试"。

- **网络上的远程文件系统** - 在这种模式下，obdfilter-survey脚本在 Lustre 客户端生成至远程 OSS 的 I/O，将数据写入文件系统。

对所有本地 OSC 运行测试，请将参数case = netdisk传递给脚本。您也可以将指定了一个或多个要运行测试的 OSC 设备的参数target= parameter传递给脚本（如luster-OST0000-osc-ffff88007754bc00）。

更多详细信息，请参见本章第 3.3 节" 测试远程磁盘性能"。

注意

obdfilter-survey具有潜在的破坏性，存在小概率丢失数据的风险。为降低这种风险，不应在数据需要保存无损的设备上运行obdfilter-survey。因此，运行obdfilter-survey的最佳时间是在 Lustre 文件系统投入生产之前。obdfilter-survey在生产文件系统上运行可能是安全的，因为它使用对象序列 2 来创建对象，而一般文件系统通常使用对象序列 0 创建对象。

如果obdfilter-survey测试在完成之前被终止，则会造成少量空间的流失。您可以忽略它或重新格式化文件系统。

obdfilter-survey脚本不能扩展到数十个 OST，它仅用于测量各个存储子系统的 I/O 性能，而不是整个系统的可扩展性。

必须根据测试组件和脚本工作文件的保存位置对obdfilter-survey脚本进行自定义。请在obdfilter-survey脚本开头描述自定义变量，并特别注意脚本中列出的每个参数的最大值。

33.3.1. 本地磁盘性能测试

obdfilter-survey脚本可以自动或手动在本地磁盘上运行。此脚本通过将不同线程数、对象数和 I/O 大小的工作负载发送 OST 到来分析存储硬件（包括管理存储的文件系统和 RAID 层）的整体吞吐量。

运行obdfilter-survey脚本将输出有关存储硬件性能的信息及其硬件饱和点。

运行plot-obdfilter脚本可实现数据可视化，它根据obdfilter-survey的输出生成一个 CSV 文件和用于导入到电子表格或 gnuplot 中的参数。

运行obdfilter-survey脚本，请创建标准 Lustre 文件系统配置，不需要特殊的设置。

执行自动运行：

1. 启动 Lustre OSTs.

   Lustre OST 应挂载在要测试的 OSS 节点上。Lustre 客户端目前不需要进行挂载。

2. 确认obdecho模块已加载。运行：

```
modprobe obdecho
```

3. 运行 obdfilter-survey 脚本，并使用参数 case=disk。

例如，运行一个两个对象 (nobjhi)，两个线程 (thrhi) 和 1024 MB 传输大小的本地测试：

```
$ nobjhi=2 thrhi=2 size=1024 case=disk sh obdfilter-survey
```

4. 写入、覆盖写、读取等的性能测试如下：

```
# example output
Fri Sep 25 11:14:03 EDT 2015 Obdfilter-survey for case=disk from
    hds1fnb6123
ost 10 sz 167772160K rsz 1024K obj   10 thr   10 write 10982.73 [
    601.97,2912.91] rewrite 15696.54 [1160.92,3450.85] read 12358.60 [
    938.96,2634.87]
...
```

文件 `./lustre-iokit/obdfilter-survey/README.obdfilter-survey` 提供了有关输出内容的解释说明：

```
ost 10          is the total number of OSTs under test.
sz 167772160K   is the total amount of data read or written (in bytes).
rsz 1024K       is the record size (size of each echo_client I/O, in bytes).
obj    10       is the total number of objects over all OSTs
thr    10       is the total number of threads over all OSTs and objects
write           is the test name.  If more tests have been specified they
      all appear on the same line.
10982.73        is the aggregate bandwidth over all OSTs measured by
      dividing the total number of MB by the elapsed time.
[601.97,2912.91] are the minimum and maximum instantaneous bandwidths seen
    on
      any individual OST.
Note that although the numbers of threads and objects are specifed per-OST
in the customization section of the script, results are reported aggregated
over all OSTs.
```

执行手动运行：

1. 启动 Lustre OSTs。

Lustre OST 应挂载在要测试的 OSS 节点上。Lustre 客户端目前不需要进行挂载。

2. 确认obdecho模块已加载。运行：

```
modprobe obdecho
```

3. 确定 OST 名。

在待测试的 OSS 节点上，运行 `lctl dl` 命令。命令输出的第四行列出了 OST 设备名，如：

```
$ lctl dl |grep obdfilter
0 UP obdfilter lustre-OST0001 lustre-OST0001_UUID 1159
2 UP obdfilter lustre-OST0002 lustre-OST0002_UUID 1159
...
```

4. 列出所有您希望测试的 OSTs 。

使用targets=parameter列出所有 OSTs，由空格间隔。按名称列出单个 OST，请使用 fsname-OSTnumber （如lustre-OST0001）的格式，不需要指定 MDS 或 LOV。

5. 运行 obdfilter-survey 脚本，并使用参数 targets=parameter。

例如，运行一个两个对象 (nobjhi)，两个线程 (thrhi) 和 1024 MB 传输大小的本地测试：

```
$ nobjhi=2 thrhi=2 size=1024 targets="lustre-OST0001 \
  lustre-OST0002" sh obdfilter-survey
```

33.3.2. 网络性能测试

obdfilter-survey 脚本只能在网络上自动运行，不能手动运行测试。
要运行网络测试，需要特定的 Lustre 文件系统设置，并确保满足配置要求。
执行自动运行：

1. 启动 Lustre OSTs。

Lustre OST 应挂载在要测试的 OSS 节点上。Lustre 客户端目前不需要进行挂载。

2. 确认obdecho模块已加载。运行：

```
modprobe obdecho
```

3. 启动lctl并检查设备列表是否为空，运行：

```
lctl dl
```

4. 运行 obdfilter-survey 脚本，使用参数 case=network 和 targets=*hostname|ip_of_server*，如：

```
  $ nobjhi=2 thrhi=2 size=1024 targets="oss0 oss1" \
  case=network sh obdfilter-survey
```

5. 在服务器端，查看统计信息：

```
lctl get_param obdecho.echo_srv.stats
```

其中，*echo_srv* 为由脚本创建的 obdecho 服务器。

33.3.3. 远程磁盘性能测试

obdfilter-survey 脚本可以自动或手动在网络磁盘上运行。运行网络磁盘测试，启动 Lustre 标准配置名，无需特殊设置。

执行自动运行：

1. 启动 Lustre OSTs。

Lustre OST 应挂载在要测试的 OSS 节点上。Lustre 客户端目前不需要进行挂载。

2. 确认obdecho模块已加载。运行：

```
modprobe obdecho
```

3. 运行 obdfilter-survey 脚本，使用参数 case=netdisk，如：

```
1 $ nobjhi=2 thrhi=2 size=1024 case=netdisk sh obdfilter-survey
```

执行手动运行：

1. 启动 Lustre OSTs。

Lustre OST 应挂载在要测试的 OSS 节点上。Lustre 客户端目前不需要进行挂载。

2. 确认obdecho模块已加载。运行：

```
modprobe obdecho
```

3. 确定 OSC 名。

在待测试的 OSS 节点上，运行 lctl dl 命令。命令输出的第四行列出了 OSC 设备名，如：

```
1 $ lctl dl |grep obdfilter
2 3 UP osc lustre-OST0000-osc-ffff88007754bc00 \
3       54b91eab-0ea9-1516-b571-5e6df349592e 5
4 4 UP osc lustre-OST0001-osc-ffff88007754bc00 \
5       54b91eab-0ea9-1516-b571-5e6df349592e 5
6 ...
```

4. 列出所有您希望测试的 OSCs 。

使用targets=parameter列出所有 OSCs，由空格间隔。按名称列出单个 OSC，请使用 fsname-OST_name-osc-instance （如lustre-OST0000-osc-ffff88007754bc00）的格式，不需要指定 MDS 或 LOV。

5. 运行 obdfilter-survey 脚本，使用参数 targets=osc 和 case=netdisk。

例如，运行一个两个对象 (nobjhi)，两个线程 (thrhi) 和 1024 MB 传输大小的本地测试：

```
1        $ nobjhi=2 thrhi=2 size=1024 \
2        targets="lustre-OST0000-osc-ffff88007754bc00 \
3        lustre-OST0001-osc-ffff88007754bc00" sh obdfilter-survey
```

33.3.4. 输出文件

obdfilter-survey 脚本运行时，它会创建大量工作文件和两个结果文件。所有文件都以变量 ${rslt}定义的值作为前缀。

文件	说明
${rslt}.summary	和 stdout 一样
${rslt}.script_*	每个主机一个测试脚本文件
${rslt}.detail_tmp*	每个 OST 一个结果文件
${rslt}.detail	收集结果文件进行事后验证

obdfilter-survey脚本遍历执行指定测试的所有线程和对象，并检查所有测试过程是否已成功完成。

注意

obdfilter-survey 如果脚本异常终止或遇到不可恢复的错误，则可能无法正常清除脚本。在这种情况下，可能需要手动清除，包括终止任何正在运行的lctl实例（本地或远程），删除脚本创建的echo_client实例以及卸载obdecho。

33.3.4.1. 脚本输出 .summary 文件及 obdfilter-survey 脚本的 stdout 包含类似以下内容：

```
1 ost 8 sz 67108864K rsz 1024 obj 8 thr 8 write 613.54 [ 64.00, 82.00]
```

其中：

参数及值	说明
ost 8	被测试的 OSTs 总数
sz 67108864K	读/写数据总量 (KB)
rsz 1024	记录大小（每个 echo_client I/O 的大小，单位为 KB）
obj 8	所有 OSTs 上的对象总数
thr 8	所有 OSTs 上的线程总数
write	测试名。如果指定了更多的测试名，它们都将在同一行显示。
613.54	所有 OSTs 上的总带宽（由数据总 MB 数除以所花时间得到）
[64, 82.00]	每个 OST 上的最低和最高的即时带宽

注意

虽然在脚本自定义时，线程和对象的数量是按每个 OST 指定的，但在报告结果时，该数量汇总了所有 OSTs。

33.3.4.2. 可视化结果 为使结果可视化，我们可以将`obdfilter-survey`脚本汇总数据（宽度固定）导入到 Excel（或任何图形包）中，并绘制图像比较带宽和线程数量（不同数量的并发区域）的关系。这将显示执行不同数量的 I/O 时的 OSS 性能情况（给定数量的并行访问对象或文件）。

另外，我们也可使用文件 `lctl get_param obdfilter.*.brw_stats` 中的'disk io size' 直方图来监视和记录每次测试期间的平均磁盘 I/O 大小。这些数字有助于在大型 I/O 未提交到底层磁盘时识别系统中的问题（可能由设备驱动程序或 Linux 块层中的问题引起）。

I/O 工具包中的`plot-obdfilter`脚本是将输出文件处理为.csv 格式并使用`gnuplot`绘制图形的示例。

33.4. OST I/O 性能测试 (`ost-survey`)

`ost-survey` 工具是一个 shell 脚本，它使用`lfs setstripe`对单个 OST 执行 I/O。该脚本将文件（当前使用`dd`）写入 Lustre 文件系统中的每个 OST，并比较读写速度。因此，`ost-survey` 可以用于检测完全相同磁盘子系统之间的异常情况。

注意

我们经常发现集群中 LUN 的性能差异很大。这可能是由磁盘故障、测试期间 RAID 重建，或网络硬件故障引起的。

要运行ost-survey脚本，请提供文件大小（以 KB 为单位）和 Lustre 文件系统挂载点。如：

```
$ ./ost-survey.sh -s 10 /mnt/lustre
```

典型的输出为：

```
Number of Active OST devices : 4
Worst  Read OST indx: 2 speed: 2835.272725
Best   Read OST indx: 3 speed: 2872.889668
Read Average: 2852.508999 +/- 16.444792 MB/s
Worst  Write OST indx: 3 speed: 17.705545
Best   Write OST indx: 2 speed: 128.172576
Write Average: 95.437735 +/- 45.518117 MB/s
Ost#  Read(MB/s)  Write(MB/s)  Read-time  Write-time
----------------------------------------------------
0     2837.440       126.918        0.035      0.788
1     2864.433       108.954        0.035      0.918
2     2835.273       128.173        0.035      0.780
3     2872.890       17.706        0.035      5.648
```

33.5. MDS 性能测试 (mds-survey)

mds-survey 脚本用于测试本地元数据性能，使用echo_client来驱动 MDS 堆栈的 MDD 层。它可以用于以下的操作：

- Open-create/mkdir/create
- Lookup/getattr/setxattr
- Delete/destroy
- Unlink/rmdir

这些操作将由可变数量的并发线程运行，对用户指定的目录进行测试。可以所有线程都执行单个目录（dir_count = 1），也可以执行各自的私有目录（dir_count=x thrlo=x thrhi=x）。

mdd 实例是直接驱动的。该脚本会根据需要自动加载 obdecho 模块、创建 echo_client 实例。

该脚本还可以通过设置大于零的 stripe_count 来创建 OST 对象。

执行运行：

1. 启动 Lustre MDT.

Lustre MDT 必须在待测试的 MDS 节点上挂载。

2. 启动 Lustre OSTs（可选，只在使用 OST 对象测试时需要此步骤）

Lustre OSTs 必须在 OSS 节点上挂载。

3. 运行 `mds-survey` 脚本。

脚本必须根据测试组件和工作文件保存位置进行自定义。自定义变量如下：

- `thrlo` - 启动测试的线程，如果比 `dir_count`少则此变量可省略。
- `thrhi` - 测试的最多线程数。
- `targets` - MDT 实例。
- `file_count` - 每个线程测试的文件数。
- `dir_count` - 测试的总目录数。必须小于或等于 `thrhi`。
- `stripe_count`- OST 对象上的条带数。
- `tests_str` - 测试操作。至少必须包含"create" 和"destroy"。
- `start_number` - 为防止名称冲突的线程基数。
- `layer` - 待测试的 MDS 堆栈层

在没有创建 OST 对象的情况下运行：

在 OST 未挂载的情况下设置 Lustre MDS。随后引用 `mds-survey` 脚本。

```
$ thrhi=64 file_count=200000 sh mds-survey
```

创建 OST 对象并运行：

在至少挂载了一个 OST 的情况下设置 Lustre MDS。随后引用 `mds-survey` 脚本，使用 `stripe_count` 参数。

```
$ thrhi=64 file_count=200000 stripe_count=2 sh mds-survey
```

注意：可以使用目标变量指定特定的 MDT 实例。

```
$ targets=lustre-MDT0000 thrhi=64 file_count=200000
stripe_count=2 sh mds-survey
```

33.5.1. 输出文件

`mds-survey` 脚本运行时，它会创建大量工作文件和两个结果文件。所有文件都以变量 `${rslt}`定义的值作为前缀。

文件	说明
`${rslt}.summary`	和 stdout 一样
`${rslt}.script_*`	每个主机一个测试脚本文件

文件	说明
${rslt}.detail_tmp*	每个 OST 一个结果文件
${rslt}.detail	收集结果文件进行事后验证

mds-survey 脚本遍历执行指定测试的所有线程和对象，并检查所有测试过程是否已成功完成。

注意

mds-survey 如果脚本异常终止或遇到不可恢复的错误，则可能无法正常清除脚本。在这种情况下，可能需要手动清除，包括终止任何正在运行的lctl实例（本地或远程），删除脚本创建的echo_client实例以及卸载obdecho。

33.5.2. 脚本输出

.summary 文件及 mds-survey 脚本的 stdout 包含类似以下内容:

```
1 mdt 1 file 100000 dir 4 thr 4 create 5652.05 [ 999.01,46940.48] destroy
    5797.79 [ 0.00,52951.55]
```

其中:

参数及值	说明
mdt 1	被测试的 MDT 总数
file 100000	每个线程操作的文件总数
dir 4	目录总数
thr 4	所有目录上的线程总数
create, destroy	测试名。如果指定了更多的测试名，它们都将在同一行显示。
565.05	所有 MDT 上的总带宽（由操作总数除以所花时间得到）
[64, 82[999.01,46940.48]	每个 MDT 上可见的最低和最高的即时操作

注意

如果脚本输出含有"ERROR"，这通常意味着在运行期间存在问题，如 MDT 或 OST 上的空间不足。更多有关调试的详细信息可在$ {rslt} .detail文件中找到。

33.6. 收集应用程序分析信息 (stats-collect)

stats-collect 实用程序包含以下用于从 Lustre 客户端和服务器收集应用程序分析信息的脚本：

- lstat.sh -在每个配置文件节点上运行的单个节点的脚本。
- gather_stats_everywhere.sh - 收集统计信息的脚本。
- config.sh - 包含自定义配置描述的脚本。

stats-collect实用程序需要：

- 在你的集群上安装和设置 Lustre 软件。
- 对这些节点的 SSH 和 SCP 免密访问。

33.6.1. stats-collect

stats-collect 通过在config.sh脚本中包含性能分析配置变量来进行配置。每个配置变量都采用以下格式，其中 0 表示仅在脚本启动和停止时才收集统计信息，而 n 表示要收集统计信息的时间间隔（以秒为单位）：

```
1 statistic_INTERVAL=0|n
```

所收集的统计信息包括：

- VMSTAT - 内存和 CPU 使用率以及总读取/写入操作
- SERVICE - Lustre OST 和 MDT RPC 服务统计信息
- BRW - OST 批量读写统计信息 (brw_stats)
- SDIO - SCSI 磁盘 IO 统计信息 (sd_iostats)
- MBALLOC - ldiskfs 块分配统计信息
- IO - Lustre 目标操作统计信息
- JBD - ldiskfs 日志信息
- CLIENT - Lustre OSC 请求信息

所收集的分析信息包括：
开始收集 config.sh 脚本中指定的每个节点的统计信息。

1. 通过输入以下命令启动每个节点上的收集配置文件守护进程：

```
sh gather_stats_everywhere.sh config.sh start
```

2. 运行测试。

3. 停止在每个节点上收集统计信息，清理临时文件并创建一个分析压缩包。

```
sh gather_stats_everywhere.sh config.sh stop log_name.tgz
```

指定了 log_name.tgz时，将创建分析概要压缩包*/tmp/log_name*.tgz.

4. 分析收集的统计信息并为指定的分析概要数据创建一个 csv 压缩包。

```
sh gather_stats_everywhere.sh config.sh analyse
log_tarball.tgz csv
```

第三十四章 Lustre 文件系统调试

34.1. 优化服务线程数量

一个 OSS 最少可以有 2 个服务线程，最多可以有 512 个服务线程。服务线程数与每个 OSS 节点上有多少 RAM 和多少个 CPU 有关，可通过（1 个线程/128MB * num_cpus）来计算。如果 OSS 节点上的负载很高，则会启动新的服务线程以并发处理更多请求，最多为线程的初始数量的 4 倍（最大为 512）。对于 2GB 2-CPU 系统，默认线程数为 32，最大线程数为 128。

在以下情况中，增加线程池的大小可能会有所帮助：

- 多个 OST 从单个 OSS 中导出
- 后端存储正在同步运行
- 由于缓慢的存储，I/O 完成时间过长

在下列情况中，减小线程池的大小可能会有所帮助：

- 客户存储容量过载
- 有很多" 缓慢" 的 I/O 或类似的消息

增加 I/O 线程数允许内核和存储将多个写入聚合在一起以获得更高效的磁盘 I/O。OSS 线程池是共享的，每个线程为内部 I/O 缓冲区分配大约 1.5 MB（即：最大 RPC 大小 +0.5 MB）的空间。

增加线程池大小时，必须考虑内存消耗情况。大量的搜索工作和专门等待 I/O 的 OST 线程导致驱动器在性能下降之前只能维持一定数量的 I/O 并行操作。在这种情况下，一种明智的做法是通过减少 OST 线程的数量来减少负载。

确定 OSS 线程的最佳数量需要反复的试验。其值随不同的配置而变化，受到每个 OSS 上的 OST 数量，磁盘数和磁盘速度，RAID 配置以及可用的 RAM 等因素的影响。一开始，您可以将该线程数设置为节点上实际磁盘轴的数量。如果使用 RAID，则需要减去未用于实际数据的死磁盘轴数（例如，RAID5 的 N 个轴中的 1 个，RAID6 的 N 个

轴中的 2 个），并监视常规工作负载期间客户端的性能。如果性能下降，请增加线程数并查看其工作情况，直到性能再次下降或达到令人满意的成都。

注意

如果线程太多，单个 I/O 请求的延迟可能会变得非常高，应该避免这种情况。请使用上述方法来永久地设置所需的最大线程数。

34.1.1. 指定 OSS 服务线程数

在 OSS 节点上模块加载时可通过 `oss_num_threads` 参数指定 OST 服务线程的数量：

```
1 options ost oss_num_threads={N}
```

启动后，OSS 的最大和最小线程数可通过 `{service}.thread_{min,max,started}` 调节，在运行时更改值：

```
1 lctl {get,set}_param {service}.thread_{min,max,started}
```

这和在 MDS 绑定线程的工作方式类似。

- `oss_cpts=[EXPRESSION]` －绑定默认 OSS 服务至由`[EXPRESSION]`定义的 CPTs。
- `oss_io_cpts=[EXPRESSION]` －绑定默认 OSS I/O 服务至由`[EXPRESSION]`定义的 CPTs。

34.1.2. 指定 MDS 服务线程数

在 MDS 节点上模块加载时可通过 `mds_num_threads` 参数指定 MDS 服务线程的数量：

```
1 options mds mds_num_threads={N}
```

启动后，MDS 的最大和最小线程数可通过 `{service}.thread_{min,max,started}` 调节，在运行时更改值：

```
1 lctl {get,set}_param {service}.thread_{min,max,started}
```

启动的 MDS 服务线程数取决于系统大小和服务器上的负载，默认最大值为 64。线程的最大潜在数（`MDS_MAX_THREADS`）为 1024。

注意

挂载时，每个 CPT 每个服务启动两个 OSS 和 MDS 线程，根据服务器负载来动态增加运行的服务线程数量。设置`* _num_threads`参数将立即为该服务启动指定数量的线程，同时禁用线程自动创建。（在 **Lustre 2.3** 中引入）

Lustre 2.3 中引入了新的参数，为管理员提供了更多的控制。

- mds_rdpg_num_threads－控制提供读取页服务的线程数。读取页服务用于处理文件关闭和 readdir 操作。

34.2. 绑定 MDS 服务线程到 CPU 分区

在 Lustre 2.3 版中引入的 Node Affinity（节点关联性），可以将 MDS 线程绑定到特定的 CPU 分区（CPT），以提高 CPU 高速缓存使用率和内存局部性。将自动选择 CPT 数和 CPU 核心绑定的默认值，以便为给定数量的 CPU 提供良好的整体性能。管理员也可更改这些设置。有关指定 CPU 内核到 CPT 的映射的详细信息，请参见本章第 4 节"libcfs 调试"。

- mds_num_cpts=[EXPRESSION] 绑定默认 MDS 服务线程至由[EXPRESSION]定义的 CPTs。如，mds_num_cpts=[0-3] 将绑定 MDS 服务线程至CPT[0,1,2,3]。
- mds_rdpg_num_cpts=[EXPRESSION] 绑定读取页服务线程至由[EXPRESSION]定义的 CPTs。读取页服务负责处理文件关闭操作及 readdir 请求。如，mds_rdpg_num_cpts=[4]将绑定读取页服务线程至 CPT4。

设置参数必须在文件 /etc/modprobe.d/lustre.conf中载入模块前。如:
options lnet networks=tcp0(eth0)options mdt mds_num_cpts=[0]

34.3. LNet 参数调试

本节主要介绍 LNet 可调参数。在某些系统上可能需要使用这些参数来提高性能。

34.3.1. 发送和接收缓冲区大小

内核在网络上分配发送和接收信息的缓冲区。
使用ksocklnd 分开设置用于发送和接收信息的缓冲区的参数。

```
1 options ksocklnd tx_buffer_size=0 rx_buffer_size=0
```

如果这些参数保留默认值（0），系统会自动调整发送和接收缓冲区大小。几乎在所有情况下，此默认设置会产生最佳性能。如果您不是网络专家，请不要尝试调整这些参数。

34.3.2. 硬件中断 (enable_irq_affinity)

网络适配器生成的硬件中断可能由系统中的任一 CPU 进行处理。在某些情况下，我们希望将网络流量保持在单个 CPU 本地，以便保持处理器缓存温度并减少环境切换

的影响。这特别有利于具有多个网络接口尤其是接口数量等于 CPU 数量时的 SMP 系统。启用enable_irq_affinity参数，请输入：

```
1 options ksocklnd enable_irq_affinity=1
```

在其它情况下，如果您运行在一个含单个快速接口（如 10Gb/s）和两个以上的 CPU 的 SMP 平台，则禁用该参数可能会提升性能：

```
1 options ksocklnd enable_irq_affinity=0
```

此参数默认为关闭。请通过测试更改此参数时的性能情况来进行调试。（在 **Lustre 2.3** 中引入）

34.3.3. 绑定针对 CPU 分区的网络接口

Lustre 2.3 及以上版本提供了高级网络接口控制。管理员可以将接口绑定到一个或多个 CPU 分区，通过 LNet 模块的选项进行指定。例如，o2ib0(ib0)[0,1] 确保了o2ib0的所有消息由在CPT0和CPT1上执行的 LND 线程处理；tcp1(eth0)[0]确保了tcp1的消息由CPT0上的线程处理。

34.3.4. 网络接口信用

网络接口（NI）信用在所有 CPU 分区（CPT）之间共享。例如，如果一台机器有四个 CPT 且 NI 信用值为 512，则每个分区有 128 个信用值。如果系统中存在大量 CPT，则 LNet 将检查并验证每个 CPT 的 NI 信用值，以确保每个 CPT 都有可用的信用值。如果一台机器有 16 个 CPT 且 NI 信用值为 256，则每个分区只有 16 个信用值，将可能会对性能产生负面影响。因此，LNet 会自动将信用值调整为8*peer_credits（默认情况下，peer_credits 为 8），因此每个分区都有 64 个信用值。

增加 credits/ peer_credits 数使得 LNet 能够将更多的飞行消息发送到特定的网络或对等节点并保持传输饱和，从而提高高延迟网络的性能（以消耗更多内存为代价）。

管理员可以使用ksoclnd或ko2iblnd修改 NI 信用值。在下面的例子中，TCP 连接的信用值被设置为 256。

```
1 ksocklnd credits=256
```

设置 IB 连接的信用值为 256：

```
1 ko2iblnd credits=256
```

注意

在 Lustre 2.3 及以上版本中，LNet 可能会重新验证 NI 积分，则管理员请求可能不会持续。

34.3.5. 路由器缓存区

当一个节点被设置为 LNet 路由器时，会分配三个缓存区：极小、小和大的缓存区。这些缓存区按 CPU 分区分配，用于缓存到达路由器待转发到下一跳的消息。三种不同大小的缓存区适应不同大小的消息。

如果消息可以放入极小缓冲区，那么使用极小的缓冲区；如果它不能放入极小的缓冲区但是可以放入小缓冲区，则使用小缓冲区；如果消息不适用于极小或小缓冲区，则使用大缓冲区。

路由器缓冲区由所有 CPU 分区共享。对于具有大量 CPT 的机器，可能需要手动指定路由器缓冲区号以获得最佳性能。路由器缓冲区数量过少可能会导致存放资源的 CPU 分区匮乏。

- `tiny_router_buffers`：用于信号和确认消息的零负荷缓冲区。
- `small_router_buffers`：用于简短消息的 4 KB 负荷缓冲区。
- `large_router_buffers`；最大负荷为 1 MB 的缓冲区（对应于 1 MB 的推荐 RPC 大小）

通常，默认路由器缓冲区设置下系统性能良好。因此，LNet 将自动将其设置为默认值以减少资源匮乏的可能性。路由器缓冲区的大小可以使用`large_router_buffers`参数修改进行更改。如，修改大缓冲区的大小：

```
1 lnet large_router_buffers=8192
```

注意

在 Lustre 2.3 及以上版本中，LNet 可能会重新验证路由器缓存区设置，则管理员请求可能不会持续。

34.3.6. 门户循环

门户循环定义了 LNet 应用的向上层传递事件和消息的策略。上层有 PLRPC 服务或 LNet 自检。

如果禁用了门户循环，则 LNet 将根据源 NID 的散列向 CPT 传递消息。因此，来自特定对等方的所有消息都将由相同的 CPT 处理。这可以减少 CPU 之间的数据流量。但是，对于某些工作负载，这种行为可能会导致整个 CPU 的负载失衡。

如果启用了门户循环，则 LNet 将对所有 CPT 中的传入事件进行循环。这可以在整个 CPU 上更好地平衡负载，但同时也可能导致 CPU 间的交互开销。

管理员可通过`echo value> /proc/sys/lnet/portal_rotor`更改当前策略。其中，`value`有以下四种选项：

- `OFF`

在所有传入请求上禁用门户循环。

- ON

在所有传入请求上启用门户循环。

- RR_RT

为路由消息启用门户循环。

- HASH_RT

默认值。路由消息将通过源 NID 的散列（而不是路由器的 NID）传递到上层。

34.3.7. LNet 对等节点健康状况

以下两个选项可用于帮助确认对等节点的健康状况：

- peer_timeout－活性查询发送给对等体节点的超时时间（以秒为单位）。例如，如果peer_timeout设置为 180 秒，则每隔 180 秒会向对等节点发送一次活性查询。此功能只在节点配置为 LNet 路由器时生效。

在路由环境中，peer_timeout功能应该始终处于打开状态。如果路由器检查程序已启用，则应在客户端和服务器上将该值设置为 0 以关闭该功能。

对于非路由环境来说，启用peer_timeout选项可提供有关健康状况的信息，如对等节点是否存活。客户端可以在发送消息时确认 MGS 或 OST 是否已启动。如果收到回复，则表明对方在线，否则将出现超时。

通常，peer_timeout应设置为不小于 LND 超时设置的值。

使用o2iblnd (IB) 驱动程序时，peer_timeout应至少为ko2iblnd 选项值的两倍。

- avoid_asym_router_failure－默认设置是 1。此时，客户端或服务器上运行的路由器检查程序会周期性地 ping 所有节点上 routes 参数设置中标识的 NID 所对应的路由器，以确定每个路由器接口的状态。

如果路由器的任一 NID 关闭，则我们认为该路由器处于关闭状态。例如，路由器 X 有三个 NID：Xnid1、Xnid2和Xnid3。客户端通过Xnid1连接到路由器。客户端启用了路由器检查程序。路由器检查器通过Xnid1定期向路由器发送一个 ping。路由器以每个 NID 的状态来回应 ping。例如，在这个例子中，响应消息为Xnid1=up，Xnid2=up，Xnid3=down。如果avoid_asym_router_failure==1且任一 NID 关

闭，则路由器处于关闭状态。我们认为该路由器 X 已关闭，不会再使用其路由消息。如果avoid_asym_router_failure==0，则将继续使用路由器 X 的路由消息。

在任何客户端或服务器上，以下路由器检查器参数必须设置为此选项对应的最大值：

- dead_router_check_interval
- live_router_check_interval
- router_ping_timeout

例如，dead_router_check_interval 参数在任何路由器上都应被设置为 MAX。（在 **Lustre 2.3** 中引入）

34.4. libcfs 调试

Lustre 2.3 通过 CPU 分区（CPT）引入了绑定服务线程，允许了系统管理员针对 Lustre 服务线程在哪些 CPU 内核上运行进行调试（在 OSS 服务、MDS 服务以及客户端上）。

CPT 有助于在 OSS 或 MDS 节点上为系统功能（如系统监视，HA heartbeat 或类似任务）预留一些核。在客户端上，可以把 Lustre RPC 服务线程限制在一小部分核中，从而避免干扰计算操作。这些核是直接连接到网络接口上的。

默认情况下，Lustre 软件将根据系统中 CPU 的数量自动生成 CPU 分区(CPT)。可以在 libcfs 模块上通过cpu_npartitions=NUMBER 设置明确的 CPT 数。cpu_npartitions的值必须是 1 到当前在线 CPU 数之间的整数。

在 Lustre 2.9 和更高版本中，默认情况下每个 NUMA 节点使用一个 CPT。在 Lustre 早期版本中，如果在线 CPU 核数量为四个或更少，则默认情况下使用单个 CPT，可根据 CPU 核数量创建额外的 CPT，通常每个 CPT 有 4-8 个核。

cpu_npartitions=1将禁用大部分 SMP 节点的 Affinity 功能。

34.4.1. CPU 分区（字符串模式）

可以使用字符串模式表示法来描述 CPU 分区。例如，cpu_pattern=N表示系统中每个 NUMA 节点有一个 CPT，每个 CPT 映射该 NUMA 节点的所有 CPU 核。

也可明确指定 CPU 核与 CPT 之间的映射，例如：

```
1 cpu_pattern="0[2,4,6] 1[3,5,7]
```

创建两个 CPT。其中，CPT0 包含核 2、4、6，CPT1 包含核 3、5、7。CPU 核 0 和 1 不会用于 Lustre 服务线程，但可以用于节点服务（如系统监视、HA heartbeat 线程等）。可使用numactl(8)或其他应用程序指定方法在用户空间中完成非 Lustre 服务与这些 CPU 核的绑定（这部分内容超出了本文的范围）。

```
cpu_pattern="N 0[0-3] 1[4-7]
```

创建两个 CPT。其中，CPT0 包含 NUMA 节点 [0-3] 上的所有 CPU，CPT1 包含 NUMA 节点 [4-7] 上的所有 CPU。

CPU 分区的当前配置可以通过`lctl get_parm cpu_partition_table`读取。例如，只有单个 CPT 包含全部四核 CPU 的简单四核系统：

```
$ lctl get_param cpu_partition_table cpu_partition_table=0  : 0 1 2 3
```

以下为更大的 NUMA 系统，有 4 个 CPT、每个 CPT 有 12 个 CPU 核：

```
$ lctl get_param cpu_partition_table
cpu_partition_table=
0   : 0 1 2 3 4 5 6 7 8 9 10 11
1   : 12 13 14 15 16 17 18 19 20 21 22 23
2   : 24 25 26 27 28 29 30 31 32 33 34 35
3   : 36 37 38 39 40 41 42 43 44 45 46 47
```

34.5. LND 调试

LND 调试允许指定每个 CPU 分区的线程数量。管理员可以使用`nscheds`参数为`ko2iblnd`和`ksocklnd`设置线程，从而调整每个分区的线程数量而不是 LND 上的线程总数。

注意

`ko2iblnd`和`ksocklnd`的默认线程数是自动设置的，该默认值在许多典型场景中保证了良好的运行，对内核数量多或少的系统均适用。

34.5.1. ko2iblnd 调试

下表对用于调试的`ko2iblnd`模块参数进行了概述：

模块参数	默认值	说明
`service`	`987`	（在 RDMA_PS_TCP 内的）服务号
`cksum`	`0`	设置为非零启用消息 (非 RDMA) 校验和。
`timeout`	`50`	超时时间（以秒为单位）。
`nscheds`	`0`	每个调度程序池中的线程数（每个 CPT）。0 表示该值为核的数量。
`conns_per_peer`	`4 (OmniPath),`	每个对等节点的连接数。消息通过

模块参数	默认值	说明
	1 (Everything else)	连接池循环发送。OmniPath 提供了显著的改进。(**Lustre 2.10** 中引入)
ntx	512	启动时为每个池分配的消息描述符的数量。在运行时会增长。由所有 CPT 共享。
credits	256	网络上的并发发送数量。
peer_credits	8	并发发送至 1 个对等点的数量。与 IB 队列大小相关（受限于 IB 队列大小)。
peer_credits_hiw	0	急于返回信用值时。
peer_buffer_credits	0	每个对等路由器的缓冲信用值。
peer_timeout	180	失去活性消息到宣布对等节点死亡之间的秒数（小于或等于 0 表示禁用)。
ipif_name	ib0	IPoIB 端口名称。
retry_count	5	未收到 ACK 时重新传输。
rnr_retry_count	6	RNR 重传。
keepalive	100	发送 keepalive 前的空闲秒数。
ib_mtu	0	IB MTU 256/512/1024/2048/4096.
concurrent_sends	0	发送工作队列的大小。0 则表示该队列大小和 map_on_demand 或peer_credits相同。
map_on_demand	0 (pre-4.8 Linux)1 (4.8 Linux	为连接预留的片段数量。如果为零，则使用全局内存（可能存在安全问题)。如果非零，则使

模块参数	默认值	说明
	onward) 32 (OmniPath)	用 FMR 或 FastReg 进行内存注册。该值需要在两个对等节点之间达成一致。
fmr_pool_size	512	每个 CPT 上的 fmr 池大小（>= ntx / 4）。在运行时会增长。
fmr_flush_trigger	384	触发池刷新的 dirty FMR 号。
fmr_cache	1	设置为非零来实现 FMR 缓存。
dev_failover	0	用于绑定的 HCA 故障切换（0 表示 OFF，1 表示 ON，其他值保留）
require_privileged_port	0	接受连接时的 require 特权端口。
use_privileged_port	1	初始化连接的 use 特权端口
wrq_sge	2	每个请求 scatter/gather 元素组的数量。用于处理可能会消耗请求工作数量两倍的碎片。（**Lustre 2.10** 中引入）

34.6. 网络请求调度程序 (NRS) 调试

网络请求调度程序（NRS）允许管理员通过应用不同策略影响服务器处理 RPC 的顺序（基于每个 PTLRPC 服务）。这样做的目的是为了更好的性能，一些未来的策略还可能提供离散的性能特征。

PTLRPC 服务的 NRS 策略状态可通过{service} .nrs_policies来读取和设置。读取 PTLRPC 服务的 NRS 策略状态，请运行：

```
1 lctl get_param {service}.nrs_policies
```

例如，读取ost_io 服务的 NRS 策略状态：

```
1 $ lctl get_param ost.OSS.ost_io.nrs_policies
2 ost.OSS.ost_io.nrs_policies=
3
```

```
regular_requests:
  - name: fifo
    state: started
    fallback: yes
    queued: 0
    active: 0

  - name: crrn
    state: stopped
    fallback: no
    queued: 0
    active: 0

  - name: orr
    state: stopped
    fallback: no
    queued: 0
    active: 0

  - name: trr
    state: started
    fallback: no
    queued: 2420
    active: 268

  - name: tbf
    state: stopped
    fallback: no
    queued: 0
    active: 0

  - name: delay
    state: stopped
    fallback: no
    queued: 0
    active: 0
```

```
40
41 high_priority_requests:
42   - name: fifo
43     state: started
44     fallback: yes
45     queued: 0
46     active: 0
47
48   - name: crrn
49     state: stopped
50     fallback: no
51     queued: 0
52     active: 0
53
54   - name: orr
55     state: stopped
56     fallback: no
57     queued: 0
58     active: 0
59
60   - name: trr
61     state: stopped
62     fallback: no
63     queued: 0
64     active: 0
65
66   - name: tbf
67     state: stopped
68     fallback: no
69     queued: 0
70     active: 0
71
72   - name: delay
73     state: stopped
74     fallback: no
75     queued: 0
```

```
76     active: 0
```

NRS 策略状态显示在一个或两个部分中，取决于所查询的 PTLRPC 服务。第一部分为regular_requests，可用于所有 PTLRPC 服务。第二部分为high_priority_requests，为可选部分。这是因为一些 PTLRPC 服务能够将某些类型的 RPC 视为较高优先级的 RPC，使它们能优先被服务器处理。对于不支持高优先级 RPC 的 PTLRPC 服务只能看到regular_requests部分。

每个 PTLRPC 服务上的每个 NRS 策略都有一个单独的实例，用于处理常规或高优先级的 RPC（如果该服务支持高优先级的 RPC)。对于每个策略实例，将显示以下字段：

字段	说明
name	策略名
state	策略状态。为 invalid、stopping、stopped、starting、started。策略完全启动为 started 状态。
fallback	是否为回退策略。回退策略用于处理其他已启用策略无法处理的 RPC 或不支持的 RPC。值为 no 或yes当前只有 FIFO 策略可以作为回退策略。
queued	策略拥有的等待服务的 RPC 数量。
active	策略正在处理的 RPC 数量

在 PTLRPC 服务上启用 NRS 策略，运行：

```
1 lctl set_param {service}.nrs_policies=
2 policy_name
```

这将为给定服务上的常规和高优先级 RPC（如果该 PLRPC 服务支持高优先级 RPC）启用策略名为 policy_name 的策略。例如，为 ldlm_cbd 服务启用 CRR-N NRS 策略，请运行：

```
1 $ lctl set_param ldlm.services.ldlm_cbd.nrs_policies=crrn
2 ldlm.services.ldlm_cbd.nrs_policies=crrn
```

对于支持高优先级 RPC 的 PTLRPC 服务，您还可以通过运行以下命令并提供可选选项 *reg|hptoken*，以便启用 NRS 策略仅用来处理给定 PTLRPC 服务上的常规或高优先级 RPC：

```
1 lctl set_param {service}.nrs_policies="
2 policy_name
3 reg|hp"
```

例如，要启用 TRR 策略用来处理ost_io服务上的常规 RPC（非高优先级 RPC），请运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_policies="trr reg"
ost.OSS.ost_io.nrs_policies="trr reg"
```

注意

启用 NRS 策略时，策略名称必须使用小写字母，否则将造成操作失败并显示错误消息。

34.6.1. 先进先出 (FIFO) 策略

先进先出 (FIFO) 策略按照从 LNet 层到达的相同顺序处理服务中的 RPC，不进行任何特殊处理来更改 RPC 处理流。FIFO 是所有 PTLRPC 服务上所有类型 RPC 的默认策略，且无论其他策略状态如何都始终处于启用状态。把 FIFO 策略作为备份策略，以防已更复杂的策略处理 RPC 失败或不支持给定类型的 RPC。

FIFO 策略没有用于调整其行为的可调参数。

34.6.2. 基于 NID 的客户端循环 (CRR-N) 策略

通过基于 NID 的客户端轮询（CRR-N）策略对所有类型 RPC 执行批量循环调度，每个批次由源自相同客户端节点的 RPC（由其 NID 标识）组成。CRR-N 旨在提高集群间的资源利用率，并通过在所有客户端之间更均匀地分配可用带宽来缩短某些情况下的作业完成时间。

可以在所有类型的 PTLRPC 服务上启用 CRR-N 策略。以下是可用于调整其行为的可调参数：

- `{service}.nrs_crrn_quantum`

`{service}.nrs_crrn_quantum` 用于确定 RPC 的最大批处理大小；度量单位是 RPC 的数量。读取 CRR-N 策略允许的最大批处理大小，请运行：

```
lctl get_param {service}.nrs_crrn_quantum
```

例如，在ost_io服务上读取 CRR-N 策略允许的最大批处理大小，运行：

```
$ lctl get_param ost.OSS.ost_io.nrs_crrn_quantum
ost.OSS.ost_io.nrs_crrn_quantum=reg_quantum:16
hp_quantum:8
```

如上所示，您可以看到常规 (`reg_quantum`) 和高优先级 (`hp_quantum`) RPCs 有两个独立的最大批处理大小。

为指定服务设置 CRR-N 策略允许的最大批处理大小，运行：

```
lctl set_param {service}.nrs_crrn_quantum=
1-65535
```

这将为常规和高优先级 RPC（如果 PLRPC 服务支持高优先级 RPC）设置给定服务上允许的最大批处理大小。

例如，将ldlm_canceld服务上允许的最大批处理大小设置为 16，请运行：

```
$ lctl set_param ldlm.services.ldlm_canceld.nrs_crrn_quantum=16
ldlm.services.ldlm_canceld.nrs_crrn_quantum=16
```

对于支持高优先级 RPC 的 PTLRPC 服务，您也可以为常规和高优先级 RPC 指定不同的最大批处理大小：

```
$ lctl set_param {service}.nrs_crrn_quantum=
reg_quantum|hp_quantum:
1-65535"
```

例如，在ldlm_canceld服务上将最大高优先级 RPC 批处理大小设置为 32：

```
$ lctl set_param
    ldlm.services.ldlm_canceld.nrs_crrn_quantum="hp_quantum:32"
ldlm.services.ldlm_canceld.nrs_crrn_quantum=hp_quantum:32
```

通过使用最后一种方法，您还可以在单个命令中将常规和高优先级 RPC 批处理最大大小设置为不同的值。

34.6.3. 基于对象的循环（ORR）策略

基于对象的循环（ORR）策略对批量读写（brw）RPC 的批量循环调度，每个批次由属于相同后端文件系统对象的 RPC（由 OST FID 标识）组成。

ORR 策略仅适用于 ost_io 服务。RPC 批处理可能包含批量读取和批量写入 RPC。根据每个 RPC 的文件偏移量或物理磁盘偏移量（仅适用于批量读取 RPC），每个批处理中的 RPC 按升序方式排序。

ORR 策略旨在通过顺序读取批量 RPC（也可能包括批量写入 RPC）来增加某些情况下的批读取吞吐量，从而最大限度地减少昂贵的磁盘查找操作。任何资源利用率的改善或更好地利用 RPC 间的相对位置都可能有助于提升性能。

ORR 策略有以下可用于调整其行为的可调参数：

- ost.OSS.ost_io.nrs_orr_quantum

ost.OSS.ost_io.nrs_orr_quantum 用于确定 RPC 的最大批处理大小；度量单位是 RPC 的数量。读取 ORR 策略允许的最大批处理大小，请运行：

```
$ lctl get_param ost.OSS.ost_io.nrs_orr_quantum
ost.OSS.ost_io.nrs_orr_quantum=reg_quantum:256
hp_quantum:16
```

如上所示，您可以看到常规 (reg_quantum) 和高优先级 (hp_quantum) RPCs 有两个独立的最大批处理大小。

设置 ORR 策略允许的最大批处理大小，运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_quantum=
1-65535
```

这将为常规和高优先级 RPC 所允许的最大批处理大小设置指定的大小。

还可以为常规和高优先级 RPC 指定不同的最大允许批处理大小，请运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_quantum=
reg_quantum|hp_quantum:
1-65535
```

例如，将常规 RPC 的最大批处理大小设置为 128，请运行

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_quantum=reg_quantum:128
ost.OSS.ost_io.nrs_orr_quantum=reg_quantum:128
```

通过使用最后一种方法，您还可以在单个命令中将常规和高优先级 RPC 批处理最大大小设置为不同的值。

- ost.OSS.ost_io.nrs_orr_offset_type

ost.OSS.ost_io.nrs_orr_offset_type 用于确定 ORR 策略是基于逻辑文件偏移量还是物理磁盘偏移量对每批次 RPC 进行排序。读取 ORR 策略的偏移类型，请运行：

```
$ lctl get_param ost.OSS.ost_io.nrs_orr_offset_type
ost.OSS.ost_io.nrs_orr_offset_type=reg_offset_type:physical
hp_offset_type:logical
```

常规 (reg_offset_type) 和高优先级 (hp_offset_type) RPC 有单独的偏移类型。

设置 ORR 策略的偏移类型，运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_offset_type=
physical|logical
```

这将设置常规和高优先级 RPC 的偏移类型为指定值。

您还可以运行以下命令为常规和高优先级 RPC 指定不同的偏移类型：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_offset_type=
reg_offset_type|hp_offset_type:
physical|logical
```

例如，将高优先级 RPC 的偏移类型设置为物理磁盘偏移量，运行：

```
$ lctl set_param
    ost.OSS.ost_io.nrs_orr_offset_type=hp_offset_type:physical
ost.OSS.ost_io.nrs_orr_offset_type=hp_offset_type:physical
```

通过使用最后一种方法，您还可以在单个命令中将常规和高优先级 RPC 批处理最大大小设置为不同的值。

注意

无论此可调参数的值为什么，只有逻辑偏移量可以用于批量写入 RPC 的排序。

- `ost.OSS.ost_io.nrs_orr_supported`

`ost.OSS.ost_io.nrs_orr_supported` 用于确定 ORR 策略处理的 RPC 类型，读取 ORR 策略支持的 RPC 类型，运行：

```
$ lctl get_param ost.OSS.ost_io.nrs_orr_supported
ost.OSS.ost_io.nrs_orr_supported=reg_supported:reads
hp_supported=reads_and_writes
```

如上所示，您可以看到常规 (`reg_quantum`) 和高优先级 (`hp_quantum`) RPCs 有不同的支持的 RPC 类型。

为 ORR 策略设置支持的 RPC 类型，运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_supported=
reads|writes|reads_and_writes
```

这将设置 ORR 策略支持的常规和高优先级 RPC 类型为指定值。

您还可以运行以下命令为常规和高优先级 RPC 指定不同的支持类型：

```
$ lctl set_param ost.OSS.ost_io.nrs_orr_supported=
reg_supported|hp_supported:
reads|writes|reads_and_writes
```

例如，为常规请求将 RPC 支持类型设置为批量读和批量写：

```
$ lctl set_param
ost.OSS.ost_io.nrs_orr_supported=reg_supported:reads_and_writes
ost.OSS.ost_io.nrs_orr_supported=reg_supported:reads_and_writes
```

通过使用最后一种方法，您还可以在单个命令中将常规和高优先级 RPC 的支持类型设置为不同的值。

34.6.4. 基于目标的循环 (TRR) 策略

基于目标的循环（TRR）策略对 brw RPC 执行批量循环调度，每个批次由属于相同 OST 的 RPC（由 OST 索引标识）构成。

除了使用 brw RPC 的目标 OST 索引而不是后端 fs 对象的 OST FID 来确定 RPC 调度顺序以外，TRR 策略与基于对象的循环（ORR）策略相同。TRR 策略和 ORR 策略的实施效果相同，它使用以下可调参数来调整其行为:

- `ost.OSS.ost_io.nrs_trr_quantum`

与 ORR 策略中的 `ost.OSS.ost_io.nrs_orr_quantum` 参数的目标和用法完全相同。

- `ost.OSS.ost_io.nrs_trr_offset_type`

与 ORR 策略中的 `ost.OSS.ost_io.nrs_orr_offset_type` 参数的目标和用法完全相同。

- `ost.OSS.ost_io.nrs_trr_supported`

与 ORR 策略中的 `ost.OSS.ost_io.nrs_orr_supported` 参数的目标和用法完全相同。

（在 **Lustre 2.6** 中引入）

34.6.5. 令牌桶过滤器 (TBF) 策略

令牌桶过滤器 (TBF) 策略通过强制限制客户端或作业的 RPC 速率而使 Lustre 服务达到一定的 QoS（服务质量）。

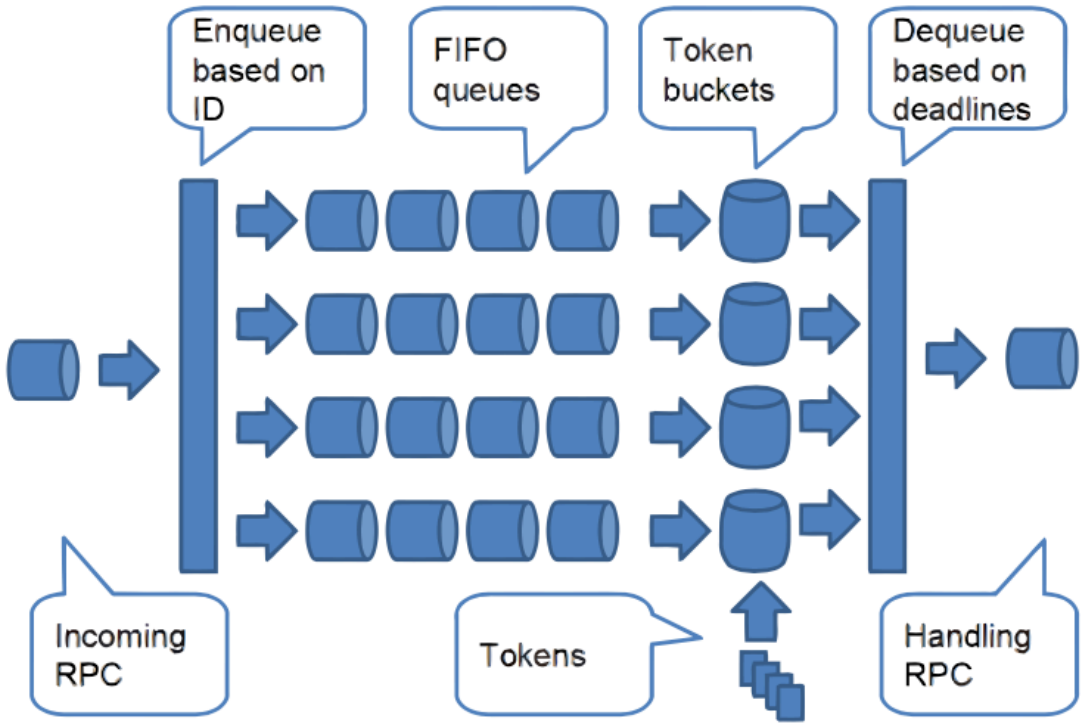


图 28: Internal stucture of TBF policy

图 34.1 TBF 策略的内部结构

当 RPC 请求到达时，TBF 策略根据它的分类将它放到一个等待队列中。根据 TBF 配置，RPC 请求的分类可以基于 RPC 的 NID 或 JobID。TBF 策略在系统中需要维护多个队列，RPC 请求分类的每个类别有一个队列。这些请求在处理之前等待 FIFO 队列中的令牌，从而使 RPC 速率保持在限制之下。

Lustre 服务太忙无法及时处理所有请求时，所有队列的处理速率都不会达到指定值。但除了一些 RPC 速率比配置慢以外，并无任何坏处。在这种情况下，速率较高的队列比速率较低的队列具有优势。

管理队列的 RPC 速率，我们不需要手动设置每个队列的速率，而是通过定义 TBF 策略匹配规则来确定 RPC 速率限制。所有定义的规则存储在有序列表中。每个新创建的队列将遍历规则列表并将第一个匹配的规则作为其规则，从而确定 RPC 令牌速率。规则可在运行时添加到列表或从列表中删除。每当规则列表发生更改时，队列将更新其匹配的规则。

34.6.5.1. 启用 TBF 策略 命令:

```
1 lctl set_param ost.OSS.ost_io.nrs_policies="tbf <policy>"
```

目前，RPC 可以根据其 NID、JOBID、OPCode 或 UID/GID 来进行分类。启用 TBF 策略时，您可以指定其中一种方式，或使用"tbf" 允许所有方式并执行细粒度 RPC 请求分类。

示例：

```
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf"
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf nid"
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf jobid"
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf opcode"
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf uid"
$ lctl set_param ost.OSS.ost_io.nrs_policies="tbf gid"
```

34.6.5.2. 启用 TBF 规则 TBF 规则在 `ost.OSS.ost_io.nrs_tbf_rule`参数中定义。

命令：

```
lctl set_param x.x.x.nrs_tbf_rule=
"[reg|hp] start rule_name arguments..."
```

其中，'*rule_name*' 是一个最多 15 个字符的字符串，用于标识 TBF 策略规则的名称，可以使用字母、数字字符和下划线（例如：``test_rule_A1"）。

'*arguments*' 是一个字符串，用于根据不同的类型指定详细的规则。

以下是 TBF 策略的不同类型：

- **基于 NID 的 TBF 策略**

命令：:

```
lctl set_param x.x.x.nrs_tbf_rule=
"[reg|hp] start rule_name nid={nidlist} rate=rate"
```

'*nidlist*' 的格式与配置 LNET 路由相同。'*rate*' 为该规则的 RPC 速率（上限）。

示例：

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start other_clients nid={192.168.*.*@tcp} rate=50"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start computes nid={192.168.1.[2-128]@tcp} rate=500"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start loginnode nid={192.168.1.1@tcp} rate=100"
```

在这个例子中，计算节点的 RPC 请求处理速率最大时是登录节点 RPC 请求处理速率的 5 倍。`ost.OSS.ost_io.nrs_tbf_rule` 的输出类似于：

```
lctl get_param ost.OSS.ost_io.nrs_tbf_rule
ost.OSS.ost_io.nrs_tbf_rule=
```

```
regular_requests:
CPT 0:
loginnode {192.168.1.1@tcp} 100, ref 0
computes {192.168.1.[2-128]@tcp} 500, ref 0
other_clients {192.168.*.*@tcp} 50, ref 0
default {*} 10000, ref 0
high_priority_requests:
CPT 0:
loginnode {192.168.1.1@tcp} 100, ref 0
computes {192.168.1.[2-128]@tcp} 500, ref 0
other_clients {192.168.*.*@tcp} 50, ref 0
default {*} 10000, ref 0
```

规则也可使用 reg 和 hp格式进行描述：

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"reg start loginnode nid={192.168.1.1@tcp} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"hp start loginnode nid={192.168.1.1@tcp} rate=100"
```

- **基于 JobID 的 TBF 策略**

  命令：

```
lctl set_param x.x.x.nrs_tbf_rule=
"[reg|hp] start rule_name jobid={jobid_list} rate=rate"
```

支持的 Wildcard 显示在 {*jobid_list*} 中。

示例：

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start iozone_user jobid={iozone.500} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start dd_user jobid={dd.*} rate=50"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start user1 jobid={*.600} rate=10"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start user2 jobid={io*.10* *.500} rate=200"
```

规则也可使用 reg 和 hp格式进行描述：

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"hp start iozone_user1 jobid={iozone.500} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"reg start iozone_user1 jobid={iozone.500} rate=100"
```

- **基于 Opcode 的 TBF 策略**

  命令:

```
$ lctl set_param x.x.x.nrs_tbf_rule=
"[reg|hp] start rule_name opcode={opcode_list} rate=rate"
```

  示例:

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start user1 opcode={ost_read} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start iozone_user1 opcode={ost_read ost_write} rate=200"
```

  规则也可使用 reg 和 hp格式进行描述:

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"hp start iozone_user1 opcode={ost_read} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"reg start iozone_user1 opcode={ost_read} rate=100"
```

- **基于 UID/GID 的 TBF 策略**

  命令:

```
$ lctl set_param ost.OSS.*.nrs_tbf_rule=\
"[reg] [hp] start rule_name uid={uid} rate=rate"
$ lctl set_param ost.OSS.*.nrs_tbf_rule=\
"[reg] [hp] start rule_name gid={gid} rate=rate"
```

  示例:

  限制 uid 500 的 RPC 请求速率:

  `$ lctl set_param ost.OSS.*.nrs_tbf_rule=\ "start tbf_name uid={500} rate=100"`

  限制 gid 500 的 RPC 请求速率:

```
$ lctl set_param ost.OSS.*.nrs_tbf_rule=\
"start tbf_name gid={500} rate=100"
```

您也可以使用以下的规则控制 MDS 上的请求。

在 MDS 上启动 tbf uid QoS：

```
$ lctl set_param mds.MDS.*.nrs_policies="tbf uid"
```

限制 uid 500 的 RPC 请求速率：

```
$ lctl set_param mds.MDS.*.nrs_tbf_rule=\
"start tbf_name uid={500} rate=100"
```

- 策略合并

为支持具有复杂条件表达式的 TBF 规则，可以使用 TBF 分类器以更细粒度的方式对 RPC 进行分类。此功能支持不同类型之间的逻辑操作。其中，" &" 代表条件与，"," 代表条件或。

示例：

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start comp_rule opcode={ost_write}&jobid={dd.0},\
nid={192.168.1.[1-128]@tcp 0@lo} rate=100"
```

在这个例子中，那些 opcode 为 *ost_write* 且 jobid 为 *dd.0*，或 nid满足 *{192.168.1.[1-128]@tcp 0@lo}* 条件的 RPC 将以 100 req/sec 的速率进行处理。ost.OSS.ost_io.nrs_tbf_rule的输出类似于：

```
$ lctl get_param ost.OSS.ost_io.nrs_tbf_rule
ost.OSS.ost_io.nrs_tbf_rule=
regular_requests:
CPT 0:
comp_rule opcode={ost_write}&jobid={dd.0},nid={192.168.1.[1-128]@tcp 0@lo}
    100, ref 0
default * 10000, ref 0
CPT 1:
comp_rule opcode={ost_write}&jobid={dd.0},nid={192.168.1.[1-128]@tcp 0@lo}
    100, ref 0
default * 10000, ref 0
high_priority_requests:
CPT 0:
comp_rule opcode={ost_write}&jobid={dd.0},nid={192.168.1.[1-128]@tcp 0@lo}
    100, ref 0
default * 10000, ref 0
```

```
14 CPT 1:
15 comp_rule opcode={ost_write}&jobid={dd.0},nid={192.168.1.[1-128]@tcp 0@lo}
       100, ref 0
16 default * 10000, ref 0
```

示例：

```
1 $ lctl set_param ost.OSS.*.nrs_tbf_rule=\
2 "start tbf_name uid={500}&gid={500} rate=100"
```

在这个例子中，那些uid 为 500 且 gid 为 500 的 RPC 将以 100 req/sec 的速率进行处理。

34.6.5.3. 更改 TBF 规则 命令:

```
1 lctl set_param x.x.x.nrs_tbf_rule=
2 "[reg|hp] change rule_name rate=rate"
```

示例：

```
1 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
2 "change loginnode rate=200"
3 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
4 "reg change loginnode rate=200"
5 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
6 "hp change loginnode rate=200"
```

34.6.5.4. 停用 TBF 规则 命令:

```
1 lctl set_param x.x.x.nrs_tbf_rule="[reg|hp] stop
2 rule_name"
```

示例：

```
1 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule="stop loginnode"
2 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule="reg stop loginnode"
3 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule="hp stop loginnode"
```

34.6.5.5. 规则选项 为支持更灵活的规则，添加了以下选项：

- **将 TBF 规则重新排序**

默认情况下，新启用的规则优先于旧规则，但在使用"start" 命令插入新规则时同时指定参数"rank =", 可以更改规则的排序。此外，还可以通过"change" 命令更改规则的排序。

命令:

```
lctl set_param ost.OSS.ost_io.nrs_tbf_rule=
"start rule_name arguments... rank=obj_rule_name"
lctl set_param ost.OSS.ost_io.nrs_tbf_rule=
"change rule_name rate=rate rank=obj_rule_name"
```

通过指定已存在的规则'*obj_rule_name*', 新规则'*rule_name*' 可被移至该条规则'*obj_rule_name*' 之前。

示例:

```
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start computes nid={192.168.1.[2-128]@tcp} rate=500"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start user1 jobid={iozone.500 dd.500} rate=100"
$ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=\
"start iozone_user1 opcode={ost_read ost_write} rate=200 rank=computes"
```

在这个例子中，规则"iozone_user1" 被添加至规则"computes" 之前，顺序如下:

```
$ lctl get_param ost.OSS.ost_io.nrs_tbf_rule
ost.OSS.ost_io.nrs_tbf_rule=
regular_requests:
CPT 0:
user1 jobid={iozone.500 dd.500} 100, ref 0
iozone_user1 opcode={ost_read ost_write} 200, ref 0
computes nid={192.168.1.[2-128]@tcp} 500, ref 0
default * 10000, ref 0
CPT 1:
user1 jobid={iozone.500 dd.500} 100, ref 0
iozone_user1 opcode={ost_read ost_write} 200, ref 0
computes nid={192.168.1.[2-128]@tcp} 500, ref 0
default * 10000, ref 0
high_priority_requests:
CPT 0:
user1 jobid={iozone.500 dd.500} 100, ref 0
iozone_user1 opcode={ost_read ost_write} 200, ref 0
```

```
18 computes nid={192.168.1.[2-128]@tcp} 500, ref 0
19 default * 10000, ref 0
20 CPT 1:
21 user1 jobid={iozone.500 dd.500} 100, ref 0
22 iozone_user1 opcode={ost_read ost_write} 200, ref 0
23 computes nid={192.168.1.[2-128]@tcp} 500, ref 0
24 default * 10000, ref 0
```

- **拥塞下的 TBF 实时策略**

在评估 TBF 期间，我们发现当所有类的 I/O 带宽需求总和超过系统容量时，具有相同速率限制的类获得的带宽要比预先均衡配置所获得得带宽要少。造成这种情况的原因是拥塞服务器上的繁重负载会导致某些类错过最后期限。在出列时，令牌的数量可能大于 1。在最初的实现中，所有类都被平等对待，以轻松丢弃超额的令牌。

随着硬令牌补偿（HTC）策略的实施，我们使用 HTC 匹配的规则对类进行配置。这个特性意味着该类队列中的请求具有较高的实时性要求，必须尽可能满足带宽分配。当错过最后期限时，该类保持最后期限不变，剩余的时间（剩余的流逝时间除以 1/r）将被补偿到下一轮。从而确保了下一个空闲 I/O 线程始终选择此类来服务，直到所有累计的超额令牌处理完毕或该类队列中没有挂起的请求。

命令:

添加实时特性的新命令格式:

```
1 lctl set_param x.x.x.nrs_tbf_rule=\
2 "start rule_name arguments... realtime=1
```

示例:

```
1 $ lctl set_param ost.OSS.ost_io.nrs_tbf_rule=
2 "start realjob jobid={dd.0} rate=100 realtime=1
```

在这个例子中，那些JobID 为 dd.0 的 RPC 将以 100 req/sec 的速率进行实时处理。(在 **Lustre 2.10** 中引入)

34.6.6. 延迟策略

NRS 延迟策略旨在通过干扰 PtlRPC 层的请求处理时间来模拟高服务器负载，从而暴露与时间有关的问题。如果启用此策略，将在请求到达时计算应该开始处理请求的时间位移量，并允许其在用户定义的范围内波动。然后使用cfs_binheap将请求按照分配的开始时间进行排序，并保存。一旦请求的开始时间已过，它将从 binheap 中移除以供处理。

延迟策略可在所有类型的 PtlRPC 服务上启用，有以下可用于调整其行为的可调参数：

- {service}.nrs_delay_min

{service}.nrs_delay_min 用于控制请求被此策略延迟的最短时间量（以秒为单位）。默认值是 5 秒。读取此值运行：

```
lctl get_param {service}.nrs_delay_min
```

例如，在 ost_io 服务上读取最小延迟设置：

```
$ lctl get_param ost.OSS.ost_io.nrs_delay_min
ost.OSS.ost_io.nrs_delay_min=reg_delay_min:5
hp_delay_min:5
```

设置 RPC 处理的最小延迟：

```
lctl set_param {service}.nrs_delay_min=0-65535
```

这将为常规和高优先级 RPC 设置给定服务的最小延迟时间。

例如，要将 ost_io 服务的最小延迟时间设置为 10，请运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_min=10
ost.OSS.ost_io.nrs_delay_min=10
```

对于支持高优先级 RPC 的 PtlRPC 服务，可为常规和高优先级 RPC 设置不同的最小延迟时间：

```
lctl set_param {service}.nrs_delay_min=reg_delay_min|hp_delay_min:0-65535
```

例如，在 ost_io 服务上将高优先级 RPC 的最小延迟时间设置为 3：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_min=hp_delay_min:3
ost.OSS.ost_io.nrs_delay_min=hp_delay_min:3
```

请注意，在任何情况下最小延迟时间都不能超过最大延迟时间。

- {service}.nrs_delay_max

{service}.nrs_delay_max 用于控制请求被此策略延迟的最长时间量（以秒为单位）。默认值是 300 秒。读取此值运行：

```
lctl get_param {service}.nrs_delay_max
```

例如，在 ost_io 服务上读取最大延迟设置：

```
$ lctl get_param ost.OSS.ost_io.nrs_delay_max
ost.OSS.ost_io.nrs_delay_max=reg_delay_max:300
hp_delay_max:300
```

设置 RPC 处理的最大延迟：

```
lctl set_param {service}.nrs_delay_max=0-65535
```

这将为常规和高优先级 RPC 设置给定服务的最大延迟时间。

例如，要将 ost_io 服务的最大延迟时间设置为 60，请运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_max=60
ost.OSS.ost_io.nrs_delay_max=60
```

对于支持高优先级 RPC 的 PtlRPC 服务，可为常规和高优先级 RPC 设置不同的最大延迟时间：

```
lctl set_param {service}.nrs_delay_max=reg_delay_max|hp_delay_max:0-65535
```

例如，在 ost_io 服务上将高优先级 RPC 的最大延迟时间设置为 30：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_max=hp_delay_max:30
ost.OSS.ost_io.nrs_delay_max=hp_delay_max:30
```

请注意，在任何情况下最长延迟时间都不能小于最短延迟时间。

- `{service}.nrs_delay_pct`

`{service}.nrs_delay_pct` 用于控制会被此延迟政策推迟的请求的百分比。默认值是 100。请注意，如果某一请求没有被延迟策略选中并推迟处理请求，该请求将由该服务定义的回退策略来处理。如果没有定义其他回退策略，则该请求由 FIFO 策略处理。读取此值请运行：

```
lctl get_param {service}.nrs_delay_pct
```

在 ost_io 服务上读取被延迟的请求的百分比，请运行：

```
$ lctl get_param ost.OSS.ost_io.nrs_delay_pct
ost.OSS.ost_io.nrs_delay_pct=reg_delay_pct:100
hp_delay_pct:100
```

设置延迟请求的百分比：

```
lctl set_param {service}.nrs_delay_pct=0-100
```

这将为常规和高优先级 RPC 设置给定服务的请求延迟的百分比。

例如，要将 ost_io 服务的请求延迟的百分比设置为 50，请运行：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_pct=50
ost.OSS.ost_io.nrs_delay_pct=50
```

对于支持高优先级 RPC 的 PtlRPC 服务，可为常规和高优先级 RPC 设置不同的请求延迟的百分比：

```
lctl set_param {service}.nrs_delay_pct=reg_delay_pct|hp_delay_pct:0-100
```

例如，在 ost_io 服务上将高优先级 RPC 的请求延迟的百分比设置为 5：

```
$ lctl set_param ost.OSS.ost_io.nrs_delay_pct=hp_delay_pct:5
ost.OSS.ost_io.nrs_delay_pct=hp_delay_pct:5
```

34.7. 无锁 I/O 可调参数

无锁 I/O 可调特性允许服务器请求客户端执行无锁 I/O（服务器代表客户端进行锁定）以避免争用文件的 ping-pong 锁定。

无锁 I/O 补丁引入了这些可调参数：

- **OST-side:**

`ldlm.namespaces.filter-fsname-*.`

`contended_locks`－如果超出`conarded_locks`指定的授权等待队列扫描中的锁冲突数量，则认为该资源为争用资源。

`contention_seconds`－该资源保持争用状态时长。

`max_nolock_bytes` －服务器锁定小于`max_nolock_bytes`的块设置的请求。如果此值被设置为零，则禁止服务器端锁定读取/写入请求。

- **Client-side:**

`/proc/fs/lustre/llite/lustre-*`

`contention_seconds`－ `llite` 节点将记住其争用状态的时长。

- **Client-side statistics:**

无锁 I/O 统计信息将会被记录在 `/proc/fs/lustre/llite/lustre-*/stats` 文件中。

`lockless_read_bytes` 和 `lockless_write_bytes`－计算读取或写入的总字节数时，如果请求大小小于`min_nolock_size`，则客户端不会与服务器通信，也不会获取客户端的锁定。

（在 **Lustre 2.9** 中引入）

34.8. 服务器端建议和提示

34.8.1. 概述

使用lfs ladvise命令为服务器提供有关文件访问的建议和提示。

```
1 lfs ladvise [--advice|-a ADVICE ] [--background|-b]
2 [--start|-s START[kMGT]]
3 {[--end|-e END[kMGT]] | [--length|-l LENGTH[kMGT]]}
4 file ...
```

选项	说明
-a ,--advice= ADVICE	提供ADVICE类型的建议或提示。ADVICE类型包括: willread－将数据预先导入服务器缓存; dontneed－清除服务器缓存; lockahead－在给定字节范围内请求给定模式的 LDLM 范围锁; noexpand禁止对此文件描述符的 I/O 的范围锁扩展行为。
-b ,--background	允许建议的发送和处理异步。
-s ,--start= START_OFFSET	文件范围起始于 START_OFFSET。
-e ,--end= END_OFFSET	文件范围终止于（不包括）END_OFFSET。该选项不能与-l 选项同时指定。
-l ,--length= LENGTH	文件范围长度为 LENGTH。该选项不能与-e 选项同时指定。
-m ,--mode= MODE	Lockahead 请求模式{READ,WRITE} 。请求一个该模式下的锁。

通常，lfs ladvise会将建议转发给 Lustre 服务器，但无法保证何时以及哪些服务器会对建议做出反应。根据不同建议的类型以及受影响的服务器端组件的实时决策情况，建议可能会触发操作也可能不会触发操作。

ladvise 的典型用例是使具有外部知识的应用程序和用户能够介入服务器端缓存管理。例如，如果大量不同的客户端正在对文件进行小的随机读取，则在随机 I/O 发生之

前以大线性读取的方式预取页到 OSS 缓存的做法效益可观。由于发送到客户端的数据还要多得多，可能无法使用 fadvise() 将数据提取到每个客户端缓存中。

`ladvise lockahead`的不同之处在于它试图通过在使用之前明确请求 LDLM 锁来控制 LDLM 锁定行为。这不会直接影响缓存行为，相反，它可以在特殊情况下用于避免正常 LDLM 锁定行为导致的病态结果（锁定交换）。

请注意，`noexpand`建议适用于特定的文件描述符，因此通过 lfs 使用它并不起作用。它只能用特定的用于 I/O 的文件描述符。

Linux 系统调用`fadvise()`和`lfs ladvise`之间的主要区别在于`fadvise()`只是一个客户端机制，它不会将建议传递给文件系统，而`ladvise`可以向 Lustre 服务器端发送建议或提示。

34.8.2. 示例

下面的例子中，持有第一个 1GB 的`/mnt/luster/ file1`得到提示：即将读取文件的前 1GB 部分。``

```
client1$ lfs ladvise -a willread -s 0 -e 1048576000 /mnt/lustre/file1/
```

下面的例子中，持有第一个 1GB 的`/mnt/luster/ file1`得到提示：文件的前 1GB 部分在近期不会被读取，所以 OST 可以在内存中清除该文件的缓存。

```
client1$ lfs ladvise -a dontneed -s 0 -e 1048576000 /mnt/lustre/file1
```

请求文件`/mnt/luster/file1`的前 1 MiB 的 LDLM 读取锁，这将尝试从保存有该文件此区域的 OST 请求一个锁：

```
client1$ lfs ladvise -a lockahead -m READ -s 0 -e 1M /mnt/lustre/file1
```

请求文件`/mnt/luster/file1`[3 MiB, 10 MiB] 范围的 LDLM 写入锁，这将尝试从保存有该文件此区域的 OST 请求一个锁：

```
client1$ lfs ladvise -a lockahead -m WRITE -s 3M -e 10M /mnt/lustre/file1
```

34.9. 大批量 I/O (16MB RPC)

34.9.1. 概述

从 **Lustre 2.9** 起，Lustre 支持的 RPC 大小最大已扩展到 16MB。在客户端和服务器之间传输相同数量的数据，启用更大的 RPC 意味着需要更少的 RPC，OSS 可以同时向底层磁盘提交更多数据，因此可以生成更大的磁盘 I/O 以充分利用磁盘日益增加的带宽。

在客户端连接时，客户端将与服务器协商允许使用的最大 RPC。客户端始终可以发送小于此最大值的 RPC。

客户端可通过在 OST 上使用参数brw_size来获知最大（首选）I/O 大小。所有与此目标交互的客户端都不能发送大于此值的 RPC。客户端可以通过 osc.*.max_pages_per_rpc 可调参数单独设置较小的 RPC 大小限制。

注意

可为 ZFS OST 设置的最小brw_size大小即该数据集的 recordsize 大小。这可以确保客户端可以随时写入完整的 ZFS 文件块，而不会强制为每个 RPC 执行读/修改/写操作。

34.9.2. 示例

为了启用更大的 RPC 大小，必须将brw_size的 IO 大小值更改为 16MB。临时更改brw_size，请在 OSS 上运行以下命令：

```
1 oss# lctl set_param obdfilter.fsname-OST*.brw_size=16
```

要持久地更改brw_size，请运行：

```
1 oss# lctl set_param -P obdfilter.fsname-OST*.brw_size=16
```

当客户端连接到 OST 目标时，它将从目标中获取brw_size，并从brw_size中获得其最大值和本地设置作为max_pages_per_rpc的实际 RPC 大小。因此，要启用 16MB 的 RPC，客户端的max_pages_per_rpc必须设置为 16M（如果 PAGESIZE 为 4KB，则为 4096）。临时更改max_pages_per_rpc请在客户端上运行以下命令：

```
1 client$ lctl set_param osc.fsname-OST*.max_pages_per_rpc=16M
```

使更改永久生效，运行：

```
1 client$ lctl set_param -P obdfilter.fsname-OST*.osc.max_pages_per_rpc=16M
```

注意

OST 的brw_size可以随时更改。但客户端必须重新安装并重新协商 RPC 最大大小。

34.10. 提升 Lustre 小文件 I/O 性能

应用程序将小文件块从多个客户端写入单个文件可能会导致较差的 I/O 性能。提高 Lustre 文件系统小文件的 I/O 性能，我们可以：

- 在将 I/O 提交到 Lustre 文件系统之前，应用程序先进行 I/O 聚合。默认情况下，Lustre 软件将强制执行 POSIX 语义一致性。因此，如果它们都同时写入同一文件会导致客户端节点之间发生 ping-pong 锁定。如果应用程序使用 MPI-IO，则实现此功能的一种直接的方法是在 Lustre ADIO 驱动程序中使用 MPI-IO Collective Write 功能。

- 让应用程序对文件执行 4kB 的O_DIRECT大小 I/O，并禁用输出文件上的锁定。这可以避免部分页面 IO 提交，以及客户端之间的争用。
- 让应用程序写入连续的数据。
- 为 OST 添加更多磁盘或使用 SSD 磁盘。这将极大地提高 IOPS 速率。为减少开销（日志，连接等）创建更大的 OST，而不是很多较小的 OST。
- 使用 RAID-1+0 OST 代替 RAID-5/6。小块数据写入磁盘存在 RAID 奇偶校验开销。

34.11. 写入性能与读取性能

通常，Lustre 集群上写操作的性能要优于读取操作。在写入时，所有客户端都异步发送写入 RPC。RPC 按照到达的顺序分配和写入磁盘。在很多情况下，这将允许后端存储高效地会聚合写入操作。

相反，客户端的读取可能会以不同的顺序出现，并且需要大量磁盘搜索。这将明显地阻碍读取吞吐量。

目前，尽管客户端进行预读，OST 本身不进行预读。如果有很多客户端正在读取，执行任何预读都将消耗大量内存（1000 个客户端的单个 RPC（1 MB）预读也会占用 1 GB 的 RAM）而导致无法进行。

对于使用 socklnd（TCP，以太网）互连的文件系统，还会产生额外的 CPU 开销。如果不从网络缓冲区复制数据，客户端将无法接收数据。而在写入案例中，客户端 CAN 无需额外的数据副本即可发送数据。这意味着比起写入操作，客户端在读取期间更有可能受 CPU 限制。

第三十五章 Lustre 文件系统故障排除

35.1. Lustre 错误消息

Lustre 提供了多种资源用于帮助解决文件系统中的问题。本节主要介绍错误代码，错误消息和日志。

35.1.1. 错误代码

错误代码由 Linux 操作系统生成，位于/usr/include/asm-generic/errno.h中。Lustre 软件没有使用所有可用的 Linux 错误代码。错误代码的确切含义取决于它的使用位置。以下是 Lustre 文件系统用户可能遇到的错误摘要。

错误代码	错误名称	**说明**
-1	-EPERM	访问被拒绝。
-2	-ENOENT	请求文件或目录不存在

错误代码	错误名称	说明
-4	-EINTR	操作被中断（通常被 ctrl+c 或终止进程中断）
-5	-EIO	操作失败，存在读/写错误。
-19	-ENODEV	该设备不可用。服务器关闭或故障。
-22	-EINVAL	参数含非法值。
-28	-ENOSPC	文件系统空间不足或索引节点不足。使用 `lfs df` 查询文件系统空间情况，使用 `lfs df -i` 查询索引节点使用情况。
-30	-EROFS	文件系统是只读的，可能由检测到的错误引起。
-43	-EIDRM	UID/GID 和 MDS 上任何已知的 UID/GID 都不匹配。在 MDS 上更新 etc/hosts 和 etc/group，添加遗失的用户或组。
-107	-ENOTCONN	客户端没有连接到服务器。
-110	-ETIMEDOUT	操作超时。
-122	-EDQUOT	操作因超过用户磁盘配额而被丢弃。

35.1.2. 查看错误消息

Lustre 软件代码在内核上运行，能够向应用程序显示一位数的错误代码，这些错误代码指示特定的问题。在节点上，`/var/log/messages`保存有含至少过去一天的所有消息的日志。有关来自该节点的所有最新内核消息，请参阅内核控制台日志（dmesg）。

错误消息在控制台日志中被初始化为"LustreError"，并提供以下简短说明：

- 问题是什么
- 哪个进程 ID 出现了问题
- 正在与哪个服务器节点进行通讯，等等

Lustre 日志被放在了 `/proc/sys/lnet/debug_path`中。

收集与问题相关的第一组消息以及在"LBUG" 或"assertion failure" 错误之前的任何消息。提到服务器节点（OST 或 MDS）的消息特指与该服务器相关的错误；您必须从相关的服务器控制台日志收集类似的消息。

另一个 Lustre 调试日志包含 Luster 软件短时间内执行操作的信息，而 Lustre 软件依赖于 Lustre 节点上的进程。使用以下命令提取每个节点上的调试日志：

```
1 $ lctl dk filename
```

注意

LBUG 通过冻结线程来捕获 panic 堆栈。需要进行系统重启来清除线程。

35.2. 报告 Lustre 文件系统 Bug

如果通过对 Lustre 文件系统进行故障排除仍无法解决问题，可尝试其他解决途径：

- 在 lustre-discuss 邮件列表发布您的问题或在档案中搜索您的问题以获得更多信息。
- 向 Lustre 软件项目的Jira* bug 追踪和项目管理工具提交故障单。首次使用需要在欢迎页面注册账号。

请按照以下步骤发起 Jira 申诉：

1. 为避免重复提交故障单，请搜索现有故障单以解决问题。有关搜索提示，请参见本章第 2.1 节" 在 Jira Bug Tracker 中搜索重复故障单"。
2. 创建申诉，请点击右上角的 **+Create Issue**。请为您想询问的每一个问题提交单独的故障单。
3. 在显示的表格中，输入：

- Project - 选择 **Lustre** 或 **Lustre Documentation** 或其它合适的项目。
- Issue type - 选择 **Bug**。
- Summary - 输入问题的简短描述。使用有利于搜索类似问题的术语，例如，LustreError 或 ASSERT/panic 通常是一个很好的总结。
- Affects version(s) - 选择您的 Lustre 版本。
- Environment - 输入您的内核及其版本。
- Description - 可见症状的详细描述，以及问题的产生方式（可能的话）。其他有用的信息包括您期望的行为，以及为诊断该问题您已尝试的方式。
- Attachments - 上传如 Lustre 调试日志、系统日志、控制台日志等。**注意：** 在 Jira 故障单中上传 Lustre 调试日志前请使用`lctl df`处理调试日志。

表单中的其他字段用于项目跟踪，与报告问题无关，可以维持默认状态。

35.2.1. 在 Jira* Tracker 中搜索重复故障单

在提交故障单之前，请在 Jira Bug Tracker 中查找与您问题有关的现有故障单。这样可以避免重复工作，并可能立即获得解决方案。

在 Jira Bug Tracker 中进行搜索，请选择 **Issues** 选项卡，然后单击 **New filter**。可使用提供的过滤器为您的搜索选择条件。搜索特定文本，请在 **Contains text** 字段中输入文本，然后单击放大镜图标。

搜索诸如 ASSERTION 或 LustreError 消息之类的文本时，您可以按照下面的示例进行搜索，请从字符串中删除 NIDS 和其他有关安装的特定文本。

原始错误消息：

"(filter_io_26.c:**791**:filter_commitrw_write())ASSERTION(oti->oti_transno <=obd->obd_last_committed)failed: oti_transno**752** last_committed**750**"

优化后的搜索字符串：

```
1 filter_commitrw_write ASSERTION oti_transno obd_last_committed failed:
```

35.3. Lustre 文件系统常见问题

本节主要介绍如何解决 Lustre 文件系统的常见问题。

35.3.1. OST 对象缺失或损坏

如果 OSS 找不到对象或找到损坏的对象，则会显示以下消息：

```
1 OST object missing or damaged (OST ” ” ost1, object 98148, error -2)
```

如果报告的错误是 -2（-ENOENT或" 没有这样的文件或目录"），则该对象丢失。这可能是因为 MDS 和 OST 没有同步，或者是 OST 对象被删除或已损坏。

如果您使用了e2fsck从磁盘故障中恢复文件系统，则这些不可恢复的对象可能已经被删除了或者在原始 OST 分区上被移至/lost+found中。由于 MDS 上的文件仍然引用着这些对象，尝试访问它们会产生此错误。

如果您从原始 MDS 或 OST 分区的备份进行了恢复，则恢复的分区很可能与集群的其余部分不同步。无论您还原的是服务器的哪个分区，MDS 上的文件都可能引用已不再存在（或备份时不存在）的对象。访问这些文件也会产生此错误。

如果上述情况都不适用，那可能存在编程错误，从而导致了服务器不同步。请提交 Jira 故障单（参见本章第 2 节" 报告 Lustre 文件系统错误"）。

如果报告的错误是其他任何内容（如 -5，"I/O error"），则可能表示存储故障。如果无法从存储设备读取数据，则低级文件系统会返回此错误。

建议的操作

如果报告的错误是 -2，则可以考虑在原始 OST 设备上的 /lost+found 中查找丢失的对象。但是，很可能这个对象已经永远丢失，且引用对象的文件现已部分或完全丢失。从备份中恢复此文件，或挽救所有您能找到的文件，再进行删除。

如果报告的错误是其他内容，则应立即检查此服务器是否存在存储问题。.

35.3.2. OSTs 变为只读

如果块设备级别的 Lustre 文件系统无法访问 SCSI 设备，则`ldiskfs`会将该设备重新安装为只读，以防止文件系统损坏。这是一种正常行为。受影响的节点上的`/proc/fs/lustre/health_check`的状态显示为"not healthy"。

确定造成"not healthy" 状况的原因：

- 检查所有服务器的控制台是否有任何错误指示
- 检查所有服务器的系统日志是否存在 LustreErrors 或 LBUG
- 检查系统硬件和网络的健康状况。（磁盘是否按预期工作，网络是否丢包?）
- 考虑当时集群上发生了什么。这与特定用户工作负载或系统负载情况有关吗？该情况是否可重现？它是否发生在特定的时间 (day, week or month)?

要从此问题中恢复，您必须使用这些文件系统重新启动 Lustre 服务。没有其他方法可以知道哪些磁盘 I/O 造成了这个问题，以及缓存与磁盘是否存在内容不一致的情况。

35.3.3. 识别丢失的 OST

如果丢失了某个 OST，您可能需要知道哪些文件受到了影响。文件系统通常在丢失一个 OST 的情况下仍可操作。请从任何已挂载的客户端节点生成位于受影响的 OST 上的文件的列表。建议将丢失的 OST 标记为" 不可用"，以防止客户端和 MDS 尝试联系它而超时。

1. 生成设备列表并确定 OST 的设备编号，运行：

   ```
   $ lctl dl
   ```

   `lctl dl`命令将列出设备名称和编号以及设备 UUID 和设备上的引用编号。

2. 停用 OST（在 MDS 的 OSS 上）。运行：

   ```
   $ lctl --device lustre_device_number deactivate
   ```

   OST 设备编号或设备名称由`lctl dl`命令生成。

   `deactivate`命令可以防止客户端在指定的 OST 上创建新的对象（尽管您仍可以读取 OST）。

   注意

   如果 OST 稍后变为可用，则需要重新激活它，运行：

   ```
   # lctl --device lustre_device_number activate
   ```

3. 确定所有在丢失 OST 上条带化的文件，运行：

   ```
   # lfs find -O {OST_UUID} /mountpoint
   ```

   这会从受影响的文件系统返回一个简单的文件名列表。

4. 如有必要，您可以阅读条带化文件的有效部分，运行：

```
# dd if=filename of=new_filename bs=4k conv=sync,noerror
```

5. 您可以使用unlink命令删除这些文件。

```
# unlink|munlink filename {filename ...}
```

注意

运行 unlink命令时，可能会返回一个 ``无法找到文件" 错误，并将 MDS 上的文件永久删除。

目前无法在文件系统不能挂载的情况下直接从 MDS 中解析元数据。如果故障 OST 没有启动，则挂载文件系统的其它方法是使用一个循环 OST 或新格式化的 OST 将其替换。在这种情况下，丢失的对象被创建，且被读为零填充。

35.3.4. 修复 OST 上错误的 LAST_ID

每个 OST 都包含一个 LAST_ID 文件，该文件保存由 MDS（预）创建的最后一个对象。MDT 包含一个 lov_objid 文件，其中的值代表 MDS 分配给文件的最后一个对象。

在正常操作期间，MDT 在 OST 上会保留一些预先创建的（但未分配的）对象，而 LAST_ID 和 lov_objid 之间的关系应为 LAST_ID > lov_objid。文件值中的差异都会导致 OST 下次连接到 MDS 时在 OST 上创建对象。这些对象从未实际分配给文件，它们的长度为 0（空）。

但是，如果 lov_objid >LAST_ID，表明 MDS 将这些对象分配给了 OST 上不存在的文件。相反，如果 lov_objid 远远小于 LAST_ID（至少 2 万个对象），则表明 OST 之前在 MDS 的请求下分配了对象（很可能包含数据），但它不知道这些对象的存在。

从 **Lustre 2.5** 开始，如果 lov_objid 和 LAST_ID 文件不同步，则 MDS 与 OSS 将自动使其重新同步。这可能会导致 OST 上的一些空间在下一次运行 LFSCK 之前无法使用，但可以避免挂载文件系统的问题。

从 **Lustre 2.6** 开始，LFSCK 会根据 OST 上存在的对象，自动修复 OST 上的 LAST_ID 文件，以防该文件被损坏。

在磁盘损坏 OST 的情况下（如由于磁盘上启用了写入缓存引起的故障，或 OST 从旧的备份或重新格式化后恢复），LAST_ID 值可能会变得不一致，并生成类似于以下内容的消息：

```
"myth-OST0002: Too many FIDs to precreate, OST replaced or
reformatted: LFSCK will clean up"
```

如果 OST 上先前创建的对象的记录与 MDS 上的先前分配的对象之间存在显着差异（例如，MDS 已损坏或从备份中恢复，如果未校验则可能导致严重的数据丢失），则可能导致类似情形。这将产生如下信息：

```
"myth-OST0002: too large difference between
MDS LAST_ID [0x1000200000000:0x100048:0x0] (1048648) and
OST LAST_ID [0x1000200000000:0x2232123:0x0] (35856675), trust the OST"
```

在这种情况下，MDS 将修改 lov_objid 的值以与 OST 的值相匹配，从而避免删除现有的可能包含数据的对象。MDT 上引用这些对象的文件不会丢失。任何未被引用的 OST 对象将在下次运行 LFSCK 布局检查时被添加到.lustre/lost+found目录中。

35.3.5. 处理"Bind: Address already in use"错误

在启动过程中，Lustre 软件可能会报告 bind: Address already in use 错误并拒绝启动操作。这是由于在 Lustre 文件系统启动之前启动了 portmap 服务（通常是 NFS 锁定），并绑定到默认端口 988。您必须在客户端、OSS 和 MDS 节点上的防火墙或 IP 表中为传入连接打开端口 988。LNet 将在可用的预留端口上为每个客户端－服务器对创建三个传出连接（从 1023、1022 和 1021 开始）。

不幸的是，您不能设置 sunprc 以避免使用端口 988。如果您收到此错误，请执行以下操作：

- 再启动任何使用 sunrpc 的服务前启动 Lustre 文件系统。
- 为 Lustre 文件系统使用 988 以外的端口。这可在 LNet 模块中的 /etc/modprobe.d/lustre.conf 配置，如：

```
options lnet accept_port=988
```

- 在使用 sunrpc 的服务之前，将 modprobe ptlrpc 添加到您的系统启动脚本中。这会使 Lustre 文件系统绑定到端口 988，sunrpc 以选择不同的端口。

注意

您还可以使用sysctl命令缓解 NFS 客户端获取 Lustre 服务端口。但这是一个解决部分问题的变通办法，因为其他用户空间 RPC 服务器仍然可以获取端口。

35.3.6. 处理错误"- 28"

在写入或同步操作期间发生的 Linux 错误 -28（ENOSPC）指示在 OST 上的现有文件由于 OST 已满（或几乎已满）而无法覆盖写或更新。要验证是否属于这种情况，请挂载该 OST 的客户端上输入：

``` client$ lfs df -h UUID bytes Used Available Use% Mounted on myth-MDT0000_UUID 12.9G 1.5G 10.6G 12% / myth[MDT: 0] myth-OST0000_UUID 3.6T 3.1T 388.9G 89%
```

/ myth[OST: 0] myth-OST0001_UUID 3.6T 3.6T 64.0K 100% / myth[OST: 1] myth-OST0002_UUID 3.6T 3.1T 394.6G 89% / myth[OST: 2] myth-OST0003_UUID 5.4T 5.0T 267.8G 95% /myth[OST:3] myth-OST0004_UUID 5.4T 2.9T 2.2T 57% /myth[OST:4]

filesystem_summary: 21.6T 17.8T 3.2T 85% /myth ```

解决这个问题，您可以扩展 OST 的磁盘空间，或使用`lfs_migrate`将文件迁移至不那么拥挤的 OST 上。

（Lustre2.6 引入）在某些情况下，一些持有打开的文件的进程消耗了大量的空间（例如：失控进程向已删除的打开的文件写入大量数据）。可以从 MDS 中获取文件系统中所有打开的文件句柄的列表列表：

```
mds# lctl get_param mdt.*.exports.*.open_files
mdt.myth-MDT0000.exports.192.168.20.159@tcp.open_files=
[0x200003ab4:0x435:0x0]
[0x20001e863:0x1c1:0x0]
[0x20001e863:0x1c2:0x0]
:
:
```

These file handles can be converted into pathnames on any client via the `lfs fid2path` command (as root):

```
client# lfs fid2path /myth [0x200003ab4:0x435:0x0] [0x20001e863:0x1c1:0x0]
    [0x20001e863:0x1c2:0x0]
lfs fid2path: cannot find '[0x200003ab4:0x435:0x0]': No such file or
    directory
/myth/tmp/4M
/myth/tmp/1G
:
:
```

在某些情况下，如果文件已经从文件系统中删除，fid2path 会返回一个 ``文件没有找到" 的错误。你可以使用客户端的 NID(如上面的例子中的 192.168.20.159@tcp) 来确定文件是在哪个节点上打开的，而 lsof 则可以找到并杀死持有该文件的进程。

```
# lsof /myth
COMMAND   PID   USER  FD TYPE    DEVICE      SIZE/OFF               NODE
    NAME
logger  13806 mythtv  0r REG  35,632494 1901048576384 144115440203858997
    /myth/logs/job.1283929.log (deleted)
```

在创建新文件时发生的 Linux 错误 -28（ENOSPC）可能表示 MDS 的 inode 资源已耗尽，MDS 需要扩展。新创建的文件不会写入满的 OST，而现有文件将继续存在最初创建的 OST 中。要查看 MDS 上的 inode 信息，请输入：

```
lfs df -i
UUID                      Inodes       IUsed       IFree IUse% Mounted on
myth-MDT0000_UUID        1910263     1910263           0 100% /myth[MDT:0]
myth-OST0000_UUID         947456      360059      587397  89% /myth[OST:0]
myth-OST0001_UUID         948864      233748      715116  91% /myth[OST:1]
myth-OST0002_UUID         947456      549961      397495  89% /myth[OST:2]
myth-OST0003_UUID        1426144      477595      948549  95% /myth[OST:3]
myth-OST0004_UUID        1426080      465248     1420832  57% /myth[OST:4]

filesystem_summary:      1910263     1910263           0 100% /myth
```

通常，Lustre 软件会将此错误报告给您的应用程序。如果应用程序正在从函数调用中检查返回代码，它会将其解码为文本的错误消息（如 No space left on device）。这两个版本的错误信息都会出现在系统日志中。

你也可以使用 lctl get_param 命令来监控任一客户端的 OSTs 和 MDTs 上的空间和对象使用情况。

```
lctl get_param {osc,mdc}.*.{kbytes,files}{free,avail,total}
```

注意

您可以在/usr/include/asm/errno.h中找到其他数字错误代码以及简短的名称和文本说明。

35.3.7. 触发 PID NNN 看门狗定时器

在某些情况下，服务器节点会触发看门狗定时器，这会导致进程堆栈转储到控制台，Lustre 内核调试日志转储到/tmp（默认情况下）。触发看门狗定时器并不意味着线程的 OOPS 错误，而是它完成给定操作将需要比预期更长的时间。

在某些情况下，可能会出现这种情况。例如，RAID 重建实际上减慢了 OST 上的 I/O 速度，它可能会触发看门狗定时器跳闸。但不久之后又有一条消息，表明有问题的线程已经完成了处理（几秒钟后）。一般来说，这表示这只是一个暂时的问题。在其他情况下，它可能会指示线程因软件错误（如锁反转）而卡住了。

```
Lustre: 0:0:(watchdog.c:122:lcw_cb())
```

以上消息表明看门狗已为 pid 933 启动：

它在 100000ms 内关闭：

```
Lustre: 0:0:(linux-debug.c:132:portals_debug_dumpstack())
```

显示进程的堆栈：

```
933 ll_ost_25     D F896071A     0   933      1    934   932 (L-TLB)
f6d87c60 00000046 00000000 f896071a f8def7cc 00002710 00001822 2da48cae
0008cf1a f6d7c220 f6d7c3d0 f6d86000 f3529648 f6d87cc4 f3529640 f8961d3d
00000010 f6d87c9c ca65a13c 00001fff 00000001 00000001 00000000 00000001
```

调用追踪：

```
filter_do_bio+0x3dd/0xb90 [obdfilter]
default_wake_function+0x0/0x20
filter_direct_io+0x2fb/0x990 [obdfilter]
filter_preprw_read+0x5c5/0xe00 [obdfilter]
lustre_swab_niobuf_remote+0x0/0x30 [ptlrpc]
ost_brw_read+0x18df/0x2400 [ost]
ost_handle+0x14c2/0x42d0 [ost]
ptlrpc_server_handle_request+0x870/0x10b0 [ptlrpc]
ptlrpc_main+0x42e/0x7c0 [ptlrpc]
```

35.3.8. 处理初始 Lustre 文件系统设置的超时

如果您遇到 Lustre 文件系统初始设置的超时或挂起，请查看服务器和客户端的名称解析是否正常工作。某些版本配置/etc/hosts将本地计算机的名称（由'hostname' 命令指示）映射到本地主机（127.0.0.1）而不是正确的 IP 地址。

这可能会产生这个错误：

```
LustreError:(ldlm_handle_cancel()) received cancel for unknown lock cookie
0xe74021a4b41b954e from nid 0x7f000001 (0:127.0.0.1)
```

35.3.9. 处理"LustreError: xxx went back in time" 错误

MDS 或 OSS 每次为客户修改 MDT 或 OST 磁盘文件系统的状态时，都会为该操作记录一个按目标增加的事务编号，并将其与该操作的回复一起返回给客户段。当服务器将这些事务提交到磁盘时，最后提交的事务编号被定期地返回给客户端，并允许从内存中丢弃未提交的操作，因为在服务器故障的情况下，未提交的操作不再需要恢复。

在某些情况下，在服务器重启或发生故障转移后，观察到类似以下的错误信息：

```
LustreError: 3769:0:(import.c:517:ptlrpc_connect_interpret())
testfs-ost12_UUID went back in time (transno 831 was previously committed,
```

```
3 server now claims 791)!
```

出现这种情况的原因是：

- 您正在使用在数据写入实际执行前就声称有数据写入的磁盘设备（如具有大缓存的设备）。如果该磁盘设备的故障或断电导致缓存丢失，那么您认为已完成的约定交易也将丢失。这非常严重，您应该在重新启动 Lustre 文件系统之前对该存储运行 e2fsck。
- 根据 Lustre 软件的要求，用于故障切换的共享存储是缓存一致的。这确保了如果一台服务器接管另一台服务器，它可以看到最新的准确数据副本。当服务器进行故障切换时，如果共享存储未提供所有端口之间的缓存一致性，则 Lustre 软件可能会产生错误。

如果您知道错误的确切原因，则无需采取进一步行动。如果您不知道，请与您的磁盘供应商进行深入探讨。

如果错误发生在故障转移期间，请检查您的磁盘缓存设置。如果错误发生在未进行故障切换的重启后，请尝试如何能让磁盘写入成功，然后解决数据设备损坏问题或磁盘错误。

35.3.10. Lustre 错误: "Slow Start_Page_Write"

当操作花很长的时间分配一批内存页时，会出现slow start_page_write消息。请先使用这些内存页接收网络通信，然后再用于写入磁盘。

35.3.11. 多客户端 O_APPEND 写入的劣势

多客户端通过O_APPEND写入单个文件是可能的，但存在很多缺点，使它成为次优解决方案。

- 每个客户端都需要对所有 OST 进行EOF锁定。这是由于在检查所有 OST 之前，很难知道哪个 OST 保存了文件的结尾。所有的客户端都使用同一个O_APPEND，因此存在很大的锁定开销。
- 第二个客户端在第一个客户端完成写入之前不能获取所有锁，客户端只能顺序写入。
- 为避免死锁，它们以已知的一致顺序获取锁。对于条带化文件来说，客户端在获取所有 OSTs 的锁前无法知道哪个 OST 持有文件的下一部分。

35.3.12. Lustre 文件系统启动时的减速

当 Lustre 文件系统启动时，它需要从磁盘读入数据。重启后运行的第一个 mdsrate，MDS 需要等待所有 OST 完成对象预创建，这将导致文件系统启动时的减速。

文件系统运行一段时间后，缓存中将包含更多的数据，从磁盘读取关键元数据引起的可变性将大大地消除。文件系统现在从缓存中读取数据。

35.3.13. OST 上的日志信息"Out of Memory"

规划 OSS 节点硬件时，请把 Lustre 文件系统中多个组件的内存使用情况列入考虑。如果内存不足，"out of memory" 消息将被记录。

在正常操作期间，以下几种状况表明服务器节点内存不足：

- 内核"Out of memory" 和/或"oom-killer" 消息
- Lustre "kmalloc of 'mmm' (NNNN bytes)failed..." 消息
- Lustre 或内核堆栈跟踪显示进程卡在"try_to_free_pages" 消息

35.3.14. 设置 SCSI I/O 大小

某些 SCSI 驱动程序默认的最大 I/O 大小对于高性能的 Lustre 文件系统而言仍然过小。我们已经调整了不少驱动程序，但您仍然可能会发现某些驱动程序使用 Lustre 文件系统时性能不理想。由于默认值是硬编码的，您需要重新编译驱动程序来更改默认值。另外，一些驱动程序的默认设置可能是错误的。

如果您察觉到 I/O 性能较差，且 Lustre 文件系统统计信息的分析表明其 I/O 不是 1 MB，请检查 /sys/block/device/queue/max_sectors_kb。如果max_sectors_kb值小于 1024，请将其设置为 1024 或更大，从而提高性能。如果更改max_sectors_kb值没有改变 Lustre I/O 大小，您可能需要检查 SCSI 驱动程序代码。

第三十六章故障恢复

36.1. 在备份 ldiskfs 文件系统上恢复错误或损坏

OSS，MDS 或 MGS 服务器崩溃时，无需在文件系统上运行 e2fsck，ldiskfs journaling 会确保文件系统在系统崩溃时仍保持一致。客户端不直接访问 ldiskfs 文件系统，因此客户端崩溃与服务器文件系统一致性无关。

只有当有事件导致了 ldiskfs journaling 无法处理的问题时（如硬件设备故障或 I/O 错误），才需要在设备上运行 e2fsck。如果 ldiskfs 内核代码检测到磁盘损坏，它会将文件系统挂载为只读，以防止进一步损坏，但仍允许该设备的读取访问。这在服务器的系统日志中显示为"-30"（EROFS）错误，例如：

```
1 Dec 29 14:11:32 mookie kernel: LDISKFS-fs error (device sdz):
2            ldiskfs_lookup: unlinked inode 5384166 in dir #145170469
```

```
3 Dec 29 14:11:32 mookie kernel: Remounting filesystem read-only
```

在这种情况下，通常只需要在损坏设备上运行 e2fsck，然后再重新启动设备。

在绝大多数情况下，Lustre 软件可以应对磁盘上或文件系统其他设备间发现的任何不一致情况。

强烈建议在记录器（如脚本）下运行e2fsck来记录对文件系统所做的所有输出和更改，以备稍后用于问题分析。

如果时间允许，请首先在非修复模式下运行e2fsck（-n选项）以评估文件系统损害类型和程度。但这么做的缺点是，e2fsck在这种模式下不能恢复文件系统日志，因此可能会出现看起来存在文件系统损坏但实际却没有的情况。

为了分清损坏是真实的还是由于未重播日志造成的假象，您可以使用类似于以下的命令，直接在已关闭 Lustre 文件系统的节点上挂载和卸载ldiskfs文件系统：

```
1 mount -t ldiskfs /dev/{ostdev} /mnt/ost; umount /mnt/ost
```

这会引起日志的恢复。

e2fsck工具在修复文件系统损坏（比类似的文件系统恢复工具更好，这也是为什么选择 ldiskfs 的原因）方面表现良好。尽管如此，确定损害的类型非常重要。一旦知道了损害的类型，ldiskfs专家可以针对需要修复的问题代替e2fsck做出明智的决定。

```
1 root# {stop lustre services for this device, if running}
2 root# script /tmp/e2fsck.sda
3 Script started, file is /tmp/e2fsck.sda
4 root# mount -t ldiskfs /dev/sda /mnt/ost
5 root# umount /mnt/ost
6 root# e2fsck -fn /dev/sda   # don't fix file system, just check for
    corruption
7 :
8 [e2fsck output]
9 :
10 root# e2fsck -fp /dev/sda   # fix errors with prudent answers (usually yes)
```

36.2. 在 Lustre 文件系统上恢复损坏

如果 ldiskfs MDT 或 OST 损坏，您需要运行e2fsck来修复本地文件系统一致性，然后使用LFSCK在文件系统上运行分布式检查，以解决 MDT 和 OST 之间或 MDT 之间的不一致问题。

1. 关闭 Lustre 文件系统。

2. 在有问题的单个 MDT/OST 上运行`e2fsck -f`修复任何本地文件系统的损坏。

我们建议在脚本下运行`e2fsck`，创建记录文件系统更改的日志以供日后使用。运行`e2fsck`后，如有必要，调出文件系统以减少停机窗口。

36.2.1. 处理孤立对象

孤立对象问题是最简单的问题。运行 LFSCK 布局检查时，这些对象将链接到新文件，并放入 Lustre 文件系统中的 `.lustre/lost+found/MDTxxxx`（其中，MDTxxxx 是找到孤立对象的 MDT 的索引），再根据需要对其进行检查、保存或删除。

Lustre 2.7 及以上版本中，LFSCK 也负责识别和处理 MDT 上的孤立对象。

36.3. 从不可用的 OST 中恢复

在 Lustre 文件系统环境中会遇到一个问题：由于网络分区、OSS 节点崩溃等，OST 变得不可用。发生这种情况时，OST 的客户端将暂停并等待 OST 再次变为可用（在主 OSS 或故障切换节点 OSS）。当 OST 重新上线时，Lustre 文件系统将启动恢复以使客户端重新连接到 OST。Lustre 服务器对于等待恢复并重连客户端设置了时间限制。

在恢复过程中，客户端按照他们原来的顺序重新连接并重播他们的请求。在收到已将某交易写入稳定存储的确认之前，客户端会保留该交易，以便在需要时重播。定期向日志输出进度消息，并说明有客户端如何实现重新连接以及多少客户端已完成重新连接。如果恢复被中止，此日志将显示有多少客户端曾设法重新连接。当所有客户端都已完成恢复，或恢复超时时，恢复期结束，OST 恢复正常的请求处理。

即使某些客户端在恢复期间无法重播他们的请求，不会阻止恢复完成。您可能会遇到在 OST 进行恢复时某些客户端无法参与恢复（如网络问题或客户端故障）的情况，此时它们将被驱逐，他们的请求将无法重播。因此，对被驱逐的客户端进行任何操作都将失败，包括正在进行的写入操作，从而导致缓存的写入操作丢失。文件系统在客户端发生故障时将挂起，恢复也不能无限期地等待。很不幸这可能导致交易丢失，但这是正常的结果。

注意

客户端恢复失败并不表示（或导致）文件系统损坏。MDT 和 OST 能够处理类似的正常事件，不会导致服务器之间的任何不一致。

版本的恢复（VBR）功能使故障客户端能够被" 略过"，以便剩余的客户端重放他们的请求，从而使故障 OST 更加成功地进行恢复。

36.4. 使用 LFSCK 检查文件系统

LFSCK 是一种管理工具，用于检查和修复已挂载的 Lustre 文件系统的特殊属性。在概念上，它与用于本地文件系统的离线修复工具类似，但在实现上，在 Lustre 文件系

统挂载使用时，LFSCK 作为文件系统的一部分运行。这样，Lustre 特殊元数据的一致性检查和修复可以在不停机的情况下进行，因此对正常操作的影响可以忽略不计。

LFSCK 能够验证并修复对象索引表（Object Index，OI），对象索引表用于从 Lustre 文件标识符（FID）映射到 MDT 内部 ldiskfs inode 编号。LFSCK 包含一种被称为 OI 清理（OI Scrub）的过程，它会遍历对象索引表，并在必要时进行修正。在对 MDT 进行文件级的备份恢复后，或者在对象索引表受损的情况下，都需要触发 OI Scrub。在 OI Scrub 之后，LFSCK 将在后续阶段进一步检查 Lustre 分布式文件系统状态。LFSCK 的命名空间扫描功能可以验证和修复目录 FID-in-dirent 和 LinkEA 之间的一致性。无论 Lustre 文件系统规模有多大，LFSCK 都可以上面运行。

在 **Lustre 2.6** 中，LFSCK 布局扫描能够验证并修复 MDT-OST 文件布局的不一致性。MDT 对象和 OST 对象之间的文件布局不一致包括悬挂引用、未引用的 OST 对象、不匹配的引用和多重引用。

在 **Lustre 2.7** 中的 LFSCK 布局扫描可以支持多个 MDT 之间的验证和不一致性修复。

LFSCK 的控制和监视是通过 LFSCK 和`lctl get_param`命令实现的。LFSCK 支持三种类型的接口：switch 接口，status 接口和 adjustment 接口。

36.4.1. LFSCK switch 接口

36.4.1.1. 手动启动 LFSCK

36.4.1.1.1. 说明

LFSCK 可在 MDT 挂载后使用 `lctl lfsck_start` 命令启动。

36.4.1.1.2. 示例

```
lctl lfsck_start <-M | --device [MDT,OST]_device> \
                    [-A | --all] \
                    [-c | --create_ostobj on | off] \
                    [-C | --create_mdtobj on | off] \
                    [-d | --delay_create_ostobj on | off] \
                    [-e | --error {continue | abort}] \
                    [-h | --help] \
                    [-n | --dryrun on | off] \
                    [-o | --orphan] \
                    [-r | --reset] \
                    [-s | --speed ops_per_sec_limit] \
                    [-t | --type check_type[,check_type...]] \
                    [-w | --window_size size]
```

36.4.1.1.3. 选项

下表列出了`lfsck_start` 的各种选项。输入 `lctl lfsck_start -h`查看所有可用选项的完整列表。

选项	说明
`-M | --device`	启动 LFSCK 的 MDT 或 OST 目标。
`-A | --all`	在所有服务器上的所有目标上启动 LFSCK。默认情况下，布局和命名空间的一致性检查、修复也同时启动。（**Lustre 2.6** 中引入）
`-c | --create_ostobj`	为悬挂 LOV EA 创建丢失的 OST 对象，值为 off（默认）或 on。如果没有指定该值，那么默认为保留悬挂的 LOV EA 而不创建丢失的 OST 对象。（**Lustre 2.6** 中引入）
`-C | --create_mdtobj`	为悬挂名称条目创建丢失的 MDT 对象，值为 off（默认）或 on。如果没有指定该值，那么默认为保留悬挂名称条目而不创建丢失的 MDT 对象。（**Lustre 2.7** 中引入）
`-d | --delay_create_ostobj`	延迟为悬挂 LOV EA 创建丢失的 OST 对象，先完成孤立 OST 对象处理。值为 off（默认）或 on。（**Lustre 2.9** 中引入）
`-e | --error`	错误处理，值为 `continue`（默认）或 `abort`，用于指定修复失败时 LFSCK 是否停止运行。如果没有指定该值，则使用保存值（从检查点恢复）。LFSCK 运行时该选项不能更改。
`-h | --help`	帮助信息操作。
`-n | --dryrun`	用于不做任何更改的测试，值为 off（默认）或 on。
`-o | --orphan`	针对布局 LFSCK 修复孤立的 OST 对象。（**Lustre 2.6** 中引入）
`-r | --reset`	为指定 MDT 重置对象循环的起始点。默认情况

选项	说明
	下，迭代器会从上一个检查点（如果可用的话，检查点通常由 LFSCK 定期保存）恢复扫描。
-s | --speed	设置 LFSCK 处理的速率上限（每秒的对象数）。如果未指定，则使用保存值（从检查点恢复）或默认值 0（0 即尽可能快地运行）。该速率可通过 adjustment 接口在 LFSCK 运行时更改。
-t | --type	应执行的检查或修复的类型。新的 LFSCK 框架为各种不同系统的一致性检查和修复操作提供单一的接口，包括：没有指定的选项－上次运行且未完成的 LFSCK 组件或与已知的造成某些系统不一致的相对应组件将启动。只要触发 LFSCK，OI scrub 将自动运行，因此无需在此情况下指定 OI_scrub。namespace－检查并修复 FID-in-dirent 和 LinkEA 一致性（**Lustre 2.4** 中引入，**Lustre 2.7** 中增强了 DNE 模式下的名称空间一致性验证）。layout－检查和修复 MDT-OST 不一致性。（**Lustre 2.6** 中引入）。
-w | --window_size	异步请求通道的窗口大小。LFSCK 异步请求通道的输入/输出可能具有完全不同的处理速度，过多的请求也可能导致异常的内存/网络压力。如果未指定，则窗口默认值为 1024

36.4.1.2. 手动关闭 LFSCK 36.4.1.2.1. 说明

在 MDT 挂载后关闭 LFSCK，请使用 lctl lfsck_stop 命令。

36.4.1.2.2. 示例

```
1 lctl lfsck_stop <-M | --device [MDT,OST]_device> \
```

```
2                   [-A | --all] \
3                   [-h | --help]
```

36.4.1.2.3. 选项

下表列出了lfsck_stop 的各种选项。输入 lctl lfsck_stop -h查看所有可用选项的完整列表。

选项	说明
-M | --device	需关闭 LFSCK 的 MDT 或 OST 目标。
-A | --all	同时关闭所有服务器上所有目标的 LFSCK。
-h | --help	帮助信息操作。

36.4.2. 查看 LFSCK 全局状态

36.4.2.1. 说明 通过 MDS 上的一条lctl lfsck_query命令检查 LFSCK 全局状态。

```
1 lctl lfsck_query <-M | --device MDT_device> \
2                   [-h | --help] \
3                   [-t | --type lfsck_type[,lfsck_type...]] \
4                   [-w | --wait]
```

36.4.2.3. 选项 下表列出了lfsck_query 的各种选项。输入 lctl lfsck_query -h查看所有可用选项的完整列表。

Option	**Description**
-M | --device	要查询 LFSCK 状态的设备。
-h | --help	操作帮助信息。
-t | --type	应该被查询的 LFSCK 类型，包括：布局、命名空间。
-w | --wait	如果 LFSCK 在扫描中，查询将等待。

36.4.3. LFSCK status 接口

36.4.3.1. 通过 procfs查看 OI Scrub 的 LFSCK 状态 36.4.3.1.1. 说明

对于每个 LFSCK 组件，都有一个专用的 procfs 接口来跟踪相应的 LFSCK 组件状态。对于 OI Scrub 来说是 OSD 层 procfs 接口，名为 oi_scrub。可使用标准的`lctl get_param`命令来查看 OI Scrub 状态。

36.4.3.1.2. 示例

```
lctl get_param -n osd-ldiskfs.FSNAME-[MDT_target|OST_target].oi_scrub
```

36.4.3.1.3. 输出

信息类型	详细内容
普通信息	名称：OI_scrub. OI scrub magic id：OI scrub 的唯一标识符。 OI 文件数量。 状态：`init`，`scanning`，`completed`，`failed`，`stopped`，`paused`或`crashed`。 标志：包括`recreated`（OI 文件被移除或重建），`inconsistent`（从文件级备份恢复），`auto`（由非 UI 机制触发），`upgrade`（**Lustre 1.8** 起引入，IGIF 格式） 参数：OI scrub 参数，如 `failout`。 上次完成至今的时间。 上次启动至今的时间。 上次检查点至今的时间。 上次启动位置：上一个 scrub 启动位置。 上次检查点位置。 上次失败位置：待修复的第一个对象的位置。 当前位置。
统计信息	`Checked`：已扫描对象总数。 `Updated`：已修复对象总数。 `Failed`：修复失败的对象总数。 `No Scrub`：已标记为`LDISKFS_STATE_LUSTRE_NOSCRUB`和`skipped`对象的总数。 `IGIF`：IGIF扫描对象总数。

信息类型	详细内容
	Prior Updated：并行 RPC 触发的对象修复个数。
	Success Count：目标上已完成的 OI_scrub 运行总数。
	Run Time：Scrub 运行时间。从指定 MDT 目标开始扫描时间起，不包括检查点之间的暂停时间或失败时间。
	Average Speed：由 Checked 除以 run_time计算所得。
	Real-Time Speed：上一个检查点至今的速率（OI_scrub 正在运行）。
	Scanned：/lost+found目录下已扫描的对象总数。
	Repaired：/lost+found目录下已修复的对象总数。
	Failed：/lost+found目录下扫描和修复失败的对象总数。

36.4.3.2. 通过 procfs查看命名空间的 LFSCK 状态

36.4.3.2.1. 说明

namespace 组件负责第 4 节" 使用 LFSCK 检查文件系统" 中所描述的检查。此组件的procfs接口位于 MDD 层中，名为lfsck_namespace。请按照下面的示例使用lctl get_param查看该组件状态。

LFSCK 命名空间状态输出分为阶段 1 和阶段 2。在阶段 1 中，每个在 MDT 上运行的 LFSCK 主引擎对本地设备进行线性扫描，以确保所有本地对象都完成了检查。但在某些情况下，LFSCK 无法知道对象是否一致，或无法修复其不一致性。因此，阶段 2 将检查有多个硬链接的对象、具有远程父项的对象以及在阶段 1 期间无法验证的其他对象。

36.4.3.2.2. 示例

```
1 lctl get_param -n mdd. FSNAME-MDT_target.lfsck_namespace
```

36.4.3.2.3. 输出

信息类型	**详细内容**
普通信息	名称：lfsck_namespace
	LFSCK namespace magic.
	LFSCK namespace 版本。
	状态：init，scanning-phase1，scanning-phase2，

信息类型	**详细内容**
	completed，failed，stopped,paused，partial， co-failed，co-stopped或 co-paused。
	标志：包括 scanned-once（第一个循环扫描已完成）， inconsistent（已发现一个或多个 FID-in-dirent 或 LinkEA 条目不 一致），upgrade（Lustre 1.8 起，IGIF 格式）
	参数：包括dryrun，all_targets，failout， broadcast，orphan，create_ostobj 和create_mdtobj。
	上次完成至今的时间。
	上次启动至今的时间。
	上次检查点至今的时间。
	上次启动位置：最近开始的扫描的启动位置。
	上次检查点位置。
	上次失败位置：待修复的第一个对象的位置。
	当前位置。
统计信息	Checked Phase1：scanning-phase1期间扫描对象总数。
	Checked Phase2：scanning-phase2期间扫描对象总数。
	Updated Phase1：scanning-phase1期间修复对象总数。
	Updated Phase2：scanning-phase2期间修复对象总数。
	Failed Phase1：scanning-phase1期间修复失败的对象总数。
	Failed Phase2：scanning-phase2期间修复失败的对象总数。
	directories：已扫描目录总数。
	multiple_linked_checked 已扫描的多链接对象总数。
	dirent_repaired：已修复的 FID-in-dirent 条目总数。
	linkea_repaired：已修复的 linkEA 条目总数
	unknown_inconsistency：scanning-phase2期间发现的未定义不 一致总数。

信息类型 **详细内容**

unmatched_pairs_repaired： 已修复的不匹配对总数。

dangling_repaired: 已修复/发现的悬挂名称条目总数。

multi_referenced_repaired： 已修复/发现的多引用名称条目总数。

bad_file_type_repaired： 已修复的错误文件类型的文件总数。

lost_dirent_repaired： 重新添加的丢失名称条目总数。

striped_dirs_scanned: 已扫描的条带化目录（主服务）总数。

striped_dirs_repaired： 已修复的条带化目录（主服务）总数。

striped_dirs_failed： 验证失败的条带化目录（主服务）总数。

striped_dirs_disabled： 失效的条带化目录（主服务）总数。

striped_dirs_skipped: 因丢失主服务 LMV EA 而略过碎片验证的条带化目录（主服务）总数。

striped_shards_scanned： 已扫描的条带化目录碎片（从服务）总数。

striped_shards_repaired： 已修复的条带化目录碎片（从服务）总数。

striped_shards_failed： 验证失败的条带化目录碎片（从服务）总数。

striped_shards_skipped：因 LFSCK 无法知晓从 LMV EA 是否有效而略过名称哈希验证的条带化目录碎片（从服务）总数。

name_hash_repaired: 条带化目录下已修复的名称哈希错误的名称条目总数。

nlinks_repaired： 已修复 nlink 的对象总数。

mul_linked_repaired 已修复的多链接对象总数。

local_lost_found_scanned: /lost+found目录下已扫描的对象总数。

local_lost_found_moved: /lost+found目录下已移至可见目录的对象总数。

local_lost_found_skipped： /lost+found目录下略过的对象总数。

local_lost_found_failed： /lost+found目录下处理失败的对象总数。

Success Count： 目标上完成 LFSCK 运行总数。

Run Time Phase1： scanning-phase1期间 LFSCK 运行时间（去除检

信息类型	**详细内容**
	查点间的暂停时间)。 Run Time Phase2: scanning-phase2期间 LFSCK 运行时间（去除检查点间的暂停时间)。 Average Speed Phase1：由 checked_phase1 除以 run_time_phase1 计算所得。 Average Speed Phase2：由 checked_phase2 除以 run_time_phase2 计算所得。 Real-Time Speed Phase1：上一个检查点至今的速率（LFSCK 正运行 scanning-phase1阶段)。 Real-Time Speed Phase2：上一个检查点至今的速率（LFSCK 正运行 scanning-phase1阶段)。

36.4.3.3. 通过 procfs查看布局的 LFSCK 状态

36.4.3.3.1. 说明

layout 组件负责检查和修复 MDT-OST 不一致性。此组件的procfs接口位于 MDD 层（名为lfsck_layout）和 OBD 层（名为lfsck_layout)。请按照下面的示例使用lctl get_param查看该组件状态。

LFSCK 布局状态输出分为阶段 1 和阶段 2。在阶段 1 中，每个在 MDT/OST 上运行的 LFSCK 主引擎对本地设备进行线性扫描，以确保所有本地对象都完成了检查。在此期间，未被任何 MDT 对象引用的 OST 对象被记录在位图中。在阶段 2 中，位图中的 OST 对象被重新扫描以检查它们是否真的是孤立对象。

36.4.3.3.2. 用例

```
1 lctl get_param -n mdd.
2 FSNAME-
3 MDT_target.lfsck_layout
4 lctl get_param -n obdfilter.
5 FSNAME-
6 OST_target.lfsck_layout
```

36.4.3.3.3. 输出

信息	说明
普通信息	名称：lfsck_layout LFSCK namespace magic. LFSCK namespace 版本。 状态：init，scanning-phase1，scanning-phase2， completed，failed，stopped,paused，partial， co-failed，co-stopped或 co-paused。 标志：包括 scanned-once（第一个循环扫描已完成）， inconsistent（已发现一个或多个 FID-in-dirent 或 LinkEA 条目不一致）， incomplete（某些 MDT 或 OST 没有参与 LFSCK，或未能完成 LFSCK）， crashed_lastid（OST 上的 lastid 文件崩溃，必须重建）。 参数：包括dryrun，all_targets，failout。 上次完成至今的时间。 上次启动至今的时间。 上次检查点至今的时间。 上次启动位置：最近开始的扫描的启动位置。 上次检查点位置。 上次失败位置：待修复的第一个对象的位置。 当前位置。
统计信息	Success Count：目标上完成 LFSCK 运行总数。 Repaired Dangling：在scanning-phase1期间修复的有悬挂引 用的 MDT 对象总数。 Repaired Unmatched Pairs：在scanning-phase1期间修复的 MDT 和 OST 对象不匹配对总数。 Repaired Multiple Referenced：在scanning-phase1期间修复 的有多个引用的 OST 对象总数。 Repaired Orphan：在scanning-phase2期间修复的孤立 OST 对象总数。

信息	说明
	Repaired Inconsistent: 在scanning-phase1期间修复的所有者信息错误的 OST 对象总数。 Repaired Others: 扫描期间修复的其他类型不一致的对象总数。 Skipped : 被略过的对象总数。 Checked Phase1: scanning-phase1期间扫描对象总数。 Checked Phase2: scanning-phase2期间扫描对象总数。 Failed Phase1: scanning-phase1期间修复失败的对象总数。 Failed Phase2: scanning-phase2期间修复失败的对象总数。 Run Time Phase1: scanning-phase1期间 LFSCK 运行时间（去除检查点间的暂停时间)。 Run Time Phase2: scanning-phase2期间 LFSCK 运行时间（去除检查点间的暂停时间)。 Average Speed Phase1: 由 checked_phase1 除以 run_time_phase1计算所得。 Average Speed Phase2: 由 checked_phase2 除以 run_time_phase2计算所得。 Real-Time Speed Phase1: 上一个检查点至今的速率（LFSCK 正运行scanning-phase1阶段)。 Real-Time Speed Phase2: 上一个检查点至今的速率（LFSCK 正运行scanning-phase1阶段)。

36.4.4. LFSCK adjustment 接口

36.4.4.1. 速率控制 36.4.4.1.1. 说明

如下用例所示，使用 lctl set_param 命令更改 LFSCK 速率上限：

36.4.4.1.2. 用例

```
1 lctl set_param mdd.${FSNAME}-${MDT_target}.lfsck_speed_limit=
2 N
```

```
lctl set_param obdfilter.${FSNAME}-${OST_target}.lfsck_speed_limit=
N
```

36.4.4.1.3. 值域

值	说明
0	无速率限制（以最大的速率运行）。
正整数	每秒扫描对象的最大值。

36.4.4.2. auto_scrub 36.4.4.2.1. 说明

auto_scrub参数用于控制在 OI 查找期间检测到不一致时是否触发 OI scurb。可按照下面的用例和值的说明进行设置。

如果挂载时检测到文件级备份，也可使用noscrub禁用 OI scrub。如果指定了noscrub安装选项，则auto_scrub也会随之被禁用，即使检测到 OI 不一致也不会触发 OI scrub。auto_scrub可在挂载后使用以下命令重新启用。挂载后手动启动 LFSCK 有助于更好地控制启动条件。

36.4.4.2.2. 用例

```
lctl set_param osd_ldiskfs.${FSNAME}-${MDT_target}.auto_scrub=N
```

其中，N 为整数，值域如下表所示。

36.4.4.2.3. 值域

值	说明
0	不自动启动 OI。
正整数	OI 查找期间检测到不一致则自动启动 OI Scrub。

注意

Lustre 2.5 及以上版本支持使用-p选项使set_param永久生效。

第三十七章 Lustre 文件系统调试

37.1. 诊断调试工具

各种诊断和分析工具都可用于调试 Lustre 软件的问题。其中一些由 Linux 发行版提供，另一些则是由 Lustre 开发项目提供。

37.1.1. Lustre 调试工具

以下的内核调试的机制已被整合到 Lustre 软件中：

- **Debug logs** － Lustre 内部调试消息将被输出至循环调试缓冲区（错误消息通常输出至系统日志或控制台）。Lustre 调试日志的条目由`/proc/sys/lnet/debug`设置的掩码控制。日志默认大小为每个 CPU 5MB。系统繁忙时可增加日志大小。当缓冲区填满时，旧的信息将被丢弃。
- **`lctl get_param debug`** - 该命令显示当前的调试掩码，用于分隔写入内核日志的调试信息。
- **`lctl debug_kernel file`** - 将 Lustre 内核调试日志以 ASCII 文本的形式转储至指定文件中，以便进一步调试和分析。
- **`lctl set_param debug_mb=size`** - 该命令设置了 Lustre 内核调试缓冲区的最大大小，单位为 MiB。
- **Debug daemon** －该调试进程用于控制将调试信息连续记录到用户空间的日志文件中。

Lustre 软件也提供了以下工具：

- **lctl** －此工具与 debug_kernel 选项一起使用，可手动转储 Lustre 调试日志或处理已被自动转储的调试日志。有关 lctl 工具的更多信息，请参见本章第 2.2 节" 使用 lctl 工具查看调试消息"。
- Lustre 子系统声明 - 内核中的恐慌式断言（LBUG）会导致 Lustre 文件系统将调试日志转储到文件`/tmp/lustre-log.*timestamp*`中，以便重启后还能进行检索。
- *`lfs`*－此实用程序提供对 Lustre 文件的布局以及其他与用户相关的信息的访问。更多关于`lfs`的信息请参考第 40.1 章节 ``lfs"。

37.1.2. 扩展调试工具

本节中介绍的工具一部分来自于 Linux 内核，一部分来自于外部网站。

37.1.2.1. 管理和开发工具 Standard Linux Distribution 中提供了一些常用调试工具：

- **strace** －该工具允许跟踪系统调用。
- **/var/log/messages** － `syslogd`将严重的致命错误输出至此日志。
- **Crash dumps** －在启用了 Crash dumps 的内核中，sysrq c 生成 crash 转储信息，Lustre software 在其上添加日志信息（日志的最后 64KB）并输出至控制台。
- **debugfs** －交互式文件系统调试器。

以下日志记录和数据收集工具负责收集信息用于调试 Lustre 内核问题：

- **kdump** －用于调试运行有 Red Hat Enterprise Linux 系统的 Linux 内核 crash 工具。有关 kdump的更多信息，请参阅 Red Hat knowledge base 的相关文章 How to troubleshoot kernel crashes, hangs, or reboots with kdump on Red Hat Enterprise Linux。下载 kdump，可通过yum install kexec-tools安装 RPM 包。
- **netconsole** －在 UDP 启用内核级网络日志记录。系统需求（SysRq）允许用户通过 netconsole收集相关数据。
- *wireshark* －网络数据包检查工具。它允许调试各种 Lustre 节点之间发送的信息。该工具建立在tcpdump之上，可以读取由它生成的数据包转储。wireshark 2.6.0 版本后包含一些分解 LNet 和 Lustre 协议的插件。有关更多详细信息，另请参阅 Wireshark Website 。

37.1.2.2. 开发工具 本节介绍的工具对于在开发环境中调试 Lustre 文件系统可能很有用。

大众最感兴趣的可能是：

- *leak_finder.pl*－ Lustre 软件提供的这个程序对于在代码中查找内存泄漏非常有用。

虚拟机通常用于创建独立的开发和测试环境。一些常用的虚拟机有：

- **VirtualBox Open Source Edition** －为所有主要平台提供企业级虚拟化功能，可在VirtualBox免费获取。
- **VMware Server** －可作为入门软件的虚拟化平台，可在 Download VMware Server免费获取。
- **Xen** －具有类似于 VMware Server 和 Virtual Box 虚拟化功能的半虚拟化环境。不同的是，Xen 允许使用经过修改的内核来提供接近本机的性能，因此也具备模拟共享存储的能力。更多信息请查看xen.org。

更有多种调试器和分析工具可供使用，包括：

- **kgdb** － Linux 内核源代码级调试器kgdb可与 GNU Debugger gdb 一起用于调试 Linux 内核。更多有关 kgdb 和 gdb的使用介绍，请参阅《*Red Hat Linux 4 Debugging with GDB* guide》中的第六章 Chapter 6. Running Programs Under gdb。
- **crash** －用于在系统发生恐慌、锁定或无响应时分析保存的故障转储数据 (crash dump)。有关使用 crash 分析 crash dump 的更多信息，请参阅：
- 作者关于如何使用 crash 的概述：白皮书 Red Hat Crash Utility

37.2. Lustre 调试过程

以下过程对于调试 Lustre 文件系统的管理员或开发人员可能很有用。

37.2.1. 了解 Lustre 调试消息格式

Lustre 调试消息按发起子系统、消息类型和在源代码中所处位置来分类。有关子系统和消息类型的列表，请参见本章第 2.1.1 节"Lustre 调试消息"。

注意

有关子系统和调试消息类型的最新列表，请参阅 Luster 软件树中的 `libcfs/include/libcfs/libcfs_debug.h`。

组成 Lustre 调试消息的元素在稍后的第 2.1.2 节"Lustre 调试消息的格式" 中进行了描述。

37.2.1.1. Lustre 调试消息 每个 Lustre 调试消息都含有标签纪录了发起子系统、消息类型和在源代码中所处的位置。使用的子系统和调试类型如下所示：

- 标准子系统

mdc, mds, osc, ost, obdclass, obdfilter, llite, ptlrpc, portals, lnd, ldlm, lov.

- 调试类型

类型	说明
trace	入口/出口标记
dlmtrace	锁定相关的信息
inode	
super	
malloc	分配或释放内存的相关消息
cache	缓存相关消息
info	普通消息
dentry	内核空间缓存处理
mmap	内存映射的 IO 接口
page	页缓存和块数据传输
info	杂项信息

类型	说明
net	调试相关的 LNet 网络
console	输出到控制台的重大系统事件
warning	输出到控制台的重大但不致命的异常
error	输出到控制台的关键错误信息
neterror	重大 LNet 错误信息
emerg	输出到控制台的致命系统错误
config	配置和设置，默认启动
ha	故障切换及恢复相关消息，默认启动
hsm	分层空间管理
ioctl	IOCTL 相关信息，默认启动
layout	文件布局处理 (PFL, FLR, DoM)
lfsck	文件系统一致性检查，默认启动
other	其他调试信息
quota	空间统计和管理
reada	客户端读管理
rpctrace	远程请求/应答跟踪和调试
sec	安全性，Kerberos，共享密钥处理
snapshot	文件系统快照管理
vfstrace	内核 VFS 接口操作

-

37.2.1.2. Lustre 调试信息格式 Lustre 软件使用`CDEBUG()`和`CERROR()`宏来打印调试/错误消息。例如，`CDEBUG()`宏使用函数`libcfs_debug_msg()` (`libcfs/libcfs/tracefile.c`)。消息格式如下所示：

描述	参数
subsystem	800000

描述	参数
debug mask	000010
smp_processor_id	0
seconds.microseconds	1081880847.677302
stack size	1204
pid	2973
host pid (UML only) or zero	31070
(file:line #:function_name())	(obd_mount.c:2089:lustre_fill_super())
debug message	kmalloced '*obj': 24 at a375571c (tot 17447717) |

37.2.1.3. Lustre 调试消息缓冲区 Lustre 调试消息保存在缓冲区中，由debug_mb参数（lctl get_param debug_mb）指定最大缓冲区大小（以 MB 为单位）。缓冲区是循环的，也就是说达到所分配的缓冲区得限制时，旧的消息被覆盖。

37.2.2. 使用 lctl 工具查看调试信息

lctl 工具允许根据子系统和消息类型对调试消息进行过滤，从而从内核调试日志中提取对故障排除有用的信息。

你可以使用 lctl 工具来：

- 获取所有类型和子系统的列表：

```
lctl > debug_list subsystems|types
```

- 过滤调试日志：

```
lctl > filter subsystem_name|debug_type
```

注意

当lctl过滤时，它会从显示的输出中删除不需要的行，但这并不影响内核内存中调试日志的内容。因此，您可以使用不同的过滤级别多次输出日志，而不必担心丢失数据。

- 显示属于特性类型或子系统的调试信息：

```
lctl > show subsystem_name|debug_type
```

debug_kernel从内核日志中提取数据，对其进行适当的过滤，并根据指定的选项显示或保存数据。

```
lctl > debug_kernel [output filename]
```

如果调试是在用户模式 Linux（UML）上完成的，那么您可能希望把日志保存在主机上以供日后使用。

- 如果您已将调试日志保存到磁盘（可能是从 crash 转储的），请在磁盘上过滤日志：

```
lctl > debug_file input_file [output_file]
```

在调试会话期间，您可以向日志添加标记或中断：

```
lctl > mark [marker text]
```

标记文本默认为调试日志中的当前日期和时间（类似于以下示例）：

```
DEBUG MARKER: Tue Mar 5 16:06:44 EST 2002
```

- 彻底刷新内核调试缓冲区：

```
lctl > clear
```

注意

使用lctl显示的调试消息也受内核调试掩码的限制。可添加过滤器。

37.2.2.1. lctl 运行范例 以下是使用lctl 运行的范例：

```
bash-2.04# ./lctl
lctl > debug_kernel /tmp/lustre_logs/log_all
Debug log: 324 lines, 324 kept, 0 dropped.
lctl > filter trace
Disabling output of type "trace"
lctl > debug_kernel /tmp/lustre_logs/log_notrace
Debug log: 324 lines, 282 kept, 42 dropped.
lctl > show trace
Enabling output of type "trace"
lctl > filter portals
Disabling output from subsystem "portals"
lctl > debug_kernel /tmp/lustre_logs/log_noportals
Debug log: 324 lines, 258 kept, 66 dropped.
```

37.2.3. 将缓冲区内容转储到文件 (debug_daemon)

lctl debug_daemon命令用于将debug_kernel缓冲区连续转储到用户指定的文件。此功能使用内核线程来连续转储来自内核调试日志的消息，使比内核缓冲区大得

多的调试日志得以更长时间保存。

debug_daemon高度依赖于文件系统的写入速度。如果 Lustre 文件系统负载过大且debug_buffer持续写入调试消息，则文件系统写操作可能无法快到可以及时刷新debug_buffer。

用户可以使用lctl debug_daemon命令启动或停止转储debug_buffer到文件的 Lustre 守护进程。

37.2.3.1. lctl debug_daemon 命令 初始化debug_daemon，开始将debug_buffer 转储到文件，使用 root 用户运行：

```
1 lctl debug_daemon start filename [megabytes]
```

调试日志将从内核写入指定的文件名。该文件将被限制指定的兆字节内（可选）。

当输出文件大小超过用户指定文件大小的限制时，守护程序会将数据存在文件的开头。要将转储的文件解码为 ASCII 并按时间对日志条目进行排序，请运行：

```
1 lctl debug_file filename > newfile
```

输出由 lctl 命令进行内部排序：

停止debug_daemon 操作并刷新文件输出，运行：

```
1 lctl debug_daemon stop
```

否则，debug_daemon将作为 Lustre 文件系统关闭过程的一部分关闭。用户可以在停止命令发出后使用 start 命令重新启动debug_daemon。

下面是一个使用debug_daemon和lctl的交互模式将调试日志转储到一个 40 MB 的文件的示例。

```
1 lctl
```

```
1 lctl > debug_daemon start /var/log/lustre.40.bin 40
```

```
1 run filesystem operations to debug
```

```
1 lctl > debug_daemon stop
```

```
1 lctl > debug_file /var/log/lustre.bin /var/log/lustre.log
```

要启动另一个不限制文件大小的守护进程，请运行：

```
1 lctl > debug_daemon start /var/log/lustre.bin
```

文本消息 *** End of debug_daemon trace log *** 将会在每个输出文件末尾显示。

37.2.4. 写入内核调试日志的控制信息

`lctl set_param subsystem_debug=subsystem_mask` 和 `lctl set_param debug=*debug_mask`用于确定将哪些信息写入调试日志。`subsystem_debug`掩码根据代码的功能区域（例如 lnet，osc 或 ldlm）确定写入日志的信息。调试掩码根据消息类型（如 info，error，trace 或 malloc）控制信息。使用 `lctl debug_list types` 命令查看调试掩码的完整列表。

完全关闭 Lustre 调试功能：

```
lctl set_param debug=0
```

完全打开 Lustre 调试功能：

```
lctl set_param debug=-1
```

列出所有可能的调试掩码：

```
lctl debug_list types
```

记录与网络通信有关的消息：

```
lctl set_param debug=net
```

记录当前调试标志以及网络通信相关的消息：

```
lctl set_param debug=+net
```

不再记录网络通信中变化的调试标志：

```
lctl set_param debug=-net
```

内核调试日志写入的各种选项可在 `libcfs/include/libcfs/libcfs.h`中找到。

37.2.5. 使用 `strace`进行故障排除

Linux 发行版提供的实用程序 `strace` 可以通过拦截进程所做的所有系统调用并记录系统调用名称、参数和返回值来跟踪系统调用。

在程序中调用 `strace` ，请输入：

```
$ strace program [arguments]
```

有时，系统调用可能会分叉成子进程。在这种情况下，可使用`strace`的`-f`选项来跟踪子进程：

```
$ strace -f program [arguments]
```

将`strace`输出重定向到文件，请输入：

```
$ strace -o filename program [arguments]
```

使用-ff选项和-o将跟踪输出保存在filename.pid中，其中pid是被跟踪进程的进程 ID。使用-ttt选项为strace输出中的所有行加上时间戳，以便它们可以与 Luster 内核调试日志中的操作相关联。

37.2.6. 查看磁盘内容

在 Lustre 文件系统中，元数据服务器上的 inode 包含存储有文件条带信息的扩展属性（EA）。EA 包含所有对象 ID 及其位置（即存储它们的 OST）的列表。可以使用lfs工具通过getstripe子命令获取给定文件的相关信息。使用相应的lfs setstripe命令为新文件或目录指定条带属性。

lfs getstripe命令将 Lustre 文件名作为输入，并列出构成此文件一部分的所有对象。要获取 Lustre 文件系统中文件/mnt/testfs/frog的这些信息，请运行：

```
$ lfs getstripe /mnt/testfs/frog
lmm_stripe_count:   2
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  2
        obdidx           objid          objid           group
             2          818855        0xc7ea7               0
             0          873123        0xd52a3               0
```

e2fsprogs包中提供了debugfs工具。它可以用于ldiskfs文件系统的交互式调试，检查文件系统的状态信息或修改文件系统中的信息。在 Lustre 文件系统中，属于文件的所有对象都存储在 OST 的底层 ldiskfs 文件系统中。文件系统使用对象 ID 作为文件名。可通过对象 ID 使用debugfs工具从不同的 OST 获取所有对象的属性。

以下是上例中的/mnt/testfs/frog文件的运行模版：

```
$ debugfs -c -R "stat O/0/d$((818855 % 32))/818855" /dev/vgmyth/lvmythost2

debugfs 1.41.90.wc3 (28-May-2011)
/dev/vgmyth/lvmythost2: catastrophic mode - not reading inode or group
    bitmaps
Inode: 227649   Type: regular    Mode:  0666   Flags: 0x80000
Generation: 1375019198    Version: 0x0000002f:0000728f
User:  1000   Group:  1000   Size: 2800
File ACL: 0    Directory ACL: 0
```

```
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x4e177fe5:00000000 -- Fri Jul  8 16:08:37 2011
 atime: 0x4d2e2397:00000000 -- Wed Jan 12 14:56:39 2011
 mtime: 0x4e177fe5:00000000 -- Fri Jul  8 16:08:37 2011
crtime: 0x4c3b5820:a37.4117c -- Mon Jul 12 12:00:00 2010
Size of extra inode fields: 28
Extended attributes stored in inode body:
  fid = "08 80 24 00 00 00 00 00 28 8a e7 fc 00 00 00 00 a7 7e 0c 00 00 00
      00 00
 00 00 00 00 00 00 00 00 " (32)
  fid: objid=818855 seq=0 parent=[0x248008:0xfce78a28:0x0] stripe=0
EXTENTS:
(0):63331288
```

37.2.7. 查找 OST 的 Lustre UUID

要确定 OST 磁盘的 Lustre UUID（例如，如果您混淆了 OST 设备上的电缆或 SCSI 总线编号突然变化且 SCSI 设备获得了新名称），可以使用debugfs从 last_rcvd 文件中提取此信息：

```
debugfs -c -R "dump last_rcvd /tmp/last_rcvd" /dev/sdc
strings /tmp/last_rcvd | head -1
myth-OST0004_UUID
```

使用dumpe2fs命令也可以（也更容易）从文件系统标签中提取它：

```
dumpe2fs -h /dev/sdc | grep volume
dumpe2fs 1.41.90.wc3 (28-May-2011)
Filesystem volume name:   myth-OST0004
```

debugfs和dumpe2fs命令在 debugfs(8) 和 dumpe2fs(8) 手册页中有详细说明。

37.2.8. 打印调试消息至控制台

将调试消息转储到控制台 (/var/log/messages)，请在 printk 标志中设置如下调试掩码：

```
lctl set_param printk=-1
```

但这将极大地降低系统的速度。可使用以下选项为特定标志选择性地启用或禁用此功能：`lctl set_param printk=+vfstrace`和`lctl set_param printk=-vfstrace`。

尽管我们强烈建议您运行类似于`lctl debug_daemon`的功能，从而将此数据捕获到本地文件系统以进行故障检测，但您也可以禁用这些警告、错误和控制台消息。

37.2.9. 锁流量跟踪

Lustre 软件为跟踪锁流量提供了特定的调试类型类别。使用：

```
lctl> filter all_types
lctl> show dlmtrace
lctl> debug_kernel [filename]
```

37.2.10. 控制台消息速率限制

由 Luster 打印的一些控制台消息的速率是有限制的。当处理这样的消息时，可能会在随后出现一条消息："Skipped N previous similar message(s)"，其中 N 是跳过的消息的数量。通过设置名为`libcfs_console_ratelimit`的 libcfs 模块参数可完全禁用这种速率限制。禁用控制台消息速率限制，请将以下这行命令添加到`/etc/modprobe.d/lustre.conf`文件中，然后重新加载 Lustre 模块。

```
options libcfs libcfs_console_ratelimit=0
```

使用模块参数`libcfs_console_max_delay`和`libcfs_console_min_delay`可以设置速率限制的控制台消息之间的最小和最大延迟。在`/etc/modprobe.d/lustre.conf`中进行设置，然后重新加载 Lustre 模块。有关 libcfs 模块参数的更多信息可通过`modinfo`获得：

```
modinfo libcfs
```

37.3. Lustre 开发调试

本节中介绍的内容可能对调试 Lustre 源代码的开发人员非常有用。

37.3.1. 在 Lustre 源代码中添加调试功能

调试基础架构提供了许多可在 Lustre 源代码中使用的宏，以帮助进行调试或报告严重错误。

使用这些宏，您需要在文件顶部设置`DEBUG_SUBSYSTEM`变量，如下所示：

```
#define DEBUG_SUBSYSTEM S_PORTALS
```

下表提供了可用的宏列表及其描述。

宏	说明
LBUG()	内核中引起 Lustre 文件系统将其循环日志转储到 /tmp/lustre-log文件的恐慌式断言。该文件可以在重启后检索。LBUG() 将冻结线程以完成恐慌堆栈的捕获。清除线程需重启系统。
LASSERT()	验证给定的表达式为 true，否则将调用 LBUG()。失败的表达式在控制台上输出，但不会显示构成表达式的值。
LASSERTF()	和 LASSERT() 类似，但允许打印无格式消息，如 printf/printk。
CDEBUG()	最基本的、常用的调试宏，它只比标准的 printf() 多一个参数，即调试类型。设置了相应的调试掩码后，该消息将添加到调试日志。用户稍后检索日志进行故障排除时，可以根据此类型进行过滤。如：CDEBUG(D_INFO, "debug message: rc=%d\n", number);。
CDEBUG_LIMIT()	与 CDEBUG() 行为类似，但打印到控制台时对速度进行了限制（消息类型为 D_WARN，D_ERROR 和 D_CONSOLE），这对使用可变调试掩码的消息很有用：CDEBUG(mask, "maybe bad: rc=%d\n", rc);。
CERROR()	在内部使用 CDEBUG_LIMIT（D_ERROR，…），它无条件地将消息打印到调试日志和控制台中。这适合用于严重错误或致命条件。打印到控制台的消息以：LustreError为前缀，并且速率受限，以避免重复播放控制台。CERROR("Something bad happened: rc=%d\n", rc);

宏	说明
CWARN()	与CERROR()行为类似，但消息须加上前缀 Lustre:。这适合重要但非致命的错误。打印到控制台的消息的速率受限。
CNETERR()	与CERROR()行为类似，但如果在调试掩码中设置了D_NETERR，则打印 LNet 的相关错误消息。这适合用于严重的网络错误。打印到控制台的消息的速率受限。
DEBUG_REQ()	打印给定 ptlrpc_request 结构相关消息。 DEBUG_REQ(D_RPCTRACE, req, "Handled RPC: rc=%d\n", rc);
ENTRY	将消息添加到函数的入口以帮助进行调用跟踪（不带任何参数)。使用这些宏时，请使用单个EXIT，GOTO()或RETURN()宏覆盖所有退出条件，以避免调试日志报告函数进入后未退出时出现混淆。
EXIT	标记函数的出口以匹配 ENTRY （不带任何参数)。
GOTO()	标记代码通过goto跳转到函数末尾以匹配 ENTRY，并以有符号和无符号十进制和十六进制格式打印goto标签和函数返回码。
RETURN()	标记函数的出口以匹配 ENTRY，并以有符号和无符号十进制和十六进制格式打印函数返回码。
LDLM_DEBUG() 和 **LDLM_DEBUG_NOLOCK()**	用于跟踪 LDLM 锁操作。这些宏将构建一个精简的跟踪以显示节点上的锁请求。也可使用打印的锁定手柄在客户端和服务器节点之间将这些宏链接起来。
OBD_FAIL_CHECK()	允许将故障点插入 Lustre 源代码中。这对于生成用于实现特定事件序列的回归测试非常有用。与 "lctl set_param fail_loc=fail_loc" 一

宏	说明
	起工作可设置一个特定的故障点，并使用给定的 OBD_FAIL_CHECK()进行测试。
OBD_FAIL_TIMEOUT()	与OBD_FAIL_CHECK()类似。用于模拟挂起、阻塞或繁忙的进程或网络设备。如果命中fail_loc，则OBD_FAIL_TIMEOUT()将等待指定的秒数。
OBD_RACE()	与 OBD_FAIL_CHECK()类似。用于让多个进程同时执行相同的代码来触发锁竞争。第一个命中 OBD_RACE()的进程会休眠，直到第二个进程命中 OBD_RACE()，然后两个进程都将继续。
OBD_FAIL_ONCE	在fail_loc断点上设置的标志，用于限定 OBD_FAIL_CHECK()条件仅能被命中一次。否则，在使用"lctl set_param fail_loc=0" 清除之前，fail_loc将永久存在。
OBD_FAIL_RAND	在fail_loc断点上设置的标志，使 OBD_FAIL_CHECK()随机失效；平均为（1/fail_val）次。
OBD_FAIL_SKIP	在fail_loc断点上设置的标志，使 OBD_FAIL_CHECK()在成功 fail_val次后永久失效或只能再被命中一次，即转变为标志 OBD_FAIL_ONCE。
OBD_FAIL_SOME	在fail_loc断点上设置的标志，使 OBD_FAIL_CHECK()在失效fail_val次后恢复。

37.3.2. 访问ptlrpc请求历史

每个服务负责维护一个请求历史记录，这对于首次出现的故障排除很有用。

ptlrpc是 LNet 上的一个 RPC 协议，它处理状态性服务器，并且具有语义和内置的恢复支持。

ptlrpc请求历史记录的工作原理如下：

1. request_in_callback()添加新请求至服务的请求历史记录。
2. 请求缓冲区空闲时，添加服务请求缓冲区历史列表至缓冲区。
3. 如果缓冲区大小比req_buffer_history_max还大时，则从服务请求缓冲区历史记录中剔除该缓冲区，其请求从服务请求历史记录中删除。

使用服务目录下/proc文件访问和控制请求历史记录：

- req_buffer_history_len

历史记录中当前的请求缓冲区的数量。

- req_buffer_history_max

允许保留的请求缓冲区的最大大小。

- req_history

请求历史。

历史请求包括当前正在处理的" 实时" 请求。req_history 中的每一行看起来如下所示：

```
1 sequence:target_NID:client_NID:cliet_xid:request_length:rpc_phase
    service_specific_data
```

参数	说明
seq	请求序列号
target NID	传入请求的目的 NID
client ID	客户端的 PID 和 NID
xid	rq_xid
length	请求消息大小
phase	新（等待处理或无法解压）解析（解压或处理）完成
svc specific	特定服务的请求打印输出。目前，唯一能做到这一点的服务是 OST（如果消息已成功解压，将打印操作码）

37.3.3. 使用 leak_finder.pl查找内存泄漏

分配内存后，一旦不再需要时必须释放内存，否则将造成内存泄漏。leak_finder.pl程序提供了一种查找内存泄漏的方法。

在运行此程序之前，您必须启用调试功能以收集所有malloc和free条目，运行：

```
1 lctl set_param debug=+malloc
```

随后，完成以下步骤：

1. 使用lctl将日志转储到用户指定的日志文件中（请参见本章第 2.2 节" 使用 lctl 工具查看调试消息")。
2. 在新创建的日志转储上运行leak_finder.pl：

   ```
   perl leak_finder.pl ascii-logname
   ```

   输出为：

```
1 malloced 8bytes at a3116744 (called pathcopy)
2 (lprocfs_status.c:lprocfs_add_vars:80)
3 freed 8bytes at a3116744 (called pathcopy)
4 (lprocfs_status.c:lprocfs_add_vars:80)
```

发现的泄漏显示如下：

```
1 Leak:32bytes allocated at a23a8fc(service.c:ptlrpc_init_svc:144,debug file
    line 241)
```

第三十八章 Lustre 文件系统恢复

38.1. 概述

Lustre 软件提供的系统恢复功能负责处理节点或网络故障，并将集群恢复到一致、高效的状态。由于 Lustre 软件允许服务器对磁盘上文件系统执行异步更新操作（即服务器可以不等待更新同步提交到磁盘就进行回复），因此可能存在客户端内存中的状态比崩溃后服务器可从磁盘恢复的状态还新的情况。

以下几种不同类型的故障可能导致恢复操作：

- 客户端（计算机节点）故障
- MDS 故障（切换）
- OST 故障（切换）
- 瞬态网络分区

对于 Lustre 文件系统，故障和恢复操作都基于连接失败的概念; 即给定连接相关的任何读写失败即视为失败。将在本章第 6 节中介绍的 ``强制恢复" 功能，允许 MGS 在目标发生故障、故障转移或其他中断后并重启时主动通知客户端，以加快恢复速度。

有关 Lustre 文件系统恢复的相关信息，请参见本章第 2 节" 元数据重放"。从损坏的文件系统中恢复的相关内容请参见本章第 3.5 节" 提交共享"。有关命令性恢复的信息，请参见本章第 6 节" 强制恢复"。

38.1.1. 客户端故障

Lustre 文件系统中的客户端故障恢复基于锁定撤销和其他资源，因此幸存的客户端可以不间断地继续工作。如果客户端未能及时响应分布式锁管理器（DLM）的阻塞锁回调或在很长一段时间都未能内与服务器通信（即 ping 无回复），则会将客户端从群集中强制删除（被驱逐）。这使得其他客户端可以获取该死亡客户端锁所阻止的锁，与该客户端关联的资源（文件句柄，导出数据）也将被释放。请注意，此状况可能是由网络分或客户端节点系统故障引起的。第 1.5 节" 网络分区" 对这种情况进行了更详细的描述。

38.1.2. 客户端驱逐

如果服务器认为某客户端表现不正常，它将被逐出。这是为了确保在存在行为不当或故障客户端时整个文件系统继续运行。必须使被驱逐的客户端的所有锁无效，这将导致所有缓存 inode 也变为无效，所有缓存的数据都将被刷新。

客户端被驱逐的原因可能有:

- 未能及时响应服务器请求
- 阻塞锁回调（即客户端持有另一个客户端/服务器想要的锁）
- 锁完成回调（即客户端被授予之前由另一个客户端持有的锁定）
- 锁 glimpse 回调（即客户端被另一个客户端询问对象大小）
- 服务器关闭通知（简化的互操作性）
- 在服务器接收到 RPC 流量时，无法及时 ping 通服务器（指向网络分区）。

38.1.3. MDS 故障（切换）

高可用性（HA）的 Lustre 文件系统操作要求元数据服务器有故障切换配置的对等设备，包括用于 MDT 后备文件系统的共享存储设备。对等设备故障检测、对等设备断

电（STONITH，用于防止其继续修改共享磁盘）以及在备份节点上 Lustre MDS 服务的接管等的实际机制取决于外部 HA 软件（如 Heartbeat）。也可使用单个 MDS 节点进行 MDS 恢复，但此时恢复将花费重启单个 MDS 所需的时间。

启用强制恢复功能，将通知客户端 MDS 重新启动（备份或恢复的主服务器）的消息。客户端可以通过 in-flight 请求超时或空闲时间的 ping 消息来检测 MDS 故障。在这两种情况下，客户端都会连接到新的备份 MDS 并使用元数据重放协议。元数据重放负责确保备份 MDS 重新获取客户端可见但未提交给磁盘的事务产生的状态。

重新连接到新的（或重启的）MDS 在首次装入文件系统时由客户端加载的文件系统配置进行管理。如果已配置故障切换 MDS（使用`mkfs.lustre`或`tunefs.lustre`的`--failnode=`选项），客户端将尝试重新连接到主 MDS 或备用 MDS，直至其中一个响应使故障 MDT 再次可用。此时，客户端开始恢复。有关更多信息，请参见本章第 2 节" 元数据重放"。

事务编号用于确保重放按最初的顺序执行，从而使它们成功地呈现与失败前相同的文件系统状态。此外，客户端将通知新服务器其现有的锁状态（包括尚未授予的锁）。在允许新的非恢复操作执行之前，必须完成所有元数据和锁重放。此外，只允许 MDS 发生故障时连接的客户端在恢复窗口期间进行重新连接，以避免引入可能与先前连接的客户端重放相冲突的状态更改。

如果正在使用多个 MDT，则可以进行主动－主动故障转移（例如，两个 MDS 节点，每个节点主动为同一文件系统的一个或多个 MDT 提供服务）。

38.1.4. OST 故障（切换）

当 OST 故障或客户端出现通信问题时，默认操作是相应的 OSC 进入恢复状态，并阻止该 OST 的 I/O 请求直到 OST 恢复或完成故障切换。可在客户端上以管理方式将 OSC 标记为非活动状态，在这种情况下，与故障 OST 相关的文件操作将返回 IO 错误（`-EIO`）。否则，应用程序将保持等待状态，一直到 OST 恢复或客户端进程被中断（如使用 CTRL-C）。

MDS（通过 LOV）检测到 OST 不可用，则在分配对象给新文件时将跳过它。重新启动 OST 或重新建立与 MDS 的通信时，MDS 和 OST 会自动执行孤立恢复，以销毁属于在 OST 不可用时删除的文件的任何对象。

虽然 OSC 到 OST 操作恢复协议与 MDC 和 MDT 之间的元数据重放协议相同，但通常 OST 会向磁盘同步提交批量写入操作，且每个回复都表明请求已提交、数据无需保存（以用于恢复）。在某些情况下，OST 会在操作（如 Lustre 软件较新版本中的 truncate，destroy，setattr 和 I / O 操作）提交到磁盘以及正常重放和重新发送（包括重新发送批量写入）完成前回复客户端。在这种情况下，客户端会保留内存中可用数据的副本，直到服务器显示写入已提交到磁盘。

要强制执行 OST 恢复，请先卸载 OST，然后重新挂载。如果 OST 在故障之前就连

接到客户端，则在重新挂载之后恢复将启动，客户端重新连接到 OST 并重放其队列中的事务。当 OST 处于恢复模式时，所有新客户端连接都将被拒绝，直到恢复完成。所有先前连接的客户端完成重新连接和重放事务或客户端连接超时，恢复完成。如果连接超时，则等待重连的所有客户端（及其事务）都将丢失。

注意

如果您知道 OST 将无法恢复之前连接的客户端（如客户端已崩溃），则可以使用以下命令手动中止恢复：

```
1 oss# lctl --device lustre_device_number abort_recovery
```

确定 OST 设备编号和名称，请运行`lctl dl` 命令。相关输出示例如下：

```
1 7 UP obdfilter ddn_data-OST0009 ddn_data-OST0009_UUID 1159
```

在此示例中，OST 设备编号为 7，名称为`ddn_data-OST0009`。在大多数情况下，可以使用设备名称代替设备编号。

38.1.5. 网络分区

网络故障可能是暂时的。为避免触发恢复操作，客户端一开始会尝试将所有超时请求重新发送到服务器。如果重新发送也失败，则客户端会尝试重建与服务器的连接。当服务器无理由驱逐客户端时，客户端可在重新连接时检测到无害分区。

如果服务器处理了请求，但删除了答复（即没有返回到客户端），则服务器必须在客户端重新发送请求时重新生成答复，而不是执行两次相同的请求。

38.1.6. 恢复失败

恢复失败则客户端将被服务器驱逐，并且必须参照上面 1.2 节" 客户端驱逐" 中所述，在该服务器相关的已保存状态被刷新后重新连接。以下是导致恢复失败可能的原因：

- 恢复失败
- 如果一个客户端的操作直接依赖于未能参与恢复的另一个客户端的操作，则恢复将失败。否则，基于版本的恢复（VBR）允许对所有连接的客户端进行恢复，并仅驱逐丢失的客户端。
- 手动中止恢复
- 管理员手动执行驱逐操作

38.2. 元数据重放

高可用性的 Lustre 文件系统操作要求 MDS 配置有用于故障转移的对等设备，包括用于 MDT 后备文件系统的共享存储设备。当客户端检测到 MDS 故障时，它会连接到新的 MDS 并使用元数据重放协议来重放其请求。

元数据重放可确保完成故障转移的 MDS 重新收集客户端可见但未提交给磁盘的事务状态信息。

38.2.1. XID 编号

客户端发送的每个请求都包含一个 XID 编号，该编号是客户端唯一的单调递增的 64 位整数。XID 会进行初始化定义，因此在重启后重新连接到同一服务器的同一客户端节点具有相同的 XID 序列的可能性非常小。客户端使用 XID 对其发送的所有请求进行排序，直到请求被分配事务编号。XID 还可用于重新生成回复，以唯一地标识服务器上的每个客户端的请求。

38.2.2. 事务编号

服务器会分配一个事务编号给服务器处理的每个涉及状态更改（元数据更新、文件打开、写入等，具体取决于服务器类型）的客户端请求。该事务编号对于目标来说是唯一的，工作于服务器范围，是单调递增的 64 位整数。每个文件系统修改请求的事务编号将与客户端请求的回复一起发回客户端。事务编号允许客户端和服务器明确地对每个文件系统更改进行排序，以便需要时进行恢复。

发送给客户端的每个回复（无论请求类型如何）还包含最后提交事务的编号，显示了提交给文件系统的事务编号的最大值。Lustre 软件使用的`ldiskfs`和`ZFS`后备文件系统确保了在随后的磁盘操作开始之前将早期磁盘操作提交到磁盘，最后提交的事务的编号还指示了任何具有更小事务编号的请求已被提交到磁盘。

38.2.3. 重放和重发

恢复 Lustre 文件系统可以分为两种不同类型的操作：重放（replay）和重发（resend）。

重放操作针对的是客户端已从服务器收到操作成功的回复的那些操作。在服务器重启后，需要以和服务器故障前报告的完全相同的方式重新执行这些操作。只有在服务器发生故障时才能进行重放，否则内存中并不会丢失任何状态。

重发操作针对的是客户端从未收到回复的那些操作，也就是说客户端并不知道它们的最终状态。客户端按照 XID 的顺序再次向服务器发送未应答的请求，并等待每个请求的回复。在某些情况下，重新发送的请求已由服务器处理并提交到磁盘（可能还提交了相关操作），则服务器将重新生成丢失的回复。在其他情况下，服务器根本没有收到请求（网络中断会发生这种状况），将像处理任何正常请求一样重新处理这些请求。服务器也可能收到了请求，但在发送故障前无法回复或提交到磁盘。

38.2.4. 客户端重放列表

在服务器发生故障的情况下，进行服务器状态恢复（重放）可能需要所有文件系统修改请求。所收到的来自服务器的包含比最后提交的事务编号更大的事务标号的回复将被保留重放列表中，每个服务器都有一个这样的重放列表。也就是说，当从服务器接收到回复时，检查它是否具有比先前的最后提交的事务编号还大的事务编号。大多数具有较小事务编号的请求可以安全地从重放列表中删除。请注意，" 打开请求" 在这里是一个例外，它需要保存在重放列表中直到文件关闭，以便 MDS 可以正确引用 open-unlinked 文件的计数。

38.2.5. 服务器恢复

如果服务器未完全关闭，则会进入恢复状态。服务器启动时，如果先前连接的客户端在`last_rcvd`文件中有任何客户端条目，则服务器进入恢复模式，等待这些客户端重新连接并开始重放或重发其请求。这将允许服务器重建已暴露给客户端（成功完成的请求）但在故障前未提交到磁盘的状态。

不进行任何客户端连接尝试的情况下，服务器将无限期地等待客户端重新连接。这旨在处理服务器存在网络问题时客户端无法重连或需要反复重启服务器来解决硬件或软件问题的情况。一旦服务器检测到客户端的连接尝试（新客户端或先前连接的客户端），无论先前连接的客户端是否可用，恢复计时器都将启动并强制在有限时间内完成恢复。

如果`last_rcvd`文件中没有客户端条目，或管理员手动中止恢复，则服务器不会等待客户端重新连接，而是允许所有客户端进行连接。

当客户端连接时，服务器从每个连接处收集信息以确定需要多长时间来完成恢复。每个客户端将报告其连接 UUID，服务器在`last_rcvd`文件中查找此 UUID 来确定此客户端之前是否已连接。如果没有，将拒绝此客户端的连接直到恢复完成。每个客户端会报告最近一次的事务，以便服务器获知何时所有事务完成重放。客户端还会报告先前等待请求完成的时间，用于帮助服务器估计某些客户端可能需要多长时间来检测服务器故障并重新连接。

如果客户端在重放期间超时，则会尝试重新连接。如果客户端无法重新连接，则`REPLAY`失败并返回`DISCON`状态。客户端可能会在`REPLAY`期间频繁地超时，因此重新连接不应该使已经很慢的进程延迟过久。我们可以通过在重放期间增加超时时间来缓解这种情况。

38.2.6. 请求重放

如果客户端先前已连接，则会从服务器获得响应，得知服务器正在进行恢复，并获知磁盘上最后提交的事务编号。然后，客户端便可以遍历其重放列表并使用此最后提交的事务编号来删除任何先前提交的请求。它按照事务编号的顺序向服务器重放任何较新

的请求，一次一个，收到服务器的回复后再重放下一个请求。

重放列表上的" 打开请求" 的事务编号可能小于服务器上次提交事务的编号。服务器将立即处理这些打开请求，然后再按照事务编号顺序处理来自客户端的重放请求。从最后提交事务的编号开始，确保状态在磁盘上以与故障之前完全相同的方式更新。在处理每个重放请求时，最后提交的事务编号将递增。如果服务器从客户端收到大于当前的最后提交事务编号的重放请求，则该请求会被搁置，直到其他客户端发起干预事务。服务器以这种方式按照先前在服务器上执行的相同顺序重放请求，直到所有客户端无请求可重放或序列中存在间隙。

38.2.7. 重放序列中的间隙

在某些情况下，回复序列中可能会出现间隙。这可能是回复丢失引起的，即请求已处理并提交到磁盘，但客户端未收到回复；也可能是由于部分网络故障或客户端崩溃导致回复无法发送至客户端造成的。

在所有客户端都已重新连接但重放序列仍存在间隙的情况下，唯一的可能是服务器处理了一些请求但是回复丢失了。客户端必须在其重发列表中包含这些请求，以便恢复完成后进行重发。

如果所有客户端都未重新连接，则故障客户端可能有不会再被重放的请求。VBR 功能可用于确定间隙之后的请求是否可以被安全地重放。文件系统中的每个条目（MDS inode 或 OST 对象）将在磁盘上存储被修改的最后事务编号。来自服务器的每个回复都包含它所作用的对象先前的版本号。在 VBR 重放期间，服务器将重新发送请求中的先前版本号与当前版本号进行匹配。如果版本匹配，则请求将作用于对象，且可以安全地进行重放。有关更多信息，请参见本章第 4 节" 基于版本的恢复"。

38.2.8. 锁恢复

如果所有请求都成功重放且所有客户端都重新连接，客户端会进行锁重放。每个客户端都会发送它从此服务器获取的每个锁的信息以及其状态（无论何时被授予、什么模式、什么属性等），随后恢复成功完成。

目前，Lustre 软件不进行锁验证，而是信任客户端呈现准确的锁状态。这不会带来任何安全问题，因为 Lustre 软件版本 1.x 客户端的其他信息（如用户 ID）在正常操作期间也是可信任的。

在重放了所有已保存的请求和锁之后，客户端发送一个`MDS_GETSTATUS`请求并设置`last-replay`标志。在所有客户端都完成重放（发送带有相同标记的 getstatus 请求）前，该请求的回复将被阻止，以便客户端在恢复完成之前不发送非恢复请求。

38.2.9. 请求重发

一旦服务器上恢复了所有先前共享的状态（目标文件系统更新至客户端缓存，且服务器已重建客户端持有的锁），客户端就可以重新发送任何之前没有得到答复的请求。该处理与正常请求的处理类似，在一些情况下，服务器可以进行重新生成回复。

38.3. 重建回复

当回复丢失时，MDS 需要能够在原始请求被重新发送时重建回复。在保持锁定系统的完整性的同时，必须在不重复任何非幂等操作的情况下完成此操作。MDS 故障切换时，用于重建回复的信息必须在与磁盘上进行组合或嵌套事务的序列化。

38.3.1. 所需状态

对于大多数请求来说，服务器在`last_rcvd`文件中存储三种数据就足够了：

- 请求的 XID
- 产生的事务编号（如果有的话）
- 结果代码 (`req->rq_status`)

对于" 打开请求" 来说，请求的处置信息也必须保存。

38.3.2. 重建" 打开请求" 的回复

" 打开请求" 的回复最多包含三条信息（除了" 请求日志" 的内容）：

- 文件句柄
- 锁句柄
- `mds_body` 以及所创建文件的相关信息（`O_CREAT`）

处置、状态和请求数据（由客户端重新发送的完整数据）足以确定所授予的是哪种类型的锁句柄、是否创建了打开文件句柄，以及应在`mds_body`中描述的资源。

38.3.2.1. 查找文件句柄 文件句柄可以在请求的 XID 和每个导出的打开文件句柄列表中找到。

38.3.2.2. 查找资源/FID 文件句柄包含资源/FID。

38.3.2.3. 查找锁句柄 可以通过遍历相应远程文件句柄（显示在重发的请求中）下资源所授予的锁列表来查找锁句柄。验证锁的模式是否正确（通过执行上面的处置/请求/状态分析来确定），以及是否被授予至适当的客户端。

38.3.3. 客户端上的多个回复数据

从 **Lustre 2.8** 起，MDS 可为每个客户端保存多个回复数据。回复数据存储在 MDT 的内部文件`reply_data`中。除了请求的 XID、事务编号、结果代码和打开请求的处置信息外，得益于`last_rcvd`文件的内容，回复数据包含了可用于标识客户端的版本号。

38.4. 基于版本的恢复

可使用基于版本的恢复（VBR）功能来处理在恢复期间无法重放的客户端请求（RPC），从而提高 Lustre 文件系统的可靠性。

在无 VBR 功能的之前的 Lustre 版本中，如果 MGS 或 OST 发生故障将触发恢复操作，客户端会尝试重放其请求。客户端只允许按顺序重放 RPC。如果特定客户端无法重播其请求，那么这些请求以及后续序列中的客户端请求都将丢失。由于必须等待更早的 RPC 完成，" 下游" 客户端将永远不会重放它们的请求。最终，恢复期将超时（因此组件可以接受新请求），导致一些客户被驱逐，其请求和数据丢失。

使用 VBR 后，恢复机制不会导致客户端或其数据丢失，这是因为对 inode 版本的更改进行了跟踪，更多客户端能够重新集成到集群中。使用 VBR 进行 inode 跟踪：

- 每个 inode 存储一个版本号，即 inode 更改的最后事务编号（transno）。
- 当要更改 inode 时，inode 的操作前版本号将被保存在客户端的数据中。

- 客户端保留操作前 inode 版本号和操作后版本号（事务编号），并在服务器发生故障后发送它们。
- 如果操作前后版本匹配，则重放请求。在请求中修改的所有 inode 上分配操作后的版本号。

注意

因为操作中可能涉及多个 inode，RPC 最多可包含四个预操作版本。进行" 重命名" 操作时，可以修改四个不同的 inode。

在正常操作期间，服务器：

- 更新给定操作中涉及的所有 inode 的版本。
- 将旧的和新的 inode 版本返回给客户端。

当恢复正在进行时，VBR 遵循以下步骤：

1. 只有当受影响的 inode 有与原始执行事务时版本相同时，VBR 才允许客户端重放事务（即使因客户端丢失导致事务序列存在间隙）。
2. 服务器尝试执行客户端发起的每个事务（即使重新集成失败）。

3. 重放完成后，客户端和服务器会检查是否有事务因 inode 版本不匹配而失败。如果版本匹配，则客户端会收到成功完成重新集成的消息。如果版本不匹配，则客户端被驱逐。

VBR 恢复对用户完全透明。如果集群在服务器恢复期间有多个客户端丢失，则可能会延长恢复时间。

38.4.1. VBR 消息

VBR 功能内置于 Lustre 文件系统恢复功能，它无法被禁用。以下是可能会显示的一些 VBR 消息：

```
1 DEBUG_REQ(D_WARNING, req, "Version mismatch during replay\n");
```

该消息提示了客户端被驱逐的原因。无需任何操作。

```
1 CWARN("%s: version recovery fails, reconnecting\n");
```

该消息提示了恢复失败的原因。无需任何操作。

38.4.2. VBR 使用建议

一般来说，VBR 在不与其他客户端共享数据的客户端上会成功。因此，为了更可靠地使用 VBR，应尽可能将客户端的数据存储在自己的目录中。在这种情况下，即使其他客户端丢失，VBR 也可以恢复这些客户端。

38.5. 共享提交

共享时提交（Commit On Share，COS）功能增加了 Lustre 文件系统恢复的可靠性，因为该功能可以防止被驱逐的客户端连带着引起其他客户端被驱逐。启用 COS 后，如果一些 Lustre 客户端在服务器重启或故障后错过了恢复窗口，剩下的客户端不会因此被驱逐。

注意

COS 功能默认启用。

38.5.1. COS 的工作原理

为了说明 COS 是如何工作的，让我们先看一下没有 COS 的恢复方式。在服务重启后，MDS 将启动并进入恢复模式。客户端开始重新连接并重新执行他们未提交的事务。客户端可以独立地重新执行事务，只要这些事务不相互依赖（一个客户端的事务不依赖另一个客户端的事务）。MDS 能够通过基于版本的恢复（Version-based Recovery）这一功能来确定一个事务是否依赖于另一个事务。

如果客户端事务之间存在着依赖关系（例如，创建和删除同一个文件），而其中一个或多个客户端没有及时地重新连接，那么这些客户端可能因为它们的事务依赖于被驱逐的客户端的事务，因而跟着被驱逐。而驱逐这些客户端又回导致更多的客户端被驱逐，从而导致客户端接二连三地被级联驱逐。

COS 通过消除客户端之间的事务依赖来解决级联驱逐的问题。如果另一个客户端的事务依赖于此客户端的某事务，COS 会确保将该事务提交到磁盘。由于客户端不会依赖于其他客户端的未提交事务，因此客户端可以独立地重放其请求而不会被驱逐。

38.5.2. COS 调试

可以使用`mdt.commit_on_sharing`可调参数（0/1）来启用或禁用 COS。此参数可以使用`lctl set/get_param`或`lctl conf_param`命令在创建 MDS（`mkfs.lustre`）或 Lustre 文件系统处于活动状态时进行设置。

在文件系统创建时为 COS（禁用/启用）设置默认值，请使用：

```
1 --param mdt.commit_on_sharing=0/1
```

在文件系统运行时启用或禁用 COS，请使用：

```
1 lctl set_param mdt.*.commit_on_sharing=0/1
```

注意

启用 COS 可能会导致 MDS 执行大量同步磁盘操作，从而损害性能。可将`ldiskfs`日志放在低延迟外部设备上来提高文件系统性能。

38.6. 强制恢复

大规模 Lustre 文件系统在其生命周期中难免遇到服务器硬件故障等问题。在发生这种故障后，服务能够及时恢复显得尤为重要。高可用软件可自动将存储目标服务转移到备份服务器上。客户端可以通过 RPC 超时来检测服务器故障的出现，而 RPC 超时时间必须随着系统规模的扩大而进行调整，以防止在负载较大的情况下错误地判定服务器死亡。祈使式恢复（Imperactive Recovery）的目的是，通过主动告知客户端服务器发生了故障，来缩短恢复窗口，并由此最大限度地减少目标停机时间，从而提高整个系统的可用性。

祈使式恢复并没有覆盖以前的恢复机制，当祈使式恢复启用时，仍然可以在集群中进行基于客户端超时的恢复，因为每个客户端仍然可以独立地从目标上断开和重新连接。在支持祈使式恢复的客户端和不支持祈使式恢复的客户端混合连接到 OST 或 MDT 的情况下，祈使式恢复不能缩短服务器的恢复超时窗口，因为不能确保所有客户端都及时收到了服务器重新启动的通知。即使在这样的混合环境中，完成恢复的时间也可能缩短，因为支持祈使式恢复的客户端仍然会接到通知，及时重新连接到服务器，一旦最后一个不支持祈使式恢复的客户端检测到服务器故障，就能完成恢复。

38.6.1. MGS 的作用

在祈使式恢复机制中，MGS 以目标状态表（Target Status Table）的形式持有关于 Lustre 目标的额外信息。在 MGS 上，每当注册一个目标时，在该表中就要增加相应的条目来识别该目标。该条目包含了 NID 信息，以及目标的状态/版本信息。当客户端挂载文件系统时，会以 Lustre 配置日志的形式，缓存并锁定该表的一个副本。当目标重启时，MGS 撤销了客户端的锁，强制所有客户端重新加载该表。所有的新目标将获得一个新的版本号，客户端检测到了版本号的更新，就会重新连接到重启的目标上。祈使式恢复要能成功将服务器的重启通知给所有客户端，有赖于客户端已经在 MGS 上的注册好，而在 MGS 重启的情况下，因为没有其他节点可以通知客户端，所以 MGS 在第一次启动时将禁用 IR 一段时间。这个时间间隔是可以配置的，将在 6.2 节"IR 调试" 中进行介绍。

由于 MGS 在恢复中至关重要，因此强烈建议 MGS 节点与 MDS 分开。如果 MGS 位于 MDS 节点上，那么在 MDS/MGS 故障的情况下，MDS 的重启将无法使用祈使式恢复机制，客户端只能始终对 MDS 使用基于超时的恢复。在 OSS 故障和恢复的情况下，仍然会使用祈使式恢复机制。

不幸的是，MGS 无法知晓有多少客户端已成功收到通知，或某个特定客户端是否已收到重新启动的目标信息。MGS 唯一能做到的就是，告诉目标所有客户端都具有祈使式恢复能力，因此没有必要等所有客户端完成重新连接。出于这个原因，我们仍需使用目标端的超时策略，但是此超时值可能比正常恢复的超时值短得多。

38.6.2. IR 调试

IR 的参数存在默认设置，这意味无需额外进行配置它就可以工作。但是，默认参数仅适用于通用配置。以下讨论了 IR 的配置项。

38.6.2.1. ir_factor Ir_factor 用于控制目标的恢复窗口。如果启用了 IR，则可通过以下方式计算重新启动放入目标的恢复超时窗口：`new timeout = recovery_time * ir_factor / 10`。

Ir_factor 必须在 [1,10] 范围内，其默认值为 5。

为目标`testfs-OST0000`将 IR 超时设置为正常恢复超时时间的 80%：

```
1 lctl conf_param obdfilter.testfs-OST0000.ir_factor=8
```

注意

如果此值对于系统来说太小，则可能导致不必要的客户端驱逐。可使用`lctl get_param`以标准方式读取当前参数值：

```
1 # lctl get_param obdfilter.testfs-OST0000.ir_factor
2 # obdfilter.testfs-OST0000.ir_factor=8
```

38.6.2.2. 禁用 IR 可以通过挂载选项手动禁用 IR。例如，通过以下方式在 OST 上禁用 IR：

```
# mount -t lustre -onoir /dev/sda /mnt/ost1
```

IR 也可在客户端上通过同样的挂载选项禁用：

```
# mount -t lustre -onoir mymgsnid@tcp:/testfs /mnt/testfs
```

注意

当通过这种方式停用某个客户端的 IR 时，MGS 将停用整个集群的 IR。启用 IR 的客户端仍将获得目标重启的通知，但不允许目标缩短恢复窗口。

您还可以通过将"state = disabled" 写入控制 procfs 条目来全局禁用 MGS 上的 IR。

```
# lctl set_param mgs.MGS.live.testfs="state=disabled"
```

以上命令将禁用文件系统 testfs 的 IR。

38.6.2.3. 查看 IR 状态－ MGS 您可以从 MGS 上获取 IR 状态信息。我们来看下面的例子：

```
[mgs]$ lctl get_param mgs.MGS.live.testfs
...
imperative_recovery_state:
    state: full
    nonir_clients: 0
    nidtbl_version: 242
    notify_duration_total: 0.470000
    notify_duation_max: 0.041000
    notify_count: 38
```

条目	说明
state	**full:** IR 正在工作，所有客户端已连接上并能够接收到通知。 **partial:** 部分客户端没有启用 IR。 **disabled:** IR 被禁用，没有客户端能接收到通知 **startup:** MGS 刚刚启动，并非所有的客户端重新连接到 MGS。
nonir_clients	系统中不支持 IR 的客户端数量。
nidtbl_version	目标状态表的版本号。客户端版本必须与 MGS 匹配。

条目	说明
notify_duration_total	［秒/毫秒］MGS 通知所有客户端所花费的总时间
notify_duration_max	［秒/毫秒］MGS 通知单个 IR 客户端花费的最长时间
notify_count	通知客户端的数量。（可通过`notify_duration_total`除以`notify_count`来计算平均通知时间）

38.6.2.4. 查看 IR 状态－客户端 IR 中的" 客户端" 指的是 Lustre 客户端或 MDT。您可以在运行客户端或 MDT 的任何节点上获取 IR 状态，这些节点将始终运行 MGC。我们来看下面的例子：

```
[client]$ lctl get_param mgc.*.ir_state
mgc.MGC192.168.127.6@tcp.ir_state=
imperative_recovery: ON
client_state:
    - { client: testfs-client, nidtbl_version: 242 }
```

以及来自 MDT 的例子：

```
mgc.MGC192.168.127.6@tcp.ir_state=
imperative_recovery: ON
client_state:
    - { client: testfs-MDT0000, nidtbl_version: 242 }
```

条目	说明
imperative_recovery	`imperative_recovery`可为 ON 或 OFF。一般情况下应为 ON。如果为 OFF，则管理员在挂载时禁用了 IR。
client_state: client:	客户端名称
client_state: nidtbl_version	目标状态表的版本号。客户端版本必须与 MGS 匹配。

38.6.2.5. 目标实例编号 目标实例编号用于确定客户端是否连接到目标的最新实例。我们使用挂载计数的最低的 32 位作为目标实例编号。在 OST 上获取 testfs-OST0001 的目标实例编号：

```
$ lctl get_param obdfilter.testfs-OST0001*.instance
obdfilter.testfs-OST0001.instance=5
```

从客户端上查询相关 OSC：

```
$ lctl get_param osc.testfs-OST0001-osc-*.import |grep instance
    instance: 5
```

38.6.3. IR 配置建议

过去，我们通常在同一个目标上创建 MGS 和 MDT0000 来保存服务器节点。但为了使 IR 更高效地工作，我们强烈建议您在单独的节点上运行 MGS。这样做有三个主要优点：

1. 在 MDT0000 恢复时能通知客户端。
2. 改进负载平衡。MDS 上的负载可能非常大，从而导致 MGS 可能无法及时通知客户端。
3. 健壮性。与 MDS 代码相比，MGS 代码更简单也更小。这意味着软件错误导致 MGS 停机的可能性非常低。

38.7. Ping 抑制

在具有大量客户端和 OST 的集群上，`OBD_PING`消息可能会带来显着的性能开销。此处存在一个抑制 ping 的选项作为中间解决方案，大大减少了 ping 的开销。在启用此选项之前，管理员应认真考虑以下要求并权衡得失：

- 当抑制 ping 时，目标无法检测到客户端的死亡，因为客户端不发送仅为保持其连接存活的 ping。因此，需另行设置 Lustre 文件系统外部机制用于及时通知 Lustre 目标客户端死亡情况，从而使过时连接不会存在太久、死亡客户端的锁回调不需要总是等待超时。
- 如果没有 ping，客户端必须依靠 IR 来通知目标故障以及时加入恢复。这表明客户端应该保持其 MGS 连接的活跃性。因此，一方面建议使用高可用性的独立 MGS，另一方面，无论选项如何设置，都应始终发送 MGS ping。
- 如果客户端有未提交的请求，且没有在连接上发送任何新请求，则即使应该抑制 ping，它还是会继续 ping 该目标。这是因为客户端需要查询目标的最后提交事务编号，以释放本地未提交的请求（以及可能的其他相关资源）。但是，一旦释放了所有未提交的请求，或者需要发送新请求，这些 ping 就应该停止。

38.7.1. 内核模块参数"suppress_pings"

用于控制 ping 是否被抑制的新选项是作为 ptlrpc 内核模块参数"suppress_pings" 来实现的。在服务器上将其设置为"1"，则对该服务器上的所有目标都启用 ping 抑制。保留默认值"0" 则会继续进行先前的 ping 操作。在客户端和 MGS 上，该参数被忽略。我们建议您通过 modprobe.conf(5) 机制来永久地设置该参数，也可以通过 sysfs 进行在线更改。请注意，在线更改仅影响以后建立的连接，现有连接的 ping 行为保持不变。

38.7.2. 客户端死亡通知

在客户端死亡通知中应将死亡客户端的 UUID 写入目标的"evict_client" procfs 条目中:

```
1 /proc/fs/lustre/obdfilter/testfs-OST0000/evict_client
2 /proc/fs/lustre/obdfilter/testfs-OST0001/evict_client
3 /proc/fs/lustre/mdt/testfs-MDT0000/evict_client
```

客户端的 UUID 可通过它们的"uuid" procfs 条目获取:

```
1 /proc/fs/lustre/llite/testfs-ffff8800612bf800/uuid
```

第三十九章 Lustre 参数

39.1. 简介

Lustre 参数和统计文件为内核中的内部数据结构提供了接口，从而监视和调试 Lustre 文件系统和应用程序性能。这些数据结构包括组件（如内存、网络、文件系统和内核管理程序）的设置和指标，在整个分层文件布局中都可用。

一般来说，通过`lctl get_param`文件获取指标结果，通过`lctl set_param`更改设置。有些数据只存在于服务器上，有些数据只存在于客户端上，有些数据是从客户端导出到服务器的，因此在两个位置都有

注意

在本章的例子中，# 表明命令在运行在 root 用户下。Lustre 服务器按照`fsname*-*MDT|OSTnumber`的惯例命名。这里我们使用了 UNIX 标准通配符 (*)。

以下是一些示例:

- 从 Lustre 客户端获取数据:

```
1 # lctl list_param osc.*
2 osc.testfs-OST0000-osc-ffff881071d5cc00
3 osc.testfs-OST0001-osc-ffff881071d5cc00
```

```
4 osc.testfs-OST0002-osc-ffff881071d5cc00
5 osc.testfs-OST0003-osc-ffff881071d5cc00
6 osc.testfs-OST0004-osc-ffff881071d5cc00
7 osc.testfs-OST0005-osc-ffff881071d5cc00
8 osc.testfs-OST0006-osc-ffff881071d5cc00
9 osc.testfs-OST0007-osc-ffff881071d5cc00
10 osc.testfs-OST0008-osc-ffff881071d5cc00
```

可在客户端获取的 OST 连接相关消息显示如上。

- 查看不同级别的参数，请使用多个通配符：

```
1 # lctl list_param osc.*.*
2 osc.testfs-OST0000-osc-ffff881071d5cc00.active
3 osc.testfs-OST0000-osc-ffff881071d5cc00.blocksize
4 osc.testfs-OST0000-osc-ffff881071d5cc00.checksum_type
5 osc.testfs-OST0000-osc-ffff881071d5cc00.checksums
6 osc.testfs-OST0000-osc-ffff881071d5cc00.connect_flags
7 osc.testfs-OST0000-osc-ffff881071d5cc00.contention_seconds
8 osc.testfs-OST0000-osc-ffff881071d5cc00.cur_dirty_bytes
9 ...
10 osc.testfs-OST0000-osc-ffff881071d5cc00.rpc_stats
```

- 使用 `lctl get_param`查看指定文件：

```
# lctl get_param osc.lustre-OST0000*.rpc_stats
```

使用带有文件完整路径的`cat`命令也可以查看数据。`cat`命令的格式与`lctl get_param`类似，但也有一些差异。多年来，Linux 内核在不断变化，统计信息和参数文件的位置也随之发生了变化。这意味着 Lustre 参数文件可能位于`/proc`目录、`/sys`目录或（和）`/sys/kernel/debug`目录，具体取决于内核版本和正在使用的 Lustre 版本。`lctl`命令将脚本与这些更改隔离，除非作为高性能监视系统的一部分，否则优先使用直接文件访问方式。`cat`命令：

- 将路径中的'.' 替换为'/'。
- 在路径前面附加相应的如下内容：

```
/{proc,sys}/{fs,sys}/{lustre,lnet}
```

`lctl get_param` 命令可能如下所示：

```
# lctl get_param osc.*.uuid
osc.testfs-OST0000-osc-ffff881071d5cc00.uuid=594db456-0685-bd16-f59b-e72ee90e9819
osc.testfs-OST0001-osc-ffff881071d5cc00.uuid=594db456-0685-bd16-f59b-e72ee90e9819
...
```

相应地，cat 命令可能如下所示：

```
# cat /proc/fs/lustre/osc/*/uuid
594db456-0685-bd16-f59b-e72ee90e9819
594db456-0685-bd16-f59b-e72ee90e9819
...
```

或：

```
# cat /sys/fs/lustre/osc/*/uuid
594db456-0685-bd16-f59b-e72ee90e9819
594db456-0685-bd16-f59b-e72ee90e9819
...
```

llstat 工具可用于监控指定的一段时间内的一些 Lustre 文件系统 I/O 活动。

某些数据是从连接的客户端导入的，位于 Lustre 服务器上相应的服务目录中名为exports的目录中。例如：

```
oss:/root# lctl list_param obdfilter.testfs-OST0000.exports.*
# hash ldlm_stats stats uuid
```

39.1.1. 识别 Lustre 文件系统和服务器

MGS 上的几个参数文件列出了现有的 Lustre 文件系统和文件系统服务器。以下示例适用于名为testfs的 Lustre 文件系统（包含一个 MDT 和三个 OST）。

- 查看所有已知 Lustre 文件系统，输入：

  ```
  mgs# lctl get_param mgs.*.filesystems testfs
  ```

- 在运行有至少一个服务器的文件系统上查看所有服务器名：

  ```
  lctl get_param mgs.*.live.<filesystem name>
  ```

  如：

```
```
mgs# lctl get_param mgs.*.live.testfs
fsname: testfs
```
```

```
flags: 0x20     gen: 45
testfs-MDT0000
testfs-OST0000
testfs-OST0001
testfs-OST0002

Secure RPC Config Rules:

imperative_recovery_state:
state: startup
nonir_clients: 0
nidtbl_version: 6
notify_duration_total: 0.001000
notify_duation_max:  0.001000
notify_count: 4
```

- 查看文件系统中所有在线的服务器，即/proc/fs/lustre/devices的列表，输入：

```
# lctl device_list
0 UP mgs MGS MGS 11
1 UP mgc MGC192.168.10.34@tcp 1f45bb57-d9be-2ddb-c0b0-5431a49226705
2 UP mdt MDS MDS_uuid 3
3 UP lov testfs-mdtlov testfs-mdtlov_UUID 4
4 UP mds testfs-MDT0000 testfs-MDT0000_UUID 7
5 UP osc testfs-OST0000-osc testfs-mdtlov_UUID 5
6 UP osc testfs-OST0001-osc testfs-mdtlov_UUID 5
7 UP lov testfs-clilov-ce63ca00 08ac6584-6c4a-3536-2c6d-b36cf9cbdaa04
8 UP mdc testfs-MDT0000-mdc-ce63ca00 08ac6584-6c4a-3536-2c6d-
   b36cf9cbdaa05
9 UP osc testfs-OST0000-osc-ce63ca00 08ac6584-6c4a-3536-2c6d-b36cf9cbdaa05
10 UP osc testfs-OST0001-osc-ce63ca00 08ac6584-6c4a-3536-2c6d-b36cf9cbdaa05
```

每一行包括以下内容：

- 设备编号
- 设备状态（UP、INactive 或 STopping）

- 设备名
- 设备 UUID
- 引用计数（该设备有多少用户）

- 显示任一服务器名，查看设备标签：

```
mds# e2label /dev/sda testfs-MDT0000
```

39.2. 多块分配的调试 (mballoc)

mballoc功能包括：

- 单个文件的预分配，减少碎片。

- 组文件的预分配，将小文件打包成大的、连续的块。

- 流分配，降低搜索率。

以下是可用的mballoc可调参数:

参数	说明
mb_max_to_scan	在最终决定前mballoc搜索的最多的空闲块数，用于避免活锁情况。
mb_min_to_scan	在分配最佳块前mballoc搜索的最少的空闲块数，用于避免大容量空间块被小的请求碎片化。
mb_order2_req	对于大小为 2^N（N>=mb_order2_req）的请求，使用基数为 2 的伙伴分配服务进行快速搜索。
mb_small_req	mb_small_req －定义小请求的上限（以 MB 为单位） 请求根据大小进行不同的处理，当其小于 mb_small_req 时，请求会被打包在一起形成大型的聚合请求；当其大于mb_small_req 且小于mb_large_req时，请求基本是按线性分配的。当其大于 mb_large_req 时，请求会被立即分配（此时硬盘搜索时间问题不大）。 也就是说，通常会把小请求组合成大请求，然后再将这些请求靠近放置从而把访问数据所需的查找次数降到最低。

参数	说明
mb_large_req	mb_large_req －定义大请求的下限（以 MB 为单位）
prealloc_table	收到新请求时预分配空间的相应值列表。默认情况下，表格为：prealloc_table 4 8 16 32 64 128 256 512 1024 2048。收到新请求时，会预分配表格中指定的下一个更高的增量。例如，对于少于 4 个文件系统块的请求，预先分配 4 个空间块；对于 4 到 8 之间的请求，预先分配 8 个块。虽然可以在表格中使用自定义值，但修改表格通常不会提高文件系统的通用性能（注意，在 ext4 系统中，表值是固定的）。但是，对于某些专用工作负载，调整prealloc_table值可能会产生更明智的预分配决策。
mb_group_prealloc	为一组小请求预分配的空间大小（以 KB 为单位）

/sys/fs/ldiskfs/disk_device/mb_groups中的伙伴组缓存信息可用于评估磁盘碎片。例如：

```
cat /proc/fs/ldiskfs/loop0/mb_groups
#group: free free frags first pa [ 2^0 2^1 2^2 2^3 2^4 2^5 2^6 2^7 2^8 2^9
     2^10 2^11 2^12 2^13]
#0    : 2936 2936 1     42    0  [ 0   0   0   1   1   1   1   2   0   1
     2    0    0    0   ]
```

各列内容为：

- # 组编号
- 组内可用块数
- 磁盘空闲块数
- 空闲片段号
- 组内第一个空闲块
- 预分配组块（chunk）编号
- 一串大小不同的可用组块（chunk）

39.3. Lustre 文件系统 I/O 监控

有许多系统实用程序能够在 Lustre 文件系统中收集 I/O 活动相关数据。通常，所收集的数据描述了：

- Lustre 文件系统外部的数据传输速率和输入输出吞吐量，例如网络请求或执行的磁盘 I/O 操作

- Lustre 文件系统内部数据的吞吐量或传输速率的数据，例如锁或分配情况。

注意

强烈建议您完成 Lustre 文件系统的基准测试，以确定硬件、网络和系统工作负载的正常 I/O 活动。通过基准数据，您可以轻松地判断系统性能何时可能会降低。以下是两个特别有用的基准测试的统计数据：

- `brw_stats` －描述对 OST 的 I/O 请求有关数据的直方图。更多详细信息请参见本章第 3.5 节"OST 块 I/O 流监控"。
- `rpc_stats` -- 描述客户端 RPC 有关数据的直方图。更多详细信息请参见本章第 3.1 节" 客户端 RPC 流监控"。

39.3.1. 客户端 RPC 流监控

文件包含了显示自上次清除此文件以来进行的远程过程调用（RPC）信息的直方图数据。将任何值写入`rpc_stats` 文件将清除直方图数据。

示例：

```
# lctl get_param osc.testfs-OST0000-osc-ffff810058d2f800.rpc_stats
snapshot_time:            1372786692.399858 (secs.usecs)
read RPCs in flight:      0
write RPCs in flight:     1
dio read RPCs in flight:  0
dio write RPCs in flight: 0
pending write pages:      256
pending read pages:       0

                        read                    write
pages per rpc   rpcs   % cum % |       rpcs   % cum %
1:                 0   0   0   |          0   0   0
2:                 0   0   0   |          1   0   0
4:                 0   0   0   |          0   0   0
8:                 0   0   0   |          0   0   0
16:                0   0   0   |          0   0   0
32:                0   0   0   |          2   0   0
```

```
64:                0   0   0   |          2   0   0
128:               0   0   0   |          5   0   0
256:             850 100 100   |      18346  99 100

                     read                 write
rpcs in flight  rpcs   % cum % |       rpcs   % cum %
0:               691  81  81   |       1740   9   9
1:                48   5  86   |        939   5  14
2:                29   3  90   |       1059   5  20
3:                17   2  92   |       1052   5  26
4:                13   1  93   |        920   5  31
5:                12   1  95   |        425   2  33
6:                10   1  96   |        399   2  35
7:                30   3 100   |      11373  61  97
8:                 0   0 100   |        460   2 100

                     read                 write
offset          rpcs   % cum % |       rpcs   % cum %
0:               850 100 100   |      18347  99  99
1:                 0   0 100   |          0   0  99
2:                 0   0 100   |          0   0  99
4:                 0   0 100   |          0   0  99
8:                 0   0 100   |          0   0  99
16:                0   0 100   |          1   0  99
32:                0   0 100   |          1   0  99
64:                0   0 100   |          3   0  99
128:               0   0 100   |          4   0 100
```

题头信息包括：

- `snapshot_time` －文件读取的 UNIX epoch 瞬间。
- `read RPCs in flight` － OSC 发出的在此时还未完成的 read RPCs 数。该值应该永远小于或等于 `max_rpcs_in_flight`。
- `write RPCs in flight` － OSC 发出的在此时还未完成的 write RPCs 数。该值应该永远小于或等于 `max_rpcs_in_flight`。
- `dio read RPCs in flight` －已发起但尚未完成的 read RPCs 的直接 I/O（对应于阻塞 I/O)。

- `dio write RPCs in flight` － 已发起但尚未完成的 write RPCs 的直接 I/O（对应于阻塞 I/O）。
- `pending write pages` － OSC 上 I/O 队列中挂起的写页面数。
- `pending read pages` － OSC 上 I/O 队列中挂起的读页面数。

下面列出了上表中统计数据各条目的含义，各行显示了读取或写入次数（ios）、占总读取或写入的相对百分比（%）以及至该点为止的累积百分比（cum%）。

条目	说明
pages per RPC	按照 RPC 中的页数显示累积的 RPC 读取和写入。例如，单页 RPC 的数据将显示在`0:`行。
RPCs in flight	显示发送 RPC 时挂起的 RPC 数。第一个 RPC 发送后，`0:`行将递增。如果在另一个 RPC 挂起时发送第一个 RPC，则`1:`行将递增。依此类推。
offset	RPC 读取或写入对象的第一页的页面索引。

分析：

此表提供了一种将 RPC 流的并发性可视化的方法。在理想情况下，您会看到很多值聚集在`max_rpcs_in_flight`值周围，这表明网络一直处于忙碌状态。

有关客户端 I/O RPC 流优化的相关信息，请参见本章第 4.1 节" 客户端 I/O RPC 流的调试"。

39.3.2. 客户端活动监控

`stats`文件负责维护在 Lustre 文件系统的 VFS 接口上的客户端的典型操作期间累积的统计信息。文件中仅显示非零参数。

默认启用客户端统计信息功能。

注意

所有挂载文件系统的统计信息可通过输入以下命令得到：

```
lctl get_param llite.*.stats
```

示例：

```
client# lctl get_param llite.*.stats
snapshot_time           1308343279.169704 secs.usecs
dirty_pages_hits        14819716 samples [regs]
dirty_pages_misses      81473472 samples [regs]
read_bytes              36502963 samples [bytes] 1 26843582 55488794
```

```
write_bytes             22985001 samples [bytes] 0 125912 3379002
brw_read                2279 samples [pages] 1 1 2270
ioctl                   186749 samples [regs]
open                    3304805 samples [regs]
close                   3331323 samples [regs]
seek                    48222475 samples [regs]
fsync                   963 samples [regs]
truncate                9073 samples [regs]
setxattr                19059 samples [regs]
getxattr                61169 samples [regs]
```

可以通过将空字符串回显到stats文件中或使用以下命令来清除统计信息：

```
lctl set_param llite.*.stats=0
```

下表介绍了所显示统计信息的详细内容：

条目	说明
snapshot_time	读取stats文件的 UNIX epoch 瞬间。
dirty_page_hits	满足脏页面缓存的写入操作数。有关 Lustre 文件系统中脏缓存行为的更多信息，请参见本章第 4.1 节" 客户端 I/O RPC 流的调试"。
dirty_page_misses	不满足脏页面缓的写入操作数。
read_bytes	已发生的读操作数。将显示三个附加参数： min －自计数器重置以来单个请求读取的最小字节数； max －自计数器重置以来单个请求读取的最大字节数； sum －自计数器重置以来所有读请求的累计字节数。
write_bytes	已发生的写操作数。将显示三个附加参数： min －自计数器重置以来单个请求写入的最小字节数； max －自计数器重置以来在单个请求写入的最大字节数； sum －自计数器重置以来所有写请求的累计字节数。
brw_read	已读取的页数。将显示三个附加参数： min －自计数器重置以来，单个brw读请求中读取的最小字节数； max －自计数器重置以来，单个brw读请求中读取的最大字节数； sum －自计数器重置以来，所有brw读请求中累计字节数。

条目	说明
ioctl	组合文件和目录ioctl操作的数量。
open	已成功的打开操作数量。
close	已成功的关闭操作数量。
seek	调用 seek 的次数。
fsync	调用 fsync 的次数。
truncate	调用有锁和无锁 truncate的总数。
setxattr	已设置扩展属性的次数。
getxattr	已调取扩展属性值的次数。

分析：

提供客户端上正在进行的 I/O 活动的数量和类型有关信息。

39.3.3. 客户端读写位移统计信息监控

设置offset_stats参数后，在访问下一个顺位前，将对进程的一系列读取或写入调用信息进行维护。每当读取或写入不同的文件时，OFFSET字段将被重置为 0。

注意

默认情况下，为减少监控开销，非必要的话统计信息不会被收集在 offset_stats、extents_stats和extents_stats_per_process文件中。可通过在任何一个文件中写入除 0 和"disable" 以外的任何内容来激活这三个文件的统计信息收集功能。

示例：

```
# lctl get_param llite.testfs-f57dee0.offset_stats
snapshot_time: 1155748884.591028 (secs.usecs)
                RANGE   RANGE    SMALLEST   LARGEST
R/W   PID     START   END      EXTENT     EXTENT     OFFSET
R     8395    0       128      128        128        0
R     8395    0       224      224        224        -128
W     8395    0       250      50         100        0
W     8395    100     1110     10         500        -150
W     8394    0       5233     5233       5233       0
R     8395    500     600      100        100        -610
```

在上述示例中，snapshot_time 是读取文件时的 UNIX epoch 瞬间。显示的表格内容介绍如下：

offset_stats 文件可通过以下命令进行清除：

```
lctl set_param llite.*.offset_stats=0
```

条目	说明
R/W	指示非顺序调用是读取还是写入。
PID	调用读/写操作的进程 ID。
RANGE START/RANGE END	顺序读/写调用的范围。
SMALLEST EXTENT	相应范围内的最小的单次读/写（以字节为单位）。
LARGEST EXTENT	相应范围内的最大的单次读/写（以字节为单位）。
OFFSET	前一个范围结束点和当前范围开始点之间的差距。

分析：

此数据提供了数据连续或分段的信息。例如，上面示例中的第四个条目显示了此 RPC 的写入在 100 到 1110 范围内时顺序的，并且最小写入 10 个字节，最大写入 500 个字节。该范围开始于从前一个条目的 RANGE END 位移 -150 处。

39.3.4. 客户端读写范围统计信息监控

要进行深入的故障排除，可以通过查看客户端读写扩展统计信息来获取针对文件系统或特定进程的详细的 I/O 范围信息。

注意

默认情况下，为减少监控开销，非必要的话统计信息不会被收集在 offset_stats、extents_stats和 extents_stats_per_process 文件中。可通过在任何一个文件中写入除 0 和"disable" 以外的任何内容来激活这三个文件的统计信息收集功能。

39.3.4.1. 基于客户端的 I/O 范围大小调查 llite 目录中的 extents_stats 直方图显示了读写 I/O 范围大小的统计信息。此文件不对每个进程的统计信息进行维护。

示例：

```
# lctl get_param llite.testfs-*.extents_stats
snapshot_time:                    1213928728.348516 (secs.usecs)
                        read           |           write
extents          calls  %      cum%   |     calls  %     cum%
```

```
5
6 0K - 4K :        0      0      0      |     2      2     2
7 4K - 8K :        0      0      0      |     0      0     2
8 8K - 16K :       0      0      0      |     0      0     2
9 16K - 32K :      0      0      0      |     20     23    26
10 32K - 64K :     0      0      0      |     0      0     26
11 64K - 128K :    0      0      0      |     51     60    86
12 128K - 256K :   0      0      0      |     0      0     86
13 256K - 512K :   0      0      0      |     0      0     86
14 512K - 1024K :  0      0      0      |     0      0     86
15 1M - 2M :       0      0      0      |     11     13    100
```

在这个例子中，snapshot_time是读取文件时的 UNIX epoch 瞬间。该表显示了根据大小排列的累计范围，并分别为读取和写入提供了统计信息。表中的每一行分别显示读取和写入的 RPC 数 (calls)，，占总调用的相对百分比（%）以及到该点为止所占的累积百分比（cum%）。

此文件可通过以下命令进行清除：

```
1 # lctl set_param llite.testfs-*.extents_stats=1
```

39.3.4.2. 基于进程的客户端 I/O 统计信息 extents_stats_per_process文件用于维护基于每个进程的 I/O 范围大小统计信息。

示例：

```
1 # lctl get_param llite.testfs-*.extents_stats_per_process
2 snapshot_time:                    1213928762.204440 (secs.usecs)
3                           read           |             write
4 extents            calls   %      cum%   |      calls   %       cum%
5
6 PID: 11488
7    0K - 4K :       0       0       0      |      0       0       0
8    4K - 8K :       0       0       0      |      0       0       0
9    8K - 16K :      0       0       0      |      0       0       0
10   16K - 32K :     0       0       0      |      0       0       0
11   32K - 64K :     0       0       0      |      0       0       0
12   64K - 128K :    0       0       0      |      0       0       0
13   128K - 256K :   0       0       0      |      0       0       0
14   256K - 512K :   0       0       0      |      0       0       0
```

```
   512K - 1024K :  0       0       0       |       0       0       0
   1M - 2M :       0       0       0       |       10      100     100

PID: 11491
   0K - 4K :       0       0       0       |       0       0       0
   4K - 8K :       0       0       0       |       0       0       0
   8K - 16K :      0       0       0       |       0       0       0
   16K - 32K :     0       0       0       |       20      100     100

PID: 11424
   0K - 4K :       0       0       0       |       0       0       0
   4K - 8K :       0       0       0       |       0       0       0
   8K - 16K :      0       0       0       |       0       0       0
   16K - 32K :     0       0       0       |       0       0       0
   32K - 64K :     0       0       0       |       0       0       0
   64K - 128K :    0       0       0       |       16      100     100

PID: 11426
   0K - 4K :       0       0       0       |       1       100     100

PID: 11429
   0K - 4K :       0       0       0       |       1       100     100
```

该表显示了根据每个进程大小排列的累计范围，并分别为读取和写入提供了统计信息。表中的每一行分别显示读取和写入的 RPC 数 (calls)，占总调用的相对百分比（%）以及到该点为止所占的累积百分比（cum%）。### 39.3.5. OST 阻塞 I/O 流监控

在osd-ldiskfs或osd-zfs目录下的brw_stats参数文件包含直方图数据，该图显示了发送到磁盘的 I/O 请求的数量、大小、以及它们在磁盘上是否连续的统计信息。

示例：

在 OSS 或 MDS 上输入：

```
# oss# lctl get_param osd-*.*.brw_stats
snapshot_time:         1372775039.769045 (secs.usecs)
                            read      |      write
pages per bulk r/w     rpcs  % cum % |  rpcs   % cum %
1:                     108 100 100   |    39   0   0
2:                       0   0 100   |     6   0   0
```

```
4:                      0   0 100   |     1   0   0
8:                      0   0 100   |     0   0   0
16:                     0   0 100   |     4   0   0
32:                     0   0 100   |    17   0   0
64:                     0   0 100   |    12   0   0
128:                    0   0 100   |    24   0   0
256:                    0   0 100   | 23142  99 100

                           read     |      write
discontiguous pages    rpcs  % cum % |  rpcs   % cum %
0:                      108 100 100   | 23245 100 100

                           read     |      write
discontiguous blocks   rpcs  % cum % |  rpcs   % cum %
0:                      108 100 100   | 23243  99  99
1:                      0   0 100   |     2   0 100

                           read     |      write
disk fragmented I/Os   ios   % cum % |   ios   % cum %
0:                      94  87  87   |     0   0   0
1:                      14  12 100   | 23243  99  99
2:                      0   0 100   |     2   0 100

                           read     |      write
disk I/Os in flight    ios   % cum % |   ios   % cum %
1:                      14 100 100   | 20896  89  89
2:                      0   0 100   |  1071   4  94
3:                      0   0 100   |   573   2  96
4:                      0   0 100   |   300   1  98
5:                      0   0 100   |   166   0  98
6:                      0   0 100   |   108   0  99
7:                      0   0 100   |    81   0  99
8:                      0   0 100   |    47   0  99
9:                      0   0 100   |     5   0 100

                           read     |      write
```

```
I/O time (1/1000s)     ios   % cum % |   ios   % cum %
1:                      94  87  87   |     0   0   0
2:                       0   0  87   |     7   0   0
4:                      14  12 100   |    27   0   0
8:                       0   0 100   |    14   0   0
16:                      0   0 100   |    31   0   0
32:                      0   0 100   |    39   0   0
64:                      0   0 100   | 18979  81  82
128:                     0   0 100   |   943   4  86
256:                     0   0 100   |  1233   5  91
512:                     0   0 100   |  1825   7  99
1K:                      0   0 100   |    99   0  99
2K:                      0   0 100   |     0   0  99
4K:                      0   0 100   |     0   0  99
8K:                      0   0 100   |    49   0 100

                           read      |      write
disk I/O size          ios   % cum % |   ios   % cum %
4K:                     14 100 100   |    41   0   0
8K:                      0   0 100   |     6   0   0
16K:                     0   0 100   |     1   0   0
32K:                     0   0 100   |     0   0   0
64K:                     0   0 100   |     4   0   0
128K:                    0   0 100   |    17   0   0
256K:                    0   0 100   |    12   0   0
512K:                    0   0 100   |    24   0   0
1M:                      0   0 100   | 23142  99 100
```

下面列出了上表中统计数据各条目的含义，各行显示了读取或写入次数（ios）、占总读取或写入的相对百分比（%）以及至该点为止的累积百分比（cum%）。

条目	说明
`pages per bulk r/w`	每个 RPC 请求的页数，应与客户端`rpc_stats`匹配（请参见本章第 3.1 节" 客户端 RPC 流监控"）。
`discontiguous pages`	单个 RPC 中每个页面的文件逻辑偏移量中的不连续数。
`discontiguous blocks`	单个 RPC 中文件系统物理块分配的不连续数。

条目	说明
`disk fragmented I/Os`	未完全按顺序写入的 I/O 数。
`disk I/Os in flight`	当前挂起的磁盘 I/O 数。
`I/O time (1/1000s)`	完成每个 I/O 操作所需时间。
`disk I/O size`	每个 I/O 操作的大小。

分析：

此数据提供了文件系统中的范围大小和分布的相关信息。

39.4. Lustre 文件系统 I/O 调试

每个 OSC 都有自己的可调参数树。例如：

```
$ lctl lctl list_param osc.*.*
osc.myth-OST0000-osc-ffff8804296c2800.active
osc.myth-OST0000-osc-ffff8804296c2800.blocksize
osc.myth-OST0000-osc-ffff8804296c2800.checksum_dump
osc.myth-OST0000-osc-ffff8804296c2800.checksum_type
osc.myth-OST0000-osc-ffff8804296c2800.checksums
osc.myth-OST0000-osc-ffff8804296c2800.connect_flags
:
:
osc.myth-OST0000-osc-ffff8804296c2800.state
osc.myth-OST0000-osc-ffff8804296c2800.stats
osc.myth-OST0000-osc-ffff8804296c2800.timeouts
osc.myth-OST0000-osc-ffff8804296c2800.unstable_stats
osc.myth-OST0000-osc-ffff8804296c2800.uuid
osc.myth-OST0001-osc-ffff8804296c2800.active
osc.myth-OST0001-osc-ffff8804296c2800.blocksize
osc.myth-OST0001-osc-ffff8804296c2800.checksum_dump
osc.myth-OST0001-osc-ffff8804296c2800.checksum_type
:
:
```

下面介绍了一些 Lustre 文件系统的可调参数。

39.4.1. 客户端 I/O RPC 流的调试

在理想情况下，每个 I/O RPC 会将刚好数据量大小刚好的数据打包，且每时每刻都会有数量一致的已发起 RPC 在处理中。为优化客户端 I/O RPC 流，Lustre 提供了几个可调参数以根据网络条件和集群大小来调整行为。相关内容请参见本章第 3.1 节" 客户端 RPC 流监控"。

RPC 流可调参数包括：

- `osc.osc_instance.cksums` －用于控制客户端是否计算传输到 OST 的批量数据的数据完整性校验和。默认情况下，启用数据完整性校验和。使用的算法可以通过 `checksum_type` 参数来设置。

- `osc.osc_instance.cksum_type` －用于控制客户端使用的数据完整性校验和算法。可用的算法是由算法集决定的。默认使用的校验和算法是通过首先选择 OST 上可用的最快的一些算法，然后在客户端上选择这些算法中最快的算法，这依赖于 CPU 硬件和内核中的可用优化。默认的算法可以通过在`checksum_type` 参数中写入算法名称来设置。通过读取 `checksum_type`参数可以在客户端上看到可用的校验和类型。目前支持的校验和类型有：`adler`, `crc32`, `crc32c`。在 **Lustre 2.12** 版本中，增加了额外的校验和类型，允许与 T10-PI 功能的硬件进行端到端的校验和集成。客户端将根据存储所使用的校验和类型，为 RPC 校验和计算出适当的校验和类型，由服务器验证并传递给存储。T10-PI 校验和类型有: `t10ip512`, `t10ip4K`, `t10crc512`, `t10crc4K`。

- `osc.*osc_instance*.max_dirty_mb` －用于控制 OSC 中允许写入客户端页缓存中的脏数据量。达到限额时，先前缓存的写入同步到服务器前其他写入将停止。此限额可通过`lctl set_param`命令修改，其值必须在 0 到 2048MiB 之间或等于 1/4 的 RAM。当客户端不能在每个 OSC 中聚合足够的数据以形成一个完整的 RPC 参数（由 max_pages_per_rpc 设置）时，如果您没有使用较大的写入，性能可能会明显受损。

为使性能最优化，我们推荐您将 `max_dirty_mb` 设置为 `max_pages_per_rpc* max_rpcs_in_flight`的四倍。

- `osc.*osc_instance*.cur_dirty_bytes` －只读值，返回此 OSC 上当前写入和缓存的字节数。

- `osc.*osc_instance*.max_pages_per_rpc` －对 OST 的单个 RPC I/O 的最大页数。最小值为 1 页，最大值为 16MiB（对于`PAGE_SIZE`为 4KB 的系统为 4096 页），RPC 的默认最大值为 4MiB。上限也可能受到 OSS 上的`ofd.*.brw_size`设置的限制，适用于连接到该 OSS 的所有客户端。也可指定

单位后缀（如max_pages_per_rpc=4M）以便独立于客户端的PAGE_SIZE而单独指定 RPC 的大小。

- osc.*osc_instance*.max_rpcs_in_flight － OSC 到其 OST 的 RPC 的最大并发处理数。如果 OSC 尝试启动 RPC 但发现已经有与此设置数相同数量的未完成 RPC，它将等待发起 RPC，直到某些 RPC 完成。最小值为 1，最大值为 256。默认值为 8 RPC.

为提升小文件 I/O 性能，请提高 max_rpcs_in_flight 值。

- llite.*fsname-instance*/max_cached_mb － 客户端缓存的最大 read +write 数据量（默认为 RAM 的 1/2）。

注意

osc_instance和fsname_instance的值对于每个挂载点来说都是唯一的，以便于将 osc、mdc、lov、lmv 和 llite 参数与同一挂载点关联。但是，脚本通常会使用通配符 `*' 或文件系统专用的通配符 fsname-* 来统一指定所有客户端上的参数设置。比如说

```
lctl get_param osc.testfs-OST0000-osc-ffff88107412f400.rpc_stats
osc.testfs-OST0000-osc-ffff88107412f400.rpc_stats=
snapshot_time:         1375743284.337839 (secs.usecs)
read RPCs in flight:  0
write RPCs in flight: 0
```

39.4.2. 文件 Readahead 和目录 Statahead 的调试

文件预读和目录 statahead 能够在进程请求数据之前将数据读入内存。文件预读为read()相关的调用将文件内容数据预读取到内存中，而目录 statahead 为readdir()和stat()相关的调用将文件元数据取到内存中。当 readahead 和 statahead 工作良好时，访问数据的进程会发现，其所需要的信息可以立刻在客户端的内存中得到，而无需忍受网络 I/O 的延迟。

39.4.2.1. 文件 Readahead 当应用程序的两次或多次连续读取未能命中 Linux 缓冲区缓存中的数据时，就会触发文件 readahead。首次 readahead 的大小由 RPC 大小和文件条带大小决定，通常至少为 1MiB。后续的预读尺寸会保持线性增长，直到预读缓存到达了客户端上每个文件或每个系统的上限。

Readahead 相关可调参数有：

- llite.fsname-instance.max_read_ahead_mb －用于控制所有文件预读的最大数据量。在文件描述符上第二次顺序读取之后，预读文件至 RPC 大小的块

（4MiB 或更大的read()大小）中。随机读取的大小只能为read()调用大小（无预读）。读取文件至非连续区域会重置预读算法，并且在再次顺序读取之前不会再次触发预读。

这是对所有文件的全局限制，不能大于客户端 RAM 的 1/2。要禁用 readahead，请设置max_read_ahead_mb=0。

- llite.fsname_instance.max_read_ahead_per_file_mb－当获取到一个文件上的读取顺序时，用于控制客户端应该预读取的最大数据兆字节数(MiB)。这是每文件的预读取限制，不能大于max_read_ahead_mb。

- llite.fsname-instance.max_read_ahead_whole_mb －用于控制最多多大的文件会在被读到时被整个预取到客户端，不管read()调用读取的大小有多大，单位为 MiB。这样的整体预取，可以避免浪费多个小 RPC 来读取相对较小的文件，因为在整个文件读取完之前，系统没有办法有效地检测到连续读取模式。

默认值为 2 MiB 或一个 RPC 的大小 (由max_pages_per_rpc 决定)，取两者中的较大者。。

39.4.2.2. 目录 Statahead 和 AGL 的调试 许多系统命令（如ls -l、du和find）按顺序遍历目录。为使这些命令高效运行，可以启用目录 statahead 来提高目录遍历性能。

statahead 相关可调参数有：

- statahead_max －用于控制由 statahead 线程预取的最大文件属性数量。statahead 默认启用，statahead_max默认为 32 个文件。

禁用 statahead，请在客户端上设置 =statahead_max0：

```
lctl set_param llite.*.statahead_max=0
```

在客户端上更改最大 statahead 窗口大小：

```
lctl set_param llite.*.statahead_max=n
```

最大statahead_max 为 8192 个文件。

目录 statahead 线程同时也会从 OST 预取文件大小和消耗的空间，以便应用程序需要时，可以直接从客户端上获取所有的文件属性。这是由异步 glimpse 锁（AGL）设置控制，可通过以下命令禁用 AGL 行为：

```
lctl set_param llite.*.statahead_agl=0
```

- statahead_stats －只读接口，可提供当前 statahead 和 AGL 统计信息，如自上次挂载以来已触发 statahead/AGL 的次数、由于预测错误或其他原因导致的 statahead/AGL 故障次数等。

注意

AGL 处理的 inode 是由 statahead 线程构建的，AGL 行为因此受 statahead 的影响。如果禁用了 statahead，则 AGL 也会被禁用。

39.4.3. 服务器读缓存的调试

服务器读缓存功能是指在 OSS 或 MDS 上提供文件数据（Data-on-MDT）的只读缓存，通过 Linux 页面缓存来存储数据。它会使用分配的所有物理内存。

服务器读缓存可在以下情况提高 Lustre 文件系统性能：

- 许多客户端访问相同的数据集（如在 HPC 应用程序中或无盘客户端从 Lustre 文件系统引导时）。
- 一个客户端正在写入数据，而另一个客户端正在读取数据（即客户端通过文件系统交换数据）。
- 客户端自身的缓存非常有限。

服务器读缓存提供了以下好处：

- 允许服务器更频繁地缓存读取数据。
- 改进重复读取以匹配网络速度而不是存储速度。
- 提供构建服务器写缓存（小数据写入聚合）的块。

39.4.3.1. 服务器读缓存的使用 服务器读缓存是在 OSS 和 MDS 上实现的，不需要客户端的任何特殊支持。由于服务器读缓存使用了 Linux 页面缓存中的可用内存，因此应根据 I/O 模式来确定适当的缓存内存量。如果主要是读取数据，则服务器需要比主要为写入的 I/O 模式需要更多读缓存。

可使用以下可调参数管理服务器读缓存。许多参数对`osd-ldiskfs`和`osd-zfs`均可用，但在某些情况下，`osd-zfs`的实现方式使参数无法使用。

- `read_cache_enable` －用于总体控制在读取请求期间从磁盘读取的数据是否保留在内存，以便于应付随后对相同数据的读取请求而无需从磁盘重新读取。默认情况下对 HDD OSDs 为启用状态 (`read_cache_enable=1`)，对闪存 OSD 为自动禁用状态(`nonrotational=1`).。

当 OSS 收到来自客户端的读取请求时，会从磁盘上读取数据到其内存中，并将数据作为请求的回复发送。如果读取缓存启用，那么在满足客户端的请求后，这些数据会

留在内存中。当收到相同数据的后续读取请求时，OSS 会跳过从磁盘读取数据，而直接从缓存的数据中满足请求。这些读缓存由 Linux 内核在该 OSS 的所有 OST 上进行全局管理，这样当空闲内存不足时，最近使用最少的缓存页会从内存中删除。

如果禁用读缓存，在完成了对客户端的读取请求的服务后，OSS 会丢弃数据，对后续的读取请求，OSS 会再次从磁盘读取数据。

在服务器的所有目标上禁用读缓存，请运行:

```
oss1# lctl set_param osd-*.*.read_cache_enable=0
```

重新在目标上启用读缓存，请运行:

```
oss1# lctl set_param osd-*.{target_name}.read_cache_enable=1
```

查看服务器的所有目标上都启用了读缓存，请运行:

```
oss1# lctl get_param osd-*.*.read_cache_enable
```

- `writethrough_cache_enable` －用于总体控制发送到服务端的写入请求数据是保留在读缓存用于后续读取，还是在写入完成后丢弃。默认情况下对 HDD OSD 为启用状态（`writethrough_cache_enable=1`），对闪存 OSD 为自动禁用(`nonrotational=1`)。对`osd-zfs`来说，写缓存不能被禁用，因此禁用参数对该后端不可用。

当服务端从客户端接获取请求时，则将从客户端接收数据放入内存中并写入存储。如果目标启用了写缓存，且 RPC 和对象大小也满足下述的其他标准后，则此数据在写入请求完成后将留存在内存中。如果后续收到对相同数据的读取请求或修改部分页面的写入请求，且数据仍在内存中时，服务端会跳过从存储读取此数据的步骤。

如果禁用了写缓存（`writethrough_cache_enabled=0`），或者写区域或对象足够大以至于缓存失去作用，则服务端在完成客户端的写入请求后丢弃数据。处理后续读取请求或部分页面写入请求时，服务端必须从磁盘重新读取数据。

当客户端正在执行小数据写入或会导致部分页面更新的未对齐写入，或者其他节点需要立即读取另一个节点刚写入的文件时，建议启用写缓存。例如，在生产者 -消费者 I/O 模型中，或者在未进行 4096 字节边界对齐的共享文件写入等情况下，启用写缓存可能会非常有用。

相反，当大部分 I/O 为文件写入且在短时间内不会被重新读取，或者文件仅由同一节点写入和重新读取时，无论 I/O 是否对齐，都建议禁用写缓存。

要在服务端的所有目标上禁用写缓存，请运行:

```
oss1# lctl set_param osd-*.*.writethrough_cache_enable=0
```

重新在 OST 上启用写缓存，请运行:

```
oss1# lctl set_param osd-*.{OST_name}.writethrough_cache_enable=1
```

查看是否启用了写缓存，请运行:

```
oss1# lctl get_param osd-*.*.writethrough_cache_enable
```

- readcache_max_filesize－用于控制read_cache和writethrough_cache尝试保留在内存中的对象的最大大小。大于readcache_max_filesize的对象，无论进行读取或写入，无论是否设置了writethrough_cache_enable或read_cache_enable，都不会保存在缓存中。

设置该参数对于下面这种工作负载非常有用：相对较小的文件（比如工作启动文件、可执行文件、日志文件等）被许多客户端重复访问，而大文件通常只被读或写一次。不把大文件放入缓存，就意味着更多较小的对象有更大概率能在缓存中保留更长的时间。

设置readcache_max_filesize时，输入值可以以字节为单位指定，也可以使用后缀来指示其他二进制单位（如 K（千字节）、M（兆字节）、G（千兆字节）、T（太字节）、P（千兆字节））。

在服务端所有的目标上将最大缓存对象大小限制为 64 MB，请运行：

```
oss1# lctl set_param osd-*.*.readcache_max_filesize=64M
```

在所有目标上禁用最大缓存对象大小的限制，请运行：

```
oss1# lctl set_param osd-*.*.readcache_max_filesize=-1
```

查看是否服务端所有的目标上都启用了当前最大缓存目标大小的限制，请运行：

```
oss1# lctl get_param osd-*.*.readcache_max_filesize
```

- readcache_max_io_mb - 控制能够缓存在内存中的单个读操作的最大大小。大于readcache_max_io_mb的读取将直接从存储中读取，完全绕过页面缓存，从而避免了在高 IO 率下 CPU 的大量开销。osd-zfs中无法禁用读缓存，因此这个参数对该后端不可用。

当设置readcache_max_io_mb时，输入值可以以 mebibytes 为单位来指定，也可以用后缀来表示其他二进制单位，如 K（kibibytes），M（mebibytes），G（gibibytes），T（tebibytes），或 P（pebibytes）。

- writethrough_max_io_mb - 控制能够缓存在内存中的单个写操作的最大大小。大于writeethrough_max_io_mb的写入将直接写入存储，完全绕过页面缓存，从而避免了在高 IO 率下 CPU 的大量开销。osd-zfs中无法禁用写缓存，因此这个参数对该后端不可用。

当设置writeethrough_max_io_mb时，输入值可以以 mebibytes 为单位来指定，也可以用后缀来表示其他二进制单位，如 K（kibibytes）、M（mebibytes）、G（gibibytes）、T（tebibytes）或 P（pebibytes）。

39.4.4. 启用 OSS 异步日志提交

OSS 异步日志提交功能将数据异步地写入磁盘，而不强制进行日志刷新。这将减少搜索次数，并显著提高了某些硬件的性能。

注意

异步日志提交无法用于 Direct I/O 的写入（设置了O_DIRECT标志）。对这种 IO，将强制执行日志刷新。

启用异步日志提交功能后，客户端节点会将数据保留在页面缓存中（增加页面引用）。Lustre 客户端将监视从 OSS 发送到客户端的消息中的最后提交的交易号（Transaction Number，transno）。当客户端看到 OSS 报告的最后一个提交的transno至少等于批量写入的transno时，它会在相应的页面上释放引用。为了避免批量写入后，持有页面引用对时间过长，客户端在收到批量写入的回复后将发起 7 秒的 ping 请求（OSS 文件系统提交默认时间间隔为 5 秒），以便 OSS 报告最后提交的transno。

如果 OSS 在日志提交之前崩溃，则中间数据将丢失。但是，结合异步日志提交的 OSS 恢复功能能够使客户端重放其写入请求，并通过恢复文件系统的状态来补偿丢失的磁盘更新。

默认情况下，sync_journal为启用状态（sync_journal=1），以便同步提交日记条目。启用异步日志提交，请输入以下内容将sync_journal参数设置为 0：

```
1 $ lctl set_param obdfilter.*.sync_journal=0
2 obdfilter.lol-OST0001.sync_journal=0
```

sync-on-lock-cancel 解决下面场景下的数据一致性问题：在多个客户端向一个对象的交叉区域写入数据后，如果这个 OSS 崩溃，而且不巧其中一个客户端也崩溃了，这种情况就有可能会违反 POSIX 对连续写入的语义要求，而且数据可能遭受损坏。在启用了 sync-on-lock-cancel 功能后，如果被取消的锁上附加了任何易失性的写入，OSS 会在撤销锁时同步将文件系统日志写到磁盘。禁用锁取消同步日志功能可以提高并发写的性能，但不推荐禁用这一功能。

sync_on_lock_cancel参数可设置为以下值：：

- always －在锁取消时强制执行日志更新（async_journal启用时的默认值）。
- blocking－只在因阻塞回调引起的锁取消时强制执行日志更新。
- never －不强制执行任何日志更新（async_journal禁用时的默认值）。

例如，将 sync_on_lock_cancel 设置为不强制执行日志更新，使用以下类似命令：

```
1 $ lctl get_param obdfilter.*.sync_on_lock_cancel
2 obdfilter.lol-OST0001.sync_on_lock_cancel=never
```

39.4.5. 客户端元数据 RPC 流的调试

客户端元数据 RPC 流表示客户端并行发起的到 MDT 目标的元数据 RPC。元数据 RPC 可以分为两类：不更改文件系统的请求（如 getattr 操作）和更改文件系统的请求（如 create、unlink、setattr 操作）。为优化客户端元数据 RPC 流，Lustre 提供了几个可调参数来根据网络条件和集群大小调整行为。

请注意，增加并行发起的元数据 RPC 的数量可能会改善元数据密集型并行应用程序的性能，但会在客户端和 MDS 上消耗更多的内存。

（在 **Lustre 2.8** 中引入）

39.4.5.1. 配置客户端元数据 RPC 流 MDC 的 `max_rpcs_in_flight` 参数定义了客户端并行发送到 MDT 目标的元数据 RPC 的最大数量，包括更改和不更改文件系统的 RPC。这包含了所有文件系统元数据操作，如文件或目录统计、创建、取消链接等。其默认值为 8，最小值为 1，最大值为 256。

在 Lustre 客户端上运行以下命令设置 `max_rpcs_in_flight` 参数：

```
1 client$ lctl set_param mdc.*.max_rpcs_in_flight=16
```

MDC 的 `max_mod_rpcs_in_flight` 参数定义了客户端并行发送到 MDT 目标的更改文件系统的 RPC 的最大数量。例如，Lustre 客户端在执行文件或目录创建、取消链接、访问权限修改、所有权修改时会发送更改式 RPC。其默认值为 7，最小值为 1，最大值为 256。

在 Lustre 客户端上运行以下命令设置`max_mod_rpcs_in_flight` 参数：

```
1 client$ lctl set_param mdc.*.max_mod_rpcs_in_flight=12
```

`max_mod_rpcs_in_flight`值必须比`max_rpcs_in_flight` 值小，同时也必须小于或等于 MDT 的 `max_mod_rpcs_per_client` 值。如果未满足其中一个条件，设置将失败，并在 Lustre 日志中写入明确的错误消息。

MDT 的 `max_mod_rpcs_per_client`参数是内核模块`mdt`的可调参数，它定义了每个客户端所允许的处理中的最大更改式 RPC 数量。该参数可以在运行时进行更新，但此更改仅对新客户端连接有效。其默认值为 8。

在 MDS 上运行以下命令设置 `max_mod_rpcs_per_client` 参数：

```
1 mds$ echo 12 > /sys/module/mdt/parameters/max_mod_rpcs_per_client
```

39.4.5.2. 客户端元数据 RPC 流监控 `rpc_stats` 文件包含了显示更改式 RPC 相关信息的直方图，可用于确定应用程序执行更改文件系统的元数据操作时所实现的并行级别。

示例：

```
client$ lctl get_param mdc.*.rpc_stats
snapshot_time:         1441876896.567070 (secs.usecs)
modify_RPCs_in_flight:  0

                        modify
rpcs in flight        rpcs   % cum %
0:                       0   0   0
1:                      56   0   0
2:                      40   0   0
3:                      70   0   0
4                       41   0   0
5:                      51   0   1
6:                      88   0   1
7:                     366   1   2
8:                    1321   5   8
9:                    3624  15  23
10:                   6482  27  50
11:                   7321  30  81
12:                   4540  18 100
```

文件内容包括：

- `snapshot_time` －读取文件时的 UNIX epoch 瞬间。
- `modify_RPCs_in_flight` － MDC 发起但当前还未完成的更改式 RPC 数。该值必须永远小于或等于`max_mod_rpcs_in_flight`。
- `rpcs in flight` －发送 RPC 时当前挂起的更改式 RPC 数量，包括相对百分比 (`%`) 和累积百分比 (`cum %`) 。

如果大部分更改式元数据 RPC 在发送时已经有大量的接近`max_mod_rpcs_in_flight`值的挂起元数据 RPC，则意味着可以增加`max_mod_rpcs_in_flight`值来提高元数据更改性能。

39.5. Lustre 文件系统超时配置

在 Lustre 文件系统中，RPC 超时使用自适应超时机制（默认为启用）。服务端跟踪 RPC 完成时间并向客户端报告，以便估计未来 RPC 的完成时间。客户端使用这些估计值来设置 RPC 超时值。当服务器请求处理因某种原因而减慢时，服务器 RPC 完成时间延长，客户端则随之修改 RPC 超时值以允许更多的时间来完成 RPC。

如果服务器上排队的 RPC 接近客户端指定的 RPC 超时，为避免 RPC 超时和断开和重新连接的循环，服务器会向客户端发送" 早期回复"，告知客户端以允许更多的处理时间。相反，随着服务器处理速度的加快，RPC 超时值会降低，从而能够更快地检测到服务器无响应、更快地连接到服务器的故障转移伙伴。

39.5.1. 配置自适应超时

下表中的自适应超时参数可以使用 MGS 上的`lctl conf_param`命令在系统范围内进行永久设置。例如，为与文件系统`testfs`关联的所有服务器和客户端设置`at_max`值：

```
lctl conf_param testfs.sys.at_max=1500
```

注意

访问多个 Lustre 文件系统的客户端必须对所有文件系统使用相同的参数值。

参数	说明
`at_min`	自适应超时的最小值（以秒为单位），即服务器会报告的最小处理时间。默认值为 0。理想情况下，应将其设置为默认值。客户端不直接使用此值但将基于此值来设置超时时间。如果由于未知原因（通常为临时网络中断）导致自适应超时值太小而客户端处理 RPC 超时，可增大`at_min`值。
`at_max`	自适应超时的最大值（以秒为单位），是服务估计时间的上限。如果达到 `at_max`，RPC 请求超时。将`at_max` 设置为 0 则表明禁用自适应超时，转而采用固定超时时间设置方法。注意，如果慢速硬件导致服务估计值增加直至超出默认值`at_max`，可将`at_max`增加到您愿意等待 RPC 完成的最长时间。
`at_history`	自适应超时记忆的发生最慢事件的时间段（以秒为单位）。默认值为 600。
`at_early_margin`	超过该时间，Lustre 服务器将发送早期回复（以秒为单位）。默认值为 5。
`at_extra`	服务器发送每次早期回复时请求的时间增量（以秒为单位）。服务器不知道 RPC 还需花费多少时间，因此会要求一个固定的值，默认为 30。该默认值在发送过多早期回复和高估实际完成时间之间寻求

参数	说明
	了一个平衡。当服务器发现排队请求即将超时并需要发送早期回复时，服务器会加大at_extra值。如果超时，Lustre 服务器将丢弃请求，客户端进入恢复状态并重新连接到正常状态。如果同一 RPC 发生了多个要求增加 30 秒的早期回复，请将at_extra值更改为一个较大的数字以减少早期回复的发送，从而减少网络负载。
ldlm_enqueue_min	最小锁入队时间（以秒为单位），默认值为 100。锁入队所需的时间ldlm_enqueue通过入队估计所需时间的最大值（受at_min和at_max参数影响）乘以加权因子和ldlm_enqueue_min计算所得。测量所得的入队时间增加时，锁入队的时间增加（类似于自适应超时）。

39.5.1.1. 解析自适应超时信息 自适应超时信息可在每个服务器上使用命令lctl get_param {ost,mds}.*.*.timeouts和在客户端上使用命令lctl get_param {osc,mdc}.*.timeouts获取。从 timeouts 中读取信息，请输入:

```
# lctl get_param -n ost.*.ost_io.timeouts
service : cur 33  worst 34 (at 1193427052, 0d0h26m40s ago) 1 1 33 2
```

在此示例中，此节点上的ost_io服务报告了 RPC 服务时间估计为 33 秒。最长的 RPC 服务时间发生在 26 分钟前，为 34 秒。

该输出还提供了服务时间的历史记录，显示了四个自适应超时历史记录，分别报告了其最大的 RPC 时间。在 0-150s bin 和 150-300s bin 中，最大的 RPC 时间为 1。300-450s bin 中，最大 RPC 时间为 33 秒。450-600s bin 中，最大 RPC 时间为 2 秒。估计的服务时间则取这四条记录中的最大值（在本例中为 33 秒）。

客户端 OBD 也跟踪服务时间（由服务器报告），如下例所示:

```
# lctl get_param osc.*.timeouts
last reply : 1193428639, 0d0h00m00s ago
network    : cur  1 worst  2 (at 1193427053, 0d0h26m26s ago)  1  1  1  1
portal 6   : cur 33 worst 34 (at 1193427052, 0d0h26m27s ago) 33 33 33  2
portal 28  : cur  1 worst  1 (at 1193426141, 0d0h41m39s ago)  1  1  1  1
portal 7   : cur  1 worst  1 (at 1193426141, 0d0h41m39s ago)  1  0  1  1
portal 17  : cur  1 worst  1 (at 1193426177, 0d0h41m02s ago)  1  0  0  1
```

在此示例中，portal 6（ost_io服务入口）显示了该入口报告的服务时间估计历史记录。

服务器统计文件还显示了估计值的范围，包括 min、max、sum 和 sumsq。例如：

```
# lctl get_param mdt.*.mdt.stats
...
req_timeout               6 samples [sec] 1 10 15 105
...
```

39.5.2. 设置静态超时

在未启用自适应超时时使用，Lustre 软件提供两组静态（固定）超时：LND 超时和 Lustre 超时。

- **LND timeouts** - LND 超时可确保网络中的点对点通信在出现故障（如程序包丢失或连接断开）时在有限时间内完成。每个 LND 有单独的 LND 超时参数设置。

设置S_LND标志记录 LND 超时。它们不通过控制台打印消息，请查看 Lustre 日志中的D_NETERROR消息，或使用以下命令将D_NETERROR消息打印到控制台：

```
lctl set_param printk=+neterror
```

拥塞的路由器可能造成 LND 假性超时。为避免这种情况，请增加 LNet 路由器缓冲区的数量来减少背压，或增加网络上所有节点的 LND 超时。同时，也可考虑增加系统中 LNet 路由器节点总数，从而使路由器总带宽与服务器总带宽相匹配。

- **Lustre timeouts** －在未启用自适应超时时，Lustre 超时可确保 RPC 出现故障时在有限时间内完成。自适应超时默认为启用状态，要在运行时禁用自适应超时，请在 MGS 上将at_max设置为 0：

```
# lctl conf_param fsname.sys.at_max=0
```

注意

在运行时更改自适应超时的状态可能会导致客户端暂时的超时、恢复和重连。

Lustre 超时的消息将始终打印在控制台上。

如果 Lustre 超时未伴随 LND 超时，请增加服务器和客户端上的 Lustre 超时时间。使用如下命令进行设置：

```
# lctl set_param timeout=30
```

Lustre 超时参数：

参数	说明
timeout	客户端等待服务器完成 RPC 的时间（默认为 100 秒）。服务器等

参数	说明
	待正常客户端完成 RPC 的时间为此时间的一半，等待单个批量请求（最多读取或写入 4MB）完成的时间为此时间的四分之一。客户端在超时时间的四分之一处 ping 可恢复目标（MDS 和 OST），服务器将等待超时时间的一倍半再驱逐客户端、将其设置为"stale"。Lustre 客户端定期向指定的时间段内没有通信的服务器发送"ping"消息。文件系统中客户端和服务器之间的任何网络活动和 ping 的效用相同。
`ldlm_timeout`	服务器等待客户端回复初始 AST（锁取消请求）的时间。对于 OST，默认值为 20 秒；对于 MDS，默认值为 6 秒。如果客户端回复 AST，服务器将给它一个正常的超时（客户端超时时间的一半）来刷新任何脏数据并释放锁。
`fail_loc`	内部调试故障钩。默认值为 0，表示不会触发或注入任何故障。
`dump_on_timeout`	超时时触发 Lustre 调试日志的转储。默认值为 0，表示不会触发 Lustre 调试日志的转储。
`dump_on_eviction`	发生驱逐时触发 Lustre 调试日志的转储。默认值 0，表示不会触发 Lustre 调试日志的转储。

39.6. LNet 监控

LNet 信息位于`/proc/sys/lnet`的以下文件中：

- `peers` - 显示此节点已知的所有 NID，并提供有关队列状态的信息。

示例：

```
1 # lctl get_param peers
2 nid                refs   state  max  rtr  min   tx    min   queue
3 0@lo               1      ~rtr   0    0    0     0     0     0
4 192.168.10.35@tcp  1      ~rtr   8    8    8     8     6     0
5 192.168.10.36@tcp  1      ~rtr   8    8    8     8     6     0
6 192.168.10.37@tcp  1      ~rtr   8    8    8     8     6     0
```

表中各条目含义如下：

条目	说明
refs	引用计数。
state	如果节点是路由器，则表示路由器的状态。对应值有：NA －表示节点不是路由器。up/down－指示节点（路由器）是否为启动状态。
max	此对等节点的最大并发发送数。
rtr	可用的路由缓冲区信用值。
min	历史最低路由缓冲区信用值。
tx	可用的发送信用值。
queue	活动/排队中的发送总字节数。

信用值被初始化以允许一定数量的操作（如上方示例所示，max列为 8）。LNet 跟踪了监控时间段内看到的最低信用值，以显示此时间段内的高峰拥挤。低的信用值表示资源更加拥挤。

当前可用的信用值（传输信用值）显示在tx列中。最大发送信用额显示在max中，且永远不会发生变化。只要tx>=0，就可以通过（max - tx）得出当前活动的传输数量。一旦tx<0，则该值表示该对等体上因信用值低而排队的传输数量。

rtr列中显示的是可供对等节点使用的路由器缓冲区数量。通过使用相应模块的 peer_buffer_credits 模块参数，可以在 LND 层或 LNet 层单独配置路由信用值。如果没有明确设置路由信用值，则默认为 peer_credits 模块参数所定义的最大发送信用值。每当网关路由一个对等体的消息时，则递减该对等体的可用路由信用值。如果该值为零，那么消息将排队。如该值为负，则显示等待路由的排队消息的数量。如需获取从一个对等体路由的消息数量，可以通过（max_rtr_credits - rtr）得到。

LNet 还限制了并发发送和分配给单个对等节点的路由器缓冲区数量，从而避免对等节点占用所有资源。

- nis －显示该节点上队列当前健康状况。

示例：

```
# lctl get_param nis nid                refs  peer   max
tx   min  0@lo                3     0      0    0    0
192.168.10.34@tcp  4     8      256  256  252
```

表中条目的含义如下：

条目	说明
nid	网络接口。
refs	内部引用数。
peer	此 NID 上点对点的发送信用数，用于调整缓冲池的大小。
max	此 NID 的最大发送信用值。
tx	此 NID 当前可用的发送信用值。
min	此 NID 当前可用的最低信用值
queue	活动/排队中的发送总字节数。

分析：

(max - tx) 为当前活动的发送数量。活动发送量很大或越来越多则表示可能存在问题。

39.7. 在 OST 上分配空闲空间

可用空间分配使用循环法还是加权法，由 OST 之间可用空间的不平衡状况决定。OST 之间的可用空间相对平衡时，使用更快的循环分配器。任何两个 OST 的可用空间差别超过指定阈值时，使用加权分配器。

可以使用以下两个可调参数调整可用空间分布：

- lod.*.qos_threshold_rr －在此文件中设置从循环法切换到加权法的阈值。默认情况下，任何两个 OST 的不平衡度达到 17% 时，切换到加权算法。
- lod.*.qos_prio_free －可在该文件中调整加权分配器使用的加权优先级。增加qos_prio_free的值会增加每个 OST 上可用空间量的权重，减少条带在 OST 之间的分布。默认值为 91% 的权重基于可用空间重新平衡，9% 的权重基于 OST 平衡。当可用空间优先级设置为 100 时，加权器则完全基于可用空间，且不再适用条带化算法。
- osp.*.reserved_mb_low－如果可用空间低于此标准，则停止分配对象。默认值为总 OST 大小的 0.1%。（在 **Lustre 2.9** 中引入）
- osp.*.reserved_mb_high －如果可用空间高于此标准，则开始分配对象。默认值为总 OST 大小的 0.2%。（在 **Lustre 2.9** 中引入）

39.8. 配置锁

lru_size参数用于控制 LRU 缓存锁队列中的客户端锁数量。LRU 的大小是基于负载来进行动态优化的，具有不同工作负载（如登录/构建节点和计算/备份节点不同）的节点可用锁的数量也不同。

可用锁的总数是服务器 RAM 的函数。默认限制为每 1MB RAM50 个锁。如果内存压力过大，LRU 则更小。服务器上的锁数量被限制为每个服务器的 OST 数量、客户端数量、客户端上所设置的lru_size值三者的乘积，如下所示：

- 启用 LRU 大小自动调整，请将lru_size参数设置为 0。在这种情况下，lru_size参数将显示导出时使用的当前锁数量。LRU 大小自动调整默认启动。

- 指定最大锁数量，请将lru_size参数设置为非零值，通常是客户端的 CPU 数量的 100 倍左右。建议您仅在用户以交互方式访问文件系统的几个登录节点上增加 LRU 大小。

清除单个客户端上的 LRU，刷新客户端缓存而不更改lru_size值，请运行：

```
1 $ lctl set_param ldlm.namespaces.osc_name|mdc_name.lru_size=clear
```

如果将 LRU 大小设置得比现有未使用锁数量更小，则未使用的锁将被立即取消。使用clear取消所有锁而不更改该值。

注意

lru_size参数只能通过lctl set_param进行暂时设置（不能进行永久设置）。

禁用 LRU 大小调整，请在 Lustre 客户端上运行：

```
1 $ lctl set_param ldlm.namespaces.*osc*.lru_size=5000
```

确定授予的动态 LRU 大小调整的锁数，请运行：

```
1 $ lctl get_param ldlm.namespaces.*.pool.limit
```

lru_max_age参数用于控制 LRU 缓存锁队列中客户端锁的锁龄（时长）。这样可以限制未使用的锁在客户端缓存的时间，避免闲置的客户端持有锁的时间过长，从而减少了客户端和服务器的内存占用，同时也减少了服务器恢复期间的工作。

lru_max_age以毫秒为单位进行设置和打印，默认为 3900000 毫秒（65 分钟）。

从 **Lustre 2.11** 开始，除了以毫秒为单位设置最大锁龄外，还可以用s或ms作为后缀分别表示秒或毫秒。例如将客户端的最大锁龄设置为 15 分钟（900s）运行：

```
1 # lctl set_param ldlm.namespaces.*MDT*.lru_max_age=900s
2 # lctl get_param ldlm.namespaces.*MDT*.lru_max_age
3 ldlm.namespaces.myth-MDT0000-mdc-ffff8804296c2800.lru_max_age=900000
```

39.9. 设置 MDS 和 OSS 线程计数

MDS 和 OSS 线程计数的可调参数可用于设置最小和最大线程计数，或获取下表中所列服务的当前运行的线程数。

服务	说明
mds.MDS.mdt	主要元数据操作
mds.MDS.mdt_readpage	元数据 readdir
mds.MDS.mdt_setattr	元数据 setattr/close 操作
ost.OSS.ost	主要数据操作
ost.OSS.ost_io	批量数据 I/O
ost.OSS.ost_create	OST 对象预创建
ldlm.services.ldlm_canceld	DLM 锁取消
ldlm.services.ldlm_cbd	DLM 锁授予

对于每个服务，可调参数如下所示：

- 暂时地设置此参数：

```
# lctl set_param service.threads_min|max|started=num
```

- 永久地设置此参数：

```
# lctl conf_param obdname|fsname.obdtype.threads_min|max|started
```

Lustre 2.5 及以上版本请运行：

```
# lctl set_param -P service.threads_min|max|started
```

以下示例显示了如何设置线程计算及如何使用 service.threads_min|max|started参数获取 ost_io服务当前运行的线程数。

- 获取运行的线程数：

```
1 # lctl get_param ost.OSS.ost_io.threads_started
2 ost.OSS.ost_io.threads_started=128
```

- 设置线程数的最大值 (512)：

```
1 # lctl get_param ost.OSS.ost_io.threads_max
2 ost.OSS.ost_io.threads_max=512
```

- 为避免存储重载或针对请求数组，设置线程数的最大值 (256)：

```
1 # lctl set_param ost.OSS.ost_io.threads_max=256
2 ost.OSS.ost_io.threads_max=256
```

- 将线程数的最大值永久地设置为 256：

```
# lctl conf_param testfs.ost.ost_io.threads_max=256
```

Lustre 2.5 及以上版本请运行：

```
# lctl set_param -P ost.OSS.ost_io.threads_max=256
ost.OSS.ost_ io.threads_max=256
```

- 查看threads_max 设置已激活，请运行：

```
1 # lctl get_param ost.OSS.ost_io.threads_max
2 ost.OSS.ost_io.threads_max=256
```

注意

如果在文件系统运行时更改了服务线程数，则此更改在文件系统停止运行前可能不会生效。超过新设置的threads_max值的正在运行的服务线程不会被停止。

39.10. 调试日志

Lustre 会默认生成所有操作的详细日志以辅助调试。可通过lctl get_param debug找到调试的相关标志。

调试的开销会影响 Lustre 文件系统的性能。因此，为最小化调试对性能的影响，可以降低调试级别。这会影响存储在内部日志缓冲区中的调试信息量，但不会改变 syslog 的信息量。当您需要收集日志用于调试各种问题时，可以提高调试级别。

可以使用" 符号名称" 来设置调试掩码，其具体格式如下：

- 验证使用的调试级别，请运行以下命令来检查用于控制调试的参数：

```
# lctl get_param debug debug= ioctl neterror warning error
emerg ha config console
```

- 关闭调试（网络错误调试除外），请在所有相关节点上运行以下命令：

```
# sysctl -w lnet.debug="neterror" debug = neterror
```

- 如需完全关闭调试，请在所有相关节点上运行以下命令：

```
# sysctl -w lnet.debug=0 debug = 0
```

- 为生产环境设置适当的调试级别，请运行：

```
# lctl set_param debug="warning dlmtrace error emerg ha
rpctrace vfstrace" debug=warning dlmtrace error emerg ha
rpctrace vfstrace
```

此示例中显示的标志收集了足够的高级信息以帮助调试，但它们不会对性能造成任何严重影响。

- 为已经设置的标志添加新标志，请在每个标志前面加上"+"：

```
# lctl set_param debug="+neterror +ha" debug=+neterror +ha
# lctl get_param debug debug=neterror warning error emerg ha
console
```

- 移除标志，请在标志前附加"-"：

```
# lctl set_param debug="-ha" debug=-ha # lctl get_param
debug debug=neterror warning error emerg console
```

调试参数包括：

- subsystem_debug －控制子系统的调试日志。
- debug_path －指示被自动或手动触发时调试日志转储的位置。
- 默认路径是/tmp/lustre-log。

可使用以下命令设置这些参数：

```
sysctl -w lnet.debug={value}
```

其他参数：

- panic_on_lbug －当 Lustre 软件检测到内部问题（LBUG日志条目）时，会调用"panic"，从而导致节点崩溃。在配置内核崩溃转储实用程序时，这尤其有用。Lustre 软件检测到内部不一致时，将触发故障转储。
- upcall －允许您指定在遇到LBUG日志条目时调用的二进制文件的路径。使用以下四个参数调用此二进制文件：
- 字符串"LBUG"
- LBUG发生的文件
- 函数名称
- 文件中的行号

39.10.1. 解析 OST 统计数据

OST stats 文件可用于提供每个 OST 活动的统计信息。例如：

```
# lctl get_param osc.testfs-OST0000-osc.stats
snapshot_time                     1189732762.835363
ost_create                1
ost_get_info              1
ost_connect               1
ost_set_info              1
obd_ping                  212
```

可使用llstat实用程序监视一段时间内的统计信息。

要清除统计信息，请使用lcstat的-c选项。指定报告统计信息的频率（以秒为单位），请使用-i选项。在下面的示例中，使用了-c选项先清除统计信息，-i10选项设置为每 10 秒报告一次统计信息：

```
$ llstat -c -i10 ost_io

/usr/bin/llstat: STATS on 06/06/07
        /proc/fs/lustre/ost/OSS/ost_io/ stats on 192.168.16.35@tcp
snapshot_time                             1181074093.276072

/proc/fs/lustre/ost/OSS/ost_io/stats @ 1181074103.284895
Name        Cur.  Cur. #
            Count Rate Events Unit  last   min    avg       max    stddev
req_waittime 8    0    8    [usec]  2078   34     259.75    868    317.49
req_qdepth   8    0    8    [reqs]  1      0      0.12      1      0.35
req_active   8    0    8    [reqs]  11     1      1.39      2      0.52
reqbuf_avail 8    0    8    [bufs]  511    63     63.88     64     0.35
ost_write    8    0    8    [bytes] 169767 72914  212209.62 397579 91874.29

/proc/fs/lustre/ost/OSS/ost_io/stats @ 1181074113.290180
Name        Cur.  Cur. #
            Count Rate Events Unit  last    min   avg       max    stddev
req_waittime 31   3    39   [usec]  30011   34    822.79    12245  2047.71
req_qdepth   31   3    39   [reqs]  0       0     0.03      1      0.16
req_active   31   3    39   [reqs]  58      1     1.77      3      0.74
reqbuf_avail 31   3    39   [bufs]  1977    63    63.79     64     0.41
```

```
ost_write    30   3    39   [bytes] 1028467 15019 315325.16 910694 197776.51

/proc/fs/lustre/ost/OSS/ost_io/stats @ 1181074123.325560
Name        Cur.  Cur. #
            Count Rate Events Unit  last    min    avg       max    stddev
req_waittime 21   2    60   [usec]  14970   34     784.32    12245  1878.66
req_qdepth   21   2    60   [reqs]  0       0      0.02      1      0.13
req_active   21   2    60   [reqs]  33      1      1.70      3      0.70
reqbuf_avail 21   2    60   [bufs]  1341    63     63.82     64     0.39
ost_write    21   2    59   [bytes] 7648424 15019  332725.08 910694
   180397.87
```

此示例中每一列的含义如下：

参数	说明
Name	服务事件的名称。有关所跟踪的服务事件的说明，请参阅下表。
Cur. Count	在最后一个间隔中发送的每种类型的事件数。
Cur. Rate	最后一个时间间隔内每秒的事件数。
# Events	自事件清除以来此类事件的总数。
Unit	该统计信息的度量单位（微秒、请求数、缓冲区数）。
last	这些事件在它们到达的最后一个时间间隔内的平均速率（每单位/事件）。 例如，在上面的ost_destroy例子中，对于前 10 秒内的 400 个对象来说，每次ost_destroy平均需要 736 微秒。
min	服务启动以来的最低速率（每单位/事件）。
avg	平均速率。
max	最高速率。
stddev	标准差（一些情况下不列入计算）。

下表列出了所有服务共有的事件：

参数	说明
req_waittime	请求在可用服务器线程处理之前在队列中等待的时间。
req_qdepth	此服务的队列中等待处理的请求数。

参数	说明
req_active	当前正在处理的请求数。
reqbuf_avail	此服务的未经请求的 lnet 请求缓冲区数。

下表列出了一些与服务有关的具体事件：| 参数 | 说明 | | ---------- | ---------- | | ldlm_enqueue | 查询锁入队的时间（包括在 MDS 上打开文件的时间）。| | mds_reint | 处理一条 MDS 修改记录所需的时间（包括create, mkdir, | | unlink, rename和setattr）。|

39.10.2. MDT 统计数据解析

MDT stats 文件可用于帮助 MDS 跟踪 MDT 统计信息。以下为 MDT 统计信息文件的输出示例。

```
# lctl get_param mds.*-MDT0000.stats
snapshot_time                   1244832003.676892 secs.usecs
open                            2 samples [reqs]
close                           1 samples [reqs]
getxattr                        3 samples [reqs]
process_config                  1 samples [reqs]
connect                         2 samples [reqs]
disconnect                      2 samples [reqs]
statfs                          3 samples [reqs]
setattr                         1 samples [reqs]
getattr                         3 samples [reqs]
llog_init                       6 samples [reqs]
notify                          16 samples [reqs]
```

第四十章用户实用程序

40.1. lfs

lfs实用程序可用于用户配置和监控。

40.1.1. 梗概

```
lfs
```

```
lfs changelog [--follow] mdt_name [startrec [endrec]]
lfs changelog_clear mdt_name id endrec
lfs check mds|osts|servers
lfs data_version [-nrw] filename
lfs df [-i] [-h] [--pool]-p fsname[.pool] [path] [--lazy]
lfs find [[!] --atime|-A [-+]N] [[!] --mtime|-M [-+]N]
         [[!] --ctime|-C [-+]N] [--maxdepth|-D N] [--name|-n pattern]
         [--print|-p] [--print0|-P] [[!] --obd|-O ost_name[,ost_name...]]
         [[!] --size|-S [+-]N[kMGTPE]] --type |-t {bcdflpsD}]
         [[!] --gid|-g|--group|-G gname|gid]
         [[!] --uid|-u|--user|-U uname|uid]
         dirname|filename
lfs getname [-h]|[path...]
lfs getstripe [--obd|-O ost_name] [--quiet|-q] [--verbose|-v]
              [--stripe-count|-c] [--stripe-index|-i]
              [--stripe-size|-s] [--pool|-p] [--directory|-d]
              [--mdt-index|-M] [--recursive|-r] [--raw|-R]
              [--layout|-L]
              dirname|filename ...
lfs setstripe [--size|-s stripe_size] [--stripe-count|-c stripe_count]
                            [--overstripe-count|-C stripe_count]
              [--stripe-index|-i start_ost_index]
              [--ost-list|-o ost_indicies]
              [--pool|-p pool]
              dirname|filename
lfs setstripe -d dir
lfs osts [path]
lfs pool_list filesystem[.pool]| pathname
lfs quota [-q] [-v] [-h] [-o obd_uuid|-I ost_idx|-i mdt_idx]
          [-u username|uid|-g group|gid|-p projid] /mount_point
lfs quota -t -u|-g|-p /mount_point
lfs setquota {-u|--user|-g|--group|-p|--project} uname|uid|gname|gid|projid
             [--block-softlimit block_softlimit]
             [--block-hardlimit block_hardlimit]
             [--inode-softlimit inode_softlimit]
             [--inode-hardlimit inode_hardlimit]
```

```
38              /mount_point
39 lfs setquota -u|--user|-g|--group|-p|--project uname|uid|gname|gid|projid
40              [-b block_softlimit] [-B block_hardlimit]
41              [-i inode-softlimit] [-I inode_hardlimit]
42              /mount_point
43 lfs setquota -t -u|-g|-p [--block-grace block_grace]
44              [--inode-grace inode_grace]
45              /mount_point
46 lfs setquota -t -u|-g|-p [-b block_grace] [-i inode_grace]
47              /mount_point
48 lfs help
```

注意

在上面的例子中，*/mount_point*参数指的是 Lustre 文件系统的挂载点。

在老版本中，`lfs quota`的输出非常详细，包含集群范围的配额统计信息（包括用户/组的集群范围限制以及用户/组的集群范围使用情况），以及每个 MDS/OST 的统计信息。现在，默认情况下，最新的`lfs quota`仅提供集群范围的统计信息。要获取集群范围限制、使用情况以及统计信息的完整报告，请在`lfs quota`中使用`-v`选项。

（Lustre 2.8 引入）

`quotacheck`、`quotaon`和`quotaoff`子命令在 Lustre 2.4 版本中弃用，并在 Lustre 2.8 版本中完全删除。关于配置和检查配额的细节，请参见第 25.2 节，``启用磁盘配额"。

40.1.2. 说明

`lfs`实用程序可用于创建具有特定条带模式的新文件、确定默认条带模式、收集特定文件的扩展属性（对象编号和位置）、查找具有特定属性的文件、列出 OST 信息或设置配额限制。它可以在没有任何参数的情况下以交互方式进行调用，也可以在非交互模式下使用支持的某一参数进行调用。

40.1.3. 选项

下表列出了 `lfs` 的一些不同选项。获取完整列表请在 `lfs` 窗口下输入`help`。

选项	说明
changelog	显示 MDT 上的元数据更改。起点和终点是可选的。 `--follow`选项会阻止新的更改；此选项仅当直接运行在 MDT 节点上时有效。

选项	说明
changelog_clear	表示某使用者*id*已对 *endrec*之前的 changelog 记录不再感兴趣，即允许 MDT 释放磁盘空间。*endrec *为 0 则表示当前的最后一条记录。必须使用lctl在 MDT 节点上注册 Changelog 使用者。
check	显示 MDS 或 OST（在命令中指定）或所有服务器（MDS 和 OST）的状态。
data_version [-nrw] filename	显示文件数据的当前版本。如果指定了 -n，则读取数据版本，不加锁。因此，如果文件系统客户端上有脏的缓存，数据版本可能已经过时，但这个选项不会强制刷新数据，对文件系统的影响较小。如果指定了 -r，不会强制刷新数据，对文件系统的影响较小。如果指定了 -r，数据版本会在客户端上的脏页被刷新后读取。如果指定了 -w，数据版本会在客户端的所有缓存页面被刷新后读取。即使使用-r或-w，也有可能出现竞争的情况，所以在操作前后都要检查数据版本，以确定数据在操作过程中没有发生变化。数据版本是文件中所有数据对象的最后一次提交的事务编号之和。HSM 策略引擎使用它验证文件数据在归档操作期间或在发布操作之前是否被更改。在非阻塞模式下进行 OST 迁移时也使用它，主要用于验在数据复制证文件数据过程中没有被更改。
df [-i] [-h] [--pool|-p fsname [. pool] [path] [--lazy]	使用-i报告每个 MDT 或 OST 的文件系统磁盘空间使用情况、inode 使用情况，或者 OST 子集的实用情况（如果使用-p选项指定了池）。默认报告所有已挂载 Lustre 文件系统的使用情况。如果包含path 选项，则仅报告指定文件系统的使用情况。如果包含-h选项，则以方便人类查看的格式输出结果，为 Mega-、Giga-、Tera-、Peta- 或 Exabytes 使用 SI base-2 后缀。如果指定了--lazy选项，则将跳过当前与客户端断开连接的所有 OST。使用--lazy选项可阻止 OST 脱机时的df输出，仅返回当前可以访问的 OST 上的空间。可以启用llite.*.lazystatfs可调参数，使其成为所有statfs()操作的默认行为。
find	搜索以给定目录/文件名为根的目录树以查找与给定参数匹配的文件。在选项前使用!表示否定（即与参数不匹配的文件）。在数值之前使用+表示搜索与参数本身或更大值匹配的文件；使用-表示搜索与参数本身或更小值匹配的文件。

选项	说明
--atime	最后一次访问是 N*24 小时前的文件（无法保证atime在集群中保持一致）。客户端处理读请求时会更新atime，默认情况下，atime值将暂时写入 OST。文件关闭时，atime值将持久写入 MDT。但磁盘atime只有在atime超过 60 秒 (mds.*.atime_diff) 时才会更新。（Lustre 2.14 引入）在 Lustre 2.14 中，可以将 OST 设置为与每个对象一起持久存储 atime，这样在使用类似名为 obdfilter.*.atime_diff 参数长时间打开的文件时，能够获得更准确的持久 atime 更新。Lustre 软件考虑了所有 OST 和 MDT 的最新时间。如果asetattr 由用户设置，则其在 MDT 和 OST 上都会更新，并允许atime值更改为更新的值。
--ctime	上次文件状态变更发生在 N*24 小时前的文件。
--mtime	上次文件内容变更发生在 N*24 小时前的文件。
--obd	在特定 OST 上有对象的文件。
--size	特定文件大小的文件。文件大小默认单位为 bytes，或者给出后缀为 kilo-, Mega-, Giga-, Tera-, Peta-的不同单位。
--type	具有类型 block、character、directory、pipe、file、symlink、socket、door 的文件 (在 Solaris 操作系统中使用)。
--uid	有指定用户数字 ID 的文件。
--user	指定用户（可使用用户数字 ID）所有的文件。
--gid	有指定组 ID 的文件。
--group	指定组（可使用组数字 ID）所有的文件。
--maxdepth	查找目标树的最多下降 N 级。
--print/--print0	打印完整文件名，新的一行或 NULL 字符跟随其后。
osts [path]	列出文件系统的所有 OST。如果指定了挂载 Lustre 文件系统的路径，则仅显示属于此文件系统的 OST。
getname [path…]	列出与每个 Lustre 挂载点关联的所有 Lustre 文件系统实例。如果未指定路径，则会询问所有 Lustre 挂载点。如果提供了路径列表，则将给出相应的路径实例。如果某路径不是 Lustre 实例，则将返回"No such device"。

选项	说明
getstripe	列出给定文件名或目录的条带信息。默认返回条带计数、条带大小和偏移量。如果您只需要特定的条带信息，可选择 `--stripe-count`、`--stripe-size`、`--stripe-index`、`--layout`或`--pool`以及这些选项的各种组合以用于检索特定信息。如果指定了`--raw`选项，则打印条带信息时不会将文件系统默认值值替换为未指定的字段。如果未设置条带化 EA，则将分别打印条带计数、大小和偏移量为 0、0 和 -1。`--mdt-index` 打印给定目录下 MDT 的索引（在 **Lustre 2.4** 中引入）。
--obd ost_name	列出在特定 OST 上具有对象的文件。
--quiet	列出有关文件的对象 ID 的详细信息。
--verbose	打印附加的条带信息。
--stripe-count | 列	出条带计数（使用的 OST 个数）。
--index	列出文件系统每个 OST 的索引。
--offset	列出文件条带开始的 OST 索引。
--pool	列出文件所属的池。
--size	列出条带大小（在移至下一个 OST 前写入当前 OST 的数据量）
--directory	列出指定目录的条目而不是其内容（与`ls -d`的方式相同）。
--recursive	递归到所有子目录。
setstripe	使用指定文件布局（条带模式）创建新文件。（在使用`setstripe`之前，目录必须存在，文件不能存在）
--stripe-count stripe_cnt	用于将文件条带化的 OST 数。当`stripe_cnt`为 0 时使用文件系统范围的默认条带计数（默认值为 1）。当`stripe_cnt`为 -1 时，在所有可用 OST 上进行条带化。
--overstripe-count stripe_cnt	| 与`--stripe-count`相同，但允许 overstriping，当`stripe_cnt`大于 OST 的数量，就会将每个 OST 条带化为 2 及上个条带。overstriping 对于将条带数与进程数匹配很用帮助，且对于非常快的 OST 来说，每个 OST 只对应一个条带，不足以获得该 OST 的 | 全部性能。

选项	说明
--size stripe_size	移至下一个 OST 之前在当前 OST 上存储的字节数。stripe_size为 0 时，使用文件系统的默认条带大小（默认为 1 MB）。可使用 k（KB）、m（MB）或 g（GB）进行指定。（默认stripe_size为 0，默认的start-ost为 -1，注意不要混淆！如果把start-ost设置为 0 ，则所有新文件创建都发生在 OST 0 上，这一般不是个好主意）
--stripe-index start_ost_index	文件条带化开始的 OST 索引（基数为 10，从 0 开始）。start_ost_index值为 -1（默认值），允许 MDS 选择起始索引。这意味着 MDS 会根据需要选择起始 OST。我们强烈建议选择此默认值，它允许了 MDS 根据需要实现空间和负载平衡。start_ost_index的值与 MDS 对文件中的剩余条带使用循环算法还是 QoS 加权分配无关。
--ost-index	文件条带化开始的 OST 索引（基数为 10，从 0 开始）。
--pool pool	用于条带化的预定义 OST 池名称。还使用了stripe_cnt，stripe_size 和start_ost值。start-ost值必须是池的一部分，否则将返回错误。
setstripe -d	删除指定目录上的默认条带化设置。
pool_list {filesystem}[.poolname]| {pathname}	列出文件系统或路径名中的池，或文件系统池中的 OST。
quota [-q][-v] [-o obd_uuid| -i mdt_idx| -I ost_idx][-u|-g|-p uname uid|gname|gid| projid] /mount_point	显示完整文件系统或特定 OBD 上对象的磁盘使用情况和限制。可以指定用户、组名称或 usr，组和项目 ID。如果所有用户、组项目 ID 都被省略了，则显示当前 UID/GID 的配额。使用-q选项将不会打印其他描述（包括列标题），它使用零来填充宽限期那一列中的空格（当没有设置宽限期时）来确保列数一致。使用-v选项将提供更详细（每 OBD 统计信息）的输出。
quota -t -u|-g| -p / mount_point	显示用户（-u）、组（-g）或项目（-p）配额的块和 inode 宽限时间。

选项	说明
setquota {-u|-g -p *uname|uid| gname|gid| projid} [--block -softlimit block_softlimit [--block- hardlimit block_hardlimit] --inode- softlimit inode_ softlimit [--inode- hardlimit inode_hardlimit] / mount_point	为用户、组或项目设置文件系统配额。可以使用 --{block|inode} --{softlimit|hardlimit}或它们的简略版-b，-B，-i，-I指定限制。用户可以设置 1、2、3 或 4 个限制。此外，可以使用特殊后缀-b，-k，-m，-g，-t和-p指定限制，分别表示 -b，-k，-m，-g，-t和-p指定限制，分别表示 1、2^{10}、^{2}20、2^{30}、^{2}40 和 2^50 的单位。默认情况下，块限制单位为 1 千字节（1024），块限制始终为千字节（即以字节为单位）。请参见下一小节的示例。（支持旧的setquota接口，但在将来的 Lustre 版本中可能被移除。）
setquota -t -u| -g|-p [--block- grace block_grace [--inode-grace inode_grace] / mount_point	设置用户或组的文件系统配额宽限时间。宽限时间以"XXwXXdXXhXXmXXs" 格式或整数秒进行指定。请参见下一小节的示例。
help exit/quit	提供有关不同 lfs 参数的简略帮助。退出lfs交互会话。

40.1.4. 示例

创建在两个 OST 上条带化的文件，每个条带为 128 KB。

```
1 $ lfs setstripe -s 128k -c 2 /mnt/lustre/file1
```

删除给定目录上的默认条带模式。新文件使用默认条带模式。

```
1 $ lfs setstripe -d /mnt/lustre/dir
```

列出给定文件的对象分配的详细信息。

```
1 $ lfs getstripe -v /mnt/lustre/file1
```

列出所有已挂载的 Lustre 文件系统和相应的 Lustre 实例。

```
1 $ lfs getname
```

高效地列出给定目录及其子目录中的所有文件。

```
$ lfs find /mnt/lustre
```

递归地列出给定目录中超过 30 天的所有常规文件。

```
$ lfs find /mnt/lustre -mtime +30 -type f -print
```

递归地列出给定目录中在 OST2-UUID 上有对象的所有文件。lfs服务器查看命令可用于检查所有服务器（MDT 和 OST）的状态。

```
$ lfs find --obd OST2-UUID /mnt/lustre/
```

列出文件系统中的所有 OST。

```
$ lfs osts
```

以方便人类查看的格式列出每个 OST 和 MDT 的空间使用情况。

```
$ lfs df -h
```

列出每个 OST 和 MDT 的 inode 使用情况。

```
$ lfs df -i
```

列出特定 OST 池的空间或 inode 使用情况。

```
$ lfs df --pool
filesystem[.
pool] |
pathname
```

列出用户'bob' 的配额情况。

```
$ lfs quota -u bob /mnt/lustre
```

列出项目号为'1' 的配额情况。

```
$ lfs quota -p 1 /mnt/lustre
```

显示在 /mnt/lustre上的用户配额宽限时间。

```
$ lfs quota -t -u /mnt/lustre
```

设置用户'bob' 的配额，为 1 GB 的组配额硬限制，2 GB 组配额软限制。

```
$ lfs setquota -u bob --block-softlimit 2000000 --block-hardlimit 1000000
/mnt/lustre
```

设置用户配额的宽限时间：块配额为 1000 秒，inode 配额为 1 周和 4 天。

```
$ lfs setquota -t -u --block-grace 1000 --inode-grace 1w4d /mnt/lustre
```

检查所有服务器（MDT 和 OST）的状态。

```
$ lfs check servers
```

在池my_pool中创建在两个 OST 上条带化的文件。

```
$ lfs setstripe --pool my_pool -c 2 /mnt/lustre/file
```

列出已挂载的 Lustre 文件系统/mnt/luster定义的池:

```
$ lfs pool_list /mnt/lustre/
```

列出作为文件系统my_fs中池my_pool的成员的 OST。

```
$ lfs pool_list my_fs.my_pool
```

查找与poolA关联的所有目录/文件。

```
$ lfs find /mnt/lustre --pool poolA
```

查找与任何池都没有关联的目录/文件。

```
$ lfs find /mnt//lustre --pool ""
```

查找与池有关联的所有目录/文件。

```
$ lfs find /mnt/lustre ! --pool ""
```

将目录与池my_pool关联，以便在池中创建所有新文件和目录。

```
$ lfs setstripe --pool my_pool /mnt/lustre/dir
```

40.2. lfs_migrate

lfs_migrate 实用程序是在 OST 间迁移文件数据的一个简单的工具。.

40.2.1. 梗概

```
lfs_migrate [lfs_setstripe_options]
    [-h] [-n] [-q] [-R] [-s] [-y] [file|directory ...]
```

40.2.2. 说明

lfs_migrate实用程序是用于帮助文件数据在 Lustre OST 之间迁移的工具。它使用提供的lfs setstripe选项将指定的文件复制到临时文件（如果有的话），有选择性地验证文件内容是否未更改，然后交换临时文件和原始文件的布局（即 OST 对象）（**Lustre 2.5** 及更高版本），或将临时文件重命名为原始文件名。这允许用户或管理员平衡 OST 之间的空间使用，或者将文件从开始显示硬件问题（尽管仍然可用）或将被删除的 OST 上移除。

注意

在 **Lustre 2.5** 之前的版本中，lfs_migrate 没有与 MDS 整合。使用它作用在正在被其他应用程序修改的文件是不安全的，因为该文件迁移是通过文件的副本和重命名实现的。**Lustre 2.5** 及更高版本中，新文件布局将与现有文件布局交换，从而确保保留用户可见的 inode 编号、文件的打开文件句柄和锁。

要迁移的文件可使用命令行参数指定。如果在命令行中指定了目录，则表示迁移目录中的所有文件。如果在命令行中未指定文件，则将从标准输入读取文件列表（如使用 lfs find查找特定 OST 上的文件或匹配其他文件属性，或其他工具标准生成输出的文件列表）。

除非通过命令行选项另行指定，否则由 MDS 上的文件分配策略决定新文件的位置。同时，把是否通过lctl在 MDS 上禁用特定 OST（防止在其上分配新文件），是否有一些 OST 过度拥挤（减少放置在这些 OST 上的文件数），是否父目录有指定的默认文件条带化设置（可能会影响新文件的条带数，条带大小，OST 池或 OST 索引）都列入考虑。

注意

在某些情况下，lfs_migrate实用程序还可以用来减少文件碎片。文件碎片通常会降低 Lustre 文件系统的性能，多在在老化的文件系统上或许多线程写入文件时发生。如果相对正在复制的文件有足够的没那么碎片化的可用空间（或者文件是在文件系统接近满溢时写入），则使用lfs_migrate重写文件将导致迁移的文件中碎片的减少。filefrag工具可用于报告文件碎片，请参见本章第 3 节"filefrag" 。

只要文件的扩展长度为几十兆字节（read_bandwidth * seek_time）或更大，碎片便不会显著影响文件的读取性能。这是因为读取管道可以通过磁盘上的大量读取来填充（即使偶尔需要进行磁盘搜索）。

40.2.3. 选项

lfs_migrate 支持的选项如下：

选项	说明
-c stripecount	使用指定的条带计数重新划分条带文件。此选项可能不会与-R 选项同时指定。
-h	显示帮助信息。
-l	使用硬链接迁移文件（默认情况下为跳过）。具有多个硬链接的文件将被lfs_migrate拆分为多个单独的文件，因此默认情况下会跳过它们以避免破坏硬链接。
-n	只打印将被迁移的文件名。
-q	静默运行，不打印文件名和状态。

选项	说明
-R	使用默认目录条带设置重新划分文件。此选项可能不会与-c 选项同时指定。
-s	在迁移完成后跳过文件数据校对。默认情况下，会将迁移的文件与原始文件进行比较以验证其迁移正确性。
-y	在没有提示的情况下针对使用警告应答'y'（对于脚本，请谨慎使用）。
-0	在标准输入中，期待有 NUL 结尾的文件名，如lfs find -print0或find -print0生成的文件名。这样就可以正确处理带有内嵌新行的文件名。

40.2.4. 示例

重新平衡 /mnt/lustre/dir中的所有文件。

```
1 $ lfs_migrate /mnt/lustre/file
```

迁移 OST004 上/test文件系统中所有一天前且超过 4 GB 的文件。

```
1 $ lfs find /test -obd test-OST0004 -size +4G -mtime +1 | lfs_migrate -y
```

40.3. filefrag

e2fsprogs包中包含了filefrag工具，该工具将报告文件碎片的范围。

40.3.1. 梗概

```
1 filefrag [ -belsv ] [ files...  ]
```

40.3.2. 说明

filefrag实用程序可用于报告给定文件的碎片范围，它使用FIEMAP ioctl从 Lustre 文件中获取范围信息。这是非常高效和快速的，即使文件非常大。

在默认模式中，filefrag将打印文件中物理上不连续的范围数量。在范围或详细模式下，将打印每个范围的详细信息（如在每个 OST 上分配的块）。Lustre 文件系统中，范围以设备偏移顺序（首先是当前 OST 的所有范围，然后是下一个 OST 等）而不是文

件逻辑偏移顺序进行打印。如果使用文件逻辑偏移顺序，则 Lustre 条带化设置将使输出非常冗长且难以查看是否存在文件碎片。

注意

只要文件的范围长度为几十兆字节或更长（即read_bandwidth * seek_time> extent_length），文件的读取性能就不怎么会受碎片的影响。这是因为文件 readahead 可以充分利用磁盘带宽（即使存在偶尔的磁盘搜索）。

在默认模式中，filefrag将返回文件中物理上不连续的范围数量。在范围或详细模式下，会显示每个范围的详细信息。对于 Lustre 文件系统来说，范围按照设备偏移顺序而不是逻辑偏移顺序打印。

40.3.3. 选项

filefrag的选项和其说明如下所示：

选项	说明
-b	输出使用 1024 字节的块大小，为 Lustre 文件系统的默认块大小设置。OST 可能使用不同的块大小。
-e	输出使用范围模式。这也是详细模式下 Lustre 文件的默认值。
-l	以 LUN 偏移顺序显示范围。这是 Lustre 唯一可用的模式。
-s	在请求映射之前，将任何未写入的文件数据同步到磁盘。
-v	检查文件碎片时，使用详细模式打印文件布局（包括文件中每个范围的逻辑到物理的映射和 OST 索引）。

40.3.4. 示例

默认输出：

```
1 $ filefrag /mnt/lustre/foo
2 /mnt/lustre/foo: 13 extents found
```

详细模式下的范围信息：

```
1 $ filefrag -v /mnt/lustre/foo
2 Filesystem type is: bd00bd0
3 File size of /mnt/lustre/foo is 1468297786 (1433888 blocks of 1024 bytes)
4  ext:     device_logical:        physical_offset: length:  dev: flags:
5    0:        0..  122879: 2804679680..2804802559: 122880: 0002: network
```

```
  1:   122880..  245759: 2804817920..2804940799: 122880: 0002: network
  2:   245760..  278527: 2804948992..2804981759:  32768: 0002: network
  3:   278528..  360447: 2804982784..2805064703:  81920: 0002: network
  4:   360448..  483327: 2805080064..2805202943: 122880: 0002: network
  5:   483328..  606207: 2805211136..2805334015: 122880: 0002: network
  6:   606208..  729087: 2805342208..2805465087: 122880: 0002: network
  7:   729088..  851967: 2805473280..2805596159: 122880: 0002: network
  8:   851968..  974847: 2805604352..2805727231: 122880: 0002: network
  9:   974848.. 1097727: 2805735424..2805858303: 122880: 0002: network
 10:  1097728.. 1220607: 2805866496..2805989375: 122880: 0002: network
 11:  1220608.. 1343487: 2805997568..2806120447: 122880: 0002: network
 12:  1343488.. 1433599: 2806128640..2806218751:  90112: 0002: network
/mnt/lustre/foo: 13 extents found
```

40.4. mount

挂载 Lustre 文件系统，可使用标准的mount(8)Linux 命令，它将执行/sbin/mount.lustre命令以完成安装。mount 命令支持 Lustre 文件系统的这些特定选项:

服务器选项	说明
abort_recov	启动目标时中止恢复
nosvc	仅启动 MGS/MGC 服务器
nomgs	在没有启动 MGS 的情况下，启动并置有 MGS 的 MDT 。
exclude	启动已死亡的 OST
md_stripe_cache_size	使用条带 raid 配置为服务器端磁盘设置条带缓存大小

客户端选项	说
flock/noflock/localflock	启用或禁用全局或本地 flock 支持
user_xattr/nouser_xattr	启用或禁用用户扩展属性
user_fid2path/nouser_fid2path	启用或禁用常规用户的 FID 到路径转换（**Lustre 2.3** 中引入）
retry=	客户端挂载文件系统的重试次数

客户端选项	说

40.5. 处理超时

超时是应用程序挂起的最常见原因。在涉及 MDS 或 OST 故障切换的超时发生后，应用程序将在连接建立后再尝试访问之前断开连接的资源。

当客户端执行远程操作时，它会允许服务器一个合理的响应时间。如果由于网络故障、服务器挂起或任何其他原因，服务器未回复，则会发生超时。超时需要恢复。

发生超时时，类似的相应消息将在客户端控制台或/var/log/messages中显示:

```
1 LustreError: 26597:(client.c:810:ptlrpc_expire_one_request()) @@@ timeout
2
3 req@a2d45200 x5886/t0 o38->mds_svc_UUID@NID_mds_UUID:12 lens 168/64 ref 1 fl
4
5 RPC:/0/0 rc 0
```

第四十一章程序接口

41.1. 用户/组的上行调用（upcall）

本节描述了补充用户/组的上行调用，它将允许 MDS 进行检索和验证分配特定用户的补充组，从而避免使用 RPC 将所有补充组从客户端传递到 MDS。

41.1.1. 梗概

MDS 使用lctl get_param mdt.${FSNAME}-MDT{xxxx}.identity_upcall指定的实用程序来查找 UID，以便检索用户是否为补充组成员。结果暂时缓存在内核中(保存时间默认为五分钟)，以消除重复调用用户空间的开销。

41.1.2. 说明

identity_upcall参数包含用于将数字 UID 解析为组成员身份列表的可执行文件的路径。该工具将打开/proc/fs/lustre/mdt/${FSNAME}-MDT{xxxx}/identity_info参数文件并按identity_downcall_data数据结构（请参见本章第 1.4 节" 数据结构"）进行填充。回调可以通过lctl set_param mdt.${FSNAME}-MDT{xxxx}.identity_upcall进行配置。

有关回调程序的示例，请参阅 Lustre 源代码分发中的luster/utils/l_getidentity.c。

41.1.2.1. 主/备组 主/备组的机制如下：

- MDS 发出回调（每个 MDS）将数字 UID 映射到补充组。
- 如果没有回调或回调失败，则最多将添加一个由客户端提供的补充组。
- 默认的回调是/usr/sbin/l_getidentity，它通过与用户/组数据库交互来获取 UID/GID/suppgid(补充 GID)。用户/组数据库依赖于身份验证的配置方式，例如本地的/etc/passwd、网络信息服务（NIS）、轻量级目录访问协议（LDAP）或 SMB 域服务。禁用回调，请将回调接口被设置为 NONE。MDS 使用客户端提供的 UID/GID/suppgid。
- MDS 将在有限的时间内等待组回调程序完成，避免 MDS 线程和客户端因错误访问远程服务节点而挂起。在 MDS 在没有补充数据的情况下，回调程序必须在 30s 内完成。使用 lctl set_param mdt.*.identity_acquire_expire=seconds可以在 MDS 上设置回调超时(以秒为单位)。
- 默认组回调由mkfs.lustre设置，请使用tunefs.lustre --param或lctl set_param -P mdt.FSNAME-MDTxxxx.identity_upcall=path设置自定义回调。
- 组调用 (downcall) 数据由内核缓存，以避免同一用户的重复回调导致 MDS 速度变慢。默认情况下，这个内核中的缓存会在 1200s 后（20 分钟）过期。MDS 上的缓存时长（以秒为单位）可以通过以下方法设置：lctl set_param mdt.*.identity_expire=seconds。
- 为了强制驱逐缓存的身份数据（例如在添加或删除补充组中的用户后，可以通过 lctl set_param mdt.*.identity_flush=UID 在 MDS 上刷新特定数字 UID 的缓存条目，如需缓存中刷新所有用户的缓存记录，使用 -1 的 UID，可通过：lctl set_param mdt.*.identity_flush=-1 。

41.1.3. 数据结构

```
1 struct perm_downcall_data{
2       __u64 pdd_nid;
3       __u32 pdd_perm;
4       __u32 pdd_padding;
5 };
6
```

```
struct identity_downcall_data{
     __u32        idd_magic;
     :
     :
```

第四十二章在 C 程序中设置 Lustre 属性 (llapi)

42.1. llapi_file_create

使用 llapi_file_create 为新文件设置 Lustre 属性。

42.1.1. 梗概

```
#include <lustre/lustreapi.h>

int llapi_file_create(char *name, long stripe_size, int stripe_offset, int
   stripe_count, int stripe_pattern);
```

42.1.2. 说明

定义llapi_file_create()函数对文件描述符的 Lustre 文件系统条带信息进行设置，并随后使用open()访问文件描述符。

选项	说明
llapi_file_create()	如果文件已存在，则此参数返回EEXIST。如果条带参数无效，则此参数返回EINVAL。
stripe_size	此值必须是系统页面大小的偶数倍，如getpagesize()所示。Lustre 条带默认大小为 4MB。
stripe_offset	表示此文件的起始 OST。
stripe_count	指示此文件条带化使用的 OST 数。
stripe_pattern	表示此文件的 RAID 模式。

注意

目前，仅支持RAID 0。使用系统默认值，请设置: stripe_size = 0, stripe_offset = -1, stripe_count = 0, stripe_pattern = 0。

42.1.3. 示例

系统默认大小为 4MB。

```
1 char *tfile = TESTFILE;
2 int stripe_size = 65536
```

以默认配置启动：

```
1 int stripe_offset = -1
```

以默认配置启动：

```
1 int stripe_count = 1
```

设置单个条带，请运行：

```
1 int stripe_pattern = 0
```

目前，仅支持RAID 0。

```
1 int stripe_pattern = 0;
2 int rc, fd;
3 rc = llapi_file_create(tfile, stripe_size,stripe_offset,
    stripe_count,stripe_pattern);
```

结果代码被反转，可能会返回EINVAL或ioctl错误。

```
1 if (rc) {
2 fprintf(stderr,"llapi_file_create failed: %d (%s) 0, rc,
    strerror(-rc));return -1; }
```

llapi_file_create 将关闭文件描述符。您必须重新打开描述符，请运行：

```
1 fd = open(tfile, O_CREAT | O_RDWR | O_LOV_DELAY_CREATE, 0644); if (fd < 0)
    \ {
2 fprintf(stderr, "Can't open %s file: %s0, tfile,
3 str-
4 error(errno));
5 return -1;
6 }
```

42.2. llapi_file_get_stripe

使用 llapi_file_get_stripe 获取 Lustre 文件系统上的文件或目录的条带信息。

42.2.1. 梗概

```
#include <lustre/lustreapi.h>

int llapi_file_get_stripe(const char *path, void *lum);
```

42.2.2. 说明

llapi_file_get_stripe() 函数将以下列格式之一返回 *lum*（应指向足够大的内存区域）中文件或目录 *path* 的条带信息：

```
struct lov_user_md_v1 {
__u32 lmm_magic;
__u32 lmm_pattern;
__u64 lmm_object_id;
__u64 lmm_object_seq;
__u32 lmm_stripe_size;
__u16 lmm_stripe_count;
__u16 lmm_stripe_offset;
struct lov_user_ost_data_v1 lmm_objects[0];
} __attribute__((packed));
struct lov_user_md_v3 {
__u32 lmm_magic;
__u32 lmm_pattern;
__u64 lmm_object_id;
__u64 lmm_object_seq;
__u32 lmm_stripe_size;
__u16 lmm_stripe_count;
__u16 lmm_stripe_offset;
char lmm_pool_name[LOV_MAXPOOLNAME];
struct lov_user_ost_data_v1 lmm_objects[0];
} __attribute__((packed));
```

选项	说明
lmm_magic	指定返回的条带化信息的格式。LOV_MAGIC_V1用于 lov_user_md_v1。LOV_MAGIC_V3用于 lov_user_md_v3。

选项	说明
lmm_pattern	存有条纹图案。此 Lustre 软件版本中只支持LOV_PATTERN_RAID0。
lmm_object_id	存有 MDS 对象 ID。
lmm_object_gr	存有 MDS 对象组。
lmm_stripe_size	存有条带大小（字节）。
lmm_stripe_count	存有文件条带化的 OST 数。
lmm_stripe_offset	存有文件条带化的起始 OST 索引。
lmm_pool_name	存有文件所属的 OST 池名称。
lmm_objects	一组包含以下格式的文件信息（per OST）的 lmm_stripe_count 成员： struct lov_user_ost_data_v1 { __u64 l_object_id; __u64 l_object_seq; __u32 l_ost_gen; __u32 l_ost_idx; } __attribute__((packed));
l_object_id	存有 OST 对象 ID。
l_object_seq	存有 OST 对象组。
l_ost_gen	存有 OST 的索引生成。
l_ost_idx	存有 LOV 中的 OST 索引。

42.2.3. 返回值

llapi_file_get_stripe() 将返回：
成功：0；
失败：!= 0，同时，将设置相应的 errno 。

42.2.4. 错误

错误	说明
ENOMEM	分配内存失败
ENAMETOOLONG	路径过长
ENOENT	路径没有指向文件或目录
ENOTTY	路径没有指向 Lustre 文件系统
EFAULT	*lum* 指向的内存区域未正确映射

42.2.5. 示例

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <lustre/lustreapi.h>
5
6 static inline int maxint(int a, int b)
7 {
8     return a > b ? a : b;
9 }
10 static void *alloc_lum()
11 {
12     int v1, v3, join;
13     v1 = sizeof(struct lov_user_md_v1) +
14         LOV_MAX_STRIPE_COUNT * sizeof(struct lov_user_ost_data_v1);
15     v3 = sizeof(struct lov_user_md_v3) +
16         LOV_MAX_STRIPE_COUNT * sizeof(struct lov_user_ost_data_v1);
17     return malloc(maxint(v1, v3));
18 }
19 int main(int argc, char** argv)
20 {
21     struct lov_user_md *lum_file = NULL;
22     int rc;
23     int lum_size;
24     if (argc != 2) {
25         fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
```

```
        return 1;
    }
    lum_file = alloc_lum();
    if (lum_file == NULL) {
        rc = ENOMEM;
        goto cleanup;
    }
    rc = llapi_file_get_stripe(argv[1], lum_file);
    if (rc) {
        rc = errno;
        goto cleanup;
    }
    /* stripe_size stripe_count */
    printf("%d %d\n",
            lum_file->lmm_stripe_size,
            lum_file->lmm_stripe_count);
cleanup:
    if (lum_file != NULL)
        free(lum_file);
    return rc;
}
```

42.3. llapi_file_open

llapi_file_open 命令用于在 Lustre 文件系统上打开（或创建）文件或设备。

42.3.1. 梗概

```
#include <lustre/lustreapi.h>
int llapi_file_open(const char *name, int flags, int mode,
   unsigned long long stripe_size, int stripe_offset,
   int stripe_count, int stripe_pattern);
int llapi_file_create(const char *name, unsigned long long stripe_size,
   int stripe_offset, int stripe_count,
   int stripe_pattern);
```

42.3.2. 说明

llapi_file_create() 调用相当于在标志为O_CREAT|O_WRONLY，模式为0644下调用llapi_file_open后再关闭文件。llapi_file_open()在 Lustre 文件系统上打开给定名称的文件。

选项	说明
flags	可以是 O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC, O_APPEND, O_NONBLOCK, O_SYNC, FASYNC, O_DIRECT, O_LARGEFILE, O_DIRECTORY, O_NOFOLLOW, O_NOATIME的任意组合。
mode	指定使用O_CREAT时用于新文件的权限位。
stripe_size	指定条带大小（字节）。须为 64 KB 的倍数且不超过 4GB。
stripe_offset	指定文件的起始 OST 索引。默认值为 -1。
stripe_count	指定文件条带化的 OST 数量。默认值为 -1。
stripe_pattern	指定条带模式。在 Lustre 发行版中，仅 LOV_PATTERN_RAID0可用。默认值为 0。

42.3.3. 返回值

llapi_file_open() 和 llapi_file_create() 返回:
成功: >=0，llapi_file_open 的返回值为文件描述符;
失败: <0，其绝对值为错误代码。

42.3.4. 错误

错误	说明
EINVAL	stripe_size、stripe_offset、stripe_count 或 stripe_pattern 无效。
EEXIST	条带信息已被设置且不能更改；名称已存在。
EALREADY	条带信息已被设置且不能更改。
ENOTTY	name 没有指向 Lustre 文件系统。

42.3.5. 示例

```
#include <stdio.h>
#include <lustre/lustreapi.h>

int main(int argc, char *argv[])
{
    int rc;
    if (argc != 2)
        return -1;
    rc = llapi_file_create(argv[1], 1048576, 0, 2, LOV_PATTERN_RAID0);
    if (rc < 0) {
        fprintf(stderr, "file creation has failed, %s\n",
           strerror(-rc));
        return -1;
    }
    printf("%s with stripe size 1048576, striped across 2 OSTs,"
            " has been created!\n", argv[1]);
    return 0;
}
```

42.4. llapi_quotactl

使用 llapi_quotactl管理 Lustre 文件系统的磁盘配额。

42.4.1. 梗概

```
#include <lustre/lustreapi.h>
int llapi_quotactl(char" " *mnt," " struct if_quotactl" " *qctl)

struct if_quotactl {
        __u32                   qc_cmd;
        __u32                   qc_type;
        __u32                   qc_id;
        __u32                   qc_stat;
        struct obd_dqinfo       qc_dqinfo;
        struct obd_dqblk        qc_dqblk;
        char                    obd_type[16];
```

```
        struct obd_uuid         obd_uuid;
};
struct obd_dqblk {
        __u64 dqb_bhardlimit;
        __u64 dqb_bsoftlimit;
        __u64 dqb_curspace;
        __u64 dqb_ihardlimit;
        __u64 dqb_isoftlimit;
        __u64 dqb_curinodes;
        __u64 dqb_btime;
        __u64 dqb_itime;
        __u32 dqb_valid;
        __u32 padding;
};
struct obd_dqinfo {
        __u64 dqi_bgrace;
        __u64 dqi_igrace;
        __u32 dqi_flags;
        __u32 dqi_valid;
};
struct obd_uuid {
        char uuid[40];
};
```

42.4.2. 说明

llapi_quotactl() 命令用于操作挂载的 Lustre 文件系统上的磁盘配额。qc_cmd表示命令将应用于 UIDqc_id或 GIDqc_id。

选项	说明
LUSTRE_Q_GETQUOTA	获取用户或组 *qc_id* 的磁盘配额限制和当前使用情况。*qc_type* 有USRQUOTA或GRPQUOTA。*uuid* 可以通过OBD UUID字符串填充以查询特定节点的配额信息。*dqb_valid* 可设置为非零以查询来自 MDS 的信息。如果 *uuid* 是空字符串且 *dqb_valid* 为零，则返回集群范围上的限制和使用情况。返回时，*obd_dqblk* 将包含所请求的信息（块限制的单位为千字节）。在使用此命令之

选项	说明
	前必须打开配额功能。
LUSTRE_Q_SETQUOTA	为用户或组 *qc_id* 设置磁盘配额限制。*qc_type* 有USRQUOTA或GRPQUOTA。根据更新的限制，必须将 *dqb_valid* 设置为QIF_ILIMITS，QIF_BLIMITS或QIF_LIMITS（inode 限制和块限制）。必须使用限制值填充 *obd_dqblk*（如 *dqb_valid*，块限制单位为千字节）。在使用此命令之前必须打开配额功能。
LUSTRE_Q_GETINFO	获取有关配额的信息。*qc_type* 为USRQUOTA或GRPQUOTA。返回时，*dqi_igrace* 为 inode 宽限时间（以秒为单位），*dqi_bgrace* 是块宽限时间（以秒为单位），当前 Lustre 软件发行版不使用 *dqi_flags*。
LUSTRE_Q_SETINFO	获取有关配额的信息。*qc_type* 为USRQUOTA或GRPQUOTA。返回时，*dqi_igrace* 为 inode 宽限时间（以秒为单位），*dqi_bgrace* 是块宽限时间（以秒为单位）。当前 Lustre 软件发行版不使用 *dqi_flags*，并且必须将其归零。

42.4.3. 返回值

llapi_quotactl() 返回:
成功: 0 ;
失败: -1 ，同时将设置错误便好 (errno) 。

42.4.4. 错误

llapi_quotactl 的错误类型如下所示:

错误	说明
EFAULT	*qctl* 无效。
ENOSYS	内核或 Lustre 模块尚未使用QUOTA选项进行编译。
ENOMEM	没有足够的内存完成操作。
ENOTTY	*qc_cmd* 无效。

错误	说明
ENOENT	*uuid* 无法对应 OBD 或 *mnt* 不存在。
EPERM	调用享有特权，但调用者不适超级用户。
ESRCH	找不到指定用户的磁盘配额。尚未为此文件系统启用配额。

42.5. llapi_path2fid

使用 llapi_path2fid 从路径名获取 FID。

42.5.1. 梗概

```
1 #include <lustre/lustreapi.h>
2
3 int llapi_path2fid(const char *path, unsigned long long *seq, unsigned long
    *oid, unsigned long *ver)
```

42.5.2. 说明

llapi_path2fid 函数将返回路径名的 FID （序列：对象 ID：版本）。

42.5.3. 返回值

llapi_path2fid 返回:
成功：0；
失败：非零值。

42.6. llapi_ladvise

（在 **Lustre 2.9** 中引入）
使用 llapi_ladvise 为服务器提供有关 Lustre 文件的 IO 建议。

42.6.1. 梗概

```
1 #include <lustre/lustreapi.h>
2 int llapi_ladvise(int fd, unsigned long long flags,
3                   int num_advise, struct llapi_lu_ladvise *ladvise);
4
5 struct llapi_lu_ladvise {
```

```
6   __u16 lla_advice;       /* advice type */
7   __u16 lla_value1;       /* values for different advice types */
8   __u32 lla_value2;
9   __u64 lla_start;        /* first byte of extent for advice */
10  __u64 lla_end;          /* last byte of extent for advice */
11  __u32 lla_value3;
12  __u32 lla_value4;
13 };
```

42.6.2. 说明

`llapi_ladvise`函数将 ladvise 中的 *num_advise I/O* 的一组来自应用程序的针对文件描述符 *fd* 的提示建议（最多有 *LAH_COUNT_MAX* 个条目）传递到一个或多个 Lustre 服务器。*flags* 可以有选择性地按位或值来修改处理建议的方式：

- LF_ASYNC：客户端在提交 ladvise RPC 后立即返回用户空间，服务器线程将异步处理建议。

- LF_UNSET：取消或清除以前的建议（目前仅支持`LU_ADVISE_LOCKNOEXPAND`）。

每个 *ladvise* 元素都是一个包含以下字段的 *llapi_lu_ladvise* 结构：

字段	说明
`lla_ladvice`	有关给定文件范围的建议，当前支持：`LU_LADVISE_WILLREAD`：使用服务器的最优 I/O 大小将数据预取到服务器缓存中；`LU_LADVISE_DONTNEED`：清除服务器上指定文件范围的缓存数据。
`lla_start`	此建议开始的偏移量（以字节为单位）。
`lla_end`	此建议结束的偏移量（不包含）。
`lla_value1`、`lla_value2`、`lla_value3`、`lla_value4`	用于未来建议类型的附加参数，例如，供以下这些字段以使用特定建议。如果对给定的建议类型没有明确要求，则应设置为零。
`lla_lockahead_mode`	使用`LU_ADVISE_LOCKAHEAD`时，`'lla_value1'`字段可用于传达请求的锁模式，可使用`lla_lockahead_mode`引用。

字段	说明
lla_peradvice_flags	当使用支持它们的建议时，'lla_value2'字段用于传递特定于每个建议的标志，可使用lla_peradvice_flags引用。支持LF_ASYNC和LF_UNSET。
lla_lockahead_result	使用LU_ADVISE_LOCKAHEAD时，'lla_value3' 字段可用于传达请求结果，可使用lla_lockahead_result引用。

llapi_ladvise()将建议转发给 Lustre 服务器，但不保证服务器如何以及何时对建议作出反应。服务器收到建议可能会也可能不会触发操作，具体取决于建议的类型以及接收建议的服务器端组件的实时决策。

llapi_ladvise()的典型用法是使应用程序和用户（通过lfs ladvise）获取关于应用程序 I/O 模式的扩展知识，以干预服务器端 I/O 处理。例如，如果一组不同的客户端正在对文件进行小的随机读取，则在处理随机 IO 之前将页面预取到具有大线性读取能力的 OSS 缓存中有百利而无一害。由于要向客户端发送更多数据，使用fadvise()将数据提取到每个客户端的缓存中可能没有什么好处。

值得一提的是LU_LADVISE_LOCKAHEAD的使用。尽管您可以（我们也推荐您）在应用程序中直接使用它以避免锁争用（主要是从多个客户端写入单个文件），但您同样也可通过 ANL 的 MPI-I/O / MPICH 库的 i/o 聚合模式来获取它。这也是使用此功能的主要方式。

目前，此功能仅作为补丁，尚未合并到公共 ANL 代码库中。建议用户查看 MPICH 文档或从其供应商处获取更多支持。

虽然在概念上类似于 *posix_fadvise* 和 *Linux fadvise* 系统调用，llapi_ladvise()与它们的主要区别在于：fadvise() 和posix_fadvise()是客户端机制，不会将建议传递给文件系统；而llapi_ladvise()会发送建议或提示至存储文该件的一个或多个 Lustre 服务器。在某些情况下，可能需要使用两种接口。

42.6.3. 返回值

llapi_ladvise 返回：

成功：0；

失败：-1，同时将设置errno。

42.6.4. 错误

错误	说明
ENOMEM	没有足够的内存完成操作。
EINVAL	一个以上的无效参数。
EFAULT	ladvise 指向的内存区域未正确映射。
ENOTSUPP	不支持的 advice 类型。

42.7. llapi 库使用示例

使用llapi_file_create为新文件设置 Lustre 软件属性。

您可以使用ioctl等内部程序设置条带化。编译以下示例程序，您需要安装 Lustre 客户端源 RPM。

用于演示 API 条带化的简单 c 程序 - libtest.c

```
1 /* -*- mode: c; c-basic-offset: 8; indent-tabs-mode: nil; -*-
2  * vim:expandtab:shiftwidth=8:tabstop=8:
3  *
4  * lustredemo - a simple example of lustreapi functions
5  */
6 #include <stdio.h>
7 #include <fcntl.h>
8 #include <dirent.h>
9 #include <errno.h>
10 #include <stdlib.h>
11 #include <lustre/lustreapi.h>
12 #define MAX_OSTS 1024
13 #define LOV_EA_SIZE(lum, num) (sizeof(*lum) + num *
       sizeof(*lum->lmm_objects))
14 #define LOV_EA_MAX(lum) LOV_EA_SIZE(lum, MAX_OSTS)
15
16 /*
17  * This program provides crude examples of using the lustreapi API functions
18  */
19 /* Change these definitions to suit */
20
21 #define TESTDIR "/tmp"          /* Results directory */
```

```
22 #define TESTFILE "lustre_dummy"  /* Name for the file we create/destroy */
23 #define FILESIZE 262144                    /* Size of the file in words */
24 #define DUMWORD "DEADBEEF"       /* Dummy word used to fill files */
25 #define MY_STRIPE_WIDTH 2                  /* Set this to the number of OST
       required */
26 #define MY_LUSTRE_DIR "/mnt/lustre/ftest"
27
28 int close_file(int fd)
29 {
30         if (close(fd) < 0) {
31                 fprintf(stderr, "File close failed: %d (%s)\n", errno,
                       strerror(errno));
32                 return -1;
33         }
34         return 0;
35 }
36
37 int write_file(int fd)
38 {
39         char *stng =  DUMWORD;
40         int cnt = 0;
41
42         for( cnt = 0; cnt < FILESIZE; cnt++) {
43                 write(fd, stng, sizeof(stng));
44         }
45         return 0;
46 }
47 /* Open a file, set a specific stripe count, size and starting OST
48  *    Adjust the parameters to suit */
49 int open_stripe_file()
50 {
51         char *tfile = TESTFILE;
52         int stripe_size = 65536;    /* System default is 4M */
53         int stripe_offset = -1;     /* Start at default */
54         int stripe_count = MY_STRIPE_WIDTH;  /*Single stripe for this demo*/
55         int stripe_pattern = 0;     /* only RAID 0 at this time */
```

```
56          int rc, fd;
57
58          rc = llapi_file_create(tfile,
59                                 stripe_size,stripe_offset,stripe_count,stripe_pattern);
60          /* result code is inverted, we may return -EINVAL or an ioctl error.
61           * We borrow an error message from sanity.c
62           */
63          if (rc) {
64                  fprintf(stderr,"llapi_file_create failed: %d (%s) \n", rc,
                        strerror(-rc));
65                  return -1;
66          }
67          /* llapi_file_create closes the file descriptor, we must re-open */
68          fd = open(tfile, O_CREAT | O_RDWR | O_LOV_DELAY_CREATE, 0644);
69          if (fd < 0) {
70                  fprintf(stderr, "Can't open %s file: %d (%s)\n", tfile,
                        errno, strerror(errno));
71                  return -1;
72          }
73          return fd;
74  }
75
76  /* output a list of uuids for this file */
77  int get_my_uuids(int fd)
78  {
79          struct obd_uuid uuids[1024], *uuidp;         /* Output var */
80          int obdcount = 1024;
81          int rc,i;
82
83          rc = llapi_lov_get_uuids(fd, uuids, &obdcount);
84          if (rc != 0) {
85                  fprintf(stderr, "get uuids failed: %d (%s)\n",errno,
                        strerror(errno));
86          }
87          printf("This file system has %d obds\n", obdcount);
88          for (i = 0, uuidp = uuids; i < obdcount; i++, uuidp++) {
```

```
89                 printf("UUID %d is %s\n",i, uuidp->uuid);
90         }
91         return 0;
92 }
93
94 /* Print out some LOV attributes. List our objects */
95 int get_file_info(char *path)
96 {
97
98         struct lov_user_md *lump;
99         int rc;
100        int i;
101
102        lump = malloc(LOV_EA_MAX(lump));
103        if (lump == NULL) {
104                return -1;
105        }
106
107        rc = llapi_file_get_stripe(path, lump);
108
109        if (rc != 0) {
110                fprintf(stderr, "get_stripe failed: %d (%s)\n",errno,
                    strerror(errno));
111                return -1;
112        }
113
114        printf("Lov magic %u\n", lump->lmm_magic);
115        printf("Lov pattern %u\n", lump->lmm_pattern);
116        printf("Lov object id %llu\n", lump->lmm_object_id);
117        printf("Lov stripe size %u\n", lump->lmm_stripe_size);
118        printf("Lov stripe count %hu\n", lump->lmm_stripe_count);
119        printf("Lov stripe offset %u\n", lump->lmm_stripe_offset);
120        for (i = 0; i < lump->lmm_stripe_count; i++) {
121                printf("Object index %d Objid %llu\n",
                    lump->lmm_objects[i].l_ost_idx,
                    lump->lmm_objects[i].l_object_id);
```

```
122         }
123
124         free(lump);
125         return rc;
126
127 }
128
129 /* Ping all OSTs that belong to this filesystem */
130 int ping_osts()
131 {
132         DIR *dir;
133         struct dirent *d;
134         char osc_dir[100];
135         int rc;
136
137         sprintf(osc_dir, "/proc/fs/lustre/osc");
138         dir = opendir(osc_dir);
139         if (dir == NULL) {
140                 printf("Can't open dir\n");
141                 return -1;
142         }
143         while((d = readdir(dir)) != NULL) {
144                 if ( d->d_type == DT_DIR ) {
145                         if (! strncmp(d->d_name, "OSC", 3)) {
146                                 printf("Pinging OSC %s ", d->d_name);
147                                 rc = llapi_ping("osc", d->d_name);
148                                 if (rc) {
149                                         printf("  bad\n");
150                                 } else {
151                                         printf("  good\n");
152                                 }
153                         }
154                 }
155         }
156         return 0;
157
```

```
158 }
159
160 int main()
161 {
162         int file;
163         int rc;
164         char filename[100];
165         char sys_cmd[100];
166
167         sprintf(filename, "%s/%s",MY_LUSTRE_DIR, TESTFILE);
168
169         printf("Open a file with striping\n");
170         file = open_stripe_file();
171         if ( file < 0 ) {
172                 printf("Exiting\n");
173                 exit(1);
174         }
175         printf("Getting uuid list\n");
176         rc = get_my_uuids(file);
177         printf("Write to the file\n");
178         rc = write_file(file);
179         rc = close_file(file);
180         printf("Listing LOV data\n");
181         rc = get_file_info(filename);
182         printf("Ping our OSTs\n");
183         rc = ping_osts();
184
185         /* the results should match lfs getstripe */
186         printf("Confirming our results with lfs getstripe\n");
187         sprintf(sys_cmd, "/usr/bin/lfs getstripe %s/%s", MY_LUSTRE_DIR,
                TESTFILE);
188         system(sys_cmd);
189
190         printf("All done\n");
191         exit(rc);
192 }
```

上述程序的 Makefile 文件：

```
gcc -g -O2 -Wall -o lustredemo libtest.c -llustreapi
clean:
rm -f core lustredemo *.o
run:
make
rm -f /mnt/lustre/ftest/lustredemo
rm -f /mnt/lustre/ftest/lustre_dummy
cp lustredemo /mnt/lustre/ftest/
```

第四十三章配置文件和模块参数

43.1. 简介

网络硬件和路由通过模块参数进行配置，这些参数应在/etc/modprobe.d/lustre.conf文件中进行指定，如：

```
options lnet networks=tcp0(eth2)
```

以上选项指定此节点应在 eth2 网络接口上使用 TCP 协议。

首次加载模块时会读取模块参数。当 LNet 启动时（通常在modprobe ptlrpc上），LNet 模块自动加载特定类型的 LND 模块（如ksocklnd）。

LNet 配置参数可在 /sys/module/lnet/parameters/下查看，LND 特定参数可在相应 LND 名下查看，例如用于 socklnd（TCP）LND 的 /sys/module/ksocklnd/parameters/ 。

对于以下参数，默认选项设置显示在括号中。标记为W的参数更改会影响正在运行的系统，标记为Wc的参数更改仅在建立连接时有效（现有连接不受这些更改的影响），而无标记的参数只能在 LNet 第一次加载时设置.

43.2. 模块选项

- 在路由或其他多网络配置中，请使用ip2nets而不是网络，从而使所有节点都可以使用相同的配置。

- 在路由网络中，请在任何位置使用相同的" 路由" 配置。指定为路由器的节点将自动启用转发，并忽略与特定节点无关的路由。请保持使用通用配置以保证所有节点有一致的路由表。

- 单独的`lustre.conf`文件使配置分发更加容易。

- 如果设置了`config_on_load=1`，LNet 将在`modprobe`期间启动，而不会等待 Lustre 文件系统启动。这确保了路由器在模块加载时开始工作。

```
# lctl
# lctl> net down
```

- 请记得使用`lctl ping {nid}`命令来快速确认您的 LNet 配置是否正确。

43.2.1. LNet 选项

此小节将介绍 LNet 选项。

43.2.1.1. 网络拓扑 网络拓扑模块参数用于确定每个节点应加入哪些网络，是否应该在这些网络之间添加路由，以及它非本地网络之间如何通信。

以下是各种网络和其支持的软件堆栈的列表：

网络	软件堆栈
o2ib	OFED Version 2

注意

Lustre 软件会忽略环回接口（`lo0`），但可使用别名为环回的任何 IP 地址（默认情况下）。如有疑问，请明确指定网络。

`ip2nets`（""）是一个列出全局可用的网络的字符串，每个网络都有一组 IP 地址范围。LNet 通过将 IP 地址范围与节点的本地 IP 进行匹配来确定此列表中的本地可用网络。此选项的目的是为了能够在不同网络上的各种节点上使用相同的`modules.conf`文件。该字符串的语法如下：

```
<ip2nets> :== <net-match> [ <comment> ] { <net-sep> <net-match> }
<net-match> :== [ <w> ] <net-spec> <w> <ip-range> { <w> <ip-range> }
[ <w> ]
<net-spec> :== <network> [ "(" <interface-list> ")" ]
<network> :== <nettype> [ <number> ]
<nettype> :== "tcp" | "elan" | "o2ib" | ...
<iface-list> :== <interface> [ "," <iface-list> ]
<ip-range> :== <r-expr> "." <r-expr> "." <r-expr> "." <r-expr>
```

```
<r-expr> :== <number> | "*" | "[" <r-list> "]"
<r-list> :== <range> [ "," <r-list> ]
<range> :== <number> [ "-" <number> [ "/" <number> ] ]
<comment :== "#" { <non-net-sep-chars> }
<net-sep> :== ";" | "\n"
<w> :== <whitespace-chars> { <whitespace-chars> }
```

<net-spec> 包含了足够的信息来标识唯一的网络并加载适当的 LND。LND 根据它可以使用的接口来确定 NID 中缺少的" 网络内地址" 部分。

<iface-list>用于指定网络可以使用的硬件接口。如果此选项被省略，则表示可使用所有接口。不支持<iface-list>语法的 LND 不能为其配置使用的特定接口，则将使用任何可用的接口。此时，一个节点上在任何时候都只能有这些 LND 的单个实例，并且必须省略<iface-list>。

<net-match>条目将按声明的顺序进行扫描，逐个查看是否有节点的 IP 地址与<ip-range>表达式之一匹配。如果匹配，<net-spec>将指定要实例化的网络。请注意，我们只采用指定网络匹配的第一个表达式。因此，为简化匹配表达式，我们在特殊条件之后放置通用条件，例如：

```
ip2nets="tcp(eth1,eth2) 134.32.1.[4-10/2]; tcp(eth1) *.*.*.*"
```

网络 134.32.1.* 上的四个节点（134.32.1.{4,6,8,10}）有两个接口，其他节点只有一个接口。

```
ip2nets="o2ib 192.168.0.*; tcp(eth2) 192.168.0.[1,7,4,12]"
```

上述语句描述了 192.168.0.* 上的 IB 集群。其中四个节点有 IP 接口，可被用作路由器。

请注意，match-all 表达式（如*.*.*.*）有效地覆盖了随后指定的所有其他<net-match>条目，应谨慎使用。

以下是一个更复杂的情况，有如下路由参数：

- 两个 TCP 子网
- 一个 Elan 子网
- 设置为路由器的机器，且有 TCP 和 Elan 接口
- Elan 上配置有 IP，但只用于标记节点。

```
options lnet ip2nets=€âtcp 198.129.135.* 192.128.88.98; \
        elan 198.128.88.98 198.129.135.3; \
        routes='cp 1022@elan # Elan NID of router; \
        elan  198.128.88.98@tcp # TCP NID of router  '
```

43.2.1.2. ip2nets(``tcp'') ip2nets是一个字符串，列出了全球可用的网络，每个网络都有一组 IP 地址范围。LNet 通过将 IP 地址范围与节点的本地 IP 相匹配，从这个列表中确定本地可用的网络。目的是允许同一个 modules.conf 文件在不同网络的不同节点上使用。该字符串的语法如下:

```
<ip2nets> :== <net-match> [ <comment> ] { <net-sep> <net-match> }
<net-match> :== [ <w> ] <net-spec> <w> <ip-range> { <w> <ip-range> }
[ <w> ]
<net-spec> :== <network> [ "(" <interface-list> ")" ]
<network> :== <nettype> [ <number> ]
<nettype> :== "tcp" | "elan" | "o2ib" | ...
<iface-list> :== <interface> [ "," <iface-list> ]
<ip-range> :== <r-expr> "." <r-expr> "." <r-expr> "." <r-expr>
<r-expr> :== <number> | "*" | "[" <r-list> "]"
<r-list> :== <range> [ "," <r-list> ]
<range> :== <number> [ "-" <number> [ "/" <number> ] ]
<comment :== "#" { <non-net-sep-chars> }
<net-sep> :== ";" | "\n"
<w> :== <whitespace-chars> { <whitespace-chars> }
```

<net-spec>包含足够的信息来唯一地识别网络并加载一个适当的 LND。LND 根据可以使用的接口来确定 NID 中缺少的 ``网络内地址'' 部分。

<iface-list>用于指定网络可以使用哪个硬件接口。如果省略，则使用所有接口。不支持<iface-list>语法的 LND 不能配置为使用特定的接口，只是使用现有的接口。在任何时候，一个节点上只能存在这些 LND 的一个实例，且<iface-list>必须省略。

<net-match>条目按照声明的顺序进行扫描，看节点的 IP 地址是否与<ip-range>表达式之一相匹配。如果有匹配的，则<net-spec>指定要实例化的网络。请注意，特定网络应与第一个匹配才算数。可以通过把通配表达式放在特殊情况的后面，来简化匹配表达式，如:

```
ip2nets="tcp(eth1,eth2) 134.32.1. [4-10/2]; tcp(eth1) *.*.*.*"
```

网络 134.32.1.* 上的 4 个节点包含 2 个接口（134.32.1.{4,6,8,10}），其他的都只有 1 个接口。

```
ip2nets="o2ib 192.168.0.*; tcp(eth2) 192.168.0. [1,7,4,12]"
```

上述表达式表示 192.168.0.* 上的一个 IB 集群。其中四个节点也包含 IP 接口；这四个节点可以作为路由器使用。

请注意，通配表达式（例如，*.*.*.*）有效地屏蔽了所有其他<net-match>条目，因此应谨慎使用。

下面是一个更复杂的情况，路由参数如下: - 两个 TCP 子网 - 一个 Elan 子网 - 一台设置为路由器的机器，同时有 TCP 和 Elan 接口 - 配置了 IP over Elan，但只有 IP 会被用来标记节点。

```
options lnet ip2nets=€âtcp 198.129.135.* 192.128.88.98; \
 elan 198.128.88.98 198.129.135.3; \
 routes='cp 1022@elan # Elan NID of router; \
 elan 198.128.88.98@tcp # TCP NID of router '
```

43.2.1.3. networks("tcp") 用于替代ip2nets，可用于指定要显式实例化的网络。语法为逗号分隔的<net-spec>列表（见上文）。仅当未指定ip2nets和networks时才使用默认值。

43.2.1.4. routes ("") 这是一个列出转发的路由器网络和 NID 的字符串。

语法如下（<w>是一个或多个空白字符）:

```
<routes> :== <route>{ ; <route> }
<route> :==
    [<net>[<w><hopcount>]<w><nid>[:<priority>]{<w><nid>[:<priority>]}
```

请注意，**Lustre 2.5** 中添加了优先级参数。

tcp1 上的节点必须经过路由器到达 Elan 网络:

```
options lnet networks=tcp1 routes="elan 1 192.168.2.2@tcpA"
```

跳数和优先级用于帮助在多路由配置之间选择最佳路径。

以下提供了一种用于描述目标网络和路由器 NID 的简单但功能强大的扩展语法:

```
<expansion> :== "[" <entry> { "," <entry> } "]"
<entry> :== <numeric range> | <non-numeric item>
<numeric range> :== <number> [ "-" <number> [ "/" <number> ] ]
```

扩展部分是用方括号括起来的列表，列表中的数字项可以是单个数字、连续的数字范围或跨步数字范围。例如，routes="elan 192.168.1.[22-24]@tcp" 表示网络elan0为相邻节点（hopcount默认为 1），且可以通过tcp0网络上的 3 个路由器（192.168.1.22@tcp，192.168.1.23@tcp和192.168.1.24@tcp）进行访问。

routes="[tcp,o2ib] 2 [8-14/2]@elan"表示网络tcp0和o2ib0可通过 4 个路由器（8@elan，10@ elan，12@elan和14elan）进行访问。跳数为 2 意味着这

两个网络的流量将经过 2 个路由器，首先是此条目中指定的第一个路由器，然后是另一个。

重复条目、到本地网络的路由条目以及非本地网络上的路由器的条目将被忽略。

在 **Lustre 2.5** 之前，通过选择更短跳数的路由器来解决等效条目之间的冲突。跳数省略时默认为 1（远程网络相邻）。

至 **Lustre 2.5** 起，如果优先级相等，则将选择 priority 号更低或跳数更少的路由条目。优先级省略时默认为 0。跳数省略时默认为 1（远程网络相邻）。

使用不同本地网络上的路由器来指点同一目标的路由是错误的。

如果目标网络字符串不包含扩展部分，则跳数默认为 1，可以省略（即远程网络是相邻的）。事实上，大多数多网络配置都是如此。为给定目标网络指定不一致的跳数是错误的，这也是为什么当目标网络字符串指定来多个网络时需要指定显式跳数。

43.2.1.5. `forwarding ("")` 该字符串可设置为" 启用" 或" 禁用"，用于明确控制此节点是否应充当路由器的角色，从而在所有本地网络之间转发消息进行通信。使用适当的网络拓扑选项启动 LNet（`modprobe ptlrpc`）可启动独立路由器。

一个独立的路由器可以通过简单地启动 LNet（`modprobe ptlrpc'）以及适当的网络拓扑结构选项来启动。

43.2.1.6. `accept(secure)` `acceptor`是一些 LND 用于建立通信的 TCP/IP 服务。如果本地网络需要它并且它尚未禁用，则`acceptor`可用于在单个端口上监听并将连接请求重定向到适当的本地网络。`acceptor`是 LNet 模块的一部分，可通过以下选项进行配置：

变量	说明
`acceptor`	接受器将允许来自远程节点的连接类型。
`secure`	`secure`－仅接受来自预留 TCP 端口（1023 以下的端口号）的这是默认的，防止用户空间的进程试图连接到服务器。 `all` －接受来自任何 TCP 端口的连接。这可能需要允许非特权端口上的连接，例如来自在用户空间中运行的虚拟机的客户端。 `none`－不运行`acceptor`。如果 TCP 连接丢失，而服务器因某种原因需要联系客户端（例如 LDLM 锁回调或查看大小），
`accept_port(988)`	`acceptor`监听连接请求的端口号。站点配置中需要`acceptor`的所有节点必须使用相同的端口。

变量	说明
accept_backlog(127)	挂起连接队列可能的最大长度。
accept_timeout(5, W)	与对等节点通信时允许acceptor阻塞的最长时间（以秒为单位）。
accept_proto_version	传出连接请求应使用的acceptor协议的版本。默认为最新的acceptor协议版本，也可设置为更早的版本以允许节点启动与仅适用该版本acceptor协议的节点的连接。acceptor在某些条件的限制下可以处理任一版本的协议（也就是说，它可以接受来自" 旧" 或" 新" 的对等节点连接）。对于当前版本的acceptor协议（v1），如果只有单个本地网络需要，acceptor与" 旧" 的对等节点兼容。

43.2.1.7. rnet_htable_size rnet_htable_size表示内部 LNet 哈希表配置处理的远程网络数，为整数值。rnet_htable_size用于优化哈希表的大小，并不限制您可以拥有的远程网络的数量。未指定此参数时，默认哈希表大小为 128。（在 **Lustre 2.3** 中引入）

43.2.2. SOCKLND 内核 TCP/IP LND

SOCKLND 内核 TCP/IP LND (socklnd) 是基于连接的，使用 acceptor 通过套接字与其对等节点建立通信。

它支持多个实例，在多个接口间使用动态负载平衡。如果ip2nets或网络模块参数未指定接口，则使用所有非环回 IP 接口。网络内的地址由socklnd遇到的第一个 IP 接口的地址决定。

如果有一个 InfiniBand 网络" 边缘" 上的节点，配置有低带宽管理以太网（eth0）、IB 上的 IP（ipoib0），以及提供集群外连接的一对 GigE NIC（eth1，eth2）。则此节点应配置'networks=o2ib,tcp(eth1,eth2)' 以确保socklnd忽略管理以太网和 IPoIB。

变量	说明
timeout (50,W)	在返回 LND 失败前通信可能会等待的时间（以秒为单位）。

变量	说明
nconnds (4)	设置连接的守护程序数量。
min_reconnectms (1000,W)	最小连接重试间隔（以毫秒为单位）。连接尝试失败后进行第一次重试之前必须等待的时间。当连接尝试失败时，此间隔时间将在每次连续重试时加倍，直到达到max_reconnectms。
max_reconnectms (6000,W)	最大连接重试间隔（以毫秒为单位）。
eager_ack (0 on linux, 1 on darwin,W)	布尔值，用于确定socklnd是否尝试刷新消息边界上的发送。
typed_conns (1,Wc)	布尔值，用于确定socklnd是否应该为不同类型的消息使用不同的套接字。清除时，与特定对等节点的所有通信都在同一个套接字上进行。否则，单独的套接字用于批量发送，批量接收和其他所有内容。
min_bulk (1024,W)	确定何时将信息视为" 批量" 数据。
tx_buffer_size, rx_buffer_size (8388608,Wc)	套接字缓冲区大小。将此选项设置为零（0），表示允许系统自动调整缓冲区大小。注意：请谨慎更改此值，不恰当的大小会损害性能。
nagle (0,Wc)	布尔值，用于确定 nagle 是否启用。不能在生产系统设置此值。
keepalive_idle (30,Wc)	在发送 keepalive probe 前套接字保持空闲的时间（以秒为单位）。将此值设置为零（0）表示禁用 keepalive。
keepalive_intvl (2,Wc)	重复未应答的 keepalive probe 的时间（以秒为单位）。将此值设置为零（0）表示禁用 keepalive。
keepalive_count (10,Wc)	在发出套接字死亡（对等节点也随之死亡）之前未应答的 keepalive probe 的数量。
enable_irq_affinity (0,Wc)	布尔值，用于确定是否启用 IRQ affinity。默认值

变量	说明
	为零（0）。设置时，socklnd 将尝试通过处理特定 CPU 上特定（硬件）接口的设备中断和数据移动来最大化性能。并非所有平台都提供此选项。此选项需要具备 SMP 系统，并使用多个 NIC 以获取最佳性能。具有多个 CPU 和单个 NIC 的系统可能会禁用此选项来提高性能。
`zc_min_frag (2048,W)`	确定 zero-copy 发送应的最小消息片段。如果将其设置为 PAGE_SIZE 还大的数，则将禁用所有 zero-copy。并非所有平台都提供此选项。

第四十四章系统配置工具

44.1. e2scan

e2scan 实用程序是 ext2 文件系统更改的 inode 扫描程序。e2scan 程序使用 libext2fs 查找 ctime 或 mtime 比给定时间更新的 inode 并打印出它们的路径名。使用 e2scan 可以快速地生成已更改的文件列表。e2scan 工具包含在 e2fsprogs 包中，可在中找到。

44.1.1. 梗概

```
1 e2scan [options] [-f file] block_device
```

44.1.2. 说明

被调用时，e2scan 实用程序会遍历块设备上的所有 inode，查找已更改的 inode，并打印其 inode 编号。另一个类似的迭代器将使用`libext2fs(5)`创建一个表（称为父数据库）列出了每个 inode 的父节点。使用查找功能，您可以从根用户重建更改的路径名。

44.1.3. 选项

选项	说明
`-b inode buffer blocks`	设置 readahead inode 块以获取扫描块设备时的更优性能。
`-o output file`	如果指定了输出文件，则将更改的路径名写入此文件。

选项	说明
	否则，更改参数将写入 stdout。
-t inode| pathname	如果为 inode，则将 e2scan 类型设置为 inode。 e2scan 实用程序将更改的 inode 编号打印到 stdout。 默认类型设置为路径名。e2scan 实用程序根据更改的 inode 编号列出更改的路径名。
-u	从头开始重建父数据库。否则使用当前父数据库。

44.2. l_getidentity

l_getidentity 实用程序负责处理 Lustre 用户/组缓存回调。

44.2.1. 梗概

```
l_getidentity { $FSNAME-MDT{xxxx}| -d} {uid}
```

44.2.2. 说明

在 MDS 中调用l_getidentity实用程序中，将 UID 数值映射到该 UID 的补充组值列表中，并将其写入mdt.*.identity_info参数文件中。补充组的列表被缓存在内核中，以避免重复回调。

l_getidentity工具也可以直接运行以调试，通过使用 -d参数代替 MDT 名称，确保特定用户的 UID 映射配置正确。

44.2.3. 选项

选项	说明
${FSNAME}-MDT{xxxx}	元数据服务器目标名称
uid	用户标识符

44.2.4. 文件

l_getidentity 文件位于：

```
/proc/fs/lustre/mdt/${FSNAME}-MDT{xxxx}/identity_upcall
```

44.3. lctl

lctl 实用程序用于根用户的控制和配置。使用 lctl，您可以通过 ioctl 接口直接控制 Lustre，从而进行各种配置、维护和调试。

44.3.1. 梗概

```
lctl [--device devno] command [args]
```

44.3.2. 说明

可以通过发出 lctl 命令在交互模式下调用 lctl 实用程序。最常见的 lctl 命令有：

```
dl
dk
device
network up|down
list_nids
ping nidhelp
quit
```

获取可用命令的完整列表，请在`lctl`提示符下键入`help`。获得有关命令的含义和语法，请键入`help command`。使用 TAB 键可补全命令（取决于编译选项），使用上下箭头键可查询命令的历史记录。

对于非交互式使用，请使用二次调用，即在连接到设备后运行该命令。

44.3.3. 使用 lctl 设置参数

由于平台的不同，使用 procfs 接口并不总是可以成功访问 Lustre 参数。`lctl {get,set} param`作为独立于平台接口的解决方案，已被 Lustre 引入为可调参数，从而避免直接引用`/proc/{fs,sys}/{lustre,lnet}`。考虑到将来使用的可移植性，请使用 `lctl {get,set} param`。

文件系统运行时，在受影响的节点上使用`lctl set_param`命令设置临时参数（映射到`/proc/{fs,sys}/{lnet,lustre}`中的项目）。`lctl set_param`命令使用以下语法：

```
lctl set_param [-n] [-P] [-d] obdtype.obdname.property=value
```

如：

```
mds# lctl set_param mdt.testfs-MDT0000.identity_upcall=NONE
```

（在 **Lustre 2.5** 中引入）

使用 -P 选项设置永久参数，使用 -d选项删除永久参数。例如：mgs# lctl set_param -P mdt.testfs-MDT0000.identity_upcall=NONE mgs# lctl set_param -P -d mdt.testfs-MDT0000.identity_upcall

很多参数也可通过 lctl conf_param进行永久设置。lctl conf_param 通常可用于指定任何在文件/proc/fs/lustre可设置的 OBD 设备参数。lctl conf_param 命令必须在 MGS 节点上运行，并使用以下语法：

```
1 obd|fsname.obdtype.property=value)
```

如：

```
1 mgs# lctl conf_param testfs-MDT0000.mdt.identity_upcall=NONE
2 $ lctl conf_param testfs.llite.max_read_ahead_mb=16
```

注意

lctl conf_param 命令可在文件系统配置中为指定类型的所有节点设置永久参数。

要获取当前 Lustre 参数设置，请在相应节点上使用lctl get_param命令，其参数名称与lctl set_param中使用的相同：

```
1 lctl get_param [-n] obdtype.obdname.parameter
```

如：

```
1 mds# lctl get_param mdt.testfs-MDT0000.identity_upcall
```

使用 lctl list_param 命令列出所有可设置的 Lustre 参数：

```
1 lctl list_param [-R] [-F] obdtype.obdname.*
```

例如，列出 MDT 上的所有参数：

```
1 oss# lctl list_param -RF mdt
```

网络配置

选项	说明
network up|down|tcp|elan	启动或关闭 LNet；为其他lctl LNet 命令选择网络类型。
list_nids	打印本地节点上的所有 NID。必须运行 LNet。
which_nid nidlist	从远程节点的 NID 列表中，标识出将发生接口通信的 NID。
ping nid	通过 LNet ping 检查 LNet 连接，将使用适合指定 NID 的结构。
interface_list	打印给定网络类型的网络接口信息。
peer_list	打印给定网络类型的对端节点信息。

选项	说明
conn_list	打印给定网络类型的所有已连接的远端 NID。
active_tx	打印活动传输，仅适用于 Elan 网络。
route_list	打印完整的路由表。

设备选择

选项	说明
device devname	选择指定的 OBD 设备。所有其他命令以此命令所设置的设备为基础。
device_list	显示本地 Lustre OBD，a/k/a dl.

设备操作

选项	说明
list_param [-F|-R] parameter [parameter ...]	列出 Lustre 或 LNet 参数名。
-F	分别为目录，符号链接和可写文件添加'/'，'@'或'='。
-R	递归列出指定路径下的所有参数。如果未指定 param_path，则显示所有参数。
get_param [-n|-N|-F] parameter [parameter ...]	从指定路径获取 Lustre 或 LNet 参数值。
-n	仅打印参数值而不打印参数名称。
-N	仅打印匹配的参数名称而不打印值；在使用模式时特别有用。
-F	指定了-N 时，分别为目录，符号链接和可写文件添加'/'，'@'或'='。

选项	说明
set_param [-n] parameter=value	设置指定路径中 Lustre 或 LNet 参数的值。
-n	打印值时禁用 key 名称。
conf_param [-d]device fsname parameter=value	通过 MGS 为设备设置永久配置参数。此命令必须在 MGS 节点上运行。`lctl list_param`下的所有可写参数（如`lctl list_param -F osc.*.*| grep`）可使用`lctl conf_param`进行永久设置，但格式略有不同。`conf_param`需要先指定设备后指定 obdtype，且不支持通配符。此外，可以添加（或删除）故障转移节点，也可以设置一些系统范围的参数 (sys.at_max, sys.at_min, sys.at_extra, sys.at_early_margin, sys.at_history, sys.timeout, sys.ldlm_timeout)。对于系统范围的参数，*device* 将被忽略。
-d device|fsname.parameter	删除参数设置（下次重启时使用默认值）。将值设置为空也会删除参数设置。
activate	在停用操作后重新激活导入。此设置仅在重新启动后有效（见 `conf_param`）。
deactivate	停用导入，特别是不要将新文件条带分配给 OSC。在 MDS 上运行`lctl deactivate`会在 OST 上阻止其分配新对象。在 Lustre 客户端上运行`lctl deactivate`会导致它们在访问 OST 上对象时返回 -EIO 而不是持续等待恢复。
abort_recovery	在重新启动 MDT 或 OST 时中止恢复过程。

注意

使用 procfs 接口并不总是可以访问 Lustre 可调参数，这取决于平台。而 `lctl`

{get,set,list}_param可作为独立于平台的解决方案，从而避免直接引用/proc/{fs,sys}/{lustre,lnet}。考虑到未来使用过程中的可移植性，请使用lctl {get,set,list}_param 。

虚拟块设备操作

Lustre 可以在常规文件上模拟虚拟块设备。当您尝试通过文件设置空间交换时，需要使用此功能。

选项	说明
blockdev_attach filename/dev/lloop_device	将常规 Lustre 文件添加到块设备。如果设备节点不存在，则使用lctl创建它。由于模拟器使用的是动态主编号，我们建议您使用lctls创建设备节点。
blockdev_detach /dev/lloop_device	删除虚拟块设备。
blockdev_info /dev/lloop_device	提供有关附加到设备节点的 Lustre 文件的信息。

Changelogs

选项	说明
changelog_register	为特定设备注册新的 changelog 用户。每个文件系统操作发生时，相应 changelog 条目将永久保存在 MDT 上，仅在超出所有注册用户的最小设置点时进行清除（请参阅lfs changelog_clear)。如果 changelog 用户注册了却从不使用这些记录，则可能导致 changelog 占用大量空间，最终填满 MDT。
changelog_deregister *id*	注销现有的 changelog 用户。如果用户的" 清除" 记录号是该设备的最小值，则 changelog 记录将被清除，直到出现下一个设备最小值。

调试

选项	说明
debug_daemon	启动和停止调试守护程序，并控制输出文件名和大小。
debug_kernel [file] [raw]	将内核调试缓冲区转储到 stdout 或文件中。
debug_file input_ file [output_file]	将内核转储的调试日志从二进制转换为纯文本格式。
clear	清除内核调试缓冲区。
mark text	在内核调试缓冲区中插入标记文本。
filter subsystem_id|debug_mask	通过子系统或掩码过滤内核调试消息。
show subsystem_id|debug_mask	显示特定类型的消息。
debug_list subsystems|types	列出所有子系统和调试类型。
modules path	提供 GDB 友好的模块信息。

44.3.4. 选项

使用以下选项调用 lctl。

选项	说明
--device	用于操作的设备（由名称或编号指定）。请参阅 device_list。
--ignore_errors | ignore_errors	在脚本处理期间忽略错误。

44.3.5. 示例

lctl

```
$ lctl
lctl > dl
   0 UP mgc MGC192.168.0.20@tcp btbb24e3-7deb-2ffa-eab0-44dffe00f692 5
   1 UP ost OSS OSS_uuid 3
   2 UP obdfilter testfs-OST0000 testfs-OST0000_UUID 3
lctl > dk /tmp/log Debug log: 87 lines, 87 kept, 0 dropped.
```

```
lctl > quit
```

也可参见"14. mkfs.lustre"，"15. mount.lustre"，"3. lctl"。

44.4. ll_decode_filter_fid

ll_decode_filter_fid实用程序用于显示 Lustre 对象 ID 和 MDT 的父 FID。

44.4.1. 梗概

```
ll_decode_filter_fid object_file [object_file ...]
```

44.4.2. 说明

ll_decode_filter_fid 实用程序为指定 OST 对象解码并打印 Lustre OST 对象 ID、MDT FID 和条带索引，这些信息存储在每个 OST 对象的"trusted.fid" 属性中。当 OST 文件系统在本地挂载为 ldiskfs 类型时，可通过 ll_decode_filter_fid 访问。

"trusted.fid" 扩展属性在首次修改（数据写入或属性集）时即被存储在 OST 对象上，并在此之后不可被 Lustre 访问或修改。

即使通常情况下 LFSCK 可以重建整个 OST 对象目录层次结构（详见 36.4 节，``用 LFSCK 检查文件系统"），OST 对象 ID（objid）在 OST 目录损坏的情况下仍可能有用。MDS FID 可用于确定 OST 对象（曾经）所使用的 MDS inode。条带索引可以在 MDT inode 丢失的情况下联合其他 OST 对象来重建文件布局。

44.4.3. 示例

```
root@oss1# cd /mnt/ost/lost+found
root@oss1# ll_decode_filter_fid #12345[4,5,8]
#123454: objid=690670 seq=0 parent=[0x751c5:0xfce6e605:0x0]
#123455: objid=614725 seq=0 parent=[0x18d11:0xebba84eb:0x1]
#123458: objid=533088 seq=0 parent=[0x21417:0x19734d61:0x0]
```

上面的例子中显示了 lost + found 中的三个十进制对象 ID 为 690670、614725 和 533088 的文件。当前所有 OST 对象的对象序列号（以前的对象组）为 0。

MDT 父节点 FID 是序列格式为oid:idx的十六进制数。由于在所有这些情况下序列号都低于 0x100000000，因此 FID 位于传统的 Inode 和 Generation In FID（IGIF）命名空间中，并直接映射到 MDT inode = seq 和 generation = oid 值；MDT inode 分别为 0x751c5、0x18d11 和 0x21417。对于 MDT 父序列号大于 0x200000000 的对象，则需要通过 MDT 上的 MDT 对象索引（OI）文件来将 FID 映射到内部 inode 编号。

idx 字段将显示 Lustre RAID-0 条带文件中此 OST 对象的条带编号。

也可参见"5. ll_recover_lost_found_objs "。

44.5. ll_recover_lost_found_objs

ll_recover_lost_found_objs 实用程序有助于从lost+found目录中恢复 Lustre OST 对象（文件数据），并根据存储在每个包含数据的 OST 对象上的 trusted.fid扩展属性中的信息，将它们返回到正确的位置。

注意

在 **Lustre 2.6** 和更高版本中不需要这个实用程序，并且已在 Lustre 2.8 中被移除，因为LFSCK在线扫描会自动将对象从 lost+found 恢复到 OST 中的正确位置上。

44.5.1. 梗概

```
llobdstat ost_name [interval]
```

44.5.2. 说明

llobdstat 实用程序按照间隔时间显示给定 ost_name 的 OST 统计信息。它应该直接在 OSS 节点上运行。键入CTRL-C停止统计信息的打印。

44.5.3. 示例

```
# llobdstat liane-OST0002 1
/usr/bin/llobdstat on /proc/fs/lustre/obdfilter/liane-OST0002/stats
Processor counters run at 2800.189 MHz
Read: 1.2144.1e+07, Write: 9.93363e+08, create/destroy: 24/1499, stat: 34,
    p\
unch: 18
[NOTE: cx: create, dx: destroy, st: statfs, pu: punch ]
Timestamp Read-delta ReadRate Write-delta WriteRate
--------------------------------------------------------
1217026053 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026054 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026055 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026056 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026057 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026058 0.00MB 0.00MB/s 0.00MB 0.00MB/s
1217026059 0.00MB 0.00MB/s 0.00MB 0.00MB/s st:1
```

44.5.4. 文件

```
1 /proc/fs/lustre/obdfilter/ostname/stats
```

44.6. llog_reader

llog_reader 实用程序将 Lustre 配置日志转换为易于人们阅读的格式。

44.6.1. 梗概

```
1 llog_reader filename
```

44.6.2. 说明

llog_reader 实用程序可解析 Lustre 磁盘配置日志的二进制格式文件。Llog_reader 只能读取日志，请使用 tunefs.lustre 进行写入操作。

检查已停止的 Lustre 服务器上的日志文件，请将其后备文件系统挂载为 ldiskfs 或 zfs，然后使用 llog_reader 转储日志文件的内容，例如：

```
1 mount -t ldiskfs /dev/sda /mnt/mgs
2 llog_reader /mnt/mgs/CONFIGS/tfs-client
```

在正在运行的 Lustre 服务器上检查相同的日志文件，请使用启用了 ldiskfs 的 debugfs 实用程序（在某些版本中称为 debug.ldiskfs）来提取文件，例如：

```
1 debugfs -c -R 'dump CONFIGS/tfs-client /tmp/tfs-client' /dev/sda
2 llog_reader /tmp/tfs-client
```

注意

虽然它们存储在 CONFIGS 目录中，但 mountdata 文件不使用配置日志格式，会混淆 llog_reader 实用程序。

也可参见'18. tunefs.lustre'。

44.7. llstat

llstat 实用程序将显示 Lustre 统计信息。

44.7.1. 梗概

```
1 llstat [-c] [-g] [-i interval] stats_file
```

44.7.2. 说明

llstat 实用程序将显示使用通用格式的所有 Lustre 统计文件的统计信息，并每隔`interval`秒更新一次。停止打印统计信息，请使用`ctrl-c`。

44.7.3. 选项

选项	说明
-c	清除统计文件
-i	指定轮询周期（以秒为单位）
-g	指定图形输出格式
-h	输出帮助信息
stats_file	指定统计文件的完整路径或快捷引用（mds或ost）

44.7.4. 示例

监控/proc/fs/lustre/ost/OSS/ost/stats 文件，时间间隔为 1 秒，运行：

```
1 llstat -i 1 ost
```

44.7.5. 文件

llstat 文件位于：

```
1 /proc/fs/lustre/mdt/MDS/*/stats
2 /proc/fs/lustre/mdt/*/exports/*/stats
3 /proc/fs/lustre/mdc/*/stats
4 /proc/fs/lustre/ldlm/services/*/stats
5 /proc/fs/lustre/ldlm/namespaces/*/pool/stats
6 /proc/fs/lustre/mgs/MGS/exports/*/stats
7 /proc/fs/lustre/ost/OSS/*/stats
8 /proc/fs/lustre/osc/*/stats
9 /proc/fs/lustre/obdfilter/*/exports/*/stats
10 /proc/fs/lustre/obdfilter/*/stats
11 /proc/fs/lustre/llite/*/stats
```

44.8. llverdev

llverdev 用于验证块设备是否全设备运行正常。

44.8.1. 梗概

```
1 llverdev [-c chunksize] [-f] [-h] [-o offset] [-l] [-p] [-r] [-t timestamp]
    [-v] [-w] device
```

44.8.2. 说明

有时，内核驱动程序错误或硬件设备故障影响了对完整的设备的正常访问。或者，磁盘上存在的坏扇区妨碍了数据的正确存储。通常情况下，主要为系统边界相关的缺陷（如 2^32 bytes, 2^31 sectors, 2^31 blocks, 2^32 blocks 上）。

llverdev 实用程序在整个设备上写入并验证唯一的测试模式来确保数据在写入后可访问，且写入磁盘某一部分的数据不会覆盖磁盘另一部分上的数据。

llverdev 应在大型设备（TB）上运行。在 verbose 模式下运行 llverdev 总是更好，以便设备测试可以轻松地从停止点再次启动。

在非常大的设备上运行完整验证可能非常耗时。我们建议您可以从部分验证开始，从而在进行完整验证之前确保设备至少部分可用。

44.8.3. 选项

选项	说明
-c|--chunksize	I/O 组块大小（字节，默认值为 1048576）。
-f|--force	强制运行测试，不进行是否设备会被覆盖或所有数据被永久销毁的确认。
-h|--help	显示简短的帮助消息。
-o offset	测试开始时的偏移量（千字节，默认值为 0）。
-l|--long	运行完整检查，即写入然后读取并验证磁盘上的每个块。
-p|--partial	运行部分检查，仅对设备进行定期检查（每次 1 GB）。
-r|--read	在以-w 模式运行测试之后，仅在只读（验证）模式下运行测试。
-t timestamp	将测试开始时间设置为先前中断测试开始时打印的时间，以确保整个文件系统中的验证数据相同（默认值为当前时间）。
-v|--verbose	在 verbose 模式下运行测试，列出所有读写操作。
-w| --write	在写模式 (测试模式) 下运行测试 (默认运行读和写测试)

44.8.4. 示例

在/dev/sda 上运行部分设备验证：

```
1 llverdev -v -p /dev/sda
2 llverdev: permanently overwrite all data on /dev/sda (yes/no)? y
```

```
3 llverdev: /dev/sda is 44.98046511104 bytes (4096.0 GB) in size
4 Timestamp: 1009839028
5 Current write offset: 4096 kB
```

使用与上一次运行相同的时间戳，在相对起始位置偏移量为 4096kB 处继续中断的验证：

```
1 llverdev -f -v -p --offset=4096 --timestamp=1009839028 /dev/sda
2 llverdev: /dev/sda is 44.98046511104 bytes (4096.0 GB) in size
3 Timestamp: 1009839028
4 write complete
5 read complete
```

44.9. lshowmount

lshowmount 将显示 Lustre 导出信息。

44.9.1. 梗概

```
1 lshowmount [-ehlv]
```

44.9.2. 说明

lshowmount 实用程序将显示有 Lustre 挂载到服务器的主机，并查找 MGS、MDS 和 obdfilter 的导出信息。

44.9.3. 选项

选项	说明
-e|--enumerate	迫使 lshowmount 在单独一行中列出所有挂载的客户端，而不是将客户端列表压缩为 hostrange 字符串。
-h|--help	打印这些命令的用法相关帮助。
-l|--lookup	迫使 lshowmount 查找看起来像 IP 地址的 NID 主机名。
-v|--verbose	迫使 lshowmount 输出每个服务的导出信息，而不是仅显示服务器上所有 Lustre 服务的总体信息。

44.10.4. 文件

```
1 /proc/fs/lustre/mgs/server/exports/uuid/nid
2 /proc/fs/lustre/mds/server/exports/uuid/nid
3 /proc/fs/lustre/obdfilter/server/exports/uuid/nid
```

44.10. lst

lst 将启动 LNet 自检。

44.10.1. 梗概

```
1 lst
```

44.10.2. 说明

LNet 自检可帮助站点管理员确认 Lustre Networking（LNet）是否已正确安装和配置，以及 LNet 及其网络软件和硬件是否按预期运行。

每个 LNet 自检都在会话环境中运行。一个节点一次只能与一个会话相关联，以确保会话独占其运行的节点。每个会话由从单个节点进行创建、控制和监视，即自检控制台。

任何节点都可以充当自检控制台。节点被命名并分配给组中的自检会话。这将允许组中的所有节点都能被单个名称引用。

测试配置通过描述和运行测试批次来进行创建。测试批次即命名的测试的集合，每个测试由并行运行的多个单独的点对点测试组成。这些单独的点对点测试在被添加到测试批次时根据指定的不同的测试类型、源组、目标组和分布来进行实例化。

44.10.3. 模块

运行 LNet 自检，请加载以下模块：libcfs、lnet、lnet_selftest 和任何一个 klnds（ksocklnd，ko2iblnd …）。加载所有必需的模块，请运行 modprobe lnet_selftest，它将以递归方式加载 lnet_selftest 所依赖的模块。

LNet 自检有两种类型的节点：控制台节点和测试节点。两种节点类型都需要加载所有先前指定的模块。（用户空间测试节点不需要这些模块）

测试节点可以位于内核或用户空间中。控制台用户可以通过运行`lst add_group NID`来邀请内核测试节点加入测试会话，但用户无法主动将用户空间测试节点添加到测试会话。当测试节点运行 lst 客户端以连接到控制台时，控制台用户可以被动地接受测试节点添加到测试会话。

44.10.4. 功能

LNet 自检包括两个用户实用程序，lst 和 lstclient。

lst 为自检控制台的用户界面（在控制台节点上运行）。它提供控制整个测试系统的命令列表，例如创建会话、创建测试组等。

lstclient 为用户空间自检程序，它与用户空间 LND 和 LNet 链接。用户可以调用 lstclient 来加入自检会话：

```
lstclient -sesid CONSOLE_NID group NAME
```

44.10.5. 脚本示例

这是一个 LNet 自检脚本的示例，它模拟了 TCP 网络上一组 Lustre 服务器的流量模式，由 IB 网络上的 Lustre 客户端（通过 LNet 路由器连接）访问，一半客户端读，一半客户端写。

```
#!/bin/bash
export LST_SESSION=$$
lst new_session read/write
lst add_group servers 192.168.10.[8,10,12-16]@tcp
lst add_group readers 192.168.1.[1-253/2]@o2ib
lst add_group writers 192.168.1.[2-254/2]@o2ib
lst add_batch bulk_rw
lst add_test --batch bulk_rw --from readers --to servers     brw read check\
=simple size=1M
lst add_test --batch bulk_rw --from writers --to servers     brw write chec\
k=full size=4K
# start running
lst run bulk_rw
# display server stats for 30 seconds
lst stat servers & sleep 30; kill $!
# tear down
lst end_session
```

44.11. lustre_rmmod.sh

lustre_rmmod.sh 实用程序将删除所有 Lustre 和 LNet 模块（假设没有运行 Lustre 服务）。它位于/usr/bin中。

注意

如果正在使用 Lustre 模块或您已手动运行`lctl network up`命令，则 lustre_rmmod.sh 无法工作。

44.12. lustre_rsync

lustre_rsync 实用程序可将 Lustre 文件系统同步（复制）到目标文件系统。

44.12.1. 梗概

```
lustre_rsync --source|-s src --target|-t tgt
   --mdt|-m mdt [--user|-u userid]
   [--xattr|-x yes|no] [--verbose|-v]
   [--statuslog|-l log] [--dry-run] [--abort-on-err]

lustre_rsync --statuslog|-l log

lustre_rsync --statuslog|-l log --source|-s source
   --target|-t tgt --mdt|-m mdt
```

44.12.2. 说明

lustre_rsync 实用程序旨在将 Lustre 文件系统（源）同步（复制）到另一个文件系统（目标）。目标可以是 Lustre 文件系统或任何其他类型，只要它是正常、可用的文件系统。此同步操作非常有效，由于 lustre_rsync 使用 Lustre MDT 更改日志来识别 Lustre 文件系统中的更改，因此不需要遍历目录。

在使用 lustre_rsync 前：

- 必须注册 changelog 用户（lctl (8) changelog_register）
- 在注册 changelog 用户前，验证 Lustre 文件系统（源）和副本文件系统（目标）是否相同。如果文件系统不一致，请使用实用程序（如常规 rsync，注意，不是 lustre_rsync）将它们统一。

44.12.3. 选项

选项	说明
`--source=src`	被同步的 Lustre 文件系统（源）的根路径。如果未指定在先前同步操作期间创建的有效状态日志（--statuslog），则这是强制选项。

选项	说明
--target=tgt	源文件系统被同步到的根路径（目标）。如果未指定在先前同步操作期间创建的有效状态日志（--statuslog），则这是强制选项。如果有多个目标，则可以重复此选项。
--mdt=mdt	被同步的元数据设备。必须为此设备注册 changelog 用户。如果未指定在先前同步操作期间创建的有效状态日志（--statuslog），则这是强制选项。
--user=userid	指定 MDT 的更改日志用户标识。使用 lustre_rsync，必须注册 changelog 用户。有关详细信息，请参阅 lctl 手册中的 changelog_register 参数介绍。如果未指定在先前同步操作期间创建的有效状态日志（--statuslog），则这是强制选项。
--statuslog=log	同步状态的日志文件。当 lustre_rsync 启动时，将从此处读取先前复制的状态。如果指定了先前同步操作的状态日志，则可以跳过如 --source、--target 或 --mdt 等强制选项。除了 --statuslog 选项外，通过指定 --source，--target 或 --mdt 等选项也可以覆盖状态日志中的参数。命令行选项优先于状态日志中的选项。
--xattryes|no	指定是否同步其扩展属性（xattrs）。默认为同步扩展属性。注意：禁用 xattrs 会导致 Lustre 条带化信息无法同步。
--verbose	输出详细信息。
--dry-run	显示目标文件系统上 lustre_rsync 命令（copy、mkdir 等）的输出，而不实际执行它们。
--abort-on-err	显示目标文件系统上 lustre_rsync 命令（copy、mkdir 等）的输出，而不实际执行它们。

44.12.4. 示例

为某一 MDT （如 lustre-MDT0000）注册一个 changelog 用户。

```
1 $ ssh
2 $ MDS lctl changelog_register \
```

```
3           --device lustre-MDT0000 -n
4 cl1
```

将 Lustre 文件系统（/mnt/lustre）同步/复制到目标文件系统（/mnt/target）。

```
1 $ lustre_rsync --source=/mnt/lustre --target=/mnt/target \
2           --mdt=lustre-MDT0000 --user=cl1 \
3           --statuslog replicate.log  --verbose
4 Lustre filesystem: lustre
5 MDT device: lustre-MDT0000
6 Source: /mnt/lustre
7 Target: /mnt/target
8 Statuslog: sync.log
9 Changelog registration: cl1
10 Starting changelog record: 0
11 Errors: 0
12 lustre_rsync took 1 seconds
13 Changelog records consumed: 22
```

文件系统变更后，将更改同步到目标文件系统。仅需指定 statuslog 名称，其他所有参数已在之前传递。

```
1 $ lustre_rsync --statuslog replicate.log --verbose
2 Replicating Lustre filesystem: lustre
3 MDT device: lustre-MDT0000
4 Source: /mnt/lustre
5 Target: /mnt/target
6 Statuslog: replicate.log
7 Changelog registration: cl1
8 Starting changelog record: 22
9 Errors: 0
10 lustre_rsync took 2 seconds
11 Changelog records consumed: 42
```

将 Lustre 文件系统（/mnt/lustre）同步到两个目标文件系统（/mnt/target1 和/mnt/target2）。

```
1 $ lustre_rsync --source=/mnt/lustre \
2   --target=/mnt/target1 --target=/mnt/target2 \
3   --mdt=lustre-MDT0000 --user=cl1
```

```
4    --statuslog replicate.log
```

44.13. mkfs.lustre

mkfs.lustre 实用程序为 Lustre 服务格式化磁盘。

44.13.1. 梗概

```
1 mkfs.lustre target_type [options] device
```

其中，*target_type* 必须为以下列表中的其中一种：

选项	说明
--ost	对象存储目标 (OST)
--mdt	元数据存储目标 (MDT)
--network=net,...	此 OST/MDT 限制的网络。可以根据需要重复此选项。
--mgs	配置管理服务（MGS），每个站点一个。可以将此服务与 --mdt 服务结合使用来指定两种类型。

44.13.2. 说明

mkfs.lustre可用于格式化磁盘设备并将其作为 Lustre 文件系统的一部分。格式化后，可以装入磁盘并启动此命令定义的 Lustre 服务。

创建文件系统时，可以简单地将参数作为--param选项添加到mkfs.lustre命令中。

选项	说明
--backfstype=fstype	强制设置后备文件系统的格式，如 ldiskfs（默认）或 zfs。
--comment=comment	设置有关此磁盘的用户注释，会被 Lustre 软件忽略。
--device-size=N(KB)	设置 loop 设备的大小。
--dryrun	仅打印执行的输出结果; 它不会影响磁盘。
--servicenode=nid,...	设置所有服务节点的 NID，包括主服务器节

选项	说明
	点和故障转移服务节点。--servicenode 选项不能与--failnode 选项一起使用。
--failnode=nid,...	为目标的主服务器设置故障转移服务节点的 NID。--failnode 选项不能与 --servicenode 选项一起使用。注意，使用--failnode 选项时会有一些限制。
--fsname=filesystem_name	该服务/节点将成为此指定 Lustre 文件系统的一部分。默认文件系统名称为lustre。注意，文件系统名称最长为 8 个字符。
--index=index_number	用于指定 OST 或 MDT 编号（0 … N）。这将允许 OSS 和 MDS 节点与 OST 或 MDT 所在设备之间的映射。
--mkfsoptions=opts	格式化备份文件系统的选项（如，可设置为 ext3）。
--mountfsoptions=opts	挂载备份文件系统时使用的挂载选项。请注意，与早期版本的mkfs.lustre不同，此版本完全将默认挂载选项替换为命令行中指定的挂载选项，如果省略任何默认挂载选项，则会在 stderr 上发出警告信息。ldiskfs 的默认值为：MGS/MDT －errors=remount-ro,iopen_nopriv,user_xattr; OST －errors=remount-ro,extents, mballoc（在 **Lustre 2.5** 中为 OST －errors=remount-roUse)。请谨慎更改默认的挂载选项。
--network=net,...	此 OST/MDT 限制的网络。可以根据需要重

选项	说明
	复此选项。
--mgsnode=nid,...	设置 MGS 节点的 NID（除 MGS 以外的所有目标都需要指定此选项）。
--param key=value	将永久参数 *key* 的值设置为 *value*。可以根据需要重复此选项。以下为常用设置：
`--param sys.timeout=40`	系统 obd 超时时间
`--param lov.stripesize=2M`	默认条带大小
`param lov.stripecount=2`	默认条带数量
`--param failover.mode=failout`	返回错误，不等待恢复
--quiet	打印简明信息。
--reformat	重新格式化已有的 Lustre 磁盘。
--stripe-count-hint=stripes	用于优化 MDT 的 inode 大小。
--verbose	打印更多信息。

44.13.3. 示例

在文件系统 `testfs` 的节点`cfs21`上创建组合的 MGS 和 MDT：

```
1 mkfs.lustre --fsname=testfs --mdt --mgs /dev/sda1
```

在文件系统 `testfs` 的任一节点上创建一个 OST（使用以上 MGS）：

```
1 mkfs.lustre --fsname=testfs --mgsnode=cfs21@tcp0 --ost --index=0 /dev/sdb
```

在节点`cfs22`上创建独立的 MGS：

```
1 mkfs.lustre --mgs /dev/sda1
```

在文件系统 `myfs1` 的任一节点上创建一个 MDT（使用以上 MGS）：

```
1 mkfs.lustre --fsname=myfs1 --mdt --mgsnode=cfs22@tcp0 /dev/sda2
```

也可参见" 本章滴 14. mkfs.lustre"，"15. mount.lustre"。

44.14. mount.lustre

mount.lustre 实用程序可用于启动 Lustre 客户端或目标服务。

44.14.1. 梗概

```
1 mount -t lustre [-o options] device mountpoint
```

44.14.2. 说明

使用 mount.lustre 实用程序启动 Lustre 客户端或目标服务，不应直接调用。它是通过 mount(8) 调用的辅助程序。使用 umount 命令停止 Lustre 客户端和目标。

device 选项有两种形式，具体取决于客户端或目标服务是否已启动：

选项	说明
`mgsname:/fsname[/subdir]`	通过联系 *mgsname* 上的 Lustre Management Service，在目录 mountpoint 中的客户端上挂载名为 fsname 的 Lustre 文件系统（如果指定了 *subdir*，则从文件系统的子目录 *subdir* 启动）。*mgsname* 的格式定义如下。可在`fstab(5)`中列出客户端文件系统，以便在启动时自动挂载。客户端文件系统即可像其他本地文件系统一样使用，并提供完整的 POSIX 标准兼容接口。
block_device	在物理磁盘 *block_device* 上启动由`mkfs.lustre(8)`命令定义的目标服务。指定 *block_device*，可使用`-L label`来查找具有该标签（如`testfs-MDT0000`）的第一个块设备，或通过`-U uuid`选项使用 UUID。如果在同一节点上存在目标文件系统的设备级备份，请格外小心。这是因为如果目标文件系统没有使用`tune2fs(8)`或类似命令进行更改，会产生重复的标签和 UUID。挂载在 *mountpoint* 上的目标服务文件系统仅对`df(1)`操作有用，并会出现在`/proc/mounts`中，表明该设备正在使用中。

44.14.3. 选项

选项	说明
`mgsname=mgsnode[:mgsnode]`	*mgsname* 是以冒号分隔的 *mgsnode* 名称列表，可运行 MGS 服务。如果 MGS 服务配置为 HA 故障切换模式且可能在任何一个节点上运行，则可指定多个 *mgsnode* 值。
`mgsnode=mgsnid[,mgsnid]`	如果 *mgsnode* 有不同的 LNet 接口，则每个 *mgsnode* 通过逗号分隔的 NID 列表进行指定。
`mgssec=flavor`	指定连接 MGS 的初始网络 RPC 的加密特性。非安全的特性有：`null`，`plain`和`gssnull`，分别表示用于测试目的的禁用、无加密功能或非完整性功能。Kerberos 特性有：`krb5n`，`krb5a`，`krb5i`和`krb5p`。共享密钥的风格有：`skn`，`ska`，`ski`和`skpi`。客户端到服务器连接的安全特性在客户端从 MGS 获取的文件系统配置中指定。
`skpath=file|directory`	为此 mount 命令加载的密钥文件的文件路径或目录路径。密钥将被插入到内核的`KEY_SPEC_SESSION_KEYRING`密钥环中，并附带有包含`lustre:`字样及后缀的说明。该后缀取决于 mount 命令的会话是用于 MGS，MDT/OST 还是客户端。
`exclude=ostlist`	启动客户端或 MDT，指定不尝试连接的已知的非活动 OST 列表（由冒号分隔）。

除了标准的 mount(8) 选项外，Lustre 还能读懂以下特定于客户端的选项：

选项	说明
always_ping	即使服务器ptlrpc模块配置了suppress_pings选项，客户端也会在空闲时定期 ping 服务器。这使得客户端即使不是外部客户端运行状况监视机制的一部分也能够可靠地使用文件系统。（在 **Lustre 2.9** 中引入）
flock	使用flock(2)系统调用在参与的应用程序之间启用文件锁定支持，以便文件锁定在所有使用此挂载选项的客户端节点上保持一致。这将在应用程序需要跨多个客户端节点进行一致的用户空间文件锁定时非常有用，但为了保持此一致性同时也增加了通信开销。
localflock	启用客户端本地flock(2)支持，仅使用客户端本地的文件锁定。这比使用全局flock选项更快，并且可以用于依赖于flock(2)但仅在单个节点上运行的应用程序。它通过仅使用 Linux 内核锁实现了最小开销。
noflock	完全禁用flock(2)，为默认选项。调用flock(2)的应用程序会出现ENOSYS错误。管理员可以根据需要选择localflock或flock挂载选项。可使用不同的选项挂载客户端，但只有那些使用flock挂载的客户端才能相互保持一致性。
lazystatfs	在某些 OST 或 MDT 无响应或已在配置中暂时或永久禁用时仍允许返回statfs(2)（被df(1)和lfs-df(1)使用），从而避免所有目标都可用前的阻塞。这是自 **Lustre 2.9.0** 以来的默认行为。
nolazystatfs	使statfs(2)阻塞，直到所有 OST 和 MDT 都可用后再返回空间使用情况。
user_xattr	允许user.*命名空间中的普通用户获取/设置扩展属性。有关更多详细信息，请参见attr(5)手册页。
nouser_xattr	禁用user.*命名空间中的普通用户使用扩展属性。root 和系统进程仍可以使用扩展属性。

选项	说明
verbose	启用额外的 mount/umount 控制台消息。
noverbose	禁用额外的 mount/umount 控制台消息。
user_fid2path	允许普通用户进行 FID 的路径转换。注意：此选项存在潜在的安全漏洞，因为它允许了普通用户绕过基于 POSIX 路径的权限检查（会阻止用户访问他们无权访问的目录中的文件）而直接通过其文件 ID 访问文件。仍然会对文件本身执行常规权限检查，因此用户无法访问他们没有访问权限的文件。（在 **Lustre 2.3** 中引入）
nouser_fid2path	禁止普通用户进行 FID 的路径转换。root 和系统进程仍可以使用 CAP_DAC_READ_SEARCH进行 FID 的路径转换。

除了标准安装选项和后备磁盘类型（如 ldiskfs）选项之外，Lustre 还能读懂以下特定于服务器的挂载选项：

选项	说明
nosvc	为目标服务而不是实际服务启动 MGC（以及 MGS(co-located) ）。
nomgs	仅启动 MDT（以及 MGS (co-located) ），不启动 MGS。
abort_recov	中止该服务器上的客户端恢复并立即启动目标服务。
max_sectors_kb=KB	设置挂载的 MDT 或 OST 目标的块设备参数max_sectors_kb（千字节数）。当未指定max_sectors_kb为挂载选项时，将自动设置为该块设备的max_hw_sectors_kb（最大为 16MiB）。此默认行为适用于大多数用户。设置max_sectors_kb =0时，将保留此可调

选项	说明
	参数的当前值。（在 **Lustre 2.10** 中引入）
`md_stripe_cache_size`	使用条带化 RAID 配置为服务器端磁盘设置条带高速缓存大小。
`recovery_time_soft=timeout`	允许客户端在服务器崩溃后重新连接，超时秒数设置为`timeout`。如果该超时即将到期而服务器仍在处理来自可恢复客户端的新连接，则此超时时间将逐步增加。默认软恢复超时值（5 分钟）是 Lustre 超时参数值（100 秒）的 3 倍。软恢复超时在挂载时设置，即使 Lustre 超时值在挂载后更改，软恢复超时值不变。
`recovery_time_hard=timeout`	允许服务器将其超时递增延长到最大为`timeout`的硬恢复超时值。默认的硬恢复超时（15 分钟）是 Lustre 超时参数值（100 秒）的 9 倍。硬恢复超时在挂载时设置，即使 Lustre 超时值在挂载后更改，硬恢复超时值不变。
`noscrub`	通常，MDT 将在挂载期间检测文件级备份的恢复，以避免挂载 MDT 时 OI Scrub 自动启动。在挂载后手动启动 LFSCK 可以更好地控制启动条件。此挂载选项还可以防止在检测到 OI 不一致时 OI Scrub 自动启动。

44.14.4. 示例

在挂载点 */mnt/chip* 上启动客户端的 Lustre 文件系统 chipfs。此客户端可通过 cfs21@tcp0 NID 访问管理服务正在运行的节点。

```
1 mount -t lustre cfs21@tcp0:/chipfs /mnt/chip
```

将 *chipfs* 的子目录作为文件集进行挂载（在 **Lustre 2.9** 中引入）。

```
1 mount -t lustre cfs21@tcp0:/chipfs/v1_0 /mnt/chipv1_0
```

从挂载点/mnt/test/mdt 上的/dev/sda1 启动 Lustre 元数据目标服务。

```
1 mount -t lustre /dev/sda1 /mnt/test/mdt
```

启动 testfs-MDT0000 服务（使用磁盘标签），但中止恢复过程。

```
1 mount -t lustre -L testfs-MDT0000 -o abort_recov /mnt/test/mdt
```

也可见本章第 14 节"mkfs.lustre"，第 18 节"unefs.lustre" 和第 3 节"lctl"。

44.15. plot-llstat

plot-llst 实用程序可用于绘制 Lustre 统计信息。

44.15.1. 梗概

```
1 plot-llstat results_filename [parameter_index]
```

44.15.2. 说明

plot-llstat 实用程序从 llstat 的输出生成用于 gnuplot 的 CSV 文件和说明文件。由于 llstat 本质上是通用的，因此 plot-llstat 也是一个通用脚本。

plot-llstat 使用用户指定的操作数创建.dat（CSV）文件。CSV 文件中的列数与操作数相等，这些列中的值对应于输出文件中 parameter_index 值。

plot-llstat 还会创建一个.scr 文件，其中包含了 gnuplot 绘制图形的说明。生成.dat 和.scr 文件后，plot-llstat 工具调用 gnuplot 来显示相应的图形。

44.15.3. 选项

选项	说明
`results_filename`	plot-llstat 生成的输出
`parameter_index`	值可以为：1 －每个时间间隔的计数；2 －每秒的计数（默认值）；3 －总计数。

44.15.4. 示例

```
1 llstat -i2 -g -c lustre-OST0000 > log
2 plot-llstat log 3
```

44.16. routerstat

routerstat 实用程序可用于打印 Lustre 路由器统计信息。

44.16.1. 梗概

```
1 routerstat [interval]
```

44.16.2. 说明

routerstat 实用程序将显示 LNet 路由器统计信息。如果未指定*interval*，则仅对统计信息采样及打印一次。否则，将以指定的时间间隔（*interval*以秒为单位）对统计信息进行采样和打印。

44.16.3. 输出

routerstat 的输出包含了以下内容：

条目	说明
M	LNet 当前正在处理的消息数（LNet 并行处理的最大消息数）
E	LNet 错误数
S	发送的消息总大小（字节长度）/发送的消息数
R	接收的消息总大小（字节长度）/接收的消息数
F	路由转发的消息总大小（字节长度）/路由转发的消息数
D	丢失的消息总大小（字节长度）/丢失的消息数

指定了 *interval* 时，还将打印以下附加的统计信息：

条目	说明
M	LNet 当前正在处理的消息数（LNet 并行处理的最大消息数）
E	每秒发生的 LNet 错误数
S	数据发送率（Mbytes/s）/每秒发送的消息数
R	数据接收率（Mbytes/s）/每秒接受的消息数
F	数据转发率（Mbytes/s）/每秒转发的消息数
D	数据丢失率（Mbytes/s）/每秒丢失的消息数

44.16.4. 示例

```
# routerstat 1
M 0(13) E 0 S 117379184/4250 R 878480/44.56 F 0/0 D 0/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    8.00/     8 R    0.00/    16 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    8.00/     8 R    0.00/    16 F    0.00/     0 D 0.00/0
M   0( 13) E 0 S    7.00/     7 R    0.00/    14 F    0.00/     0 D 0.00/0
...
```

44.16.5. 文件

routerstat 可从以下文件中提取统计信息：

```
/proc/sys/lnet/stats
```

44.17. tunefs.lustre

The tunefs.lustre 实用程序可用于修改 Lustre 目标磁盘上的配置信息。

44.17.1. 梗概

```
tunefs.lustre [options] /dev/device
```

44.17.2. 说明

tunefs.lustrek 可用于修改 Lustre 目标磁盘上的配置信息。这不会重新格式化磁盘或擦除目标信息，但修改配置信息可能会导致文件系统无法使用。

注意

此处所做的更改只在下次挂载目标时产生效果。

使用 tunefs.lustre 时，参数是" 附加的" 。即除旧参数外，指定的新参数不会替换它们，而是附加上去的。要删除所有旧的 tunefs.lustre 参数并仅使用新指定的参数，请运行：

```
$ tunefs.lustre --erase-params --param=new_parameters
```

tunefs.lustre命令可用于设置 /proc/fs/lustre文件中可设置具有自己的 OBD 设备的任何参数，因此可以将其指定为 *bd|fs-nameobd|fsname.obdtype.proc_file_name=value*。例如：

```
1 $ tunefs.lustre --param mdt.identity_upcall=NONE /dev/sda1
```

44.17.3. 选项

tunefs.lustre 选项如下所示：| **选项** | **说明** | | ---------- | ---------- | | --comment=comment | 设置有关此磁盘的用户注释，会被 Lustre 忽略。| | --dryrun | 只打印命令的输出，不执行命令。| | --erase-params | 删除所有先前的参数信息。| | --servicenode=nid,... | 设置所有服务节点的 NID，包括主服务器节点和 | | | 故障切换服务节点。--servicenode选项不能 | | | 与--failnode选项一起使用。| | --failnode=nid,... | 为目标的主服务器设置故障切换服务节点的 NID。| | | --servicenode选项不能与--failnode选项一 | | | 起使用。注意使用 --failnode 选项时有一些限 | | | 制。| | --fsname=filesystem_name | 该服务将成所指定 Lustre 文件系统其中的一部分。| | | 默认文件系统名称为lustre。| | --index=index | 强制设置特定的 OST 或 MDT 索引。| | --mountfsoptions=opts | 设置备份文件系统挂载时使用的挂载选项。注意，| | | 与早期版本的 tunefs.lustre 不同，此版本完全将现 | | | 有挂载选项替换为命令行中指定的挂载选项。如果 | | | 省略任何默认挂载选项，将在 stderr 上发出警告。| | | ldiskfs 的默认值为：MGS/MDT －errors=remount- | | | ro,iopen_nopriv,user_xattr; OST － | | | errors=remount-ro,extents,mballoc | | | （在 **Lustre 2.5** 中，OST －errors=remount-）。ro | | | 请不要在不明状况时轻易更改默认挂载选项。| | --network=net,... | OST / MDT 限制的网络。可以根据需要重复此选项。| | --mgs | 添加此目标的配置管理服务。| | --msgnode=nid,... | 设置 MGS 节点的 NID（除 MGS 之外的所有目标）。| | --nomgs | 删除此目标的配置管理服务。| | --quiet | 打印简短的信息。| | --verbose | 打印更多信息。| | --writeconf | 擦除此 MDT 所属的文件系统的所有配置日志，并重新 | | | 生成它们。这是非常危险的操作，请务必卸载所有客 | | | 户端并停止此文件系统的服务器。随后，请重启所有 | | | 目标（OST / MDT）以重新生成日志。在重新启动所 | | | 有目标之前，请不要启动任何客户端。正确的操作顺 | | | 序是：1. 卸载文件系统上的所有客户端，2. 卸载文件系统 | | | 上的 MDT 和所有 OST，3. 在每个服务器上运行 | | | tunefs.lustre --writeconf device，4. 挂载 MDT 和 | | | OST，5. 挂载客户端。|

44.17.4. 示例

更改 MGS 的 NID 地址。（在每个目标磁盘上执行，它们都应联系同一个 MGS。）

```
tunefs.lustre --erase-param --mgsnode=new_nid --writeconf /dev/sda
```

为此目标添加故障转移 NID 位置。

```
tunefs.lustre --param="failover.node=192.168.0.13@tcp0" /dev/sda
```

也可见本章第 14 节"mkfs.lustre"，第 15 节"mount.lustre" 和第 3 节"lctl"。

44.18. 附加系统配置程序

本节主要介绍 Lustre 的其他系统配置实用程序。

44.18.1. 应用程序分析工具

`lustre_req_history.sh`位于/usr/bin 中，它从客户端运行，从本地节点和连接的服务器收集尽可能多的 Lustre RPC 请求历史记录，从而更好地了解协调网络活动。

44.18.2. More/proc 统计信息

`vfs_ops_stats`提供了更多统计信息，它通过 PID, PPID, GID 等来跟踪 Linux VFS 操作调用。

```
/proc/fs/lustre/llite/*/vfs_ops_stats
/proc/fs/lustre/llite/*/vfs_track_[pid|ppid|gid]
```

`extents_stats`可用于显示来自客户端的 I/O 调用的大小分布（累计值和每进程值）。

```
/proc/fs/lustre/llite/*/extents_stats, extents_stats_per_proces
```

`offset_stats`通过偏移和范围显示客户端的读/写搜索活动。

```
/proc/fs/lustre/llite/*/offset_stats
```

Lustre 也包含了 Per-client（每个客户端的）和优化的 MDT 统计信息：

- 服务器上追踪的 Per-client 统计信息

每个 MDS 和 OSS 都会跟踪每个连接客户端的 LDLM 和操作统计信息，以便对分发的作业的统计信息进行更方便的收集和比较。

```
/proc/fs/lustre/mds|obdfilter/*/exports/
```

- 优化的 MDT 统计信息

收集更详细的 MDT 操作统计信息以获得更好的分析。

```
/proc/fs/lustre/mdt/*/md_stats
```

44.18.3. 测试和调试工具

Lustre 提供了以下测试和调试实用程序。

44.18.3.1. lr_reader lr_reader 实用程序将 last_rcvd 和 reply_data 文件的内容转换为易于人们阅读的格式。

以下工具也是 Lustre I/O 工具包的一部分。

44.18.3.2. sgpdd-survey sgpdd-survey 实用程序可绕过尽可能多的内核从而测试" 裸机" 性能。它不需要 Lustre，但需要 sgp_dd 包。

注意 `sgpdd-survey` 将擦除设备上所有数据。

44.18.3.3. obdfilter-survey obdfilter-survey 实用程序是一个 shell 脚本，用于测试被隔离的 OST 的性能、echo 客户端网络，以及端到端测试。

44.18.3.4. ior-survey ior-survey 实用程序是用于运行 IOR 基准测试的脚本。Lustre 支持 IOR 2.8.6。

44.18.3.5. ost-survey ost-survey 实用程序可用于调查 OST 性能，将测试 Lustre 文件系统中各个 OST 的客户端到磁盘的性能。

44.18.3.6. stats-collect stats-collect 实用程序包含用于从 Lustre 客户端和服务器收集应用程序分析信息的脚本。

44.18.4. Fileset（文件集）功能

（在 **Lustre 2.9** 中引入）

Lustre 通过文件集功能来提供子目录挂载支持。子目录挂载（也称为文件集）允许客户端挂载父文件系统的子目录，从而限制文件系统命名空间在特定客户端上的可见性。一个常见的用法是：为防止挂载的子目录之外的文件的意外，客户端可以使用子目录挂载，以限制整个文件系统命名空间的可见性。

值得注意的是，是否调用子目录挂载是客户端自愿的，这不会影响对多个子目录中硬链接可见的文件的访问。此外，它也不会影响客户端随后在没有指定子目录的情况下挂载整个文件系统。

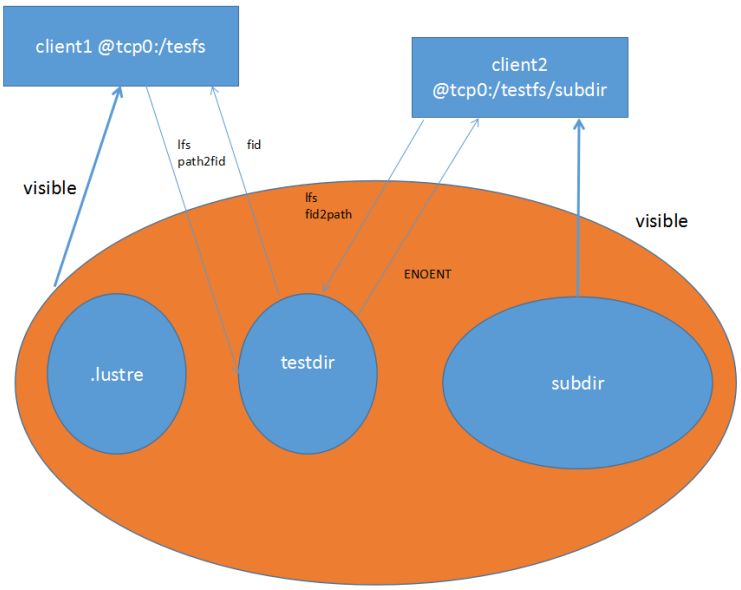


图 29: Lustre file system fileset feature

图 44.1 Lustre 文件集

44.18.4.1. 示例 以下示例将在 client1 上挂载chipfs文件系统，并在该文件系统中创建子目录v1_1。随后，Client2 将把 v1_1 子目录挂载为文件集，从而限制 client2 访问chipfs文件系统中的任何其他内容。

```
client1# mount -t lustre mgs@tcp:/chipfs /mnt/chip
client1# mkdir /mnt/chip/v1_1
```

```
client2# mount -t lustre mgs@tcp:/chipfs/v1_1 /mnt/chipv1_1
```

您可以在/etc/mtab 中检查所创建的挂载。它应该如下所示：

```
client1
mds@tcp0:/chipfs/ /mnt/chip lustre rw          0       0
```

```
client2
mds@tcp0:/chipfs/v1_1 /mnt/chipv1_1 lustre rw          0       0
```

在/mnt/chip 下创建一个目录，并获取其 FID：

```
client1# mkdir /mnt/chip/v1_2
client1# lfs path2fid /mnt/chip/v1_2
[0x200000400:0x2:0x0]
```

如果您尝试在 client2 上解析 /mnt/chip/v1_2 路径的 FID（如上例中所示），则会返回错误。无法在 client2 上解析 FID 是因为它在该客户端上不属于已挂载文件集的一部分（client2 上的文件集挂载在chipfs文件系统根目录下的v1_1子目录）。

```
client2# lfs fid2path /mnt/chip/v1_2 [0x200000400:0x2:0x0]
fid2path: error on FID [0x200000400:0x2:0x0]: No such file or directory
```

子目录挂载不包含.lustre目录，这将阻止客户端通过 FID 直接打开或访问文件。

```
client1# ls /mnt/chipfs/.lustre
        fid  lost+found
client2# ls /mnt/chipv1_1/.lustre
        ls: cannot access /mnt/chipv1_1/.lustre: No such file or directory
```

第四十五章 LNet 配置的 C-API

45.1. API 通用信息

45.1.1. API 返回代码

```
LUSTRE_CFG_RC_NO_ERR                0
LUSTRE_CFG_RC_BAD_PARAM            -1
LUSTRE_CFG_RC_MISSING_PARAM        -2
LUSTRE_CFG_RC_OUT_OF_RANGE_PARAM   -3
LUSTRE_CFG_RC_OUT_OF_MEM           -4
LUSTRE_CFG_RC_GENERIC_ERR          -5
```

45.1.2. API 普通输入参数

所有 API 都将序列号作为输入，这是一个由 API 的调用者分配的数字，并且会包含在 YAML 错误返回块中。它用于将请求与响应相关联。它在通过 YAML 接口进行配置时尤其有用，因为 YAML 接口通常用于配置多个项目，而在返回错误块中，需要知道哪些项目已正确配置、哪些项目未正确配置。序列号正好达到了这个目的。

45.1.3. API 普通输出参数

45.1.3.1. YAML 内部表征 (cYAML) YAML 块完成解析后，需要进行结构化存储它，以便于将其传递给不同的函数、查询或打印。此外，还需要能够从内核返回的数据构建此内部表征，并将其返回给调用者以供调用者查询和打印。此结构表征用于 Error 和 Show API Out 参数。YAML 在内部被结构化表示为:

```
typedef enum {
    EN_YAML_TYPE_FALSE = 0,
    EN_YAML_TYPE_TRUE,
    EN_YAML_TYPE_NULL,
```

```
    EN_YAML_TYPE_NUMBER,
    EN_YAML_TYPE_STRING,
    EN_YAML_TYPE_ARRAY,
    EN_YAML_TYPE_OBJECT
} cYAML_object_type_t;

typedef struct cYAML {
    /* next/prev allow you to walk array/object chains. */
    struct cYAML *cy_next, *cy_prev;
    /* An array or object item will have a child pointer pointing
       to a chain of the items in the array/object. */
    struct cYAML *cy_child;
    /* The type of the item, as above. */
    cYAML_object_type_t cy_type;
    /* The item's string, if type==EN_YAML_TYPE_STRING */
    char *cy_valuestring;
    /* The item's number, if type==EN_YAML_TYPE_NUMBER */
    int cy_valueint;
    /* The item's number, if type==EN_YAML_TYPE_NUMBER */
    double cy_valuedouble;
    /* The item's name string, if this item is the child of,
       or is in the list of subitems of an object. */
    char *cy_string;
    /* user data which might need to be tracked per object */
    void *cy_user_data;
} cYAML;
```

45.1.3.2. 错误块 所有 API 都会返回一个 cYAML 错误块。打印输出时，所有配置错误都应以 YAML 序列表示，具有以下格式：

```
<cmd>:
  - <entity>:
    errno: <error number>
    seqno: <sequence number>
    descr: <error description>

```

```
Example:
add:
  - route
      errno: -2
      seqno: 1
      descr: Missing mandatory parameter(s) - network
```

45.1.3.3. 显示块 所有显示 API 都会返回一个 cYAML 显示块。此显示块将表征以 YAML 格式请求的信息。每个配置项都有自己的 YAML 语法。所有支持的配置项的 YAML 语法将在本文档的后面部分介绍。以下是显示块的示例：

```
net:
    - nid: 192.168.206.130@tcp4
      status: up
      interfaces:
          0: eth0
      tunables:
          peer_timeout: 10
          peer_credits: 8
          peer_buffer_credits: 30
          credits: 40
```

45.2. LNet 配置 C-API

45.2.1. 配置 LNet

```
/*
 * lustre_lnet_config_ni_system
 *   Initialize/Uninitialize the LNet NI system.
 *
 *   up - whether to init or uninit the system
 *   load_ni_from_mod - load NI from mod params.
 *   seq_no - sequence number of the request
 *   err_rc - [OUT] struct cYAML tree describing the error. Freed by
 *            caller
 */
int lustre_lnet_config_ni_system(bool up, bool load_ni_from_mod,
```

```
12                                      int seq_no, struct cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_CONFIGURE 或 IOC_LIBCFS_UNCONFIGURE

说明：

- 配置 LNet：

如果设置了load_ni_from_mod，则初始化 LNet 内部并加载模块参数中指定的任何网络。否则不要加载任何网络接口。

- 取消 LNet 的配置：

关闭 LNet 并清除网络接口、路由和所有 LNet 内部信息。

- 返回值：

成功为 0；失败为相应的 errno。

45.2.2. 启用/禁用路由

```
1 /*
2  * lustre_lnet_enable_routing
3  *   Send down an IOCTL to enable or disable routing
4  *
5  *   enable - 1 to enable routing, 0 to disable routing
6  *   seq_no - sequence number of the request
7  *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
8  */
9 extern int lustre_lnet_enable_routing(int enable,
10                                        int seq_no,
11                                        cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ENABLE_RTR

说明：

- 启用路由：

使用默认值分配路由器缓冲池。然后在内部将节点标记为路由器节点（即该节点此时起可以用作路由器）。

- 禁用路由器：

释放未使用的路由器缓冲池。目前正在使用的缓冲区在返回到未使用列表之前不会被释放。在内部将节点的路由标志关闭。不是发往此节点的任何后续消息都将被删除。

- 在已启用的节点上启用路由器 (或相反):

在这两种情况下，LNet Kernel 模块都会忽略这个请求。

- 返回值:

成功为 0；如果没有足够的内存分配缓冲池则为 ENOMEM。

45.2.3. 添加路由

```
/*
 * lustre_lnet_config_route
 *   Send down an IOCTL to the kernel to configure the route
 *
 *   nw - network
 *   gw - gateway
 *   hops - number of hops passed down by the user
 *   prio - priority of the route
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_route(char *nw, char *gw,
                    int hops, int prio,
                    int seq_no,
                    cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ADD_ROUTE

说明:

LNet 内核模块将此路由添加到现有路由列表（如果还不在此列表中)。如果未指定跳数参数（IE：-1)，则将跳数设置为 1。如果未指定优先级参数（IE：-1)，则将优先级设置为 0。使用循环法处理具有相同跳数和优先级的路由，优选选取较低跳数和/或较高优先级（0 代表最高优先级）的路由。

如果路由已存在，则忽略此添加路由请求。

- 返回值:
- EINVAL：该条路由指向本地网络。
- ENOMEM：无足够内存。

- EHOSTUNREACH：主机不在本地网络上。
- 0：成功。

45.2.4. 删除路由

```
/*
 * lustre_lnet_del_route
 *   Send down an IOCTL to the kernel to delete a route
 *
 *   nw - network
 *   gw - gateway
 */
extern int lustre_lnet_del_route(char *nw, char *gw,
                 int seq_no,
                 cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_DEL_ROUTE

说明:

LNet 将删除与传入的网络和网关匹配的路由。如果没有路由匹配，则操作将失败并显示相应的错误编号。

- 返回值:

成功为 0；如果没有要删除的条目不存在为 ENOENT。

45.2.5. 显示路由

```
/*
 * lustre_lnet_show_route
 *   Send down an IOCTL to the kernel to show routes
 *   This function will get one route at a time and filter according to
 *   provided parameters. If no filter is provided then it will dump all
 *   routes that are in the system.
 *
 *   nw - network.  Optional.  Used to filter output
 *   gw - gateway. Optional. Used to filter ouptut
 *   hops - number of hops passed down by the user
 *          Optional.  Used to filter output.
```

```
 *   prio - priority of the route.  Optional.  Used to filter output.
 *   detail - flag to indicate whether detail output is required
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_show_route(char *nw, char *gw,
                  int hops, int prio, int detail,
                  int seq_no,
                  cYAML **show_rc,
                  cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_GET_ROUTE

说明:

根据传入的参数进行过滤后，将从内核中逐个获取路径并打包在 cYAML 块中。cYAML 块随后将返回给 API 的调用者。

以下为 detail 参数设置为 1 时的示例:

```
route:
    net: tcp5
    gateway: 192.168.205.130@tcp
    hop: 1.000000
    priority: 0.000000
    state: up
```

以下为 detail 参数设置为 0 时的示例:

```
route:
    net: tcp5
    gateway: 192.168.205.130@tcp
```

- 返回值:

成功为 0；如果没有足够的内存则为 ENOMEM。

45.2.6. 添加网络接口

```
/*
 * lustre_lnet_config_net
 *   Send down an IOCTL to configure a network.
```

```
 *
 *   net - the network name
 *   intf - the interface of the network of the form net_name(intf)
 *   peer_to - peer timeout
 *   peer_cr - peer credit
 *   peer_buf_cr - peer buffer credits
 *       - the above are LND tunable parameters and are optional
 *   credits - network interface credits
 *   smp - cpu affinity
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_net(char *net,
                  char *intf,
                  int peer_to,
                  int peer_cr,
                  int peer_buf_cr,
                  int credits,
                  char *smp,
                  int seq_no,
                  cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ADD_NET

说明:

添加并初始化一个新的网络。这与从模块参数配置网络具有相同的效果。API 允许指定如对等超时、对等信用，对等缓冲信用和信用等网络参数，同时也可指定添加的网络接口的 CPU 亲和性。这些参数在动态 LNet 配置（DLC）下变为特定于网络，而不是特定于 LND。

如果添加的网络已经存在，该请求将被忽略。

- 返回值:
- EINVAL：传入的网络无法识别。
- ENOMEM：无足够内存。
- 0：成功。

45.2.7. 删除网络接口

```
/*
```

```
 * lustre_lnet_del_net
 *   Send down an IOCTL to delete a network.
 *
 *   nw - network to delete.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_del_net(char *nw,
                   int seq_no,
                   cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_DEL_NET

说明:

指定的网络接口将被删除，与此网络接口关联的所有资源将被释放，通过该网络接口的所有路由也都将被清除。

- 返回值:

成功时为 0；该请求指向无存在的网络时为 EINVAL。

45.2.8. 显示网络接口

```
/*
 * lustre_lnet_show_net
 *   Send down an IOCTL to show networks.
 *   This function will use the nw paramter to filter the output.  If it's
 *   not provided then all networks are listed.
 *
 *   nw - network to show.  Optional.  Used to filter output.
 *   detail - flag to indicate if we require detail output.
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_show_net(char *nw, int detail,
                int seq_no,
                cYAML **show_rc,
                cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_GET_NET

说明:

在网络上过滤（EX：tcp）后，将从内核逐个查询网络接口并打包在 cYAML 块中。如果 detail 字段设置为 1，则显示块的可调节部分将被包含在返回值中。

详细输出示例如下:

```
net:
    nid: 192.168.206.130@tcp4
    status: up
    interfaces:
        intf-0: eth0
    tunables:
        peer_timeout: 10
        peer_credits: 8
        peer_buffer_credits: 30
        credits: 40
```

非详细输出示例如下:

```
net:
    nid: 192.168.206.130@tcp4
    status: up
    interfaces:
        intf-0: eth0
```

- 返回值:

成功为 0；如果没有足够的内存分配错误块或显示块则为 ENOMEM。

45.2.9. 调整路由器缓冲池

```
/*
 * lustre_lnet_config_buf
 *   Send down an IOCTL to configure buffer sizes.  A value of 0 means
 *   default that particular buffer to default size. A value of -1 means
 *   leave the value of the buffer unchanged.
 *
 *   tiny - tiny buffers
 *   small - small buffers
 *   large - large buffers.
```

```
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_buf(int tiny,
                  int small,
                  int large,
                  int seq_no,
                  cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ADD_BUF

说明：

此 API 用于动态配置微型、小型和大型路由器缓冲区。这些缓冲区用于缓存正在被路由到其他节点的消息。这些缓冲区（每个 CPT）的最小值是：

```
#define LNET_NRB_TINY_MIN     512
#define LNET_NRB_SMALL_MIN    4096
#define LNET_NRB_LARGE_MIN    256
```

这些缓冲区的默认值是：

```
#define LNET_NRB_TINY         (LNET_NRB_TINY_MIN * 4)
#define LNET_NRB_SMALL        (LNET_NRB_SMALL_MIN * 4)
#define LNET_NRB_LARGE        (LNET_NRB_LARGE_MIN * 4)
```

这些默认值在所有 CPT 中均匀分配。但是每个 CPT 最低不能低于上述最小值。

使用相同的值多次调用此 API 为无效操作。

- 返回值：

成功为 0；如果没有足够的内存分配缓冲池则为 ENOMEM。

45.2.10. 显示路由信息

```
/*
 * lustre_lnet_show_routing
 *   Send down an IOCTL to dump buffers and routing status
 *   This function is used to dump buffers for all CPU partitions.
 *
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] struct cYAML tree describing the error. Freed by caller
 */
```

```
extern int lustre_lnet_show_routing(int seq_no, struct cYAML **show_rc,
                                    struct cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_GET_BUF

说明：

此 API 将返回一个 cYAML 块，用于描述以下值（每个 CPT）：

1. 每个缓冲区的页数。这是一个常数。
2. 分配的缓冲区数。这是一个常数。
3. 缓冲信用积分。这是当前可用的缓冲信用数的实时值。如果此值为负，则表示的是排队的消息数。
4. 系统中曾到达的最低信用积分。这是历史数据。

显示块同时也将返回路由状态（无论启用或禁用）。

以下是该 YAML 块的一个示例：

```
routing:
    - cpt[0]:
          tiny:
              npages: 0
              nbuffers: 2048
              credits: 2048
              mincredits: 2048
          small:
              npages: 1
              nbuffers: 16384
              credits: 16384
              mincredits: 16384
          large:
              npages: 256
              nbuffers: 1024
              credits: 1024
              mincredits: 1024
    - enable: 1
```

- 返回值：

成功为 0；如果没有足够的内存分配错误块或显示块则为 ENOMEM。

45.2.11. 显示 LNet 流量统计数据

```
/*
 * lustre_lnet_show_stats
 *   Shows internal LNet statistics.  This is useful to display the
 *   current LNet activity, such as number of messages route, etc
 *
 *     seq_no - sequence number of the command
 *     show_rc - YAML structure of the resultant show
 *     err_rc - YAML strucutre of the resultant return code.
 */
extern int lustre_lnet_show_stats(int seq_no, cYAML **show_rc,
                  cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_GET_LNET_STATS

说明:

此 API 将返回用于描述 LNet 流量统计信息的 cYAML 块。当 LNet 存活时，统计数据会不断累加。此 API 在 API 调用时返回统计信息。统计数据包括以下内容:

1. 已分配的消息数。
2. 系统中做的最大消息数。
3. 分配或发送消息的错误。
4. 发送的累积消息数。
5. 收到的累积消息数。
6. 路由的累积消息数。
7. 丢弃的累积消息数。
8. 发送的累积字节数。
9. 收到的累积字节数。
10. 路由的累积字节数。
11. 丢弃的累积字节数。

以下是该 YAML 块的一个示例:

```
statistics:
    msgs_alloc: 0
    msgs_max: 0
    errors: 0
    send_count: 0
```

```
    recv_count: 0
    route_count: 0
    drop_count: 0
    send_length: 0
    recv_length: 0
    route_length: 0
    drop_length: 0
```

- 返回值：

成功为 0；如果没有足够的内存分配错误块或显示块则为 ENOMEM。

45.2.12. 添加/删除/显示参数

```
/*
 * lustre_yaml_config
 *   Parses the providedYAMLfile and then calls the specific APIs
 *   to configure the entities identified in the file
 *
 *   f -YAMLfile
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_config(char *f, cYAML **err_rc);

/*
 * lustre_yaml_del
 *   Parses the providedYAMLfile and then calls the specific APIs
 *   to delete the entities identified in the file
 *
 *   f -YAMLfile
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_del(char *f, cYAML **err_rc);

/*
 * lustre_yaml_show
 *   Parses the providedYAMLfile and then calls the specific APIs
```

```
 *   to show the entities identified in the file
 *
 *   f - YAML file
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_show(char *f,
                cYAML **show_rc,
                cYAML **err_rc);
```

IOCTL to Kernel:

依赖于配置的实体。

说明:

这些 API 将分别添加、删除、显示在 YAML 文件中指定的参数。这些实体不必是统一，可以在一个 YAML 块中添加/移除/显示多个不同的实体。

以下是该 YAML 块的一个示例:

```
---
net:
    - nid: 192.168.206.132@tcp
      status: up
      interfaces:
          0: eth3
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
          SMP: "[0]"
route:
   - net: tcp6
     gateway: 192.168.29.1@tcp
     hop: 4
     detail: 1
     seq_no: 3
   - net: tcp7
     gateway: 192.168.28.1@tcp
```

```
    hop: 9
    detail: 1
    seq_no: 4
buffer:
  - tiny: 1024
    small: 2000
    large: 512
...
```

- 返回值：

返回值将对应于将在配置项上操作的 API 的返回值。

45.2.13. 添加路由的代码示例

```
int main(int argc, char **argv)
{
    char *network = NULL, *gateway = NULL;
    long int hop = -1, prio = -1;
    struct cYAML *err_rc = NULL;
    int rc, opt;
    optind = 0;

    const char *const short_options = "n:g:c:p:h";
    const struct option long_options[] = {
        { "net", 1, NULL, 'n' },
        { "gateway", 1, NULL, 'g' },
        { "hop-count", 1, NULL, 'c' },
        { "priority", 1, NULL, 'p' },
        { "help", 0, NULL, 'h' },
        { NULL, 0, NULL, 0 },
    };

    while ((opt = getopt_long(argc, argv, short_options,
                    long_options, NULL)) != -1) {
        switch (opt) {
        case 'n':
            network = optarg;
```

```
            break;
        case 'g':
            gateway = optarg;
            break;
        case 'c':
            rc = parse_long(optarg, &hop);
            if (rc != 0) {
                /* ignore option */
                hop = -1;
                continue;
            }
            break;
        case 'p':
            rc = parse_long(optarg, &prio);
            if (rc != 0) {
                /* ingore option */
                prio = -1;
                continue;
            }
            break;
        case 'h':
            print_help(route_cmds, "route", "add");
            return 0;
        default:
            return 0;
        }
    }

    rc = lustre_lnet_config_route(network, gateway, hop, prio, -1, &err_rc);

    if (rc != LUSTRE_CFG_RC_NO_ERR)
        cYAML_print_tree2file(stderr, err_rc);

    cYAML_free_tree(err_rc);

    return rc;
```

```
60 }
```

其他代码示例请参照：

```
1 lnet/utils/lnetctl.c
```