NFS 测试报告

- 前言
  - 关于测试目的
  - 关于测试维度
  - 关于测试环境
  - 关于 NFS 实现
  - 关于测试结果
- 场景 1: ls 可见性
  - 1.1: 创建文件
  - 1.2: 创建目录
  - 1.3: 删除文件
  - 1.4: 删除目录
  - 1.5: rename 文件
  - 1.6: rename 目录
  - 总结
- 场景 2: ls -l 查看文件属性
  - 2.1: 创建文件
  - 2.2: 修改文件属性 <不一致>
  - 2.3: 修改文件大小 <不一致>
  - 总结
- 场景 3: stat 查看文件属性
  - 3.1: 创建文件
  - 3.2: 删除文件
  - 3.3: rename 文件
  - 3.3: stat 后删除 <不一致>
  - 3.4: ls 刷新缓存后再 stat
  - 3.5: 缩短缓存时间再 stat
  - 3.6: 关闭 lookup 缓存再 stat
  - 3.7: rename 后 stat
  - 总结
- 场景 4: cat 读取文件
  - 4.1: 删除后 cat
  - 4.2: 缓存属性后 cat
  - 4.3: rename 后 cat
  - 4.4: chmod 后执行
  - 总结
- 场景 5: 混合场景
  - 5.1: 删除后创建同名目录
  - 5.2: 删除不存在文件
  - 总结
- 场景 6: 数据一致性
  - 6.1：读取文件
  - 6.2：修改内容后读取
  - 步骤：
  - 6.3: 轮流读写
  - 总结
- 场景 7: 性能测试
  - 7.1: 对比各文件系统
  - 7.2: 元数据性能 (ls)
  - 7.3：元数据性能 (stat)
  - 7.4：数据性能
  - 总结
- 总结

前言

关于测试目的

我们目前的方案是和 NFS v4 目前处理元数据和数据缓存是基本保持一致性的，测试及分析 NFS 可以帮助我们更好的了解各个场景下实际的效果和性能

关于测试维度

测试主要关注以下几方面：

- 文件可见性：即在一个客户端执行文件的创建、删除、重命名等操作，另外一个客户端要立马能看到
- 属性一致性：指一个文件的属性（如 atime, ctime, mtime, mode, size...）在一个客户端修改后，另外一个能立即看到
- 内容一致性：指用户在符合 close-to-open 的操作顺序下，数据得保证一致性

关于测试环境

- 我们在 nuc 机器上搭建 JuiceFS 集群和客户端，在 A, B 这 2 台机器分别挂载 NFS 客户端
- 为了让不一致性问题更容易暴露，我们特地将缓存超时时间（actimeo）调到 3600 秒

关于 NFS 实现

- NFS 在开启 ac 后会缓存 inode attribute，但是这个有超时时间 (actimeo) 过期后会从服务端重新校验，如果没有更改，仍然使用本地缓存，并将其在内核中重新设置 timeout (actimeo)
- NFS 在开启 lookupcache=positive 后，会保存 lookup 结果（即 dentry），
- 对于 opendir() 操作，NFS 无论如何都会向后服务端发送 getattr() 请求，来获取该目录的属性，通过该目录 mtime（或者 change_info）和本地缓存的目录 mtime 作对比，来判断该目录内容是否有更改（如在该目录下创建文件、删除文件、rename 文件），如果更改了则扔掉该目录下的全部缓存，重新从服务端获取，否则该目录下的所有文件将重新在内核中获取 actimeo 的超时时间
- 具体的相关使用及基本原理可参考：nfs(5) - Linux man page

关于测试结果

- 对于部分元数据缓存不一致性的情况，我们分析了其触发的场景，以及实际对业务的影响

场景 1: ls 可见性

```
mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs /mnt/nfs
```

1.1: 创建文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls /mnt/nfs/dir1/dir2
```

结果：f1 文件可见

```
root@B:~# ls /mnt/nfs/dir1/dir2
f1
```

1.2：创建目录

步骤：

```
A: mkdir /mnt/nfs/dir1/dir2/d1
B: ls /mnt/nfs/dir1/dir2
```

结果：d1 目录可见

```
root@B:~# ls /mnt/nfs/dir1/dir2
d1
```

1.3：删除文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls /mnt/nfs/dir1/dir2
A: rm /mnt/nfs/dir1/dir2/f1
B: ls /mnt/nfs/dir1/dir2
```

结果：最终 f1 文件不可见

© XXX

```
root@B:~# ls /mnt/nfs/dir1/dir2
```

1.4: 删除目录

步骤:

```
A: mkdir /mnt/nfs/dir1/dir2/d1
B: ls /mnt/nfs/dir1/dir2
A: rm -r /mnt/nfs/dir1/dir2/d1
B: ls /mnt/nfs/dir1/dir2
```

结果：最终 d1 目录不可见

```
root@B:~# ls /mnt/nfs/dir1/dir2
```

1.5: rename 文件

步骤:

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls /mnt/nfs/dir1/dir2
A: mv /mnt/nfs/dir1/dir2/f1 /mnt/nfs/dir1/dir2/f2
B: ls /mnt/nfs/dir1/dir2
```

结果：最终只可见 f2

```
root@B:~# ls /mnt/nfs/dir1/dir2
f2
```

© XXX

1.6: rename 目录

步骤：

```
A: mkdir /mnt/nfs/dir1/dir2/d1
B: ls /mnt/nfs/dir1/dir2
A: mv /mnt/nfs/dir1/dir2/d1 /mnt/nfs/dir1/dir2/d2
B: ls /mnt/nfs/dir1/dir2
```

结果：最终只可见 d2

```
root@B:~# ls /mnt/nfs/dir1/dir2
d2
```

总结

- ls 总是能看到目录下最新的文件列表，不管是创建，删除还是 rename

场景 2: ls -l 查看文件属性

> mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs /mnt/nfs

2.1: 创建文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
```

结果：f1 文件属性保持一致

```
root@B:~# ls /mnt/nfs/dir1/dir2
total 4
-rw-r--r-- 1 root root 3897 Feb 20 11:02 f1
```

2.2: 修改文件属性 <不一致>

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
A: chmod a+x /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
```

结果：f1 文件的 +x 属性未显示

```
root@B:~# ls -l /mnt/nfs/dir1/dir2
total 0
-rw-r--r-- 1 root root 0 Feb 20 19:09 f1
root@B:~# ls -l /mnt/nfs/dir1/dir2
total 0
-rw-r--r-- 1 root root 0 Feb 20 19:09 f1
```

注意：虽然显示的文件权限不对，但是不影响其执行，见以下 4.4 相关测试。

备注：该场景在我们的方案中已修复！（如果 JuiceFS 的做法）

2.3: 修改文件大小 <不一致>

步骤：

© XXX

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
A: echo "hello" > /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
```

结果：f1 文件可见

```
root@B:~# ls -l /mnt/nfs/dir1/dir2
total 0
-rw-r--r-- 1 root root 0 Feb 20 19:09 f1
root@B:~# ls -l /mnt/nfs/dir1/dir2
total 0
-rw-r--r-- 1 root root 0 Feb 20 19:09 f1
```

注意：虽然显示的文件大小不对，但是不影响读取，一旦读取，还是能读取到正确的内容，见以下场景 4 、场景 6相关测试。

备注：该场景在我们的方案中已修复!（如果 JuiceFS 的做法）

总结

- ls -l 总是能看到最新的目录的列表，但是文件属性因为有缓存，可能有延时（取决于用户配置的缓存时长）

场景 3：stat 查看文件属性

> mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs /mnt/nfs

3.1：创建文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：f1 文件属性正确

```
root@B:~# stat /mnt/nfs/dir1/dir2
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4bh/75d Inode: 201         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:09:34.518847433 +0000
Modify: 2023-02-20 11:09:34.518847433 +0000
Change: 2023-02-20 11:09:34.518847433 +0000
 Birth: -
```

3.2: 删除文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：stat f1 文件属性正确

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
stat: cannot stat '/mnt/nfs/dir1/dir2/f1': No such file or directory
```

3.3: rename 文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
A: mv /mnt/nfs/dir1/dir2/f1 /mnt/nfs/dir1/dir2/f2
B: stat /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f2
```

结果：f1 文件不可见，f2 文件属性正确

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
stat: cannot stat '/mnt/nfs/dir1/dir2/f1': No such file or directory
root@B:~# stat /mnt/nfs/dir1/dir2/f2
  File: /mnt/nfs/dir1/dir2/f2
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4bh/75d Inode: 206         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:25:42.442617602 +0000
Modify: 2023-02-20 11:25:42.442617602 +0000
Change: 2023-02-20 11:25:47.979584130 +0000
 Birth: -
```

3.3: stat 后删除 <不一致>

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 stat f1 依旧存在，存在不一致

© XXX

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4bh/75d Inode: 207         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:30:47.715743145 +0000
Modify: 2023-02-20 11:30:47.715743145 +0000
Change: 2023-02-20 11:30:47.715743145 +0000
 Birth: -
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4ch/76d Inode: 208         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:33:42.607287035 +0000
Modify: 2023-02-20 11:33:42.607287035 +0000
Change: 2023-02-20 11:33:42.607287035 +0000
 Birth: -
```

因为在 B 节点 stat 操作后，inode 已经缓存在内核中，所以即使另外一个节点删除了，还是能 stat 到。

- 但是 NFS 的实现是 ls 的时候会刷新缓存，所以 ls 后再 stat 即能看到最新的变化，参见下面 3.4 的测试用例
- 同样的，如果我们将缓存时长设的特别短，等缓存过期后，也可以看到最新的变化，参加下面 3.5 测试用例
- 或者，我们关系 lookup 缓存，可以看到最新的变化，参加下面 3.6 的测试用例

3.4: ls 刷新缓存后再 stat

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: ls /mnt/nfs/dir1/dir2
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 stat f1 文件不存在，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4ch/76d Inode: 210         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:38:32.145187883 +0000
Modify: 2023-02-20 11:38:32.145187883 +0000
Change: 2023-02-20 11:38:32.145187883 +0000
 Birth: -
root@B:~# ls /mnt/nfs/dir1/dir2
root@B:~# stat /mnt/nfs/dir1/dir2/f1
stat: cannot stat '/mnt/nfs/dir1/dir2/f1': No such file or directory
```

3.5: 缩短缓存时间再 stat

> B 挂载参数修改缓存时间为 1 秒：
>
> *mount -t nfs4 -o ac -o cto -o actimeo=1 -o lookupcache=positive 10.221.103.160:/mnt/jfs2 /mnt/nfs*

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 stat f1 文件不存在，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4bh/75d Inode: 207         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:30:47.715743145 +0000
Modify: 2023-02-20 11:30:47.715743145 +0000
Change: 2023-02-20 11:30:47.715743145 +0000
 Birth: -
root@B:~# stat /mnt/nfs/dir1/dir2/f1
stat: cannot stat '/mnt/nfs/dir1/dir2/f1': No such file or directory
```

3.6: 关闭 lookup 缓存再 stat

> B 挂载参数修改 lookupcache 为 none:
>
> ```
> mount -t nfs4 -o ac -o cto -o actimeo=3699 -o lookupcache=none 10.221.103.160:/mnt/jfs2 /mnt/nfs
> ```

步骤:

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 stat f1 文件不存在，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4bh/75d Inode: 207         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-20 11:30:47.715743145 +0000
Modify: 2023-02-20 11:30:47.715743145 +0000
Change: 2023-02-20 11:30:47.715743145 +0000
 Birth: -
root@B:~# stat /mnt/nfs/dir1/dir2/f1
stat: cannot stat '/mnt/nfs/dir1/dir2/f1': No such file or directory
```

3.7: rename 后 stat

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rename /mnt/nfs/dir1/dir2/f1 /mnt/nfs/dir1/dir2/f2
B: stat /mnt/nfs/dir1/dir2/f2
```

结果：第一次 stat f1 属性正确，第二次 stat f2 显示正确，存在不一致

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4ch/76d Inode: 213         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 02:30:48.952976196 +0000
Modify: 2023-02-21 02:30:48.952976196 +0000
Change: 2023-02-21 02:30:48.952976196 +0000
 Birth: -
root@B:~# stat /mnt/nfs/dir1/dir2/f2
  File: /mnt/nfs/dir1/dir2/f2
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4ch/76d Inode: 213         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 02:30:48.952976196 +0000
Modify: 2023-02-21 02:30:48.952976196 +0000
Change: 2023-02-21 02:31:06.604456187 +0000
 Birth: -
```

总结

在缓存 attribute 和 dentry 后，确实会出现一些场景不一致，而这些场景我认为对我们的业务几乎是没有影响的，主要因为：

- 相同的配置下，取决于用户的使用方法，不同的使用方法可能造成不一样的结果（有可能是一致的，也可能是不一致的），见 3.3 vs 3.4
- 而当用户确实出现这类情况时，我们可以通过将缓存时间调小来避免这类情况的发生，见 3.5
- 如果缓存调小仍不能避免（这类情况在实际应用中少之又少，可见 JuiceFS 的使用），我们可以选择关闭一部分缓存来获得一致性，见 3.6

所以我们从上面的测试可以看出，灵活配置带来的好处，我们可以一步步牺牲性能（长缓存 → 短缓存 → 关闭某一部分缓存），来获取一致性，以达到用户的实际需求。

场景 4: cat 读取文件

> mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs /mnt/n

4.1: 删除后 cat

步骤:

```
A: seq 1 3 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 cat 显示文件不存在，符合预期

```
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
2
3
root@B:~# cat /mnt/nfs/dir1/dir2/f1
cat: /mnt/nfs/dir1/dir2/f1: No such file or directory
```

4.2：缓存属性后 cat

步骤：

```
A: seq 1 3 > /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
```

结果：第一次 stat f1 属性正确，第二次 cat 显示文件不存在，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 6               Blocks: 1          IO Block: 1048576 regular file
Device: 4ch/76d Inode: 217         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 02:51:15.673075053 +0000
Modify: 2023-02-21 02:51:15.700538184 +0000
Change: 2023-02-21 02:51:15.700538184 +0000
 Birth: -
root@B:~# cat /mnt/nfs/dir1/dir2/f1
cat: /mnt/nfs/dir1/dir2/f1: No such file or directory
```

4.3: rename 后 cat

步骤：

```
A: seq 1 3 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: mv /mnt/nfs/dir1/dir2/f1 /mnt/nfs/dir1/dir2/f2
B: cat /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f2
```

结果：f1 不存在，f2 文件内容正常，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
1
2
3
root@B:~# cat /mnt/nfs/dir1/dir2/f1
cat: /mnt/nfs/dir1/dir2/f1: No such file or directory
root@B:~# cat /mnt/nfs/dir1/dir2/f2
123
```

4.4: chmod 后执行

步骤：

```
A: echo "hostname" > /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: chmod a+x /mnt/nfs/dir1/dir2/f1
B: bash /mnt/nfs/dir1/dir2/f1
```

结果：文件，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 9               Blocks: 1          IO Block: 1048576 regular file
Device: 4ch/76d Inode: 402         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 05:50:00.827721810 +0000
Modify: 2023-02-21 05:50:00.847376729 +0000
Change: 2023-02-21 05:50:00.847376729 +0000
 Birth: -
root@B:~# bash /mnt/nfs/dir1/dir2/f1
B
```

之所以能执行，是因为执行前需要读取文件内容，执行了 open() 操作，而 open() 操作会利用 ESTALE 机制刷新缓存，刷新后再次去 stat 就是最新的属性了

总结

因为 NFS 在 open 的时候会从服务端校验 inode，不一致时利用 ESTALE 会重新打开文件，所以可以保证 cat 在全部场景下（属性、内容）的一致性。

场景 5：混合场景

```
mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs  /mnt/n
```

5.1：删除后创建同名目录

步骤：

```
A: mkdir /mnt/nfs/dir1/dir2/d1
B: stat /mnt/nfs/dir1/dir2/d1
A: rm -r /mnt/nfs/dir1/dir2/d1
B: mkdir /mnt/nfs/dir1/dir2/d1
```

结果：stat 目录正常，mkdir 能正常创建目录，符合预期

```
root@B:~# stat /mnt/nfs/dir1/dir2/d1
  File: /mnt/nfs/dir1/dir2/d1
  Size: 4096            Blocks: 8          IO Block: 32768  directory
Device: 4ch/76d Inode: 219         Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 03:01:47.644806639 +0000
Modify: 2023-02-21 03:01:47.644806639 +0000
Change: 2023-02-21 03:01:47.644806639 +0000
 Birth: -
root@B:~# mkdir /mnt/nfs/dir1/dir2/d1
```

5.2：删除不存在文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: stat /mnt/nfs/dir1/dir2/f1
A: rm /mnt/nfs/dir1/dir2/f1
B: rm /mnt/nfs/dir1/dir2/f1
```

结果：stat 文件正常，rm 文件显示文件不存在，符合预期

© XXX

```
root@B:~# stat /mnt/nfs/dir1/dir2/f1
  File: /mnt/nfs/dir1/dir2/f1
  Size: 0               Blocks: 0          IO Block: 1048576 regular empty file
Device: 4ch/76d Inode: 222         Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-02-21 03:08:27.743277959 +0000
Modify: 2023-02-21 03:08:27.743277959 +0000
Change: 2023-02-21 03:08:27.743277959 +0000
 Birth: -
root@B:~# rm /mnt/nfs/dir1/dir2/d1
rm: cannot remove '/mnt/nfs/dir1/dir2/f1': No such file or directory
```

总结

这些混合场景下，如 mkdir/rmdir/unlink，即使有缓存也会下发相应的请求到 NFS client，而此时 client 会请求服务端，以服务端的响应为主，所以一致性都是可以保证的

场景 6：数据一致性

> mount -t nfs4 -o ac -o cto -o actimeo=3600 -o lookupcache=positive 10.221.103.160:/mnt/nfs /mnt/n

6.1：读取文件

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: ls -l /mnt/nfs/dir1/dir2
A: seq 1 3 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
```

结果：可以正确读到 f1 文件

```
root@B:~# ls /mnt/nfs/dir1/dir2
total 0
-rw-r--r-- 1 root root 0 Feb 21 02:33 f1
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
2
3
```

6.2：修改内容后读取

步骤：

```
A: touch /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: seq 1 3 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: seq 1 5 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: seq 1 1 > /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
```

结果：每次都能正确读到文件内容

```
root@B:~# cat /mnt/nfs/dir1/dir2
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
2
3
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
2
3
4
5
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
```

6.3：轮流读写

步骤：

```
A: echo "1" >> /mnt/nfs/dir1/dir2/f1
A: cat /mnt/nfs/dir1/dir2/f1
B: echo "2" >> /mnt/nfs/dir1/dir2/f1
B: cat /mnt/nfs/dir1/dir2/f1
A: echo "3" >> /mnt/nfs/dir1/dir2/f1
A: cat /mnt/nfs/dir1/dir2/f1
```

结果：f1 文件内容正常

```
root@A:~# cat /mnt/nfs/dir1/dir2/f1
1
root@B:~# cat /mnt/nfs/dir1/dir2/f1
1
2
root@A:~# cat /mnt/nfs/dir1/dir2/f1
1
2
3
```

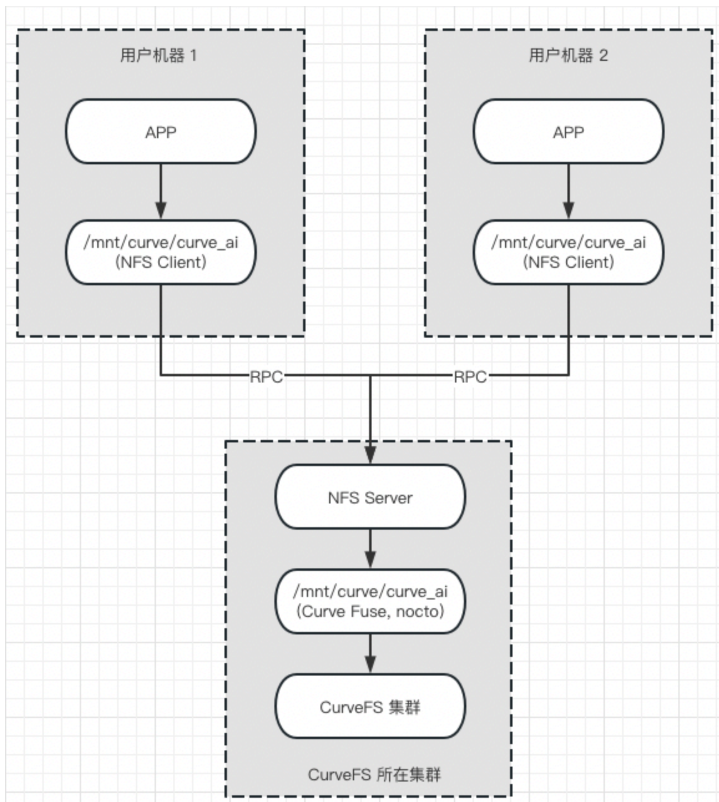
总结

NFS 在数据一致性方面表现非常好，只要你符合 close-to-open 的操作，总能看到一致的数据。

场景 7：性能测试

7.1：对比各文件系统

© XXX



> ls 测试都为单个目录下创建 100 个目录

部署方案	耗时（pipestream）	耗时（特征提取任务）	耗时（ls）	耗时（ls -l）	耗时（删一文件后 ls）	耗时（删一文件后 ls -l）
CurveFS (cto)	573 ms	16 m 23 s	74 ms	931 ms	79 ms	965 ms
CurveFS (nocto)	114 ms		22 ms	35 ms	20 ms	32 ms
JuiceFS (redis)	50 ms		7 ms	10 ms	7 ms	10 ms
CephFS (fuse)	50 ms		6 ms	42 ms	15 ms	44 ms
CurveFS (cto) + NFS	16 ms	9 m 32 s	3 ms	3 ms	178 ms	215 ms
CurveFS (nocto) + NFS	16 ms		2 ms	3 ms	5 ms	14 ms
CephFS (kernel)	16 ms		2 ms	3 ms	7 ms	9 ms

7.2: 元数据性能 (ls)

前置条件：

- 2 个节点（A、B） 通过 NFS 挂载同一个 CurveFS（nocto)
- 在 A 节点根目录下创建 100 个文件
- A 节点不断 ls 根目录，并查看其耗时
- B 节点在根目录下做不同动作，如创建文件、删除文件、写入文件、读取文件

在 B 节点上操作	观测 A 节点耗时
新建一文件	目录缓存全部失效，耗时 9ms → 41ms
删除一文件	目录缓存全部失效，耗时 9ms → 41ms
rename 一文件	目录缓存全部失效，耗时 9ms → 48ms
往一存在文件不断写入新内容	目录缓存不变，耗时 9ms → 9ms

7.3：元数据性能（stat)

前置条件：

- 2 个节点（A、B） 通过 NFS 挂载同一个 CurveFS（nocto)
- 在 A 节点用 watch 每隔一秒统计 stat f1 的耗时
- 在 B 节点不断在创建文件（约定创建的文件名依次为 f1, f2, f3...）
- 以下图表代表在 B 节点创建相应文件后，A 节点 stat f1 文件的耗时

B 节点创建文件	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
A 节点 stat f1 文件耗时 (ms)	29	4	2	3	2	4	5	4	4	3

结论：创建文件不会导致已有的文件元数据缓存失效（通过抓取 CurveFS Fuse 请求也可以确定），但是 NFS 自身的缓存超时可能会导致其失效 （默认 60 s）

7.4：数据性能

前置条件：

- 2 个节点（A、B） 通过 NFS 挂载同一个 CurveFS（nocto)
- 在 A 节点不断读取根目录下的所有文件，并统计读取各个文件的耗时
- 在 B 节点不断在根目录下创建文件，并写入 1 MB 数据（约定创建的文件名依次为 f1, f2, f3...）

- 以下图表代表在 B 节点创建相应文件后，A 节点读取各个文件的耗时

B 节点创建文件	A 节点依次读取各文件：f1	f2	f3	f4	f5
f1	246 ms				
f2	131 ms	279 ms			
f3	111 ms	131 ms	220 ms		
f4	106 ms	141 ms	113 ms	324 ms	

f5	153 ms	119 ms	117 ms	121 ms	219 ms

```
root@debian10-001:/mnt/nfs_sharedir/dir# mkdir d1 d2 d3
root@debian10-001:/mnt/nfs_sharedir/dir# ls
d1  d2  d3
root@debian10-001:/mnt/nfs_sharedir/dir# rm -r d1
root@debian10-001:/mnt/nfs_sharedir/dir# ls
d2  d3
root@debian10-001:/mnt/nfs_sharedir/dir# ls d1
ls: cannot access 'd1': No such file or directory
root@debian10-001:/mnt/nfs_sharedir/dir#
```

```
root@debian10-002:/mnt/nfs_sharedir/dir# ls
d1  d2  d3
root@debian10-002:/mnt/nfs_sharedir/dir# ls
d2  d3
root@debian10-002:/mnt/nfs_sharedir/dir# ls d1
ls: cannot access 'd1': No such file or directory
root@debian10-002:/mnt/nfs_sharedir/dir#
```

```
root@debian10-001:/mnt/nfs_sharedir/dir# seq 1 3 > f1
root@debian10-001:/mnt/nfs_sharedir/dir# seq 1 3 > f2
root@debian10-001:/mnt/nfs_sharedir/dir# cat f1
1
2
3
root@debian10-001:/mnt/nfs_sharedir/dir# cat f2
1
2
3
root@debian10-001:/mnt/nfs_sharedir/dir# cat f1
Write from 002
root@debian10-001:/mnt/nfs_sharedir/dir# cat f2
Write from 002
root@debian10-001:/mnt/nfs_sharedir/dir#
```

```
root@debian10-002:/mnt/nfs_sharedir/dir# echo "Write from 002" > f1
root@debian10-002:/mnt/nfs_sharedir/dir# echo "Write from 002" > f2
root@debian10-002:/mnt/nfs_sharedir/dir# cat f1
Write from 002
root@debian10-002:/mnt/nfs_sharedir/dir# cat f2
Write from 002
root@debian10-002:/mnt/nfs_sharedir/dir#
```

总结

NFS 对于这些没改动的文件，充分利用缓存来加速读取

总结

从上面的测试情况来看：

- 文件可见性：NFS 可以保证文件可见性的一致性
- 属性一致性：大部分保持一致性，在一些场景下（见 3.3）属性存在延时，属于弱一致性，而这类情况触发跟用户使用情况及配置有关。当然 NFS 提供了丰富的配置，如果用户有需要，可根据实际业务情况调整配置，牺牲一部分性能来保证一致性（见场景 3 测试总结）。另外从我们上面的测试用例可以看到，虽然 ls 看到不一致，但是其实并不影响其实际使用，见 (测试用例 2.2, 2.3, 4.4)
- 内容一致性：NFS 可以保证只要符合 close-to-open 的操作，所有场景下的内容一致性

另外，我们可以发现，对于 NFS 来说，除非某个系统调用只会下发 lookup/getattr 这 2 个请求到 client（如 stat），这类情况由于 VFS 缓存会导致 NFS Client 无法接受到任何请求，导致不一致。而对于其他任何 Client 可以接收到的请求（如 open/opendir/mkdir/rmdir/unlink...），NFS 都可以在这一层做动作来保证一致性，如：

- opendir 时通过 getattr 向服务端做校验
- open 是通过 getattr 向服务端校验后返回 ESTALE 促使 VFS 重新下发 open
- mkdir/rmdir/unlink/mknod 这些本来就要向服务端操作元数据的来说，直接以服务端的响应为主
- 而这些操作相当于一次缓存刷新的操作，之前不一致的请求在执行这些请求后，缓存就是最新的一致的，见 4.4 测试用例