NFS Delegations

Table of Contents

This is a demonstration of how NFS v4 read delegations operate

Delegations

When the client opens a file, the nfs v4 server has the option of providing a delegation to the client for that particular open file. When the server grants a delegation, the server guarantees the client has full control over the file (according to the parameters of the delegation), and the client assumes management of the file. The client no longer needs to communicate with the server before taking various actions. This greatly reduces the interactions between the client & server, reducing network traffic, and increasing performance.

The server may choose not to grant a delegation, or may recall the delegation at any time, at which point the client must stop assuming full control over the file, and must resume interacting with the server as previously.

When does an nfs server grant a delegation

Speaking for the Linux nfs v4 server in RHEL 7 and newer, the nfs server may grant a read delegation when the client opens a file, the file has not been opened for writing by any clients, and the server is configured to permit file leases.

File leases are configured through the sysctl **fs.leases-enable**. When set to '1' at the time that the nfs client first establishes a session with the server, the client and server will negotiate a backchannel for the server to initiate communications back to the client, if necessary.

```
16  2019-10-12 17:32:18.200013125 192.168.122.60 → 192.168.122.71 NFS 434 V4 Call
(Reply In 18) EXCHANGE_ID
18  2019-10-12 17:32:18.200264649 192.168.122.71 → 192.168.122.60 NFS 286 V4 Reply
(Call In 16) EXCHANGE_ID
21  2019-10-12 17:32:18.200831332 192.168.122.60 → 192.168.122.71 NFS 350 V4 Call
(Reply In 22) CREATE_SESSION
    nfs.create_session.flags.conn_back_chan == 1
22  2019-10-12 17:32:18.201186973 192.168.122.71 → 192.168.122.60 NFS 262 V4 Reply
(Call In 21) CREATE_SESSION
    nfs.create_session.flags.conn_back_chan == 1
```

What does this demonstration show?

This particular demonstration mounts the filesystem using the 'noac' option, which prevents the client from caching file attributes. Ordinarily, the client will cache file attributes, and will perform a check every few seconds to see whether the nfs server’s version of each file’s attributes has had any updates. Because of this, the client may not know about changes to the file which occur during that small interval. Specifying 'noac' requires the client to perform these checks before every operation, so that attribute changes are detected immediately. This causes a significant performance penalty.

This demonstration shows how that performance penalty can be mitigated when read delegations can be used.

Example delegation using packet capture

The demonstration environment includes 2 nfs clients, both mounting the same nfs export using nfs version 4.1.

```
[root@client1 ~]# mount | grep server
server:/exports on /mnt/server type nfs4
(rw,relatime,sync,vers=4.1,rsize=524288,wsize=524288,namlen=255,acregmin=0,acregmax=0,
acdirmin=0,acdirmax=0,hard,noac,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.1
68.122.60,local_lock=none,addr=192.168.122.71)
```

```
[root@client2 ~]# mount | grep server
server:/exports on /mnt/server type nfs4
(rw,relatime,sync,vers=4.1,rsize=524288,wsize=524288,namlen=255,acregmin=0,acregmax=0,
acdirmin=0,acdirmax=0,hard,noac,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.1
68.122.73,local_lock=none,addr=192.168.122.71)
```

on 'client1', open a file read-only; the server grants a read delegation:

```
[root@client1 ~]# exec 10</mnt/server/trace.dat
```

```
121  27.950367616      client1 → server       NFS 350 V4 Call (Reply In 122) SEQUENCE
| PUTFH | OPEN DH: 0x8e2272a5/trace.dat | GETFH | ACCESS FH: 0x00000000, [Check: RD MD
XT XE] | GETATTR
                       rpc.xid == 0x75438f20
                   nfs.fh.hash == 0xde61574d
                   nfs.fh.hash == 0x8e2272a5
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_READ
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
           nfs.open.claim_type == CLAIM_NULL
             nfs.open.opentype == OPEN4_NOCREATE
122  27.950603133       server → client1      NFS 474 V4 Reply (Call In 121) SEQUENCE
| PUTFH | OPEN StateID: 0xc262 | GETFH | ACCESS, [Access Denied: XE], [Allowed: RD MD
XT] | GETATTR
                       rpc.xid == 0x75438f20
                   nfs.fh.hash == 0x8e2272a5
                   nfs.fh.hash == 0xde61574d
        nfs.open.delegation_type == OPEN_DELEGATE_READ
```

on 'client1', run 'dd' to open a file, read its contents, and close the file again:

```
[root@client1 ~]# dd if=/mnt/server/trace.dat of=/dev/null
14128+1 records in
14128+1 records out
7233628 bytes (7.2 MB) copied, 0.0870879 s, 83.1 MB/s
```

Because the nfs client already had a read delegation, and because 'dd' was opening the file read-only again, the client did not need to communicate with the server in order to do the file 'OPEN' or 'CLOSE' operations when opening or closing the file. Only the 'READ' calls and replies were exchanged between the client and server. In addition, due to the delegation, the client did not need to check the file attributes prior to each read.

on 'client2', open the file read-only:

```
[root@client2 ~]# exec 10</mnt/server/trace.dat
```

the nfs server also grants a read delegation to 'client2':

```
1387  63.374995537      client2 → server       NFS 350 V4 Call (Reply In 1388)
SEQUENCE | PUTFH | OPEN DH: 0x8e2272a5/trace.dat | GETFH | ACCESS FH: 0x00000000,
[Check: RD MD XT XE] | GETATTR
                       rpc.xid == 0xfb8f09da
                   nfs.fh.hash == 0xde61574d
                   nfs.fh.hash == 0x8e2272a5
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_READ
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
           nfs.open.claim_type == CLAIM_NULL
             nfs.open.opentype == OPEN4_NOCREATE
1388  63.375491192       server → client2      NFS 474 V4 Reply (Call In 1387)
SEQUENCE | PUTFH | OPEN StateID: 0xdfd3 | GETFH | ACCESS, [Access Denied: XE],
[Allowed: RD MD XT] | GETATTR
                       rpc.xid == 0xfb8f09da
                   nfs.fh.hash == 0x8e2272a5
                   nfs.fh.hash == 0xde61574d
        nfs.open.delegation_type == OPEN_DELEGATE_READ
```

run 'dd' on 'client2' as well:

```
[root@client2 ~]# dd if=/mnt/server/trace.dat of=/dev/null
14128+1 records in
14128+1 records out
7233628 bytes (7.2 MB) copied, 0.12299 s, 58.8 MB/s
```

again, because the client had a read delegation, the 'OPEN' and 'CLOSE' operations were not communicated with the server, and the client did not check file attributes prior to each 'READ' operation.

Now, on 'client2', open the file in a read-write mode:

```
[root@client2 ~]# exec 11<>/mnt/server/trace.dat
```

Because the client knows that opening the file read-write will conflict with its read delegation, the client calls 'DELEGRETURN' to inform the server that it will no longer use the delegation:

```
1819  81.896349549      client2 → server       NFS 290 V4 Call (Reply In 1820)
SEQUENCE | PUTFH | GETATTR FH: 0xde61574d | DELEGRETURN StateID: 0xc368
                       rpc.xid == 0x0d9009da
                   nfs.fh.hash == 0xde61574d
1820  81.896668788       server → client2      NFS 230 V4 Reply (Call In 1819)
SEQUENCE | PUTFH | GETATTR | DELEGRETURN
                       rpc.xid == 0x0d9009da
                   nfs.fh.hash == 0xde61574d
```

After returning the delegation, 'client2' then makes an 'OPEN' call to open the file in read-write mode:

```
1821  81.897544067      client2 → server       NFS 338 V4 Call (Reply In 1822)
SEQUENCE | PUTFH | OPEN DH: 0xde61574d/ | ACCESS FH: 0xde61574d, [Check: RD MD XT XE]
| GETATTR FH: 0xde61574d
                       rpc.xid == 0x0e9009da
                   nfs.fh.hash == 0xde61574d
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_BOTH
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
           nfs.open.claim_type == CLAIM_FH
             nfs.open.opentype == OPEN4_NOCREATE
```

Because the nfs server has also given a read delegation to 'client1', the server must first recall the delegation from 'client1' before allowing 'client2' to open the file with a mode that permits writing. The server therefore responds to the 'OPEN' call with 'NFS4ERR_DELAY' to instruct 'client2' to wait briefly before making the call again:

```
1822  81.897906741       server → client2      NFS 166 V4 Reply (Call In 1821)
SEQUENCE | PUTFH | OPEN Status: NFS4ERR_DELAY
                       rpc.xid == 0x0e9009da
                   nfs.fh.hash == 0xde61574d
```

The server next informs 'client1' that it must return its read delegationa. For this, it makes a 'CB_RECALL' call using the rpc program named 'NFS_CB', rather than making a call using the 'NFS' program. The nfs client acknowledges that it has received the message that it must return the delegation:

```
1823  81.898057606       server → client1      NFS CB 278 V1 CB_COMPOUND Call (Reply
In 1824) <EMPTY> CB_SEQUENCE;CB_RECALL
                       rpc.xid == 0xdd4fbbb4
                   nfs.fh.hash == 0xde61574d
1824  81.898703209      client1 → server       NFS CB 154 V1 CB_COMPOUND Reply (Call
In 1823) <EMPTY> CB_SEQUENCE;CB_RECALL
                       rpc.xid == 0xdd4fbbb4
                   nfs.fh.hash == 0xde61574d
```

Note that the nfs callback is initiating a call back to the client, not responding to a call from the client; when using nfs v4.0, this callback will over a separate tcp port, negotiated at the time that the session was initially set up. Beginning with nfs v4.1, the existing tcp port 2049 connection is reused for the backchannel communication.

As instructed, 'client1' now returns its read delegation:

```
1825  81.898707139      client1 → server       NFS 290 V4 Call (Reply In 1827)
SEQUENCE | PUTFH | GETATTR FH: 0xde61574d | DELEGRETURN StateID: 0xded9
                       rpc.xid == 0x89438f20
                   nfs.fh.hash == 0xde61574d
1827  81.899046375       server → client1      NFS 230 V4 Reply (Call In 1825)
SEQUENCE | PUTFH | GETATTR | DELEGRETURN
                       rpc.xid == 0x89438f20
                   nfs.fh.hash == 0xde61574d
```

After 'client2' received the 'NFS4ERR_DELAY' response, it slept briefly. It now reattempts the 'OPEN' call to open the file read-write:

```
1830  82.001078714      client2 → server       NFS 338 V4 Call (Reply In 1831)
SEQUENCE | PUTFH | OPEN DH: 0xde61574d/ | ACCESS FH: 0xde61574d, [Check: RD MD XT XE]
| GETATTR FH: 0xde61574d
                       rpc.xid == 0x0f9009da
                   nfs.fh.hash == 0xde61574d
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_BOTH
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
           nfs.open.claim_type == CLAIM_FH
             nfs.open.opentype == OPEN4_NOCREATE
1831  82.001388279       server → client2      NFS 386 V4 Reply (Call In 1830)
SEQUENCE | PUTFH | OPEN StateID: 0xd124 | ACCESS, [Access Denied: XE], [Allowed: RD MD
XT] | GETATTR
                       rpc.xid == 0x0f9009da
                   nfs.fh.hash == 0xde61574d
        nfs.open.delegation_type == OPEN_DELEGATE_NONE
```

The server responded that the 'OPEN' was successful, but gives no delegation to the client.

Because it no longer has a delegation in place, when 'client1' runs the 'dd' command again, the 'dd' progresses much more slowly, since the client has to regularly verify that the file attributes have not changed:

```
[root@client1 ~]# dd if=/mnt/server/trace.dat of=/dev/null
14128+1 records in
14128+1 records out
7233628 bytes (7.2 MB) copied, 4.66734 s, 1.5 MB/s
```

When 'client2' closes the read-write file, the client simply reduces its access level to read-only, using an 'OPEN_DOWNGRADE' call:

```
30105  120.627189249      client2 → server       NFS 286 V4 Call (Reply In 30106)
SEQUENCE | PUTFH | OPEN_DOWNGRADE
                       rpc.xid == 0x109009da
                   nfs.fh.hash == 0xde61574d
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_READ
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
30106  120.627525362       server → client2      NFS 182 V4 Reply (Call In 30105)
SEQUENCE | PUTFH | OPEN_DOWNGRADE
                       rpc.xid == 0x109009da
                   nfs.fh.hash == 0xde61574d
```

Note that neither 'client1' nor 'client2' is given a read delegation again; this is done at the time of an 'OPEN' call.

on 'client2', close the file which was open read-only:

```
[root@client2 ~]# exec 10>&-
```

```
30108  124.662539565      client2 → server       NFS 294 V4 Call (Reply In 30109)
SEQUENCE | PUTFH | GETATTR FH: 0xde61574d | CLOSE StateID: 0xd489
                       rpc.xid == 0x119009da
                   nfs.fh.hash == 0xde61574d

30109  124.662875472       server → client2      NFS 246 V4 Reply (Call In 30108)
SEQUENCE | PUTFH | GETATTR | CLOSE
                       rpc.xid == 0x119009da
                   nfs.fh.hash == 0xde61574d
```

on 'client1', close the file as well:

```
[root@client1 ~]# exec 10>&-
```

```
58386  146.150172048      client1 → server       NFS 294 V4 Call (Reply In 58387)
SEQUENCE | PUTFH | CLOSE StateID: 0xc262 | GETATTR FH: 0xde61574d
                       rpc.xid == 0xf6b18f20
                   nfs.fh.hash == 0xde61574d
58387  146.150582719       server → client1      NFS 246 V4 Reply (Call In 58386)
SEQUENCE | PUTFH | CLOSE | GETATTR
                       rpc.xid == 0xf6b18f20
                   nfs.fh.hash == 0xde61574d
```

On 'client1', run the 'dd' again:

```
[root@client1 ~ ]# dd if=/mnt/server/trace.dat of=/dev/null
14128+1 records in
14128+1 records out
7233628 bytes (7.2 MB) copied, 0.0465882 s, 155 MB/s
```

```
58398  151.263072172      client1 → server       NFS 338 V4 Call (Reply In 58399)
SEQUENCE | PUTFH | OPEN DH: 0xde61574d/ | ACCESS FH: 0xde61574d, [Check: RD MD XT XE]
| GETATTR FH: 0xde61574d
                       rpc.xid == 0xfbb18f20
                   nfs.fh.hash == 0xde61574d
        nfs.open4.share_access == OPEN4_SHARE_ACCESS_READ
          nfs.open4.share_deny == OPEN4_SHARE_DENY_NONE
           nfs.open.claim_type == CLAIM_FH
             nfs.open.opentype == OPEN4_NOCREATE
58399  151.263419750       server → client1      NFS 422 V4 Reply (Call In 58398)
SEQUENCE | PUTFH | OPEN StateID: 0xfb14 | ACCESS, [Access Denied: XE], [Allowed: RD MD
XT] | GETATTR
                       rpc.xid == 0xfbb18f20
                   nfs.fh.hash == 0xde61574d
        nfs.open.delegation_type == OPEN_DELEGATE_READ
```

Note that the server has granted a read delegation again, and the file read progresses quickly again.

Because there are no additional file handles open for the file, 'client1' closes the file immediately:

```
58401  151.309305617      client1 → server       NFS 278 V4 Call (Reply In 58402)
SEQUENCE | PUTFH | CLOSE StateID: 0xfb14
                       rpc.xid == 0xfcb18f20
                   nfs.fh.hash == 0xde61574d
58402  151.309690377       server → client1      NFS 182 V4 Reply (Call In 58401)
SEQUENCE | PUTFH | CLOSE
                       rpc.xid == 0xfcb18f20
                   nfs.fh.hash == 0xde61574d
```

Conclusions from the demonstration

When an nfs client is granted a read delegation, it has a guarantee from the server that the file will not change; it is therefore unnecessary for the client to perform any checking of the file attributes. This dramatically improves the client performance.

After the client had to return its delegation, it needed to constantly check the file attributes, significantly reducing the overall transfer rate.

Note that the benefits provided by a read delegation are not always this dramatic. Mounting with 'noac' shows an extreme, worst-case scenario.