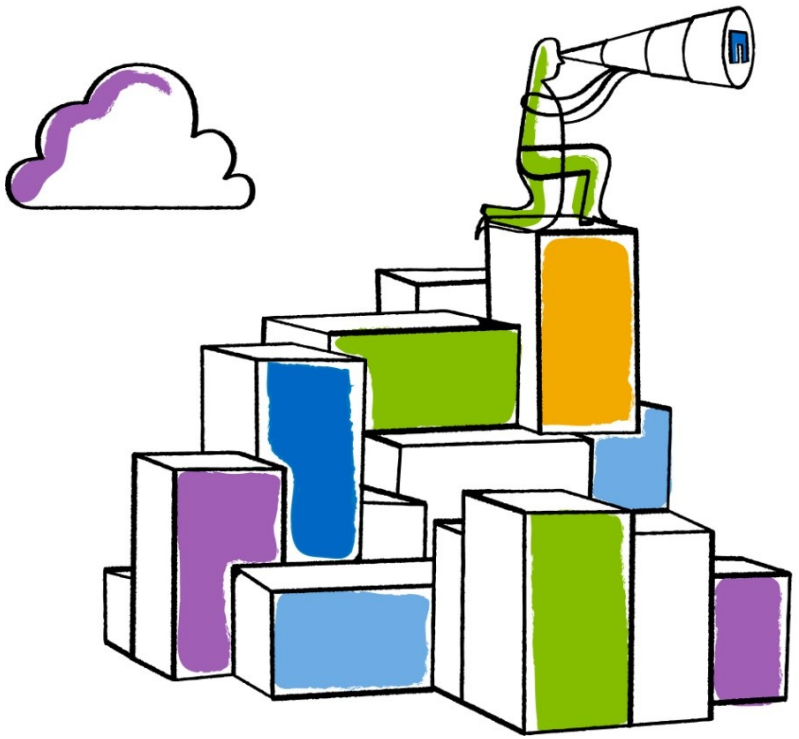
NetApp®

Go further, faster®



NFS v4.2 Prototyping

Anna Schumaker
anna.schumaker@netapp.com

Overview

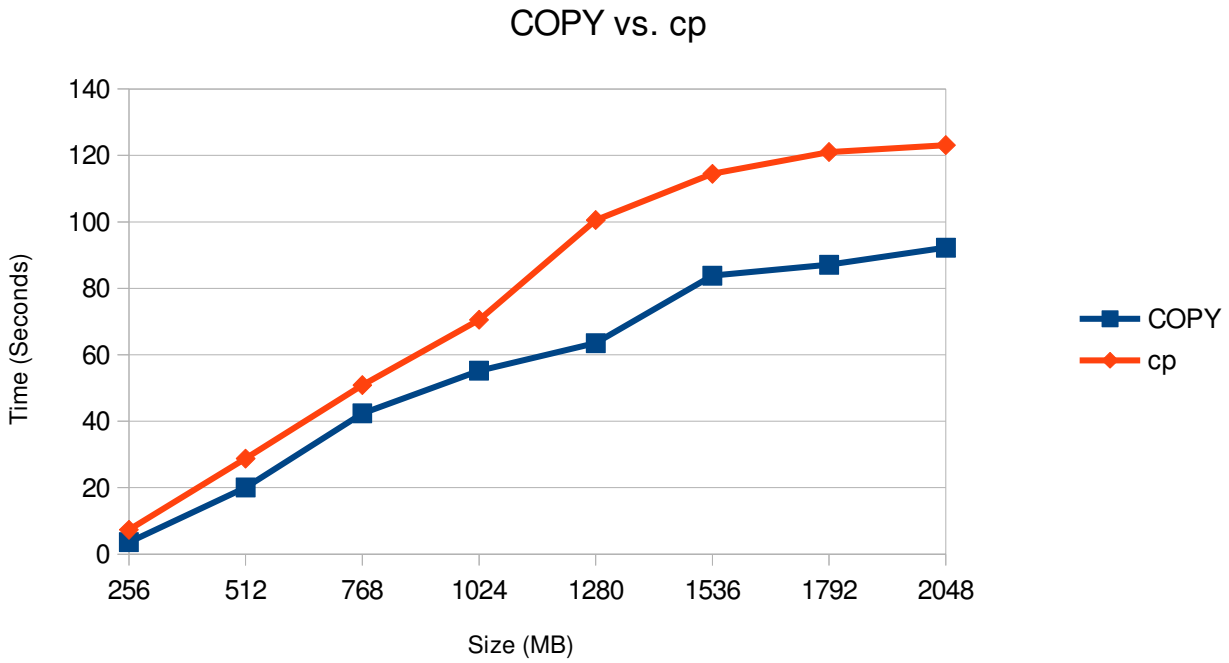
- Wrote Linux client and server code for
  - COPY
  - SEEK
  - WRITE_PLUS
  - READ_PLUS

Copy

- Single server (intra-server) patches have been written
  - Andy is looking at a server-to-server (inter-server) implementation
- No Linux VFS-level interface yet
  - Waiting on Zack Brown (Red Hat) to finish copy syscall
- Async implemented poorly
  - Code is buggy and not recently updated
  - Current state makes my server crash

Copy – Testing

- Wrote a python program (nfscopy.py) to call the splice() syscall
- Use `dd` to create files with random content (if=/dev/urandom)
  - Call either cp or nfscopy.py to perform the copy
  - Rebooted machines to copy with a cold cache



Copy – Spec Issues

- Argument for creating destination file as part of the COPY
  - But how to get a destination filehandle or stateid?
  - Feature was removed after discussion
- Server does not return the sequential number of bytes copied
  - If there is an error or the server decides to limit the copy size then the client has to copy the entire range again

Copy – Lesson Learned

- Clients may want to break copies into smaller chunks
  - If there is a problem then there will be a smaller range to retry
  - If the server copies synchronously then an RPC slot won't be tied up

Seek

- Server calls the VFS-level lseek and encodes the result
- Ranges of zeros in a data section are not seen as a hole
  - SEEK_DATA → di_allocated == true
  - SEEK_HOLE → di_allocated == false
- Client should be able to preemptively cache holes for READ_PLUS
  - I do not have code for this (… yet)

Seek - Testing

- Tested using xfstests #285 (seek sanity check)
- Does not pass Test 10: Testing a huge file for offset overflow
  - I expect I need to do a better job checking arguments on the server

```
10. Test a huge file for offset overflow
10.01 SEEK_HOLE expected 1048576 or 0, got 8588886016.           FAIL
10.02 SEEK_HOLE expected 1048576 or 0, got 8588886016.           FAIL
10.03 SEEK_DATA expected 0 or 0, got 0.                          succ
10.04 SEEK_DATA expected 1 or 1, got 1.                          succ
10.05 SEEK_HOLE expected 8588886016 or 0, got 8588886016.        succ
10.06 SEEK_DATA expected 8587837440 or 8587837440, got 8587837440. succ
10.07 SEEK_DATA expected 8587837441 or 8587837441, got 8587837441. succ
10.08 SEEK_DATA expected 8587837440 or 8587837440, got 8586788864. FAIL
```

Seek – Potential performance problem

- Each SEEK operation is really two seeks
  - One to find offset
  - One to find length
- lseek only cares about offset
- This could be slow depending on underlying filesystem implementation

Write Plus

- Only implemented the CONTENT_HOLE arm
- Only wrote sync version
  - Async would follow same codepath as async COPY

Write Plus - Testing

- Tested using /usr/bin/fallocate to create a 30G sparse file with 1MB data at beginning and end
- Hole punching:
  - `fallocate -o 1048576 -l 32212254720 -p /nfs/test.file`
  - 0.008 total
- Zero range instead:
  - `fallocate -o 1048576 -l 32212254720  /nfs/test.file`
  - 0.158 total

Write Plus – Spec Issue

- Christoph Hellwig suggested creating an ALLOCATE operation
  - Decouples hole punching and preallocation
  - No discussion since late November

Read Plus

- Linux server only supports one section of XDR pages
  - Means we can only encode one data section in a reply
- Can reply to call with:
  - <HOLE>
  - <DATA>
  - <HOLE><DATA>
  - <DATA><HOLE>
  - <HOLE><DATA><HOLE>
- Noticed huge performance hit when not caching holes
  - Client will try to read one memory page at a time after the initial result

Read Plus - Testing

- Create files with various hole and data segments
- Python script to read file and find zeros

```
File[     0]: Hole length: 65536
File[ 65536]: Text data: Test data to check caching
File[ 65562]: Hole length: 196608
File[262170]: EOF
```

```
File[     0]: Hole length: 65536
File[ 65536]: b'Test data to check caching' (26 bytes)
File[ 65562]: Hole length: 196608
File[262170]: EOF
```

Read Plus – Spec issue

- I did not have any issues while implementing READ_PLUS
- Christoph suggested a few edits
  - NFS4ERR_UNION_NOTSUPP is not a valid error code

Next Steps

- Error recovery has not been implemented
- Async client (and server?) needs some work
- Bruce won't merge server code until the spec is finished
  - I expect the same for client side code

NetApp

Thank you