Flex Files v2: Protection Types in NFSv4.2

IETF 120, NFSv4 WG

Tom Haynes July 26th, 2024

RFC8435

Focus of Flex Files v1

- Separating metadata from data
  - Leveraging NFSv3 data servers
  - Pointers to data
  - Client-Side Mirroring

draft-haynes-nfsv4-flexfiles-v2

Focus of Flex Files v2

- Still leverage Flex Files format over Files format
- Protection Types:
  - Client-Side Mirroring
  - Client-Side Erasure Encoding
    - Mojette Transformation
  - *Client-Side Reed Solomon*
    - *Will need help from WG*

Issues to tackle

- Security
- Encoding Protection Types
- Projection Header
- WRITE Holes

Security

- Krb5
  - Avoided it in v1 because how do we extend trust with NFSv3
  - Could have a per cluster ticket server
    - Cluster is metadata and data servers
- TLS
  - Wasn’t a thing in v1
  - Can extend NFSv4.2 to allow registration of proxy server

Encoding Protection Types

- Separate Layout Type for each Protection Type
  - Cumbersome
- New IANA Registry for Protection Types
  - Will possibly need an equivalent to layout_content4
  - Enough information to allow interoperability
  - Avoid opaque

Projection Header

- READ and WRITE payload for Mojette are data files
  - 4k block size because size of Linux page
- Payload contains a projection header
  - Fields are not aligned in 4 byte boundaries
  - Endianness
    - Record in metadata file
  - Performance cost to pack in XDR

mojette_projection_header

- //
- // The first 16 bytes of data<> in the READ and WRITE requests
- // are the projection header. As they are part of an opaque,
- // they need to be interpreted in the endianness of the data.
- //
- // A C structure describing the projection header is:
- //
- // struct mojette_projection_header {
- // uint64_t cinfo; /* The change info describing the first write */
- // uint16_t effective_len; /* effective len of user block within a 4kb block */
- // uint8_t id; /* Identifier of projection */
- // uint8_t version; /* version id - how to specify different ones? */
- // uint32_t crc_32c; /* crc of the payload at the file offset */
- //};
- //
-

PNFS_OSD_RAID_PQ

- What did it do for the same type of data files?

WRITE Holes

- Assume that a file already has data Old at block N
- Client writes new data New at block N
- Some of the data files get an error which is not sufficient to allow data reconstruction
  - With 8 projections
    - 4 Old
    - 4 New
- How to fix the file to a consistent view?
- What if the client goes down?

Repair the file

- Need to have an API to fix the WRITE Hole
- Provide this as an extension to NFSv4.2

Stage Writes

Possible Implementation

- Write to a temporary file
- When the contents are verified, overwrite blocks of old file with those of new file
  - 2 phase
    - What happens if error on 2nd Phase?
- API to allow different implementations