pNFS for Block Storage

Sridhar Balasubramanian, Stanley Skelton

HSG Advanced Development

Agenda

- pNFS Background Info
- pNFS Block Storage Access
  - RFC5663
  - SCSI Layout type
  - pNFS Block volume topology
  - Protocol operations
- pNFS Server
  - H/L Architecture Overview
  - Client Fencing
  - HA support
- Next Steps

pNFS Background Information

- Parallel Data storage foundations are built into NFSv4.1
  - Supports Global namespace
- Head and storage scaling capabilities
  - High availability through Client and Server lease management w/ failover
- Multiple read, write, delete operations per RPC call
- Delegate locks, read and write procedures to clients
  - File delegations allow client workloads for single writer and multiple reader scheme
- Supports storage layout types: Files (NFSv4.1), Blocks (SCSI), Objects (OSD T10)

pNFS Block Storage Access

RFC 5663 – pNFS for Block Storage

- Defines additional extensions (primarily data structures) for use of pNFS with block- and volume-based storage
- Enables direct access from pNFS clients to storage systems over storage protocols based on blocks and volumes
  - For example, SCSI protocol family including Fibre Channel Protocol, iSCSI, IB, and FCoE

pNFS Block Storage Access Model

- Server's filesystem is built from one or more block storage volumes (SCSI targets)
- Client mounts and open a file from the server's filesystem
- Server grants the open and provides the block layout map (device ID and the list of blocks within the device) of the requested file to the client
- Based on the block layout (read/write) obtained, client can read/write in parallel directly to the SCSI targets

SCSI Layout Type Extension

https://tools.ietf.org/html/draft-ietf-nfsv4-scsi-layout-05

- RFC5662 defined Layout type 4 to support File, Object and Block volume layout type.
- The SCSI layout type (*LAYOUT4_SCSI*) proposed in RFC draft-ietf-nfsv4-scsi-layout-05 for a SCSI class of storage allows mapping from an NFS file (or portion of a file) to the blocks of storage volumes that contain the file.
  - The blocks are expressed as extents with 64-bit offsets and lengths using the existing NFSv4 offset4 and length4 types

```
 enum layouttype4 {
LAYOUT4_NFSV4_1_FILES   = 0x1,
LAYOUT4_OSD2_OBJECTS    = 0x2,
LAYOUT4_BLOCK_VOLUME    = 0x3,
LAYOUT4_SCSI            = TBD
};
```

GETDEVICEINFO - Block Volume Identification

- SCSI targets implementing [SPC4] export unique LU names for each LU through the Device Identification VPD page (page code 0x83)
- This solution uses a subset of this information to identify LUs backing pNFS SCSI layouts.
  - It is similar to the "Identification Descriptor Target Descriptor" specified in [SPC4], but limits the allowed values to those that uniquely identify a LU
  - The "CODE SET" VPD page field is stored in the "sbv_code_set" field of the "pnfs_scsi_base_volume_info4" structure, the "DESIGNATOR TYPE" is stored in "sbv_designator_type", and the DESIGNATOR is stored in "sbv_designator"
- The Device Identification VPD page MAY contain multiple descriptors with the same association, code set and designator type.
  - NFS clients thus MUST check all the descriptors for a possible match to "sbv_code_set", "sbv_designator_type" and "sbv_designator"
  - Additionally, the server returns a Persistent Reservation key in the "sbv_pr_key" field

pNFS SCSI Layout Block Volume Topology

- The pNFS SCSI layout block volume topology is expressed as an arbitrary combination of base volume types enumerated in the following data structures
- The individual components of the topology are contained in an array and components may refer to other components by using array indices

```
enum pnfs_scsi_volume_type4 {
        PNFS_SCSI_VOLUME_SLICE          = 1,  /* volume is a slice of another volume */
        PNFS_SCSI_VOLUME_CONCAT         = 2,  /* volume is a concatenation of multiple volumes */
        PNFS_SCSI_VOLUME_STRIPE         = 3,  /* volume is striped across multiple volumes */
        PNFS_SCSI_VOLUME_BASE           = 4   /* volume maps to a single LU */
};
```

pNFS SCSI Layout Extents and Extent States

- The SCSI layout consists of a list of extents that map the regions of the file to locations on a block volume
- The "*se_storage_offset*" field within each extent identifies a location on the volume specified by the "*se_vol_id*" field in the extent

```
struct pnfs_scsi_extent4 {
    deviceid4    se_vol_id;      /* id of logical volume on which extent of file is stored. */
    offset4      se_file_offset;  /* the starting byte offset in the file */
    length4      se_length;      /* the size in bytes of the extent */
    offset4      se_storage_offset;  /* the starting byte offset in the volume */
    pnfs_scsi_extent_state4 se_state; /* the state of this extent */
};
```

pNFS Protocol Operations via Network

- *GETDEVICEINFO*
  - Retrieve mapping information for a device
- *GETDEVICELIST*
  - Client requests the list of all data servers participating in a storage cluster
  - Very rarely used
- *LAYOUTGET*
  - Obtains the data server map from the metadata server
  - Returns the device ID and the list of blocks within the device
- *LAYOUTCOMMIT*
  - Commit data so that it is visible to other clients, and update the metadata maps
- *LAYOUTRETURN*
  - Returns the layout or the new layout, if the data is modified.

PNFS Call Back Operations

- *CB_LAYOUTRECALL*
  - Recall a layout or all layouts from a file
  - Server recalls the data layout from a client; for example, if conflicts are detected
- *CB_RECALLABLE_OBJ_AVAIL*
  - Inform a client that a layout is now available
- *CB_NOTIFY_DEVICEID*
  - Inform client about changed device mappings
- *CB_RECALL_ANY*
  - The server may decide that it cannot hold all of the state for layouts without running out of resources. In such a case, it is free to recall individual layouts using CB_LAYOUTRECALL to reduce the load, or it may choose to request that the client return any layout

pNFS Block Metadata Server

pNFS Block Metadata Server (MDS) for Block Storage

- Open source initiative under HSG AD incubator project
- Complies to pNFS storage access protocol for block storage (RFC5663)
- Pertains to implementation of a pNFS block metadata server
- Extends NFSv4.x server core to coordinate pNFS client access through block layout driver to highly parallelized set of E-Series block storage systems
- Complements existing NetApp pNFS solutions in file space

pNFS Block MDS Internals

- The MDS defines the complex volume structure based on top of E-Series block storage LUNs and provides the layout information that maps the file to the block storage device extents in terms of byte address locations to pNFS clients
- The pNFS client uses the layout information to direct IO to the block storage
- Low-level locking implemented using layout recalls, with the notion that the MDS is free to recall the layout at any time
  - Client commits the changes upon IO completion. Client may return the layout in case of
    - MDS can request layout on implicit close
    - Client must return layout on recall

Persistent Reservations – Block Storage

- SCSI persistent reservations provides the basic mechanism for dynamic contention resolution in systems with multiple initiator ports accessing a logical unit.
- Reservations may be used to allow E-Series storage device to process commands from a single initiator port or a selected set of initiator ports and reject commands from initiator ports outside the selected set.
- The *PERSISTENT RESERVE IN* and *PERSISTENT RESERVE OUT* commands are used for persistent reservations in SCSI-3 (SPC-3)
  - The *PERSISTENT RESERVE OUT* command is used to request service actions that reserve a logical unit for the exclusive or shared use of a particular initiator port, actually of a specific combination of initiator port and target port referred to as an I_T nexus in SCSI-3)
  - The *PERSISTENT RESERVE IN* command is used to obtain information about persistent reservations and reservation keys (i.e., registrations) that are active within a device server

pNFS Client Fencing Enforcement

- The pNFS SCSI protocol must handle situations in which a system failure, typically a network connectivity issue, requires the server to unilaterally revoke extents from a client after the client fails to respond to a CB_LAYOUTRECALL request.
  - This is implemented by fencing off a non-responding client from access to the storage device
- The pNFS SCSI protocol implements fencing using Persistent Reservations (PRs), similar to the fencing method used by existing shared disk file systems.
  - By placing a PR of type "Exclusive Access - All Registrants" on each SCSI LU exported to pNFS clients the MDS prevents access from any client that does not have an outstanding device device ID that gives the client a reservation key to access the LU, and allows the MDS to revoke access to the logic unit at any time

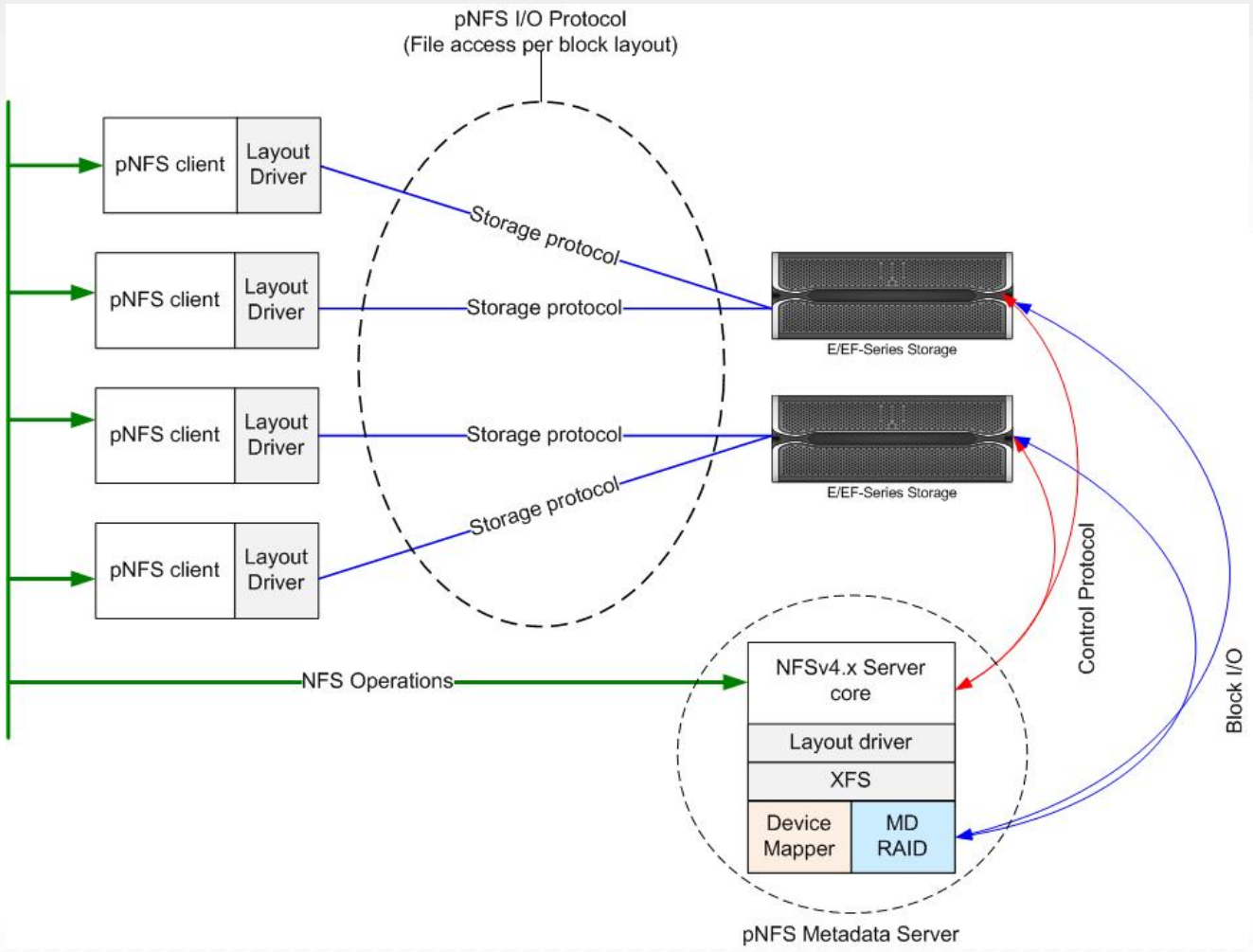
PR based Interactions for Client Fencing

- PRs - Key Generation
  - To allow fencing individual systems, each system must use a unique Persistent Reservation key. This solution places responsibility on MDS to generate unique keys, first for itself before exporting a volume, and a key for each client that accesses a scsi layout volumes
- PRs – MDS Registration and Reservation
  - Before returning a PNFS_SCSI_VOLUME_BASE volume to the client, the MDS needs to prepare the volume for fencing using PRs. This is done by registering the reservation generated for the MDS with the device using the "PERSISTENT RESERVE OUT" command with a service action of "REGISTER", followed by a "PERSISTENT RESERVE OUT" command, with service action of "RESERVE" and the type field set to 8h (Exclusive Access - All Registrants)
- PRs – Client Registration
  - Before performing the first IO to a device returned from a GETDEVICEINFO operation the client will register the registration key returned in sbv_pr_key with the storage device by issuing a "PERSISTENT RESERVE OUT" command with a service action of REGISTER with the "SERVICE ACTION RESERVATION KEY" set to the reservation key returned in sbv_pr_key
- PRs – Fencing Action
  - In case of a non-responding client the MDS fences the client by issuing a "PERSISTENT RESERVE OUT" command with the service action set to "PREEMPT" or "PREEMPT AND ABORT", the reservation key field set to the server's reservation key, the service action reservation key field set to the reservation key associated with the non-responding client, and the type field set to 8h (Exclusive Access – All Registrants)
- Client recovery after fencing
  - A client that detects a "RESERVATION CONFLICT" SCSI status (I/O error) on the storage devices MUST commit all layouts that use the storage device through the MDS, return all outstanding layouts for the device, forget the device ID and unregister the reservation key

Crash Recovery Process

- Beyond RFC5661, this solution addresses crash recovery issues specifically in the context of SCSI layouts
- When the server crashes while the client holds a writable layout, and the client has written data to blocks covered by the layout, and the blocks are still in the PNFS_SCSI_INVALID_DATA state, the client has two options for recovery
  - If the data that has been written to these blocks is still cached by the client, the client can simply re-write the data via NFSv4, once the server has come back online
  - If the data is no longer in the client's cache, the client MUST NOT attempt to source the data from the data servers. Instead, it should attempt to commit the blocks in question to the server during the server's recovery grace period, by sending a LAYOUTCOMMIT with the "loca_reclaim" flag set to true

pNFS Block MDS for E/EF-Series Block Storage - Overview



Block Target Access - Device Mapper (DM) vs. Software RAID (MD)

- MD: Original Linux software RAID driver. No H/L volume features (snapshot, dedupe, etc.)
  - Handles all metadata in kernel space
- DM: Low-level component of the volume manager (LVM2). Also supports basic RAID types
  - Handles metadata in user space in LVM2
- We require our pNFS solution to support complex volume structure laid on top of block LUNs to support striping and or concatenation
  - Supported by both MD and DM approaches

High Availability Proposal for MDS w/ SCSI Layout

- It is currently possible to support simple active/passive high-availability NFS service.
- Goal is to support HA when using pNFS SCSI Layout implementation in kernel upstream
- The proposal for initial approach is to treat server migration (triggered by HA failover) as a reboot, similar to how it is handled for other NFSv4 state;
  - As there's no shared NFSv4 state, and the state being private to each server (tied to the server's lifecycle), it will be the responsibility of the clients to replay the state to the new takeover server
- After migration, a client that attempts to use preexisting SCSI reservations or pNFS layout state id's will get errors
- Those errors will trigger recovery behavior on the client which will cause it to reclaim any NFSv4 state and reacquire layouts as necessary

pNFS Block Server – Ph1 Content

- A pNFS block metadata server using the XFS filesystem on top of an iSCSI external iSCSI target supporting full read and write operations on simple layouts
- A device management layer, which exports physical storage layout information to the pNFS metadata server
- A XFS filesystem that can export physical file layout so that the NFS client can directly access the storage without going through the server
  - Development complete as of Q2 CY14
  - Performance results from single client with 10GB iSCSI link meets the target
  - Performance evaluation with multiple clients is in progress

Phase1 – Performance Results

- Performance evaluation between the server and the array with 10Gb iSCSI link, @ 1MB IO size, and file size (20 to 30 GB) larger than cache size on the controller:
  - Sequential reads: 850 MB/s
  - Sequential writes: 630 MB/s
- Max throughput with single IO path with DM devices
  - Reads: 1000MB/s
  - Writes 970 MB/s

Phase 2 – Content

- Development complete as of Q4 2015
- RFC draft and code implementation pertaining to for block layout support, shared reserve/release mechanism and client fencing implementation
- Supports block volume capacity expansion to allow on-line hot-add of LUNs
  - Requires *CB_NOTIFY_DEVICEID* support on server and client
- pNFS client bug fixes for data integrity related issues, rolling client fixes into kernel upstream submission
  - Required for productization with stable client versions (RHEL, SLES, Fedora) in phase 3
- Bug fixes on server code, Server refactor, porting server core to latest upstream

Next Steps

Phase 3 (Productization) - Scope

- HA support on Block Layout/MDS
- Extensions to support other storage protocols beyond iSCSI (FC and IB, in particular)
- Server scalability
  - Back-port NFSv4.1
- Rolling MDS into kernel upstream submission
- Sizing of the server, inter-connect to the array, topology standardization

Go-To-Market Strategy

- Different pathways for market reach:
  - Reseller
    - Service, Install, and Application expertise
    - Pre-configured solution to include Server and Storage
    - Alternative option to be considered – Appliance solution
  - Server OS Vendors
    - RedHat, SuSe
  - Reference Architecture with Block storage devices
    - Characterized on the basis of performance

Acknowledgements

- **Christoph Hellwig** for his contributions throughout this project
- **Trond Myklebust** for initial concept and proposal
- **Linux community** for timely reviews against all commits and RFC draft reviews and approval
- **RedHat NFS/HA team** for reviewing and providing constructive feedback against HA proposal for the server

Thank you