NFSv4 C. Hellwig
Internet-Draft December 03, 2015
Intended status: Standards Track
Expires: June 5, 2016

# Parallel NFS (pNFS) SCSI Layout
# draft-ietf-nfsv4-scsi-layout-05.txt

Abstract

The Parallel Network File System (pNFS) allows a separation between the metadata (onto a metadata server) and data (onto a storage device) for a file. The SCSI Layout Type is defined in this document as an extension to pNFS to allow the use SCSI based block storage devices.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 5, 2016.

Table of Contents

## 1. Introduction

Figure 1 shows the overall architecture of a Parallel NFS (pNFS) system:

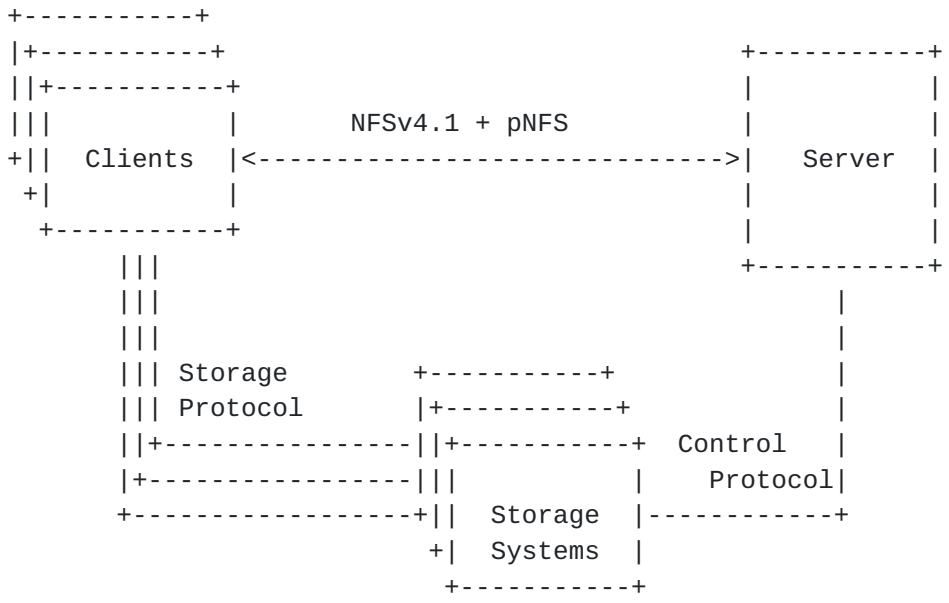


Figure 1

The overall approach is that pNFS-enhanced clients obtain sufficient information from the server to enable them to access the underlying storage (on the storage systems) directly. See the Section 12 of [RFC5661] for more details. This document is concerned with access from pNFS clients to storage devices over block storage protocols based on the the SCSI Architecture Model ([SAM-4]), e.g., Fibre Channel Protocol (FCP) for Fibre Channel, Internet SCSI (iSCSI) or Serial Attached SCSI (SAS). pNFS SCSI layout requires block based SCSI command sets, for example SCSI Block Commands ([SBC3]). While SCSI command set for non-block based access exist these are not supported by the SCSI layout type, and all future references to SCSI storage devices will imply a block based SCSI command set.

The Server to Storage System protocol, called the "Control Protocol", is not of concern for interoperability, although it will typically be the same SCSI based storage protocol.

This document is based on [RFC5663] and makes changes to the block layout type to provide a better pNFS layout protocol for SCSI based storage devices. Despite these changes, [RFC5663] remains the defining document for the existing block layout type. [RFC6688] is unnecessary in the context of the SCSI layout type because the new

layout type provides mandatory disk access protection as part of the layout type definition. In contrast to [RFC5663], this document uses SCSI protocol features to provide reliable fencing by using SCSI Persistent Reservations, and it can provide reliable and efficient device discovery by using SCSI device identifiers instead of having to rely on probing all devices potentially attached to a client for a signature. This new layout type also optimizes the I/O path by reducing the size of the LAYOUTCOMMIT payload

## 1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 1.2. General Definitions

The following definitions are provided for the purpose of providing an appropriate context for the reader.

Byte This document defines a byte as an octet, i.e., a datum exactly 8 bits in length.

Client The "client" is the entity that accesses the NFS server's resources. The client may be an application that contains the logic to access the NFS server directly. The client may also be the traditional operating system client that provides remote file system services for a set of applications.

Server The "server" is the entity responsible for coordinating client access to a set of file systems and is identified by a server owner.

## 1.3. Code Components Licensing Notice

The external data representation (XDR) description and scripts for extracting the XDR description are Code Components as described in Section 4 of "Legal Provisions Relating to IETF Documents" [LEGAL]. These Code Components are licensed according to the terms of Section 4 of "Legal Provisions Relating to IETF Documents".

## 1.4. XDR Description

This document contains the XDR [RFC4506] description of the NFSv4.1 SCSI layout protocol. The XDR description is embedded in this document in a way that makes it simple for the reader to extract into a ready-to-compile form. The reader can feed this document into the following shell script to produce the machine readable XDR

description of the NFSv4.1 SCSI layout:

```
#!/bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^ *///$??'
```

That is, if the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
sh extract.sh < spec.txt > scsi_prot.x
```

The effect of the script is to remove leading white space from each line, plus a sentinel sequence of "///".

The embedded XDR file header follows. Subsequent XDR descriptions, with the sentinel sequence are embedded throughout the document.

Note that the XDR code contained in this document depends on types from the NFSv4.1 nfs4_prot.x file [RFC5662]. This includes both nfs types that end with a 4, such as offset4, length4, etc., as well as more generic types such as uint32_t and uint64_t.

```
/// /*
///  * This code was derived from RFCTBD10
///  * Please reproduce this note if possible.
///  */
/// /*
///  * Copyright (c) 2010,2015 IETF Trust and the persons
///  * identified as the document authors.  All rights reserved.
///  *
///  * Redistribution and use in source and binary forms, with
///  * or without modification, are permitted provided that the
///  * following conditions are met:
///  *
///  * - Redistributions of source code must retain the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer.
///  *
///  * - Redistributions in binary form must reproduce the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer in the documentation and/or other
///  *   materials provided with the distribution.
///  *
///  * - Neither the name of Internet Society, IETF or IETF
///  *   Trust, nor the names of specific contributors, may be
///  *   used to endorse or promote products derived from this
///  *   software without specific prior written permission.
///  *
```

```
///  *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
///  *   AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
///  *   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
///  *   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
///  *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO
///  *   EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
///  *   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
///  *   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
///  *   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
///  *   SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
///  *   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
///  *   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
///  *   OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
///  *   IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
///  *   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
///  */
///
/// /*
///  *      nfs4_scsi_layout_prot.x
///  */
///
/// %#include "nfsv41.h"
///
```

## 2. SCSI Layout Description

### 2.1. Background and Architecture

The fundamental storage model supported by SCSI storage devices is a Logical Unit (LU) consisting of a sequential series of fixed-size blocks. Logical units used as devices for NFS scsi layouts, and the SCSI initiators used for the pNFS Metadata Server and clients MUST support SCSI persistent reservations.

A pNFS layout for this SCSI class of storage is responsible for 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. Clients MUST be able to perform I/O to the block extents without affecting additional areas of storage (especially important for writes); therefore, extents MUST be aligned to 512-byte boundaries.

The pNFS operation for requesting a layout (LAYOUTGET) includes the "layoutiomode4 loga_iomode" argument, which indicates whether the requested layout is for read-only use or read-write use. A read-only layout may contain holes that are read as zero, whereas a read-write

layout will contain allocated, but un-initialized storage in those holes (read as zero, can be written by client). This document also supports client participation in copy-on-write (e.g., for file systems with snapshots) by providing both read-only and un-initialized storage for the same range in a layout. Reads are initially performed on the read-only storage, with writes going to the un-initialized storage. After the first write that initializes the un-initialized storage, all reads are performed to that now-initialized writable storage, and the corresponding read-only storage is no longer used.

The SCSI layout solution expands the security responsibilities of the pNFS clients, and there are a number of environments where the mandatory to implement security properties for NFS cannot be satisfied. The additional security responsibilities of the client follow, and a full discussion is present in Section 4, "Security Considerations".

- Typically, SCSI storage devices provide access control mechanisms (e.g., Logical Unit Number (LUN) mapping and/or masking), which operate at the granularity of individual hosts, not individual blocks. For this reason, block-based protection must be provided by the client software.

- Similarly, SCSI storage devices typically are not able to validate NFS locks that apply to file regions. For instance, if a file is covered by a mandatory read-only lock, the server can ensure that only readable layouts for the file are granted to pNFS clients. However, it is up to each pNFS client to ensure that the readable layout is used only to service read requests, and not to allow writes to the existing parts of the file.

Since SCSI storage devices are generally not capable of enforcing such file-based security, in environments where pNFS clients cannot be trusted to enforce such policies, pNFS SCSI layouts SHOULD NOT be used.

## 2.2. layouttype4

The layout4 type defined in [RFC5662] is extended with a new value as follows:

```
      enum layouttype4 {
          LAYOUT4_NFSV4_1_FILES   = 1,
          LAYOUT4_OSD2_OBJECTS    = 2,
          LAYOUT4_BLOCK_VOLUME    = 3,
          LAYOUT4_SCSI            = 0x80000005
  [[RFC Editor: please modify the LAYOUT4_SCSI
    to be the layouttype assigned by IANA]]
      };
```

This document defines structure associated with the layouttype4 value LAYOUT4_SCSI. [RFC5661] specifies the loc_body structure as an XDR type "opaque". The opaque layout is uninterpreted by the generic pNFS client layers, but obviously must be interpreted by the Layout Type implementation.

## 2.3. GETDEVICEINFO

### 2.3.1. Volume Identification

SCSI targets implementing [SPC4] export unique LU names for each LU through the Device Identification VPD page (page code 0x83), which can be obtained using the INQUIRY command with the EVPD bit set to one. This document 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. Device Identification VPD page descriptors used to identify LUs for use with pNFS SCSI layouts must adhere to the following restrictions:

1. The "ASSOCIATION" MUST be set to 0 (The DESIGNATOR field is associated with the addressed logical unit).

2. The "DESIGNATOR TYPE" MUST be set to one of four values that are required for the mandatory logical unit name in [SPC4], as explicitly listed in the "pnfs_scsi_designator_type" enumeration:

   PS_DESIGNATOR_T10 T10 vendor ID based

   PS_DESIGNATOR_EUI64 EUI-64-based

   PS_DESIGNATOR_NAA NAA

   PS_DESIGNATOR_NAME SCSI name string

   Any other association or designator type MUST NOT be used. Use of T10 vendor IDs is discouraged when one of the other types can be used.

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". Due to the use of a XDR array the "DESIGNATOR LENGTH" field does not need to be set separately. Only certain combinations of "sbv_code_set" and "sbv_designator_type" are valid, please refer to [SPC4] for details, and note that ASCII may be used as the code set for UTF-8 text that contains only printable ASCII characters. Note that a 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".

Storage devices such as storage arrays can have multiple physical network ports that need not be connected to a common network, resulting in a pNFS client having simultaneous multipath access to the same storage volumes via different ports on different networks. Selection of one or multiple ports to access the storage device is left up to the client.

Additionally the server returns a Persistent Reservation key in the "sbv_pr_key" field. See Section 2.4.10 for more details on the use of Persistent Reservations.

### 2.3.2. Volume Topology

The pNFS SCSI layout 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 */
/// };
///
```

```
    /// /*
    ///  * Code sets from SPC-3.
    ///  */
    /// enum pnfs_scsi_code_set {
    ///     PS_CODE_SET_BINARY     = 1,
    ///     PS_CODE_SET_ASCII      = 2,
    ///     PS_CODE_SET_UTF8       = 3
    /// };
    ///
    /// /*
    ///  * Designator types from taken from SPC-3.
    ///  *
    ///  * Other values are allocated in SPC-3, but not mandatory to
    ///  * implement or aren't Logical Unit names.
    ///  */
    /// enum pnfs_scsi_designator_type {
    ///     PS_DESIGNATOR_T10      = 1,
    ///     PS_DESIGNATOR_EUI64    = 2,
    ///     PS_DESIGNATOR_NAA      = 3,
    ///     PS_DESIGNATOR_NAME     = 8
    /// };
    ///
    /// /*
    ///  * Logical Unit name + reservation key.
    ///  */
    /// struct pnfs_scsi_base_volume_info4 {
    ///     pnfs_scsi_code_set             sbv_code_set;
    ///     pnfs_scsi_designator_type      sbv_designator_type;
    ///     opaque                         sbv_designator<>;
    ///     uint64_t                       sbv_pr_key;
    /// };
    ///


    /// struct pnfs_scsi_slice_volume_info4 {
    ///     offset4  ssv_start;            /* offset of the start of
    ///                                       the slice in bytes */
    ///     length4  ssv_length;           /* length of slice in
    ///                                       bytes */
    ///     uint32_t ssv_volume;           /* array index of sliced
    ///                                       volume */
    /// };
    ///
```

```
///
/// struct pnfs_scsi_concat_volume_info4 {
///     uint32_t  scv_volumes<>;       /* array indices of volumes
///                                       which are concatenated */
/// };


///
/// struct pnfs_scsi_stripe_volume_info4 {
///     length4  ssv_stripe_unit;      /* size of stripe in bytes */
///     uint32_t ssv_volumes<>;        /* array indices of
///                                       volumes which are striped
///                                       across -- MUST be same
///                                       size */
/// };


///
/// union pnfs_scsi_volume4 switch (pnfs_scsi_volume_type4 type) {
///     case PNFS_SCSI_VOLUME_BASE:
///         pnfs_scsi_base_volume_info4 sv_simple_info;
///     case PNFS_SCSI_VOLUME_SLICE:
///         pnfs_scsi_slice_volume_info4 sv_slice_info;
///     case PNFS_SCSI_VOLUME_CONCAT:
///         pnfs_scsi_concat_volume_info4 sv_concat_info;
///     case PNFS_SCSI_VOLUME_STRIPE:
///         pnfs_scsi_stripe_volume_info4 sv_stripe_info;
/// };
///


/// /* SCSI layout-specific type for da_addr_body */
/// struct pnfs_scsi_deviceaddr4 {
///     pnfs_scsi_volume4 sda_volumes<>; /* array of volumes */
/// };
///
```

The "pnfs_scsi_deviceaddr4" data structure is a structure that allows arbitrarily complex nested volume structures to be encoded. The types of aggregations that are allowed are stripes, concatenations, and slices. Note that the volume topology expressed in the pnfs_scsi_deviceaddr4 data structure will always resolve to a set of pnfs_scsi_volume_type4 PNFS_SCSI_VOLUME_BASE. The array of volumes is ordered such that the root of the volume hierarchy is the last element of the array. Concat, slice, and stripe volumes MUST refer to volumes defined by lower indexed elements of the array.

The "pnfs_scsi_device_addr4" data structure is returned by the server

as the storage-protocol-specific opaque field da_addr_body in the "device_addr4" structure by a successful GETDEVICEINFO operation [RFC5661].

As noted above, all device_addr4 structures eventually resolve to a set of volumes of type PNFS_SCSI_VOLUME_BASE. Complicated volume hierarchies may be composed of dozens of volumes each with several signature components; thus, the device address may require several kilobytes. The client SHOULD be prepared to allocate a large buffer to contain the result. In the case of the server returning NFS4ERR_TOOSMALL, the client SHOULD allocate a buffer of at least gdir_mincount_bytes to contain the expected result and retry the GETDEVICEINFO request.

## 2.4. Data Structures: Extents and Extent Lists

A pNFS SCSI layout is a list of extents within a flat array of data blocks in a volume. The details of the volume topology can be determined by using the GETDEVICEINFO operation. The SCSI layout describes the individual block extents on the volume that make up the file. The offsets and length contained in an extent are specified in units of bytes.

```
/// enum pnfs_scsi_extent_state4 {
///     PNFS_SCSI_READ_WRITE_DATA = 0, /* the data located by
///                                       this extent is valid
///                                       for reading and
///                                       writing. */
///     PNFS_SCSI_READ_DATA      = 1,  /* the data located by this
///                                       extent is valid for
///                                       reading only; it may not
///                                       be written. */
///     PNFS_SCSI_INVALID_DATA   = 2,  /* the location is valid; the
///                                       data is invalid. It is a
///                                       newly (pre-) allocated
///                                       extent. The client MUST
///                                       not read from this
///                                       space */
///     PNFS_SCSI_NONE_DATA      = 3   /* the location is invalid.
///                                       It is a hole in the file.
///                                       The client MUST NOT read
///                                       from or write to this
///                                       space */
/// };
```

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


/// /* SCSI layout-specific type for loc_body */
/// struct pnfs_scsi_layout4 {
///     pnfs_scsi_extent4 sl_extents<>;
///                                    /* extents which make up this
///                                       layout. */
/// };
///
```

The SCSI layout consists of a list of extents that map the regions of the file to locations on a 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. The se_vol_id itself is shorthand for the whole topology of the volume on which the file is stored. The client is responsible for translating this volume-relative offset into an offset on the appropriate underlying SCSI LU.

Each extent maps a region of the file onto a portion of the specified LU. The se_file_offset, se_length, and se_state fields for an extent returned from the server are valid for all extents. In contrast, the interpretation of the se_storage_offset field depends on the value of se_state as follows (in increasing order):

PNFS_SCSI_READ_WRITE_DATA means that se_storage_offset is valid, and points to valid/initialized data that can be read and written.

PNFS_SCSI_READ_DATA means that se_storage_offset is valid and points to valid/initialized data that can only be read. Write operations are prohibited; the client may need to request a read-write layout.

PNFS_SCSI_INVALID_DATA means that se_storage_offset is valid, but points to invalid un-initialized data. This data must not be read from the disk until it has been initialized. A read request for a PNFS_SCSI_INVALID_DATA extent must fill the user buffer with zeros, unless the extent is covered by a PNFS_SCSI_READ_DATA extent of a copy-on-write file system. Write requests must write whole server-sized blocks to the disk; bytes not initialized by the user must be set to zero. Any write to storage in a PNFS_SCSI_INVALID_DATA extent changes the written portion of the extent to PNFS_SCSI_READ_WRITE_DATA; the pNFS client is responsible for reporting this change via LAYOUTCOMMIT.

PNFS_SCSI_NONE_DATA means that se_storage_offset is not valid, and this extent may not be used to satisfy write requests. Read requests may be satisfied by zero-filling as for PNFS_SCSI_INVALID_DATA. PNFS_SCSI_NONE_DATA extents may be returned by requests for readable extents; they are never returned if the request was for a writable extent.

An extent list contains all relevant extents in increasing order of the se_file_offset of each extent; any ties are broken by increasing order of the extent state (se_state).

### 2.4.1. Layout Requests and Extent Lists

Each request for a layout specifies at least three parameters: file offset, desired size, and minimum size. If the status of a request indicates success, the extent list returned must meet the following criteria:

- A request for a readable (but not writable) layout returns only PNFS_SCSI_READ_DATA or PNFS_SCSI_NONE_DATA extents (but not PNFS_SCSI_INVALID_DATA or PNFS_SCSI_READ_WRITE_DATA extents).

- A request for a writable layout returns PNFS_SCSI_READ_WRITE_DATA or PNFS_SCSI_INVALID_DATA extents (but not PNFS_SCSI_NONE_DATA extents). It may also return PNFS_SCSI_READ_DATA extents only when the offset ranges in those extents are also covered by PNFS_SCSI_INVALID_DATA extents to permit writes.

- The first extent in the list MUST contain the requested starting offset.

- The total size of extents within the requested range MUST cover at least the minimum size. One exception is allowed: the total size MAY be smaller if only readable extents were requested and EOF is encountered.

- Extents in the extent list MUST be logically contiguous for a read-only layout. For a read-write layout, the set of writable extents (i.e., excluding PNFS_SCSI_READ_DATA extents) MUST be logically contiguous. Every PNFS_SCSI_READ_DATA extent in a read-write layout MUST be covered by one or more PNFS_SCSI_INVALID_DATA extents. This overlap of PNFS_SCSI_READ_DATA and PNFS_SCSI_INVALID_DATA extents is the only permitted extent overlap.

- Extents MUST be ordered in the list by starting offset, with PNFS_SCSI_READ_DATA extents preceding PNFS_SCSI_INVALID_DATA extents in the case of equal se_file_offsets.

According to [RFC5661], if the minimum requested size, loga_minlength, is zero, this is an indication to the metadata server that the client desires any layout at offset loga_offset or less that the metadata server has "readily available". Given the lack of a clear definition of this phrase, in the context of the SCSI layout type, when loga_minlength is zero, the metadata server SHOULD:

- when processing requests for readable layouts, return all such, even if some extents are in the PNFS_SCSI_NONE_DATA state.

- when processing requests for writable layouts, return extents which can be returned in the PNFS_SCSI_READ_WRITE_DATA state.

### 2.4.2. Layout Commits

```
///
/// /* SCSI layout-specific type for lou_body */
///
/// struct pnfs_scsi_range4 {
///     offset4      sr_file_offset;   /* starting byte offset
///                                       in the file */
///     length4      sr_length;        /* size in bytes */
/// };
///
/// struct pnfs_scsi_layoutupdate4 {
///     pnfs_scsi_range4 slu_commit_list<>;
///                                    /* list of extents which
///                                     * now contain valid data.
///                                     */
/// };
```

The "pnfs_scsi_layoutupdate4" structure is used by the client as the SCSI layout-specific argument in a LAYOUTCOMMIT operation. The "slu_commit_list" field is a list covering regions of the file layout that were previously in the PNFS_SCSI_INVALID_DATA state, but have

been written by the client and should now be considered in the PNFS_SCSI_READ_WRITE_DATA state. The extents in the commit list MUST be disjoint and MUST be sorted by sr_file_offset. Implementors should be aware that a server may be unable to commit regions at a granularity smaller than a file-system block (typically 4 KB or 8 KB). As noted above, the block-size that the server uses is available as an NFSv4 attribute, and any extents included in the "slu_commit_list" MUST be aligned to this granularity and have a size that is a multiple of this granularity. Since the block in question is in state PNFS_SCSI_INVALID_DATA, byte ranges not written should be filled with zeros. This applies even if it appears that the area being written is beyond what the client believes to be the end of file.

### 2.4.3. Layout Returns

A LAYOUTRETURN operation represents an explicit release of resources by the client. This may be done in response to a CB_LAYOUTRECALL or before any recall, in order to avoid a future CB_LAYOUTRECALL. When the LAYOUTRETURN operation specifies a LAYOUTRETURN4_FILE return type, then the layoutreturn_file4 data structure specifies the region of the file layout that is no longer needed by the client.

The LAYOUTRETURN operation is done without any SCSI layout specific data. The opaque "lrf_body" field of the "layoutreturn_file4" data structure MUST have length zero.

### 2.4.4. Layout Revocation

Layouts may be unilaterally revoked by the server, due to the client's lease time expiring, or the client failing to return a layout which has been recalled in a timely manner. For the SCSI layout type this is accomplished by fencing off the client from access to storage as described in Section 2.4.10. When this is done, it is necessary that all I/Os issued by the fenced-off client be rejected by the storage This includes any in-flight I/Os that the client issued before the layout was revoked.

Note, that the granularity of this operation can only be at the host/LU level. Thus, if one of a client's layouts is unilaterally revoked by the server, it will effectively render useless *all* of the client's layouts for files located on the storage units comprising the volume. This may render useless the client's layouts for files in other file systems. See Section 2.4.10.5 for a discussion of recovery from from fencing.

### 2.4.5. Client Copy-on-Write Processing

Copy-on-write is a mechanism used to support file and/or file system snapshots. When writing to unaligned regions, or to regions smaller than a file system block, the writer must copy the portions of the original file data to a new location on disk. This behavior can either be implemented on the client or the server. The paragraphs below describe how a pNFS SCSI layout client implements access to a file that requires copy-on-write semantics.

Distinguishing the PNFS_SCSI_READ_WRITE_DATA and PNFS_SCSI_READ_DATA extent types in combination with the allowed overlap of PNFS_SCSI_READ_DATA extents with PNFS_SCSI_INVALID_DATA extents allows copy-on-write processing to be done by pNFS clients. In classic NFS, this operation would be done by the server. Since pNFS enables clients to do direct block access, it is useful for clients to participate in copy-on-write operations. All SCSI pNFS clients MUST support this copy-on-write processing.

When a client wishes to write data covered by a PNFS_SCSI_READ_DATA extent, it MUST have requested a writable layout from the server; that layout will contain PNFS_SCSI_INVALID_DATA extents to cover all the data ranges of that layout's PNFS_SCSI_READ_DATA extents. More precisely, for any se_file_offset range covered by one or more PNFS_SCSI_READ_DATA extents in a writable layout, the server MUST include one or more PNFS_SCSI_INVALID_DATA extents in the layout that cover the same se_file_offset range. When performing a write to such an area of a layout, the client MUST effectively copy the data from the PNFS_SCSI_READ_DATA extent for any partial blocks of se_file_offset and range, merge in the changes to be written, and write the result to the PNFS_SCSI_INVALID_DATA extent for the blocks for that se_file_offset and range. That is, if entire blocks of data are to be overwritten by an operation, the corresponding PNFS_SCSI_READ_DATA blocks need not be fetched, but any partial-block writes must be merged with data fetched via PNFS_SCSI_READ_DATA extents before storing the result via PNFS_SCSI_INVALID_DATA extents. For the purposes of this discussion, "entire blocks" and "partial blocks" refer to the server's file-system block size. Storing of data in a PNFS_SCSI_INVALID_DATA extent converts the written portion of the PNFS_SCSI_INVALID_DATA extent to a PNFS_SCSI_READ_WRITE_DATA extent; all subsequent reads MUST be performed from this extent; the corresponding portion of the PNFS_SCSI_READ_DATA extent MUST NOT be used after storing data in a PNFS_SCSI_INVALID_DATA extent. If a client writes only a portion of an extent, the extent may be split at block aligned boundaries.

When a client wishes to write data to a PNFS_SCSI_INVALID_DATA extent that is not covered by a PNFS_SCSI_READ_DATA extent, it MUST treat

this write identically to a write to a file not involved with copy-on-write semantics. Thus, data must be written in at least block-sized increments, aligned to multiples of block-sized offsets, and unwritten portions of blocks must be zero filled.

### 2.4.6. Extents are Permissions

Layout extents returned to pNFS clients grant permission to read or write; PNFS_SCSI_READ_DATA and PNFS_SCSI_NONE_DATA are read-only (PNFS_SCSI_NONE_DATA reads as zeroes), PNFS_SCSI_READ_WRITE_DATA and PNFS_SCSI_INVALID_DATA are read/write, (PNFS_SCSI_INVALID_DATA reads as zeros, any write converts it to PNFS_SCSI_READ_WRITE_DATA). This is the only means a client has of obtaining permission to perform direct I/O to storage devices; a pNFS client MUST NOT perform direct I/O operations that are not permitted by an extent held by the client. Client adherence to this rule places the pNFS server in control of potentially conflicting storage device operations, enabling the server to determine what does conflict and how to avoid conflicts by granting and recalling extents to/from clients.

If a client makes a layout request that conflicts with an existing layout delegation, the request will be rejected with the error NFS4ERR_LAYOUTTRYLATER. This client is then expected to retry the request after a short interval. During this interval, the server SHOULD recall the conflicting portion of the layout delegation from the client that currently holds it. This reject-and-retry approach does not prevent client starvation when there is contention for the layout of a particular file. For this reason, a pNFS server SHOULD implement a mechanism to prevent starvation. One possibility is that the server can maintain a queue of rejected layout requests. Each new layout request can be checked to see if it conflicts with a previous rejected request, and if so, the newer request can be rejected. Once the original requesting client retries its request, its entry in the rejected request queue can be cleared, or the entry in the rejected request queue can be removed when it reaches a certain age.

NFSv4 supports mandatory locks and share reservations. These are mechanisms that clients can use to restrict the set of I/O operations that are permissible to other clients. Since all I/O operations ultimately arrive at the NFSv4 server for processing, the server is in a position to enforce these restrictions. However, with pNFS layouts, I/Os will be issued from the clients that hold the layouts directly to the storage devices that host the data. These devices have no knowledge of files, mandatory locks, or share reservations, and are not in a position to enforce such restrictions. For this reason the NFSv4 server MUST NOT grant layouts that conflict with mandatory locks or share reservations. Further, if a conflicting

mandatory lock request or a conflicting open request arrives at the server, the server MUST recall the part of the layout in conflict with the request before granting the request.

### 2.4.7. Partial-Bock Updates

SCSI storage devices do not provide byte granularity access and can only perform read and write operations atomically on a block granularity. WRITES to SCSI storage devices thus require read-modify-write cycles to write data smaller than the block size or which is otherwise not block-aligned. Write operations from multiple clients to the same block can thus lead to data corruption even if the byte range written by the applications does not overlap. When there are multiple clients who wish to access the same block, a pNFS server MUST avoid these conflicts by implementing a concurrency control policy of single writer XOR multiple readers for a given data block.

### 2.4.8. End-of-file Processing

The end-of-file location can be changed in two ways: implicitly as the result of a WRITE or LAYOUTCOMMIT beyond the current end-of-file, or explicitly as the result of a SETATTR request. Typically, when a file is truncated by an NFSv4 client via the SETATTR call, the server frees any disk blocks belonging to the file that are beyond the new end-of-file byte, and MUST write zeros to the portion of the new end-of-file block beyond the new end-of-file byte. These actions render any pNFS layouts that refer to the blocks that are freed or written semantically invalid. Therefore, the server MUST recall from clients the portions of any pNFS layouts that refer to blocks that will be freed or written by the server before effecting the file truncation. These recalls may take time to complete; as explained in [RFC5661], if the server cannot respond to the client SETATTR request in a reasonable amount of time, it SHOULD reply to the client with the error NFS4ERR_DELAY.

Blocks in the PNFS_SCSI_INVALID_DATA state that lie beyond the new end-of-file block present a special case. The server has reserved these blocks for use by a pNFS client with a writable layout for the file, but the client has yet to commit the blocks, and they are not yet a part of the file mapping on disk. The server MAY free these blocks while processing the SETATTR request. If so, the server MUST recall any layouts from pNFS clients that refer to the blocks before processing the truncate. If the server does not free the PNFS_SCSI_INVALID_DATA blocks while processing the SETATTR request, it need not recall layouts that refer only to the PNFS_SCSI_INVALID_DATA blocks.

When a file is extended implicitly by a WRITE or LAYOUTCOMMIT beyond the current end-of-file, or extended explicitly by a SETATTR request, the server need not recall any portions of any pNFS layouts.

### 2.4.9. Layout Hints

The layout hint attribute specified in [RFC5661] is not supported by the SCSI layout, and the pNFS server MUST reject setting a layout hint attribute with a loh_type value of LAYOUT4_SCSI_VOLUME during OPEN or SETATTR operations. On a file system only supporting the SCSI layout a server MUST NOT report the layout_hint attribute in the supported_attrs attribute.

### 2.4.10. Client Fencing

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.

#### 2.4.10.1. PRs - Key Generation

To allow fencing individual systems, each system must use a unique Persistent Reservation key. [SPC4] does not specify a way to generate keys. This document assigns the burden to generate unique keys to the MDS, which must generate a key for itself before exporting a volume, and a key for each client that accesses a scsi layout volumes. Individuals keys for each volume that a client can access are permitted but not required.

#### 2.4.10.2. 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 a

service action of "RESERVE" and the type field set to 8h (Exclusive Access - All Registrants). To make sure all I_T nexuses are registered, the MDS SHOULD set the "All Target Ports" (ALL_TG_PT) bit when registering the key, or otherwise ensure the registration is performed for each initiator port.

#### 2.4.10.3. 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. To make sure all I_T nexus are registered, the client SHOULD set the "All Target Ports" (ALL_TG_PT) bit when registering the key, or otherwise ensure the registration is performed for each initiator port.

When a client stops using a device earlier returned by GETDEVICEINFO it MUST unregister the earlier registered key by issuing a "PERSISTENT RESERVE OUT" command with a service action of "REGISTER" with the "RESERVATION KEY" set to the earlier registered reservation key.

#### 2.4.10.4. 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).

After the MDS preempts a client, all client I/O to the LU fails. The client should at this point return any layout that refers to the device ID that points to the LU. Note that the client can distinguish I/O errors due to fencing from other errors based on the "RESERVATION CONFLICT" SCSI status. Refer to [SPC4] for details.

#### 2.4.10.5. Client Recovery After a Fence Action

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. Future GETDEVICEINFO calls may refer to the storage device again, in which case the client will perform a new registration based on the

key provided (via sbv_pr_key) at that time.

## 2.5. Crash Recovery Issues

A critical requirement in crash recovery is that both the client and the server know when the other has failed. Additionally, it is required that a client sees a consistent view of data across server restarts. These requirements and a full discussion of crash recovery issues are covered in the "Crash Recovery" section of the NFSv41 specification [RFC5661]. This document contains additional crash recovery material specific only to the SCSI layout.

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. However, 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. This process is described in detail in Section 18.42.4 of [RFC5661].

## 2.6. Recalling Resources: 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.

The NFSv4.1 spec [RFC5661] defines the following types:

```
const RCA4_TYPE_MASK_BLK_LAYOUT = 4;

struct CB_RECALL_ANY4args {
       uint32_t      craa_objects_to_keep;
       bitmap4       craa_type_mask;
};
```

When the server sends a CB_RECALL_ANY request to a client specifying the RCA4_TYPE_MASK_BLK_LAYOUT bit in craa_type_mask, the client should immediately respond with NFS4_OK, and then asynchronously return complete file layouts until the number of files with layouts cached on the client is less than craa_object_to_keep.

## 2.7. Transient and Permanent Errors

The server may respond to LAYOUTGET with a variety of error statuses. These errors can convey transient conditions or more permanent conditions that are unlikely to be resolved soon.

The error NFS4ERR_RECALLCONFLICT indicates that the server has recently issued a CB_LAYOUTRECALL to the requesting client, making it necessary for the client to respond to the recall before processing the layout request. A client can wait for that recall to be receive and processe or it can retry as for NFS4ERR_TRYLATER, as described below.

The error NFS4ERR_TRYLATER is used to indicate that the server cannot immediately grant the layout to the client. This may be due to constraints on writable sharing of blocks by multiple clients or to a conflict with a recallable lock (e.g. a delegation). In either case, a reasonable approach for the client is to wait several milliseconds and retry the request. The client SHOULD track the number of retries, and if forward progress is not made, the client should abandon the attempt to get a layout and perform READ and WRITE operations by sending them to the server

The error NFS4ERR_LAYOUTUNAVAILABLE may be returned by the server if layouts are not supported for the requested file or its containing file system. The server may also return this error code if the server is the progress of migrating the file from secondary storage, there is a conflicting lock that would prevent the layout from being granted, or for any other reason that causes the server to be unable to supply the layout. As a result of receiving NFS4ERR_LAYOUTUNAVAILABLE, the client should abandon the attempt to get a layout and perform READ and WRITE operations by sending them to the MDS. It is expected that a client will not cache the file's layoutunavailable state forever. In particular, when the file is closed or opened by the client, issuing a new LAYOUTGET is appropriate.

## 2.8. Volatile write caches

Many storage devices implement volatile write caches that require an explicit flush to persist the data from write operations to stable storage. Storage devices implementing [SBC3] should indicate a volatile write cache by setting the WCE bit to 1 in the Caching mode page. When a volatile write cache is used, the pNFS server must ensure the volatile write cache has been committed to stable storage before the LAYOUTCOMMIT operation returns by using one of the SYNCHRONIZE CACHE commands.

## 3. Enforcing NFSv4 Semantics

The functionality provided by SCSI Persistent Reservations makes it possible for the MDS to control access by individual client machines to specific LUs. Individual client machines may be allowed to or prevented from reading or writing to certain block devices. Finer-grained access control methods are not generally available.

For this reason, certain responsibilities for enforcing NFSv4 semantics, including security and locking, are delegated to pNFS clients when SCSI layouts are being used. The metadata server's role is to only grant layouts appropriately and the pNFS clients have to be trusted to only perform accesses allowed by the layout extents they currently hold (e.g., and not access storage for files on which a layout extent is not held). In general, the server will not be able to prevent a client that holds a layout for a file from accessing parts of the physical disk not covered by the layout. Similarly, the server will not be able to prevent a client from accessing blocks covered by a layout that it has already returned. The pNFS client must respect the layout model for this mapping type to appropriately respect NFSv4 semantics.

Furthermore, there is no way for the storage to determine the specific NFSv4 entity (principal, openowner, lockowner) on whose behalf the IO operation is being done. This fact may limit the functionality to be supported and require the pNFS client to implement server policies other than those describable by layouts. In cases in which layouts previously granted become invalid, the server has the option of recalling them. In situations in which communication difficulties prevent this from happening, layouts may be revoked by the server. This revocation is accompanied by changes in persistent reservation which have the effect of preventing SCSI access to the LUs in question by the client.

### 3.1. Use of Open Stateids

The effective implementation of these NFSv4 semantic constraints is complicated by the different granularities of the actors for the different types of the functionality to be enforced:

- To enforce security constraints for particular principals.
- To enforce locking constraints for particular owners (openowners and lockowners)

Fundamental to enforcing both of these sorts of constraints is the principle that a pNFS client must not issue a SCSI IO operation unless it possesses both:

- A valid open stateid for the file in question, performing the IO that allows IO of the type in question, which is associated with the openowner and principal on whose behalf the IO is to be done.

- A valid layout stateid for the file in question that covers the byte range on which the IO is to be done and that allows IO of that type to be done.

As a result, if the equivalent of IO with an anonymous or write-bypass stateid is to be done, it MUST NOT by done using the pNFS SCSI layout type. The client MAY attempt such IO using READs and WRITEs that do not use pNFS and are directed to the MDS.

When open stateids are revoked, due to lease expiration or any form of administrative revocation, the server MUST recall all layouts that allow IO to be done on any of the files for which open revocation happens. When there is a failure to successfully return those layouts, the client MUST be fenced.

## 3.2. Enforcing Security Restrictions

The restriction noted above provides adequate enforcement of appropriate security restriction when the principal issuing the IO is the same as that opening the file. The server is responsible for checking that the IO mode requested by the open is allowed for the principal doing the OPEN. If the correct sort of IO is done on behalf of the same principal, then the security restriction is thereby enforced.

If IO is done by a principal different from the one that opened the file, the client SHOULD send the IO to be performed by the metadata server rather than doing it directly to the storage device.

## 3.3. Enforcing Locking Restrictions

Mandatory enforcement of whole-file locking by means of share reservations is provided when the pNFS client obeys the requirement set forth in Section 2.1 above. Since performing IO requires a valid open stateid an IO that violates an existing share reservation would only be possible when the server allows conflicting open stateids to exist.

The nature of the SCSI layout type is such implementation/enforcement of mandatory byte-range locks is very difficult. Given that layouts are granted to clients rather than owners, the pNFS client is in no position to successfully arbitrate among multiple lockowners on the same client. Suppose lockowner A is doing a write and, while the IO is pending, lockowner B requests a mandatory byte-range for a byte

range potentially overlapping the pending IO. In such a situation, the lock request cannot be granted while the IO is pending. In a non-pNFS environment, the server would have to wait for pending IO before granting the mandatory byte-range lock. In the pNFS environment the server does not issue the IO and is thus in no position to wait for its completion. The server may recall such layouts but in doing so, it has no way of distinguishing those being used by lockowners A and B, making it difficult to allow B to perform IO while forbidding A from doing so. Given this fact, the MDS need to successfully recall all layouts that overlap the range being locked before returning a successful response to the LOCK request. While the lock is in effect, the server SHOULD respond to requests for layouts which overlap a currently locked area with NFS4ERR_LAYOUTUNAVAILABLE. To simplify the required logic a server MAY do this for all layout requests on the file in question as long as there are any byte-range locks in effect.

Given these difficulties it may be difficult for servers supporting mandatory byte-range locks to also support SCSI layouts. Servers can support advisory byte-range locks instead. The NFSv4 protocol currently has no way of determining whether byte-range lock support on a particular file system will be mandatory or advisory, except by trying operation which would conflict if mandatory locking is in effect. Therefore, to avoid confusion, servers SHOULD NOT switch between mandatory and advisory byte-range locking based on whether any SCSI layouts have been obtained or whether a client that has obtained a SCSI layout has requested a byte-range lock.

## 4. Security Considerations

Access to SCSI storage devices is logically at a lower layer of the I/O stack than NFSv4, and hence NFSv4 security is not directly applicable to protocols that access such storage directly. Depending on the protocol, some of the security mechanisms provided by NFSv4 (e.g., encryption, cryptographic integrity) may not be available or may be provided via different means. At one extreme, pNFS with SCSI layouts can be used with storage access protocols (e.g., serial attached SCSI ([SAS3]) that provide essentially no security functionality. At the other extreme, pNFS may be used with storage protocols such as iSCSI ([RFC7143]) that can provide significant security functionality. It is the responsibility of those administering and deploying pNFS with a SCSI storage access protocol to ensure that appropriate protection is provided to that protocol (physical security is a common means for protocols not based on IP). In environments where the security requirements for the storage protocol cannot be met, pNFS SCSI layouts SHOULD NOT be used.

When security is available for a storage protocol, it is generally at a different granularity and with a different notion of identity than NFSv4 (e.g., NFSv4 controls user access to files, iSCSI controls initiator access to volumes). The responsibility for enforcing appropriate correspondences between these security layers is placed upon the pNFS client. As with the issues in the first paragraph of this section, in environments where the security requirements are such that client-side protection from access to storage outside of the layout is not sufficient, pNFS SCSI layouts SHOULD NOT be used.

## 5. IANA Considerations

IANA is requested to assign a new pNFS layout type in the pNFS Layout Types Registry as follows (the value 5 is suggested): Layout Type Name: LAYOUT4_SCSI Value: 0x00000005 RFC: RFCTBD10 How: L (new layout type) Minor Versions: 1

## 6. Normative References

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents", November 2008, <http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[RFC5662] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, January 2010.

[RFC5663] Black, D., Ed., Fridella, S., Ed., and J. Glasgow, Ed., "Parallel NFS (pNFS) Block/Volume Layout", RFC 5663, January 2010.

[RFC6688] Black, D., Ed., Glasgow, J., and S. Faibish, "Parallel NFS (pNFS) Block Disk Protection", RFC 6688, July 2012.

[RFC7143] Chadalapaka, M., Meth, K., and D. Black, "Internet Small

Computer System Interface (iSCSI) Protocol
(Consolidated)", RFC RFC7143, April 2014.

[SAM-4] INCITS Technical Committee T10, "SCSI Architecture Model - 4 (SAM-4)", ANSI INCITS 447-2008, ISO/IEC 14776-414, 2008.

[SAS3] INCITS Technical Committee T10, "Serial Attached Scsi-3", ANSI INCITS ANSI INCITS 519-2014, ISO/IEC 14776-154, 2014.

[SBC3] INCITS Technical Committee T10, "SCSI Block Commands-3", ANSI INCITS INCITS 514-2014, ISO/IEC 14776-323, 2014.

[SPC4] INCITS Technical Committee T10, "SCSI Primary Commands-4", ANSI INCITS 513-2015, 2015.

## Appendix A. Acknowledgments

Large parts of this document were copied verbatim, and others were inspired by [RFC5663]. Thank to David Black, Stephen Fridella and Jason Glasgow for their work on the pNFS block/volume layout protocol.

David Black, Robert Elliott and Tom Haynes provided a throughout review of early drafts of this document, and their input lead to the current form of the document.

David Noveck provided ample feedback to various drafts of this document, wrote the section on enforcing NFSv4 semantics and rewrote various sections to better catch the intent.

## Appendix B. RFC Editor Notes

[RFC Editor: please remove this section prior to publishing this document as an RFC]

[RFC Editor: prior to publishing this document as an RFC, please replace all occurrences of RFCTBD10 with RFCxxxx where xxxx is the RFC number of this document]

## Author's Address

Christoph Hellwig

Email: hch@lst.de