

NFSv4  
Internet-Draft  
Intended status: Standards Track  
Expires: January 25, 2015

M. Benjamin  
CohortFS, LLC  
Bodley  
Emerson

Ersani  
NetApp  
Honeyman  
July 24, 2014

pNFS Metadata Striping  
draft-mbenjamin-nfsv4-pnfs-metastripe-03

Abstract

This Internet-Draft describes a means to add metadata striping to pNFS. The text of this draft is substantially based on prior drafts by Eisler, M., with some departures. The current draft attempts to define a somewhat lighter-weight protocol, in particular, seeks to permit striping for "filehandle only" operations such as LOCK and OPEN + CLAIM\_FH, without clients having to obtain metadata layouts on regular files. We gratefully acknowledge the primary contributions of Mike Eisler, Pranoop Ersani, and others.

Internet Draft Comments

Comments regarding this draft are solicited.

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 January 25, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| 1. Introduction and Motivation . . . . .                                | 4  |
| 2. Short List of Protocol Changes from Previous Drafts . . . . .        | 4  |
| 2.1. File-system wide Striping for Filehandle-Only Operations . . . . . | 4  |
| 2.2. Uniform Filehandles . . . . .                                      | 4  |
| 2.3. Simplified Multipath Device Model . . . . .                        | 4  |
| 2.4. Cookie Model . . . . .                                             | 5  |
| 2.5. LAYOUTCOMMIT . . . . .                                             | 5  |
| 2.6. Recommended Attributes . . . . .                                   | 5  |
| 2.6.1. meta_stripe_deviceid (deviceid4) . . . . .                       | 5  |
| 2.6.2. meta_stripe_count (uint32_t) . . . . .                           | 5  |
| 2.7. PREADDIR (Operation) . . . . .                                     | 5  |
| 3. Terminology . . . . .                                                | 6  |
| 4. Scope of Metadata Layouts . . . . .                                  | 6  |
| 4.1. Filehandle Striping . . . . .                                      | 7  |
| 4.2. Directory Striping . . . . .                                       | 7  |
| 4.2.1. Name-Based Operations . . . . .                                  | 8  |
| 4.2.2. Directory Enumeration . . . . .                                  | 8  |
| 5. The Metadata Striping Layout . . . . .                               | 9  |
| 5.1. Name . . . . .                                                     | 9  |
| 5.2. Value . . . . .                                                    | 9  |
| 5.3. Data Type Definitions . . . . .                                    | 9  |
| 5.3.1. Layout Hint . . . . .                                            | 9  |
| 5.3.2. Devices . . . . .                                                | 9  |
| 5.3.3. Metadata Layout . . . . .                                        | 10 |
| 5.3.4. Layoutupdate4 lou_body . . . . .                                 | 11 |
| 5.4. Metadata Layout Semantics . . . . .                                | 11 |
| 5.4.1. LAYOUTGET Argument Conventions . . . . .                         | 11 |
| 5.4.2. Filehandle Striping Layouts . . . . .                            | 12 |
| 5.4.2.1. Filehandle Stripe Hints . . . . .                              | 12 |
| 5.4.3. Directory Striping Layouts . . . . .                             | 13 |
| 5.4.3.1. L-MDS Selection for Name-based Operations . . . . .            | 13 |

|                                                              |    |
|--------------------------------------------------------------|----|
| 5.4.3.2. Directory Enumeration . . . . .                     | 15 |
| 5.5. LAYOUTCOMMIT . . . . .                                  | 16 |
| 5.6. Operation: PREADDIR - Parallel Read Directory . . . . . | 16 |
| 5.6.1. ARGUMENTS . . . . .                                   | 16 |
| 5.6.2. RESULTS . . . . .                                     | 16 |
| 5.6.3. DESCRIPTION . . . . .                                 | 16 |
| 5.6.4. IMPLEMENTATION . . . . .                              | 16 |
| 6. Further Considerations . . . . .                          | 16 |
| 6.1. Storage Access Protocols . . . . .                      | 17 |
| 6.2. Revocation of Layouts . . . . .                         | 17 |
| 6.3. Stateids . . . . .                                      | 17 |
| 6.4. Lease Terms . . . . .                                   | 18 |
| 6.5. Layout Operations Sent to an L-MDS . . . . .            | 18 |
| 6.6. Filehandles in Metadata Layouts . . . . .               | 18 |
| 6.7. Restriping . . . . .                                    | 19 |
| 6.7.1. Layout Recall Cases . . . . .                         | 19 |
| 6.7.2. Hint Invalidation . . . . .                           | 19 |
| 6.8. Recovery . . . . .                                      | 19 |
| 6.9. Failure and Restart of Client . . . . .                 | 19 |
| 6.10. Failure and Restart of Server . . . . .                | 19 |
| 7. Negotiation . . . . .                                     | 20 |
| 8. Usage Examples . . . . .                                  | 20 |
| 8.1. open-lock-write-close . . . . .                         | 20 |
| 8.2. parallel create-layoutcommit . . . . .                  | 20 |
| 8.3. parallel directory listing . . . . .                    | 21 |
| 9. Operational Recommendation for Deployment . . . . .       | 21 |
| 10. Acknowledgements . . . . .                               | 21 |
| 11. Security Considerations . . . . .                        | 21 |
| 12. IANA Considerations . . . . .                            | 22 |
| 13. References . . . . .                                     | 22 |
| 13.1. Normative References . . . . .                         | 22 |
| 13.2. Informative References . . . . .                       | 23 |
| Authors' Addresses . . . . .                                 | 23 |

## 1. Introduction and Motivation

The NFSv4.1 specification describes pNFS [NFSv4.1]. pNFS distributes (stripes) file data across multiple storage devices. In NFSv4.1, parallel access is limited to the data contents of regular files. Metadata is not distributed or striped: the model presented in the NFSv4.1 specification is that of a single metadata server. This document describes a means to add metadata striping to pNFS, which includes the notion of multiple metadata servers. With metadata striping, multiple metadata servers may work together to provide a higher parallel performance.

Two methods are described. The first, called filehandle striping, directs metadata operations associated with a file handle to a preferred metadata server. The second, called directory striping, distributes directory operations across a collection of metadata servers.

## 2. Short List of Protocol Changes from Previous Drafts

### 2.1. File-system wide Striping for Filehandle-Only Operations

Stripe hints redirect clients to a preferred metadata server for filehandle-only operations (below), but are backed by a single layout per-file system, rather than per-file, as in [METASTRIPE]. The new model is lighter weight, but since it remains layout-based retains the advantages of pNFS device indirection and garbage collection.

### 2.2. Uniform Filehandles

[METASTRIPE] offers implementations the option to propagate layout filehandles for all metadata layout types. Since it would be impossible to reasonably support this under the new proposed model for filehandle-only operations, we propose instead that L-MDS filehandles always be equivalent to I-MDS filehandles.

### 2.3. Simplified Multipath Device Model

[METASTRIPE] defines two different methods for encoding metadata server locations, only the "simple" model uses the pNFS device mechanism. In this draft, we propose a single model based on pNFS devices, in which there is a one-to-one mapping between devices and L-MDS servers. This approach facilitates sharing device addresses across layouts which have servers in common and also minimizes the difficulty of reclaiming devices no longer in use by any metadata layout.

## 2.4. Cookie Model

NFSv4 associates with each entry in a directory a unique value of type `cookie4`, a 64-bit integer. [METASTRIPE] involves cookies in stripe selection, and imposes specific requirements on cookie values. In the current proposal we treat cookies as opaque values except as specified in ordinary NFSv4.1. We concur with [METASTRIPE] that cookies **MUST** be unique within any logical directory regardless of the striping pattern. As in ordinary NFSv4.1, the behavior of `READDIR` (or `PREADDIR`, below) when cookie has a value previously returned to a client by the same server, but no longer associated with any directory entry, is not defined.

## 2.5. LAYOUTCOMMIT

In this draft, we introduce layout-subtype specific data for the `LAYOUTCOMMIT` operation.

## 2.6. Recommended Attributes

We propose two new recommended attributes.

`meta_stripe_deviceid` (`deviceid4`)

`meta_stripe_count` (`uint32_t`)

### 2.6.1. `meta_stripe_deviceid` (`deviceid4`)

An attribute of type `meta_stripe_deviceid` represents a filehandle stripe hint. This attribute **MUST NOT** be offered to clients unless they hold a valid filehandle striping layout on the containing file system.

### 2.6.2. `meta_stripe_count` (`uint32_t`)

The `meta_stripe_count` attribute represents, for directory objects, the directory's current stripe count, which may help the client decide if it will request a directory striping layout on the directory. This attribute **MAY** be offered only to clients which hold a filehandle striping layout on the containing file system.

## 2.7. `PREADDIR` (Operation)

The NFSv4.1 `READDIR` operation has insufficient information to perform all possible enumerations required in the proposed directory striping model. We propose a new `PREADDIR` operation which takes, in addition to all the current `READDIR` operations, also a controlling metadata layout `stateid` and stripe number.

### 3. Terminology

Initial Metadata Server (I-MDS). The I-MDS is the metadata server from which the client obtains a filehandle prior to acquiring any layout on the file.

Layout Metadata Server (L-MDS). The L-MDS is the metadata server from which the client obtains a filehandle from after redirection from a layout.

Regular file: An object of file type NF4REG or NF4NAMEDATTR.

Filehandle striping. Hint-based indirection to a preferred MDS for filehandle-based operations, backed by a filesystem-wide metadata layout.

Directory striping. Fine-grained, layout-based indirection for parallel operations on directories, using a striping pattern.

### 4. Scope of Metadata Layouts

This proposal assumes a model where there are two or more servers capable of supporting NFSv4.1 operations. At least one server is an I-MDS, and the I-MDS should be thought of as a normal NFSv4.1 server, with the additional capability of granting metadata layouts on demand. The I-MDS might also be capable of granting non-metadata layouts, but this is orthogonal to the scope of metadata striping.

The model also requires at least one additional server, an L-MDS, that is capable of supporting NFSv4.1 operations that are directed to the server by the I-MDS. It is permissible for an I-MDS to also be an L-MDS, and an L-MDS to also be an I-MDS. Indeed, a simple submodel is for every NFSv4.1 server in a set to be both an I-MDS and L-MDS.

For convenience, we divide NFSv4.1 metadata operations into three classes:

Filehandle-only. These are operations that take only filehandles as arguments, i.e. the current filehandle, or both the current filehandle and the saved filehandle, and no component names of files (e.g., LOCK, LAYOUTGET).

Name-based. These are operations that take one or two filehandles (i.e. the current filehandle, or both the current file handle and the saved filehandle) and one or two component names of files (e.g., LINK, RENAME).

Directory-enumeration. These are operations that take one filehandle and return the contents of a directory. Currently, NFSv4 has only one such operation, READDIR. This draft adds a section, PREADDIR.

Metadata striping applies to all of the foregoing NFSv4.x operations, and is of two types:

filehandle striping uses hints (attribute-based indications) backed by a filesystem-wide layout to direct clients to a preferred MDS on which to perform filehandle-only operations

directory striping uses fine-grained metadata layouts on directories to support execution of name-based operations (directory enumeration, creates) on a set of MDS servers in parallel

#### 4.1. Filehandle Striping

To avoid an explosion of new client state, a coarse-grained hinting mechanism is used to direct filehandle-only operations to a preferred metadata server.

As specified in 5.12.1 of [NFSv4.1], when a client encounters a file system which supports LAYOUT4\_METADATA, it can obtain a metadata layout of subtype LAYOUTMETA4\_FILEHANDLE, whose scope is the entire file system, using the LAYOUTGET operation on any filehandle object in the file system which it is permitted to access.

Then using ordinary READDIR and GETATTR requests, the client can obtain for any object in the file system a meta\_stripe\_deviceid attribute that indicates the preferred device to send filehandle-only or name-based operations for that object.

For example, suppose that after obtaining an ordinary filehandle via OPEN, a LAYOUTMETA4\_FILEHANDLE layout on the containing file system, and a meta\_stripe\_deviceid hint from a previous GETATTR, READDIR, or PREADDIR,, the client wants to get a byte range lock on the file. The client sends the LOCK request to the network address (pNFS device, L-MDS) indicated by the meta\_stripe\_deviceid attribute.

#### 4.2. Directory Striping

For name-based and directory enumeration operations, a more fine-grained, layout-based redirection mechanism is used.

When a client obtains a filehandle for an object that is of type directory and wishes to take advantage of metadata striping, the

client first obtains a metadata layout of subtype `LAYOUTMETA4_DIRECTORY` on the directory. The client is provided with a directory-specific list of network addresses (devices) to which to send requests specific to objects in that directory.

#### 4.2.1. Name-Based Operations

For name-based operations, the directory striping layout indicates the preferred destinations in the network to send name-based operations for that directory (e.g., `CREATE`). The preferred destinations **MUST** apply to the current filehandle that the operation uses. In other words, for `LINK` and `RENAME`, which take both the saved filehandle and the current filehandle as parameters, the pNFS client would use the stripe hint of the target directory (indicated in the current filehandle) for guidance where to send the operation. Note that if an L-MDS accepts a `LINK` or `RENAME` operation, the L-MDS **MUST** perform the operation atomically. If it cannot, then the L-MDS **MUST** return the error `NFS4ERR_XDEV`, and the client **MUST** send the operation to the I-MDS.

The choice of destination is a function of the name the client is requesting. For example, after the client obtains the filehandle of a directory via `LOOKUP` and the metadata layout via `LAYOUTGET`, the client wants to open a regular file within the directory. As with the `LAYOUT4_NFSV4_1_FILES` layout type, the client has a list network addresses to which to send requests. With the `LAYOUT4_NFSV4_1_FILES` layout, the choice of the index in the list of network addresses was computed from the offset of the read or write request. With the metadata layout, the choice of the index is derived from the name (or some other method, such as the name and one or more attributes of the directory, such as the filehandle, fileid, as below.) passed to `OPEN`.

#### 4.2.2. Directory Enumeration

For directory-enumeration operations, the directory striping layout indicates the preferred destination in the network to send `(P)READDIR` operations for that directory. For example, after the client obtains the filehandle of a directory via `LOOKUP` and the metadata layout via `LAYOUTGET`, the client wants to read the directory. As with the `LAYOUT4_NFSV4_1_FILES` layout type, the client has a list network addresses to which to send requests. With the `LAYOUT4_NFSV4_1_FILES` layout, the choice of the index in list of network addresses was computed from the offset of the read or write request. For directory striping layouts, the index counts from 0 to the directory stripe count, less 1.



## 5. The Metadata Striping Layout

### 5.1. Name

The name of the metadata striping layout type is LAYOUT4\_METADATA.

### 5.2. Value

The value of the metadata striping layout type is TBD1.

### 5.3. Data Type Definitions

#### 5.3.1. Layout Hint

```

///  %
///  /* Encoded in the loh_body field of type layouthint4: */
///  %
///  struct md_dirsize_layouthint4 {
///      uint64_t *mdlh_min_est;
///      uint64_t *mdlh_avg_est;
///      uint64_t *mdlh_max_est;
///      uint32_t *mdlh_stripe_count;
///      uint32_t *mdlh_stripe_modulus;
///  };

```

Figure 1

The layout-type specific layouthint4 content for the LAYOUT4\_METADATA layout type is composed of five fields, each optional. Using some combination of the mdlh\_min\_est, mdlh\_avg\_est, and mdlh\_max\_est fields, the client is enabled to give an indication of the directory workload it expects for a new directory. The client also may suggest an explicit stripe count or modulus preference in mdlh\_stripe\_count or mdlh\_stripe\_modulus, which SHOULD be congruent if specified together.

#### 5.3.2. Devices

```

///  % /*
///  % * Encoded in the da_addr_body field of data type
///  % * device_addr4:
///  % */
///  struct md_layout_addr4 {
///      multipath_list4      mdla_multipath_list<>;
///  };

```

Figure 2

## 5.3.3. Metadata Layout

```

/// enum md_layout_subtype4 {
///     LAYOUTMETA4_FILEHANDLE = 0,
///     LAYOUTMETA4_DIRECTORY
/// };

///
/// enum md_namebased_alg4 {
///     MDN_ALG_CITYHASH64 = 0,
///     MDN_ALG_CEPHFRAG = 1,
///     /* XXX TBD2 */
/// };
///

/// typedef uint32_t cephfrag4;
///
/// struct cephfragsplit4 {
///     cephfrag4 frag;
///     uint32_t bits;
/// };
///
/// enum cephash4 {
///     MDC_HASH_LINUX_DCACHE = 0,
///     MDC_HASH_RJENKINS = 1,
///     MDC_HASH_CITYHASH32 = 2,
/// };
///
/// struct md_namebased_alg_cephfrag4 {
///     enum cephash4 hash;
///     cephfragsplit4 fragtree<>;
/// };

/// struct md_layout_directory {
///     switch(enum md_namebased_alg4 mdl_n_namebased_alg) {
///         case MDN_ALG_CITYHASH64:
///             uint32_t mdl_n_cityhash_seed;
///         case MDN_ALG_CEPHFRAG:
///             md_namebased_alg_cephfrag4 mdl_n_cephfrag;
///     };
///
///     deviceid4 mdl_n_devicelist<>;
///     uint32_t mdl_n_stripe_pattern<>;
/// };

/// struct md_layout4 {
///     union md_layout_type
///         switch (enum md_layout_subtype4 subtype) {

```

```

    ///          case LAYOUTMETA4_FILEHANDLE:
    ///              void;
    ///          case LAYOUTMETA4_DIRECTORY:
    ///              md_layout_directory mdl_layout;
    ///      };
    ///  };

```

Figure 3

#### 5.3.4. Layoutupdate4 lou\_body

```

    ///
    /// struct md_directory_layoutupdate4 {
    ///     int32_t mdlu_entries_added;
    ///     int32_t mdlu_entries_removed;
    ///     nfstime4 mdlu_last_update;
    /// };
    ///
    /// % /*
    /// % * Encoded in the lou_body field of datatype
    /// % * layoutupdate4:
    /// % */
    /// struct md_layout_update4 {
    ///     union md_layout_type switch (enum md_layout_subtype4 subtype) {
    ///         case LAYOUTMETA4_FILEHANDLE:
    ///             void;
    ///         case LAYOUTMETA4_DIRECTORY:
    ///             md_directory_layoutupdate4 mlu_directory;
    ///     };
    /// };
    ///
    layoutupdate4 lou_body

```

Figure 4

#### 5.4. Metadata Layout Semantics

The reply to a successful LAYOUTGET request MUST contain exactly one element in `logr_layout`. The element contains the metadata layout.

##### 5.4.1. LAYOUTGET Argument Conventions

When a client requests a layout of type `LAYOUT4_METADATA`, it specifies the desired subtype, which MUST be one of `LAYOUTMETA4_FILEHANDLE` or `LAYOUTMETA4_DIRECTORY`, as the value of the LAYOUTGET `loga_iomode` argument. Server implementations should reject LAYOUTGET requests with other values for `loga_iomode`.

The value provided for `loga_stateid` may be any valid `stateid` for the related file or directory, or else the anonymous `stateid`.

The values provided for `loga_offset`, `loga_length`, and `loga_minlength` are not defined for metastripe layouts, and server implementations MUST NOT interpret these values.

#### 5.4.2. Filehandle Striping Layouts

If the requested layout is of subtype `LAYOUTMETA4_FILEHANDLE`, the value of the layout is void. The filehandle redirection information issued under auspices of the layout will be entirely in the form of filehandle striping attribute hints.

As noted in Section 4, the scope of filehandle striping layouts is an entire file system. The client can acquire the (singleton) filehandle striping layout for a given file system using any corresponding file handle which it happens to hold, and whose object the client is permitted to access. For example, the client could use the file handle of the first directory it traverses on a given file system, provided the file server is an NFSv4.x file server that supports layouts of type `LAYOUT4_METADATA`.

##### 5.4.2.1. Filehandle Stripe Hints

Filehandle stripe hints are objects of type `deviceid4`, and are the value of a new recommended, get-only attribute `meta_stripe_deviceid`.

A client may successfully obtain the `meta_stripe_deviceid` attribute on any file object if and only if it has successfully obtained a filehandle striping layout on the containing file system. Since the `meta_stripe_deviceid` hint is an ordinary NFSv4 attribute, the client may acquire it from a `GETATTR`, `READDIR`, or `PREADDIR` request. A server implementation SHOULD interpret a `PREADDIR` operation (which has a controlling metadata layout `stateid`) as a request for just those attributes that are appropriate for the layout `stateid` that has been presented.

At all events, when a client holds a filehandle stripe hint for a file object, it uses the `GETDEVICEINFO` operation to map the hint value to a device address of data type `md_layout_addr4` in the ordinary pNFS manner.

The server ensures that each such device remains accessible (unrecalled) for at least as long as any filehandle striping layout exists for which the device has been named in a hint.

### 5.4.3. Directory Striping Layouts

If the requested layout is of subtype `LAYOUTMETA4_DIRECTORY`, then the layout contains a <device list, striping pattern, algorithm> triple enabling the client to perform both parallel directory enumeration operations and stripe-aware name-based operations, as outlined in Section 4.

When the layout subtype is `LAYOUTMETA4_DIRECTORY`, the layout content provides an integer identifying a hashing algorithm, a list of deviceids, and a striping pattern. Then `mdl_n_namebased_alg` identifies an algorithm that maps a name, as a component4, to an integer. Each entry in the `mdl_n_devicelist` specifies a set of metadata servers that may be treated as equally valid for metadata requests to the same block in the partitioned namespace. Each entry in the stripe pattern is an index into the device list.

To perform a name based operation, the client maps the name to a number with the name based algorithm, looks that number up in the stripe pattern (modulo the length of the stripe pattern), yielding a device id that may be interpreted with `GETDEVICEINFO`, in the ordinary pNFS manner. After resolving the device id as a device address of data type `md_layout_addr4`, the client sends the request to any of the devices specified in the corresponding entry in the device list.

#### 5.4.3.1. L-MDS Selection for Name-based Operations

Clients with layouts of type `LAYOUTMETA4_DIRECTORY` may use the algorithm supplied in field `mdl_n_namebased_alg` of the layout content to compute a preferred L-MDS to use when performing name-based operations, as follows:

Let `F` be the function specified in `mdl_n_namebased_alg`;

Let `X = (x1, x2, x3, ...)` some set of inputs for function `F`, such that `x1` SHOULD be the component name of the file, and `x2, x3, ...` any additional parameters required for the chosen `F`, their arguments asserted to be values available to the client.

```
Let stripe_unit_number = F(X);
Let stripe_count = number of elements in mdl_layout.mdl_n_stripe_pattern;
Let idx =
    mdl_layout.mdl_n_stripe_pattern(stripe_unit_number % stripe_count);
Let deviceid = mdl_layout.mdl_n_devicelist[idx];
```

pseudocode

Figure 5

The client then selects an L-MDS indicated by the deviceid (using GETDEVICEINFO in the normal manner), and sends the name-based operation to that server.

#### 5.4.3.1.1. MDN\_ALG\_CITYHASH64

A layout with MDN\_ALG\_CITYHASH64 as the mdl\_n\_namebased\_alg indicates the use of the 64-bit CityHash non-cryptographic hashing function [CITY] for directory placement, with x1 the desired component name, and x2 the 32-bit seed value returned in the layout.

#### 5.4.3.1.2. MDN\_ALG\_CEPHFRAG

A layout with MDN\_ALG\_CEPHFRAG as the mdl\_n\_namebased\_alg indicates the use of Ceph's directory fragmentation algorithm for directory placement.

Ceph uses a recursive algorithm to partition the hash space of a directory into fragments, which are represented by an ordered list of splits called the fragtree. Fragments are split into powers of two, so each split stores this exponent in the field 'bits'.

Similarly, the cephfrag4 encodes in its high 8 bits the total number of bits 'n' it has split from the root fragment. In the next highest 'n' bits, it encodes its position in the hash space. If a given hash value 'v' matches these 'n' bits, the fragment is said to contain 'v'.

For example, starting with the root fragment root=0x00000000 and splitting by 2 bits, we generate the four fragments f1=0x02000000, f2=0x02400000, f3=0x02800000 and f4=0x02C00000. Further splitting f3 by 1 bit, we generate two new fragments g1=0x03800000 and g2=0x03A00000. The resulting fragtree for this structure would be { {0x00000000, 2}, {0x02800000, 1} }.

To place a given filename, calculate its hash value 'v' using the hash function indicated by the 'hash' enum. Then, starting with the root fragment f=0x00000000, follow these step recursively: \* Search for a split in the fragtree matching frag=f. If no split is found, place the file in fragment f. \* Given a split of 'n' bits, find which of the 2^n child fragments contains the hash value 'v'. Assign this child fragment to 'f' and continue.

#### 5.4.3.1.2.1. MDC\_HASH\_LINUX\_DCACHE

Specifies the use of the Linux dentry cache (needs reference) hashing function.

## 5.4.3.1.2.2. MDC\_HASH\_RJENKINS

Specifies the use of Robert Jenkins' [JENKINS] hashing function.

## 5.4.3.1.2.3. MDC\_HASH\_CITYHASH32

Specifies the use of the 32-bit CityHash [CITY] hashing function.

## 5.4.3.2. Directory Enumeration

Clients with layouts of type LAYOUTMETA4\_DIRECTORY may use the following algorithm to perform enumeration of striped directories preferred metadata servers, in parallel:

```

For stripe_number in 0 .. length(mdl_layout.mdln_stripe_pattern) -1
do
    Let stripe =
        mdl_layout.mdln_stripe_pattern[stripe_number];
    Let device = mdl_layout.mdln_devicelist[stripe];
    <PREADDIR at device, layout_stateid, stripe_number>

```

pseudocode

Figure 6

That is, for each logical stripe in the directory, the client notes stripe number (merely the stripe's offset in the sequence), and derives from it the corresponding index into mdln\_devicelist by indirection on mdln\_stripe\_pattern. The object at mdln\_devicelist[stripe\_number] is a device id, which the client maps to an L-MDS using GETDEVICEINFO, and performs a sequence of PREADDIR operations on that server. The PREADDIR operation behaves exactly as described in section 18.23.3 of [NFSv4.1], but takes in addition to the arguments of READDIR, a metadata layout stateid and stripe number.

As in ordinary NFSv4.1, to perform a full enumeration of the directory entries at each component L-MDS, the client commences iteration by sending a cookie argument of zero for the first PREADDIR operation in the current stripe, and continues performing PREADDIR operations supplying for the cookie argument the value of last cookie value returned in the prior PREADDIR operation in the same logical (L-MDS) enumeration only, until a PREADDIR operation indicates that no further entries are available. The client and server behavior for subsequent re-traversals of a previously-enumerated logical directory are exactly as in ordinary NFSv4.1, except with respect to entry and cookie partitioning as described here. The client SHOULD present to a component L-MDS only cookie values previously returned to that

client by that same L-MDS, or 0 to commence iteration. An L-MDS MAY reject with NFS4ERR\_BADCOOKIE PREADDIR operations using cookie values that are valid cookies for the logical directory, but which are local to another L-MDS segment.

### 5.5. LAYOUTCOMMIT

As filehandle striping layouts are effectively read-only, clients SHOULD NOT attempt commits on filehandle striping layouts. If a server implementation receives a LAYOUTCOMMIT for a valid filehandle striping layout, it SHOULD return NFS4ERR\_OK.

For metastripe layouts of subtype LAYOUTMETA4\_DIRECTORY, the layout specific data for LAYOUTCOMMIT contains the signed count of items added to and removed from the directory since the last LAYOUTCOMMIT operation.

### 5.6. Operation: PREADDIR - Parallel Read Directory

#### 5.6.1. ARGUMENTS

```

/// struct READDIR4args {
///     /* CURRENT_FH: directory */
///     nfs_cookie4      cookie;
///     verifier4        cookieverf;
///     count4           dircount;
///     count4           maxcount;
///     bitmap4          attr_request;
///     stateid4         layout_stateid;
///     uint32_t         stripe_number;
/// };

```

Figure 7

#### 5.6.2. RESULTS

```

/// typedef struct READDIR4res PREADDIR4res;

```

Figure 8

#### 5.6.3. DESCRIPTION

#### 5.6.4. IMPLEMENTATION

## 6. Further Considerations



### 6.1. Storage Access Protocols

The LAYOUT4\_METADATA layout type uses NFSv4.1 operations (and potentially, operations of higher minor versions of NFSv4, subject to the definition of a minor version of NFSv4) to access striped metadata. The LAYOUT4\_METADATA does not affect access to storage devices, and indeed, in the protocol described here, layouts of type LAYOUT4\_METADATA and ordinary pNFS layouts for parallel data access (e.g., LAYOUT4\_NFSV4\_1\_FILES, LAYOUT4\_OSD2\_OBJECTS, or LAYOUT4\_BLOCK\_VOLUME, or a future flexible files layout), are orthogonal.

### 6.2. Revocation of Layouts

Servers MAY revoke layouts of type LAYOUT4\_METADATA. A client detects if layout has been revoked if the operation is rejected with NFS4ERR\_PNFS\_NO\_LAYOUT. In NFSv4.1, the error NFS4ERR\_PNFS\_NO\_LAYOUT could be returned only by READ and WRITE. When the server returns a layout of type LAYOUT4\_METADATA, the set of operations that can return NFS4ERR\_PNFS\_NO\_LAYOUT is: ACCESS, CLOSE, COMMIT, CREATE, DELEGRETURN, GETATTR, LINK, LOCK, LOCKT, LOCKU, LOOKUP, LOOKUPP, NVERIFY, OPEN, OPENATTR, OPEN\_DOWNGRADE, PREADDIR, READ, READDIR, READLINK, REMOVE, RENAME, SECINFO, SETATTR, VERIFY, WRITE, GET\_DIR\_DELEGATION, SECINFO, SECINFO\_NO\_NAME, and WANT\_DELEGATION.

### 6.3. Stateids

The pNFS specification for LAYOUT4\_NFSV4\_1\_FILES states data servers MUST be aware of the stateids granted by MDS so that the stateids passed to READ and WRITE can be properly validated. Similarly, in layouts of type LAYOUT4\_METADATA, the L-MDS MUST be aware of layout stateids issued by the controlling I-MDS in the corresponding layout.

In addition, the L-MDS MUST be aware of any non-layout stateids granted by the I-MDS, if and only if the client is in contact the L-MDS under direction of a metadata layout returned by the I-MDS, and the I-MDS has not recalled or revoked that layout. In addition, because an L-MDS can accept operations like OPEN and LOCK that create or modify stateids, the I-MDS MUST be aware of stateids that an L-MDS has returned to a client, if and only if the I-MDS granted the client a metadata layout that directed the client to the L-MDS.

In some cases, one L-MDS MUST be aware of a stateid generated by another L-MDS. For example a client can obtain a stateid from the L-MDS serving as the destination of name-based operations, which includes OPEN. However, operations that use the stateid will be filehandle-only operations, and the L-MDS the OPEN operation is sent to might differ from the L-MDS the LOCK operation for the same target

file is sent to.

When a client obtains a non-layout stateid from an L-MDS, for example, as the result of an OPEN operation, the stateid is asserted to be valid at the issuing L-MDS, and also the associated I-MDS, as noted above. In addition, if the client holds a filehandle striping layout on the current file system, it SHOULD request the associated stripe hint on the object, ideally in the same COMPOUND.

When responding to client LAYOUTGET requests, server implementations MUST accept the anonymous stateid as a valid stateid for both LAYOUTMETA4\_FILEHANDLE and LAYOUTMETA4\_DIRECTORY layouts, but MAY return NFS4ERR\_BADSTATEID for other stateids, when appropriate.

#### 6.4. Lease Terms

Any state the client obtains from an I-MDS or L-MDS is guaranteed to last for an interval lasting as long as the maximum of the lease\_time attribute of the the I-MDS, and any L-MDS the client is directed to as the result of a metadata layout. The client has a lease for each client ID it has with an I-MDS or L-MDS, and each lease MUST be renewed separately for each client ID.

#### 6.5. Layout Operations Sent to an L-MDS

An L-MDS MAY allow a LAYOUTGET operation of type LAYOUT4\_METADATA. One reason the L-MDS might allow such a LAYOUTGET operation is to allow hierarchical striping. For example, for name-based operations, the pNFS server might use a radix tree, (which the field mdln\_namebased\_alg would indicate). The first four bytes of the component name would be combined to form a 32-bit stripe\_unit\_number. Once the client contacted the L-MDS, it would repeat the algorithm on the second four bytes of the component, and so on until the component name was exhausted.

More typically, an L-MDS MAY allow a LAYOUTGET operation of type LAYOUT4\_NFSV4\_1\_FILES, LAYOUT4\_OSD2\_OBJECTS, or LAYOUT4\_BLOCK\_VOLUME. Naturally, a reason to allow this would be for increased pNFS MDS scalability.

Once an L-MDS grants a layout, the client MUST use only the L-MDS that granted the layout to send LAYOUTUPDATE, LAYOUTCOMMIT, and LAYOUTRETURN.

#### 6.6. Filehandles in Metadata Layouts

Metadata layouts do not present filehandles.

## 6.7. Restriping

### 6.7.1. Layout Recall Cases

When a server implementation intends to perform restriping, it MUST ensure that it has successfully recalled any metadata layout which would be invalidated by the restriping.

If the implementation wishes to restripe a directory on which there are outstanding layouts of type LAYOUTMETA4\_DIRECTORY, it must first successfully recall these layouts at their controlling I-MDS servers, as described in [NFSv4.1].

If the implementation wishes to perform filehandle restriping which would invalidate any filehandle stripe hint which it has issued to clients, it MUST successfully recall all controlling layouts of type LAYOUTMETA4\_FILEHANDLE which would conflict with the restriping.

Naturally, if a client requests an L-MDS to perform any operation under the auspices of a metadata layout which is no longer valid, the L-MDS is not required to perform it. The L-MDS SHOULD fail the operation with NFS4ERR\_PNFS\_NO\_LAYOUT.

### 6.7.2. Hint Invalidation

When an implementation wishes to perform filehandle restriping that would invalidate an filehandle stripe hint or hints it has issued to clients, it can use ordinary NFSv4.1 invalidation to reclaim the hints. Since filehandle stripe hints are recommended attributes, the controlling I-MDS or L-MDS does this by updating the change attribute on the file being updated, as it would for any other file update.

## 6.8. Recovery

[[Comment.1: it is likely this section will follow that of the files layout type specified in the NFSv4.1 specification.]]

## 6.9. Failure and Restart of Client

TBD

## 6.10. Failure and Restart of Server

TBD

## 7. Negotiation

The NFSv4.x client sends a GETATTR operation for attribute `fs_layout_type`. If the reply contains the metadata layout type, then either or both of filehandle or directory striping are supported, subject to further verification by subsequent LAYOUTGET operations. If not, the client cannot use metadata striping.

## 8. Usage Examples

This section contains illustrative examples of the protocol.

### 8.1. open-lock-write-close

```
I-MDS: LAYOUTGET for filehandle layout -> fh_stateid
I-MDS: OPEN('foo') -> open_stateid
I-MDS: GETATTR(meta_stripe_deviceid) -> in_deviceid
I-MDS: GETDEVICEINFO(in_deviceid) -> [L-MDS]

L-MDS: LOCK(open_stateid) -> lock_stateid
L-MDS: WRITE(lock_stateid)
I-MDS: CLOSE(open_stateid)
I-MDS: LAYOUTRETURN(fh_stateid)
```

Figure 9

### 8.2. parallel create-layoutcommit

```
I-MDS: LAYOUTGET for filehandle layout -> fh_stateid
I-MDS: LAYOUTGET(dir) for directory layout
-> {dir_stateid, dir_deviceid, dir_placement}
I-MDS: GETDEVICEINFO(dir_deviceid)
-> [L-MDS1, L-MDS2, L-MDS3]

dir_placement('foo') -> L-MDS1
L-MDS1: CREATE(dir, 'foo')

dir_placement('bar') -> L-MDS2
L-MDS2: CREATE(dir, 'bar')

dir_placement('baz') -> L-MDS3
L-MDS3: CREATE(dir, 'baz')

I-MDS: LAYOUTCOMMIT(dir_stateid, +3)
I-MDS: LAYOUTRETURN(dn_stateid)
I-MDS: LAYOUTRETURN(fh_stateid)
```

Figure 10

### 8.3. parallel directory listing

```
I-MDS: LAYOUTGET for filehandle layout -> fh_stateid
I-MDS: LAYOUTGET(dir) for directory layout
-> dn_stateid, dn_deviceid
I-MDS: GETDEVICEINFO(dn_deviceid)
-> [L-MDS1, L-MDS2, L-MDS3]

L-MDS1: PREADDIR(dn_stateid, stripe=0, cookie=0)
-> [a, b, c]
L-MDS2: PREADDIR(dn_stateid, stripe=1, cookie=0)
-> [d, e, f]
L-MDS3: PREADDIR(dn_stateid, stripe=2, cookie=0)
-> [g, h, i]

I-MDS: LAYOUTRETURN(dn_stateid)
I-MDS: LAYOUTRETURN(fh_stateid)
```

Figure 11

## 9. Operational Recommendation for Deployment

Deploy the metadata striping layout when it is anticipated that the workload will involve a high fraction of non-I/O operations on filehandles.

## 10. Acknowledgements

We gratefully acknowledge the primary contributions of Mike Eisler, Pranoop Ersani, and others, in [METASTRIPE].

From prior drafts, Brent Welch had the idea of returning a separate device ID for filehandle-only operations in the metadata layout. Pranoop Erasani, Dave Noveck, and Richard Jernigan provided valuable feedback.

## 11. Security Considerations

The security considerations of Section 13.12 of [NFSv4.1] which are specific to data servers apply to l-MDSes. In addition, each l-MDS server and client are, respectively, a complete NFSv4.1 server and client, and so the security considerations of [NFSv4.1] apply to any client or server using the metadata layout type.

## 12. IANA Considerations

This specification requires an addition to the Layout Types registry described in Section 22.4 of [NFSv4.1]. The five fields added to the registry are:

1. Name of layout type: LAYOUT4\_METADATA.
2. Value of layout type: TBD1.
3. Standards Track RFC that describes this layout: RFCTBD2, which would be the RFC of this document.
4. How the RFC Introduces the specification: minor revision (we believe).
5. Minor versions of NFSv4 that can use the layout type: [TBD].

This specification requires the creation of a registry of hash algorithms for supporting the field `mdl_namebased_alg`. Additional details TBD.

This specification introduces two new recommended attributes (`meta_stripe_deviceid` and `meta_stripe_count`).

This specification introduces a new operation (`PREADDIR`).

## 13. References

### 13.1. Normative References

- [CITY] Pike and Alakuijala, "Introducing CityHash", April 2011, <<http://google-opensource.blogspot.com/2011/04/introducing-cityhash.html>>.
- [JENKINS] Jenkins, "Hash Functions for Hash Table Lookup", <<http://burtleburtle.net/bob/hash/evahash.html>>.
- [METASTRIPE] Eisler, "Metadata Striping for pNFS", October 2010, <<http://tools.ietf.org/html/draft-eisler-nfsv4-pnfs-metastripe-03>>.
- [NFSv4.1] Shepler, Eisler, and Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", January 2010, <<http://tools.ietf.org/html/rfc5661>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

### 13.2. Informative References

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

### Authors' Addresses

Matt Benjamin  
CohortFS, LLC  
206 S. Fifth Ave, Suite 150  
Ann Arbor, MI 48104  
USA

Phone: +1 734 761 4689  
Email: matt@cohortfs.com

Casey Bodley  
Email: casey@cohortfs.com

Adam C. Emerson  
Email: aemerson@cohortfs.com

Pranoop Ersani  
NetApp  
Email: Pranoop.Erasani@netapp.com

Peter Honeyman  
Email: peter.honeyman@gmail.com

