NFSv4 M. Eisler
Internet-Draft NetApp
Intended status: Standards Track October 18, 2010
Expires: April 21, 2011

Metadata Striping for pNFS
draft-eisler-nfsv4-pnfs-metastripe-02.txt

Abstract

This Internet-Draft describes a means to add metadata striping to pNFS.

Requirements Language

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 RFC 2119 [1].

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 April 21, 2011.

Copyright Notice

Table of Contents

1. Introduction and Motivation

The NFSv4.1 specification describes pNFS [2]. In NFSv4.1, pNFS is limited to the data contents of regular files. The content of regular files is distributed (striped) across multiple storage devices. Metadata is not distributed or striped, and indeed, 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.

This document does not require a new minor version of NFSv4. Instead, it requires a new layout type.

The XDR description is provided 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 metadata layout:

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

I.e. 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 > md.x
```

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

2. Terminology

- Initial Metadata Server (I-MDS). The I-MDS is the metadata server the client obtains a filehandle from prior to acquiring any layout on the file.
- Layout Metadata Server (L-MDS). The L-MDS is the metadata server the client obtains a filehandle from after direction from a layout.
- Regular file: An object of file type NF4REG or NF4NAMEDATTR.

3. Scope of Metadata Striping

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 irrelevant 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.

Metadata striping applies to all NFSv4.1 operations that operate on file objects. These operations can be broken down into three classes:

- Filehandle-only. These are operations that take just filehandles as arguments, i.e. the current filehandle, or both the current filehandle and the saved filehandle, and no component names of files. When a client obtains a filehandle of an file object from an NFS server, it can obtain a metadata layout that indicates the optimal destination in the network to send filehandle-only operations for that file object. For example, after obtaining the filehandle via OPEN, and the metadata layout via LAYOUTGET, the client wants to get a byte range lock on the file. The client sends the LOCK request to the network address specified in the metadata layout.

- Name-based. These are operations that take one or two filehandles (i.e. the current file handle, or both the current file handle and the saved filehandle) and one or two component names of files. When a client obtains a filehandle of a file object that is of type directory, it can obtain a metadata layout that indicates the optimal destinations in the network to send name-based operations for that directory. The optimal 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 metadata layout 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, etc.) passed to OPEN.

- o Directory-reading. These are operations that take one filehandle and return the contents of a directory (currently, NFSv4 has just one such operation, READDIR). When a client obtains a filehandle of a file object that is of type directory, it can obtain a metadata layout that indicates the optimal destination in the network to send directory reading 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. Since directories have cookies which resemble offsets, the choice of the index is computed from the the "cookie" argument to the operation.

4. The Definition of Metadata Striping Layout

4.1. Name of Metadata Striping Layout Type

The name of the metadata striping layout type is LAYOUT4_METADATA.

4.2. Value of Metadata Striping Layout Type

The value of the metadata striping layout type is TBD1.

4.3. Definition of the da_addr_body Field of the device_addr4 Data Type

```
///  %#include "nfs4_prot.h"
///  union md_layout_addr4 switch (bool mdla_simple) {
///    case TRUE:
///      multipath_list4              mdla_simple_addr;
///    case FALSE:
///      nfsv4_1_file_layout_ds_addr4 mdla_complex_addr;
///  };
```

Figure 1

If mdla_simple is TRUE, the remainder of the device address contains a list of elements (mdla_simple_addr), where each element represents a network address of an L-MDS which can serve equally as the target of metadata operations (typically the filehandle-only operations). See Section 13.5 of [2] for a description of how the multipath_list4 data type supports multi-pathing.

If mdla_simple is FALSE, the remainder of the device address is the same as the LAYOUT4_NFSV4_1_FILES device address, consisting of an array of lists of L-MDSes servers (nflda_multipath_ds_list), and an array of indices (nflda_stripe_indices). Each element of nflda_multipath_ds_list contains one or more subelements, and each subelement represents a network address of an L-MDS which may serve equally as the target of name-based and directory-reading operations (see Section 13.5 of [2]). The number of elements in nflda_multipath_ds_list array might be different than the stripe count. The stripe count is the number of elements in nflda_stripe_indices. The value of each element of nflda_stripe_indices is an index into nflda_multipath_ds_list, and thus the value of each element of nflda_stripe_indices MUST be less than the number of elements in nflda_multipath_ds_list.

4.4. Definition of the loh_body Field of the layouthint4 Data Type

```
///  enum md_layout_hint_care4 {
///          MD4_CARE_STRIPE_UNIT_SIZE    = 0x040,
///          MD4_CARE_STRIPE_CNT_NAMEOPS  = 0x080,
///          MD4_CARE_STRIPE_CNT_DIRRDOPS = 0x100
///  };
///  %
///  %/* Encoded in the loh_body field of type layouthint4: */
///  %
///  struct md_layouthint4 {
///          uint32_t        mdlh_care;
///          count4          mdlh_stripe_cnt_nameops;
///          count4          mdlh_stripe_cnt_dirrdops;
///          nfs_cookie4     mdlh_stripe_unit_size;
///  };
```

Figure 2

The layout-type specific content for the LAYOUT4_METDATA layout type is composed of four fields. The first field, mdlh_care, is a set of flags indicating which values of the hint the client cares about. If MD4_CARE_STRIPE_CNT_NAMEOPS is set, then the client indicates in the second field, mdlh_stripe_cnt_nameops the preferred stripe count for name-based operations. If MD4_CARE_STRIPE_CNT_DIRRDOPS is set, then the client indicates in the third field, mdlh_stripe_cnt_dirrdops, the preferred stripe count for directory-reading operations. If MD4_CARE_STRIPE_UNIT_SIZE is set, then the client indicates in the fourth field, mdlh_stripe_unit_size, the preferred stripe unit size for directory-reading operations.

4.5. Definition of the loc_body Field of the layout_content4 Data Type

```
///  struct md_layout_fhonly {
///    deviceid4   mdlf_devid;
///    nfs_fh4     mdlf_fh<1>;
///  };
///
///  struct md_layout_namebased {
///    deviceid4   mdln_devid;
///    uint32_t    mdln_namebased_alg;
///    uint32_t    mdln_first_index;
///    nfs_fh4     mdln_fh_list<>;
///  };
///
///  union md_layout_dirread_fhlist
///        switch (bool mdldf_use_namebased) {
///    case TRUE:
///      void;
///    case FALSE:
///      nfs_fh4     mdldf_fh_list<>;
///  };
///
///  struct md_layout_dirread {
///    deviceid4                 mdld_devid;
///    nfs_cookie4               mdld_first_cookie;
///    nfs_cookie4               mdld_unit_size;
///    uint32_t                  mdld_first_index;
///    md_layout_dirread_fhlist mdld_fh_list;
///  };
///
///  struct md_layout4 {
///    md_layout_fhonly    mdl_fhops_layout<1>;
///    md_layout_namebased mdl_nameops_layout<1>;
///    md_layout_dirread   mdl_dirrdops_layout_segments<>;
///  };
```

Figure 3

The reply to a successful LAYOUTGET request MUST contain exactly one element in logr_layout. The element contains the metadata layout. The metadata layout consists of three variable length arrays. At least one of the arrays MUST be of non-zero length.

- mdl_fhops_layout. This is an array of up to one element. If there is one element, the element indicates the preferred set L-MDSes as the target of filehandle-only operations. The element contains two fields, mdlf_devid, the pNFS device ID of the L-MDS

and mdlf_fh, an array of up to one filehandle.

When the client receives a layout that has a mdl_fhops_layout array with one element, it uses GETDEVICEINFO to map mdlf_devid to a device address, of data type md_layout_addr4. The value of the device address field mdla_simple MUST be TRUE. The client can then select any element in mdla_simple_addr to send a filehandle-only operation. The field mdlf_devid MUST map to a device address with mdla_simple set to TRUE. The current filehandle REQUIRED for use with the filehandle-only operation is either mdlf_fh[0] (if and only if mdlf_fh has one element) or it is the filehandle the pNFS client used as the current filehandle to the LAYOUTGET operation that returned the metadata layout.

- mdl_nameops_layout. This is an array of up to one element. If there is one element, the element indicates the preferred set of L-MDS servers to as the target of name-based operations. The list of L-MDSes is mapped from the mdln_devid device ID. The array mdln_fh_list is used to select a filehandle for accessing an L-MDS. The number of elements in this array MUST be one of three values:

  * Zero. The means that filehandles used for each L-MDS are the same as the filehandle used as the current filehandle to LAYOUTGET.

  * One. This means that every L-MDS uses filehandle in mdln_fh_list[0].

  * The same number of elements as mdla_complex_addr.nflda_multipath_ds_list. Thus, when sending a name-based operation to any L-MDS in mdla_complex_addr.nflda_multipath_ds_list[X], the filehandle in mdln_fh_list[X] MUST be used.

  The field mdld_first_index is the index into the first element of the of mdla_complex_addr.nflda_stripe_indices array to use. The field mdln_namebased_alg identifies the algorithm used to compute the actual element in the mdla_complex_addr.nflda_stripe_indices array to use.

  When the client receives a layout that has a mdl_nameops_layout array with one element, it uses GETDEVICEINFO to map mdln_devid to a device address of data type md_layout_addr4. The value of the device address field mdla_simple MUST be set to FALSE.

  The client determines the filehandle and the set of L-MDS network addresses to send a name-based operation via the following

algorithm:

```
let F be the function designated by
  mdln_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;

stripe_unit_number = F(X);
stripe_count = number of elements in
  mdla_complex_addr.nflda_stripe_indices;

j = (stripe_unit_number + mdln_first_index) %
  stripe_count;

idx = nflda_stripe_indices[j];

fh_count = number of elements in mdln_fh_list;
lmds_count = number of elements in
mdla_complex_addr.nflda_multipath_ds_list;

switch (fh_count) {
case lmds_count:
  fh = mdln_fh_list[idx];
  break;

case 1:
  fh = mdln_fh_list[0];
  break;

case 0:
  fh = current filehandle passed to LAYOUTGET;
  break;

default:
  throw a fatal exception;
  break;
}

address_list =
  mdla_complex_addr.nflda_multipath_ds_list[idx];
```

Figure 4

The client would then select an L-MDS from address_list, and send the name-based operation using the filehandle specified in fh.

If value of stripe_count is one, then in the above, the value of the stripe_unit_number derived from mdln_namebased_alg and the value of mdln_first_index will not change the index into nflda_stripe_indices because that index will always be zero. Hence when stripe_count is one, the value mdln_namedbased_alg does not matter. Thus, when mdla_complex_addr.nflda_stripe_indices has a length of one, the client MUST ignore the value of mdln_namebased_alg. This means that all name-based operations on the directory can be be sent any among the set of L-MDSes indicated in one element of mdla_complex_addr.nflda_multipath_ds_list. This serves the common case of where whole directories are distributed across a set L-MDSes, but the directories themselves are not striped.

- o mdl_dirops_layout_segments. This is an array of zero or more elements. Each element indicates the preferred set of L-MDSes as the preferred destination for directory reading operations and the pattern over which directory reading operations iterates over the L-MDSes. The set of L-MDSes is mapped from the value of the device ID in the field mdld_devid. The field mdld_first_cookie indicates the first directory entry cookie that a directory reading operation can use for the first unit of the pattern in this element. E.g., the value of mdld_first_cookie can be used as the value of the "cookie" field in READDIR4args. In the first element, mdld_first_cookie MUST be zero. The last cookie that can be used on the pattern can be no higher than one less than the value of mdld_first_cookie of the next element. If there is no next element, then the pattern is valid for all cookies from mdld_first_cookie through NFS4_UINT64_MAX inclusive. The field mdld_unit_size indicates the maximum number of cookies that can be read from each unit of a pattern, and thus indicates the lowest value of the "cookie" field in READDIR4args for each unit after the first unit. For example, if mdld_unit_size is 100000, and mdld_first_cookie is zero, then value of the "cookie" field in the READDIR4args of the READDIR operation sent to the second unit MUST be greater than or equal to 100000, and less than 200000. The field mdld_fh_list is used to select a filehandle for accessing an L-MDS. It is a switched union with a boolean discriminator mdldf_use_namebased. If mdldf_use_namebased is TRUE, then the array mdl_nameops_layout MUST be of length equal to one and the filehandle MUST be selected from mdl_nameops_layout.mdln_fh_list. Note however, that the device address MUST still be mapped from mdld_devid and not mdln_devid.

- o If mdldf_use_namebased is FALSE, then mlld_fh_list is present, and number of elements in mdld_fh_list MUST be one of three values:

* Zero. The means that filehandles used for each L-MDS are the same as the filehandle used as the current filehandle to LAYOUTGET.

* One. This means that every L-MDS uses the filehandle in mdld_fh_list[0].

* The same number of elements as mdld_complex_addr.nflda_multipath_ds_list. Thus, when sending a directory-reading operation to any L-MDS in mdld_complex_addr.nflda_multipath_ds_list[X], the filehandle in mdld_fh_list[X] MUST be used.

The field mdld_first_index is the index into the first element of the mdld_complex_addr.nflda_stripe_indices array to use.

When the client receives a layout that has a mdl_dirops_layout_segments array with more than zero elements, it uses GETDEVICEINFO to map the mdln_devid of each element of the array to a device address of data type md_layout_addr4. The value of the device address field mdla_simple MUST be set to FALSE. The client determines the filehandle and the set of L-MDS network addresses to send a name-based operation via the following algorithm:

```
let cookie_arg be the cookie the pNFS client will
  use as the value of the cookie argument to a
  directory reading operation;

segment_count = number of elements in
  mdl_dirrdops_layout_segments;

find index k, such that (cookie_arg >=
  mdl_dirrdops_layout_segments[k].mdld_first_cookie)
  && ((k == (segment_count - 1)) || (cookie_arg
  < mdl_dirrdops_layout_segments[k+1]));

relative_cookie = cookie_arg -
  mdl_dirrdops_layout_segments[k].mdld_first_cookie;

address = the result of GETDEVICEINFO on
  mdl_dirrdops_layout_segments[k].mdld_devid;

i = floor(relative_cookie /
  mdl_dirrdops_layout_segments[k].mdld_unit_size);

stripe_count = number of elements in
  address.mdla_complex_addr.nflda_stripe_indices;
```

```
  j = (stripe_unit_number + mdld_first_index) % stripe_count;

  idx = nflda_stripe_indices[j];
  lmds_count = number of elements in
      address.mdla_complex_addr.nflda_multipath_ds_list;

  if (mdl_dirrdops_layout_segments[k].
      mdldf_use_namebased == TRUE) {
    fh_count = number of elements in mdl_nameops_layout[0].mdln_fh_list;
      address.mdla_complex_addr.nflda_multipath_ds_list;
  } else {
    fh_count = number of elements in
      mdl_dirrdops_layout_segments[k].mdld_fh_list.
        mdldf_fh_list;
  }

  switch (fh_count) {
    case lmds_count:
      if (mdl_dirrdops_layout_segments[k].
          mdldf_use_namebased == TRUE) {
        fh = mdln_fh_list[idx];
      } else {
        fh = mdl_dirrdops_layout_segments[k].mdld_fh_list.
          mdldf_fh_list[idx];
      }
      break;

    case 1:
      if (mdl_dirrdops_layout_segments[k].
          mdldf_use_namebased == TRUE) {
        fh = mdln_fh_list[0];
      } else {
        fh = mdl_dirrdops_layout_segments[k].mdld_fh_list.
          mdldf_fh_list[0];
      }
      break;

    case 0:
      fh = current filehandle passed to LAYOUTGET;
      break;

    default:
      throw a fatal exception;
      break;
  }

  address_list =  address.mdla_complex_addr.
    nflda_multipath_ds_list[idx];
```

Figure 5

The client would then select an L-MDS from address_list, and send the directory-reading operation using the filehandle specified in fh. When the client is reading the beginning of the directory, cookie_arg is always zero. Subsequent directory-reading operations to read the rest of the directory will use the last cookie returned by the L-MDS. An MDS returning a metadata layout SHOULD return cookies that can be used directly to the I-MDS that returned the layout. However this might not always be possible. For example, the directory design of the filesystem of the MDS, might not return cookies in ascending order, or any order at all for that matter. Whereas, striping by definition requires an ordering. In such cases, if a directory is restriped while a pNFS client is reading its contents from the L-MDSes, it is possible that client will be unable to complete reading the directory, and as a result an error is returned to process reading the directory. To mitigate this, servers that have sent a CB_LAYOUTRECALL on the directory SHOULD NOT revoke the layout as long as they detect that the client is completing a read of the entire directory. Once a client has received a CB_LAYOUTRECALL, it SHOULD NOT send a directory-reading operation to an L-MDS with a cookie argument of zero. If the server has sent a CB_LAYOUTRECALL, the L-MDS SHOULD reject requests to read the directory that have a cookie argument zero and return the error NFS4ERR_PNFS_NO_LAYOUT.

4.6. Definition of the lou_body Field of the layoutupdate4 Data Type

```
///  %/*
///  % * LAYOUT4_METADATA.
///  % * Encoded in the lou_body field of type layoutupdate4:
///  % *      Nothing. lou_body is a zero length array of octets.
///  % */
///  %
```

Figure 6

The LAYOUT4_METADATA layout type has no content for lou_body filed of the layoutupdate4 data type.

4.7. 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. Thus a client might be able to obtain both a LAYOUT4_METADATA layout, and a non-LAYOUT4_METADATA layout type

(e.g., LAYOUT4_NFSV4_1_FILES, LAYOUT4_OSD2_OBJECTS, or LAYOUT4_BLOCK_VOLUME) on the same regular file. Of course, for a non-regular file, a pNFS client will be unable to get layouts of types LAYOUT4_NFSV4_1_FILES, LAYOUT4_OSD2_OBJECTS, or LAYOUT4_BLOCK_VOLUME).

4.8. 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, READ, READDIR, READLINK, REMOVE, RENAME, SECINFO, SETATTR, VERIFY, WRITE, GET_DIR_DELEGATION, SECINFO, SECINFO_NO_NAME, and WANT_DELEGATION.

4.9. 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. This requirement extends to the LAYOUT4_METADATA layout type: 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.

4.10. 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.

4.11. Layout Operations Sent to an L-MDS

An L-MDS MAY allow a LAYOUTGET operation. One reason the L-MDS might allow 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.

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

4.12. Filehandles in Metadata Layouts

The filehandles returned in a metadata layout are subject to becoming stale at any time. The L-MDS SHOULD NOT return NFS4ERR_STALE unless the I-MDS has recalled or revoked the corresponding layout.

4.13. READ and WRITE Operations

READ and WRITE are filehandle-only operations, and thus the pNFS client SHOULD attempt to obtain a non-metadata layout for a regular file. If it cannot, then it MAY use the metadata layout to send READ and WRITE operations to an L-MDS. An L-MDS MUST accept a READ or WRITE operation if the layout the I-MDS returned to the client included a filehandle-only layout.

4.14. Recovery

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

4.14.1. Failure and Restart of Client

TBD

4.14.2. Failure and Restart of Server

TBD

4.14.3. Failure and Restart of Storage Device

TBD

5. Negotiation

An pNFS client sends a GETATTR operation for attribute fs_layout_type. If the reply contains the metadata layout type, then metadata striping is supported, subject to further verification by a LAYOUTGET operation. If not, the client cannot use metadata striping.

6. 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.

7. Acknowledgements

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.

8. Security Considerations

The security considerations of Section 13.12 of [2] which are specific to data servers apply to lMDSes. In addition, each lMDS server and client are, respectively, a complete NFSv4.1 server and client, and so the security considerations of [2] apply to any client or server using the metadata layout type.

9. IANA Considerations

This specification requires an addition to the Layout Types registry described in Section 22.4 of [2]. The five fields added to the regisity are:

1. Name of layout type: LAYOUT4_METADATA

2. Value of layout type: TBD1.

3. Standards Track RFC that describes this layout: RFCTBD2, which is the RFC of this document.

4. How the RFC Introduces the specification: L.

5. Minor versions of NFSv4 that can use the layout type: 1.

This specification requires the creation of a registry of hash algorithms for supporting the field mdln_namebased_alg. Details TBD.

10. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[2] Shepler, S., Eisler, M., and D. Noveck, "NFS Version 4 Minor Version 1", draft-ietf-nfsv4-minorversion1-26 (work in progress), Sep 2008.

Author's Address

Mike Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
US

Phone: +1-719-599-9026
Email: mike@eisler.com