NFSv4 T. Haynes
Internet-Draft Hammerspace
Updates: 5661 (if approved) April 19, 2018
Intended status: Standards Track
Expires: October 21, 2018

Requirements for pNFS Layout Types
draft-ietf-nfsv4-layout-types-12.txt

Abstract

This document defines the requirements which individual pNFS layout types need to meet in order to work within the parallel NFS (pNFS) framework as defined in RFC5661. In so doing, it aims to clearly distinguish between requirements for pNFS as a whole and those specifically directed to the pNFS File Layout. The lack of a clear separation between the two set of requirements has been troublesome for those specifying and evaluating new Layout Types. In this regard, this document updates RFC5661.

Table of Contents

1. Introduction

The concept of layout type has a central role in the definition and implementation of Parallel Network File System (pNFS) (see [RFC5661]). Clients and servers implementing different layout types behave differently in many ways while conforming to the overall pNFS framework defined in [RFC5661] and this document. Layout types may differ as to:

- o The method used to do I/O operations directed to data storage devices.

- o The requirements for communication between the metadata server (MDS) and the storage devices.

- o The means used to ensure that I/O requests are only processed when the client holds an appropriate layout.

o The format and interpretation of nominally opaque data fields in pNFS-related NFSv4.x data structures.

Each layout type will define the needed details for its usage in the specifciation for that layout type; layout type specifications are always standards-track RFCs. Except for the files layout type, which was defined in Section 13 of [RFC5661], existing layout types are defined in their own standards-track documents and it is anticipated that new layout types will be defined in similar documents.

The file layout type was defined in the Network File System (NFS) version 4.1 protocol specification [RFC5661]. The block layout type was defined in [RFC5663] while the object layout type was defined in [RFC5664]. Subsequently, the SCSI layout type was defined in [RFC8154].

Some implementers have interpreted the text in Sections 12 ("Parallel NFS (pNFS)") and 13 ("NFSv4.1 as a Storage Protocol in pNFS: the File Layout Type") of [RFC5661] as both being applying only to the file layout type. Because Section 13 was not covered in a separate standards-track document such as those for both the block and object layout types, there had been some confusion as to the responsibilities of both the metadata server and the data servers (DS) which were laid out in Section 12.

As a consequence, authors of new specifications (see [FlexFiles] and [Lustre]) may struggle to meet the requirements to be a pNFS layout type. This document gathers the requirements from all of the original layout type standard documents and then specifies the requirements placed on all layout types independent of the particular type chosen.

2. Definitions

control communication requirement: is the specification for information on layouts, stateids, file metadata, and file data which must be communicated between the metadata server and the storage devices. There is a separate set of requirements for each layout type.

control protocol: is the particular mechanism that an implementation of a layout type would use to meet the control communication requirement for that layout type. This need not be a protocol as normally understood. In some cases the same protocol may be used as a control protocol and storage protocol.

storage protocol: is the protocol used by clients to do I/O operations to the storage device. Each layout type specifies the set of storage protocols.

loose coupling: is when the control protocol is a storage protocol.

tight coupling: is an arrangement in which the control protocol is one designed specifically for that purpose. It may be either a proprietary protocol, adapted specifically to a a particular metadata server, or one based on a standards-track document.

(file) data: is that part of the file system object which contains the data to be read or written. It is the contents of the object rather than the attributes of the object.

data server (DS): is a pNFS server which provides the file's data when the file system object is accessed over a file-based protocol. Note that this usage differs from that in [RFC5661] which applies the term in some cases even when other sorts of protocols are being used. Depending on the layout, there might be one or more data servers over which the data is striped. While the metadata server is strictly accessed over the NFSv4.1 protocol, the data server could be accessed via any file access protocol that meets the pNFS requirements.

See Section 2.1 for a comparison of this term and "data storage device".

storage device: is the target to which clients may direct I/O requests when they hold an appropriate layout. Note that each data server is a storage device but that some storage device are not data servers. See Section 2.1 for further discussion.

fencing: is the process by which the metadata server prevents the storage devices from processing I/O from a specific client to a specific file.

layout: is the information a client uses to access file data on a storage device. This information will include specification of the protocol (layout type) and the identity of the storage devices to be used.

The bulk of the contents of the layout are defined in [RFC5661] as nominally opaque, but individual layout types are responsible for specifying the format of the layout data.

layout iomode: is a grant of either read or read/write I/O to the client.

layout stateid: is a 128-bit quantity returned by a server that uniquely defines the layout state provided by the server for a specific layout that describes a layout type and file (see Section 12.5.2 of [RFC5661]). Further, Section 12.5.3 describes differences in handling between layout stateids and other stateid types.

layout type: is a specification of both the storage protocol used to access the data and the aggregation scheme used to lay out the file data on the underlying storage devices.

recalling a layout: is a graceful recall, via a callback, of a specific layout by the metadata server to the client. Graceful here means that the client would have the opportunity to flush any writes, etc., before returning the layout to the metadata server.

revoking a layout: is an invalidation of a specific layout by the metadata server. Once revocation occurs, the metadata server will not accept as valid any reference to the revoked layout and a storage device will not accept any client access based on the layout.

(file) metadata: is that part of the file system object that contains various descriptive data relevant to the file object, as opposed to the file data itself. This could include the time of last modification, access time, end-of-file (EOF) position, etc.

metadata server (MDS): is the pNFS server which provides metadata information for a file system object. It also is responsible for generating, recalling, and revoking layouts for file system objects, for performing directory operations, and for performing I /O operations to regular files when the clients direct these to the metadata server itself.

stateid: is a 128-bit quantity returned by a server that uniquely defines the set of locking-related state provided by the server. Stateids may designate state related to open files, to byte-range locks, to delegations, or to layouts.

2.1. Use of the Terms "Data Server" and "Storage Device"

In [RFC5661], these two terms of "Data Server" and "Storage Device" are used somewhat inconsistently:

o In chapter 12, where pNFS in general is discussed, the term "storage device" is used.

- o In chapter 13, where the file layout type is discussed, the term "data server" is used.

- o In other chapters, the term "data server" is used, even in contexts where the storage access type is not NFSv4.1 or any other file access protocol.

As this document deals with pNFS in general, it uses the more generic term "storage device" in preference to "data server". The term "data server" is used only in contexts in which a file server is used as a storage device. Note that every data server is a storage device but storage devices which use protocols which are not file access protocols (such as NFS) are not data servers.

Since a given storage device may support multiple layout types, a given device can potentially act as a data server for some set of storage protocols while simultaneously acting as a storage device for others.

2.2. 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 [RFC2119].

This document differs from most standards-track documents in that it specifies requirements for those defining future layout types rather than defining the requirements for implementations directly. This document makes clear whether:

(1) any particular requirement applies to implementations.

(2) any particular requirement applies to those defining layout types.

(3) the requirement is a general requirement which implementations need to conform to, with the specific means left to layout type definitions type to specify.

3. The Control Protocol

A layout type has to meet the requirements that apply to the interaction between the metadata server and the storage device such that they present to the client a consistent view of stored data and lock state (Section 12.2.6 of [RFC5661]). Particular implementations may satisfy these requirements in any manner they choose and the mechanism chosen need not be described as a protocol. Specifications defining layout types need to clearly show how implementations can

meet the requirements discussed below, especially with respect to those that have security implications. In addition, such specifications may find it necessary to impose requirements on implementations of the layout type to ensure appropriate interoperability.

In some cases, there may be no control protocol other than the storage protocol. This is often described as using a "loose coupling" model. In such cases, the assumption is that the metadata server, storage devices, and client may be changed independently and that the implementation requirements in the layout type specification need to ensure this degree of interoperability. This model is used in the block and object layout type specification.

In other cases, it is assumed that there will be a purpose-built control protocol which may be different for different implementations of the metadata server and data server. The assumption here is that the metadata server and data servers are designed and implemented as a unit and interoperability needs to be assured between clients and metadata-data server pairs, developed independently. This is the model used for the files layout.

Another possibility is for the definition of a control protocol to be specified in a standards-track document. There are two subcases to consider:

o A new layout type includes a definition of a particular control protocol whose use is obligatory for metadata servers and storage devices implementing the layout type. In this case the interoperability model is similar to the first case above and the defining document should assure interoperability among metadata servers, storage devices, and clients developed independently.

o A control protocol is defined in a standards-track document which meets the control protocol requirements for one of the existing layout types. In this case, the new document's job is to assure interoperability between metadata servers and storage devices developed separately. The existing definition document for the selected layout type retains the function of assuring interoperability between clients and a given collection of metadata servers and storage devices. In this context, implementations that implement the new protocol are treated in the same way as those that use an internal control protocol or a functional equivalent.

An example of this last case is the SCSI layout type [RFC8154], which extends the block layout type. The block layout type had a requirement for fencing of clients, but did not present a way for the

control protocol (in this case the SCSI storage protocol) to fence the client. The SCSI layout type remedies that in [RFC8154] and in effect has a tightly coupled model.

3.1. Control Protocol Requirements

The requirements of interactions between the metadata server and the storage devices are:

(1) The metadata server MUST be able to service the client's I/O requests if the client decides to make such requests to the metadata server instead of to the storage device. The metadata server must be able to retrieve the data from the constituent storage devices and present it back to the client. A corollary to this is that even though the metadata server has successfully given the client a layout, the client MAY still send I/O requests to the metadata server.

(2) The metadata server MUST be able to restrict access to a file on the storage devices when it revokes a layout. The metadata server typically would revoke a layout whenever a client fails to respond to a recall or a client's lease is expired due to non-renewal. It might also revoke the layout as a means of enforcing a change in locking state or access permissions that the storage device cannot directly enforce.

Effective revocation may require client co-operation in using a particular stateid (files layout) or principal (e,g., flexible files layout) when performing I/O.

In contrast, there is no requirement to restrict access to a file on the storage devices when a layout is recalled. It is only after the metadata server determines that the client is not gracefully returning the layout and starts the revocation that this requirement is enforced.

(3) A pNFS implementation MUST NOT allow the violation of NFSv4.1's access controls: ACLs and file open modes. Section 12.9 of [RFC5661] specifically lays this burden on the combination of clients, storage devices, and the metadata server. However the specification of the individual layout type might create requirements as to how this is to be done. This may include a possible requirement for the metadata server to update the storage device so that it can enforce security.

The file layout requires the storage device to enforce access whereas the flex file layout requires both the storage device and the client to enforce security.

(4) Interactions between locking and I/O operations MUST obey existing semantic restrictions. In particular, if an I/O operation would be invalid when directed at the metadata server, it is not to be allowed when performed on the storage device.

For the block and SCSI layout, as the storage device is not able to reject the I/O operation, the client is responsible for enforcing this requirement.

(5) Any disagreement between the metadata server and the data server as to the value of attributes such as modify time, the change attribute, and the EOF position MUST be of limited duration with clear means of resolution of any discrepancies being provided. Note that

(a) Discrepancies need not be resolved unless any client has accessed the file in question via the metadata server, typically by performing a GETATTR.

(b) A particular storage device might be striped such it has no information regarding the EOF position.

(c) Both clock skew and network delay can lead to the metadata server and the storage device having different values of the time attributes. As long as those differences can be accounted for in what is presented to the client in a GETATTR, then no violation results.

(d) A LAYOUTCOMMIT requires that changes in attributes resulting from operations on the storage device need to be reflected in the metadata server by the completion of the operation.

These requirements may be satisfied in different ways by different layout types. As an example, while the file layout type uses the stateid to fence off the client, there is no requirement that other layout types use this stateid approach.

Each new standards-track document for a layout types MUST address how the client, metadata server, and storage devices are to interact to meet these requirements.

3.2. Previously Undocumented Protocol Requirements

While not explicitly stated as requirements in Section 12 of [RFC5661], the existing layout types do have more requirements that they need to enforce.

The client has these obligations when making I/O requests to the storage devices:

(1) Clients MUST NOT perform I/O to the storage device if they do not have layouts for the files in question.

(2) Clients MUST NOT perform I/O operations outside of the specified ranges in the layout segment.

(3) Clients MUST NOT perform I/O operations which would be inconsistent with the iomode specified in the layout segments it holds.

Under the file layout type, the storage devices are able to reject any request made not conforming to these requirements. This may not be possible for other known layout types, which puts the burden of enforcing such violations solely on the client. For these layout types:

(1) The metadata server MAY use fencing operations to the storage devices to enforce layout revocation against the client.

(2) The metadata server MUST allow the clients to perform data I/O against it, even if it has already granted the client a layout. A layout type might discourage such I/O, but it can not forbid it.

(3) The metadata server MUST be able to do storage allocation, whether that is to create, delete, extend, or truncate files.

The means to address these requirements will vary with the layout type. A control protocol will be used to effect these, whether a purpose-built one, one identical to the storage protocol, or a new standards-track control protocol.

3.3. Editorial Requirements

This section discusses how the protocol requirements discussed above need to be addressed in documents specifying a new layout type. Depending on the interoperability model for the layout type in question, this may involve the imposition of layout-type-specific requirements that ensure appropriate interoperability of pNFS components which are developed separately.

The specification of the layout type needs to make clear how the client, metadata server, and storage device act together to meet the protocol requirements discussed previously. If the document does not impose implementation requirements sufficient to ensure that these

semantic requirements are met, it is not appropriate for publication as an IETF-stream RFC.

Some examples include:

o If the metadata server does not have a means to invalidate a stateid issued to the storage device to keep a particular client from accessing a specific file, then the layout type specification has to document how the metadata server is going to fence the client from access to the file on that storage device.

o If the metadata server implements mandatory byte-range locking when accessed directly by the client, then the layout type specification must require that this also be done when data is read or written using the designated storage protocol.

4. Specifications of Original Layout Types

This section discusses how the original layout types interact with Section 12 of [RFC5661], which enumerates the requirements of pNFS layout type specifications. It is not normative with regards to the file layout type presented in Section 13 of [RFC5661], the block layout type [RFC5663], and the object layout type [RFC5664]. These are discussed here only to illuminate the updates made in Section 3 of this document to Section 12 of [RFC5661].

4.1. File Layout Type

Because the storage protocol is a subset of NFSv4.1, the semantics of the file layout type comes closest to the semantics of NFSv4.1 in the absence of pNFS. In particular, the stateid and principal used for I/O MUST have the same effect and be subject to the same validation on a data server as it would have if the I/O were being performed on the metadata server itself. The same set of validations are applied whether pNFS is in effect or not.

And while for most implementations the storage devices can do the following validations:

(1) client holds a valid layout,

(2) client I/O matches the layout iomode, and,

(3) client does not go out of the byte ranges,

these are each presented as a "SHOULD" and not a "MUST". Actually, the first point is presented in [RFC5661] as both:

"MUST": in Section 13.6

> "As described in Section 12.5.1, a client MUST NOT send an I/O to a data server for which it does not hold a valid layout; the data server MUST reject such an I/O."

"SHOULD": in Section 13.8

> "The iomode need not be checked by the data servers when clients perform I/O. However, the data servers SHOULD still validate that the client holds a valid layout and return an error if the client does not."

It should be noted that it is just these layout specific checks that are optional, not the normal file access semantics. The storage devices MUST make all of the required access checks on each READ or WRITE I/O as determined by the NFSv4.1 protocol. If the metadata server would deny a READ or WRITE operation on a file due to its ACL, mode attribute, open access mode, open deny mode, mandatory byte-range lock state, or any other attributes and state, the storage device MUST also deny the READ or WRITE operation. Also while the NFSv4.1 protocol does not mandate export access checks based on the client's IP address, if the metadata server implements such a policy, then that counts as such state as outlined above.

The data filehandle provided by the PUTFH operation to the data server provides sufficient context to enable the data server to ensure that for the subsequent READ or WRITE operation in the compound, that the client has a valid layout for the I/O being performed.

Finally, the data server can check the stateid presented in the READ or WRITE operation to see if that stateid has been rejected by the metadata server in order to cause the I/O to be fenced. Whilst it might just be the open owner or lock owner on that client being fenced, the client should take the NFS4ERR_BAD_STATEID error code to mean it has been fenced from the file and contact the metadata server.

4.2. Block Layout Type

With the block layout type, the storage devices are generally not able to enforce file-based security. Typically, storage area network (SAN) disk arrays and SAN protocols provide coarse-grained access control mechanisms (e.g., Logical Unit Number (LUN) mapping and/or masking), with a target granularity of disks rather than individual blocks and a source granularity of individual hosts rather than of users or owners. Access to block storage is logically at a lower

layer of the I/O stack than NFSv4. Since NFSv4 security is not directly applicable to protocols that access such storage directly, Section 2.1 [RFC5663] specifies that:

> "in environments where pNFS clients cannot be trusted to enforce such policies, pNFS block layout types SHOULD NOT be used."

Due to these granularity issues, the security burden has been shifted from the storage devices to the client. Those deploying implementations of this layout type need to be sure that the client implementation can be trusted This is not a new sort of requirement in the context of SAN protocols. In such environments, the client is expected to provide block-based protection.

This shift of the burden also extends to locks and layouts. The storage devices are not able to enforce any of these and the burden is pushed to the client to make the appropriate checks before sending I/O to the storage devices. For example, the server may use a layout iomode only allowing reading to enforce a mandatory read-only lock, In such cases, the client has to support that use by not sending WRITEs to the storage devices. The fundamental issue here is that the storage device is treated by this layout type in the same fashion as a local disk device. Once the client has access to the storage device, it is able to perform both READ and WRITE I/O to the entire storage device. The byte ranges in the layout, any locks, the layout iomode, etc, can only be enforced by the client. Therefore, the client is required to provide that enforcement.

In the context of fencing off of the client upon revocation of a layout, these limitations come into play again, i.e., the granularity of the fencing can only be at the host/logical-unit level. Thus, if one of a client's layouts is revoked by the server, it will effectively revoke all of the client's layouts for files located on the storage units comprising the logical volume. This may extend to the client's layouts for files in other file systems. Clients need to be prepared for such revocations and reacquire layouts as needed.

4.3. Object Layout Type

With the object layout type, security checks occur during the allocation of the layout. The client will typically ask for layouts covering all of the file and may do so for either READ or READ/WRITE. This enables it to do subsequent I/O operations without the need to obtain layouts for specific byte ranges. At that time, the metadata server should verify permissions against the layout iomode, the file mode bits or ACLs, etc. As the client may be acting for multiple local users, it MUST authenticate and authorize the user by issuing

respective OPEN and ACCESS calls to the metadata server, similar to having NFSv4 data delegations.

Upon successful authorization, the client receives within the layout a set of object capabilities allowing it I/O access to the specified objects corresponding to the requested iomode. These capabilities are used to enforce access control and locking semantics at the storage devices. Whenever one of the following occur on the metadata server:

- o the permissions on the object change,
- o a conflicting mandatory byte-range lock is granted, or
- o a layout is revoked and reassigned to another client,

then the metadata server MUST change the capability version attribute on all objects comprising the file to in order to invalidate any outstanding capabilities before committing to one of these changes.

When the metadata server wishes to fence off a client to a particular object, then it can use the above approach to invalidate the capability attribute on the given object. The client can be informed via the storage device that the capability has been rejected and is allowed to fetch a refreshed set of capabilities, i.e., re-acquire the layout.

5. Summary

In the three original layout types, the burden of enforcing the security of NFSv4.1 can fall to either the storage devices (files), the client (blocks), or the metadata server (objects). Such choices are conditioned by the native capabilities of the storage devices - if a control protocol can be implemented, then the burden can be shifted primarily to the storage devices.

In the context of this document, we treat the control protocol as a set of requirements. And as new layout types are published, the defining documents MUST address:

(1) The fencing of clients after a layout is revoked.

(2) The security implications of the native capabilities of the storage devices with respect to the requirements of the NFSv4.1 security model.

In addition, these defining documents need to make clear how other semantic requirements of NFSv4.1 (e.g., locking) are met in the context of the proposed layout type.

6. Security Considerations

This section does not deal directly with security considerations for existing or new layout types. Instead, it provides a general framework for understating security-related issues within the pNFS framework. Specific security considerations will be addressed in the Security Considerations sections of documents specifying layout types. For example, in Section 5 of [RFC5663], the lack of finer-than-physical disk access control necessitates that the client is delegated the responsibility to enforce the access provided to them in the layout extent which they were granted by the metadata server.

The layout type specification must ensure that only data accesses consistent with the NFSV4.1 security model are allowed. It may do this directly, by providing that appropriate checks be performed at the time each access is performed. It may do it indirectly by allowing the client or the storage device to be responsible for making the appropriate checks. In the latter case, I/O access rights are reflected in layouts and the layout type must provide a way to prevent inappropriate access due to permissions changes between the time a layout is granted and the time the access is performed.

The metadata server MUST be able to fence off a client's access to the data file on a storage device. When it revokes the layout, the client's access MUST be terminated at the storage devices. The client has a subsequent opportunity to re-acquire the layout and perform the security check in the context of the newly current access permissions.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

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

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

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

[RFC5664] Halevy, B., Welch, B., and J. Zelenka, "Object-Based Parallel NFS (pNFS) Operations", RFC 5664, January 2010.

[RFC8154] Hellwig, C., "Parallel NFS (pNFS) Small Computer System Interface (SCSI) Layout", RFC 8154, DOI 10.17487/RFC8154, May 2017, <http://www.rfc-editor.org/info/rfc8154>.

8.2. Informative References

[FlexFiles] Halevy, B. and T. Haynes, "Parallel NFS (pNFS) Flexible File Layout", draft-ietf-nfsv4-flex-files-16 (Work In Progress), January 2018.

[Lustre] Faibish, S. and P. Tao, "Parallel NFS (pNFS) Lustre Layout Operations", draft-faibish-nfsv4-pnfs-lustre-layout-07 (Work In Progress), April 2014.

Appendix A. Acknowledgments

Dave Noveck provided an early review that sharpened the clarity of the definitions. He also provided a more comprehensive review of the document.

Both Chuck Lever and Christoph Helwig provided insightful comments during the WGLC.

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

Thomas Haynes
Hammerspace
4300 El Camino Real Ste 105
Los Altos, CA 94022
USA

Email: loghyr@gmail.com