NFSv4 T. Haynes
Internet-Draft Primary Data
Updates: 5661 (if approved) January 30, 2015
Intended status: Standards Track
Expires: August 3, 2015

Requirements for pNFS Layout Types
draft-ietf-nfsv4-layout-types-03.txt

Abstract

This document provides help in distinguishing between the requirements for Network File System (NFS) version 4.1's Parallel NFS (pNFS) and those 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. As this document clarifies RFC5661, it effectively updates RFC5661.

Table of Contents

1. Introduction

Both Parallel Network File System (pNFS) and the File Layout Type were defined in the Network File System (NFS) version 4.1 protocol specification, [RFC5661]. The Block Layout Type was defined in [RFC5663] and the Object Layout Type was in turn defined in [RFC5664].

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 strictly for the File Layout Type. I.e., since Section 13 was not covered in a separate RFC like those for both the Block and Object Layout Types, there is some confusion as to the responsibilities of both the Metadata Server (MDS) and the Data Servers (DS) which were laid out in Section 12.

As a consequence, new internet drafts (see [FlexFiles] and [Lustre]) may struggle to meet the requirements to be a pNFS Layout Type. This document clarifies what are the Layout Type independent requirements

placed on all Layout Types, whether one of the original three or any new variant.

2. Definitions

control protocol: is a set of requirements for the communication of information on layouts, stateids, file metadata, and file data between the metadata server and the storage devices.

(file) data: is that part of the file system object which describes the payload and not the object. E.g., it is the file contents.

Data Server (DS): is one of the pNFS servers which provide the contents of a file system object which is a regular file. Depending on the layout, there might be one or more data servers over which the data is striped. Note that while the metadata server is strictly accessed over the NFSv4.1 protocol, depending on the Layout Type, the data server could be accessed via any protocol that meets the pNFS requirements.

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

layout: informs a client of which storage devices it needs to communicate with (and over which protocol) to perform I/O on a file. The layout might also provide some hints about how the storage is physically organized.

layout iomode: describes whether the layout granted to the client is for read or read/write I/O.

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 the difference between a layout stateid and a normal stateid.

Layout Type: describes both the storage protocol used to access the data and the aggregation scheme used to lays out the file data on the underlying storage devices.

(file) metadata: is that part of the file system object which describes the object and not the payload. E.g., it could be the time since last modification, access, etc.

Metadata Server (MDS): is the pNFS server which provides metadata information for a file system object. It also is responsible for

generating layouts for file system objects. Note that the MDS is responsible for directory-based operations.

recalling a layout: is when the metadata server uses a back channel to inform the client that the layout is to be returned in a graceful manner. Note that the client could be able to flush any writes, etc., before replying to the metadata server.

revoking a layout: is when the metadata server invalidates the layout such that neither the metadata server nor any storage device will accept any access from the client with that layout.

stateid: is a 128-bit quantity returned by a server that uniquely defines the open and locking states provided by the server for a specific open-owner or lock-owner/open-owner pair for a specific file and type of lock.

storage device: is another term used almost interchangeably with data server. See Section 2.1 for the nuances between the two.

2.1. Difference Between a Data Server and a Storage Device

We defined a data server as a pNFS server, which implies that it can utilize the NFSv4.1 protocol to communicate with the client. As such, only the File Layout Type would currently meet this requirement. The more generic concept is a storage device, which can use any protocol to communicate with the client. The requirements for a storage device to act together with the metadata server to provide data to a client are that there is a Layout Type specification for the given protocol and that the metadata server has granted a layout to the client. Note that nothing precludes there being multiple supported Layout Types (i.e., protocols) between a metadata server, storage devices, and client.

As storage device is the more encompassing terminology, this document utilizes it over data server.

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].

3. The Control Protocol

In Section 12.2.6 of [RFC5661], the control protocol is introduced. There have been no specifications for control protocols, and indeed there need not be such a protocol in use for any given

implementation. The control protocol is actually a set of requirements provided to describe the interaction between the metadata server and the storage device. When specifying a new Layout Type, the defining document MUST show how it meets these requirements, especially with respect to the security implications.

3.1. Protocol Requirements

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

(1) NFSv4.1 clients MUST be able to access a file directly through the metadata server and not the storage device. I.e., the metadata server must be able to retrieve the data from the constituent storage devices and present it back to the client via normal NFSv4.1 operations. Whether the metadata server allows access over other protocols (e.g., NFSv3, Server Message Block (SMB), etc) is strictly an implementation choice.

(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 fails to renew its lease in time. It might also revoke the layout as a means of enforcing a change in state that the storage device cannot directly enforce with the client.

(3) Storage devices MUST NOT remove NFSv4.1's access controls: ACLs and file open modes.

(4) Locking MUST be respected.

(5) The metadata server and the storage devices MUST agree on attributes like modify time, the change attribute, and the end-of-file (EOF) position.

    Note that "agree" here means that some state changes need not be propagated immediately, although all changes SHOULD be propagated promptly.

Note that there is no requirement on how these are implemented. While the File Layout Type does use the stateid to fence off the client, there is no requirement that other Layout Types use this stateid approach. But the other Layout Types MUST document how the client, metadata server, and storage devices interact to meet these requirements.

3.2. Non-protocol Requirements

In gathering the requirements from Section 12 of [RFC5661], there are some which are notable in their absence:

(1) Storage device MUST honor the byte range restrictions present in the layout. I.e., if the layout only provides access to the first 2 MB of the file, then any access after that MUST NOT be granted.

(2) The enforcement of authentication and authorization so that restrictions that would be enforced by the metadata server are also enforced by the storage device. Examples include both export access checks and if the layout has an iomode of LAYOUTIOMODE4_READ, then if the client attempts to write, the I/O may be rejected.

While storage devices should make such checks on the layout iomode, [RFC5661] does not mandate that all Layout Types have to make such checks.

(3) The allocation and deallocation of storage. I.e., creating and deleting files.

Of these, the first two are of concern to this draft and Layout Types SHOULD honor them if at all possible,

3.3. Editorial Requirements

In addition to these protocol requirements, there are two editorial requirements for drafts that present a new Layout Type. At a minimum, the specification needs to address:

(1) The approach the new Layout Type takes towards fencing clients once the metadata server determines that the layout is revoked.

(2) The security considerations of the new Layout Type.

While these could be envisioned as one section in that the fencing issue might be the only security issue, it is recommended to deal with them separably.

The specification of the Layout Type should discuss how the client, metadata server, and storage device act together to meet the protocol requirements. I.e., if the storage device cannot enforce mandatory byte-range locks, then how can the metadata server and the client interact with the layout to enforce those locks?

4. Implementations in Existing Layout Types

4.1. File Layout Type

Not surprisingly, the File Layout Type comes closest to the normal semantics of NFSv4.1. In particular, the stateid used for I/O MUST have the same effect and be subject to the same validation on a data server as it would if the I/O was being performed on the metadata server itself in the absence of pNFS.

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

- client holds a valid layout,
- client I/O matches the layout iomode, and,
- client does not go out of the byte ranges,

these are each presented as a "SHOULD" and not a "MUST". However, 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. And note that 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.

As the data filehandle provided by the PUTFH operation and the stateid in the READ or WRITE operation are used to ensure that the client has a valid layout for the I/O being performed, the client can be fenced off for access to a specific file via the invalidation of either key.

4.2. Block Layout Type

With the Block Layout Type, the storage devices are not guaranteed to be able to enforce file-based security. Typically, storage area network (SAN) disk arrays and SAN protocols provide access control mechanisms (e.g., Logical Unit Number (LUN) mapping and/or masking), which operate at the granularity of individual hosts, not individual blocks. Access to block storage is logically at a lower layer of the I/O stack than NFSv4, and hence NFSv4 security is not directly applicable to protocols that access such storage directly. As such,

[RFC5663] is very careful to define that in environments where pNFS clients cannot be trusted to enforce such policies, pNFS Block Layout Types SHOULD NOT be used.

The implication here is that the security burden has shifted from the storage devices to the client. It is the responsibility of the administrator doing the deployment to trust the client implementation. However, this is not a new requirement when it comes to SAN protocols, the client is expected to provide block-based protection.

This implication also extends to ACLs, locks, and layouts. The storage devices might not be 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. As an example, if the metadata server uses a layout iomode for reading to enforce a mandatory read-only lock, then the client has to honor that intent by not sending WRITEs to the storage devices. The basic issue here is that the storage device can be treated as a local dumb disk such that once the client has access to the storage device, it is able to perform either READ or 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.

While the Block Layout Type does support client fencing upon revoking a layout, the above restrictions come into play again: the granularity of the fencing can only be at the host/logical-unit level. Thus, if one of a client's layouts is unilaterally revoked by the server, it will effectively render useless *all* of the client's layouts for files located on the storage units comprising the logical volume. This may render useless the client's layouts for files in other file systems.

4.3. Object Layout Type

The Object Layout Type focuses security checks to occur during the allocation of the layout. The client will typically ask for a layout for each byte-range of either READ or READ/WRITE. At that time, the metadata server should verify permissions against the layout iomode, the outstanding locks, 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, inside the layout, the client receives 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 at the storage devices. Whenever the metadata server detects one of:

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

then it MUST change the capability version attribute on all objects comprising the file to implicitly 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 published 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 decisions seem to be forced by the native capabilities of the storage devices - if a real control protocol can be implemented, then the burden can be shifted primarily to the storage devices.

But as we have seen, the control protocol is actually a set of requirements. And as new Layout Types are published, the enclosing documents minimally 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.

6. Security Considerations

The metadata server MUST be able to fence off a client's access to a file stored on a storage device. When it revokes the layout, the client's access MUST be terminated at the storage devices.

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.

8.2. Informative References

[FlexFiles] Halevy, B. and T. Haynes, "Parallel NFS (pNFS) Flexible File Layout", draft-ietf-nfsv4-flex-files-02 (Work In Progress), October 2014.

[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.

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
Primary Data, Inc.
4300 El Camino Real Ste 100
Los Altos, CA 94022
USA

Phone: +1 408 215 1519
Email: thomas.haynes@primarydata.com