Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 31 October 2026

T. Haynes
Hammerspace

29 April 2026

Deviceid-Scoped Layout Recall for NFSv4.2
draft-haynes-nfsv4-recalldevice-02

Abstract

The Parallel Network File System (pNFS) allows the metadata server to recall a layout from a client by file id, by file system id, or across all of a client's layouts. It also lets the server delete a deviceid via CB_NOTIFY_DEVICEID. It does not provide a mechanism for the metadata server to recall, in a single operation, all layouts that reference a specific deviceid. This document presents an extension to RFC7862 that adds a deviceid-scoped layout recall: a single CB_LAYOUTRECALL operation recalls every layout a client holds that references a given deviceid, leaving unrelated layouts in place. Without this capability, device unavailability can trigger large volumes of failed WRITE and LAYOUTERROR traffic, and administrators lack a surgical way to retire a deviceid without disturbing layouts that reference healthy devices.

Note to Readers

Discussion of this draft takes place on the NFSv4 working group mailing list (nfsv4@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Working Group information can be found at https://datatracker.ietf.org/wg/nfsv4/about/.

Status of This Memo

This Internet-Draft will expire on 31 October 2026.

Table of Contents

1. Introduction 2
1.1. Problem 1: Traffic amplification during device unavailability 3
1.2. Problem 2: Cleanly retiring a deviceid 4
1.3. What this document specifies 4
2. Requirements Language 5
3. Client Capability Advertisement 5
4. Extension to CB_LAYOUTRECALL - Recall Layout from Client . . 6
5. Extraction of XDR 7
6. Security Considerations 8
7. IANA Considerations 9
8. References 9
8.1. Normative References 9
8.2. Informative References 10
Acknowledgments 10
Author's Address 10

1. Introduction

In the Network File System version 4 (NFSv4) with a Parallel NFS (pNFS) metadata server ([RFC8881]), there is no mechanism for the metadata server to recall layouts from a client based on the deviceid (see Section 3.3.14 of [RFC8881]) those layouts reference. The Flex Files layout type ([RFC8435]) is the primary motivating use case: a Flex Files layout describes data spread across multiple data servers, each identified by a distinct deviceid, often deliberately deployed across separate power or fault domains so that a single domain failure does not lose data.

The absence of a deviceid-scoped recall has two distinct operational consequences. This section describes each, shows why the existing recall and notification mechanisms are the wrong shape to address it, and motivates the extension specified in the rest of the document.

1.1. Problem 1: Traffic amplification during device unavailability

When a deviceid becomes unavailable -- because a power fault domain has been isolated, because a device has been taken down for maintenance, or because of a transient I/O failure -- clients holding layouts that reference the deviceid continue to issue I/O against it. Each such attempt produces a WRITE (see Section 18.32 of [RFC8881]) that fails with NFS4ERR_NXIO (see Section 15.1.16.3 of [RFC8881]), followed by a LAYOUTERROR (see Section 15.6 of [RFC7862]) returned to the metadata server. The metadata server learns about the failure once per (client, layout) pair; the network and the metadata server pay for the retries.

Structurally, the wasted RPC count is approximately *2 x C x L x W*, where:

* *C* is the number of clients holding layouts that reference the affected deviceid,

* *L* is the average number of such layouts per client, and

* *W* is the number of write attempts per layout during the unavailability window.

The factor of two captures the WRITE-then-LAYOUTERROR pair generated per attempt. For a representative deployment with *C = 100*, *L = 50*, and a several-minute unavailability window in which each layout sees *W = 1000* write attempts, the wasted RPC count is on the order of *10 million*. These numbers are illustrative order-of-magnitude estimates derived from the structural relationship; deployment-specific measurements are out of scope for this specification, but the structural multiplier is what the extension eliminates.

A single deviceid-scoped recall delivered to each affected client replaces the (C x L x W) retry traffic with a single CB_LAYOUTRECALL per client plus the client-side processing to return the affected layouts.

The existing recall mechanisms are the wrong shape for this problem. LAYOUTRECALL4_ALL recalls every layout a client holds, including layouts pointing at healthy deviceids; this fixes the traffic amplification at the cost of evicting unrelated layouts and forcing them to be re-issued. LAYOUTRECALL4_FILE recalls layouts one file at

a time, which adds an RPC per affected file to the recovery cost; for a deployment where the number of affected files is large, the per-file recall traffic is itself a scaling concern.

1.2. Problem 2: Cleanly retiring a deviceid

The metadata server can signal to clients that a deviceid no longer exists by setting NOTIFY4_DEVICEID_DELETE in the CB_NOTIFY_DEVICEID callback (see Section 20.12 of [RFC8881]). This flag cannot be set while any layout still references the deviceid: once a delete is announced, a client cannot have in-flight I/O against a deviceid that no longer exists.

The result is that an administrator who wants to retire a deviceid -- for hardware refresh, decommission, capacity rebalancing, or removal of a failed device from the namespace -- has no surgical option. The available choices are:

* Wait for the affected layouts to expire naturally. The wait is bounded by a lease period and by the slowest client.

* Use LAYOUTRECALL4_ALL to recall every layout from every client. This evicts layouts pointing at healthy deviceids in the same operation, and may produce a thundering herd of re-LAYOUTGET requests once the recall completes.

* Use LAYOUTRECALL4_FILE for every affected file. This generates one recall RPC per file and is expensive at scale.

* Fence the deviceid (see Section 12.5.5 of [RFC8881]). This is disruptive: clients with active I/O see I/O failures rather than orderly recall, and the operation may invalidate concurrent legitimate I/O.

A deviceid-scoped recall provides the missing surgical option: the metadata server recalls exactly the layouts referencing the to-be-retired deviceid, waits for client acknowledgement, and then sends CB_NOTIFY_DEVICEID with NOTIFY4_DEVICEID_DELETE. The deviceid is retired without disturbing layouts that reference other deviceids.

1.3. What this document specifies

Using the process detailed in [RFC8178], the revisions in this document become an extension of NFSv4.2 [RFC7862]. They are built on top of the external data representation (XDR) [RFC4506] generated from [RFC7863].

The extension adds a deviceid-scoped recall to CB_LAYOUTRECALL: a single callback identifies a deviceid and the client returns every layout it holds that references that deviceid. The extension closes both operational gaps described above with a single new layout-recall scope.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Client Capability Advertisement

Before the server may send a CB_LAYOUTRECALL with LAYOUTRECALL4_DEVICEID, it MUST know that the client supports the new union arm. Per [RFC8178] Section 6, a server MUST NOT send a new callback operation or new union arm to a client that has not indicated support for it.

A client that supports LAYOUTRECALL4_DEVICEID signals this by setting a new flag in the eia_flags field of the EXCHANGE_ID operation (see Section 18.35 of [RFC8881]):

```
///    const EXCHGID4_FLAG_SUPP_RECALL_DEVICEID = 0x02000000;
///
```

A client that sets EXCHGID4_FLAG_SUPP_RECALL_DEVICEID in its EXCHANGE_ID request advertises that it supports handling CB_LAYOUTRECALL with a LAYOUTRECALL4_DEVICEID recall type.

A server MUST NOT send a CB_LAYOUTRECALL with LAYOUTRECALL4_DEVICEID to a client that did not set EXCHGID4_FLAG_SUPP_RECALL_DEVICEID in its EXCHANGE_ID request. A server that does not wish to support this capability MAY ignore the flag.

If the client does not set EXCHGID4_FLAG_SUPP_RECALL_DEVICEID, the server MAY fall back to recalling individual layouts via LAYOUTRECALL4_FILE (one CB_LAYOUTRECALL per layout file that references the unavailable deviceid), as described in Section 20.3 of [RFC8881]. This is less efficient but correct, and preserves interoperability with clients that predate this extension.

4. Extension to CB_LAYOUTRECALL - Recall Layout from Client

The original union layoutrecall4 (see Section 20.3.1 of [RFC8881]) is:

```
enum layoutrecall_type4 {
        LAYOUTRECALL4_FILE = LAYOUT4_RET_REC_FILE,
        LAYOUTRECALL4_FSID = LAYOUT4_RET_REC_FSID,
        LAYOUTRECALL4_ALL  = LAYOUT4_RET_REC_ALL
};

union layoutrecall4 switch(layoutrecall_type4 lor_recalltype) {
   case LAYOUTRECALL4_FILE:
           layoutrecall_file4 lor_layout;
   case LAYOUTRECALL4_FSID:
           fsid4              lor_fsid;
   case LAYOUTRECALL4_ALL:
           void;
   };
```

The proposed extension is:

```
///    const LAYOUT4_RET_REC_DEVICEID  = 4;
///
///    enum layoutrecall_type4 {
///           LAYOUTRECALL4_FILE     = LAYOUT4_RET_REC_FILE,
///           LAYOUTRECALL4_FSID     = LAYOUT4_RET_REC_FSID,
///           LAYOUTRECALL4_ALL      = LAYOUT4_RET_REC_ALL,
///           LAYOUTRECALL4_DEVICEID = LAYOUT4_RET_REC_DEVICEID
///   };
///
/// union layoutrecall4 switch(layoutrecall_type4 lor_recalltype) {
///   case LAYOUTRECALL4_FILE:
///           layoutrecall_file4 lor_layout;
///   case LAYOUTRECALL4_FSID:
///           fsid4              lor_fsid;
///   case LAYOUTRECALL4_DEVICEID:
///           deviceid4          lor_deviceid;
///   case LAYOUTRECALL4_ALL:
///           void;
///   };
```

Note that LAYOUT4_RET_REC_* constants are shared between layoutrecall_type4 (used in CB_LAYOUTRECALL) and layoutreturn_type4 (used in LAYOUTRETURN, see Section 18.44.1 of [RFC8881]). Adding LAYOUT4_RET_REC_DEVICEID = 4 therefore also extends layoutreturn_type4 with a LAYOUTRETURN4_DEVICEID value. A client that receives a LAYOUTRETURN with lor_recalltype set to LAYOUTRETURN4_DEVICEID and does not recognise it MUST return NFS4ERR_UNION_NOTSUPP per [RFC8178].

The server MUST NOT send CB_LAYOUTRECALL with LAYOUTRECALL4_DEVICEID to a client that has not set EXCHGID4_FLAG_SUPP_RECALL_DEVICEID (see Section 3). This satisfies the requirement in Section 6 of [RFC8178] that a server establish client awareness before sending new callback extensions.

The existing clora_iomode field in CB_LAYOUTRECALL4args (see Section 20.3.1 of [RFC8881]) applies normally: the client MUST return all layouts matching both the given deviceid and the given iomode. The server can determine that the client no longer has any layouts with the given deviceid and iomode once the client replies with NFS4ERR_NOMATCHING_LAYOUT.

5. Extraction of XDR

This document contains the external data representation (XDR) [RFC4506] description of the extension to CB_LAYOUTRECALL. The XDR description is presented in a manner that facilitates easy extraction into a ready-to-compile format. To extract the machine-readable XDR description, use the following shell script:

```
<CODE BEGINS>
#!/bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^ *///$??'
<CODE ENDS>
```

For example, if the script is named 'extract.sh' and this document is named 'spec.txt', execute the following command:

```
<CODE BEGINS>
sh extract.sh < spec.txt > recalldevice.x
<CODE ENDS>
```

This script removes leading blank spaces and the sentinel sequence '///' from each line. XDR descriptions with the sentinel sequence are embedded throughout the document.

Note that the XDR code contained in this document depends on types from the NFSv4.2 nfs4_prot.x file (generated from [RFC7863]). This includes both nfs types that end with a 4, such as offset4, length4, etc., as well as more generic types such as uint32_t and uint64_t.

The extracted XDR extends types that are already defined in [RFC7863] rather than introducing new types. An implementer MUST integrate the extracted block into the base XDR by *replacing* the existing layoutrecall_type4 enum declaration and layoutrecall4 union declaration with the extended versions in this document, and by adding the LAYOUT4_RET_REC_DEVICEID constant in the same file area as the existing LAYOUT4_RET_REC_* constants. Appending the extracted block verbatim to the RFC 7863 XDR produces duplicate- type errors at compile time.

The EXCHGID4_FLAG_SUPP_RECALL_DEVICEID constant is purely additive and is added to the EXCHGID4 flag constant block without replacing anything.

6. Security Considerations

The extension introduced by this document inherits the Security Considerations of CB_LAYOUTRECALL in Section 20.3 of [RFC8881] and of NFSv4.2 in [RFC7862]. Three aspects of the deviceid-scoped recall warrant explicit mention.

Blast radius: A single CB_LAYOUTRECALL with LAYOUTRECALL4_DEVICEID can cause a client to return many layouts in one callback exchange. This is not a new risk category -- LAYOUTRECALL4_FSID and LAYOUTRECALL4_ALL already allow wider-scoped recalls -- but administrators sizing callback-channel timeouts and client-side recall-handler resources SHOULD account for the possibility that a deviceid-scoped recall triggers per-layout cleanup work proportional to the number of layouts that reference the deviceid.

Cross-client scope: In deployments where a single deviceid is shared across clients serving different tenants or applications, a deviceid-scoped recall is cross-client by construction: every client that holds a layout referencing the deviceid is recalled. This is the same property as LAYOUTRECALL4_FSID and does not introduce a new information-exposure surface. Deployments that prefer per-tenant failure-domain isolation MAY treat this extension as an operational benefit: by assigning distinct deviceids per tenant, deviceid-scoped recall becomes a per-tenant operation with no cross-tenant effect.

Capability negotiation as defence in depth: The capability flag

defined in Section 3 ensures that a server cannot silently change the callback wire format for a client that did not opt in. A client that does not advertise EXCHGID4_FLAG_SUPP_RECALL_DEVICEID continues to receive only the layoutrecall_type4 values defined in [RFC8881]. A server that sends LAYOUTRECALL4_DEVICEID to a client that did not advertise support is in violation of Section 6 of [RFC8178]; clients that encounter such a violation MUST return NFS4ERR_UNION_NOTSUPP and SHOULD log the server for operator attention.

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <https://www.rfc-editor.org/rfc/rfc4506>.

[RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/rfc/rfc7862>.

[RFC7863] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 External Data Representation Standard (XDR) Description", RFC 7863, DOI 10.17487/RFC7863, November 2016, <https://www.rfc-editor.org/rfc/rfc7863>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/rfc/rfc8178>.

[RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/rfc/rfc8881>.

8.2. Informative References

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/rfc/rfc1813>.

[RFC8435] Halevy, B. and T. Haynes, "Parallel NFS (pNFS) Flexible File Layout", RFC 8435, DOI 10.17487/RFC8435, August 2018, <https://www.rfc-editor.org/rfc/rfc8435>.

Acknowledgments

Trond Myklebust and Paul Saab were involved in the initial requirements for this functionality.

Brian Pawlowski and Gorry Fairhurst helped guide this process.

Author’s Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com