Internet Engineering Task Force (IETF)
Request for Comments: 9766
Category: Standards Track
ISSN: 2070-1721

T. Haynes
T. Myklebust
Hammerspace

April 2025

Extensions for Weak Cache Consistency in NFSv4.2's Flexible File Layout

Abstract

This document specifies extensions to NFSv4.2 for improving Weak Cache Consistency (WCC). These extensions introduce mechanisms that ensure partial writes performed under a Parallel NFS (pNFS) layout remain coherent and correctly tracked. The solution addresses concurrency and data integrity concerns that may arise when multiple clients write to the same file through separate data servers. By defining additional interactions among clients, metadata servers, and data servers, this specification enhances the reliability of NFSv4 in parallel-access environments and ensures consistency across diverse deployment scenarios.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9766.

Table of Contents

1. Introduction

In the Parallel NFS (pNFS) flexible file layout (see [RFC8435]), there is no mechanism for the data servers to update the metadata servers when the data portion of the file is modified. The metadata server needs this knowledge to correspondingly update the metadata portion of the file. If the client is using NFSv3 as the protocol with the data server, it can leverage Weak Cache Consistency (WCC) to update the metadata server of the attribute changes. In this document, we introduce a new operation called LAYOUT_WCC to NFSv4.2, which allows the client to periodically report the attributes of the data files to the metadata server.

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].

1.1. Definitions

For a more comprehensive set of definitions, see Section 1.1 of [RFC8435].

(file) data: that part of the file system object that 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): a pNFS server that provides the file's data when the file system object is accessed over a file-based protocol.

(file) metadata: the 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, EOF position, etc.

metadata server (MDS): the pNFS server that provides metadata information for a file system object.

storage device: 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 of [RFC8434] for a discussion on the difference between a data server and a storage device.)

weak cache consistency (WCC): the mechanism in NFSv3 that allows the client to check for file attribute changes before and after an operation (see Section 2.6 of [RFC1813]).

1.2. 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.

2. Weak Cache Consistency (WCC)

A pNFS layout type enables the metadata server to inform the client of both the storage protocol and the locations of the data that the client should use when communicating with the storage devices. The flexible file layout type, as specified in [RFC8435], describes how data servers using NFSv3 can be accessed. The client is restricted to performing the following NFSv3 operations on the filehandles provided in the layout: READ, WRITE, and COMMIT (see Sections 3.3.6, 3.3.7, and 3.3.21 of [RFC1813], respectively). In other words, the client may only use NFSv3 operations that act directly on the data portion of the file.

Because there is no control protocol (see [RFC8434]) possible with all data servers, NFSv3 is used as the control protocol. As such, the following NFSv3 operations are commonly used by the metadata server: CREATE, GETATTR, and SETATTR (see Sections 3.3.8, 3.3.1, and 3.3.2 of [RFC1813], respectively). That is, the metadata server is only allowed to use NFSv3 operations that directly act on the metadata portion of the data file. GETATTR allows the metadata server to mainly retrieve the mtime (modify time), ctime (change time), and atime (access time). The metadata server can use this information to determine if the client modified the file whilst it held an iomode of LAYOUTIOMODE4_RW (see Section 3.3.20 of [RFC8881]). Then it can determine the following for the metadata file: time_modify, time_metadata, and time_access (see Sections 5.8.2.43, 5.8.2.42, and 5.8.2.37 of [RFC8881], respectively). That is, it can determine the information to return to clients in an NFSv4.2 GETATTR response.

For example, the metadata server might issue an NFSv3 GETATTR operation to the data server, which is typically triggered by a client's NFSv4 GETATTR request to the metadata server. In addition to the cost of each individual GETATTR operation, the data server can be overwhelmed by a large volume of such requests. NFSv3 addressed a similar challenge by including a post-operation attribute in the READ and WRITE operations to report WCC data (see Section 2.6 of [RFC1813]).

Each NFSv3 operation entails a single round trip between the client and server. Consequently, issuing a WRITE followed by a GETATTR would require two round trips. In that situation, the retrieved attribute information is regarded as having strict server-client consistency. By contrast, NFSv4 enables a WRITE and GETATTR to be combined within a compound operation, which requires only one round trip. This combined approach is likewise considered to have strict server-client consistency. Essentially, NFSv4 READ and WRITE operations omit post-operation attributes, allowing the client to determine whether it requires that information.

Whilst NFSv4 got rid of the requirement for WCC information to be supplied by the WRITE or READ operations, the introduction of pNFS reintroduces the same problem. The metadata server has to communicate with the data server in order to get the data that could be provided by a WCC model.

With the flexible file layout type, the client can leverage the NFSv3 WCC to service the proxying of times (see Section 5 of [RFC9754]), but the granularity of this data is limited. With client-side mirroring (see Section 8 of [RFC8435]), the client has to aggregate the N mirrored files in order to send one piece of information instead of N pieces of information. Also, the client is limited to sending that information only when it returns the delegation.

This document introduces a new NFSv4.2 operation, LAYOUT_WCC, which enables the client to provide the metadata server with information obtained from the data server. The client is responsible for gathering the NFSv3 WCC data, returned by the three permissible NFSv3 operations, and conveying it back to the metadata server as part of NFSv4.2 attributes. The metadata server MAY therefore avoid issuing costly NFSv3 GETATTR calls to the data servers. Because this approach relies on a weak model, the metadata server MAY still perform these calls if it chooses to strengthen the model.

3. Operation 77: LAYOUT_WCC - Layout Weak Cache Consistency

3.1. ARGUMENT

```
<CODE BEGINS>
/// struct LAYOUT_WCC4args {
///         stateid4        lowa_stateid;
///         layouttype4     lowa_type;
///         opaque          lowa_body<>;
/// };
<CODE ENDS>
```

stateid4 is defined in Section 3.3.12 of [RFC8881]. layouttype4 is defined in Section 3.3.13 of [RFC8881].

3.2. RESULT

```
<CODE BEGINS>
/// struct LAYOUT_WCC4res {
///         nfsstat4                lowr_status;
/// };
<CODE ENDS>
```

nfsstat4 is defined in Section 3.2 of [RFC8881].

3.3. DESCRIPTION

The current filehandle and the lowa_stateid identify the specific layout for the LAYOUT_WCC operation. The lowa_type indicates how to interpret the layout-type-specific payload contained in the lowa_body field. The lowa_type is the corresponding value from the "pNFS Layout Types" IANA registry for the layout type being used.

The lowa_body contains the data file attributes. The client is responsible for mapping NFSv3 post-operation attributes to the fattr4 representation. Similar to the behavior of post-operation attributes, the client may ignore these attributes, and the server may also choose to ignore any attributes included in LAYOUT_WCC. However, the server can use these attributes to avoid querying the data server for data file attributes. Because these attributes are optional and the client has no recourse if the server opts to disregard them, there is no requirement to return a bitmap4 indicating which attributes have been accepted in the LAYOUT_WCC result.

3.4. Implementation

3.4.1. Examples of When to Use LAYOUT_WCC

The only way for the metadata server to detect modifications to the data file is to probe the data servers via a GETATTR. It can compare the mtime results across multiple calls to detect an NFSv3 WRITE operation by the client. Likewise, the atime results indicate the client having issued an NFSv3 READ operation. As such, the client can leverage the LAYOUT_WCC operation whenever it has the belief that

the metadata server would need to refresh the attributes of the data files. While the client can send a LAYOUT_WCC at any time, there are times it will want to do this operation in order to avoid having the metadata server issue NFSv3 GETATTR requests to the data servers:

* Whenever it sends a GETATTR for any of the following attributes:

  - size (see Section 5.8.1.5 of [RFC8881])

  - space_used (see Section 5.8.2.35 of [RFC8881])

  - change (see Section 5.8.1.4 of [RFC8881])

  - time_access (see Section 5.8.2.37 of [RFC8881])

  - time_metadata (see Section 5.8.2.42 of [RFC8881])

  - time_modify (see Section 5.8.2.43 of [RFC8881])

* Whenever it sends an NFS4ERR_ACCESS error via LAYOUTRETURN or LAYOUTERROR. It could have already gotten the NFSv3 uid and gid values back in the WCC of the WRITE, READ, or COMMIT operation that got the error. Thus, it could report that information back to the metadata server, saving it from querying that information via an NFSv3 GETATTR.

* Whenever it sends a SETATTR to refresh the proxied times (see Section 5 of [RFC9754]). The metadata server will correlate these times in order to detect later modification to the data file.

3.4.2. Examples of What to Send in LAYOUT_WCC

The NFSv3 attributes returned in the WCC of WRITE, READ, and COMMIT operations are a smaller subset of what can be transmitted as an NFSv4 attribute. The mapping of NFSv3 to NFSv4 attributes is shown in Table 1. The LAYOUT_WCC MUST provide all of these attributes to the metadata server. Both the uid and gid are stringified into their respective attributes of owner and owner_group. In the case of NFS4ERR_ACCESS, the reason to provide these two attributes is that the metadata server can compare what it expects the values of the uid and gid of the data file to be versus the actual values. It can then repair the permissions as needed or modify the expected values it has cached.

NFSv3 Attribute	NFSv4.2 Attribute
size	size
used	space_used
mode	mode
uid	owner
gid	owner_group
atime	time_access
mtime	time_modify
ctime	time_metadata

Table 1: NFSv3 to NFSv4.2 Attribute Mappings

3.5. Allowed Errors

The LAYOUT_WCC operation can raise the errors listed in Table 2. When an error is encountered, the metadata server can decide to ignore the entire operation, or depending on the layout-type-specific payload, it could decide to apply a portion of the payload. Note that there are no new errors introduced for the LAYOUT_WCC operation and the errors in Table 2 are each defined in Section 15.1 of [RFC8881]. Table 2 can be considered as an extension of Section 15.2 of [RFC8881].

Operation	Errors
LAYOUT_WCC	NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_EXPIRED, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NO_GRACE, NFS4ERR_OLD_STATEID, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_UNKNOWN_LAYOUTTYPE, NFS4ERR_WRONG_CRED, NFS4ERR_WRONG_TYPE

Table 2: Operations and Their Valid Errors

3.6. Extension of Existing Implementations

The new LAYOUT_WCC operation is OPTIONAL for both NFSv4.2 [RFC7863] and the flexible file layout type [RFC8435].

3.7. Flexible File Layout Type

```
<CODE BEGINS>
/// struct ff_data_server_wcc4 {
///             deviceid4           ffdsw_deviceid;
///             stateid4            ffdsw_stateid;
///             nfs_fh4             ffdsw_fh_vers<>;
///             fattr4              ffdsw_attributes;
/// };
///
/// struct ff_mirror_wcc4 {
///             ff_data_server_wcc4 ffmw_data_servers<>;
/// };
///
/// struct ff_layout_wcc4 {
///             ff_mirror_wcc4      fflw_mirrors<>;
/// };
<CODE ENDS>
```

The results specific to the flexible file layout type MUST correspond to the ff_layout4 data structure as defined in Section 5.1 of [RFC8435]. There MUST be a one-to-one correspondence between the following:

* ff_data_server4 -> ff_data_server_wcc4

* ff_mirror4 -> ff_mirror_wcc4

* ff_layout4 -> ff_layout_wcc4

Each ff_layout4 has an array of ff_mirror4, which has an array of ff_data_server4. Based on the current filehandle and the lowa_stateid, the server can match the reported attributes.

But the positional correspondence between the elements is not sufficient to determine the attributes to update. Consider the case where a layout has three mirrors and two of them have updated attributes but the third does not. A client could decide to present all three mirrors, with one mirror having an attribute mask with no attributes present. Or it could decide to present only the two mirrors that had been changed.

In either case, the combination of ffdsw_deviceid, ffdsw_stateid, and ffdsw_fh_vers will uniquely identify the attributes to be updated. All three arguments are required. A layout might have multiple data files on the same storage device, in which case the ffdsw_deviceid and ffdsw_stateid would match, but the ffdsw_fh_vers would not.

The ffdsw_attributes are processed similar to the obj_attributes in the SETATTR arguments (see Section 18.30 of [RFC8881]).

4. Extraction of XDR

This document contains the XDR [RFC4506] description of the new NFSv4.2 operation LAYOUT_WCC. The XDR description is embedded 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 new NFSv4.2 operation LAYOUT_WCC.

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

That is, 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:

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

The effect of the script is to remove leading blank space from each line, plus a sentinel sequence of '///'. 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 and length4) as well as more generic types (such as uint32_t and uint64_t).

While the XDR can be appended to that from [RFC7863], the various code snippets belong in their respective areas of that XDR.

5. Security Considerations

There are no new security considerations beyond those in [RFC8435].

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. 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/info/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/info/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/info/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/info/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/info/rfc8174>.

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

[RFC8434] Haynes, T., "Requirements for Parallel NFS (pNFS) Layout Types", RFC 8434, DOI 10.17487/RFC8434, August 2018, <https://www.rfc-editor.org/info/rfc8434>.

[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/info/rfc8435>.

[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/info/rfc8881>.

[RFC9754] Haynes, T. and T. Myklebust, "Extensions for Opening and Delegating Files in NFSv4.2", RFC 9754, DOI 10.17487/RFC9754, March 2025, <https://www.rfc-editor.org/info/rfc9754>.

7.2. 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/info/rfc1813>.

Acknowledgments

Dave Noveck, Tigran Mkrtchyan, and Rick Macklem provided reviews of the document.

Authors' Addresses

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com

Trond Myklebust
Hammerspace
Email: trondmy@hammerspace.com