Network File System Version 4 T. Haynes
Internet-Draft Hammerspace
Intended status: Standards Track 7 January 2026
Expires: 11 July 2026

Adding an Uncacheable File Data Attribute to NFSv4.2
draft-ietf-nfsv4-uncacheable-files-01

Abstract

The Network File System version 4.2 (NFSv4.2) allows a client to cache data for file objects. Applications on some clients can control the caching of data, but there is no way to achieve this at a system level. This document introduces a new uncacheable file data attribute for NFSv4.2. Files marked as uncacheable file data SHOULD NOT have their data stored in client-side caches. This document extends NFSv4.2 (see RFC7862).

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/search/?email_list=nfsv4. Source code and issues list for this draft can be found at https://github.com/ietf-wg-nfsv4/uncacheable-files.

Working Group information can be found at https://github.com/ietf-wg-nfsv4.

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 https://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 11 July 2026.

Table of Contents

1. Introduction

With a remote filesystem, the client typically caches file contents in order to improve performance. Several assumptions are made about the rate of change in the number of clients trying to concurrently access a file. With NFSv4.2, this could be mitigated by file delegations for the file contents.

There are prior efforts to bypass file caching. In Highly Parallel Computing (HPC) workloads, file caching is bypassed in order to achieve consistent work flows and to allow concurrent access from many writers.

Applications can use O_DIRECT on open (see [OPEN-O_DIRECT]) to force the client to bypass the page cache, but the limitation is that each application must be modified to use this flag.

This document introduces the uncacheable file data attribute to NFSv4.2 to bypass file caching on the client. As such, it is an OPTIONAL attribute to implement for NFSv4.2. However, if both the client and the server support this attribute, then the client SHOULD follow the semantics of the uncacheable file data attribute.

The uncacheable file data attribute is read-write and per file. The data type is bool.

A client can easily determine whether or not a server supports the uncacheable file data attribute with a simple GETATTR on any file. If the server does not support the uncacheable file data attribute, it will return an error of NFS4ERR_ATTRNOTSUPP.

The only way that the server can determine that the client supports the attribute is if the client sends either a GETATTR or SETATTR with the uncacheable file data attribute.

As bypassing file caching is file based, it is only applicable for objects which are of type attribute value of NF4REG.

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

file caching A client cache, normally called the page cache, which caches the contents of a regular file. Typical usage would be to accumulate changes to be bunched together for writing to the server.

write hole A write hole is a data corruption scenario where either two clients are trying to write to the same erasure encoded block of dataor one client is overwriting an existing erasure encoded block of data. (Adapted from [I-D.haynes-nfsv4-flexfiles-v2].) Note the hole occurs when multiple data servers are not consistent with the encoded block.

Further, the definitions of the following terms are referenced as follows:

* COMMIT ((Section 18.3 of [RFC8881])

* file delegations (Section 10.2 of [RFC8881])

* GETATTR (Section 18.7 of [RFC8881])

* NF4REG (Section 5.8.1.2 of [RFC8881])

* NFS4ERR_ATTRNOTSUPP (Section 15.1.15.1 of [RFC8881])

* SETATTR (Section 18.30 of [RFC8881])

* system (Section 5.8.2.36 of [RFC8881])

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. Caching of File Data

The uncacheable file data attribute instructs the client to bypass its page cache for the file. This often includes write-behind caching used to combine multiple pending WRITEs into a single operation for more efficient remote processing. The uncacheable file data attribute, inhibits this behavior with an effect similar to the user using the O_DIRECT flag with the open call ([OPEN]). Under some conditions clients might be able to determine that files are not shared or do not exhibit access patterns suitable for write-behind caching. In such situations, setting this attribute provides a way to inform other clients of this judgment.

The most pressing need for this feature is in connection with HPC workloads. These often involve massive data transfers and require extremely low latency. Write-behind caching can introduce unpredictable latency, as data is buffered and flushed later.

Another aspect of such workloads is the need to share data between multiple writers. As the application data may span a data block in a page cache, data needs to be flushed immediately for the detection of write holes.

2.1. Uncacheable File Data

If a file object is marked as uncacheable file data, all modifications to the file SHOULD be immediately sent from the client to the server. I.e., if a NFSv4.2 client fails to query this attribute, then it can not meet the requirements of the attribute.

For uncacheable data, the client MUST NOT retain file data in its local data cache for the purpose of satisfying subsequent READ requests or delaying transmission of WRITE data. Reads MUST bypass the client data cache, and WRITE data MUST NOT be retained for read-after-write satisfaction or for the purpose of combining multiple WRITE requests.

Caching of UNSTABLE WRITE data required to support the NFSv4.2 COMMIT operation is permitted and unaffected by this requirement.

If the fattr4_uncacheable_file_data is not set when a client opens a file and is changed whilst the file is open, the client is not responsible for bypassing the page cache. It could flush the page cache and make all subsequent IO be direct, but until the client closes the file and reopens it, it is not required to meet the requirements of the attribute.

If the client has a OPEN_DELEGATE_WRITE delegation on the file (see Section 10.4 of [RFC8881]), then the uncacheable file data attribute takes precedence over the caching of file data. The server can control this by not issuing file delegations for files with this attribute.

3. Setting the Uncacheable File Data Attribute

The uncacheable file data attribute can allow for applications which do not support O_DIRECT to be able to use O_DIRECT semantics. One approach to support this would be to add a new parameter to mount [MOUNT] that specifies all newly created files under that mount point would have their data not cacheable. Then the NFSv4.2 client would use a SETATTR to set fattr4_uncacheable_file_data. This approach is similar to the Solaris forcedirectio (See [SOLARIS-FORCEDIRECTIO]) mount option.

4. Implementation Status

There is a prototype Hammerspace server which implements the uncacheable file data attribute and a prototype Linux client which treats the uncacheable file data attribute as meaning use O_DIRECT. For the prototype, all files created under the mount point have the fattr4_uncacheable_file_data set to be true.

5. XDR for Uncacheable Attribute

```
///
/// typedef bool            fattr4_uncacheable_file_data;
///
/// const FATTR4_UNCACHEABLE_FILE_DATA       = 87;
///
```

6. Extraction of XDR

This document contains the external data representation (XDR) [RFC4506] description of the uncacheable file attribute. 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 > uncacheable_prot.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.

While the XDR can be appended to that from [RFC7863], the code snippets should be placed in their appropriate sections within the existing XDR.

7. Security Considerations

This document imposes no new security considerations to NFSv4.2.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. 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>.

9.2. Informative References

[I-D.haynes-nfsv4-flexfiles-v2] Haynes, T., "Parallel NFS (pNFS) Flexible File Layout Version 2", Work in Progress, Internet-Draft, draft-haynes-nfsv4-flexfiles-v2-02, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-haynes-nfsv4-flexfiles-v2-02>.

[MOUNT] Linux man-pages project, "mount(2) - mount filesystem", Linux Programmer's Manual, 2024, <https://man7.org/linux/man-pages/man2/mount.2.html>.

[OPEN] Linux man-pages project, "open(2) - open and possibly create a file", Linux Programmer's Manual, 2024, <https://man7.org/linux/man-pages/man2/open.2.html>.

[OPEN-O_DIRECT] Linux man-pages project, "open(2) - Linux system call for opening files (O_DIRECT)", 2024, <https://man7.org/linux/man-pages/man2/open.2.html>.

[SOLARIS-FORCEDIRECTIO] Oracle Solaris Documentation, "mount -o forcedirectio - Solaris forcedirectio mount option", Solaris Administration Guide, 2023, <https://docs.oracle.com/en/operating-systems/solaris/oracle-solaris/11.4/manage-nfs/mount-options-for-nfs-file-systems.html>.

Acknowledgments

Trond Myklebust, Mike Snitzer, and Thomas Haynes all worked on the prototype at Hammerspace.

Rick Macklem, Chuck Lever, and Dave Noveck reviewed the document.

Chris Inacio, Brian Pawlowski, and Gorry Fairhurst helped guide this process.

Author's Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com