

Network File System Version 4  
Internet-Draft  
Intended status: Standards Track  
Expires: 19 September 2026

T. Haynes  
Hammerspace  
18 March 2026

Adding an Uncacheable File Data Attribute to NFSv4.2  
draft-ietf-nfsv4-uncacheable-files-04

## Abstract

Network File System version 4.2 (NFSv4.2) clients commonly perform client-side caching of file data in order to improve performance. On some systems, applications may influence client data caching behavior, but there is no standardized mechanism for a server or administrator to indicate that particular file data should not be cached by clients for reasons of performance or correctness. This document introduces a new file data caching attribute for NFSv4.2. Files marked with this attribute are intended to be accessed with client-side caching of file data suppressed, in order to support workloads that require predictable data visibility. 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](mailto:nfsv4@ietf.org)), which is archived at [https://mailarchive.ietf.org/arch/search/?email\\_list=nfsv4](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 19 September 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Definitions . . . . .	4
1.2. Requirements Language . . . . .	4
2. Client-Side Caching of File Data . . . . .	4
2.1. Non-Goals . . . . .	5
2.2. Uncacheable File Data . . . . .	6
3. Setting the Uncacheable File Data Attribute . . . . .	7
4. Implementation Status . . . . .	7
5. XDR for Uncacheable Attribute . . . . .	8
6. Extraction of XDR . . . . .	8
7. Security Considerations . . . . .	9
8. IANA Considerations . . . . .	9
9. References . . . . .	9
9.1. Normative References . . . . .	9
9.2. Informative References . . . . .	10
Acknowledgments . . . . .	10
Author's Address . . . . .	10

## 1. Introduction

Clients of remote filesystems commonly perform client-side caching of file data in order to improve performance. Such caching may include retaining data read from the server to satisfy subsequent READ requests, as well as retaining data written by applications in order to delay or combine WRITE requests before transmitting them to the server. While these techniques are effective for many workloads, they may be unsuitable for workloads that require predictable data visibility or involve concurrent modification of shared files by multiple clients.

In some cases, Network File System version 4.2 (NFSv4.2) (see [RFC7862]) mechanisms such as file delegations can reduce the impact of concurrent access. However, delegations are not always available or effective, particularly for workloads with frequent concurrent writers or rapidly changing access patterns.

There have been prior efforts to bypass file data caching in order to address these issues. In High-Performance Computing (HPC) workloads, file data caching is often bypassed to improve predictability and to avoid read-modify-write hazards when multiple clients write disjoint byte ranges of the same file.

Applications on some systems can request bypass of the client data cache by opening files with the `O_DIRECT` flag (see [OPEN-O\_DIRECT]). However, this approach has limitations, including the requirement that each application be explicitly modified and the lack of a standardized mechanism for communicating this intent between servers and clients.

This document introduces the uncacheable file data attribute to NFSv4.2. This OPTIONAL attribute allows a server to indicate that client-side caching of file data for a particular file is unsuitable. When both the client and the server support this attribute, the client is advised to suppress client-side caching of file data for that file, in accordance with the semantics defined in this document.

The uncacheable file data attribute is read-write, applies on a per-file basis, and has a data type of boolean.

Support for the uncacheable file data attribute is specific to the exported filesystem and may differ between filesystems served by the same server. A client can determine whether the attribute is supported for a given file by issuing a GETATTR request and examining the returned attribute list.

The uncacheable file data attribute applies only to regular files (NF4REG). Attempts to query or set this attribute on objects of other types MUST result in an error of NFS4ERR\_INVALID. Since the uncacheable file data attribute applies only to regular files, attempts to apply it to other object types represent an invalid use of the attribute.

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

## 1.1. Definitions

**client-side caching of file data** The retention of file data by a client in a local data cache, commonly referred to as the page cache, for the purpose of satisfying subsequent READ requests or delaying transmission of WRITE data to the server.

**write-behind caching** A form of file data caching in which WRITE data is retained by the client and transmission of the data to the server is delayed in order to combine multiple WRITE operations or improve efficiency.

**direct I/O** An access mode in which file data is transferred between application buffers and the underlying storage without populating or consulting the client's file data cache. Direct I/O suppresses both read caching and write-behind caching of file data.

**write hole** A write hole is an instance of data corruption that arises when multiple clients modify disjoint byte ranges within the same encoded data block without having a consistent view of the existing contents. This can result in stale data overwriting newer updates, particularly in environments that use erasure encoding or striped storage. (Adapted from [I-D.haynes-nfsv4-flexfiles-v2].)

This document assumes familiarity with the NFSv4 protocol operations, error codes, object types, and attributes as defined in [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. Client-Side Caching of File Data

The uncacheable file data attribute advises the client to bypass its page cache for a file in certain cases. These include forms of client-side caching of file data such as write-behind caching, in which multiple pending WRITES are combined and transmitted to the server at a later time for efficiency. The uncacheable file data attribute inhibits such behavior with an effect similar to that of using the O\_DIRECT flag with the open call ([OPEN-O\_DIRECT]).

While similar in intent to `O_DIRECT`, the uncacheable file data attribute applies at the protocol level and therefore may influence client behavior beyond application-requested direct I/O semantics.

When honoring the uncacheable file data attribute, clients **SHOULD** ensure that file-associated metadata used to make I/O decisions (such as file size or timestamps) is revalidated before exposing cached state to applications. Reuse of stale metadata may otherwise lead applications to make decisions based on outdated file state even when file data caching itself has been suppressed.

The intent of this attribute is to allow a server or administrator to indicate that client-side caching of file data for a particular file is unsuitable. The server is often in a better position than individual clients to determine sharing patterns, access behavior, or correctness requirements associated with a file. By exposing this information via an attribute, the server can advise clients to suppress file data caching in a consistent manner.

One important use case for this attribute arises in connection with High-Performance Computing (HPC) workloads. These workloads often involve large data transfers and concurrent access by multiple clients. In such environments, client-side caching of file data can introduce unpredictable latency or correctness hazards when data is buffered and flushed at a later time.

Another aspect of such workloads is the need to support concurrent writers to shared files. When application data spans a data block in a client cache, delayed transmission of `WRITE` data can result in clients modifying stale data and overwriting updates written by others. Prompt transmission of `WRITE` data enables the prompt detection of write holes and reduces the risk of data corruption.

## 2.1. Non-Goals

This attribute does not require clients to provide strict coherency, does not replace existing NFS cache consistency mechanisms, and does not mandate any specific client implementation strategy. It provides advisory guidance intended to reduce latency and correctness risks in selected workloads.

## 2.2. Uncacheable File Data

When a file object is marked as uncacheable file data, the attribute advises the client that client-side caching of file data for the file is unsuitable. In particular, the client is advised to transmit modifications to the file promptly rather than retaining them in a local data cache. Note that a client that does not query this attribute cannot be expected to observe the behavior described in this section.

For uncacheable file data, the client is advised not to retain file data in its local data cache for the purpose of satisfying subsequent READ requests or delaying transmission of WRITE data. In such cases, READ operations bypass the client data cache, and WRITE data is not retained for read-after-write satisfaction or for the purpose of combining multiple WRITE requests.

Caching of unstably written data used to reissue WRITES lost because of server failure prior to COMMIT is not affected by the advice provided by the uncacheable file data attribute. This is because the server is made aware of the WRITE operation without the sort of delays introduced by write-behind caching.

Suppressing read caching in addition to suppressing write-behind caching reduces the risk of stale-data overwrite in multi-writer workloads. If a client retains cached READ data while other clients concurrently modify disjoint byte ranges of the same file, the client may perform a read-modify-write operation using stale data and overwrite updates written by others. This risk exists even when WRITE operations are transmitted promptly.

Disabling READ caching allows clients to observe the most recent data prior to modification and reduces read-modify-write hazards for shared files. This behavior is consistent with direct I/O semantics such as those provided by the `O_DIRECT` flag in Linux and the `directio/forcedirectio` mechanisms in Solaris.

If the `fattr4_uncacheable_file_data` attribute is not set when a file is opened and is changed while the file is open, the client is not expected to retroactively alter its caching behavior. A client may choose to flush cached data and apply the advice to subsequent I/O, but such behavior is not required until the file is closed and reopened.

The presence of the uncacheable file data attribute does not invalidate file delegations. A server that wishes to ensure prompt client I/O may choose not to issue write delegations for files marked as uncacheable, but clients are not required to suppress delegations solely due to the presence of this attribute.

### 3. Setting the Uncacheable File Data Attribute

The uncacheable file data attribute provides a mechanism by which applications that do not support `O_DIRECT` can request `DIRECT-I/O`-like semantics for file access. In particular, the attribute allows a server to advise clients that client-side caching of file data for a file is unsuitable, including both read caching and write-behind caching.

Suppressing read caching is necessary in addition to suppressing write-behind caching to avoid read-modify-write hazards in multi-writer workloads. If clients retain cached `READ` data while other clients concurrently modify disjoint byte ranges of the same file, stale cached data may be merged with new `WRITE` data and overwrite updates written by others. This risk exists even when `WRITE` data is transmitted promptly and is not addressed by suppressing write-behind caching alone.

One possible deployment model is for a server or administrator to configure a mount (see [MOUNT]) option such that newly created files under a given export are marked as uncacheable file data. In such a configuration, the NFSv4.2 client could use `SETATTR` to set the `fattnr4_uncacheable_file_data` attribute at file creation time.

This approach is conceptually similar in intent to the Solaris `forcedirectio` mount option (see [SOLARIS-FORCEDIRECTIO]), but differs in scope and visibility in that it allows `DIRECT-I/O`-like behavior to be applied without requiring changes to individual applications. However, unlike the Solaris option, the NFSv4.2 attribute is visible to all clients accessing the file and is intended to convey server-side knowledge or policy in a distributed environment.

### 4. Implementation Status

Note to RFC Editor: please remove this section prior to publication.

There is a prototype Hammerspace server which implements the uncacheable file data attribute and a prototype Linux client which treats the attribute as an indication to use `O_DIRECT`-like behavior for file access and to revalidate file-associated metadata before exposing cached state.

For the prototype, all files created under the mount point have the `fat4_uncacheable_file_data` set to be true.

Experience with the prototype indicates that the uncacheable file data attribute can provide many of the practical benefits of `O_DIRECT` without requiring application modification. For applications that issue well-formed I/O requests, this approach has been observed to improve performance in many cases, while also reducing memory pressure and CPU utilization in the NFS client.

## 5. XDR for Uncacheable Attribute

```
///  
// typedef bool          fat4_uncacheable_file_data;  
///  
// const FAT4_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 '^ *///  
<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

The attribute does not introduce new authorization mechanisms or alter existing access control semantics; existing NFSv4.2 security mechanisms continue to apply.

## 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-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, Jon Flynn, Keith Mannthey, 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 Gorrry Fairhurst helped guide this process.

## Author's Address

Thomas Haynes  
Hammerspace  
Email: [loghyr@gmail.com](mailto:loghyr@gmail.com)