Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 22 September 2025

T. Haynes
Hammerspace

21 March 2025

Adding an Uncacheable Attribute to NFSv4.2
draft-haynes-nfsv4-uncacheable-03

Abstract

The Network File System version 4.2 (NFSv4.2) allows a client to cache both metadata and data for file objects, as well as metadata for directory objects. While caching directory entries (dirents) can improve performance, it can also prevent the server from enforcing access control on individual dirents. Similarly, caching file data can lead to performance issues if the cache hit rate is low. This document introduces a new uncacheable attribute for NFSv4. Files and dirents marked as uncacheable MUST NOT be stored in client-side caches. This ensures data consistency and integrity by requiring clients to always retrieve the most recent data directly from the server. 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.

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 22 September 2025.

Table of Contents

1. Introduction 2
1.1. Definitions 3
1.2. Requirements Language 4
2. Caching of Dirents 5
2.1. Uncacheable Dirents 6
3. Caching of File Data 6
3.1. Uncacheable Files 6
4. XDR for Offline Attribute 6
5. Extraction of XDR 6
6. Security Considerations 7
7. IANA Considerations 7
8. References 7
8.1. Normative References 7
8.2. Informative References 8
Acknowledgments 9
Author's Address 9

1. Introduction

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

There are prior efforts to bypass both the dirent and file caching. Access Based Enumeration (ABE) is used in Server Message Block (SMB) [SMB2] protocol to effectively limit the namespace visibility per user. In Highly Parallel Computing (HPC) workloads, file caching is bypassed in order to achieve consistent work flows.

This document introduces the uncacheable attribute to NFSv4.2 to implement ABE and to bypass file caching on the client. As such, it is an OPTIONAL to implement attribute for NFSv4.2. However, if both the client and the server support this attribute, then the client MUST follow the semantics of uncacheable.
// What about mixed modes?

A client can easily determine whether or not a server supports the uncacheable attribute with a simple GETATTR on any dirent. If the server does not support the uncacheable 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 a SETATTR with the uncacheable attribute.

While some argument could be made to introduce two new attributes, the functionality of the uncacheable attribute dictates which cache is to be bypassed. As ABE is concerned with walking the namespace, it is only applicable to be acted on dirents which are of type attribute value of NF4DIR. And as bypassing file caching is file based, it is only applicable for dirents which are of type attribute value of NF4REG.
// What about the other file types?

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

Access Based Enumeration (ABE) When servicing a READDIR or GETATTR operation, the server provides results based on the access permissions of the user making the request.

dirent A directory entry, representing either a file or a subdirectory. In the context of NFSv4, a dirent marked as uncacheable MUST NOT be cached by clients.

dirent caching A client cache that is used to avoid looking up attributes.

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.

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

* directory delegations (Section 10.9 of [RFC8881])

* file delegations (Section 10.2 of [RFC8881])

* GETATTR (Section 18.7 of [RFC8881])

* hidden (Section 5.8.2.15 of [RFC8881])

* Mandatory Access Control (MAC) ([RFC4949])

* NF4DIR (Section 5.8.1.2 of [RFC8881])

* NF4REG (Section 5.8.1.2 of [RFC8881])

* NFS4ERR_ATTRNOTSUPP (Section 15.1.15.1 of [RFC8881]

* mode (Section 6.2.4 of [RFC8881])

* offline (Section 2 of [I-D.ietf-nfsv4-delstid])

* owner (Section 5.8.2.26 of [RFC8881])

* owner_group (Section 5.8.2.27 of [RFC8881])

* READDIR (Section 18.23 of [RFC8881])

* SETATTR (Section 18.30 of [RFC8881])

* system (Section 5.8.2.36 of [RFC8881])

* type (Section 5.8.1.2 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 Dirents

With a remote filesystem, the client typically caches directory entries (dirents) locally to improve performance. This cooperation succeeds because both the server and client operate under POSIX semantics ([POSIX.1]) and agree to interpretation of mode bits with respect to the uid and gid in NFSv3 [RFC1813]. For NFSv4.2, these would respectively be the mode, owner, and owner_group attributes defined in Section 5 of [RFC8881]. Note that this cooperation does not apply to Access Control List (ACLs) entries as NFSv4.2 does not implement a strict POSIX style ACL.

NFSv4.2 does implement NFSv4.1 ACLs, which are enforced on the server and not the client. As such, ACL enforcement requires the client to bypass the dirent cache to have checks done when a new user attempts to access the dirent.

Another consideration is that not all server implementations natively support the SMB [SMB2]. Instead, they layer Samba [Samba] on top of the NFSv4.2 service. The attributes of hidden, system, and offline have already been introduced in the NFSv4.2 protocol to support Samba. The Samba implementation can utilize these attributes to provide SMB semantics. While private protocols can supply these features, it is better to drive them into open standards.

Another concept that can be adapted from SMB is that of Access Based Enumeration (ABE). If a share or a folder has ABE enabled, then the user can only see the files and sub-folders for which they have permissions.

Under the POSIX model, this can be done on the client and not the server. However, that only works with uid, gid, and mode bits. If we consider identity mappings, ACLS, and server local policies, then the determination of ABE MUST be done on the server.

Since cached dirents are shared by all users on a client, and the client cannot determine access permissions for individual dirents, all users are presented with the same set of attributes. To address this, this document introduces the new uncacheable attribute. This attribute instructs the client not to cache the dirent for a file or directory object. Consequently, each time a client queries for these attributes, the server’s response can be tailored to the specific user making the request.

2.1. Uncacheable Dirents

If a file object or directory has the uncacheable attribute set, then the client MUST NOT cache its dirent attributes. This means that even if the client has previously retrieved the attributes for a user, it MUST query the server again for those attributes on subsequent requests. Additionally, the client MUST NOT share attributes between different users.

3. Caching of File Data

In addition to caching metadata, clients can also cache file data. The uncacheable attribute also instructs the client to bypass its page cache for the file. This behavior is similar to using the O_DIRECT flag with the open call ([open_2_]). This can be beneficial for files that are not shared or for files that do not exhibit access patterns suitable for caching.

However, the real need for bypassing write caching is evident in HPC workloads. In general, these involve massive data transfers and require extremely low latency. Write caching can introduce unpredictable latency, as data is buffered and flushed later.

3.1. Uncacheable Files

If a file object is marked as uncacheable, all modifications to the file MUST be immediately sent from the client to the server. In other words, the file data is also not cacheable.

4. XDR for Offline Attribute

```
///
/// typedef bool            fattr4_uncacheable;
///
/// const FATTR4_UNCACHEABLE            = 87;
///
```

5. Extraction of XDR

This document contains the external data representation (XDR) [RFC4506] description of the uncacheable 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:

```
#!/bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^ *///$??'
```

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

```
sh extract.sh < spec.txt > uncacheable_prot.x
```

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.

6. Security Considerations

For a given user A, a client MUST NOT make access decisions for uncacheable dirents retrieved for another user B. These decisions MUST be made by the server. If the client is Labeled NFS aware ([RFC7204]), then the client MUST locally enforce the MAC security policies.

The uncacheable attribute allows dirents to be annotated such that attributes are presented to the user based on the server's access control decisions.

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[I-D.ietf-nfsv4-delstid]
Haynes, T. and T. Myklebust, "Extending the Opening of Files in NFSv4.2", Work in Progress, Internet-Draft, draft-ietf-nfsv4-delstid-08, 2 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-delstid-08>.

[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>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC7204] Haynes, T., "Requirements for Labeled NFS", RFC 7204, DOI 10.17487/RFC7204, April 2014, <https://www.rfc-editor.org/rfc/rfc7204>.

[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

[open_2_] "open and create files.", Linux Programmer’s Manual , n.d..

[POSIX.1] IEEE, "The Open Group Base Specifications Issue 7", IEEE Std 1003.1, 2013 Edition , 2013.

[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>.

[Samba] "Samba.org. Samba Project Website.", n.d., <https://www.samba.org/>.

[SMB2] Microsoft Learn, "Server Message Block (SMB) Protocol Versions 2 and 3", n.d..

Acknowledgments

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

Author’s Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com