Network File System Version 4 T. Haynes
Internet-Draft Hammerspace
Intended status: Standards Track 8 April 2026
Expires: 10 October 2026

Adding an Atomic EXCHANGE_RANGE Operation to NFSv4.2
draft-haynes-nfsv4-swap-07

Abstract

The Network File System version 4.2 (NFSv4.2) does not provide support for atomic multi-block updates to file data. This document introduces a new EXCHANGE_RANGE operation which provides for such atomic updates. 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/exchange_range.

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 10 October 2026.

Copyright Notice

Table of Contents

1. Introduction

With the Network File System version 4.2 (NFSv4.2), atomic updates to a file are not guaranteed. A single WRITE operation might span multiple data blocks; another client doing a READ might then encounter a partial WRITE. In addition, multiple WRITE operations, even within the same compound, may not be atomically applied to a file. In some implementations, multiple WRITE operations within the same compound may appear to be applied atomically, but this behavior is implementation-specific and not guaranteed by the protocol.

This document introduces the EXCHANGE_RANGE operation, which is OPTIONAL to implement, to NFSv4.2. EXCHANGE_RANGE atomically exchanges a range of content between two files. A client can easily determine whether or not a server supports the EXCHANGE_RANGE operation by examining the return code of the operation. If the server does not support the EXCHANGE_RANGE operation, the server returns NFS4ERR_NOTSUPP.

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].

2. Definitions

The definitions of the following terms are referenced as follows:

* change_info4 (Section 3.3.3 of [RFC8881])

* CLONE (Section 15.13 of [RFC7862])

* clone_blksize (Section 12.2.1 of [RFC7862])

* NFS4ERR_NOTSUPP (Section 15.1.1.5 of [RFC8881])

* READ (Section 18.22 of [RFC8881])

* WRITE (Section 18.32 of [RFC8881])

3. 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.

4. Use Cases

The EXCHANGE_RANGE operation addresses a class of problems that cannot be solved without loss of atomicity by existing NFSv4.2 operations. This section motivates the operation by contrasting it with a hypothetical one-way transfer primitive and then presenting five concrete use cases.

Consider an operation called XFER_RANGE that atomically copies data from a source range to a destination range -- essentially CLONE (Section 15.13 of [RFC7862]) with an atomicity guarantee. XFER_RANGE

would be a useful primitive in its own right for cases where one-way data movement is all that is needed. EXCHANGE_RANGE differs from it in one critical respect: EXCHANGE_RANGE atomically swaps the contents of both ranges. Neither range is "the source" and neither is "the destination" -- both hold meaningful data before the operation and both hold meaningful data after. While the protocol distinguishes SAVED_FH and CURRENT_FH for access control purposes, the semantics of the operation treat both ranges symmetrically. The following examples illustrate when the distinction matters.

The use cases below share a common requirement: both ranges contain valid data before the operation, and both must remain valid after the operation, with no externally visible intermediate state. A sequence of one-way operations (e.g., CLONE, WRITE, or a hypothetical XFER_RANGE) cannot provide this guarantee, as they either destroy one side of the data or introduce a window in which observers may see partially updated state.

A/B deployment: A server pre-computes version 2 of a dataset into fileB while version 1 remains live in fileA. At the moment of go-live, the operator wants to atomically swap which file is current. XFER_RANGE(B->A) overwrites fileA and destroys version 1. XFER_RANGE(A->B) does the same in the other direction. Neither preserves both versions. EXCHANGE_RANGE atomically swaps the two, leaving version 1 intact in fileB for inspection or rollback. This example applies when the two versions reside in fixed locations or ranges that cannot be renamed or redirected atomically (e.g., shared file layouts or range-based consumers).

Checkpoint and restore: A process maintains a live range and a checkpoint range. A checkpoint is performed by EXCHANGE_RANGE(live, checkpoint): the checkpoint range now holds the saved state and the live range holds what was the checkpoint. A restore is performed by the identical call. The self-inverse property means the mechanism for save and restore are the same operation. XFER_RANGE requires two distinct one-way operations and cannot guarantee that a restore returns exactly what was saved without additional bookkeeping.

Circular log rotation: Segment 0 is being read by consumers; segment 1 has been filled by a writer and is ready to become the new read segment. Both segments hold valuable data -- neither is scratch space. EXCHANGE_RANGE atomically swaps their roles. XFER_RANGE(1->0) destroys segment 0’s content before readers have finished with it. In systems with multiple independent readers, coordination is not feasible, and readers must never observe a partially rotated state.

Erasure coding stripe rebuild: During a pNFS Flex Files ([I-D.haynes-nfsv4-flexfiles-v2]) reconstruction, a data server has rebuilt a stripe from parity into a staging buffer. Both the live stripe and the staging buffer exist and are valid. EXCHANGE_RANGE atomically promotes the staging buffer to live and moves the old live data into the staging buffer, where it remains available for integrity verification or rollback if the new stripe fails a subsequent check. XFER_RANGE would overwrite the live stripe before that verification can occur. This avoids both the need to copy data and any window in which parity and data are inconsistent.

Atomic file update via clone-then-swap: This is the case where XFER_RANGE would be sufficient, and it is worth stating clearly. A client CLONEs a source file to a temporary working copy, modifies the copy, and then wants to atomically publish the result. XFER_RANGE(temp->source) accomplishes this: the temporary file is discarded and source is updated atomically. EXCHANGE_RANGE handles this case as well, and adds one benefit: after the swap, the temporary file holds the previous content of source, providing a no-cost rollback path -- invoke EXCHANGE_RANGE again and the original state is restored. An implementation that does not need rollback can simply discard the old content; one that does need it gets it without additional operations.

A composition of existing operations (e.g., CLONE followed by WRITE, or two XFER_RANGE operations) cannot provide the same guarantees as EXCHANGE_RANGE. Such compositions either:

* destroy one side of the data, or

* expose an intermediate state to other clients, or

* require external coordination not available at the protocol level.

EXCHANGE_RANGE provides these guarantees as a single atomic operation.

5. Operation 81: EXCHANGE_RANGE - Exchange a range of a file into another file

5.1. ARGUMENTS

```
/// const OP_EXCHANGE_RANGE = 81;
///
/// struct EXCHANGE_RANGE4args {
///         /* SAVED_FH: source file */
///         /* CURRENT_FH: destination file */
///         stateid4        era_src_stateid;
///         stateid4        era_dst_stateid;
///         offset4         era_src_offset;
///         offset4         era_dst_offset;
///         length4         era_count;
/// };
```

Figure 1: XDR for EXCHANGE_RANGE4args

5.2. RESULTS

```
/// struct EXCHANGE_RANGE4resok {
///         change_info4    err_src_cinfo;
///         change_info4    err_dst_cinfo;
/// };
///
/// union EXCHANGE_RANGE4res switch (nfsstat4 err_status) {
///  case NFS4_OK:
///          EXCHANGE_RANGE4resok err_resok4;
///  default:
///          void;
/// };
```

Figure 2: XDR for EXCHANGE_RANGE4res

The following case arms are added to the nfs_argop4 and nfs_resop4 dispatch unions in the base XDR:

```
/// /* In nfs_argop4: */
/// case OP_EXCHANGE_RANGE:
///         EXCHANGE_RANGE4args opexchange_range;
///
/// /* In nfs_resop4: */
/// case OP_EXCHANGE_RANGE:
///         EXCHANGE_RANGE4res opexchange_range;
```

Figure 3: Dispatch union case arms for EXCHANGE_RANGE

5.3. DESCRIPTION

The EXCHANGE_RANGE operation is used to exchange file content from a source file specified by the SAVED_FH value into a destination file specified by CURRENT_FH without actually copying the data, e.g., by using a block exchange mechanism.

Both SAVED_FH and CURRENT_FH MUST be regular files. If either SAVED_FH or CURRENT_FH is not a regular file, the operation MUST fail and return NFS4ERR_WRONG_TYPE.

The era_dst_stateid MUST refer to a stateid that is valid for a WRITE operation and follows the rules for stateids in Section 8.2.5 of [RFC8881] and Section 18.32.3 of [RFC8881]. The era_src_stateid MUST refer to a stateid that is valid for a READ operation and follows the rules for stateids in Section 8.2.5 of [RFC8881] and Section 18.22.3 of [RFC8881]. If either stateid is invalid, then the operation MUST fail.

The era_src_offset is the starting offset within the source file from which the data to be exchanged will be obtained, and the era_dst_offset is the starting offset of the target region into which the exchanged data will be placed. An offset of 0 (zero) indicates the start of the respective file. The number of bytes to be exchanged is obtained from era_count, except that an era_count of 0 (zero) indicates that the number of bytes to be exchanged is the count of bytes between era_src_offset and the EOF of the source file; in this case the derived count ends at the source EOF, so the alignment exception below always applies. Both era_src_offset and era_dst_offset MUST be aligned to the clone block size (Section 12.2.1 of [RFC7862]). The number of bytes to be exchanged MUST be a multiple of the clone block size, except in the case in which era_src_offset plus the number of bytes to be exchanged is equal to the source file size.

If era_src_offset or era_src_offset + era_count is greater than the size of the source file, the operation MUST fail with NFS4ERR_INVAL. It is valid for era_dst_offset or era_dst_offset + era_count to be greater than the current size of the destination file.

If SAVED_FH and CURRENT_FH refer to the same file and the source and target ranges overlap, the operation MUST fail with NFS4ERR_INVAL. This restriction avoids undefined behavior that could arise from overlapping atomic replacement of data within a single file.

If the target area of the EXCHANGE_RANGE operation ends beyond the end of the destination file, the new size of the destination file MUST be set to era_dst_offset plus the number of bytes exchanged. The contents of any block not part of the target area MUST be the same as if the file size were extended by a WRITE.

If the number of bytes to be exchanged (using the derived count when era_count is 0) is not a multiple of the clone block size and era_dst_offset plus the derived count is less than the current size of the destination file, the operation MUST fail with NFS4ERR_INVAL.

This restriction avoids modifying a portion of a clone block while leaving the remainder of that clone block within the destination file unchanged, which could otherwise lead to implementation-dependent results and potential data integrity issues.

If a conflicting byte-range lock exists on either the source or the destination range at the time of the operation, the server MUST return NFS4ERR_LOCKED before performing any exchange. The source range requires at minimum read access; the destination range requires write access.

The EXCHANGE_RANGE operation is atomic in that other operations MUST NOT see any intermediate states between the state of the two files before the operation and after the operation. A READ of the destination file MUST NOT see some blocks of the target area exchanged without all of them being exchanged. A WRITE to the source area MUST either have no effect on the data of the target file or be fully reflected in the target area of the destination file.

Atomicity is defined with respect to NFSv4.2 READ and WRITE operations issued by other clients; the protocol makes no guarantees regarding the visibility of intermediate states to server-internal mechanisms.

Upon successful completion, the server MUST update the change, mtime, and ctime attributes of both the source and destination files.

The completion status of the operation is indicated by err_status.

Upon success, the response includes err_resok4, which contains two change_info4 values: err_src_cinfo for the source file and err_dst_cinfo for the destination file. Each change_info4 carries the change attribute value before (cinfo_before) and after (cinfo_after) the exchange, sampled atomically with the operation. Clients MAY use these values to update their attribute caches for both files without issuing a subsequent GETATTR. When cinfo_atomic is TRUE, the client MAY treat the before/after pair as a precise

record of the transition; when FALSE, the client MUST treat the cached change attribute as potentially stale and revalidate on next access. Because EXCHANGE_RANGE is its own inverse, a client confirming execution MUST compare cinfo_before against a previously recorded value -- comparing cinfo_after is insufficient, as two executions leave cinfo_after equal to the original cinfo_before.

Within an NFSv4.1 or NFSv4.2 session, the SEQUENCE operation (Section 18.46 of [RFC8881]) provides exactly-once execution semantics via the slot identifier and sequenceid carried in each compound: the server's session reply cache detects retransmissions and returns the cached reply without re-executing the operation. The discussion below addresses replay protection across server reboots, where session state is not preserved.

Stateids supplied with the EXCHANGE_RANGE operation provide protection against replay across server reboots, lease expiration, or state revocation. After a server reboot, all previously issued stateids are invalidated, and replay of a prior EXCHANGE_RANGE operation MUST fail with an appropriate error.

Because EXCHANGE_RANGE is its own inverse -- applying it twice to the same ranges returns both files to their original state -- clients MUST NOT assume that a successfully replayed EXCHANGE_RANGE operation produced a net change. Clients that require confirmation MUST validate the resulting file contents or metadata, such as via the change attribute.

Per Section 3.3.1 of [RFC7862], EXCHANGE_RANGE is not an operation which can be sent to a data server.

5.4. ERRORS

The EXCHANGE_RANGE operation returns the full set of errors defined in Section 6, Paragraph 2. The following errors are specific to this operation:

NFS4ERR_WRONG_TYPE: Either SAVED_FH or CURRENT_FH is not a regular file.

NFS4ERR_INVAL: era_src_offset or era_dst_offset is not aligned to the clone block size; or the count is not a valid multiple of the clone block size and the range does not end at the source EOF; or the source range extends beyond the source file; or SAVED_FH and CURRENT_FH are the same file and the source and target ranges overlap.

NFS4ERR_BAD_STATEID, NFS4ERR_EXPIRED, NFS4ERR_ADMIN_REVOKED,

NFS4ERR_DELEG_REVOKED, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE: The era_src_stateid or era_dst_stateid is not valid for the required access mode, has expired, or has been revoked.

NFS4ERR_LOCKED: A conflicting byte-range lock on the source or destination range prevents the exchange.

6. Operations and Their Valid Errors

The operations and their valid errors are presented in Section 6, Paragraph 2. All error codes not defined in this document are defined in Section 15 of [RFC8881] and Section 11 of [RFC7862].

EXCHANGE_RANGE: NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_ISDIR, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_NOTSUPP, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PNFS_IO_HOLE, NFS4ERR_PNFS_NO_LAYOUT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE

If the destination file has active pNFS layouts that prevent atomic modification of the target range, the server MAY return an appropriate pNFS-related error.

7. Extraction of XDR

This document contains the external data representation (XDR) [RFC4506] description of the EXCHANGE_RANGE operation. 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 'exchange_range.txt', execute the following command:

```
sh extract.sh < exchange_range.txt > exchange_range_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.

8. Security Considerations

The EXCHANGE_RANGE operation is subject to the same security considerations as other NFSv4.2 operations that modify file data.

The caller MUST have appropriate access to both the source file (read access) and the destination file (write access). Servers MUST enforce authorization checks on both filehandles before performing the exchange.

Within a session, the SEQUENCE mechanism (Section 18.46 of [RFC8881]) prevents duplicate execution via the slot identifier and sequenceid. After a server reboot, previously issued stateids are invalidated, preventing replay of prior operations across reboots.

Because EXCHANGE_RANGE is its own inverse, a double execution returns both files to their original state. A client checking only the final value of the change attribute cannot distinguish zero executions from two executions, since both cases leave the files in their original state while the change attribute may have been incremented an even number of times. The change_info4 values returned in err_src_cinfo and err_dst_cinfo enable precise confirmation: a client MUST record the change attribute of both files before the first attempt and compare cinfo_before from the response against that recorded value to confirm a single net execution. Comparing only cinfo_after is insufficient for this purpose.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. 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>.

10.2. 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-v2-03, 27 March 2026, <https://datatracker.ietf.org/doc/html/draft-haynes-nfsv4-flexfiles-v2-03>.

[XFS-EXCHANGE-RANGE]
Linux man-pages project, "ioctl_xfs_exchange_range(2) - Exchange data between files (Accessed January 2026)", Linux 6.10, July 2024.

Acknowledgments

Christoph Helwig brought the XFS exchange range implementation ([XFS-EXCHANGE-RANGE]) to the author’s attention, which prompted this document.

Darrick Wong, David Noveck, Pali Rohar, Rick Macklem, and Christoph Helwig helped review the document.

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

Author’s Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com