Internet Engineering Task Force	M. Eisler, Ed.
Internet-Draft	NetApp
Intended status: Informational	M. Susairaj, Ed.
Expires: April 17, 2011	Oracle
	October 14, 2010

TOC

Extending NFS to Support Enterprise Applications
draft-eisler-nfsv4-enterprise-apps-01

Abstract

This document proposes a new operating to efficiently initialize files.

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 http://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 April 17, 2011.

Copyright Notice

Table of Contents

1. Introduction

TOC

Enterprise applications (such as databases) have requirements that go beyond the traditional use cases for NFS. The requirements falls into two broad categories: (1) data integrity and (2) quality of service. This document proposes a set of operatons for a future minor version of NFSv4 to support requirements of enterprise applications.

1.1. Requirements Language

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

2. Operation XX: INITIALIZE - Initialize File

TOC

2.1. ARGUMENT

TOC

```
struct INITIALIZE4args {
          /* CURRENT_FH: file */
          stateid4        ia_stateid;
          offset4         ia_offset;
          length4         ia_blocksize
          length4         ia_blockcount;
          length4         ia_reloff_pattern;
          length4         ia_reloff_blocknum;
          opaque          ia_pattern<>;
};
```

2.2. RESULT

TOC

```
nfsstat4;
```

2.3. MOTIVATION

TOC

Most enterprise applications that use files almost always need to initialize such files to a known state. Even with existing files, after such a file grows, the application needs to initialize the expanded region the file. The most trivial initial state is intialize every byte to zero. The problem with initializing to zero is that it is often difficult to distinguish a byte-range of initialized to all zeroes from data corruption, since a pattern of zeroes is a probable pattern for corruption. Instead, some applications, such as database management systems, use pattern consisting of bytes or words of non-zero values. Ideally one would like to efficiently initialize an entire file to a specified pattern without having to send WRITE requests for the entire file. The INITIALIZE operation is bandwidth conserving operation for initializing file state.

2.4. DESCRIPTION

TOC

The INITIALIZE operation is used to initialize an open file to an iterated pattern. The pattern consists of a fixed string, and a block number. The pattern is defined by the arguments.

*ia_offset: where to start the iterated pattern. This value is specified in bytes.

*ia_blocksize: the size of each iteration of the pattern. Each iteration is called a block.

*ia_reloff_pattern: the relative offset within a block where to write the specified pattern encoded in ia_pattern.

*ia_reloff_blocknum: the relative offset within a block where to write a 64 bit block number. The block number is incremented once a block is written. The block number is always written in little endian order. If ia_reloff_blocknum is set to NFS4_UINT64_MAX, then this informs the server that no block number is to be written.

*ia_pattern: a fixed string written to every block. If the length of ia_pattern is zero, then this informs the server that no string is to be written.

The field ia_stateid is the stateid corresponding to the current filehandle's share reservation, delegation, or byte range lock.
An example will illustrate how the client uses INITIALIZE. Suppose the arguments (except for ia_stateid) are: { 0, 500, 1000, 8, 0, "DeadBeef" }. Then starting with offset zero, the content of the file will have these contents.

```
offset value (decimal or ASCII)
0      0   0   0   0   0   0   0   0
8      'D' 'e' 'a' 'd' 'B' 'e' 'e' 'f'
16-499 zeroes

500    0   0   0   0   0   0   0   1
508    'D' 'e' 'a' 'd' 'B' 'e' 'e' 'f'
516-999 zeroes

...

499500  0   0   0   0   0   0   3   231
499508  'D' 'e' 'a' 'd' 'B' 'e' 'e' 'f'
499516-499999 zeroes
```

2.5. IMPLEMENTATION

TOC

When an NFS server receives this operation, instead of writing the iterated pattern over each block, it should de-allocate the data of affected range of the file and record the the values of ia_offset, ia_blocksize, ia_blockcount, ia_reloff_pattern, ia_reloff_blocknum, and ia_pattern<> in the file's system metadata. When a client sends a READ request, instead of returning zeroes, it should construct a response corresponding to the pattern specified in the arguments to INITIALIZE. An application likely has a legacy pattern for initialized blocks which cannot be mapped to that specified for INITIALIZE. The application should modified to detect that the block corresponds to INITIALIZE's pattern. When the application sees such a block, it can overwrite the block with the legacy pattern. Note that will cause the block to be allocated on the NFS server.
When the length of ia_pattern is zero and the value of ia_reloff_blocknum is NFS4_UINT64_MAX, then the client is requesting that a hole be punched into the file.

3. Operation XX: IO_ADVISE - Advise server of client's intended I/O access pattern

TOC

3.1. ARGUMENT

TOC

```
enum io_advise_type {
   IO_ADVISE4_SEQUENTIAL_CACHE       = 0,
   IO_ADVISE4_SEQUENTIAL_DONTCACHE       = 1,
   IO_ADVISE4_RANDOM           = 2,
   IO_ADVISE4_PREFETCH         = 3,
   IO_ADVISE4_PREFETCH_OPPORTUNISTIC = 4,
   IO_ADVISE4_INTENT_TO_WRITE = 5,
   IO_ADVISE4_RECENTLY_USED = 6
};


struct io_directions {
        stateid4        iod_stateid;
        offset4         iod_offset;
        bitmap4         iod_flags;
};

struct IO_ADVISE4args {
       /* CURRENT_FH: file */
       io_directions ioaa_directions;
       length4       ioaa_count;
};
```

TOC

3.2. RESULT

```
struct IO_ADVISE4resok {
        bitmap4              ioar_flags;
};

union IO_ADVISE4res switch (nfsstat4 ioar_status) {
 case NFS4_OK:
         IO_ADVISE4resok  ioar_resok4;
 default:
         void;
};
```

3.3. MOTIVATION

TOC

The client is in a better position to deduce the intended I/O pattern than the server, especially if the application provides this information. With this information, the server can optimize I/O to the file.

3.4. DESCRIPTION

TOC

The IO_ADVISE operation is used advise the server as to how the holder of the stateid intends to access the file over the specified byte range (iod_offset through iod_offset + ioaa_count - 1).

*IO_ADVISE4_SEQUENTIAL_CACHE: Sequential access to data expected. The server should leave data in its cache.

*IO_ADVISE4_SEQUENTIAL_DONTCACHE: Sequential access to data expected. The server does not need to leave data in its cache.

*IO_ADVISE4_RANDOM: Random access to data expected.

*IO_ADVISE4_PREFETCH: Stateid holder expects to access the data soon; prefetch data in preparation.

*IO_ADVISE4_PREFETCH_OPPORTUNISTIC: Stateid holder expects to access the data soon; prefetch if it can be done at a marginal cost.

* IO_ADVISE4_INTENT_TO_WRITE: Byte range will be written soon so no point in caching data.

* IO_ADVISE4_RECENTLY_USED: The client has recently accessed the byte range in its own cache. This informs the server that the data in the byte range remains important to the client. When the server reaches resource exhaustion, knowing which data is more important allows the server to make better choices about which data to, for example purge from a cache, or move to secondary storage. It also informs the server which delegations are more important, since if delegations are working correctly, once delegated to a client, a server might never receive another I/O request for the file.

The results indicate which advice the server intends to follow. The server MUST NOT return an error if it does not recognize or does not support the requested advice. The server MAY return different advice than what the client requested. If it does, then this might be due to one of several conditions, including, but not limited to: another client advising of a different I/O access pattern; a different I/O access pattern from another client that that the server has heuristically detected; or the server is not able to support the requested I/O access pattern, perhaps due to a temporary resource limitation (for example, a request for IO_ADVISE4_SEQUENTIAL_CACHE might not be supported because the server cannot afford to cache data, and/or cannot afford to queue read-a-head requests).

---

4. Operation XX: READ_WITH_ADVICE - READ with advice

TOC

TOC

4.1. ARGUMENT

```
      enum io_advise_type {
         IO_ADVISE4_SEQUENTIAL_CACHE       = 0,
         IO_ADVISE4_SEQUENTIAL_DONTCACHE       = 1,
         IO_ADVISE4_RANDOM           = 2,
         IO_ADVISE4_PREFETCH         = 3,
         IO_ADVISE4_PREFETCH_OPPORTUNISTIC = 4,
         IO_ADVISE4_INTENT_TO_WRITE = 5,
         IO_ADVISE4_RECENTLY_USED = 6
      };


      struct io_directions {
              stateid4        iod_stateid;
              offset4         iod_offset;

              bitmap4         iod_flags;
      };

   struct READ_WITH_ADVICE4args {
          /* CURRENT_FH: file */
          io_directions rwaa_directions;
          length4       rwaa_count;
   };
```

TOC

4.2. RESULT

```
const NFS4_TWO_GB           = 0x80000000;

typedef opaque twoGB_byte_array4[NFS4_INT32_MAX];

struct fourGB_buffer4 {
     twoGB_byte_array4[2];
}

struct large_buffer4 {
        fourGB_buffer4 lb_big_buffers<>;
        opaque         lb_small_buffer<>;
}


struct READ_WITH_ADVICE4resok {
        bool                rwar_eof;
        bitmap4             rwar_flags;
        large_buffer4       rwar_data;

};

union READ_WITH_ADVICE4res switch (nfsstat4 rwar_status) {
 case NFS4_OK:
         READ_WITH_ADVICE4resok  rwar_resok4;
 default:
         void;
};
```

4.3. MOTIVATION

TOC

Under some circumstances, the IO_ADVISE operation is insufficient when the client is also performing a READ operation. Some advice needs to be communicated atomically with the READ operation and an IO_ADVISE in the same COMPOUND operation as the READ operation would fail to provide the necessary advice. For example, if IO_ADVISE proceeded READ, and the server was given advice to not cache the data requested by READ, the IO_ADVISE would be too late, because the server might already have cached the data. If IO_ADVISE preceded READ, in order to be effective, the advice would have to be communicated across two operations in the same COMPOUND. This would complicate the server implementation.

4.4. DESCRIPTION

TOC

The READ_WITH_ADVICE operation is used read from a file and to advise the server as to how the reader of intends to access the file over the specified byte range (iod_offset through iod_offset + rwaa_count - 1).

- *IO_ADVISE4_SEQUENTIAL_CACHE: Sequential access to data expected. The server should leave data in its cache.

- *IO_ADVISE4_SEQUENTIAL_DONTCACHE: Sequential access to data expected. The server does not need to leave data in its cache.

- *IO_ADVISE4_RANDOM: Random access to data expected.

- *IO_ADVISE4_PREFETCH: Not applicable.

- *IO_ADVISE4_PREFETCH_OPPORTUNISTIC: Not applicable.

- *IO_ADVISE4_INTENT_TO_WRITE: Byte range will be written soon so no point in caching data.

- *IO_ADVISE4_RECENTLY_USED: Explicit hint to keep data of byte range in cache.

The results indicate which advice the server intends to follow. The server MUST NOT return an error if it does not recognize or does not support the requested advice.
The intent is that READ_WITH_ADVICE is preferred over READ. In addition to providing I/O hints, READ_WITH_ADVICE uses 64 bit data lengths, which anticipates the expected improvements in average network speeds and network buffer capacities. Because the XDR standard does not support 64 bit array lengths, the large_buffer4 data type is introduced to encode an array of zero or more buffers of fixed size of 2^32 bytes, followed by a variable length array of up to 2^32 - 1 bytes

---

5. Operation XX: WRITE_WITH_ADVICE - WRITE with advice

TOC

TOC

5.1. ARGUMENT

```
            enum stable_how4 { /* from NFSv4.0 */
             UNSTABLE4       = 0,
             DATA_SYNC4      = 1,
             FILE_SYNC4      = 2,
             LAYOUT_SYNC4    = 3 /* new */
            };

         enum io_advise_type {
            IO_ADVISE4_SEQUENTIAL_CACHE       = 0,
            IO_ADVISE4_SEQUENTIAL_DONTCACHE       = 1,
            IO_ADVISE4_RANDOM           = 2,
            IO_ADVISE4_PREFETCH         = 3,
            IO_ADVISE4_PREFETCH_OPPORTUNISTIC = 4,
            IO_ADVISE4_INTENT_TO_WRITE = 5,
            IO_ADVISE4_RECENTLY_USED = 6
         };

         struct io_directions {
                 stateid4        iod_stateid;
                offset4         iod_offset;
                bitmap4         iod_flags;
         };

      struct WRITE_WITH_ADVICE4args {
             /* CURRENT_FH: file */
             stable_how4     wwaa_stable;
             io_directions   wwaa_directions;
             large_buffer4   wwaa_data<>;
      };
```

TOC

5.2. RESULT

```
struct WRITE_WITH_ADVICE4resok {
        length4              wwar_count;
        stable_how4          wwar_committed;
        bitmap4              wwar_flags;


};

union WRITE_WITH_ADVICE4res switch (nfsstat4 wwar_status) {
 case NFS4_OK:
         WRITE_WITH_ADVICE4resok  wwar_resok4;
 default:
         void;
};
```

5.3. MOTIVATION

TOC

Under some circumstances, the IO_ADVISE operation is insufficient when the client is also performing a WRITE operation. Some advice needs to be communicated atomically with the WRITE operation and an IO_ADVISE in the same COMPOUND operation as the WRITE operation would fail to provide the necessary advice. For example, if IO_ADVISE proceeded WRITE and the server was given advice to not cache the data requested by WRITE the IO_ADVISE would be too late, because the server might already have cached the data. If IO_ADVISE preceded WRITE in order to be effective, the advice would have to be communicated across two operations in the same COMPOUND. This would complicate the server implementation.
This operation adds a new enumerated value for stable_how4 called LAYOUT_SYNC4 in order to reduce the need for LAYOUT_COMMIT operations.

5.4. DESCRIPTION

TOC

The WRITE_WITH_ADVICE operation is used write to a file and to advise the server as to how the writer intends to access the file over the

specified byte range (iod_offset through iod_offset + amount of data in wwaa_data - 1).

- *IO_ADVISE4_SEQUENTIAL_CACHE: Sequential access to data expected. The server should leave data in its cache.

- *IO_ADVISE4_SEQUENTIAL_DONTCACHE: Sequential access to data expected. The server does not need to leave data in its cache.

- *IO_ADVISE4_RANDOM: Random access to data expected.

- *IO_ADVISE4_PREFETCH: Not applicable.

- *IO_ADVISE4_PREFETCH_OPPORTUNISTIC: Not applicable.

- *IO_ADVISE4_INTENT_TO_WRITE: Byte range will be over-written soon so no point in caching data.

- *IO_ADVISE4_RECENTLY_USED: Explicit hint to keep data of byte range in cache.

The results indicate which advice the server intends to follow. The server MUST NOT return an error if it does not recognize or does not support the requested advice.
The intent is that WRITE_WITH_ADVICE is preferred over WRITE. In addition to providing I/O hints, WRITE_WITH_ADVICE uses 64 bit data lengths, which anticipates the expected improvements in average network speeds and network buffer capacities. Because the XDR standard does not support 64 bit array lengths, the large_buffer4 data type is introduced to encode an array of zero or more buffers of fixed size of 2^32 bytes, followed by a variable length array of up to 2^32 - 1 bytes
If general, if the value of wwaa_stable is valid, then the value of wwar_committed in the reply MUST NOT be less than the value of wwaa_stable. The exception is if the wwaa_stable is LAYOUT_SYNC4. LAYOUT_SYNC4 is an enumerated value that can be used by the client when the server is an pNFS data server, and the client has a layout that covers the byte range specified by iod_offset and the amlount of data in wwaa_data. If the client sends a WRITE_WITH_ADVICE to a data server with wwaa_stable set to LAYOUT_SYNC4, then a successful reply MUST return value of wwar_committed equal to LAYOUT_SYNC4 or FILE_SYNC4. Regardless what value wwaa_stable is, if the server is a pNFS data server, it MAY return a value of wwar_committed equal to LAYOUT_SYNC4. Whenever wwar_committed is LAYOUT_SYNC4, this indicates that range of the layout covered by iod_offset and wwar_count has been committed to the metadata server, and there is not need to send a LAYOUT_COMMIT for that range.

TOC

6. Operation XX: SET_WORKFLOW_TAG - Sets the workflow tag of a given session

6.1. ARGUMENT

TOC

```
struct SET_WORKFLOW_TAG 4args {

        uint64_t  swta_tag;
};
```

6.2. RESULT

TOC

```
nfsstat4
```

6.3. MOTIVATION

TOC

Enterprise applications require guarantees of quality and/or priority of service Providing end-to-end guarantees requires awareness at the file services level of the necessary quality and/or priority.

6.4. DESCRIPTION

TOC

Sets the workflow tag of a given session. All operations in progress before the server receives SET_WORKFLOW_TAG use the previous tag (if any). All operations received after the server receives SET_WORKFLOW_TAG use the new tag.

TOC

7. Operation XX: SESSION_CTL - Adjust session parameters

7.1. ARGUMENT

TOC

```
 struct channel_attrs4 { /* from NFSv4.1 */
        count4                  ca_headerpadsize;
        count4                  ca_maxrequestsize;
        count4                  ca_maxresponsesize;
        count4                  ca_maxresponsesize_cached;
        count4                  ca_maxoperations;
        count4                  ca_maxrequests;
        uint32_t                ca_rdma_ird&lt1>;
 };

 /* from NFSv4.1 */

 const CREATE_SESSION4_FLAG_PERSIST              = 0x00000001;
 const CREATE_SESSION4_FLAG_CONN_BACK_CHAN       = 0x00000002;
 const CREATE_SESSION4_FLAG_CONN_RDMA            = 0x00000004;

struct session_ctl4 {
       uint32_t               sc_flags;
       channel_attrs4         sc_fore_chan_attrs;
       channel_attrs4         sc_back_chan_attrs;
};

typedef session_ctl SESSION_CTL4args;
```

TOC

7.2. RESULT

```
union SESSION_CTL4res switch (nfsstat4 scr_status) {
case NFS4_OK:
        session_ctl4  scr_resok4;
default:
        void;
};
```

7.3. MOTIVATION

TOC

The introduction of the session model in NFSv4.1 imposes an explicit limitation on the number of outstanding requests a client can make of an NFS server. In enterprise applications, it is possible each NFS request corresponds to a single application request. Thus, the size of the slot table can bound the number of outstanding application requests. While there are workarounds (examples include (1)implement a mapping layer between application's request slot list and the client's slot table (2) create additional sessions in order to preserve a one-to-one mapping between application and client slots), these workarounds introduce complexity. The application's needs for more slots are dynamic. The NFSv4.1 model assumes a dynamic slot table, but the size of the slot table is driven by the server via the reply to the SEQUENCE operation and the CB_RECALL_SLOT operation. What is missing is a method for the client to request a larger slot table.

7.4. DESCRIPTION

TOC

This operation allows the client to request changes to the session's parameters. There are three major fields in the arguments and results:

*sc_flags. These flags correspond to the csa_flags and csr_flags argument and result of CREATE_SESSION. In the result, the value of a bit in sc_flags MUST be one of:

-The corresponding bit in sc_flags of the arguments to SESSION_CTL.

-The corresponding bit in sc_flags of the result of the previous SESSION_CTL that the server executed.

- If the server has not executed a previous SESSION_CTL, then the corresponding bit in the csr_flags field of the reply the CREATE_SESSION operation that created the session.

* sc_fore_chan_attrs. In the arguments of SESSION_CTL, the fields within sc_fore_chan_attrs correspond to the fields of the argument csa_fore_chan_attrs in the arguments of CREATE_SESSION. In the results of SESSION_CTL, the values fields within sc_fore_chan_attrs correspond to the fields of the result csr_fore_chan_attrs in the response to CREATE_SESSION. The values of the fields in the result sc_fore_chan_attrs are governed according to the same rules that govern the values of the fields of csr_fore_chan_attrs.

* sc_back_chan_attrs. In the arguments of SESSION_CTL, the fields within sc_back_chan_attrs correspond to the fields of the argument csa_back_chan_attrs in the arguments of CREATE_SESSION. In the results of SESSION_CTL, the values fields within sc_back_chan_attrs correspond to the fields of the result csr_back_chan_attrs in the response to CREATE_SESSION. The values of the fields in the result sc_back_chan_attrs are governed according to the same rules that govern the values of the fields of csr_back_chan_attrs.

The SESSION_CTL operation MUST be sent on a COMPOUND operation prefixed by a SEQUENCE operation with the sa_slotid argument set to zero. If SESSION_CTL requests a smaller slot table on the fore channel, and there are operations in progress on other slots of the fore channel, the server MUST do one of (1) return NFS4ERR_FORE_CHAN_BUSY (a new error); (2) allow SESSION_CTL to succeed, wait for the in progress operations to complete and reply to those operations before replying to SESSION_CTL; or (3) if all the in progress operations allow the one or both of the errors NFS4ERR_DELAY or NFS4ERR_SERVERFAULT, allow SESSION_CTL to succeed, abort the in progress operations, reply with to those operations with either NFS4ERR_DELAY or NFS4ERR_SERVERFAULT, and then reply to SESSION_CTL. Because a server is free to return NFS4ERR_FORE_CHAN_BUSY, it is strongly RECOMMENDED that when a client sends a SESSION_CTL operation that it have no other requests in progress.

If SESSION_CTL request a smaller slot table on the backchannel and there are operations in progress on other slots of the backchannel, the server MUST do one of (1) return NFS4ERR_BACK_CHAN_BUSY; (2) allow SESSION_CTL to succeed, and for wait replies to the in progress backchannel operations before replying to SESSION_CTL; or (3) if all the in progress operations allow the one or both of the errors NFS4ERR_DELAY or NFS4ERR_SERVERFAULT, allow SESSION_CTL to succeed, abort the in progress operations, reply with to those operations with either NFS4ERR_DELAY or NFS4ERR_SERVERFAULT, and then reply to SESSION_CTL. Before a client sends a SESSION_CTL operation, it SHOULD

reply to all in progress backchannel requests of the same session as the SESSION_CTL operation.

---

8. Modification to Operation 42: EXCHANGE_ID - Instantiate Client ID

TOC

8.1. ARGUMENT

TOC

```
/* new */
const EXCHGID4_FLAG_SUPP_FENCE_OPS     = 0x00000004;
```

8.2. RESULT

TOC

Unchanged

8.3. MOTIVATION

TOC

Enterprise applications require guarantees that an operation has either aborted or completed. NFSv4.1 provides this guarantee as long as the session is alive: simply send a SEQUENCE operation on the same slot with a new sequence number, and the successful return of SEQUENCE indicates the previous operation has completed. However, if the session is lost, there is no way to know when any in progress operations have aborted or completed. In hindsight, the NFSv4.1 specification should have mandated that DESTROY_SESSION abort/complete all outstanding operations.

TOC

8.4. DESCRIPTION

A client SHOULD request the EXCHGID4_FLAG_SUPP_FENCE_OPS capability when it sends an EXCHANGE_ID operation. The server SHOULD set this capability in the EXCHANGE_ID reply whether the client requests it or not. If the client ID is created with this capability then the following will occur:

*The server will not reply to DESTROY_SESSION until all operations in progress are completed or aborted.

*The server will not reply to subsequent EXCHANGE_ID invoked on the same Client Owner with a new verifier until all operations in progress on the Client ID's session are completed or aborted.

*When DESTROY_CLIENTID is invoked, if there are sessions (both idle and non-idle), opens, locks, delegations, layouts, and/or wants (Section 18.49) associated with the client ID are removed. Pending operations will be completed or aborted before the sessions, opens, locks, delegations, layouts, and/or wants are deleted.

*The NFS server SHOULD support client ID trunking, and if it does and the EXCHGID4_FLAG_SUPP_FENCE_OPS capability is enabled, then a session ID created on one node of the storage cluster MUST be destroyable via DESTROY_SESSION. In addition, DESTROY_CLIENTID and an EXCHANGE_ID with a new verifier affects all sessions regardless what node the sessions were created on.

---

9. Acknowledgements

TOC

Contributors to this document include: Sumanta Chatterjee, Steve Daniel, Mike Eisler, Jeff Kimmel, Akshay Shah, Margaret Susairaj, and Lynne Thieme. Reviewers of this document include: Dave Noveck.

---

10. IANA Considerations

TOC

The IO_ADVISE4 flags are considered extendable. Values 32 through 63 are reserved for private use. All others are standards track.

TOC

11. Security Considerations

None.

---

12. References

TOC

12.1. Normative References

TOC

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

12.2. Informative References

TOC

[I-D.eisler-nfsv4-pnfs-dedupe]	Eisler, M., "Storage De-Duplication Awareness in NFS," draft-eisler-nfsv4-pnfs-dedupe-00 (work in progress), October 2008 (TXT).
[I-D.eisler-nfsv4-pnfs-metastripe]	Eisler, M., "Metadata Striping for pNFS," draft-eisler-nfsv4-pnfs-metastripe-01 (work in progress), October 2008 (TXT).
[I-D.faibish-nfsv4-pnfs-access-permissions-check]	Faibish, S., Black, D., Eisler, M., and J. Glasgow, "pNFS Access Permissions Check," draft-faibish-nfsv4-pnfs-access-permissions-check-03 (work in progress), July 2010 (TXT).
[I-D.ietf-nfsv4-minorversion1]	Shepler, S., Eisler, M., and D. Noveck, "NFS Version 4 Minor Version 1," draft-ietf-nfsv4-minorversion1-29 (work in progress), December 2008 (TXT).
[I-D.lentini-nfsv4-server-side-copy]	Lentini, J., Eisler, M., Kenchammana, D., Madan, A., and R. Iyer, "NFS Server-side Copy," draft-lentini-nfsv4-server-side-copy-05 (work in progress), July 2010 (TXT).
[I-D.myklebust-nfsv4-pnfs-backend]	Myklebust, T., "Network File System (NFS) version 4 pNFS back end protocol extensions," draft-myklebust-nfsv4-pnfs-backend-00 (work in progress), July 2009 (TXT).
[I-D.quigley-nfsv4-sec-label]	Quigley, D. and J. Morris, "MAC Security Label Support for NFSv4," draft-quigley-nfsv4-sec-label-01 (work in progress), February 2010 (TXT).
[RFC3530]	

	Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, “Network File System (NFS) version 4 Protocol,” RFC 3530, April 2003 (TXT).

Authors' Addresses

TOC

	Michael Eisler (editor)
	NetApp
	5765 Chase Point Circle
	Colorado Springs, CO 80919
	US
Phone:	+1 719 599 9026
Email:	mike@eisler.com
	Margaret Susairaj (editor)
	Oracle
	7806 Garden Bend
	Sugar Land, TX 77479
	US
Phone:	+1 408 431 7405
Email:	Margaret.Susairaj@oracle.com