nfsv4
Internet-Draft
Expires: September 15, 2011

D. Noveck
EMC
P. Erasani
L. Bairavasundaram
NetApp
P. Dai
C. Karamonolis
Vmware

March 14, 2011

Storage Control Extensions for NFS Version 4
draft-dnoveck-storage-control-01

Abstract

Developments in storage systems have made it important for applications to have control over the characteristics of the storage that will be used for their particular files. The development of pNFS has added to the usefulness of such control mechanisms as it has created the opportunity for the hierarchical organization of file names to be separated from the control of storage characteristics for individual files, including the assignment to storage locations to reflect the performance or other needs of those specific files. This document proposes extensions to NFS version 4 to allow storage requirements to be communicated to the NFS version 4 server.

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 September 15, 2011.

Copyright Notice

Table of Contents

1. Storage Control Issues

Storage to which files may be assigned can differ in a number of ways, raising the issue of how to control the choice of storage for specific files. The range of such choices is not static but can be expected to increase as flash memory becomes an option whose use needs to be controlled, or various choices of types of local caching need to be made. Although all files may well be helped by such approaches, the degree to which they will be helped will vary with the type of file and the typical application reference pattern for it. In addition, the value of improved access will differ with quick access to certain files being of much greater value, thereby justifying the allocation of more expensive storage resources to such files.

The traditional way that user decisions regarding assignment of storage resources have been effected is by assigning specific file systems to specific disks or sets of disks. Files placed in that file system thereby get the storage characteristics assigned to that file system. Where file systems contain storage of various types, various heuristics are used to assign files or pieces thereof, to storage of various types, generally without any external input about application needs.

The creation of pNFS modifies this pattern in that data and metadata are separated. Where pNFS is used, assigning a file to a specific file system now controls only where the metadata is located. Different files may have their data assigned to different sorts of storage, potentially located on different servers. This gives rise to the need for a means by which the storage choice for a particular file may be made.

NFS version 4.1 contains a layouthint attribute but this does not really address the problem. The focus of the layouthint attribute is on the striping configuration, but there is a need to control storage characteristics other than this. This is the case even when there is only a single stripe (that is, no striping). Even though this is not "parallel NFS," using pNFS in this way to provide a separation of data and metadata, with the ability to choose locations for data based on its characteristics subject to later change in a user-transparent manner is very powerful, particularly if the storage location is subject to intelligent management.

Additionally, more sophisticated storage management arrangements make it desirable to have a way to specify details for storage handling, even when pNFS is not used. When a file system contains different sorts of storage, input regarding desired or necessary storage characteristics can be used to make storage assignment choices more

in line with application needs.

As a result, the ability to specify desired storage characteristics can provide benfits, both when pNFS is used and when it is not, although pNFS has the most immediate set of needs for means by which to control storage selection.

2. Storage Choice and API Definition

It needs to be noted that existing API's may not provide means by which some of the storage characteristics described herein may be communicated to NFSv4 in-kernel clients and from there, to NFSv4 servers. Nevertheless, definition of a means by which these storage characteristics may be communicated to the NFSv4 server is still useful for a number of reasons:

- Embedded clients for particular applications may specify this information even without any API deinition.

- Client implementations may use various less-than-perfect ways of specifying storage characteristics, assigning storage chatcteristics based on file ownership or other nominally unrealated characteristics that that corelate well with customer intentions.

Note that if the absence of a standard kernel API were sufficient to stop this work, it also probably be the case that the absence of a means to communicate the information to remote servers might make the definition of that API not worth the effort. By defining some storage characteristics and a general means of communicating them and others (via an extension mechanism) we allow for either:

- The later development of API's to specify these storage characteristics.

- The developemt of API's to specify different sets of storage characteristics that can then be easily assimilated to this mechanism as extensions.

3. Modes of Storage Choice

There are a number of different ways in which storage choices may be indicated:

- The specific file system location(s) might be specified.
- Specific types of storage might be specified with selection of such choices as SSD, SATA, or fiber channel SAN drives being made by the client and effected by the MDS.
- Desired characteristics of storage including speed (latency and/or throughput), amount of storage that will be needed, safety (raid-level). Available storage would be selected to meet the required characteristics and would be subject to active management as the environment changes.

These different modes of storage choice are all useful in different environments. Specification of a specific file system imposes the least need for a storage management infrastructure but it requires user/application knowledge.

The other modes imply a sequence of progressively greater infrastructure requirements to map specifications to specific storage systems and a correspondingly smaller need for user/application knowledge of the storage environment. However, such modes of operation are very different from existing storage management paradigms and the precise ways in which applications and storage might communicate are not fully understood.

4. Assuring Extensability

4.1. Requirements for Extensability

As the examples of different modes of storage choice suggest, there are potentially a large number of specific items that might be specified in order to effect storage choice. Further, in many cases, expected future developments in the area of storage can be expected to extend and otherwise modify the characteristics which might be specified.

The need for extensibility is important as one might expect many ongoing developments, including those in the areas of storage hardware, and file systems, to create corresponding needs to specify relevant storage chatacteristics.

For example, local caching, including writeback caching using flash, creates the opportunity for greatly improved performance, at the risk of greater complexity in dealing with network failures. This raises the issue of allowing the user to make the choice of whether this greater performance is worth the risks and difficulties.

Similarly, the development of distributed file systems raises many choices where performance will need to be balanced against various forms of safety issues, with specific choices reflecting the specific needs of applications dealing with the storage.

These situations and others that we may not be able to predict, require that any attribute scheme in this area allow the specification of multiple storage characteristics with the ability to easily extend the specification so that it incorporates new characteristics to govern storage selection. Further, the need for actual use testing before incorporation in an IETF standard, imposes new requirements as far as organizing specification of the characteristics.

Having "working code" to effect characteristic selection is not sufficient to demonstrate usefulness. The working code may be trivial while finding out whether this set of characteristics make sense for applications to use or requires extension or modification before assuming its final form is not trivial. This may require significant trial use among a large set of users running different applications, before the details are ready to be standardized.

These factors increase the need for flexibility, including non-private use of characteristics not yet standardized. Accommodating this need for flexibility has the potential for unduly interfering with interoperability and the design of this feature will need to

avoid that.

4.2. XDR Encoding for Extensability

While each storage property could conceivably be made its own attribute, the burden that this would place on the IETF process would be immense. There would be necessary co-ordination (and almost certain confusion) as individual experimental properties needed temporary attribute numbers and then had to shift them to other more permanent numbers. Further, and even more of an issue, storage property definition would seem to require a minor version, which seems too heavyweight. This would slow down the process beyond what should be for something which was its own standard-track RFC.

In order to address these issues, individual properties will be treated as sub-attributes within a single storage_ctl attribute. To simplify assignment of sub-attribute numbers, mainly in support of experimental use, multiple sub-attribute spaces will be supported, to allow independent development of features each involving multiple storage properties. Once such a feature is standardized, the definition of the specific sub-atribute space could simply be made the subject of a standards-track RFC, with no change to those using it.

```
typedef uin32_t  spacenum_sc;    /* Individual property space id. */
typedef uint32_t bitmap_sc<*>;   /* Bit map for the presence or
                                    absence of individual properties
                                    using bit numbers assigned for
                                    the space. Like bitmap4.      */
typedef opaque   proplist_sc<*>; /* Data associated with each of the
                                    properties in the bitmap_sc.
                                    Like attrlist4.               */

struct section_sc {
   spacenum_sc   SpaceSection;   /* Section number.                */
   bitmap_sc     WhichProperties;/* Bit map of properties present. */
   proplist_sc   PropertyData;   /* Data for each of the properties
                                    specified in this section.     */
};

typedef section_sc fattr4_storage_ctl<*>;
                                 /* The attribute may have one or
                                    more property sections. */
```

This form of property encoding allows the property set to be extended without requiring a new minor version. Also, by allowing property

space numbers to be assigned, property sets can be developed independently, and converted to a standard state without undue interruption to those using the earlier form.

5. Storage Control

Storage, along with compute, memory, and network, is an integral part of an application's resources. Much like the other types of resources consumed by an application, storage needs can be described using a set of properties. These properties may serve to describe the characteristics of the storage, the intended usage both temporal and spatial, quality of service expectations, physical layout over available storage media, data access locations, geographical distribution, just to name a few. The collection of such properties together define the control an application ultimately wants to have on storage; conversely, they enable the storage system to more effectively and dynamically meet the application's needs as specifically expressed, rather than inferred, based on fallible heuristics. Henceforth, we will use the term control to refer to the property collection.

It is not difficult to conceive various storage properties. In fact, there are numerous of them, due to the diversity of applications and the corresponding workload characteristics, the ever increasing storage value-adds in the form of data services, and the fast changing business requirements. It is an impossible task to capture all of them here. Rather, the goal of this document is to define a framework in which new properties can be easily added and new semantics of the properties can be introduced as necessary without disruption. It is desired that they be capable of being used in more limited situations, refined as necessary.

5.1. Property Types

There may be numerous storage properties as mentioned above. We need, however, to distinguish at least two types, namely, informative properties and enforceable properties. There may very well be other systems or criteria when it comes to the classification of storage properties; and extensibility shall apply in this case just as it does to adding new storage properties. However, there is a need to explicitly capture the distinctions between informative and enforceable properties in the data model, due to the impact on the storage protocol semantics.

5.1.1. Informative Properties

An informative property, as the name suggests, provides some descriptive information about the storage in question. Such information is furnished in a single direction from the application to the storage system with absolutely no "contractual" implications. The storage system may use the information captured in such a property for storage optimization. But it is not obligated to do so.

More importantly, the application is not offered any transparency as to how the storage system may utilize this information. As such, the information flow is strictly one-way without the prospect for any feedback. Examples of informative properties are the access pattern of the storage in use, the expected capacity need, and the estimated growth rate.

5.1.2. Enforceable Properties

In contrast, an enforceable property may have embedded in it varying degrees of binding effect. By that, it means the application specifying the property has expectations that the storage system not only acts upon but also conveys the action status back in some way. Unlike the case of an informative property, the information flow in this case is truly bi-directional, with the backward direction for monitoring property status, including information on whether a property has been satisfied or is in the process of being satisfied. In that sense, an enforceable property has a resemblance to an agreement, where one might monitor the performance of the other party.

Applications seeking tighter control of the storage may resort to the enforceable properties. Examples of enforceable properties could include the type and speed of sorage but could also include the availability, reliability, and average throughput and latency.

5.1.2.1. Enforcement Level

To allow varying degrees of control, an enforcement level may be associated with an enforceable property. There are two levels of control possible, namely, advisory and mandatory. Regardless of the level, the storage system should strive to fulfill an enforceable property. The difference lies in the treatment of an inability to do so. With an advisory enforcement level, the storage system shall continue to carry out the operation even if the property could not be fulfilled; whereas with mandatory, the storage shall fail the operation without making any modification. In any case, the failure to fulfill an enforceable property can be communicated to the application.

5.1.2.2. Compliance Status

While control may suffice to describe the ultimate storage requirements, i.e., the intended behavior once it has been fully implemented, it does not by itself capture the dynamic aspects of the implementation process. This is encompassed by the concept of "compliance" which indicates the extent to which requested storage properties have or have not been provided or whether they are still

in the process of being provided. Note that the word "compliance" as used here has no connection with this word as used to describe issues conformance with a set of legal requirements for record-keeping, among other matters.

Control implementation can be a fairly heavyweight process by nature due to the data intensity involved. This may be true whether it is during the initial provisioning of storage, or the subsequent change management, or the remediation of compliance violation. The data intensive nature of the control implementation process implies that the transition from non-compliance to compliance will not be instantaneous in the general case. In other words, the implementation process remains asynchronous relative to the operation that triggers it.

The asynchronous nature of the control implementation process may be captured by the compliance status. The compliance status may have three different values, namely, Current, Complying, and Failed. The value Current represents a fully compliant state. The value Complying refers to a transient state in which the transition to current is in progress.

The value Failed represents an indefinite state of non-compliance. In the last case, the storage system may have made the determination that it is unable to fulfill some or all of the storage properties given the physical resources available. The application will work without, but its performance may not be what is desired.

The compliance status describes the state of the control fulfillment as it pertains to each property. It applies to an enforceable property only. Its presence is not a syntactic requirement as defined by the XDR specification. Depending on the operational context in which the enforceable property is specified, specification of compliance status may be either invalid, required, or optional with the specification of more that one such status values possible in some cases.

5.1.2.3. XDR Encoding for Enforceable Properties

Enforceable properties contain a word which is of type enforce_sc and allows the enforcement level and compliance status to be specified. To allow greatest flexibility, all enforcement statuses and compliance status values are specified as bit values, allowing sets of enforcement levels and complicance status, to be specified, as appropriate.

```
typedef uint32_t enforce_sc;

const enforce_sc ENFORCE_MANDATORY = 0x1;
const enforce_sc ENFORCE_ADVISORY = 0x2;
const enforce_sc ENFORCE_CURENT = 0x10;
const enforce_sc ENFORCE_COMPLYING = 0x20;
const enforce_sc ENFORCE_FAILED = 0x40;
```

For most purposes, enforcement words should have a single enforcement level, either ENFORCE_MANDATORY ENFORCE_ADVISORY. Any enforcement word containing both bits will result in NFS4ERR_SCTL_BADENF being returned. Specification of an enforcement word containing neither will generally result in in NFS4ERR_SCTL_BADENF being returned. However, it may be specified, when doing a SETATTR that specifies a reserved empty parameter value to remove a property specification. Also, it may be specified when doing an VERIFY or NVERIFY to specify a property without a defined enforcement level.

When specifying a storage property as part of a OPEN, CREATE. or SETATTR, no enforcement level bits should be specified. If they are, the error NFS4ERR_SCTL_BADENF is returned. For values returned by the server in response to GETATTR, enforcement words, containing exactly one compliance status bit will be returned. When using storage properties as part of VERIFY or NVERIFY compliance words containing no compliance bits or any subset of the valid compliance status bits may be specified.

5.2. Base Property Specifications

The goal for initial inclusion in an NFS version 4 minor version is to define a small set of property specifications that are generally useful and do not require a large management infrastructure to implement. The following are the three property specifications that fit the description.

```
const spacenum_sc SCNUM_BASE = 1;   /* Base property space id for
                                       all properties in this
                                       group. */

const uint32_t SCBASE_SIZE = 0;     /* Informative property for
                                       size. */
const uint32_t SCBASE_DURATION = 1; /* Informative property for
                                       duration. */
const uint32_t SCBASE_DEVFAIL = 2;  /* Enforceable property for
                                       a device failure limit. */
const uint32_t SCBASE_SYSFAIL = 3;  /* Enforceable property for
                                       a system failure limit. */
const uint32_t SCBASE_FAIL_RPO = 4; /* Enforceable property for
                                       a recovery point objective
                                       in the event of failure. */
const uint32_t SCBASE_SFAIL_RTO = 5;/* Enforceable property for
                                       a recovery time objective
                                       in the event of system
                                       failure. */
const uint32_t SCBASE_DLOSS_RTO = 6;/* Enforceable property for
                                       a recovery time objective
                                       in the event of data loss. */
const uint32_t SCBASE_DISASTER_RTO = 7;/* Enforceable property for a
                                          recovery time objective in
                                          the event of disaster. */
```

5.2.1. Storage Size

The storage size is an informative property that allows the specification of the expected amount of storage to be needed. It may be used by the server in seeing if appropriate space is available and in reserving space. It is specified as a 64-bit unsigned value giving a quantity of storage expressed in bytes.

```
typedef uint64_t propbase_size;
```

This value may be different from the expected file size. Areas not allocated, because of holes for example, are not included. This amount of storage may not be required immediately if the file starts small and grows. Any derating of specified values is purely a matter of server implementation choice and will typically reflect the ability to move data to respond to storage overcommitment.

A value of zero is invalid and would result in the error NFS4ERR_SCTL_BADPARM when used in an OPEN or CREATE. When used in SETATTR, it causes deletion of a previous storage size specification.

5.2.2. Storage Use Duration

The storage use duration is an informative property that allows the specification of the amount of time that the storage is expected to be needed. It may be used in assigning files to storage so that space conflicts are reduced. It is specified as a 64-bit unsigned value giving a duration in milliseconds.

```
typedef uint64_t propbase_duration;
```

This allows times from 1 millisecond up to approximately 500 million years to be specified. A value of zero is invalid and would result in the error NFS4ERR_SCTL_BADPARM when used in an OPEN or CREATE. When used in SETATTR, it causes deletion of a previous storage duration specification.

5.2.3. Storage Device Failure Limit

The storage device failure limit is an enforceable property that allows the specification of a number of disk drives (or other devices) that can fail simultaneously with no data loss and that incurs zero recovery time. It must be the case that any set of devices of the specified can fail without data loss and with zero recovery time.

Even though there is no recovery time, there may be a significant recovery period of modestly reduced performance while adaptation to the failure is done and until the completion of which, additional device failures will be considered simultaneous.

The limit is specified as a 32-bit unsigned value giving the minimum count of simultaneous failures that can result in data loss to clients accessing the file. Storage is assigned which either matches this specification or provides a greater value. When pNFS is involved the specification applies to storage for the MDS and each DS.

```
typedef uint32_t prop_dev_fail_lim;

struct propbase_device_failure_limit {
    enforce_sc        DflEnforce;
    prop_dev_fail_lim DflLimit;
};
```

This allows values from zero to approximately 4 billion to be specified. A value of zero is valid and specifies that data loss is tolerable in the event of single device failure. (e.g. RAID-0)

5.2.4. Storage System Failure Limit

The storage system failure limit is an enforceable property that allows the specification of the number of storage systems that must be able to fail simultaneously without complete data loss. Storage is assigned which either matches this specification or provides a greater value. When pNFS is involved the specification applies to storage for the MDS and DS's as a unit.

```
typedef uint32_t prop_sys_fail_lim;

struct propbase_system_failure_limit {
    enforce_sc        SflEnforce;
    prop_sys_fail_lim SflLimit;
};
```

This allows values from zero to approximately four billion to be specified. A value of zero is valid and specifies data loss in the event of a single storage system failure is tolerable.

5.2.5. Storage System Failure RPO

The recovery point objective (RPO) is the age of files that must be recovered from backup storage for normal operations to resume if a computer, system, device, or network failure results in data loss. The RPO is expressed backward in time (that is, into the past) from the instant at which the failure occurs, and can be specified in seconds. It is an important consideration in disaster recovery planning.

```
typedef uint64_t prop_sys_fail_RPO;

struct propbase_system_failure_RPO {
    enforce_sc        SfrpoEnforce;
    prop_sys_fail_RPO SfrpoTime;
};
```

This allows values from zero seconds to a value far beyond the age of the universe to be specified. A value of zero is valid and indiactes that a real-time backup that reflects changes immediately as made is required.

5.2.6. Storage System Failure RTO Properties

Recovery time objective (RTO) properties specify is the maximum tolerable length of time that storage assigned may be unavailable in the event of various classes of failures. There are three associated properties, each of which specifies this value for a particular class

of failure:

The system failure RTO property, with the property id SCBASE_SFAIL_RTO, defines the recovery time objective in the event of failures that do not not involve data loss or data corruption.

The data loss RTO property, with the property id SCBASE_DLOSS_RTO, defines the recovery time objective in the event of failures that do not not involve the occurrence of a disaster, defined as a major environmental event such as a hurricane, earthquake, or flood, etc.

The system failure RTO property, with the property id SCBASE_DISASTER_RTO, defines the recovery time objective in the event of any falure including disasters.

The actual RTO is a function of the extent to which the interruption disrupts normal operations and the provisions made to ameliorate this situation. The desired RTO is a function of the urgency to re-establish operations and the consequences of failure to promptly do so. It is an important consideration in recovery planning.

```
typedef uint64_t propbase_sys_fail_RTO;

struct propbase_system_failure_RTO {
    enforce_sc        SfrtoEnforce;
    prop_sys_fail_RTO SfrtoTime;
};
```

RTO values for all of these properties is specified as a 64-bit integer which specifies a number of microseconds. Although sub-second RTO values may be difficult, the specification allows small values which might be useful in the future. The maximum value is approximately five-hundred thousand years.

6. Uses of the Attribute storage_ctl

There are four occasions in which the storage_ctl attribute is referred to as part of an fattr4 when the storage_ctl mask is present.

- As an attribute specified when creating a file or similar object by means of an OPEN or CREATE operation, in order to specify the specific storage properies to control the locations on which the data is to be put and other associated properties.

- As an attribute set in a SETATTR operation to change the requested location properties. Servers may or may not have the ability to change locations on request, but the operation structure will indicate whether the server has or doesn’t have this ability when it is requested.

- As an attribute read in a GETATTR or READDIR operation to determine the currently requested storage properties and the degree to which they are current being complied with.

- As an attribute specified in VERIFY or NVERIFY to test for current location property compliance status.

In addition to the above, a fattr4_storage_ctl of the of the same structure as storage_ctl attribute (although not within an fattr) also appears within the response data in the following situations.

For the OPEN, CREATE, and SETATTR operations, when the error returned is NFS4ERR_SCTL_FAIL. (See Use of storage_ctl when creating a file and Use of storage_ctl in SETATTR for details).

For the response to the FETCH_SCNOTE operation, when there is a pending storage control note to be reported.

For most purposes, a fattr4_storage_ctl which appears in OPEN, CREATE, and SETATTR requests are handled the same and a fattr4_storage_ctl which appears in the responses for OPEN, CREATE, and SETATTR are handled similarly, while the VERIFY and NVERIFY requests form a third similarity group.

6.1. Use of storage_ctl when creating a file

When the storage_ctl attribute is specified when creating a file, it helps decide on the location selected for the file data. If all enforceable properties can be immediately satisfied, then the operation proceeds normally.

If an enforceable property specified as with the manadatory enforcement level cannot be satisfied then the operation fails with the error NFS4ERR_SCTL_FAIL. The response contains, for the case NFS4ERR_SCTL_FAIL, a fattr4_storage_ctl value which consists all such enforceable properties which could not be satisfied.

If there is a situation which is not as serious as the failure above, but still of note, then information relevant to that situation is stored as a pending storage control note, where it can be fetched (in the same COMPOUND) by the FETCH_SCNOTE operation.

The following three classes of items are included in situations leading to a pending storage control note being created.

- An enforceable property of the advisory enforcment level which could not be satisfied, i.e its compliance status is indicated as failed.
- An enforceable property of the advisory enforcement level which could not be immediately satisfied, i.e. its compliance status is indicated as Complying.
- An enforceable property of the mandatory enforcement level which could not be immediately satisfied, i.e. its compliance status is indicated as Complying.

6.2. Use of storage_ctl in SETATTR

A value of the storage_ctl attribute with a structure similar to the OPEN case is used to change properties for an existing file. Existing elements properties, not changed by the storage_ctl attribute remain in effect.

An enforceable property type and the same enforcement level status is overridden by a corresponding one in the new attributes. To delete such an enforceable property element without setting a new one, an enforceable property with no parameter values is used. Similarly, an informative property will override an existing one of the same type and use of the that property specification with no parameters is used to delete an existing informative propety specification without replacing it.

Failures and notifications are indicated via the error code NFS4ERR_SCTL_FAILED and creation of pending storage control notes, just as in the case of OPEN.

6.3. Use of storage_ctl in GETATTR/READDIR

When the storage_ctl attribute is requested as part of GETATTR or READDIR, the fattr4_storage_ctl returned within the file attributes reflects the current informative properties together with the enforceable properties and together with its current compliance status.

The order of the elements need not reflect that used when the attribute was first set. When enforceable properties specify a range of multiple possible values, the one returned in the attribute will reflect the value actually assigned.

6.4. Use of storage_ctl in VERIFY/NVERIFY

The storage_ctl attribute presented to VERIFY or NVERIFY is interpreted as a series of properties each of which results in a truth value. When the truth value for all properties presented is true, VERIFY succeeds and NVERIFY fails. Conversely when not all properties have that truth value, VERIFY fails and NVERIFY succeeds.

When informative properties are present they are compared to the value set at OPEN, CREATE, or the last SETATTR. If no such value had been previously set, the result is treated as non-matching.

Enforceable properties are classified according to three criteria:

- Whether they have parameters that indicate specific values (With-P) or are the special values defined for that purpose for each parameter, which are treated as without parameters (Non-P) where the parameter values taken are those specified in the corresponding property within the file's attributes.

- Whether they are, an enforcement level specified (With-Enf) or not (Non-Enf).

- Whether they are together with one or more compliance level levels specified (With-Comp) or not (Non-Comp).

Given the above classifications, the following sets of characteristics for enforceable properties in the context of storage_ctl for VERIFY, NVERIFY are treated as errors and should cause the return of the error NFS4ERR_SCTL_BAD.

- Non-Comp/Non-Enf/Non-P

- Non-Comp/Non-Enf/With-P

- With-Comp/non-Enf/Non-P

- With-Comp/With-Enf/With-P

Given the above classifications, the following sets of characteristics for enforceable properties in the context of storage_ctl for VERIFY, NVERIFY are handled as discussed below.

Non-Comp/With-Enf/Non-P: is true iff there exists an enforceable property containing elements of the associated enforcement status as part of the storage_ctl attribute of the file.

Non-Comp/With-Enf/With-P: is true iff the enforceable proeprty specified is compatible with the corresponding enforceable property of the associated enforcement level, i.e. if it is possible to satisfy both at the same time, without reference to whether both or either actually is satisfied.

With-Comp/Non-Enf/With-P: is true iff the enforceable property (including a set of of property specifications of the same type) which appear in the storage_ctl attribute passed to the op is consistent with the set of compliance levels (often a single level but sometimes two) in the specification. That is, the actual compliance level must be one of the ones that is specified.

With-CompB/With-Enf/Non-P: is true iff the enforceable property designated by this specification (i.e. that being of the same type of specification and the same enforcement level) is consistent with the set of compliance levels (often a single level but sometimes two) in this specification. That is, the actual compliance level must be one of the ones that is specified.

7. The FETCH_SCNOTE Operation

7.1. SYNOPSIS

```
(cfh) -> note_pres, note_fattr
```

7.2. ARGUMENT

```
/* CURRENT_FH: */
void;
```

7.3. RESULT

```
enum SCFres_type {
        SCFres_ABSENT = 0,
        SCFres_PRESENT = 1
};

union SCFresok switch (SCFres_type note_pres) {
 case FETCH_PRES:
        fattr4_storage_ctl  note_attr;

 case FETCH_ABS:
        void;
};

union FETCHres switch (nfsstat4 status) {
 case NFS4_OK:
        /* CURRENT_FH: opened file */
        FETCH4resok      resok4;
 default:
        void;
};
```

7.4. DESCRIPTION

The FETCH_SCNOTE operation is used to fetch a pending storage control note for a specified file handle (the current file handle). Note that these notes are stored according to the current file handle when the operation which gave rise to them was executed. Thus it will be the directory on (most) OPENs, and the specific file in the event of SETATTR.

This operation uses the current filehandle value to identify the storage control note being sought.

The operation returns an indication of whether the note is present

and if it is a fattr4_storage_ctl value which consists all enforceable properties where there is a lack of adequate compliance to be noted. The use of the the enum scnote_respval rather than a boolean value allows later extension.

If the note is present, it ceases to be so once the operation is executed.

7.5. IMPLEMENTATION

Storage control note items are maintained on a per-COMPOUND-request basis and cease to exist when a COMPOUND fails due to completion or an the occurrence of an error. This makes it desirable to place the FETCH_SCNOTE operation close to, generally immediately after the operation capable of generating the storage control note.

8. Attribute Extension

8.1. Experimental and Other Non-standardized Extensions

In order to support development of extensions to allow control of new file system support attributes, extensions may be defined, each with their own proper space id. The goal is to allow quick deployment of new features, including those that are vendor-specific at the time with the definitions of extensions being publicly available.

Each such extension set should be registered with IANA. The registration will include

- A short name (a few words) by which the extension will be known.
- The name or corporate identity of the owner of the extension.
- Data for the first version of the namespace extension, as described below.

IANA will assign a space id by which the extension will be known.

Successive versions of spaceid properties should be registered by the owner of the extension. The registration should include:

- The namespace name and number.
- The namespace version number. The version number is in the form a series of small (< 256) integers. The length of the series will probably be restricted to something between four and six. The version numbers will not be checked for order but only that they are unique for a given extension.
- A document in the form of an internet draft with information on the namespace elements paralleling this one. The document will contain definitions and property numbers with the space id for all of properties within the extension.

  Successive version may add properties but may not delete them, clarifications to the semantics of existing properties may be made but substantive changes in their semantics should not be made.

  Existing properties may not be defines as invalid or mandatory-to-not-implement but they may be defined as incompatible with some set of new properties.

The definitional document should be subject to expert review but the purpose of the review is to ensure that the document describes the

extension adequately. It should not be rejected simply because the expert would do things differently or believe the specified properties are useful.

8.2. Standardized Extensions

Storage properties may be extended via a standards-track document in a number of ways. Such an extension may be part of a new minor version, but may also be done independent of in a standards-track document other than for a new NFSv4 minor version. When the extension occurs in a new minor version the document should make clear whether the additional properties are recommended (as is normally the case) or mandatory.

The following forms of extension are all valid options:

Adding additional properties to existing standardized property set such as PROP_BASE.

Creating a new property set its own property set id.

Converting a previous experimental property set to standards-track status based on the publication of the RFC [Need to clarify any possible transfer of ownership issues.]

8.3. The storage_ext attribute

The storage_ext attribute is a per-fs attribute which contains information on the storage_ctl extensions suported by the server when used on the associated file system. Servers will often report the same value of the storage_ext attribute for all file systems, but client should not assume that this is the case.

```
struct section_se {
   spacenum_sc   SpaceSction;    /* Section number. */
   bitmap_sc     WhichProperties;/* Supported properties. */
};

typedef section_se fattr4_storage_ext&lt;&#65533;>;
```

The storage_ext attribute consists of section_se arrays, each of which specify the supported properties for a specific space_id. The section_se arrays should be reported in ascending numeric order of spacenum_sc values.

9. Summary

This chapter serves a reference guide to things discussed above. For a more discursive treatment, with less attention due syntax details, see above.

9.1. Errors

This proposal would involve adding the following new errors to the NFS version 4 minor version in which it is included.

NFS4ERR_SCTL_BADPROP Returned when the storage_ctl attribute contains properties with a space id unknown to the server, or with property bits whose diplacement in the bitmap corresponds to property numbers not known to the server as being associated with the current space id.

This error is returnable by OPEN, CREATE, SETATTR, VERIFY, and NVERIFY.

NFS4ERR_SCTL_BADPARM Returned when the storage_ctl attribute contains parameters defined as not valid in connection with the current property. This includes situations in which multiple properties contain values that are defined as inconsistent (as opposed to not being satisfiable).

This error is returnable by OPEN, CREATE, SETATTR, VERIFY, and NVERIFY.

NFS4ERR_SCTL_BADENF Returned when the the storage_ctl attribute contains a enforceable property whose enforce_sc is invalid, in that it contain multiple enforcement level bits, contains no enforcement level bits, in a context in which that is not allowed or contains a set of compliance specification bits that is not appropriate in the current context.

This error is returnable by OPEN, CREATE, SETATTR, VERIFY, and NVERIFY.

NFS4ERR_SCTL_BADDATA Returned when the storage_ctl contains a section_sc whose PropertyData array does not match the length of the properties specified in the associated WhichProperties.

This error is returnable by OPEN, CREATE, SETATTR, VERIFY, and NVERIFY.

NFS4ERR_SCTL_FAIL Returned when a required storage_ctl element cannot be satisfied. This is as opposed to the case in which it is not being able to be satisfied immediately but is in the process of being satisfied.

This error is returnable by OPEN, CREATE, and SETATTR only.

9.2. Semantic constraints

This section lists the semantic contraints on property specifications. We will have situations in which the attribute will fully match specified XDR specification but the specification will not be in line with appropriate contextual constraints. This section will list those constraints, in order to complement the XDR definition above.

There are four categories of constraints that need to be dealt with:

- Whether the properties have the associated parameters specified.
- Whether the properties have an associated enforcement level specified.
- Whether the properties have associated compliance level(s) specified.
- Constraints that involve the validity of combinations of what are otherwise allowed situations with regard to the above.

Each property specifies a particuar value which is invalid and is to be treated as inicateing the absence of property parameters (zero values, zero-length arays, etc.). Specification of the parameters associated with storage properties are generally required and so these special value result in NFS4ERR_SCTL_BADPARM being returned. The only exceptions are SETATTR, for which a storage property without parameters serves to delete the corresponding storage propery in the existing attribute, and VERIFY/NVERIFY where it is allowed under some circumstances, to be discussed below.

Specification of the enforcement level is generally required for enforceable properties. The only exception is VERIFY/NVERIFY where it is allowed under some circumstances, to be discussed below.

Specification of the compliance status for enforceable properties depends on the context in which the properties appears. For OPEN, CREATE, and SETATTR, specification of compliance status is not allowed. VERIFY/NVERIFY specification of multiple compliance status values is allowed, subject to the specific combination constraints

appropriate to VERIFY and NVERIFY as listed below. For all other contexts, whether in GETATTR, READDIR, the responses in the NFS4ERR_SCTL_FAIL case, or in the response to the FETCH_SCNOTE operation, specification of compliance status is required but only a single compliance status must appear.

In addition to the constraints listed above, in the case of a storage_ctl attribute within VERIFY/NVERIFY, the properties within the attribute must meet the additional constraints described in the section Use of storage_ctl in VERIFY/NVERIFY

When sending responses to GETATTR, READDIR, OPEN, CREATE, and SETATTR, the server MUST obey these constraints. When receiving OPEN, SETATTR, VERIFY, and NVERIFY requests that contain the storage_ctl attribute, the server MUST return the error NFS4ERR_SCTL_BADENF if the attribute does not follow the specified constraints and is otherwise valid (matching the XDR property deinition).

These constraints apply to properties introduced by extensions to the storage_ctl attirbute unless explicitly overridden in the document defining the extension. Such a document may add other contextual constraints that apply to the properties defined by that extension.

10. Possible Future Work

This document describes a basic framework for storage control and a basic set of properties. It is a base for development of this feature and could have considerable additions before incorporation in NFSv4 an minor version. On the other hand, the feature is intended to be defined with sufficient flexibility that many of these additions to the feature might be done as subsequent extensions, after the basic feature is made part of an NFSv4 minor version.

The question of which additions are required for an initial version of the feature, which are best deferred to later and which proposed extensions don't really belong is a complex one and will be a major subject of the development of the feature.

The following list, illustrates some of the possible additions that have had some preliminary discussion. It is not intended to be exhaustive, and the examination of other additions not yet thought of is definitely part of the work to be done:

- Addition of other properties to those in this document, that make sense as a basic set of properties, both informative and enforceable, for an initial set to be part of an NFSv4 minor version.

- Mechanisms to allow a set of properties to be applied to a large set of files, including those that are directory-based (with inheritance a possible part of the mix), by bulk attribute change on a client-specified set of files, or by allowing the client to store some set of properties as a persistent object in file system, and allowing subsequent storage control attributes to reference that persistent object.

- Mechanisms to enable the client to determine possible choices (or ranges) for some properties within the context of a given server. This would be to simplify and streamline property negotation.

- Mechanisms by which a server could advertise various possible sets of property choices to deal with environments where there only exists a small set of possible choices each effecting a particular choice for many properties, as opposed to a case where multiple independent property choices are possible.

11. Acknowledgments

Mike Eisler reviewed early drafts of this work and made important contributions in helping define the direction of the effort.

David Black reviewed many drafts of this work and made many helpful suggestions that improved the quality of the result.

Authors' Addresses

David Noveck
EMC
228 South St.
Hopkinton, MA 01748
US

Phone: +1 508 249 5748
Email: david.noveck@emc.com

Pranoop R. Erasani
NetApp
48980 Oat Grass Terrace
Fremont, CA 94539
US

Phone: +1 408 822 3282
Email: pranoop@netapp.com

Lakshmi N. Bairavasundaram
NetApp
475 East Java Drive
Sunnyvale, CA 94089
US

Phone: +1 408 419 5616
Email: lakshmib@netapp.com

Peng Dai
Vmware
5 Cambridge Center
Cambridge, MA 02142
US

Phone: +1 617 528 7592
Email: pdai@vmware.com

Christos Karamonolis
Vmware
3401 Hillview Ave.
Palo Alto, CA 94304
US

Phone: +1 650 427 2329
Email: ckaramonolis@vmware.com