

INTERNET-DRAFT
Expires: April 2006

David Noveck
Network Appliance, Inc.
Rodney C. Burnett
IBM, Inc.

October 2005

Next Steps for NFSv4 Migration/Replication
draft-noveck-nfsv4-migrep-00.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt> The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

Copyright Notice

Copyright (C) The Internet Society (2005). All Rights Reserved.

Internet-Draft Next Steps for NFSv4 Migration/Replication October 2005

Abstract

The `fs_locations` attribute in NFSv4 provides support for fs migration, replication and referral. Given the current work on supporting these features, and the new needs such as support for global namespace, it is time to look at this area and see what further development of this protocol area may be required. This document makes suggestions for the further development of these features in NFSv4.1 and also presents ideas for work that might be done as part of future minor versions.

Table Of Contents

1.	Introduction	3
1.1.	History	3
1.2.	Areas to be Addressed	3
2.	Clarifications/Corrections to V4.0 Functionality	4
2.1.	Attributes Returned by GETATTR and READDIR	4
2.1.1.	fsid	5
2.1.2.	mounted_on_fileid	5
2.1.3.	fileid	6
2.1.4.	filehandle	6
2.2.	Issues with the Error NFS4ERR_MOVED	6
2.2.1.	Issue of when to check current filehandle	7
2.2.2.	Issue of GETFH	7
2.2.3.	Handling of PUTFH	7
2.2.4.	Inconsistent handling of GETATTR	8
2.2.5.	Ops not allowed to return NFS4ERR_MOVED	8
2.2.6.	Summary of NFS4ERR_MOVED	9
2.3.	Issues of Incomplete Attribute Sets	9
2.3.1.	Handling of attributes for READDIR	10
2.4.	Referral Issues	11
2.4.1.	Editorial Changes Related to Referrals	12
3.	Feature Extensions	13
3.1.	Attribute Continuity	13
3.1.1.	filehandle	14
3.1.2.	fileid	14
3.1.3.	change attribute	15
3.1.4.	fsid	15
3.2.	Additional Attributes	16
3.2.1.	fs_absent	16
3.2.2.	fs_location_info	16
3.2.3.	fh_replacement	28

3.2.4.	fs_status	31
4.	Migration Protocol	33
4.1.	NFSv4.x as a Migration Protocol	34
	Acknowledgements	36
	Normative References	36

Internet-Draft Next Steps for NFSv4 Migration/Replication October 2005

	Informative References	37
	Authors' Addresses	37
	Full Copyright Statement	37

[1.](#) Introduction

[1.1.](#) History

When the `fs_locations` attribute was introduced, it was done with the expectation that a server-to-server migration protocol was in the offing. Including the `fs_locations`-related features provided client support which would be used to allow clients to use migration when that protocol was developed and also could provide support for vendor-specific homogeneous server migration, until that point.

As things happened, development of a server-to-server migration protocol stalled. In part, this was due to the demands of NFSv4 implementation itself. Also, until V4 clients which supported these features were widely deployed, it was hard to justify the long-term effort for a new server-to-server protocol.

Now that serious implementation work has begun, a number of issues have been discovered with the treatment of these features in [RFC3530](#). There are no significant protocol bugs, but there are numerous cases in which the text is not clear or contradictory on significant points. Also, a number of suggestions have been made regarding small things left undone in the original specification, leading to the question of whether it is now an appropriate time to rectify those inadequacies.

Another important development has been the idea of referrals. Referrals, a limiting case of migration, were not recognized when the spec was written, even though the protocol defined therein does

support them. See [[referrals](#)] for an explanation of referrals implementation. Also, it has turned out that referrals are an important building-block for the development of a global namespace for NFSv4.

[1.2.](#) Areas to be Addressed

This document is motivated in large part by the opportunity represented by NFSv4.1. First, this will provide a way to revise the treatment of these features in the spec, to make it clearer, to avoid ambiguities and contradictions, and to incorporate explicit discussion of referrals into the text.

NFSv4.1 also affords the opportunity to provide small extensions to these facilities, to make them more generally useful, in particular in environments in which migration between servers of different types is to be performed. Use of these features in a global-namespace environment will also motivate certain extensions.

The remaining issue in this area is the development of a vendor-independent migration mechanism. This is definitely not something that can be done immediately (in v4.1) but the working group needs to figure out when this effort can be revived. This document will examine a somewhat lower-overhead alternative to development of a separate server-to-server migration protocol.

The alternative that will be explored is the use of NFSv4 itself, with a small set of additions, by a server operating as an NFSv4 client to either pull or push file system state to or from another server. It seems that this sort of incremental development can provide a more efficient way of getting a migration mechanism than development of a new protocol that will inevitably duplicate a lot of NFSv4. Since NFSv4 must have the general ability to represent fs state that is accessible via NFSv4, using the core protocol as the base and adding only the extensions needed to do data transfer efficiently and transfer locking state should be more efficient in terms of design time. The needed extensions could be introduced within a minor version. It is not proposed or expected that these extensions would be in NFSv4.1.

[2.](#) Clarifications/Corrections to V4.0 Functionality

All of the sub-sections below deal with the basic functionality described, explicitly or implicitly, in [RFC3530](#). While the majority of the material is simply corrections, clarifications, and the resolution of ambiguities, in some cases there is cleanup to make things more consistent in v4.1, without adding any new functionality. Functional changes are addressed in separate sections.

[2.1](#). Attributes Returned by GETATTR and REaddir

While the [RFC3530](#) allows the server to return attributes in addition to `fs_locations`, when GETATTR is used with a current filehandle within an absent filesystem, not much guidance is given to help clarify what is appropriate. Such vagueness can result in serious interoperability issues.

Instead of simply allowing an undefined set of attributes to be returned, the NFSv4.1 spec should clearly define the circumstances under which attributes for absent filesystems are to be returned.

While some leeway may be necessary to accommodate different NFSv4.1 servers, unnecessary leeway should be avoided.

In particular, there are a number of attributes which most server implementations should find relatively easy to supply which are of critical importance to clients, particularly in those cases in which NFS4ERR_MOVED is returned when first crossing into an absent file system that the client has not previously referenced, i.e. a referral.

NFSv4.1 should require servers to return `fsid` for an absent file system as well as `fs_locations`. In order for the client to properly determine the boundaries of the absent filesystems, it needs access to `fsid`. In addition when at the root of absent filesystem, `mounted_on_fileid` needs to be returned.

On the other hand, a number of attributes pose difficulties when returned for an absent filesystem. While not prohibiting the server from returning these, the NFSv4.1 spec should explain the issues which may result in problems, since these are not always obvious. Handling of some specific attributes is discussed below.

[2.1.1.](#) fsid

The fsid attribute allows clients to recognize when fs boundaries have been crossed. This applies also when one crosses into an absent filesystem. While it might seem that returning fsid is not absolutely required, since fs boundaries are also reflected, in this case, by means of the fs_root field of the fs_locations attribute, there are renaming issues that make this unreliable. Returning fsid is necessary for clients and servers should have no difficulty in providing it.

To avoid misunderstanding, the NFSv4.1 spec should note that the fsid provided in this case is solely so that the fs boundaries can be properly noted and that the fsid returned will not necessarily be valid after resolution of the migration event. The logic of fsid handling for NFSv4 is that fsid's are only unique within a per-server context. This would seem to be a strong indication that they need not be persistent when file systems are moved from server to server, although [RFC 3530](#) does not specifically address the matter.

[2.1.2.](#) mounted_on_fileid

The mounted_on_fileid attribute is of particular importance to many clients, in that they need this information to form a proper response to a readdir() call. When a readdir() call is done within

UNIX, the d_ino field of each of the entries needs to have a unique value normally derived from the NFSv4 fileid attribute. It is in the case in which a file system boundary is crossed that using the fileid attribute for this purpose, particularly when crossing into an absent fs, will pose problems. Note first that the fileid attribute, since it is within a new fs and thus a new fileid space, will not be unique within the directory. Also, since the fs, at its new location, may arrange things differently, the fileid decided on at the directing server may be overridden at the target server, making it of little value. Neither of these problems arises in the case of mounted_on_fileid since that fileid is in the context of the mounted-on fs and unique within it.

[2.1.3.](#) fileid

For reasons explained above under `mounted_on_fileid`, it would be difficult for the referring server to provide a `fileid` value that is of any use to the client. Given this, it seems much better for the server never to return `fileid` values for files on an absent fs.

[2.1.4.](#) `filehandle`

Returning file handles for files in the absent fs, whether by use of `GETFH` (discussed below) or by using the `filehandle` attribute with `GETATTR` or `REaddir` poses problems for the client as the server to which it is referred is likely not to assign the same `filehandle` value to the object in question. Even though it is possible that volatile `filehandles` may allow a change, the referring server should not prejudge the issue of `filehandle` volatility for the server which actually has the fs. By not providing the `filehandle`, the referring server allows the target server freedom to choose the `filehandle` value without constraint.

[2.2.](#) Issues with the Error `NFS4ERR_MOVED`

[RFC3530](#), in addition to being somewhat unclear about the situations in which `NFS4ERR_MOVED` is to be returned, is self-contradictory. In particular in [section 6.2](#), it is stated, "The `NFS4ERR_MOVED` error is returned for all operations except `PUTFH` and `GETATTR`.", which is contradicted by the error lists in the detailed operation descriptions. Specifically,

- o `NFS4ERR_MOVED` is listed as an error code for `PUTFH` ([section 14.2.20](#)), despite the statement noted above.
- o `NFS4ERR_MOVED` is listed as an error code for `GETATTR` ([section 14.2.7](#)), despite the statement noted above.

- o Despite the "all operations except" in the statement above, six operations (`PUTROOTFH`, `PUTPUBFH`, `RENEW`, `SETCLIENTID`, `SETCLIENTID_CONFIRM`, `RELEASE_OWNER`) are not allowed to return `NFS4ERR_MOVED`.

[2.2.1.](#) Issue of when to check current `filehandle`

In providing the definition of `NFS4ERR_MOVED`, [RFC 3530](#) refers to

the "filesystem which contains the current filehandle object" being moved to another server. This has led to some confusion when considering the case of operations which change the current filehandle and potentially the current file system. For example, a LOOKUP which causes a transition to an absent file system might be supposed to result in this error. This should be clarified to make it explicit that only the current filehandle at the start of the operation can result in NFS4ERR_MOVED.

[2.2.2.](#) Issue of GETFH

While [RFC 3530](#) does not make any exception for GETFH when the current filehandle is within an absent filesystem, the fact that GETFH is such a passive, purely interrogative operation, may lead readers to wrongly suppose that an NFSERR_MOVED error will not arise in this situation. Any new NFSv4 RFC should explicitly state that GETFH will return this error if the current filehandle is within an absent filesystem.

This fact has a particular importance in the case of referrals as it means that filehandles within absent filesystems will never be seen by clients. Filehandles not seen by clients can pose no expiration or consistency issues on the target server.

[2.2.3.](#) Handling of PUTFH

As noted above, the handling of PUTFH regarding NFS4ERR_MOVED is not clear in [RFC3530](#). Part of the problem is that there is felt to be a need for an exception for PUTFH, to enable the sequence PUTFH-GETATTR(fs_locations). However, if one clearly establishes, as should be established, that the check for an absent filesystem is only to be made at the start of each operation, then no such exception is required. The sequence PUTFH-GETATTR(fs_locations) requires an exception for the GETATTR but not the PUTFH.

PUTFH can return NFS4ERR_MOVED but only if the current filehandle, as established by a previous operation, is within an absent filesystem. Whether the filehandle established by the PUTFH, is within an absent filesystem is of no consequence in determining

whether such an error is returned, since the check is to be done at

the start of the operation.

[2.2.4.](#) Inconsistent handling of GETATTR

While, as noted above, [RFC 3530](#) indicates that NFS4ERR_MOVED is not returned for a GETATTR operation, NFS4ERR_MOVED is listed as an error that can be returned by GETATTR. The best resolution for this is to limit the exception for GETATTR to specific cases in which it is required.

- o If all of the attributes requested can be provided (e.g. fsid, fs_locations, mounted_on_fileid in the case of the root of an absent filesystem), then NFS4ERR_MOVED is not returned.
- o If an attribute which indicates that the client is aware of the likelihood of migration having happened (such as fs_locations) then NFS4ERR_MOVED is not returned, irrespective of what additional attributes are requested. The newly-proposed attributes fs_absent and fs_location_info (see sections [3.2.1](#) and [3.2.2](#)) would, like fs_locations, also cause NFS4ERR_MOVED not to be returned. For the rest of this document, the phrase "fs_locations-like attributes" is to be understood as including fs_locations, and the new attributes fs_absent and fs_location_info, if added to the protocol.

In all other cases, if the current filesystem is absent, NFS4ERR_MOVED is to be returned.

[2.2.5.](#) Ops not allowed to return NFS4ERR_MOVED

As noted above, [RFC 3530](#) does not allow the following ops to return NFS4ERR_MOVED:

- o PUTROOTFH
- o PUTPUBFH
- o RENEW
- o SETCLIENTID
- o SETCLIENTID_CONFIRM
- o RELEASE_OWNER

All of these are ops which do not require a current file handle, although two other ops that also do not require a current file handle, DELEGPURGE and PUTFH are allowed to return NFS4ERR_MOVED.

There is no good reason to continue these as exceptions. In future NFSv4 versions it should be the case that if there is a current filehandle and the associated filesystem is not present an NFS4ERR_MOVED error should result, as it does for other ops.

2.2.6. Summary of NFS4ERR_MOVED

To summarize, NFSv4.1 should:

- o Make clear that the check for an absent filesystem is to occur at the start (and only at the start) of each operation.
- o Allow NFS4ERR_MOVED to be returned by all ops including those not allowed to return it in [RFC3530](#).
- o Be clear about the circumstances in which GETATTR will or will not return NFS4ERR_MOVED.
- o Delete the confusing text regarding an exception for PUTFH.
- o Make it clear that GETFH will return NFS4ERR_MOVED rather than a filehandle within an absent filesystem.

2.3. Issues of Incomplete Attribute Sets

Migration or referral events naturally create situations in which all of the attributes normally supported on a server are not obtainable. [RFC3530](#) is in places ambivalent and/or apparently self-contradictory on such issues. Any new NFSv4 RFC should take a clear position on these issues (and it should not impose undue difficulties on support for migration).

The first problem concerns the statement in the third paragraph of [section 6.2](#): "If the client requests more attributes than just fs_locations, the server may return fs_locations only. This is to be expected since the server has migrated the filesystem and may not have a method of obtaining additional attribute data."

While the above seems quite reasonable, it is seemingly contradicted by the following text from [section 14.2.7](#) the second paragraph of the DESCRIPTION for GETATTR: "The server must return a value for each attribute that the client requests if the attribute

is supported by the server. If the server does not support an attribute or cannot approximate a useful value then it must not

return the attribute value and must not set the attribute bit in the result bitmap. The server must return an error if it supports an attribute but cannot obtain its value. In that case no attribute values will be returned."

While the above is a useful restriction in that it allows clients to simplify their attribute interpretation code since it allows them to assume that all of the attributes they request are present often making it possible to get successive attributes at fixed offsets within the data stream, it seems to contradict what is said in [section 6.2](#), where it is clearly anticipated, at least when `fs_locations` is requested, that fewer (often many fewer) attributes will be available than are requested. It could be argued that you could harmonize these two by being creative with the interpretation of the phrase "if the attribute is supported by the server". One could argue that many attributes are not supported by the server for an absent fs even though the text by talking about attributes "supported by a server" seems to indicate that this is not allowed to be different for different fs's (which is troublesome in itself as one server might have filesystems that do support and don't support acl's for example).

Note however that the following paragraph in the description says, "All servers must support the mandatory attributes as specified in the section 'File Attributes'". That's reasonable enough in general, but for an absent fs it is not reasonable and so [section 14.2.7](#) and [section 6.2](#) are contradictory. NFSv4.1 should remove the contradiction, by making an explicit exception for the case of an absent filesystem.

[2.3.1](#). Handling of attributes for READDIR

A related issue concerns attributes in a READDIR. There has been discussion, without any resolution yet, regarding the server's obligation (or not) to return the attributes requested with READDIR. There has been discussion of cases in which this is inconvenient for the server, and an argument has been made that the attributes request should be treated as a hint, since the client

can do a GETATTR to get requested attributes that are not supplied by the server.

Regardless of how this issue is resolved, it needs to be made clear that at least in the case of a directory that contains the roots of absent filesystems, the server must not be required to return attributes that it is simply unable to return, just it cannot with GETATTR.

The following rules, derived from section 3.1 of [[referrals](#)], modified for suggested attribute changes in NFSv4.1 represent a good base for handling this issue, although the resolution of the general issue regarding the attribute mask for READDIR will affect the ultimate choices for NFSv4.1.

- o When any of the fs_locations-like attributes is among the attributes requested, the server may provide a subset of the other requested attributes together with the request fs_locations-like attributes for roots of absent fs's, without causing any error for the READDIR as a whole. If rgetattr_error is also requested and there are attributes which are not available, then rgetattr_error will receive the value NFS4ERR_MOVED.
- o When no fs_locations-like attributes are requested, but all of the attributes requested can be provided, then they will be provided and no NFS4ERR_MOVED will be generated. An example would be READDIR's that request mounted_on_fileid either with or without fsid.
- o When none of the fs_locations-like attributes are requested, but rgetattr_error is and some attributes requested are not available because of the absence of the filesystem, the server will return NFS4ERR_MOVED for the rgetattr_error attribute and, in addition, the requested attributes that are valid for the root of an absent filesystem.
- o When none of fs_locations-like attributes are requested and there is a directory within an absent fs within the directory being read, if some unavailable attributes are requested, the handling will depend on the overall decision about READDIR

referred to above. If the attribute mask is to be treated as a hint, only available attributes will be returned. Otherwise, no data will be returned and the REaddir will get an NFS4ERR_MOVED error.

[2.4.](#) Referral Issues

[RFC 3530](#) defines a migration feature which allows the server to direct clients to another server for the purpose of accessing a given file system. While that document explains the feature in terms of a client accessing a given file system and then finding that it has moved, an important limiting case is that in which the clients are redirected as part of their first attempt to access a given file system.

[2.4.1.](#) Editorial Changes Related to Referrals

Given the above framework for implementing referrals, within the basic migration framework described in [RFC 3530](#), we need to consider how future NFSv4 RFC's should be modified, relative to [RFC 3530](#), to address referrals.

The most important change is to include an explanation of how referrals fit into the v4 migration model. Since the existing discussion does not specifically call out the case in which the absence of a filesystem is noted while attempting to cross into the absent file system, it makes it hard to understand how referrals work and how they relate to other sorts of migration events.

It makes sense to present a description of referrals in a new sub-section following the "Migration" section, and would be [section 6.2.1](#), given the current numbering scheme of [RFC 3530](#). The material in [[referrals](#)], suitably modified for the changes proposed for v4.1, would be very helpful in providing the basis for this sub-section.

There are also a number of cases in which the existing wording of [RFC 3530](#) seems to ignore the referral case of the migration feature. In the following specific cases, some suggestions are made for edits to tidy this up.

- o In [section 1.4.3.3](#), in the third sentence of the first paragraph, the phrase "In the event of a migration of a filesystem" is unnecessarily restrictive and having the sentence read "In the event of the absence of a filesystem, the client will receive an error when operating on the filesystem and it can then query the server as to the current location of the file system" would be better.
- o In [section 6.2](#), the following should be added as a new second paragraph: "Migration may be signaled when a file system is absent on a given server, when the file system in question has never actually been located on the server in question. In such a case, the server acts to refer the client to the proper fs location, using fs_locations to indicate the server location, with the existence of the server as a migration source being purely conventional."
- o In the existing second paragraph of [section 6.2](#), the first sentence should be modified to read as follows: "Once a filesystem has been successfully established at a new server location, the error NFS4ERR_MOVED will be returned for subsequent requests received by the server whose role is as

the source of the filesystem, whether the filesystem actually resided on that server, or whether its original location was purely nominal (i.e. the pure referral case)."

- o The following should be added as an additional paragraph to the end of [section 6.4](#), the: "Note that in the case of a referral, there is no issue of filehandle recovery since no filehandles for the absent filesystem are communicated to the client (and neither is the fh_expire_type)".
- o The following should be added as an additional paragraph to the end of [section 8.14.1](#): "Note that in the case of referral, there is no issue of state recovery since no state can have been generated for the absent filesystem."
- o In [section 12](#), in the description of NFS4ERR_MOVED, the first sentence should read, "The filesystem which contains the current filehandle object is now located on another server."

[3.](#) Feature Extensions

A number of small extensions can be made within NFSv4's minor versioning framework to enhance the ability to provide multi-vendor implementations of migration and replication where the transition from server instance to server instance is transparent to client users. This includes transitions due to migration or transitions among replicas due to server or network problems. These same extensions would enhance the ability of the server to present clients with multiple replicas in referral situation, so that the most appropriate one might be selected. These extensions would all be in the form of additional recommended attributes.

[3.1.](#) Attribute Continuity

There are a number of issues with the existing protocol that revolve around the continuity (or lack thereof) of attribute values across a migration event. In some cases, the spec is not clear about whether such continuity is required and different readers may make different assumptions. In other cases, continuity is not required but there are significant cases in which there would be a benefit and there is no way for the client to take advantage of attribute continuity when it exists. A third situation is that attribute continuity is generally assumed (although not specified in the spec), but allowing change at a migration event would add greatly to flexibility in handling a global namespace.

[3.1.1.](#) filehandle

The issue of filehandle continuity is not fully addressed in [RFC3530](#). In many cases of vendor-specific migration or replication (where an entire fs image is copied, for instance), it is relatively easy to provide that the same persistent filehandles used on the source server be recognized on the destination server.

On the other hand, for many forms of migration, filehandle continuity across a migration event cannot be provided, requiring that filehandles be re-established. Within [RFC3530](#), volatile

filehandles (FH4_VOL_MIGRATION) is the only mechanism to satisfy this need and in many environments they will work fine.

Unfortunately, in the case in which an open file is renamed by a another client, the re-establishment of the filehandle on the destination target will give the wrong result and the client will attempt to re-open an incorrect file on the target.

There needs to be a way to address this difficulty in order to provide transparent switching among file system instance, both in the event of migration or when transitioning among replicas.

3.1.2. fileid

[RFC3530](#) gives no real guidance on the issue of continuity of fileid's in the event of migration or a transition between two replicas. The general expectation has been that in situations in which the two filesystem instances are created by a single vendor using some sort of filesystem image copy, fileid's will be consistent across the transition while in the analogous multi-vendor transitions they will not. This latter can pose some difficulties.

It is important to note that while clients themselves may have no trouble with a fileid changing as a result of a filesystem transition event, applications do typically have access to the fileid (e.g. via stat), and the result of this is that an application may work perfectly well if there is no filesystem instance transition or if any such transition is among instances created by a single vendor, yet be unable to deal with the situation in which a multi-vendor transition occurs, at the wrong time.

Providing the same fileid's in a multi-vendor (multiple server vendors) environment has generally been held to be quite difficult. While there is work to be done, it needs to be pointed out that this difficulty is partly self-imposed. Servers have typically

identified fileid with inode number, i.e. with a quantity used to find the file in question. This identification poses special difficulties for migration of an fs between vendors where assigning the same index to a given file may not be possible. Note here that

a fileid does not require that it be useful to find the file in question, only that it is unique within the given fs. Servers prepared to accept a fileid as a single piece of metadata and store it apart from the value used to index the file information can relatively easily maintain a fileid value across a migration event, allowing a truly transparent migration event.

In any case, where servers can provide continuity of fileids, they should and the client should be able to find out that such continuity is available, and take appropriate action.

[3.1.3.](#) change attribute

Currently the change attribute is defined as strictly the province of the server, making it necessary for the client to re-establish the change attribute value on the new server. This has the further consequence that the lack of continuity between change values on the source and destination servers creates a window during which we have no reliable way of determining whether caches are still valid. Where there is a transition among writable filesystem instances, even if most of the access is for reading (in fact particularly if it is), this can be a big performance issue.

Where the co-operating servers can provide continuity of change number across the migration event, the client should be able to determine this fact and use this knowledge to avoid unneeded attribute fetches and client cache flushes.

[3.1.4.](#) fsid

Although [RFC3530](#) does not say so explicitly, it has been the general expectation that although the fsid is expected to change as part of migration (since the fsid space is per-server), the boundaries of a server when migrated will be the same as they were on the source.

The possibility of splitting an existing filesystem into two or more as part of migration can provide important additional functionality in a global namespace environment. When one divides up pieces of a global namespace into convenient-sized fs's (to allow their independent assignment to individual servers), difficulties will arise over time. As the sizes of directories grow, what was once a convenient set of files, embodied as a separate fs, may become inconveniently large. This requires a

means to divide it into a new set of pieces which are of a convenient size. The important point is that while there are many ways to do that currently, they are all disruptive. A method is needed which allows this division to occur without disrupting access.

[3.2.](#) Additional Attributes

A small number of additional attributes in V4.1 can provide significant additional functionality, by addressing the attribute continuity issues discussed above and allowing more complete information about the possible replicas, post-migration locations, or referral targets for a given filesystem that allows the client to choose the one most suited to its needs, and to more effectively handle the transition to a new target server.

All of the proposed attributes would be defined as validly requested when the current filehandle is within an absent filesystem, i.e. an attempt to obtain these attributes would not result in NFS4ERR_MOVED. In some cases, it may be optional to actually provide the requested attribute information based on the presence or absence of the filesystem. The specifics will be discussed under each of the individual attributes.

[3.2.1.](#) fs_absent

In NFSv4.0, fs_locations is the only attribute which, when fetched, indicates that the client is aware of the possibility that the current filesystem may be absent. Since fs_locations is a complicated attribute and the client may simply want an indication of whether the filesystem is present, we propose the addition of a boolean attribute named "fs_absent" to provide this information simply.

As noted above, this attribute, when supported, may be requested of absent filesystems without causing NFS4ERR_MOVED to be returned and it should always be available. Servers are strongly urged to support this attribute on all filesystems if they support it on any filesystem.

[3.2.2.](#) fs_location_info

The fs_location_info attribute is intended as a more functional replacement for fs_locations which will continue to exist and be supported. Clients which need the additional information provided by this attribute will interrogate it and get the information from servers that support it. When the server does not support

fs_location_info, fs_locations can be used to get a subset of the

Internet-Draft Next Steps for NFSv4 Migration/Replication October 2005

information. A server which supports fs_location_info MUST support fs_locations as well.

There are several sorts of additional information present in fs_location_info, that aren't available in fs_locations:

- o Attribute continuity information to allow a client to select a location which meets the transparency requirements of the applications accessing the data and to take advantage of optimizations that server guarantees as to attribute continuity may provide (e.g. change attribute).
- o Filesystem identity information which indicates when multiple replicas, from the clients point of view, correspond to the same target filesystem, allowing them to be used interchangeably, without disruption, as multiple paths to the same thing.
- o Information which will bear on the suitability of various replicas, depending on the use that the client intends. For example, many applications need an absolutely up-to-date copy (e.g. those that write), while others may only need access to the most up-to-date copy reasonably available.
- o Server-derived preference information for replicas, which can be used to implement load-balancing while giving the client the entire fs list to be used in case the primary fails.

Attribute continuity and filesystem identity information define a number of identity relations among the various filesystem replicas. Most often, the relevant question for the client will be whether a given replica is identical-with/continuous-to the current one in a given respect but the information should be available also as to whether two other replicas match in that respect as well.

The way in which such pairwise filesystem comparisons are relatively compactly encoded is to associate with each replica a 32-bit integer, the location id. The fs_location_info attribute then contains for each of the identity relations among replicas a 32-bit mask. If that mask, when anded with the location ids of the

two replicas, result in fields which are identical, then the two replicas are defined as belonging to the corresponding identity relation. This scheme allows the server to accommodate relatively large sets of replicas distinct according to a given criteria without requiring large amounts of data to be sent for each replica.

Server-specified preference information is also provided in a fashion that allows a number of different relations (in this case order relations) in a compact way. In this case each `location4_server` structure contains a 32-bit priority word which can be broken into fields devoted to these relations in any way the server wishes. The `location4_info` structure contains a set of 32-bit masks, one for each relation. Two replicas can be compared via that relation by anding the corresponding mask with the priority word for each replica and comparing the results.

The `fs_location_info` attribute consists of a root pathname (just like `fs_locations`), together with an array of `location4_item` structures.

```
struct location4_server {
    uint32_t      priority;
    uint32_t      flags;
    uint32_t      location_id;
    int32_t       currency;
    utf8str_cis   server;
};

const LIF_FHR_OPEN   = 0x00000001;
const LIF_FHR_ALL    = 0x00000002;
const LIF_MULTI_FS   = 0x00000004;
const LIF_WRITABLE   = 0x00000008;
const LIF_CUR_REQ    = 0x00000010;
const LIF_ABSENT     = 0x00000020;
const LIF_GOING      = 0x00000040;

struct location4_item {
    location4_server entries<>;
    pathname4      rootpath;
};

struct location4_info {
    pathname4      fs_root;
    location4_item items<>;
    uint32_t       fileid_keep_mask;
    uint32_t       change_cont_mask;
```

```

        uint32_t      same_fh_mask;
        uint32_t      same_state_mask;
        uint32_t      same_fs_mask;
        uint32_t      valid_for;
        uint32_t      read_rank_mask;
        uint32_t      read_order_mask;
        uint32_t      write_rank_mask;
        uint32_t      write_order_mask;

};

```

The `fs_location_info` attribute is structured similarly to the `fs_locations` attribute. A top-level structure (`fs_locations4` or `location4_info`) contains the entire attribute including the root pathname of the fs and an array of lower-level structures that define replicas that share a common root path on their respective servers. Those lower-level structures in turn (`fs_locations4` or `location4_item`) contain a specific pathname and information on one or more individual server replicas. For that last lowest-level information, `fs_locations` has a server name in the form of `utf8str_cis`, while `fs_location_info` has a `location4_server`

structure that contains per-server-replica information in addition to the server name.

The `location4_server` structure consists of the following items:

- o The `priority` word is used to implement server-specified ordering relations among replicas. These relations are intended to be used to select a replica when migration (including a referral) occurs, when a server appears to be down, when the server directs the client to find a new replica (see `LIF_GOING`) and, optionally, when a new filesystem is first entered. See the `location4_info` fields `read_order`, `read_rank`, `write_order`, and `write_rank` for details of the ordering relations.
- o A word of flags providing information about this replica/target. These flags are defined below.
- o An indication of file system up-to-date-ness (currency) in

terms of approximate seconds before the present. A negative value indicates that the server is unable to give any reasonably useful value here. A zero indicates that filesystem is the actual writable data or a reliably coherent and fully up-to-date copy. Positive values indicate how out-of-date this copy can normally be before it is considered for update. Such a value is not a guarantee that such updates will always be performed on the required schedule but instead serve as a hint about how far behind the most up-to-date copy of the data, this copy would normally be expected to be.

- o A location id for the replica, to be used together with masks in the location4_info structure to determine whether that replica matches other in various respects, as described above. See below (after the mask definitions) for an example of how the location_id can be used to communicate filesystem information.

When two location id's are identical, then access to the corresponding replicas are defined as identical in all respects. They access the same filesystem with the same filehandles and share v4 file state. Further, multiple connections to the two replicas may be done as part of the same session. Two such replicas will share a common root path and are best presented within two location4_server entries in a common location4_item. These replicas should have identical values for the currency field although the flags and priority fields may be different.

Clients may find it helpful to associate all of the location4_server structures that share a location_id value and treat this set as representing a single fs target. When they do so, they should take proper care to note that priority fields for these may be different and the selection of location4_server needs to reflect rank and order considerations (see below) for the individual entries.

- o The server string. For the case of the replica currently being accessed (via GETATTR), a null string may be used to indicate the current address is using for the RPC call.

The flags field has the following bits defined:

- o LIF_FHR_OPEN indicates that the server will normally make a replacement filehandle available for files that are open at the time of a filesystem image transition. When this flag is associated with an alternative filesystem instance, the client may get the replacement filehandle to be used on the new filesystem instance from the current server. When this flag is associated with the current filesystem instance, a replacement for filehandles from a previous instance may be obtained on this one. See [section 3.2.2](#), fh_replacement, for details. Because of the possibility of hardware and software failures, this is not a guarantee, but when this bit returned, the server should make all reasonable efforts to provide the replacement filehandle.
- o LIF_FHR_ALL indicates that a replacement filehandle will be made available for all files when there is a migration event or a replica switch. Like LIF_FHR_OPEN, it may indicate replacement availability on the source or the destination, and the details are described in [section 3.2.3](#).
- o LIF_MULTI_FS indicates that when a transition occurs from the current filesystem instance to this one, the replacement may consist of multiple filesystems. In this case, the client has to be prepared for the possibility that objects on the same fs before migration will be on different ones after. Note that LIF_MULTI_FS is not incompatible with the two filesystems agreeing with respect to the fileid-keep mask since, if one has a set of fileid's that are unique within an fs, each subset assigned to a smaller fs after migration would not have any conflicts internal to that fs.

A client, in the case of split filesystem will interrogate existing files with which it has continuing connection (it is free simply forget cached filehandles). If the client

remembers the directory filehandle associated with each open file, it may proceed upward using LOOKUPP to find the new fs boundaries.

Once the client recognizes that one filesystem has been split

into two, it could maintain applications running without disruption by presenting the two filesystems as a single one until a convenient point to recognize the transition, such as a reboot. This would require a mapping of fsid's from the server's fsids to fsids as seen by the client but this already necessary for other reasons anyway. As noted above, existing fileids within the two descendant fs's will not conflict. Creation of new files in the two descendent fs's may require some amount of fileid mapping which can be performed very simply in many important cases.

- o LIF_WRITABLE indicates that this fs target is writable, allowing it to be selected by clients which may need to write the on this filesystem. When the current filesystem instance in writable, then any other filesystem to which the client might switch must incorporate within its data any committed write made on the current filesystem instance. See below, in the section on the same-fs mask, for issues related to uncommitted writes. While there is no harm in not setting this flag for a filesystem that turns out to be writable, turning the flag on for read-only filesystem can cause problems for clients who select a migration or replication target based on it and then find themselves unable to write.
- o LIF_VLCACHE indicates that the server is a cached copy where the measured latency of operation may differ very significantly depending on the particular data requested, in that already cached data may be provided with very low latency while other data may require transfer from a distant source.
- o LIF_CUR_REQ indicates that this replica is the one on which the request is being made. Only a single server entry may have this flag set and in the case of a referral, no entry will have it.
- o LIF_ABSENT indicates that this entry corresponds an absent filesystem replica. It can only be set if LIF_CUR_REQ is set. When both such bits are set it indicates that a filesystem instance is not usable but that the information in the entry can be used to determine the sorts of continuity available when switching from this replica to other possible replicas. Since this bit can only be true if LIF_CUR_REQ is true, the value could be determined using the fs_absent attribute but

the information is also made available here for the convenience of the client. An entry with this bit, since it represents a true filesystem (albeit absent) does not appear in the event of a referral, but only where a filesystem has been accessed at this location and subsequently been migrated.

- o LIF_GOING indicates that a replica, while still available, should not be used further. The client, if using it, should make an orderly transfer to another filesystem instance as expeditiously as possible. It is expected that filesystems going out of service will be announced as LIF_GOING some time before the actual loss of service and that the valid_for value will be sufficiently small to allow servers to detect and act on scheduled events while large enough that the cost of the requests to fetch the fs_location_info values will not be excessive. Values on the order of ten minutes seem reasonable.

The location4_item structure, analogous to an fs_locations4 structure, specifies the root pathname all used by an array of server replica entries.

The location4_info structure, encoding the fs_location_info attribute contains the following:

- o The fs_root field which contains the pathname of the root of the current filesystem on the current server, just as it does the fs_locations4 structure.
- o An array of location4_item structures, which contain information about replicas of the current filesystem. Where the current filesystem is actually present, or has been present, i.e. this is not a referral situation, one of the location4_item structure will contain a location4_server for the current server. This structure will have LIF_ABSENT set if the current filesystem is absent, i.e. normal access to it will return NFS4ERR_MOVED.
- o The fileid-keep mask indicates, in combination with the appropriate location ids, that fileids will not change (i.e. they will be reliably maintained with no lack of continuity) across a transition between the two filesystem instances, whether by migration or a replica transition. This allows transition to safely occur without any chance that applications that depend on fileids will be impacted.
- o The change-cont mask indicates, in combination with the appropriate location ids, that the change attribute is

Internet-Draft Next Steps for NFSv4 Migration/Replication October 2005

continuous across a migration event between the server within any pair of replicas. In other words if the change attribute has a given value before the migration event, then it will have that same value after, unless there has been an intervening change to the file. This information is useful after a migration event, in avoiding any need to refetch change information or any requirement to needlessly flush cached data because of a lack of reliable change information. Although change attribute continuity allows the client to dispense with any migration-specific refetching of change attributes, it still must fetch the attribute in all cases in which would normally do so if there had been no migration. In particular, when an open-reclaim is not available and the file is re-opened, a check for an unexpected change in the change attribute must be done.

- o The same-fh mask indicates, in combination with the appropriate location ids, whether two replicas will have the same fh's for corresponding objects. When this is true, both filesystems must have the same filehandle expiration type. When this is true and that type is persistent, those filehandles may be used across a migration event, without disruption.
- o The same-state mask indicates, in combination with the appropriate location ids, whether two replicas will have the same state environment. This does not necessarily mean that when performing migration, the client will not have to reclaim state. However it does mean that the client may proceed using his current clientid just as if there were no migration event and only reclaim state when an NFS4ERR_STALE_CLIENTID or NFS4ERR_STALE_STATEID error is received.

Filesystems marked as having the same state should also have same filehandles. In other words the same-fh mask should be a subset (not necessarily proper) of the same-state mask.

- o The same-fs mask indicates, in combination with the appropriate location ids, whether two replicas in fact designate the same filesystem in all respects. If so, any action taken on one is immediately on the other and the client can consider them as effectively the same thing.

The same-fs mask must include all bits in the same-fh mask, the change-cont mask, and same-state mask. Thus, filesystem instances marked as same-fs must also share state, have the same filehandles, and be change continuous. These considerations imply that a transition can occur with no

application disruption and no significant client work to update state related to the filesystem.

When the same-fs mask indicates two filesystems are the same the clients are entitled to assume that there will also be no significant delay for the server to re-establish its state to effectively support the client. Where same-fs is not true and the other constituent continuity indication are true (fileid-keep, change-cont, same-fh), there may be significant delay under some circumstances, in line with the fact that the filesystems are being represented as being carefully kept in complete synchronization yet they are not the same.

When two filesystems on separate servers have location ids which match on all the bits within the same-fs mask, clients should present the same nfs_client_id to both with the expectation the servers may be able to generate a shared clientid to be used when communicating with either. Such servers are expected to co-ordinate at least to the degree that they will not provide the same clientid to a client while not actually sharing the underlying state data.

In handling of uncommitted writes, two servers with any pair of filesystems having the same-fs relation, write verifiers must be sufficiently unique that a client switching between the servers can determine whether previous async writes need to be reissued. This is unlike the general case of filesystems not bearing this relation, in which it must be assumed that asynchronous writes will be lost across a filesystem transition.

When two replicas' location ids, match on all the bits within the same-fs mask, but are not identical, the client using sessions will establish separate sessions to each which together share any such common clientid.

- o The `valid_for` field specifies a time for which it is reasonable for a client to use the `fs_location_info` attribute without refetch. The `valid_for` value does not provide a guarantee of validity since servers can unexpectedly go out of service or become inaccessible for any number of reasons. Clients are well-advised to refetch this information for actively accessed filesystem at every `valid_for` seconds. This is particularly important when filesystem replicas may go out of service in a controlled way using the `LIF_GOING` flag to communicate an ongoing change. The server should set `valid_for` to a value which allows well-behaved clients to notice the `LIF_GOING` flag and make an orderly switch before

the loss of service becomes effective. If this value is zero, then no refetch interval is appropriate and the client need not refetch this data on any particular schedule.

In the event of a transition to a new filesystem instance, a new value of the `fs_location_info` attribute will be fetched at the destination and it is to be expected that this may have a different `valid_for` value, which the client should then use, in the same fashion as the previous value.

- o The `read-rank`, `read-order`, `write-rank`, and `write-order` masks are used, together with the priority words of various replicas to order the replicas according to the server's preference. See the discussion below for the interaction of rank, order, and the client's own preferences and needs. `Read-rank` and `read-order` are used to direct clients which only need read access while `write-rank` and `write-order` are used to direct clients that require some degree of write access to the filesystem.

Depending on the potential need for write access by a given client, one of the pairs of rank and order masks is used, together with priority words, to determine a rank and an order for each instance under consideration. The read rank and order should only be used if the client knows that only reading will ever be done or if it is prepared to switch to a different replica in the event that any write access capability is required in the future. The rank is obtained by anding the selected rank mask with the priority and the

order is obtained similarly by anding the selected order mask with the priority. The resulting rank and order are compared as described below with lower always being better (more preferred).

Rank is used to express a strict server-imposed ordering on clients, with lower values indicating "more preferred." Clients should attempt to use all replicas with a given rank before they use one with a higher rank. Only if all of those servers are unavailable should the client proceed to servers of a higher rank.

Within a rank, the order value is used to specify the server's preference to guide the client's selection when the client's own preferences are not controlling, with lower values of order indicating "more preferred." If replicas are approximately equal in all respects, clients should defer to the order specified by the server. When clients look at server latency as part of their selection, they are free to use this criterion but it is suggested that when latency differences are not significant, the server-specified order should guide selection.

The server may configure the rank and order masks to considerably simplify the decisions if it so chooses. For example, if read vs. write is not to be important in the selection process, then the `location4_info` should be one in which the read-rank and write-rank mask, and the read-order and write-order mask are equal. If the server wishes to totally direct the process via rank, leaving no room for client choice, it may simply set the write-order mask and the read-order mask to zero. Conversely, if it wishes to give general preferences with more scope for client choice, it may set the read-rank mask and the write-rank mask to zero. A server may even set all the masks to zero and allow the client to make its own choices. The protocol allows multiple policies to be used as found appropriate.

The use of location id together with the masks in `location4_info` structure can be illustrated by an example.

Suppose one has the following sets of servers:

- o Server A with four IP addresses A1 through A4.

- o Servers B, C, D sharing a cluster filesystem with A and each having four IP addresses, B1, B2, ... D3, D4.
- o A point-in-time copy of the filesystem created using image copy which shares filehandles and is change-attribute continuous with the filesystem on A-D and has two IP address X1 and X2.
- o A point-in-time-copy of the filesystem which was created at a higher level but shares fileid's with the one on A-D but is accessed (via a clustered filesystem) by servers Ya and Yb.
- o A copy of the of the filesystem made by simple user-level copy tools and which is served from server Z.

Given the above, one way of presenting these relationships is to assign the following location id's:

- o A1-4 would get 0x1111
- o B1-4 would get 0x1112
- o C1-4 would get 0x1113
- o D1-4 would get 0x1114

- o X1-2 would get 0x1125
- o Ya would get 0x1236
- o Yb would get 0x1237
- o Z would get 0x2348

And then the following mask values would be used:

- o The same-fs and same-state masks would all be 0xfff0.
- o The same-fh and change-cont mask would be 0xff00.

- o The keep-fileid mask would be 0xf00

This scheme allows the number of bits devoted to various kinds of similarity classes to be adjusted as needed with no change to the protocol. The total of thirty-two bits is expected to suffice indefinitely.

As noted above, the `fs_location_info` attribute, when supported, may be requested of absent filesystems without causing `NFS4ERR_MOVED` to be returned and it is generally expected that will be available for both present and absent filesystems even if only a single `location_server` entry is present, designating the current (present) filesystem, or two `location_server` entries designating the current (and now previous) location of an absent filesystem and its successor location. Servers are strongly urged to support this attribute on all filesystems if they support it on any filesystem.

[3.2.3.](#) `fh_replacement`

The `fh_replacement` attribute provides a way of providing a substitute filehandle to be used on a target server when a migration event or other fs instance switching event occurs. This provides an alternative to maintaining access via the existing persistent filehandle (which may be difficult) or using volatile filehandles (which will not give the correct result in all cases).

When a migration event occurs, information on the new location (or location choices) will be available via the `fs_location_info` attribute applied to any filehandle within the source filesystem. When `LIF_FHR_OPEN` or `LIF_FHR_ALL` is present, the `fh_replacement` attribute may be used to get the corresponding filehandle for filehandles that the client has accessed.

Similarly, after such an event, when the `fs_location_info` attribute is fetched on the new server, `LIF_FHR_OPEN` or `LIF_FHR_ALL` may be present in the server entry corresponding to the current filesystem instance. In this case, the `fh_replacement` attribute can be used to get the new filehandles corresponding to each of the now outdated filehandles on the previous instance. In either of these ways, the client may be assured of a consistent mapping from old to

new filehandles without relying on a purely name-based mapping, which in some cases will not be correct.

The choice of providing replacement on the source filesystem instance or the target will normally be based on which server has the proper mapping. Generally when the image is created by a push from the source, the source server naturally has the appropriate filehandles corresponding to its files and can provide them to the client. When the image transfer is done via pull, the target server will be aware of the source filehandles and can provide the appropriate mapping when the client requests it. Note that the target server can only provide replacement filehandles if it can assure filehandle uniqueness, i.e. that filehandles from the source do not conflict with valid filehandles on the destination server. In the case where such uniqueness can be assured, source filehandles can be accepted for the purpose of providing replacements with NFS4ERR_FHEXPIRED returned for any use other than interrogation of the fh_replacement attribute via GETATTR.

Multiple fh replacement on different migration targets may be provided via multiple fhrep4 entries. Each fhrep4_entry provides a replacement filehandle applying to all targets whose location id, when anded with the fh-same mask (from the fs_location_info attribute) matches the location_set value in the fhrep4_entry. This set of replicas share the same filehandle and thus can a single entry can provide replacement filehandles for all of the members. Note that the location_set value will only match that of the current filesystem instance, when the client presents a filehandle from the previous filesystem instance and the target filesystem provides its own replacement filehandles.

```
union fhrep4_entry switch (bool present) {
    uint32_t      location_set;
    nfs_fh4      replacement;
};

struct fh4_replacement {
    fhrep4_entry  entries<>;
};
```

When a filesystem becomes absent, the server in responding to requests for the fh_replacement attribute is not required to validate all fields of the filehandle if it does not maintain per-file information. This matches current handle of fs_locations (and applies as well to fs_location_info). For example, if a server has an fsid field within its filehandle implementation, it may simply recognize that value and return filehandles with the corresponding new fsid without validating other information within the handle. This can result in filesystem accepting a filehandle, which under other circumstances might result in NFS4ERR_STALE, just as it can when interrogating the fs_locations or fs_location_info attributes. Note that when it does so, it will return a replacement which, when presented to the new filesystem, will get an NFS4ERR_STALE there.

Use of the fh_replacement attribute can allow wholesale change of filehandles to implement storage re-organization even within the context of a single server. If NFS4ERR_MOVED is returned, the client will fetch fs_location_info which may refer to a location on the original server. Use of fh_replacement in this context allows a new set of filehandles to be established as part of storage reconfiguration (including possibly a split into multiple fs's) without requiring the client to maintain name information against the possibility of such a reconfiguration (for volatile filehandles).

Servers are not required to maintain the availability of replacement filehandles for any particular length of time, but in order to maintain continuity of access in the face of network disruptions, servers should generally maintain the mapping from the pre-replacement file handles persistently across server reboots, and for a considerable time. It should be the case that even under severe network disruption, any client that received pre-replacement filehandles is given an opportunity to obtain the replacements. When this mapping no longer made available, the pre-replacement filehandles should not be re-used, just as is the case for any other superseded file handle.

As noted above, this attribute, when supported, may be requested of absent filesystems without causing NFS4ERR_MOVED to be returned, and it should always be available. When it is requested and the attribute is supported, if no replacement file handle information is present, either because the filesystem is still present and there is no migration event or because there are currently no replacement filehandles available, a zero-length array of fhrep4_entry structures should be returned.

[3.2.4.](#) fs_status

In an environment in which multiple copies of the same basic set of data are available, information regarding the particular source of such data and the relationships among different copies, can be very helpful in providing consistent data to applications.

```
enum status4_type {
    STATUS4_FIXED = 1,
    STATUS4_UPDATED = 2,
    STATUS4_INTERLOCKED = 3,
    STATUS4_WRITABLE = 4,
    STATUS4_ABSENT = 5
};

struct fs4_status {
    status4_type    type;
    utf8str_cs      source;
    utf8str_cs      current;
    nfstime4        version;
};
```

The type value indicates the kind of filesystem image represented. This is of particular importance when using the version values to determine appropriate succession of filesystem images. Five types are distinguished:

- o STATUS4_FIXED which indicates a read-only image in the sense that it will never change. The possibility is allowed that as a result of migration or switch to a different image, changed data can be accessed but within the confines of this instance, no change is allowed. The client can use this fact to aggressively cache.
- o STATUS4_UPDATED which indicates an image that cannot be updated by the user writing to it but may be changed exogenously, typically because it is a periodically updated copy of another writable filesystem somewhere else.
- o STATUS4_VERSIONED which indicates that the image, like the STATUS4_UPDATED case, is updated exogenously, but it provides

a guarantee that the server will carefully update the associated version value so that the client, may if it chooses, protect itself from a situation in which it reads data from one version of the filesystem, and then later reads data from an earlier version of the same filesystem. See

below for a discussion of how this can be done.

- o STATUS4_WRITABLE which indicates that the filesystem is an actual writable one. The client need not of course actually write to the filesystem, but once it does, it should not accept a transition to anything other than a writable instance of that same filesystem.
- o STATUS4_ABSENT which indicates that the information is the last valid for a filesystem which is no longer present.

The opaque strings source and current provide a way of presenting information about the source of the filesystem image being present. It is not intended that client do anything with this information other than make it available to administrative tools. It is intended that this information be helpful when researching possible problems with a filesystem image that might arise when it is unclear if the correct image is being accessed and if not, how that image came to be made. This kind of debugging information will be helpful, if, as seems likely, copies of filesystems are made in many different ways (e.g. simple user-level copies, filesystem-level point-in-time copies, cloning of the underlying storage), under a variety of administrative arrangements. In such environments, determining how a given set of data was constructed can be very helpful in resolving problems.

The opaque string 'source' is used to indicate the source of a given filesystem with the expectation that tools capable of creating a filesystem image propagate this information, when that is possible. It is understood that this may not always be possible since a user-level copy may be thought of as creating a new data set and the tools used may have no mechanism to propagate this data. When a filesystem is initially created associating with it data regarding how the filesystem was created, where it was created, by whom, etc. can be put in this attribute in a human-readable string form so that it will be available when propagated

to subsequent copies of this data.

The opaque string 'current' should provide whatever information is available about the source of the current copy. Such information as the tool creating it, any relevant parameters to that tool, the time at which the copy was done, the user making the change, the server on which the change was made etc. All information should be in a human-readable string form.

The version field provides a version identification, in the form of a time value, such that successive versions always have later time values. When the filesystem type is anything other than

STATUS4_VERSIONED, the server may provide such a value but there is no guarantee as to its validity and clients will not use it except to provide additional information to add to 'source' and 'current'.

When the type is STATUS4_VERSIONED, servers should provide a value of version which progresses monotonically whenever any new version of the data is established. This allows the client, if reliable image progression is important to it, to fetch this attribute as part of each COMPOUND where data or metadata from the filesystem is used.

When it is important to the client to make sure that only valid successor images are accepted, it must make sure that it does not read data or metadata from the filesystem without updating its sense of the current state of the image, to avoid the possibility that the fs_status which the client holds will be one for an earlier image, and so accept a new filesystem instance which is later than that but still earlier than updated data read by the client.

In order to do this reliably, it must do a GETATTR of fs_status that follows any interrogation of data or metadata within the filesystem in question. Often this is most conveniently done by appending such a GETATTR after all other operations that reference a given filesystem. When errors occur between reading filesystem data and performing such a GETATTR, care must be exercised to make sure that the data in question is not used before obtaining the proper fs_status value. In this connection, when an OPEN is done within such a versioned filesystem and the associated GETATTR of

fs_status is not successfully completed, the open file in question must not be accessed until that fs_status is fetched.

The procedure above will ensure that before using any data from the filesystem the client has in hand a newly-fetched current version of the filesystem image. Multiple values for multiple requests in flight can be resolved by assembling them into the required partial order (and the elements should form a total order within it) and using the last. The client may then, when switching among filesystem instances, decline to use an instance which is not of type STATUS4_VERSIONED or whose version field is earlier than the last one obtained from the predecessor filesystem instance.

4. Migration Protocol

As discussed above, it has always been anticipated that a migration protocol would be developed, to address the issue of migration of a filesystem between different filesystem implementations. This need remains, and it can be expected that as client implementations of

migration become more common, it will become more pressing and the working group needs to seriously consider how that need may be best addressed.

We are going to suggest that the working group should seriously consider what may be a significantly lighter-weight alternative, the addition of features to support server-to-server migration within NFSv4 itself, but taking advantage of existing NFSv4 facilities and only adding the features needed to support efficient migration, as items within a minor version.

One thing that needs to be made clear is that a common migration protocol does not mean a common migration approach or common migration functionality. Thus the need for the kinds of information provided by fs_location_info. For example, the fact that the migration protocol will make available on the target the file id, file handle, and change attribute from the source, does not mean that the receiving can store these values natively, or that it will choose to implement translation support to accommodate the values exported by the source. This will remain an implementation choice. Clients will need information about those various choices, such as would be provided by fs_location_info, in

order to deal with the various implementations.

4.1. NFSv4.x as a Migration Protocol

Whether the following approach or any other is adopted, considerable work will still be required to flesh out the details, requiring a number of drafts for a problem statement, initial protocol spec, etc. But to give an idea of what would be involved in this kind of approach, a rough sketch is given below.

First, let us fix for the moment on a pull model, in which the target server, selected by a management application pulls data from the source using NFSv4.x. The server acts as a client, albeit a specially privileged one, to copy the existing data.

The first point to be made is that using NFSv4 means that we have a representation for all data that is representable within NFSv4 and that that is maintained automatically as minor versioning proceeds. That is, when attributes are added to a minor version of NFSv4, they are "automatically" added to the migration copy protocol, because the two are the same.

The presence of COMPOUND is a further help in that implementations will be able to maintain high throughput when copying without creating a special protocol devoted to that purpose. For example, when copying a large set of small files, these files can all be

read with a single COMPOUND. This means that the benefit of creating a stream format for the entire fs is much reduced and allows existing servers (with small modifications) to simply support the kinds of access they have to support anyway. The servers acting as clients would probably use a non-standard implementation but they would share lots of infrastructure with more standard clients, so this would probably be a win on the implementation side as well as on the specification side.

One other point is that if the migration protocol were in fact an NFSv4.x, NFSv4 developments such as pNFS would be available for high-performance migration, with no special effort.

Clearly, there is still considerable work to do this, even if it is not of the same order as a new protocol. The working group needs

to discuss this and see if there is agreement that a means of cross-server migration is worthwhile and whether this is the best way to get there.

Here is a basic list of things that would have to be dealt with to effect a transfer:

- o Reads without changing access times. This is probably best done as a per-session attribute (it is best to assume sessions here).
- o Reads that ignore share reservations and mandatory locks. It may be that the existing all-ones special stateid is adequate.
- o A way to obtain the locking state information for the source fs: the locks (byte-range and share reservations) for that fs including associated stateids and owner opaque strings, clientid's and the other identifying client information for all clients with locks on that fs. This is all protocol-defined, rather than implementation-specific data.
- o A way to lock out changes on a filesystem. This would be similar to a read delegation on the entire filesystem, but would have a greater degree of privilege, in that the holder would be allowed to keep it as long as his lease was renewed.
- o A way to permanently terminate existing access to the filesystem (by everyone except the calling session) and report it MOVED to the users.

Conventions as far as appropriate security for such operations would have to be developed to assure interoperability, but it is a

question of establishing conventions rather than defining new mechanisms.

Given the facilities above, you could get an initial image of a filesystem, and then rescan and update the destination until the amount of change to be propagated stabilized. At this point, changes could be locked out and a final set up updates propagated while read-only access to the filesystem continued. At that point

further access would be locked out, and the locking state and any final changes to access time would be propagated. The access time scan would be manageable since the client could issue long COMPOUND's with many PUTFH-GETATTR pairs and many such requests could be in flight at a time.

If it was required that the disruption to access be smaller, some small additions to the functionality might be quite effective:

- o Notifications for a filesystem, perhaps building on the notifications proposed in the directory delegations document would limit the rescanning for changes, and so would make the window in which additional changes could happen much smaller. This would greatly reduce the window in which write access would have to be locked out.
- o A facility for global scans for attribute changes could help reduce lockout periods. Something that gave a list of object filehandles that met a given attribute search criterion (e.g. attribute x greater than, less than, equal to, some value) could reduce rescan update times and also rescan times for accesstime updates.

These lists assume that the server initiating the transfer is doing its own writing to disk. Extending this to writing the new fs via NFSv4 would require further protocol support. The basic message for the working group is that the set of things to do is of moderate size and builds in large part on existing or already proposed facilities.

Acknowledgements

The authors wish to thank Ted Anderson and Jon Haswell for their contributions to the ideas within this document.

[RFC3530]

S. Shepler, et. al., "NFS Version 4 Protocol", Standards Track RFC

Informative References

[referrals]

D. Noveck, R. Burnett, "Implementation Guide for Referrals in NFSv4", Internet Draft [draft-ietf-nfsv4-referrals-00.txt](#), Work in progress

Authors' Addresses

David Noveck
Network Appliance, Inc.
375 Totten Pond Road
Waltham, MA 02451 USA

Phone: +1 781 768 5347
EMail: dnoveck@netapp.com

Rodney C. Burnett
IBM, Inc.
13001 Trailwood Rd
Austin, TX 78727 USA

Phone: +1 512 838 8498
EMail: cburnett@us.ibm.com

Full Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

