INTERNET-DRAFT
Expires: January 2006

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

July 2005

Implementation Guide for Referrals in NFSv4
draft-ietf-nfsv4-referrals-00.txt

Status of this Memo

Copyright Notice

Abstract

RFC3530 describes a migration feature using the NFS4ERR_MOVED error code and the fs_locations attribute. The description focuses on the case of migration (i.e. relocation) of a file system already known to the client. The simpler limiting case in a which a file system not previously known to the client was located elsewhere, which we here call a referral, was not clearly described. Because of this and also because of some inconsistencies and ambiguities in the text of RFC3530, there has been confusion about how the client and server should interact in performing a referral. This document provides a guide to the implementation of referrals, and in so doing, addresses the relevant problems in RFC3530.

Table Of Contents

1. Introduction

RFC3530 describes a migration feature using the NFS4ERR_MOVED error code and the fs_locations attribute. The description focuses on the case of migration of a file system already known to the client, while the alternate case in a which a file system not previously known to the client was located elsewhere, which we here call a referral, was not clearly described.

Analysis of how this case would be dealt with has shown that the protocol in RFC3530 is capable of dealing properly with this sub-case. However, there are difficulties for implementers in determining how precisely this is to be done, because the text of RFC3530 does not address this case. In addition, there are a number of ambiguities and inconsistencies in the text, that increase the difficulties for implementers and that raise the probability that clients and servers may not properly interact.

This document is an attempt to deal with these spec issues and to provide a clear guide to follow in developing inter-operable referral implementations.

This document may deal with some aspects of issues related to the migration case where the data actually relocated from one server to another while under client access. For example, clarification of apparent contradictions in the spec may relate to both referrals and the general migration case. However, the focus of this document is on the case of referrals and other aspects of migration will only be dealt with as necessary to accomplish that main purpose.

1.1. Referrals as a Limiting Case of Migrations

If a server implements file system migration, individual clients, not being synchronized with the migration event, will encounter different situations from the client point of view. That is, even though some clients will have received filehandles within the migrated filesystem, others may at that time be first referencing the root of the migrated filesystem and need to be redirected to the new fs location, just as those clients that have already accessed the filesystem (when it was present) need to be redirected.

Thus referrals are a limiting sub-case of migration and when a migration event occurs some clients will see an ordinary migration in the case in which access (by that client) to the filesystem being moved has already occurred, while others will see a referral when they first attempt to access the absent filesystem.

1.2. Pure Referrals

Given that clients supporting migration need to be prepared for a migration event, a server can establish a situation, which we refer to as a pure referral, in which all clients see a referral. Since no client can enter the absent filesystem, the presumptive migration of the absent filesystem, assumes a purely conventional character, in that for each client it appears to have happened

before all access by that client. This allows the server to establish a situation in which referral to an absent filesystem may occur for filesystems that were never present on the server.

This functionality may be used to allow construction of multi-server namespaces and may serve as a building block for the construction of wider global namespaces.

2. Issues as to When NFS4ERR_MOVED May/Should be Returned

RFC3530 contains the following statement in section 6.2: "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. PUTFH and GETATTR, while the statement above suggests otherwise, have NFS4ERR_MOVED listed as error codes, whereas there are six other ops (PUTROOTFH, PUTPUBFH, RENEW, SETCLIENTID, SETCLIENTID_CONFIRM, RELEASE_OWNER) which are not allowed to return NFS4ERR_MOVED despite the "all operations except" clause given above.

The individual ops will be discussed below but in order to understand how these contradictions arose and how they are to be dealt with, it is important to be clear on what point the current filesystem (i.e. the filesystem associated with the current filehandle) is to be tested for absence in deciding whether to return NFS4ERR_MOVED. For operations which change the current filehandle, (i.e. PUTFH, PUTROOTFH, PUTPUBFH, RESTOREFH, LOOKUP, LOOKUPP), this distinction is critical. The spec is, however, not very clear on this issue so we need to consider the alternatives.

Let's first consider a check on the current filehandle after the operation. In part, that seems dubious because it returns an error for an operation where the circumstances that give rise to the error (i.e. changing the filehandle) depend on the successful execution of the operation giving rise to the error. However, the main reason such an interpretation is insupportable is that it would make some situations, wherein fs_locations is needed, impossible to deal with. For example, LOOKUP followed by GETATTR(fs_locations) would not be possible if the LOOKUP returned NFS4ERR_MOVED when the current filehandle at the start of the LOOKUP was in a present filesystem while the current filehandle at the end of the LOOKUP was at the root of an absent filesystem. An NFS4ERR_MOVED returned by LOOKUP would not allow execution of the GETATTR, any exception for GETATTR of fs_locations notwithstanding.

On the other hand, making the check at the start of each operation (except some GETATTR's) allows fs_locations to be determined whenever necessary and is the best way to resolve the issue.

Except where RFC3530 has made this impossible by not listing NFS4ERR_MOVED as an error code for certain ops, each op should check at the beginning of the operation for a current filehandle within an absent fs. If so, NFS4ERR_MOVED is returned. A partial exception is GETATTR for which the return of NFS4ERR_MOVED (or not) is dependent on the set of attributes requested. This is discussed further below.

2.1. PUTFH Returning NFS4ERR_MOVED

As noted above, section 6.2 indicates that PUTFH will not return NFS4ERR_MOVED while the detailed description for PUTFH (section 14.2.20), lists NFS4ERR_MOVED as a valid return. How are we to resolve the contradiction?

It would seem that the exception for PUTFH in section 6.2, is unnecessary and derives from a confusion about the logic of NFS4ERR_MOVED. Clearly the sequence of PUTFH(fh within absent filesystem) followed by GETATTR(fs_locations) must be allowed but it requires no specific exception for PUTFH to do so, since only GETATTR, and not PUTFH, has a filehandle within an absent filesystem as the current handle at the start of the operation. Thus no exception for PUTFH is required, as is recognized by section 14.2.20.

PUTFH will never return NFS4ERR_MOVED because the filehandle it is establishing belongs to an absent filesystem. This is because the current filehandle/filesystem is checked for absence at the start of each op and the new filehandle for PUTFH has not been established at the start of the PUTFH. This allows the sequence PUTFH-GETATTR(fs_locations) to proceed without any specific exception for PUTFH or any analysis of the op stream that encompasses multiple ops (e.g. looking ahead to the GETATTR).

NFS4ERR_MOVED can happen as a result of a PUTFH, but only in rather odd circumstances such as the following:

o   PUTFH (fh within absent filesystem)

o   PUTFH (fh within present filesystem)

In this case, NFS4ERR_MOVED would be returned on the second PUTFH since the current filehandle at the start of that op is within an absent filesystem. It may seem odd that the PUTFH which is establishing a current filehandle within a present filesystem should get an error while the one that established a current filehandle within an absent filesystem does not but this behavior is a consequence of a uniform set of rules for NFS4ERR_MOVED, the

basis of which is that the current filehandle is tested at the start of an operation.

2.2. GETFH Returning NFS4ERR_MOVED

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.

In fact, GETFH will result in NFS4ERR_MOVED being returned if the current file handle is for an absent filesystem. This is particularly helpful in dealing with pure referral situations since a filehandle that a client does not ever see can pose no difficulties. For example, it is irrelevant whether the target filesystem has persistent or volatile filehandles. An inherently unobserved filehandle poses no expiration difficulties regardless of whether the target filesystem (on the target server) has persistent or volatile filehandles. Servers should never return filehandles within absent filesystems for pure referral cases.

2.3. GETATTR Returning NFS4ERR_MOVED

As noted above, Section 6.2 indicates that NFS4ERR_MOVED is not returned for a GETATTR operation, but NFS4ERR_MOVED is listed as an error that can be returned by GETATTR (in section 14.2.7). Since the purpose of the exception for GETATTR is to allow fs_locations to be fetched, the best resolution of this contradiction is for NFS4ERR_MOVED to be returned by GETATTR's that ask for some unavailable attribute and that do not interrogate the fs_locations attribute. This maintains the exception which allows GETATTR to be used to get fs_locations information by establishing the rule that GETATTR's which interrogate fs_locations (with or without additional attributes) will not return NFS4ERR_MOVED. Neither will GETATTR's that only request attributes that are available. These considerations are further discussed in section 3.

Sometimes clients need a quickly checked indication of whether a filesystem is absent or not. In most cases, because fs_locations is not requested, this indication will be the NFS4ERR_MOVED being returned by the GETATTR. When fs_locations is requested, a convenient way to provide this indication is to request the filehandle attribute. Since this attribute will not be provided for referrals (see section 3.4.2), examining the mask of returned attributes for the presence of the filehandle attribute is quick way to determine whether the directory in question is part of an absent filesystem.

2.4. Ops Not Allowed to Return NFS4ERR_MOVED

As noted above, despite the fact that section 6.2 suggests otherwise, a number of ops, all of which do not require a current filehandle, are not listed as returning NFS4ERR_MOVED. These ops are PUTROOTFH, PUTPUBFH, RENEW, SETCLIENTID, and RELEASE_LOCKOWNER. Note also that DELEGPURGE, another op which does not require a current filehandle, is listed as able to return NFS4ERR_MOVED.

Although this contradiction could be resolved by changing the op description to match 6.2, it does not seem that there is any strong reason to do so. The protocol will function adequately, if these are treated as exceptions, and NFS4ERR_MOVED is returned by the next op allowed to do so, as long as the current filehandle at the start of that op is one within an absent fs. Note that PUTROOTFH and PUTPUBFH could cause a pending absent fs indication not to result in an error code, since they change the current filehandle to be within a new, presumably present, filesystem. However, in this case the client would not have issued any operations prior to the PUTROOTFH or PUTPUBFH that actually operated on the absent fs.

3. Attribute Issues

RFC3530 anticipates that clients will request additional attributes beyond fs_locations (see section 6.2 of that document) and allows the server to return fs_locations only. The return of other attributes, implicit in the statement "server may return fs_locations only" is not clearly dealt with. While some of these, such as fsid, are important for all forms of migration, this omission is most critical in the case of referrals. There are a number of attributes besides fs_locations that need to be provided at the root of an absent filesystem for server and client to properly collaborate to perform a referral.

There are also a large number of other attributes which the server should not provide for absent filesystem since providing them is the province of the referral's target server. Providing possibly conflicting values on the server that is the source of the referral can only contribute to confusion.

3.1. Number of fs_locations

It should be noted that with referrals, it is valid for an fs_locations response to describe multiple locations if the target of the referral is a replicated filesystem. This allows the client to find one of the replicas when one or more of the supplied locations are unavailable. Once the client establishes contact with

a target server, it will then obtain fs_locations as appropriate from the server where the referral target data resides.

Although it is valid for fs_locations to specify more than one location in the case of referral, it must specify at least one. This is in contrast to the case in which fs_locations is fetched for a filesystem which is present in order to determine replica locations in the case of failure. In that case, it is valid for a server to return an empty fs_locations attribute, since the client already has a valid location for the server and is only requesting additional locations and when there aren't any, an empty fs_locations gives him exactly the information being sought. When handling the returned attribute, the client should consider the replying server as the implied last entry in the list when it is not present in the returned fs_locations.

In the referral case, the client is getting NFS4ERR_MOVED and has no current valid location for the filesystem and thus at least one location is required. In other cases, although a server may include his own location for the filesystem in the fs_locations attribute, he is not required to do so.

When the server validly returns an fs_locations which has a null array of locations, it may choose to return an empty (zero component) fs_root since the only purpose of fs_root is to aid in the interpretation of the associated locations in the fs_locations attribute.

3.2. General 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.

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 in 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."

The above is a helpful restriction in that it allows clients to simplify their attribute interpretation code. 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. However, 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 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 some filesystems that do support ACLs and some that don't support then, 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.

The text in section 14.2.7 would create great obstacles in the implementation of migration given section 6.2. In order to get the additional attributes mentioned, the client would have to guess at the set to be provided and, if that guess were not valid, get an error, presumably NFS4ERR_MOVED, returned. For the purposes of this document, we will treat the specific statement regarding fs_locations in section 6.2 as controlling. The contradictory text in section 14.2.7 will be treated as an overgeneralization to which an exception, for the case of migration and referrals, must be understood.

When fs_locations is requested, then a subset of the requested attributes may be provided by the server without generating the error NFS4ERR_MOVED. The subset returned must always include fs_locations. In addition, if all of the attributes requested can be provided, then NFS4ERR_MOVED will also not be returned but in this case, a subset of the requested attributes may not be provided. The following sections give guidance on the attributes the server should provide for filehandles within an absent filesystem.

A related issue concerns attributes in a READDIR. There has been discussion, not yet fully resolved, on the working group list about whether all the attributes specified in the attribute mask for READDIR must be provided for all readdir entries when those attributes are supported by the associated filesystems. It can be argued that that would be implied if one combined the words of section 14.2.7 with text that (even though somewhat more loosely) implies that the attributes in READDIR are to be provided via GETATTR. It has also been argued that the role of READDIR with attributes as a replacement for v3 READDIRPLUS makes it more appropriate to treat the attribute mask parameters as a hint with the server allowed to omit any attributes requested (possibly with some exceptions) whenever inconvenient. In the discussion, there have also been occasional suggestions that an attribute's status (as mandatory or recommended) should have role in deciding whether servers are obliged to provide them when requested for READDIR.

Regardless of how this debate is to be resolved in the general case, it is clear in the case of an absent filesystem, if there is a general obligation to provide all requested attributes, that an exception must be made for directory entries that represent the root of an absent fs (a referral). For these entries, the full set of requested attributes may simply not be available, as they likewise would not be available for GETATTR.

Further, we will assume in our examples, that for the few attributes necessary to perform the referral (see below for details), the server will always provide these when requested as it should be easy for the server to do.

So in this document, we will assume that regardless of the general resolution of this issue, in the case of an absent filesystem some flexibility must be provided, even if it is determined that making the attribute mask for READDIR a hint in general is not justified.

So we will assume that for READDIR:

o   When fs_locations is among the attributes requested, the server may provide a subset of the other requested attributes together with fs_locations for roots of absent fs's, without causing any error for the READDIR as a whole. If rdattr_error is also requested and there are attributes which are not available, then rdattr_error will receive the value NFS4ERR_MOVED.

o   When fs_locations is not 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.

- When fs_locations is not requested, but rdattr_error is and some attributes requested are not available because of the absence of the filesystem, the server will return NFS4ERR_MOVED for the rdattr_error attribute and, in addition, the requested attributes that are valid for the root of an absent filesystem.

- When neither fs_locations nor rdattr_error is requested and there is a directory within an absent fs within the directory being read, if some unavailable attributes are requested, no data will be returned and the READDIR will get an NFS4ERR_MOVED error. (The examples will suppose this but clients will not depend on this behavior and clients should work with a more liberal interpretation if this is decided on.

3.3. Attributes Needed for Root of an Absent Fs

The following options need to be provided for referrals. While it is true that some of these are not mandatory in NFSv4, and the spec treats providing others as optional in the case of an absent filesystem, they do need to be provided to do a referral as envisioned herein.

3.3.1. fsid

The fsid attribute allows clients to recognize when fs boundaries have been crossed. This applies also when one crosses into an absent filesystem and servers should provide this information when fs_locations is being requested for a filehandle within an absent filesystem, most typically at the root of that absent filesystem.

To avoid misunderstanding, it should be noted 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 referral or 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.

A key reason for always providing an fsid can be seen by considering client behavior. When a client gets the error NFS4ERR_MOVED, it has a number of key steps to get through. It must determine if it has previously accessed the filesystem on the

server that the moved error involves. If so, then it represents a data migration event in the strict sense, rather than a referral. The client must recover state and properly use rootpath data from the old and new server to graft the new server's namespace into the client's local space. If a client gets an NFS4ERR_MOVED error and it cannot get an fsid, then it must use fs_locations, fs_root, and rootpath data to locally determine if it has been accessing the filesystem or if the error applies to an fh in a different. (absent) filesystem. If it uses fs_root and there have been renames along the path to the fh, then the client will not be able to distinguish a migration from a referral. The best solution here is for the client to always get an fsid. All a server has to do is generate an fsid value that cannot represent actual exported data at the server. We want to have a model where referrals can still function even if renames or other events change the path to the referral point. Always providing fsid's improves the usefulness of the NFS namespace since referral and migration events can continue to be handled in the face of namespace management. This also helps to reduce client complexity.

3.3.2. 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, need 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 arise in the case of mounted_on_fileid since that fileid is in the context of the mounted-on fs and unique within it. Servers need to support and always return valid data for mounted-on-fileid, even for the case of an absent filesystem. The returned data represents a fileid for the entry in the directory that represents the absent fs. It's not the fileid of the root of the absent fs itself. Per the attribute handling rules presented in section 3.1, the server may return mounted-on-fileid as a subset of the requested attributes. The presence of fs_locations and rdattr_error attributes in a READDIR request determines if and how the server sets rdattr_error in the partial return case.

Clients are strongly encouraged to always include mounted-on-fileid and rdattr_error in READDIR requests. This increases the potential that the client will receive information needed to satisfy the request and avoid further READDIR requests with a more restricted set of attributes. As a result, client performance is improved and network traffic is reduced.

3.3.3. rdattr_error attribute

This attribute needs to be provided for READDIR's or else a READDIR of a directory that contains the root of absent fs's cannot be done effectively. In order to avoid NFS4ERR_MOVED on the directory as a whole, providing this error via the rdattr_error attribute allows attributes for the entry in question, which can be provided to the client, as well allowing the READDIR to provide information about other entries within the directory.

3.4. Attributes Valid for Root of an Absent Fs

Beyond the essential attributes discussed above, there are several other attributes that a server could decide to return for an absent fs. These should pose little potential for inconsistencies or client side problems. They are mentioned below, but clients should not expect servers to return them.

3.4.1. type attribute

For the root of absent fs, providing the value NF4DIR poses no difficulties as the target will provide the same value at the root of the target filesystem.

Such a value isn't really needed as clients can assume that when there is a change of fsid value, the root of new filesystem will be a directory, as it must be.

3.4.2. change attribute

When a server implements pure referrals for a large number of filesystems, it may be convenient for clients to cache the destination fs_locations and interrogate the change attribute to see if there has been any change in the locations attribute to require a refetch. Therefore, if the server does return a value for the change attribute when fetched for an absent filesystem, the server must update the change attribute when the target of the referral, i.e. the fs_locations value, changes.

If the server does provide this information, the client must understand that there is no necessary continuity between the change values on referring server and on the target, as there may not be in the general migration case. Clients are best advised to refetch such values on the target server and assume that it is possible that a change to the object may have occurred before the fetch of the change attribute on the target, and that this fact means that data and attribute caches on he client are best flushed.

3.5. Attributes Not Valid for Root of an Absent Fs

Attributes not listed in the sections above should not be provided in the case of the root of an absent filesystem (the referral case). The general principle is that such information is appropriately provided at the destination specified by the fs_locations attribute and that specifying a value at the point of referral will either be incorrect or superfluous.

We present a few examples of this principle below for particular attributes but the same sort of logic applies to all of the other attributes, to some degree or other.

3.5.1. supp_attr attribute

Since the referring server has no basis to determine what attributes are supported on the target, it is best not to provide this attribute on the referring server.

It could be argued a value limited to attributes actually provided by the referring server could be provided, with the clear understanding that the value on the target will be different. However, there is very little value in doing that since there is only a single accessible object on the referring server for each fs, and the supported attributes are simply those that the server returns together with fs_locations when requested.

3.5.2. filehandle attribute

Just as a filehandle for the absent fs cannot be provided to the client via GETFH, it should similarly not be provided as the filehandle attribute and for the same reasons.

A filehandle provided for an absent filesystem poses the difficulty of possible expiration or remapping. This is an issue when migration happens after the file system has been accessed, but in the pure referral case, this issue can and should be avoided by never allowing the client to see such a filehandle at all.

3.5.3. fh_expire_type attribute

In the pure referral case, the issue of filehandle expiration type is rendered moot by the lack of visibility of filehandles within the absent filesystem. Providing a value for this attribute would limit the value on the target server to no purpose.

By not providing a value, the referring server allows the target server flexibility to choose the value without fear of confusing the client.

Note that the same logic applies to such attributes as link_support, symlink_support, case_insensitive, case_preserving, chown_restricted, and homogeneous. The referring server should support any choice of such values on the target and the client should have no interest in guesses on the part of the server.

3.5.4. fileid attribute

Specifying a value for the root of the absent filesystem would serve no purpose as the value at the target would have to be definitive and any value at the referring server might conflict with a value at the target server.

4. Referral Examples

The details of how referrals proceed are implicit in the specification of migration in RFC 3530. However, because the details of handling of this case are so different from those in the cases discussed therein, examples tailored to the referral situation are needed to clarify matters and allow correct and consistent implementations.

The examples given in the sections below are somewhat artificial in that an actual client will not typically do a multi-component lookup, but will have cached information regarding the upper levels of the name hierarchy. However, these example are chosen to make the required behavior clear and easy to put within the scope of a small number of requests, without getting unduly into details of how specific clients might choose to cache things.

4.1. Referral Example (LOOKUP)

Let us suppose that the following COMPOUND is issued in an environment in which /src/linux/2.7/latest is absent from the target server. This may be for a number of reasons. It may be the case that the file system has moved, or, it may be the case that

the target server is functioning mainly, or solely, to refer clients to the servers on which various file systems are located.

- PUTROOTFH

- LOOKUP "src"

- LOOKUP "linux"

- LOOKUP "2.7"

- LOOKUP "latest"

- GETFH

- GETATTR fsid,fileid,size,ctime

Under the given circumstances, the following will be the result.

- PUTROOTFH --> NFS_OK

  Current fh is root of pseudo-fs.

- LOOKUP "src" --> NFS_OK

  Current fh is for /src and is within pseudo-fs.

- LOOKUP "linux" --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- LOOKUP "2.7" --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- LOOKUP "latest" --> NFS_OK

  Current fh is for /src/linux/2.7/latest and is within a new, absent fs, but ...

  The client will never see the value of that fh.

- GETFH --> NFS4ERR_MOVED

  Fails because current fh is in an absent fs at the start of the operation and the spec makes no exception for GETFH.

o   GETATTR fsid,fileid,size,ctime

Not executed because the failure of the GETFH stops processing of the COMPOUND.

Given the failure of the GETFH, the client has the job of determining the root of the absent file system and where to find that file system, i.e. the server and path relative to that server's root fh. Note here that in this example, the client did not obtain filehandles and attribute information (e.g. fsid) for the intermediate directories, so that he would not be sure where the absent file system starts. It could be the case, for example, that /src/linux/2.7 is the root of the moved filesystem and that the reason that the lookup of "latest" succeeded is that the filesystem was not absent on that op but was moved between the last LOOKUP and the GETFH (since COMPOUND is not atomic). Even if we had the fsid's for all of the intermediate directories, we could have no way of knowing that /src/linux/2.7/latest was the root of a new fs, since we don't yet have its fsid.

In order to get the necessary information, let us re-issue the chain of lookup's with GETFH's and GETATTR's to at least get the fsid's so we can be sure where the appropriate fs boundaries are. The client could choose to get fs_locations at the same time but in most cases the client will have a good guess as to where fs boundaries are (because of where NFS4ERR_MOVED was gotten and where not) making fetching of fs_location info unnecessary.

o   PUTROOTFH --> NFS_OK

Current fh is root of pseudo-fs.

o   GETATTR(fsid) --> NFS_OK

Just for completeness. Normally, clients will know the fsid of the pseudo-fs as soon as they establish communication with a server.

o   LOOKUP "src" --> NFS_OK

o   GETATTR(fsid) --> NFS_OK

Get current fsid to see where fs boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

o   GETFH --> NFS_OK

Current fh is for /src and is within pseudo-fs.

- LOOKUP "linux" --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- GETATTR(fsid) --> NFS_OK

  Get current fsid to see where fs boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

- GETFH --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- LOOKUP "2.7" --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- GETATTR(fsid) --> NFS_OK

  Get current fsid to see where fs boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

- GETFH --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- LOOKUP "latest" --> NFS_OK

  Current fh is for /src/linux/2.7/latest and is within a new, absent fs, but ...

  The client will never see the value of that fh

- GETATTR(fsid, fs_locations) --> NFS_OK

  We are getting the fsid to know where the fs boundaries are. While RFC 3530 does not oblige the server to give us anything but fs_locations, we are assuming that the server is following the recommendations herein and providing it. Note that the fsid we are given will not necessarily be preserved at the new location. That fsid might be different and in fact the fsid we have for this fs might a valid fsid of a different fs on that new server.

In this particular case, we are pretty sure anyway that what has moved is /src/linux/2.7/latest rather than /src/linux/2.7 since we have the fsid of the latter and it is that of the pseudo-fs, which presumably cannot move. However, in other examples, we might not have this kind of information to rely on (e.g. /src/linux/2.7 might be a non-pseudo filesystem separate from /src/linux/2.7/latest), so we need to have another reliable source information on the boundary of the fs which is moved. If, for example, the filesystem "/src/linux" had moved we would have a case of migration rather than referral and once the boundaries of the migrated filesystem was clear we could fetch fs_locations.

We are fetching fs_location because the fact that we got an NFS4ERR_MOVED at this point means that it most likely that this is a referral and we need the destination. Even if it is the case that "/src/linux/2.7" is a filesystem which has migrated, we will still need the location information for that file system.

- GETFH --> NFS4ERR_MOVED

  Fails because current fh is in an absent fs at the start of the operation and the spec makes no exception for GETFH. Note that this has the happy consequence that we don't have to worry about the volatility or lack thereof of the fh. If the root of the fs on the new location is a persistent fh, then we can assume that this fh, which we never saw is a persistent fh, which, if we could see it, would exactly match the new fh. At least, there is no evidence to disprove that. On the other hand, if we find a volatile root at the new location, then the filehandle which we never saw must have been volatile or at least nobody can prove otherwise.

Given the above, the client knows where the root of the absent file system is, by noting where the change of fsid occurred. The fs_locations attribute also gives the client the actual location of the absent file system, so that the referral can proceed. The server gives the client the bare minimum of information about the absent file system so that there will be very little scope for problems of conflict between information sent by the referring server and information of the file system's home. No filehandles and very few attributes are present on the referring server and the client can treat those it receives as basically transient information with the function of enabling the referral.

4.2. Referral Example (READDIR)

Another context in which a client may encounter referrals is when it does a READDIR on directory in which some of the sub-directories are the roots of absent file systems.

Suppose such a directory is read as follows:

- PUTROOTFH

- LOOKUP "src"

- LOOKUP "linux"

- LOOKUP "2.7"

- READDIR (fsid, size, ctime, mounted_on_fileid)

In this case, because rdattr_error is not requested, fs_locations is not requested, and some of attributes cannot be provided the result will be an NFS4ERR_MOVED error on the READDIR, with the detailed results as follows:

- PUTROOTFH --> NFS_OK

  Current fh is root of pseudo-fs.

- LOOKUP "src" --> NFS_OK

  Current fh is for /src and is within pseudo-fs.

- LOOKUP "linux" --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- LOOKUP "2.7" --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- READDIR (fsid, size, ctime, mounted_on_fileid) --> NFS4ERR_MOVED

  Note that the same error would have been returned if /src/linux/2.7 had migrated, when in fact it is because the directory contains the root of an absent fs.

So now suppose that we reissue with rdattr_error:

- PUTROOTFH
- LOOKUP "src"
- LOOKUP "linux"
- LOOKUP "2.7"
- READDIR (rdattr_error, fsid, size, ctime, mounted_on_fileid)

The results will be:

- PUTROOTFH --> NFS_OK

  Current fh is root of pseudo-fs.

- LOOKUP "src" --> NFS_OK

  Current fh is for /src and is within pseudo-fs.

- LOOKUP "linux" --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- LOOKUP "2.7" --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- READDIR (rdattr_error, fsid, size, ctime, mounted_on_fileid) --> NFS_OK

  The attributes for "latest" will only contain rdattr_error with the value will be NFS4ERR_MOVED, together with an fsid value and an a value for mounted_on_fileid.

So suppose we do another READDIR to get fs_locations info, although we could have used a GETATTR directly, as in the previous section.

- PUTROOTFH
- LOOKUP "src"
- LOOKUP "linux"
- LOOKUP "2.7"
- READDIR (rdattr_error, fs_locations, mounted_on_fileid, fsid, size, ctime)

The results would be:

- PUTROOTFH --> NFS_OK

  Current fh is root of pseudo-fs.

- LOOKUP "src" --> NFS_OK

  Current fh is for /src and is within pseudo-fs.

- LOOKUP "linux" --> NFS_OK

  Current fh is for /src/linux and is within pseudo-fs.

- LOOKUP "2.7" --> NFS_OK

  Current fh is for /src/linux/2.7 and is within pseudo-fs.

- READDIR (rdattr_error, fs_locations, mounted_on_fileid, fsid, size, ctime) --> NFS_OK

  The attributes for "latest" will only contain

  + rdattr_error (value: NFS4ERR_MOVED)

  + fs_locations (value: target:/path/on/target)

  + mounted_on_fileid (value: unique fileid within referring fs)

  + fsid (value: unique value within referring server)

  The attribute entry for "latest" will not contain size or ctime.

5. Additional Client-side Considerations

When clients make use of servers that implement referrals and migration, care should be taken so that a user who mounts a given filesystem that includes a referral or a relocated filesystem continue to see a coherent picture of that user-side filesystem despite the fact that it contains a number of server-side filesystems which may be on different servers.

One important issue is upward navigation from the root of a server-side filesystem to its parent (specified as ".." in UNIX). The client needs to determine when it hits an fsid root going up the filetree. When at such a point, and needs to ascend to the parent,

it must do so locally instead of sending a LOOKUPP call to the server. The LOOKUPP would normally return the ancestor of the target filesystem on the target server, which may not be part of the space that the client mounted.

Another issue concerns refresh of referral locations. When referrals are used extensively, they may change as server configurations change. It is expected that clients will cache information related to traversing referrals so that future client side requests are resolved locally without server communication. This is usually rooted in client-side name lookup caching. Clients should periodically purge this data for referral points in order to detect changes in location information. When the change attribute changes for directories that hold referral entries or for the referral entries themselves, clients should consider any associated cached referral information to be out of date.

6. Acknowledgements

The authors wish to thank Neil Brown and Ted Anderson for their helpful comments on various drafts of the material presented here.

7. Normative References

[RFC3530]
S. Shepler, et. al., "NFS Version 4 Protocol", Standards Track RFC

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

Intellectual Property

Acknowledgement

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