

NFSv4  
Internet-Draft  
Intended status: Informational  
Expires: January 8, 2013

D. Noveck, Ed.  
EMC  
P. Shivam  
C. Lever  
B. Baker  
ORACLE  
July 7, 2012

NFSv4 migration: Implementation experience and spec issues to resolve  
draft-ietf-nfsv4-migration-issues-01

## Abstract

The migration feature of NFSv4 provides for moving responsibility for a single filesystem from one server to another, without disruption to clients. Recent implementation experience has shown problems in the existing specification for this feature. This document discusses the issues which have arisen and explores the options available for curing the issues via clarification and correction of the NFSv4.0 and NFSv4.1 specifications.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
2. Conventions . . . . .	4
3. NFSv4.0 Implementation Experience . . . . .	5
3.1. Implementation issues . . . . .	5
3.1.1. Failure to free migrated state on client reboot . . . . .	5
3.1.2. Server reboots resulting in a confused lease situation . . . . .	6
3.1.3. Client complexity issues . . . . .	7
3.2. Sources of Protocol difficulties . . . . .	9
3.2.1. Issues with nfs_client_id4 generation and use . . . . .	9
3.2.2. Issues with lease proliferation . . . . .	11
4. Issues to be resolved in NFSv4.0 . . . . .	11
4.1. Possible changes to nfs_client_id4 client-string . . . . .	11
4.2. Possible changes to handle differing nfs_client_id4 string values . . . . .	12
4.3. Other issues within migration-state sections . . . . .	13
4.4. Issues within other sections . . . . .	13
5. Proposed resolution of NFSv4.0 protocol difficulties . . . . .	14
5.1. Proposed changes: nfs_client_id4 client-string . . . . .	14
5.2. Client-string Approaches (AS PROPOSED) . . . . .	14
5.2.1. Non-Uniform Client-string Approach . . . . .	16
5.2.2. Uniform Client-string Approach . . . . .	16
5.2.3. Mixing Client-string Approaches . . . . .	18
5.2.4. Trunking Determination using Uniform Client-strings . . . . .	19
5.3. Proposed changes: merged (vs. synchronized) leases . . . . .	24
5.4. Other proposed changes to migration-state sections . . . . .	25
5.4.1. Proposed changes: Client ID migration . . . . .	25
5.4.2. Proposed changes: Callback re-establishment . . . . .	26
5.4.3. Proposed changes: NFS4ERR_LEASE_MOVED rework . . . . .	26
5.5. Proposed changes to other sections . . . . .	27
5.5.1. Proposed changes: callback update . . . . .	27
5.5.2. Proposed changes: clientid4 handling . . . . .	27
5.5.3. Proposed changes: NFS4ERR_CLID_INUSE . . . . .	29
5.6. Migration, Replication and State (AS PROPOSED) . . . . .	29
5.6.1. Migration and State . . . . .	30
5.6.2. Replication and State . . . . .	32
5.6.3. Notification of Migrated Lease . . . . .	32
5.6.4. Migration and the Lease_time Attribute . . . . .	35
6. Results of proposed changes for NFSv4.0 . . . . .	35

6.1.	Results: Failure to free migrated state on client reboot . . . . .	36
6.2.	Results: Server reboots resulting in confused lease situation . . . . .	36
6.3.	Results: Client complexity issues . . . . .	38
6.4.	Result summary . . . . .	39
7.	Issues for NFSv4.1 . . . . .	39
7.1.	Addressing state merger in NFSv4.1 . . . . .	39
7.2.	Addressing pNFS relationship with migration . . . . .	40
7.3.	Addressing server owner changes in NFSv4.1 . . . . .	40
8.	Lock State and File System Transitions (AS PROPOSED) . . . . .	41
8.1.	File System Transitions with Matching Server Scopes . . . . .	42
8.2.	File System Transitions with Non-Matching Server Scopes . . . . .	43
8.3.	FS Transitions Involving Reobtaining Locking State . . . . .	44
9.	Security Considerations . . . . .	45
10.	IANA Considerations . . . . .	45
11.	Acknowledgements . . . . .	45
12.	References . . . . .	46
12.1.	Normative References . . . . .	46
12.2.	Informative References . . . . .	46
	Authors' Addresses . . . . .	46

## 1. Introduction

This document is in the informational category, and while the facts it reports may have normative implications, any such normative significance reflects the readers' preferences. For example, we may report that the reboot of a client with migrated state results in state not being promptly cleared and that this will prevent granting of conflicting lock requests at least for the lease time, which is a fact. While it is to be expected that client and server implementers will judge this to be a situation that is best avoided, the judgment as to how pressing this issue should be considered is a judgment for the reader, and eventually the nfsv4 working group to make.

We do explore possible ways in which such issues can be avoided, with minimal negative effects, in the expectation that the working group will choose to address these issues, but the choice of exactly how to address these is best given effect in one or more standards-track documents and/or errata.

This document focuses on NFSv4.0, since that is where the majority of implementation experience has been. Nevertheless, there is some discussion of the implications of the NFSv4.0 experience for migration in NFSv4.1.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In the context of this informational document, these normative keywords will always occur in the context of a quotation, most often direct but sometimes indirect. The context will make it clear whether the quotation is from:

- o The current definitive definition of the NFSv4.0 protocol, whether that is the original NFSv4.0 specification [RFC3530], the current pending draft of RFC3530bis expected to become the definitive definition of NFSv4.0 once certain procedural steps are taken [cur-v4.0-bis], or an eventual RFC3530bis RFC, taking over the role of definitive definition of NFSv4.0 from RFC3530.

As the identity of that document may change during the lifetime of this document, we will often refer to the current or pending definition of NFSv4.0 and quote from portions of the documents that are identical among all existing drafts. Given that RFC3530 and all RFC3530bis drafts agree as to the issues under discussion,

this should not cause undue difficulty. Note that to simplify document maintenance, section names rather than section numbers are used when referring to sections in existing documents so that only minimal changes will be necessary as the identity of the document defining NFSv4.0 changes.

- o The current definitive definition of the NFSv4.1 protocol [RFC5661].
- o A proposed or possible text to serve as a replacement for the current definitive document text. Sometimes, a number of possible alternative texts may be listed and benefits and detriments of each examined in turn.

### 3. NFSv4.0 Implementation Experience

#### 3.1. Implementation issues

Note that the examples below reflect current experience which arises from clients implementing the recommendation to use different `nfs_client_id4` id strings for different server addresses, i.e. using what is later referred to herein as the "non-uniform client-string approach"

This is simply because that is the experience implementers have had. The reader should not assume that in all cases, this practice is the source of the difficulty. It may be so in some cases but clearly it is not in all cases.

##### 3.1.1. Failure to free migrated state on client reboot

The following sort of situation has proved troublesome:

- o A client C establishes a `clientid4` C1 with server ABC specifying an `nfs_client_id4` with id string value "C-ABC" and boot verifier 0x1111.
- o The client begins to access files in filesystem F on server ABC, resulting in generating stateids S1, S2, etc. under the lease for `clientid4` C1. It may also access files on other filesystems on the same server.
- o The filesystem is migrated from ABC to server XYZ. When transparent state migration is in effect, stateids S1 and S2 and `clientid4` C1 are now available for use by client C at server XYZ. So far, so good.

- o Client C reboots and attempts to access data on server XYZ, whether in filesystem F or another. It does a SETCLIENTID with an `nfs_client_id4` with id string value "C-XYZ" and boot verifier 0x112. There is thus no occasion to free stateids S1 and S2 since they are associated with a different client name and so lease expiration is the only way that they can be gotten rid of.

Note here that while it seems clear to us in this example that C-XYZ and C-ABC are from the same client, the server has no way to determine the structure of the "opaque" id string. In the protocol, it really is treated as opaque. Only the client knows which `nfs_client_id4` values designate the same client on a different server.

### 3.1.2. Server reboots resulting in a confused lease situation

Further problems arise from scenarios like the following.

- o Client C talks to server ABC using an `nfs_client_id4` id string such as "C-ABC" and a boot verifier `v1`. As a result, a lease with `clientid4 c.i` is established: `{v1, "C-ABC", c.i}`.
- o `fs_a1` migrates from server ABC to server XYZ along with its state. Now server XYZ also has a lease: `{v1, "C-ABC", c.i}`.
- o Server ABC reboots.
- o Client C talks to server ABC using an `nfs_client_id4` id string such as "C-ABC" and a boot verifier `v1`. As a result, a lease with `clientid4 c.j` is established: `{v1, "C-ABC", c.j}`.
- o `fs_a2` migrates from server ABC to server XYZ. Now server XYZ also has a lease: `{v1, "C-ABC", c.j}`.
- o Now server XYZ has two leases that match `{v1, "C-ABC", *}`, when the protocol clearly assumes there can be only one.

Note that if the client used "C" (rather than "C-ABC") as the `nfs_client_id4` id string, the exact same situation would arise.

One of the first cases in which this sort of situation has resulted in difficulties is in connection with doing a SETCLIENTID for callback update.

The SETCLIENTID for callback update only includes the `nfs_client_id4`, assuming there can only be one such with a given `nfs_client_id4` value. If there were multiple, confirmed client records with identical `nfs_client_id4` id string values, there would be no way to

map the callback update request to the correct client record. Apart from the migration handling specified in [RFC3530], such a situation cannot arise.

One possible accommodation for this particular issue that has been used is to add a RENEW operation along with SETCLIENTID (on a callback update) to disambiguate the client.

When the client updates the callback info to the destination, the client would, by convention, send a compound like this:

```
{ RENEW clientid4, SETCLIENTID nfs_client_id4,verf,cb }
```

The presence of the clientid4 in the compound would allow the server to differentiate among the various leases that it knows of, all with the same nfs\_client\_id4 value.

While this would be a reasonable patch for an isolated protocol weakness, interoperable clients and servers would require that the protocol truly be updated to allow such a situation, specifically that of multiple clientid4's with the same nfs\_client\_id4 value. The protocol is currently designed and implemented assuming this can't happen. We need to either prevent the situation from happening, or fully adapt to the possibilities which can arise. See Section 4 for a discussion of such issues.

### 3.1.3. Client complexity issues

Consider the following situation:

- o There are a set of clients C1 through Cn accessing servers S1 through Sm. Each server manages some significant number of filesystems with the filesystem count L being significantly greater than m.
- o Each client Cx will access a subset of the servers and so will have up to m clientid's, which we will call Cxy for server Sy.
- o Now assume that for load-balancing or other operational reasons, numbers of filesystems are migrated among the servers. As a result, each client-server pair will have up to m clientid's and each client will have up to m\*\*2 clientids. If we add the possibility of server reboot, the only bound on a client's clientid count is L.

Now, instead of a clientid4 identifying a client-server pair, we have many more entities for the client to deal with. In addition, it isn't clear how new state is to be incorporated in this structure.

The limitations of the migrated state (inability to be freed on reboot) would argue against adding more such state but trying to avoid that would run into its own difficulties. For example, a single lockowner string presented under two different clientids would appear as two different entities.

Thus we have to choose between:

- o indefinite prolongation of foreign clientid's even after all transferred state is gone.
- o having multiple requests for the same lockowner-string-named entity carried on in parallel by separate identically named lockowners under different clientid4's
- o Adding serialization at the lock-owner string level, in addition to that at the lockowner level.

In any case, we have gone (in adding migration as it was described) from a situation in which

- o Each client has a single clientid4/lease for each server it talks to.
- o Each client has a single nfs\_client\_id4 for each server it talks to.
- o Every state id can be mapped to an associated lease based on the server it was obtained from.

To one in which

- o Each client may have multiple clientid4's for a single server.
- o For each stateid, the client must separately record the clientid4 that it is assigned to, or it must manage separate "state blobs" for each fsid and map those to clientid4's.
- o Before doing an operation that can result in a stateid, the client must either find a "state blob" based on fsid or create a new one, possibly with a new clientid4.
- o There may be multiple clientid4's all connected to the same server and using the same nfs\_clientid4.

This sort of additional client complexity is troublesome and needs to be eliminated.



### 3.2. Sources of Protocol difficulties

#### 3.2.1. Issues with nfs\_client\_id4 generation and use

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Client ID" says:

The second field, id is a variable length string that uniquely defines the client.

There are two possible interpretations of the phrase "uniquely defines" in the above:

- o The relation between strings and clients is a function from such strings to clients so that each string designates a single client.
- o The relation between strings and clients is a bijection between such strings and clients so that each string designates a single client and each client is named by a single string.

The first interpretation would make these client-strings like phone numbers (a single person can have several) while the second would make them like social security numbers.

Endless debate about the true meaning of "uniquely defines" in this context is quite possible but not very helpful. The following points should be noted though:

- o The second interpretation is more consistent with the way "uniquely defines" is used elsewhere in the spec.
- o The spec as now written intends the first interpretation (or is internally inconsistent). In fact, it recommends, although it doesn't "RECOMMEND" that a single client have at least as many client-strings as server addresses that it interacts with. It says, in the third bullet point regarding construction of the string (which we shall henceforth refer to as client-string-BP3):

The string should be different for each server network address that the client accesses, rather than common to all server network addresses.

- o If internode interactions are limited to those between a client and its servers, there is no occasion for servers to be concerned with the question of whether two client-strings designate the same client, so that there is no occasion for the difference in interpretation to matter.

- o When transparent migration of client state occurs between two servers, it becomes important to determine when state on two different servers is for the same client or not, and this distinction becomes very important.

Given the need for the server to be aware of client identity with regard to migrated state, either client-string construction rules will have to change or there will be a need to get around current issues, or perhaps a combination of these two will be required. Later sections will examine the options and propose a solution.

One consideration that may indicate that this cannot remain exactly as it is today has to do with the fact that the current explanation for this behavior is not correct. The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Client ID" says:

The reason is that it may not be possible for the client to tell if the same server is listening on multiple network addresses. If the client issues SETCLIENTID with the same id string to each network address of such a server, the server will think it is the same client, and each successive SETCLIENTID will cause the server to begin the process of removing the client's previous leased state.

In point of fact, a "SETCLIENTID with the same id string" sent to multiple network addresses will be treated as all from the same client but will not "cause the server to begin the process of removing the client's previous leased state" unless the server believes it is a different instance of the same client, i.e. if the id string is the same and there is a different boot verifier. If the client does not reboot, the verifier should not change. If it does reboot, the verifier will change, and the server should "begin the process of removing the client's previous leased state."

The situation of multiple SETCLIENTID requests received by a server on multiple network addresses is exactly the same, from the protocol design point of view, as when multiple (i.e. duplicate) SETCLIENTID requests are received by the server on a single network address. The same protocol mechanisms that prevent erroneous state deletion in the latter case prevent it in the former case. There is no reason for special handling of the multiple-network-appearance case, in this regard.

### 3.2.2. Issues with lease proliferation

It is often felt that this is a consequence of the client-string construction issues, and it is certainly the case that the two are closely connected in that non-uniform client-strings make it impossible for the server to appropriately combine leases from the same client. See Section 5.2.1 for a discussion of non-uniform client-strings.

However, even where the server could combine leases from the same client, it needs to be clear how and when it will do so, so that the client will be prepared. These issues will have to be addressed at various places in the spec.

This could be enough only if we are prepared to do away with the "should" recommending non-uniform client-strings and replace it with a "should not" or even a "SHOULD NOT". Current client implementation patterns make this an unpalatable choice for use as a general solution, but it is reasonable to "RECOMMEND" this choice for a well-defined subset of clients. One alternative would be to create a way for the server to infer from client behavior which leases are held by the same client and use this information to do appropriate lease mergers. Prototyping and detailed specification work has shown that this could be done but the resulting complexity is such that a better choice is to "RECOMMEND" use of the uniform approach for clients supporting the migration feature.

Because of the discussion of client-string construction in [RFC3530], most existing clients implement the non-uniform client-string approach. As a result, existing servers may not have been tested with clients implementing uniform client-strings. As a consequence, care must be taken to preserve interoperability between UCS-capable clients and servers that don't tolerate uniform client strings for one reason or another. See Section 5.2.3 for details.

## 4. Issues to be resolved in NFSv4.0

### 4.1. Possible changes to nfs\_client\_id4 client-string

The fact that the reason given in client-string-BP3 is not valid makes the existing "should" insupportable. We can't either

- o Keep a reason we know is invalid.
- o Keep saying "should" without giving a reason.

What are often presented as reasons that motivate use of the non-

uniform approach always turn out to be cases in which, if the uniform approach were used, the server will treat a client which accesses that server via two different IP addresses as part of a single client, as it in fact is. This may be disconcerting to a client unaware that the two IP addresses connect to the same server. This is thus not a reason to use the non-uniform approach but rather an illustration of the fact that those using the uniform approach must use server behavior to determine whether any trunking of IP addresses exists, as is described in Section 5.2.2.

It is always possible that a valid new reason will be found, but so far none has been proposed. Given the history, the burden of proof should be on those asserting the validity of a proposed new reason.

So we will assume for now that the "should" will have to go. The question is what to replace it with.

- o We can't say "MUST NOT", despite the problems this raises for migration since this is pretty late in the day for such a change. Many currently operating clients obey the existing "should". Similar considerations would apply for "SHOULD NOT" or "should not".
- o Dropping client-string-BP3 entirely is a possibility but, given the context and history, it would just be a confusing version of "SHOULD NOT".
- o Using "MAY" would clearly specify that both ways of doing this are valid choices for clients and that servers will have to deal with clients that make either choice.
- o This might be modified by a "SHOULD" (or even a "MUST") for particular groups of clients.
- o There will have to be some text explaining why a client might make either choice but, except for the particular cases referred to above, we will have to make sure that it is truly descriptive, and not slanted in either direction.

#### 4.2. Possible changes to handle differing nfs\_client\_id4 string values

Given the difficulties caused by having different nfs\_client\_id4 client-string values for the same client, we have two choices:

- o Deprecate the existing treatment and basically say the client is on its own doing migration, if it follows it.

- o Introduce a way of having the client provide client identity information to the server, if it can be done compatibly while staying within the bounds of v4.0.

#### 4.3. Other issues within migration-state sections

There are a number of issues where the existing text is unclear and/or wrong and needs to be fixed in some way.

- o Lack of clarity in the discussion of moving clientids (as well as stateids) as part of moving state for migration.
- o The discussion of synchronized leases is wrong in that there is no way to determine (in the current spec) when leases are for the same client and also wrong in suggesting a benefit from leases synchronized at the point of transfer. What is needed is merger of leases, which is necessary to keep client complexity requirements from getting out of hand.
- o Lack of clarity in the discussion of LEASE\_MOVED handling, including failure to fully address situations in which transparent state migration did not occur.

#### 4.4. Issues within other sections

There are a number of cases in which certain sections, not specifically related to migration, require additional clarification. This is generally because text that is clear in a context in which leases and clientids are created in one place and live there forever may need further refinement in the more dynamic environment that arises as part of migration.

Some examples:

- o Some people are under the impression that updating callback endpoint information for an existing client, as used during migration, may cause the destination server to free existing state. There need to be additions to clarify the situation.
- o The handling of the sets of clientid4's maintained by each server needs to be clarified. In particular, the issue of how the client adapts to the presumably independent and uncoordinated clientid4 sets needs to be clearly addressed
- o Statements regarding handling of invalid clientid4's need to be clarified and/or refined in light of the possibilities that arise due to lease motion and merger.

- o Confusion and lack of clarity about NFS4ERR\_CLID\_INUSE.

## 5. Proposed resolution of NFSv4.0 protocol difficulties

### 5.1. Proposed changes: nfs\_client\_id4 client-string

We propose replacing client-string-BP3 with the following text and adding the following proposed Section 5.2 to provide implementation guidance.

- o The string MAY be different for each server network address that the client accesses, rather than common to all server network addresses.
- o The considerations that might influence a client to use different strings for different network server addresses are explained in Section 5.2.
- o Despite the use of the word "string" for this identifier, and the fact that using strings will often be convenient, it should be understood that the protocol defines this as opaque data. In particular, those receiving such an id should not assume that it will be in UTF-8 format. Servers MUST NOT reject an nfs\_client\_id4 simply because the id string is not in UTF-8 format.

### 5.2. Client-string Approaches (AS PROPOSED)

One particular aspect of the construction of the nfs4\_client\_id4 string has proved recurrently troublesome. The client has a choice of:

- o Presenting the same id string to multiple server addresses. This is referred to as the "uniform client-string approach" and is discussed in Section 5.2.2.
- o Presenting different id strings to multiple server addresses. This is referred to as the "non-uniform client-string approach" and is discussed in Section 5.2.1.

Note that implementation considerations, including compatibility with existing servers, may make it desirable for a client to use both approaches, based on configuration information, such as mount options. This issue will be discussed in Section 5.2.3.

Construction of the client-string has been a troublesome issue because of the way in which the NFS protocols have evolved.

- o NFSv3 as a stateless protocol had no need to identify the state shared by a particular client-server pair. Thus there was no occasion to consider the question of whether a set of requests come from the same client, or whether two server IP addresses are connected to the same server. As the environment was one in which the user supplied the target server IP address as part of incorporating the remote filesystem in the client's file name space, there was no occasion to take note of server trunking. Within a stateless protocol, the situation was symmetrical. The client has no server identity information and the server has no client identity information.
- o NFSv4.1 is a stateful protocol with full support for client and server identity determination. This enables the server to be aware when two requests come from the same client (they are on sessions sharing a clientid4) and the client to be aware when two server IP addresses are connected to the same server (they return the same server name in responding to an EXCHANGE\_ID).

NFSv4.0 is unfortunately halfway between these two. The two client-string approaches have arisen in attempts to deal with the changing requirements of the protocol as implementation has proceeded and features that were not very substantial in [RFC3530], got more substantial.

- o In the absence of any implementation of the fs\_locations-related features (replication, referral, and migration), the situation is very similar to that of NFSv3, with the addition of state but with no concern to provide accurate client and server identity determination. This is the situation that gave rise to the non-uniform client-string approach.
- o In the presence of replication and referrals, the client may have occasion to take advantage of knowledge of server trunking information. Even more important, migration, by transferring state among servers, causes difficulties for the non-uniform client-string approach, in that the two different client-strings sent to different IP addresses may wind up on the same IP address, adding confusion.
- o A further consideration is that client implementations typically provide NFSv4.1 by augmenting their existing NFSv4.0 implementation, not by providing two separate implementations. Thus the more NFSv4.0 and NFSv4.1 can work alike, the less complex are clients. This is a key reason why those implementing NFSv4.0 clients might prefer using the uniform client string model, even if they have chosen not to provide fs\_locations-related features in their NFSv4.0 client.

Both approaches have to deal with the asymmetry in client and server identity information between client and server. Each seeks to make the client's and the server's views match. In the process, each encounters some combination of inelegant protocol features and/or implementation difficulties. The choice of which to use is up to the client implementer and the sections below try to give some useful guidance.

#### 5.2.1. Non-Uniform Client-string Approach

The non-uniform client-string approach is an attempt to handle these matters in NFSv4.0 client implementations in as NFSv3-like a way as possible.

For a client using the non-uniform approach, all internal recording of clientid4 values is to include, whether explicitly or implicitly, the server IP address so that one always has an (IP-address, clientid4) pair. Two such pairs from different servers are always distinct even when the clientid4 values are the same, as they may occasionally be. In this approach, such equality is always treated as simple happenstance.

Making the client-string different on different servers means that a server has no way of tying together information from the same client and so will treat a single client as multiple clients with multiple leases for each server network address. Since there is no way in the protocol for the client to determine if two network addresses are connected to the same server, the resulting lack of knowledge is symmetrical and can result in simpler client implementations in which there is a single clientid/lease per server network addresses.

Support for migration, particularly with transparent state migration, is more complex in the case of non-uniform client-strings. For example, migration of a lease can result in multiple leases for the same client accessing the same server addresses, vitiating many of the advantages of this approach. Therefore, client implementations that support migration with transparent state migration SHOULD NOT use the non-uniform client-string approach, except where it is necessary for compatibility with existing server implementations (For details of arranging use of multiple client-string approaches, see Section 5.2.3).

#### 5.2.2. Uniform Client-string Approach

When the client-string is kept uniform, the server has the basis to have a single clientid4/lease for each distinct client. The problem that has to be addressed is the lack of explicit server identity information, which is made available in NFSv4.1.



When the same client-string is given to multiple IP addresses, the client can determine whether two IP addresses correspond to a single server, based on the server's behavior. This is the inverse of the strategy adopted for the non-uniform approach in which different server IP addresses are told about different clients, simply to prevent a server from manifesting behavior that is inconsistent with there being a single server for each IP address, in line with the traditions of NFS. So, to compare:

- o In the non-uniform approach, servers are told about different clients because, if the server were to use accurate information as to client identity, two IP addresses on the same server would behave as if they were talking to the same client, which might prove disconcerting to a client not expecting such behavior.
- o In the uniform approach, the servers are told about there being a single client, which is, after all, the truth. Then, when the server uses this information, two IP addresses on the same server will behave as if they are talking to the same client, and this difference in behavior allows the client to infer the server IP address trunking configuration, even though NFSv4.0 does not explicitly provide this information.

The approach given in the section below shows one example of how this might be done.

The uniform client-string approach makes it necessary to exercise more care in the definition of the `nfs_client_id4` boot verifier:

- o In [RFC3530], the client is told to change the boot verifier when reboot occurs, but there is no explicit statement as to the converse, so that any requirement to keep the verifier constant unless rebooting is only present by implication.
- o Many existing clients change the boot verifier every time they destroy and recreate the data structure that tracks an <IP-address, clientid4> pair. This might happen if the last mount of a particular server is removed, and then a fresh mount is created. And, note that this might result in each <IP-address, clientid4> pair having its own boot verifier that is independent of the others.
- o Within the uniform client-string approach, an `nfs_client_id4` designates a globally known client instance, so that the boot verifier should change if and only if a new client instance is created, typically as a result of a reboot.

The following are advantages for the implementation of using the

uniform client-string approach:

- o Clients can take advantage of server trunking (and clustering with single-server-equivalent semantics) to increase bandwidth or reliability.
- o There are advantages in state management so that, for example, we never have a delegation under one clientid revoked because of a reference to the same file from the same client under a different clientid.
- o The uniform client-string approach allows the server to do any necessary automatic lease merger in connection with migration, without requiring any client involvement. This consideration is of sufficient weight to cause us to RECOMMEND use of the uniform client-string approach for clients supporting transparent state migration.

The following implementation considerations might cause issues for client implementations.

- o This approach is considerably different from the non-uniform approach, which most client implementations have been following. Until substantial implementation experience is obtained with this approach, reluctance to embrace something so new is to be expected.
- o Mapping between server network addresses and leases is more complicated in that it is no longer a one-to-one mapping.

How to balance these considerations depends on implementation goals.

### 5.2.3. Mixing Client-string Approaches

As noted above, a client which needs to use the uniform client-string approach (e.g. to support migration), may also need to support existing servers with implementations that do not work properly in this case.

Some examples of such server issues include:

- o Some existing NFSv4 server implementations of IP-address failover depend on clients' use of a non-uniform client-string approach. In particular, when a server supports both its own IP address and one failed over from a partner server, it may have separate sets of state applicable to the two IP addresses, owned by different servers but residing on a single one.

In this situation, some servers have relied on clients' use of the non-uniform client-string approach, as suggested but not mandated by [RFC3530], to keep these sets of state separate, and will have problems in handling clients using the uniform client-string approach, in that such clients will see changes in trunking relationships whenever server failover and giveback occur.

- o Some existing servers incorrectly return NFS4ERR\_CLID\_INUSE in a way which interferes with clients using the uniform client-string approach. See Section 5.5.3 for details.

In order to support such servers, the client can use different approaches for different mounts, as long as:

- o The uniform client-string approach is used when accessing servers that may return NFS4ERR\_MOVED.
- o The non-uniform client-string approach is used when accessing servers whose implementations make them incompatible with the uniform client-string approach

One effective way for clients to handle this is to support the uniform client-string approach as the default, but allow a mount option to specify use of the non-uniform client-string approach for particular mount points, as long as such mount points are not used when migration is to be supported.

In the case in which the same server has multiple mounts, and both approaches are specified for the same server, the client could have multiple clientids corresponding to the same server, one for each approach and would then have to keep these separate.

#### 5.2.4. Trunking Determination using Uniform Client-strings

This section provides an example of how trunking determination could be done by a client following the uniform client-string approach (whether this is used for all mounts or not). Clients need not follow this procedure but implementers should make sure that the issues dealt with by this procedure are all properly addressed.

We need to clarify the various possible purposes of trunking determination and the corresponding requirements as to server behavior. The following points should be noted:

- o The primary purpose of the trunking determination algorithm is to make sure that, if the server treats client requests on two IP addresses as part of the same client, the client will not be blind-sided and encounter disconcerting server behavior, as

mentioned in Section 5.2.2. Such behavior could occur if the client were unaware that all of its client requests for the two IP addresses were being handled as part of a single client talking to a single server.

- o A second purpose to be able to use knowledge of trunking relationships for better performance, etc
- o If a server were to give out distinct clientid's in response to receiving the same nfs\_client\_id4 on different network addresses, and acted as if these were separate clients, the primary purpose of trunking determination would be met, as long as the server did not treat them as part of the same client. In this case, the server would be acting, with regard to that client, as if it were two distinct servers. This would interfere with the secondary purpose of trunking determination but there is nothing the client can do about that.
- o Suppose a server were to give such a client two different clientid's but act as if they were one. That it is the only way that the server could behave in a way that would defeat the primary purpose of the trunking determination algorithm.

Servers MUST NOT do that.

For a client using the uniform approach, clientid4 values are treated as important information in determining server trunking patterns. For two different IP addresses to return the same clientid4 value is a necessary, though not a sufficient condition for them to be considered as connected to the same server. As a result, when two different IP addresses return the same clientid4, the client needs to determine, using the procedure given below or otherwise, whether the IP addresses are connected to the same server. For such clients, all internal recording of clientid4 values needs to include, whether explicitly or implicitly, identification of the server from which the clientid4 was received so that one always has a (server, clientid4) pair. Two such pairs from different servers are always considered distinct even when the clientid4 values are the same, as they may occasionally be.

In order to make this approach work, the client must have accessible, for each nfs4\_client\_id4 used by the uniform approach (only one in general) a list of all server IP addresses, together with the associated clientid4 values and authentication flavors. As a part of the associated data structures, there should be the ability to mark a server IP structure as having the same server as another and to mark an IP-address as currently unresolved. One way to do this is to allow each such entry to point to another with the pointer value

being one of:

- o A pointer to another entry for an IP address associated with the same server, where that IP address is the first one referenced to access that server.
- o A pointer to the current entry if there is no earlier IP address associated with the same server, i.e. where the current IP address is the first one referenced to access that server. We'll refer to such an IP address as the lead IP address for a given server.
- o The value NULL if the address's server identity is currently unresolved.

In order to keep the above information current, in the interests of the most effective trunking determination, RENEWS should be periodically done on each server. However, even if this is not done, the primary purpose of the trunking determination algorithm, to prevent confusion due to trunking hidden from the client, will be achieved.

Given this apparatus, when a SETCLIENTID is done and a clientid4 returned, the data structure can be searched for a matching clientid4 and if such is found, further processing can be done to determine whether the clientid4 match is accidental, or the result of trunking.

In this algorithm, when SETCLIENTID is done it will use the common nfs\_client\_id4 and specify the current target IP address as part of the callback parameters. We call the clientid4 and SETCLIENTID verifier returned by this operation XC and XV.

Note that when the client has done previous SETCLIENTID's, to any IP addresses, with more than one authentication flavor, we have the possibility of receiving NFS4ERR\_CLID\_INUSE, since we do not yet know which of our connections with existing IP addresses might be trunked with our current one. In the event that the SETCLIENTID fails with NFS4ERR\_CLID\_INUSE, one must try all other authentication flavors currently in use and eventually one will be correct and not return NFS4ERR\_CLID\_INUSE.

Note that at this point, no SETCLIENTID\_CONFIRM has yet been done. This is because our SETCLIENTID has either established a new clientid4 on a previously unknown server or changed the callback parameters on a clientid4 associated with some already known server. Given that we don't want to confirm something that we are not sure we want to happen, what is to be done next depends on information about existing clientid4's.

- o If no matching clientid4 is found, the IP address X and clientid4 XC are added to the list and considered as having no existing known IP addresses trunked with it. The IP address is marked as a lead IP address for a new server. A SETCLIENTID\_CONFIRM is done using XC and XV.
- o If a matching clientid4 is found which is marked unresolved, processing on the new IP address is suspended. In order to simplify processing, there can only be one unresolved IP address for any given clientid4.
- o If one or more matching clientid4's is found, none of which is marked unresolved, the new IP address is entered and marked unresolved. After applying the steps below to each of the lead IP addresses with a matching clientid4, the address will have been resolved: either it will be part of the same server as a new IP address to be added to an existing set of IP addresses for a server, or it will be recognized as a new server. At the point at which this determination is made, the unresolved indication is cleared and any suspended SETCLIENTID processing is restarted

So for each lead IP address IPn with a clientid4 matching XC, the following steps are done.

- o If the authentication flavor for IPn does not match that for X, the IP address is skipped, since it is impossible for IPn and X to be trunked in these circumstances. This avoids any possibility that NFS4ERR\_CLID\_INUSE will be returned for the SETCLIENTID and SETCLIENTID\_CONFIRM to be done below, as long as the server(s) at IP addresses IPn and X are correctly implemented.
- o A SETCLIENTID is done to update the callback parameters to reflect the possibility that X will be marked as associated with the server whose lead IP address is IPn. The specific callback parameters chosen, in terms of cb\_client4 and callback\_ident, are up to the client and should reflect its preferences as to callback handling for the common clientid, in the event that X and IPn are trunked together. So assume that we do that SETCLIENTID on IP address IPn and get back a setclientid\_confirm value (in the form of a verifier4) SCn.

Note that the v4.0 spec requires the server to make sure that such value are very unlikely to be regenerated. Given that it is already highly unlikely that the clientid XC is duplicated by distinct servers, the probability that SC is duplicated as well has to be considered vanishingly small. Note also that the callback update procedure can be repeated multiple times to reduce the probability of spurious matches further.

- o Note that we don't want this to happen if address X is not associated with this server. So we do a SETCLIENTID\_CONFIRM on address X using the setclientid\_confirm value SCn.
- o If the setclientid\_confirm value generated on X is accepted on IPn, then X and IPn are recognized as connected to the same server and the entry for X is marked as associated with IPn. The entry is now resolved and processing can be restarted for IP addresses whose clientid4 matched XC but whose resolution had been deferred.
- o If the confirm value generated on IPn is not accepted on X, then X and IPn are distinct and the callback update will not be confirmed. So we go on to the next IPn, until we run out of them. If it happens that we run out of potential matches, then we can treat X as connected to a distinct server and then update and confirm its callback parameters on that basis.

Note here that we may set a number of possible values for the callback parameters to be used for XC, one for the possibility that X is untrunked, and others for each potential match with an existing IPn. Although there are multiple such updates at most one will be confirmed and, if X is untrunked, its original callback parameters will be put in effect by its SETCLIENTID\_CONFIRM.

The procedure above has made no explicit mention of the possibility that server reboot can occur at any time. To address this possibility the client should periodically use the clientid4 XC in RENEW operations, directed to both the IP address X and the current lead IP address that is currently being tested for identity.

- o When XC becomes invalid on X, the resolution process should be terminated, subject to being redone later. Before redoing the resolution, XC should be checked on all the lead IP addresses on which it was valid. Once a new clientid4 is established on any servers on which XC became invalid, a new clientid4 can be established on X and the resolution process for X can be restarted.
- o When XC does not become invalid on X, but becomes invalid on the current IPn being tested, it should be concluded that X and IPn do not match and that it is time to advance to the next IPn, if any.
- o In the event of a reboot detected on any server lead IP, the set of IP addresses associated with the server should not change and state should be re-established for the lease as a whole, using all available connected server IP addresses. It is prudent to verify connectivity by doing a RENEW using the new clientid4 on each such server address before using it, however.

If we have run out of IPn's without finding a matching server, X is considered as having no existing known IP addresses trunked with it. The IP address is marked as a lead IP address for a new server. A SETCLIENTID\_CONFIRM is done using XC and XV.

### 5.3. Proposed changes: merged (vs. synchronized) leases

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Migration and State" says:

As part of the transfer of information between servers, leases would be transferred as well. The leases being transferred to the new server will typically have a different expiration time from those for the same client, previously on the old server. To maintain the property that all leases on a given server for a given client expire at the same time, the server should advance the expiration time to the later of the leases being transferred or the leases already present. This allows the client to maintain lease renewal of both classes without special effort:

There are a number of problems with this and any resolution of our difficulties must address them somehow.

- o The current v4.0 spec recommends that the client make it essentially impossible to determine when two leases are from "the same client".
- o It is not appropriate to speak of "maintain[ing] the property that all leases on a given server for a given client expire at the same time", since this is not a property that holds even in the absence of migration. A server listening on multiple network addresses may have the same client appear as multiple clients with no way to recognize the client as the same.
- o Even if the client identity issue could be resolved, advancing the lease time at the point of migration would not maintain the desired synchronization property. The leases would be synchronized until one of them was renewed, after which they would be unsynchronized again.

To avoid client complexity, we need to have no more than one lease between a single client and a single server. This requires merger of leases since there is no real help from synchronizing them at a single instant.

For the uniform approach, the destination server would simply merge leases as part of state transfer, since two leases with the same



nfs\_client\_id4 values must be for the same client.

We have made the following decisions as far as proposed normative statements regarding for state merger. They reflect the facts that we want to support fully migration support in the simplest way possible and that we can't say MUST since we have older clients and servers to deal with.

- o Clients SHOULD use the uniform client-string approach in order to get good migration support.
- o Servers SHOULD provide automatic lease merger during state migration so that clients using the uniform id approach get the support automatically.

If the clients and the servers obey the SHOULD's, having more than a single lease for a given client-server pair will be a transient situation, cleaned up as part of adapting to use of migrated state.

Since clients and servers will be a mixture of old and new and because nothing is a MUST we have to ensure that no combination will show worse behavior than is exhibited by current (i.e. old) clients and servers.

#### 5.4. Other proposed changes to migration-state sections

##### 5.4.1. Proposed changes: Client ID migration

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Migration and State" says:

In the case of migration, the servers involved in the migration of a filesystem SHOULD transfer all server state from the original to the new server. This must be done in a way that is transparent to the client. This state transfer will ease the client's transition when a filesystem migration occurs. If the servers are successful in transferring all state, the client will continue to use stateids assigned by the original server. Therefore the new server must recognize these stateids as valid. This holds true for the client ID as well. Since responsibility for an entire filesystem is transferred with a migration event, there is no possibility that conflicts will arise on the new server as a result of the transfer of locks.

This poses some difficulties, mostly because the part about "client ID" is not clear:

- o It isn't clear what part of the paragraph the "this" in the statement "this holds true ..." is meant to signify.
- o The phrase "the client ID" is ambiguous, possibly indicating the clientid4 and possibly indicating the nfs\_client\_id4.
- o If the text means to suggest that the same clientid4 must be used, the logic is not clear since the issue is not the same as for stateids of which there might be many. Adapting to the change of a single clientid, as might happen as a part of lease migration, is relatively easy for the client.

We have decided to address this issue as follows, with the relevant changes all reflected in Section 5.6.

- o Make it clear that both clientid4 and nfs\_client\_id4 (including both id string and boot verifier) are to be transferred.
- o Indicate that the initial transfer will result in the same clientid4 after transfer but this is not guaranteed since there may conflict with an existing clientid4 on the destination server and because lease merger can result in a change of the clientid4.

#### 5.4.2. Proposed changes: Callback re-establishment

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Migration and State" says:

A client SHOULD re-establish new callback information with the new server as soon as possible, according to sequences described in sections "Operation 35: SETCLIENTID - Negotiate Client ID" and "Operation 36: SETCLIENTID\_CONFIRM - Confirm Client ID". This ensures that server operations are not blocked by the inability to recall delegations.

The above will need to be fixed to reflect the possibility of merging of leases and the text to do this appears as part of Section 5.6.

#### 5.4.3. Proposed changes: NFS4ERR\_LEASE\_MOVED rework

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Notification of Migrated Lease" says:

Upon receiving the NFS4ERR\_LEASE\_MOVED error, a client that supports filesystem migration MUST probe all filesystems from that server on which it holds open state. Once the client has

successfully probed all those filesystems which are migrated, the server MUST resume normal handling of stateful requests from that client.

There is a lack of clarity that is prompted by ambiguity about what exactly probing is and what the interlock between client and server must be. This has led to some worry about the scalability of the probing process, and although the time required does scale linearly with the number of fs's that the client may have state for with respect to a given server, the actual process can be done efficiently.

To address these issues we propose replacing the above with the text addressing NFS4RR\_LEASE\_MOVED as given in Section 5.6.3.

#### 5.5. Proposed changes to other sections

##### 5.5.1. Proposed changes: callback update

Some changes are necessary to reduce confusion about the process of callback information update and in particular to make it clear that no state is freed as a result:

- o Make it clear that after migration there are confirmed entries for transferred clientid4/nfs\_client\_id4 pairs.
- o Be explicit in the sections headed "otherwise," in the descriptions of SETCLIENTID and SETCLIENTID\_CONFIRM, that these don't apply in the cases we are concerned about.

##### 5.5.2. Proposed changes: clientid4 handling

To address both of the clientid4-related issues mentioned in Section 4.4, we propose replacing the last three paragraphs of the section entitled "Client ID" with the following:

Once a SETCLIENTID and SETCLIENTID\_CONFIRM sequence has successfully completed, the client uses the shorthand client identifier, of type clientid4, instead of the longer and less compact nfs\_client\_id4 structure. This shorthand client identifier (a client ID) is assigned by the server and should be chosen so that it will not conflict with a client ID previously assigned by same server. This applies across server restarts or reboots.

Distinct servers MAY assign clientid4's independently, and will generally do so. Therefore, a client has to be prepared to deal with multiple instances of the same clientid4 value received on

distinct IP addresses, denoting separate entities. When trunking of server IP addresses is not a consideration, a client should keep track of (IP-address, clientid4) pairs, so that each pair is distinct. For a discussion of how to address the issue in the face of possible trunking of server IP addresses, see Section 5.2.

When a clientid4 is presented to a server and that clientid4 is not recognized, the server will reject the request with the error NFS4ERR\_STALE\_CLIENTID. This can occur for a number of reasons:

- \* A server reboot causing loss of the server's knowledge of the client
- \* Client error sending an incorrect clientid4 or valid clientid4 to the wrong server.
- \* Loss of lease state due to lease expiration.
- \* Client or server error causing the server to believe that the client has rebooted (i.e. receiving a SETCLIENTID with an nfs\_client\_id4 which has a matching id string and a non-matching boot verifier).
- \* Migration of all state under the associated lease causes its non-existence to be recognized on the source server.
- \* Merger of state under the associated lease with another lease under a different clientid causes the clientid4 serving as the source of the merge to cease being recognized on its server.

In the event of a server reboot, or loss of lease state due to lease expiration, the client must obtain a new clientid4 by use of the SETCLIENTID operation and then proceed to any other necessary recovery for the server reboot case (See the section entitled "Server Failure and Recovery"). In cases of server or client error resulting in this error, use of SETCLIENTID to establish a new lease is desirable as well.

In the last two cases, different recovery procedures are required. See Section 5.6 for details. Note that in cases in which there is any uncertainty about which sort of handling is applicable, the distinguishing characteristic is that in reboot-like cases, the clientid4 and all associated stateids cease to exist while in migration-related cases, the clientid4 ceases to exist while the stateids are still valid.

The client must also employ the SETCLIENTID operation when it receives a NFS4ERR\_STALE\_STATEID error using a stateid derived

from its current clientid4, since this indicates a situation, such as server reboot which has invalidated the existing clientid4 and associated stateids (see the section entitled "lock-owner" for details).

See the detailed descriptions of SETCLIENTID and SETCLIENTID\_CONFIRM for a complete specification of the operations.

#### 5.5.3. Proposed changes: NFS4ERR\_CLID\_INUSE

It appears to be the intention that only a single authentication flavor be used for client establishment between any client-server pair. However:

- o There is no explicit statement to this effect.
- o The error that indicates an authentication flavor conflict has a name which does not clarify this issue: NFS4ERR\_CLID\_INUSE.
- o The definition of the error is also not very helpful: "The SETCLIENTID operation has found that a client id is already in use by another client".

As a result, servers exist which reject a SETCLIENTID simply because there already exists a clientid for the same client, established using a different IP address. Although this is generally understood to be erroneous, such servers still exist and the spec should make the correct behavior clear.

Although the error name cannot be changed, the following changes should be made to avoid confusion:

- o The definition of the error should be changed to read, "The SETCLIENTID operation has found that the specified nfs\_client\_id4 was previously presented with a different authentication flavor and that client instance currently holds an active lease."
- o In the description of SETCLIENTID, the phrase "then the server returns a NFS4ERR\_CLID\_INUSE error" should be expanded to read "then the server returns a NFS4ERR\_CLID\_INUSE error, since use of a single client with multiple principals is not allowed."

#### 5.6. Migration, Replication and State (AS PROPOSED)

When responsibility for handling a given filesystem is transferred to a new server (migration) or the client chooses to use an alternate server (e.g., in response to server unresponsiveness) in the context

of filesystem replication, the appropriate handling of state shared between the client and server (i.e., locks, leases, stateids, and client IDs) is as described below. The handling differs between migration and replication.

If a server replica or a server immigrating a filesystem agrees to, or is expected to, accept opaque values from the client that originated from another server, then it is a wise implementation practice for the servers to encode the "opaque" values in network byte order. When doing so, servers acting as replicas or immigrating filesystems will be able to parse values like stateids, directory cookies, filehandles, etc. even if their native byte order is different from that of other servers cooperating in the replication and migration of the filesystem.

#### 5.6.1. Migration and State

In the case of migration, the servers involved in the migration of a filesystem SHOULD transfer all server state from the original to the new server. This must be done in a way that is transparent to the client. This state transfer will ease the client's transition when a filesystem migration occurs. If the servers are successful in transferring all state, the client will continue to use stateids assigned by the original server. Therefore the new server must recognize these stateids as valid.

If transferring stateids from server to server would result in a conflict for an existing stateid for the destination server with the existing client, transparent state migration MUST NOT happen for that client. Servers participating in using transparent state migration should co-ordinate their stateid assignment policies to make this situation unlikely or impossible. The means by which this might be done, like all of the inter-server interactions for migration, are not specified by the NFS version 4.0 protocol.

Handling of clientid values is similar but not identical. The clientid4 and nfs\_client\_id4 information (id string and boot verifier) will be transferred with the rest of the state information and the destination server should use that information to determine appropriate clientid4 handling. Although the destination server may make state stored under an existing lease available under the clientid4 used on the source server, the client should not assume that this is always so. In particular,

- o If there is an existing lease with an nfs\_client\_id4 that matches a migrated lease (same id string and boot verifier), the server SHOULD merge the two, making the union of the sets of stateids available under the clientid4 for the existing lease. As part of

the lease merger, the expiration time of the lease will reflect renewal done within either of the ancestor leases (and so will reflect the latest of the renewals).

- o If there is an existing lease with an `nfs_client_id4` that partially matches a migrated lease (same id string and a different boot verifier), the server MUST eliminate one of the two, possibly invalidating one of the ancestor `clientid4`'s. Since boot verifiers are not ordered, the later lease renewal time will prevail.

When leases are not merged, the transfer of state should result in creation of a confirmed client record with empty callback information but matching the `{v, x, c}` for the transferred client information. This should enable establishment of new callback information using `SETCLIENTID` and `SETCLIENTID_CONFIRM`.

A client may determine the disposition of migrated state by using a `stateid` associated with the migrated state and in an operation on the new server and using the associated `clientid4` in a `RENEW` on the new server.

- o If the `stateid` is not valid and an error `NFS4ERR_BAD_STATEID` is received, either transparent state migration has not occurred or the state was purged due to boot verifier mismatch.
- o If the `stateid` is valid and an error `NFS4ERR_STALE_CLIENTID` is received on the `RENEW`, transparent state migration has occurred and the lease has been merged with an existing lease on the destination server.
- o If the `stateid` is valid and the `clientid4` is valid, the lease has been transferred intact.

Since responsibility for an entire filesystem is transferred with a migration event, there is no possibility that conflicts will arise on the new server as a result of the transfer of locks.

The servers may choose not to transfer the state information upon migration. However, this choice is discouraged, except where specific issues such as `stateid` conflicts make it necessary. In the case of migration without state transfer, when the client presents state information from the original server (e.g. in a `RENEW` op or a `READ` op of zero length), the client must be prepared to receive either `NFS4ERR_STALE_CLIENTID` or `NFS4ERR_STALE_STATEID` from the new server. The client should then recover its state information as it normally would in response to a server failure. The new server must take care to allow for the recovery of state information as it would

in the event of server restart.

When a lease is transferred to a new server (as opposed to being merged with a lease already on the new server), a client SHOULD re-establish new callback information with the new server as soon as possible, according to sequences described in sections "Operation 35: SETCLIENTID - Negotiate Client ID" and "Operation 36: SETCLIENTID\_CONFIRM - Confirm Client ID". This ensures that server operations are not blocked by the inability to recall delegations.

In those situation in which state has not been transferred, as shown by a return of NFS4ERR\_BAD\_STATEID, the client may attempt to reclaim the locks in order to take advantage of cases in which destination server has set up a file-system-specific grace period in support of the migration.

#### 5.6.2. Replication and State

Since client switch-over in the case of replication is not under server control, the handling of state is different. In this case, leases, stateids and client IDs do not have validity across a transition from one server to another. The client must re-establish its locks on the new server. This can be compared to the re-establishment of locks by means of reclaim-type requests after a server reboot. The difference is that the server has no provision to distinguish requests reclaiming locks from those obtaining new locks or to defer the latter. Thus, a client re-establishing a lock on the new server (by means of a LOCK or OPEN request), may have the requests denied due to a conflicting lock. Since replication is intended for read-only use of filesystems, such denial of locks should not pose large difficulties in practice. When an attempt to re-establish a lock on a new server is denied, the client should treat the situation as if its original lock had been revoked.

#### 5.6.3. Notification of Migrated Lease

In the case of lease renewal, the client may not be submitting requests for a filesystem that has been migrated to another server. This can occur because of the implicit lease renewal mechanism. The client renews a lease containing state of multiple filesystems when submitting a request to any one filesystem at the server.

In order for the client to schedule renewal of leases that may have been relocated to the new server, the client must find out about lease relocation before those leases expire. Similarly, when migration occurs but there has not been transparent state migration, the client needs to find out about the change soon enough to be able to reclaim the lock within the destination server's grace period. To



accomplish this, all operations which implicitly renew leases for a client (such as OPEN, CLOSE, READ, WRITE, RENEW, LOCK, and others), will return the error NFS4ERR\_LEASE\_MOVED if responsibility for any of the leases to be renewed has been transferred to a new server. Note that when the transfer of responsibility leaves remaining state for that lease on the source server, the lease is renewed just as it would have been in the NFS4ERR\_OK case, despite returning the error. The transfer of responsibility happens when the server receives a GETATTR(fs\_locations) from the client for each filesystem for which a lease has been moved to a new server. Normally it does this after receiving an NFS4ERR\_MOVED for an access to the filesystem but the server is not required to verify that this happens in order to terminate the return of NFS4ERR\_LEASE\_MOVED. By convention, the compounds containing GETATTR(fs\_locations) SHOULD include an appended RENEW operation to permit the server to identify the client getting the information.

Note that the NFS4ERR\_LEASE\_MOVED error is only required when responsibility for at least one stateid has been affected. In the case of a null lease, where the only associated state is a clientid, no NFS4ERR\_LEASE\_MOVED error need be generated.

Upon receiving the NFS4ERR\_LEASE\_MOVED error, a client that supports filesystem migration MUST perform the necessary GETATTR operation for each of the filesystems containing state that have been migrated and so give the server evidence that it is aware of the migration of the filesystem. Once the client has done this for all migrated filesystems on which the client holds state, the server MUST resume normal handling of stateful requests from that client.

One way in which clients can do this efficiently in the presence of large numbers of filesystems is described below. This approach divides the process into two phases, one devoted to finding the migrated filesystems and the second devoted to doing the necessary GETATTRs.

The client can find the migrated filesystems by building and issuing one or more COMPOUND requests, each consisting of a set of PUTFH/GETFH pairs, each pair using an fh in one of the filesystems in question. All such COMPOUND requests can be done in parallel. The successful completion of such a request indicates that none of the fs's interrogated have been migrated while termination with NFS4ERR\_MOVED indicates that the filesystem getting the error has migrated while those interrogated before it in the same COMPOUND have not. Those whose interrogation follows the error remain in an uncertain state and can be interrogated by restarting the requests from after the point at which NFS4ERR\_MOVED was returned or by issuing a new set of COMPOUND requests for the filesystems which

remain in an uncertain state.

Once the migrated filesystems have been found, all that is needed is for the client to give evidence to the server that it is aware of the migrated status of filesystems found by this process, by interrogating the `fs_locations` attribute for an `fh` within each of the migrated filesystems. The client can do this by building and issuing one or more COMPOUND requests, each of which consists of a set of PUTFH operations, each followed by a GETATTR of the `fs_locations` attribute. A RENEW follows to help tie the operations to the lease returning NFS4ERR\_LEASE\_MOVED. Once the client has done this for all migrated filesystems on which the client holds state, the server will resume normal handling of stateful requests from that client.

In order to support legacy clients that do not handle the NFS4ERR\_LEASE\_MOVED error correctly, the server SHOULD time out after a wait of at least two lease periods, at which time it will resume normal handling of stateful requests from all clients. If a client attempts to access the migrated files, the server MUST reply NFS4ERR\_MOVED.

When the client receives an NFS4ERR\_MOVED error, the client can follow the normal process to obtain the new server information (through the `fs_locations` attribute) and perform renewal of those leases on the new server. If the server has not had state transferred to it transparently, the client will receive either NFS4ERR\_STALE\_CLIENTID or NFS4ERR\_STALE\_STATEID from the new server, as described above. The client can then recover state information as it does in the event of server failure.

Aside from recovering from a migration, there are other reasons a client may wish to retrieve `fs_locations` information from a server. When a server becomes unresponsive, for example, a client may use cached `fs_locations` data to discover an alternate server hosting the same `fs` data. A client may periodically request `fs_locations` data from a server in order to keep its cache of `fs_locations` data fresh.

Since a GETATTR(`fs_locations`) operation would be used for refreshing cached `fs_locations` data, a server could mistake such a request as indicating recognition of an NFS4ERR\_LEASE\_MOVED condition. Therefore a compound which is not intended to signal that a client has recognized a migrated lease SHOULD be prefixed with a guard operation which fails with NFS4ERR\_MOVED if the file handle being queried is no longer present on the server. The guard can be as simple as a GETFH operation.

Though unlikely, it is possible that the target of such a compound could be migrated in the time after the guard operation is executed

on the server but before the GETATTR(fs\_locations) operation is encountered. When a client issues a GETATTR(fs\_locations) operation as part of a compound not intended to signal recognition of a migrated lease, it SHOULD be prepared to process fs\_locations data in the reply that shows the current location of the fs is gone.

#### 5.6.4. Migration and the Lease\_time Attribute

In order that the client may appropriately manage its leases in the case of migration, the destination server must establish proper values for the lease\_time attribute.

When state is transferred transparently, that state should include the correct value of the lease\_time attribute. The lease\_time attribute on the destination server must never be less than that on the source since this would result in premature expiration of leases granted by the source server. Upon migration in which state is transferred transparently, the client is under no obligation to re-fetch the lease\_time attribute and may continue to use the value previously fetched (on the source server).

In the case in which lease merger occurs as part of state transfer, the lease\_time attribute of the destination lease remains in effect. The client can simply renew that lease with its existing lease\_time attribute. State in the source lease is renewed at the time of transfer so that it cannot expire, as long as the destination lease is appropriately renewed.

If state has not been transferred transparently (i.e., the client need to reclaim or re-obtain its locks), the client should fetch the value of lease\_time on the new (i.e., destination) server, and use it for subsequent locking requests. However the server must respect a grace period at least as long as the lease\_time on the source server, in order to ensure that clients have ample time to reclaim their locks before potentially conflicting non-reclaimed locks are granted. The means by which the new server obtains the value of lease\_time on the old server is left to the server implementations. It is not specified by the NFS version 4.0 protocol.

### 6. Results of proposed changes for NFSv4.0

The purpose of this section is to examine the troubling results reported in Section 3.1. We will look at the scenarios as they would be handled within the proposal.

Because the choice of uniform vs. non-uniform nfs\_client\_id4 id strings is a "SHOULD" in these cases, we will designate clients that

follow this recommendation by SHOULD-UF-CID.

We will also have to take account of any merger-related "SHOULD" clauses to better understand how they have addressed the issues seen. We abbreviate as follows:

- o SHOULD-SVR-AM refers to the server obeying the SHOULD which RECOMMENDS that they merge leases with identical nfs\_client\_id4 id strings and boot verifiers.

#### 6.1. Results: Failure to free migrated state on client reboot

Let's look at the troublesome situation cited in Section 3.1.1. We have already seen what happens when SHOULD-UF-CID does not hold. Now let's look at the situation in which SHOULD-UF-CID holds, whether SHOULD-SVR-AM is in effect or not.

- o A client C establishes a clientid4 C1 with server ABC specifying an nfs\_client\_id4 with id string value "C" and boot verifier 0x111.
- o The client begins to access files in filesystem F on server ABC, resulting in generating stateids S1, S2, etc. under the lease for clientid C1. It may also access files on other filesystems on the same server.
- o The filesystem is migrated from ABC to server XYZ. When transparent state migration is in effect, stateids S1 and S2 and lease {0x111, "C", C1} are now available for use by client C at server XYZ. So far, so good.
- o Client C reboots and attempts to access data on server XYZ, whether in filesystem F or another. It does a SETCLIENTID with an nfs\_client\_id4 with id string value "C" and boot verifier 0x112. The state associated with lease {0x111, "C", C1} is deleted as part of creating {0x112, "C", C2}. No problem.

The correctness signature for this issue is

SHOULD-UF-CID

so if you have clients and servers that obey the SHOULD clauses, the problem is gone regardless of the choice on the MAY.

#### 6.2. Results: Server reboots resulting in confused lease situation

Now let's consider the scenario given in Section 3.1.2. We have already seen what happens when SHOULD-UF-CID does not hold. Now

let's look at the situation in which SHOULD-UF-CID holds and SHOULD-SVR-AM holds as well.

- o Client C talks to server ABC using an `nfs_client_id4` id string such as "C-ABC" and boot verifier `v1`. As a result a lease with `clientid4 c.i` established: `{v1, "C-ABC", c.i}`.
- o `fs_a1` migrates from server ABC to server XYZ along with its state. Now server XYZ also has a lease: `{v1, "C-ABC", c.i}`
- o Server ABC reboots.
- o Client C talks to server ABC using an `nfs_client_id4` id string such as "C-ABC" and boot verifier `v1`. As a result a lease with `clientid4 c.j` established: `{v1, "C-ABC", c.j}`.
- o `fs_a2` migrates from server ABC to server XYZ. As part of migration the incoming lease is seen to denote same `Nfs_client_id4` and so is merged with `{v1, "C-ABC", c.i}`.
- o Now server XYZ has only one lease that matches `{v1, "C-ABC", *}`, so the problem is solved

Now let's consider the same scenario in the situation in which SHOULD-UF-CID holds and SHOULD-SVR-AM holds as well.

- o Client C talks to server ABC using an `nfs_client_id4` id string "C" and boot verifier `v1`. As a result a lease with `clientid4 c.i` is established: `{v1, "C", c.i}`.
- o `fs_a1` migrates from server ABC to server XYZ along with its state. Now XYZ also has a lease: `{v1, "C", c.i}`
- o Server ABC reboots.
- o Client C talks to server ABC using an `nfs_client_id4` id string "C" and boot verifier `v1`. As a result a lease with `clientid4 c.j` is established: `{v1, "C", c.j}`.
- o `fs_a2` migrates from server ABC to server XYZ. As part of migration the incoming lease is seen to denote the same `nfs_client_id4` and so is merged with `{v1, "C", c.i}`.
- o Now server XYZ has only one lease that matches `{v1, "C", *}`, so the problem is solved

The correctness signature for this issue is

## SHOULD-SVR-AM

so if you have clients and servers that obey the SHOULD clauses, the problem is gone regardless of the choice on the MAY.

## 6.3. Results: Client complexity issues

Consider the following situation:

- o There are a set of clients C1 through Cn accessing servers S1 through Sm. Each server manages some significant number of filesystems with the filesystem count L being significantly greater than m.
- o Each client Cx will access a subset of the servers and so will have up to m clientid's, which we will call Cxy for server Sy.
- o Now assume that for load-balancing or other operational reasons, numbers of filesystems are migrated among the servers. As a result, depending on how this handled, the number of clientids may explode. See below.

Now look what will happen under various scenarios:

- o We have previously (in Section 3.1.3) looked at this in case of client following the non-uniform client-string approach. In that case, each client-server pair could have up to m clientid's and each client will have up to  $m \times 2$  clientids. If we add the possibility of server reboot, the only bound on a client's clientid count is L.
- o If we look at this in the SHOULD-UF-CID case in which the SHOULD-SVR-AM condition holds, the situation is no different. Although the server has the client identity information that could enable same-client-same-server leases to be combined, it does not do so. We still have up to L clientid's per client.
- o On the other hand, if we look at the SHOULD-UF-CID case in which SHOULD-SVR-AM holds, the problem is gone. There can be no more than m clientids per client, and n clientid's per server.

The correctness signature for this issue is

(SHOULD-UF-CID & SHOULD-SVR-AM)

so if you have clients and servers that obey the SHOULD clauses, the problem is gone regardless of the choice on the MAY.

#### 6.4. Result summary

We have seen that (SHOULD-SVR-AM & SHOULD-UF-CID) are sufficient to solve the problems people have experienced.

#### 7. Issues for NFSv4.1

Because NFSv4.1 embraces the uniform client-string approach, addressing migration issues is simpler. In the terms of Section 6, we already have SHOULD-UF-CID, for NFSv4.1, as advised by section 2.4 of [RFC5661], simplifying the work to be done.

Nevertheless, there are some issues that will have to be addressed. Some examples:

- o The other necessary part of addressing migration issues, which we call above SHOULD-SVR-AM, is not currently addressed by NFSv4.1 and changes need to be made to make it clear that state needs to be appropriately merged as part of migration, to avoid multiple clientids between a client-server pair.
- o There needs to be some clarification of how migration, and particularly transparent state migration, should interact with pNFS layouts.
- o The current discussion (in [RFC5661]), of the possibility of server\_owner changes is incomplete and confusing.

Discussion of how to resolve these issues will appear in the sections below.

##### 7.1. Addressing state merger in NFSv4.1

The existing treatment of state transfer in [RFC5661], has similar problems to that in [RFC3530] in that it assumes that the state for multiple fs's on different servers will not be merged so that it appears under a single common clientid. We've already seen the reasons that this is a problem, with regard to NFSv4.0.

Although we don't have the problems stemming from the non-uniform client-string approach, there are a number of complexities in the existing treatment of state management in the section entitled "Lock State and File System Transitions" in [RFC5661] that make this non-trivial to address:

- o Migration is currently treated together with other sorts of file system transitions including transitioning between replicas

without any NFS4ERR\_MOVED errors.

- o There is separate handling and discussion of the cases of matching and non-matching server scopes.
- o In the case of matching server scopes, the text calls for an impossible degree of transparency.
- o In the case of non-matching server scopes, the text does not mention transparent state migration at all, resulting in a functional regression from NFSV4.0

## 7.2. Addressing pNFS relationship with migration

This is made difficult because, within the PNFS framework, migration might mean any of several things:

- o Transfer of the MDS, leaving DS's alone.

This would be minimally disruptive to those using layouts but would require the pNFS control protocol to support the DS being directed to a new MDS.

- o Transfer of a DS, leaving everything else in place.

Such a transfer can be handled without using migration at all. The server can recall/revoke layouts, as appropriate.

- o Transfer of the file system to a new file system with both MDS and DS's moving.

In such a transfer, an entirely different set of DS's will be at the target location. There may even be no pNFS support on the destination FS at all.

Migration needs to support both the first and last of these models.

## 7.3. Addressing server owner changes in NFSv4.1

Section 2.10.5 of [RFC5661] states the following.

The client should be prepared for the possibility that `eir_server_owner` values may be different on subsequent `EXCHANGE_ID` requests made to the same network address, as a result of various sorts of reconfiguration events. When this happens and the changes result in the invalidation of previously valid forms of trunking, the client should cease to use those forms, either by dropping connections or by adding sessions. For a discussion of



lock reclaim as it relates to such reconfiguration events, see Section 8.4.2.1.

While this paragraph is literally true in that such reconfiguration events can happen and clients have to deal with them, it is confusing in that it can be read as suggesting that clients have to deal with them without disruption, which in general is impossible.

A clearer alternative would be:

It is always possible that, as a result of various sorts of reconfiguration events, `eir_server_scope` and `eir_server_owner` values may be different on subsequent `EXCHANGE_ID` requests made to the same network address.

In most cases such reconfiguration events will be disruptive and indicate that an IP address formerly connected to one server is now connected to an entirely different one.

Some guidelines on client handling of such situations follow:

- \* When `eir_server_scope` changes, the client has no assurance that any id's it obtained previously (e.g. file handles) can be validly used on the new server, and, even if the new server accepts them, there is no assurance that this is not due to accident. Thus it is best to treat all such state as lost/stale although a client may assume that the probability of inadvertent acceptance is low and treat this situation as within the next case.
- \* When `eir_server_scope` remains the same and `eir_server_owner.so_major_id` changes, the client can use filehandles it has and attempt reclaims. It may find that these are now stale but if `NFS4ERR_STALE` is not received, he can proceed to reclaim his opens.
- \* When `eir_server_scope` and `eir_server_owner.so_major_id` remain the same, the client has to use the now-current values of `eir_server-owner.so_minor_id` in deciding on appropriate forms of trunking.

## 8. Lock State and File System Transitions (AS PROPOSED)

In dealing with file system transitions, the client needs to handle cases in which the two servers have cooperated in state management and cases in which they have not.

The primary means by which a client finds out about state management co-operation is by comparing `eir_server_scope` values returned by each server. If the scope values do not match, then any co-operation of the servers in state management, is limited to transferring state in event of migration and making arrangements for the safe reclamation of locking state. If the scope values match, then this indicates the servers have cooperated in assigning client IDs and stateids to the point that the same id will not refer to different things on different servers. Servers may reject client IDs that refer to state they do not know about. See the section entitled "Server Scope" for more information about the use of server scope.

How the client needs to deal with locking state with regard to these situations will depend upon:

- o The type of file system transition occurring.
- o The type of state involved (e.g. layout state may sometimes be handled differently).
- o The specific level of state handling co-ordination between the two servers for the specific transition.

We will divide the basic description of these possibilities into three sections

- o In Section 8.1, we will discuss handling specific to the case of matching server scopes.
- o In Section 8.2, we will discuss handling specific to the case of non-matching server scopes.
- o In Section 8.3, we will discuss issues relating to handling common to both cases.

#### 8.1. File System Transitions with Matching Server Scopes

In the case of migration, the servers involved in the migration of a file system SHOULD transfer all server state relevant to the migrating file system from the original to the new server. When this is done, it needs to be done in a way that is maximally transparent to the client in that all stateids used by the client to access state on the filesystem in question can be used on the new server, albeit possibly under different client IDs.

When layouts are active for a migrated file system, layout state SHOULD be included as part of the state transferred. Even if it is the case that there are circumstances preventing the layout from

being supported on the new server, this should be dealt with by recalling layouts either before or after the transition. Where this cannot be done, layout revocation is possible but any such revocation should appear to the client just as any other layout revocation would.

With replication, such a degree of common state is typically not the case. Clients, however, should use the information provided by the `eir_server_scope` returned by `EXCHANGE_ID` (as modified by the validation procedures described in the section entitled "Server Scope") to determine whether such sharing may be in effect in non-migration cases, rather than making assumptions based solely on the reason for the transition.

This state transfer will reduce disruption to the client when a file system transition occurs. If the servers are successful in transferring all state, the client can access existing stateids, using either existing or new sessions between the client and the new server instance. If the server accepts such a transferred stateid as valid, then the client may use that stateid to access the same state that it represented on the old server.

When the two servers belong to the same server scope, it does not mean that when dealing with the transition, the client will not have to reclaim or otherwise reobtain state. However, it does mean that the client may proceed using its current stateids when communicating with the new server, and the new server will either recognize the stateids as valid or reject them, in which case locking state must be reobtained by the client.

File systems cooperating in state management may actually share state or simply divide the identifier space so as to recognize (and reject as stale) each other's stateids and client IDs. Servers that do share state may not do so under all conditions or at all times. If the server cannot be sure when accepting a stateid that it reflects the locks the client was given, the server must treat the state as stale and report it as such to the client.

## 8.2. File System Transitions with Non-Matching Server Scopes

When the two file system instances are on servers that do not share a server scope value, the client must establish a new client ID on the destination, if it does not have one already, to obtain access to its locks. Depending on the type of file system transition and facilities provided by the server, it may re-establish its connection to locking and layout state in a number of ways.

In the case of migration, the servers may have transferred stateids,

making it possible for the client to access his state on the new server, simply by using the existing stateid. The server may transfer all state or a subset and the client can use TEST\_STATEID to determine what state has been transferred and what needs to be reclaimed or otherwise reobtained as described in Section 8.3.

Lock reclaim may be used by the client for any sort of file system transition, but the server is not required to support it in any particular case.

Note that in this case, lock reclaim may be attempted even when the servers involved in the transfer have different server scope values (see Section 8.4.2.1 for the contrary case of reclaim after server reboot). Servers with different server scope values may cooperate to allow reclaim for locks associated with the transfer of a file system even if they do not cooperate sufficiently to share a server scope.

### 8.3. FS Transitions Involving Reobtaining Locking State

In either case, when actual locks are not known to be maintained, the destination server may establish a grace period specific to the given file system, with non-reclaim locks being rejected for that file system, even though normal locks are being granted for other file systems. Clients should not infer the absence of a grace period for file systems being transitioned to a server from responses to requests for other file systems.

In the case of lock reclamation for a given file system after a file system transition, edge conditions can arise similar to those for reclaim after server restart (although in the case of the planned state transfer associated with migration, these can be avoided by securely recording lock state as part of state migration). Unless the destination server can guarantee that locks will not be incorrectly granted, the destination server should not allow lock reclaims and should avoid establishing a grace period.

Once all locks have been reclaimed, or there were no locks to reclaim, the client indicates that there are no more reclaims to be done for the file system in question by sending a RECLAIM\_COMPLETE operation with the rca\_one\_fs parameter set to true. Once this has been done, non-reclaim locking operations may be done, and any subsequent request to do a reclaim will be rejected with the error NFS4ERR\_NO\_GRACE.

Information about client identity may be propagated between servers in the form of a client\_owner4 and associated verifiers, under the assumption that the client presents the same values to all the servers with which it deals.

Servers are encouraged to provide facilities to allow locks to be reclaimed on the new server after a file system transition. Often, however, in cases in which the two servers do not share a server scope value, such facilities may not be available and the client should be prepared to re-obtain locks, even though it is possible that the client may have its LOCK or OPEN request denied due to a conflicting lock.

Layouts may be reobtained when necessary even without special facilities for lock reclamation. However, the client **MUST NOT** depend on being able to obtain such layout since pNFS or the desired mapping type might not be supported on the new server.

The consequences of having no facilities available to reclaim locks on the new server will depend on the type of environment. In some environments, such as the transition between read-only file systems, such denial of locks should not pose large difficulties in practice. When an attempt to re-establish a lock on a new server is denied, the client should treat the situation as if its original lock had been revoked. Note that when the lock is granted, the client cannot assume that no conflicting lock could have been granted in the interim. Where change attribute continuity is present, the client may check the change attribute to check for unwanted file modifications. Where even this is not available, and the file system is not read-only, a client may reasonably treat all pending locks as having been revoked.

## 9. Security Considerations

The current definitive definition of the NFSv4.0 protocol [RFC3530], and the current pending draft of RFC3530bis [cur-v4.0-bis] both agree. The section entitled "Security Considerations" encourages that clients protect the integrity of the SECINFO operation, any GETATTR operation for the fs\_locations attribute, and the operations SETCLIENTID/SETCLIENTID\_CONFIRM. A migration recovery event can use any or all of these operations. We do not recommend any change here.

## 10. IANA Considerations

This document does not require actions by IANA.

## 11. Acknowledgements

The editor and authors of this document gratefully acknowledge the contributions of Trond Myklebust of NetApp and Robert Thurlow of

Oracle. We also thank Tom Haynes of NetApp and Spencer Shepler of Microsoft for their guidance and suggestions.

Special thanks go to members of the Oracle Solaris NFS team, especially Rick Mesta and James Wahlig, for their work implementing an NFSv4.0 migration prototype and identifying many of the issues documented here.

## 12. References

### 12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.
- [RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

### 12.2. Informative References

- [cur-v4.0-bis] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", 2011, <<http://www.ietf.org/id/draft-ietf-nfsv4-rfc3530bis-18.txt>>.

Work in progress.

## Authors' Addresses

David Noveck (editor)  
EMC Corporation  
228 South Street  
Hopkinton, MA 01748  
US

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

Piyush Shivam  
Oracle Corporation  
5300 Riata Park Ct.  
Austin, TX 78727  
US

Phone: +1 512 401 1019  
Email: piyush.shivam@oracle.com

Charles Lever  
Oracle Corporation  
1015 Granger Avenue  
Ann Arbor, MI 48104  
US

Phone: +1 248 614 5091  
Email: chuck.lever@oracle.com

Bill Baker  
Oracle Corporation  
5300 Riata Park Ct.  
Austin, TX 78727  
US

Phone: +1 512 401 1081  
Email: bill.baker@oracle.com

