Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: October 9, 2011

Dipankar Roy
Mike Eisler
Alex RN
NetApp

April 7, 2011

NFS Pathless Objects
draft-dipankar-nfsv4-pathless-objects-02.txt

Abstract

This document describes a set of NFS operations for creating, maintaining and searching filesystem objects independent of the traditional hierarchical namespace.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 October 9, 2011.

Copyright Notice

Table of Contents

1. Terminology

NFS: Used to refer to Network File System irrespective of the version.

NFSv3: Network File System Version 3

NFSv4: Network File System Version 4

NFSv4.1: Network File System Version 4.1

ACL: Access Control List

HTTP: Hyper Text Transfer Protocol

REST: Representational State Transfer

2. Introduction

The NFS protocol is presently capable of interacting with objects which can be represented by a pathname and a filehandle, residing in a hierarchical namespace exported by the NFS server. However, such a hierarchical namespace which tries to resemble the UNIX filesystem layout and interface imposes restrictions on the filesystem object locations and does not scale well in the case we need to store billions of files inside a flat directory structure.

The rapidly developing distributed web applications of today, such as those implementing the HTTP REST protocol, need to store billions of objects, which do not need any directory hierarchy and instead must have the capability to specify custom object attributes and quickly search for the objects based on these attributes. To facilitate this, an object needs to become independent of the filesystem directory hierarchy that has been mandated by the NFS server until now. In other words, the requirement is to have filesystem objects which can be created, queried for and destroyed without being associated with a pathname. These objects do not need to become visible under a standard NFS exported hierarchical pathname but need to be looked up based on tags or custom attributes. This RFC presents the operations that are required by the NFS protocol to implement such a feature of pathless filesystem objects.

Separation of the pathname from the filesystem object provides the implementation greater flexibility on where to store the object, which can lead to an optimal distribution of the filesytem objects based on application requirements. Such an filesystem object is looked up by the client based on the attributes of the object, rather

than it's location in a directory, and is accessed by it's NFS filehandle directly. This new object, though it does not have a pathname, can still optionally have a name, which MAY be implemented as an attribute for the object. The existing set of NFS attributes needs to be extended to support a model where lookup of filesystem objects can be done based on attributes.

With the introduction of NFSv4, the NFS protocol enforces statefulness for interacting with filesystem objects. There are many applications which require the stateful model of NFSv4. But there are also many web oriented object stores, which can be simultaneously accessed over other stateless protocols such as http, ftp etc. and hence are not very interested in statefulness. Rather, they would like to have the flexibility of using the stateless nature of NFSv3 along with some interesting features of NFSv4 such as ACLs, Named Attributes, Compound Operations etc. which do not depend on the statefulness of the NFS server for functionality. Stateless operations provide the benefit of better performance as functional and maintenance costs for the implementations are significantly less. The object stores which will potentially handle billions of objects and have no need for state maintenance can greatly benefit from the improved performance of a stateless NFS implementation.

In light of the above, anonymous states are RECOMMENDED to be used with this RFC, which means a stateid of all zeroes SHOULD be used for NFSv4 and NFSv4.1 READ, WRITE and SETATTR operations on pathless objects. At the same time, also keeping in mind the requirements of applications which need to maintain locks at the server, advisory locking SHOULD be supported. Delegations and shared locking support is OPTIONAL with the implementation of this RFC.

3. Protocol Overview

3.1. Pathless Objects and Object Sets

To create an object independent of a pathname, the client sends a request to the server to create the object without specifying any name or pathname and the server returns the NFS filehandle for the object thus created. A new object type called NF4NOPATHOBJ is defined in this RFC, which SHOULD be used to create pathless objects. Since pathless objects cannot be looked up based on pathname, a new type of attribute, called Search Attribute is defined in this RFC, which SHOULD be used to lookup the pathless objects. The NFSv4.1 READ and WRITE operations SHOULD be used to perform I/O on pathless objects and attributes, including search attributes, can be set using SETATTR and retrieved using GETATTR operations. A pathless object SHOULD support the mandatory set of attributes defined in RFC 5661

for a filesystem object.

Along with creating objects which are independent of pathnames, it is REQUIRED to have a mechanism to classify together such objects, for accessibility and scoped access resolution. This RFC uses the term "Object Set" for representing such a collection of objects. An Object Set works as a container for pathless objects and MUST be defined and created before a pathless object is created. It is analogous to a directory, where the object is analogous to a file inside the directory. An Object Set contains objects and MAY OPTIONALLY also contain other Object Sets.

When the client has created an Object Set or has access to an existing Object Set within the server, it can create pathless objects that are contained in the Object Set. The pathless object, even though it is contained in a Set, can be physically located anywhere. It is not necessary to implement a pathless object as a file. It can be any physical entity, which has a unique identifier in the form of a NFS filehandle. So the filehandle serves as an unique identifier for the object and there is no requirement that it SHOULD represent a file. The server is REQUIRED to maintain the linkage between the object and it's Set and is free to distribute and store the objects in the best possible way to satisfy the needs of the application.

An Object Set is expected to have certain access primitives associated with it, which are used by the server to provide access control for the objects contained in the Set. The server can implement a policy based mechanism to grant specific clients or groups of clients access to an Object Set. Such an implementation can be analogous to the exports mechanism commonly used with the NFS exported directories. The NFS server MUST keep track of all the Object Sets it has. The server SHOULD make visible all the exported Object Sets to the clients, subject to access control policies at the server. This RFC does not pose any other requirements on the implementation of an access policy for an Object Set.

An Object Set MUST have a name, which MAY be implemented as an attribute of the Set. However, unlike the name of a pathless object, the name of an Object Set MUST be unique for the NFS server. When the client first creates an Object Set, it MUST specify a name for the Object Set. The server returns a filehandle for the Object Set to the client.

3.2. Object Root Filehandle

The NFS server supporting the Object Set and pathless object creation MUST also have a well known public filehandle, hereby named as "Object Root Filehandle", in short form objrootfh. This public

filehandle is required for the purpose of informing the client that the server implements support for this RFC.The client SHOULD send an operation named PUTOBJROOTFH to set the current filehandle to objrootfh. The server which implements this RFC, MUST set the current filehandle to objrootfh and return NFS4_OK. This objrootfh filehandle SHOULD be different from the public filehandle that an NFSv4 server supports under the PUTROOTFH operation. The Object Root Filehandle serves as a master container for all the Object Sets. The client can send a query to the server to list all the Object Sets that are available to it for access under the Object Root Filehandle. Such a query can be specified as {PUTOBJROOTFH, READDIR}, where the requested attributes specified in the READDIR request would indicate if the server should reply with the Object Sets under the current filehandle.

3.3. Optional Features

Since the pathless objects MAY not be implemented as files, and this RFC RECOMMENDS stateless operation as much as possible, the following features are explicitly being made OPTIONAL:

1. Supporting the POSIX semantics for interaction with pathless objects.

2. Specifying a name to create a pathless object. Note that Object Sets MUST have unique names.

3. Support for either Exclusive Create or Soft and Hard links.

4. Support for non-regular filesystem objects such as device files.

5. Support for delegations and share locks

If links are implemented, it SHOULD link to the object based on the object name attribute, rather than a pathname. So it is a matter of having multiple values for the attribute name, which is already a feature for the search attributes.

3.4. Interaction with stateful NFS operations

The pathless objects are RECOMMENDED to be stateless. As such, the anonymous stateid of zero SHOULD be used for operations like READ, WRITE, SETATTR etc. However, if a server wants to implement stateful NFSv4.1 operations with pathless objects, it can do so given that it conforms to the specifications of NFSv4.1 RFC. So, existing NFSv4 READ, WRITE operations will work with pathless objects without any changes to the operation definitions as stated in NFSv4.1 RFC. The NFSv4.1 locking model is applicable to pathless objects, but only

advisory locking MUST be supported. But if a server decides to implement support for share locks and delegations, it MUST follow the NFSv4.1 RFC locking semantics.

4. New file types

The pathless objects are not necessarily files or any other filesystem object that can be defined with the existing nfs_fype4 type as specified in RFC 5661. The same holds for Object Sets. So two new types are being introduced with this RFC, namely NF4NOPATH and NF4OBJSET. So the definition of nfs_ftype4 is changed to include the new file types and is as follows:

```
enum nfs_ftype4 {
    NF4REG       = 0x1;
    NF4DIR       = 0x2;
    NF4BLK       = 0x3;
    NF4CHR       = 0x4;
    NF4LNK       = 0x5;
    NF4SOCK      = 0x6;
    NF4ATTRDIR   = 0x7;
    NF4NAMEDATTR = 0x8;
    NF4NOPATHOBJ = 0x9;
    NF4OBJSET    = 0x10;
}
```

5. Search Attributes

5.1. Search Attributes Definition

In a hierarchical filesystem, the NFS client can do LOOKUP operations based on pathname for a filesystem object but this will not work in the case of pathless objects. So with pathless objects, the server SHOULD support some kind of attributes which can be used to search for such objects. These attributes are hereby called "Search Attributes". These attributes have a name and a list of values. The values can be of type integer or string. Search attributes can be combined to form a query which looks up objects matching the attributes specified in the query, as per the query semantics.

Two new attributes are added to the existing set of RECOMMENDED attributes for NFSv4.1. One is a boolean attribute called sattrsupport and the other is an array of strings called srchattrlist. The sattrsupport denotes whether the server supports search attributes. The srchattrlist contains the search attributes.

The GETATTR and SETATTR operations can be used to retrieve and set the search attributes. The sattrsupport applies to the filesystem and the srchattrlist applies to an object in that filesystem.

The basic data types used in this RFC are same as the data types defined in RFC 5661 Section 3.2. Some new structured data types are added in this section to define the Search Attributes. One is a type specifier for the Search Attribute value i. whether it is a number or a string and is defined as "svaltype". The "sval" represents a single value for a Search Attribute, of type svaltype. The "svalist" is a set of such values for the Search Attribute. The Search Attribute itself is defined as "srchattr" and contains a name of type component4, the svaltype and svalist. A collection of Search Attributes is defined as srchattrlist.

Name	Id	Data Type	Acc
sattrsupport	75	bool	R
srchattrlist	76	srchattr<>	R W

```
bool sattrsupport; /* indicates search attributes are supported */

enum svaltype {
    SVAL_TYPE_NUM = 0; /* Search Attribute value is a number */
    SVAL_TYPE_STR = 1; /* Search Attribute value is a string */
};

/* single search attribute value */
union sval switch (svaltype type) {
    case SVAL_TYPE_NUM:
        int64_t svalnum;
    case SVAL_TYPE_STR:
        component4 svalstr;
    default:
        void;
};

typedef struct sval svalist<>; /* array of attribute values */

struct srchattr {
    component4  srchattrname; /* name of the search attribute */
    svaltype    type;         /* type of the search attribute */
    svalist     srchvalist;   /* list of values for this attr */
};

typedef struct srchattr srchattrlist<>;
```

5.2. Search Attributes usage

Since there is no pathname, we cannot use the NFSv4 LOOKUP operation to lookup a pathless object. Instead, the new search attributes are used to look up a pathless object. These are represented as name value pairs where the name is a string and the value is an array of numbers or strings. A search attribute can have multiple values for the same object. A search can be done for one or more of these values. For example, a pathless object can have the attribute name as "weather" and values can be "sunny", "cloudy", "rainy" etc. If there are no values specified for a search attribute, a search is made for the objects having the search attribute, with or without any values.

Access control for a search attribute is governed by the access control for the corresponding object. The permissions to read or write the object's attributes apply to search attributes as well.

To lookup pathless objects, the client sends a list of the search attributes. A new operation PUTSRCHATTR is added to lookup objects based on search attributes. The search attributes as well as the operations are applicable to both pathless objects and Object Sets. To retrieve or set search attributes, GETATTR and SETATTR are used.

The PUTSRCHATTR operation MUST be used in conjunction with the READDIR operation to make use of the features provided by the READDIR operation, namely, a reply cursor, requested set of object attributes and maximum count of bytes in the reply. The PUTSRCHATTR operation MUST be immediately followed by a READDIR operation in the same COMPOUND operation to this effect. The client MUST request the object filehandles in the bitmap for requested attributes in the READDIR request. The READDIR reply contains the filehandles of all the objects matching the search attributes specified in PUTSRCHATTR.

A special search attribute with srchattrname as "objname" of type SVAL_TYPE_STR MUST always be present for a pathless object and denotes the name that the object was created with. For Object Sets, this should have a unique value in the NFS server. For pathless objects it defaults to an empty string i.e. "".

5.3. Search Attributes Query

The Search Attributes can be queried using the semantics defined in this section. A simple query is based on the Search Attribute name and on whether the Search Attribute matches a set of Search Attribute values. The match can be based on whether the Search Attribute name has a value that equals the value specified in the query or the match can also be based on whether the Search Attribute has a value lesser

than or greater than the value in the query. The lesser than and greater than comparisons are more effective when the Search Attribute has a svaltype of a number. Such queries can be logically joined or chained together using logical primitives such as AND and OR. A NOT primitive is also present to provide a logical negation of the query, which will match those objects that do not match the Search Attribute values specified in the query. Each query has a priority assigned to it and queries with higher priority will execute earlier. For example, let's say that there are 3 queries, which are joined like this: query 1 AND query 2 OR query 3". Suppose query 2 and query 3 have a higher priority than query 1. So the server would execute this query like this : query 1 AND (query 2 OR query 3), where brackets indicate a logical grouping and execution of queries of the same priority.

Some new structured data types are added in this section to define a Search Attribute query. The type "srelation" determines what is the nature of the match being done for the Search Attribute and it's values i.e. whether a match is done for equals, lesser than or greater than. The type "srchqueryjointype" specifies how two queries can be logically chained together or if a query needs to be logically negated. A Search Attribute query is defined as a combination of:

1. A srchattrlist, as defined in the section 5.1 of this RFC

2. A srelation, which specifies how the match for the Search Attributes inside the srachattrlist and their corresponing values are done.

3. A srchqueryjoinype called sqjtypenext which specified how the query should be chained with the next query. The last query in a chain of queries MUST have this set to SQUERY_NONE.

4. A priority which determines an ordering of the queries. A priority of 0 SHOULD be considered as the highest priority, followed by 1 and so on.

5. A flag which tells if the query is a logical NOT. A value of 1 for this flag SHOULD be interpreted as a logical NOT.

```
enum srelation {
    SRELN_EQUALS  = 0;
    SRELN_GREATER = 1;
    SRELN_LESSER  = 2;
};

enum srchqueryjointype {
    SQUERY_NONE   = 0;
    SQUERY_AND    = 1;
    SQUERY_OR     = 2;
};

struct srchquery {
    srchattrlist     search_attrs;
    srelation        search_relation;
    srchqueryjoinype sqjtypenext;
    uint32_t         priority;
    uint32_t         flag;
};

typedef struct srchquery srchquerylist<>;
```

The flag in srchquery denotes whether the query is a NOT. A value of 1 for the flag means it's a NOT. For example, if a seach attribute list i.e. srchattrlist has 2 search attributes (A and B), each with multiple values, and if the flag has a value of 1, it can be described as NOT ((srchattrnameA (search_relation i.e. EQ, LT, GT) srchattrvalues) && (srchattrnameB (search_relation i.e. EQ, LT, GT) srchattrvalues))

The priority in srchquery determines the precedance. For example, in the searchquery ((A == B) AND (C == D)) OR (F == G), we want A==B and C==D computed before F==G. So assign equal priorities to query 1 and query 2, i.e. A==B and C==D and then assign a lower priority to query 3 i.e. F==G. Similarly to effect the query (A == B) AND ((C == D)) OR (F == G)), query 2 and query 3 are assigned a higher priority than query 1. Since each query has a priority number, we can group queries that need to be executed in the same priority bucket, the same priority number. If the precedance is expressed in the form of brackets, then the priority is directly proportional to the number of brackets enclosing a query.

NOT combined with Greater gives us Lesser than or equal to. Similarly NOT combined with Lesser than gives us Greater than or equal to. So combining NOT with the search_relation gives the flexibility to specify these special relations in a query.

6. New Operations

6.1. Operation 1: PUTOBJROOTFH - Set Object Root Filehandle

SYNOPSIS

```
- -> (cfh)
```

ARGUMENT

```
void;
```

RESULT

```
struct PUTOBJROOTFHres {
       /* CURRENT_FH: objrootfh */
       nfsstat4        status;
};
```

DESCRIPTION

Replaces the current filehandle with the filehandle that represents the root of all the Objects Sets that the server contains.

IMPLEMENTATION

This is the first operator in a NFS request to set the context for the following operations. A READDIR following a PUTOBJROOTFH SHOULD list all the Object Sets with respect to this filehandle. The READDIR operation SHOULD list only Object Sets and not individual pathless objects.

ERRORS

```
NFS4ERR_BADSESSION
NFS4ERR_DELAY
NFS4ERR_NOTSUPP
NFS4ERR_RESOURCE
NFS4ERR_SERVERFAULT
```

6.2. Operation 2: PUTSRCHATTR: Search for an Object based on Search Attributes

SYNOPSIS

(cfh), srchquerylist, -> (cfh)

ARGUMENT

```
struct PUTSRCHATTRargs {
       /* CURRENT_FH: Object Set filehandle */
       srchquerylist    search_query;
};
```

RESULT

```
struct PUTSRCHATTRres {
       /* CURRENT_FH:  Object Set filehandle */
       nfsstat4        status;
};
```

DESCRIPTION

The PUTSRCHATTR operation searches for objects matching the search attributes. The scope is the Object Set as specified in the current filehandle (cfh). Instead of returning the matching filehandles, it just returns a status and uses READDIR operation's reply to construct the proper reply. The READDIR reply is used to return a variable list of filehandles of all the objects that matches the search query. The READDIR reply contains the filehandles for the matching objects in the set of attributes for the object. The object name is an empty string by default for pathless objects. A PUTSRCHATTR in a Compound request MUST be followed by a READDIR. If the PUTSRCHATTR is not followed by a READDIR, then NFS4ERR_OP_ILLEGAL MUST be returned by the NFS server.

IMPLEMENTATION

The compound operation for implementing the lookup based on search attributes is like this:
PUTFH (filehandle of object set), PUTSRCHATTR (search attributes), READDIR (cookie, verifier, dircount, maxcount, requested_attrs). It is RECOMMENDED that dircount be set to a value for zero for this sequence of operations as clients are not supposed to implement "ls" based on search attribute lookup.

ERRORS

NFS4ERR_ACCESS
NFS4ERR_ATTRNOTSUPP

NFS4ERR_BADCHAR
NFS4ERR_BADNAME
NFS4ERR_BADSESSION
NFS4ERR_BADXDR
NFS4ERR_DELAY
NFS4ERR_INVAL
NFS4ERR_IO
NFS4ERR_NAMETOOLONG
NFS4ERR_NOENT
NFS4ERR_NOFILEHANDLE
NFS4ERR_NOTSUPP
NFS4ERR_REP_TOO_BIG
NFS4ERR_RESOURCE
NFS4ERR_SERVERFAULT
NFS4ERR_STALE
NFS4ERR_WRONG_TYPE

7. Modifications to existing NFSv4.1 operations

7.1. CREATE: Modifications

Object Sets MUST be created with the CREATE call. Pathless objects are RECOMMENDED to be created with the CREATE call, though they can also be created with the OPEN call. For an Object Set, an unique objname is MANDATORY. For a pathless object, objname can be an empty string, namely, "". In case any other objname is supplied with the CREATE call for a pathless object, it MAY be allowed. The objname SHOULD become part of the search attributes for the pathless object or the Object Set.

7.2. OPEN: Modifications

OPEN with a empty objname SHOULD create a pathless object under the current filehandle. The current filehandle MUST be the filehandle for an Object Set.

7.3. LOOKUP: Modifications

Since pathless objects can have a name associated with them, LOOKUP of an objname under the current filehandle of an Object Set can return a filehandle which maps to the name. However, there can be multiple objects which map to the same name and in that case, it MAY not be correct to return the name of any one of them. So if an objname maps to a single object filehandle, the LOOKUP operation MAY return that filehandle. Otherwise, it SHOULD return NFS4ERR_WRONG_TYPE.

7.4. READDIR: Modifications

READDIR MUST be used immediately following a PUTSRCHATTR to lookup pathless objects. If the pathless objects do not have unique names READDIR will return empty names. If a READDIR operation is used standalone with current filehandle being set to the Object Set filehandle, the client MUST request filehandles in the requested set of attributes and the server SHOULD return filehandles for all the pathless objects in the Object Set. The READDIR following a PUTSRCHATTR MUST be used only to return the filehandles and attributes for the objects matching the query in PUTSRCHATTR. Any other intended use of READDIR following a PUTSRCHATTR SHOULD NOT be implemented.

8. Migration and Replication

The RFC 5661 specifies the attributes fs_locations and fs_locations_info that can be used for migration and replication. For details, please refer to sections 11.9 and 11.10 of RFC 5661. Pathless objects can use the same attributes for migration and replication with some minor modifications. The Object Sets which act as containers for pathless objects are similar to the root path of a filesystem within a server. Hence, for pathless objects, "rootpath" and "fs-root" in fs_location4 SHOULD be Object Set names. Similarly the "fli_rootpath" and "fli_fs_root" for fs_locations_info4 SHOULD contain Object Set names.

9. Acknowledgements

The authors would like to acknowledge Manjunath Shankararao for reviews of the various early versions of the draft. Thomas Haynes and Daniel Muntz have provided additional comments.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

All considerations from RFC 3530 Section 16 [RFC3530]

12. References

12.1. Normative References

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

[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.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

[RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009.

[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

[RFC1094] Nowicki, B., "NFS: Network File System Protocol specification", RFC 1094, March 1989.

[RFC2624] Shepler, S., "NFS Version 4 Design Considerations", RFC 2624, June 1999.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

Authors' Addresses

Dipankar Roy
NetApp
495 East Java Drive
Sunnyvale, CA 94089
USA

Phone: +1-408-822-4931
Email: dipankar@netapp.com

Mike Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
USA

Phone: +1-719-599-9026
Email: mike@eisler.com

Alex RN
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Phone: +91-80-41843352
Email: rnalex@netapp.com