Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 6, 2009

D. Ellard
C. Everhart
NetApp, Inc.
R. Tewari
M. Naik
IBM Almaden

August 5, 2008

NSDB Protocol for Federated Filesystems
draft-tewari-nfsv4-federated-fs-protocol-03.txt

Status of this Memo

Copyright Notice

Abstract

This document describes a file system federation protocol that enables file access and namespace traversal across collections of independently administered fileservers. The protocol specifies a set of interfaces by which fileservers and collections of fileservers with different administrators can form a fileserver federation that provides a namespace composed of the filesystems physically hosted on and exported by the constituent fileservers.

Table of Contents

1. Requirements notation 4
2. Introduction 5
2.1. Protocol Goals 5
3. Overview of Features and Concepts 7
3.1. Namespace 7
3.2. Fileset 7
3.3. Fileset Location (FSL) 8
3.3.1. Mutual Consistency across Fileset Locations 9
3.4. Namespace Repository (NSDB) 9
3.5. Mount Points, Junctions and Referrals 10
3.6. Federation Root FileServers 11
3.7. Federation Root FileSet 11
3.8. Fileservers 11
3.9. File-access Clients 11
4. Interaction with NFSv4 12
5. Finding the local NSDB 13
6. Examples 14
6.1. Create a Fileset and its FSL(s) 14
6.1.1. Creating a Fileset and a FSN 14
6.1.2. Adding a Replica of a Fileset 15
6.2. Junction Resolution 15
6.3. Example use case for fileset annotations 16
7. Error Definitions 17
8. Mapping the NSDB onto LDAP 19
8.1. Basic LDAP Configuration 19
8.2. LDAP Attributes 19
8.2.1. fedfsUuid 19
8.2.2. fedfsNetAddr 20
8.2.3. fsnUuid 20
8.2.4. nsdbName 20
8.2.5. fslHost 20
8.2.6. fslPath 20
8.2.7. annotation 21
8.2.8. descr 21
8.2.9. fslUuid 21

8.2.10. junctionKey 21
8.2.11. childFsnUuid 21
8.2.12. childNsdbName 22
8.3. LDAP Objects 22
8.3.1. FsnObject 22
8.3.2. FslObject 22
8.3.3. JunctionObject 22
9. NSDB Protocol Operations 24
9.1. Administrative NSDB Operations 24
9.1.1. Creating an FSN 25
9.1.2. Deleting an FSN 26
9.1.3. Mount an FSN 26
9.1.4. Unmount an FSN 27
9.1.5. Create an FSL 28
9.1.6. Delete an FSL 28
9.1.7. Update an FSL 29
9.1.8. Examining an FSL 29
9.1.9. Finding the children FSNs of a fileset 29
9.2. Fileserver to NSDB Operations 30
9.2.1. Looking up FSLs for an FSN 30
10. Security Considerations 31
11. IANA Requirements 32
12. Conclusions 33
13. Glossary 34
14. Normative References 37
Authors' Addresses 38
Intellectual Property and Copyright Statements 39

1. Requirements notation

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].

2. Introduction

A federated filesystem enables file access and namespace traversal in a uniform, secure and consistent manner across multiple independent fileservers within an enterprise (and possibly across multiple enterprises) with reasonably good performance.

The first requirement of a federated filesystem is the ability to traverse the data exported by different fileservers without requiring a static client configuration. The second requirement is that the location of the data should be dynamically discovered and the discovery process should be transparent to the clients. The third requirement is that it should be possible for all clients, with sufficient privilege, to view the same namespace regardless of the fileserver they connect to.

Traditionally, fileserver collections are administered by a single entity. Fileservers may provide proprietary management tools and in some cases an administrator may be able to use the proprietary tools to build a shared namespace out of the exported filesystems. Relying on vendor-proprietary tools does not work in larger enterprises or when collaborating across enterprises because it is likely that the system will contain fileservers running different software, each with their own interfaces, with no common protocol to manage the namespace or exchange namespace information. There may also be independently-administered singleton servers that export some or all of their filesystem resources. A filesystem federation protocol enables the interoperation across multi-vendor fileservers managed by the same administrative entity, across singleton independent fileservers, and across independent administrative entities that may manage a collection of fileservers. The scope of the filesystem federation protocol is limited to NFSv4 capable fileservers. The support for NFSv3 fileservers is optional.

2.1. Protocol Goals

The objective of this draft is to specify a set of interfaces by which fileservers and collections of fileservers with different administrators can form a fileserver federation that provides a namespace composed of the filesystems physically hosted on and exported by the fileservers of the federation. It should be possible, using a system that implements the interfaces, to share a common namespace across all the fileservers in the federation. It should also be possible for different fileservers in the federation to project different namespaces and enable clients to traverse them. Such a federation may contain an arbitrary number of namespace repositories, each belonging to a different administrative entity, and each rendering a part of the namespace. Such a federation may

also have an arbitrary number of administrative entities responsible for administering disjoint subsets of the fileservers. In the rest of the document the term fileserver implies a fileserver that is part of the federation. A fileserver not part of the federation is called an external fileserver.

3. Overview of Features and Concepts

3.1. Namespace

The goal of a unified namespace is to make all managed data available to all clients via the same path in a common filesystem-like namespace. This should be achieved with minimal or zero client configuration. In particular, updates to the common namespace should not require configuration changes at the client. Filesets, which are the unit of data management, are a set of files and directories accessible from a single mount. Depending on the implementation, they may be anything between an individual directory of an exported filesystem to an entire exported filesystem at a fileserver. From the perspective of the clients, the common namespace is constructed by logically mounting filesets that are physically located on different fileservers. The namespace, which is defined in terms of fileset definitions, fileset identifiers, the location of each fileset in the namespace, and the physical location of the implementation(s) of each fileset, is stored in a set of namespace repositories, each managed by an administrative entity. The namespace schema defines the model used for populating, modifying, and querying the namespace repositories. It is not required by the federation that the namespace be common across all fileservers. It should be possible to have several independently rooted namespaces that should permit traversal into another namespace at defined junction points.

3.2. Fileset

A fileset is defined to be a container of data and is the basic unit of data management. It is uniquely represented by the fileset name (FSN). An FSN is considered unique across the federation. An FSN contains information sufficient to locate the namespace repository (NSDB) that holds authoritative information about it and an identifier, called fsn_uuid, that identifies it on that NSDB. After an FSN is created, it is associated with a fileset location (FSL) on a fileserver. A fileset can be implemented by one or more FSLs. The attributes of an FSN are:

NsdbName: the fully qualified domain name of an NSDB location that contains authoritative information for this FSN.

FsnUuid: a 128-bit UUID (universally unique identifier), conforming to [RFC4122], that is used to uniquely identify an FSN.

3.3. Fileset Location (FSL)

An FSL represents the location where the fileset data resides. Each FSL maps to a host:path pair on a file server. An FSL may also have additional attributes. Each location has an associated type that determines the protocol(s) that may be used to access its data. Type information can be used to decide the list of locations that will be returned to the client. It also has associated status information. Other attributes associated with an FSL are based on the NFSv4.1 fs_locations_info attribute[RFCTBD].

```
    struct FSL {
       utf8string    host_fqdn;
       utf8string    pathname;
       FSL_ATTR      attrs;
    };
```

Each FSL consists of:

host_fqdn: the name of the host fileserver storing the physical data

pathname: the exported pathname at that host fileserver

attrs: additional attributes for this FSL, as described in the FSL_ATTR structure

```
    struct FSL_ATTR {
        protocol_t   type;
        int32_t      currency;
        annotation_t annotations<>;
        fs_status_t  status;
        opaque_t     info<>;
    }
```

The attributes associated with each FSL are:

type: the protocol(s) supported by the fileserver hosting this FSL

currency: the time lag of this FSL represented as the number of time units it lags the latest version as defined by the NFSv4.1 fs_locations_info attribute. A currency value of 0 represents the latest version. Currency values are less than or equal to zero

annotations: a list of name/value pairs that can be interpreted by an individual NSDB. The semantics of the name/value pair is not defined by this protocol and is intended to be used by higher-level administration protocols

status: fls_status as defined by the NFSv4.1 status attribute

info: as defined in NFSv4.1 fs_locations_info attribute

3.3.1. Mutual Consistency across Fileset Locations

All of the FSLs that have the same FSN (thereby reference the same fileset) are equivalent from the point of view of client access; the different locations of a fileset represent the same data, though potentially at different points in time. Fileset locations are equivalent but not identical. Locations may either be read- only or read-write. Typically, multiple read-write locations are backed by a clustered filesystem while read-only locations are replicas created by a federation-initiated or external replication. Read-only locations may represent consistent point-in-time copies of a read-write location. The federation protocols, however, cannot prevent subsequent changes to a read-only location nor guarantee point-in-time consistency of a read-only location if the read-write location is changing.

Regardless of the type, all locations exist at the same mount point in the namespace and, thus, one client may be referred to one location while another is directed to a different location. Since updates to each fileset location are not controlled by the federation protocol, it is the responsibility of administrators to guarantee the functional equivalence of the data.

The federation protocol does not guarantee that the different locations are mutually consistent in terms of the currency of the data. It relies on the client file-access protocol (i.e., NFSv4) to contain sufficient information to help the clients determine the currency of the data at each location in order to ensure that the clients do not revert back in time when switching locations. This raises a concern for NFSv3 fileservers, which the federation protocol may support, that may lack such control.

3.4. Namespace Repository (NSDB)

The NSDB service is a federation-wide service that provides interfaces to define, update, and query FSN information and FSN to FSL mapping information. An individual repository of namespace information is called an NSDB location. Each NSDB location is managed by a single administrative entity. A single admin entity can manage multiple NSDB locations.

The difference between the NSDB service and an NSDB location is analogous to that between the DNS service and a particular DNS server.

The term local NSDB is shorthand for an NSDB location that is known a priori to a server. It is typically located within the same federation member as the server, but this is not required. A local NSDB is not required.

Each NSDB location stores the definition of the FSNs for which it is authoritative. It also stores the definitions of the FSLs associated with those FSNs. An NSDB location is authoritative for the filesets that it defines. An NSDB location can cache information from a peer NSDB location. The fileserver can always contact a local NSDB location (if it has been defined) or directly contact any NSDB location to resolve a junction. Each NSDB location supports an LDAP interface and can be accessed by an LDAP client.

3.5. Mount Points, Junctions and Referrals

A mount point is a directory in a parent fileset where a target fileset may be attached. If a client traverses the path leading from the root of the namespace to the mount point of a fileset it should be able to access the data in that fileset (assuming appropriate permissions).

The directory where a fileset is mounted is represented by a junction in the underlying filesystem. In other words, a junction can be viewed as a reference from a directory in one fileset to the root of the target fileset. A junction can be implemented as a special marker on a directory that is interpreted by the fileserver as a mount point, or by some other mechanism in the underlying file system.

What data is used by the underlying file system to represent the junction is not defined by this protocol. The essential property is that the server must be able to find, given the junction, the FSN for the target fileset. The FSN (as described earlier) contains both the NSDB location of the authoritative NSDB location and the FsnUuid (a UUID for the fileset).

When a client traversal reaches a junction, the client is referred to a list of FSLs associated with the FSN that was the target of the junction. The client can then redirect its connection to one of the FSLs. This act is called a referral. For NFSv4 clients, the FSL information is returned in the fs_locations or fs_locations_info attributes.

The federation-fs interfaces do not limit where and how many times a fileset is mounted in the namespace. Filesets can be nested -- a fileset can be mounted under another fileset.

3.6. Federation Root FileServers

A set of designated fileservers that render the common federation-wide namespace are called the federation root fileservers. The federation protocol does not mandate that federation root fileservers be defined. When a client mounts the root of the namespace from a root fileserver it can traverse the entire federation-wide namespace. It is not required for a client to mount from one of the root fileservers. If a client mounts from a non-root fileserver then it can traverse the part of the namespace that is visible starting from that fileserver. A client can mount multiple individual filesets from multiple non-root fileservers and chose to navigate the namespace in any manner. How the client discovers the root fileserver(s), if one is defined, is not in the scope of the federation protocol. Numerous external techniques such as DNS SRV records can be used for this.

3.7. Federation Root FileSet

The root fileset is the optional, top-level fileset of the federation-wide namespace. The root of the namespace is the top level directory of this fileset. The fileset can contain an arbitrary number of virtual directories. The leaf directories of the root fileset serve as the mount points for other filesets. It is desirable that the leaf directories not contain data. The root fileset is a simple combination of internal nodes and leaf nodes where each leaf node is a junction to a target fileset. The root fileset is replicated at all the root fileservers. The recommended replication protocols for root fileset replication are: an external protocol such as rsync or NDMP.

3.8. Fileservers

Fileservers are NFSv4 servers that store the physical fileset data or fileservers that refer the client to other fileservers.

3.9. File-access Clients

File access clients are standard off-the-shelf NAS clients that access file data using the NFSv4 protocol.

4. Interaction with NFSv4

The federation protocol is compatible with the requirements of NFSv4 referral mechanisms as defined in [RFC3530].

5. Finding the local NSDB

The local NSDB may be used for finding the mapping from the server's local representation of a junction to an FSN. How the mapping is resolved is implementation-specific. The fed-fs protocol does not mandate how and if a local NSDB is defined or located. A fileserver could choose to have a special configuration setup for defining the local or default NSDB in a manner similar to a resolv.conf file for DNS.

6. Examples

In this section we provide examples and discussion of the basic operations facilitated by the federated file system protocol: creating a fileset, adding a replica of a fileset, resolving a junction, and creating a junction.

6.1. Create a Fileset and its FSL(s)

A fileset is the abstraction of a set of files and their containing directory tree. The fileset abstraction is the fundamental unit of data management in the federation. This abstraction is implemented by an actual directory tree whose root location is specified by a fileset location (FSL).

In this section, we describe the basic requirements for starting with a directory tree and creating a fileset that can be used in the federation protocols. Note that we do not assume that the process of creating a fileset requires any transformation of the files or the directory hierarchy. The only thing that is required by this process is assigning the fileset a fileset name (FSN) and expressing the location(s) of the implementation of the fileset as FSL(s).

There are many possible variations to this procedure, depending on how the FSN that binds the FSL is created, and whether other replicas of the fileset exist, are known to the federation, and need to be bound to the same FSN.

It is easiest to describe this in terms of how to create the initial implementation of the fileset, and then describe how to add replicas.

6.1.1. Creating a Fileset and a FSN

1. Choose the NSDB node that will keep track of the FSL(s) and related information for the fileset.

2. Request that the NSDB node register a new FSN for the fileset.

   The FSN may either be chosen by the NSDB node or by the server. The latter case is used if the fileset is being restored, perhaps as part of disaster recovery, and the server wishes to specify the FSN in order to permit existing junctions that reference that FSN to work again.

   At this point, the FSN exists, but its location is unspecified.

3. Send the FSN, the local volume path, the export path, and the export options for the local implementation of the fileset to the

NSDB node. Annotations about the FSN or the location may also be sent.

The NSDB node records this info and creates the initial FSL for the fileset.

6.1.2. Adding a Replica of a Fileset

Adding a replica is straightforward: the NSDB node and the FSN are already known. The only remaining step is to add another FSL.

Note that the federation interfaces do not include methods for creating or managing replicas: this is assumed to be a platform-dependent operation (at least at this time). The only interface required is the ability to register or remove the registration of replicas for a fileset.

6.2. Junction Resolution

A fileset may contain references to other filesets. These references are represented by junctions. If a client requests access to a fileset object that is a junction, the server resolves the junction to discover the FSL(s) that implements the referenced fileset.

There are many possible variations to this procedure, depending on how the junctions are represented and how the information necessary to perform resolution is represented by the server. In this example, we assume that the only thing directly expressed by the junction is the junction key; its mapping to FSN can be kept local to the server hosting the junction.

Step 5 is the only step that interacts directly with the federation interfaces. The rest of the steps may use platform-specific interfaces.

1. The server determines that the object being accessed is a junction.

2. The server determines the junction key for the junction.

3. Using the junction key, the server does a local lookup to find the FSN of the target fileset.

4. Using the FSN, the server finds the NSDB node responsible for the target object.

5. The server contacts that NSDB node and asks for the set of FSLs that implement the target FSN. The NSDB node responds with a set

of FSLs.

6.3. Example use case for fileset annotations

The fileset annotations can be used to define relationships between filesets that can be used by an auxiliary replication protocol. Consider the scenario where a fileset is created and mounted at some point in the namespace. A snapshot of the read-write FSL of that fileset is taken periodically at different frequencies say a daily snapshot or a weekly snapshot. The different snapshots are mounted at different locations in the namespace. The daily snapshots are considered as a different fileset from the weekly ones but both are related to the source fileset. For this we can define an annotation labeling the filesets as source and replica. The replication protocol can use this information to copy data from one or more FSLs of the source fileset to all the FSLs of the replica fileset. The replica filesets are read-only while the source fileset is read-write.

This follows the traditional AFS model of mounting the read-only volume at a path in the namespace different from that of the read-write volume.

The federation protocol does not control or manage the relationship among filesets. It merely enables annotating the filesets with user-defined relationships.

7. Error Definitions

ERR_OK Indicates the operation completed successfully.

ERR_ACCESS Permission denied. The caller does not have the correct permission to perform the requested operation. Contrast this with ERR_PERM, which restricts itself to owner or privileged user permission failures.

ERR_BADCHAR A UTF-8 string contains a character which is not supported in the context in which it being used.

ERR_BADNAME A name string in a request consists of valid UTF-8 characters supported by the server but the name is not supported by the server as a valid name for current operation.

ERR_BADTYPE An attempt was made to create an object of a type not supported by the server.

ERR_DENIED An attempt to lock a file is denied. Since this may be a temporary condition, the client is encouraged to retry the lock request until the lock is accepted.

ERR_EXIST Object exists. The object specified already exists.

ERR_INVALID Invalid argument or unsupported argument for an operation.

ERR_IO I/O error. A hard error (for example, a disk error) occurred while processing the requested operation.

ERR_NAMETOOLONG The filename in an operation was too long.

ERR_NOENT No such object. The object being accessed does not exist.

ERR_NOTDIR Not a directory. The caller specified a non- directory in a directory operation.

ERR_NOTEMPTY An attempt was made to remove an object that was not empty. An FSN which has FSLs still defined for it.

ERR_NOTSUPP Operation is not supported.

ERR_PERM Not owner. The operation was not allowed because the caller is either not a privileged user (root) or not the owner of the target of the operation.

ERR_WRONGSEC The security mechanism being used by the client for the operation does not match the server's security policy. The client should change the security mechanism being used and retry the operation.

ERR_WRONGNSDB The NSDB location is not the one to be used for this operation.

8. Mapping the NSDB onto LDAP

This section describes the LDAP schema used to define the LDAP implementation of the NSDB service. The first part of the section describes the basic properties of the LDAP configuration that MUST be used in order to ensure compatibility between different implementations. The second section defines the new LDAP attribute types and the subsequent sections describe the new object types and specifies how the distinguished name of each object instance MUST be constructed.

8.1. Basic LDAP Configuration

The base name (or suffix) for all of DNs used by the NSDB schema is "dc=fed-fs,dc=com".

The DN of the priviledged LDAP user is, by convention, "cn=admin,dc=fed-fs,dc=com". This user is able to modify the contents of the LDAP database. It is permitted to use a different DN (or add additional priviledged users) but if a different DN is used then every admin entity that needs to modify the contents of the database or view privilidged information must be made aware of the new DN.

It MUST be possible for the anonymous (unauthenticated) user perform LDAP queries that access the NSDB data.

All implementation SHOULD use the same schema, or, at minimum, a schema that includes all of the objects, with each of the attributes, named in the following sections. The complete schema SHOULD be defined as part of the protocol (or as a separate RFC) when its definition is complete.

8.2. LDAP Attributes

This section describes the required attributes of the NSDB LDAP schema.

8.2.1. fedfsUuid

A fedfsUuid is the base type for all of the universally unique identifiers (UUIDs) used by the federated file system protocols.

This SHOULD be defined in terms of the text representation of the standard UUID (as defined in [RFC4122]).

It MAY also be useful, for purposes of debugging or annotation, to permit a fedfsUuid to include members of a more general class of

strings.

A fedfsUuid is a single-valued attribute.

8.2.2. fedfsNetAddr

An fedfsNetAddr is the locative name of a TCP/IP-based network service. It MUST be able to express network locations as IPv4, IPv6, and DNS FQDN notations. It may include a port specifier, or the port may be implicit in context.

There MAY be a special syntax at some point for specifying a SVR record (for a DNS FQDN).

This attribute is single-valued.

8.2.3. fsnUuid

A fsnUuid represents the fsnUuid component of an FSN.

The fsnUuid is a subclass of fedfsUuid.

This attribute is single-valued.

8.2.4. nsdbName

A nsdbName is the NSDB component of an FSN.

The nsdbName attribute is a subclass of fedfsNetAddr.

This attribute is single-valued.

8.2.5. fslHost

A fslHost is the hostname/port component of an FSL.

The fslHost attribute is a subclass of fedfsNetAddr.

This attribute is single-valued.

8.2.6. fslPath

The path component of an FSL.

This attribute is single-valued.

8.2.7. annotation

An annotation of an NSDB object.

This attribute is multi-valued; an object type that permits annotations may have any number of annotations per instance.

This attribute is a placeholder; it has not been well-defined at the date of this draft.

8.2.8. descr

A descriptive attribute containing information about an NSDB object.

This attribute is single-valued.

This attribute is a placeholder; it has not been well-defined at the date of this draft.

8.2.9. fslUuid

Each FSL must have a UUID associated with it, which serves as part of its DN.

The fslUuid attribute is a subclass of fedfsUuid.

This attribute is single-valued.

8.2.10. junctionKey

Each junction has a unique junctionKey that is used to distinguish it from other junctions that may refer to the same child fileset and/or appear within the same parent fileset.

The junctionKey attribute is a subclass of fedfsUuid.

This attribute is single-valued.

8.2.11. childFsnUuid

The fsnUuid of the target of a junction.

The childFsnUuid attribute is a subclass of fsnUuid.

This attribute is single-valued.

8.2.12. childNsdbName

The nsdbName of the target of a junction.

The childNsdbName attribute is a subclass of nsdbName.

This attribute is single-valued.

8.3. LDAP Objects

8.3.1. FsnObject

An FsnObject represents an FSN.

The required attributes of an FsnObject are an fsnUuid and nsdbName.

An FsnObject MAY also have descr and annotation attributes, but neither is required.

The DN of an FSN is assumed to take the following form: "fsnUuid=FSNUUID,dc=fed-fs,dc=com", where fsnUuid is the UUID of the FSN.

An FsnObject MAY also have additional attributes, but these attributes MUST NOT be referenced by any part of this draft.

8.3.2. FslObject

An FslObject represents an FSL.

The required attributes of an FslObject are an fsnUuid, nsdbName, fslHost, fslPath, and fslUuid.

An FslObject MAY also have descr and annotation attributes, but neither is required.

The DN of an FSL is required to take the following form: "fslUuid=UUID,fsnUuid=FSNUUID,dc=fed-fs,dc=com".

To find all the FSLs that match a given FSN, query for the children of the object with DN "fsnUuid=FSNUUID,dc=fed-fs,dc=com" with a filter for "objectType = fslObject". (If you want to be doubly careful, you can also filter by the nsdbName.)

8.3.3. JunctionObject

An JunctionObject captures the relationship between a fileset and its children (if any). The children FSNs are FSNs that appear in

junctions in the fileset named by the fsnUuid and nsdbName attributes of the parent FSN.

The required attributes of a JunctionObject are a junctionKey, fsnUuid, nsdbName, childFsnUuid, and childNsdbName.

A JunctionObject MAY also have descr and annotation attributes, but neither is required.

The required form of a DN for an JunctionObject is: "junctionKey=KEY,fsnUuid=FSNUUID,dc=fed-fs,dc=com" where KEY is a unique key chosen for this relationship (the junctionKey) and FSNUUID is the fsnUuid of the parent fileset's FSN.

Note that the reason why KEY might be something other than simply the fsnUuid of the child's FSN is that a child FSN may appear as the target of several junctions within the same fileset, and we must have a way to distinguish each of these junctions.

To find all the junctions within a given fileset, query for the children of the object with DN "fsnUuid=FSNUUID,dc=fed-fs,dc=com" and filter for "objectType = JunctionObject". (If you want to be doubly careful, you can also filter by the nsdbName.)

9. NSDB Protocol Operations

The operations defined by the protocol can be described as several sub-protocols that are used by entities within the federation to perform different roles.

The first of these sub-protocols defines how the state of an NSDB location can be initialized and updated. The primary use of this sub-protocol is by an administrator to add, edit, or delete filesets, their properties, and their fileset locations.

The second of these sub-protocols defines the queries that are sent to an NSDB location in order to perform resolution (or find other information about the information stored within that NSDB location) and the responses returned by the NSDB location. The primary use of this sub-protocol is by a fileset server in order to perform resolution, but it may also be used by an administrator to query the state of the system.

The first and second sub-protocols are defined as LDAP operations, using the schema defined in the previous section. If each NSDB location is a standard LDAP server, then, in theory, it is unnecessary to describe the LDAP operations in detail, because the operations are ordinary LDAP operations to query and update records. However, we do not require that an NSDB location implement a complete NSDB service, and therefore we define in these sections the minimum level of LDAP functionality required to implement an NSDB location.

The NSDB sub-protocols are defined in the next two sub-sections.

The third sub-protocol defines the queries or other requests that are sent to a fileset server in order to get information from it or to modify the state of the fileset server in a manner related to the federation protocols. The primary purpose of this for an administrator to create or delete a junction or fileset or discover related information about a particular fileset server.

The third sub-protocol is defined as ONC/RPC operations. The reason for using a different RPC mechanism (instead of mapping these operations onto LDAP) is to minimize the changes required to the fileset server. This protocol is described in a separate document.

9.1. Administrative NSDB Operations

The admin entity initiates and controls the commands to manage fileset and namespace information. The admin entity, however, is stateless. All state is maintained at the NSDB locations or at the fileserver.

We require that each NSDB location be able to act as an LDAP server and that the protocol used for communicating between the admin entity and each NSDB is LDAP.

The names we assign to these operations are entirely for the purpose of exposition in this document, and are not part of the LDAP dialogs.

In the description of the LDAP messages and LDIF, we use the following notation: constant strings and literal names are specified in lower or mixed case, while variables or values are specified in uppercase. One important exception to this rule is that the names of the error codes follow the convention (used widely in other protocols, including NFS) of having names that are entirely uppercase.

NEED TO UPDATE THE TEXT HERE TO REFER TO THE OTHER DRAFT. -DJE

9.1.1. Creating an FSN

The administrator uses this operation to create a new FSN by requesting the NSDB to create a new FsnObject in its LDAP database with an fsnUuid of FSNUUID and an NsdbName of NSDB.

The NSDB location that receives the request SHOULD check that the NSDB matches its own value and return an ERR_WRONGNSDB error if does not. This is to ensure that an FSN is always created by the NSDB location encoded within the FSN as its owner.

The NSDB location that receives the request SHOULD check all of the attributes for validity and consistency, but this is not generally possible for LDAP servers because the consistency requirements cannot be expressed in the LDAP schema (although many LDAP servers can be extended, via plug-ins or other mechanisms, to add functionality beyond the strict definition of LDAP).

PARAGRAPH DESCRIBING ERRORS

9.1.1.1. LDAP Request

The admin chooses the fsnUuid and NsdbName of the FSN. The fsnUuid is a UUID and should be chosen via a standard process for creating a UUID (described in [RFC4122]). The NsdbName is the name of the NSDB location that will serve as the source of definitive information about an FSN for the life of that FSN. In the example below, the admin server chooses a fsnUuid of FSNUUID and the NsdbName of NSDB and then sends an LDAP ADD request, described by the LDIF below, to the NSDB location NSDB. This will create a new FsnObject on that NSDB location with the given attributes in the LDAP database.

```
dn: fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: add
objectClass: FsnObject
fsnUuid: FSNUUID
nsdbName: NSDB
```

9.1.2. Deleting an FSN

Deletes the Fileset with the given FSN. This assumes that all the FSLs related to that FSN have already been deleted. If FSL records for this FSN still exist in the database of the NSDB that receives this request, then this function MUST return with an ERR_NOTEMPTY error.

Note that the FSN delete function only removes the fileset from the namespace (by removing the records for that FSN from the NSDB location that receives this request). The fileset and its data are not deleted. Any junction that has this FSN as its target may continue to point to this non-existent FSN. A dangling reference may be detected when a client tries to resolve the target of a junction that refers to the deleted FSN and the NSDB returns ERR_NOTFOUND.

PARAGRAPH DESCRIBING ERRORS

9.1.2.1. LDAP Request

The admin then sends an LDAP DELETE request to the NSDB server to remove the FsnObject from the NSDB server. An example LDIF for the delete request is shown below.

```
dn: fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: delete
```

9.1.3. Mount an FSN

NOTE: the semantics of this operation have changed significantly, and "mount" might be a quite unintuitive name at this point.

The mount operation records that a given fileset (called the parent fileset) contains a junction. The target of that fileset is called the child fileset.

The NSDB of the parent fileset (as identified by the FSN of the parent) maintains this information.

The parent/child relation is used to indicate how the filesets in the federation are related. The names "parent" and "child" should not be taken literally. A fileset can have no parent (if it is a root

fileset). A fileset may also have any number of parents. In theory, the parent of a fileset may also be its child, although in practice this is deprecated.

A fileset may be mounted in multiple places, and may have the same parent multiple times. For example, /home/ellard and /home/daniel.ellard might both be junctions from the /home fileset to the fileset that represents the home directory of user Daniel Ellard. In order to be able to distinguish each mount, each mount is given a unique identifier (in addition to the fsnUuids of the parent and child).

PARAGRAPH DESCRIBING ERRORS

9.1.3.1. LDAP Request

On fileset mount operation the admin will generate an LDAP ADD request to the NSDB server using the example LDIF below. This creates a new FsnJunctionObject that establishes the mount relationship between the parent and target FSNs.

```
dn: key=KEY,fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: add
objectClass: JunctionObject
fsnUuid: FSNUUID
nsdbName: NSDBNAME
childFsnUuid: CHILDFSNUUID
childNsdbName: CHILDNSDB
```

9.1.4. Unmount an FSN

Removes the record of a junction between a parent and child fileset.

PARAGRAPH DESCRIBING ERRORS

9.1.4.1. LDAP Request

In case a target_FSN is to be unmmounted, the associated JunctionObject is deleted from the NSDB maintaining the parent fileset. An example delete request is shown below.

```
dn: key=KEY,fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: delete
```

9.1.5. Create an FSL

Creates a new Fileset location at the given location denoted by HOST and PATH for the given FSN. An fsl_uuid may be provided as an optional UUID for the FSL. Normally an FSL is identified by the HOST:PATH pair. A UUID is an optional way to identify an FSL if it is recovered to a different HOST:PATH after a backup/restore. If the FSL belongs to an FSN that has another FSN mounted under it then there would be a related junction_create operation.

PARAGRAPH DESCRIBING ERRORS

The FSL create command will result in the admin server sending an LDAP ADD request to create a new FslObject at the NSDB maintaining the given FSN. The example LDIF is shown below. The PATH is the pathname where the fileset is located on that host.

9.1.5.1. LDAP Request

```
dn:fslUuid=UUID,fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: add
objectClass: FslObject
fsnUuid: FSNUUID
nsdbName: NSDB
fslUuid: UUID
fslHost: HOST
fslPath: PATH
type: nfs4
version: VERSION
```

9.1.6. Delete an FSL

Deletes the given Fileset location. The admin requests the NSDB location storing the FslObject to delete it from its database. This operation does not result in the fileset location's data being deleted at the fileserver.

PARAGRAPH DESCRIBING ERRORS

9.1.6.1. LDAP Request

```
dn: fslUuid=UUID,fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: delete
```

9.1.7. Update an FSL

Update the attributes of a given FSL. This command results in a change in the attributes of the FslObject at the NSDB server maintaining this FSL. The attributes that must not change are the fslUuid and the fsnUuid of the fileset this FSL implements.

PARAGRAPH DESCRIBING ERRORS

9.1.7.1. LDAP Request

```
dn: fslUuid=UUID,fsnUuid=FSNUUID,dc=fed-fs,dc=com
changeType: modify
replace: ATTRIBUTE-TYPE
```

9.1.8. Examining an FSL

Find all attributes of a given FSL from the FSLObject stored at the NSDB location.

ERRORS: ERR_OK ERR_NOTFOUND ERR_INVALID ERR_PERM

WHERE IS THE LDAP FOR THIS? -DJE

9.1.9. Finding the children FSNs of a fileset

The NSDB also tracks information about which filesets are "children" of others. A fileset X is a child of fileset Y if there is a junction in fileset Y referencing fileset X. (note that this definition permits a fileset to be its own child, although this use is deprecated)

In addition to keeping track of whether one fileset has another as its child, the NSDB also records additional information to simplify management -- each parent/child relation is associated with an additional key that is used to disambiguate the relationship. For example, one fileset may have several junctions targeting the same child, but each has a seperate key that can be used to differentiate them. This permits junctions to be removed without necessarily removing the underlying relationship.

NOTE: if it is decided to require that there can only be one junction from one fileset to a second, then the key should simply be the FSN of the target. This restriction would greatly simplify some aspects of the implementation (but it may also eliminate some very useful functionality).

```
LDAP Request
Search base: fsnUuid=FSNUUID, dc=fed-fs, dc=com
Search scope: onelevel
Search filter:  (objectClass=JunctionObject)
```

9.2. Fileserver to NSDB Operations

9.2.1. Looking up FSLs for an FSN

Return the list of FSLs for the FSN with an fsnUuid that matches the filter. The fileserver will convert the list of FSLs to the NFSv4 fs_locations.

The filter may also specify the type of protocol (v4, v3), or type of data access (ro, rw).

ERRORS: ERR_OK ERR_NOTFOUND ERR_INVALID ERR_PERM

```
LDAP Request
Search base: fsnUuid=FSNUUID, dc=fed-fs, dc=com
Search scope: onelevel
Search filter:  (objectClass=FslObject)
```

The server can scan through the results and find results whose type corresponds to the type of the client on whose behalf the server is performing the request, extracting the fslHost and fslPath (and possibly additional attributes) and using them to create a list of fs_locations that the client can use.

10. Security Considerations

To be added.

11. IANA Requirements

This document has no actions for IANA.

12. Conclusions

The federated filesystem protocol manages multiple independently administered fileservers to share namespace and referral information to enable clients to traverse seamlessly across them.

13. Glossary

Administrator: user with the necessary authority to initiate administrative tasks on one or more servers.

Admin entity: A server or agent that administers a collection of fileservers and persistently stores the namespace information.

Client: Any client that accesses the fileserver data using a supported filesystem access protocol.

Federation: A set of server collections and singleton servers that use a common set of interfaces and protocols in order to provide to their clients a federated namespace accessible through a filesystem access protocol.

Fileserver: A server exporting a filesystem via a network filesystem access protocol.

Fileset: The abstraction of a set of files and their containing directory tree. A fileset is the fundamental unit of data management in the federation.

Note that all files within a fileset are descendants of one directory, and that filesets do not span filesystems.

Filesystem: A self-contained unit of export for a fileserver, and the mechanism used to implement filesets. The fileset does not need to be rooted at the root of the filesystem, nor at the export point for the filesystem.

A single filesystem MAY implement more than one fileset, if the client protocol and the fileserver permit this.

Filesystem access protocol: A network filesystem access protocol such as NFSv2 [RFC1094], NFSv3 [RFC1813], NFSv4 [RFC3530], or CIFS.

FSL (Fileset location): The location of the implementation of a fileset at a particular moment in time. A FSL MUST be something that can be translated into a protocol-specific description of a resource that a client can access directly, such as a fs_location (for NFSv4), or share name (for CIFS). Note that not all FSLs need to be explicitly exported as long as they are contained within an exported path on the fileserver.

FSN (Fileset name): A platform-independent and globally unique name for a fileset. Two FSLs that implement replicas of the same fileset MUST have the same FSN, and if a fileset is migrated from one location to another, the FSN of that fileset MUST remain the same.

Junction: A filesystem object used to link a directory name in the current fileset with an object within another fileset. The server-side "link" from a leaf node in one fileset to the root of another fileset.

Junction key: The UUID of a fileset, used as a key to lookup an FSN within an NSDB node or a local table of information about junctions.

Namespace: A filename/directory tree that a sufficiently-authorized client can observe.

NSDB (Namespace Database Service): A service that maps FSNs to FSLs. The NSDB may also be used to store other information, such as annotations for these mappings and their components.

NSDB Node: The name or location of a server that implements part of the NSDB service and is responsible for keeping track of the FSLs (and related info) that implement a given partition of the FSNs.

Referral: A server response to a client access that directs the client to evaluate the current object as a reference to an object at a different location (specified by an FSL) in another fileset, and possibly hosted on another fileserver. The client re-attempts the access to the object at the new location.

Replica: A replica is a redundant implementation of a fileset. Each replica shares the same FSN, but has a different FSL.

Replicas may be used to increase availability or performance. Updates to replicas of the same fileset MUST appear to occur in the same order, and therefore each replica is self-consistent at any moment.

We do not assume that updates to each replica occur simultaneously If a replica is offline or unreachable, the other replicas may be updated.

Server Collection: A set of fileservers administered as a unit. A server collection may be administered with vendor-specific software.

The namespace provided by a server collection could be part of the federated namespace.

Singleton Server: A server collection containing only one server; a stand-alone fileserver.

14. Normative References

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

[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.

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, September 1997.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[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.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC4513] Harrison, R., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

Authors' Addresses

Daniel Ellard
NetApp, Inc.
1601 Trapelo Rd, Suite 16
Waltham, MA 02451
US

Phone: +1 781-768-5421
Email: ellard@netapp.com

Craig Everhart
NetApp, Inc.
7301 Kit Creek Rd
Research Triangle Park, NC 27709
US

Phone: +1 919-476-5320
Email: everhart@netapp.com

Renu Tewari
IBM Almaden
650 Harry Rd
San Jose, CA 95120
US

Email: tewarir@us.ibm.com

Manoj Naik
IBM Almaden
650 Harry Rd
San Jose, CA 95120
US

Email: manoj@almaden.ibm.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).