Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 30, 2009

D. Ellard
BBN Technologies
C. Everhart
J. Lentini
NetApp
R. Tewari
M. Naik
IBM Almaden

September 26, 2008

Requirements for Federated File Systems
draft-ietf-nfsv4-federated-fs-reqts-00.txt

Status of this Memo

Copyright Notice

Abstract

This draft describes and lists the functional requirements of a federated file system and defines related terms. Our intent is to use this draft as a starting point and refine it, with input and feedback from the file system community and other interested parties, until we reach general agreement. We will then begin, again with the help of any interested parties, to define standard, open federated file system protocols that satisfy these requirements and are suitable for implementation and deployment.

Table of Contents

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].

Note, that this is a requirements document, and in many instances where these words are used in this document they refer to qualities of a specification for a system that satisfies the document, or requirements of a system that matches that specification. These cases are distinguished when there is potential for ambiguity.

2. Draft Goals

This draft describes and lists the functional requirements of a federated file system and defines related terms. Our intent is to use this draft as a starting point and refine it, with input and feedback from the file system community and other interested parties, until we reach general agreement. We will then begin, again with the help of any interested parties, to define standard, open federated file system protocols that satisfy these requirements and are suitable for implementation and deployment.

We do not describe the mechanisms that might be used to implement this functionality except in cases where specific mechanisms, in our opinion, follow inevitably from the requirements. Our focus is on the interfaces between the entities of the system, not on the protocols or their implementations.

For the first version of this document, we are focused on the following questions:

- Are any "MUST" requirements missing?
- Are there any "MUST" requirements that should be "SHOULD" or "MAY"?
- Are there any "SHOULD" requirements that should be "MAY"?
- Are there better ways to articulate the requirements?

3. Overview

Today, there are collections of fileservers that inter-operate to provide a single namespace comprised of filesystem resources provided by different members of the collection, joined together with inter-filesystem junctions. The namespace can either be assembled at the fileservers, the clients, or by an external namespace service -- the mechanisms used to assemble the namespace may vary depending on the filesystem access protocol used by the client.

These fileserver collections are, in general, administered by a single administrative entity. This administrator builds the namespace out of the filesystem resources and junctions. There are also singleton servers that export some or all of their filesystem resources, but which do not contain junctions to other filesystems.

Current server collections that provide a shared namespace usually do so by means of a service that maps filesystem names to filesystem locations. We refer to this as a namespace database service (NSDB). In some distributed file systems, this service is embodied as a volume location database (VLDB), and may be implemented by LDAP, NIS, or any number of other mechanisms.

We use the term "fileset" to represent the abstraction of a filesystem. The fileset abstraction implies very little about how the fileset is implemented, although in the simplest case a fileset can be implemented by an exported filesystem. A fileset is a directory tree that may contain files and references, called "junction", to other filesets. Each fileset has a fileset globally unique name (FSN) that is used as an identifier for the fileset. Each implementation of a given fileset is specified by its fileset location (FSL).

The primary purpose of the NSDB service is to provide a level of indirection between the FSN the FSLs. If the NSDB service permits updates to the set of mappings, then the FSLs may be changed (e.g., moved or replicated) in a manner that is transparent to the referring fileset and its server(s).

Current approaches are unsuitable to build common namespaces across systems with multiple administrative domains and multiple NSDB nodes. An approach which requires changing existing NSDB nodes to collaborate or replacing them with a single NSDB node, while possible, is not desirable.

Figure Figure 1 shows an example of a federation. This federation has two members, named ALPHA and BETA. Federation members may contain an arbitrary number of file servers and NSDB nodes; in this

illustration ALPHA and BETA each have three servers and one NSDB node.

A federation with two members, ALPHA and BETA. ALPHA and BETA each have their own NSDB node and several file servers, but are administered separately.

Figure 1

4. Purpose

Our objective is to specify a set of interfaces (and corresponding protocols) by which such fileservers and collections of fileservers, with different administrators, can form a federation of fileservers and NSDB nodes that provides a namespace composed of the filesets hosted on the different fileservers and fileserver collections.

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 NSDB nodes, each belonging to a different administrative entity, and each providing the mappings that define a part of a namespace. Such a federation may also have an arbitrary number of administrative entities, each responsible for administering a subset of the servers and NSDB nodes. Acting in concert, the administrators should be able to build and administer this multi-fileserver, multi-collection namespace.

Each singleton server can be presumed to provide its own NSDB node, for example with a trivial mapping to local FSLs.

It is not the intent of the federation to guarantee namespace consistency across all client views. Since different parts of the namespace may be administered by different entities, it is possible that a client could be accessing a stale area of the namespace managed by one entity because a part of the namespace above it, managed by another entity, has changed.

5. Examples and Discussion

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.

5.1. 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.

5.1.1. 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.

5.1.2. 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.

5.2. 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. The server converts the FSL to the location type used by the client (e.g., fs_location for NFSv4, as described in [RFC3530]).

7. The server redirects (in whatever manner is appropriate for the client) the client to the location(s).

These steps are illustrated in Figure 2. The client sends request 1 to server X, in federation member ALPHA, in an attempt to reference an object (which appears to the client as a directory). Server X recognizes that the referenced object is actually a junction that refers to a directory in a different fileset. Server X finds, from the FSN in the junction, that the NSDB responsible for knowing the location of the target of the junction is the NSDB of federation member BETA. Server X sends request 2 to the NSDB of BETA, asking for the current location of the directory. The NSDB sends response 3 to server X, telling the server that the directory is located on server Y. Server X sends response 4 to the client, indicating that the directory is in a "new" location on server Y. The client then sends request 5 to server Y, repeating the initial request.

Given the current requirements and definitions, this resolution method MUST work. However, there is no requirement that this is the only resolution method that can be used. This method may be used as the fallback when all else fails (or, for a simple implementation, it could be the only method). This is a degenerate implementation of the NSDB service as a simple composition of NSDB nodes; we expect that large federations will use more sophisticated methods to share the FSN and FSL information among multiple NSDB nodes.

Figure 2

5.3. Junction Creation

Given a local path, a remote export and a path relative to that export, create a junction from the local path to the path within the remote export.

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 1 is the only step that uses the federation interfaces. The rest of the steps may use platform-specific interfaces.

1. Contact the server named by the export and ask for the FSN for the fileset, given its path relative to that export.

2. Create a new local junction key.

3. Insert, in the local junction info table, a mapping from the local junction key to the FSN.

4. Insert the junction, at the given path, into the local filesystem.

6. Glossary

The phrase "USING THE FEDERATION INTERFACES" implies that the subsequent requirement must be satisfied, in its entirety, via the federation interfaces.

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.

7. Proposed Requirements

Note that the requirements are described in terms of correct behavior by all entities. We do not address the requirements of the system in the presence of faults.

7.1. Basic Assumptions

Several of the requirements are so fundamental that we treat them as basic assumptions; if any of these assumptions are violated, the rest of the requirements must be reviewed in their entirety.

A1: The federation protocols do not require any changes to existing client-facing protocols, and MAY be extended to incorporate new client-facing protocols.

A2: A client SHOULD NOT require any a priori knowledge of the general structure or composition of the federation.

The client may require some specific knowledge in order to find and access an instance of the fileset that defines the root of its view of the namespace. As the client traverses the namespace, the client discovers the information it needs in order to locate the filesets it accesses.

A3: All requirements MUST be satisfiable via the federation protocols and the standard protocols used by the fileservers (i.e., NFS, CIFS, DNS, etc).

USING THE FEDERATION INTERFACES, a federation operation that requires an interaction between two (or more) entities that are members of the federation MUST be possible without requiring any proprietary protocols.

A4: All the entities participating in a federation operation MUST be able to authenticate each other.

All principals (clients, users, administrator of a singleton or server collection, hosts, NSDB nodes, etc) that can assume a role defined by the federation protocol can identify themselves to each other via an authentication mechanism. This mechanism is not defined or further described in this document.

The authority of a principal to request that a second principal perform a specific operation is ultimately determined by the second. Authorization may be partitioned by server collection or set of servers as well as by operation. For example, if a user has administrative privileges on one server in the

federation, this does not imply that they have administrative privileges (or, for that matter, any privileges whatsoever) on any other server in the federation.

In order to access the functionality provided by the federation interfaces, it may be necessary to have elevated privileges or authorization. The authority required by different operations may be different. For example, the authority required to query the NSDB about the FSLs bound to an FSN may be different than the authority required to change the bindings of that FSN.

An operation attempted by an unauthorized entity MUST fail in a manner that indicates that the failure was due to insufficient authorization.

This document does not enumerate the authorization necessary for any operation.

A5: The federation protocols MUST NOT require changes to existing authentication/authorization mechanisms in use at the fileservers for client-facing protocols.

A user's view of the namespace may be limited by the authentication and authorization privileges it has on the different fileservers in the federation. As such, users may only be able to traverse the parts of the namespace that they have access to.

The federation protocols do not impose any restrictions on how users are represented within the federation. For example, a single enterprise could employ a common identity for users across the federation. A grid environment could utilize user mapping or translations across different administrative domains.

A6: In a federated system, we assume that a FSN MUST express, or can be used to discover, the following two pieces of information:

1. The location of the NSDB node that is responsible for knowing the filesystem location(s) (FSLs) of the named fileset.

   The NSDB node must be specified because there may be many NSDB nodes in a federation. We do not assume that any single entity knows the location of all of the NSDB nodes, and therefore exhaustive search is not an option.

   There are several ways in which a fileserver can locate the NSDB node responsible for a given fileset. One approach,

given a DNS infrastructure, is to specify the location of the NSDB node by the FQDN of the server hosting the NSDB node. Another approach is to use a separate DNS-style hierarchy to resolve the location of the NSDB node.

2. The junction key.

   The junction key is the index used by the NSDB node to identify the FSN of the target fileset.

   There are several ways to represent junction keys. One approach could use 128-bit UUIDs as described described in [RFC4122].

As an example, an FSN could be represented by a URL of the form nsdb.example.com/UUID where nsdb.example.com is the FQDN of the server hosting the NSDB node and UUID is the string representation of the junction key.

Note that it is not assumed that it is always required for a server to contact the NSDB node specified by the FSN in order to find the FSLs. The relevant information stored in that NSDB node may also be cached local to the server or on a proxy NSDB node "near" the server.

A7: All federation servers and NSDB nodes are assumed to execute the federation protocols correctly. The behavior of the federation is undefined in the case of Byzantine behavior by any federation server or NSDB node.

A8: The locations of federation services (such as NSDBs and FSLs) can be specified in a manner such that they can be correctly interpreted by all members of the federation that will access them.

For example, if an NSDB node is specified by a FQDN, then this implies that every member of the federation that needs to access this NSDB node can resolve this FQDN to an IP address for that NSDB node. (It is not necessary that the FQDN always resolve to the same address; the same service may appear at different addresses on different networks.)

It is the responsibility of each federation member to ensure that the resources it wishes to expose have accessible network locations and that the necessary resolution mechanisms (i.e., DNS) are given the necessary data to perform the resolution correctly.

7.2. Requirements

R1: Requirements of each FSN:

a. Each FSN MUST be globally unique.

b. The FSN MUST be sufficiently descriptive to locate an instance of the fileset it names within the federation at any time.

c. An FSN is a name of a fileset. (An FSL is not the name of a fileset, but only a locator of an instance of a fileset at some point in time. For example, the same FSL may implement different filesets at different times.)

+ If a fileset instance is moved to a new location, it will have a new FSL, but its FSN is unchanged.

+ An instance of a different fileset may be placed at a FSL previously occupied by an instance of a different fileset.

d. If a fileset instance is migrated to another location, the FSN remains the same in the new location.

e. If the fileset is replicated using the federation interfaces, then all of the replicas have the same FSN.

Not all filesets in the federation are required to have a FSN or be reachable by a FSL. Only those filesets that are the target of a junction (as described in R3) are required to have an FSN.

NOTE: this requirement has been called into question.

R2: USING THE FEDERATION INTERFACES, it MUST be possible to create an FSN for a fileset, and it must be possible to bind an FSL to that FSN. These operations are NSDB operations and do not require any action on the part of an NFS server.

It is possible to create an FSN for a fileset that has not actually been created. It is also possible to bind a nonexistant FSL to an FSN. It is also possible to create a fileset without assigning it an FSN. The binding between an FSN and an FSL is defined entirely within the context of the NSDB; the servers do not "know" whether the filesets they host have been assigned FSNs (or, if so, what those FSNs are).

The requirement that filesets can exist prior to being assigned an FSN, and the requirement that FSNs can exist independent of filesets are intended to simplify the construction of the namespace in a convenient manner. For example, they permit an admin to assign FSNs to existing filesets and thereby incorporate existing filesets into the namespace. They also permit the structure of the namespace to be defined prior to creation of the component filesets. In either case, it is the responsibility of the entity updating the NSDB with FSNs and FSN-to-FSL mappings to ensure that the namespace is constructed in a consistent manner. (The simplest way to accomplish this is to ensure that the FSN and FSN-to-FSL mappings are always recorded in the NSDB prior to the creation of any junctions that refer to that FSN.)

a. An administrator MAY specify the entire FSN (including both the NSDB node location and the junction key) of the newly-created FSL, or the administrator MAY specify only the NSDB node and have the system choose the junction key.

   The admin can choose to specify the FSN explicitly in order to recreate a lost fileset with a given FSN (for example, as part of disaster recovery). It is an error to assign an FSN that is already in use by an active fileset.

   Note that creating a replica of an existing filesystem is NOT accomplished by assigning the FSN of the filesystem you wish to replicate to a new filesystem.

b. USING THE FEDERATION INTERFACES, it MUST be possible to create a federation FSL by specifying a specific local volume, path, export path, and export options.

R3: USING THE FEDERATION INTERFACES, and given the FSN of a target fileset, it MUST be possible to create a junction to that fileset at a named place in another fileset.

After a junction has been created, clients that access the junction transparently interpret it as a reference to the FSL(s) that implement the FSN associated with the junction.

a. It SHOULD be possible to have more than one junction whose target is a given fileset. In other words, it SHOULD be possible to mount a fileset at multiple named places.

b. If the fileset in which the junction is created is replicated, then the junction MUST eventually appear in all of its replicas.

The operation of creating a junction within a fileset is treated as an update to the fileset, and therefore obey the general rules about updates to replicated filesets.

R4: USING THE FEDERATION INTERFACES, it MUST be possible to delete a specific junction from a fileset.

If a junction is deleted, clients who are already viewing the fileset referred to by the junction after traversing the junction MAY continue to view the old namespace. They might not discover that the junction no longer exists (or has been deleted and replaced with a new junction, possibly referring to a different FSN).

After a junction is deleted, another object with the same name (another junction, or an ordinary filesystem object) may be created.

The operation of deleting a junction within a fileset is treated as an update to the fileset, and therefore obey the general rules about updates to replicated filesets.

R5: USING THE FEDERATION INTERFACES, it MUST be possible to invalidate an FSN.

a. If a junction refers to an FSN that is invalid, attempting to traverse the junction MUST fail.

An FSN that has been invalidated MAY become valid again if the FSN is recreated (i.e., as part of a disaster recovery process).

If an FSN is invalidated, clients who are already viewing the fileset named by the FSN MAY continue to view the old namespace. They might not discover that the FSN is no longer valid until they try to traverse a junction that refers to it.

R6: USING THE FEDERATION INTERFACES, it MUST be possible to invalidate a FSL.

a. An invalid FSL MUST NOT be returned as the result of resolving a junction.

An FSL that has been invalidated MAY become valid again if the

FSL is recreated (i.e., as part of a disaster recovery process).

If an FSL is invalidated, clients who are already viewing the fileset implemented by the FSL MAY continue to use that FSL. They might not discover that the FSL is no longer valid until they try to traverse a junction that refers to the fileset implemented by the FSL.

Note that invalidating an FSL does not imply that the underlying export or share (depending on the file access protocol in use) is changed in any way -- it only changes the mappings from FSNs to FSLs on the NSDB.

R7: It MUST be possible for the federation of servers to provide multiple namespaces.

R8: USING THE FEDERATION INTERFACES, it MUST be possible to perform queries about the state of objects relevant to the implementation of the federation namespace.

It MUST be possible to query the fileserver named in an FSL to discover whether a junction exists at a given path within that FSL.

R9: The projected namespace (and the objects named by the namespace) MUST be accessible to clients via at least one standard filesystem access protocol.

a. The namespace SHOULD be accessible to clients via the CIFS protocol.

b. The namespace SHOULD be accessible to clients via the NFSv4 protocol as described in [RFC3530].

c. The namespace SHOULD be accessible to clients via the NFSv3 protocol as described in [RFC1813].

d. The namespace SHOULD be accessible to clients via the NFSv2 protocol as described in [RFC1094].

It must be understood that some of these protocols, such as NFSv3 and NFSv2, have no innate ability to access a namespace of this kind. Where such protocols have been augmented with other protocols and mechanisms (such as autofs or amd for NFSv3) to provide an extended namespace, we propose that these protocols and mechanisms may be used, or extended, in order to satisfy the requirements given in this draft, and different

clients may use different mechanisms.

R10: USING THE FEDERATION INTERFACES, it MUST be possible to modify the NSDB mapping from an FSN to a set of FSLs to reflect the migration from one FSL to another.

R11: FSL migration SHOULD have little or no impact on the clients, but this is not guaranteed across all federation members.

Whether FSL migration is performed transparently depends on whether the source and destination servers are able to do so. It is the responsibility of the administrator to recognize whether or not the migration will be transparent, and advise the system accordingly. The federation, in turn, MUST advise the servers to notify their clients, if necessary.

For example, on some systems, it may be possible to migrate a fileset from one system to another with minimal client impact because all client-visible metadata (inode numbers, etc) are preserved during migration. On other systems, migration might be quite disruptive.

R12: USING THE FEDERATION INTERFACES, it MUST be possible to modify the NSDB mapping from an FSN to a set of FSLs to reflect the addition/removal of a replica at a given FSL.

R13: Replication SHOULD have little or no negative impact on the clients.

Whether FSL replication is performed transparently depends on whether the source and destination servers are able to do so. It is the responsibility of the administrator initiating the replication to recognize whether or not the replication will be transparent, and advise the federation accordingly. The federation MUST advise the servers to notify their clients, if necessary.

For example, on some systems, it may be possible to mount any FSL of an FSN read/write, while on other systems, there may be any number of read-only replicas but only one FSL that can be mounted read-write.

R14: USING THE FEDERATION INTERFACES, it SHOULD be possible to annotate the objects and relations managed by the federation protocol with arbitrary name/value pairs.

These annotations are not used by the federation protocols --

they are intended for use by higher-level protocols. For example, an annotation that might be useful for a system administrator browsing the federation would be the "owner" of each FSN (i.e., "this FSN is for the home directory of Joe Smith."). As another example, the annotations may express hints used by the clients (such as priority information for NFSv4.1).

Both FSNs and FSLs may be annotated. For example, an FSN property might be "This is Joe Smith's home directory", and an FSL property might be "This instance of the FSN is at the remote backup site."

a. USING THE FEDERATION INTERFACES, it MUST be possible to query the system to find the annotations for a junction.

b. USING THE FEDERATION INTERFACES, it MUST be possible to query the system to find the annotations for a FSN.

c. USING THE FEDERATION INTERFACES, it MUST be possible to query the system to find the annotations for a FSL.

8. Non-Requirements

N1: It is not necessary for the namespace to be known by any specific fileserver.

In the same manner that clients do not need to have a priori knowledge of the structure of the namespace or its mapping onto federation members, the projected namespace can exist without individual fileservers knowing the entire organizational structure, or, indeed, without knowing exactly where in the projected namespace the filesets they host exist.

Fileservers do need to be able to handle referrals from other fileservers, but they do not need to know what path the client was accessing when the referral was generated.

N2: It is not necessary for updates and accesses to the federation data to occur in transaction or transaction-like contexts.

One possible requirement that is omitted from our current list is that updates and accesses to the data stored in the NSDB (or individual NSDB nodes) occur within a transaction context. We were not able to agree whether the benefits of transactions are worth the complexity they add (both to the specification and its eventual implementation) but this topic is open for discussion.

Below is the the draft of a proposed requirement that provides transactional semantics:

> "There MUST be a way to ensure that sequences of operations, including observations of the namespace (including finding the locations corresponding to a set of FSNs) and changes to the namespace or related data stored in the system (including the creation, renaming, or deletion of junctions, and the creation, altering, or deletion of mappings between FSN and filesystem locations), can be performed in a manner that provides predictable semantics for the relationship between the observed values and the effect of the changes."
>
> "It MUST be possible to protect sequences of operations by transactions with NSDB-wide or server-wide ACID semantics."

9. IANA Requirements

This document has no actions for IANA.

10. Security Considerations

Assuming the Internet threat model, the federated resolution mechanism described in this document MUST be implemented in such a way to prevent loss of CONFIDENTIALITY, DATA INTEGRITY and PEER ENTITY AUTHENTICATION, as described in [RFC3552].

CONFIDENTIALITY may be violated if an unauthorized party is able to eavesdrop on the communication between authorized servers and NSDB nodes and thereby learn the locations or other information about FSNs that they would not be authorized to discover via direct queries. DATA INTEGRITY may be compromised if a third party is able to undetectably alter the contents of the communication between servers and NSDB nodes. PEER ENTITY AUTHENTICATION is defeated if one server can masquerade as another server without proper authority, or if an arbitrary host can masquerade as a NSDB node.

Well-established techniques for providing authenticated channels may be used to defeat these attacks, and the protocol MUST support at least one of them.

For example, if LDAP is used to implement the query mechanism [RFC4511], then TLS may be used to provide both authentication and integrity [RFC4346] [RFC4513]. If the query protocol is implemented on top of ONC/RPC, then RPCSEC_GSS may be used to fill the same role [RFC2203] [RFC2743].

11. References

11.1. Normative References

[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.

11.2. Informational 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.

Authors' Addresses

Daniel Ellard
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
US

Phone: +1 617-873-8000
Email: ellard@gmail.com

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

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

James Lentini
NetApp
1601 Trapelo Rd, Suite 16
Waltham, MA 02451
US

Phone: +1 781-768-5359
Email: jlentini@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).