Internet Engineering Task Force (IETF)
Request for Comments: 5716
Category: Informational
ISSN: 2070-1721

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

January 2010

Requirements for Federated File Systems

Abstract

This document describes and lists the functional requirements of a federated file system and defines related terms.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5716.

Table of Contents

1. Overview

This document describes and lists the functional requirements of a federated file system and defines related terms.

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.

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 references. The namespace can either be assembled at the fileservers, the clients, or by an external namespace service, and is often not easy or uniform to manage. The requirements in this document are meant to lead to a uniform server-based namespace that is capable of spanning a whole enterprise and that is easy to manage.

We define some terms to better describe the solution space. A "fileset" is the abstract view of a filesystem in a uniform namespace, and may be implemented behind that abstraction by one or more physical filesystems at any given time. Each fileset has a name called an "FSN" (fileset name), and each physical filesystem has a fileset location ("FSL"). A fileset is a directory tree containing files and directories, and it may also contain references to other filesets. These references are called "junctions". To provide location independence, a junction does not contain information about the location of the real resource(s), but instead contains an FSN that can be used to look up the location information. The service that can be used to map from the FSN to the FSL(s) is called a namespace database (NSDB) service. The NSDB provides a level of indirection from the virtual paths in the uniform namespace to the actual locations of files. By design, the NSDB does not store the junctions. This allows junction administration and NSDB administration to be separate roles.

The servers direct clients to the proper locations by existing mechanisms (e.g., the referrals mechanism within [RFC3530] and [RFC5661]). Updates to the locations make it possible to support migration and replication of physical filesystems that comprise the namespace, in a way that is transparent to filesystem applications.

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 fileservers and NSDB nodes; in this illustration, ALPHA and BETA each have three servers, one NSDB node, and are administered separately.

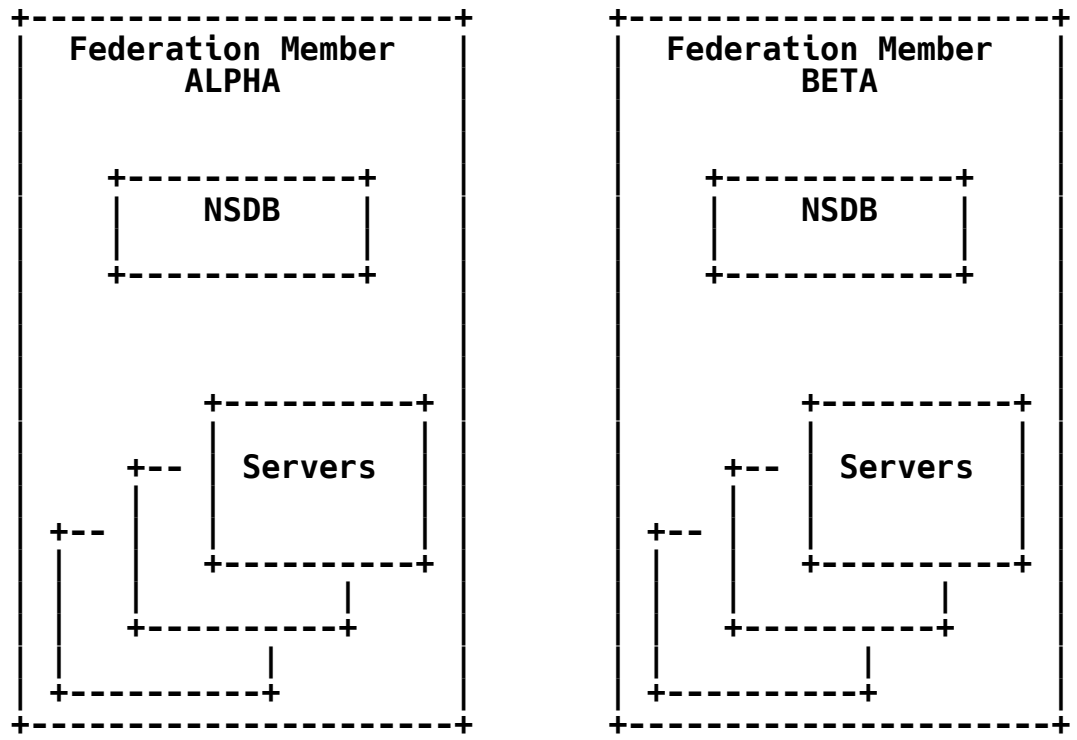

Figure 1

1.1. 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 [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. Purpose

Our objective is to specify a set of protocols by which fileservers or 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 protocols, 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 fileservers and NSDB nodes. Acting in concert, the administrators should be able to build and administer this multi-fileserver, multi-collection namespace.

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.

3. 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.

3.1. Create a Fileset and Its FSL(s)

A fileset is the abstraction of a set of files and the directory tree that contains them. 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.

3.1.1. Creating a Fileset and an 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 information and creates the initial FSL for the fileset.

3.1.2. 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 protocols 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 requirement is that these fileset replicas be registered and unregistered with the NSDB.

3.2. 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.

Step 4 is the only step that interacts directly with the federation protocols. The rest of the steps may use platform-specific interfaces.

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

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

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

4. 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.

5. The server converts one or more of the FSLs to the location type used by the client (e.g., a Network File System (NFSv4) fs_location, as described in [RFC3530]).

6. 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.

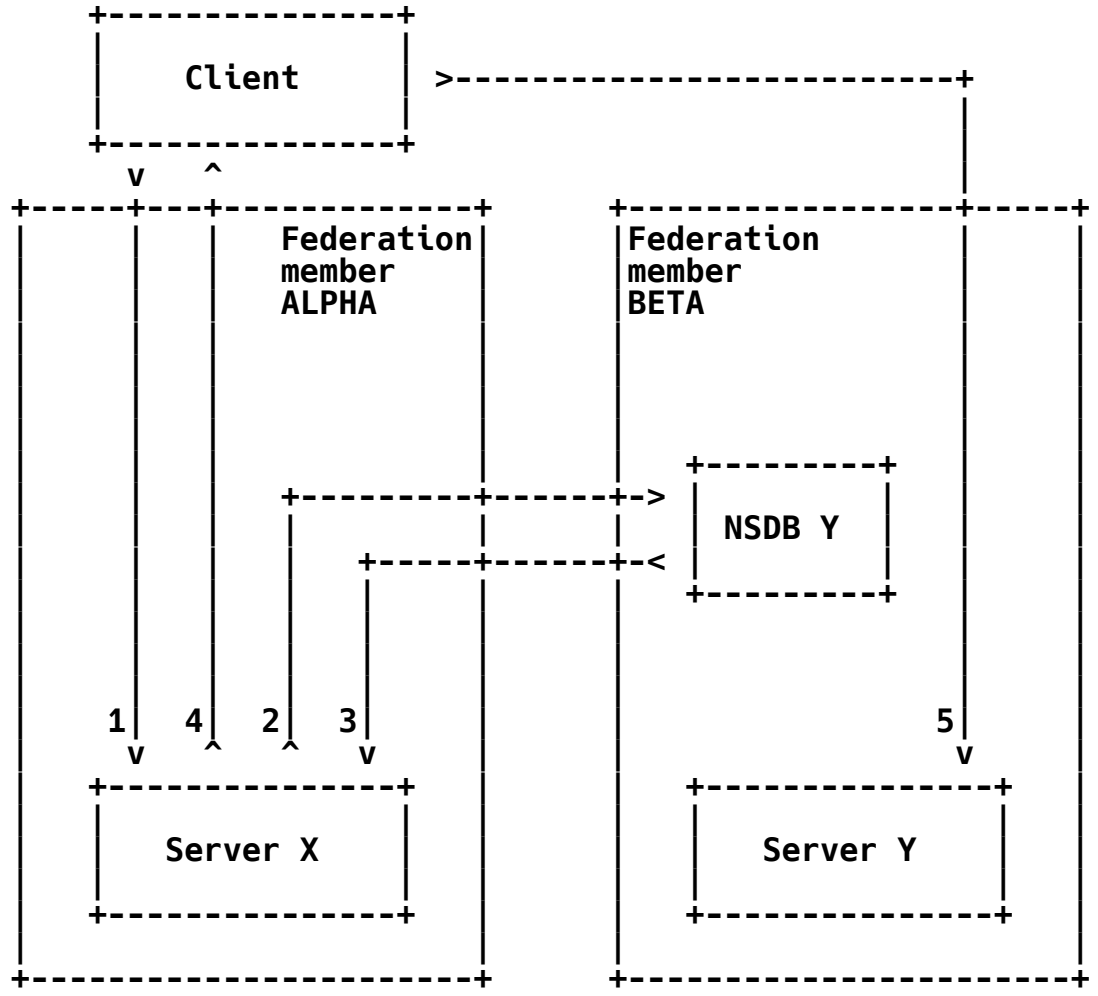

Figure 2

3.3. Junction Creation

Given a local path and the FSN of a remote fileset, an administrator can create a junction from the local path to the remote 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.

Step 1 is the only step that uses the federation interfaces. The remaining step may use platform-specific interfaces.

1. The administrator requests the server create a junction to the FSN of the remote fileset at the given path.

2. The server inserts the junction to the FSN, at the given path, into the local filesystem.

4. 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 the directory tree that contains them. 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 (Common Internet File System) [MS-SMB] [MS-SMB2] [MS-CIFS].

FSL (Fileset Location): The location of the implementation of a fileset at a particular moment in time. An FSL MUST be something that can be translated into a protocol-specific description of a resource that a client can access directly, such as an 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.

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.

5. Proposed Requirements

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

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.

5.1. 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 to which they have access.

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 an 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 Fully-Qualified Domain Name (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 FSN identifier.

   The FSN identifier is the index used by the NSDB node to identify the target fileset.

   There are several ways to represent FSN identifiers. One approach could use 128-bit Universally Unique IDentifiers (UUIDs) as described in [RFC4122].

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

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 an 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.

5.2. Requirements

R1: Requirements of each FSN:

a. Each FSN MUST be unique within the scope of its NSDB (so that the FSN is 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. All FSNs MUST be invariant when their underlying filesystems move or are replicated; only mappings from FSN to FSL(s) change under these transformations.

d. All files accessible from the global namespace MUST be part of a fileset that has an assigned FSN.

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

The FSN format MAY be of variable size. If the format is variable in size, fileserver implementations MAY have a maximum supported FSN size. By bounding the FSN size, some fileserver implementations might be able to efficiently organize FSNs in stable storage. For interoperability, the federation protocols SHOULD define an FSN size that all fileservers support.

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 a file server.

It is possible to create an FSN for a fileset that has not actually been created. It is also possible to bind a nonexistent 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 identifier) of the newly created FSL, or the administrator MAY specify only the NSDB node and have the system choose the identifier.

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 obeys 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 obeys 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 an 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:

a. 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.

b. It MAY be possible to query the fileserver named in an FSL to discover the junctions, if any, in that FSL. If this feature is implemented, the fileserver SHOULD report each junction's path within the FSL and the targeted FSN.

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

a. The namespace SHOULD be accessible to clients via versions of the CIFS (Common Internet File System) protocol as described in [MS-SMB] [MS-SMB2] [MS-CIFS].

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 document, 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 as 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 an FSN.

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

R15: It MUST be possible for the federation to project a namespace with a common root.

a. It SHOULD be possible to define a root fileset that is exported by one or more fileservers in the federation as the top level of a namespace. (Corollary: There is one root fileset per namespace and it is possible to support multiple namespaces per federation.)

b. It SHOULD be possible for a fileserver to locate an NSDB that stores the layout of a root fileset.

c. It SHOULD be possible to access, store, and update information related to a root fileset using the federation protocols.

d. It SHOULD be possible to replicate root fileset information across multiple repositories.

e. If a root fileset is defined, it SHOULD be possible to enable a fileserver to export that root fileset for client access.

f. If a root fileset is defined, it SHOULD be possible for multiple fileservers to project a common root with defined consistency semantics.

g. If a root fileset is defined, it SHOULD be stored using a compact representation that is compatible with heterogeneous fileserver implementations. The root fileset's internal format SHOULD contain enough information to generate any attributes, including referrals, required by the standard filesystem access protocols in R9.

6. 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 NSDB 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 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 Atomicity, Consistency, Isolation, and Durability (ACID) semantics.

7. 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 Lightweight Directory Access Protocol (LDAP) is used to implement the query mechanism [RFC4510], then Transport Layer Security (TLS) may be used to provide both authentication and integrity [RFC5246] [RFC4513]. If the query protocol is implemented on top of Open Network Computing / Remote Procedure Call (ONC/RPC), then RPCSEC_GSS may be used to fill the same role [RFC2203] [RFC2743].

A federation could contain multiple Public Key Infrastructure (PKI) trust anchors [RFC5280]. The federation protocols SHOULD define a mechanism for managing a fileserver's NSDB trust anchors [TA-MGMT-REQS]. A general purpose trust anchor management protocol [TAMP] would be appropriate, though it might be desirable for the federation protocols to facilitate trust anchor management by, for example, using trust anchor interchange formats [TA-FORMAT].

It is useful to note that the requirements described in this document lead naturally to a system with distributed authorization, which has scalability and manageability benefits.

FSNs are likely to be long-lived resources. Therefore, the privilege to create FSNs SHOULD be carefully controlled. To assist in determining if an FSN is referenced by a junction somewhere in the federation, the NSDB records SHOULD include non-authoritative informational annotations recording the locations of any such junctions. These annotations are non-authoritative because a

junction might be created, deleted, or modified by an individual that does not have permission to modify the NSDB records.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[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.

[RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System Version 4 Minor Version 1", RFC 5661, January 2010.

8.2. Informative References

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol Specification", MS-CIFS 2.0, November 2009.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", MS-SMB 17.0, November 2009.

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2 Protocol Specification", MS-SMB2 19.0, November 2009.

[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.

[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.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[TA-FORMAT] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", Work in Progress, October 2009.

[TA-MGMT-REQS] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", Work in Progress, September 2009.

[TAMP] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", Work in Progress, December 2009.

Appendix A. Acknowledgments

We would like to thank Robert Thurlow of Sun Microsystems for helping to author this document.

We would also like to thank Peter McCann and Nicolas Williams for their comments and suggestions.

Authors' Addresses

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

Phone: +1 781-768-5359
EMail: jlentini@netapp.com

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

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

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

Phone: +1 617-873-8000
EMail: dellard@bbn.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