Network Working Group
Internet Draft
Document: draft-ietf-nfsv4-repl-mig-design-00.txt

Robert Thurlow
December 2002

Server-to-Server Replication/Migration Protocol Design Principles

Status of this Memo

Abstract

NFS Version 4 [RFC3010] provided support for client/server interactions to support replication and migration, but left unspecified how replication and migration would be done. This document discusses the nature of a protocol to be used to transfer filesystem data and metadata for use with replication and migration services for NFS Version 4.

Table of Contents

1. Introduction

Though used in different circumstances, replication of data and migration of data share a common problem: how to accurately transfer data (which may be in use by applications) from one location to another with reasonable bandwidth usage and in reasonable time. Years ago, this was done by taking storage offline (or at least preventing write access), making a tape copy of the data files, and walking it to the new machine, after warning the twenty or so people who cared about it. Networks reduced wear on sneakers, but many of the data formats we use for filesystem copies tend to be little improved - they are either lowest-common-denominator standards like "tar" and "cpio" or internal dump formats which are non-standard. Today, with distributed filesystems like NFS Version 4, richer metadata including Access Control Lists (ACLs) and extended attributes, and potential users all over the enterprise and the Internet, we need something better - a standard, complete and extensible protocol to transfer filesystems. Functionality like this has been available in AFS and DCE/DFS for many years, though the protocols have not been published.

Though data replication and transfer are needed in many areas, this document will focus primarily on solving the problem of providing replication and migration support between NFS Version 4 servers. It is assumed that the reader has familiarity with NFS Version 4 [RFC3010].

1.1. Definitions of terms

1.1.1. Replication

Filesystem replication is the creation of a functionally identical copy of a filesystem, usually to enhance availability or provide for redundancy or disaster recovery. For example, a company may set up replicas of a customer database accessed by employees in different geographies. The data sets are often read-only, and initial creation of a replica is not as interesting a problem as maintaining the replica efficiently over time via incremental updates, which will likely be set up to push automatically.

1.1.2. Migration

Filesystem migration is the moving of a filesystem to another server for load balancing purposes or because a user or server has moved. For example, a user may have moved from one building to another, or across the country, and want his home directory to follow him, or it may just be time to decommission an old server and move data to a new

one. Only one data transfer is done, and it is important for this to be done efficiently and with the lowest possible impact on users.

1.2. Current practice

System administrators typically have several options available to them to replicate or migrate files, but none of them cover the problem space:

- The pax, cpio and tar tape archivers as defined by IEEE 1003.1 or ISO/IEC 9945-1 are often used without tape over a network for data transfer; these support only generic Unix-specific metadata and do not support ACLs or extended attributes

- The rdist (http://www.magnicomp.com/rdist) and rsync (http://samba.anu.edu.au/rsync) applications focus on propagating changes to replicas, but are documented only by source code, are not available on all platforms, and do not support more than generic Unix-specific metadata

- "cp -r" or its equivalent over NFS Version 4 could work in cases where capabilities of servers were the same, but if the destination did not support ACLs or extended attributes, would it do what the user wanted?

- Most server filesystems have a "dump" format of some kind, which can preserve all data and metadata as long as there are no architectural differences in the servers

- Most server vendors have products which can keep replicas in sync by monitoring changes at the block level below the server filesystem, which are again inherently tied to one architecture

- Most of the above tools are not set up to properly deal with exotic metadata which may be present on filesystems like MacOS's HFS or Windows' NTFS, which can result in loss of data even when transferring to the same platform

1.3. The problem

1.3.1. NFS clients today

Replication and migration events both cause problems for NFS clients, which may have applications operating on data when the event occurs. Past versions of NFS did not provide any support in protocol for the client, and typical clients did not even attempt to find another

replica which might provide service.

1.3.2. NFS Version 4

NFS Version 4 [RFC3010] introduced some extra error codes and attributes to improve this situation. For replication, the new "fs_locations" attribute could be retrived by the client to determine if multiple locations were available, so that when a server became unavailable, the client could fail over to a new location without hoping updated information was available in its name service. For migration and in the case of a decommissioned replica, the NFS4ERR_MOVED error would inform a client that it should consult "fs_locations" and make contact with a new server responsible for the data. In both cases, a client is required to establish a relationship with a new server, which may involve state recovery and using saved pathname information to discover new filehandles.

1.4. The need for a transfer protocol

To support NFS Version 4, a method is needed to transfer functionally complete filesystem data from one server to another. The shortcomings listed previously in the common tools in use demonstrate that there is value in a standard protocol to transfer filesystem data.

2. Requirements

The requirements for a replication and migration protocol are to be addressed in a separate document, but are approximately these:

2.1. Interoperability

The replication/migration protocol must first and foremost be one which can potentially be implemented on any server. Several vendors already have a replication mechanism in their product lines which takes advantage of known properties of their servers to replicate at the block level, but this is inherently tied to one system.

2.2. Transparency

When a client has been using a file which has been migrated, it should be able to detect this and recover the file state on the new server without applications needing to take action. Similarly, when a client has availability problems with a particular replica, it should be able to adapt to the use of the new replica without application involvement. This implies that, as far as possible, the replication/migration protocol must copy all filesystem data, as much

metadata as possible, and all non-recoverable transient state such as outstanding lock and delegation state, completely and correctly. It is acceptable that the client must recover some state as occurs in the event of a server reboot.

2.3. Security

NFS Version 4 supported strong mandatory-to-implement security mechanisms to protect the integrity and privacy of file data and metadata. The replication/migration protocol must specify mandatory-to-implement security to protect data in transit, and provide a security payload and an encryption mechanism to ensure strong security for each message. It is expected that the security mechanisms will correlate well with NFS Version 4 [RFC3010].

2.4. Efficiency

The replication/migration protocol must get the job of data movement done as efficiently as possible in terms of both bandwidth and time. Components of this are:

- the protocol will conserve bandwidth by streaming data in large blocks with limited header overhead
- the protocol will transfer changed regions in files rather than complete files whenever possible
- the protocol will permit restart in the event of a server failure or lost connection

2.5. Scalability

The replication/migration protocol must be able to handle both huge files and huge filesystems, while maintaining low enough overhead to work well with small filesystems as well.

3. Non-requirements

There have been discussions about the things a good replication protocol could do which are not considered part of the scope of this work at this time. Some of these things could be specified by future RFCs, so we do not wish to preclude them from being done later on. These non-requirements include:

- being an "rdist" or "rsync" replacement
- being a tool to permit unprivileged users to copy file trees

- being used for replication of other types of data

4. Design considerations

4.1. Basic structure

For best performance, a replication/migration protocol should be able to move large amounts of data without frequent small packets in the direction of data movement. Use of RPC [RFC1831] and XDR [RFC1832] will be subject to analysis of their overhead for this purpose. However they are formatted, groups of messages would probably include:

- Initialization and negotiation messages
- Filesystem information messages
- Data transfer messages
- Finalization messages

4.2. Administrative Control

The replication and migration protocol should include nothing specifying how an administrative user contacts a server to initiate replication or migration. A separate document should define a mechanism suitable for this purpose.

4.3. Basic environment

The replication/migration protocol should be available to a privileged context on a well-known TCP port on an NFSv4 server, able to authenticate and act on control messages from administration clients and general messages from other servers.

4.4. Handling file changes

For replication, it should be possible to handle large files changed in small ways without transferring the entire file. The protocol needs to be able to express changes to byte ranges within a file; ideally, the server will be able to extract such changes from some kind of change log or from internal filesystem data. However, this may not be practical. The existence of "rdist" shows that a bidirectional protocol can determine differences in files at a reasonable bandwidth cost, and it would be good for the replication/migration protocol to be able to operate in this mode.

4.5. Replication model

Replication is usually set up as a series of read-only replicas, with the master copy of the filesystem generally unaccessible to the client or accessible through a different mount point. It is possible to envision a case where, along with several read-only replicas, a single writer is available and "marked" as such in the fs_locations attribute. The client would have to ensure that all reads and writes were directed to the writable copy from the time a particular file on the filesystem was first written to the time the client ceased caring about the file. This is considered beyond our current scope at this time.

5. Security considerations

NFS Version 4 is the primary impetus behind a replication/migration protocol, so this protocol should mandate a strong security scheme and security negotiation in a manner compatible with NFS Version 4. Since NFS Version 4 specifies RPCSEC_GSS [RFC2203], which in turn builds on GSS-API [RFC2078], it makes sense for a replication/migration protocol to specify RPCSEC_GSS if it is based on RPC, and GSS-API if it is not based on RPC. Kerberos Version 5 will be used as described in [RFC1964] to provide one security framework. The LIPKEY GSS-API mechanism described in [RFC2847] will be used to provide for the use of user password and server public key. An initial message exchange will permit security negotiation. The replication/migration protocol will also specify a NULL security mechanism to optimize its performance when used with strong host-based security mechanism such as SSH and IPSec.

6. Implementation considerations

6.1. Filehandle preservation

Filahandles are the basic shorthand used by clients to perform most operations on files. The are opaque to the client, but are usually derived from:

- the fsid of the filesystem
- the fileid or "inode number" of the directory shared by the server
- the fileid or "inode number" of the file
- the "generation number", an internal field to support inode reuse.

It is, in some circumstances, desireable to preserve persistant filehandles across a replication or migration event. The most likely circumstance for this is when both servers are of the same architecture, and when the destination server can assign values to these fields as data is accepted. To support this case, the filehandle should be available as an attribute which can be passed to the new server. Some operating environments will not have interfaces to support access to this data or a way to recreate it anew, so this should be negotiated so that this data is not sent unnecessarily.

Even if a server implementation can transfer and accept persistent filehandles, it must ensure that the client is not falsely promised that this will happen. [RFC3010] specifies that a server may migrate a filesystem with persistent filehandles as long as the new server also uses persistent filehandles and the same filehandles will correspond to the same files after migration. In the general case, the decision to migrate a filesystem, perhaps to a heterogeneous server with different filehandles, will be made after clients have accessed filesystems and learned of the value of the "fh_expire_type" attribute. Thus it seems necessary that servers return an "fh_expire_type" of at least FH4_VOL_MIGRATION so that clients will always store partial pathnames for later use. It is possible for clients to attempt to use pre-event filehandles with the new server in the hope that persistent filehandles would have been transferred intact, but there is no way for the server to promise this unless it will never transfer to a server of a different implementation.

6.2. Data transfer phases

For both replication and migration, transfer most generally happens in two phases: first, the bulk of the data is copied to the target while access to the source filesystem continues, and second, changes made since the start of the first phase are transferred while write access to the source filesystem is curtailed. This reduces the window during which clients will see restrictions, at the cost of needing a method to lock out writes to files in the file tree. For replication, it would be possible to bypass locking by the use of multiple point-in-time copies ("snapshots"), since the delta represented by each snapshot could be used to update the replicas.

6.3. Operation on filesystem subsets

When NFSv4 clients discover that they must react to a replication or migration event, [RFC3010] states that they will recover at the granularity of an entire filesystem, i.e. a set of files sharing the same "fsid" attribute. It is possible that this protocol could be useful for splitting up of large filesystems to permit them to be replicated and migrated separately. This can most easily be done if

the server can arrange to return distinct "fsid"s for subdirectories of what it manages as a single filesystem.

7. Difficult issues

7.1. Transparency violations

When being used between servers that are sufficiently different, it may be impossible for the new server to support some metadata enumerated in the data stream, or it may be that metadata critical to the new server are not supported on the old. When this happens, the client may notice and react badly to the loss of transparency. Sources of this kind of problem include:

- Filename encoding differences
- Attributes supported on one server and not the other
- A failure of atomicity during transfer
- Incomplete or no transfer of locking, delegation and other state

7.2. Directory access

When a directory is read, a series of RPCs is used to get the entries in small parts. The sequence of RPCs is tied together by a "cookie" returned by the server in each reply and used by the client in the next request. The sequence can be interrupted by a replication or migration event, which can lead to NFS4ERR_BAD_COOKIE on the new server, even if the servers are the same architecture, due to different orders of creation of the directory entries and compaction.

8. Bibliography

[RFC1831]
R. Srinivasan, "RPC: Remote Procedure Call Protocol Specification Version 2", RFC1831, August 1995.

[RFC1832]
R. Srinivasan, "XDR: External Data Representation Standard", RFC1832, August 1995.

[RFC3010]
S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck, "NFS version 4 Protocol", RFC3010, December 2000.

[RDIST]
MagniComp, Inc., "The RDist Home Page", http://www.magnicomp.com/rdist.

[RSYNC]
The Samba Team, "The rsync web pages", http://samba.anu.edu.au/rsync.

9. Author's Address

Address comments related to this memorandum to:

nfsv4-wg@sunroof.eng.sun.com

Robert Thurlow
Sun Microsystems, Inc.
500 Eldorado Boulevard, UBRM05-171
Broomfield, CO 80021

Phone: 877-718-3419
E-mail: robert.thurlow@sun.com