# DEVELOPING AND INTEGRATING A CLUSTERING PLUGIN FOR NFS-GANESHA

[1]Rohan Koul , [2]Ashutosh Bhadoria , [3]Sukhada Bhingarkar, [4]Nikhil Ajgaonkar , [5]Aditya Kamath
[1,2,4,5]B.E Student, [3]Assistant Professor
Department of Computer Engineering
MIT College of Engineering , Pune
Email:[1]rohansainath@gmail.com , [2]ashu.f1117@gmail.com, [3]sukhada.bhingarkar@mitcoe.edu.in, [4]nikhilnma@gmail.com, [5]adikamath.99@gmail.com

**Abstract— NFS-GANESHA is an NFS version 2-4 server that runs in the user address space instead of as part of the operating system kernel. GANESHA is not a replacement for the NFSv4 server implemented in the kernel; it is a brand new program, with its own advantages and disadvantages. Using the NFS protocol in user mode provides huge benefits like allocating large pieces of memory, making internal caches , portability etc . Inspite of having so many benefits NFS GANESHA server is still a single point of failure in a network. Should a server fail , all the clients serviced by the server are denied service leading to huge losses . This paper proposes a clustering support for existing NFS GANESHA server . We propose to provide communication links between existing servers and transform them into a cluster such that each node is aware about the working of all other nodes . If a node fails, control is rapidly transferred to another node in transparent manner so that the client does not face any hindrance .**

**Index Terms— CTDB , Clustering , Distributed File System , NFS-GANESHA , Samba ,VFS**

## I. INTRODUCTION

Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984[1] , allowing a user on a client computer to access files over a network much like how a local storage is accessed. NFS is a client/server application that lets a computer user view and optionally store and update file on a remote computer as though they were on the user's own computer. This file access is completely transparent to the client, and works across a variety of server and host architectures. The user's system acts as a NFS client while the system from where data is to be accessed or manipulated acts as the NFS server. Both of them require that you also have TCP/IP installed since the NFS server and client use TCP/IP as the program that sends and updates the files back and forth.

When someone accesses a file over NFS, the kernel places an RPC call to nfsd (the NFS daemon) on the server machine. This call takes the file handle, the name of the file to be accessed, and the user's user and group id as parameters. These are used in determining access rights to the specified file. In order to prevent unauthorized users from reading or modifying files, user and group ids must be the same on both hosts. On most implementations, the NFS functionality of both client and server are implemented as kernel-level daemons that are started from user space at system boot. These are the NFS daemon (nfsd) on the server host, and the Block I/O Daemon (biod) running on the client host.

Currently NFS server is implemented in the kernel space as opposed to a userspace application. Implementing NFS server in kernel space gives a boost to performance as well as interoperability. In general, a kernel server's strengths are closer integration with the vfs and the exported filesystem. We can make up for that by providing more kernel interfaces (such as the filehandle system calls), but that's not easy. On the other hand, some of the filesystems people want to export these days (like gluster) actually live mainly in userspace. Those can be exported by the kernel nfsd using FUSE but again extensions to the FUSE[2] interfaces may be required for newer features, and there may be performance issues. Due to above reasons the kernel-space NFS server is preferred over the user-space one.

NFS-GANESHA is a NFS version 2-4 server that runs in the user address space instead of as part of the operating system kernel. Filesystem in Userspace (FUSE) lets you run a filesystem in the user address space instead of as part of the Linux kernel, but the FUSE support in the Linux kernel from many Linux distributions does not allow you to export FUSE through NFS. NFS-GANESHA lets you expose FUSE through NFS without patching your kernel. NFS-GANESHA accesses the underlying data through a File System Abstraction Layer (FSAL)[3] , allowing you to plug in your own storage mechanism and access it from any NFS client. NFS-GANESHA provides a FUSE-compatible FSAL to allow you to quickly access a FUSE filesystem over NFS while avoiding the need for data to bounce through the kernel FUSE mechanism on the NFS server. Inspite of all this NFS-GANESHA is still a single point of failure . Should a server fail , all the clients connected to it get halted and the whole network collapses . This paper proposes a innovative and efficient solution in the form of an clustering plugin to the existing servers . A communication link is created amongst existing servers to form a cluster so that every server is aware about the working status of all other servers . Should one of the servers fail control is transferred to another server within the cluster . This switching happens in the background without the client being aware of it . Hence the client doesnt face any hindrance in work in case of a node failure .

The paper is organized as follows : Section II deals with Literature survey of the various technologies closely related to NFS-GANESHA. Section III presents a brief description of the various modules used in GANESHA . Section IV proposes an innovative design strategy for providing clustering support to GANESHA . Section V concludes the paper .

## II. LITERATURE SURVEY

NFS protocol has been in existence for a long time now . Sun used version 1 only for in-house experimental purposes. When the development team added substantial changes to NFS version 1 and released it outside of Sun, they decided to release the new version as v2, so that version interoperation and RPC version fallback could be tested [4] . As time passed by further refinements and additional features were incorporated into the protocol which led to development of NFSv3 [5] and NFSv4 [6] . Version 3 added support for 64-bit file sizes and offsets, support for asynchronous writes on the server, additional file attributes in many replies . Version 4 added strong security, and introduces a stateful protocol. Version 4 became the first version developed with the Internet Engineering Task Force (IETF) after Sun Microsystems handed over the development of the NFS protocols. NFS version 4.1 aims to provide protocol support to take advantage of clustered server deployments including the ability to provide scalable parallel access to files distributed among multiple servers (pNFS extension). NFS version 4.2 is currently being developed [7].

A . Samba Server

Samba[8] is important to understand for understanding of development of protocols in user-space . Samba server runs on Unix and Linux/GNU operating systems. Windows clients can talk to Linux/GNU/Unix systems through Samba server. It provides the interoperability between Windows and Linux/Unix systems. Initially it was created to provide printer sharing and file sharing mechanisms between Unix/Linux and Windows. As of now Samba project is doing much more than just file and printer sharing like autorization, authentication , service announcement etc . Samba server basically works as a semantic translation engine/machine. Windows clients talk in Windows syntax e.g. SMB protocol. And Unix/Linux/GNU file-systems understand requests in POSIX. Samba converts Windows syntax to Unix/GNU syntax and vice versa. Samba code is very modular in nature. Samba VFS code is divided in to two parts i.e. Samba VFS layer and VFS modules. The purpose of Samba VFS layer is to act as an interface between Samba server and below layers. When Samba server get requests from Windows clients through SMB[9] protocol requests, it passes it to Samba VFS modules. Samba VFS modules i.e. plugin is a shared library (.so) and it implements some or all functions which Samba VFS layer i.e. interface makes available. Samba VFS modules can be stacked on each other(if they are designed to be stacked). Samba VFS layer passes the request to VFS modules. If the Samba share is done for a native Linux/Unix file-system, the call goes to default VFS module. The default VFS module forwards call to System layer i.e. operating system. For User space file-system like GlusterFS, VFS layer calls are implemented through a VFS module i.e. VFS plugin for GlusterFS .The plugin redirects the requests (i.e fops) to GlusterFS[10] APIs i.e. libgfapi. It implements or maps all VFS layer calls using libgfapi. Fig.1 is the schematic diagram of how communication works between different layers.

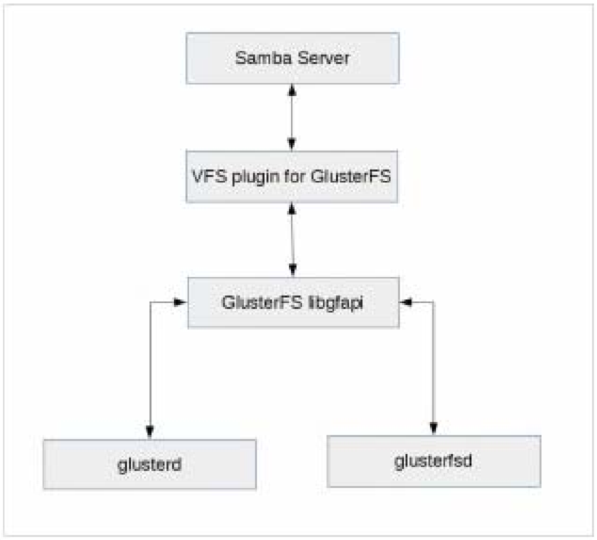

Figure 1 : Working of SAMBA

The client requests come to Samba server and Samba servers redirects the calls to GlusterFS's VFS plugin through Samba VFS layer. VFS plugin calls relevant libgfapi fucntions. Libgfapi acts as a client, contacts glusterd for vol file information ( i.e. information about gluster volume, translators, involved nodes) , then forward requests to appropriate glusterfsd i.e. brick processes where requests actually get serviced.

Without GlusterFS VFS plugin, we can still share GlusterFS volume through Samba server. This can be done through native glusterfs mount i.e. FUSE (file system in user space). We need to mount the volume using FUSE i.e .glusterfs native mount in the same machine where Samba server is running, then share the mount point using Samba server. As we are not using the VFS plugin for GlusterFS here, Samba will treat the mounted GlusterFS volume as a native file-system. The default VFS module will be used and the file-system calls will be sent to operating system. The flow is same as any native file system shared through Samba.

B . Filesystem Abstraction Layer :

This abstraction layer enables an operating system to support different filesystems, even if they differ greatly in supported features or behavior. Such a generic interface for any type of filesystem is feasible only because the kernel itself implements an abstraction layer around its low-level filesystem interface. This is possible because the VFS provides a common file model that is capable of representing any conceivable filesystem's general features and behavior. This abstraction layer enables the kernel to support many types of filesystems easily and cleanly. The figure below shows the actual working of VFS . The clients connected to NFS-GANESHA can be using different types of filesystems , which can be very dissimilar in their nature and implementation . The FSAL sits on top of the distributed file system and acts as a translation engine to make requests to the underlying file system . This layer performs trasnslation according to the type of underlying filesystem protocol . It is a very important part in creating a portable distributed server .

C . CLUSTER DRC

The NFS reply cache, also known as the Duplicate Request/Reply Cache (DRC)[11] helps a server to identify duplicate requests and avoid repeated processing of non-idempotent operations. Generally, idempotent operations can be safely repeated and will cause no harm, but non-idempotent operations, can only be executed once. For example exclusive create can be a success on first attempt,

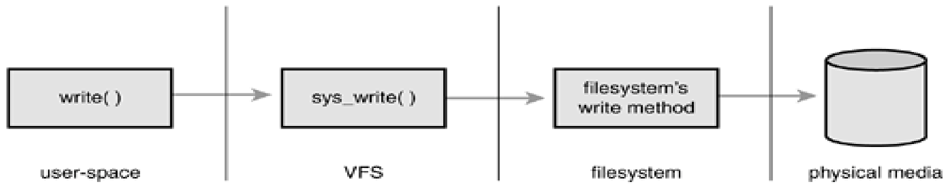

FIG.2
WORKING OF FSAL

but if the same operation is retried, it will fail because the file is already created. NFS-Ganesha maintains a Duplicate Reply Cache in node's local memory. In a clustered environment, there are issues because the duplicate reply cache is not cluster-aware and in the case of recovery or fail-over, if the NFS request is replayed on another server, then that server has no knowledge of the reply cache and may result in incorrect behavior. This clustered DRC design is based on "best effort". It does not make any guarantees that the all the DRC entries will synced to the backup node and will be available immediately after fail-over. It could happen that there are some DRC entries that never make it to the backup server before the server fails.

D . CMAL **(** Cluster Management Abstraction Layer **) :**

The CMAL, provides an abstraction layer for cluster manager support. It is a generic layer that would interface with different backend cluster managers. The backend cluster manager will be a dynamically linked layer. So the CMAL layer would have function pointers based on the CMAL that is dynamically linked. The backend CMAL would interface with the available cluster manager .

The CMAL has following basic functions associated with it :

**init_cmal :** Initialize the CMAL interface. Set up backup node information.

**add_drc_entry(xid, entry) :** Stores the DRC entry in the cluster. This functionality will be provided by the backend CMAL layer, which will talk to the available cluster manager.

**retrieve_drc_entries(ip_address) :** Retrieve all the DRC entries for a particular sever node.

**shutdown_cmal() :** Shutdown the CMAL interface **.**

## III. NFS-GANESHA ARCHITECTURE

GANESHA is designed as a layered product. Each layer is a module dedicated to a specific task. Data and meta-data caching, RPC, SEC_GSS and protocol management, accessibility to the file system, etc . All these functionalities are handled by specific modules. Each module has a well-defined interface . Such a modular design is good for future code maintenance. Furthermore, one can write new. algorithms within a layer leaving the rest of the source code unchanged . Fig 3 shows the architecture of GANESHA .

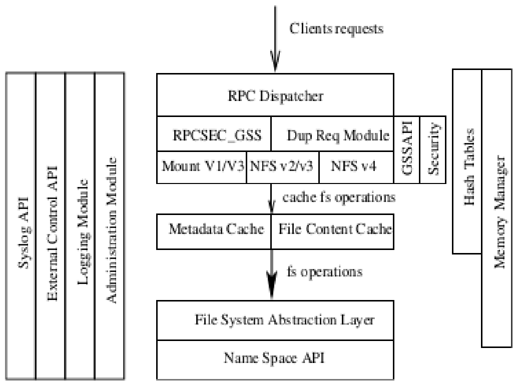

FIG 3 Architecture of NFS-GANESHA

The various modules which are used by GANESHA are as follows :

A. Managing memory

Memory management is one of the most crucial task . Almost all modules within GANESHA's architecture will have to perform dynamic memory allocation. For example, a thread managing a NFS request may need to allocate a buffer for storing the requested result. If the regular LibC malloc/free calls are used, there are risks of fragmenting memory because some modules will allocate large buffers when others will use much smaller ones. This could lead to a situation where part of the memory used by the program is swapped to disk, and performance would quickly drop.

For this reason, GANESHA implements its own memory manager. This module, which is used by all the other parts of GANESHA, allows each thread to allocate its own piece of memory at startup. When a thread needs a buffer, it will look into this space to find an available chunk of the correct size. This allocation is managed by the Buddy Malloc algorithm, the same that is used by the kernel .

B. Managing the CPU resource :

The second most important resource is CPU which is easier to manage than memory . GANESHA ismassively multi-threaded, and will have dozens of threads at the same time POSIX calls for managing threads help us a lot here, we can use them to tell the Linux scheduler not to manage the pack of threads as a whole, but to consider each of them separately.[5]. With a multi-processor machine, such an approach will allow the workload to "spread across" all of the CPUs. What is also to be considered is potential deadlocks. In a multi-threaded environment, it is logical to have mutexes to protect some resources from concurrent accesses. But having bunches of threads is not useful if most of them are stuck on a bottleneck. Design considerations were taken into account to avoid this situation.

First, reader/writer locks were preferred to simple mutexes. Because the behavior of reader/ writer locks may differ from one system to another, a small library was written to provide this service (which was a required enhancement in terms of portability). Second, if threads share resources, this common pool could turn to a bottleneck when many threads exist together. This was avoided by allocating resources per thread. This consideration has a strong impact on the threads' behavior, because there can't be a dedicated garbage collector. Each thread has to perform its own garbage collection and has to reassemble its resources regularly.

C. The Hash Tables: a core module for associative addressing :

Associative addressing is a service that is required by many modules in GANESHA—for example, finding an inode knowing its parent and name, or finding the structure related to a NFSv4 client, knowing its client ID. The API for this kind of service is to be called very often: it has to be very efficient to enhance the daemon's global performance. The choice was made to use an array of Red- Black Trees[12]. RBTs have an interesting feature: they re-balance themselves automatically. after add/update operations and so stay wellbalanced. RBTs use a computed value, defined as the RBT value in this document, to identify a specific contiguous region of the tree. Bottlenecks could occur if a single RBT is used: several threads could perform add/update operations at the same time, causing a conflicting re-balance simultaneously. It then appears that RBTs are to be protected by read/writer locks and this could quickly become a bottleneck. Working around this issue is not difficult: using several RBTs (stored in an array) will solve it. If the number of RBTs used islarge (more than 15 times bigger) that the number of concurrent threads that can access them, then the robability of having two of them working on the same tree becomes pretty small.

D. Huge Cache Management :

GANESHA uses a large piece of memory to build large caches. Data and meta-data caches will be the largest caches in GANESHA. Let's focus first on the meta-data cache, located in the Cache Inode Layer. Each of itsentries is associated with an entry in the namespace (a file, a symbolic link, or a directory [13] ).This entry is itself associated with a related object in the File System Abstraction Layer identified by a unique FSAL handle. The meta-data cache layer will map in memory the structure it reads from the FSAL calls, and it tries to keep in memory as many entries as possible, with their parent-children dependencies. Meta-data cache use hash tables intensively to address the entries, using the FSAL handle to address the entry associatively. With the current version of GANESHA, a simple write-through cache policy is implemented.

The data cache is not managed separately: if the content of a file is stored in data cache, this will become a characteristic of the meta-data cached entry. The data cache is then a 'child cache' to the meta-data cache: if a file is datacached, then it is also meta-data cached. This avoid incoherencies between this two caches since they are two sides of the same coin. Contents of the files which are cached are stored in dedicated directories in a local file system. A data-cache entry will correspond to two files in this directory: the index file and the data file. The index files contain the basic metadata information about the file; the most important one is its FSAL handle. The data file is the actual data corresponding to the cached file. The index file is used to rebuild the data-cache, in the event that the server crashes without cleanly flushing it: the FSAL Handle willbe read from this file and then the corresponding meta-data cache entry will be re-inserted as well, making it point to the data file for reconstructing the data cached entry.

## IV. PROPOSED ARCHITECTURE/DESIGN

In the current scenario NFS-GANESHA server is a single point of failure with no scope of recovery . This can prove to a problem in a scenario where a lot of important data is being accessed and transferred to and from the server by the clients . If the server breaksdown due to some reason a lot of important data can be lost . The entire network will be halted and no fruitful work can be done until the server is up again . Currently there is an FSAL layer which sits on top of the shared file system to process the requets of the clients . We propose a design to impart clustering support to the existing NFS – GANESHA servers . The basic idea is to establish communication links between a group of servers to form a cluster . As a result each node in the cluster will be aware about the status of all other nodes . Also messages can be sent from one server to another . Fig 4 shows the basic architecture of the proposed design .

A. Stackable FSAL

NFS-GANESHA works normally by having a FSAL sit on top of Distributed file system which acts as a middle layer to process the requests of the client . In order to add a clustering plugin we need to stack multiple FSAL layers on top of each other . There is a dummy stackable FSAL called FSAL_NULL in the current version of

NFS-GANESHA which serves no purpose as of now . We propose to use this design to stack FSAL_CTDB layer on top of the existing FSAL layer . The purpose of this layer is to add clustering support to the servers . We also propose to use the inbuilt features of Clustered Trivial Database for managing the cluster . The Clustered TDB is a shared TDB approach to distributing locking state. In this approach, all cluster nodes access the same TDB files. CTDB provides the same types of functions as TDB but in a clustered fashion, providing a TDB-style database that spans multiple physical hosts in a cluster. The cluster filesystem takes care of ensuring the TDB contents are consistent across the cluster . CTDB also provides failover mechanism to ensure data is not lost if any node goes down while serving data. It does this with the use of Virtual IP addresses or Public IP addresses .

The information regarding the state of all the servers is stored in small databases called Trivial Databases . Samba used these databases to maintain information like password , mapping information , session information . CTDB extends this concept to share this information across a cluster . All these databases are shared by all nodes such that each one of them can access and update the records. This means that these databases must be stored in the shared file system . To make it simpler, each node of the cluster has CTDB daemon ctdbd running and will have a local, old-style tdb stored in a fast local filesystem. The daemons negotiate only the metadata for the TDBs over the network. The actual data read and writes always happens on the local copy. Ideally this filesystem will be in-memory, such as on a small ramdisk, but a fast local disk will also suffice if that is more administratively convenient. This makes the read write approach really fast. The contents of this database on each node will be a subset of the records in the CTDB (clustered tdb). However, for Persistent database, when a node wants to write to a persistent CTDB, it locks the whole database on the network with a transaction, performs its read and write, commits and finally distributes the changes to all the nodes and write locally too. This way the persistent database is consistent across all nodes .

CTDB uniquely identifies each of the nodes in the cluster by the Virtual Node number, VNN. It maps the physical addresses with the VNN. Also, CTDB works with two IP networks. The internal network on the InfiniBand for the CTDB communication between the nodes. This is same as that for the clusters internal network for communication between the nodes. The second type is the public addresses through which the clients access the nodes for data.

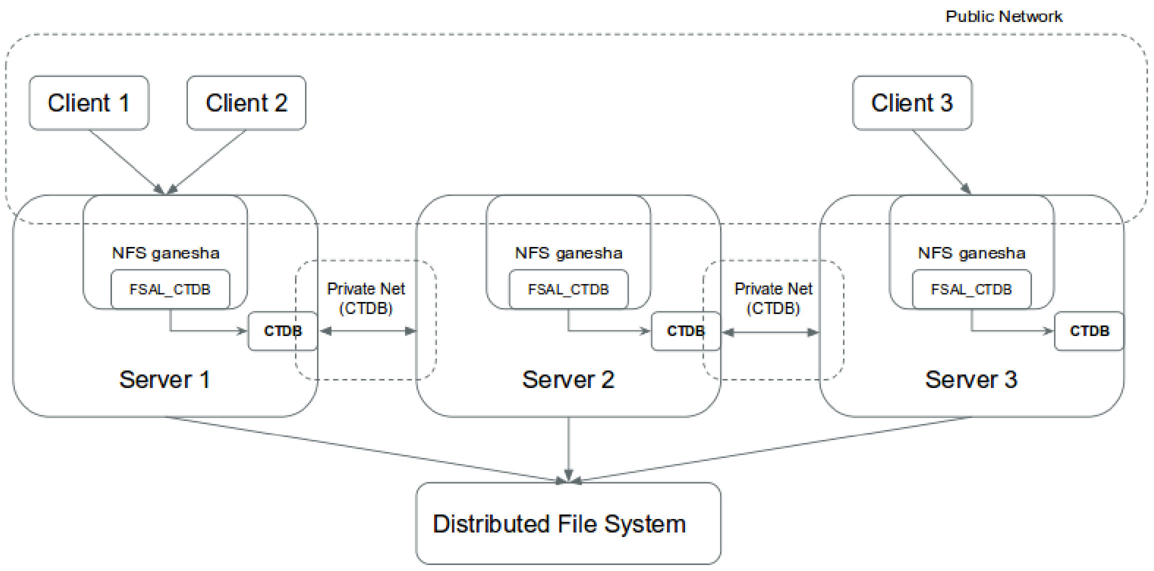

FIG 4 Proposed architecture for NFS-GANESHA

## V. CONCLUSION

NFS GANESHA is a latest one in the series of NFS servers . Its large cache management capability allowed an increase of the incoming NFS requests on the related machines, a need that was critical for several other projects . At the same time being developed in the user space has provided a great scope for development of many more features by the external developers . The proposed clustering support can help to ensure consistency and provide failover mechanism to the existing servers which will go a long way in making GANESHA one of the most robust and efficient servers.

## REFERENCES

[1] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, Bob Lyon (1985). "Design and Implementation of the Sun Network Filesystem”

[2] Aditya Rajgarhia , Ashish Gehani , “Performance and Extension of User Space File Systems” , SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing Pages 206-213

[3] “Filesystem Abstraction Layer” Available online at : http://www.makelinux.net/books/lkd2/ch12lev1sec2

[4] NFS Illustrated (2000) by Brent Callaghan - ISBN 0-201-32570-5

[5] Callaghan, B., Pawlowski, B. and P.Staubach, “NFS Version 3 Protocol Specification,” RFC 1813, The Internet Society, June, 1995.

[6] S. Shepler, B. Callaghan, D. Robinson, Sun Microsystems Inc., C. Beame, Hummingbird Ltd., M. Eisler, D. Noveck, Network Aplliance Inc. “Network File System (NFS) version 4 Protocol,” RFC 3530, The Internet Society, 2003.

[7] Haynes, Thomas (2013-03-14). "NFS Version 4 Minor Version 2"

[8] “Samba : An Introduction ” Available online at : https://www.samba.org/samba/docs/SambaIntro.html

[9] EMC corporation White Paper “EMC VNXe Series: Introduction to SMB 3.0 Support” January 2013

[10] “Storage for your Cloud” , Available online at : http://www.gluster.org/

[11] “ Cluster DRC ” Available online at : https://github.com/nfs-ganesha/nfs-ganesha/wiki/Cluster-DRC

[12] Sun Microsystems, Inc., “NFS: Network File System Protocol Specification,” RFC 1094, The Internet Society, March, 1989.

[13] Eisler, M., “NFS Version 2 and Version 3 Security Issues and the NFS Protocol’s Use of RPCSEC_GSS and Kerberos V5,” RFC 2623, The Internet Society, June, 1999