ORNL/TM-2015/212

Secure Storage Architectures

Ferrol Aderholdt
Blake Caldwell
Susan Hicks
Scott Koch
Thomas Naughton
James Pogge
Stephen L. Scott
Galen Shipman
Lawrence Sorrillo

January 2015

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2015/212

Computing & Computational Sciences Directorate

DoD-HPC Program

Secure Storage Architectures

Ferrol Aderholdt[2], Blake Caldwell[1], Susan Hicks[1], Scott Koch[1],
Thomas Naughton[1], James Pogge[2],
Stephen L. Scott[1,2], Galen Shipman[2] and Lawrence Sorrillo[1]

[1] Oak Ridge National Laboratory
Oak Ridge, TN 37831

[2] Tennessee Technological University
Cookeville, TN, 38501

Date Published: January 2015

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACL	Access Control List
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
BGP	Border Gateway Protocol
CLI	Command Line Interface
CPU	Central Processing Unit
C/R	Checkpoint/Restart
DNAT	Dynamic Network Address Translation
DOE	Department of Energy
GRE	Generic Routing Encapsulation
HPC	High-Performance Computing
HPCCG	High-Performance Computing Conjugate Gradient
HPL	High Performance Linpack
I/O	Input/Output
IPC	Inter-Process Communication
LACP	Link Aggregation Control Protocol
LUN	Logical Unit
LXC	Linux Containers
MD5	Message Digest Algorithm V5
MLAG	Multichassis Link Aggregation
MPI	Message Passing Interface
NAT	Network Address Translation
NFV	Network Function Virtualization
NIDS	Network Intrusion Detection System
ORNL	Oak Ridge National Laboratory
OS	Operating System
OSPF	Open Shortest Path First
OVS	Open Virtual Switch
PC	Personal Computer
POSIX	Portable Operating System Interface for Unix
PVM	Parallel Virtual Machine
QOS	Quality of Service
RAS	Reliability, Availability, and Serviceability
SDM	Security Device Manager
SDN	Software Defined Networking
SNAT	Source Network Address Translation
SNMP	Simple Network Management Protocol
SSH	Secure Shell
VLAN	Virtual Local Area Network
VMS	Virtual Modular Switch
VNIC	Virtualized Network Interface Control
VPC	Virtual Port Channel

VRF	Virtual Routing and Forwarding
VXLAN	Virtual eXtensible Local Area Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Executive Summary

Secure Storage Architectures

The purpose of this report is to clarify the challenges associated with storage for secure enclaves. The major focus areas for the report are:

- review of relevant parallel filesystem technologies to identify assets and gaps;
- review of filesystem isolation/protection mechanisms, to include native filesystem capabilities and auxiliary/layered techniques;
- definition of storage architectures that can be used for customizable compute enclaves (i.e., clarification of use-cases that must be supported for shared storage scenarios);
- investigate vendor products related to secure storage.

This study provides technical details on the storage and filesystem used for HPC with particular attention on elements that contribute to creating secure storage. We outline the pieces for a a shared storage architecture that balances protection and performance by leveraging the isolation capabilities available in filesystems and virtualization technologies to maintain the integrity of the data.

Key Points There are a few existing and in-progress protection features in Lustre related to secure storage, which are discussed in (Chapter 3.1). These include authentication capabilities like GSSAPI/Kerberos and the in-progress work for GSSAPI/Host-keys. The GPFS filesystem provides native support for encryption, which is not directly available in Lustre. Additionally, GPFS includes authentication/authorization mechanisms for inter-cluster sharing of filesystems (Chapter 3.2). The limitations of key importance for secure storage/filesystems are: (i) restricting sub-tree mounts for parallel filesystem (which is not directly supported in Lustre or GPFS), and (ii) segregation of hosts on the storage network* and practical complications with dynamic additions to the storage network, e.g., LNET. A challenge for VM based use cases will be to provide efficient IO forwarding of the parallel filessytem from the host to the guest (VM). There are promising options like para-virtualized filesystems to help with this issue, which are a particular instances of the more general challenge of efficient host/guest IO that is the focus of interfaces like `virtio`. A collection of bridging technologies have been identified in Chapter 4, which can be helpful to overcome the limitations and challenges of supporting efficient storage for secure enclaves. The synthesis of native filesystem security mechanisms and bridging technologies led to an isolation-centric storage architecture that is proposed in Chapter 5, which leverages isolation mechanisms from different layers to facilitate secure storage for an enclave.

Recommendations The following highlights recommendations from the investigations done thus far.

- The Lustre filesystem offers excellent performance but does not support some security related features, e.g., encryption, that are included in GPFS. If encryption is of paramount importance, then GPFS may be a more suitable choice.
- There are several possible Lustre related enhancements that may provide functionality of use for secure-enclaves. However, since these features are not currently integrated, the use of Lustre as a secure storage system may require more direct involvement (support).

*The network that connects the storage subsystem and users, e.g., Lustre's LNET.

- The use of OpenStack with GPFS will be more streamlined than with Lustre, as there are available drivers for GPFS.
- The Manilla project offers "Filesystem as a Service" for OpenStack and is worth further investigation. Manilla has some support for GPFS.
- The proposed Lustre enhancement of Dynamic-LNET should be further investigated to provide more dynamic changes to the storage network which could be used to isolate hosts and their tenants.
- The Linux namespaces offer a good solution for creating efficient restrictions to shared HPC filesystems. However, we still need to conduct a thorough round of storage/filesystem benchmarks.
- Vendor products should be more closely reviewed, possibly to include evaluation of performance/protection of select products. (Note, we are investigation the option of evaluating equipment from Seagate/Xyratex.)

Outline The remainder of this report is structured as follows:

- Section 1: Describes the growing importance of secure storage architectures and highlights some challenges for HPC.
- Section 2: Provides background information on HPC storage architectures, relevant supporting technologies for secure storage and details on OpenStack components related to storage. Note, that background material on HPC storage architectures in this chapter can be skipped if the reader is already familiar with Lustre and GPFS.
- Section 3: A review of protection mechanisms in two HPC filesystems; details about available isolation, authentication/authorization and performance capabilities are discussed.
- Section 4: Describe technologies that can be used to bridge gaps in HPC storage and filesystems to facilitate more secure storage.
- Section 5: We describe an isolation-centric approach for secure storage in an HPC environment, and provide example instances to clarify potential usage for secure-enclaves.
- Section 6: A short analysis of storage vendors with products that could potentially be of use for creating secure enclaves.
- Section 7: A brief synopsis and observations are provided along with concluding remarks for the report. We also highlight the next steps for subsequent evaluations in the secure-enclaves project.

Chapter 1

Introduction

Storage systems are core components of computing. Today computing systems range in size from cellphones to supercomputers and are all data producers. As more of the world's population enjoy Internet access and access to mobile networked devices there has been a corresponding explosion in data production and consumption. This trend has been augmented by the asymmetrical advancement in computer processors and basic storage technologies, where breakthroughs in processors are of a far greater magnitude than those in storage systems. Not surprisingly not only is there a growing demand for bigger, faster storage systems but there is also a demand for them to be secure.

A secure storage product would provide authentication, authorization, data integrity and confidentially and the ability to isolate portions of the storage to appropriate users processes only. A secure HPC storage product would provide all the aforementioned protections in addition to good parallel I/O performance and scaling abilities.

This report provides background on two major HPC storage architectures, Lustre and GPFS. It especially focuses on their security profiles; highlighting strengths, discussing weaknesses and possible workarounds. Particular attention will be paid to areas where security gaps can be bridged by using other technologies. The resulting architectures will be described in detail and evaluated for scaling, performance and security. Recommendations will follow these evaluations.

1.1 Report Outline

The report is structured as follows, Chapter 2 provides background information and terminology, which may be skipped if the reader is already familiar with the technology. Chapter 3 provides details on security related features of selected HPC filesystems, namely Lustre and GPFS, and contrasts some of their differences with respect to secure storage. Chapter 4 discusses technologies that can help to bridge gaps identified in Chapter 3. A brief analysis of currently available secure storage vendor products is given in Chapter 6. We conclude the report in Chapter 7.

Chapter 2

Background

Storage systems are integral components of High-Performance Computing (HPC) environments. Historically, HPC storage systems were mostly installed in government labs, large universities and a few commercial research and development labs. The emphasis was on achieving performance and scalability suitable for a HPC context and less on implementing strong security controls. Often very basic protections were judged sufficient. The shift toward more ubiquitous Internet access and the corresponding increase in shared storage across compute platforms, especially cloud-based platforms, has increased the demand for security in HPC storage. We focus on two HPC storage solutions, Lustre and GPFS, as we believe they collectively represent the more mature, feature-rich, performant, scalable, cost effective and actively developed architectures.

Note, as done in previous reports, we generalize virtualization technologies into two groups: hypervisor-based and container-based. Throughout the report we use the term **virtual machine (VM)** when referring to hypervisor-based virtualization, e.g., KVM. We use the term **virtual environment (VE)** when referring to container-based virtualization, e.g., Docker/LXC.

2.1 The Lustre Storage Architecture

Lustre is a popular storage architecture in the HPC world. The Lustre filesystem has been utilized today by 7 out of 10 of the TOP10 supercomputing sites and over 60% of the TOP100 [30]. It is open source (GPLv2) and is available for several Linux variants. It is a massively parallel filesystem, capable of tremendous I/O performance. Its ability to easily scale capacity and performance by adding more servers is another reason why Lustre is so popular. Clients and servers on a Lustre network communicate via a special networking API called LNET, Lustre networking.

LNET supports a variety of transport protocols including TCP, Infiniband's o2ib, Cray's Seastar and Gemini, Myrinet's MX, RapidArray's *ra* and Quadrics' Elan [30]. LNET also supports routers which provide the capability of routing traffic between different IP networks whose underlying Layer 2 technology might differ. A common use of routers is to bridge between an Ethernet network and an Infiniband fabric [30], often to extend the storage resources on an Infiniband fabric to clients without Infiniband connectivity. Infiniband-to-Infiniband routers are also used to physically partition a large fabric, while still allowing Lustre filesystem access between the partitions. An example of a potential Lustre configuration for basic cluster installation is shown in Figure 2.1.

In Lustre, the management server (MGS) provides the clients with configuration information, such as the location of the metadata server (MDS), storage servers, and filesystem parameters. It also serves as the

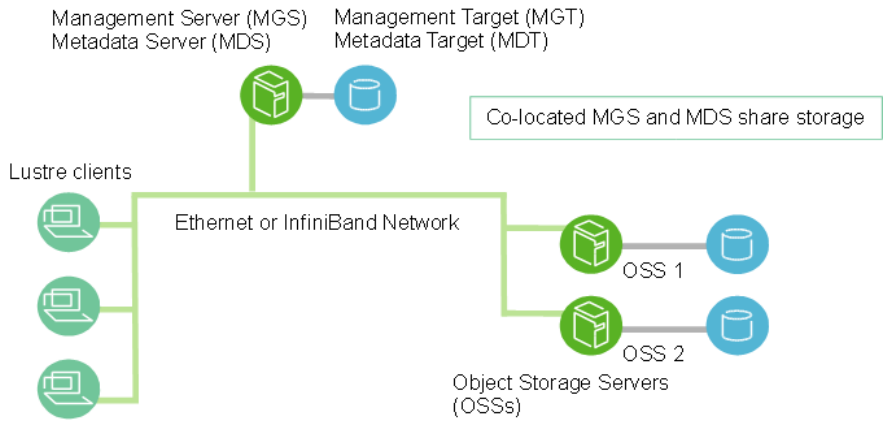


Figure 2.1. Illustration of Lustre configuration for basic cluster installation (figure source: [32]).

first point of contact for a client that wishes to mount the filesystem. To open a particular file, the client contacts a MDS server to receive the metadata for that file. This is stored in an inode similar to traditional filesystems such as *ext4*, but information on the layout of the file is stored in the extended attributes portion. The layout information consists of Object Storage Target (OST) indexes and object numbers for each chunk of the file, which reside throughout the cluster. The client can then contact the Object Storage Server (OSS) on which the specified OSTs reside and gather all chucks from the file. This interaction is illustrated in Figure 2.2, which shows the steps for a Lustre client involved in writing data to the filesystem.

Lustre Components

Metadata Server (MDS) The MDS provides the interface between Lustre clients and the Metadata Target (MDT). The metadata includes file and directory names and their associated inode (location) in the Lustre filesystem [32].

Management Server (MGS) The MGS provides an information service for Lustre, which includes configuration details about all Lustre filesystems in the cluster. The MGS is independent of an individual filesystem, instead it provides general configuration data for end-users and Lustre components themselves [32].

Metadata Target (MDT) The MDT is responsible for maintaining Lustre metadata, e.g., file and directory names, permissions. The MDT is the storage portion of the Lustre metadata, and is accessed via the MDS [32]. The MDT is associated with a filesystem (one filesystem, one MDT) and the shared target (MDT) can be accessed by many MDSs, but for consistency only one should use it. In the event of an MDS failure, a secondary MDS can serve out the MDT to clients [32].

Object Storage Servers (OSS) The OSS provides I/O services for network file requests. The OSS obtains the file data from OST. An OSS is generally paired with 2-8 OSTs [32], which can each be as large as 128 TB [23]. The MDT, OST and Lustre clients can run on a single node but generally are separated to different machines, e.g., MDT on node, OSTs on OSS node, Lustre client on compute nodes [32].

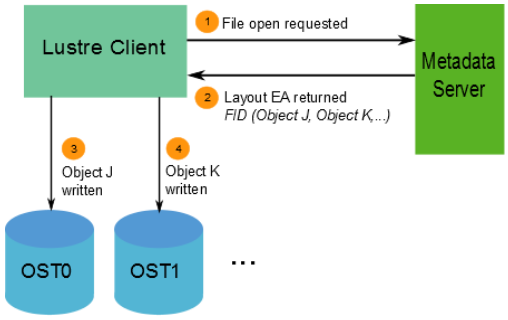


Figure 2.2. Diagram showing steps for Lustre client writing data (figure source: [32]).

Object Storage Target (OST) The OST stores file data as objects, i.e., "chunks of user files" [32]. Multiple OSTs are generally used for a single Lustre filesystem with file "chunks" (objects) spread across the different OSTs. The mapping between file and OST is not necessarily one-to-one, and in many cases to improve performance the files are spread across several OSTs [32]. The management of these "file striping" over OSTs is maintained by a Logical Object Volume (LOV) [32].

Lustre Lock Manager (LDLM) The LDLM coordinates filesystem access, which is run by the MDS [32]. The lock manager used by Lustre is based on the design employed by the VAX distributed lock manager [15].

Portal RPC (PTLRPC) The Portal Remote Procedure Call (RPC), `PTLRPC`, is the underlying mechanism used within LNET for the client/server exchanges. The mechanism is responsible for [41]:

- sending requests through imports[1] and receiving replies,
- receiving and processing requests through exports and sending replies,
- performing bulk data transfer, and
- error recovery.

Lustre Clients The clients are Linux kernel modules that interfaces with the Linux Virtual File System (VFS) and the backend Lustre data servers [32]. The clients mount the Lustre filesystem to provide a seamless view of the underlying parallel infrastructure. The client ensures POSIX compatibility for the user of the filesystem, which include coherent, synchronized filesystem access at all times [32]. There are different clients for the various Lustre components [32]:

- Metadata Client (MDC) to interface with MDT,
- Object Storage Client (OSC) to interface with OST, and
- Management Client (MGC) to interface with MGS.

Lustre Network Driver (LND) A LND provides the interface between the underlying physical network and the abstracted LNET. There are several drivers available for Lustre, e.g., Ethernet (TCP/IP), Infiniband, Quadrics Elan, Myrinet, and Cray (SeaStar Or Gemini) [32].

[1] Lustre import/export are communication pairings used for receiving/sending over LNET.

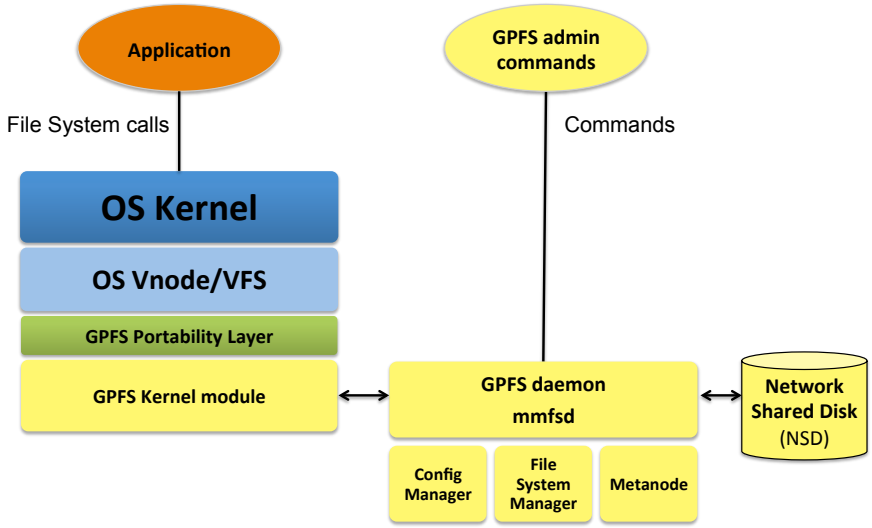


Figure 2.3. System structure for GPFS (adapted from figure in source: [14]).

Lustre Networking (LNET) Lustre uses an internal network abstraction layer that offers an API for managing metadata and file I/O via server and clients [32]. The LNET layer abstracts the underlying network fabric to allow various network transport layers to inter-operate; the Lustre clients and servers use LNET to have a common communication substrate independent of the underlying network technology [32]. LNET uses Lustre Network Drivers (LNDs) to communicate with the underlying networks, e.g., Ethernet, Infiniband, SeaStar.

2.2 The GPFS Storage Architecture

General Parallel File System (GPFS) is a propriety storage architecture from IBM that supplies a feature-rich filesystem. This is a cluster filesystem, providing concurrent access to files from multiple clients. While it possesses many enterprise class storage attributes, GPFS is also a popular choice for HPC. Its features include replication, information life-cycle management (ILM), cloning, snapshots, global namespace, encryption and authentication. Also, GPFS is available for several operating systems: Linux, AIX and Windows.

There are two main methods by which GPFS shares data across clusters: GPFS multicluster and Active File Management (AFM). Multicluster allows GPFS filesystems to share files amongst themselves after the appropriate authentications and authorizations. Multicluster is more suited for sharing filesystems in a Personal Computer (PC) environment as it assumes reliable links and offers higher performance.

AFM is not an HPC appropriate feature as it creates caches of filesets that are asynchronously maintained with the home fileset location. AFM is engineered to prioritize creating global namespaces of filesets rather than getting the most performance. Some of the filesets in the namespace can be read-only.

By default all nodes in a GPFS storage architecture perform the same functions. Thus we describe the GPFS architecture by enumerating the different functions in GPFS and their relationships to each other. A general overview of the system structure is shown in Figure 2.3.

GPFS Components

GPFS Cluster Manager There is one cluster manager per cluster. This node is chosen internally through an election amongst the quorum nodes. The cluster manager has many important duties including:

- Monitoring disk leases,
- Detecting failures and managing recovery from node failures,
- Determining whether a quorum exists so that the GPFS daemon can start,
- Distribute configuration changes to other nodes in the cluster, and
- Manage UID mapping from remote clusters.

Filesystem Manager There is one filesystem manager per filesystem. This node is chosen by the cluster manager and manages all the nodes mounting the filesystem. The primary functions of the Filesystem Manager are:

- Managing adding disks to the filesystem,
- Changing disk availability,
- Managing all mount and umount requests for the filesystem.

Token Manager Server The Token Manager may be the filesystem manager. The Token Manager coordinates access to files on shared disks by granting tokens that convey the right to read or write data or metadata of a file.

Metanode The metanode handles file metadata. Unlike other storage architectures the metanode for a file is determined on a per-file basis and then on which node in the GPFS cluster has recently opened the file for the longest period. Thus over the lifetime of a file the metanode is dynamic. The distributed nature of metadata handling lends itself to good scale-out and metadata performance capabilities.

Network Shared Disk (NSD) Disks containing user data are called Network Shared Disk, NSD. Upon joining a filesystem each disk is tagged with a filesystem descriptor to uniquely identity it belongs to the filesystem. NSD is also the name of a protocol for network access to the disks. The NSD protocol must also be running on the nodes to which the NSD logical units (LUNs) are attached.

Quorum Server The quorum procedure is used for preserving data consistency in GPFS. Quorum means one plus one-half. When a majority, quorum, of nodes are communicating in a cluster then core cluster functions like filesystems mounts can proceed. This scheme prevents filesystem corruption by preventing nodes that have become cut-off from the rest of the cluster from performing filesystem mounts and data access.

2.3 Supporting Security Technologies

2.3.1 NIST

National Institute of Standards and Technology (NIST) is a government body that "develops and issues standards, guidelines, and other publications to assist federal agencies in implementing the Federal Information Security Management Act (FISMA) and in managing cost-effective programs to protect their information and information systems [9]".

2.3.2 FIPS

Federal Information Processing Standards (FIPS) are issued by the National Institute of Standards and Technology. FIPS are "publicly announced standardizations developed by the United States federal government for use in computer systems by all non-military government agencies and by government contractors, when properly invoked and tailored on a contract [44]".

2.3.3 GSSAPI

The General Security Services Application Programming Interface (GSSAPI) is a general application programming interface (API) for security schemes [42]. The GSSAPI allows security vendors to write GSSAPI compliant security modules which is sure to work with clients respecting the standard. A major benefit of the GSSAPI is that it removes the need for vendors to know details specific to the client. For example, the popular Kerberos [26] security mechanism is often used via GSSAPI to ensure a consistent API due to variations in Kerberos implementations.

2.3.4 Kerberos

Kerberos is a security system that provides authentication in a distributed system [26]. The Kerberos authentication system is widely used by many other systems, to include commercial products like Windows, GPFS, Lustre, etc. Kerberos itself does not provide authorization, but can be used to build authorization in other services [26]. Briefly, a client contacts a Kerberos authentication server and once authenticated obtains a *ticket*. This ticket contains cryptographic keys that ensure the identity of the authentication server and client, so that when the client speaks to a service it can transmit the $client_{ticket}$, which can be verified based on the established keys known within the infrastructure. A key element to this process is that the $client_{ticket}$ contains all the information needed for the service to verify the identity of the client and the authorizing server, which enables the security protocol to be more efficient by reducing the amount of communication required to authenticate a client [26]. A more thorough description of the Kerberos authentication service is provided in [26] and RFC-4120 [27].

2.4 Storage Related Components in OpenStack

2.4.1 Swift – Object Storage

OpenStack [31] contains an object storage service known as "Swift" [39]. The purpose of which is to provide massively scalable, redundant storage across commodity hardware platforms. The basic model for this service is along the lines of Amazon S3 storage severs and is managed in similar fashion. The redundant nature of this storage server allows for VM instance templates used by OpenStack compute nodes.

Object Storage Support Structure OpenStack uses many services to complete a storage environment setup. The following is a list of services necessary to run the object storage service in an OpenStack environment:

- Swift: This service handles all of the common files shared between other OpenStack object storage servers and packages, including the Swift client itself;
- Swift-Proxy: The proxy service is the outward facing component that clients connect to.

- Swift-account: The account management service clients use to gain access to OpenStack Storage;
- Swift-Object: Service package that manages object storage and *rsync* file transfer synchronization methods;
- Swift-Container: The package that handles the OpenStack Object Storage Container Class and server;
- Swift-recon: Middleware used on a Swift server node to collect access and usage statistics;
- Memcache: Memory object caching system;
- Network time protocol (NTP): Protocol used in multi-node environments to synchronize the nodes such that the transfers, snapshots and cache scheduling can all be synchronized;
- Xfprogs: This module controls the XFS filesystem used in many OpenStack object storage systems.

OpenStack relies on an NTP server to keep all of its nodes and transfers synchronized. In Swift storage, NTP is used to allow multiusers access to shared data by using a scheduler to make sure one device does not lock out the other requesting client nodes. A maximum resolution is 5 seconds; if the synchronization exceeds this limit the results are unpredictable. Careful load balancing is necessary to provide adequate networking between storage and compute nodes. If the number of compute instances wishing access to data conflicts with the scheduling or desired latencies, Swift allows for replication and redundant repositories that self-synchronize. The effect is the data and storage scales with the demand.

In keeping with the Amazon S3 storage model, Swift is designed to be both highly scalable and highly redundant. It can be installed on any commodity hardware that has sharable storage to be used across the entire installation. The visible structure of the storage service is the folder or directory tree rather than the disk itself. The physical disk is invisible to the virtual plane and may contain one single tree or multiple top directory trees that will appear as individual storage sites depending on group and user permissions.

Object Replication A key element in a highly scalable and redundant filesystem is the ability to handle replication with minimum overhead. In the case of multiple VMs that are set up to share similar resources, such as configuration files and operation parameters, Swift must be able to not only have replicated structure, but synchronization so that all nodes sharing common resources can be maintained simultaneously. The Swift object server relies on the `rsync` service to maintain this replication requirement. The attributes set to achieve this are found in the configuration file `rsyncd.conf`. Each active synchronized instance has values indicating individual performance, such as access rules, the maximum connections allowed. The file attributes, such as read-only and locks, are set in the configuration for the instance. The max connections attribute is used to set a value based on the limitations of the physical host where the storage server is located. If this machine has high throughput networking, large amounts of RAM and multiple high speed disks, the max connections value can be set higher. If the storage node is lacking in physical services, such as available cores, networking throughput and cache memory, then setting this value lower helps keep the system from being overwhelmed by the virtualization of the cloud environment.

Client Connections – Proxy Server The use of a proxy server to handle client connections allows the system administrator to scale out the object storage environment without affecting the connection to the front end. User authentication, access logs and connectivity rules are set in the node's `/etc/swift/proxy-server.conf` file. This allows the system to direct connection requests specifically to a pre-determined source for authentication verification, without the necessity of having to change the basic authentication structure locally. This configuration file also contains the IP address and port to use for both physical or virtual LAN support in this authentication process.

Account Server The account server provides a list of containers available on the node. The accounts are actually in this case the `rsync` account numbers for the connected containers within the storage system. The account server also is used to assign directory folders from the physical disk to a specific proxy server account. Access to specific data could be limited to a single server and account number or shared on multiple servers or accounts.

Container Server The configuration of the container server is very similar to that of the account server, the purpose is to establish server nodes that support the previously generated storage accounts.

Object Server The object server associates the data object with a particular account and container server such that the data is available to an authorized user from a virtualized source on the cloud cluster. This is accomplished by tying the three servers together Account, Container and Object servers in what is called a service ring. The purpose of this structure is for data to be available even if it is non-contiguous and is located across multiple physical disks and multiple hardware platforms. The system will see one or more server storage devices that may be made up of scattered physical drives. The ability to replicate the data on companion storage allows systems to be brought out of service without affecting the perceived location and availability of the data source.

Isolation The storage can be isolated using zones and rings structure. A zone is a group of storage nodes isolated from other nodes in the system. This terminology can be confusing as it is also used in reference to the use of separate physical appliances, network connections, power source, or geographical location. A zone will refer to storage nodes that share common networking and access rules. The Swift Ring structure allows multiple Swift servers and services to locate objects. Swift uses zones to store backup copies or replicas of data on separate systems. Zones are also used to add client access capacity, for example if the storage objects are high demand, and access is requested for multiple users causing access slow down, a zone can be stood up that copies the data and brings up an additional server to share the requested data. In this example we see how the account and synchronization servers come into play. Updates to the data object will be synchronized across all zones as the data source is authenticated. New data appears in each replicated zone within the collective ring such that all sources are up to date.

In a similar fashion, storage nodes can be removed or taken offline from the cluster. This is done by setting a node's priority weight to zero and issuing a ring rebalance command. The node will be removed from service and replications will not include copies and verifications to that node. Once this rebalancing is complete a remove node command can be issued.

System Health and Physical Audits Swift also contains the ability to perform health checks and physical audits on the physical drives attached to a server node. Automated audits help detect physical drives with defects or damaged drives that allow the redundancy systems to replicate the data and prepare for the replacement of the physical disk. Once a physical drive is replaced, the configuration files that describe the Account, Container and Object servers can replicate the missing data and re-attach the drive to the storage ring. This is done by re-enabling the drive back into the ring descriptor, the replicator will reload and check the data on the device and notify the system that the storage object is back in service. This is accomplished by issuing a ring rebalancing command, once the physical drive is replaced. Statistical analysis is accomplished by using the middleware application `swift-recon`. This command provides usage statistics including bytes transferred, read, written and errors. The command line allows for both full reporting on the storage disks and zone based reports. The average loading on the system can also

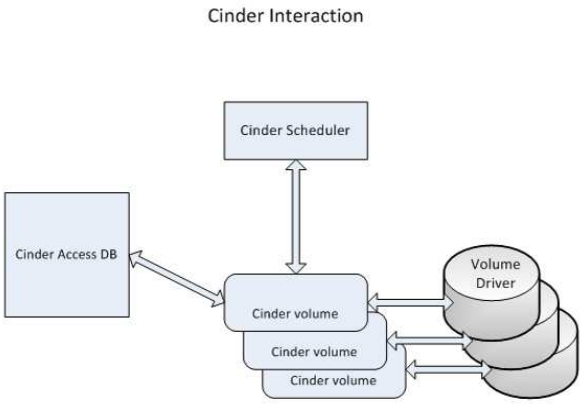


Figure 2.4. Diagram of Cinder component interactions.

be reported as can the time averaged usage statistics. `swift-recon` also reports on quarantined data objects and replication metrics such as success, failure and any discrepancies between replicated objects.

2.4.2 Cinder – Block Storage

OpenStack allows the user to maintain persistent storage methods. When data is written to virtualized node instances the data is not persistent, meaning when the instance is lost the written data can be lost as well. Block storage volumes are a form of persistent storage that can be attached to existing running compute nodes. The block storage methods used in OpenStack are similar to those in Amazon EC2 elastic storage. The running compute node uses an iSCSI based LVM grouping listed as `cinder-volumes`. This is in keeping with the OpenStack Block Storage service called *Cinder* [3]. The service that supports this is called Cinder and the LVM (Logical Volume Manager) volume group is referred to as `cinder-volumes`. Before any such connections are made the `open-iscsi` service must be mounted. Figure 2.4 shows a high-level view of the interactions between the Cinder block storage components. A logical diagram of a Cinder Storage Node is shown in Figure 2.5.

Preparing the Physical Disk Before launching the `cinder-volumes` service the physical disk or part of it must be properly prepared. This is accomplished by creating a partition configured as an LVM volume.

Authentication Cinder uses the existing authentication methods established during the installation of OpenStack and requires keys and ACL checking just as any of the other services in OpenStack, by defining an endpoint, authentication service and user credentials. This includes container class user ID and namespace identification as well as network ID. In a secure environment a Cinder volume can be assigned access only through a VLAN port on a localized Neutron node within the container.

Replicating Instance Once a Cinder volume service is established and its group credentials created, the instances can be copied like any other instance template and only the group ID and VLAN access credentials changed. This is used to allow multiple users access to a data block without specific use information regarding what data and who is accessing it available to the other users.

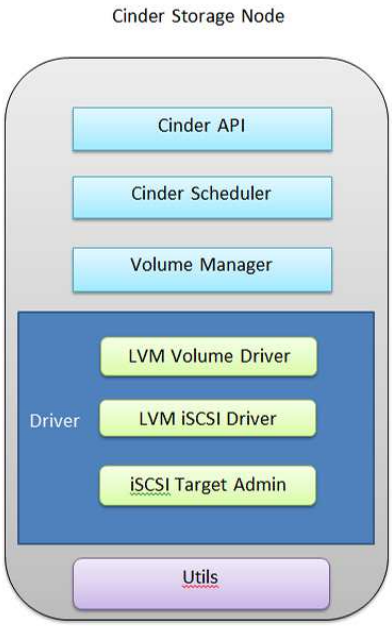


Figure 2.5. Diagram showing components in a Cinder Storage Node.

Compute Node Attachment Once a Cinder volume is established it can be attached to a compute node with the `nova volume-attach` command from the `nova` client. This will occur only after the authentication service has verified the permissions and network node has authenticated the connection credentials. The perspective from the compute node has been compared to the plugging in of a USB storage device, once it appears on the compute node request layer (after authenticating) it is automatically mounted and the connection takes on characteristics native to that of the attached node. Example, it may be the third attached LVM Cinder volume and therefore appear on this device with a different drive name. During authentication the configuration file allows the administrator to make the drive volume name unique to that user. This helps to obfuscate the use or importance of the drive from other nodes and systems sharing a similar attachment to the same data volume. Once a Cinder volume is attached, only that compute node has access to the block. This follows the USB analogy completely, if another node requests the volume from the system, it must first be relinquished by the current user, then after release it is available for attachment to the new compute node.

2.4.3 Manila – File Share Service

A file sharing service called Manila [24] is currently under development. It is based on the Cinder architecture and provides an OpenStack interface for distributed/parallel filesystems, e.g., NFS, GPFS. The Manila project was started in 2013 and was added to the Juno release with an “incubation” status. There are reference implementations for developers as well as initial support for a few vendors/filesystems, e.g., IBM’s GPFS, NetApp, etc.

The currently supported protocols for file sharing are NFS and CIFS (i.e., Samba). The access restrictions are host based (IP addresses). The project is exploring additional access controls beyond the

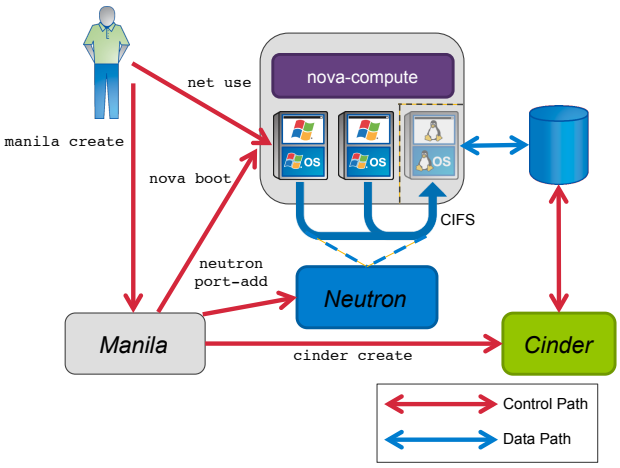


Figure 2.6. Example of the 'generic' Manila file share component for OpenStack (figure source [25]).

basic host-based (IP) approach, e.g., LDAP user/group [25]. The Manila file share service interfaces with the Neutron networking layer, which creates a new neutron subnet that exposes (exports) the share. Currently, this share is limited to a single network and is available to the set of hosts on that network [25]. A snapshot of the share can be created, which as of early 2014 was limited to read-only [25].

An overview presentation from the 2014 OpenStack *Juno* Summit in Atlanta is available off of the Manila project's wiki [25]. This includes a demonstration of the code and details from several vendors supporting or developing Manila modules for the OpenStack component. An example of the reference 'generic' Manila component, useful or development/testing, is shown in Figure 2.6. This 'generic' component dynamically instantiates a VM that is used as the file server, which is attached to the respective compute node subnets. This file server exports the share via the NFS or CIFS protocol. At the time of the presentation, the filesystem `mount` from within the tenant VM of the network share (e.g., NFS export) is done manually, e.g., via a remote SSH to the compute node. They mention that they are currently investigating ways to try and automate this mounting process [25].

Chapter 3

Security in HPC Storage

3.1 Lustre

To employ Lustre in a secure enclave model we outline its security features. Lustre release 2.7 adds an important capability, dynamic LNET configuration, which we leverage to automate the isolation of environments sharing access to the filesystem. Another upcoming feature which enhances Lustre's authentication capabilities, is GSSAPI support. Since Lustre is POSIX compliant, file locking mechanisms promote serialized file access and POSIX ACLs provide user-managed authorizations. Administrators of Lustre can employ root-squash[1] to restrict root clients on the mounted filesystem. There are other features under development for the Lustre 2.8 release scheduled for Q2 2015 [18].

3.1.1 Isolation

A Lustre server can communicate with clients and servers via the LNET API over one or more network interfaces. The common use case is a single LNET on a single physical medium, on which the server can reach all other servers and all clients. However, the identifier Lustre uses for each network node could specify a different network interface (NI) (e.g. `@tcp2` instead of `@tcp1`, where each are in different subnets). If the server has LNET configured on both `@tcp1` and `@tcp2` NIs, then it could reach a client `172.20.0.1@tcp1` as well as `192.168.0.100@tcp2`. These NIs can be separate physical interfaces or different VLANs, but in each case traffic between the two NIs is isolated. Only servers that are dual-homed on both NIs have visibility into both network segments.

This has been a feature of Lustre to have so-called "multi-rail" LNET configurations, but until Lustre 2.7 with the *Dynamic LNET Configuration* feature, it has not been possible to make changes to add or remove an NI from a server without taking the entire filesystem down. Now a command-line tool `lcmd` and a YAML specification can be used to add and remove these interfaces on-demand.

3.1.2 Authentication

GSSAPI New authentication mechanisms can be added to Lustre through its support of the GSSAPI [42]. Currently support for Kerberos is included this way and soon a shared host-key mechanism is due for release.

[1] The "root-squash" option restricts a local `root` user's permissions from being applied to the remote system providing the share, i.e., squashing local `root` permissions from transferring to the remote system.

Kerberos According to OpenSFS, the organization currently responsible for ensuring that Lustre "remains vendor-neutral, open, and free [29]". Lustre's support for Kerberos is in some disrepair [43]. An older work describes Kerberos intended uses in Lustre. Kerberos allows mutual authentication amongst clients, OSSes and MDSes. It also provides both privacy and integrity for `PTLRPC` messages. All entities, users and services alike, are represented as principals to the Kerberos server. Each principal shares a secret key with the Kerberos server that allows them to verify messages from the server. Kerberos implementation is simplest if the Key Distribution Center (KDC) and all the other services share the same user database [32].

Shared Key Authentication Shared Key Authentication and Encryption in Lustre is currently in development and is expected to be completed in Lustre version 2.8. This mechanism will provide host-based authentication and encryption and will use Lustre's existing support for GSSAPI. In this scheme a single key is generated for each client and is installed on client and server [5]. This key is created using a notion of cluster ID, "a string used to uniquely identify a cluster." Data integrity will be provided by creating a message digest, HMAC, of each message or block of data. The keys used to create this HMAC will be obtained from the Linux keyring. The scheme proposes to use userspace tools to create the keys and LNET control utilities, `lctl`, to make the keys available to Lustre. For encryption, a Diffie-Hellman key exchange will be performed to generate a per-session encryption key. Lustre's `PTLRPC` will be tasked with performing the Diffie-Hellman key exchange.

3.1.3 Authorization

POSIX & ACLs Lustre employs a POSIX compliant UNIX filesystem interface [11, 30]. The full suite of POSIX tests completes on a Lustre filesystem just as they do on a regular `ext4` filesystem [30]. This means that regular UNIX users encounter familiar filesystem interfaces and behaviors and can almost immediately begin using a Lustre filesystem.

Root-Squash Since version 2.6 Lustre has supported "root-squash." This is the ability to specify to which local UID/GID should `root` on an accessing client should be mapped [32]. Thus filesystem administrators can apply arbitrary restrictions on clients accessing the filesystem as `root`.

3.1.4 Integrity

Lustre can provide data integrity checks by computing checksums on data [30]. A 32-bit checksum for data read or written on the client and server is computed to guard against corruption in transit. Alder32 and CRC32 are amongst the common algorithms utilized. It should be noted that the backend filesystem does not do any persistent checksumming and so cannot determine whether data residing on disk is corrupted.

3.1.5 Features in Development

Two features currently under development that could add to the overall security of a shared Lustre filesystem are UID/GID mapping scheme and support for clients to mount sub-trees of the filesystem.

Developers from Indiana University have put forth a plan for implementing a nodemap scheme for UID and GID mapping within Lustre funded by an OpenSFS grant [38]. It's functionality was demonstrated with Lustre 1.6 and 1.8 releases, but work is under way to bring the code up to date for a Lustre 2.8 re-implementation. There have been patches submitted for review [21], but the development is not far

enough along that we could perform an evaluation of the feature. Briefly, the nodemap defines a relationship between NID ranges (clients) and UID/GID maps. The nodemap is distributed via LNET to each Lustre OSS and MDS for enforcing what system IDs map to IDs on the filesystem based on the NID of the client. This is not unlike our proposed use of user namespaces to map the UIDs/GIDs within a container to IDs on the host. The combination of these two mapping techniques would provided a layered approach to securely isolating UID/GID ranges. For example, user namespaces can map the `root` user within a container to a normal user on the host, where the nodemap defines what IDs on the host are allowed to map to filesystem IDs. This could potentially limit the filesystem access rights of an adversary if a VE host were to be compromised.

The ability for a client to mount a subdirectory of a Lustre filesystem is proposed in a patch currently under review [22]. This patch proposes to add a capability similar to NFS, where instead of mounting the Lustre root directory, the client could choose to limit the filesystem namespace that is exposed to a subdirectory of the Lustre filesystem. It is true that this doesn't add any security in enforcing isolation at the client level because if the client is capable of mounting a subdirectory, then there's nothing preventing the client form mounting the root directory instead.

3.1.6 Gaps

Having discussed the security features of Lustre, we note significant gaps that appear in the current Lustre release. The first is the absence of server-enforced subtree mounts, instead granting full namespace access to client. There is not a method to limit the subtrees of the filesystem to export to specific clients. This leaves little protection for data on the filesystem if an adversary has escalated to root on any one client. We view this as significant component of shared filesystem security, and we explore alternative ways to mitigate this risk with Lustre in Section 4.

A second gap is the lack of encryption support at rest. Encryption of data in flight is made possible by the GSSAPI support in Lustre, however, it is not likely tested enough to be used in production. Furthermore, objects and metadata are stored in unencrypted `ldiskfs` (a variant of `ext4`) format on the the OST devices. Other non-native techniques would be needed to achieve encryption with Lustre.

As evidenced by the frequent appearance of Lustre on Top500 lists, the focus has been more on the scalability and performance aspects rather than security features seen in other filesystems targeted at the "enterprise-class market."

3.2 GPFS

GPFS is a storage architecture rich in security features. It is POSIX compliant; GPFS implements file locking algorithms that ensures serialization of file updates and uses POSIX ACLs authorizations to manage file access. GPFS also provides both authentication and encryption between clusters owning a filesystem and cluster wishing to mount that filesystem. Apart from encryption in authentication, GPFS also provides encryption for "at rest" files on the filesystem. Below we explore these features in greater detail.

3.2.1 Authentication

GPFS supports mutual cluster authentication and authorization in Multicluster [12]. This allows distinct GPFS clusters to authorize and authenticate each other and then share filesystems. The cluster

owning the filesystem must explicitly grant access to other clusters wishing to mount that filesystem and also explicitly grant access for the specific filesystem to be mounted. On the other hand, clusters wanting to mount a remote filesystem must define the cluster owning the remote filesystem as well as the filesystem it wishes to mount. For this GPFS uses RSA authentication; each cluster generate key pairs then exchanging their public keys. GPFS also supports client clusters at multiple security levels[12]. In this model key pairs of the appropriate strength are exchanged with the different clusters[12]. It is important to note this authentication is between two clusters so nodes within each cluster use the cluster keys for authentication. In addition, this mechanism does not include user authentication[12]; it is assumed the users authenticate to the operating system of the client node by means external to GPFS.

3.2.2 Authorization

POSIX & ACLs GPFS is fully POSIX compliant [1]. Locking, POSIX ACLs and other shell utilities makes the GPFS filesystem experience very similar to a standard Linux filesystem installation.

Kerberos in GPFS Kerberos can be used in combination with SSH as a means of authenticating administrative commands in GPFS [40]. However, it appears to have limited applicability in authenticating clusters to each other. In addition, Kerberos can be used to authenticate users at login into nodes belonging to a GPFS cluster.

GSSAPI in GPFS In GPFS the GSSAPI seem mostly confined to usages within SSH authentication of clients to nodes belonging to a GPFS cluster at login and not as a means of authentication between GPFS services[12].

Root-Squash in GPFS Root squash in GPFS is the ability to map a root user on a client mounting the filesystem to another user with little authorization on the filesystem. This is a restriction imposed by the cluster owning the filesystem and not a device of any UID mapping application nor is it a function of the client cluster mounting the filesystem.

Multicluster Authorization GPFS multicluster requires both filesystem cluster and client cluster various authorization episodes. First, each cluster must authorize participating in multicluster sharing. Then each cluster must authorize connecting to each other; the cluster owning the filesystem must authorize clusters mounting the filesystem and then authorize each connecting client to mount a specific filesystem. Similarly, the client cluster must authorize the cluster with the filesystem and the particular filesystem [12].

3.2.3 Encryption

GPFS supports on-disk encryption in GPFS Advanced Edition and then only in the latest version of the 4.1 filesystem. Encryption is managed through keys and encryption policies and supports several different ciphers. This encryption only applies to data and not metadata. GPFS also advertises secure deletion where data is effectively inaccessible because the encryption keys are deleted from the filesystem.

Encryption from authentication coupled with the on-disk encryption effectively provide end-to-end data encryption in GPFS. Encrypted files at rest on disk are transported through the wire in its encrypted format, to be decrypted in the memory of the client mounting the filesystem. GPFS also supports encryption in a multicluster environment.

In GPFS, Master Encryption Keys (MEKs) are used to encrypt File Encryption Keys (FEKs). FEKs encrypt portions of a file when the file is first created. The FEK is stored, in encrypted format, in an attribute of the file. MEKs are stored on Remote Key Management Servers (RKMs). The RKMs also contain other encryption and policy information.

All GPFS security mechanisms are NIST compliant. To ensure other compliances like FIPS 140-2 or NIST SP800-131A GPFS uses variables, like `FIPS1402mode=yes`. These variables must be set before generating the key-store.

3.2.4 Features & Gaps

GPFS possesses attractive protections. Its authentication and encryption capabilities as well as the "at rest" file encryption capabilities make GPFS a highly secure storage architecture. GPFS also lacks subtree export control capabilities in its NSD protocol thus client cluster mounts cannot be restricted to a subtree of the filesystem. This is a serious shortcoming.

3.3 Discussion

3.3.1 Comparisons of Security with Lustre and GPFS

While both the Lustre and GPFS storage architectures produce highly performant scalable filesytems, there are significant differences in their security capabilities. Both suffer from the inability to only export subtrees of their global filesystem. This is a significant shortcoming as it means that any client that mounts one of these filesystem mounts all the data on the filesystem. This an aspect that we discuss in Section 4, where modern OS technologies can be used to provide an additional layer of isolation between end-users and access to the root filesystem namespace. Both GPFS and Lustre support GSSAPI and Kerberos but in different manners. In GPFS, Kerberos functions to authenticate users for login to systems at the OS level and possibly to authenticate administrative commands. However, in Lustre, Kerberos is used for authentication amongst Lustre specific services. A significant difference between the security stance of the two filesystem is that GPFS natively supports data encryption "at rest," while Lustre does not. Another area where there are significant differences is authentication amongst storage specific technologies. GPFS authenticates pairs of clusters to allow filesystem mounts whereas Lustre authentication is host-based. So client node wanting to mount the Lustre filesystem would have to be authenticated against each Lustre service. Pointedly lacking in both storage architectures is end-user authentication to either Lustre or GPFS specific processes.

3.3.2 Performance in Lustre and GPFS

On at 18PB system with 5000+ servers using GPFS v3.4 has been documented to achieve 240GB/sec [10]. No further details were given on the setup. The Lustre Spider filesystem at ORNL has been documented to produce about the same throughput [33]. This paper predicts that the next generation of Lustre filesystem will reach 1TB/s [34].

Table 3.1. Lustre vs. GPFS

Feature	Lustre	GPFS
Authentication	yes	yes
Encryption in Authentication	yes	yes
On-disk Encryption	no	yes
Subtree mounts	no	no
POSIX Compliant	yes	yes
User authentication to storage	no	no
Kerberos Support	yes	other*
GSSAPI Compliant	yes	other*
Performance	good	good
Scalability	good	good

* denotes that feature is not applied directly to GPFS processes.

Chapter 4

Bridging Technologies for Secure Storage

There are several technologies that may be useful for bridging current gaps in HPC secure storage. The methods we highlight are generally focused on introducing isolation or protection mechanisms that can be leveraged to overcome voids in current HPC storage technologies.

4.1 Virtualization

The virtualization capabilities in enclaves can be leveraged to overcome certain issues by carefully applying the available isolation mechanisms. We briefly highlight some of the more interesting capabilities that can be of use for bridging current gaps in secure storage. Note, more details on isolation with virtualization can be found in a previous project report, "*Review of Enabling Technologies to Facilitate Secure Compute Customization*".

Virtual Machines In hypervisor-based systems, the VM can have virtual devices that do not necessarily match the exact hardware. This can be useful for interposing on a VM's device layer to provide capabilities that are transparent to the guest running inside the VM. For example, if encryption was a priority the data passing from the VM to virtual devices could be encrypted on the fly.

Another example where virtualized devices can be advantageous is when there are changes made to make the interface more efficient by adapting the VM's interface for performance reasons. This is commonly referred to as para-virtualization and is commonly used for customizing the VM's interface to better suite a given use case. For example, virtualized IO can avoid translation in some layers of the software stack because similar work is happening at the system level. Note, this is in contrast to techniques like VMM-bypass, which have the hypervisor and VM setup an interface that allows the VM to have (controlled) direct access to resources without passing through the hypervisor. The `virtio` [36] interface is a standardized API within Linux for creating efficient IO devices for VMs.

Containers & Namespaces The container-based approach to virtualization uses a single OS kernel and adds additional isolation mechanisms for running processes. In this VE-based environment, there are several mechanisms available for limiting access and visibility of the system that can be advantageous for securing storage. For example, the `mnt` namespace provides a kernel level restriction for limiting access to portions of the filesystem. This can be combined with `user` namespaces to allow for controlled mapping of UID/GID privileges within and outside the VE and host contexts. For example, a user may have `root` permissions in a VE but not outside the VE. Additionally, since there is a single kernel in the VE context,

very efficient isolation mechanisms like bind-mounting can be employed to restrict access to filesystems, e.g., restrict a user to a specific region of the filesystem.

4.2 VLAN/Network Segmentation

The ability to restrict access to different portions of the communication network is another mechanism for limiting access and protecting storage. The creation of dynamic network segments, i.e., overlay networks and VLANS, can be used to segment the network to specific hosts and users. These network restrictions may be within the network connecting the hosts, which run the VEs and VMs, such that only the hosts have the ability to make configuration changes to these segments. This would restrict users (VEs/VMs) from seeing each other. Additionally, the restrictions can be within the storage network itself. For example, limiting which hosts may access the Lustre network, i.e., LNET, can limit potential risks to the backing storage network (Chapter 3.1.1).

Note, more details on techniques for network isolation can be found in a previous project report, "*Multi-Tenant Isolation via Reconfigurable Networks*".

4.3 I/O Forwarding

The forwarding of I/O requests through some intermediate layer or service is a very general method for controlling access. A common method for performing this I/O forwarding is to use a network protocol for marshalling the interactions between a client and server. This is often useful for restricting access to filesystem subtree to limit the namespace accessible by a given user, i.e., restrict mountable filesystem shares. When employing virtualization, the forwarding layer may simply be between a host/guest. When working under a single kernel, the filesystem subtree can be restricted via a combination mechanisms, e.g., bind-mounts, cmdpivot_root, and `user` namespaces, as used by containers. However, when working with multi-kernel configurations, e.g., VMs, the sharing must be modified to suit the two kernel environment. The remaining paragraphs in this chapter describe I/O forwarding mechanisms that could be used to retain control of the filesystem tree exposed to the guest.

4.3.1 NFS

Lustre does not have the capability to restrict client mount to only subtrees of the filesystem. This is a very common feature of the NFS protocol and is immensely useful. It also provides a measure of security as it restricts client visibility of the filesystem to anything outside the mounted area. Even if a client on the storage network was compromised, such that the client machine could issue a mount of the entire filesystem, with NFS export restrictions, the NFS server would not allow mounts to succeed outside of the configured subtree for that client's IP address.

4.3.2 VirtFS

VirtFS is a para-virtualized filesystem designed to optimize passing filesystems up from the host operating system through to the guest environment. Popular methods of passing filesystems into a guest are NFS and CIFS[1]. Both these options suffer from performance deficiencies and both are unable to capitalize on the virtual nature of the environment. The ingredients for VirtFS are QEMU [2], KVM [17], VirtIO [36]

[1] Also known as *Samba*.

technologies and the 9P2000.L protocol [6, 13]. VirtFS moves away from the traditional notion of creating virtual block devices in the guest environment for the filesystem passed to it and instead passes I/O to memory objects it shares with the host. These I/Os are then relayed to the local filesystem of the host. This minimizes the number places where the same information is cached between client and server and reduces the number of layers through which data must flow between them. A QEMU/KVM server would export part of the hypervisor's filesystem hierarchy into the guest environment where it is mounted using the 9P2000.L protocol. At this point the guest uses the filesystem as if it were local. In reality however, the guest I/O is actually happening on the hypervisor filesystem [13].

4.3.3 DIOD

DIOD is an I/O forwarding server that uses the 9P protocol to share a filesystem [6]. This work is being carried out by Jim Garlick at LLNL and is currently only being tested with NFS filesystems [6]. There is potential for using this with parallel filesystems like Lustre and GPFS but the documentation indicates this has not yet been fully tested [7]. Even with the experimental Lustre support, performance is limited without a patch to the v9fs driver in Linux to increase the packet payload size beyond 64k [8]. Attempts to push this patch upstream were unsuccessful. However, there has been related work that used 9P for I/O forwarding of Lustre, which appears to be using the NFS-Ganesha server with support for 9P [35].

Chapter 5

Secure Enclave Storage Architecture

An important facet of secure-enclaves is the ability to create the perception of single-user environments out of shared resources. This is achieved through a layering of different isolation mechanisms. In this chapter we will present our view of an isolation-centric storage architecture. This architecture employs different underlying storage technologies (e.g., Chapter 3) in concert with bridging technologies (e.g., Chapter 4) to provide the requisite controls for persistent storage.

5.1 Isolation-Centric Storage Architecture

Given the security strengths and frailties of Lustre and GPFS from previous chapters, we advocate a entirely different approach that might not be immediately intuitive. We believe isolation is key to providing security as well as preserving the performance expected in a HPC environment. To this end, we redefine the storage layer up from the filesystem (Lustre or GPFS) to extend into an enclave where the user's view of storage is strongly restricted to authorized areas. These restrictions are achieved in a layered fashion using different isolation mechanisms like OS containers, virtual machines and network segmentation capabilities. The network isolation mechanisms can extend into the storage architecture by implementing a dedicated VLAN per user into the storage filesystem and another for a user's compute resources (nodes). For example, a user requiring 128 nodes for a job will have one VLAN dedicated to the 128 compute nodes and another dedicated to the storage traffic. This holistic approach to a secure-enclaves design is a consequence of the challenge to balance performance and protection. In many cases, as highlighted in earlier chapters, there are limitations in HPC filesystems that inhibit securing the storage layers through native mechanisms due to practical implementation gaps. Restated, the requisite protection and isolation mechanisms are not directly available by the HPC filesystems and must be complemented with additional layers. An illustration of this isolation-centric architecture is shown in Figure 5.1.

The bridging technologies we have mentioned before, e.g., VirtFS, DIOD, NFS, can be used to restrict the end-user to approved areas of the filesystem. For example, in Figure 5.1 end-user jobs run in the VEs and the VMs. In either case, the end-user's view of the storage has been restricted by host level mounts from Node1, Node2 and Node3 into their virtual environment. It is important to note that all the hosts (Node1..3) mount the full filesystem and pass the "user appropriate portion" into the virtual environments. Also note that there is no commingling of end-user processes due to the compute layer virtualization based isolation mechanisms. In those instances where we restrict each user's traffic to their own VLAN, both into the storage and the VEs or VMs belonging to that user, we have essentially installed a single-user environment (i.e., enclave) – though contention for the filesystem remains.

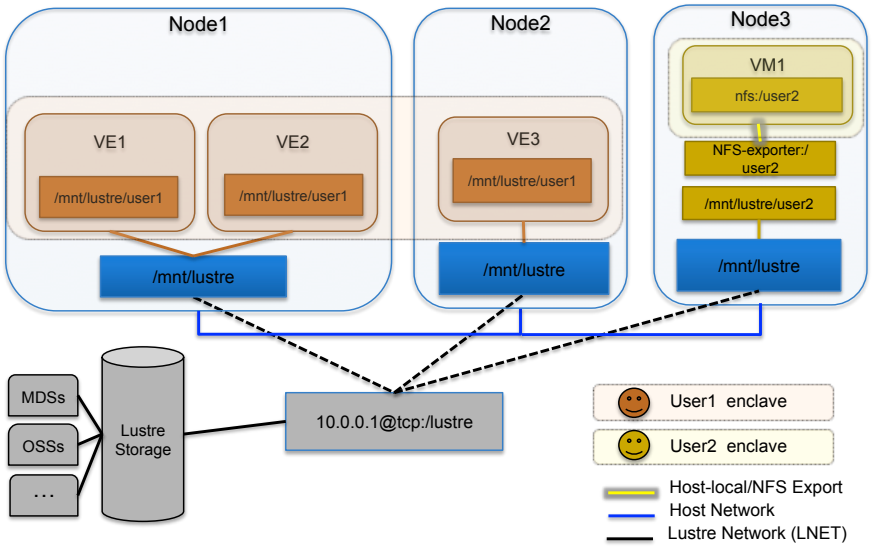


Figure 5.1. Diagram showing different layers of isolation-based storage architecture using Lustre, three VEs and one VM.

As identified in previous chapters the ability to control the namespace accessible by a tenant is a common gap in many HPC parallel filesystems. This issue of restricting subtree access for a global filesystem can be achieved in an isolation-centric model by employing kernel based namespace restrictions or via I/O forwarding methods in multi-kernel scenarios. These map to the VE and VM use cases within the secure-enclaves architecture, where VEs operate within a single kernel and VMs have distinct kernels. In both cases, the controls are implemented at the host-level, i.e., outside of the VE/VM context, and restrict access to the underlying storage services. The VE based approach may be implemented using "bind mounts" and Linux namespaces to restrict the tenant to a subset of the shared filesystem. The VM case may use NFS or 9pfs to "export" the shared filesystem at a specific depth to restrict access. As noted in earlier chapters, para-virtualized filesystem interfaces may provide more efficient "re-exporting" of host-level filesystems to the guest (VM) context by passing virtual IO devices, e.g., `virtio`.

There are also instances where it can be beneficial to limit portions of the *storage network*, which is the portion of the network dedicated to the storage LAN, e.g., Lustre's LNET. A normal configuration is to have a single storage network with all hosts directly connected. Note, these host machines are running the tenant compute VEs/VMs, which are connected via compute VLANs. The scenario of a single storage network is illustrated in Figure 5.2 where a all hosts in the cluster are directly connected to the single storage network.

An additional degree of network traffic segregation can be achieved by creating "storage VLANs." In this scenario, the hosts are grouped and each group has a distinct interface for the different storage VLANs, i.e., VLANs for the storage-facing region of the network. These interfaces can be used to segregate traffic for the storage network. For example, in a Lustre environment a separate network interface (NI) could be created on the host for the different types of groups. This multiple storage VLAN example is shown in Figure 5.3. Note, the addition of features like *Dynamic LNET Configuration* (Section 3.1.1) provide a way to create these grouping in a much more agile fashion that can be influenced by the tenant assignments for a given host.

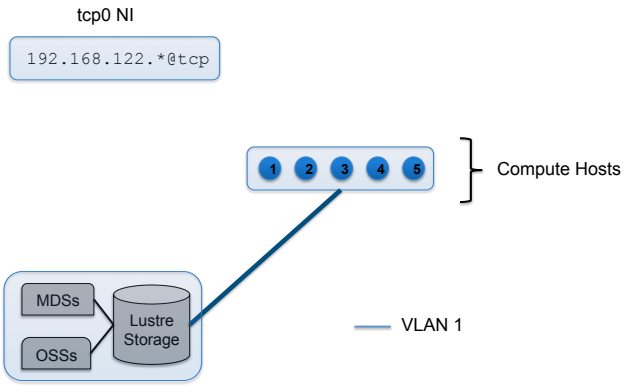


Figure 5.2. Diagram showing all hosts on a single storage network.

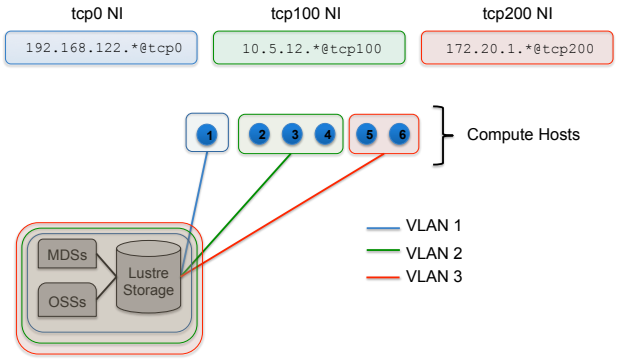


Figure 5.3. Diagram showing hosts on separate VLANs to restrict overall access to storage network.

5.2 Instances of the Isolation-Centric Storage Architecture

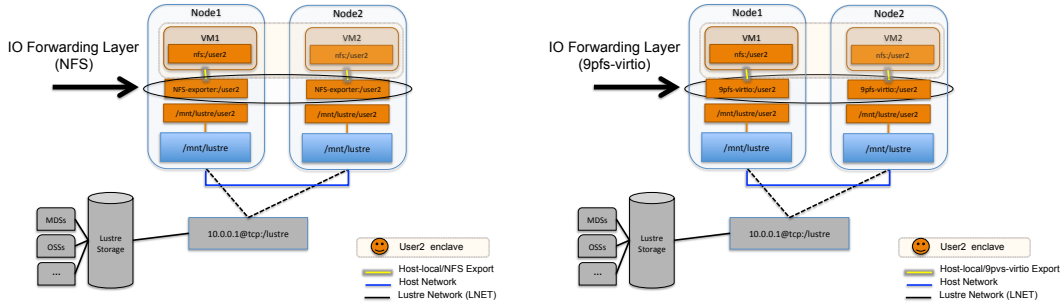
The following examples show concrete instances of configurations that employ the proposed isolation-centric storage architecture. These scenarios illustrate approaches for managing access to shared storage in a secure enclave. The configurations use different storage technologies (Chapter 3) and as necessary bridging technologies (Chapter 4) to implement the controls for isolation-centric secure storage.

5.2.1 Parallel filesystem with host-based subtree limitations for VM

Lustre, NFS re-exporter with KVM This configuration seeks to implement subtree export capability using Lustre and NFS (Figure 5.4(a)). A node which is a Lustre client also assumes two other responsibilities; that of a NFS server and host for a KVM instance. The KVM instance is in turn a NFS client mounting the filesystem served by the NFS server. In this approach, the NFS server only exports that subtree of the Lustre filesystem it wishes to make available to the user. We expect some performance penalty for using NFS.

Lustre, 9pfs re-exporter with KVM This configuration seeks to implement subtree export capability using Lustre and p9fs with KVM (Figure 5.4(b)). A node which is a Lustre client also assumes one other

responsibility; that of a KVM instance. The KVM instance in turn uses the p9fs protocol to mount the part of the host filesystem. In this approach, the KVM instance is allocated the part of the filesystem it is allowed to mount via p9fs at the time of its creation. We expect some performance penalty for using P9fs.



(a) NFS re-exporter (b) 9pfs-virtio re-exporter

Figure 5.4. Example instance of Lustre, IO re-exporter with VM. This shows two different approaches for the IO re-exporter, one that uses NFS and another that uses 9pfs with virtio.

5.2.2 Parallel filesystem with host-based subtree limitations for VE

Lustre, bind-mount with LXC namespaces This model seeks to restrict the LXC instance to a portion of the filesystem tree. A Lustre client hosts a LXC instance into which a subtree of the filesystem is bind mounted (Figure 5.5). This maintains the high performance of Lustre in the LXC instance, while protecting other areas of the global filesystem. Another benefit here is that the Lustre client authentication mechanisms are all preserved.

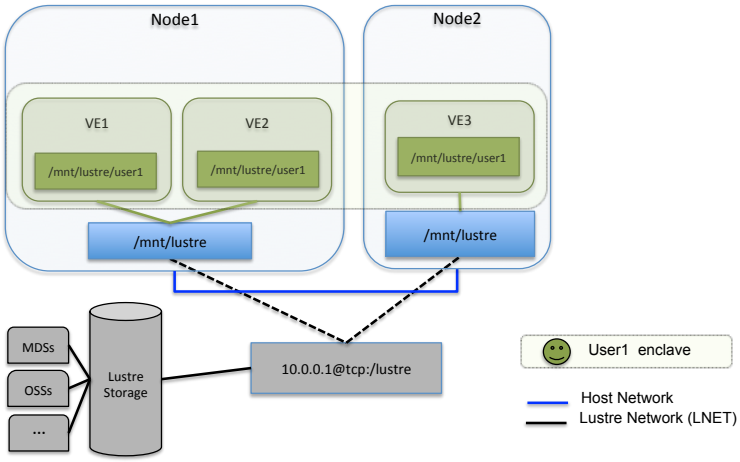


Figure 5.5. Example instance of Lustre, bind-mount with VE.

GPFS, bind-mount with LXC namespaces This model seeks to restrict the LXC instance to a portion of the filesystem tree. A GPFS client hosts a LXC instance into which a subtree of the filesystem is bind mounted (Figure 5.6). This maintains the high performance of the GPFS in the LXC instance, while

protecting other areas of the global filesystem. Another benefit here is that the GPFS client authentication mechanisms and the filesystem encryption protection are all preserved.

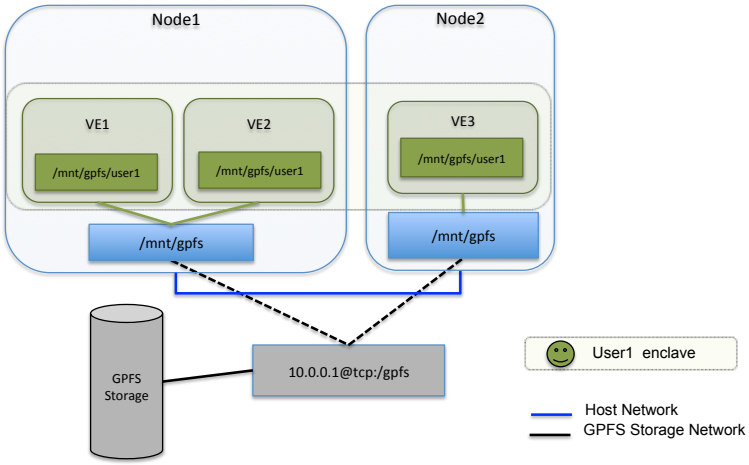


Figure 5.6. Example instance of GPFS, bind-mount with VE.

Chapter 6

Storage Vendor Analysis

6.1 Overview

There are several vendors that support network storage or more specifically NFS (Network File Systems). Most of these are a form of classic large enterprise centralized archive storage such as SAN (Storage area network) servers or just simply additional servers that have large RAID (Redundant Array of Inexpensive Disks) arrays. In an HPC deployment model they would be externally connected peripheral services, not aligned with the compute nodes directly. Some of these hardware based storage configurations can be adapted using the OpenStack services to perform a more abstracted storage service within the cloud, but few exist that have been configured to handle the rigors of the HPC compute environment. The following is a sparse list due largely because few vendors have started servicing the HPC market directly, and fewer still provide the necessary bandwidth and configuration for high demand access adopting rather the slower and more accessible archive transfer model. The following is a synopsis of the vendors that have either directly adopted the design path necessary to support HPC or systems that are part of emergent technology that can easily meet the rigors of HPC.

6.2 Seagate/Xyratex

Seagate corporation is one of the longest standing disk drive manufacturers in the world, and has supported SAN and RAID server technology for years. Recently they acquired *Xyratex* [45] and started to aggressively pursue the storage appliance market. This division of Seagate offers several high speed redundant storage offerings but further examination reveals that they all have a very common platform topology and are equally scalable. This discussion will cover one of their smaller build outs, the *ClusterStor*™ 1500 as an example, the other models in this family are mainly scaled-up size and capabilities of either racks of this model or larger versions of the same. The ClusterStor devices are built upon the successful Seagate OneStor enterprise/data center platform.

6.2.1 Security

The ClusterStor series is ICD 503 (Intelligence Community Directive) (DCID 6/3 PL4) Compliant, including MAC (Mandatory Access Control) and uses explicit auditing of both usage tracking, access logging. The policy “least privilege” is enforced using audit logging and encryption. The system supports multiple separate file zone classifications on a single server or filesystem.

6.2.2 ClusterStor Platform

This appliance is marketed directly for high performance computing and uses the open source Lustre filesystem [19] taking advantage of its parallel file structure. The intrinsic customized OS a modified SELinux (Security Enhanced Linux) loaded on the ClusterStor 1500 organizes the storage blocks base units with network interface and Expansion ports referred to as SSU (Scalable Storage Units), System administrators can alternatively attach additional physical storage devices called ESU (expandable Storage Units), these physical appliances contain access controllers and additional disks but lack the network interfaces for external connection. In most common configuration each data file is striped across the entire assigned filesystem organization on SSU and ESU so that access is handled in parallel. The physical disk drives are abstracted in a sub layer directly handling I/O read and write requests in a synchronized simultaneous manner. Custom SATA (serial AT attachment the AT from the original IBM PC AT model circa 1980's) [37], ASIC devices provide the synchronized access scalable between the nominal 1GB/s up to 100 GB/s. A single ClusterStor SSU unit can have up to three additional ESU units attached, for full physical expansion build out. The SSU contains the management unit that handles all of Lustre's metadata services.

6.2.3 Physical Drive Capability

Each of the drive bays can handle any of the available 2.5" SATA disk drives commonly available on the market, however all of the published specifications assume that the dual port 6Gb/s SAS drives are used for maximum performance. The number of drives installed is limited by the size and model of the SSU and ESU attached, the embedded Lustre filesystem abstracts the system to a single storage space that can be linearly scaled out with additional units and across other Lustre servers.

6.2.4 Data Throughput

A single physical system can provide between 1 GB/second up to 100 GB/s however a parallel cluster of systems can theoretically support throughput rates of a sustained 1 TB/s [4].

6.2.5 Specification Table ClusterStor™ 1500

The following is a table of capabilities and specification for the range of the model 1500, any of the other larger models will be similar with a higher capacity, the basic throughput will be identical to a configuration with multiple smaller devices. The difference in models appears to be mainly to consolidate physical space requirements by sharing a larger enclosure. Table 6.1 was copied from the ClusterStor 1500 product bulletin courtesy Seagate Inc [4].

6.3 Oracle ZFS Storage Appliance

The ZFS is originally designed to support VM and cloud storage requirements, it follows an architecture paradigm of mapping a HDD (hard disk drive) with large arrays of DRAM storage. This format is referred to as Hybrid Data Pool (HDP) [46].

Parameter	Description
Filesystem Performance	1.25GB/s up to 110GB/s sustained read and write
Raw OST Capacity	80TB (4TB SAS HDDs) to 10.PB (6TB SAS HDDs)
Usable File System	60.4TB (4TB SAS HDDs) to 7.9.PB (6TB SAS HDDs)
Lustre Network Protocol	Infiniband QDR / FDR or Ethernet 10Gb/s 40Gb/s
File System Lustre	2.1 (with Seagate add ons)
Maximum Files	280 Million
SSU Storage units	(2) 2.5" HDDs 300GB RAID 1 1+1 (21) 3.5" RAID 6, 8+2
ESU Storage Units	(21) 3.5" RAID 6, 8+2
Hot Swappable	HDDs, Redundant Power/Cooling supplies
Operating System	SELinux
Management Network	Dual 1 Gigabit Ethernet

Table 6.1. Seagate ClusterStor product specifications (table source: [4]).

6.3.1 Resiliency

The HDP topology uses the HDD physical drives for resiliency of the data. Access requests use the high speed scalable DRAM arrays and periodic backups are made to the slower HDD and checked against read and write/store requests. The user access is handled directly from the DRAM and mimics the smaller CACHE memory access used on most high efficiency compute nodes. Simultaneous access to the DRAM array provides the system with parallel failover and self healing filesystem capability.

6.3.2 Vitualization Support

Although designed largely for enterprise applications, the Oracle ZFS system does include plug-ins to support virtualization. Including OpenStack and other cloud implementations.

6.3.3 Filesystem Support

The Oracle ZFS specifically includes support for Oracle data base file structures, although there is no listed support for Lustre or GPFS, there is no indication that these appliances cannot be reconfigured to use either. This will require additional research and inquiries to ascertain. The system also supports the OpenStack Cinder (block storage server) [46].

Parameter	Description
Filesystem Performance	1.25GB/s up to 110GB/s sustained read and write
Usable File System	60.4TB (4TB SAS HDDs) to 7.9.PB (6TB SAS HDDs)
Lustre Network Protocol	Infiniband QDR or Ethernet 1Gb/s 10Gb/s
File System	Oracle Solaris ZFS
Hot Swappable	HDDs, Redundant Power/Cooling/ supplies
Operating System	Semi custom Linux
Management Network	1-10 1 Gigabit Ethernet ports

Table 6.2. Oracle ZFS Storage appliance specifications (table source: [46]).

6.4 Additional Systems

The other storage systems of note are not geared specifically toward HPC, but rather fit into standard Enterprise/Cloud service models. These include SAN (Storage Area Network) servers from HP and Dell, Fujitsu and others. In most cases these are currently used in OpenStack storage server applications, additional research is need to ascertain the performance specifications when used in an HPC environment with Lustre and GPFS. Certainly all of the models collected for this study can be upgraded with Mellanox Infiniband and 10 + GB Ethernet controllers, but the backplanes and server structure does not specify the advantage nor ability to increase beyond native network interface speeds. Most of these are iSCSI (Internet Small Computer System Interface) which is a relatively slow interface compared to the two identified vendor products.

Chapter 7

Conclusion

7.1 Synopsis

We set out to identify secure HPC storage technologies that provide protection and high performance consistent with the requirements for secure enclaves. In Chapter 1, we identified the protections that constitute a secure storage architecture and later identified Lustre and GPFS as two of the most popular, performant and scalable HPC storage architectures in use today. In Chapter 2, we provided some background on Lustre and GPFS to provide a working familiarity with both storage systems. In Chapter 3, we focused on the protections provided by Lustre and GPFS, highlighting protections and identifying vulnerabilities. Chapter 4 introduces "bridging technologies," whose purpose are to augment protections by addressing specific weaknesses. Chapter 5 outlines the application of "bridging technologies" to protect the gaps in Lustre and GPFS to provide more secure HPC storage solutions. Some HPC storage appliances were examined in Chapter 6 to decide if they provide more complete security protections than other products in the study. Lastly, in this chapter we mention recommendations and outline future plans for the project.

7.2 Recommendations & Observations

While presenting these recommendations, we acknowledge that there is a cost in terms of administrative complexity when combining OS isolation layers and filesystem technologies. Where practical, we suggest the use of automation tools such as OpenStack [31] and Puppet [16], both of which have significant momentum in HPC and cloud communities. This is evidenced by the rapid appearance of projects integrating new technologies, such as containers with `user` namespaces as in the `nova-compute-lxd` project [28]. The ability to automate the storage aspects such as with *Dynamic LENT Configuration* adds an additional isolation capability, but in order for it to be feasibly implemented, it must also be able to integrate with the encompassing automation workflow.

A compelling argument could be made for avoiding this complexity and adopting a complete solution stack from a vendor. Vendors such as IBM, Seagate and Oracle strive to deliver a complete secure storage solution. They will certainly have fewer layers and individual components to manage than a Lustre filesystem with virtualization-based isolation layered on. The disadvantage is in terms of flexibility. A Lustre deployment comes with enormous flexibility for customization, and features are continually being released by the community.

Throughout this report we have noted where particular security mechanisms are at odds with the performance of the storage system. This will be an ongoing competition between these goals, but we believe that the isolation techniques discussed here form a unique solution to the dilemma.

7.3 Future Plans

Our future plans include a thorough evaluation of the proposed secure-enclave storage architectures. We want to determine whether these proposals add the protections for which they were designed without adding new vulnerabilities. In particular, we want to evaluate whether isolation provides enough protections to overcome the absence of subtree export controls in both Lustre and GPFS. In addition, we are particularly eager to determine whether we can provide enough protections to overcome the lack of file encryption in Lustre. We will also try to find alternative ways of applying either disk or filesystem encryption in Lustre.

In GPFS we would like to quantify the overhead of using native encryption in conjunction with the isolation-centric storage architecture outlined in this report. We also anticipate further analysis of security vulnerabilities for selected storage related technologies used as part of our evaluations.

In our investigation of the security features of Lustre, we found many features that are in-development, where some are nearing release-readiness such as shared key authentication/encryption [5] and dynamic LNET configuration [20], while others are under heavy development as in UID/GID mapping [38], and subtree support [22]. Along with Kerberos support [43], we will evaluate these features to the extent the current code has been merged into the Lustre *master* branch.

To address the administrative burden of managing a secure storage system with several layers of isolation, we will make use of the OpenStack project and Puppet. We intend to evaluate the suitability of these tools for the scalable deployment and administration of a secure storage system for secure enclaves. In addition they will aid our efforts to rapidly deploy testing environments in a reproducible manner.

7.4 Acknowledgments

This work was supported by the United States Department of Defense (DoD) and used resources of the DoD-HPC Program and the Compute and Data Environment for Science (CADES) at Oak Ridge National Laboratory.

Bibliography

[1] IBM InfoSphere BigInsights Version 2.1.2. *Comparison of HDFS features and GPFS features*. IBM. URL: http://www-01.ibm.com/support/knowledgecenter/SSPT3X_2.1.2/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/over_filesystem_comparison.html.

[2] Fabrice Bellard. QEMU, A Fast and Portable Dynamic Translator. In *USENIX 2005 Annual Technical Conference*, Anaheim, CA, USA, April 2005.

[3] Cinder: Block Storage for OpenStack. URL: https://wiki.openstack.org/wiki/Cinder [cited 29-jan-2015].

[4] Seagate ClusterStor Product Bulletin. URL: http://www.seagate.com/products/enterprise-servers-storage/enterprise-storage-systems/clustered-file-systems/#specs [cited 29-jan-2015].

[5] Andreas Dilger. *Lustre File System: IU Shared Key Authentication and Encryption*. Open SFS, 2013. URL: https://jira.hpdd.intel.com/browse/LU-3289.

[6] diod: An I/O forwarding server based on the 9P protocol. URL: https://code.google.com/p/diod/ [cited 1-feb-2015].

[7] diod source code at github. URL: https://github.com/chaos/diod [cited 1-feb-2015].

[8] diod performance tests with Lustre. URL: https://code.google.com/p/diod/wiki/performance [cited 5-feb-2015].

[9] Computer Security Division. *Compliance With NIST Standards And Guidelines*. Nationa Institute for Standards and Technology, 2014. URL: http://csrc.nist.gov/groups/SMA/fisma/compliance.html.

[10] *General Parallel File System (GPFS) 3.5 System Administration for Linux*, November 2013. Avanet Services IBM Training Student Notebook (Course code H005 ERC 1.0).

[11] Andreas Grünbacher. POSIX Access Control Lists on Linux. In *Proceedings of the USENIX Annual Technical Conference*, pages 259–272. USENIX, June 2003. URL: http://users.suse.com/~agruen/acl/linux-acls.

[12] IBM. *GPFS Advanced Administration Guide*. IBM Knowledge Center, 2014. URL: http://www-01.ibm.com/support/knowledgecenter/#!/SSFKCN/gpfs41/gpfs.v4r1_welcome.html.

[13] Venkateswararao Jujjuri, Eric Van Hensbergen, Anthony Liguori, and Badari Pulavarty. VirtFS—A virtualization aware File System pass-through. In *Ottawa Linux Symposium*, pages 1–14, December 2010.

[14] Frank Kraemer. Software defined storage in action with GPFS v4.1, October 2014. Presentation at Linux Foundation's CloudOpen Europe 2014 Conference, Düsseldorf, Germany. URL: http://events.linuxfoundation.org/sites/events/files/slides/SDS-in-action-with-GPFSv41-kraemerf-102014.pdf [cited 27-jan-2015].

[15] Nancy P. Kronenberg, Henry M. Levy, and William D. Strecker. Vaxcluster: A closely-coupled distributed system. *ACM Trans. Comput. Syst.*, 4(2):130–146, May 1986. URL: http://doi.acm.org/10.1145/214419.214421, doi:10.1145/214419.214421.

[16] Puppet Labs. Puppet Documentation Index. URL: https://docs.puppetlabs.com/puppet/ [cited 02-dec-2014].

[17] Linux Kernel-based Virtual Machine (KVM). URL: http://www.linux-kvm.org [cited 29-nov-2014].

[18] Community Lustre Roadmap. OpenSFS community Lustre Roadmap. URL: http://lustre.opensfs.org/community-lustre-roadmap [cited 31-jan-2015].

[19] Lustre community portal. URL: http://lustre.opensfs.org [cited 28-jan-2015].

[20] Lustre Ticket LU-2456: Dynamic LNet Config Main Development Work. Descr: This ticket has been created to track the main development work for the Dynamic LNet Config project. URL: https://jira.hpdd.intel.com/browse/LU-2456 [cited 06-feb-2015].

[21] Lustre Ticket LU-3291: IU UID/GID Mapping Feature. Descr: Tracking bug for Indiana University's UID/GID mapping and cluster project. URL: https://jira.hpdd.intel.com/browse/LU-3291 [cited 27-jan-2015].

[22] Lustre Ticket LU-5989: add subdirectory mounting support for Lustre. Descr: add subdirectory mounting support for Lustre. URL: https://jira.hpdd.intel.com/browse/LU-5989 [cited 31-jan-2015].

[23] Lustre hands-on at SC2011. Lustre Hands-On at SC2011 Presentation. No Links to proceedings. URL: http://cdn.opensfs.org/wp-content/uploads/2011/11/Lustre-hands-on-SC2011.pdf [cited 01-jan-2015].

[24] Manila: Shared file system service for OpenStack. URL: https://wiki.openstack.org/wiki/Manila [cited 29-jan-2015].

[25] Welcome to Manila: An OpenStack File Share Service, 2014. Presentation from Juno (Atlanta) Summit. URL: https://wiki.openstack.org/wiki/Manila/JunoSummitPresentation [cited 30-jan-2015].

[26] B.Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994. doi:10.1109/35.312841.

[27] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC-4120: The Kerberos Network Authentication Service (V5), July 2005. URL: http://www.ietf.org/rfc/rfc4120.txt [cited 31-jan-2015].

[28] nova-compute-lxd source code at github. URL: https://github.com/zulcss/nova-compute-lxd [cited 5-feb-2015].

[29] OpenSFS. *About OpenSFS*. OpenSFS, 2015. URL: http://opensfs.org/about/.

[30] OpenSFS. *Lustre® File System, Version 2.4 Released*. OpenSFS, 2015. URL: http://opensfs.org/press-releases/lustre-file-system-version-2-4-released.

[31] OpenStack: The Open Source Cloud Operating System. URL: https://www.openstack.org/software [cited 4-feb-2015].

[32] Oracle; Intel. *Lustre File System: Operations Manual Version 2.0*, 2011. URL: https://wiki.hpdd.intel.com/display/PUB/Documentation.

[33] Sarp Oral, David A. Dillow, Douglas Fuller, Jason Hill, Dustin Leverman, Sudharshan S. Vazhkudai, Feiyi Wang, Youngjae Kim, James Rogers, James Simmons, and Ross Miller. *OLCF's 1 TB/s, Next-Generation Lustre File System*. OLCF at ORNL, 2013. URL: https://cug.org/proceedings/cug2013_proceedings/includes/files/pap151.pdf [cited 27-jan-2015].

[34] January 2015. Personal communication with administrative staff deploying and maintaining leadership class Lustre installation at ORNL.

[35] Grégoire Pichon. Experiments with io proxies over lustre, September 23, 2014. Presentation at the 2014 Lustre Administrators and Developers Workshop (LAD'14). URL: http://www.eofs.eu/fileadmin/lad2014/slides/18_Gregoire_Pichon_LAD2014_IOProxies_over_Lustre.pdf [cited 1-feb-2015].

[36] Rusty Russell. virtio: Towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, July 2008. doi:10.1145/1400097.1400108.

[37] Wikipedia: Serial ATA. URL: http://en.wikipedia.org/wiki/Serial_ATA [cited 1-feb-2015].

[38] Stephen Simms and Josh Walgenbach. Scope Statement For UID/GID Mapping in Lustre 2.X, November 10, 2012. Revision v2. URL: http://wiki.opensfs.org/images/3/31/UID_GID_Scope_Statement_v2.pdf [cited 27-jan-2015].

[39] Swift: Object Storage for OpenStack. URL: https://wiki.openstack.org/wiki/Swift [cited 29-jan-2015].

[40] Visolve SSH Team. *OpenSSH Whitepaper*. Visolve. URL: http://www.visolve.com/ssh.php#Kerberos_Authentication.

[41] Feiyi Wang, Sarp Oral, Galen Shipman, Oleg Drokin, Tom Wang, and Isaac Huang. Understanding lustre filesystem internals. Technical Report ORNL/TM-2009/117, National Center for Computational Sciences, April 2009. URL: http://wiki.lustre.org/images/d/da/Understanding_Lustre_Filesystem_Internals.pdf [cited 1-feb-2015].

[42] Wikipeda. *Generic Security Services Application Program Interface*. OpenSFS, 2015. URL: http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface.

[43] Wikipeda. *Lustre GSSAPI/Kerberos Repair*. OpenSFS, 2015. URL: http://wiki.opensfs.org/Lustre_GSSAPI/Kerberos_Repair [cited 31-jan-2015].

[44] Wikipedia. *Federal Information Processing Standards*. Wikipedia. URL: http://en.wikipedia.org/wiki/Federal_Information_Processing_Standards.

[45] Xyratex. URL: http://www.xyratex.com [cited 28-jan-2015].

[46] Oracle ZFS Product Bulliten. URL: https://go.oracle.com/LP=3687?elqCampaignId=6317&src1=ad:pas:go:dg:stor&src2=wwmk14054304mpp001&SC=sckw=WWMK14054304MPP001 [cited 29-jan-2015].