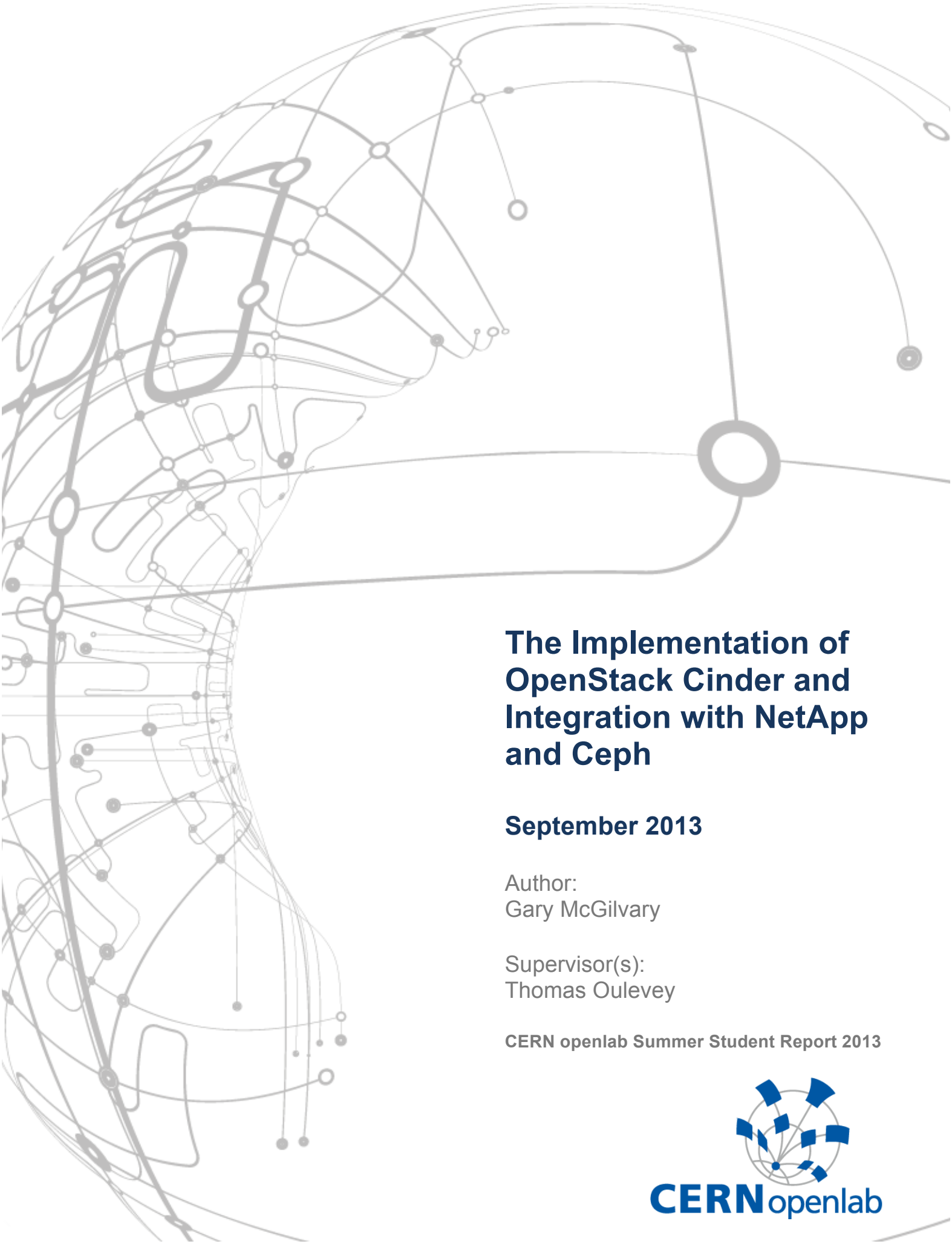

The Implementation of OpenStack Cinder and Integration with NetApp and Ceph

September 2013

Author:
Gary McGilvary

Supervisor(s):
Thomas Oulevey

CERN openlab Summer Student Report 2013

CERN openlab

Project Specification

CERN is establishing a large scale private cloud based on OpenStack as part of the expansion of the computing infrastructure for the Large Hardon Collider (LHC).

Depending on the application running on the cloud, some virtual machines require large disk capacities or high reliability/performance volumes. This project involves the configuration, deployment and testing of OpenStack Cinder as well as the integration with other block storage alternatives such as NetApp and Ceph.

A performance analysis and comparison between these storage mechanisms will be undertaken to determine the most suitable for use at CERN. Furthermore, modifications will also be made to OpenStack to allow user-specified Ceph data striping values to be set during volume creation as well as the Ceph/QEMU caching method upon volume attachment to an instance.

Abstract

With the ever increasing amount of data produced from Large Hadron Collider (LHC) experiments, new ways are sought to help analyze and store this data as well as help researchers perform their own experiments. To help offer solutions to such problems, CERN has employed the use of cloud computing and in particular OpenStack; an open source and scalable platform for building public and private clouds.

The OpenStack project contains many components such as Cinder used to create block storage that can be attached to virtual machines and in turn help increase performance. However instead of creating volumes locally with OpenStack, others remote storage clusters exist offering block based storage with features not present in the current OpenStack implementation; two popular solutions are NetApp and Ceph.

Two features Ceph offers is the ability to stripe data stored within volumes over the distributed cluster as well as locally cache this data, both with the aim of improving performance. When in use with OpenStack, Ceph performs default data striping where the number and size of stripes is fixed and cannot be changed dependent on the volume to be created. Similarly, Ceph does not perform data caching when integrated with OpenStack.

In this project we outline and document the integration of NetApp and Ceph with OpenStack as well as benchmark the performance of the NetApp and Ceph clusters already present at CERN. To allow Ceph data striping, we modify OpenStack to take the number and size of stripes input via the user to create volumes whose data is then striped according to the values they specify. Similarly, we also modify OpenStack to enable Ceph caching and allow users to select the caching policy they require per-volume. In this report, we describe how these features are implemented.

Table of Contents

1 Introduction

Since 2002, CERN has taken a globally leading role in developing continuously operational Grids such as the Worldwide LHC Computing Grid to handle the large volumes of data, simulation runs and data analyses required by researchers using the LHC. Nowadays with the ever increasing computational and data demands produced from the LHC experiments, CERN is looking at new ways to help store and analyze LHC data as well as find new ways to support their researchers and one such way is turning to the cloud; more specifically, using OpenStack (Andrade, et al. 2012).

OpenStack is a open and scalable cloud platform for building public and private clouds (OpenStack 2013). Hence the use of OpenStack allows CERN to support the data and compute demands resulting from LHC experiments as well as optimize their current IT infrastructure. As part of the expansion of the computing infrastructure for the LHC experiements, CERN is establishing a large scale private cloud based on OpenStack. Due to the large amounts of data researchers need to work with and dependent on the application running on the cloud, some virtual machines may require large disk capacities or high reliability/performance volumes.

The OpenStack Cinder project aims to provide such features by offering "block storage as a service" allowing storage volumes to be attached to virtual machine instances. Typically these volumes reside locally upon nodes within the OpenStack cloud however Cinder can leverage other block storage infrastructures served from remote clusters. These alternative block storage providers may offer different features to take advantage of or offer varying I/O performance levels.

Two prominent block storage clusters currently present at CERN that can be integrated with OpenStack Cinder are:

- **NetApp**: is a company that specializes in networked storage solutions and in turn offers a block storage platform. NetApp delivers a unified storage platform for scalability and flexibility and features clustering capabilities to maintain high availability.

- **Ceph**: is a unified, open source distributed storage platform designed to present different types of storage from a distributed cluster. It boasts excellent performance, reliability and scalability. Performance and reliability can be enhanced by enabling Ceph caching and data striping; the technique of segmenting files and distributing these over separate storage devices.

Both NetApp and Ceph aim to provide the same high-level features (e.g scalability, reliability etc) however their integration and performance when used by OpenStack Cinder are typically untested at CERN. Futhermore, the performance and data reliability of Ceph when integrated with OpenStack may be sub-optimal due to the way Cinder creates Ceph volumes; by default caching is not enabled and data striping is fixed to set

values. The latter limits the achievable performance where the default values may not be suitable to the application and data present on the volume. For example, it may be best to distribute a large number of small sized stripes over different phyiscal disks for a large application.

This project will investigate and solve the above problems via the following aims:

1. **Document the deployment and configuration of OpenStack Cinder, NetApp and Ceph:** *currently no detailed documentation exists within CERN on the actual deployment of Cinder, NetApp and Ceph*

2. **Integrate OpenStack Cinder with NetApp and Ceph**

3. **Test the performance of NetApp and Ceph**: *we do not aim to determine which block storage platform is optimal but merely outline the achievable performance of these clusters at CERN.*

4. **Modify OpenStack to allow per volume data striping values to be set**.

5. **Modify OpenStack to allow a user-defined caching method to be used per volume**.

The rest of this report is organized as follows: Section 2 gives a detailed overview of the use and architectures of OpenStack, NetApp and Ceph. Section 3 outlines the process of integrating these together and their performance results are outlined in Section 4. We then describe how we introduce per volume caching and data striping when OpenStack Cinder uses Ceph for block storage in Section 5 and finally we conclude in Section 6.

Note that we offer detailed documentation on the technical details described in this report which covers:

1. Installation of OpenStack
2. Pre and Post Installation Errors with Solutions
3. Getting Started with OpenStack
4. OpenStack Cinder Integration with NetApp
5. OpenStack Cinder Integration with Ceph

This documentation can be found at:

- https://twiki.cern.ch/twiki/bin/view/AgileInfrastructure/PackstackSlc6 (internal)
- http://garymcgilvary.co.uk/cern.html

2 Background

This section gives the necessary detailed insight into the use and architectures of OpenStack, Cinder, NetApp and Ceph.

2.1 OpenStack

The OpenStack project contains various components that individually provide compute, storage, networking and the dashboard but together create a functioning cloud operating system (OS).

2.1.1 Overview

Currently OpenStack (Grizzly) consists of seven core components as shown in Figure 1 below.

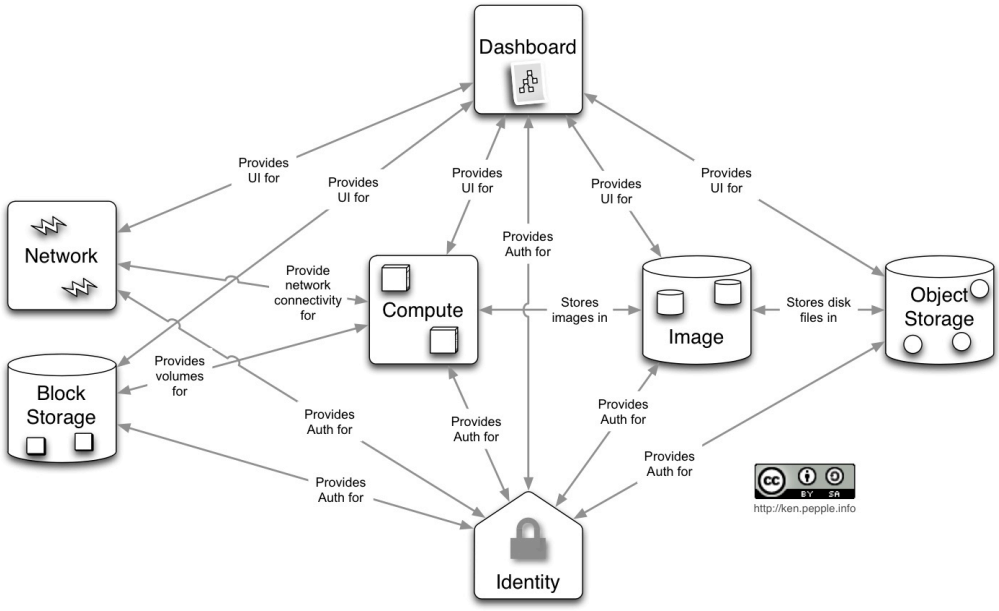

Figure 1. OpenStack Architecture (Pepple 2013)

- **Compute (Nova):** the Infrastructure as a Service (IaaS) system providing virtual machines to hosts with *nova-compute* installed.
- **Identity Service (Keystone):** provides the authentication and authorization for all OpenStack components.
- **Image Service (Glance)**: an image repository for all virtual disk images. Glance can also be configured to store these images on a remote cluster, such as Ceph.
- **Dashboard (Horizon)**: the user interface to easily control most aspects of the OpenStack components. As an alternative, the OpenStack API can be used.
- **Networking (Neutron)**: provides "networking as a service" by allowing users to create their own networks and interfaces as well as manage IPs.
- **Object Storage (Swift)**: is a highly available and distributed object/blob store.
- **Block Storage (Cinder)**: provides "block storage as a service".

The four components that are relevant to this project are Nova, Glance, Horizon and Cinder.

2.1.2 Installation

We installed OpenStack Grizzly on three SL6 CERN servers via Packstack from RDO; a tool that uses Puppet modules to deploy the various components of OpenStack on single or multiple servers (RDO 2013). The installation prerequistes as well as how to install OpenStack via Packstack is explained in our OpenStack documentation.

As Packstack is a relatively new tool, after installation, errors appeared both on the Horizon interface as well as component logs signifiying that installation was not correctly performed. The issues faced as well as the solutions are documented at the above links. After such problems were resolved, we had the setup shown in Figure 2.

One node was assigned to be the OpenStack controller where most service components were installed. In our case, Horizon, Keystone, Cinder, and Glance was installed on the controller node to provide a central server for these services to be accessed.

The compute nodes had Nova installed to allow virtual machines to run upon these servers. The controller node also had Nova installed making this a joint compute node.

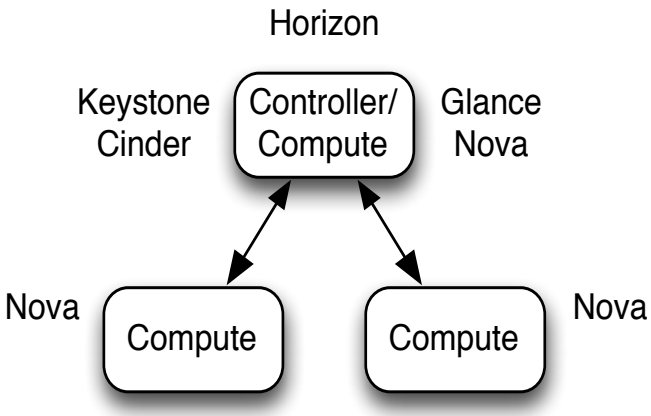

Figure 2. OpenStack Hosts and Environment

Our OpenStack hosts have the following hardware specifications:

Host	CPU	Disk	RAM	L1 Cache	L2 Cache	L3 Cache
lxfssmA	2 x 2.27GHz	~40 TB	12 GB	256KB	1 MB	8 MB
lxfssmB	2 x 2.27GHz	~40 TB	12 GB	256KB	1 MB	8 MB
lxfssmC	2 x 2.27GHz	~6 TB	12 GB	256KB	1 MB	8 MB

Table 1. Hardware Specifications of OpenStack Hosts

We see that the three hosts *lxfssmA, lxfssmB,* and *lxfssmC* have the equivalent number of CPUs, RAM and size of level one, two and three caches. The host *lxfssmC* has approximately 34TB less disk space available hence this was selected as a compute node while *lxfssmA* is the OpenStack controller node.

2.2 NetApp

Network Appliance, or NetApp for short, is a company specializing in networked storage and data management solutions.

2.2.1 Overview

NetApp offers the NetApp filer, a disk storage device that offers varying types of storage over a network. NetApp filers do not use existing commodity hardware but instead use highly customized hardware using NetApp's own Data ONTAP OS. This OS provides cluster availability, scalability and efficiency. Data ONTAP offers file-based storage by the protocols NFS, CIFS, FTP, TFTP and HTTP and block-based storage by the FC, FcoE and iSCSI protocols; we are interested in the block-based storage offerings and more particulary via the use of iSCSI.

The Internet Small Computer System Interface (iSCSI) protocol is an IP based protocol used to let clients (initiators) communicate with I/O storage devices (targets) by sending SCSI packets over TCP/IP (Huffered 2003). OpenStack offers a NetApp iSCSI direct driver meaning the iSCSI connection goes directly to the virtual machine, as shown in Figure 3.

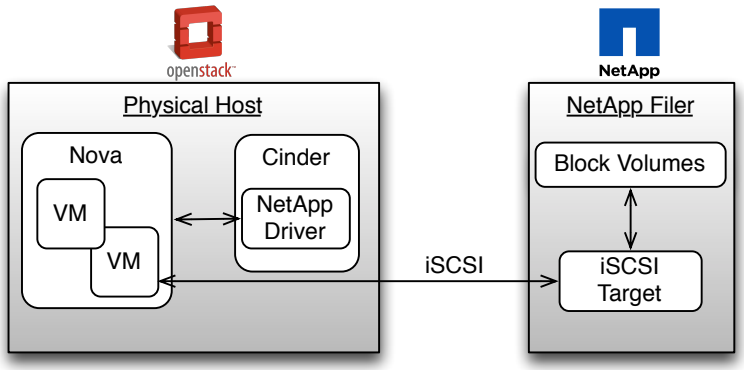

Figure 3. High-level Overview of OpenStack and NetApp iSCSI

When in use with OpenStack, NetApp receives iSCSI packets from the virtual machine which is configured to use the NetApp service via Cinder. The OpenStack virtual machines can then mount and make use of the remote volume, or Logical Unit Number (LUN); an addressable logical SCSI device that is physically connected to the SCSI target. The NetApp iSCSI direct driver is the driver we are interested in testing.

2.2.2 NetApp at CERN

The use of NetApp in CERN is highly documented mainly in conjunction with the use of Oracle Databases hence it appears many NetApp clusters are presently being used within CERN (NetApp 2013) (Daniel 2010). The specifications of the NetApp cluster we used was:

- FAS2040 Storage Systems
- Data ONTAP 8
- 52 Disks

2.3 Ceph

Ceph is a massively scalable, open source, software-defined storage system that runs on commodity hardware (Ceph 2013).

2.3.1 Overview

Ceph delivers three types of storage: *object*, *block* and *filesystem*.

These storage types are offered from the Ceph Storage Cluster known as the Reliable, Autonomic, Distributed Object Store (RADOS). RADOS is self-managing, self-healing, has no single points of failure and can be massively scalable. The object store is accessed via the RADOS Gateway (RGW) and block storage can be provisioned through the RADOS Block Device (RBD) service, both of which use ***librados*** to access RADOS. These components and their interactions are shown in the architecture diagram below.

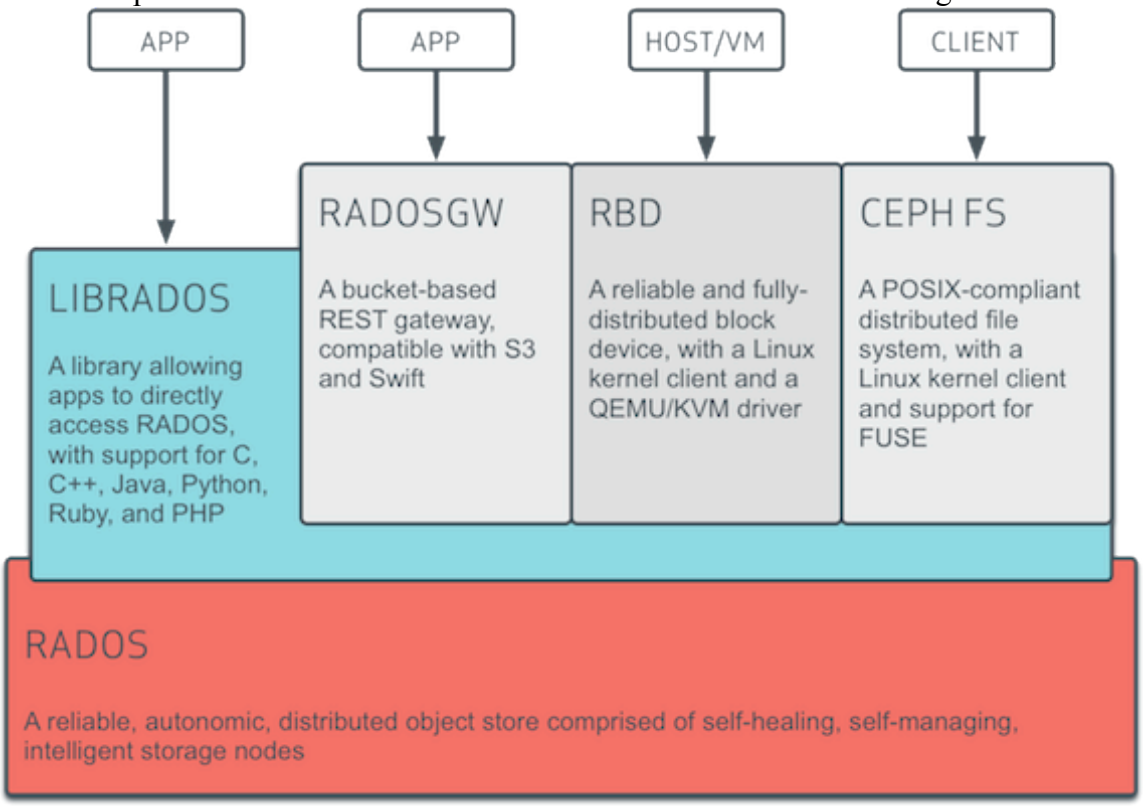

Figure 4. Ceph Architecture (Ceph 2013)

To provide high reliability and performance, the RADOS service consists of a number of monitoring node daemons (MONs) and object storage daemons (OSDs). The purpose of MONs are to maintain a master copy of the cluster map containing the cluster membership, configuration and state; many of these daemons may run where at least 3 is recommended. OSDs serve and store objects to and from clients; again many of these daemons may run concurrently upon a Ceph cluster to improve performance.

When in use with OpenStack Cinder, Ceph uses RBD, and more specifically the **librbd** library, to interact with RADOS and subsequent OSDs, as shown in Figure 5. This also shows **libvirt**, a tool to interact and manage virtualization on Linux. This configures the OpenStack QEMU/KVM virtual machines to use **librbd**, in turn allowing RBD block devices to be attached and I/O operations to be performed on the remote cluster. Not only can Ceph be integrated with Cinder to make use of block devices, it can be integrated with OpenStack's image service Glance to provide a backend for virtual machine images on RBD volumes.

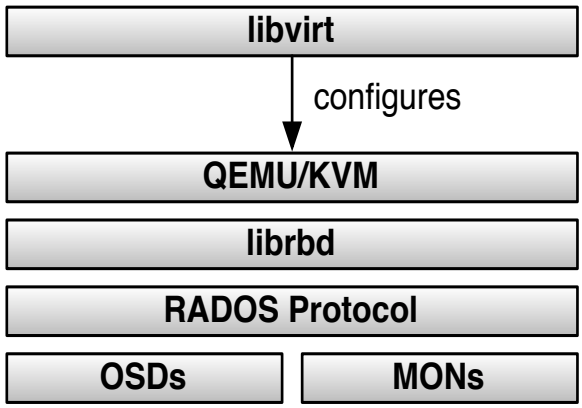

Figure 5. libvirt and librbd

2.3.2 RBD Striping

Regardless of whether Cinder or Glance is integrated with Ceph, to increase I/O performance, data stored upon these RBD volumes is striped across multiple Ceph objects and stored by RADOS meaning a single disk's performance does not become the bottleneck when performing I/O operations. Ceph uses a modified RAID 0 data distribution where data is stored upon one object up to a fixed size then the next object is used to store the data, and so on. Two important variables exist to control the how data striping operates:

- **stripe_unit**: the size of a single stripe unit.
- **stripe_count**: the number of objects to distribute segments of size 'stripe unit' before looping back to the first object.

Upon creation of a RBD volume via OpenStack, Ceph provides default values for these variables and hence data upon these volumes are always striped in the same way regardless of whether the application/data running upon the volume would best be suited to another striping setup to increase its performance. This is one of the problems we aim to solve.

2.3.3 RBD Caching

By default, OpenStack does not perform any caching however Ceph can take advantage of caching via **librbd**. As **librbd** cannot take advantage of the Linux page cache, it provides its own in-memory implementation of caching. This is however disabled by default and Ceph does this for data safety. A host failure or power disruption could corrupt data if QEMU exits uncleanly or data may even be lost.

RBD caching can be enabled in two ways:

1. Modify the */etc/ceph/ceph.conf* file – Ceph's configuration file – to include:

```
rbd cache = true            // for write-back caching     (1)
rbd cache max dirty = 0 // for write-through caching      (2)
```

If (1) is only included, write-back caching is performed by default; this is recommended by Ceph if caching is enabled. We can select write-though by including (2) meaning writes only return when the data is present on all replicas but reads can come from the cache. Other caching methods (i.e a mix of both strategies) can be found at: http://ceph.com/docs/next/rbd/rbd-config-ref/#rbd-cache-config-settings

2. Modify the */etc/nova/nova.conf* file – Nova's configuration file – to include:

```
disk_cachemodes="network=writeback"
```

This simply enables Ceph caching and sets the cache policy to writeback; writethough could also be entered here. It does this by entering the selected policy into an XML file which **virsh –** the command line interface to **libvirt** - uses to attach the previously created RBD volume to the instance. For example, this XML file would look something like:

```
<disk type='network' device='disk'>
  <driver name="qemu" type="raw" cache="writeback"/>
  <source protocol="rbd" name="volume_pool/volume_name">
    <host name='XXX.XXX.XXX.XXX' port='6789'/>
  </source>
  <target dev="vdb" bus="virtio"/>
  <auth username='volumes'>
    <secret type='ceph' uuid='232g33rf32ed12ed2ed2'/>
  </auth>
</disk>
```

This file specifies that the disk is accessible over the network and is accessed using the RBD protocol. The remote path and volume name are specified as well as the server connect to and the relevant authentication. The disk is then attached to the target device to be exposed to the guest OS.

The user-specifed caching policy is entered from the *nova.conf* file and hence writeback caching will be employed on the Ceph client. If caching was only enabled via *ceph.conf* or not enabled at all, the cache variable within the XML would be set to 'none', i.e cache="none".

2.3.4 Ceph at CERN

CERN actively make use of their own Ceph cluster. At the time of writing, CERN was currently upgrading their Ceph cuttlefish 'Andy' cluster to 'Beesly' which uses Ceph dumpling; all our work and experiments took place on the 'Andy' cluster. This cluster had the following specifications:

- 47 OSD servers, each with:
  - 24 x 3TB Hitachi CoolSpin drives (5900rpms)
  - 64 GB RAM
  - 2 x Intel(R) Xeon(R) CPU E5-2650 (16 cores in total)
  - 10Gbit ethernet NIC
- 3 MON servers where each is an OpenStack virtual machine with 2 cores with 4GB RAM.

3 Integrating OpenStack Cinder with NetApp and Ceph

We now discuss how both NetApp and Ceph can be integrated with OpenStack Cinder. OpenStack Cinder uses iSCSI that employs the use of the Logical Volume Manager (LVM), which has the purpose of virtualizing and managing disks, to expose volumes to virtual machine instances. OpenStack Cinder uses a driver to create volumes of a specific type and by default uses the LVM iSCSI driver; this can be explicitly set in the */etc/cinder/cinder.conf* file as:

```
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
```

If one wishes to create a volume of a different type, the **volume_driver** variable requires modification as well as the likely addition of other variables however this is dependent on the driver. Note that the Cinder integration with NetApp and Ceph as well as the errors and solutions faced can be found in much more detail than below in our OpenStack documentation.

3.1 NetApp

Integrating NetApp with OpenStack Cinder is a relatively easy process. After obtaining a username and password to access the NetApp cluster, we need to modify the */etc/cinder/cinder.conf* file to include the necessary directives:

```
volume_driver=cinder.volume.drivers.netapp.iscsi.NetAppDirect7mode
ISCSIDriver
netapp_server_hostname=lxfssmXXXX.cern.ch
netapp_server_port=8088
netapp_login=username
netapp_password=password
```

This configuration specifies that the NetApp iSCSI direct driver should be used and the iSCSI target is located on the specified host via port 8088. We also enter the username and password to allow OpenStack access to the NetApp service. Many other directives are available however they are not mandatory to use the CERN NetApp services.

In theory, one should now be able to create NetApp volumes via the Horizon interface however in our experience things were much more difficult due to permission problems. These originated from Cinder not having the required permission to successfully issue API calls on the NetApp filer. The required API calls to gain access to were:

```
login-*,api-clone-start,api-clone-list-status,api-nfs-exportfs-
storage-path,api-lun-*,api-volume-options-list-*,api-volume-list-
*,api-igroup-*,api-iscsi-*,api-storage-shelf-list-info,cli-version
```

After the correction of a few minor errors, OpenStack was then able to create, attach, format and mount NetApp volumes.

3.2 Ceph

To integrate Ceph successfully with OpenStack Cinder, one must first obtain a configuration file (*ceph.conf*) and a keyring (*ceph.client.pool_name.keyring*) to access a particular RADOS pool. These are segregated areas for storing data which offer replication, restricted access etc. We had access to the following pools:

- volumes: where we store the RBD block volumes
- images: where we store the virtual machine images

We discuss how to integrate both OpenStack volumes (Cinder) and images (Glance) with Ceph below but first we discuss the necessary prerequisites.

3.2.1 Prerequisites

The necessary prerequisites are discussed in our OpenStack documentation however we do highlight one important requirement here. At the time of writing, the Red Hat version of QEMU did not support the attachment of RBD volumes and when doing so resulted in the error:

"internal error unable to execute QEMU command '__com.redhat_drive_add': Device 'drive-virtio-disk1' could not be initialized"

To avoid this, QEMU needs to be replaced with the QEMU RPMs available from Ceph; (http://ceph.com/packages/qemu-kvm/centos/x86_64/).

3.2.2 Block Storage

Similar to the integration of NetApp, Ceph requires the RBD driver to be specified as well as directives relating to authentication. Again these must be placed within the */etc/cinder/cinder.conf* file and are:

```
volume_driver=cinder.volume.drivers.rbd.RBDDriver
rbd_pool=volumes
rbd_user=volumes
rbd_secret_uuid=the_secret_key
glance_api_version=2
```

This configuration shows that the RBD driver should be used and the remote volumes are accessible within the pool 'volumes' and are accessible only by user 'volumes'. The generated secret key is tied to our Ceph keyring enabling **virsh** to attach the remote volume to our instance. Note that the remote server hostname is not present in this configuration file however it must be specified within */etc/ceph/ceph.conf*. For example:

```
[global]
  mon host = ceph01.cern.ch
```

After correcting a few errors as discussed in our OpenStack documentation, we were able create, attach, format and mount Ceph volumes.

3.2.3 Image Storage

Ceph not only provides block storage but allows RBD volumes to become a backend for storing Glance images. To configure Glance to use Ceph, we must modify/edit */etc/glance/glance-api.conf*:

```
default_store=rbd
rbd_store_user=images
rbd_store_pool=images
rbd_store_ceph_conf=/etc/ceph/ceph.conf
```

The first directive is present within *glance-api.conf* and must be modified from its initial value 'file' to 'rbd'. In similar fashion of configuring Cinder, we must include the pool name and user which has access to this pool. Finally, the Ceph configuration file must be listed to ensure the Glance service knows how to access the Ceph cluster.

After creating a Glance image, we should now see that the image now resides on Ceph within the correct pool as opposed to on the OpenStack controller node.

3.3 Cinder Multi-backend

So far we have only described how a single driver (e.g NetApp or Ceph) can be used to create and attach volumes from a single source. OpenStack Cinder allows multiple drivers to be used concurrently without the need to modify the Cinder configuration file and restart the service each time a user wishes to create a volume from a different source. To do this we have to set the following directives in */etc/ceph/ceph.conf*:

```
enabled_backends=netapp-driver,rbd-driver

[netapp-driver]
#Normal NetApp directives
…
volume_backend_name=netapp_backend

[rbd-driver]
#Normal RBD directives
…
volume_backend_name=rbd_backend
```

First we define **enabled_backends** to include the names of the configuration groups and within these groups we specify the directives for the respective drivers, for example, those outlined above for NetApp and Ceph. We must also give the enabled backends a name. For each driver listed, the users have to manually create a Cinder volume type and link this type to the backend name meaning a user can select the driver they wish to use via the Horizon interface. This looks as follows:

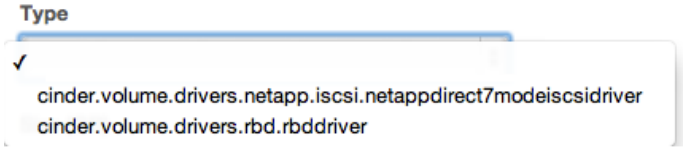

Figure 6. Volume Type Selection via Horizon

The process of manually creating Cinder volume types for each driver and linking this to the backend name is one that can be performed automatically by OpenStack with some additions. We have modified the OpenStack source (*cinder/manager.py*) to include this functionality.

```
types_reset = False

def create_volume_type(self, context, volume_driver):
   volume_types.create(context,volume_driver.lower(),extra_specs={})

def destroy_volume_type(self, context):
   vts = volume_types.get_all_types(context)
   for x in vts:
      volume_types.destroy(context,vts[x]["id"])

def init_volume_types(self,context, volume_driver):
   type_present = False

   if VolumeManager.types_reset == False:
      self.destroy_volume_type(context)
      VolumeManager.types_reset = True

   vts = volume_types.get_all_types(context)

   for x in vts:
       if volume_driver.lower() in vts[x]["name"]:
            type_present = True
   if type_present == False:
       self.create_volume_type(context, volume_driver)
```

When the first driver is loaded, we call **init_volume_types** which then destoys all previous volume types by calling the *destroy* function from the OpenStack **volume_types.py** class. We then create the volume type based on the volume driver name and link it to a self created backend name. When subsequent drivers are loaded, we ensure previous volume types remain and again create the volume type based on the drivers name. This removes the requirement of the user manually creating volume types for each of their drivers.

4 The Performance of NetApp and Ceph

We now describe the performance achieved when running a number of benchmarks upon the NetApp and Ceph clusters.

4.1 Benchmark Setup

To benchmark a cluster, we instantiated three virtual machines and attached and mounted a disk of a specific size from the respective cluster to each virtual machine. The three disk sizes were 100GB, 200GB and 400GB and were each mounted to the virtual machines A, B and C respectively; 400GB was the largest disk size we could create using the NetApp cluster. Our experimental setup is shown below.

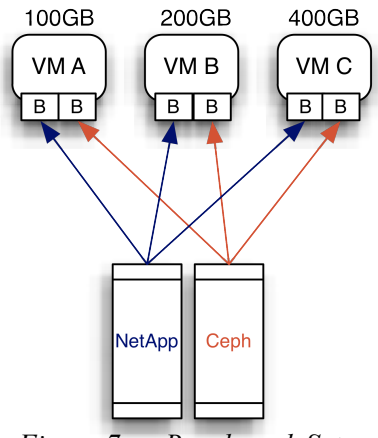

Figure 7. Benchmark Setup

The purpose of this configuration was to:

1. determine the performance difference between NetApp and Ceph in their respective setups.
2. determine if any performance differences exist dependent on the size of the disk.
3. determine if there is any performance degradation if other disks present within the cluster are being heavily used; we performed this by running the benchmarks on each disk at the same time.

These experiments were first performed on the NetApp cluster and soon after, upon the Ceph cluster, by running benchmarks upon the attached formatted disk. Each benchmark was executed five times and the results were averaged. The results shown display the results from the best performing disk from either the 100GB, 200GB and 400GB disks.

4.2 Benchmarks

We used three benchmarks to test the performance of NetApp and Ceph to get a better understanding of the performance of each cluster. The benchmarks used are described in the following subsections.

4.2.1 hdparm

hdparm is a disk performance benchmarking and configuration tool. We used *hdparm* to test the speed of buffered, cached and reads (Aspinwall 2009).

Buffered reads use the file system buffer cache hence *hdparm* measures the speed of reading through this cache to the disk without caching any data previously. Cached reads are performed directly from the cache hence *hdparm* measures the speed when reading from the cache only. Finally *hdparm* also measures the speed of direct reads where the

file system buffer cache is bypassed allowing applications to perform asynchrounous overlapping I/O operations.

4.2.2 FIO

FIO: the Flexible I/O Tester is a workload generator and benchmarking tool (FIO 2013). We use FIO to measure aggregate read and write bandwidth. To use FIO, one must create a job description file which describes the workloads to be executed. As CERN is quoted to perform 20% writes and 80% reads, we created multiple job description files that mimic the same level of I/O operations for both sequential and random read and writes. An example of the latter is displayed below:

```
[read-write]
rw=randrw
size=16g
directory=/root/disk/fio/CERN/rrw
blocksize=256k
numjobs=25
nrfiles=1
filesize=320m
runtime=10m

[data02]
rw=randread
size=16g
directory=/root/disk/fio/CERN/rrw/d2
blocksize=256k
numjobs=25
nrfiles=1
filesize=320m
runtime=10m

[data03]
rw=randread
size=16g
directory=/root/disk/fio/CERN/rrw/d3
blocksize=256k
numjobs=25
nrfiles=1
filesize=320m
runtime=10m

[data04]
rw=randread
size=16g
directory=/root/disk/fio/CERN/rrw/d4
blocksize=256k
numjobs=25
nrfiles=1
filesize=320m
runtime=10m
```

This file specifies that four processes execute: one to perform random read and write operations and the remaining three to randomly read data. In this configuration, direct I/O is disabled however can be added by specifying *direct=1*. Each process has a buffer cache of 256KB and creates 25 instances of itself to operate over its own 320MB file located within the directory listed. The read and write processes halt once 16GB of data has been read and writen. If these operations have not completed within 10 minutes, the

processes terminate and the results displayed. These results are summarized at the end of each benchmarking run; an example is shown below.

```
Run status group 0 (all jobs):
   READ:   io=28219MB,   aggrb=110442KB/s,   minb=660KB/s,   maxb=2457KB/s,
mint=133343msec, maxt=261638msec
  WRITE:   io=3781.3MB,   aggrb=14799KB/s,   minb=591KB/s,   maxb=679KB/s,
mint=227813msec, maxt=261638msec

Disk stats (read/write):
  vdb:    ios=120303/18169,    merge=0/947355,    ticks=10116683/53279160,
in_queue=61857457, util=100.00%
```

The result we are interested in the aggregate bandwidth for each process (underlined), i.e the combined bandwidth for all process instances for read and write operations.

4.2.3 dd

dd is the Linux function used to covert and copy files (GNU 2013). This function however displays the read and write speeds when writing to and reading from files. We used *dd* to read and write from 1GB and 10GB files and recorded the speeds achieved. An example of the former case is shown below.

```
dd if=/dev/zero of=1GBFile bs=1073741824 count=1 &> write.txt   (A)
dd of=/dev/zero if=1GBFile bs=1073741824 count=1 &> read.txt    (B)
```

We see that to create a file (A), we take the input from */dev/zero* with a block size of 1GB, output the contents to a file named 1GBFile and capture the function's output within *write.txt*; this process is similar of reads as shown in (B). In this configuration, direct I/O is disabled however is can be enabled by specifying *oflag=direct*. To test the affect of virtual machine caching, each execution was run when the cache was enabled and then disabled (or emptied); the latter can be performed by executing:

```
sync; echo 3 > /proc/sys/vm/drop_caches
```

4.3 Benchmark Results

We now show the results obtained from running the above benchmarks on NetApp and Ceph. Note that the NetApp and Ceph clusters have different configurations, varying numbers and types of hard disks as well as different network connections; 1Gbit and 10Gbit for the NetApp and Ceph clusters respectively. Hence our aim is not to determine which cluster is best overall but to test the cluster's achievable performance in their respective configurations.

4.3.1 NetApp versus Ceph

Firstly we see the results from the *hdparm* benchmark in Figure 8. The speed of cached reads are similar for both NetApp and Ceph where NetApp performs only slightly better at approximately 6GB/s. However, Ceph outperforms Netapp for both buffered and direct reads by being approximately two and eight times respectively.

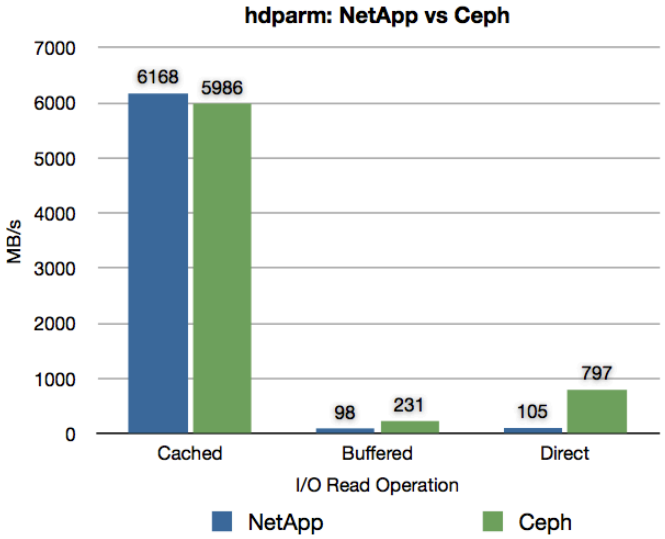

Figure 8. hdparm Benchmark Results

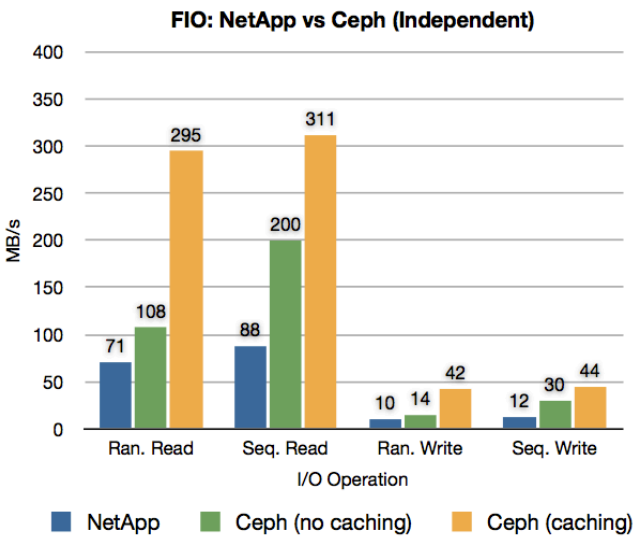

Figure 9. FIO Benchmark Results

Similar to *hdparm*, FIO (Figure 9) also shows that Ceph outperforms NetApp both when Ceph caching is disabled and enabled; by default Ceph caching is disabled. Firstly, taking the scenario when Ceph caching is disabled, the speed of sequential reads are similar to the buffered read performance of *hdparm*; the type of reads *hdparms* uses. Furthermore, random reads as well as random and sequential writes are also faster using Ceph but with less of a difference between the two.

Secondly, when Ceph writeback caching is enabled upon the host, we see that the performance gap between NetApp and Ceph widens as well as the perforance gap between the non-cached version. For random and sequential reads, Ceph can be approximately three times as faster than NetApp when caching is enabled; similarly, Ceph's write performance can be four times faster. In our experience, little performance gain is achieved using writethrough caching with Ceph.

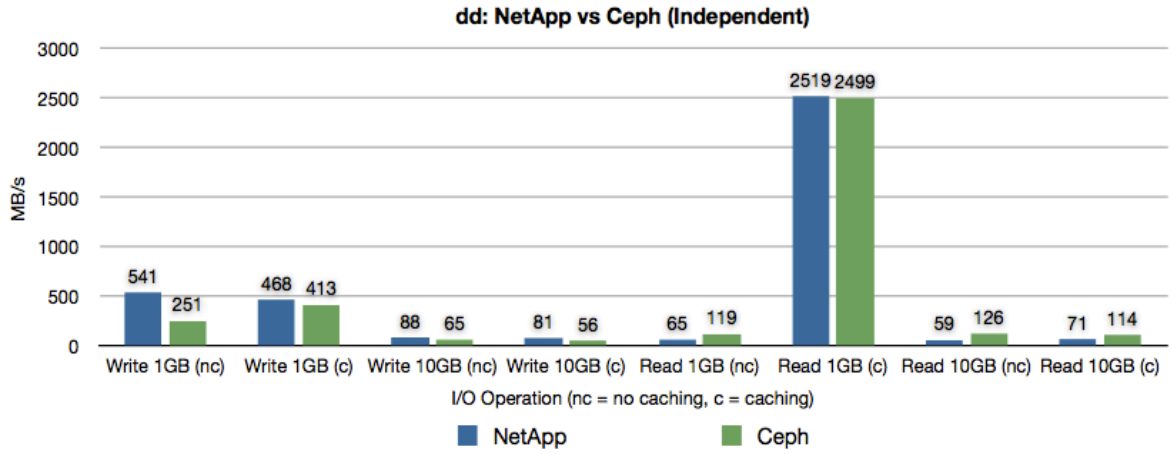

Figure 10. dd Benchmark Results

Finally, running the *dd* benchmark in the same way as *hdparm* and FIO shows some different results. We see from Figure 9 that NetApp is faster at writing to both a 1GB and 10GB file however Ceph is typically faster at reading these files. We also see the

performance differences when virtual machine caching is either enabled or disabled however it is difficult to provide an overall comparison when either method is employed. Despite this, we can see that the use of the virtual machine cache can be beneficial for both NetApp and Ceph when the speed of reading a 1GB file is approximately 2.5 GB/s.

As all benchmarks were run multiple times on the three different sized disks, we found that the performance of each disk usually remains equal despite the size being different for both NetApp and Ceph; of course there were a few exceptions where some disk's performance is much larger or less than another however no correlation could be found between disk size and achievable performance.

4.3.2 Miscellaneous: Direct and Concurrent Benchmarks

By default, the above benchmarks were performed with direct I/O disabled and each benchmark was performed independently, i.e. only upon one disk at a time. We also performed the same benchmarks with direct I/O. It was found that the random read and write performance increased for both NetApp and Ceph however the sequential read and write performance only slightly increased with Ceph and remained unchanged with NetApp.

Executing the same benchmark on all three mounted disks concurrently did however decrease NetApp's achievable performance by approximately 50% for both the FIO and *dd* benchmarks. The performance of FIO when executing concurrently on our three Ceph volumes remained unchanged however a slight decrease in performance was observed when running the *dd* benchmark. This can be explained by the different configurations of the NetApp and Ceph clusters and in particular the network speeds available to each.

5 Ceph: Adding Striping and Caching to OpenStack

Using OpenStack with Ceph by default does not allow users to specify the *stripe_count* and *stripe_unit* parameters when volumes are created; defaults values are always used for all volumes. Similarly, Ceph caching is disabled by default however when caching is enabled via the *ceph.conf* file, this caching method is used for all RBD volumes and as such, users are not able to specify the caching method they wish to use per volume.

We now describe how we have modified OpenStack to allow these features to be enabled.

5.1 Adding RBD Striping

To obtain the *stripe_count* and *stripe_unit* parameters from the user, we first have to add the respective text fields to the Horizon interface when a user creates a volume; we do this by modifying *horizon/openstack_dashboards/dashboards/project/volumes/forms.py* The stripe fields should however only appear when the RBD driver is in use. We do this by creating a volume type for each of the drivers enabled (see Section 3.3) and when the volume type associated with the RBD driver is selected (in our case this is 'cinder.volume.drivers.rbd.rbddriver') the *stripe_count* and *stripe_unit* text fields dynamically appear. This is depicted in Figure 11.

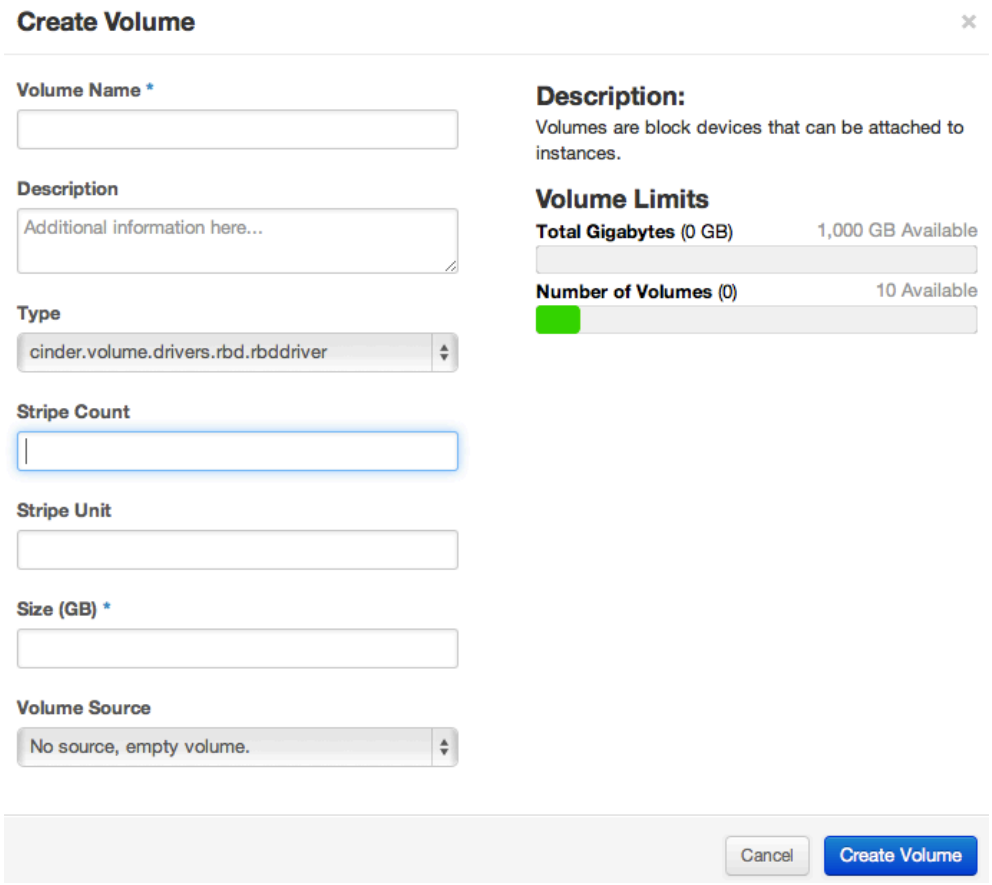

Figure 11. The modified 'Create Volume' submission form including stripe parameters

This dynamic appearance is achieved by using OpenStack's 'switchable' and 'switched' fields. By marking a field as switchable, we specify that this is a field which is used to instruct other fields (i.e the switched fields) to display or not dependent on the option selected. In our case, the volume type field is switchable and the stripe fields are of type switched:

```
type = forms.ChoiceField(label=_("Type"), widget=forms.Select(
                                attrs={'class': 'switchable',
                                       'data-slug': 'type'}),
                                       required=False)

stripe_count = forms.IntegerField(min_value=1, label=_("Stripe Count"),
        widget=forms.TextInput(attrs={
                'class': 'switched',
                'data-switch-on': 'type',
                'data-type-cinder.volume.drivers.rbd.rbddriver':
                _('Stripe Count')}), required=False)

stripe_unit = forms.IntegerField(min_value=1, label=_("Stripe Unit"),
         widget=forms.TextInput(attrs={
                'class': 'switched',
                'data-switch-on': 'type',
                'data-type-cinder.volume.drivers.Rbd.rbddriver':
                _('Stripe Unit')}), required=False)
```

The stripe fields will then appear when the string *cinder.volume.drivers.rbd.rbddriver* is selected via the type field. If stripe values are entered, the values subsequently entered are then sanitized to ensure they are valid according to the size of the disk and then are passed as parameters to many method calls which store these values in the database and create the volume. We perform these checks by using the following snippet of code:

```
try:
   stripe_count = int(data['stripe_count'])
   stripe_unit = int(data['stripe_unit'])

   if ((int(data['size']) * 1073741824) %(stripe_count * stripe_unit) != 0):
        error_message = _('Please Enter Valid Striping Parameters!')
                           raise ValidationError(error_message)
except Exception:
   stripe_count = 0
   stripe_unit = 0

volume = cinder.volume_create(request,
                              data['size'],
                              data['name'],
                              data['description'],
                              data['type'],
                              snapshot_id=snapshot_id,
                              image_id=image_id,
                              metadata=metadata,
                              stripe_count=stripe_count,
                              stripe_unit=stripe_unit)
```

First we check if values are entered or not by using *try except*. Leaving either of these fields empty will cause an exception to occur and default values to be set. If values are present, they are check to ensure that the stripe parameters are correct according to the disk size, if not an error message is displayed to the user. We then pass these values to the

cinder.volume_create method which then passes these values to subsequent method calls that are used to create volumes. The following files containing methods we have modified are:

- horizon/openstack_dashboard/api/cinder.py
- python-cinderclient/cinderclient/v1/volumes.py
- python-cinderclient/cinderclient/v2/volumes.py
- cinder/api/v1/volumes.py
- cinder/api/v2/volumes.py
- cinder/volume/api.py
- cinder/volume/flows/create_volume/__init__.py
- cinder/volume/drivers/rbd.py

Apart from the latter RBD driver, the modifications to the other files are small hence they are not described here. We show the changes we have made to the RBD driver to allow the stripe parameters to be specified during volume creation below:

```
rbd_stripe_count = int(volume['stripe_count'])
rbd_stripe_unit = int(volume['stripe_unit'])

if rbd_stripe_count != 1:
     image_format = 2
else:
     image_format = 1

args = ['rbd', 'create',
        '--pool', self.configuration.rbd_pool,
        '--size', size,
        '--stripe_count', rbd_stripe_count,
        '--stripe_unit', rbd_stripe_unit,
        '--image-format', image_format,
        volume['name']]
self._try_execute(*args)
```

We first retrieve the stripe parameter values from the database and then perform a simple check to determine if the values are default or not. Currently Ceph has two formats of images which can be used to create a volume. The second image format is the only format that allows non-default striping values to be used during creation. We therefore set the image format number dependent on whether default stripe parameter values were entered or not. Finally, we append the '—stripe_count', '—stripe_unit' and '—image-format' parameters to the command that is executed to create the volume.

One can confirm that the volume has been properly created using the specified stripe parameters by executing the *rbd info volume_name* on the command line.

We have submitted these modifications upstream to the OpenStack project and hopefully will be merged into a future release of OpenStack. These modifications can be found at:

- https://review.openstack.org/#/c/47469/
- https://review.openstack.org/#/c/47471/
- https://review.openstack.org/#/c/47473/

5.2 Adding RBD Caching

In order to allow different per-volume caching policies to be selected, we first have to make available the caching policies to the user via the Horizon interface when a user wishes to attach an existing volume to an instance. This is shown in Figure 12.

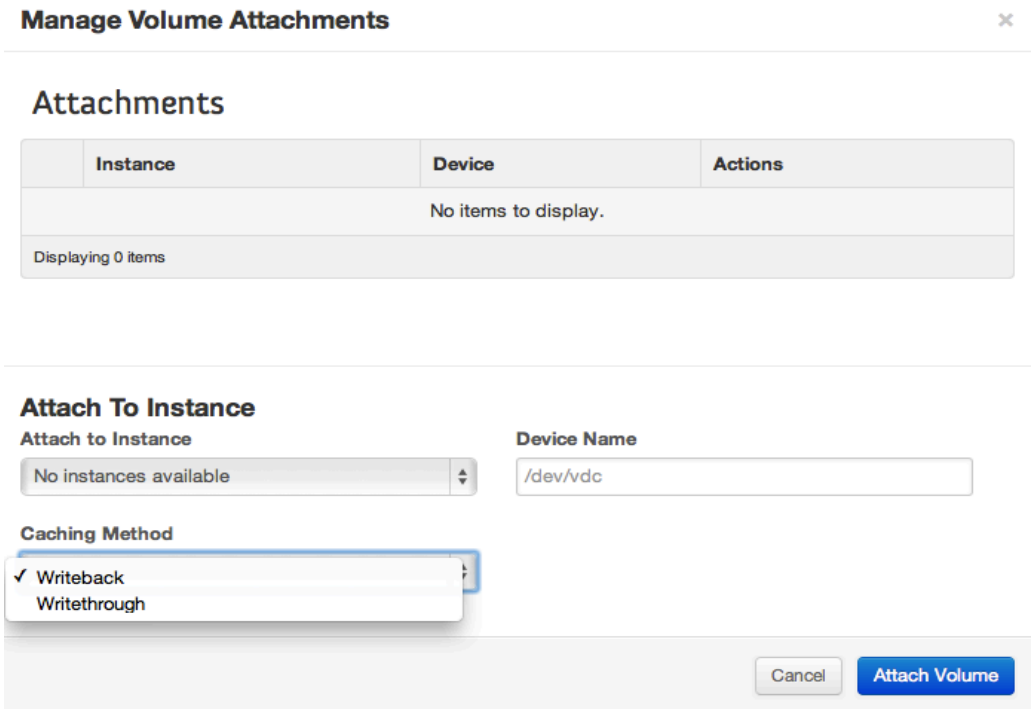

Figure 12. The modified 'Attach Volume' form including per-volume caching policies

We add the selection field similar to the way the 'type' field is added in the case of introducing striping parameters.

```
cache_policy = forms.ChoiceField(label=_("Caching Method"), required=False)
```

Afterwards the the cache policies are then populated; we only offer two types of caching: writeback and writethrough. The selected policy is then passed through many method calls and is eventually entered dynamically into the XML file created (see Section 2.3.3) to allow **libvirt** to attach the volume to the instance. This XML is created within *nova/nova/virt/libvirt/volume.py* and allows the user defined preference to be set:

```
conf.device_type = disk_info['type']
conf.driver_format = "raw"
conf.driver_cache = disk_info['cache_policy']
conf.target_dev = disk_info['dev']
conf.target_bus = disk_info['bus']
conf.serial = connection_info.get('serial')
```

Currently, we have an 'unstable' version of this feature available and requires more implementation and stability before being submitted to OpenStack for integration in future releases.

6 Conclusions

In this report, we have outlined the progress made during the project 'Implementation of OpenStack Cinder and Integration with NetApp and Ceph.' First we showed the infrastructure made available to this project and how OpenStack was deployed upon these servers; a process that was documented to allow future employees and summer students to find it easier to install the open source platform.

The presence of NetApp and Ceph clusters within CERN gave us the opportunity to test their ease of integration with OpenStack. We found that in theory, integration is easy and can be achieved by entering a small number of lines within the Cinder configuration file, however in reality many problems existed, in particular with NetApp.

The performance of both these storage clusters were tested in their respective configurations and because of this, a direct comparsion and ultimate conclusion outlining which is best cannot be offered. However our results did show that Ceph is typically faster at reading data however it is difficult to include which is best for writing files. The appropriate cluster for use at CERN would not only be which offers the best performance but also provides the required features (scalability, relability etc) dependent on the applications using block-storage volumes.

To potentially increase the performance of Ceph storage volumes, we introduced per-volume data striping when creating volumes. Currently, data striping values are set by default and cannot be changed when Ceph is integrated with OpenStack. By allowing user-specified *stripe_count* and *stripe_unit* parameters to be set, data can now be striped optimally dependent on the application or data present on the RBD volume. We outlined how this feature was implemented and we hope that it's submission upstream to OpenStack will allow this feature to be available to all in a future release.

Ceph caching – a feature which is disabled by default – is also enabled to increase performance. Dependent on the application/data present on the volume, the user can select either the writeback or writethrough caching polices for each of the volumes to be attached to an instance. At the time of leaving CERN, this implementation was still in progress and hence becomes future work. Despite this, we did however show that using writeback caching can potentially increase read and write performance by up to 3 or 4 times and writethrough caching gives only a slight performance increase.

7 Acknowledgements

Finally, I would like to thank my supervisor Thomas Oulevey for all the support received during the course of the program. Also thank you to the OpenStack group the Openlab administration team (in particular Jasime, Sebastian and Kristina) who helped significantly during my stay, especially in the first week!

8 Bibliography

Andrade, P, et al. “Review of CERN Data Centre Infrastructure.” *Journal of Physics: Conference Series*, 2012.

Aspinwall, Jim. *PC Hacks: 100 Industrial-Strength Tips & Tools.* O'Reilly Media, 2009.

Ceph. *Ceph.* 2013. http://ceph.com.

Daniel, Eric Grancher and Steve. *CERN Openlab.* 2010. http://openlab.web.cern.ch/sites/openlab.web.cern.ch/files/presentations/2010OOW-08.pdf.

FIO. *FIO - Freecode.* 2013. http://freecode.com/projects/fio.

GNU. *dd invocation - GNU Coreutils.* 2013. http://www.gnu.org/software/coreutils/manual/html_node/dd-invocation.html.

Huffered, John L. *ISCSI: The Universal Storage Connection.* Addison-Wesley Professional, 2003.

NetApp. *NetApp Storage Helps CERN Unlock the Secrets of the Universe.* 2013. http://www.netapp.com/us/company/news/press-releases/news-rel-20120926-758337.aspx.

OpenStack. *OpenStack Open Source Cloud Computing Software.* 2013. https://www.openstack.org.

Pepple, Ken. *OpenStack Grizzly Architecture.* 2013. http://www.solinea.com/2013/06/15/openstack-grizzly-architecture-revisited/.

RDO, Redhat. *Quickstart - RDO.* 2013. http://openstack.redhat.com/Quickstart.