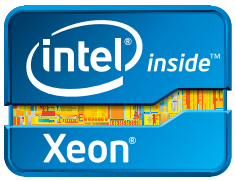
Intel® Cloud Builders Guide to Cloud Design and Deployment on Intel® Platforms

Content Cloud Storage with Red Hat Storage



Intel® Xeon® Processor E5-2600 Series

RED HAT® STORAGE

Tushar Gohad
Senior Staff Engineer Storage Division
Intel Corporation

Veda Shankar
Principal Technical Marketing Manager
Red Hat Storage, Red Hat Inc

Audience and Purpose

For companies looking to build their own cloud computing infrastructure, including both enterprise IT organizations and cloud service providers or cloud hosting providers, the decision to use a cloud for the delivery of IT services is best done by starting with the knowledge and experience gained from previous work. This white paper gathers into one place the essentials of a scale-out storage cloud reference architecture coupled with an example installation of GlusterFS, the foundational component of Red Hat Storage (RHS), optimized for the Content Cloud Storage (Large File/Object Store) use case. This workload is characterized by a large number of concurrent users accessing storage with a write-once, read-often access pattern. The reference architecture, based on Intel® Xeon® processor-based servers, creates a multi-node, capacity-optimized cloud storage deployment. The white paper contains details on the cloud topology, hardware, and software-deployed installation and configuration steps, and tests for real-world use cases that should significantly reduce the learning curve for building and operating your first cloud infrastructure.

It should be noted that the creation and operation of a storage cloud requires significant integration and customization to be based on existing IT infrastructure and business requirements. As a result, it is not expected that this paper can be used "as-is." For example, adapting to an existing network and identifying management requirements are out of scope for this paper. Therefore, it is expected that the user of this paper will make significant adjustments to the design presented in order to meet their specific requirements.

This paper also assumes that the reader has basic knowledge of cloud storage infrastructure components and services.

June 2013

Table of Contents

Executive Summary

To meet escalating capacity requirements in the cloud, providers are turning to a "scale-out" storage infrastructure. Red Hat Storage based on the GlusterFS technology combines industry-standard servers and standard storage components. The GlusterFS architectural model — basing cloud storage on a set of converged storage servers — allows providers to scale resources to boost capacity, bandwidth, and I/O performance as required with minimal time to deployment and a lower total cost of ownership (TCO). Illustrated in this paper is scale-out storage reference architecture to demonstrate a "content cloud storage" deployment with Red Hat Storage built on top of Intel® Xeon® processor family.

Introduction

Unstructured data is the fastest growing type of data in the enterprise today, and rich media content has continued to take on a larger piece of this data pie. Content clouds are emerging to provide a range of IT and consumer services for rich media consumption and distribution, requiring unprecedented amounts of data to be continually stored and managed. To meet this growing demand cost-effectively, companies have turned to standard Intel Xeon processor-based server platforms. Red Hat and Intel have created such a content cloud storage reference design, bringing together Intel Xeon processor-based servers and Red Hat's software-defined storage platform to speed organizations' ability to rapidly implement content clouds. The goals for this reference architecture are: the ability to scale out in a modular, incremental manner based on evolving demands, being optimized for storage and management of unstructured data, and being architected for cost-effectiveness based on Linux and Intel technology without the need for custom hardware.

Red Hat Storage Product Brief

Red Hat Storage (RHS) is a software-only, open source scale-out solution, designed to meet unstructured, semi- structured, and big data storage requirements. At the heart of Red Hat Storage is GlusterFS, an open source, massively scalable distributed file system that allows organizations to combine large numbers of storage and compute resources into a high-performance, virtualized, and centrally-man- aged storage pool. With Red Hat Enterprise Storage, enterprises can turn Intel Xeon processor-based servers and storage resources (either on-premise or in the public cloud) into a scale on-demand, virtualized, and centrally-managed storage pool. The global namespace capability aggregates disk, CPU, I/O, and memory into a single pool of resources with flexible back-end disk options, supporting direct attached, JBOD, or SAN storage. Storage server nodes can be added or removed without disruption to service – enabling storage to grow or shrink on-the-fly in the most dynamic environments.

Salient features of Red Hat Storage include:

POSIX-like Distributed File System

Red Hat Storage is based on GlusterFS, which is a powerful distributed filesystem written in user space which uses FUSE to hook itself with Linux VFS layer (on top of standard Linux filesystems such as XFS). GlusterFS is POSIX compliant.

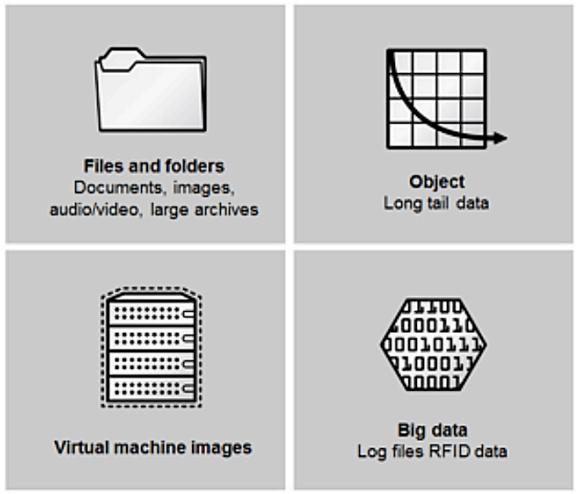


Figure 1: What can be stored in RHS?

Global Namespace

The global namespace capability aggregates disk, CPU, I/O, and memory from multiple servers into a single pool of resources with flexible back-end disk options, supporting direct attached, JBOD, or SAN storage. Storage server nodes can be added or removed without disruption to service – enabling storage to grow or shrink on-the-fly in the most dynamic environments.

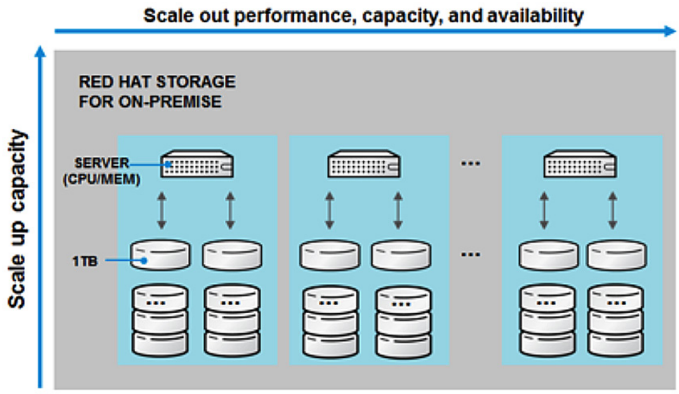


Figure 2: RHS Global Namespace, Linear Scalability

Elastic, Deployment-Agnostic, Runs on Standard Hardware

Red Hat Storage works on industry-standard Intel hardware, no purpose-built hardware needed. It allows IT admins to add/remove storage resources on-demand, without application disruption.

Scale-Out, No Metadata Server

Red Hat Storage eliminates the need for a separate metadata server and locates file algorithmically using an elastic hash algorithm. This no-metadata-server architecture ensures that there is no central point of failure improves performance and provides linear scalability.

High Availability: Replication (and Geo-Replication)

The replication feature in Red Hat Storage provides active-active mirroring within a data center while ensuring high-availability and fault-tolerance in the event of server/OS crashes and networking failures. This synchronous replication feature also supports automatic self-healing with the latest version of the data when the failed servers are recovered or restored. Geo-replication enables active-passive mirroring of data over wide-area networks. It provides the means to ensure global fault-tolerance and business continuity in the face of natural disasters and other data center-wide failures.

Predictable Performance across a Variety of Workloads

As a scale-out, software only system, Red Hat Storage was designed to perform across a wide variety of workloads, and was designed to enable customers to economically achieve very good performance levels under less than ideal conditions. It provides enterprises the ability to easily adjust configurations to achieve the optimal balance between performance, cost, manageability, and availability for their particular needs.

Red Hat Storage uses advanced performance enhancing techniques to provide a predictable performance. Some of the techniques include: write behind, asynchronous I/O, intelligent I/O scheduling, and aggregated operations.

Unified File and Object Access

RHS supports FUSE-based native client access to GlusterFS volumes. Native client is the recommended method for accessing GlusterFS volumes when high concurrency and high write performance is required.

RHS also supports CIFS and NFS protocols for providing standards-based access methods to its scaled-out volumes.

Via the Unified File and Object (UFO) interface, RHS supports object access using OpenStack Swift API. It is also the first product that enables unified access to objects as files through standard NFS and CIFS mounts as well as access to files created on GlusterFS volumes as objects with the OpenStack Swift protocol.

HDFS Compatibility

HDFS support in Red Hat Storage provides stronger protection against failure of data and name-nodes in a Hadoop deployment. Large and long-running data processing benefits from the combination of GlusterFS' high-availability and fault-tolerance.

Red Hat Storage – GlusterFS for the Enterprise

Red Hat Storage is an enterprise implementation of GlusterFS that integrates hardened and tested GlusterFS distribution and industry-standard Red Hat Enterprise Linux in an easy-to-consume distribution for rapid, seamless deployment of on-premise, public or hybrid cloud storage. Red Hat Storage uses TCP and XFS by default. In addition, Hadoop/UFO support is built-in and validated.

Red Hat Storage – Technical Overview

Red Hat Storage based on GlusterFS technology is a distributed filesystem built on top of XFS as its local file system. GlusterFS takes a layered approach to the file system, where features are added / removed as per the requirement. This modular architecture allows users to create multiple volumes with different characteristics like distribute/replicate with differing performance requirements. Though GlusterFS is a File System, it uses already tried and tested disk file systems like ext3, ext4, xfs, etc. to store the data.
Here are few concepts at the core of GlusterFS:

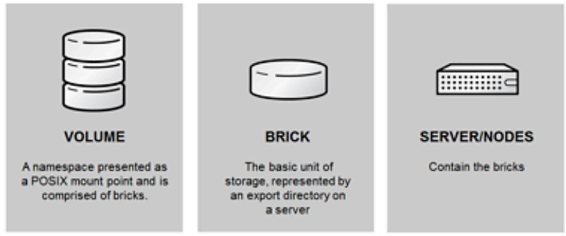


Figure 3: GlusterFS Concepts

GlusterFS Brick: "Brick" is the storage filesystem that has been assigned to a GlusterFS volume. A brick is GlusterFS' basic unit of storage, represented by an export directory on a server in a trusted storage pool. A Brick is expressed by combining a server with an export directory in the following format: **`SERVER:EXPORT`**.

For example: `<hostname>:/rhs/brick1/`.

Red Hat Storage Client: A node that mounts a GlusterFS volume (this may also be a server).

Red Hat Storage Server: A node (virtual or bare metal) that hosts the real filesystem (XFS, for example), where data will be stored. **glusterd**, the GlusterFS management daemon, runs on GlusterFS servers.

GlusterFS Translator: A translator connects to one or more sub-volumes, does something with them, and offers a sub-volume connection. An example of a GlusterFS translator is "replicate" which applies replication to bricks.

GlusterFS Sub-volume: A brick after being processed by at least one GlusterFS translator.

GlusterFS Volume: The final GlusterFS "share" after it passes through all the translators configured. The other view of a "Volume" is a logical collection of GlusterFS bricks.

Red Hat Storage – Possible Configurations

Distributed Volumes (Distribute)

Distribute is a translator that takes a list of bricks and spreads files evenly across them, effectively making one single larger storage volume from a series of smaller ones. The server that the files are written to is calculated by hashing the filename. If the filename changes, a pointer file is written to the server that the new hash code would point to, telling the distribute translator on which server the file is actually stored.

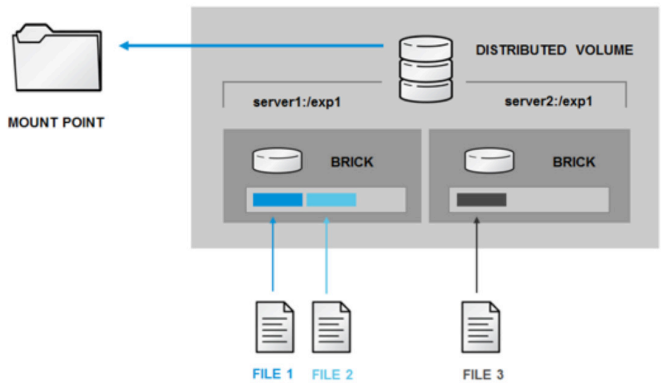


Figure 4: GlusterFS Distribute (File-level RAID0)

Advantages: A) The more servers you add, the better this scales in terms of random file access. As long as clients aren't all retrieving the same file, their access should be spread pretty evenly across all the servers. B) Increasing volume can be done by adding a new server. c) Adding servers can be done on-the-fly.

Disadvantages: A) If you lose a single server, you lose access to all the files that are hosted on that server. This is why distribute is typically graphed to the replicate translator. B) If your file is larger than the sub-volume, writing your file will fail. C) If the file is not where the hash code calculates to, an extra lookup operation must be performed, adding slightly to latency.

Replicated Volumes (Replicate)

Replicate is a translator that takes a list of bricks and replicates files across them. This is analogous to a file-level RAID-1 volume.

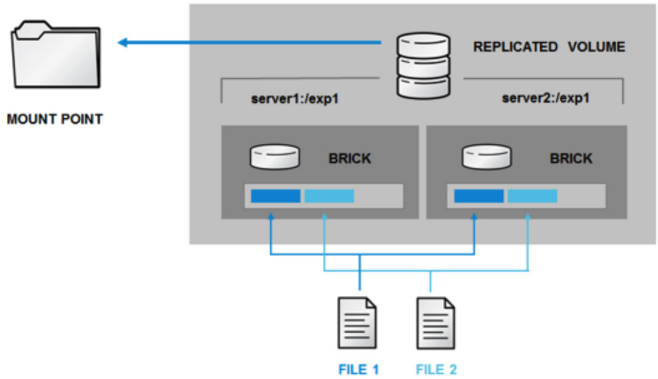


Figure 5: GlusterFS Replicate (File-level RAID1)

Distributed-Replicated Volumes

Distribute-Replicate, as the name suggests, is a translator that takes a list of bricks and replicates files across them, providing file distribution within the replicated volume at the same time. This is analogous to a file-level RAID-1+0 volume.

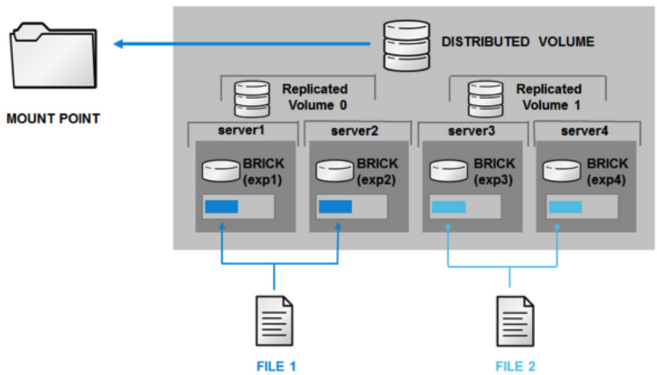


Figure 6: GlusterFS Distributed-Replicate (File level RAID 1+0)

Red Hat Storage – Use Cases

Red Hat Storage has broad use cases in scale-out cloud environment. Standardized NAS and object storage server deployments, both on-premise and in the Amazon Web Services (AWS) public cloud are possible. Common solutions based on RHS include Content Delivery Network (CDN), Backup/Disaster Recovery (DR), High Performance Computing (HPC), and Infrastructure as a Service (IaaS).

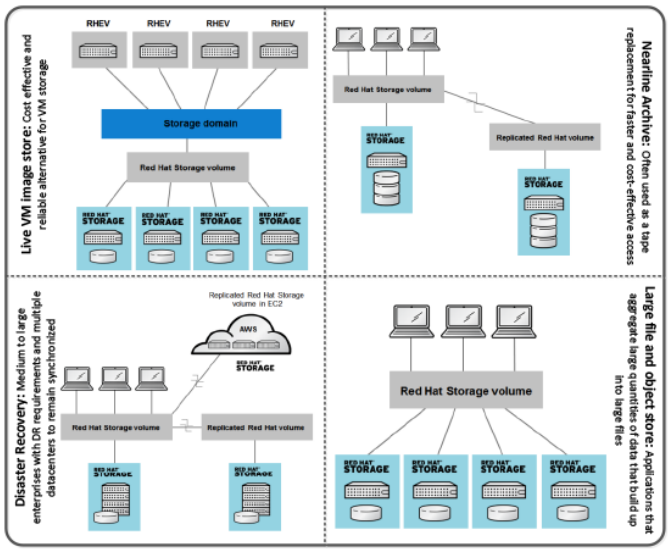


Figure 7: RHS Use Cases - Summary

Content Cloud/Large File and Object Store

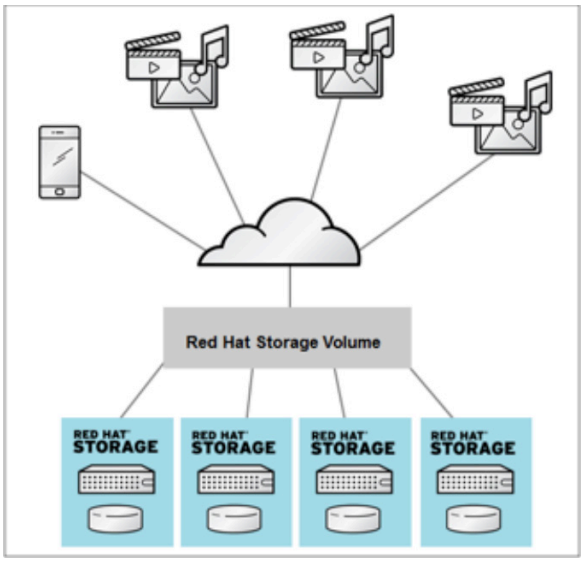


Figure 8: Content cloud: Massive simultaneous consumption of multimedia content by thousands of users

In a number of enterprises, applications generate a vast amount of data either in form files and/or objects that need to be stored for analysis and synthesis at a later time. The unstructured data piles up at an exponential rate, enterprises continue to react by adding more file servers or expensive monolithic NAS devices which over time becomes hard to manage & control and creates a file server sprawl. This in many cases results in underutilization of pricey NAS devices or lower performance because of over utilization of those file servers and NAS devices.

This is common in industries like oil/gas-exploration, Telco (data/app logs, billing), healthcare (x-ray, patient records), finance (risk-analysis, compliance and security data), scientific (gene sequencing, simulations), and miscellaneous applications such as rendering, geographical information systems, and computer aided engineering.

Workload: Several concurrent users, Read-mostly, Write-sparingly.

Average file size: Small to large, a few kilobytes to several megabytes. Total size of the data-store: Start small (a few terabytes) but will often grow to several hundred terabytes and achieve steady state.

The RHS advantage:

- Elastic scalability
- Eliminate storage silos, centralize & simplify management with global namespace technology
- Leverage volume economics & maximize.
- Multi-protocol client support for file sharing

Sizing: Recommend dual-socket Intel® Xeon® E5-2600 series processors with 64GB of DRAM for storage node system memory. Use 3TB SAS 10k rpm or 15k rpm drives for storage.

Enterprise Drop Box or Cloud Storage for Service Providers

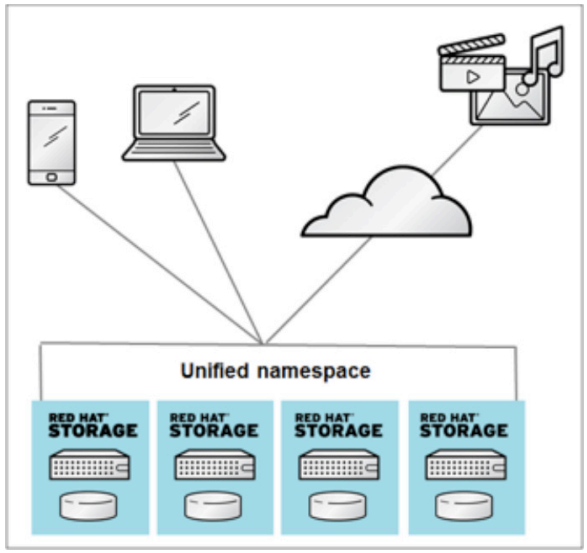


Figure 9: Enterprise drop box/ Cloud storage for SPs: SPs provide a share to end users to store content accessible from a variety of devices

Service Providers (SPs) including telcos, cable companies and other content providers are increasingly faced with competitive needs to provide differentiated services to their end customers. As a consequence service providers offer their end customers

access to free and/or paid storage for storing photos, videos and other media files that they access via the web using a smart phone, tablet or personal computer. Building a reliable back-end storage infrastructure with traditional monolithic storage gear does not meet the agility, cost-point and versatility demanded by such storage deployments. A similar use case is seen in the context of an enterprise where employees need to reliably and securely store and retrieve files from a variety of devices. Industries include Telecom and media verticals. The Enterprise Dropbox requirement applies to all verticals.

The workload: Can be characterized as initially comprising of only writes as end-users upload media files. Over time expect 50-50 reads and writes as end-users tend to retrieve and play those media files.

File sizes: A few megabytes (pictures taken using smart phones) to dozens of megabytes (HD videos taken using high end digital SLR cameras).

The total size of the data store: Directly proportional to the number of users this service is offered to. They typically start at hundreds of terabytes as there is a need to pre-provision for potentially millions of subscribers/customers. Such data stores can grow up to several petabytes over time where the data can span multiple data centers.

Data access pattern Hundreds of concurrent users reading and writing at the same time. The storage solution must be able to deliver sustained high throughput.

The RHS advantage:

- Scale-out architecture provides massive and linear scalability - capacity and performance can scale independently and linearly with no bottlenecks.
- Object storage support via OpenStack SWIFT API.
- Support for multiple Dropbox software licenses or seats that can be co-resident on the storage servers.
- Storage and retrieval of files and objects interchangeably.
- Consistent, sustained, predictable throughput performance.

Sizing: Recommend dual-socket Intel® Xeon® E5-2600 series processors with 64GB of DRAM for storage node system memory.

Red Hat Storage – Reference Architecture for Content Cloud Usages

One of the goals of this paper is to define a reference configuration for a Red Hat Storage cluster that is optimized on Intel architecture for the Content Cloud/large file/object store use case. Towards this goal, we experimented with RHS in an elaborate test setup that led to a reference architecture.

RHS Server Node – Reference Configuration

Figure 11 (RHS Reference Architecture) illustrates the test bed setup that we used for experiments/performance measurements. As shown, we deployed RHS in a 4x Storage Node cluster. This RHS "Storage Server" portion of the test bed is detailed in Figure 10 (RHS Server Node - Reference Configuration) – see nodes rhs01, rhs02, rhs03, and rhs04. A high-density (2U-4node) Intel® H2300JF system provided us with an optimal form factor for RHS server setup. Each storage node was equipped with Intel® Xeon® Processor E5-2680 with 8 CPU cores running at 2.7GHz, 64GB DDR3-1600 ECC-protected memory, dual-ported Intel® X520-T2 10GbE Ethernet adapter and an Intel® RS25SB008 SAS 6Gbps RAID controller (hardware RAID support with battery-backup). Each RHS server node was connected to an external storage array with twelve- 3TB Seagate* SAS 10k hard drives configured in RAID6 mode. This provides a total of 48 * 3TB = 144TB of storage in a 4-node RHS system. This combined with the 2U form factor makes this configuration optimal from a storage-capacity-per-rack perspective.

Compute Density – Upto 8 Intel® Xeon® Processor E5-2600 per 2U system
via 4 hot-swappable half-width nodes
(Available as
Intel® H2300JF
Supermicro 2UTwin2
Dell PowerEdge C6220)

CPU – 2x Intel® Xeon® Processor E5-2680 8cores@2.7GHz, 20MB, 130W
Memory – 64GB DDR3-1600 ECC
Motherboard – S2600JFQ

Network – Intel® X520-T2 Converged Ethernet
2x 10GbE copper ports, bonded on the switch with 802.3ad

OS Disk – 2x 600GB 2.5" SAS 10k 64MB Seagate ST960020588 (RAID0)

Storage Controller – Intel® RS25SB008 SAS RAID Controller Card, PCIe x16
battery-backed

Storage Array – via 2x external JBODs (4U-60 NDS-4600)

RHS Storage Node Reference Configuration
Intel® H2300JF System (4 dual-CPU servers in a 2U form factor)

Figure 10: Reference Hardware Configuration for RHS Server Nodes (rhs01, rhs02, rhs03, rhs04 in Figure 11)

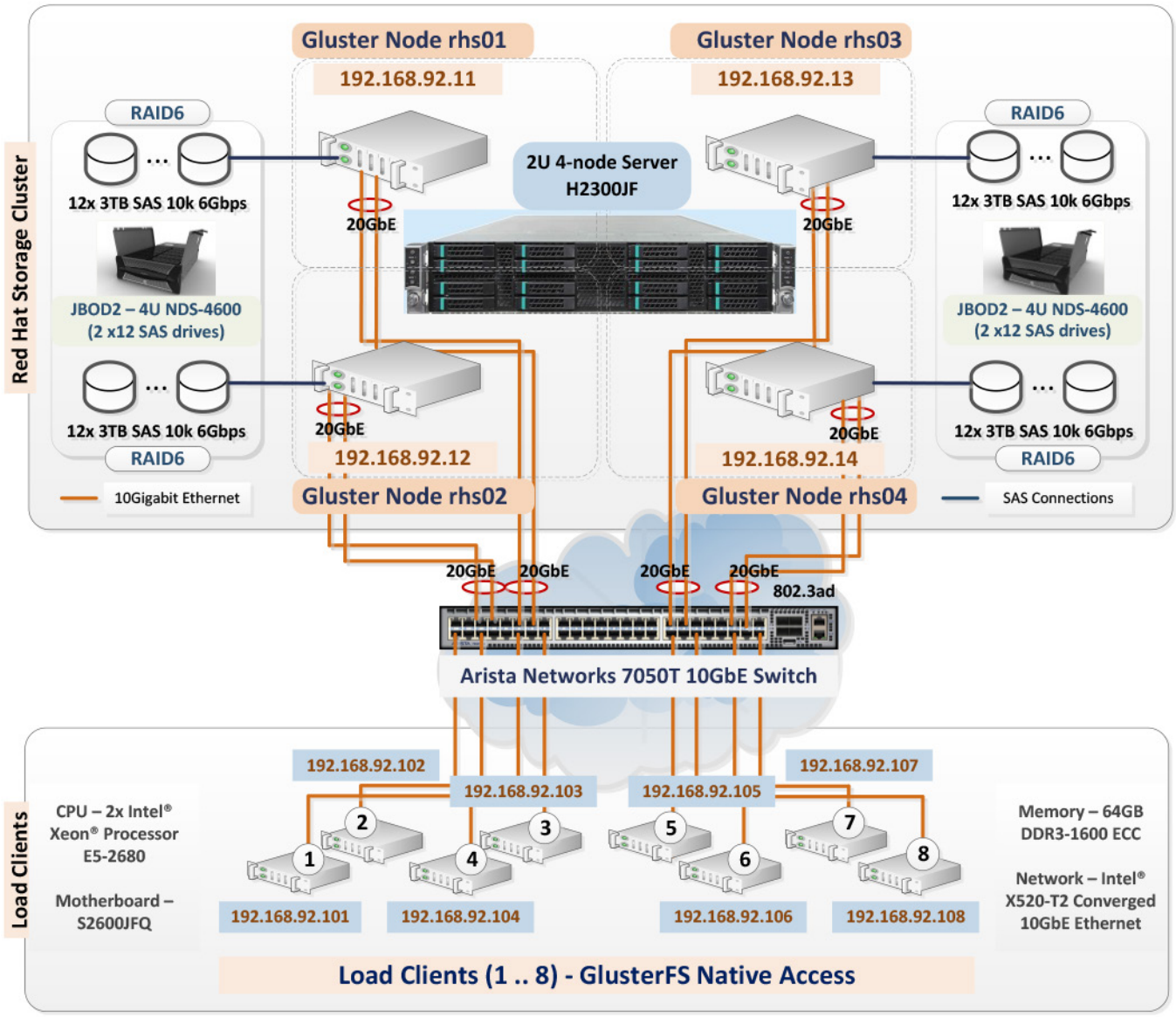


Figure 11: RHS Reference Architecture (Content Cloud)

Storage Array – Reference Configuration

We used the Newisys/Sanmina* NDS-4600 4U, 60-SAS-hard-drive JBOD as the external storage array to host the twelve- 3TB SAS drives per RHS server node. The JBOD drive array was partitioned such that only 12 SAS drives are presented to each RHS server node (this is known as "Zoning Configuration 3" in the NDS-4600 JBOD terminology. Please refer to JBOD manual for further details on how to configure this mode).

RHS Client – Test Configuration

The test setup included up to 8 RHS load generator clients (shown in Figure 11). Each client was configured for native GlusterFS access. Hardware configuration for each client was Intel Xeon Processor E5-2680 with 8 CPU cores running at 2.7GHz, 64GB DDR3-1600 ECC-protected memory, and an Intel® X520-T2 10GbE Ethernet adapter.

Test Network Configuration – 10GbE, LACP, Jumbo Frames

The RHS cluster and the test clients were networked using an Arista Networks* 7050T fully non-blocking, manageable 10GbE Ethernet switch. The switch ports were configured so that IP packets destined to test network 192.168.92.0/24 are addressed to the same VLAN. Network switch ports were configured to support jumbo frames (MTU = 9216). In addition, each of the two network switch ports connected to the RHS server nodes rhs01, rhs02, rhs03 and rhs04 were bonded in the "dynamic link aggregation" configuration with LACP (link-aggregation control protocol as described in IEEE 802.3ad standard). A tutorial on Arista switch configuration for this mode is illustrated at: https://eos.aristanetworks.com/2011/05/link-aggregation-primer. One of the important things to note here is that the aggregated-link (also called "channel-group" should be assigned the same VLAN ID as one assigned to the test subnet, 192.168.92.0/24 in the test setup described here.

Red Hat Storage Setup Cookbook

This section describes the steps involved in hardware configuration through deploying Red Hat Storage software on storage server nodes. This "cookbook" style tutorial should help the reader recreate the reference architecture/test setup shown in Figure 11: RHS Reference Architecture (Content Cloud).

Red Hat Storage Node – BIOS Configuration

On each RHS server node, make sure you have Intel® Turbo Boost and Enhanced Intel SpeedStep® Technology enabled. Intel SpeedStep will make sure your RHS servers do not run at maximum CPU frequency and thus run hot all the time.

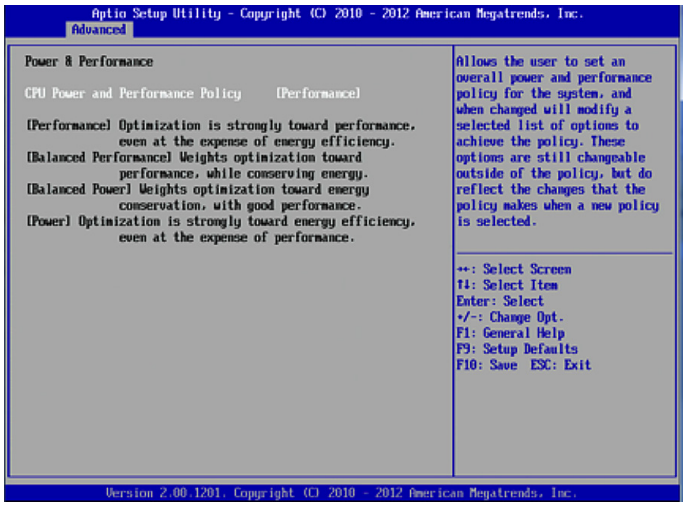


Figure 12: CPU Power and Performance Policy Chosen ("Performance")

Next, enter "Advanced" BIOS menu and choose CPU power and performance policy to be optimized for "Performance.

Red Hat Storage Node – RHS2.x install

RHS (server) nodes (rhs01, rhs02, rhs03, and rhs04) were installed with Red Hat Storage 2.0-update4. This RHS release provides a GlusterFS 3.x implementation that is validated and hardened in the Red Hat Enterprise Linux* 6.x environment. Red Hat Storage comes preinstalled with GlusterFS server software, client daemons, CIFS/NFS/UFO services, and Linux kernel tuning daemon (tuned, tuned-adm). Tuned provides predefined profiles for optimizing the system performance for the following workloads: "throughput-performance", "enterprise-storage" and "latency-storage."

OS Disk Creation (RAID1)

In our reference test environment, we used mirrored (RAID1) volume for OS install (RHS 2.0-update 4). The OS volume was created using the Intel® Rapid Storage Technology (Intel® RST) BIOS shown in Figure 13.

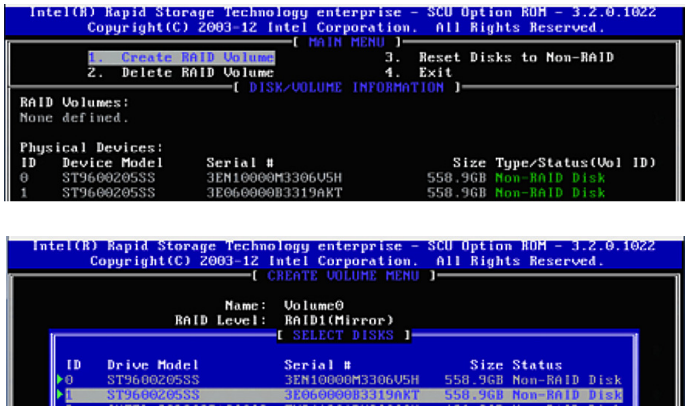
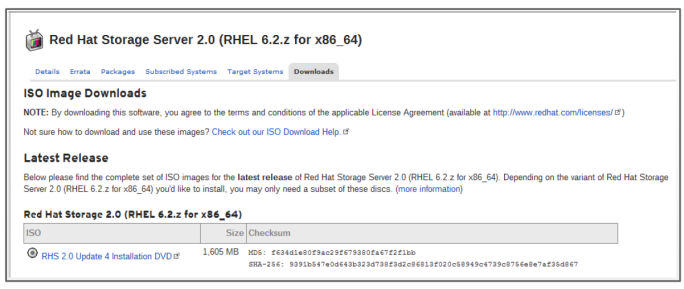


Figure 13: RAID1 Volume Creation for RHS 2.0 install

In the next step, we install Red Hat Storage using the ISO image "RHS 2.x Update y Installation DVD." You can download the ISO using your Red Hat Network account.



Connect RHS nodes to a network that can provide the node Internet connectivity. Boot the node(s) off of RHS install DVD and install Red Hat Storage with defaults (recommended).

Once RHS install is complete, use "rhn-register" utility to register the RHS node on Red Hat Network (RHN).

```
rhs01# rhn-register
```

Subscribe to the RHN channels "RHS 2.x" and "RHEL Server SFS 6.x.z", in order to receive updates.

```
rhs01# rhn-channel --add \
--channel=rhel-x86_64-server-6-rhs-2.0

rhs01# rhn-channel --add \
--channel=rhel-x86_64-server-sfs-6.2.z
```

Update the RHS node using:

```
rhs01# yum update
```

"rhs01" is used in all examples in the remainder of this document. Perform the same actions on rhs02, rhs03 and rhs04.

Red Hat Storage Node – Network Configuration

Once RHS is installed, the next step is to connect the node(s) to the 10GbE network we configured in section "Test Network Configuration – 10GbE, LACP, Jumbo Frames." It is important that the node(s) are installed with 10Gb Ethernet adapter (or adapters) providing at least two 10GbE ports for connectivity. Configure the two 10GbE interfaces, say "eth0" and "eth1" in a mode 4 (802.3ad, LACP) bonding configuration in order to match the 10GbE switch port configuration. Here is an example configuration from "rhs01":

```
[root@rhs01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BOOTPROTO=static
IPADDR=192.168.92.11
NETMASK=255.255.255.0
GATEWAY=192.168.92.1
ONBOOT=yes
NM_CONTROLLED=no
BONDING_OPTS="miimon=100 mode=4 lacp_rate=0 xmit_hash_policy=2"
MTU=9000
[root@rhs01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
MASTER=bond0
SLAVE=yes
MTU=9000
[root@rhs01 ~]# _
```

Note the parameter:

```
BONDING_OPTS="<snip> mode=4 <snip> xmit hash_policy=2"
```

This setting tells the Linux bonding driver to configure the "bond0" interface in "802.3ad" LACP mode, which is mode=4 in Linux terms. The parameter "xmit_hash_policy=2" configures the bonding driver to use a hashing function on the network Layer 2 and Layer 3 packet header fields in order to achieve better load balancing across the slave 10GbE interfaces that are part of "bond0." Also recommended is the setting:

```
MTU=9000
```

This is to enable "Jumbo Frames". Please note that, if jumbo frames are enabled on the server nodes, the same MTU should be configured on all nodes (client and any other test nodes) for best results.

Once the interfaces "eth0", "eth1" and "bond0" are configured, restart RHS network service.

Verify bonding settings with the commands shown here:

```
[root@rhs01 ~]# cat /sys/class/net/bond0/bonding/mode
802.3ad 4
[root@rhs01 ~]# cat /sys/class/net/bond0/bonding/xmit_hash_policy
layer2+3 2
[root@rhs01 ~]# cat /sys/class/net/bond0/bonding/slaves
eth0 eth1
[root@rhs01 ~]# cat /sys/class/net/bond0/bonding/lacp_rate
slow 0
```

Red Hat Storage Node – Storage Configuration (RAID6)

As described in section "RHS Server Node – Reference Configuration," an external JBOD (NDS-4600) provides the storage backend to the RHS server nodes. The nodes are connected via an Intel® RS25SB008 SAS 6Gbps battery-backed RAID controller to the 12-disk JBOD disk array. The recommended RAID configuration for the disk array is hardware-assisted RAID6 with battery-backed write-back cache. Here are instructions for recreating the setup.

Use the "CmdTool264" utility (called "megacli" in case of LSI branded RAID adapters) to configure RAID6 on each of the RHS server nodes (example configuration for one RHS node is shown here).

Determine JBOD enclosure ID and the slot numbers that have disks installed:

```
rhs01# CmdTool264 -LdPdInfo -a0 \
      | grep '^Enclosure Device ID:'
rhs01# CmdTool264 -LdPdInfo -a0 \
      | grep '^Slot Number:'
```

Use the JBOD enclosure ID and the appropriate disk slot numbers reported by the previous set of commands to create a RAID6 volume:

```
rhs01# CmdTool264 -CfgLdAdd \
       -r6 [10:4, 10:5, 10:6, 10:16, 10:17, 10:18,
       10:28, 10:29, 10:30, 10:40, 10:41, 10:42] \
       -a0 -strpsz 256
```

`-r6` specifies RAID6 creation, 10 is the enclosure ID and `4, 5, 6… 41, 42` are the `12` disk slot numbers in the JBOD that we want to be RAID6 members. The -strpsz parameter specifies the RAID stripe size.

The recommended RAID6 strip size is 256KB.

Configure RAID6 parameters:

Set caching policy to "cached".

```
rhs01# CmdTool264 -LDSetProp Cached -LALL -a0
```

Set disk cache policy to "enabled".

```
rhs01# CmdTool264 -LDSetProp EnDskCache -LALL -a0
```

Enable write policy to "WriteBack".

```
rhs01# CmdTool264 -LDSetProp WB -LALL -aALL
```

The "WriteBack" policy enables asynchronous writes and thus improves performance. For this mode to work reliably, battery-backup for Write-back cache is essential.

For other RAID controller configuration commands, please refer to the LSI/Intel MegaRAID adapter user manual.

To watch progress of RAID6 "virtual drive" creation:

```
rhs01# CmdTool264 - AdpAllLog -aALL
```

To confirm RAID6 "virtual drive" creation:

```
rhs01# CmdTool264 -LDInfo -LALL -aALL
```

Here is the "-LDInfo" output from our setup:

```
Adapter 0 -- Virtual Drive Information:
Virtual Drive: 0 (Target Id: 0)
Name                :
RAID Level          : Primary-6, Secondary-0, RAID Level Qualifier-3
Size                : 27.284 TB
Parity Size         : 5.456 TB
State               : Optimal
Strip Size          : 256 KB
Number Of Drives    : 12
Span Depth          : 1
Default Cache Policy: WriteBack, ReadAdaptive, Cached, Write Cache OK if Bad BBU
Current Cache Policy: WriteBack, ReadAdaptive, Cached, Write Cache OK if Bad BBU
Default Access Policy: Read/Write
Current Access Policy: Read/Write
Disk Cache Policy   : Enabled
Encryption Type     : None
PI type: No PI

Is VD Cached: No
```

Red Hat Storage Node – Brick Creation

A "brick" is the unit of storage in RHS. One or more bricks make an RHS "volume". The brick creation steps provided in this section are for the reader's reference only and typically, a script, rhs_system_init.sh, is used to create bricks automagically, and configure the RHS system for optimal performance.

In order to create a brick, first identify your RAID6 volume using the "lvmdiskscan" utility, say "`/dev/sdX`".

As the next step, we use Logical Volume Manager (LVM) to create a physical volume (PV). (Use of LVM is recommended, however if you do not wish to use LVM to create logical volumes on your RAID device, you may skip to the last step in this section, which is formatting the device with XFS filesystem).

Create PV:

```
rhs01# pvcreate --dataalignment 2560k /dev/sdX
```

Create a volume group named "RHS_vg1":

```
rhs01# vgcreate RHS_vg1 /dev/sdX
```

Verify volume group creation with:

```
rhs01# pvdisplay
```

Make certain your RAID device is shows up in pvdisplay output.

```
rhs01# vgdisplay
```

```
  --- Volume group ---
  VG Name               RHS_vg1
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  2
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               27.28 TiB
  PE Size               4.00 MiB
  Total PE              7152554
  Alloc PE / Size       6079670 / 23.19 TiB
  Free  PE / Size       1072884 / 4.09 TiB
  VG UUID               p0Gvx2-XowG-mdy3-Hwz4-016U-FFSs-2sPD1j
```

Create a logical volume named "RHS_lv1" under the volume group "RHS_vg1" and verify with "lvdisplay":

```
rhs01# lvcreate -l 85%FREE -n RHS_lv1 RHS_vg1

rhs01# lvdisplay
```

```
  --- Logical volume ---
  LV Path                /dev/RHS_vg1/RHS_lv1
  LV Name                RHS_lv1
  VG Name                RHS_vg1
  LV UUID                fKW03f-xNuJ-fAUW-pUNM-rcPF-90v7-0A069L
  LV Write Access        read/write
  LV Creation host, time rhs01, 2013-05-07 09:37:59 -0700
  LV Status              available
  # open                 1
  LV Size                23.19 TiB
  Current LE             6079670
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:3
```

Next, format the logical volume "RHS_lv1" with XFS filesystem:

```
rhs01# mkfs.xfs -I size=512 -n size=8192 \
       -d su=256k,sw=10 /dev/RHS _ vg1/RHS _ lv1
```

We'll use this XFS-formatted logical volume as one of the bricks.

Create a mount point for the brick and create an `/etc/fstab` entry:

```
/dev/RHS _ vg1/RHS _ lv1 /rhs/brick1 xfs \
       inode64,noatime,nodiratime 1 2
```

Mount the newly created filesystem:

```
rhs01# mount /rhs/brick1
```

The XFS volume mounted under "rhs01:/rhs/brick1" is our first brick that we can use to create an RHS volume (in the next section). Before we go there, we need to repeat this "brick creation" process on all RHS server nodes (rhs02, rhs03, rhs04).

On each of rhs01/02/03/04, start "glusterd" (restart if already running):

```
rhs01# service glusterd stop
rhs01# service glusterd start
```

Take this opportunity to also set up "tuned" on each of rhs01.. rhs04:

```
rhs01# tuned-adm profile rhs-high-throughput
```

To make this setting persist across reboots, you may want to use /etc/tuned.conf. Please see tuned.conf(5) and tuned-adm(1) man pages for more information.

Red Hat Storage Node – Volume Creation

Once we have created and mounted bricks on all RHS server nodes, we'll instantiate a few RHS volumes that are useful for recreating the test setup we describe here. We create "distribute" and "replicate" (mirrored) flavors of RHS volumes, specifically:

1. 2, 3, 4-node configurations of Distribute Volumes.
2. A 2-node Replicate Volume.
3. A 4-node Distribute-Replicate Volume

Before we do, make sure that you have "glusterd" daemon running on each of rhs01/02/03/04. If all is well, you should be able to query your gluster "peers":

```
rhs01# gluster peer status
```

and get something on the lines of what's shown below. You should see "Connected" for status on all the peers. If not, check your network firewall settings and make sure that TCP ports 111, 24007,24008, 24009-(24009 + number of bricks across all volumes) are open on all of rhs01/02/03/04. If you will be using NFS, also open ports 38465, 38466 and 38467.

```
[root@rhs01 ~]# gluster peer status
Number of Peers: 3

Hostname: rhs02
Uuid: 9ce1a894-71e3-4639-85e8-a3771f727a7f
State: Peer in Cluster (Connected)

Hostname: rhs03
Port: 24007
Uuid: e71e9136-f711-4967-9ab9-47e756ee6187
State: Peer in Cluster (Connected)

Hostname: rhs04
Port: 24007
Uuid: 817c575e-7301-4b0d-bdff-d0900a0ae6a7
State: Peer in Cluster (Connected)
```

Once all peers are responding, we can go ahead and create our volumes:

```
rhs01# gluster volume create <volume _ name> \
       <server1:brick>/<volume _ name> \
       <server2:brick>/<volume _ name> \
       <servern:brick>/<volume _ name> \
```

For example: to create a Distribute Volume named "distvol4" with bricks from all 4 of our RHS server nodes, do the following from any one of rhs01/02/03/04 nodes:

```
rhs01# gluster volume create distvol4 \
       rhs01:/rhs/brick1/distvol4 \
       rhs02:/rhs/brick1/distvol4 \
       rhs03:/rhs/brick1/distvol4 \
       rhs04:/rhs/brick1/distvol4
rhs01# gluster volume start distvol4
```

Verify volume creation with:

```
rhs01# gluster volume info distvol4
```

```
[root@rhs01 ~]# gluster volume info distvol4

Volume Name: distvol4
Type: Distribute
Volume ID: 9e7de726-55c8-4ddf-8a73-9c6cd2317d4e
Status: Started
Number of Bricks: 4
Transport-type: tcp
Bricks:
Brick1: rhs01:/rhs/brick1/distvol4
Brick2: rhs02:/rhs/brick1/distvol4
Brick3: rhs03:/rhs/brick1/distvol4
Brick4: rhs04:/rhs/brick1/distvol4
```

On similar lines, we created the following volumes for our test setup:

1. distvol2: only with rhs01 and rhs02 (2-node Distribute)
2. distvol3: with rhs01, rhs02 and rhs03 (3-node Distribute)

Next, we create a "Replicate" or a mirrored volume with 2 nodes participating:

```
rhs01# gluster volume create mirrorvol2 \
       replica 2 \
       rhs01:/rhs/brick1/mirrorvol2 \
       rhs02:/rhs/brick1/mirrorvol2

rhs01# gluster volume start mirrorvol2
```

Verify volume creation with:

```
[root@rhs01 ~]# gluster volume info mirrorvol2

Volume Name: mirrorvol2
Type: Replicate
Volume ID: 8a41e8ea-147a-4a25-bc55-77103e6bce30
Status: Started
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: rhs01:/rhs/brick1/mirrorvol2
Brick2: rhs02:/rhs/brick1/mirrorvol2
```

We create another Replicate volume with all 4-nodes, and instead of setting the number replicas to 4, we set the number to 2 so RHS divides the node-set into two – one subset being the mirror of the other, with files being distributed across nodes within a subset, essentially, creating a combo "Distributed-Replicate" volume, named, say "mirrorvol4":

```
rhs01# gluster volume create mirrorvol4 \
       replica 2 \
       rhs01:/rhs/brick1/mirrorvol4 \
       rhs02:/rhs/brick1/mirrorvol4 \
       rhs03:/rhs/brick1/mirrorvol4 \
       rhs04:/rhs/brick1/mirrorvol4

rhs01# gluster volume start mirrorvol4
```

```
[root@rhs01 ~]# gluster volume info mirrorvol4

Volume Name: mirrorvol4
Type: Distributed-Replicate
Volume ID: a3489e62-a853-4d0d-a864-ea4200821888
Status: Started
Number of Bricks: 2 x 2 = 4
Transport-type: tcp
Bricks:
Brick1: rhs01:/rhs/brick1/mirrorvol4
Brick2: rhs02:/rhs/brick1/mirrorvol4
Brick3: rhs03:/rhs/brick1/mirrorvol4
Brick4: rhs04:/rhs/brick1/mirrorvol4
```

Make sure all the volumes are started with "gluster volume info" on each volume. If all looks ok, we are ready to mount the volumes on RHS client.

RHS Native Client Configuration (Load Generators)

In test setup described in this paper, 8 RHS clients were used (rhs-client1 through rhs-client8 in Figure 11). Red Hat Enterprise Linux Server 6.4 was installed on all test clients. All clients were FUSE-based clients running in user space. Here are detailed setup instructions for the clients:

```
rhs-client1# rhn-register
```

Subscribe to the "Red Hat Storage 2.x Native Client" RHN channel

```
rhs-client1# rhn-channel --add \
    --channel=rhel-x86_64-server-rhsclient-6
```

Install Native RHS Client rpms:

```
rhs-client1# yum -y install glusterfs
glusterfs-fuse
```

Load FUSE module:

```
rhs-client1# modprobe fuse
```

Verify that FUSE module is loaded:

```
rhs-client1# dmesg | grep -i fuse
```

Key-based SSH Setup between Server/Client Nodes

Set up key-based authentication between all the RHS server and client nodes. Do the following on each node.

Generate SSH key pair:

```
rhs01#        ssh-keygen -t rsa
```

Copy the keys to other hosts:

```
rhs01# ssh-copy-id -i ~/.ssh/id_rsa.pub \
       root@rhs02 root@rhs03 root@rhs04 \
       root@rhs-client1 root@rhs-client2 \
       root@rhs-client3 root@rhs-client4 \
       root@rhs-client5 root@rhs-client6 \
       root@rhs-client7 root@rhs-client8
```

Repeat on RHS nodes rhs0{2..4} and clients rhs-client{1..8}.

Mount RHS Volumes on Client Nodes

Next mount all RHS volumes we created in section Red Hat Storage Node – Volume Creation, using "glusterfs" filesystem type (native RHS client access). For example, to mount distvol4 on rhs-client1:

```
rhs-client1# mkdir –p /mnt/fuse-distvol4

rhs-client1# mount –t glusterfs \
       rhs01:/distvol4 /mnt/fuse-distvol4
```

Repeat on all clients (rhs-client1..rhs-client8), for all volumes (distvol2, distvol3, distvol4, mirrorvol2, mirrorvol4).

Verify native RHS mounts on each client:

```
rhs-client1# mount | grep gluster
```

```
[root@rhs-client1 rhs_tools]#
[root@rhs-client1 rhs_tools]# mount | grep gluster
rhs01:/distvol2 on /mnt/fuse-distvol2 type fuse.glusterfs (rw,default_permission
s,allow_other,max_read=131072)
rhs01:/distvol3 on /mnt/fuse-distvol3 type fuse.glusterfs (rw,default_permission
s,allow_other,max_read=131072)
rhs01:/distvol4 on /mnt/fuse-distvol4 type fuse.glusterfs (rw,default_permission
s,allow_other,max_read=131072)
rhs01:/mirrorvol2 on /mnt/fuse-mirrorvol2 type fuse.glusterfs (rw,default_permis
sions,allow_other,max_read=131072)
rhs01:/mirrorvol4 on /mnt/fuse-mirrorvol4 type fuse.glusterfs (rw,default_permis
sions,allow_other,max_read=131072)
```

The clients are now ready to use the RHS cluster. Access to the cluster is provided via the glusterfs mount points `/mnt/fuse-*` shown above.

Performance and Scalability Testing with IOzone

We use IOzone to test the performance of our reference RHS cluster. To recap, we have 4 RHS server nodes, each is hosting a RAID6 based RHS brick and is running glusterd. We have created 5 RHS volumes – 3 Distribute (distvol2, distvol3, distvol4), 1 Replicate (mirrorvol2) and 1 Distributed-Replicate (mirrorvol4). We use IOzone on up to 8 RHS native clients, for IO performance / scalability testing.

IOzone Configuration

IOzone is run in "Cluster testing" mode to launch 4 IOzone tester threads on each client. With 8 clients, that gives us the ability to test with 8, 16, 24, or 32 threads performing IO to the RHS cluster concurrently.

The IOzone command line used is:

```
rhs-client1# iozone –+m \
       ${IOZONE_CONFIG_FILENAME} –i ${IOZONE_
       TEST} \
       –C –w –+n \
       –s ${IOZONE_FILESZ} \
       –r ${IOZONE_RECORDSZ} \
       –+z –c –e –t ${TEST_THREADS} –R \
       –b result.xls"
```

where, "-+m" specifies cluster testing mode, **IOZONE_CONFIG_FILENAME** is the IOzone config file for the cluster test format that lists the client hostnames and their respective glusterfs mounts (that we created in the previous section when mounting RHS volumes), **IOZONE_FILESZ is "8GB"** – we use 8GB file transfers as representative "workload" for our "Content Cloud" reference architecture testing. **IOZONE_RECORDSZ** is varied between 64KB and 4MB, considering record sizes that are powers of 2. This range of record sizes is meant to characterize the effect of record/block size (in client file requests) on IO performance.

The IOZONE_TEST parameter is varied to cover the "Sequential Read", "Sequential Write", and "Random Read/Write" test cases.

Each test outcome was averaged over 6 test runs to achieve a 90 percent confidence interval. Test results are discussed starting in the section "Test Results – Distributed Volumes."

Test Preconditions

Clear data, inodes and dentries on all RHS server nodes, before starting iozone tests in order to reduce caching effects:

```
rhs01 # sysctl –w vm.drop_caches=3
```

Test Results – Distributed Volumes

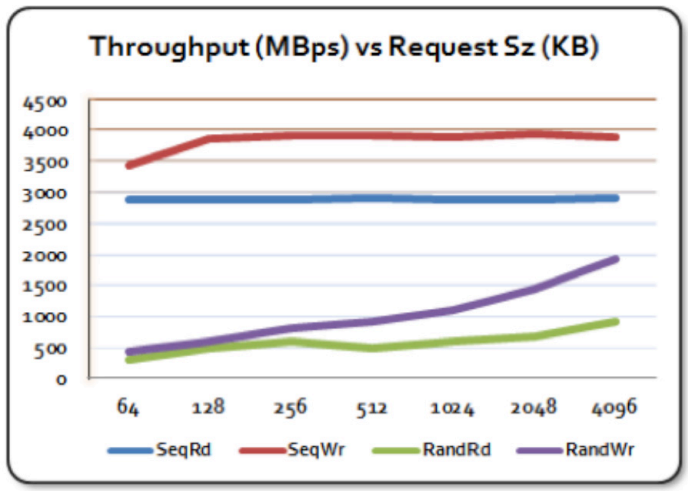
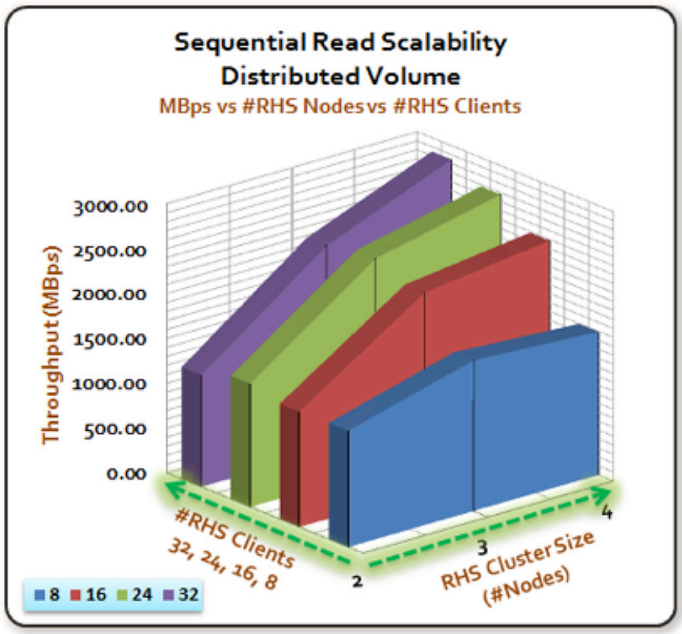


Figure 14: **RHS Distributed Volume, 4-nodes, 32 RHS clients** - SeqRd/Wr saturate at reqsz > 64KB, Random Rd/Wr Scale over all reasonable block sizes (up to 4MB and possibly beyond).

The chart in Figure 14 illustrates the effect of request size (kiloBytes) variation (IOzone "record size") on RHS Distributed volume Throughput (megaBytes per second). 32 IOzone client threads were used to measure throughput in this configuration.

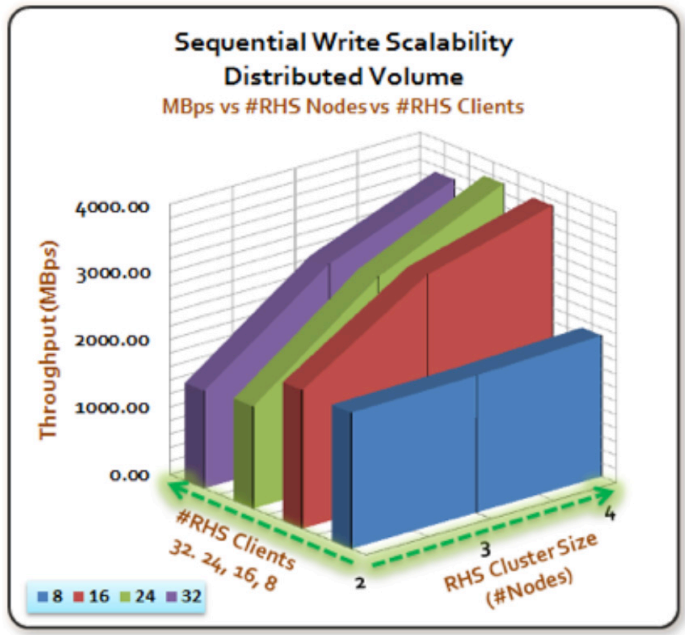
We used 1024KB as the representative record size (request size) for presenting the data gathered during this experiment.



Distributed Mode - Sequential Read Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2	3	4
8	1295.77	1679.08	1596.70
16	1284.66	2208.96	2388.97
24	1373.52	2364.80	2680.09
32	1248.14	2286.14	2848.23
iozone record size	1024KB		
iozone file/transfer size	8GB		

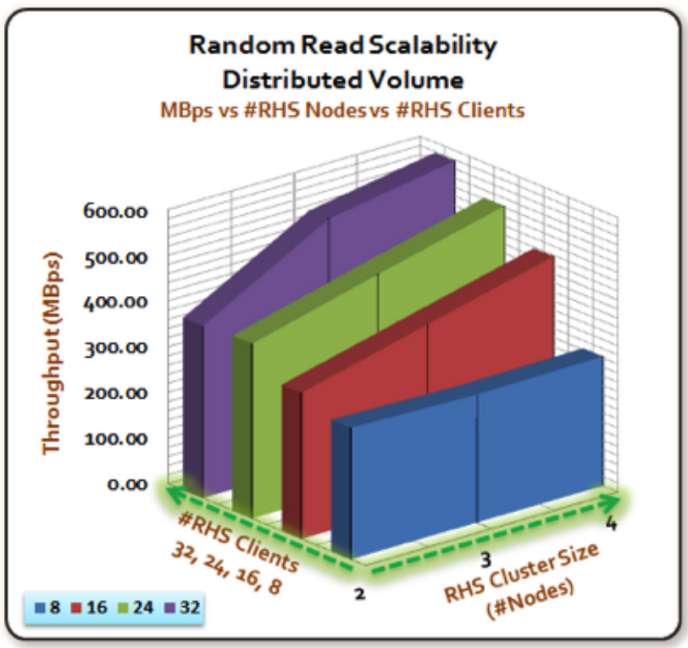
Figure 15: **SeqRd Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added



Distributed Mode - Sequential Write Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2	3	4
8	2015.73	2032.06	2087.64
16	2051.13	3225.03	3716.49
24	1511.17	2897.81	3743.87
32	1443.74	2794.47	3507.44
iozone record size	1024KB		
iozone file/transfer size	8GB		

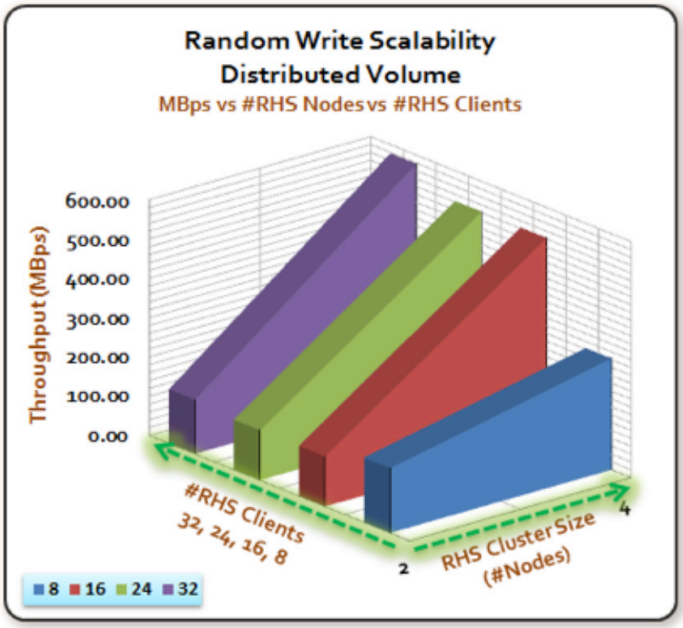
Figure 16: **SeqWr Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added



Distributed Mode - Random Read Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2	3	4
8	287.63	280.26	283.29
16	319.44	388.92	457.58
24	382.16	454.25	523.15
32	376.27	530.69	577.78
iozone record size	1024KB		
iozone file/transfer size	8GB		

Figure 17: **RandomRd Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added



Distributed Mode - Random Write Performance/Scalability			
	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2		4
8	164.73		283.29
16	152.83		457.58
24	127.34		523.15
32	137.69		577.78
iozone record size	1024KB		
iozone file/transfer size	8GB		

Figure 18: **RandomWr Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added

Test Results – Replicated Volumes

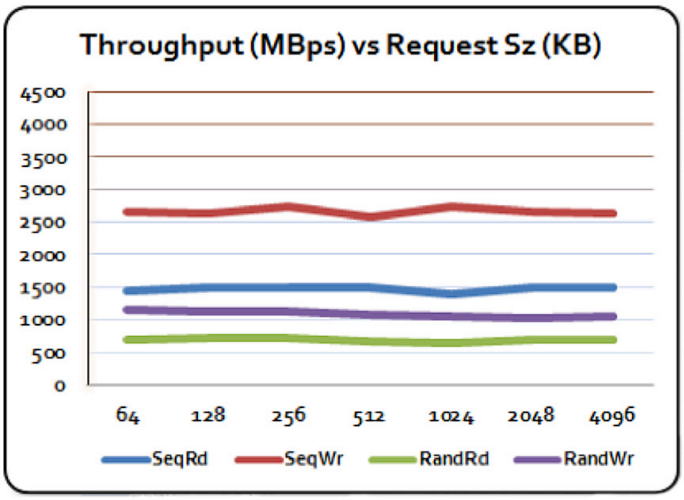


Figure 19: **RHS Replicated Volume**, 2-nodes, 32 RHS clients – Rd/Wr performance saturates at reqsz > 64KB

Chart in Figure 15 illustrates the effect of request size (kiloBytes) variation (IOzone "record size") on RHS Mirrored volume Throughput (megabytes per second). 32 IOzone client threads were used to measure throughput in this configuration.

Test results for this 2-node configuration along with the Distributed-Replicate 4-Node configuration are presented together starting on the next page.

Test Results – Distributed-Replicated Volumes

Chart in Figure 15 illustrates the effect of request size (kiloBytes) variation (IOzone "record size") on throughput (megaBytes per second) of RHS Distributed-Mirrored volume constructed with 4-RHS nodes. 32 IOzone client threads were used to measure throughput in this configuration.

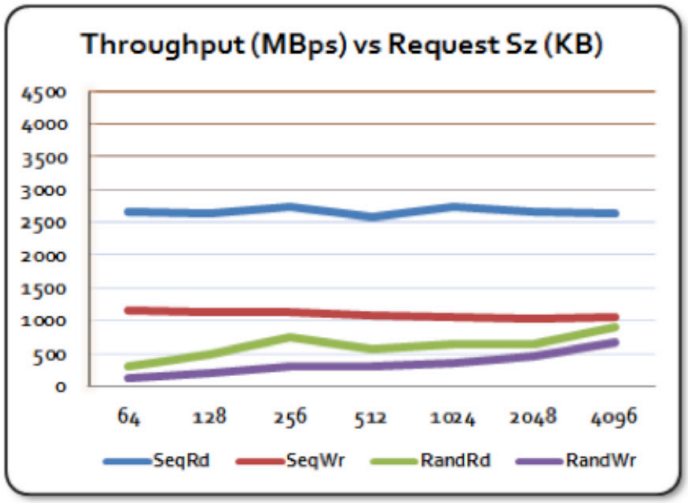
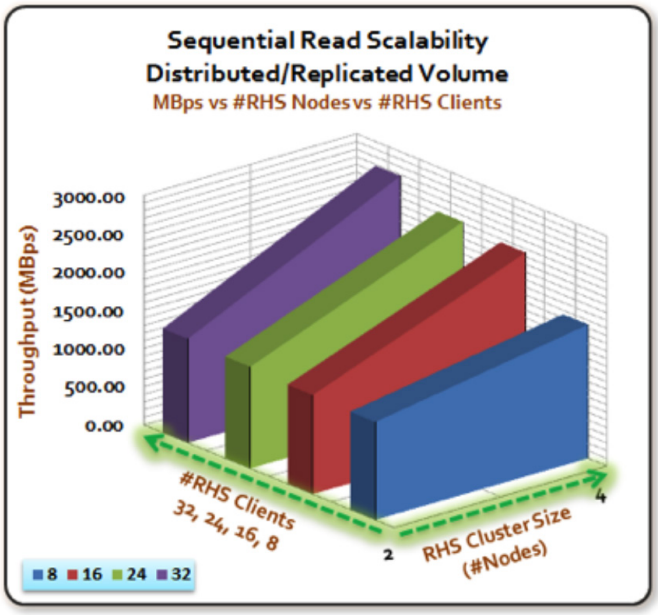


Figure 20: **RHS Distributed-Replicated Volume**, 4-nodes, 32 RHS clients – SeqRd/Wr performance saturates at reqsz > 64KB, RandRd/RandWr scale

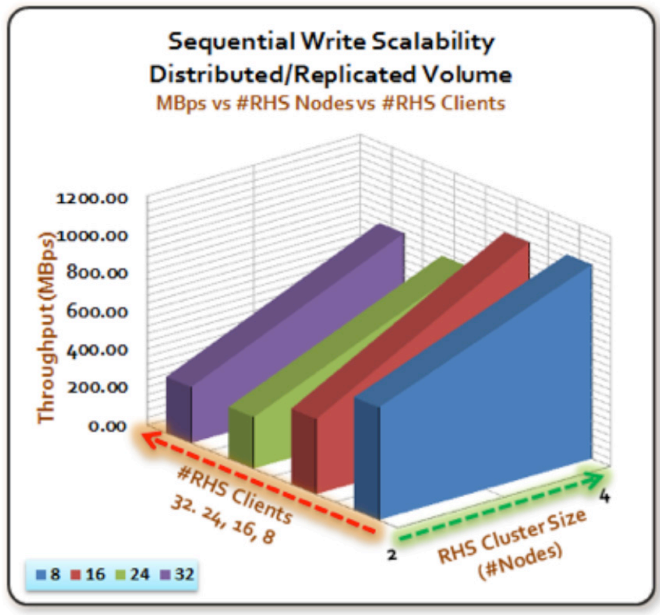
We used 1024KB as the representative record size (request size) for presenting the data gathered during this experiment.



Replicated Mode - Sequential Read Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2		4
8	1277.53		1747.54
16	1294.84		2352.97
24	1328.62		2419.96
32	1357.97		2685.61
iozone record size	1024KB		
iozone file/transfer size	8GB		

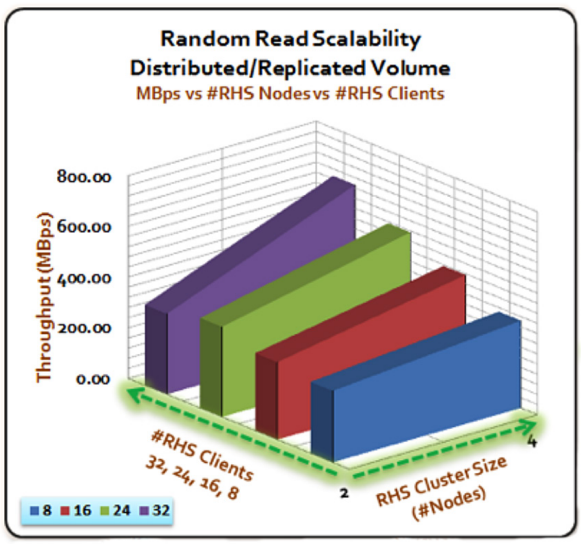
Figure 21: **SeqRd Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added



Replicated Mode - Sequential Write Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2		4
8	589.28		1015.78
16	393.98		997.21
24	269.67		740.75
32	293.44		778.69
iozone record size	1024KB		
iozone file/transfer size	8GB		

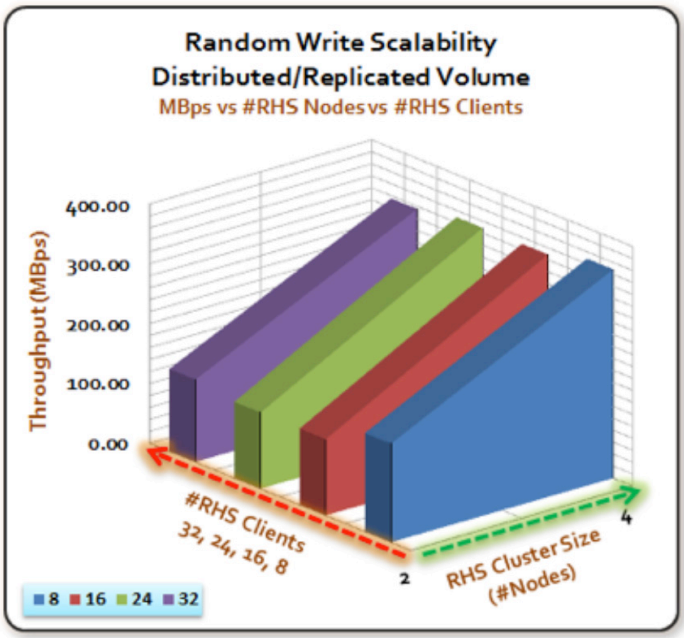
Figure 22: **SeqWr** Throughput Scales upward Linearly as **Storage Nodes** are added, however exhibits decline as **more RHS Clients** are added – this decline may be attributed to added system pressure and back-end network traffic due to replication.



Distributed/Replicated Mode - Random Read Performance/Scalability

	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2		4
8	281.38		344.10
16	305.37		435.01
24	354.81		514.11
32	317.19		611.95
iozone record size	1024KB		
iozone file/transfer size	8GB		

Figure 23: **SeqRd Throughput Scales Linearly** as Storage Nodes and/or RHS Clients are added



Distributed Mode - Random Write Performance/Scalability			
	RHS Cluster Size (#nodes) vs Throughput (MBps)		
#Client threads / #Nodes ->	2		4
8	164.73		348.75
16	152.83		329.23
24	127.34		328.94
32	137.69		315.71
iozone record size	1024KB		
iozone file/transfer size	8GB		

Figure 24: SeqWr Throughput Scales **upward** Linearly as **Storage Nodes** are added, however exhibits decline as **more RHS Clients** are added – this **decline** may be attributed to added system pressure and back-end network traffic due to replication. When compared to the drop in performance in the replicate-only configuration (Figure 22), the drop in this case is small and the advantage of using Distribute-replicate combination is evident.

Performance Considerations

This section describes the recommended system configuration settings in order to maximize performance in the Content Cloud use case.

Red Hat Storage Defaults

Red Hat Storage comes preconfigured with the following settings – the defaults work well for the Content Cloud/Large File Store use case.

1. TCP auto-tuning (tuned socket buffer sizes)
2. Deadline I/O scheduler (https://www.kernel.org/doc/Documentation/block/switching-sched.txt)
3. Reduced Linux kernel dirty page write-back frequency (dirty_background_ratio) from 10 percent to 1 percent

Tuned Configuration

Tuned is a daemon that monitors the use of system components and dynamically tunes system settings based on that monitoring information. Dynamic tuning accounts for the way that various system components are used differently throughout the uptime for any given system. Tuned monitors CPU, network, storage activity and reacts to changes in their use.

"rhs-high-throughput" tuned profile is recommended for sequential file access. This profile helps optimize a multi-stream workload at the expense of latency.

```
# tuned-adm profile rhs-high-throughput
```

Networking Configuration

10GbE infrastructure coupled with Intel® 10GbE network interface cards will yield the best performance from RHS. Jumbo frames (MTU = 9000) and LACP 802.3ad bonded interfaces will help with a jump in the cached read throughput. With jumbo frames with native GlusterFS client access, for IO sizes >= 64KB, one should expect to use most of 10GbE pipe.

Storage Array - RAID6 Configuration

Larger hardware RAID stripe element size utilizes the hardware RAID cache effectively and can provide more parallelism at RAID level for small files because fewer disks would be required to participate in the read operations. For RAID 6, the recommended stripe size is 256 KB.

LVM Configuration

If using LVM on top of RAID6 for creating RHS bricks, the "dataalignment" option to "pvcreate" helps to make sure that IO is aligned at the RAID6 stripe boundary. The value is calculated by multiplying the RAID6 stripe size by the number of data disks in the RAID volume. As noted in the cookbook section, we used 12-

disk RAID6 configuration (10 data + 2 parity disks) with 256KB stripe size. This translates to dataalignment size of 2560KB.

```
# pvcreate -dataalignment 2560k <RAIDDEVICE>
```

XFS Filesystem Configuration

XFS allows the selection of logical block size for directories that is greater than the logical block size of the file-system. Increasing the logical block size for the directories from default 4K to 8K, decreases the directory IO, this improves the performance of directory operations. For example:

```
# mkfs.xfs -I size=512 -n size=8192 \
       -d su=256k,sw=10 /dev/<RHS _ VG>/<RHS _ LV>
```

XFS Mount Options

Most applications do not require an update to the access time on files or directories. It is recommended that RHS bricks are mounted with the "noatime" and "nodiratime" option.

Also, inode64 option allows inodes to be placed near to the data which minimizes disk seeks.

Entry in fstab:

```
# /dev/<RHS _ VG>/<RHS _ LV> /rhs/brick1 xfs
inode64,noatime,nodiratime 1 2
```

RHS/GlusterFS Client Access Method

Native Client is a FUSE-based client running in user space. Native Client is the recommended method for accessing Red Hat Storage volumes when high concurrency and high write performance is required.

For additional Linux performance tuning guidelines, see Red Hat Linux Performance Tuning Guide.

Other Things to Consider

Power Optimization

Power consumption for large amounts of storage (i.e. Petabyte (1015) or even Zettabytes (1021)) can become a significant part of the total cost of ownership for a storage cloud. The scale-out storage cloud test could be augmented by profiling the power consumption of the racks when idle, power aware workload/ virtual machine scheduling, or being tested by the usage models. The power utilization could be optimized through technology such as Intel® Intelligent Power Node Manager.

Conclusion

By delivering increased scalability, flexibility, affordability, performance, and ease-of-use in concert with reduced cost of acquisition and maintenance, Red Hat Storage® is a revolutionary step forward in data management. The complete elimination of centralized metadata server and the use of the Elastic Hashing Algorithm is at the heart of many of Red Hat Storage's fundamental advantages. This dramatically reduces the risk of data loss, data corruption, and increases data availability. Red Hat Storage can be deployed in the private cloud or datacenter via Red Hat Storage ISO image installed on standard server hardware based on Intel Xeon E5-2600 series processors paired with standard storage hardware, resulting in a powerful, turn-key, massively scalable, and highly available storage environment. Extensive testing on the reference architecture deployment discussed in this paper helps prove these benefits by demonstrating the ease of Red Hat Storage setup on standard Intel hardware and shows how Red Hat Storage performance scales linearly with the addition of storage nodes or with the number of concurrent client requests for data.

For more information about Red Hat Storage services and product offerings, visit http://www.redhat.com/storage/.

Glossary

CIFS: Common Internet File System. Also known as Server Message Block (SMB) is a network protocol used to provide shared access to files, printers, serial ports, and miscellaneous communications between nodes on a network – typically a set of Microsoft Windows servers and PC clients. See: http://en.wikipedia.org/wiki/Server_Message_Block.

Deduplication: Data deduplication (or Dedup) is a specialized data compression technique for eliminating coarse-grained redundant data, typically used to improve storage utilization. In the deduplication process, duplicate data is deleted, which leaves only one copy of the data to be stored, along with references to the unique copy of data. Deduplication reduces the required storage capacity since only the unique data is stored. From: http://en.wikipedia.org/wiki/Data_deduplication.

FUSE: File System in User Space, kernel module which allows non privileged users on a Linux operating system to create a file system on the local machine.

HTTP: The Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. See http://en.wikipedia.org/wiki/HTTP.

JBOD: Just a Bunch of Disks.

Mdadm: (Multiple Device Administrator) is a Linux utility to manage software RAID.

Metadata: Metadata is loosely defined as data about data. Metadata is a concept that applies mainly to electronically archived or presented data and is used to describe the a) definition, b) structure, and c) administration of data files with all contents in context to ease the use of the captured and archived data for further use. From: http://en.wikipedia.org/wiki/Metadata.

Namespace: Namespace is an abstract container or environment created to hold a logical grouping of unique identifiers or symbols.

NAS: Network Attached Storage is a storage server or appliance that uses file-based protocols such as NFS (Network File Server) or CIFs to enable clients (typically servers and PCs) to access files over a TCP/IP network. See http://en.wikipedia.org/wiki/Network-attached_storage.

NFS: Network File System protocol designed to let client access files on the server over the network connection.

RAID: Redundant Array of Independent Disks.

Replication: Data replication is the process of sharing data so as to improve reliability between redundant storage devices. The replication is transparent to an application or end-user. In a failure scenario, failover of replicas is hidden as much as possible.

REST: Representational State Transfer. REST is an architecture for client-server communication over a TCP/IP network (e.g. Internet). Clients initiate requests to servers; servers process requests and return appropriate responses. At any particular time, a client can either be in transition between application states or "at rest." A client in a rest state is able to interact with its user, but creates no load and consumes no per-client storage on the set of servers or on the network. The client begins to send requests when it is ready to make the transition to a new state. The Hypertext Transfer Protocol (HTTP) is commonly used as the transport layer basis for REST communication. See http://en.wikipedia.org/wiki/Representational_State_Transfer.

SAN: Storage Area Network is a storage server or appliance that uses block-based protocols typically based on SCSI to access files over a fiber channel or TCP/IP network. See http://en.wikipedia.org/wiki/Storage_area_network.

Scale-out Storage (SOS): SOS is a usage model for storage that enables an enterprise to grow capacity incrementally as it adds more storage nodes (typically as a new server on an IP network). The goal of scale-out storage is to grow capacity with near-linear (versus lump sum) investment.

TCO: Total Cost of Ownership.

To learn more about deployment of cloud solutions, visit www.intel.com/cloudbuilders.

For more information on Intel® Xeon® processors, see www.intel.com/xeon.

For more information on Red Hat® Storage Products, see http://www.redhat.com/products/storage-server.

For more information on Intel® Intelligent Power Node Manager Technology, see www.intel.com/technology/nodemanager.

Disclaimers