DELL EqualLogic™

A Dell Technical White Paper

Scalable Storage for High Performance Computing: Red Hat ® Global File System on Dell EqualLogic™

By Jacob Liberman
Dell │ Enterprise Product Group

THIS WHITE PAPER IS FOR INFORMATIONAL PURPOSES ONLY, AND MAY CONTAIN TYPOGRAPHICAL ERRORS AND TECHNICAL INACCURACIES. THE CONTENT IS PROVIDED AS IS, WITHOUT EXPRESS OR IMPLIED WARRANTIES OF ANY KIND.

Contents

Introduction

Over the past decade clusters have emerged as the dominant solution for High Performance Computing (HPC) applications. Clusters combine commodity servers, standards-based interconnects, and open source software to deliver supercomputer-like performance at a fraction of the cost. However, clustering's emergence in HPC has not been without growing pains. Clusters frequently need fast access to large data sets. Large scale storage solutions remain proprietary and expensive, often accounting for more than half of a cluster's acquisition costs. The economic factors that spurred clustering's rapid growth in HPC have not influenced the storage subsystem.

This whitepaper demonstrates that departmental clusters can achieve scalable I/O performance equal to that of proprietary I/O solutions by combining standards-based iSCSI storage with the Global File System (GFS), an open-source cluster file system. Small Computer Systems Interface (SCSI) is a popular family of protocols for communicating with storage devices. The iSCSI protocol was defined as a means of transporting SCSI packets over TCP/IP. iSCSI takes advantage of ubiquitous IP networking infrastructure to provide block-level I/O with networked storage devices over long distances. For these reasons, iSCSI has emerged as a popular, low-cost alternative to other Storage Area Networking (SAN) protocols such as Fibre Channel. GFS is a 64-bit cluster file system that allows multiple clients simultaneous access to the same file system namespace. GFS was originally developed at the University of Minnesota under the GPL license. More recently GFS has been acquired and productized by Red Hat, Inc. It remains open source.

The paper begins with an overview of the required hardware and software before providing detailed installation instructions. The paper concludes with performance data from common I/O benchmarks. The results demonstrate that clusters can achieve scalable I/O performance equal to that of proprietary solutions using only standards-based hardware and open-source software.

Dell EqualLogic PS Series iSCSI SAN

Dell EqualLogic PS Series storage combines standards-based iSCSI initiators and Serial Attached SCSI (SAS) or Serial ATA (SATA) disks with unique array-based management software to provide a flexible building block for scalable storage. They feature redundant controllers, power supplies, and fans for fault tolerance. Each controller has three Gigabit Ethernet ports for host access. The controllers are active/passive, meaning that only one controller can field client I/O requests at a time. Although the PS Series storage supports both SAS and SATA disks, the arrays used in this study were all identically equipped with sixteen 73 Gigabyte, 15k RPM SAS disks.

Although comprised entirely from standards based hardware, the PS Series storage includes innovative features to help enhance performance and flexibility. Multiple PS arrays ("members") can be combined into a "group" manageable via a single IP address. The members' RAID volumes are combined into one or more storage pools. The volumes in the pool are automatically load balanced across all members at the block level. Furthermore, client connections are also automatically load balanced across each member's active controller, doubling the group's aggregate I/O capacity. A two member PS group with a three volume storage pool is depicted in Figure 1.

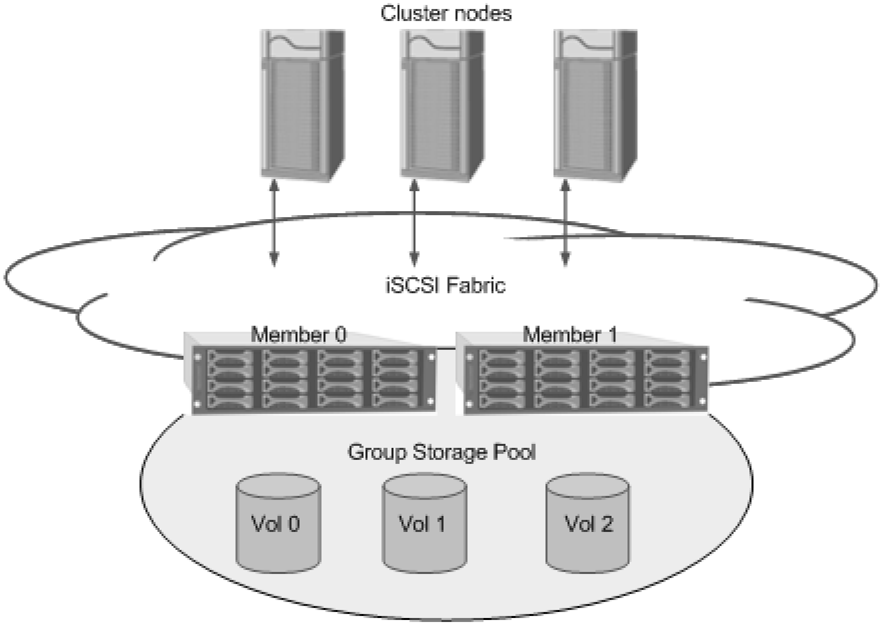

Figure 1 – A two member PS group. The cluster nodes access the volumes via a single IP address.

Although the underlying RAID levels and disk types may differ between members, clients can access the storage pool through the PS Series group portal address. When new members are added to the group, the storage pool expands to include their capacity and performance without host interruption. Furthermore, volumes can be distributed across multiple group members at the block level, automatically load balancing client requests based on their data access patterns. The PS Series storage also supports other features such as multi-volume snapshots, thin provisioning, and multi-way replication. These features and capabilities come standard with the PS Series iSCSI SAN at no additional cost.

Red Hat Global File System

HPC nodes typically access shared storage via a distributed network file system such as NFS. GFS clients mount shared external storage as a local filesystem. Data are stored as blocks on the underlying Logical Volume Manager (LVM) subsystem. A lock manager coordinates file access among the clients in order to ensure disk cache coherency. GFS supports several lock managers. This case study used the Distributed Lock Manager (DLM) for interprocess synchronization across cluster nodes. At the time of writing, GFS with DLM had been tested for scalability up to 125 nodes and theoretically supports many more. DLM lock requests require from zero to two remote operations depending on the locality of the lock request. These network operations can introduce additional processing overhead as the number of nodes scales. A large number of cluster nodes increase the likelihood that lock requests will require remote operations. In spite of the additional overhead introduced by GFS kernel space modules and network lock requests, GFS and iSCSI combine to provide scalable and reliable storage access with minimal performance impact.

GFS features include POSIX compliance, native 64-bit, and Red Hat Enterprise Linux® integration. It also includes many file system features such as file access control lists, fencing agents to ensure data integrity, multiple journals for rapid data recovery, and online volume expansion. The file locking data are also distributed across multiple nodes. This provides fault tolerance for the cluster meta-data. In the past HPC customers have been reluctant to SAN attach their compute nodes to shared storage because the high cost per port of Fibre channel initiators can quickly erode a cluster's favorable economics. By SAN attaching nodes to GFS on PS series storage via iSCSI, HPC administrators can enjoy the flexibility, performance, and fault tolerance of a SAN at a fraction of the hardware cost.

There are currently two versions of the GFS file system: GFS and GFS2. GFS2 contains many enhancements over GFS. At the time of printing, GFS2 is still under development. In this study performance measurements were made with both GFS and GFS2. All performance numbers are for GFS unless otherwise stated.

Installation

This section describes the steps necessary to duplicate the case study. The installation procedure includes network, storage, and host configuration details. Please note that the configuration outlined in this paper is not ideally suited to all workloads.

Figure 2 describes the complete installation procedure.

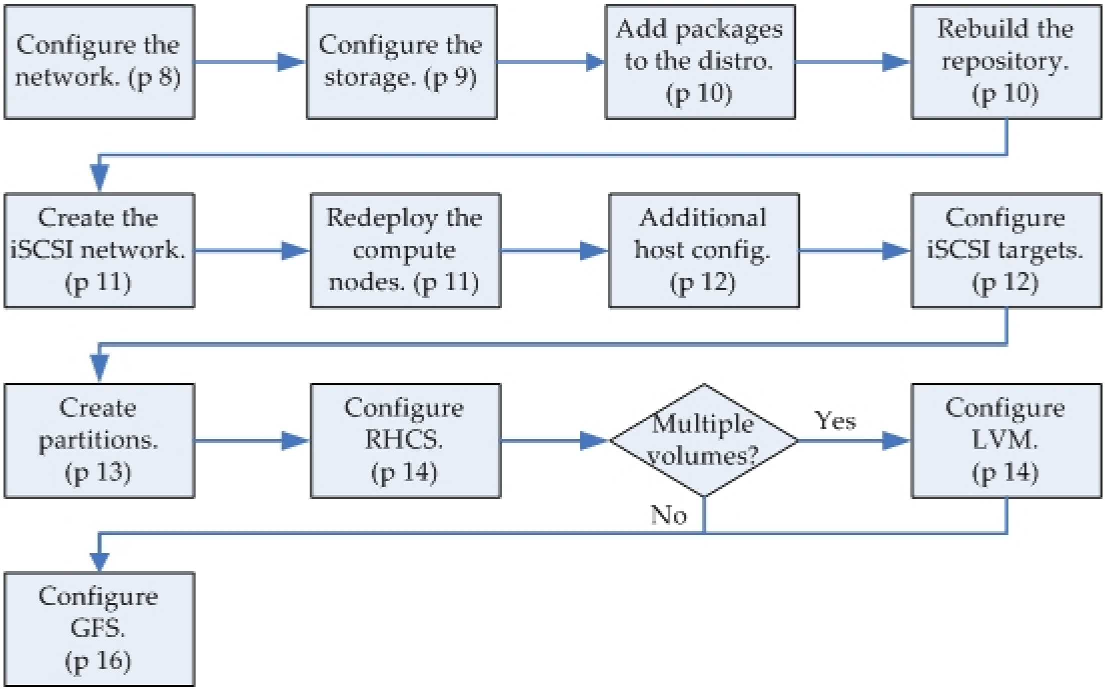
Configure the network. (p 8)
Configure the storage. (p 9)
Add packages to the distro. (p 10)
Rebuild the repository. (p 10)
Create the iSCSI network. (p 11)
Redeploy the compute nodes. (p 11)
Additional host config. (p 12)
Configure iSCSI targets. (p 12)
Create partitions. (p 13)
Configure RHCS. (p 14)
Multiple volumes?
Yes
Configure LVM. (p 14)
No
Configure GFS. (p 16)

Figure 2 -- Installation flowchart

Network Configuration

PS Series iSCSI SAN utilizes standard IP hardware and interconnect without costly additions to the networking infrastructure. However, to help meet optimal performance, reliability and scalability goals, iSCSI network component configuration should implement the following recommendations.

In general, the switch used to implement this approach should show good scaling. Although this case study does not describe best practices for fault tolerance, redundancy is a consideration of any SAN design. Layer three stackable switches provide good scaling with low blocking ratios or non-blocking configurations.

The selected switch should support a fast convergence method such as Rapid Spanning Tree in order to reduce the effect of state changes. The switch should also support Flow Control and Maximum Transmission Unit (MTU) control for best performance. Finally, users can achieve optimal performance with switches that have at least 512Kb buffers per port.

This study was conducted using a layer three Cisco Catalyst WS-6509 switch with WS-X6748-GE-TX modules. Each port was set to switchport mode access, MTU of 9216, and Incoming/Outgoing Flow Control enabled.

Storage Configuration

The backend of PS arrays utilize SAS and SATA drives. Both the number of drives and RAID policy per volume influence benchmark performance. Tradeoffs between capacity, cost, fault tolerance, and performance should guide design decisions. The PS5000 Series arrays used in this study support three RAID types: 5, 10, and 50. The PS5000 Series array has 16 SAS drives. The RAID 10 policy configures seven sets of mirrored drives and stripes across them, leaving two drives as spares. RAID 10 has the same fault tolerance as RAID 1 but provides better performance by striping segments. The RAID 50 policy creates two RAID 5 sets and stripes across them. Two drives are also left for spares. RAID 50 volumes provide higher capacity than RAID 10 although the performance in terms of I/O Operations per Second (IOP/s) can be lower than the RAID 10 equivalent. The RAID 5 policy stripes parity data across a single fifteen drive set and leaves one standby drive. RAID 5 gives the highest capacity.

Optimal RAID configuration for performance is a function of data access patterns. For HPC customers, sequential workloads dominate and I/O requests are more heavily weighted toward reads than writes. Therefore, the recommendations in this case study are geared toward maximizing sequential read workloads. Performance was compared across all three RAID types to validate these assumptions. Only SAS disks were used and only one volume was configured for each group member. Additional configuration details are described in the benchmark description for each test case.

Host Configuration

The following host configuration instructions assume that the cluster was deployed with Platform Open Cluster Stack (OCS) 5. Platform OCS is a provisioning and management solution for Linux based clusters. The host operating system is Red Hat Enterprise Linux Server 5.1 Advanced Platform with kernel 2.6.18-53.el5. For more information regarding Platform OCS, please visit http://my.platform.com/products/platform-ocs.

A. Deploy Software

Platform OCS requires an installer node with at least two network interfaces. The first network interface is used to deploy the cluster nodes. This is called the "provision" network. The second cluster interface provides a remote access gateway to the cluster. This is called the "public" network. The cluster nodes are all connected to the installer node via the provision network. Additionally, for the purposes of this study, the cluster nodes are also attached to a third network called "storage." The storage network includes the secondary network interfaces for all compute nodes and the iSCSI ports for the PS Series group members.

OCS is built around the notion of "node groups." Compute nodes are logically divided into "node groups" that define common shared characteristics of the nodes. They share identical configurations, software packages, and responsibilities. Each node group is installed from a software repository that can be expanded to include additional software packages and post-installation configuration scripts. When changes are made to a node group, such as adding additional packages, the node group members are synchronized to the new configuration. This synchronization can either be pushed through an online update service called "cfm" or pulled by reinstalling the nodes. Each node group is installed from a repository that contains the additional packages and update scripts.

The following paragraphs describe the changes made to deploy the compute nodes as iSCSI SAN clients to the PS Series storage. Although it is possible to create an additional node group, for the sake of simplicity, the following steps describe how to extend the existing "compute-rhel-5-x86_64" group to include the new functionality.

I. Add packages to distribution

First, add the following packages to the node group with ngedit:

a. cman
b. elfutils
c. gfs-utils
d. gfs2-utils
e. gnome-python2
f. gnome-python2-canvas
g. oddjob
h. openais
i. perl-Net-Telnet
j. perl-XML-Libxml
k. pygtk2
l. pygtk2-libglade
m. python-imaging
n. rpm-build
o. unifdef

These packages can be added via ngedit because they already exist in the /depot/repos/1000/Server/ directory on the installer node.

Note that the iscsi-initiatorutils package is also required, but included by default this package is included in the node group base template. Furthermore, if the cluster is deployed with OFED support, the open-iscsi package used in OFED will also be installed and supersede the iscsi-initiatorutils package.

II. Add packages to contrib and rebuild repository

Next, create symbolic links from the /depot/repos/1000/Cluster directory to the /depot/contrib/rhel/5/x86_64/ directory for the following packages:

a. Cluster_Administration-en
b. luci
c. modcluster
d. rgmanager
e. system-config-cluster

Create symbolic links from the /depot/repos/1000/ClusterStorage directory to the /depot/contrib/rhel/5/x86_64/ directory for the following packages:

a. Global_File_System-en-US
b. kmod-gfs
c. kmod-gfs2
d. lvm2-cluster

For example:

```
[root@li ~]# ln -s /depot/repos/1000/Server/luci-0.10.0-6.el5.x86_64.rpm
/depot/repos/1000/Server/
```

Once the packages are available in /depot/contrib/rhel/5/x86_64/, rebuild the repository with the repoman command.

```
[root@li ~]# repoman -u -r "Repo for rhel-5-x86_64"
```

After the repository rebuilds successfully, add these packages to the node group with ngedit.

III. Create iSCSI Network

By default the compute node group is configured with a single provision network for the first Ethernet controller. Configure the node group so that the second onboard controller can be used for iSCSI traffic. Linux software packages such as MPIO can provide hosts with additional bandwidth and fault tolerance by aggregating multiple initiators. These packages were not used in this study because the cost of adding additional network cards can quickly erode a large cluster's favorable economics.

As root, run netedit and select **New** to create an additional network. Fill in the network address, netmask, starting address, increment value, and description. Select **public** as the interface type, and then select **OK**.

For this study, the following netedit parameters were used:

```
Network: 10.40.0.0
Subnet: 255.255.0.0
Gateway: 10.40.0.254
Device: eth1
Starting IP: 10.40.40.10
Suffix: -eth1
Increment: 1
Options:
Description: iscsi
Interface type: Public
```

Next, use ngedit and select the compute node group. Select **Next** until the Network screen is displayed. Assign the new network to a node group by putting an asterisk next to the network group using the spacebar and selecting **Next**. After exiting ngedit, OCS will update the node group so that the second network interface of deployed hosts will be configured to the specifications set in netedit.

IV. Redeploy compute nodes

Update the PXE files and redeploy the compute nodes with the boothost command.

NOTE: Dell component-Dell-pxeboot must be added through ngedit before redeploying nodes if PXE is not set before the internal disk in the BIOS boot order.

```
[root@li ~]# boothost -r -n compute-rhel-5-x86_64
```

Check the status of the reinstallation with the nghosts command. When the nodes have completed reinstallation, the status will change to “Installed.”

```
[root@li ~]# nghosts -l
```

V. Additional Host Configuration

These post-installation tasks include synchronizing the system clocks and increasing the MTU size for the iSCSI interface.

Set the MTU in /etc/sysconfig.network-scripts/ifcfg-eth1.

```
[root@li ~]# pdsh -a 'echo "MTU=9000" >> /etc/sysconfig/network-
scripts/ifcfg-eth1'

[root@li ~]# pdsh -a "ifdown eth1 && ifup eth1 && ifconfig eth1 | grep -i
mtu"
```

Use either the date command or the Network Time Protocol (NTP) daemon to synchronize the time on the compute nodes. The time reported by the compute nodes must match in order for the cluster file system to work correctly. The following example shows how to set the compute nodes’ time using the date command through pdsh.

```
[root@li ~]# pdsh -a date -s 10:54
```

B. Configure the File System

The following section describes how to configure the compute node’s second Gigabit Ethernet network card as an iSCSI software initiator. The instructions below describe how to configure the iSCSI initiator using the Parallel Distributed Shell (PDSH), which executes shell commands in parallel on all hosts listed in /etc/hosts.pdsh. These commands can also be issued during installation via a custom script. An example script is included in Appendix A.

I. Configure the iSCSI initiator

1. First, start the iSCSI service and configure it to start automatically at boot on all compute nodes that will attach directly to the PS Series storage.

```
[root@li ~]# pdsh –a service iscsi start

[root@li ~]# pdsh –a chkconfig iscsi on
```

2. Next create an iSCSI software initiator for eth1.

```
[root@li ~]# pdsh –a iscsiadm -m iface -I iface1 --op=new
```

```
 [root@li ~]# pdsh –a iscsiadm -m iface -I iface1 --op=update -n
iface.net_ifacename -v eth1
```

3. Discover and list the target volumes by specifying the IP address of the PS Series group.

   ```
   [root@li ~]# pdsh –a iscsiadm -m discovery -t st -p 10.40.40.110 -I iface1
   ```

4. Login the host to initiate iSCSI sessions with each volume.

   ```
   [root@li ~]# iscsiadm -m node -L all
   ```

In the following example, host compute-00-00 has initiated two sessions with the PS Series storage. Each session corresponds to a volume.

```
[root@compute-00-00 ~]# iscsiadm -m session
tcp: [1] 10.40.40.110:3260,1 iqn.2001-05.com.equallogic:0-8a0906-
a7e233b02-f0b00007fb948f4c-vol00
tcp: [2] 10.40.40.110:3260,1 iqn.2001-05.com.equallogic:0-8a0906-
a86233b02-7fd00007fbc48f4c-vol01
```

II. Create Partitions

1. Each iSCSI session will correspond to a new disk device listed in the Linux system file /proc/partitions. In the following example, the initiator has sessions with two volumes which are discovered as sdb and sdc. For single volume configurations, create only one partition.

   ```
   [root@compute-00-00 ~]# cat /proc/partitions
   major minor  #blocks  name

      8     0   71687325 sda
      8     1     104391 sda1
      8     2   12289725 sda2
      8     3    2048287 sda3
      8     4          1 sda4
      8     5    2048256 sda5
      8     6   55191276 sda6
      8    16   71687325 sdb
      8    32   71687325 sdc
   ```

2. From a single host, create one primary partition with a GPT label on each volume.

   ```
   [root@compute-00-00 ~]# parted -s /dev/sdb mklabel gpt

   [root@compute-00-00 ~]# parted -s /dev/sdb "mkpart primary 0 -1"

   [root@compute-00-00 ~]# parted -s /dev/sdc mklabel gpt

   [root@compute-00-00 ~]# parted -s /dev/sdc "mkpart primary 0 -1"
   ```

3. Discover the newly created partitions from all the compute nodes with partprobe.

   ```
   [root@li ~]# pdsh -a partprobe -s | dshbak -c
   ----------------
   compute-00-00-eth0,compute-00-01-eth0,compute-00-02-eth0,compute-00-03-e…
   ----------------
    /dev/sda: msdos partitions 1 2 3 4 <5 6>
    /dev/sdb: gpt partitions 1
    /dev/sdc: gpt partitions 1
   ```

III. Configure Red Hat Cluster Service

Red Hat Cluster Service (RHCS) provides an underlying framework for GFS. RHCS provides startup and management services used to co-ordinate cluster numbers and ensure quorum. Not all RHCS clusters require GFS, but all GFS file systems require RHCS. A detailed description of RHCS services, locking methods, and fence devices can be found at **http://www.redhat.com/docs/manuals/csgfs/**.

1. First, create a cluster.conf file and copy it to /etc/cluster/cluster.conf on all nodes. Appendix B contains an example cluster.conf file.

NOTE: This example file has been shortened to 8 nodes for the sake of brevity. The test was performed on 32 nodes.

```
[root@li ~]# pdsh -a cp /home/jacob/cluster.conf /etc/cluster/cluster.conf
```

2. Next, load the underlying RHCS drivers. These drivers can also be loaded during RHCS service startup.

```
[root@li ~]# pdsh -a modprobe gfs

[root@li ~]# pdsh -a modprobe dlm

[root@li ~]# pdsh -a modprobe lock_dlm
```

3. Start the cluster manager service on all nodes. This will establish a quorum and create the cluster group.

```
[root@li ~]# pdsh –a service cman start
```

4. Verify all nodes have joined the cluster with cman_tool. In the following example, 32 nodes have joined the ps00 cluster.

```
[root@li ~]# ssh compute-00-00 cman_tool status
Version: 6.0.1
Config Version: 1
Cluster Name: ps00
Cluster Id: 1500
Cluster Member: Yes
Cluster Generation: 516
Membership state: Cluster-Member
Nodes: 32
Expected votes: 32
Total votes: 32
Quorum: 17
Active subsystems: 7
Flags:
Ports Bound: 0 11
Node name: compute-00-00
Node ID: 1
Multicast addresses: 239.192.5.225
Node addresses: 172.20.0.2
```

IV. Configure LVM

LVM is the Logical Volume Manager for the Linux operating system. The version used in this study is LVM2, which uses the device mapper kernel driver. Logical volume management allows storage to be flexibly allocated, resized, and moved between hosts. Once the cluster service is running on all nodes, use LVM to configure the shared storage for host access.

In this study, multiple PS Series RAID volumes were striped together via LVM2 in order to create a single file system namespace across volumes that spanned multiple arrays. The PS Series group will intelligently load balance a single volume across group members after it has acquired enough historical usage data to identify the most requested blocks. However, due to the time constraints imposed by this study, the decision to stripe multiple volumes with LVM was made to ensure that the volume capacity was evenly distributed across both group members from the outset. This section describes how two PS Series volumes were combined into a single volume for the multi-member group performance tests.

For the single member group performance tests, LVM2 was not used, as all I/O requests targeted a single volume. It is not necessary to create a logical volume for single volume configurations regardless of whether or not the volume spans multiple members. Skip to step 5 for single volume installation instructions.

1. On all nodes, change the default locking_type in /etc/lvm/lvm.conf to support clustered logical volumes.

   ```
   [root@li ~]# pdsh -a "sed -i.old 's/locking_type = 1/locking_type = 3/'
   /etc/lvm/lvm.conf"
   ```

   This can also be accomplished by running **lvmconfig –enable-cluster** on all nodes.

2. On all nodes, start clvmd.

   ```
   [root@li ~]# pdsh -a service clvmd start
   ```

3. From a single node, create physical volumes that can be added to volume groups using pvcreate.

   ```
   [root@li ~]# ssh compute-00-00 pvcreate /dev/sdb1 /dev/sdc1
     Physical volume "/dev/sdb1" successfully created
     Physical volume "/dev/sdc1" successfully created
   ```

4. Detect the physical volumes on all nodes with pvscan.

   ```
   [root@li ~]# pdsh -a pvscan
   ```

5. From one node, create a volume group with vgcreate.

   ```
   [root@li ~]# ssh compute-00-00 vgcreate ps00 /dev/sdb1 /dev/sdc1
   ```

6. Detect the volume group on all nodes with vgscan.

   ```
   [root@li ~]# pdsh -a vgscan
   ```

7. From one node, create a striped logical volume using lvcreate. Striping the logical volume will ensure that client connections are load balanced across the PS Series group members in multi-member groups.

   ```
   [root@li ~]# ssh compute-00-00 lvcreate ps00 -l 100%FREE -i 2
   ```

8. Detect the logical volume on all nodes with lvscan.

```
[root@li ~]# pdsh -a lvscan
```

V. *Configure GFS*

Red Hat GFS is a cluster file system that provides data sharing between attached clients. The following steps create a GFS file system on the clustered logical volume.

1. First, create a mount point for the GFS file system on all nodes. In the following example, the mount point is named ps00.

```
[root@li ~]# pdsh –a mkdir /ps00
```

2. From a single node, format the file system. The file system name is arbitrary. The cluster name should match the name defined in /etc/cluster/cluster.conf.

```
[root@compute-00-00 ~]# mkfs -t gfs -p lock_dlm -t ps00:test -j 64
/dev/ps00/lvol0
```

The number of journals should equal or exceed the number of cluster nodes. Additional journals allow for future expandability at the cost of reduced file system capacity.

3. Mount the GFS file system from all nodes.

```
[root@li ~]# pdsh -a mount -t gfs /dev/ps00/lvol0 /ps00
```

At this point the shared storage is configured and accessible by all nodes. For more information, please refer to **http://www.redhat.com/docs/manuals/csgfs/**.

Appendix C summarizes the commands used in this section.

Performance

Large HPC clusters typically require fast access to large data sets. The I/O requirements of large clusters are often met via either parallel file systems re-exported through SAN attached servers or distributed file systems such as NFS. In both cases, these extra servers do not participate in the computation. The configuration tested in this study leverages the parallel data distribution characteristics of the underlying PS Series hardware to reach parallel file-system-like performance without the additional expense of the file servers. Eliminating costly file servers allows greater investment in cluster nodes that perform the computation. This can be accomplished by SAN attaching the compute nodes directly to the PS Series group via iSCSI.

This performance study investigates the random and sequential I/O performance of iSCSI SAN attached compute nodes at various RAID levels. It also quantifies the overhead introduced by the cluster file system and inter-array communication as the PS Series group increases from one to two members. Finally, this study measures the scalability of the PS Series array as the client count scales from one to thirty two in order to identify an optimal array-to-client ratio for various I/O patterns and RAID policies.

Much like the memory subsystem, I/O subsystem performance can be improved by implementing multiple cache levels. In this study, attempts were made to minimize the effect of client-side caching in order to compare the performance characteristics of the underlying array configurations and RAID policies. Excessive client-side file system caching flattens the differences between RAID policies by measuring the speed at which clients can access data from their own file system buffers stored in local memory. To that end, all I/O operations utilized files at least 2x the size of the amount of RAM in each node. Each node had four one-Gigabytes DDR-667 Fully Buffered DIMMs, so the file size used for all I/O operations was eight Gigabytes.

Policy	Disks	Capacity	Spares
10	16	7 x drive size	2
50	16	12 x drive size	2
5	16	14 x drive size	1

Table 1 - RAID Policy Comparison

The PS Series array supports three RAID policies: 10, 50, and 5. The RAID policies differ in terms of capacity, performance, and fault tolerance. These differences are summarized in Table 1.

The RAID policy is configured on a per member basis, meaning the same RAID policy applies to all volumes within a single PS Series array. In a multi-member group, the RAID policy is set on a per member basis. Each member can have its own RAID policy, hard drive type, and hard drive capacity. In the multi-group member tests in this study, the RAID policies and disk types were consistent across both group members. For example, during the two member RAID 50 performance tests, both group members had identically configured volumes on RAID 50 arrays and 15K RPM SAS hard drives.

I/O Performance Results

Iozone is a file system benchmark tool that generates and measures a variety of file operations including sequential read, sequential write, random read, and random write. Iozone features include a distributed fileserver option used to measure I/O from multiple nodes in a cluster. This feature was used to gather the I/O performance of GFS on PS Series storage arrays as the client count scaled from 1 to 32 nodes. The distributed file server measurement feature was scripted to initiate one iozone thread from each node in the cluster. The benchmark aggregates and presents the cluster level performance data in Megabytes per second. Additional iozone parameters for each test and included in the test result descriptions.

The first test compared the sequential write performance of one to thirty two clients to a two member group across the three RAID policies. All clients simultaneously accessed a single GFS file system. The file system was formatted on a Linux LVM logical volume. The underlying logical volume was comprised of a single pool with two physical volumes – one on each array member – striped together with client side Logical Volume Management. The results of this test are depicted in Figure 3.

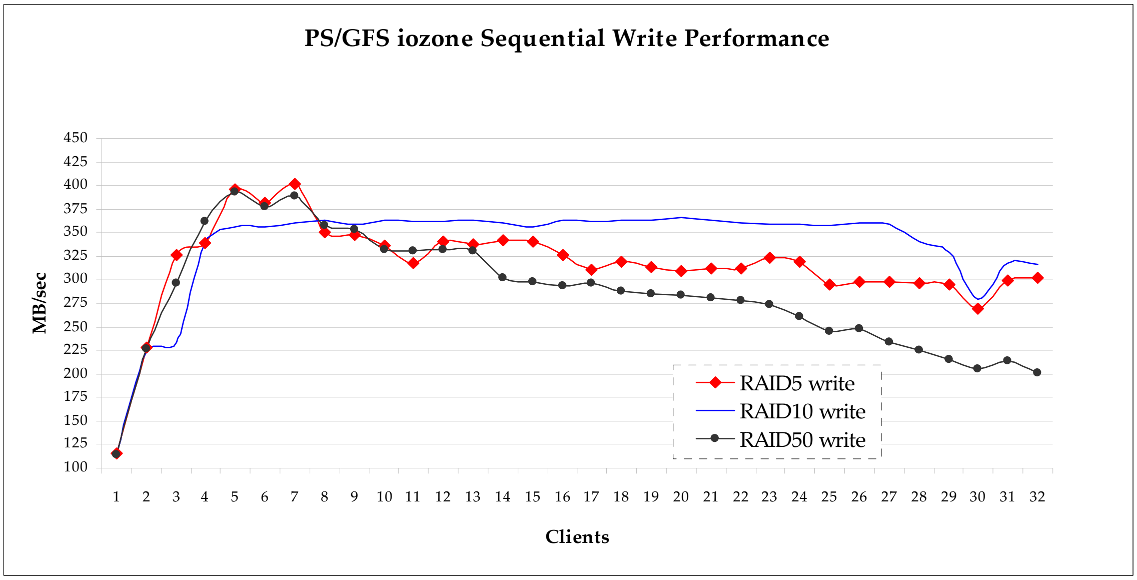

Figure 3 - Sequential iozone Write Performance to a Two Member Group

The results of this test indicate that the clients received a maximum sustained throughput of approximately 400 MB/s across all RAID levels. RAID 10 sustained the highest average write performance, although the performance of all three policies degraded as the number of clients increased. This performance decrease was due to the overhead imposed by the cluster file locking and increased communication. RAID 5 write performance approached RAID 10 write performance beyond 28 clients.

The second test compared sequential read performance in a two member group across RAID policies as the client count scaled from 1 to 32. The read performance effectively mirrored the write performance data. RAID 50 and RAID 5 both scaled well, peaking near 600 MB/s after 15 clients and sustaining throughput of approximately 550 MB/s as additional clients were added.

RAID 10 read performance never reached this level and began to decline as the client count exceeded 20. Read performance is not burdened with file locking overhead. It therefore scaled much better than write performance and nearly doubled write performance in average throughput. The results of the second test are depicted in Figure 4.

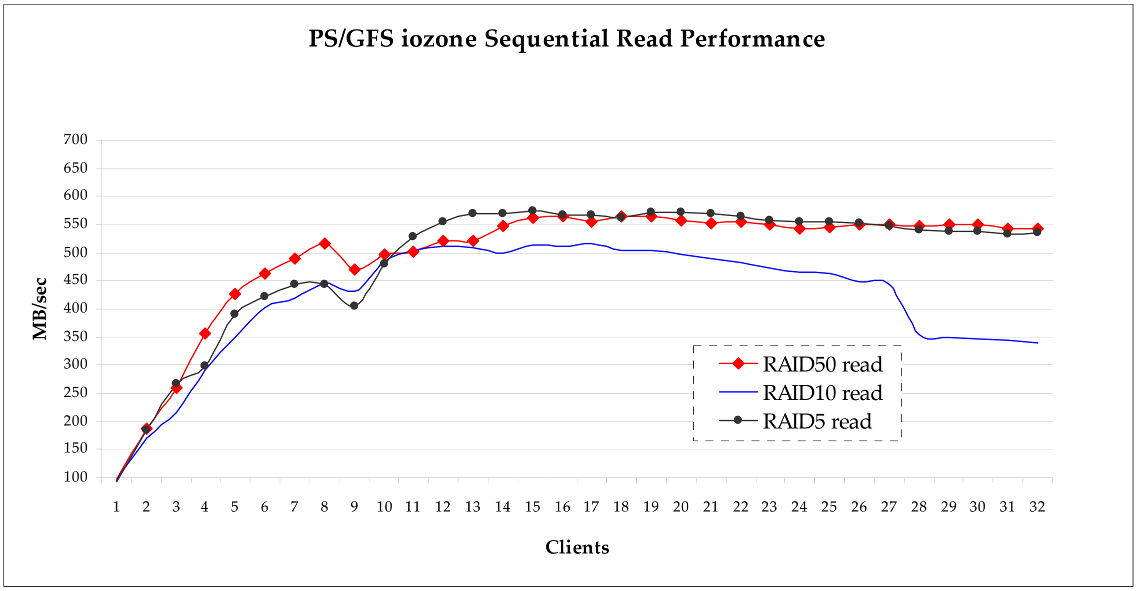

Figure 4 - Sequential iozone Read Performance to a Two Member Group

The third and fourth tests measured random write and read operations across all three RAID levels in a two member group. Random I/O eliminates many of the efficiencies that can be exploited in sequential I/O operations such as queuing and pre-fetching. The principal determinant of random I/O performance is the rotational speed of the disk and average seek time, which is a function of the data placement on the spindles. Therefore, random I/O performance is generally worse than sequential I/O when measured in MB/s. Random read and write performance data are depicted in Figure 5 and Figure 6.

Per Figure 5, random I/O performance tended to flatten as the number of clients increased beyond eight. RAID 10 had the highest average random write performance, but RAID 5 and RAID 50 tracked RAID 10 performance very closely. This trend is due to the workload characteristics. The random I/O file sizes were large, particularly relative to the controller cache size. The performance difference between RAID types on smaller sized file operations is greater with RAID 10 showing a significant advantage. This makes RAID 10 a good fit for write heavy transactional workloads with small file and record sizes, such as certain databases.

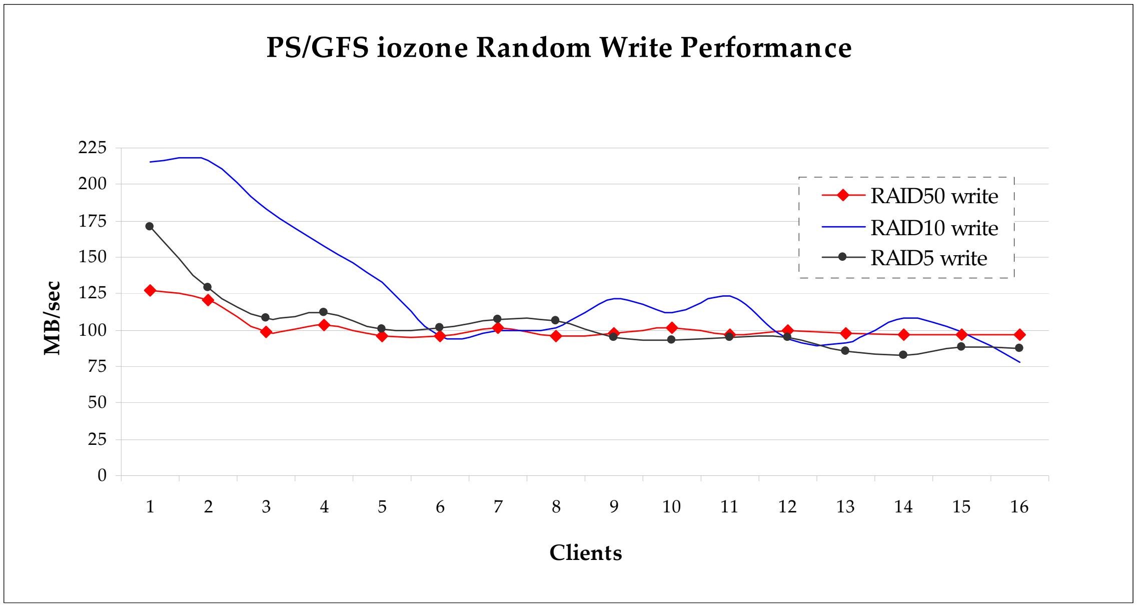

Figure 5 - Random iozone Write Performance to a Two Member group

The random read performance was also very close between RAID types. All RAID types scaled well as the client count increased with RAID 50 showing the best scalability.

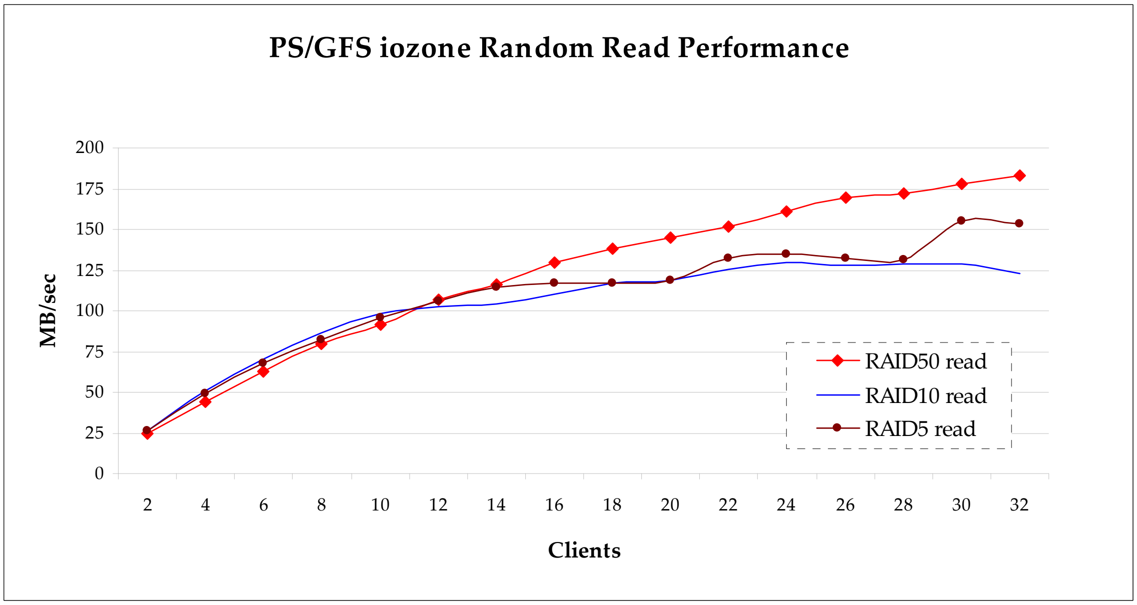

Figure 6 - Random iozone Read Performance to a Two Member Group

The preceding tests used the following iozone parameters: "**–r 64K –s 8G –c –e –w -+n**". The GFS file system was created and mounted without performance tuning parameters.

In summary, the underlying member RAID type is an important design consideration. The following conclusions can be drawn based on the results of these performance tests:

- RAID 5 and RAID 50 showed near linear scalability and sustained sequential read performance as the client count increased. Either option is recommended for read heavy (> 70%) environments.
- RAID 10 showed best write performance for both sequential and random workloads. The performance differences tended to decrease as the client count increased, making RAID 10 the preferred RAID type for write dominant environments with low client counts.
- RAID 5 tracked the leaders' performance and scalability in all four categories, making it the preferred choice for environments with mixed I/O patterns.
- RAID 10 and 50 offer superior data protection than RAID 5 for mission critical applications while RAID 5 offers highest capacity.

Overhead

The next test quantified the overhead imposed by the topology and the file system on absolute performance. All clients access the PS Series group via a single group IP address. The owner of that IP address redirects requests for blocks it does not own to the block owning member. The data are returned via that member. Therefore, there is some overhead associated with the member synchronization. Additionally, GFS imposes additional overhead during write operations due to file locking operations that ensure data consistency.

This test compared the sequential I/O performance of 32 clients accessing 32 separate volumes (16 per PS Series group member) as raw devices using O_DIRECT file operations to 32 requests to a single GFS volume. Taken together, these data quantify the overhead imposed by array communication versus file system management. Results are depicted in Figure 7.

The dashed line in Figure 7 represents the theoretical performance of the PS Series groups if all front-end Gigabit Ethernet ports are fully utilized. GFS sequential read performance exceeded raw I/O performance and trended upward as the client count scaled until 18 clients. At this point raw I/O performance continued to trend upward while the GFS read performance trended downward. This downward trend is attributed to the inter-member coordination as GFS imposes no overhead during reads. Dividing the achieved performance at all data points by the theoretical performance and averaging the results yields the array efficiency. 100% efficiency is a theoretical limit and therefore not attainable. By this formula, efficiency means the average percentage of theoretical peak performance yielded by the operation. GFS read achieved 71% efficiency and direct I/O achieved 73%. From a statistical standpoint, the performance was identical.

GFS write performance outperformed direct I/O write performance for less than 16 clients. With 16 clients, direct I/O performance trended upward while GFS write performance began a shallow decline. GFS sequential write performance showed 6% better efficiency than direct I/O overall by a margin of 6% in spite of the fact that the average write efficiency for client counts of 16-32 favored direct I/O by 7%. This indicates that the file locking operations and journaling overhead introduced by GFS impacted write performance by less than 10%.

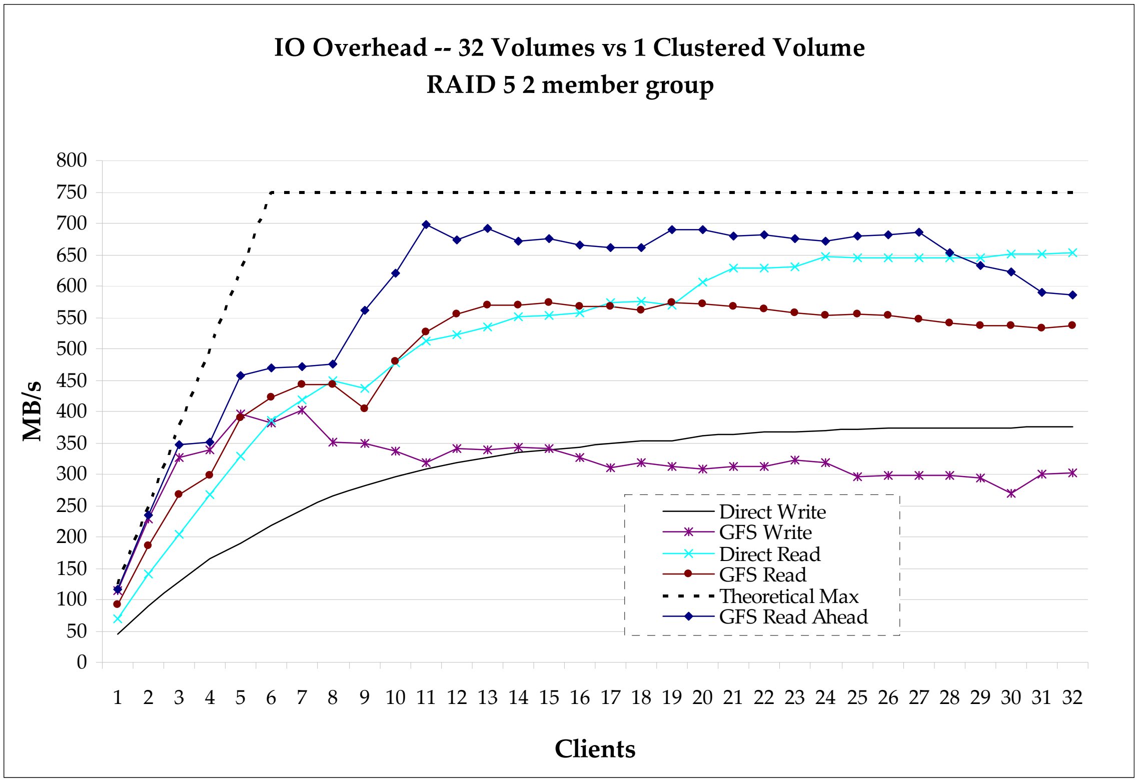

Figure 7 - Sequential iozone Performance: 32 Volumes versus 1 Volume

A final note about this study is that all performance measurements were achieved without file system or operating system performance tuning. The study should accurately reflect PS Series storage GFS performance out of the box. Higher performance in certain tests can be achieved through performance tuning. In Figure 7 above, the top most solid line represents sequential read performance of RAID 5 with GFS after setting the kernel readahead parameter for the block device to 16384. Notice that sequential read performance with this parameter peaked at 700 MB/s and sustained the highest read performance measured until 28 clients. The average efficiency with this operating system prefetching parameter was 85%, a 14% improvement over untuned sequential read performance.

The iozone parameters used in the direct I/O study were **"-c -I -e -w -+n"**. The-I switch opens files with the O_DIRECTIO flag to bypass all caching.

Block device pre-fetching was set with **"blockdev –setra 16384"** on the logical volume.

Scalability

Although this study only tested performance on one and two member groups, PS Series groups can scale to twelve members. Measuring the performance scaling as group membership increases from one to two arrays provides data for projecting scalability beyond two members. Figure 8 depicts the sequential I/O performance of a two member group normalized against single member performance.

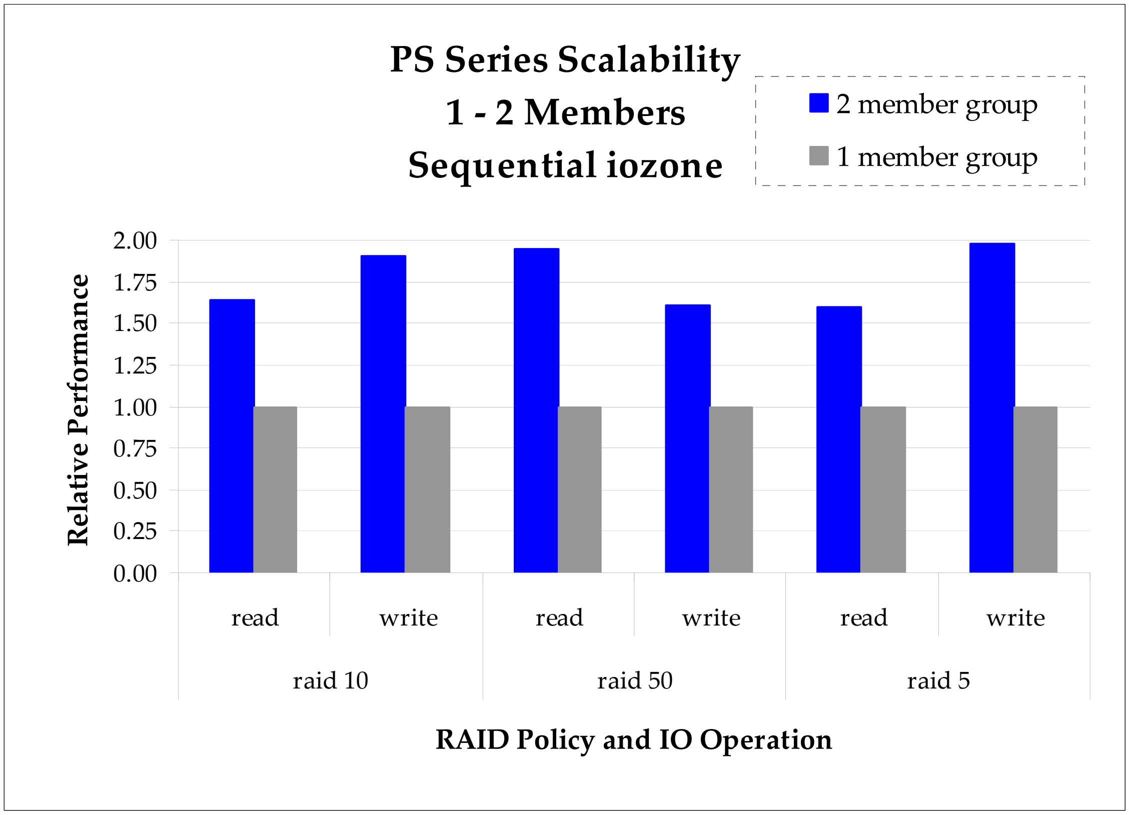

Figure 8 - One to Two Member Relative Performance

For RAID 10 members, write performance scales by approximately 80%. For RAID 5 and 50 both read and write performance scale by more than 70%. In all cases, two member performances improved by at least 50-85%. These scalability numbers were achieved without additional operating system, kernel, or file system tuning parameters.

Conclusion

Dell PS Series storage and GFS combine standards-based hardware with open source software to provide scalable I/O performance for small clusters. Although this paper is primarily geared toward a scientific computing environment, the performance results presented in this study are not HPC specific. After reviewing best practices for deploying and configuring GFS on PS Series storage, the paper described the results of performance and scalability studies as the group expanded from one to two members.

RAID 10 offered the best write performance for both random and sequential tests. The write performance difference between RAID 5 and RAID 10 diminished as the client count increased. RAID 10 also offers increased data protection over RAID 5 at the cost of reduced capacity. Therefore, RAID 10 is the preferred configuration for write-heavy environments -- particularly at lower client counts or where increased data protection is necessary. The optimal client to member ratio for read-heavy environments is 14 to 1.

RAID 5 and RAID 50 showed similar performance across random write and sequential read workloads. RAID 50 performed marginally better than RAID 5 on random reads while RAID 5 outperformed RAID 50 on sequential write performance after the client count scaled above 16. RAID 5 is therefore recommended for environments with mixed I/O patterns, particularly where large disk capacity is required. RAID 50 is appropriate for read-heavy environments (> 70%) where data protection is a primary concern. In either case, the recommended client-to-member ratio for RAID 5 and RAID 50 groups is 16 to 1. Furthermore, in exclusively read-heavy environments, performance can be improved significantly by increasing readahead on the block device.

The results of the study show that PS Series storage sequential performance scaled by approximately 75% on average including nearly linear sequential read scalability with RAID 5 and RAID 50 volumes. Both GFS and PS Series storage scaled well as arrays were added to the group. Read performance scaled better than write due to the overhead inherent in the storage architecture and the additional overhead associated with file locking in a clustered file system. However, the slight write performance penalty is offset by the additional features GFS provides including simultaneous access to a single file system name space from multiple clients, fault tolerance, online file system expansion, and journaling for quick recovery. Although Red Hat supports up to 128 GFS nodes attached to a single file system, Dell HPC Engineering does not recommend more than 64 nodes per PS/GFS iSCSI SAN until further testing can be performed.

In summary, the cost of implementing and managing a parallel file system can quickly erode the favorable economics of clusters and divert investment from the nodes actually performing computation to auxiliary expenses such as file servers. On the other hand, distributed file systems such as NFS have proven insufficient to meet escalating I/O demands. Dell PS Series storage and GFS combine standards-based hardware with open source software to provide scalable I/O performance for small clusters.

Appendix A

This is an example host configuration script that can be integrated into the OCS installer to automate the tasks associated with configuring an iSCSI interface and configuring the Dell Baseboard Management Controller (BMC) as a Red Hat Cluster Service fence device.

```
#!/bin/bash

# Disable unused services
for i in acpid auditd avahi-daemon bluetooth cpuspeed cups gmond lava lm_sensors
lvm2-monitor mcstrans mdmonitor ntpd openibd pcscd rhnsd sendmail smartd yum-
updatesd; do chkconfig $i off; done

# Enable other services
for i in iscsi ipmi; do chkconfig $i on; done

# Configure iscsi interface
service iscsi start
iscsiadm -m iface -I iface1 --op=new
iscsiadm -m iface -I iface1 --op=update -n iface.net_ifacename -v eth1
iscsiadm -m discovery -t st -p 10.40.40.110 -I iface1

# Configure BMC interface
LAST=$(ifconfig eth0 | awk ' /inet addr/ { print $2 } ' | cut -f4 -d.)
service ipmi start
ipmitool lan set 1 ipsrc static
ipmitool lan set 1 ipaddr 172.20.100.$LAST
ipmitool lan set 1 netmask 255.255.0.0

# Configure BMC remote user
ipmitool lan set 1 access on
ipmitool user set name 2 admin
ipmitool user set password 2 password
ipmitool channel setaccess 1 2 callin=on ipmi=on link=on privilege=4
ipmitool user enable 2

# Configure MTU for eth1
echo "MTU=9000" >> /etc/sysconfig/network-scripts/icfg-eth1
ifconfig eth1 mtu 9000
```

Appendix B

This is an example cluster.conf configuration file that mirrors the file used in this study. The only difference between this file and the file used in the study is the number of nodes, 10 versus 32.

```
<?xml version="1.0"?>
<cluster name="ps00" config_version="1">
 <clusternodes>
  <clusternode name="compute-00-00" nodeid="1">
        <fence>
                <method name="1">
                        <device name="c00-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-01" nodeid="2">
        <fence>
                <method name="1">
                        <device name="c01-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-02" nodeid="3">
        <fence>
                <method name="1">
                        <device name="c02-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-03" nodeid="4">
        <fence>
                <method name="1">
                        <device name="c03-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-04" nodeid="5">
        <fence>
                <method name="1">
                        <device name="c04-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-05" nodeid="6">
        <fence>
                <method name="1">
                        <device name="c05-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-06" nodeid="7">
        <fence>
                <method name="1">
                        <device name="c06-ipmi"/>
                </method>
        </fence>
  </clusternode>
  <clusternode name="compute-00-07" nodeid="8">
        <fence>
                <method name="1">
                        <device name="c07-ipmi"/>
                </method>
        </fence>
  </clusternode>
 </clusternodes>
 <fencedevices>
```

```
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.2" login="admin"
name="c00-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.3" login="admin"
name="c01-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.4" login="admin"
name="c02-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.5" login="admin"
name="c03-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.6" login="admin"
name="c04-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.7" login="admin"
name="c05-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.8" login="admin"
name="c06-ipmi" passwd="password"/>
        <fencedevice agent="fence_ipmilan" ipaddr="172.20.100.9" login="admin"
name="c07-ipmi" passwd="password"/>
 </fencedevices>
</cluster>
```

Appendix C

This is a summary of the commands used to create the cluster file system. All commands are executed from the head node in an OCS cluster.

1. Set up iSCSI sessions.

```
[root@li ~]# pdsh –a service iscsi start

[root@li ~]# pdsh –a chkconfig iscsi on

[root@li ~]# pdsh –a iscsiadm -m iface -I iface1 --op=new

[root@li ~]# pdsh –a iscsiadm -m iface -I iface1 --op=update –n
iface.net_ifacename -v eth1

[root@li ~]# pdsh –a iscsiadm -m discovery -t st -p 10.40.40.110 -I iface1

[root@li ~]# iscsiadm -m node -L all
```

2. Configure the disks. These commands assume a two array group with one volume per member.

```
[root@li ~]# ssh compute-00-00 parted -s /dev/sdb mklabel gpt

[root@li ~]# ssh compute-00-00 ‘parted -s /dev/sdb "mkpart primary 0 -1"’

[root@li ~]# ssh compute-00-00 parted -s /dev/sdc mklabel gpt

[root@li ~]# ssh compute-00-00 ‘parted -s /dev/sdc "mkpart primary 0 -1"’

[root@li ~]# pdsh -a partprobe –s
```

3. Configure the cluster services.

```
[root@li ~]# pdsh -a cp /home/jacob/cluster.conf /etc/cluster/cluster.conf

[root@li ~]# pdsh -a modprobe gfs

[root@li ~]# pdsh -a modprobe dlm

[root@li ~]# pdsh -a modprobe lock_dlm

[root@li ~]# pdsh –a service cman start
```

4. Configure the logical volume.

```
[root@li ~]# pdsh -a lvmconfig –enable-cluster

[root@li ~]# pdsh -a service clvmd start

[root@li ~]# ssh compute-00-00 pvcreate /dev/sdb1 /dev/sdc1

[root@li ~]# pdsh -a pvscan

[root@li ~]# ssh compute-00-00 vgcreate ps00 /dev/sdb1 /dev/sdc1

[root@li ~]# pdsh -a vgscan

[root@li ~]# ssh compute-00-00 lvcreate ps00 -l 100%FREE -i 2
```

```
[root@li ~]# pdsh -a lvscan
```

5. Create and mount the GFS file system on all nodes.

```
[root@li ~]# pdsh –a mkdir /ps00

[root@li ~]# ssh compute-00-00 mkfs -t gfs -p lock_dlm -t ps00:test -j 64
/dev/ps00/lvol0

[root@li ~]# pdsh -a mount -t gfs /dev/ps00/lvol0 /ps00
```