DELL™ PowerVault™ MD1200 Performance as a Network File System (NFS) Backend Storage Solution

Aziz Gulbeden

Dell HPC Engineering Team

June 2010

Contents

Introduction

Network file system (NFS) provides an easy, standard, and open shared file system solution, and is widely used in the High Performance Computing (HPC) field. It is included in virtually every Linux distribution, and is designed to work with various operating systems including Microsoft® Windows®. It provides a robust file system solution for clusters whose I/O requirements can be met by a single server.

The goal of this white paper is to evaluate the performance of the Dell PowerVault MD1200 storage arrays when they are used as backend NFS storage over 10 Gigabit Ethernet in an HPC environment, and to determine what kind of workloads benefit from this solution. The PV MD1200 arrays offer an easily-deployable and manageable direct-attached storage solution, and their high performance make them a great fit to be used as a part of HPC clusters. An additional goal of the paper is to investigate the effects of different configuration settings on storage performance, and to come up with best practices for setting up a similar storage environment by presenting basic performance tuning guidelines.

Overview of the Hardware and Software Setup

PV MD1200/1220 Series Storage

The Dell PowerVault (PV) MD1200/1220 arrays provide a simple way to expand PowerEdge™ servers to connect directly to high performance storage; both the MD1200 and MD1220 models use 6 Gigabit SAS technology that supports RAID levels 0, 1, 5, 6, 10, 50 and 60.

The PV MD1200 array can host up to twelve 3.5" hard drives, and the PV MD1220 can host up to twenty-four 2.5" hard drives. Solid-state drives, self-encrypted drives and chaining a mixture of MD1200/1220 enclosures is supported to connect differently configured arrays to the same server in order to tailor the storage to meet the capacity and performance requirements; up to four enclosures can be connected to each other using daisy chaining.

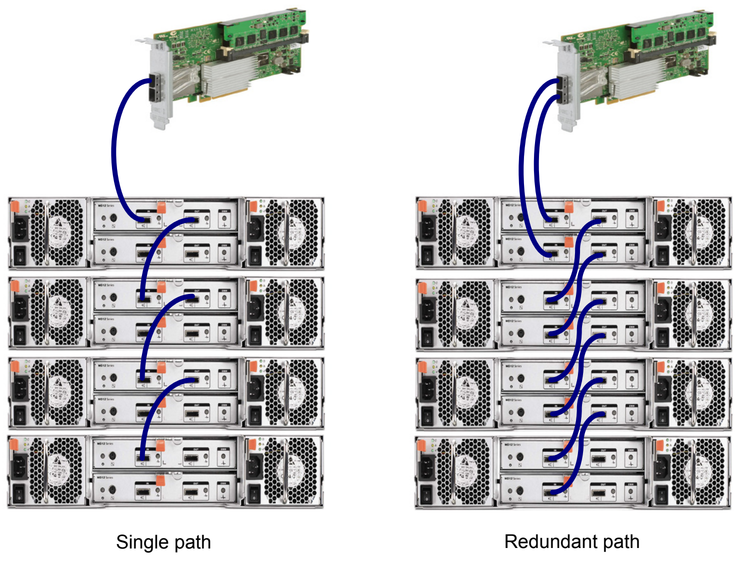
On the host side, a PERC H800 Host-RAID adapter is required to connect to the storage. The PERC H800 adapter has two ports that allow a maximum of eight arrays to be connected to a single adapter. The PERC H800 card supports redundant paths with I/O load balancing. It is possible to connect a maximum of 192 disks to a single PERC H800 card using eight MD1220 enclosures.

The MD 1200/1220 storage arrays can be daisy chained in two ways:

- *Single path*
- *Redundant path*

In a unified mode configuration, the enclosure is connected to one PERC H800 host-RAID adapter and can be one of up to four enclosures daisy-chained to a single PERC H800 port on the host server. The enclosure can also be connected to the same PERC H800 in a redundant path mode with two connections. These two different daisy chaining schemes are summarized in Figure 1, with the *Single Path* configuration on the left and the *Redundant Path* configuration on the right.

Figure 1. Daisy Chain Options for MD1200 Arrays



Another array connection method is to put it into what is called split mode that allows two servers to be connected to the same array. Each server identifies the enclosure as an independent device, with half of the total enclosure slot capacity. The split mode does not allow daisy chaining of the enclosures. For the purposes of this paper, the split mode was not utilized.

Setting up NFS

Network File System (NFS) is a standard package included in most Linux distributions. It allows sharing a file system over a network, giving the compute nodes (clients) access to the same data. Due to its simplicity and pervasiveness, NFS is widely used in the Linux space for HPC clusters since many applications and tools require a common file system for all compute nodes. Example uses of NFS by applications include storing the model information for a simulation application, storing genome sequence data, or storing the rendered movie scenes by the compute nodes.

There are a number of NFS configuration options present on the server and on the clients that mount the storage. On the server side, the `/etc/exports` file configures the NFS export directories. The `/etc/sysconfig/nfs` file specifies a number of NFS options. On the client side, the mount options are specified in the `/etc/fstab` file. The settings we have used in our cluster are described in the *Best Practices for Tuning NFS Server* section.

Benefits of using 10GbE Connected NFS Server with MD1200 Backend

In this white paper, the performance implications of using NFS storage for an HPC cluster that utilizes a 10 GbE connection on the NFS server side was studied. Below is a summary of the solution benefits:

- **Backend:** The MD 1200 provides reliable, high-performance server storage, and NFS provides a simple and high-performance method to share this storage among the HPC compute nodes. Together, the MD 1200 and NFS provide complete, easy-to-setup, dependable cluster storage.
- **Frontend:** Since the NFS server is a single machine, its network and storage bandwidth is quickly consumed by the compute nodes; therefore using a faster network interconnect has a direct impact on the overall performance. With high-speed interconnects becoming more

widespread, it is recommended to install a faster network card, such as a 10 Gigabit Ethernet (10 GbE) card, on the NFS server even for smaller clusters. For example, the Dell PowerConnect™ 6248 switch has forty-eight 1 Gigabit Ethernet ports and can accommodate up to two 10 Gigabit aggregation modules that can contain up to four 10 Gigabit Ethernet connections to a single switch (i.e. two 2-port modules).

- **Balanced Design:** 10 Gigabit Ethernet and MD1200 performance are comparable, allowing a balanced storage system to be designed. Depending on the workload, and the number of disks in the volume, the storage or the network may be a bottleneck; however in general the performance of each component is in a similar range. The performance results of the sequential tests below demonstrate that utilizing a GigE link (125MB/s theoretical bandwidth), causes the front-end network to be a bottleneck.
- **Simplicity:** 10 Gigabit Ethernet interface is a cost-effective, standard network interface with an IP address; no extra effort is required to use 10 GbE for the NFS protocol since the standard IP protocol over Ethernet is used.

Benchmarking Cluster Setup

The cluster setup used a dedicated NFS node that is recommended for larger numbers of compute nodes; however, for smaller clusters the NFS server and the head node could be the same. This configuration has little or no performance impact on the test conducted in this white paper. However, there are other configuration considerations that are beyond the scope of this paper.

A Dell PowerEdge R710 server running Red Hat Enterprise Linux® 5 U4 with kernel 2.6.18-164.9.1.el5 is used as the NFS server for this study. The NFS server has two Intel® E5540 2.53 GHz processors and 24 GB RAM. It uses an Intel 10GbE-SR PCI-E Network adapter with the ixgbe v2.0.8-k3 driver to connect to a Dell PowerConnect 6248 network switch with 10GigE modules.

A 32 node cluster was used as the compute nodes for testing; the compute nodes are Dell PowerEdge 1950 servers and run Red Hat Enterprise Linux 5 U1 that is installed using Platform OCS 5.1. Each compute node has two Intel E5450 3.0 GHz processors and 4 GB RAM. The onboard Broadcom 5708 Gigabit Ethernet interface with bnx2 v1.7.9-1 driver is used to connect to the cluster switch network.

Figure 2 is a cluster setup diagram. The onboard NIC 1 of each compute node as well as the cluster head node is connected to one of the GigE ports on the PowerConnect6248 switch. The 10 GigE NIC on the R710 NFS server is also connected directly to one of the 10GigE port on that switch.

Figure 2. Cluster Setup

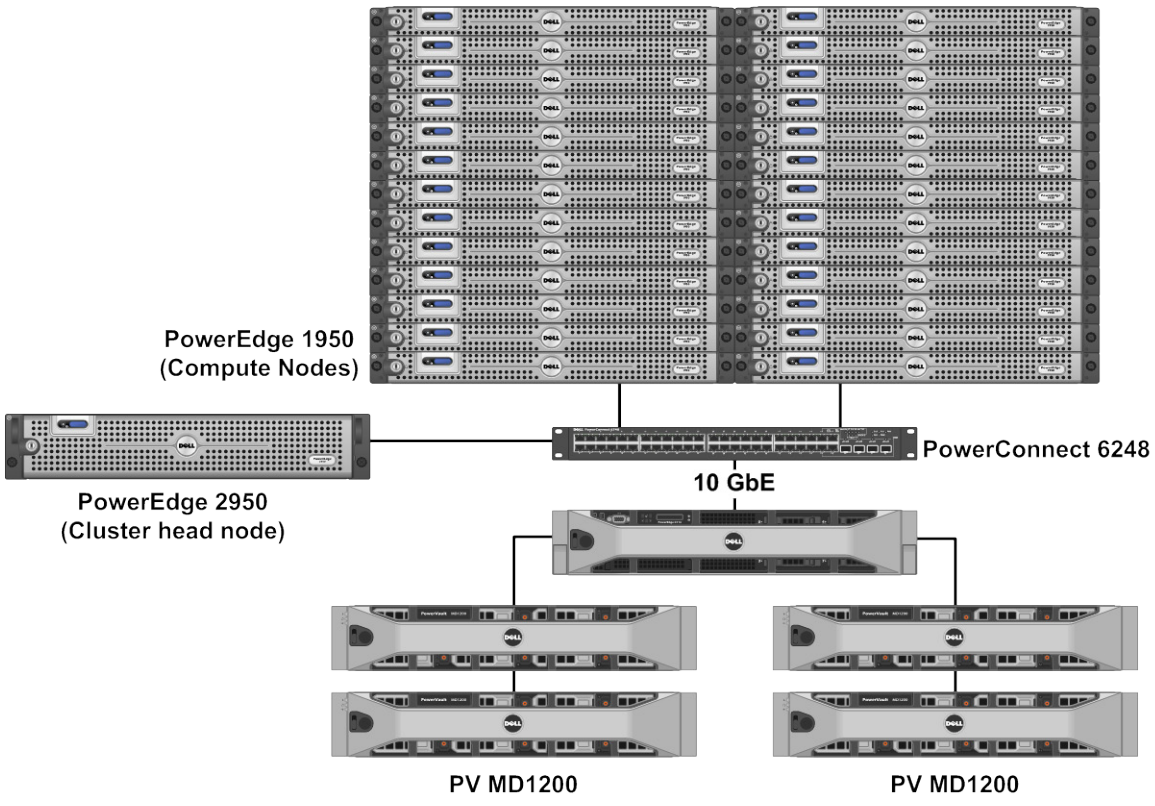
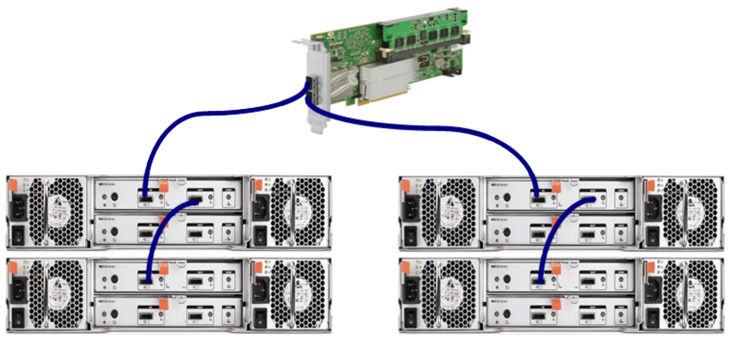


Figure 3. Connecting Four MD1200 Arrays



On the backend, the NFS server is connected to four Dell PowerVault MD 1200 RAID arrays. After running a number of tests, it was decided to connect two arrays to both ports of the H800 card as shown in Figure 3 and these experiments are summarized in the Direct Attach Performance Study section.

Each array contains twelve 450 GB 15K rpm SAS6 disks. The storage is configured as a single RAID 50 volume spanning over 48 disks. The RAID 50 span size is set to 12 in order to have one parity disk per array; increasing this number will reduce the number of parity disks that might improve the performance at the expense of reduced redundancy. The volume is formatted as ext3 using the default values.

Table 1. Cluster Setup Details

NFS Server Configuration:

Processor	Two Intel E5540 2.53 GHz quad core processors
Memory	24 GB (six 4 GB 1067MHz DDR3-Registered DIMMs)
OS	Red Hat Enterprise Linux 5 U4 with kernel 2.6.18-164.9.1.el5
PERC H800 Firmware	12.0.1-0083
Storage Driver Version	00.00.04.08-RH2
BIOS	1.2.6
10 GB Ethernet Adapter	Intel 10 GBE-SR AF Server Adapter
10 Gbe Driver	ixgbe version 2.0.8-k3

Compute Node configuration

Processor	Two Intel E5450 3 GHz quad core processors
Memory	4 GB
OS	Red Hat Enterprise Linux 5 U1
BIOS	2.6.1
GigE Driver	bnx2 version 1.7.9-1

Best Practices for Tuning the NFS Server

The first performance tuning action performed on the cluster was to enable jumbo frames on the switch and all the compute nodes, so that maximum network message size is up to 9000 bytes rather than the default 1500. On the nodes, this setting is specified in the `ifconfig` files by adding line `MTU=9000` to the `ifcfg-eth#` file in the `/etc/sysconfig/network-scripts` directory, where `eth#` is the Ethernet interface the node uses to communicate with the NFS server. On the network switch the size is set through the serial terminal; see Appendix A for more information on this procedure.

Configuring the NFS export directory to be asynchronous is another important setting that improves NFS server performance. When `async` is specified in the `/etc/exports` file, the NFS server can let a file system write operation return without actually committing the data to the disk. Instead of immediately committing every write to the disk, the NFS server keeps the operation in its cache to gather small NFS write operations into a single efficient disk write operation. On the other hand, it should be noted that using the async mode introduces a risk to data integrity in case the data in the cache is lost due to a failure.

Another important tuning option is the number of NFS daemons. The number of NFS daemons is specified in the `/etc/sysconfig/nfs` file by the `RPCNFSDCOUNT` parameter. This number was set to 128 in order to have enough threads for the compute nodes. For the sequential workloads, increasing the thread count generally increases the read performance while decreasing the write performance due to the increased randomness seen by the storage. A number of tests were run on the cluster, with various numbers of threads starting with 8 and going up to 512 threads; 128 threads was found to be a good compromise setting with reasonable sequential read and write performance for 32 clients. For more information on this setting, see the section *Performance Effect of Number of NFS Threads*.

On the compute nodes, NFS is mounted using the following options:

- `defaults`
- `tcp`
- `intr`
- `noatime`
- `hard`

On a cluster installed with Platform Cluster Manager (PCM), these settings can be specified on the head node by editing the `/etc/cfm/compute-rhel-5-x86_64/etc/fsstab.append` file, and distributed to the compute nodes using the `cfmsync –f` command. The `cfmsync` command in PCM pushes any configuration file modifications on the head node to all compute nodes. After executing this command, the changes to `fsstab.append` on the head node will be reflected in the `/etc/fstab` file on all compute nodes. The same task can be completed by manually editing the `fsstab` file and copying it to all the compute nodes using a parallel shell command; the list of command options is explained in the next paragraph.

The first option `defaults` uses the default options for the distribution; for example the default block size of 32 Kb is used. The remaining options are described below:

- The network protocol used for NFS is `tcp`; it is generally used and recommended for NFS (see the *Red Hat Enterprise Linux Deployment Guide* [1]). For this configuration, `tcp` was used since the network setup did not have dropped packets, and the network cards and switches have specific optimizations to improve `tcp` performance.
- Specifying `intr` allows processes that are waiting for NFS server I/O to be interrupted, for example by sending an interrupt using a `Ctrl-C` command; this can be useful in a benchmarking environment.
- The `noatime` parameter disables file access time tracking to improve file system performance.
- Specifying the NFS mount as a `hard` mount causes the I/O processes from the NFS server to hang when it becomes unavailable, and then to resume when the server comes back online rather than returning an error; an error is returned when using a `soft` mount. This option should not impact performance, but will change the NFS client behavior when the NFS server is temporarily out of service; therefore the `hard` option is recommended. Only select the `soft` option if the return of an error is expected, even if the service resumes after the temporary failure.

For this paper, the other settings were left as their default values. Storage tuning was not performed, however it is possible to gain further performance benefits by modifying other OS and storage settings.

Aligning the partition with the storage block size is recommended [2]. When the file system blocks do not align with the storage blocks, RAID parity calculations can reduce the performance significantly. In the configuration described in this paper, the `parted` command was used with the units set to sectors to make sure that the partitions align by specifying their start to be a multiple of the storage stripe size.

While running the tests in a direct-attached storage configuration and where NFS is not being used, the read ahead setting that specifies how many sectors should be read in advance when a read request comes was modified. The read ahead algorithm adaptively turns itself off when it identifies that the requests are not sequential. For sequential workloads, increasing this parameter from its default value of 256 results in a big performance improvement. For these experiments, the `blockdev` command was used to increase the read ahead parameter on the OS to 8192 sectors to improve sequential read performance; i.e. `blockdev --setra 8192 /dev/sdb`, where `sdb` is the device the file system is created on. On the other hand, this parameter does not have a significant performance effect once the partition is exported using NFS due to NFS using its own read ahead algorithm.

Performance Studies

This section of the white paper describes the performance studies of both the NFS server and of the clients accessing the storage using NFS. The first section examines NFS server performance.

Direct Attach Performance Studies

Comparing Different PV MD1200 Connection Topologies

The first step in studying performance was to perform tests with storage directly attached to the server, without NFS, in order to architect the storage cluster correctly. The first study performed compared different the MD1200 connection topologies shown in Figures 1 and 2.

With four MD1200 arrays there are different ways to connect them to the PERC H800 adapter. The goal was to study the performance of three possible topologies under a workload where a fixed number of threads perform sequential write and read operations. The three topologies investigated were:

- **Topology 1 (single-path):** Four MD1200 arrays daisy chained using a single connection (See Figure 1, configuration on the left).
- **Topology 2 (redundant):** Four MD1200 arrays daisy chained using redundant connections (See Figure 1, configuration on the right).
- **Topology 3:** Two MD1200 arrays connected to each port of the PERC H800 controller using a single connection (See Figure 3).

The IOzone benchmark version 3.327 from www.iozone.org ran locally to spawn 64 threads performing sequential read and write operations on separate files. Each test was performed 10 times for each topology, and the performance results plotted in Figures 4 and 5 show the average aggregate throughput. The standard deviation is represented as error bars on the charts. Since the goal is to compare these three topologies, the charts report performance relative to Topology 1.

Figure 4. Sequential Write Performance

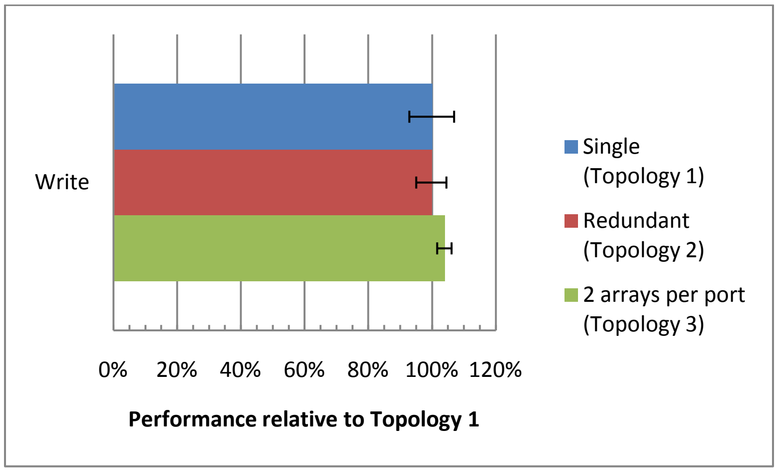
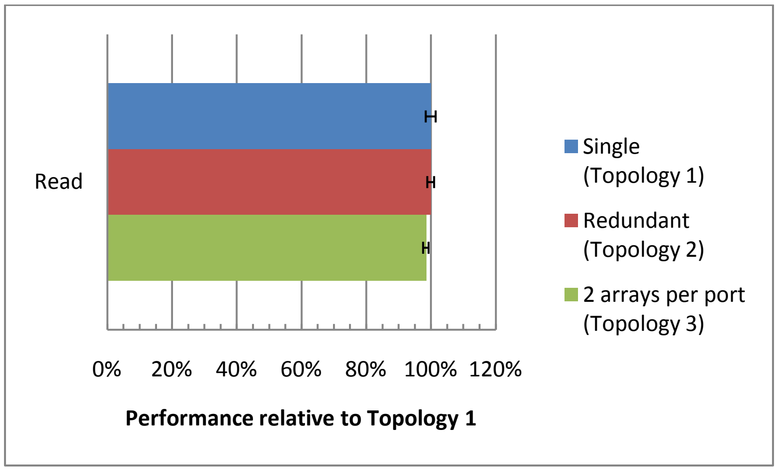


Figure 5. Sequential Read Performance



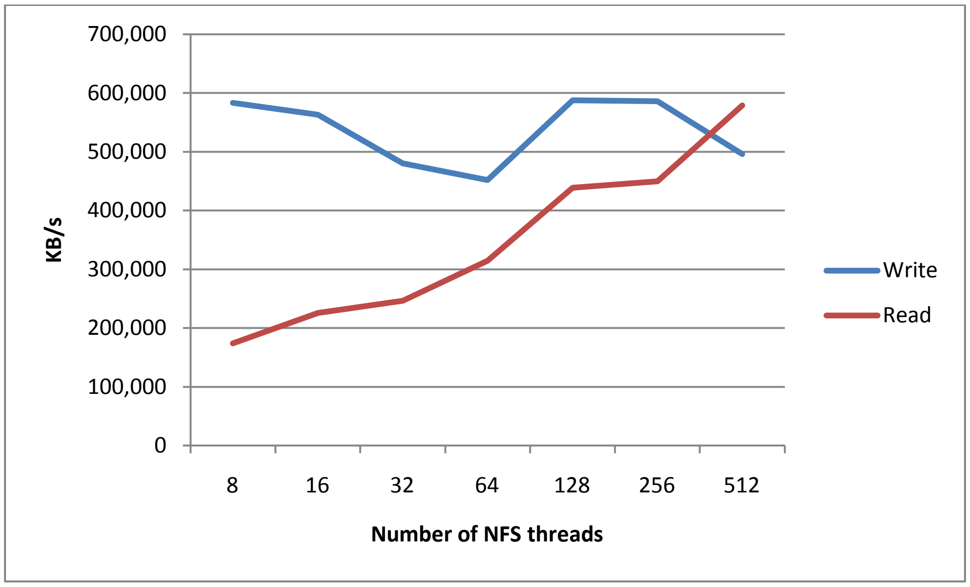
For this workload type, the third topology performed the write operations approximately 4% faster than the redundant or single connection configurations. The read performance for Topology 3, on the other hand, was approximately 1.5% lower. This experiment covers the situation where multiple threads perform sequential reads and writes only; therefore the results may not be applicable to other applications with different access patterns. Since the following NFS throughput tests will have a cluster of nodes performing sequential reads and writes, it was decided to use the third topology for the NFS performance study that uses the MD1200 backend storage.

NFS Performance Study

Number of NFS Threads

In order to find the optimal number of server side NFS threads, the sequential performance of 32 clients with various numbers of NFS threads must be studied. The number of clients, 32, was selected as a representative number that typically use a single NFS server. The 32 clients performed sequential writes and reads of 8 GB per client with 8 Kb block sizes with the results shown in Figure 6.

Figure 6. Throughput of 32 clients With Varying Number of NFS Threads



The graph illustrates the general trend that increasing the number of threads increases the read performance while reducing the write performance. One reason for the read performance improvement is that as the number of threads increases the amount of data “read ahead” increases, resulting in increased read performance for this type of workload. For write operations on the other hand, additional threads cause more access pattern randomness that decreases performance.

Based on these results for a cluster size of 32 clients, 128 and 256 threads give reasonable sequential performance. Consequently, the rest of the testing was performed with 128 NFS threads since 256 threads did not offer an increase in performance.

Throughput and IOPS Scalability

To measure throughput and the NFS server scalability, IOzone and IOR benchmarks were used. The IOzone benchmark is commonly used to measure performance, particularly throughput, but it can also be used for measuring IOPS. It can run on a cluster and report aggregate client (compute node) performance for different workloads. This testing used version 3.327 that is available from http://www.iozone.org.

IOR is another common benchmark used for measuring throughput performance. It is particularly useful for measuring performance of MPI-IO applications. This testing used IOR version 2.10.2, available from http://sourceforge.net/projects/ior-sio/. IOR uses MPI to synchronize parallel clients performing I/O; for the testing IOR was configured to use a single file for the reads and the writes in order to measure performance when all the clients write to and read from the same file. This is commonly referred to as the N-1 IO case, where N clients are writing to a single file. In contrast, IOzone was configured so that every client works on its own file, which is commonly referred to as the N-N IO case where there are N clients writing to N files.

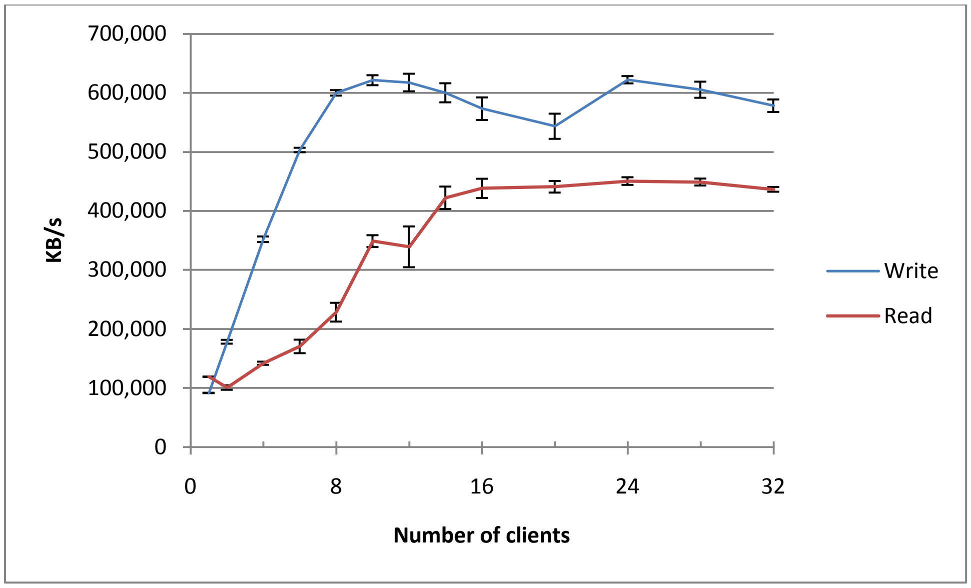
The following command was used to launch the benchmark:

```
./iozone -i0 -i1 -r 8k -s 8g -c -e -t <# of threads> -+m hosts -+n.
```

`-i0` specifies sequential write tests. `-i1` specifies sequential read tests. `-s 8g` indicates each thread writes/reads an 8GB file. `-c` is used to count include the time it takes to close the file. `-e` is used to flush the cache data. `-t` indicates the number of threads launched in this test.

Figure 7 shows the sequential read and write performance while running IOzone, where every client writes and reads an 8 GB file. The X-axis is the number of clients in the test, and the Y-axis is the resulting aggregate throughput performance in KB/s. The benchmark was run five times, and the average is plotted on the charts. The standard deviation is represented as error bars on the chart.

Figure 7. Sequential Read and Write Performance for Various Client Numbers



The experiment results show that the clients achieve around 600 MB/s sequential write and 450 MB/s sequential read performance.

Figure 8 shows the IOR N-to-1 performance in terms of I/O operations per second (IOPS), where a number of clients perform sequential read and write operations from different parts of a single file using 1kB block sizes. The block size of an ext3 file system is set to 1 KB, so a 1 KB block size setting for IOPS tests is expected to give best results. For IOPS measurements, the O_DIRECT mode in order to by-pass the cache and obtain storage performance without OS caching was used. O_DIRECT bypasses the client side cache and makes sure all requests are sent to the NFS server directly; unlike a throughput test, the IOPS test request size is relatively small (1 KB each). Without O_DIRECT, most requests could be served in client side cache and returned to benchmark in a speed almost equivalent to local memory accesses. Every client writes 8 GB and reads 8 GB, which is double the amount of memory on each of the clients ensuring that the results are outside client cache capability limits. The tests were run three times, and the charts display the mean value, and the standard deviation is represented as error bars.

Below is a list of commands used to run the benchmarks:

Write: `mpirun -np <# of clients> -machinefile hostfile IOR -a POSIX -i 3 -d 16 -r -E -k -B -o /nfs/test -s 8 -b 1G -t 1k`

Read: `mpirun -np <# of clients> -machinefile hostfile IOR -a POSIX -i 3 -d 16 -r -E -k -B -o /nfs/test -s 8 -b 1G -t 1k`

`-a POSIX` indicates to use POSIX interface. `-B` is used for O_DIRECT. `-t 1k` specifies that the request size is 1 KB. `-s 8` and `-b 1G` are combined together to specify a stride access of 8 sections with 1 GB each

Figure 8. IOR Single File IOPS Performance

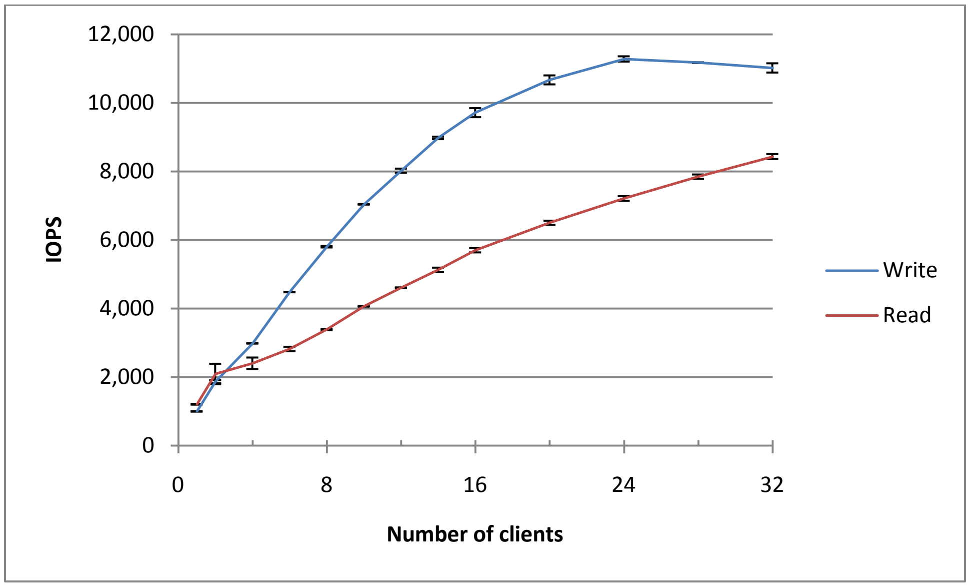


Figure 9 shows the throughput for IOR N-to-1 performance in terms of Megabytes per second. The block size is set to 1 MB, and the clients perform read and write operations to a single file stored on the NFS share.

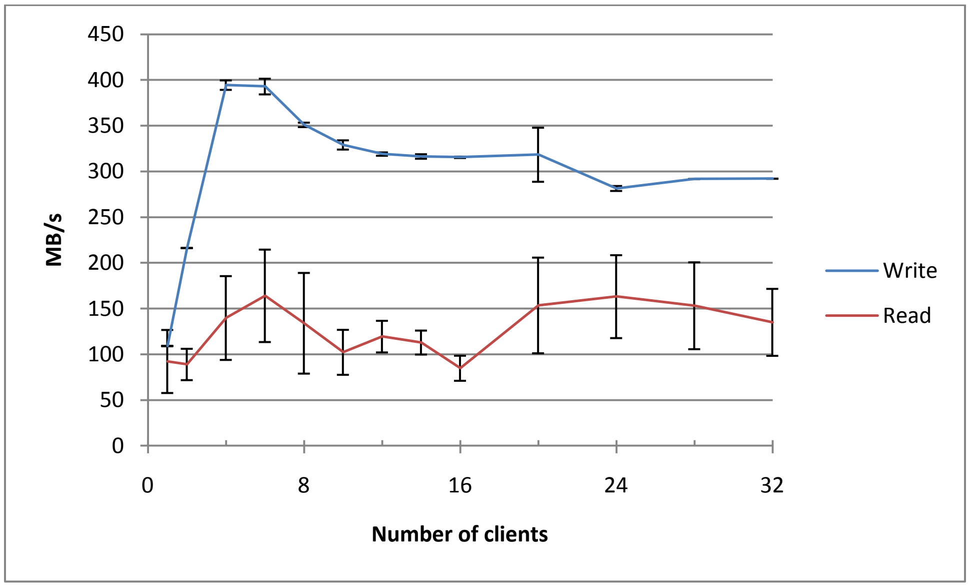
Below is the list of commands we used to run the benchmarks:

Write: `mpirun -np <# of clients> -machinefile hosts IOR -a POSIX -i 3 -d 16 -w -E -k -o /nfs/test -s 8 -b 1G -t 1m`

Read: `mpirun –np <# of clients> -machinefile hosts IOR -a POSIX -i 3 -d 16 -r -E -k`

`-t 1m` here indicates that the request size is increased to 1 MB for the throughput tests.

Figure 9. IOR single file throughput



When the clients work in the same directory, the performance is reduced since more synchronization takes place. The maximum write performance achieved was around 400 MB/s with 4 and 6 clients. The read performance was around 150 MB/s with a high variance. NFS is more efficient when there is little synchronization required, since it uses a stateless protocol in order to reduce the server load. When the clients work on the same file at the same time, the stateless protocol is no longer enough to keep the data consistent on multiple distributed client caches. One way of guaranteeing data consistency in this scenario, is to disable client metadata caches. However, a large performance penalty is to be expected by forcing clients to access the NFS server for each and every metadata operation. That is the reason for the large performance gap seen when clients work on the same file versus working on different files.

In all of the tests, the write performance was higher than the read performance. This is due the multiple cache levels that exist in the system (e.g. at the client side, at the NFS server OS and hardware level), therefore write operations can be combined together and performed more efficiently. Reads outperformed writes where the read ahead algorithms, which are present at various levels, successfully cache the soon to be read data in advance, which was the case for direct attach performance studies. However, as explained in section “Best Practice for tuning the NFS server”, the parameter used to increase the read ahead cache size for direct-attached storage does not have a significant effect on the NFS read performance due to NFS using its own read ahead algorithm. One of the practical ways observed to increase NFS read ahead caches is to increase the number of NFS server threads. Figure 6 clearly indicates that with more NFS threads, the read throughput is higher. A major reason for this is that increasing the number of NFS threads also increases the size of the NFS read ahead cache, and at a certain point, the read throughput matches the write. On the other hand, Figure 6 also indicates that a large number of NFS threads may have a negative impact on write operations. It is recommended to select a reasonable number of NFS threads based on workloads types.

Metadata Performance

In order to test the metadata performance of the cluster, the mdtest benchmark was used. This benchmark measures the performance of “create”, “stat” and “remove” operations that are common metadata operations. Version 1.7.4 of mdtest is available from http://sourceforge.net/projects/mdtest/. The benchmark was modified to measure "utime" performance as well, since this operation was not included in the original benchmark. In order to ensure consistency and to avoid caching effects, the NFS export options were modified by replacing "async" with "sync" for the NFS share. In addition, the clients mounted the file system with an additional parameter of "noac" that disables attribute caching. The "noac" parameter is required for consistency when multiple clients are working in the same directory.

Figures 10, 11, 12 and 13 show metadata performance as a function of the number of compute nodes. Figure 10 shows the performance of the "create" operation, Figure 11 shows the performance of the "stat" operation, Figure 12 shows the performance of the "utime" operation, and Figure 13 shows the performance of the "remove" operation. While running the first two series of tests ("create" and "stat") the clients have unique working directories; for the last series of tests ("utime" and "remove") that are N-to-1 operations, the clients work in the same directory. The X-axis is the number of clients used, and the Y-axis shows the aggregate number of operations performed per second. The N-to-1 case is not included for the remove operation in Figure 13, since the clients will try to remove the same file and it can only be removed once when they are all working in the same directory. Some of the small benchmark cases completed quickly, and their performance data were reported as infinity by the benchmark, therefore these cases are not included in the output.

The following commands were used to launch the tests:

```
mpirun -np <# of clients> -machinefile hosts mdtest -d $dir -n $n -i 200 -y
-N 10 -t -u
```

```
mpirun -np <# of clients> -machinefile hosts mdtest -d $dir -n 10 -i 200 -y
-N 10 -t -u -S
```

`-n 10` specifies that each thread creates 10 files/directories. `-i` is the number of iterations. `-y` is used to flush cache data after each write. `-u` indicates a unique working directory for each thread. `-N` is used to specify that each thread stats the files their neighbor created to avoid client cache effects for read operations

Figure 10. Create Command Metadata Performance

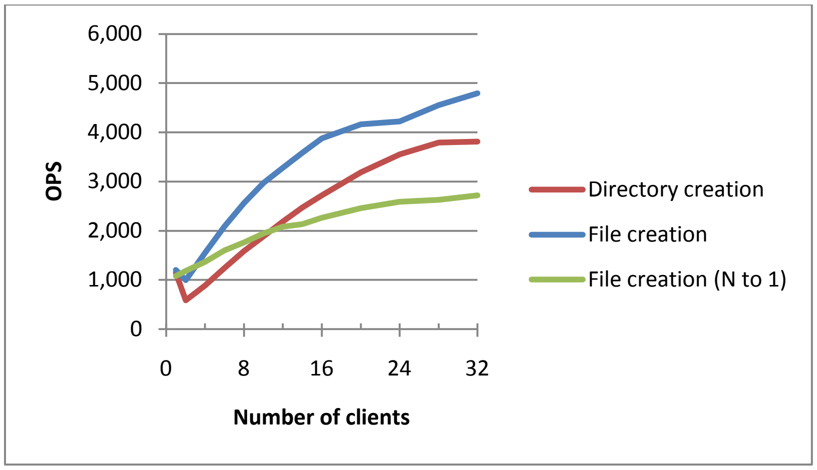


Figure 11. Stat Command Metadata Performance

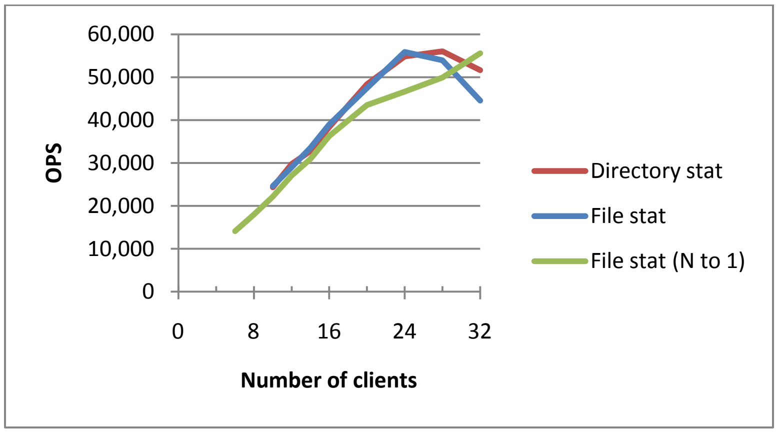


Figure 12. Utime Command Metadata Performance

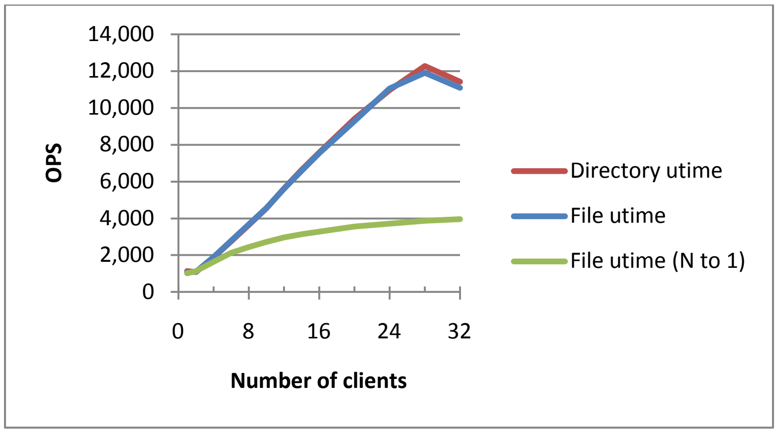
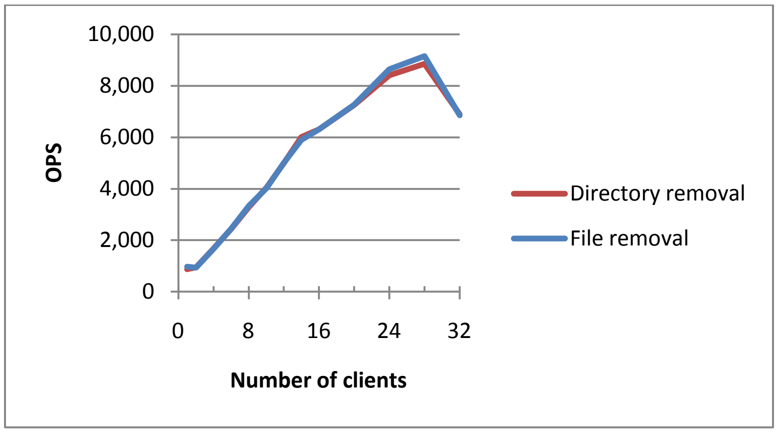


Figure 13. Remove Command Metadata Performance



The file and directory metadata operations show similar performance when clients are working in separate directories. When the working directory is shared, the performance is lower since more synchronization is required.

Random I/O Performance

Random I/O performance can be important for a number of applications, and for HPC systems that have a large number of I/O operations that occur on the NFS server. To measure random I/O performance, the IOzone benchmark was used. To reduce the cache impact, a data set much larger than the aggregated cache size of the entire system is required. However, compared to the sequential tests, random tests are significantly slower with large datasets. To speed up the tests and ensure data accuracy, the system cache was manually cleaned for each test run as follows:

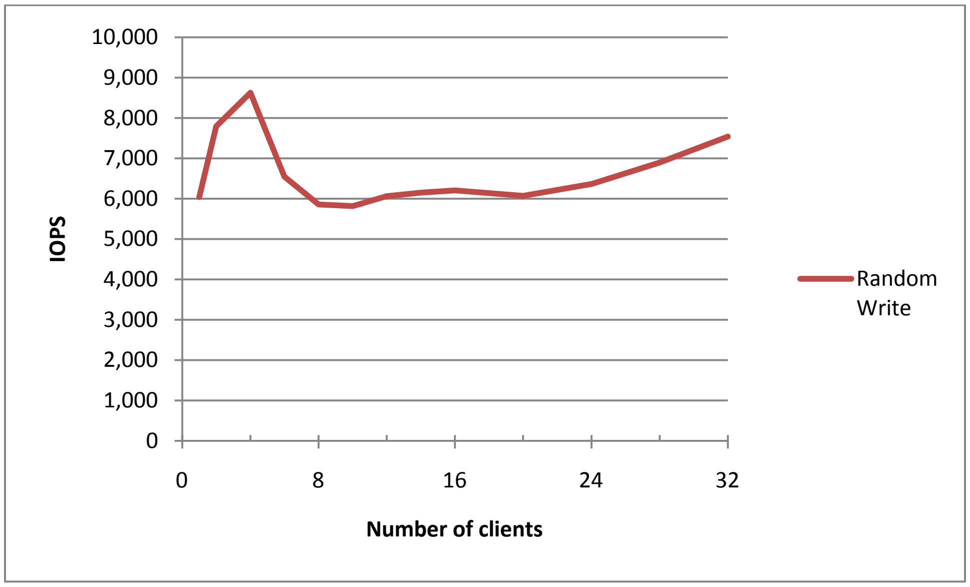
- 32 files were created in advance on the NFS share
- Before every run to make sure data is not cached on the NFS server and the clients, an `umount` of the NFS share is performed on the clients
- Next the NFS service is stopped on the server
- Remounted the storage
- Started NFS service and remount NFS share on the clients

A 1 KB block size and 512 MB file were used since the client and server caches were cleared before every test using the process previously mentioned. IOzone was launched in the O_DIRECT mode in order to make sure the data is not kept on the client cache. The following command was used to launch the benchmark:

```
./iozone -i2 -r 1k -s 512m -c -e -t <# of clients> -+m hosts -+n -w -I
```

`-i2` indicates this is a random test. `-I` is used for running in the O_DIRECT mode.

Figure 14. Random Write IOPS Performance Results



The random write performance is shown in Figure 14, and was observed at 6,000 to 9,000 I/Os per second. For the random load, only the write performance is reported since the clients do not cache any data and send all the write operations to the NFS server. On the other hand, read operations can take a shorter path by reading from the local cache if the data was read once before, and therefore may not give an accurate picture of the end-to-end performance of the system. See [3] for an in-depth study of the MD1200 direct-attach performance under different synthetic workloads.

Architecting the PowerVault MD1200 with a 10GbE interface provides a cost-effective NFS solution for HPC systems. As demonstrated by the previous performance tests, this solution has the capability to deliver up to 600 MB/s sequential throughput with a Linux ext3 local file system, and the random file access can reach 9000 IOPS. Metadata tests also indicate that this solution can support up to 5000 file creation IOPS, and 50000 file stat IOPS.

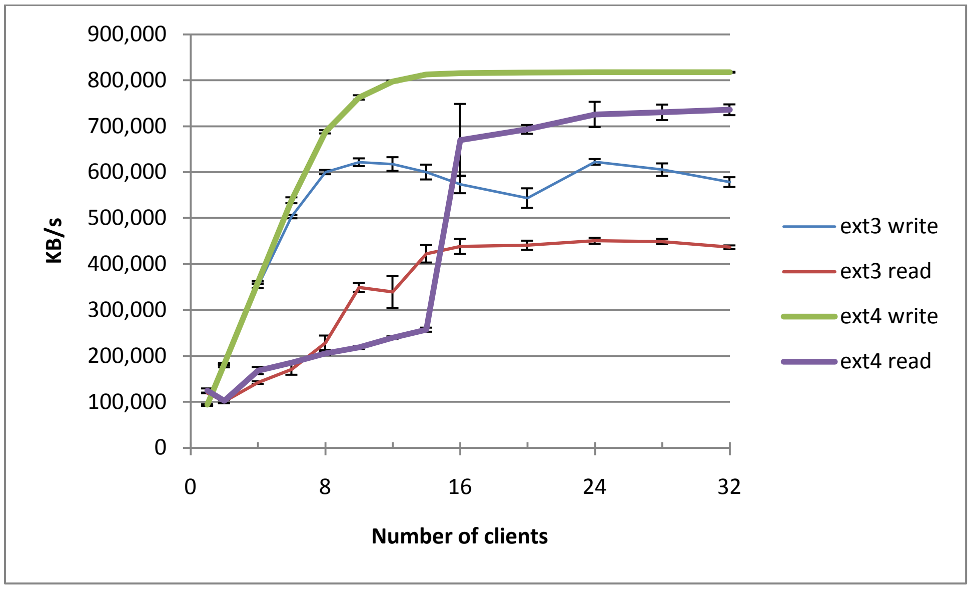
Looking Forward

Performance of ext3 versus ext4

Ext4 is a new Linux local file system that is included in RHEL5U4 as a technology preview. It is not supported in this version of RHEL, but official support is expected in upcoming Red Hat releases. Ext4 is gaining more popularity because of the new features it provides, such as creating up to 1 Exabyte volumes and up to 16 terabyte files. It introduces further enhancements, such as using checksums in the journal, using extents rather than block maps, and delayed block allocation that improves large file performance.

The goal of this section is to compare the sequential read and write performance of an NFS server using ext3 versus ext4 as the local file system. Installing `e4fsprogs` rpm allows the ext4 file system tools to build and maintain the file system. The tests described in the section *Throughput and IOPs Scalability* are repeated here. See Figure 7 for results using ext3.

Figure 15. ext3 vs. ext4 Comparison (Sequential Write and Read)



The chart above shows the benchmark run results performed on the same cluster. Every test was run 5 times, and the average is reported with the standard deviation shown as error bards. Only one IOzone thread per client was used since increasing the IOzone thread number did not result in improved performance.

Ext4 resulted in better read and write performance when compared to ext3 due to its enhancements, such as the use of extents and delayed block allocation. The sudden read performance increase for 16 clients is caused by the number of NFS threads. The 14 and 16 client tests were re-run with a different number of threads; with 512 NFS threads for example, the performance increase from 14 clients to 16 clients was 220 MB/s, which is much lower than the increase with 128 NFS threads in Figure 15.

Overall the ext4 results are very encouraging, as ext4 is considerably faster than ext3. As ext4 evolves, its performance characteristics may change, however once the file system becomes supported it is expected to be an excellent choice for Linux.

Summary

While numerous parallel file systems gradually gain popularity on cluster computing platforms, NFS is still the most popular shared file system used in high-performance clusters. For a typical HPC platform, NFS is normally built on a GigE fabric, isolated from the sensitive IPC traffic. However, as the number of clients increase, the aggregate bandwidth requirement between an NFS server and the multiple NFS clients can quickly exceed the capability of a GigE fabric. This white paper illustrates how the PowerVault MD1200 can be architected and tuned for performance using a 10Gbe interface for HPC systems that use NFS. As demonstrated by the previous testing results, this solution has the capability to deliver up to 600 MB/s aggregate bandwidth using ext3, and 800MB/s aggregate bandwidth using ext4. Some big advantages NFS has over other cluster file systems is its simplicity, it is included as a Linux standard package without any extra licensing requirements, and an NFS share can be brought up within minutes.

The results of the test cluster configuration benchmarks show approximately how much performance can be expected from an NFS over a 10 GbE solution under different workloads. The goal was to cover most test scenarios that may occur as various solution requirements. Based on the results, NFS continues to have good performance and is a cost effective solution when the system I/O performance requirements are within the parameters summarized in this paper.

The best throughput performance occurs when there is little interdependency between the NFS clients. For sequential workloads, approximately 600 MB/s write and 450 MB/s read performance is possible when using ext3. The ext4 technology preview that was tested resulted in even higher performance, a little over 800 MB/s for writes and a little over 700 MB/s for reads. NFS v3, being a stateless protocol, focuses on keeping the server load low by performing more processing on the clients. Therefore, the solution described in this white paper can scale to 32 clients. While not reported, up to 64 clients were tested in some cases with performance slowly degrading as the load increased.

For the test cases where multiple clients operate on the same file, N-1 IO operations, NFS can handle the workload without compromising data consistency by using the additional mount options that were added; however, the performance is significantly degraded due to the high level of synchronization that is required. The results reported here suggest that NFS may not provide good performance when the clients read from and write to the same files, N-to-1 IO operations, and the data consistency requirements are high.

Metadata operations are an increasingly important aspect of I/O performance, because of the number of processes accessing the NFS server. Mdtest was used to test metadata performance as the number of compute nodes was increased. Overall, metadata performance is a function of the NFS server file system, but NFS can also impact performance depending upon the operation details (i.e. N-to-1 or N-to-N or operations in different directories or all in the same directory). Metadata rates for “create”, “stat”, “utime”, and “remove” operations generally increased with the number of clients to approximately 28 clients.

Overall, the PowerVault MD1200 storage arrays are a great fit as an NFS server storage device. The MD1200 uses a new 6 Gb/s SAS backplane that improves the throughput from the drives to the host. It is a good match as a NFS server configured with a 10 Gigabit Ethernet connection to the main cluster switch. The cluster setup described in this white paper utilizes Dell’s standards based HPC architecture to provide a high performance, easily configurable, and manageable HPC storage solution.

References

[1] http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Deployment_Guide-en-US/s1-nfs-tcp.html

[2] http://kbase.redhat.com/faq/docs/DOC-2893

[3] http://principledtechnologies.com/clients/reports/Dell/6Gbps_v_3Gbps_RAID.pdf

Appendix A - Modifying MTU / Jumbo Frames on Dell Power Connect 6248 Switch

This section describes how to enable jumbo frames on a Dell PC 6248 switch.

To connect to the switch (PC6248), use any terminal application with these settings:

```
Port: Com1
Baud Rate: 9600
Data: 8 Bit
Parity: None
Stop: 1 Bit
Flow Control: None
```

Use the following commands to show current MTU/jumbo frame configuration:

```
>enable
#show running-config
```

Use the following commands to set MTU:

```
#configure
(config)#interface range ethernet all
(config-if)#mtu 9000
(config-if)#exit
(config)#interface range ethernet ch1-ch24
(config-if)#mtu 9000
(config)#exit
#show running-config
```

To show the new configuration with MTU of 9000:

```
>enable
#show running-config
```

Copy the running configuration to the start up configuration so the settings will be maintained after reboot:

```
#copy running-config startup-config
```