Technical White Paper

Dell EMC Ready Solutions for HPC BeeGFS High Performance Storage

Abstract
This Dell EMC technical white paper discusses the architecture, scalability and performance of the Dell EMC Ready Solutions for HPC BeeGFS Storage.

November 2020

ID 460

Revisions

Date	Description
January 2020	Initial release
November 2020	Updated to address language-standards

Acknowledgments

This paper was produced by the following:

Author: Nirmala Sundararajan — HPC and AI Innovation Lab

Support: N/A

Other: N/A

Table of contents

1 Introduction

High Performance Computing (HPC) is undergoing transformations that include storage systems technologies as a driving component. The first major leap in drive technology was the SSD, which owing to lack of moving parts delivered significantly better random access and read performance compared to HDDs. The second major leap in drive technology is the introduction of Non-Volatile Memory Express (NVMe) drives which offer latency savings by removing the scheduler and queueing bottlenecks[1] from the block layer. The highlight of the Dell EMC Ready Solutions for HPC BeeGFS Storage solution is the use of PowerEdge R740xd servers equipped with 24x Intel P4600 1.6 TB NVMe, Mixed Use Express Flash drives and two ConnectX-5 InfiniBand EDR adapters which offers a balanced configuration engineered for high performance. The Dell EMC HPC BeeGFS Storage Solution is delivered as an all-inclusive storage solution and is available with deployment services and full hardware and software support from Dell EMC. The solution is designed using standards-based HPC components and focuses on ease of deployment and ease of use.

This technical white paper describes the configuration of the Dell EMC Ready Solutions for HPC BeeGFS Storage[2] (referred to as Dell EMC BeeGFS Storage Solution in the rest of the document), which is the latest addition to the HPC storage portfolio. Many types of jobs generate large amount of data while the job is running, but that data is discarded when the job completes. These jobs do not perform all their computations purely in memory. Each of the worker threads loads an initial configuration and reads/writes many intermediate points to disk. In such scenarios, the scratch space is often considered as a limiting factor. It is often too small or too slow. To tackle this problem, the Dell EMC HPC BeeGFS Storage Solution is being offered as a scratch solution; a staging ground for transient data that is usually not held beyond the lifetime of the job. The Dell EMC BeeGFS storage solution uses a flexible building block approach, where individual building blocks can be combined to offer various configurations to meet customer-specific workloads and use cases and to provide for future expansion.

This technical white paper describes the architecture of the Dell EMC BeeGFS Storage Solution, the solution configuration, software, and application versions. The capabilities of the system are quantified by presenting the benchmark performance for IOzone sequential N-N read and write throughput (GBps) and random read and write I/O operations per second (IOPS), IOR N-1 performance and metadata tests. An extensive appendix with details on the benchmarks used and the testing methodology adopted is included at the end of the document.

2 BeeGFS File System

BeeGFS[3] is an open source parallel cluster file system. The software can be downloaded from www.beegfs.io. The file system software also includes enterprise features such as High Availability, Quota enforcement, and Access Control Lists. BeeGFS is a parallel file system which distributes user data across multiple storage nodes. There is parallelism of access as it maps data across many servers and drives and provides a global namespace, a directory tree, that all nodes can see. It is easy to deploy BeeGFS and integrate it with existing systems. The BeeGFS server components are user space daemons. The client is a native kernel module that does not require any patches to the kernel itself. All BeeGFS components can be installed and updated without even rebooting the machine. So, clients and servers can be added to existing systems without any downtime. BeeGFS is a highly scalable file system. By increasing the number of servers and drives, the performance and capacity can be increased to the required level from small clusters up to enterprise-class systems with thousands of nodes. As BeeGFS is software defined storage that decouples the storage software from its hardware, it offers flexibility and choice while making hardware purchasing decisions. In BeeGFS, the roles and hardware are not tightly integrated. The BeeGFS clients and servers can even run on the same machine. BeeGFS supports a wide range of Linux distributions, RHEL, CentOS, SUSE, and so on. BeeGFS is designed to be independent of the local file system used. The local storage can be formatted with any of the standard Linux file systems—xfs or ext4.

The BeeGFS architecture consists of four main services:

- Management Service
- Metadata Service
- Storage Service
- Client Service

Each BeeGFS file system or namespace has only one management service. The management service is the first service which must be set up because when we configure all other services, they must register with the management service. The Metadata Service is a scale-out service, which means that there can be many metadata services in a BeeGFS file system. However, each metadata service has exactly one metadata target to store metadata. On the metadata target, BeeGFS creates one metadata file per user created file. BeeGFS metadata is distributed on a per-directory basis. The metadata service provides the data striping information to the clients and is not involved in the data access between file open/close. The Storage Service stores the user data. A BeeGFS file system can be made of multiple storage servers where each storage service can manage multiple storage targets. On those targets the striped user data is stored in chunk files for parallel access from the client. Except for the client service which is a kernel module, the management, metadata and storage services are user space processes. Figure 1 illustrates the general architecture of the BeeGFS file system.

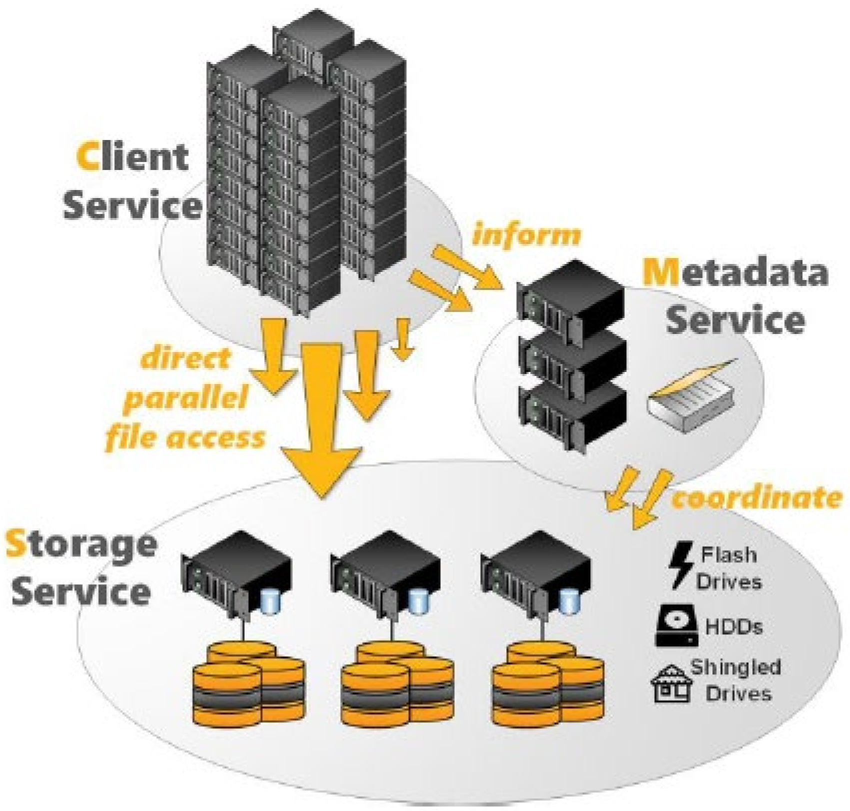

Figure 1 BeeGFS Architecture Overview[4]

3 Dell EMC BeeGFS Storage Solution Reference Architecture

Figure 2 shows the reference architecture of the solution. The management server is only connected using Ethernet to the metadata and storage servers. Each metadata and storage server have two InfiniBand links and are connected to the internal management network via Ethernet. The clients have one InfiniBand link and are connected to the internal management network using Ethernet. This is the “Medium” configuration which uses five dedicated storage servers. Other configurations with varying number of storage servers are also available as described in Section 5 of this document.

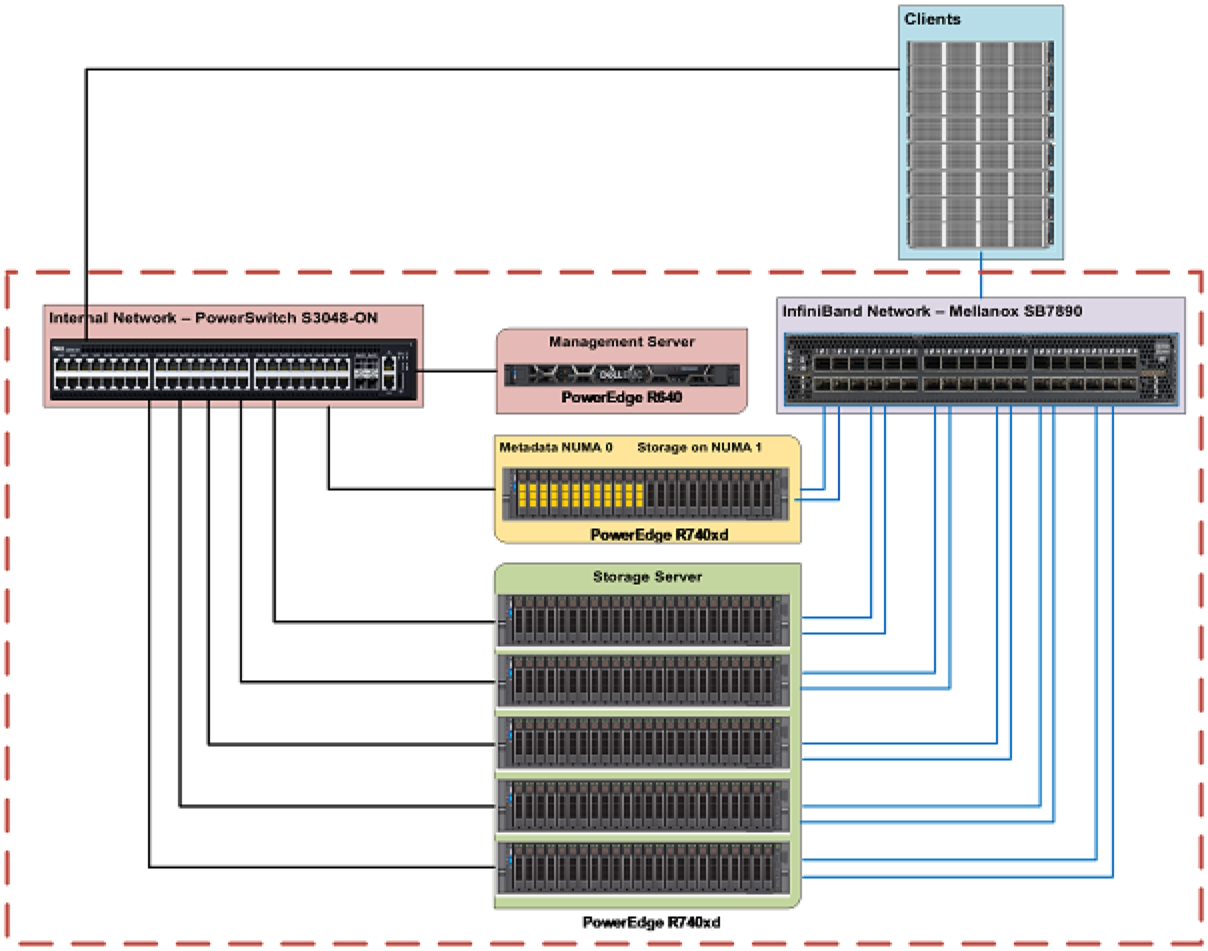

Figure 2 Dell EMC Ready Solutions for HPC BeeGFS Storage - Reference Architecture

3.1 Management Server

A PowerEdge R640 is used as the management server. In addition to hosting the management service (beegfs-mgmtd.service), it also hosts the monitoring service (beegfs-mon.service) which collects statistics from the system and provides them to the user, using the time series database InfluxDB. For visualization of data, beegfs-mon provides predefined Grafana panes that can be used out of the box. The management server has 6x 300 GB HDDs configured in RAID 10 for the Operating System and InfluxDB.

3.2 Metadata Server

A PowerEdge R740xd with 24x Intel P4600 1.6 TB NVMe drives is used for metadata storage. As the storage capacity requirements for BeeGFS metadata are small, instead of using a dedicated metadata server, only the 12 drives on NUMA zone 0 were used to host the Metadata Targets (MDTs), while the remaining 12 drives on NUMA zone 1 host Storage Targets (STs). Figure 3 shows the metadata server. The 12 drives enclosed in the yellow rectangle are the MDTs in the NUMA zone 0 whereas the 12 drives enclosed in the green rectangle are the STs in the NUMA zone 1. This configuration not only avoids NUMA issues but also provides enough metadata storage to facilitate scaling the capacity and performance as needed.

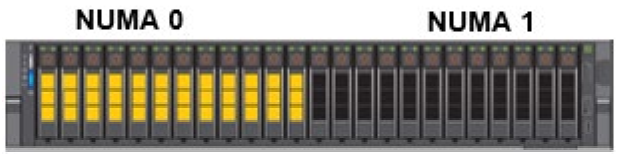

Figure 3 Metadata Server

Figure 4 shows the raid configuration of the metadata server. This figure shows how in the metadata server the drives in the NUMA zone 0 host the MDTs and those in NUMA zone 1 host the storage data, while the storage servers host the STs in both the NUMA zones.

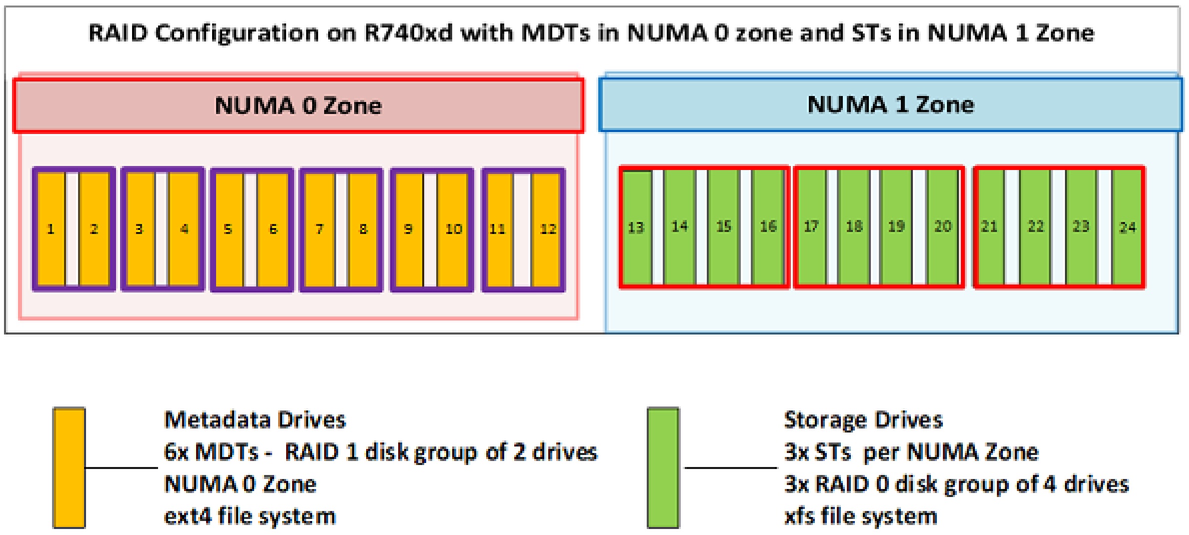

Figure 4 Configuration of drives in the Metadata Server

The 12 drives used for metadata are configured as 6x RAID 1 disk group of 2 drives, each serving as an MDT. There are 6 metadata services running each of which handles one MDT. The remaining 12 storage drives are configured in 3x RAID 0 disk groups of 4 drives each. There are three storage services running on the NUMA 1 zone, one service for each ST. So, the server which co-hosts the metadata and Storage Targets has 6 MDTs and 3 STs. It also runs 6 metadata services and three storage services. Each MDT is an ext4 file system[5] based on a RAID 1 configuration. The STs are based on XFS file system configured in RAID 0.

3.3 Storage Server

Figure 5 shows the 5x PowerEdge R740xd servers used as storage servers.

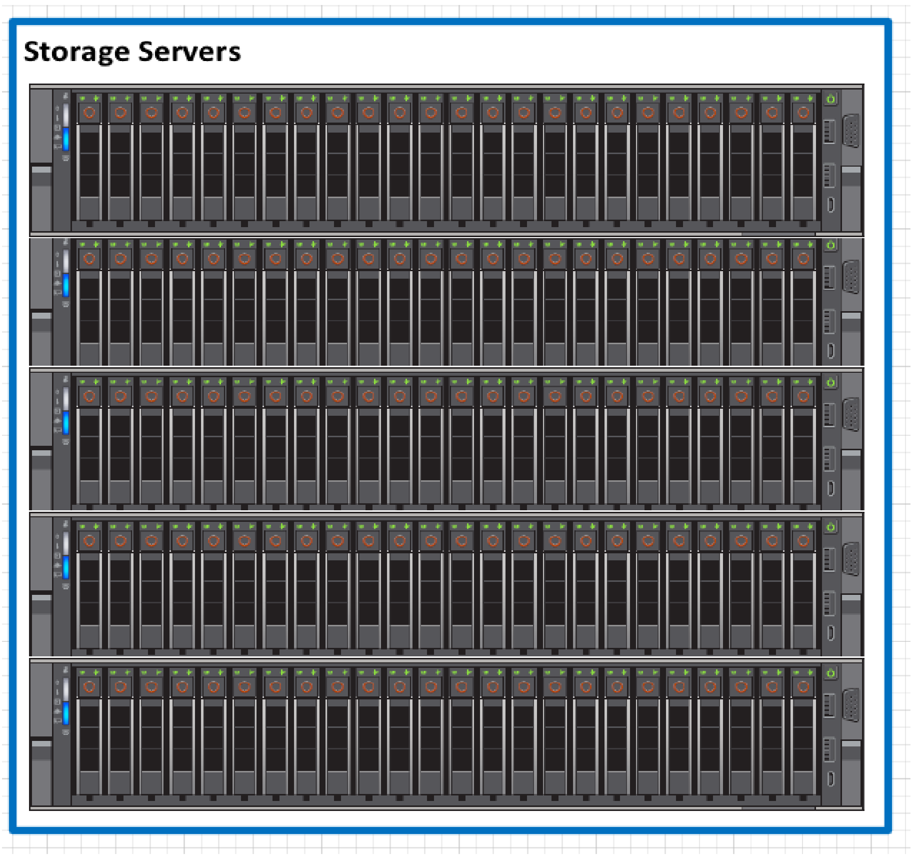

Figure 5 Dedicated Storage Servers

Each storage server has six storage targets, 3 per NUMA zone. In total, there are 33 storage targets in the configuration. The targets are configured like the STs on the Metadata Server. The NVMe drives are configured in RAID 0 disk groups of 4 drives each and XFS is used as the underlying file system for the beegfs-storage services.

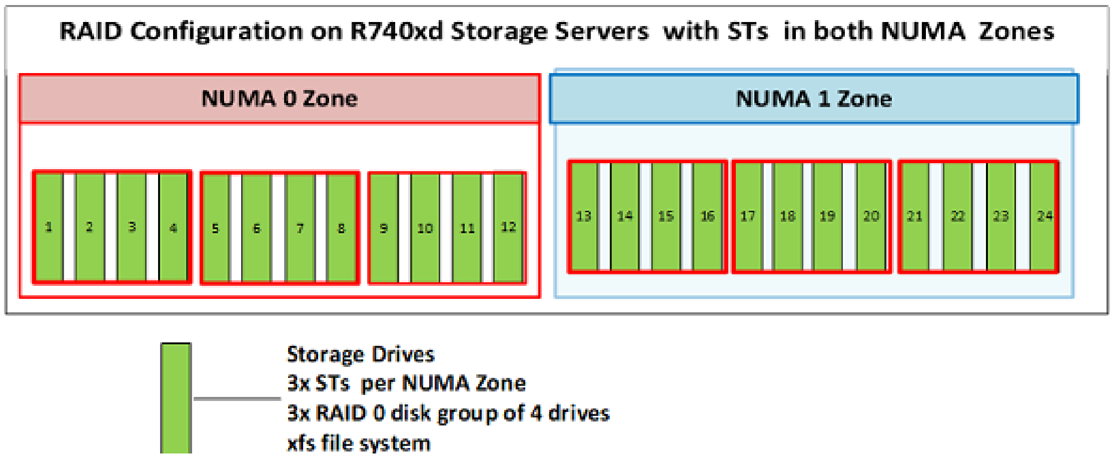

Figure 6 Configuration of drives in the Storage Servers

3.4 Clients

Thirty-two C6420 servers were used as clients. The BeeGFS client module must be loaded on to all the hosts that need to access the BeeGFS file system. When the beegfs-client is loaded, it mounts the file systems defined in the /etc/beegfs/beegfs-mounts.conf file instead of the usual approach based on /etc/fstab. Adopting this approach starts the beegfs-client like any other Linux service through the service startup script. It also enables the automatic recompilation of the BeeGFS client module after system updates. When the client module is loaded, it mounts the file systems defined in the beegfs-mounts.conf. It is possible to mount multiple beegfs instances on the same client as shown below:

```
$ cat /etc/beegfs/beegfs-mounts.conf
/mnt/beegfs-medium /etc/beegfs/beegfs-client-medium.conf
/mnt/beegfs-small /etc/beegfs/beegfs-client-small.conf
```

3.5 Hardware and Software Configuration

Dell EMC HPC BeeGFS Storage Solution is available in the standard configurations listed below. Contact a Dell Sales Representative to discuss which offering works best in your environment. Customers can order any of the preconfigured solutions or customization can be performed to fit specific needs. Tables 1 and 2 describe the hardware specifications of the management server and metadata/storage server respectively. Table 3 describes the software versions used for the solution.

Table 1 PowerEdge R640 configuration (Management Server)

Component	Details
Server Model	PowerEdge R640
Processor	2 x Intel Xeon Gold 5218 2.3 GHz, 16 cores
Memory	12 x 8 GB DDR4 2666MT/s DIMMs - 96 GB
Local Disks	6 x 300 GB 15K RPM SAS 2.5in HDDs
RAID Controller	PERC H740P Integrated RAID Controller
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller
Power Supplies	Dual 1100 W Power Supply Units
BIOS Version	2.2.11
Operating System	CentOS™ 7.6
Kernel Version	3.10.0-957.27.2.el7.x86_64

Table 2 PowerEdge R740xd Configuration (Metadata and Storage Servers)

Component	Details
Server	Dell EMC PowerEdge R740xd
Processor	2x Intel Xeon Platinum 8268 CPU @ 2.90 GHz, 24 cores
Memory	12 x 32 GB DDR4 2933MT/s DIMMs - 384 GB
BOSS Card	2x 240 GB M.2 SATA SSDs in RAID 1 for operating system
Local Drives	24x Dell Express Flash NVMe P4600 1.6 TB 2.5" U.2

Component	Details
Mellanox EDR card	2x Mellanox ConnectX-5 EDR card (Slots 1 & 8)
Out of Band Management	iDRAC9 Enterprise with Lifecycle Controller

Table 3 Software Configuration (Metadata and Storage Servers)

Component	Details
BIOS	2.2.11
CPLD	1.1.3
Operating System	CentOS 7.6
Kernel Version	3.10.0-957.el7.x86_64
iDRAC	3.34.34.34
Systems Management Tool	OpenManage Server Administrator 9.3.0-3407_A00
Mellanox OFED	4.5-1.0.1.0
NVMe SSDs	QDV1DP13
Intel Data Center Tool*	3.0.19
BeeGFS	7.1.3
Grafana	6.3.2
InfluxDB	1.7.7

* For Management and Firmware update of Intel P4600NVMe SSDs

3.6 Advantages of using NVMe devices in the R740xd servers

Figure 7 shows the storage layout in the Dell EMC PowerEdge Enterprise Servers.

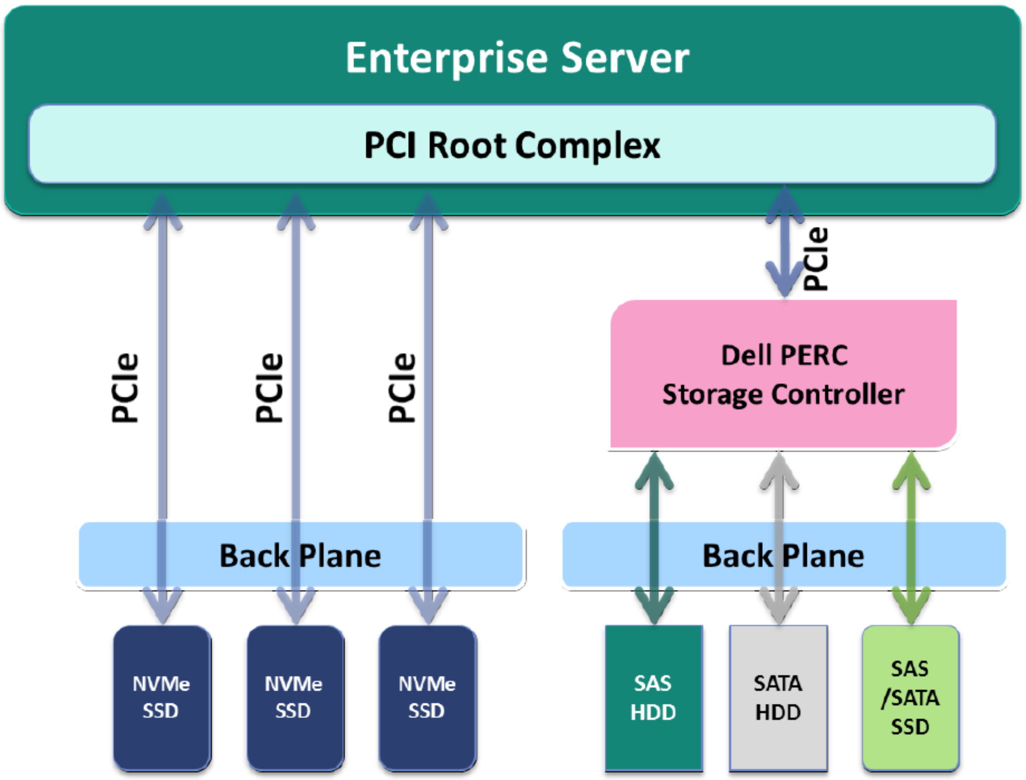

Figure 7 Storage Layout in the Dell EMC Enterprise Servers

Use of the NVMe devices in the solution provides the following distinct advantages:

1. While the legacy storage protocols such as SCSI, SAS and SATA communicate with the host using an integrated on-chip or an external Host Bus Adapter (HBA), the NVMe based devices connect through the PCIe interface as shown in Figure 1. A single Gen 3 PCIe lane allows transfers up to 985 MB/s. The P4600 SSDs used in the solution are x4 PCIe Gen 3 devices and so a x4 PCIe connection supports close to 4 GB/s.
2. The I/O paths of traditional SATA/SAS SSDs and NVMe SSDs are different. In case of a traditional I/O path, an I/O request arriving at the block layer is first inserted into a request queue (Elevator). The elevator then reorders and combines multiple requests to sequential requests. The most commonly used elevator scheduler for flash media is the noop scheduler, which implements a First in First out (FIFO) policy without any reordering. An NVMe request bypasses the conventional block layer queue. Instead it uses a paired submission and completion queue mechanism and offers latency savings by removing the scheduler and queuing bottlenecks from the block layer.
3. NVMe standard supports up to 64 K I/O queues and up to 64K commands per queue. The large pool of NVMe queues provides higher levels of I/O parallelism.
4. After a CPU writes a command to the tail of the submission queue, the CPU finishes the I/O request and can continue servicing other requests. Thus, the CPU time spent to issue an I/O request is significantly less in the case of NVMe drives unlike in case of SATA/SAS drives, where the CPU is responsible to pass the request through several functions before the command is finally issued to the device.

3.7 R740xd, 24x NVMe Drives, Details on CPU Mapping

In the 24xNVMe configuration of the PowerEdge R740xd server, there are two x16 NVMe extender cards connected to PCIe switches on the backplane which are connected to the drives. Each NVMe drive is connected to 4x PCIe lanes. This is shown in Figure 8 below:

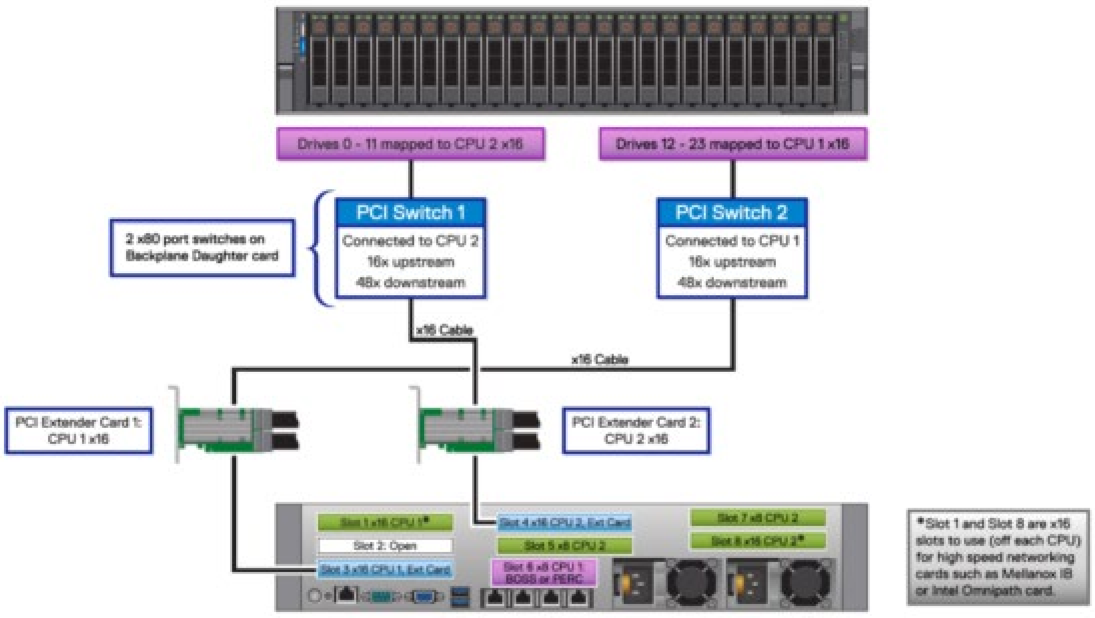
Drives 0 - 11 mapped to CPU 2 x16
Drives 12 - 23 mapped to CPU 1 x16
2 x80 port switches on Backplane Daughter card
PCI Switch 1
Connected to CPU 2
16x upstream
48x downstream
PCI Switch 2
Connected to CPU 1
16x upstream
48x downstream
x16 Cable
x16 Cable
PCI Extender Card 1: CPU 1 x16
PCI Extender Card 2: CPU 2 x16
Slot 1 x16 CPU 1*
Slot 2: Open
Slot 3 x16 CPU 1, Ext Card
Slot 4 x16 CPU 2, Ext Card
Slot 5 x8 CPU 2
Slot 6 x8 CPU 1: BOSS or PERC
Slot 7 x8 CPU 2
Slot 8 x16 CPU 2*
*Slot 1 and Slot 8 are x16 slots to use (off each CPU) for high speed networking cards such as Mellanox IB or Intel Omnipath card.

Figure 8 R740xd, 24x NVMe Details on CPU Mapping

In Non-Uniform Memory Access (NUMA), system memory is divided into zones called nodes, which are allocated to CPUs or sockets. Access to memory that is local to a CPU is faster than memory connected to remote CPUs on the system. A threaded application typically performs best when the threads are accessing memory on the same NUMA node. The performance impact of NUMA misses is significant, generally starting at a 10% performance hit or higher. To improve performance, the services are configured to use specific NUMA zones to avoid unnecessary use of UPI cross-socket links thereby reducing latency. Each NUMA zone handles 12 drives and uses one of the two InfiniBand EDR interfaces on the servers. This NUMA separation is achieved by manually configuring NUMA balancing by creating custom systemd unit files and by configuring multihoming. Hence the automatic NUMA balancing is disabled, as shown below:

```
# cat /proc/sys/kernel/numa_balancing
0
```

4 Performance Characterization

To characterize the performance of the Ready Solution, the testbed used is shown in Figure 9. In the figure, the InfiniBand connections to the NUMA zone are highlighted. Each server has two IP links and the traffic through NUMA 0 zone is handled by interface IB0 while the traffic through NUMA 1 zone is handled by interface IB1.

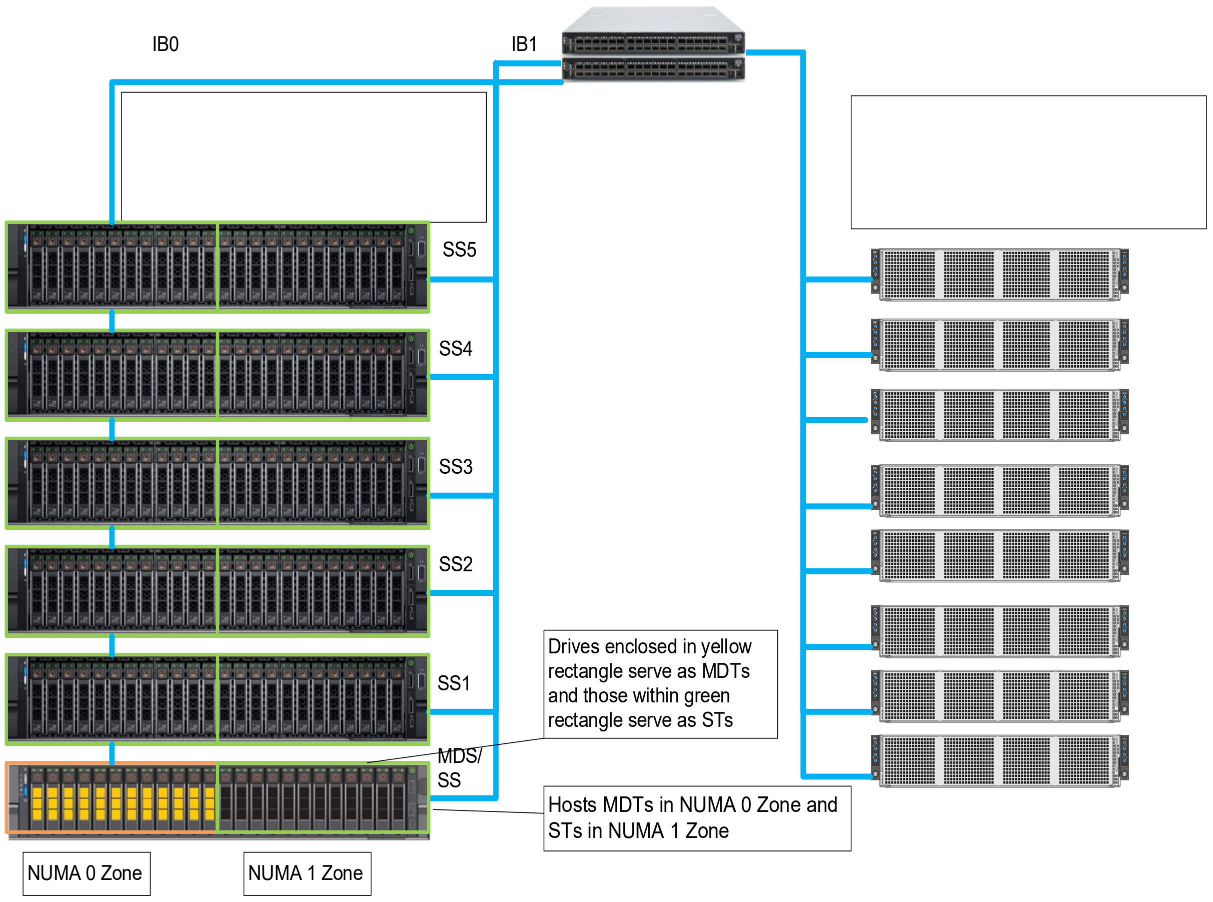
IB0
IB1
SS5
SS4
SS3
SS2
SS1
MDS/
SS
Drives enclosed in yellow rectangle serve as MDTs and those within green rectangle serve as STs
Hosts MDTs in NUMA 0 Zone and STs in NUMA 1 Zone
NUMA 0 Zone
NUMA 1 Zone

Figure 9 Testbed Configuration

The hardware and software configuration of the testbed is specified in Tables 1, 2 and 3. In order to assess the solution performance, the following benchmarks were used:

1. IOzone N to N sequential
2. IOR N to 1 sequential
3. IOzone random
4. MDtest

For all the benchmarks listed above the testbed had the clients described in Table 4. Since the number of compute nodes available for testing was 32, when a higher number of threads was required, those threads were equally distributed on the compute nodes (i.e. 64 threads = 2 threads per node, 128 threads = 4 threads per node, 256 threads = 8 threads per node, 512 threads = 16 threads per node, 1024 threads = 32 threads per node and 2048 threads = 64 threads per node). The intention was to simulate a higher number of concurrent clients with the limited number of compute nodes.

Table 4 Client Configuration

Component	Details
Clients	32x Dell EMC PowerEdge C6420 Compute Nodes
BIOS	2.2.9
Processor	2x Intel Xeon Gold 6148 CPU @ 2.40GHz, 20 cores
Memory	12x 16GB DDR4 2666 MT/s DIMMs - 192GB
BOSS Card	2x 120GB M.2 boot drives in RAID 1 for OS
Operating System	Red Hat Enterprise Linux Server release 7.6
Kernel Version	3.10.0-957.el7.x86_64
Interconnect	1x Mellanox ConnectX-4 EDR card
OFED Version	4.5-1.0.1.0

The transparent huge pages were disabled, and the following tuning options are in place on the metadata and storage servers:

```
vm.dirty_background_ratio = 5
vm.dirty_ratio = 20
vm.min_free_kbytes = 262144
vm.vfs_cache_pressure = 50
vm.zone_reclaim_mode = 2
kernel.numa_balancing = 0
```

In addition to the above, the following BeeGFS tuning options were used:

- `tuneTargetChooser` parameter was set to `"roundrobin"` in the metadata configuration file
- `tuneNumWorkers` parameter was set to 24 for metadata and 32 for storage
- `connMaxInternodeNum` parameter was set to 32 for metadata and 12 for storage and 24 for clients

4.1 Sequential writes and reads IOzone N-N

To evaluate sequential reads and writes, the IOzone benchmark v3.487 was used in the sequential read and write mode. These tests were conducted on multiple thread counts starting at 1 thread and increasing in powers of 2, up to 1024 threads. At each thread count, an equal number of files were generated since this test works on one file per thread or the N clients to N file (N-N) case. The processes were distributed across 32 physical client nodes in a round robin or cyclical fashion so that the requests are equally distributed and there is load balancing. An aggregate file size of 8TB was selected which was equally divided among the number of threads within any given test. The aggregate file size was chosen large enough to minimize the effects of caching from the servers as well as from BeeGFS clients. IOzone was run in a combined mode of write then read (-i 0, -i 1) to allow it to coordinate the boundaries between the operations. For this testing and results, we used a 1MiB record size for every run. The commands used for Sequential N-N tests are as follows:

```
iozone -i 0 -i 1 -c -e -w -r 1m -I -s $Size -t $Thread -+n -+m
/path/to/threadlist
```

To minimize the effects of caching, OS caches were also dropped or cleaned on the client nodes between iterations as well as between write and read tests by running the command:

```
# sync && echo 3 > /proc/sys/vm/drop_caches
```

The default stripe count for BeeGFS is 4. However, the chunk size and the number of targets per file can be configured on a per-directory basis. For all these tests, BeeGFS stripe size was chosen to be 2MB and stripe count was chosen to be 3 since we have three targets per NUMA zone as shown below:

```
$ beegfs-ctl --getentryinfo --mount=/mnt/beegfs /mnt/beegfs/benchmark --verbose
EntryID: 0-5D9BA1BC-1
ParentID: root
Metadata node: node001-numa0-4 [ID: 4]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 2M
+ Number of storage targets: desired: 3
+ Storage Pool: 1 (Default)
Inode hash path: 7/5E/0-5D9BA1BC-1
```

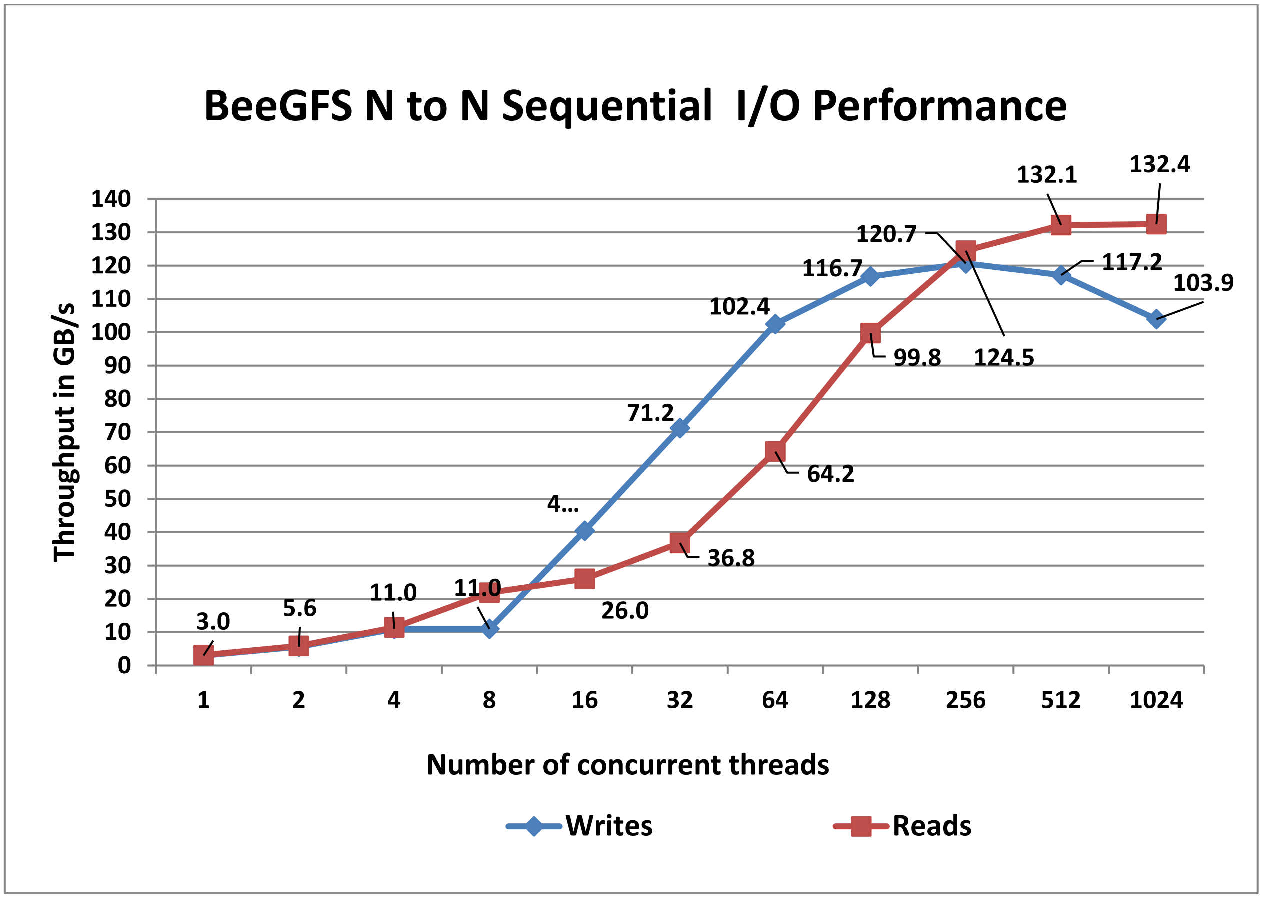

Figure 10 Sequential IOzone 8TB aggregate file size

The IOzone sequential write and read tests were performed three times and the mean value is plotted in Figure 10 above. We observe that peak read performance is 132 GB/s at 1024 threads and peak write is 121 GB/s at 256 threads. As per the technical specifications of the Intel P4600 1.6 TB NVMe SSDs, each drive can provide 3.2 GB/s peak read performance and 1.3 GB/s peak write performance, which allows a theoretical peak of 422 GB/s for reads and 172 GB/s for writes. However, here the network is the limiting factor. We have a total of 11 InfiniBand EDR links for the storage servers in the test bed. Each link can provide a theoretical peak performance of 12.4 GB/s which provides an aggregate theoretical peak performance of 136.4 GB/s. The achieved peak read and write performance are 97% and 89% respectively of the theoretical network peak performance.

The single thread write performance is observed to be ~3 GB/s and read at ~3 GB/s. We observe that the write performance scales linearly, peaks at 256 threads and then starts decreasing. At lower thread counts read and write performance are the same. Because until 8 threads, we have 8 clients writing 8 files across 24 targets which means, not all storage targets are being fully utilized. We have 33 storage targets in the system and hence at least 11 threads are needed to fully utilize all the servers. The read performance registers a steady linear increase with the increase in the number of concurrent threads and we observe almost similar performance at 512 and 1024 threads.

We also observe that the read performance is lower than writes for thread counts from 16 to 128. This is because while a PCIe read operation is a Non-Posted Operation, requiring both a request and a completion, a PCIe write operation is a fire and forget operation. Once the Transaction Layer Packet is handed over to the Data Link Layer, the operation completes. A write operation is a "Posted" operation that consists of a request only.

Read throughput is typically lower than the write throughput because reads require two transactions instead of a single write for the same amount of data. The PCI Express uses a split transaction model for reads. The read transaction includes the following steps:

- The requester sends a Memory Read Request (MRR).
- The completer sends out the acknowledgement to MRR.
- The completer returns a Completion with Data.

The read throughput depends on the delay between the time the read request is issued and the time the completer takes to return the data. However, when the application issues enough number of read requests to cover this delay, then throughput is maximized. That is the reason why while the read performance is less than that of the writes from 16 threads to 128 threads, we measure an increased throughput when the number of requests increases. A lower throughput is measured when the requester waits for completion before issuing subsequent requests. A higher throughput is registered when multiple requests are issued to amortize the delay after the first data returns.

More details about the PCI Express Direct Memory Access is available at https://www.intel.com/content/www/us/en/programmable/documentation/nik1412547570040.html[6].

4.2 Sequential writes and reads IOR N-1

The performance of sequential reads and writes with N threads to a single shared file was measured with IOR version 3.3.0, assisted by OpenMPI v1.10.7 to run the benchmark over the 32 compute nodes. Tests executed varied from single thread up to 512 threads. The OS caches were dropped between the runs on the BeeGFS servers as well as BeeGFS clients as detailed in section 4.1.

The following commands were used to execute the benchmark for writes and reads, where `threads` was the variable with the number of threads used incremented in powers of two. The transfer size is 2M and each thread wrote to or read 128G from a single file striped over all the 33 targets. Three iterations of each test have been run and the mean value has been recorded. Figure 11 shows the N to 1 sequential I/O performance. The command used to run the test is given below:

```
mpirun -machinefile $hostlist --map-by node -np $threads -mca btl openib,self
~/bin/ior -w -r -i 3 -d 3 -e -O beegfsNumTargets=33,beegfsChunkSize=2M -o
$working_dir/ior.out -s 1 -g -t 2M -b $file_size
```

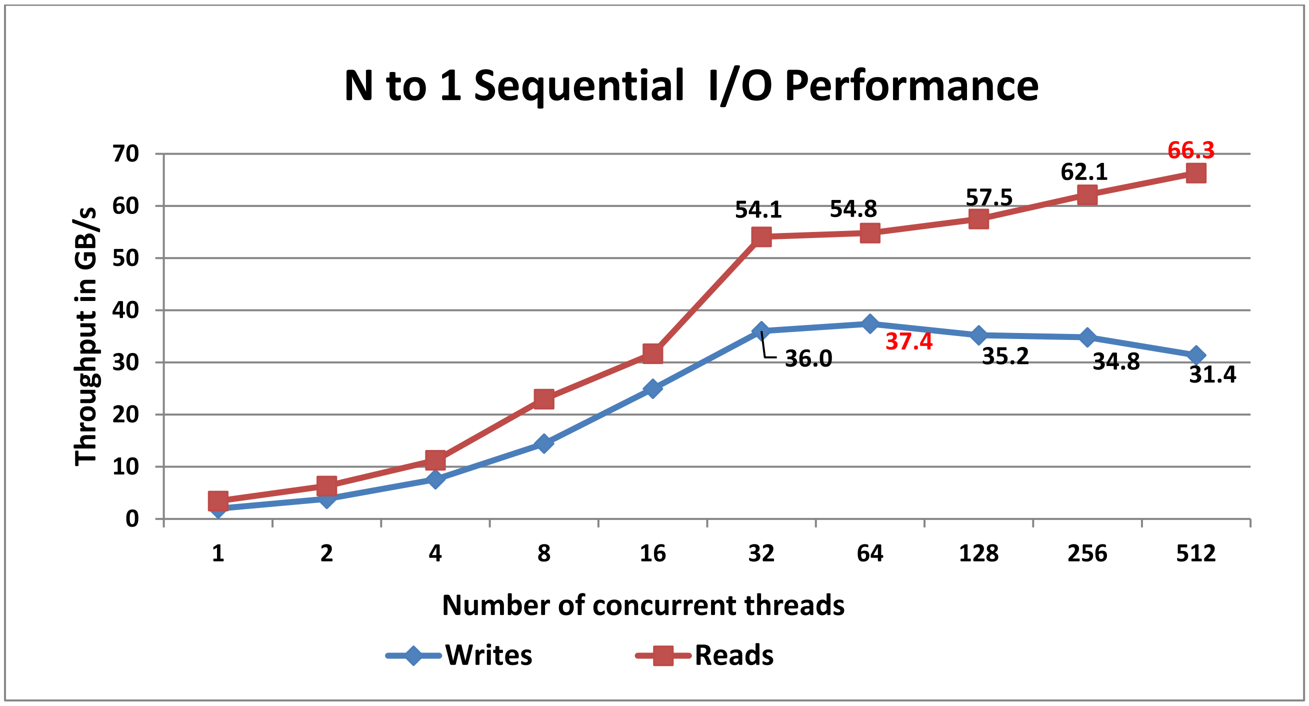

Figure 11 Sequential N to 1 Sequential I/O Performance

From the results we observe that there is a steady increase in both the read and write performance as the number of concurrent threads is increased. The performance remains stable for reads as well as writes all the way to the maximum number of threads used in this test. The peak read performance of 66.3 GB/s is observed at 512 threads. Write performance attains the peak value of 37.4 GB/s at 64 threads.

4.3 Random writes and reads N-N

To evaluate random IO performance, IOzone was used in the random mode. Tests were conducted on thread counts starting from 4 threads to up to 1024 threads. Direct IO option (-I) was used to run IOzone so that all operations bypass the buffer cache and go directly to the devices. BeeGFS stripe count of 3 and chunk size of 2MB was used. A 4KiB request size is used on IOzone. Performance is measured in I/O operations per second (IOPS). The OS caches were dropped between the runs on the BeeGFS servers as well as BeeGFS clients as detailed in section 4.1 to avoid caching effects. The command used for executing the random writes and reads is as follows:

```
iozone -i 2 -w -c -O -I -r 4K -s $Size -t $Thread -+n -+m /path/to/threadlist
```

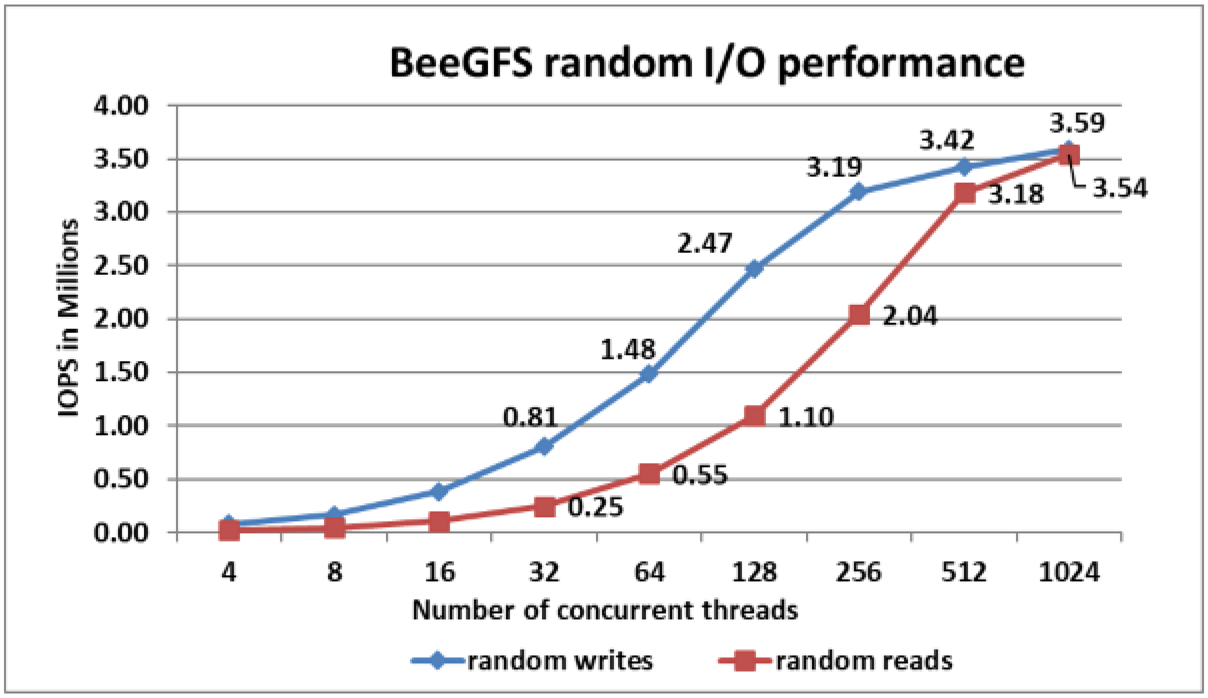

Figure 12 Random Read and Write Performance using IOzone with 8TB aggregate file size

Three iterations of the random write and read tests were done and the mean value of the results is shown in Figure 12. The random writes peak at ~3.6 Million IOPS at 512 threads and the random reads peak at ~3.5 Million IOPS at 1024 threads as shown in Figure 11. Both the write and read performance show a higher performance when there are a higher number of IO requests. This is because NVMe standard supports up to 64K I/O queue and up to 64K commands per queue. This large pool of NVMe queues provide higher levels of I/O parallelism and hence we observe IOPS exceeding 3 million.

4.4 Metadata performance with MDtest using empty files

Metadata performance was measured with MDtest version 3.3.0, assisted by OpenMPI v1.10.7 to run the benchmark over the 32 compute nodes. Tests executed varied from single thread up to 2048 threads. The benchmark was used for files only (no directories metadata), getting the number of creates, stats, reads and removes the solution can handle.

The following command was used to execute the benchmark, where threads was the variable with the number of threads used (1 to 2048 incremented in powers of two), and $hostfile is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 32 compute nodes.

```
mpirun -machinefile $hostlist --map-by node -np $threads ~/bin/mdtest -i 3 -b
$Directories -z 1 -L -I 1024 -y -u -t -F
```

Since performance results can be affected by the total number of IOPs, the number of files per directory and the number of threads, consistent number of files across tests was chosen to be able to compare them on similar grounds. The total number of files is ~ 2M in powers of two (2^21 = 2097152). The number of files per directory was fixed at 1024, and the number of directories varied as the number of threads changed as shown in Table 3. Figure 13 shows the metadata performance using empty (i.e. 0 KB) files.

Table 5 MDtest distribution of files on directories

Number of Threads	Number of files per directory	Number of directories per thread	Total number of files
1	1024	2048	2,097,152
2	1024	1024	2,097,152
4	1024	512	2,097,152
8	1024	256	2,097,152
16	1024	128	2,097,152
32	1024	64	2,097,152
l64	1024	32	2,097,152
128	1024	16	2,097,152
256	1024	8	2,097,152
512	1024	4	2,097,152
1024	1024	2	2,097,152
2048	1024	1	2,097,152

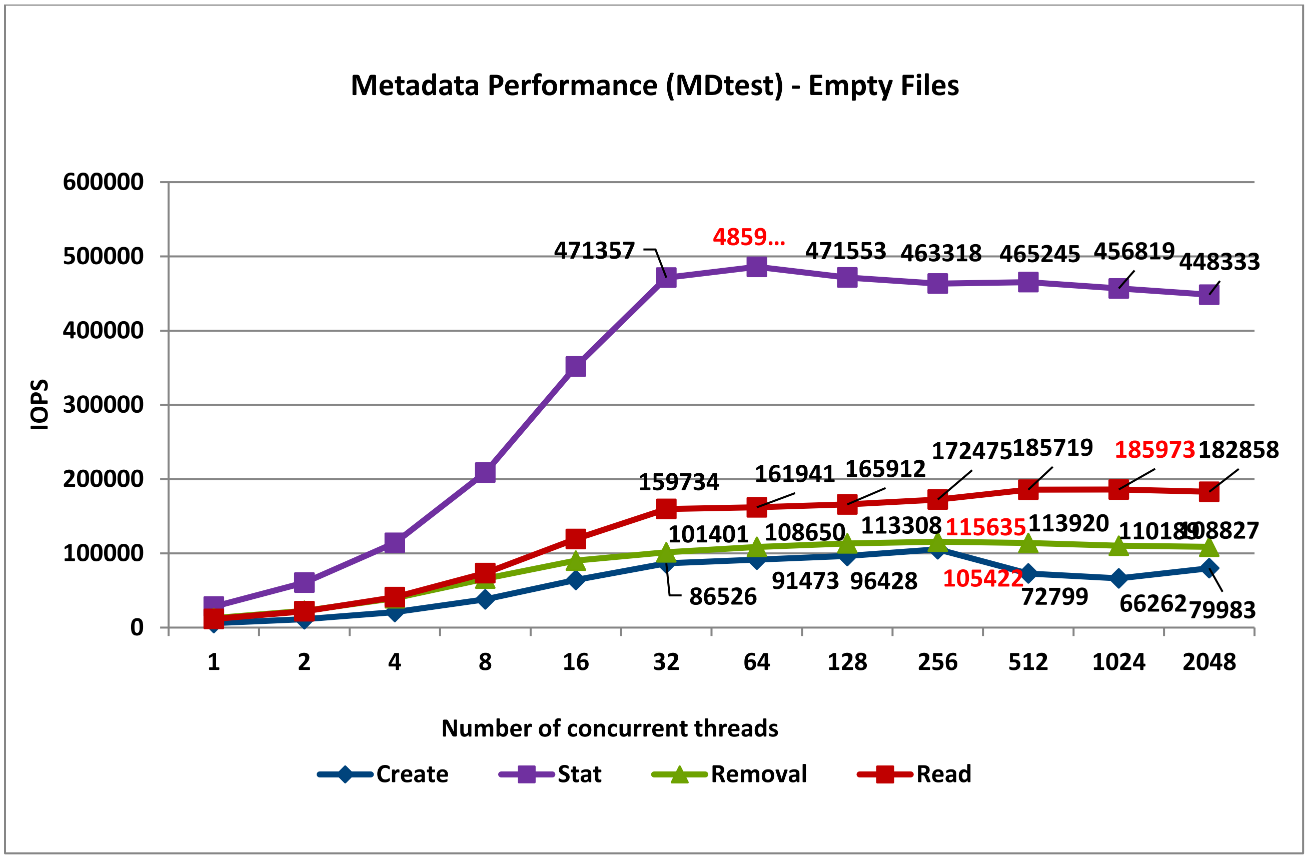

Figure 13 Metadata performance with MDtest using empty files

The system gets very good results with Stat and Read operations reaching their peak value at 64 threads with ~486K op/s and at 1024 threads with ~186K op/s respectively. Removal and create operations attained the maximum value at 256 threads with ~116K op/s and ~105K op/s respectively. Once they reach their peak value, for stat and read operations the performance does not drop below 440K op/s for Stats and 182K op/s for Reads. The create operations show some variability after it reaches a plateau. However, the file removal operations remain more stable and does not decrease below 100K op/s.

5 Scalability of Dell EMC Ready Solutions for HPC BeeGFS Storage

The Dell EMC solution uses dedicated storage servers and a dual-purpose metadata and storage server to provide a high-performance, scalable storage solution[7]. It is possible to scale the system by adding additional storage/metadata servers to an existing system. In the following sub sections, we present the performance data when we increase the number of storage servers in the system.

5.1 Base Configurations

Figure 14 shows the two base configurations that have been tested and validated extensively at the Dell EMC HPC and AI Innovation Lab.

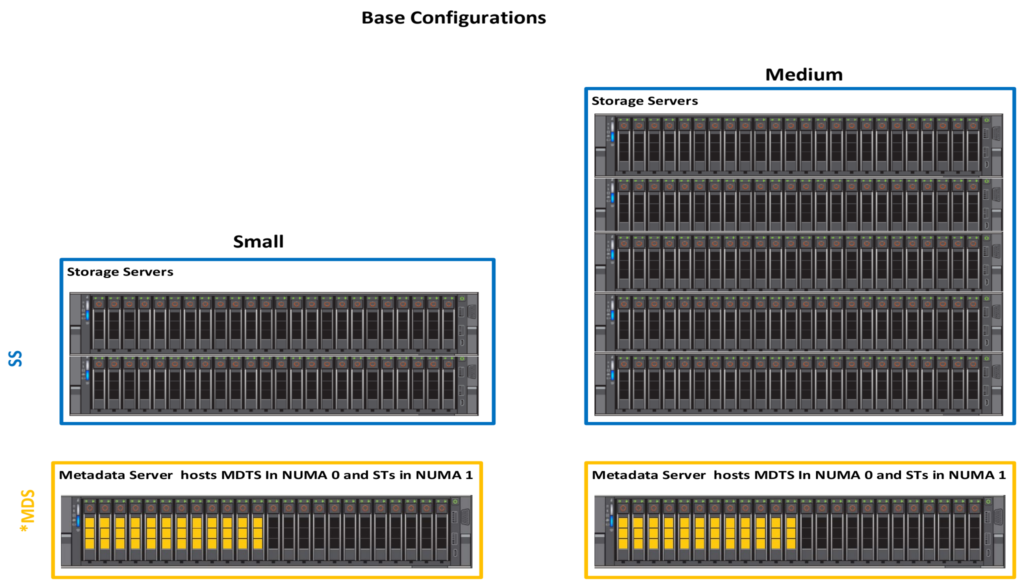
Base Configurations
Medium
Storage Servers
Small
Storage Servers
SS
Metadata Server hosts MDTS In NUMA 0 and STs in NUMA 1
*MDS
Metadata Server hosts MDTS In NUMA 0 and STs in NUMA 1

Figure 14 Base Configurations

The small configuration consists of three R740xd servers. It has a total of 15 storage targets. The medium configuration has 6xR740xd servers and has a total of 33 storage targets. The user can start with a "Small" configuration or with the "Medium" configuration and can add storage or metadata servers as needed to increase storage space and overall performance, or number of files and metadata performance, respectively. Table 6 shows the performance data for the base configurations.

Table 6 Capacity and Performance Details of Base Configurations

Component		Small	Medium
Total U (MDS+SS)		6U	12U
# of Dedicated Storage Servers		2	5
# of NVMe Drives for data storage		60	132
Estimated Usable Space	1.6 TB	86 TiB	190 TiB
	3.2 TB	173 TiB	380 TiB
	6.4 TB	346 TiB	761 TiB
Peak Sequential Read		60.1 GB/s	132.4 GB/s
Peak Sequential Write		57.7 GB/s	120.7 GB/s
Random Read		1.80 Million IOPS	3.54 Million IOPS
Random Write		1.84 Million IOPS	3.59 Million IOPS

5.2 BeeGFS Usable Space Calculation

Estimated usable space is calculated in TiB (since most tools show usable space in binary units) using the following formula:

BeeGFS Usable Space in TiB= (0.99* # of Drives* size in TB * (10^12/2^40)

In this formula, 0.99 is the factor arrived at by assuming conservatively that there is a 1% overhead from the file system. For arriving at the number of drives for storage, 12 drives from the MDS are also included. This is because, in the MDS, the 12 drives in NUMA zone 0 are used for metadata and the 12 drives in the NUMA zone 1 are used for storage. The last factor in the formula 10^12/2^40 is to convert the usable space from TB to TiB.

5.3 Scalable Configurations

The BeeGFS High Performance Storage Solution has been designed to be flexible and you can easily and seamlessly scale performance and capacity by adding additional servers.

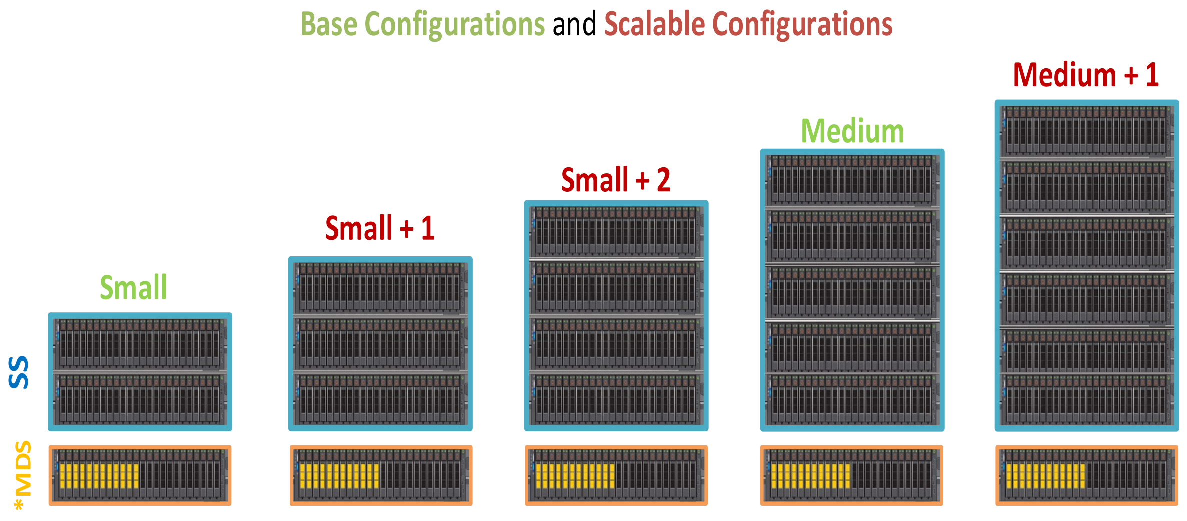

Figure 15 Scalable Configurations

The metadata portion of the stack remains the same for all the above configurations described in this blog. This is because the storage capacity requirements for BeeGFS metadata are typically 0.5% to 1% of the total storage capacity. However, it really depends on the number of directories and files in the file system. The guideline is that the user can add an additional metadata server when the percentage of metadata capacity to the storage falls below 1%. Table 7 shows the measured performance data for the different flexible configurations of the BeeGFS Storage Solution. Additional details are provided in Section 5.4.

Table 7 Capacity and Performance Details of Scaled Configurations

Component		Small	Small + 1	Small +2	Medium	Medium +1
Total U (MDS+SS)		6U	8U	10U	12U	14U
# of Dedicated Storage Servers		2	3	4	5	6
# of NVMe Drives for data storage		60	84	108	132	156
Estimated Usable Space	1.6 TB	86 TiB	121 TiB	156 TiB	190 TiB	225 TiB
	3.2 TB	173 TiB	242 TiB	311 TiB	380 TiB	449 TiB
	6.4 TB	346 TiB	484 TiB	622 TiB	761 TiB	898 TiB
Peak Sequential Read		60.1 GB/s	83.3 GB/s	105.2 GB/s	132.4 GB/s	152.9 GB/s
Peak Sequential Write		57.7 GB/s	80.3 GB/s	99.8 GB/s	120.7 GB/s	139.9 GB/s

5.4 Performance Characterization of Scalable Configurations

The performance of the various configurations was tested by creating storage pools. The small configuration has 15 storage targets and each additional storage server adds an additional six storage targets. So, for the purpose of testing the performance of the various configurations, storage pools were created from 15 to 39 storage targets (increments of six for small, small+1, small+2, medium, medium+1). For each of those pools, three iterations of IOzone benchmark were run, each with one to 1024 threads (in powers of two increments).

The testing methodology adopted is the same as that described in Section 4.1. Figures 16 and 17 show the write and read performance of the scalable configurations respectively, with the peak performance of each of the configurations highlighted for ready reference:

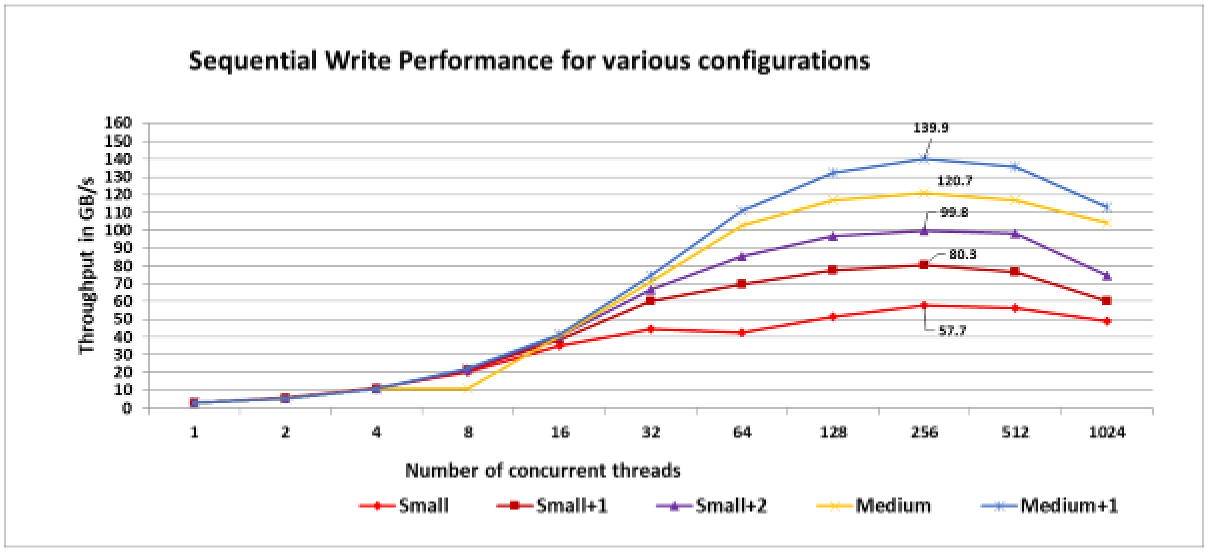

Figure 16 Write performance of Scalable Configurations

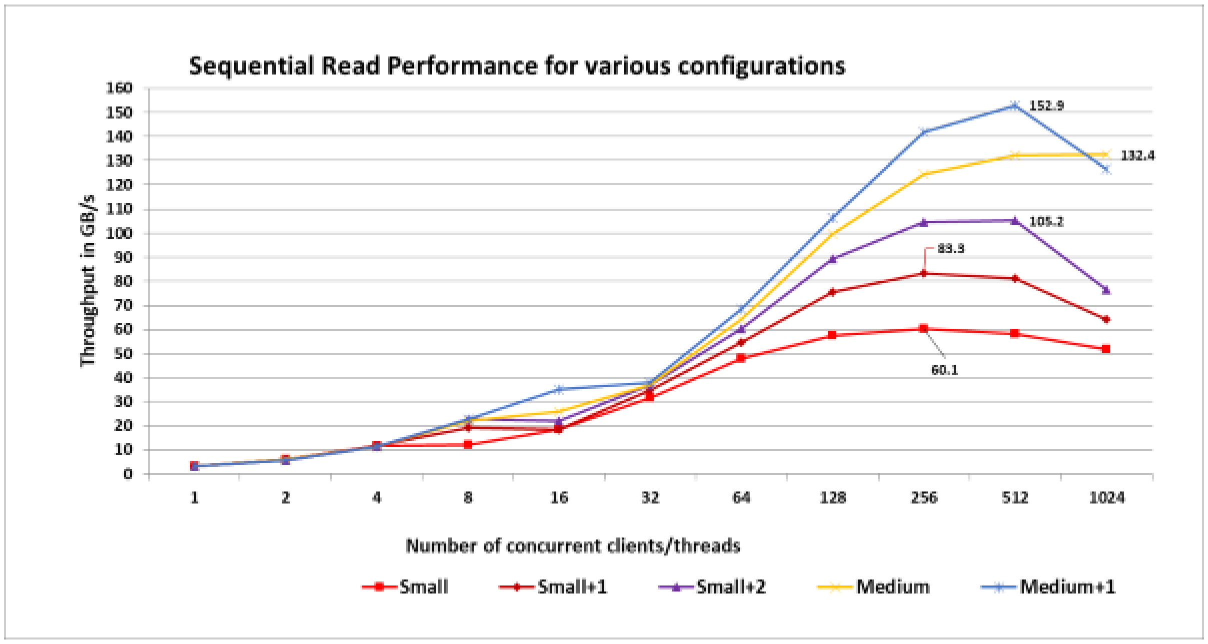

Figure 17 Read performance of Scalable Configurations

We see that the read and write performance are similar for the various configurations for lower thread counts but later the performance scales almost linearly.

Note: The storage pools referred to were created only for the explicit purpose of characterizing the performance of different configurations. While doing the performance evaluation of the medium configuration detailed in Section 4.1, all the 33 targets were in the "Default Pool" only. The output of the `beegfs-ctl --liststoragepools` command given below shows the assignment of the storage targets:

```
# beegfs-ctl --liststoragepools
Pool ID Pool Description Targets Buddy Groups
======= ================== ============================
============================
1 Default 1,2,3,4,5,6,7,8,9,10,11,12,
13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,
31,32,33
```

A Benchmarks and test tools

1) The IOzone benchmark tool was used to measure sequential N to N read- and write throughput (GB/s) and random read- and write I/O operations per second (IOPS).
2) The IOR benchmark tool was used to measure sequential N to 1 read- and write throughput (GB/s).
3) The MDtest benchmark was used for files only (no directories metadata), to get the number of creates, stats, reads and removes the solution can handle when using empty files.
4) Intel Data Center Tool was used for Management and Firmware update of Intel P4600NVMe SSDs.

The following sub sections 1 to 3 provide the command line reference and describe the various options used in the commands to run the respective benchmarks. The sub section 4 describes how the Intel Data Center Tool was used for management of the NVMe drives.

A.1 IOzone

The `IOzone` tests were run from 1–32 nodes in clustered mode. All tests were N-to-N. Meaning, N client threads would read or write N independent files. The command used to run the IOzone benchmarks are given below:

IOzone Sequential Tests

Sequential Writes

```
iozone -i 0 -i 1 -c -e -w -r 1m -I -s $Size -t $Thread -+n -+m
/path/to/threadlist
```

Sequential Reads

```
iozone -i 0 -i 1 -c -e -w -r 1m -I -s $Size -t $Thread -+n -+m
/path/to/threadlist
```

By using `-c` and `-e` in the test, `IOzone` provides a more realistic view of what a typical application is doing.

IOzone Random Writes/ Reads

```
iozone -i 2 -w -c -O -I -r 4K -s $Size -t $Thread -+n -+m /path/to/threadlist
```

The `O_Direct` command line parameter allows us to bypass the cache on the compute node on which we are running the `IOzone` thread. The following table describes `IOzone` command line arguments.

Table 8 Appendix A — IOzone command line arguments

IOzone Argument	Description
-i 0	Write test
-i 1	Read test
-i 2	Random Access test
-+n	No retest
-c	Includes close in the timing calculations

IOzone Argument	Description
-t	Number of threads
-e	Includes flush in the timing calculations
-r	Records size
-s	File size
-t	Number of threads
-+m	Location of clients to run IOzone when in clustered mode
-w	Does not unlink (delete) temporary file
-I	Use O_DIRECT, bypass client cache
-O	Give results in ops/sec

A.2 IOR

The command used to run the test is given below:

```
mpirun -machinefile $hostlist --map-by node -np $threads -mca btl openib,self
~/bin/ior -w -r -i 3 -d 3 -e -O beegfsNumTargets=33,beegfsChunkSize=$record_size
-o $working_dir/ior.out -s 1 -g -t $record_size -b $file_size
```

The following table describes the `IOR` command-line arguments.

Table 9 Appendix A — IOR command line arguments

BeeGFS Tuning Parameter	**Description**
beegfsNumTargets	Number of storage targets to use for striping
beegfsChunkSize	Striping chunk size in bytes (Accepts k=kilo, m=mega, g=giga etc)
mpirun Argument	Description
-machinefile	Tells `mpirun` where the hostfile is located
-map-by node	Launch processes one per node, cycling by node in a round robin fashion. This spreads processes evenly among nodes and assigns MPI_COMM_WORLD ranks in a round-robin, “by-node” manner.
-np	Number of processes
IOR Argument	Description
-w	Write benchmark
-r	Read benchmark
-i	Number of repetitions
-d	Delay between repetitions in seconds
-e	perform fsync upon POSIX write close (make sure reads are only started after all writes are done)

BeeGFS Tuning Parameter	Description
-O	String of IOR directives
-o	path to file for the test
-s	Segment count
-g	intraTestBarriers - use barriers between open, write/read, and close
-t	Transfer size
-b	Block size (amount of data for a process)

A.3 MDtest

`MDtest` is used with `mpirun`. For these tests, OpenMPI version < > was used. The command used to run the MDtest is given below:

```
mpirun -machinefile $hostlist --map-by node -np $threads ~/bin/mdtest -i 3 -b
$Directories -z 1 -L -I 1024 -y -u -t -F
```

The following table describes the `MDtest` command-line arguments.

Table 10 Appendix A — MDtest command line arguments

mpirun Argument	Description
-machinefile	Tells `mpirun` where the hostfile is located
-map-by node	Launch processes one per node, cycling by node in a round robin fashion. This spreads processes evenly among nodes and assigns MPI_COMM_WORLD ranks in a round-robin, "by-node" manner.
-np	Number of processes
Mdtest Argument	Description
-d	The directory MDtest should run in
-i	The number of iterations the test will run
-b	Branching factor of directory structure
-z	Depth of the directory structure
-L	Files only at leaf level of tree
-I	Number of files per directory tree
-y	Sync the file after writing
-u	Unique working directory for each task
-C	Create files and directories
-R	Randomly stat files
-T	Only stat files and directories
-r	Remove files and directories left over from run

A.4 Intel Data Center Tool

The Intel Data Center Tool was used to reformat the Intel P4600 NVMe devices with 512b blocks for metadata and 4k blocks for storage as shown below:

# On servers with meta on NUMA 0 and storage on NUMA 1

```
for i in 0 11 {16..23} 1 2 ; do isdct start -f -intelssd $i -nvmeformat
LBAFormat=0 SecureEraseSetting=0 ; done
```

```
for i in {3..10} {12..15} ; do isdct start -f -intelssd $i -nvmeformat
LBAFormat=1 SecureEraseSetting=0 ; done
```

# On servers with all storage

```
for i in {0..23} ; do isdct start -f -intelssd $i -nvmeformat LBAFormat=1
SecureEraseSetting=0 ; done
```

However, before formatting the devices, one must confirm which devices are on NUMA zone 0 and which ones are in NUMA zone 1, using the `isdct show -a` command. This is because the device IDs do not match the kernel scan order. Moreover, the LBAFormat is also device specific.

The server needs to be rebooted after changing the low-level NVMe block sizes. The following command can be used to check the details of the devices.

```
for i in nvme{0..23} ; do echo -n $i " " ; grep -h .
/sys/class/nvme/$i/{address,*/numa_node,*/*/hw_sector_size} | xargs ; done |
column -t
```

# Output on an R740XD w/ meta + storage:

```
nvme0  0000:62:00.0  0  512
nvme1  0000:63:00.0  0  512
nvme2  0000:64:00.0  0  512
nvme3  0000:65:00.0  0  512
nvme4  0000:66:00.0  0  512
nvme5  0000:67:00.0  0  512
nvme6  0000:68:00.0  0  512
nvme7  0000:69:00.0  0  512
nvme8  0000:6a:00.0  0  512
nvme9  0000:6b:00.0  0  512
nvme10  0000:6c:00.0  0  512
nvme11  0000:6d:00.0  0  512
nvme12  0000:b3:00.0  1  4096
nvme13  0000:b4:00.0  1  4096
nvme14  0000:b5:00.0  1  4096
nvme15  0000:b6:00.0  1  4096
nvme16  0000:b7:00.0  1  4096
nvme17  0000:b8:00.0  1  4096
nvme18  0000:b9:00.0  1  4096
```

```
nvme19 0000:ba:00.0 1 4096
nvme20 0000:bb:00.0 1 4096
nvme21 0000:bc:00.0 1 4096
nvme22 0000:bd:00.0 1 4096
nvme23 0000:be:00.0 1 4096
```

B Technical support and resources

1 — NVM Express Explained: https://nvmexpress.org/wp-content/uploads/2013/04/NVM_whitepaper.pdf

2 — Dell EMC Ready Solutions for HPC BeeGFS Storage: https://www.dell.com/support/article/sln319381/

3 — BeeGFS Documentation: https://www.beegfs.io/wiki/

4 — General Architecture of BeeGFS File System: https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf

5 — ext4 file system for metadata targets: https://www.beegfs.io/docs/whitepapers/Metadata_Performance_Evaluation_of_BeeGFS_by_ThinkParQ.pdf

6 — PCI Express Direct Memory Access Reference Design using External Memory: https://www.intel.com/content/www/us/en/programmable/documentation/nik1412547570040.html#nik1412547565760

7 — Scalability of Dell EMC Ready Solutions for HPC BeeGFS Storage: https://www.dell.com/support/article/sln319382/

8 — Dell.com/support: Focused on meeting customer needs with services and support for Dell EMC products.

9 — hpcatdell.com: Provides a consolidated list of all the publications from the HPC Engineering Team at the Dell EMC HPC and AI Innovation Lab.