Technical White Paper

Dell EMC Ready Solution for HPC PixStor Storage

Dell EMC HPC Solutions

Abstract

This white paper describes the architecture of the PixStor including its optional components for capacity expansion, NVMe tier and Gateways, along with performance characterization for the different components.

June 2020

Document ID

Revisions

Date	Description
July 2020	Initial release

Acknowledgements

Author: J. Mario Gallegos – HPC and AI Innovation Lab

Table of contents

Executive summary

In High-Performance Computing (HPC), all its different components need to be balanced to keep optimal performance and avoid bottlenecks. Evolution of compute nodes can make storage a bottleneck, that is normally avoided by the use of Parallel File Systems (PFS) that can scale out to meet such demands. With recent advances in storage technologies like Non-Volatile Memory Express (NVMe) SSDs, more options are available and added to PFS storage system.

A well-balanced storage system is required to achieve optimal performance, therefore its components like back-end arrays, storage controllers, disk drives, IO cards, network adapters and switches must be able to provide similar bandwidth and distribute the load among the processors and memory of the servers they use. The PFS and supporting software must be able to balance the load on the different storage system components distributing data among them, monitoring performance along the health of the components and provide administration tools to manage the solution efficiently.

The PixStor solution is highly available (HA), based on Dell EMC PowerEdge 14G servers and PowerVault ME4 storage arrays arranged in storage modules that can scale up its capacity to 8 PB of formatted storage and can scale out to customer requirements by adding more of those modules to the required capacity. For situations that require very high throughput, an NVMe tier can scale out performance by adding pairs of HA nodes with NVMe devices.

The Dell EMC Ready Solution for HPC PixStor Storage is a storage solution based on a well-established parallel file system that can be geo-distributed with components that simplify administration, expand its connectivity, add advance search capabilities, and allow access to other storage devices both on and off premises and using cloud protocols and other enterprise protocols. This storage solution is fully supported (hardware and software), easy-to-use, high-throughput, multi-tiered and it is offered with deployment services.

Solution Architecture

Introduction

Today's HPC environments have increased demands for very high-speed storage and with the higher count CPUs, faster networks and bigger and faster memory, storage was becoming the bottleneck in many workloads. Those high demand HPC requirements are typically covered by Parallel File Systems (PFS) that provide concurrent access to a single file or a set of files from multiple nodes, very efficiently and securely distributing data to multiple LUNs across several servers.

Those files systems are normally spinning media based to provide the highest capacity at the lowest cost. However, more and more often, the speed and latency of spinning media cannot keep up with the demands of many modern HPC workloads, requiring the use of flash technology in the form of burst buffers, faster tiers, or even very fast scratch, local or distributed. The **DellEMC Ready Solution for HPC PixStor Storage** uses NVMe nodes as the component to cover such new high bandwidth demands in addition to being flexible, scalable, efficient, and reliable.

In addition, very frequently data cannot be accessed using the native PFS clients normally used to access data, but instead other protocols like NFS or SMB must be used. Point in case is when customers require access data from workstations or laptops with MS-Windows or Apple macOS, or research/production systems that only offer connectivity via standard protocols The DellEMC Ready Solution for HPC PixStor Storage uses Gateway nodes as the component to allow such connectivity in a scalable, efficient, and reliable way.

Furthermore, storage solutions frequently require access to other storage devices (local or remote) to move data to and from those devices, where the PixStor Gateway is not appropriate for many cases (NFS or SMB not supported in those devices) or when it is highly desirable, to integrate those devices as another tier (e.g. Tape libraries, Object Storage, Cloud storage, etc.). Under those circumstances, the PixStor Solution can provide tiered access to other devices using enterprise protocols, including cloud protocols, using the Ngenea node with ArcaStream proprietary software to allow that level of integration while staying very cost effective.

Architecture

This whitepaper describes one of DellEMC storage solutions for HPC environments, the DellEMC Ready Solution for HPC PixStor Storage. **Figure 1** presents the reference architecture, which leverages DellEMC PowerEdge R440, R640 and R740 servers and the PowerVault ME4084 and ME4024 storage arrays, with the PixStor software from our partner company Arcastream. In addition, optional PowerVault ME484 EBOD arrays can be used to increase the capacity of the solution. **Figure 1** presents the reference architecture depicting such capacity expansion SAS additions to the existing PowerVault ME4084 storage arrays.

PixStor Software includes the widespread General Parallel File System (GPFS) also known as Spectrum Scale as the PFS component which is considered as a software defined storage due to its flexibility and scalability. In addition, PixStor Software includes many other Arcastream software components like advanced analytics, simplified administration and monitoring, efficient file search, advanced gateway capabilities and many others.

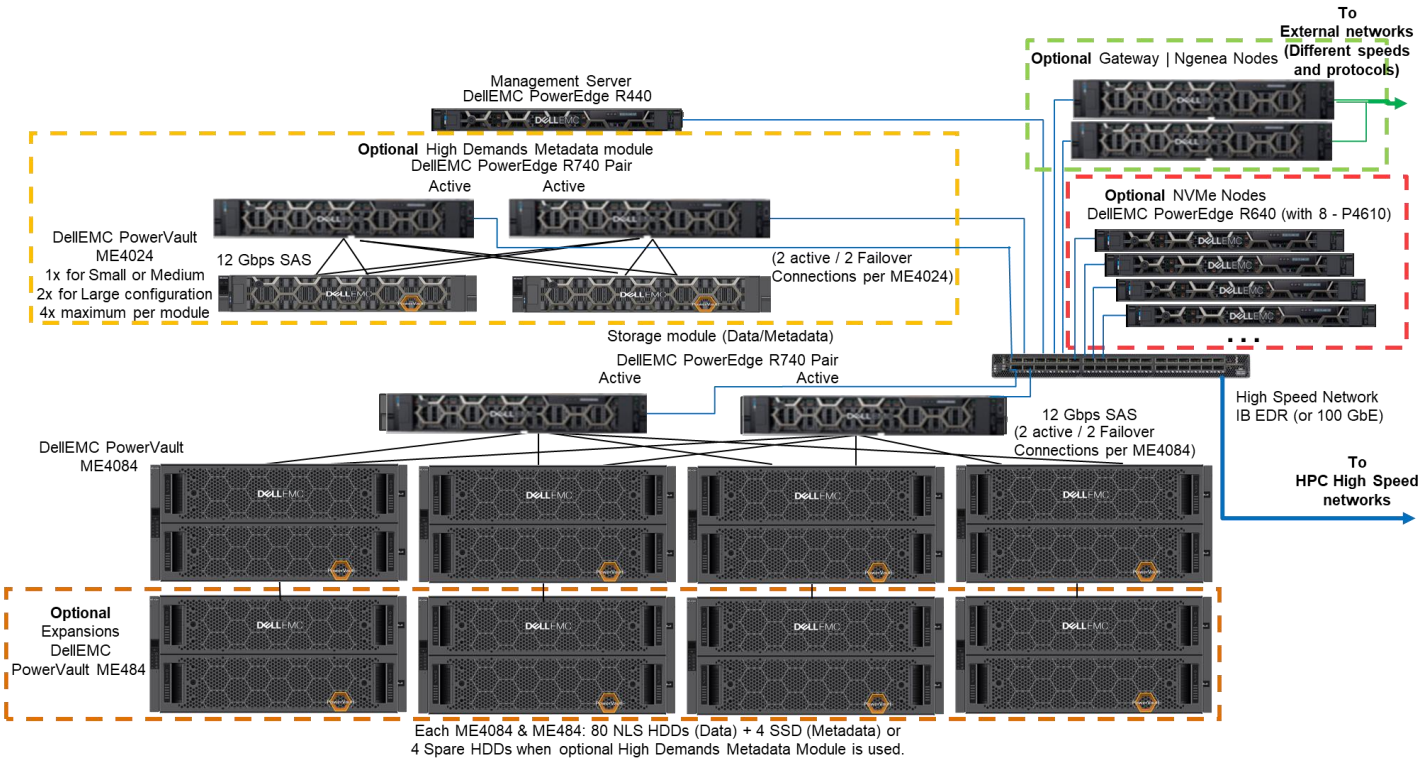


Figure 1 Reference Architecture

The main components of the PixStor solution are:

Network Shared Disks (NSDs)

Back end block devices (i.e. RAID LUNs from ME4 arrays, RAID 10 NVMeoF devices) that store information, data and metadata. In the PixStor solution file system data and metadata are stored in different NSDs, data NSDs normally use spinning media (NLS SAS3 HDDs), while metadata NSDs use SAS3 SSDs (Metadata include directories, filenames, permissions, time stamps and the location of data).
NVMeoF based NSDs are currently used only for data, but there are plans to use them also for data + metadata or even for metadata only.

Storage Server (SS)

Part of the **Storage Module**, pairs of PowerEdge R740 servers in HA (failover domains) connected to ME4084s arrays via SAS cables, manages the data NSDs and provides access to the backend data. For the standard configuration these servers have the dual role of Metadata Servers and also manage metadata NSDs (on SSDs that replace spare HDDs).

High Demand Metadata Server (HDMDS)

Part the **optional High Demand Metadata Module** (inside dotted yellow square in **Figure 1**). Pairs of PowerEdge R740 servers in HA (failover domains) connected to ME4024s arrays via SAS cables, manages the metadata NSDs and provides access to the backend data.

Backend Storage

Stores the file system data (MD4084) or metadata (ME4024). ME4084s are part of the **Data Module** and ME4024s are part of the **optional High Demand Metadata Module** inError! Reference source not found..

Expansion Storage

Part of the **optional Capacity Expansions** (inside dotted orange square in **Figure 1**). ME484s connected behind ME4084s via SAS cables to expand the capacity of a Storage Module. For PixStor solutions each ME4084 can only use one ME484 expansion, for performance and reliability (even that ME4084 supports up to 3).

Management Servers

PowerEdge R440 servers provide GUI and CLI access for management and monitoring of the PixStor solution, as well as performing the advanced search capabilities compiling some metadata in a database to speed up searches and avoid loading Metadata NSDs.

NVMe Nodes

Part of the **optional NVMe Tier Modules** (inside dotted green square in **Figure 1**). Pairs of PowerEdge R640 servers in HA (failover domains) provide a very high-performance flash-based tier for the PixStor solution. Performance and capacity for this NVMe tier can be scaled out by additional pair of NVMe nodes. Increased capacity is provided by selecting the appropriate NVMe devices supported in the PowerEdge R640. Each R640 has eight NVMe devices split in eight slices (think partitions). Then, slices from all drives in both servers are combined into eight RAID10 devices, for very high throughput. These NVMe nodes use NVMesh as the NVMe over Fabric (NVMeoF) means to have each mirror copy from the RAID10 on a different server and provide block devices to the file system to use as NSDs.

Native Client Software

Software installed on the clients to allow access to data on the file system. The file system appears as a single namespace that can be mounted for access.

Gateway Nodes

The **optional Gateway Nodes** (inside dotted red square in **Figure 1**) are PowerEdge R740 servers (same hardware as Ngenea nodes but different software) in a Samba's Clustered Trivial Data Base (CTDB) cluster providing NFS or SMB access to clients that do not have or cannot have the native client software installed, but instead use NFS or SMB protocols to access information.

Ngenea Nodes

The **optional Ngenea Nodes** (inside dotted red square in **Figure 1**) are PowerEdge R740 servers (same hardware as Gateway nodes but different software) that use Arcastream software to access to external storage devices that could be used as another tier (e.g. Tape libraries, Object Storage, Cloud storage, etc.) using enterprise protocols, including cloud protocols.

Management Switch

PowerConnect S3048-ON gigabit ethernet switch used to connect the different servers and storage arrays. It is used for administration of the solution interconnecting all the components.

High Performance Switch

Mellanox SB7800 Switches to provide high speed access via Infiniband (IB) EDR or 100 GbE.

Solution Components

This solution was planned to be released with the latest Intel Xeon 2nd generation Scalable Xeon CPUs, a.k.a. Cascade Lake CPUs and some of the servers will use the fastest RAM available to them (2933 MT/s). However, due to current hardware available to work on the prototype of the solution to characterize performance, servers with Intel Xeon 1st generation Scalable Xeon CPUs a.k.a. Skylake processors and in some cases slower RAM were used to characterize this system. Since the bottleneck of the solution is at the SAS controllers of the DellEMC PowerVault ME40x4 arrays, no significant performance disparity is expected once the Skylake CPUs and RAM are replaced with the envisioned Cascade Lake CPUs and faster RAM.

Table 1 has the list of main components for the solution, but when discrepancies were introduced for the situation mentioned above, the first description column has for the planned component to be used at release time and therefore available to customers, and the last column is the component actually used for characterizing the performance of the solution. The drives listed or data (12TB NLS) and metadata (960Gb SSD), are the ones used for performance characterization, and faster drives can provide better Random IOPs and may improve create/removal metadata operations.

Software components listed in **Table 1** describe versions during the initial testing and at release time. However, these software versions change over time to include important bug fixes, support for new hardware components or add important new features. When software versions were updated before characterizing a new component, a small table with versions used will be included.

Finally, for completeness, the list of possible data HDDs and metadata SSDs was included, which is based on the drives supported as enumerated on the DellEMC PowerVault ME4 support matrix, available online.

Table 1 Components to be used at release time and those used in the Test bed

Solution Component		At Release	Test Bed
Internal Mgmt Connectivity		Dell Networking S3048-ON Gigabit Ethernet	
Data Storage Subsystem		1x to 4x Dell EMC PowerVault ME4084 1x to 4x Dell EMC PowerVault ME484 (One per ME4084) 80 – 12TB 3.5" NL SAS3 HDD drives Options 900GB @15K, 1.2TB @10K, 1.8TB @10K, 2.4TB @10K, 4TB NLS, 8TB NLS, 12TB NLS, 16TB NLS. 8 LUNs, linear 8+2 RAID 6, chunk size 512KiB. 4 - 1.92TB SAS3 SSDs for Metadata – 2x RAID 1 (or 4 - Global HDD spares, if Optional High Demand Metadata Module is used)	
Optional High Demand Metadata Storage Subsystem		1x to 2x Dell EMC PowerVault ME4024 (4x ME4024 if needed, Large config only) 24 – 960GB 2.5" SSD SAS3 drives (Options 480GB, 960GB, 1.92TB, 3.84TB) 12 LUNs, linear RAID 1.	
RAID Storage Controllers		Redundant 12 Gbps SAS	
Capacity w/o Expansion		Raw: 4032 TB (3667 TiB or 3.58 PiB) Formatted ~ 3072 GB (2794 TiB or 2.73 PiB)	
Capacity w/Expansion		Raw: 8064 TB (7334 TiB or 7.16 PiB) Formatted ~ 6144 GB (5588 TiB or 5.46 PiB)	
Proces	**Gateway/Ngenea**	2x Intel Xeon Gold 6230 2.1G, 20C/40T, 10.4GT/s, 27.5M Cache, Turbo, HT (125W) DDR4-2933	2x Intel Xeon Gold 6136 @ 3.0 GHz, 12 cores
	High Demand Metadata		

Solution Component		At Release	Test Bed
	Storage Node		
	NVMe Node		2x Intel Xeon Gold 6230 @ 2.1GHz, 20 cores
	Management Node	2x Intel Xeon Gold 5220 2.2G, 18C/36T, 10.4GT/s, 24.75M Cache, Turbo, HT (125W) DDR4-2666	2x Intel Xeon Gold 5118 @ 2.30GHz, 12 cores
Memory	**Gateway/Ngenea**	12 x 16GiB 2933 MT/s RDIMMs (192 GiB)	24 x 16GiB 2666 MT/s RDIMMs (384 GiB)
	High Demand Metadata		
	Storage Node		
	NVMe Node		12x 16GiB 2933 MT/s RDIMMs (192 GiB)
	Management Node	12 X 16GB DIMMs, 2666 MT/s (192GiB)	12 x 8GiB 2666 MT/s RDIMMs (96 GiB)
Operating System		CentOS 7.6	CentOS 7.5
Kernel version		3.10.0-957.12.2.el7.x86_64	3.10.0-862.14.4.el7.x86_64
PixStor Software		5.1.0.0	4.8.3
Spectrum Scale (GPFS)		5.0.3	5.0.3
OFED Version		Mellanox OFED 4.6-1.0.1.0	Mellanox OFED 4.3-3.0.2
High Performance Network Connectivity		Mellanox ConnectX-5 Dual-Port InfiniBand EDR/100 GbE, and 10 GbE	Mellanox ConnectX-5 InfiniBand EDR
High Performance Switch		2x Mellanox SB7800 (HA – Redundant)	1x Mellanox SB7790
Local Disks (OS & Analysis/monitoring)		All servers except Management node 3x 480GB SSD SAS3 (RAID1 + HS) for OS PERC H730P RAID controller Management Node 3x 480GB SSD SAS3 (RAID1 + HS) for OS & Analysis/Monitoring PERC H740P RAID controller	All servers except Management node 2x 300GB 15K SAS3 (RAID 1) for OS PERC H330 RAID controller Management Node 5x 300GB 15K SAS3 (RAID 5) for OS & Analysis/monitoring PERC H740P RAID controller
Systems Management		iDRAC 9 Enterprise + DellEMC OpenManage	iDRAC 9 Enterprise + DellEMC OpenManage

High-Speed, management and SAS connections

All the servers have the iDRAC dedicated port and the first 1GbE port (either LOM or NCD) are connected to the management switch.

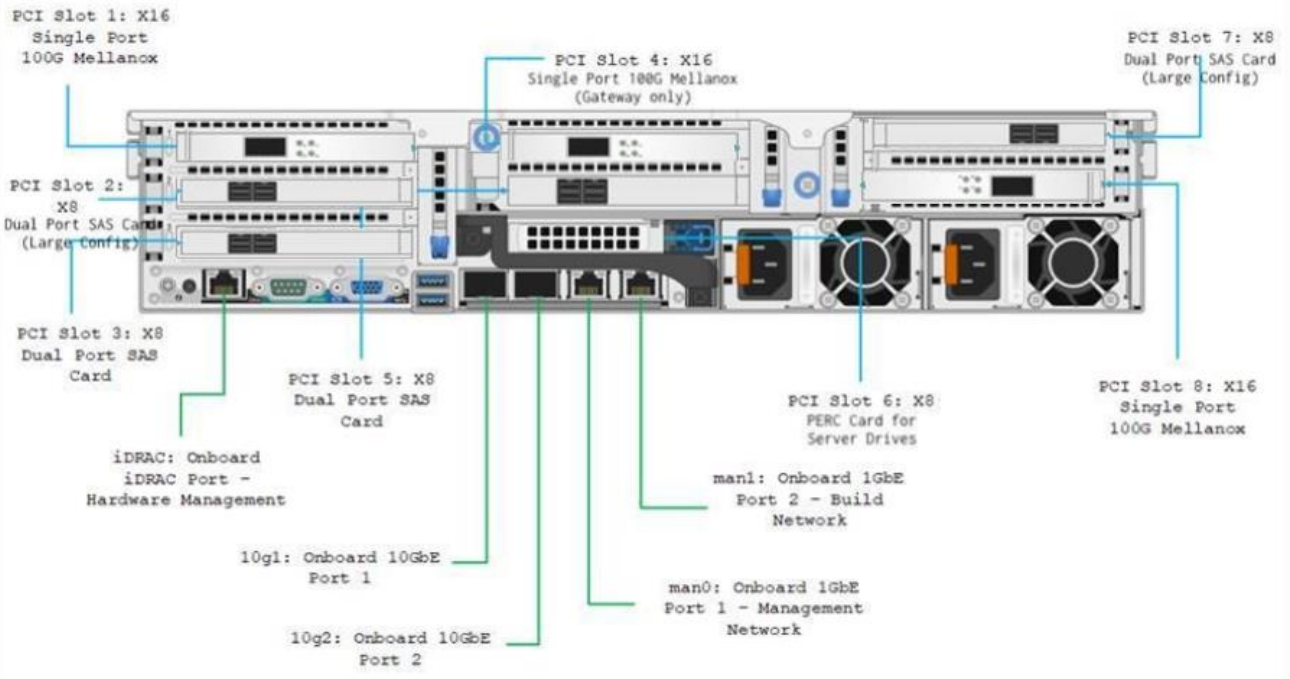
The PE R440 servers used as management servers only have two x16 slots used for CX5 adapters connected to the high-speed network switches.

The PE R640 servers used as NVMe nodes have 3 x16 slots, slots 1 and 3 are used for CX5 adapters connected to the high-speed network switches.

The PE R740 servers with the Riser configuration 6 have 8 slots, 3 x16 and 5 x8, **Figure 2** shows the slot

Allocation for the server. All R740 servers have slots 1 & 8 (x16) used for CX5 adapters connected to the high-speed network switches. Any Storage or HDMD server that is connected to one or two ME4 arrays only have two 12Gb SAS HBAs ins slots 3 & 5 (x8). Notice that slot 4 (x16) is only used for a CX5 adapter by the

Gateways/Ngenea nodes. Similarly, slots 2 & 7 (x8) are only used for the Storage Servers for Large configurations (4 ME4084s) or for High Demand Metadata servers that require 4 ME4024s.



Slot Number	Speed	Card Type
1	x16	Single Port 100G Mellanox
2	x8	Dual Port SAS Card (Large Config)
3	x8	Dual Port SAS Card
4	x16	Single Port 100G Mellanox (Gateway only)
5	x8	Dual Port SAS Card
6	x8	Perc Card for Server Drives
7	x8	Dual Port SAS Card (Large Config)
8	x16	Single Port 100G Mellanox

Figure 2 R740 Slot Allocation

To give an idea how the solution connects its SAS and high-speed links to avoid single points of failure and allow maximum performance to the different components, **Figure 3** shows such connections for a large configuration with the optional High Demand Metadata module included. Since the **DellEMC Ready Solution for HPC PixStor Storage** is sold with deployment services include, details explanation of the cabling is out of scope for this work. Similarly, the cabling of the different components to the management switch is also out of scope for this document.

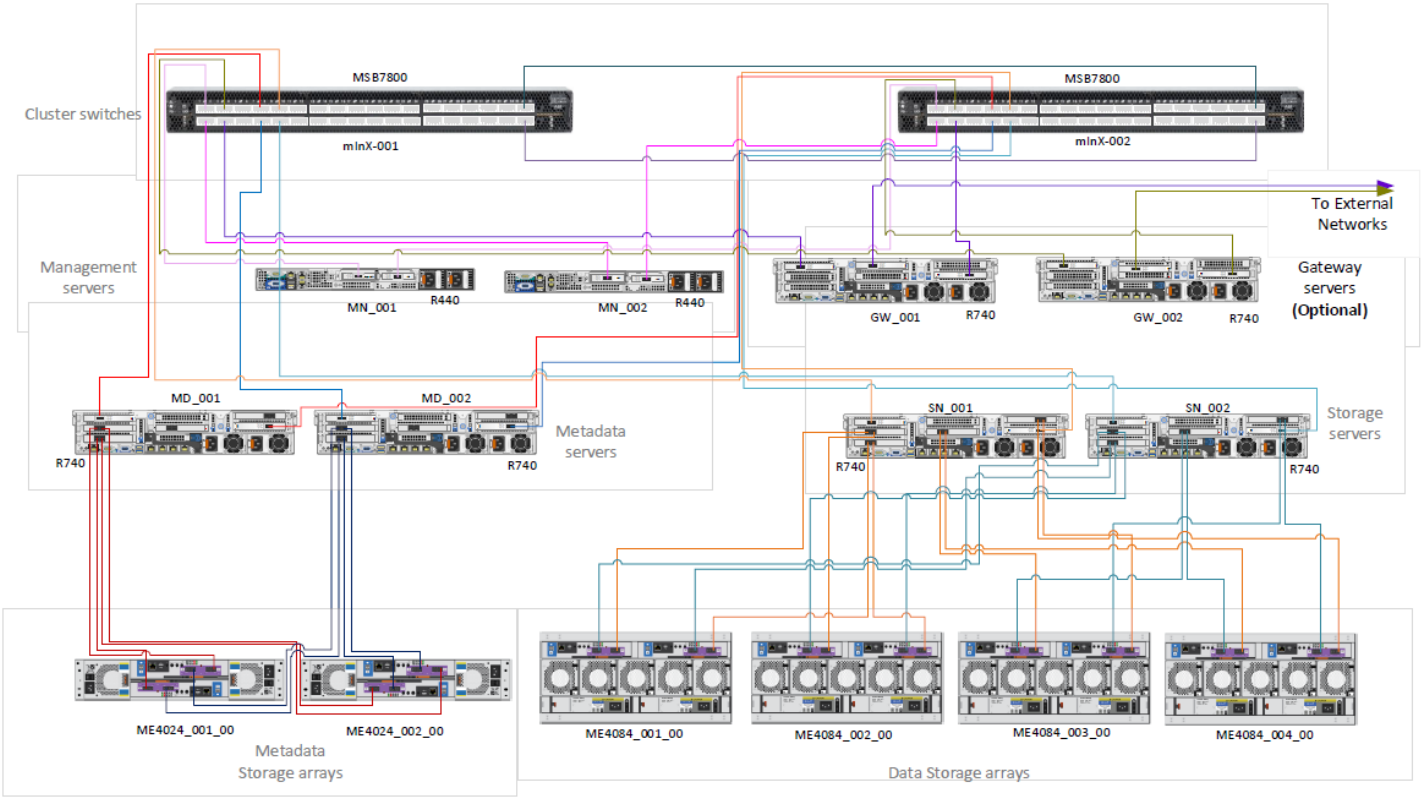


Figure 3 SAS & High-speed cable diagram

Storage configuration on ME4 arrays.

The **DellEMC Ready Solution for HPC PixStor Storage** has two variants, the **standard configuration** and the one that includes the High **Demand Meta Data module**. On the **standard configuration** the same pair of R740 servers use their ME4084 arrays to store DATA on NLS SAS3 HDDs and Metadata on SAS3 SSDs. On **Figure 4** we can see this ME4084 configuration showing how are the drives assigned to the different LUNs. Notice that each ME4084 has eight linear RAID6 (8 data disks + 2 parity disks) Virtual Disks used for DATA only where HDDs are selected alternating even numbered disk slots for one LUN, and odd numbered disk slots for the next LUN, and that pattern repeats until all 80 NLS disks are used. The last four disk slots have SAS3 SSDs, which are configured as two linear RAID 1 pairs, and only hold Meta Data. Linear RAIDs were chosen over virtual RAIDs to provide the maximum possible performance for each virtual disk. Similarly, RAID 6 was chosen over ADAPT, in spite of the advantage the later has when rebuilding after failures.

When the Storage Module has a single ME4084, then GPFS is instructed to replicate the first RAID 1 on the second RAID 1 as part of a failure group. However, when the Storage module has 2 or 4 ME4084s, those arrays are divided in pairs and GPFS is instructed to replicate each RAID 1 on one array on the other array, using different failure groups. Therefore, each RAID 1 always have a replica managed by a GPFS's failure group.

All the virtual disks have associated volumes that span their whole size, and are mapped to all ports, so that they are all accessible to any HBA port from the two R740s connected to them. Also, each R740 has one HBA port connected to each ME4084 controller from their storage arrays. Such that even if one server is

operational and only a single SAS cable remains connected to each ME4084, the solution can still provide access to all data stored in those arrays.

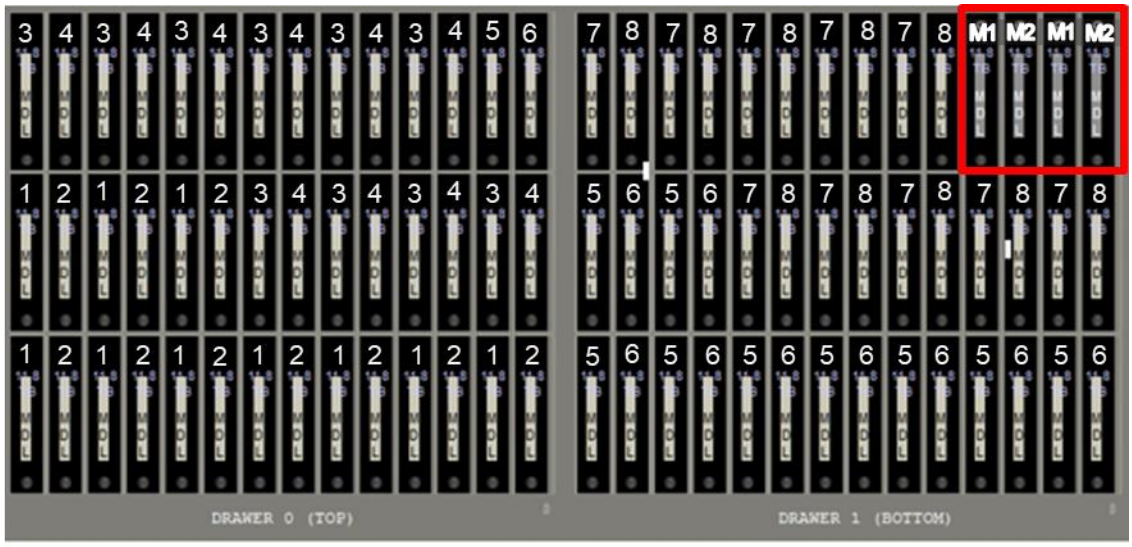
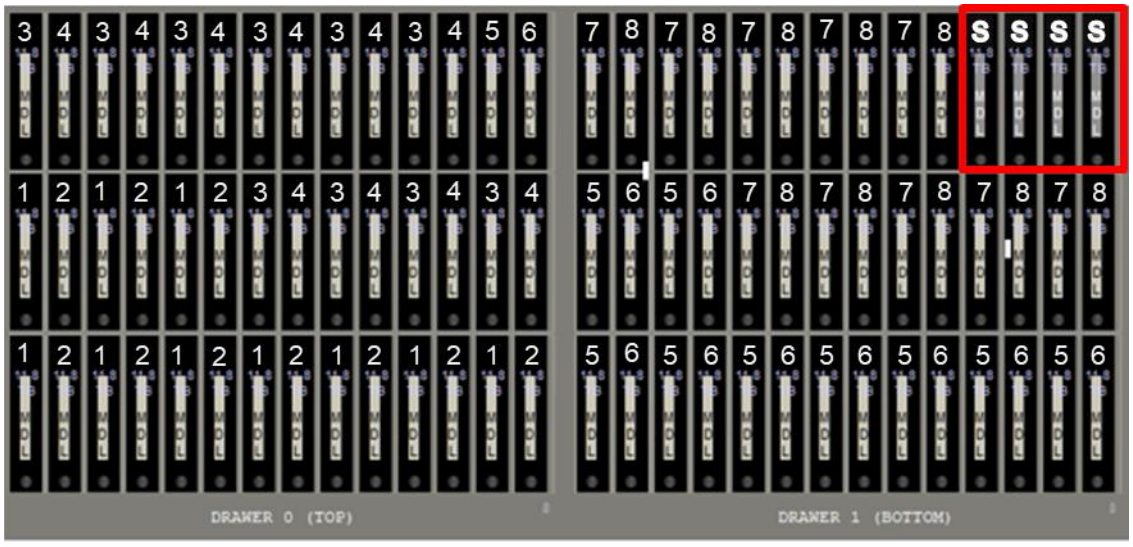


Figure 4 ME4084 drives assigned to LUN for Standard Configuration.

When the optional **High Demand Meta Data** module is used, the eight RAID 6 are assigned just like the standard configuration and are also used only to store data. However, instead of SSDs, the last four disk slots have NLS SAS3 HDDs to be used as hot spares for any failed disk within the array, see **Figure 5**.



.

Figure 5 ME4084 drives assigned to LUN for Configuration with High Demand Meta Data.

An additional pair of R740 servers are connected to one or more ME4024, which are used to store metadata. Those arrays are populated with SAS3 SSDs that are configured as twelve RAID 1 Virtual Disks (VDs), and disks for such VDs are selected starting in disk slot 1 and for its mirror moving twelve disks to the right (slot N & Slot N+12), as can be seen in **Figure 6**. Similar to the standard configuration, when a single ME4024 array is used, GPFS is instructed to replicate metadata using pairs of RAIDs 1 LUNs as part of a failure group. However, if more than one ME4024 is used, then LUNs part of failure groups are kept in separate arrays.

All the virtual disks on the storage module and HDMD module are exported as volumes that are accessible to any HBA port from the two R740s connected to them, and each R740 has one HBA port connected to each ME4 controller from their storage arrays. Such that even if one server is operational and only a single SAS cable remains connected to each ME4, the solution can still provide access to all data (or metadata) stored in those arrays.

Figure 6 ME4024 drives assigned to LUN for Configuration with High Demand Meta Data.

Finally, high-speed networks are connected via CX5 adapters to handle information exchange with clients, but also to evaluate if a node part of a module is operational.

NVMe Tier configuration

Each PE R640 has 8 NVMe devices directly connected to the CPU in Socket 2 (so this is not a balanced configuration in terms of NUMA domains), and two HCAs Mellanox ConnectX-6 Single Port VPI HDR100 adapters (one per CPU socket). For the configuration characterized, Dell P4610 devices were used, since they have the same read and write performance for large blocks (3.2 GB/s) and fairly good random IO performance for small blocks, which are nice features when trying to scale and estimate the number of pairs needed to meet the requirements of this flash tier. Nevertheless, any NVMe device supported on the PowerEdge R640 will be supported for the NVMe nodes.

Those NVMe drives are configured as eight RAID 10 devices across a pair of servers, using NVMesh as the NVMe over Fabric component to allow data redundancy not only at the devices level, but at the server level. In addition, when any data goes into or out of one of those RAID10 devices, all 16 drives in both servers are used, increasing the bandwidth of the access to that of all the drives. Therefore, the only restriction for these NVMe tier servers is that they must be sold and used in pairs.

The R640 tested in this configuration were used with EDR 100 Gb IB connections, but since they already have CX6 adapters, the NVMe nodes are ready to support HDR100 speeds when used with HDR cables and switches. Testing HDR100 on these nodes is deferred as part of the HDR100 update for the whole PixStor solution. Both CX6 interfaces are used to sync data for the RAID 10 (NVMe over fabric) and as the connectivity for the file system. In addition, they provide hardware redundancy at the adapter, port and cable. For redundancy at the switch level, dual port CX6 VPI adapters are required, but need to be procured as S&P components.

To characterize the performance of NVMe nodes, from the system depicted in **figure 1**, only the high demand metadata module and the NVMe nodes were used.

Gateway Nodes

Gateway nodes use the same PE R740 as other servers in the solution, which have eight PCIe slots, three x16 and five x8. The three x16 slots have Mellanox ConnectX-5 Single Port VPI adapters, that can be configured for either IB EDR 100 Gb or Ethernet connections at any of the speeds supported by those

adapters, at least one of those adapters must be connected to the PixStor solution to get access to the file system and any information it has stored (two connections if redundancy is required on a single gateway). In addition, the gateways can be connected to other networks adding NICs supported by the PowerEdge R740 on the four x8 slots available (one x8 slot is used by a PERC adapter to manage local SSDs for the OS).

The Samba's Clustered Trivial Data-Base (CTDB) is a clustered database used to manage the NFS and SMB services on the gateway nodes, providing high availability, load balancing and monitoring of the nodes in the CTDB cluster. For each of the gateways in the CTDB cluster, a Domain Name System (DNS) entry with an A record for their IP is added, such that all have the same hostname, a sort of "public Gateway name." That Gateway name is then used by clients to mount those services, that way the name server daemon (named) can assign all the gateways in the CTDB cluster to clients in a round robin fashion. When needed, NFS-Ganesha (an open source, user space, NFS file server) can be used as an alternative to the regular NFS server services, and it is also managed by the CTDB cluster.

Behind the Gateways, a PixStor system must be accessed and exported to the clients. For characterizing the gateways in this work, a PixStor solution with high demand metadata and the capacity expansion modules was used. That is:

- For Storing metadata, 2x PE R740 servers were connected to both controllers of a single ME4024 fully populated with 960 GB SAS3 SSDs.
- For storing data 2x PE R740 servers were connected to all controllers of 4x DellEMC PowerVault (PV) ME4084 disk arrays, each of them with an expansion box, for a total of 4x PV ME484. All arrays fully populated with 12TB SAS3 NLS HDDs.

Ngenea Nodes

The hardware for Ngenea nodes is exactly the same as for the Gateway nodes, but has different software installed and requires a different license. Since these nodes were not tested at the time of publishing this work, a future blog will describe them in more detail and present some performance characterization and use case relevant for a **DellEMC Ready Solution for HPC PixStor Storage**.

Advanced Analytics

Among PixStor capabilities, monitoring the file system via advanced analytics can be essential to greatly simplify administration, helping to proactively or reactively find problems or potential issues. Next, we will briefly review some of these capabilities.

Figure 7 shows useful information based on the file system capacity. The left side shows the file system total space used, and the top ten users based on file system capacity used. The right side provides a historical view with capacity used across many years, then the top ten file types used and top ten filesets, both based on capacity used and in pareto chart diagram. With this information, it is easy to find users getting more than their fair share of the file system, identify trends of capacity usage to decide future growth for capacity, ascertain what files are using most of the space or what projects are taking most of the capacity.

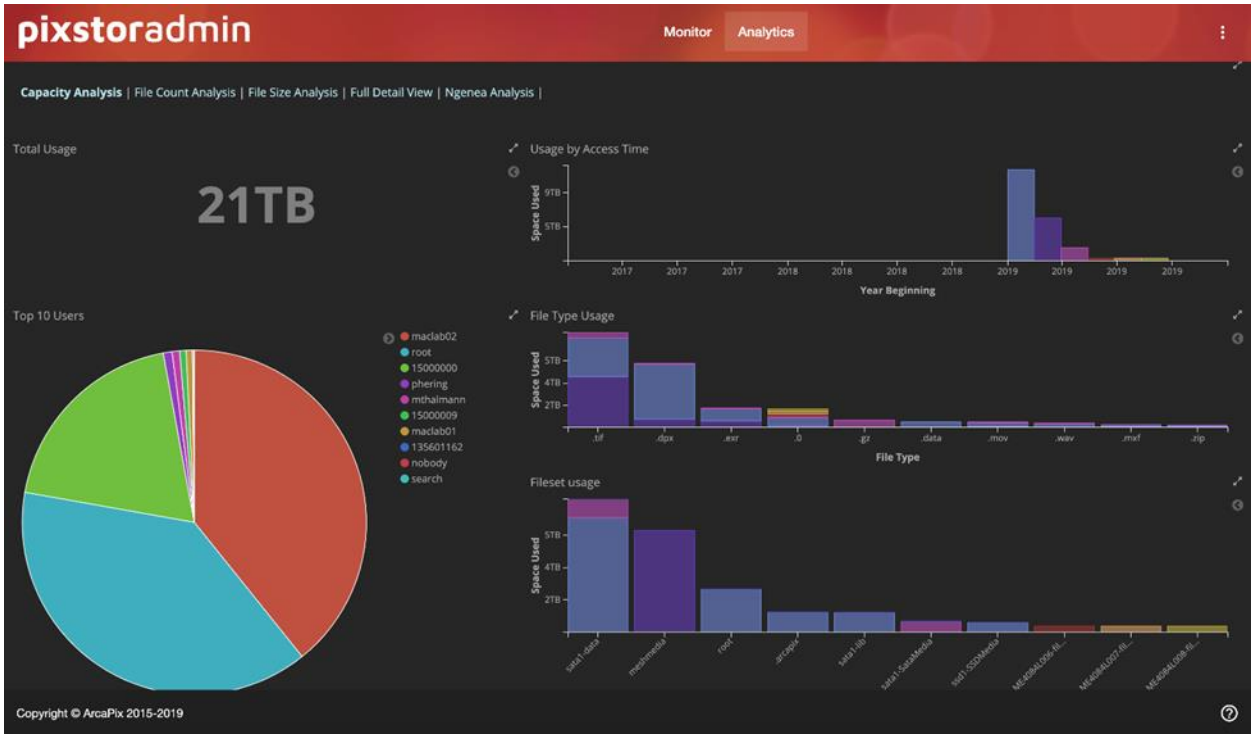


Figure 7 PixStor Analytics - Capacity view

Figure 8 provides a file count view with two very useful ways to find problems. The first half of the screen has the top ten users in a pie chart and top ten file types and top ten filesets (think projects) in pareto charts, all based on number of files. This information can be used to answer some important questions. For example, what users are monopolizing the file system by creating too many files, which type of file is creating a metadata nightmare or what projects are using most of the resources.

The bottom half has a histogram with number of files (frequency) for file sizes using 5 categories for different file sizes. This can be used to get an idea of the files sizes used across the file system, which coordinated with file types can be used to decide if compression will be beneficial.

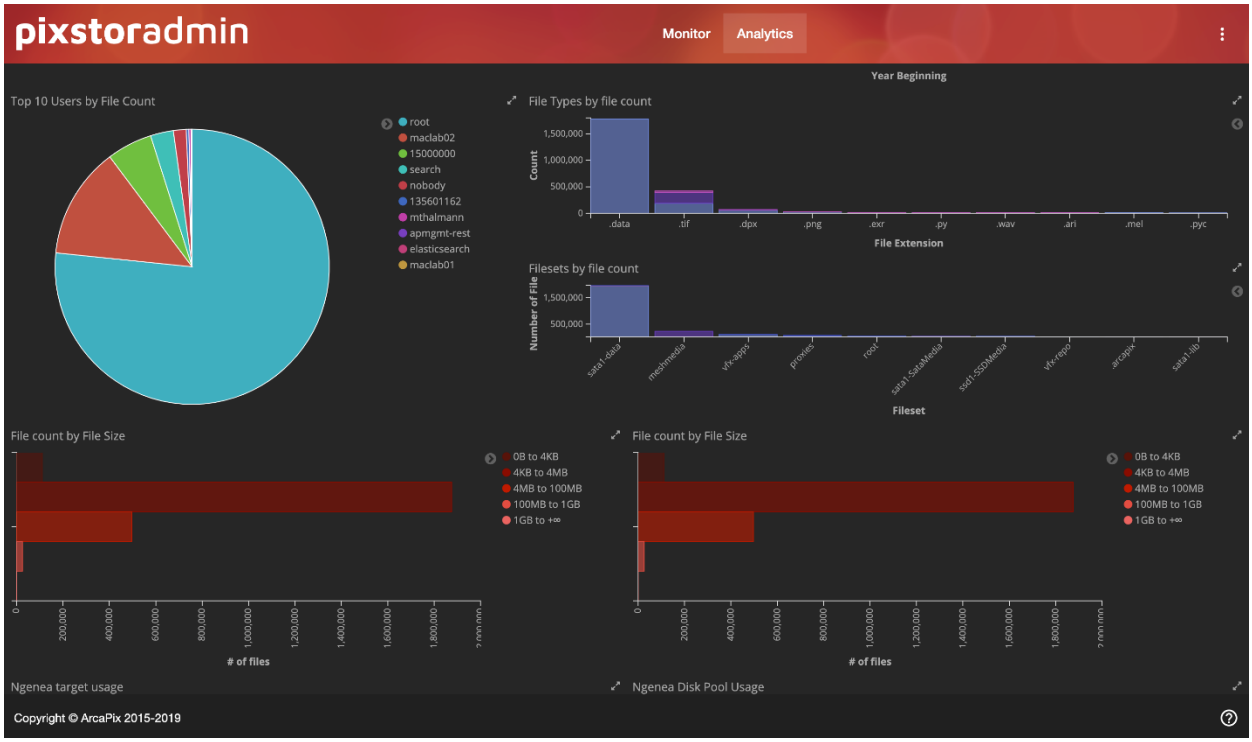


Figure 8 PixStor Analytics - File count view

Performance characterization

Benchmarks selected and test beds

To characterize the different components of this Ready Solution, we used the hardware specified in the last column of Table 1, including the optional High Demand Metadata Module. In order to assess the solution performance, the following benchmarks were selected:

- IOzone N to N sequential
- IOR N to 1 sequential
- IOzone random
- MDtest

For all benchmarks listed above, the test bed had the clients as described in the **Table 2** below, except for testing the Gateway nodes. Since the number of compute nodes available for testing was only 16, when a higher number of threads was required, those threads were equally distributed on the compute nodes (i.e. 32 threads = 2 threads per node, 64 threads = 4 threads per node, 128 threads = 8 threads per node, 256 threads =16 threads per node, 512 threads = 32 threads per node, 1024 threads = 64 threads per node). The intention was to simulate a higher number of concurrent clients with the limited number of compute nodes. Since the benchmarks support a high number of threads, a maximum value up to 1024 was used (specified for each test), while avoiding excessive context switching and other related side effects from affecting performance results.

The software versions listed were used in the initial testing, but as newer versions of the software became available, the servers and clients were updated. Each performance results section that use different versions from those listed in **Table 1** or **Table 2**, has the correct version of the software components used.

Table 2 Infiniband Client test bed

Number of Client nodes	16
Client node	C6320
Processors per client node	2 x Intel(R) Xeon(R) Gold E5-2697v4 18 Cores @ 2.30GHz
Memory per client node	12 x 16GiB 2400 MT/s RDIMMs
BIOS	2.8.0
Operating System	CentOS 7.6
OS Kernel	3.10.0-957.10.1
PixStor Software	4.8.3 (Released version 5.1.0.0)
Spectrum Scale (GPFS)	5.0.3
OFED Version	Mellanox OFED 4.3-3.0.2 (Released version Mellanox OFED 4.6-1.0.1.0)

Since the Gateways require a different network to properly test a realistic scenario where NFS and SMB protocols are used to connect to the PixStor solution, an Ethernet cluster was assembled from 13G servers in our lab. Since a limited number of machines was gathered, it was decided to use 100 GbE links to maximize the bandwidth of the clients, while still using a different network. **Table 3** below describes the 100 Gb Ethernet test bed for the Gateway nodes.

Table 3 Ethernet Client test bed (for Gateway nodes)

Number of Client nodes		16
Client node		Different 13G models with different CPUs and DIMMs
Cores per client node		10-22, Total = 492
Memory per client node		8x128 GiB & 8x256GB, Total = 3TiB For testing, all nodes were counted as 256GiB (4 TiB).
OS		CentOS 8.1
OS Kernel		4.18.0-147.el8.x86_64
PixStor Software		5.1.3.1
Spectrum Scale (GPFS)		5.0.4-3
OFED Version		Mellanox OFED 5.0-2.1.8.0
100 GbE Connectivity	**Adapter**	Mellanox ConnectX-4 InfiniBand VPI EDR/100 GbE
	Switch	DellEMC Z9100-ON

PixStor Solution with High Demand Meta-Data module (no Capacity Expansion)

This initial benchmarking was using the large configuration (two R740 servers connected to four ME4084s) with the optional HDMD module (two R740) but using a single ME4024 array instead of the two arrays a large configuration would normally have. The software versions for were those before the release versions ,as listed in Table 1 and Table 2.

Sequential IOzone Performance N clients to N files

Sequential N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 1024 threads.

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger that two times that size. It is important to notice that for GPFS that tunable sets the maximum amount of memory used for caching data, regardless the amount of RAM installed and free. Also, important to notice is that while in previous DellEMC HPC solutions the block size for large sequential transfers is 1 MiB, GPFS was formatted with 8 MiB blocks and therefore that value is used on the benchmark for optimal performance. That may look too large and apparently waste too much space, but GPFS uses subblock allocation to prevent that situation. In the current configuration, each block was subdivided in 256 subblocks of 32 KiB each.

The following commands were used to execute the benchmark for writes and reads, where Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
./iozone -i0 -c -e -w -r 8M -s 128G -t $Threads -+n -+m ./threadlist

./iozone -i1 -c -e -w -r 8M -s 128G -t $Threads -+n -+m ./threadlist
```

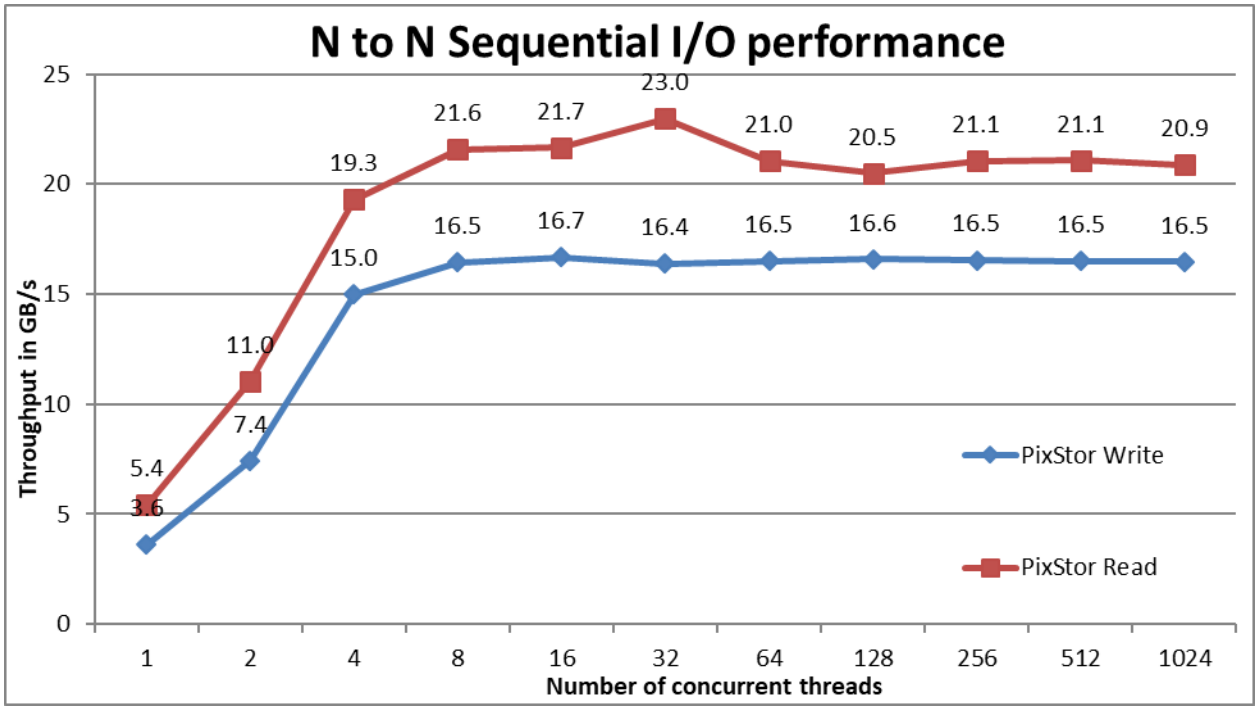


Figure 9 N to N Sequential Performance

From the results we can observe that performance rises very fast with the number of clients used and then reaches a plateau that is stable until the maximum number of threads that IOzone allow is reached, and therefore large file sequential performance is stable even for 1024 concurrent clients. Notice that the maximum read performance was 23 GB/s at 32 threads and very likely the bottleneck was the InfiniBand EDR interface, with ME4 arrays still had some extra performance available. Similarly notice that the maximum write performance of 16.7 was reached a bit early at 16 threads and it is apparently low compared to the ME4 arrays specs.

Here it is important to remember that GPFS preferred mode of operation is scattered, and the solution was formatted to use it. In this mode, blocks are allocated from the very beginning in a pseudo-random fashion, spreading data across the whole surface of each HDD. While the obvious disadvantage is a smaller initial maximum performance, that performance is maintained fairly constant regardless of how much space is used on the file system. That in contrast to other parallel file systems that initially use the outer tracks that can hold more data (sectors) per disk revolution, and therefore have the highest possible performance the HDDs can provide, but as the system uses more space, inner tracks with less data per revolution are used, with the consequent reduction of performance.

Sequential IOR Performance N clients to 1 file

Sequential N clients to a single shared file performance was measured with IOR version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from single thread up to 1024 threads.

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger that two times that size. This benchmark tests used 8 MiB blocks for optimal performance. The previous performance test section has a more complete explanation for those matters.

The following commands were used to execute the benchmark for writes and reads, where Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -w -s 1 -t 8m -b 128G
```

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -r -s 1 -t 8m -b 128G
```

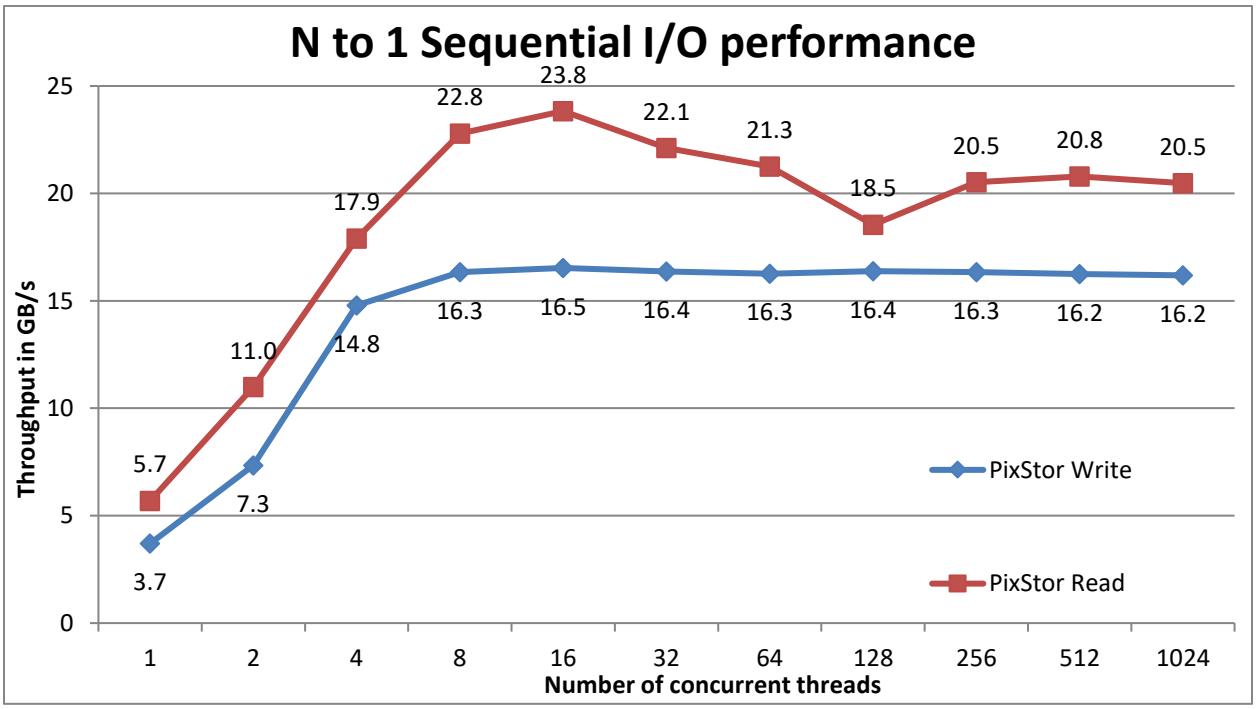


Figure 10 N to 1 Sequential Performance

From the results we can observe that performance rises again very fast with the number of clients used and then reaches a plateau that is semi-stable for reads and very stable for writes all the way to the maximum number of threads used on this test. Therefore, large single shared file sequential performance is stable even for 1024 concurrent clients. Notice that the maximum read performance was 23.7 GB/s at 16 threads and very likely the bottleneck was the InfiniBand EDR interface, with ME4 arrays still had some extra performance available. Furthermore, read performance decreased from that value until reaching the plateau at around 20.5 GB/s, with a momentary decrease to 18.5 GB/s at 128 threads. Similarly, notice that the maximum write performance of 16.5 was reached at 16 threads and it is apparently low compared to the ME4 arrays specs.

Random small blocks IOzone Performance N clients to N files

Random N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 1024 threads. This benchmark tests used 4 KiB blocks for emulating small blocks traffic.

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files two times that size. The first performance test section has a more complete explanation about why this is effective on GPFS.

The following command was used to execute the benchmark in random IO mode for both writes and reads, where Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
./iozone -i0 -c -e -w -r 8M -s 32G -t $Threads -+n -+m ./threadlist

./iozone -i2 -c -O -w -r 4K -s 32G -t $Threads -+n -+m ./threadlist
```

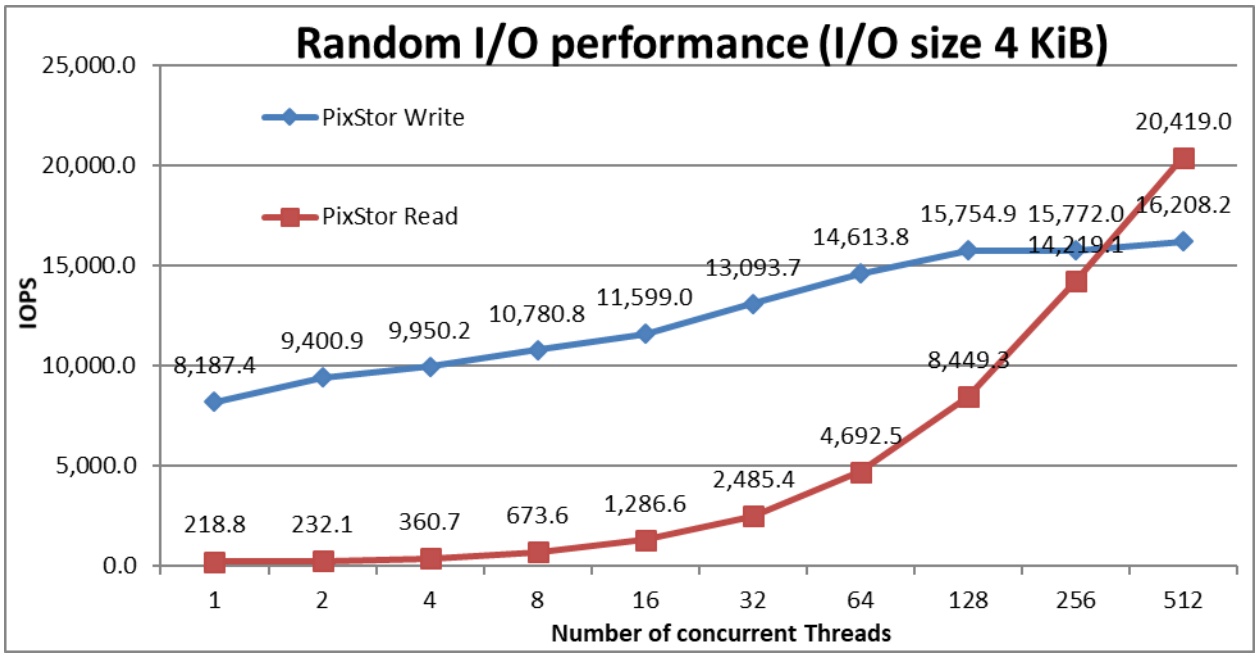


Figure 11 N to N Random Performance

From the results we can observe that write performance starts at a high value of almost 8.2K IOPS and rises steadily up to 128 threads where it reaches a plateau and remains close to the maximum value of 16.2K IOPs. Read performance on the other hand starts very small at over 200 IOPS and increases performance almost linearly with the number of clients used (keep in mind that number of threads is doubled for each data point) and reaches the maximum performance of 20.4K IOPS at 512 threads without signs of reaching the maximum. However, using more threads on the current 16 compute nodes with two CPUs each and where each CPU has 18 cores, have the limitation that there are not enough cores to run the maximum number of IOzone threads (1024) without incurring in context switching (16 x 2 x 18 = 576 cores), which limits performance considerably. A future test with more compute nodes could check what random read performance can be achieved with 1024 threads with IOzone, or IOR could be used to investigate the behavior with more than 1024 threads.

Metadata performance with MDtest using empty files

Metadata performance was measured with MDtest version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from single thread up to 512 threads. The benchmark was used for files only (no directories metadata), getting the number of creates, stats, reads and removes the solution can handle.

To properly evaluate the solution in comparison to other DellEMC HPC storage solutions, the optional High Demand Metadata Module was used, but with a single ME4024 array, even that the large configuration and tested in this work was designated to have two ME4024s.

This High Demand Metadata Module can support up to four ME4024 arrays, and it is suggested to increase the number of ME4024 arrays to 4, before adding another metadata module. Additional ME4024 arrays are expected to increase the Metadata performance linearly with each additional array, except maybe for Stat operations (and Reads for empty files), since the numbers are very high, at some point the CPUs will become a bottleneck and performance will not continue to increase linearly.

The following command was used to execute the benchmark, where Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes. Similar to the Random IO benchmark, the maximum number of threads was limited to 512, since there are not enough cores for 1024 threads and context switching would affect the results, reporting a number lower than the real performance of the solution.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -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, it was decided to keep fixed the total number of files to 2 MiB files (2^21 = 2097152), the number of files per directory fixed at 1024, and the number of directories varied as the number of threads changed as shown in **Table 4**.

Table 4 MDtest distribution of files on directories

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

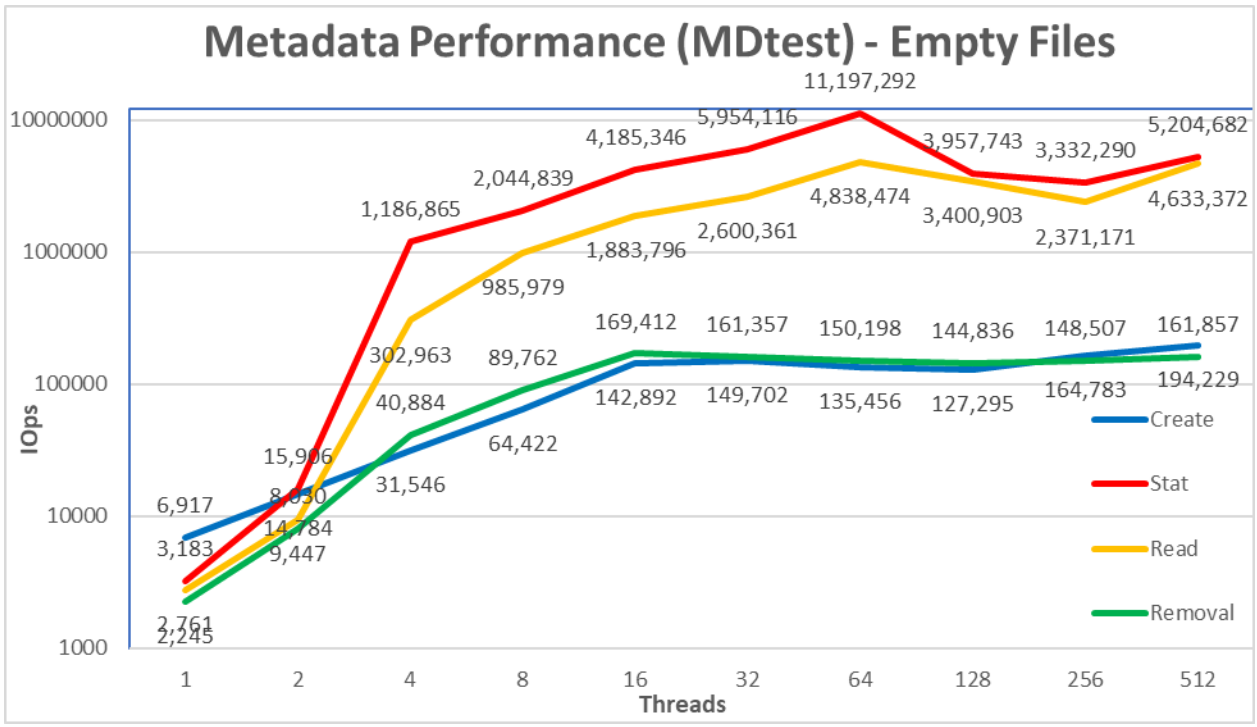


Figure 12 Metadata Performance - Empty Files

First, notice that the scale chosen was logarithmic with base 10, to allow comparing operations that have differences several orders of magnitude; otherwise some of the operations would look like a flat line close to 0 on a normal graph. A log graph with base 2 could be more appropriate, since the number of threads are increased in powers of 2, but the graph would look pretty similar, and people tend to handle and remember better numbers based on powers of 10.

The system gets very good results with Stat and Read operations reaching their peak value at 64 threads with 11.2M op/s and 4.8M op/s respectively. Removal operations attained the maximum of 169.4K op/s at 16 threads and Create operations achieving their peak at 512 threads with 194.2K op/s. Stat and Read operations have more variability, but once they reach their peak value, performance does not drop below 3M op/s for Stats and 2M op/s for Reads. Create and Removal are more stable once their reach a plateau and remain above 140K op/s for Removal and 120K op/s for Create.

Metadata performance with MDtest using 4 KiB files

This test is almost exactly identical to the previous one, except that instead of empty files, small files of 4KiB were used.

The following command was used to execute the benchmark, where Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -b $Directories
-z 1 -L -I 1024 -y -u -t -F -w 4K -e 4K
```

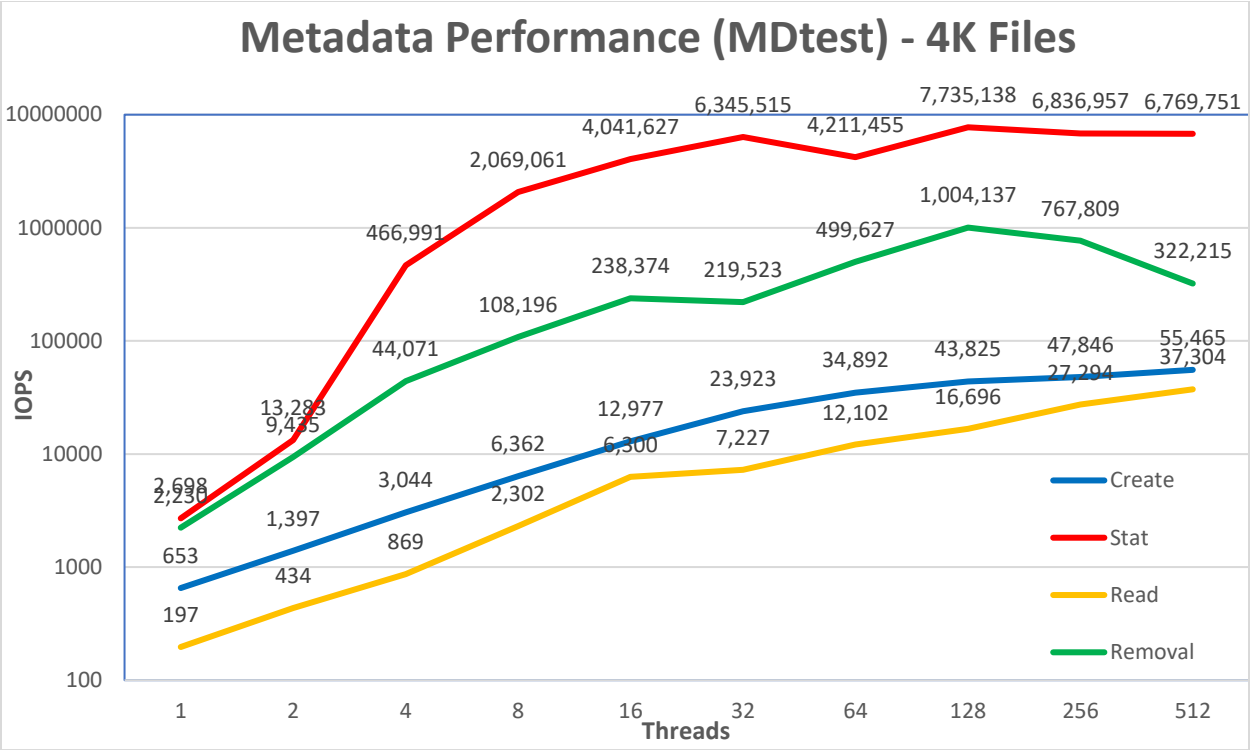


Figure 13 Metadata Performance - Small files (4K)

The system gets very good results for Stat and Removal operations reaching their peak value at 128 threads with 7.7M op/s and 1M op/s respectively. Removal operations attained the maximum of 37.3K op/s and Create operations achieving their peak with 55.5K op/s, both at 512 threads. Stat and Removal operations have more variability, but once they reach their peak value, performance does not drop below 4M op/s for Stats and 200K op/s for Removal. Create and Read have less variability and keep increasing as the number of threads grows.

Since these numbers are for a metadata module with a single ME4024, performance will increase for each additional ME4024 arrays, however we cannot just assume a linear increase for each operation. Unless the whole file fits inside the inode for such file, data targets on the ME4084s will be used to store the 4K files, limiting the performance to some degree. Since the inode size is 4KiB and it still needs to store metadata, only files around 3 KiB will fit inside and any file bigger than that will use data targets.

Metadata Performance using MDtest with 3K files

This test is almost exactly identical to the previous ones, except that small files of 3KiB were used. The main difference is that these files fit completely inside the inode. Therefore, the storage nodes and their ME4084s are not used, improving the overall speed by using only SSD media for storage and less network accesses.

The following command was used to execute the benchmark, where Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -b $Directories
-z 1 -L -I 1024 -y -u -t -F -w 3K -e 3K
```

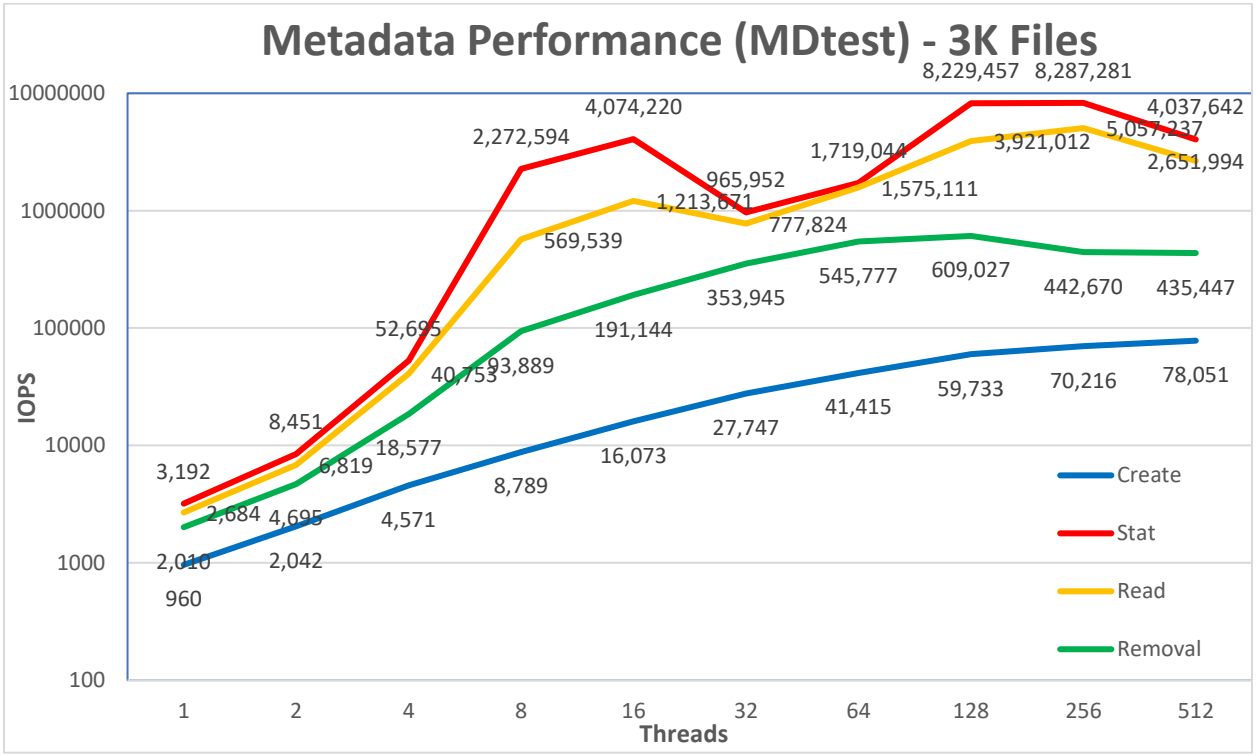


Figure 14 Metadata Performance - Small files (3K)

The system gets very good results for Stat and Read operations reaching their peak value at 256 threads with 8.29M op/s and 5.06M op/s respectively. Removal operations attained the maximum of 609K op/s at 128 threads and Create operations achieving their peak with 78K op/s at 512 threads. Stat and Read operations have more variability than Create and Removal. Removal has a small drop in performance for the two higher threads points suggesting the sustained performance after 128 threads will be slightly over 400K op/s. Creates kept increasing up to 512 threads, but looks like is reaching a plateau so the maximum performance may still be under 100K op/s.

Since small files like these are stored completely on the SSD based metadata module, applications requiring superior small files performance, can use one or more optional high demand metadata modules to increase small files performance. However, files that fit in the inode are tiny by current standards. Also, since the metadata targets use RAID1s with SSDs relatively small (max size is 19.2TB), capacity will be limited when compared to the storage nodes. Therefore, care must be exercised to avoid filling up Metadata targets, which can cause unnecessary failures and other problems.

Summary

The current solution was able to deliver fairly good performance, which is expected to be stable regardless of the used space (since the system was formatted in scattered mode), as can be seen in **Table 5** MDtest distribution of files on directories. Furthermore, the solution scales in capacity and performance linearly as

more storage nodes modules are added, and a similar performance increase can be expected from the optional high demand metadata module.

Table 5 Peak & Sustained Performance

Benchmark	Peak Performance		Sustained Performance	
	Write	Read	Write	Read
Large Sequential N clients to N files	16.7 GB/s	23 GB/s	16.5 GB/s	20.5 GB/s
Large Sequential N clients to single shared file	16.5 GB/s	23.8 GB/s	16.2 GB/s	20.5 GB/s
Random Small blocks N clients to N files	15.8KIOps	20.4KIOps	15.7KIOps	20.4KIOps
Metadata Create empty files	169.4K IOps		127.2K IOps	
Metadata Stat empty files	11.2M IOps		3.3M IOps	
Metadata Read empty files	4.8M IOps		2.4M IOps	
Metadata Remove empty files	194.2K IOps		144.8K IOps	
Metadata Create 4KiB files	55.4K IOps		55.4K IOps	
Metadata Stat 4KiB files	6.4M IOps		4M IOps	
Metadata Read 4KiB files	37.3K IOps		37.3K IOps	
Metadata Remove 4KiB files	1M IOps		219.5K IOps	

PixStor Solution with Capacity Expansion and High Demand Meta-Data

This benchmarking was on the initial configuration plus four ME484s, that is two R740 servers connected to four ME4084s and four ME484s (one behind each ME4084), with the optional HDMD module (two R740) but using a single ME4024 array instead of the two arrays a large configuration would normally have.

The software versions use during the characterization are listed in **Table 6**.

Table 6 Software Components versions during characterization

Solution Component	Version at Characterization
Operating System	CentOS 7.7
Kernel version	3.10.0-1062.9.1.el7.x86_64
PixStor Software	5.1.1.4
Spectrum Scale (GPFS)	5.0.4-2
OFED Version	Mellanox OFED-4.7-3.2.9

Sequential IOzone Performance N clients to N files

Sequential N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 1024 threads, and the results of the capacity expanded solution (4x ME4084s + 4x ME484s) are contrasted to the large size solution (4x ME4084s).

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger that two times that size. It is important to notice that for GPFS that tunable sets the maximum amount of memory used for caching data, regardless the amount of RAM installed and free. Also, important to notice is that while in previous DellEMC HPC solutions the block size for large sequential transfers is 1 MiB, GPFS was formatted with 8 MiB blocks and therefore that value is used on the benchmark for optimal performance. That may look too large and apparently waste too much space, but GPFS uses subblock allocation to prevent that situation. In the current configuration, each block was subdivided in 256 subblocks of 32 KiB each.

The following commands were used to execute the benchmark for writes and reads, where Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
./iozone -i0 -c -e -w -r 8M -s 128G -t $Threads -+n -+m ./threadlist

./iozone -i1 -c -e -w -r 8M -s 128G -t $Threads -+n -+m ./threadlist
```

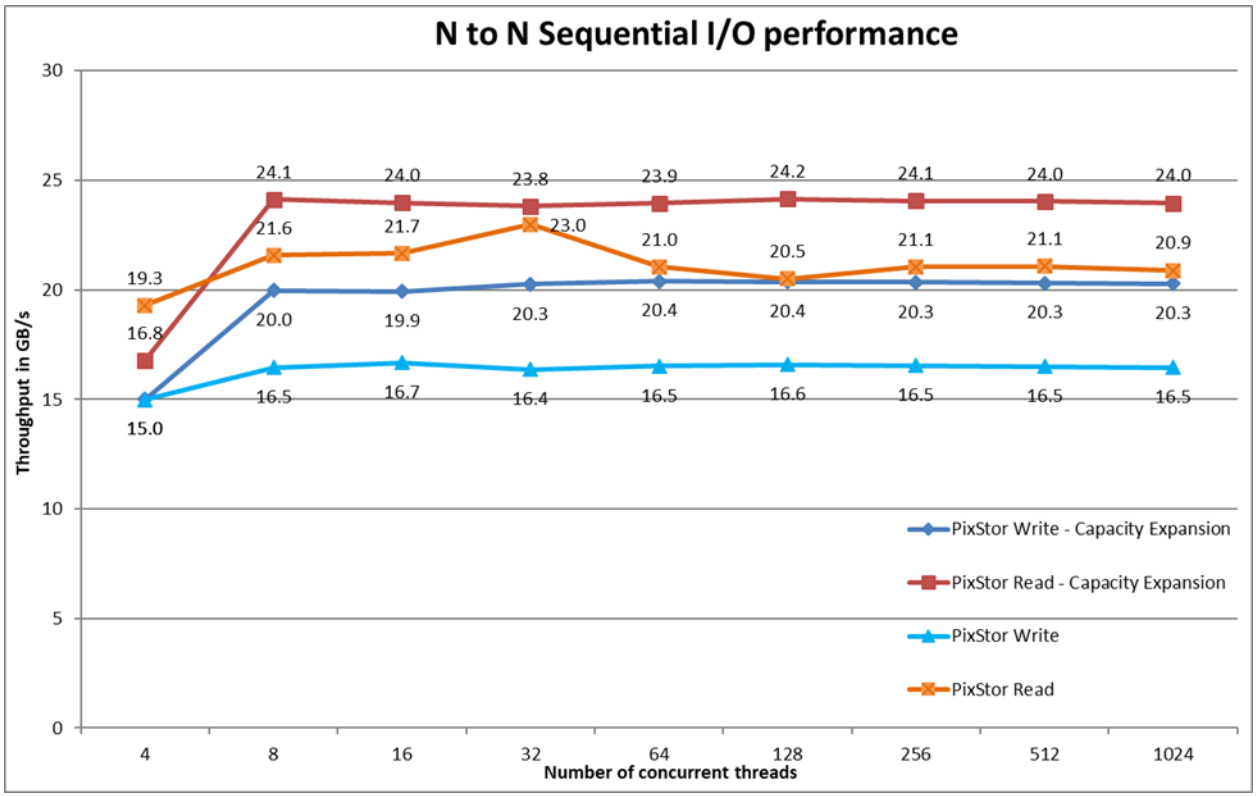


Figure 15 N to N Sequential Performance

From the results we can observe that performance rises very fast with the number of clients used and then reaches a plateau that is stable until the maximum number of threads that IOzone allow is reached, and therefore large file sequential performance is stable even for 1024 concurrent clients. Notice that both read and write performance benefited from the doubling the number of drives. The maximum read performance was limited by the bandwidth of the two IB EDR links used on the storage nodes starting at 8 threads, and ME4 arrays may have some extra performance available. Similarly notice that the maximum write performance increased from a maximum of 16.7 to 20.4 GB/s at 64 and 128 threads and it is closer to the ME4 arrays maximum specs (22 GB/s).

Here it is important to remember that GPFS preferred mode of operation is scattered, and the solution was formatted to use such mode. In this mode, blocks are allocated from the very beginning of operation in a pseudo-random fashion, spreading data across the whole surface of each HDD. While the obvious disadvantage is a smaller initial maximum performance, that performance is maintained fairly constant regardless of how much space is used on the file system. That in contrast to other parallel file systems that initially use the outer tracks that can hold more data (sectors) per disk revolution, and therefore have the highest possible performance the HDDs can provide, but as the system uses more space, inner tracks with less data per revolution are used, with the consequent reduction of performance.

Sequential IOR Performance N clients to 1 file

Sequential N clients to a single shared file performance was measured with IOR version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from a one thread up to 512 threads (since there were not enough cores for 1024 threads), and results are contrasted to the solution without the capacity expansion.

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger that two times that size. This benchmark tests used 8 MiB blocks for optimal performance. The previous performance test section has a more complete explanation for those matters.

The following commands were used to execute the benchmark for writes and reads, where Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -w -s 1 -t 8m -b 128G
```

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -r -s 1 -t 8m -b 128G
```

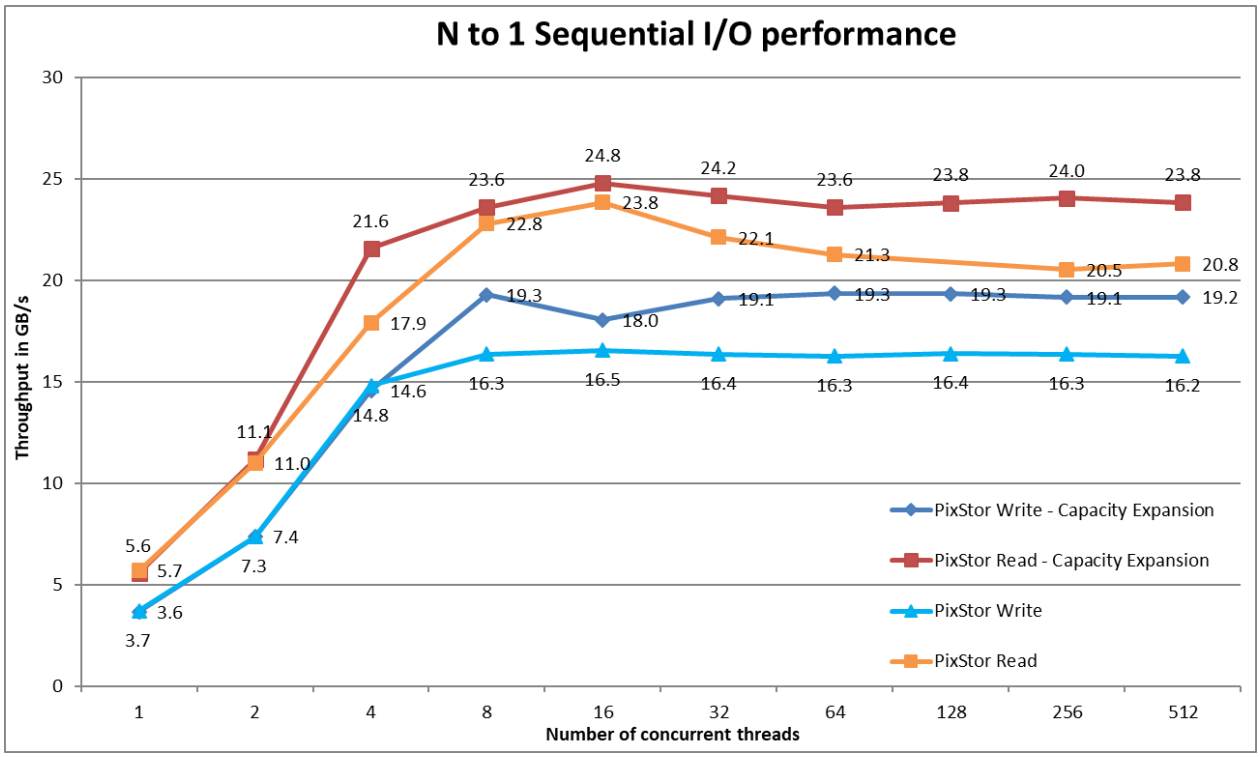


Figure 16 N to 1 Sequential Performance

From the results we can observe again that the extra drives benefit read and write performance. Performance rises again very fast with the number of clients used and then reaches a plateau that is fairly stable for reads and writes all the way to the maximum number of threads used on this test. Notice that the maximum read performance was 24.8 GB/s at 16 threads and the bottleneck was the InfiniBand EDR interface, with ME4 arrays still had some extra performance available. From that point, read performance decreased from that value until reaching the plateau at around 23.8 GB/s. Similarly, notice that the maximum write performance of 19.3 was reached at 8 threads and reach a plateau.

Random small blocks IOzone Performance N clients to N files

Random N clients to N files performance was measured with FIO version 3.7 instead of the traditional Iozone. The intention, as listed in the previous blog, was of take advantage of a larger Queue Depth to investigate the maximum possible performance that ME4084 arrays can deliver (previous tests for different ME4 solutions showed the ME4084 arrays need more IO pressure that Iozone can deliver to reach their random IO limits).

Tests executed varied from single thread up to 512 threads since there was not enough client-cores for 1024 threads. Each thread was using a different file and the threads were assigned round robin on the client nodes. This benchmark tests used 4 KiB blocks for emulating small blocks traffic and using a queue depth of 16. Results from the large size solution and the capacity expansion are compared.

Caching effects were again minimized by setting the GPFS page pool tunable to 16GiB and using files two times that size. The first performance test section has a more complete explanation about why this is effective on GPFS.

```
./iozone -i0 -c -e -w -r 8M -s 32G -t $Threads -+n -+m ./threadlist

./iozone -i2 -c -O -w -r 4K -s 32G -t $Threads -+n -+m ./threadlist
```

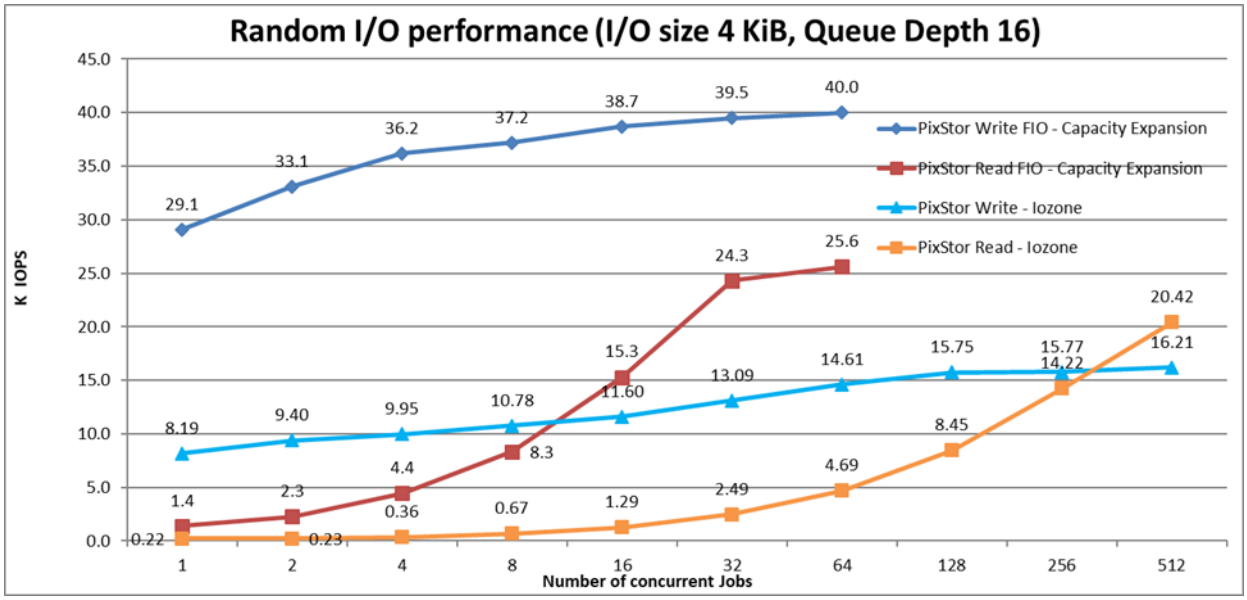


Figure 17 N to N Random Performance

From the results we can observe that write performance starts at a high value of 29.1K IOps and rises steadily up to 64 threads where it seems to reach a plateau at around 40K IOps. Read performance on the other hand starts at 1.4K IOps and increases performance almost linearly with the number of clients used (keep in mind that number of threads is doubled for each data point) and reaches the maximum performance of 25.6K IOPS at 64 threads where seems to be close to reaching a plateau. Using more threads will require more than the 16 compute nodes to avoid resources starvation and a lower apparent performance, where the arrays could in fact maintain the performance.

Metadata performance with MDtest using empty files

Metadata performance was measured with MDtest version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from single thread up to 512 threads. The benchmark was used for files only (no directories metadata), getting the number of creates, stats, reads and removes the solution can handle, and results were contrasted with the Large size solution.

To properly evaluate the solution in comparison to other DellEMC HPC storage solutions and the previous blog results, the optional High Demand Metadata Module was used, but with a single ME4024 array, even that the large configuration and tested in this work was designated to have two ME4024s.

This High Demand Metadata Module can support up to four ME4024 arrays, and it is suggested to increase the number of ME4024 arrays to 4, before adding another metadata module. Additional ME4024 arrays are expected to increase the Metadata performance linearly with each additional array, except maybe for Stat operations (and Reads for empty files), since the numbers are very high, at some point the CPUs will become a bottleneck and performance will not continue to increase linearly.

The following command was used to execute the benchmark, where Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes. Similar to the Random IO benchmark, the maximum number of threads was limited to 512, since there are not enough cores for 1024 threads and context switching would affect the results, reporting a number lower than the real performance of the solution.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -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, it was decided to keep fixed the total number of files to 2 MiB files (2^21 = 2097152), the number of files per directory fixed at 1024, and the number of directories varied as the number of threads changed as shown in **Table 4** in the first benchmark characterization section.

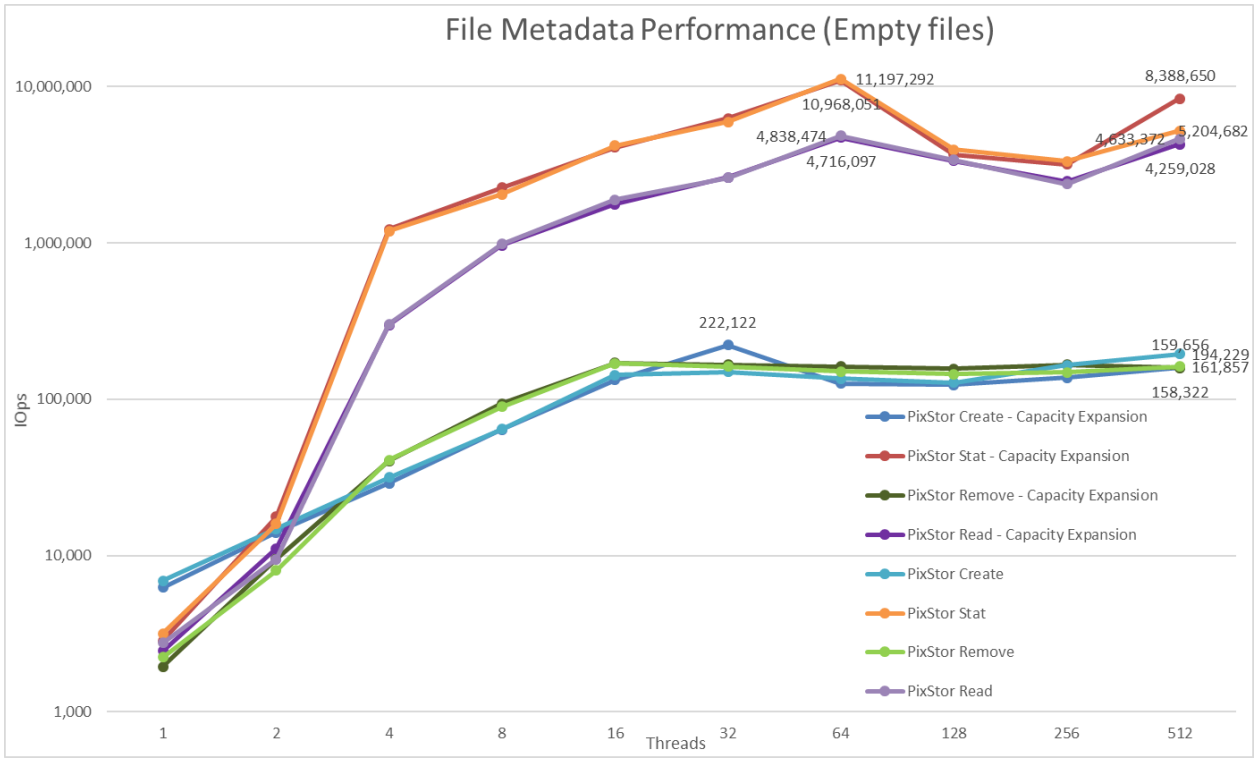


Figure 18 Metadata Performance - Empty Files

First, notice that the scale chosen was logarithmic with base 10, to allow comparing operations that have differences several orders of magnitude; otherwise some of the operations would look like a flat line close to 0 on a normal graph. A log graph with base 2 could be more appropriate, since the number of threads are increased in powers of 2, but the graph would look very similar and people tend to handle and remember better numbers based on powers of 10.

The system gets very good results with Stat and Read operations reaching their peak value at 64 threads with almost 11M op/s and 4.7M op/s respectively. Removal operations attained the maximum of 170.6K op/s at 16 threads and Create operations achieving their peak at 32 threads with 222.1K op/s. Stat and Read operations have more variability, but once they reach their peak value, performance does not drop below 3M op/s for Stats and 2M op/s for Reads. Create and Removal are more stable once their reach a plateau and remain above 140K op/s for Removal and 120K op/s for Create. Notice the extra drives do not affect much most of the metadata operations on empty files as expected.

Metadata performance with MDtest using 4 KiB files

This test is almost exactly identical to the previous one, except that instead of empty files, small files of 4KiB were used.

The following command was used to execute the benchmark, where Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -b $Directories
-z 1 -L -I 1024 -y -u -t -F -w 4K -e 4K
```

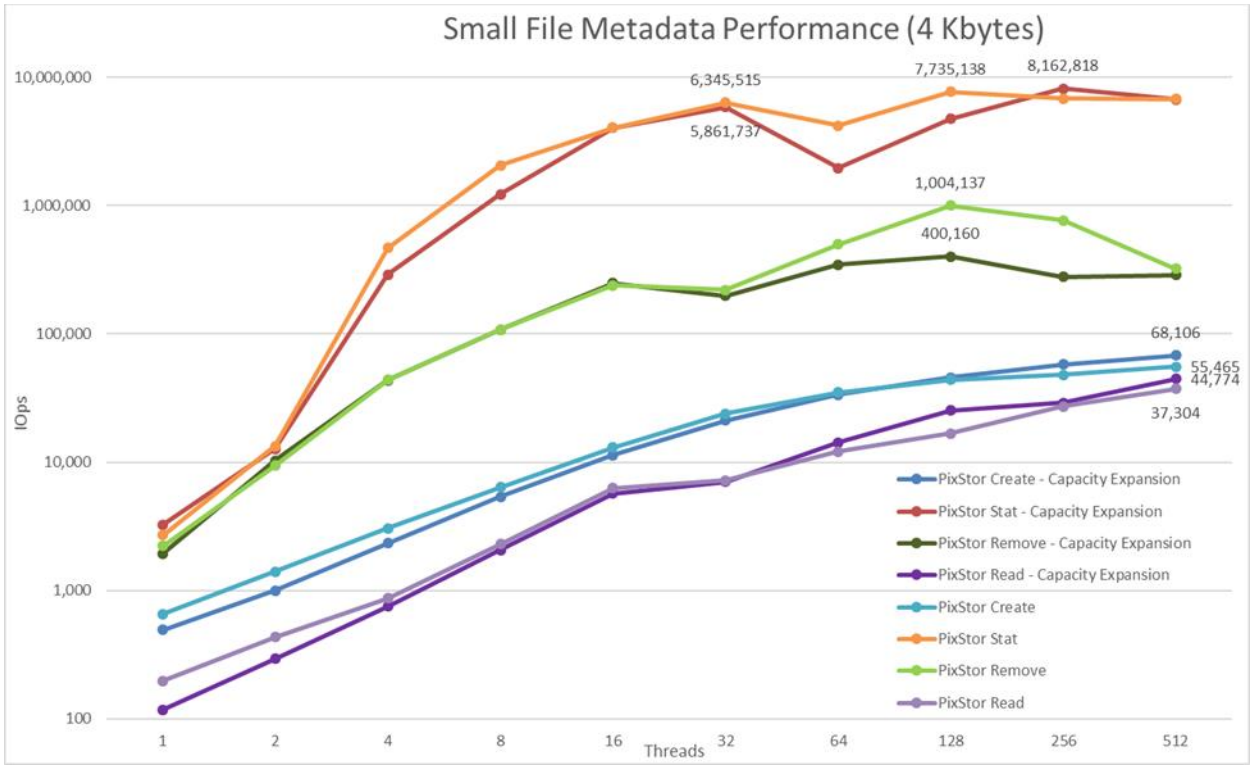


Figure 19 Metadata Performance - Small files (4K)

The system gets very good results for Stat and Removal operations reaching their peak value at 256 threads with 8.2M op/s and 400K op/s respectively. Read operations attained the maximum of 44.8K op/s and Create operations achieving their peak with 68.1K op/s, both at 512 threads. Stat and Removal operations have more variability, but once they reach their peak value, performance does not drop below 3M op/s for Stats and 280K op/s for Removal. Create and Read have less variability and keep increasing as the number of threads grows. As can be observed, the extra drives of the capacity expansions only provide marginal changes in metadata performance.

Since these numbers are for a metadata module with a single ME4024, performance will increase for each additional ME4024 arrays, however we cannot just assume a linear increase for each operation. Unless the whole file fits inside the inode for such file, data targets on the ME4084s will be used to store the 4K files,

limiting the performance to some degree. Since the inode size is 4KiB and it still needs to store metadata, only files around 3 KiB will fit inside and any file bigger than that will use data targets.

Summary

The current solution was able to deliver fairly good performance, which is expected to be stable regardless of the used space (since the system was formatted in scattered mode), as can be seen in

Benchmark	Peak Performance		Sustained Performance	
	Write	Read	Write	Read
Large Sequential N clients to N files	16.7 GB/s	23 GB/s	16.5 GB/s	20.5 GB/s
Large Sequential N clients to single shared file	16.5 GB/s	23.8 GB/s	16.2 GB/s	20.5 GB/s
Random Small blocks N clients to N files	15.8KIOps	20.4KIOps	15.7KIOps	20.4KIOps
Metadata Create empty files	169.4K IOps		127.2K IOps	
Metadata Stat empty files	11.2M IOps		3.3M IOps	
Metadata Read empty files	4.8M IOps		2.4M IOps	
Metadata Remove empty files	194.2K IOps		144.8K IOps	
Metadata Create 4KiB files	55.4K IOps		55.4K IOps	
Metadata Stat 4KiB files	6.4M IOps		4M IOps	
Metadata Read 4KiB files	37.3K IOps		37.3K IOps	
Metadata Remove 4KiB files	1M IOps		219.5K IOps	

Table 7. Furthermore, the solution scales in capacity and performance linearly as more storage nodes modules are added, and a similar performance increase can be expected from the optional high demand metadata module.

Table 7 Peak & Sustained Performance

Benchmark	Peak Performance		Sustained Performance	
	Write	Read	Write	Read
Large Sequential N clients to N files	20.4 GB/s	24.2 GB/s	20.3 GB/s	24 GB/s
Large Sequential N clients to single shared file	19.3 GB/s	24.8 GB/s	19.3 GB/s	23.8 GB/s
Random Small blocks N clients to N files	40K IOps	25.6K IOps	40K IOps	25.6KIOps
Metadata Create empty files	169.4K IOps		123.5K IOps	
Metadata Stat empty files	11M IOps		3.2M IOps	
Metadata Read empty files	4.7M IOps		2.4M IOps	
Metadata Remove empty files	170.6K IOps		156.5K IOps	
Metadata Create 4KiB files	68.1K IOps		68.1K IOps	
Metadata Stat 4KiB files	8.2M IOps		3M IOps	
Metadata Read 4KiB files	44.8K IOps		44.8K IOps	
Metadata Remove 4KiB files	400K IOps		280K IOps	

PixStor Solution – NVMe Tier

This benchmarking was performed on four R640 NVMe nodes, each with eight Intel P4610 NVMe SSDs arranged as eight NVMe over Fabric RAID10 using NVMesh, as previously described in this document. Such RAID 10 were used as Block devices to create NSDs for data only, so the optional HDMD module (two R740) but using a single ME4024 array was used to store all the metadata.

While this works pretty well as the results can show, The NSDs can store data and metadata, creating a self-contained tier 0 where metadata could also benefit from the NVMe devices speed. Even more, if such a need arises, the NVMe NSDs could be used for metadata only, to create an extreme demand metadata module.

The software versions use during the NVMe characterization are listed in **Table 8**.

Table 8 Software Components versions during characterization

Solution Component	Version at Characterization
Operating System	CentOS 7.7
Kernel version	3.10.0-1062.12.1.el7.x86_64
PixStor Software	5.1.3.1
Spectrum Scale (GPFS)	5.0.4-3
NVMesh	2.0.1
OFED Version	Mellanox OFED-5.0-2.1.8.0

Sequential IOzone Performance N clients to N files

Sequential N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 1024 threads in increments of powers of two.

Caching effects were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger than two times that size. It is important to notice that for GPFS that tunable sets the maximum amount of memory used for caching data, regardless the amount of RAM installed and free. Also, important to notice is that while in previous DellEMC HPC solutions the block size for large sequential transfers is 1 MiB, GPFS was formatted with 8 MiB blocks and therefore that value is used on the benchmark for optimal performance. That may look too large and apparently waste too much space, but GPFS uses subblock allocation to prevent that situation. In the current configuration, each block was subdivided in 256 subblocks of 32 KiB each.

The following commands were used to execute the benchmark for writes and reads, where $Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

To avoid any possible data caching effects from the clients, the total data size of the files was twice the total amount of RAM in the clients used. That is, since each client has 128 GiB of RAM, for threads counts equal or above 16 threads the file size was 4096 GiB divided by the number of threads (the variable $Size below was used to manage that value). For those cases with less than 16 threads (which implies each thread was running on a different client), the file size was fixed at twice the amount of memory per client, or 256 GiB.

```
./iozone -i0 -c -e -w -r 8M -s ${Size}G -t $Threads -+n -+m ./threadlist

./iozone -i1 -c -e -w -r 8M -s ${Size}G -t $Threads -+n -+m ./threadlist
```

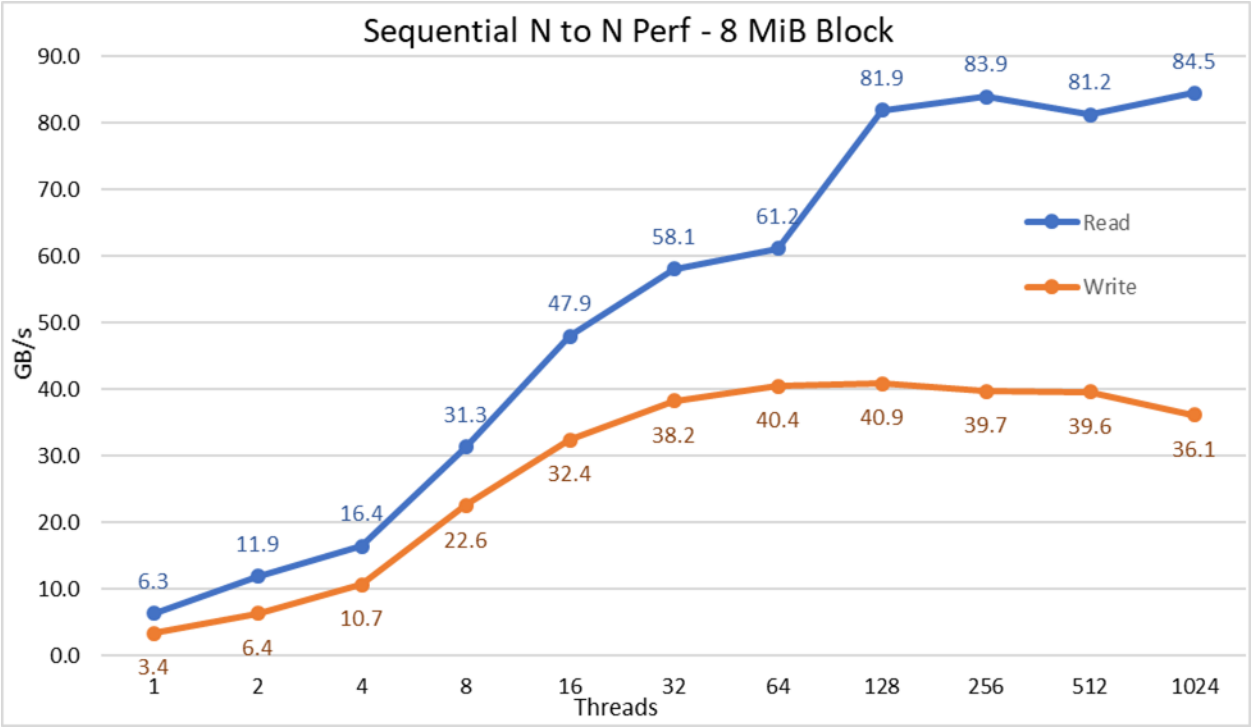


Figure 20 N to N Sequential Performance

From the results we can observe that write performance rises with the number of threads used and then reaches a plateau at around 64 threads for writes and 128 threads for reads. Then read performance also rises fast with the number of threads, and then stays stable until the maximum number of threads that IOzone allow is reached, and therefore large file sequential performance is stable even for 1024 concurrent clients. Write performance drops about 10% at 1024 threads. However, since the client cluster has less than that number of cores, it is uncertain if the drop in performance is due to swapping and similar overhead not observed in spinning media(since NVMe latency is very low compared to spinning media), or if the RAID 10 data synchronization is becoming a bottleneck. More clients are needed to clarify that point. An anomaly on the reads was observed at 64 threads, where performance did not scale at the rate that was observed for previous data points, and then on the next data point moves to a value very close to the sustained performance. More testing is needed to find the reason for such anomaly but is out of the scope of this blog.

The maximum read performance for reads was below the theoretical performance of the NVMe devices (~102 GB/s), or the performance of EDR links, even assuming that one link was mostly used for NVMe over fabric traffic (4x EDR BW ~96 GB/s).

However, this is not a surprise since the hardware configuration is not balanced with respect to the NVMe devices and IB HCAs under each CPU socket. One CX6 adapter is under CPU1, while the CPU2 has all the NVMe devices and the second CX6 adapters. Any storage traffic using the first HCA must use the UPIs to access the NVMe devices. In addition, any core in CPU1 used must access devices or memory assigned to CPU2, so data locality suffers, and UPI links are used. That can explain the reduction for the maximum performance, compared to the max performance of the NVMe devices or the line speed for CX6 HCAs. The alternative to fix that limitation is having a balanced hardware configuration which implies reducing density to half by using an R740 with four x16 slots and use two x16 PCIe expanders to equally distribute NVMe devices on two CPUs and having one CX6 HCA under each CPU.

Sequential IOR Performance N clients to 1 file

Sequential N clients to a single shared file performance was measured with IOR version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from a one thread up to 512 threads since there were not enough cores for 1024 or more threads. This benchmark tests used 8 MiB blocks for optimal performance. The previous performance test section has a more complete explanation for why that matters.

Data caching effects were minimized by setting the GPFS page pool tunable to 16GiB and the total file size was twice the total amount of RAM in the clients used. That is, since each client has 128 GiB of RAM, for threads counts equal or above 16 threads the file size was 4096 GiB, and an equal amount of that total was divided by the number of threads (the variable $Size below was used to manage that value). For those cases with less than 16 threads (which implies each thread was running on a different client), the file size was twice the amount of memory per client used times the number of threads, or in other words, each thread was asked to use 256 GiB.

The following commands were used to execute the benchmark for writes and reads, where $Threads was the variable with the number of threads used (1 to 1024 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -w -s 1 -t 8m -b ${Size}G
```

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --mca
btl_openib_allow_ib 1 --mca pml ^ucx --oversubscribe --prefix
/mmfs1/perftest/ompi /mmfs1/perftest/lanl_ior/bin/ior -a POSIX -v -i 1 -d 3 -e -
k -o /mmfs1/perftest/tst.file -r -s 1 -t 8m -b ${Size}G
```

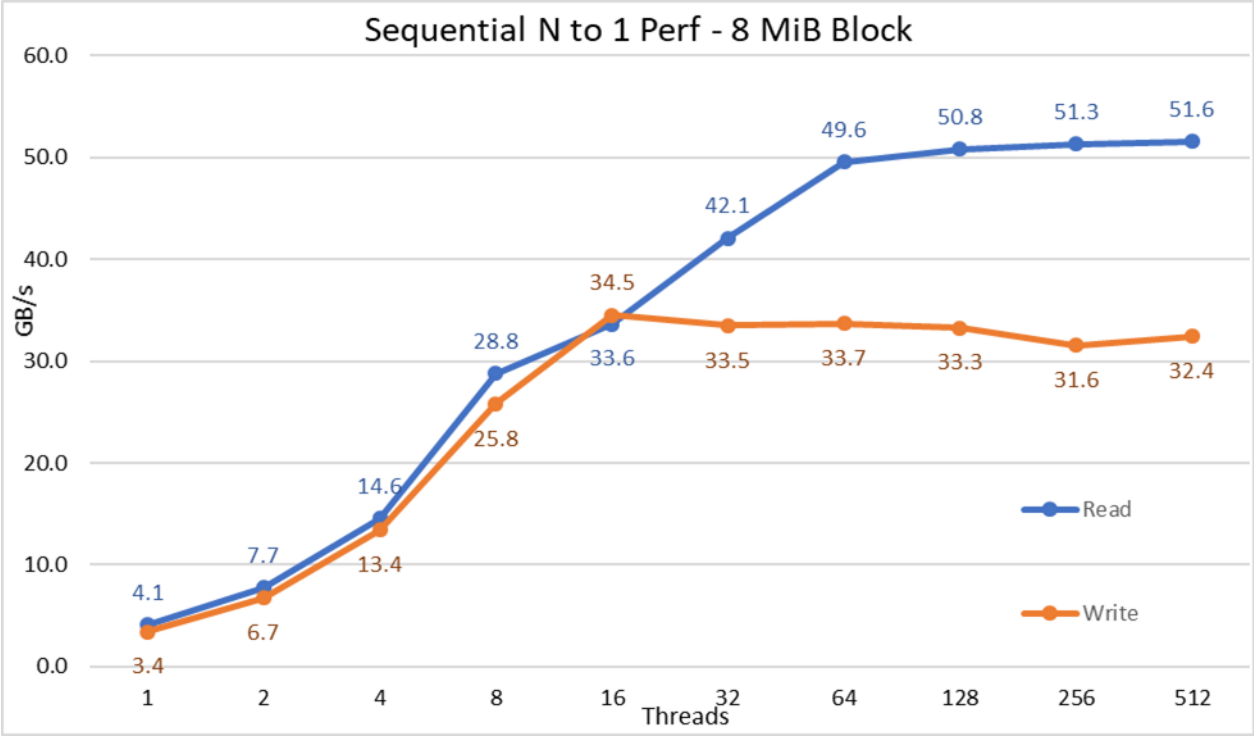


Figure 21 N to 1 Sequential Performance

From the results we can observe read and write performance are high regardless of the implicit need for locking mechanisms since all threads access the same file. Performance rises again very fast with the number of threads used and then reaches a plateau that is relatively stable for reads and writes all the way to the maximum number of threads used on this test. Notice that the maximum read performance was 51.6 GB/s at 512 threads, but the plateau in performance is reach at about 64 threads. Similarly, notice that the maximum write performance of 34.5 GB/s was achieved at 16 threads and reached a plateau that can be observed until the maximum number of threads used.

Random small blocks IOzone Performance N clients to N files

Random N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 1024 threads in increments of powers of two.

Tests executed varied from single thread up to 512 threads since there was not enough client-cores for 1024 threads. Each thread was using a different file and the threads were assigned round robin on the client nodes. This benchmark tests used 4 KiB blocks for emulating small blocks traffic and using a queue depth of 16. Results from the large size solution and the capacity expansion are compared.

Caching effects were again minimized by setting the GPFS page pool tunable to 16GiB and to avoid any possible data caching effects from the clients, the total data size of the files was twice the total amount of RAM in the clients used. That is, since each client has 128 GiB of RAM, for threads counts equal or above 16 threads the file size was 4096 GiB divided by the number of threads (the variable $Size below was used to manage that value). For those cases with less than 16 threads (which implies each thread was running on a different client), the file size was fixed at twice the amount of memory per client, or 256 GiB.

```
./iozone -i0 -I -c -e -w -r 8M -s ${Size}G -t $Threads -+n -+m ./nvme_threadlist

./iozone -i2 -I -c -O -w -r 4k -s ${Size}G -t $Threads -+n -+m ./nvme_threadlist
```

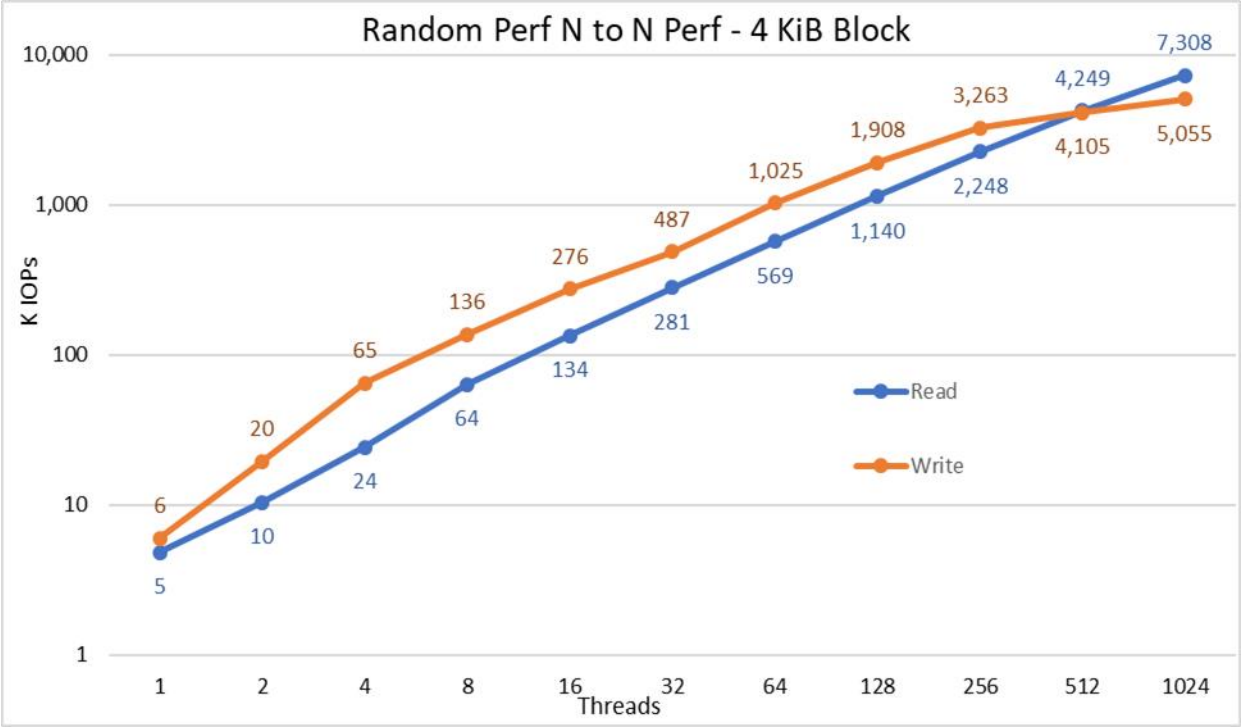


Figure 22 N to N Random Performance

From the results we can observe that write performance starts at a high value of 6K IOps and rises steadily up to 1024 threads where it seems reach a plateau with over 5M IOPS if more threads could be used. Read performance on the other hand starts at 5K IOPSs and increases performance steadily with the number of threads used (keep in mind that number of threads is doubled for each data point) and reaches the maximum performance of 7.3M IOPS at 1024 threads without signs of reaching a plateau. Using more threads will require more than the 16 compute nodes to avoid resources starvation and excessive swapping that can lower apparent performance, where the NVMe nodes could in fact maintain the performance.

Metadata performance with MDtest using 4 KiB files

Metadata performance was measured with MDtest version 3.3.0, assisted by OpenMPI v4.0.1 to run the benchmark over the 16 compute nodes. Tests executed varied from single thread up to 512 threads. The benchmark was used for files only (no directories metadata), getting the number of creates, stats, reads and removes the solution can handle, and results were contrasted with the Large size solution.

The optional High Demand Metadata Module was used, but with a single ME4024 array, even that the large configuration and tested in this work was designated to have two ME4024s. The reason for using that metadata module is that currently these NVMe nodes are used as Storage targets for data only. However, the nodes could be used to store data and metadata, or even as a flash alternative for the high demand metadata module, if extreme metadata demands call for it. Those configurations were not tested as part of this work.

Since the same High Demand Metadata module has been used for previous benchmarking of the DellEMC Ready Solution for HPC PixStor Storage solution, metadata results will be very similar compared to previous blog results. For that reason, the study with empty files was not done, and instead 4 KiB files were used.

Since 4KiB files cannot fit into an inode along with the metadata information, NVMe nodes will be used to store data for each file. Therefore, MDtest can give a rough idea of small files performance for reads and the rest of the metadata operations.

The following command was used to execute the benchmark, where $Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and my_hosts.$Threads is the corresponding file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes. Similar to the Random IO benchmark, the maximum number of threads was limited to 512, since there are not enough cores for 1024 threads and context switching would affect the results, reporting a number lower than the real performance of the solution.

```
mpirun --allow-run-as-root -np $Threads --hostfile my_hosts.$Threads --prefix
/mmfs1/perftest/ompi --mca btl_openib_allow_ib 1
/mmfs1/perftest/lanl_ior/bin/mdtest -v -d /mmfs1/perftest/ -i 1 -b $Directories
-z 1 -L -I 1024 -y -u -t -F -w 4K -e 4K
```

Since performance results can be affected by the total number of IOPS, the number of files per directory and the number of threads, it was decided to keep fixed the total number of files to 2 MiB files (2^21 = 2097152), the number of files per directory fixed at 1024, and the number of directories varied as the number of threads changed as shown in **Table 4**.

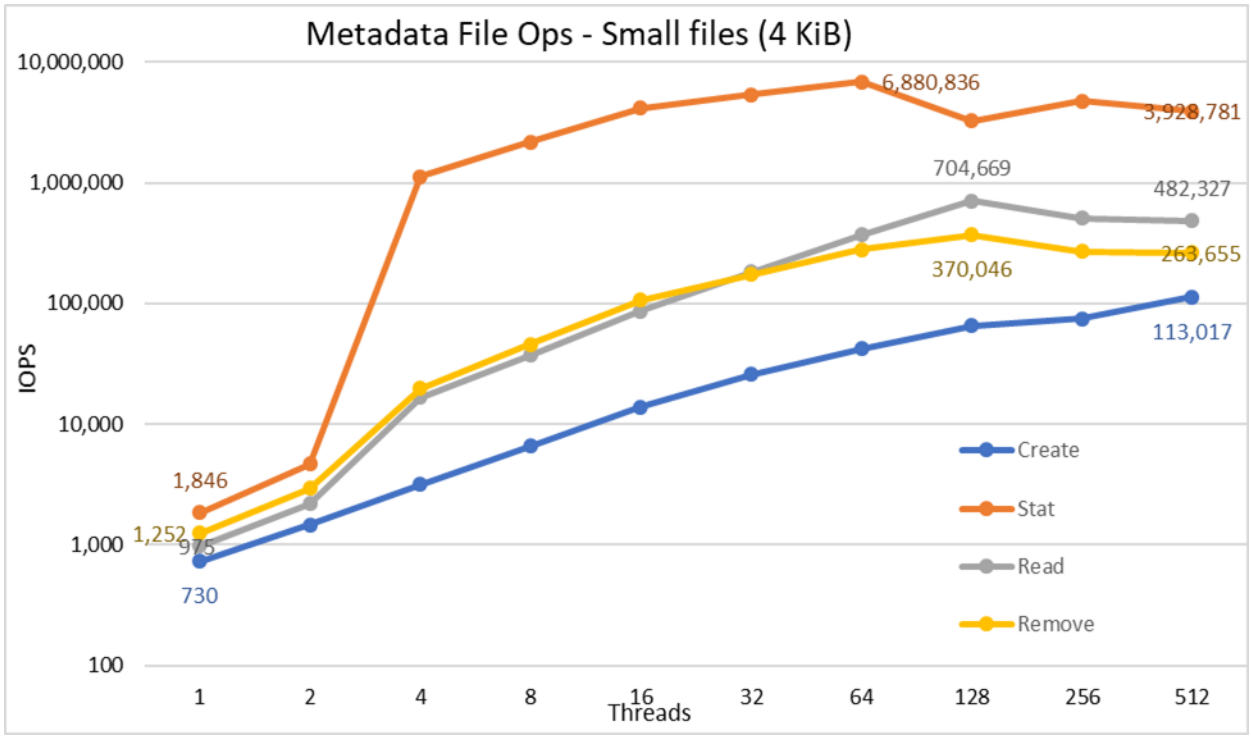


Figure 23 Metadata Performance – 4 KiB Files

First, notice that the scale chosen was logarithmic with base 10, to allow comparing operations that have differences with several orders of magnitude; otherwise some of the operations would look like a flat line close to 0 on a linear scale. A log graph with base 2 could be more appropriate, since the number of threads are increased in powers of 2, but the graph would look very similar and people tend to handle and remember better numbers based on powers of 10.

The system gets very good results as previously reported with Stat operations reaching the peak value at 64 threads with almost 6.9M op/s and then is reduced for higher thread counts reaching a plateau. Create operations reach the maximum of 113K op/s at 512 threads, so is expected to continue increasing if more client nodes (and cores) are used. Reads and Removes operations attained their maximum at 128 threads, achieving their peak at almost 705K op/s for Reads and 370K op/s for removes, and then they reach plateaus. Stat operations have more variability, but once they reach their peak value, performance does not drop below 3.2M op/s for Stats. Create and Removal are more stable once their reach a plateau and remain above 265K op/s for Removal and 113K op/s for Create. Finally, reads reach a plateau with performance above 265K op/s.

Summary

The current solution was able to deliver fairly good performance, which is expected to be stable regardless of the used space (since the system was formatted in scattered mode), as can be seen in **Table 9**. Furthermore, the solution scales in capacity and performance linearly as more storage nodes modules are added, and a similar performance increase can be expected from the optional high demand metadata module.

Table 9 Peak & Sustained Performance

Benchmark	Peak Performance		Sustained Performance	
	Write	Read	Write	Read
Large Sequential N clients to N files	40.9 GB/s	84.5 GB/s	40 GB/s	81 GB/s
Large Sequential N clients to single shared file	34.5 GB/s	51.6 GB/s	31.5 GB/s	50 GB/s
Random Small blocks N clients to N files	5.06M IOps	7.31M IOps	5M IOps	7.3M IOps
Metadata Create 4KiB files	113K IOps		113K IOps	
Metadata Stat 4KiB files	6.88M IOps		3.2M IOps	
Metadata Read 4KiB files	705K IOps		500K IOps	
Metadata Remove 4KiB files	370K IOps		265K IOps	

PixStor Solution – Gateway Nodes

This benchmarking was run with the file system running on the Large configuration plus four ME484s, that is two R740 servers connected to four ME4084s and four ME484s (one behind each ME4084), with the optional HDMD module (two R740) but using a single ME4024 array and two gateways using a native client to access the file system, and exporting that file system using the NFS or SMB protocols.

To characterize this component, the IOzone benchmark was used to characterize sequential performance for N clients to N files using large transfer blocks. Since the gateways can be used for NFS or SMB clients, both protocols were used for this work.

To allow proper testing with a limited number of clients, the cluster described in **Table 3** was created with heterogeneous 13G PowerEdge servers including from R630, R730 and R730XD. Clients have different CPU models type E5-26XX v4 with different number of cores per socket (10, 12, 18 or 22), adding a total number of cores to 492, and speeds ranging from 2.2 to 2.6 GHz (2.2, 2.3 & 2.6).
Similarly, half of the clients had memory 8GiB DIMMs and the other half had 16GiB DIMMs, always using all memory channels and with speeds mostly 2400 MT/s (except two clients @ 2133 MT/s). Therefore, each

server had either 128 GiB, or 256 GiB with a total of 3 TiB. However, to simplify testing and avoid caching, all clients were counted as having 256GiB.

Regarding network connectivity, all clients have a Mellanox CX4 VPI adapter configured for 100 Gb Ethernet and connected using a Dell EMC Z9100 switch, so that the limited number of clients could provide the highest possible load for the Gateways. The two gateways tested are also connected to the Z9100 switch using a single 100 GbE link each.

Since the number of compute nodes available for testing was only 16, when a higher number of threads was required, those threads were equally distributed on the compute nodes (i.e. 32 threads = 2 threads per node, 64 threads = 4 threads per node, 128 threads = 8 threads per node, 256 threads =16 threads per node, 512 threads = 32 threads per node, 1024 threads = 64 threads per node). The intention was to simulate a higher number of concurrent clients with the limited number of compute nodes available. Since some benchmarks support a high number of threads, a maximum value up to 1024 was used (specified for each test), while avoiding excessive context switching and other related side effects from affecting performance results.

Table 3 has the software versions used on the clients. The software versions used on the servers during the characterization are listed in **Table 10**.

Table 10 Software Components versions during characterization

Solution Component	Version at Characterization
Operating System	CentOS 7.7
Kernel version	3.10.0-1062.12.1.el7.x86_64
PixStor Software	5.1.3.1
Spectrum Scale (GPFS)	5.0.4-3
OFED Version	Mellanox OFED-5.0-2.1.8.0

NFS testing

For this testing, two PixStor Gateways were used exporting the PixStor file system via native OS NFS server. Clients were mounted the FS via NFSv3 using the IP of each gateway in a round robin fashion (odd numbered clients to first gateway and even numbered clients to the second gateway). This manual configuration was used to deterministically have half of the clients mounted from each Gateway. While NFSv4 is supported, NFSv3 is still widely used and it was selected as the protocol version to use.

Sequential IOzone Performance N clients to N files

Sequential N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 512 threads in increments of powers of two (since there are not enough cores on the clients for executing efficiently 1024 threads).

Caching effects on the servers were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger than two times that size. It is important to notice that for GPFS that tunable sets the maximum amount of memory used for caching data, regardless the amount of RAM installed and free.

To avoid any possible data caching effects from the clients, the total data size of the files was twice the total amount of RAM in the clients used. That is, since each client was considered as having 256 GiB of RAM, for threads counts equal or above 16 threads the file size was 8192 GiB divided by the number of threads (the variable $Size below was used to manage that value). For those cases with less than 16 threads (which

implies each thread was running on a different client), the file size was fixed at twice the amount of memory per client, or 512 GiB.

Even that for the PixStor native client the optimum block transfer size is 8 MiB, the block size for large sequential transfers was set to 1 MiB, since that is the maximum size used by NFS for reads and writes.

The following commands were used to execute the benchmark for writes and reads, where $Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
./iozone -i0 -c -e -w -r 1M -s ${Size}G -t $Threads -+n -+m ./threadlist

./iozone -i1 -c -e -w -r 1M -s ${Size}G -t $Threads -+n -+m ./threadlist
```

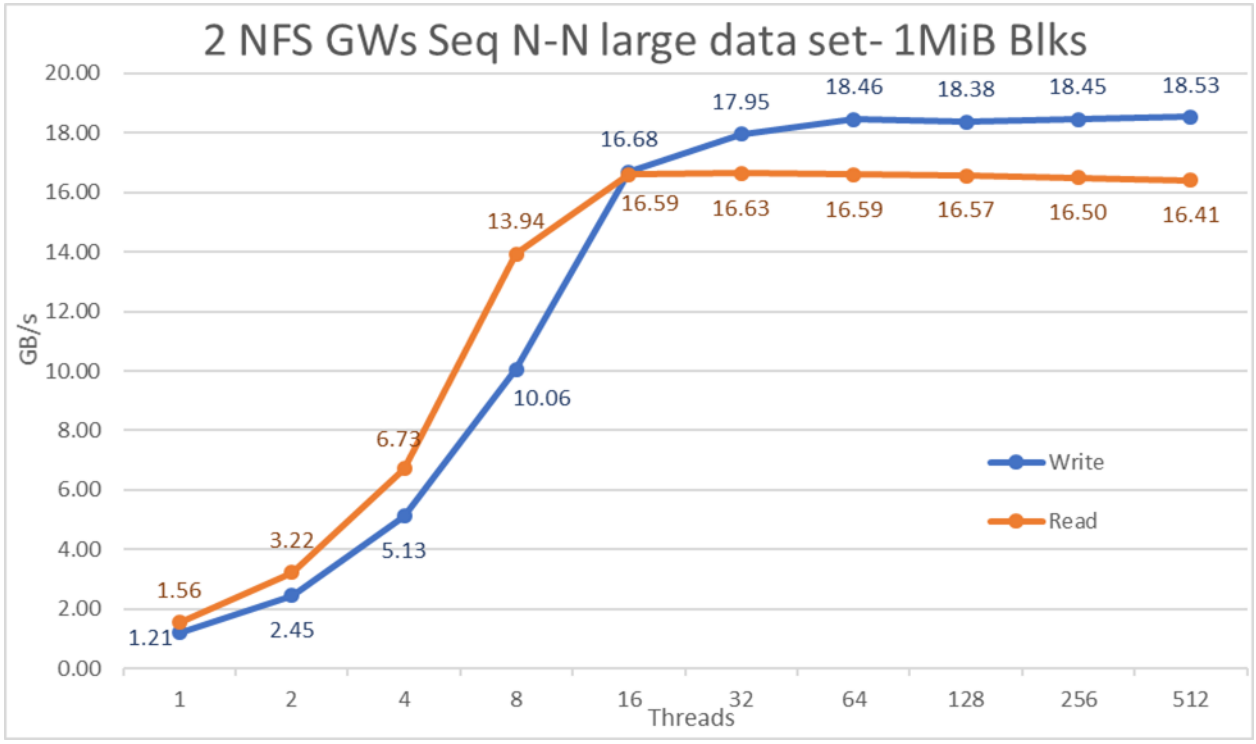


Figure 24 N to N Sequential Performance - NFS

From the results we can observe that read performance rises steadily with the number of threads, reaching the plateau of about 16.5 GB/s with 16 threads and reaching the maximum performance of 16.63 GB/s at 32 threads.

The write performance also rises steadily until 16 threads getting to 90% pf the sustain value, and then more slowly until reaching a plateau at 64 threads with sustained performance of about 18.3 GB/s. The maximum read performance of 18.53 GB/s was reached at 512 threads.

SMB testing

For this testing, the same two PixStor Gateways were used exporting the PixStor file system via native OS SMB server. Clients were mounted the FS via SMBv3 using the IP of each gateway in a round robin fashion

(odd numbered clients to first gateway and even numbered clients to the second gateway). This manual configuration was used to deterministically have half of the clients mounted from each Gateway.

Sequential IOzone Performance N clients to N files

Sequential N clients to N files performance was measured with IOzone version 3.487. Tests executed varied from single thread up to 512 threads in increments of powers of two (since there are not enough cores on the clients for executing efficiently 1024 threads).

Caching effects on the servers were minimized by setting the GPFS page pool tunable to 16GiB and using files bigger than two times that size. It is important to notice that for GPFS that tunable sets the maximum amount of memory used for caching data, regardless the amount of RAM installed and free.

To avoid any possible data caching effects from the clients, the total data size of the files was twice the total amount of RAM in the clients used. That is, since each client was considered as having 256 GiB of RAM, for threads counts equal or above 16 threads the file size was 8192 GiB divided by the number of threads (the variable $Size below was used to manage that value). For those cases with less than 16 threads (which implies each thread was running on a different client), the file size was fixed at twice the amount of memory per client, or 512 GiB.

Even that for the solution native client, the optimum block transfer size is 8 MiB, for consistency with benchmarks executed on NFS, the block size for large sequential transfers was set to 1 MiB, since Samba has smaller maximum transfer sizes.

The following commands were used to execute the benchmark for writes and reads, where $Threads was the variable with the number of threads used (1 to 512 incremented in powers of two), and threadlist was the file that allocated each thread on a different node, using round robin to spread them homogeneously across the 16 compute nodes.

```
./iozone -i0 -c -e -w -r 1M -s ${Size}G -t $Threads -+n -+m ./threadlist

./iozone -i1 -c -e -w -r 1M -s ${Size}G -t $Threads -+n -+m ./threadlist
```

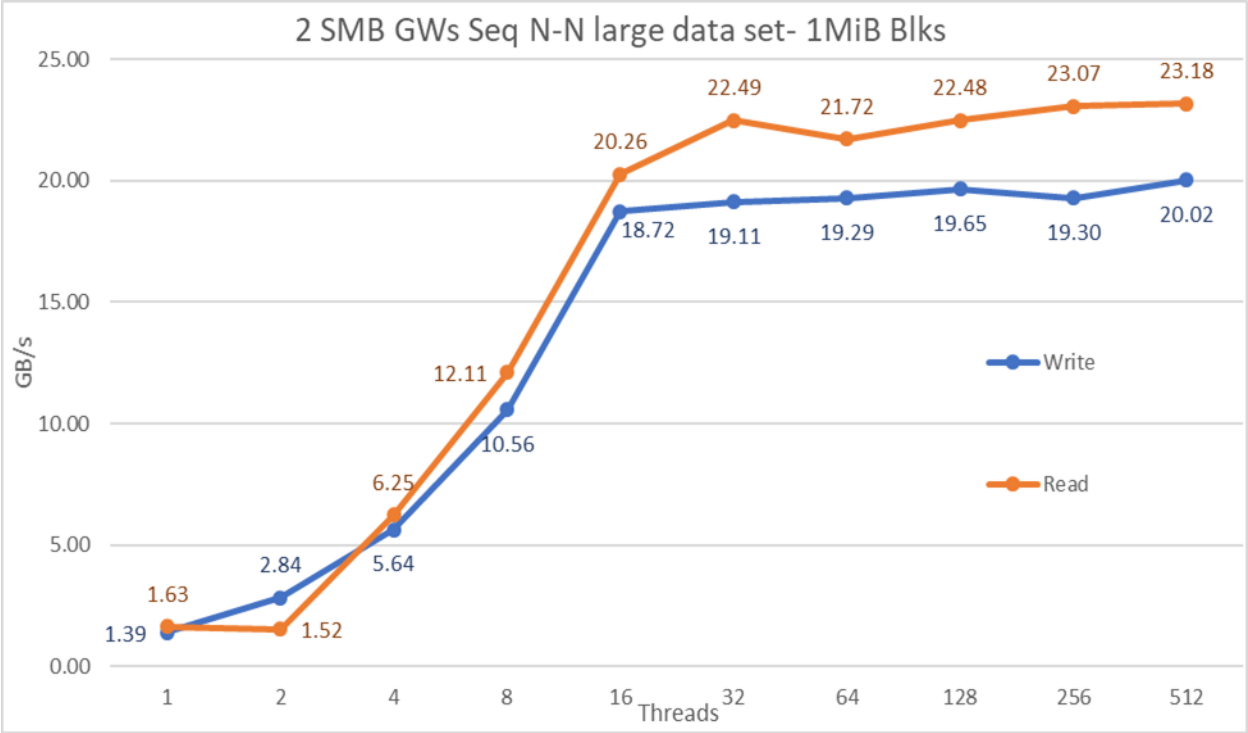


Figure 25 N to N Sequential Performance - SMB

From the results we can observe that write performance rises steadily with the number of threads, almost reaching the plateau of about 19 GB/s with 16 threads and attaining the maximum performance of 20.2 GB/s at 512 threads.

The read performance does not immediately rise as expected but stays almost flat at 2 threads. After that, it rises with the number of threads fast and steadily, almost reaching the plateau at 16 threads, attaining the maximum read performance of 23.18 GB/s at 512 threads and having a sustained performance of about 21.5 GB/s.

Summary

The Gateway nodes provide connectivity at reasonably high sequential performance when accessing different files (N to N).

The solution scales out in performance as more Gateway nodes are added. The N to N sequential performance from two Gateways nodes can be reviewed on **Table 11**, and it is expected to be stable. Those values can be used to estimate the performance for a different number of Gateway nodes, keeping in mind that **numbers reported were for two Gateways, each contributing half of the bandwidth reported.**

Table 11 Peak & Sustained Performance

Benchmark	Peak Performance		Sustained Performance	
	Write	Read	Write	Read
NFS: Large Sequential N clients to N files	18.53 GB/s	16.63 GB/s	18.3 GB/s	16.5 GB/s
SMB: Large Sequential N clients to N files	20.20 GB/s	23.18 GB/s	19 GB/s	21.5 GB/s

Conclusion and Future Work

The **DellEMC Ready Solution for HPC PixStor Storage** is a high-performance file system solution that is very efficient, easy to manage, fully supported, multi-tier, scalable in throughput and in capacity and with components to allow connecting it via standard protocols like NFS, SMB or Cloud. The solution is based on PowerEdge servers and PowerVault ME4 storage arrays, and PixStor software from ArcaStream, with components like Spectrum Scale (GPFS) and NVMesh.

Next project for the solution will be an update to support HDR100, along with some performance validation to verify that performance remains very similar to those values reported in this document.

Next, there are plans for the addition of AMD based nodes to the NVMe Tier, along with performance characterization of such nodes.

Finally, we want to try the Ngenea nodes in typical scenarios for HPC storage.

The vehicle to report results for the different pieces of future work is very likely technical Blogs.

References

Dell EMC Ready Solution for HPC PixStor Storage

Dell EMC Ready Solution for HPC PixStor Storage - Capacity Expansion

Dell EMC Ready Solution for HPC PixStor Storage - NVMe Tier

Dell EMC Ready Solution for HPC PixStor Storage – Gateway Nodes

Spectrum Scale (GPFS) Overview

NVMesh Datasheet

Clustered Trivial Data-Base (CTDB)

NFS-Ganesha

PowerVault ME4 support matrix

Dell EMC ME4 Series Storage System Administrator's Guide

IOzone Benchmark

IOR & MDtest Benchmarks

Open-MPI Software

Benchmark Reference

This section describes the commands that were used to benchmark the DellEMC HPC Storage solutions.

IOzone

The following commands are examples to run sequential and random IOzone tests, the results of which are recorded in Performance evaluation.

For sequential writes:

```
iozone -i 0 -c -e -w -r 1024K -s $Size -t $Thread -+n -+m /path/to/threadlist
```

For sequential reads:

```
iozone -i 1 -c -e -w -r 1024K -s $Size -t $Thread -+n -+m /path/to/threadlist
```

For IOPS random reads/writes:

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

Table 12 describes the IOzone command line options. The O_Direct command line option, -I, enables us to bypass the cache on the compute nodes where the IOzone threads are running.

Table 12 IOzone command line options

Option	Description
-i 0	Write test
-I 1	Read test
-I 2	Random IOPS test
-+n	No retest
-c	Includes close in the timing calculations
-e	Includes flush in the timing calculations
-r	Records size
-s	File size
-+m	Location of clients to run IOzone on when in clustered mode
-I	Use O_Direct
-w	Does not unlink (delete) temporary file
-O	Return results in OPS

IOR (N to 1)

The following command is an example to run the IOR N-1 performance tests:

```
mpirun --allow-run-as-root -np $Threads --hostfile $hostlist --map-by node -np
$threads ~/bin/ior -a POSIX -v -w -r -i 3 -t 8m -b $file_size -g -d 3 -e -E -k
-o $working_dir/ior.out -s 1
```

Table 13 describes the IOR command line options.

Table 13 IOR command line options

Option	Description
-g	intraTestBarriers – use barriers between open, write/read, and close
-d	interTestDelay – delay between reps in seconds
-e	fsync – perform fsync upon POSIX write close
-E	useExistingTestFile – do not remove test file before write access
-k	keepFile – don’t remove the test file(s) on program exit
-o	testFile – full name for test
-s	segmentCount – number of segments

MDtest

The following command is an example to run the metadata tests:

```
mpirun --allow-run-as-root -machinefile $hostlist --map-by node -np $threads
$mdtest -v -d $working_dir -i ${repetitions} -b $n_directories -z 1 -L -I
$n_files -y -u -t -F -w 4K -e 4K
```

Table 14 describes the MDtest command line options.

Table 14 MDtest command line options

Option	Description
-d	Directory in which the tests will run
-v	Verbosity (each instance of option increments by one)
-i	Number of iterations that the test will run
-b	Branching factor, how many directories to create, used in conjunction with -z
-z	Depth of hierarchical directory structure
-L	Files only at leaf level of tree
-I	Number of files to create per directory
-y	Sync file after writing
-u	Unique working directory for each task
-t	Time unique working directory overhead
-F	Perform test on files only (no directories)
-w=N	bytes to write to each file after it is created
-e=N	bytes to read from each file