Spectrum Scale with NVMe-over-Fabrics: Deploying NetApp EF600

Last update: 01 July 2021

Introduces Spectrum Scale "SAN Mode" with NVMe-over-Fabrics

Describes how to configure NetApp EF6000 storage in an Infiniband HPC/AI fabric

Explains NVMe-over-Fabrics discovery, connect, and multipathing setup

Provides capacity and performance sizing guidance

Michael Hennecke

LENOVO PRESS

Abstract

NVMe-over-Fabrics on Infiniband networks makes the Spectrum Scale “SAN Mode” available to HPC and AI clusters. This deployment model, which is popular in enterprise Fibre Channel SAN networks, does not require any Spectrum Scale NSD servers as all application nodes can directly access all block storage volumes over the fabric. This guide shows how to use the NetApp EF600 storage system in a Spectrum Scale “SAN Mode” setup, using NVMe-over-Fabrics on Infiniband.

This Planning and Implementation Guide is intended for sales and technical sales specialists, solution architects, and storage administrators who want to deploy NVMe-over-Fabrics storage solutions like the NetApp EF600 with IBM Spectrum Scale. The paper will be most useful for technical professionals who have a working knowledge of high performance storage systems.

At Lenovo Press, we bring together experts to produce technical publications around topics of importance to you, providing information and best practices for using Lenovo products and solutions to solve IT challenges.

See a list of our most recent publications at the Lenovo Press web site:

http://lenovopress.com

Table of Contents

Overview

IBM Spectrum Scale is a high-performance, highly scalable, and highly reliable parallel file system and data management infrastructure. It is the foundation of several of Lenovo's HPC & AI storage solutions. In particular, the Lenovo DSS-G solution uses Spectrum Scale RAID to provide high performance software RAID functionality at scale without the need for block storage controllers. Complementing DSS-G, Spectrum Scale can also be used on top of traditional hardware RAID storage controllers, like the Lenovo DE-Series storage systems. In both of these solutions, the typical deployment mechanism uses a Client/Server architecture where the physical storage devices are connected to only a small number of NSD Servers, and the application nodes are accessing the Spectrum Scale file system through those NSD servers.

The NVMe-over-Fabrics technology provides an industry standard mechanism to map NVMe bock storage volumes to application nodes over a high-speed fabric. Its typical usage model is to map *different* volume to *separate* hosts, which then use those volumes in lieu of locally attached physical storage. But if an upper layer software stack can coordinate the concurrent access to NVMe-over-Fabrics volumes, a very different usage model is possible, where the *same* NVMe block storage volumes are mapped concurrently to *many* application nodes.

The Spectrum Scale file system has the ability to work directly with storage volumes that are accessible to all Spectrum Scale nodes through a storage-area network (SAN). For this reason, using Spectrum Scale in this "SAN Mode" with NVMe-over-Fabrics volumes on an InfiniBand fabric is a good combination of both technologies. It eliminates the NSD Client/Server layer, which should reduce the overall solution cost as well as improve I/O latencies. Note that the Spectrum Scale "SAN Mode" does not scale as high as the NSD Client/Server model, so it is targeted at Spectrum Scale clusters of less than a hundred nodes, not hundreds or thousands of nodes.

This planning and implementation guide first presents the different Spectrum Scale architectures. It then gives an overview of the NetApp EF600 storage system, which is available from Lenovo through the Lenovo *Vendor Logo Hardware* (VLH) program, and outlines the various InfiniBand topologies in which an EF600 could be deployed. The main part of the document provides the details of the EF600 storage setup, Linux host setup steps for NVMe-over-Fabrics in a typical HPC&AI Infiniband fabric, and the Spectrum Scale setup. The guide closes with guidelines for capacity sizing and performance sizing.

Note:
Some of the work presented in this Lenovo Press paper has been performed on pre-GA versions of the NetApp EF600 storage system firmware, and/or pre-GA versions of IBM Spectrum Scale. It should not be implied that a combination of software and firmware levels mentioned in this document is supported by Lenovo or NetApp in production environments. The Lenovo Scalable Infrastructure (LeSI) "best recipe" and the NetApp Interoperability Matrix Tool (IMT) are the definitive references that define which combinations of code levels are supported (see "Additional Resources" on page 55). Always check these documents before deploying an EF600 in a Lenovo cluster in production.

Spectrum Scale Architectures

The predominant Spectrum Scale architecture is show in Figure 1: In this model, only a small subset of the Spectrum Scale nodes have a direct physical connection to the disks (which can be physical disks, or LUNs served by a block storage controller). The connection between the disks and those severs is shown as a Storage Area Network (SAN) network in Figure 1. In the early days of GPFS, it was common that this was actually a Fibre Channel SAN, with FC-attached block storage controllers and FC host adapters in this subset of servers. It is still possible to use an FC SAN, but a more common and more cost effective way to build this type of Spectrum Scale Architecture is to use direct point-to-point (P2P) connections between a pair of servers (for redundant access) and the external storage (LUNs in block storage systems, or physical disks in JBODs). As the storage infrastructure grows, several of those building blocks can be deployed in a scale-out manner. The Lenovo DSS-G solution based on Spectrum Scale RAID follows exactly this architecture, where each DSS-G building block uses P2P connections between two DSS-G servers and up to eight JBODs per building block.

The Spectrum Scale nodes that are connected to the "SAN" (P2P, or in an actual SAN) then act as Spectrum Scale Network Shared Disk (NSD) Servers, and provide a typically much larger set of NSD Client nodes access to those "disks" through a TCP/IP based "LAN" network. The NSD "disks" can be LUNs on a hardware RAID controller, physical disks inside NSD servers, physical disks in external JBODs, or in the case of Spectrum Scale RAID they can also be virtual disks (vdisks).

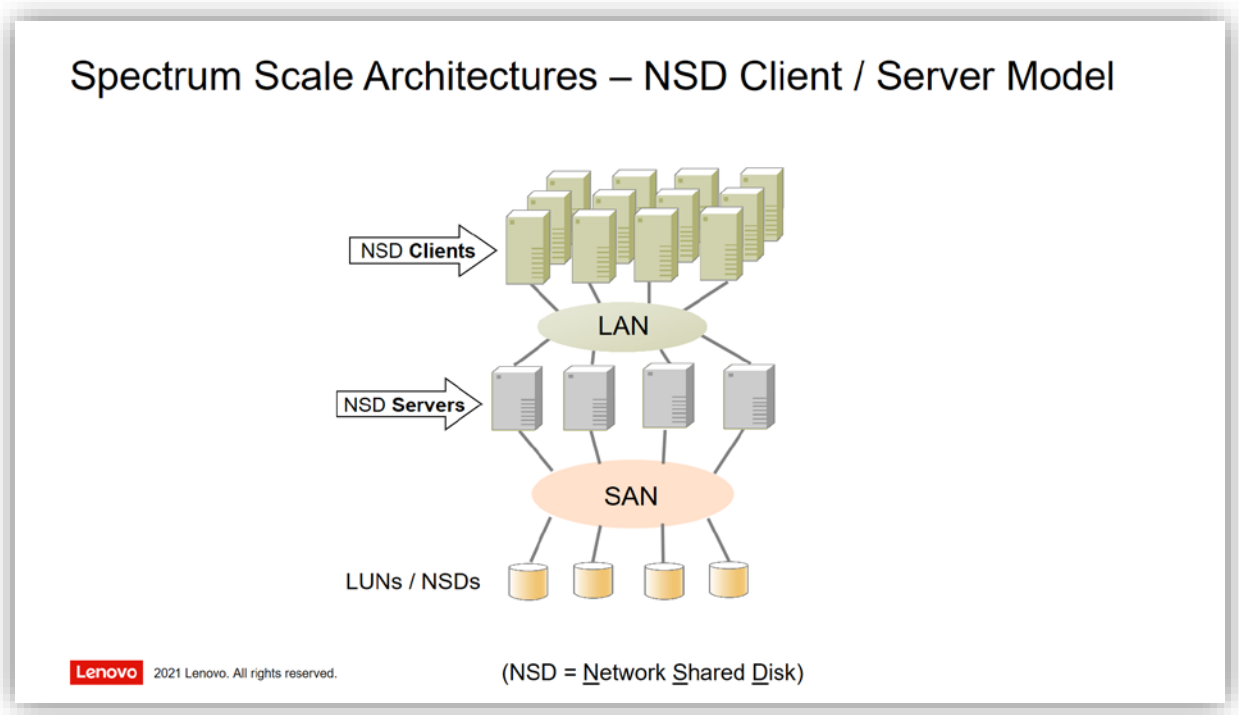

Figure 1: Spectrum Scale Architectures – NSD Client / Server Model.

The two biggest advantages of the NSD Client/Server model are its scalability (clusters with over ten thousand nodes have been implemented) and its cost effectiveness: In HPC/AI clusters, a high-speed "LAN" usually exists for the communication traffic of the applications, so the "SAN" can be limited to a few NSD Server nodes. Especially for heterogeneous environments, another advantage of the NSD Client/Server model is that the servers and (potentially multiple) client clusters can be put into different Spectrum Scale clusters with different administrative responsibility, using the Multi-Cluster feature of Spectrum Scale.

For smaller deployments, the Spectrum Scale *SAN Model* is an alternative to the NSD Client/Server model. This topology is shown in Figure 2. The Spectrum Scale file system layer itself is *not* based on a Client/Server architecture: All it requires is that the participating nodes can access the NSDs ("disks") on which the Spectrum Scale file system is built. To provide this access, the physical SAN infrastructure can be extended to all application nodes, and the LUNs of the SAN-connected block storage controllers are directly mapped/exported to all the application nodes.

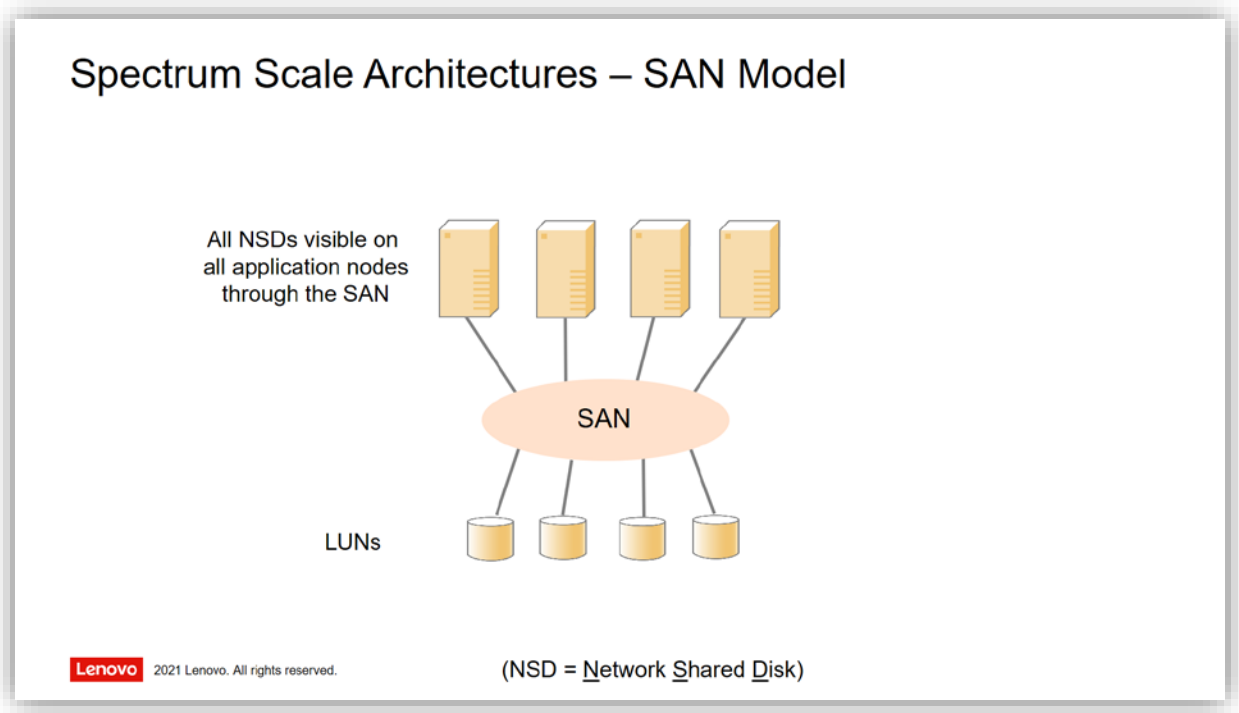

Figure 2: Spectrum Scale Architectures – SAN Model.

The SAN model can be very cost effective when the cost of a SAN port in the application nodes is smaller than the cost of additional NSD Servers (and a high-performance "LAN" network, if one does not already exist). It is also a simpler design, as it avoids the NSD Client/Server "middle layer". On the performance side, some workloads may actually perform better in the SAN Model: NSD Servers introduce an additional software layer that impacts storage latencies, and they are stateless by design which may have a negative performance impact for some types of workloads.

The Spectrum Scale SAN model is very popular with enterprise customers who are operating redundant Fibre Channel SAN fabrics, and who have a moderate number of application nodes that need access to the Spectrum Scale filesystem. (The testing limit for Spectrum Scale FC SAN deployments has been around 64 nodes.) On the other hand, in scientific computing the clusters are typically much larger, and already have a high-performance interconnect like InfiniBand or high-speed Ethernet, often with RDMA (Remote Direct Memory Access) transport. In those environments, the traditional FC SAN Model has not been an option, and the NSD Client/Server model is the dominant Spectrum Scale architecture in the scientific computing space.

In this document, we will use the emerging NVMe-over-Fabrics technology to implement the Spectrum Scale SAN Model on an InfiniBand fabric. Figure 3 shows this concept: The "SAN" is the Infiniband fabric of the HPC/AI cluster, with its lossless RDMA transport. Alternatively, RDMA over Converged Ethernet (RoCE) could also be used with a high-speed Ethernet fabric. The "SAN storage" is an end-to-end NVMe storage solution: In this document, we use the NetApp EF600 block storage controllers with InfiniBand Host Interface Cards (HICs) to provision the NVMe-over-Fabrics volumes.

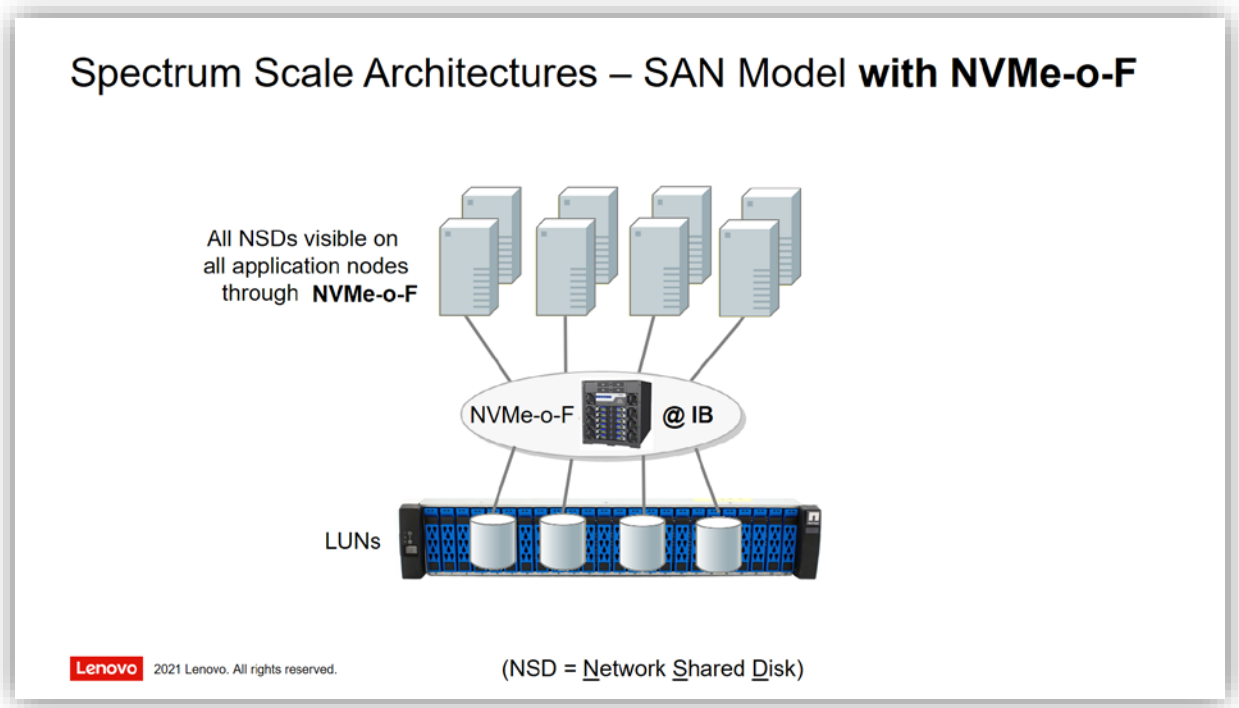

Figure 3: Spectrum Scale Architectures – SAN Model with NVMe-over-Fabrics.

InfiniBand Fabrics have been scaled to a much higher number of fabric endpoints than FC SAN infrastructures. The hardware scalability of this approach is therefore far exceeding the hardware scalability of traditional FC SAN fabrics. However, it should be noted that the software scalability will still be somewhat limited: As the block storage controller has to map all the LUNs to all client nodes, and managing those connections does consume resources on the controller, this deployment model should not be expected to scale to thousands of nodes (like the NSD Client/Server model does). But for moderately sized deployments, the NVMe-over-Fabrics model makes the high IOPS, high bandwidth, and low latency of end-to-end NVMe storage solutions directly accessible to the Spectrum Scale filesystem layer – without introducing the software overhead of the NSD Client/Server model and without the need for dedicated NSD server nodes.

NetApp EF600 Overview

The NetApp EF600 NVMe all-flash array is a classical block storage controller, whose functionality and manageability is comparable to the other NetApp E-Series storage arrays and the Lenovo DE-Series storage arrays. Based on its end-to-end NVMe design, its performance capabilities enable data intensive High Performance Computing applications as well as analytics and artificial intelligence (AI) workloads to run much faster than on previous generations of storage arrays.

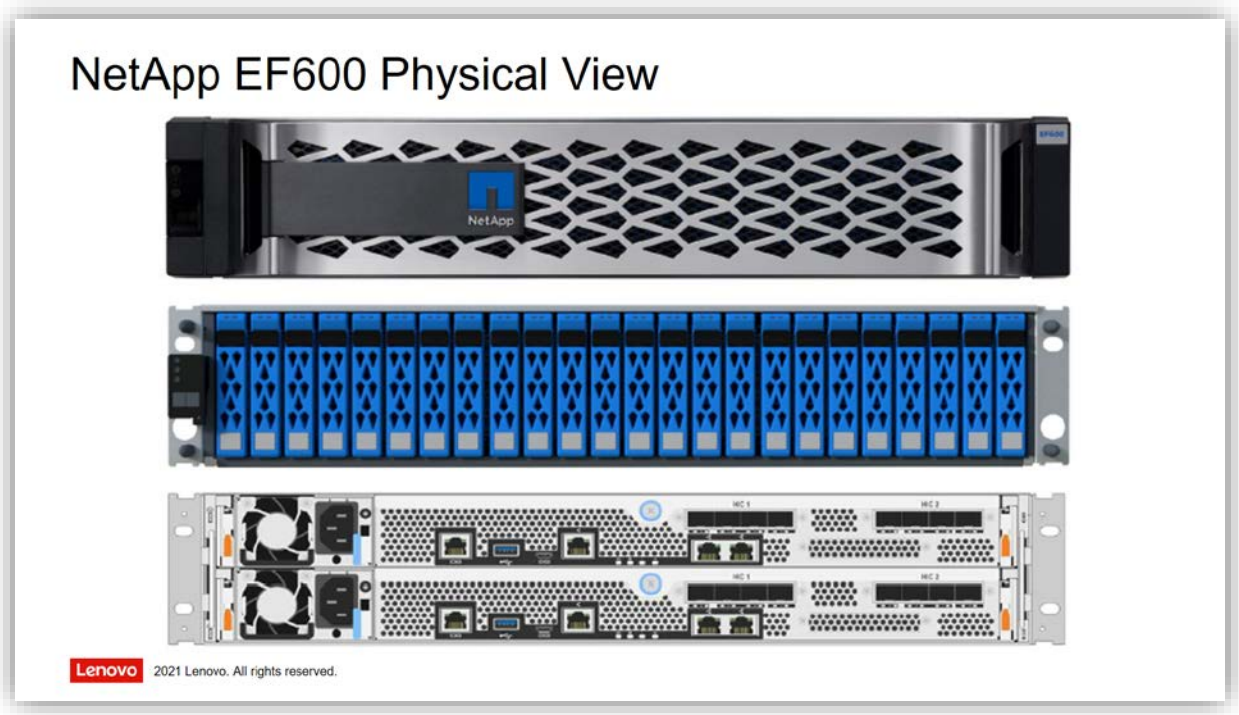

Figure 4: NetApp EF600 Physical View.

The EF600 is described in detail in the NetApp Technical Report TR-4800: *Introduction to NetApp EF600 Array* (https://www.netapp.com/pdf.html?item=/media/17009-tr4800pdf.pdf). As shown in Figure 9, it is a 2U storage system with 24 dual-ported NVMe drive slots in the front, and two redundant storage controller modules ("A" and "B") in the back. The EF600 currently supports the following NVMe drive capacities:

- 1.92 TB NVMe → 46 TB raw capacity with 24 drives (38 TB with 20 drives)
- 3.84 TB NVMe → 92 TB raw capacity with 24 drives (76 TB with 20 drives)
- 7.68 TB NVMe → 184 TB raw capacity with 24 drives (153 TB with 20 drives)
- 15.36 TB NVMe → 368 TB raw capacity with 24 drives (307 TB with 20 drives)

Note that "scale-up" expansion of the EF600 with additional disk enclosures is not supported. For performance-oriented solutions, it is always advisable to balance the controller capabilities with the NVMe disk capabilities, and 24 NVMe drives are already more than sufficient to saturate the EF600 controller capabilities (see "Performance Sizing" on page 48 for details). To grow an EF600 storage solution beyond 24 NVMe drives, multiple EF600 storage arrays can be used in a "scale-out" approach. Spectrum Scale then provides the parallel file system layer with a global namespace across all the EF600 storage arrays, and the aggregate performance of the solution will scale out accordingly.

The performance characteristics of the EF600 populated with 24 NVMe drives are as follows:

- Up to 2 million 4 kiB random I/O operations per second (IOPS) at 250 µsec response times
- Up to 44 GByte/s sequential read bandwidth
- Up to 12.5 GByte/s sequential write bandwidth in *Cache Mirroring Enabled* (CME) mode
- Up to 24 GByte/s sequential write bandwidth in *Full-Stripe Write Acceleration* (FSWA) mode

Note: These performance numbers are best-case results on the *block storage layer*. They do not include any filesystem layer with its associated overheads. In particular, local "IOPS" rates on the block storage level cannot be compared with "IOPS" or "metadata operations per second" in a parallel file system, where many more operations are required to ensure global consistency and to properly manage the file system metadata.

The EF600 is available with two different controller cache sizes: Either 32 GiB or 128 GiB per controller (64 GiB or 256 GiB in total for the dual-controller storage system). The EF600 also supports several different Host Interface Cards (HICs), and different storage protocols over those HIC ports. In this document we are focusing on InfiniBand, so the two HIC choices that are relevant here are:

- One 2-port 200 Gbps HDR Infiniband HIC per controller (four HDR IB ports per system)
- Two 2-port 100 Gbps EDR Infiniband HICs per controller (eight EDR IB ports per system)

The desired storage protocol is set at ordering time through a *feature pack sub-model ID* (FP-SMID). The correct FP-SMID for NVMe-over-Fabrics with InfiniBand is "**444**". To use NVMe-over-Fabrics with RoCE (using the same HIC modules), the correct FP-SMID would be "443".

In terms of cabling and network planning, the EF600 needs one 1 Gbps Ethernet connection per controller (two for the dual-controller storage system) for out-of-band management access. It also needs one InfiniBand cable and dedicated IP address in the InfiniBand fabric for *each* of the EF600's IB host ports (eight EDR cables and IP addresses when using the two EDR HICs per controller, and four HDR cables and IP addresses when using the one HDR HIC per controller). As an example when using eight EDR host ports, Figure 5 shows the `/etc/hosts` entries for the EF600 storage system in Lenovo's HPC Benchmarking system, named "de0704".

```
$ grep de0704 /etc/hosts

172.30.25.107    de0704a         de0704a.hpc.eu.lenovo.com
172.30.25.108    de0704b         de0704b.hpc.eu.lenovo.com

172.30.57.107    de0704a-ib0     de0704a-ib0.hpc.eu.lenovo.com
172.30.57.108    de0704a-ib1     de0704a-ib1.hpc.eu.lenovo.com
172.30.57.109    de0704a-ib2     de0704a-ib2.hpc.eu.lenovo.com
172.30.57.110    de0704a-ib3     de0704a-ib3.hpc.eu.lenovo.com

172.30.57.111    de0704b-ib0     de0704b-ib0.hpc.eu.lenovo.com
172.30.57.112    de0704b-ib1     de0704b-ib1.hpc.eu.lenovo.com
172.30.57.113    de0704b-ib2     de0704b-ib2.hpc.eu.lenovo.com
172.30.57.114    de0704b-ib3     de0704b-ib3.hpc.eu.lenovo.com
```

Figure 5: NetApp EF600 IP addresses for NVMe-over-Fabrics.

The minimum SANtricity software level to support the EF600 is 11.60. However, we strongly recommend to upgrade the storage array to the latest available level (at least 11.70.1).

EF600 and InfiniBand Fabric Topologies

In traditional enterprise SAN infrastructures using Fibre Channel, high availability is typically achieved by using two separate *SAN fabrics*, and connecting the SAN storage controllers to both of those SAN fabrics. The application nodes typically also have multiple FC SAN host ports, and are connected to both SAN fabrics as well. This design provides redundancy at all levels, including the failure of one of the two SAN fabrics. The SAN could be implemented by a single (zoned) 1U FC switch, a pair of SAN switches, or a pair of FC director switches with hundreds of ports. Figure 6 shows this traditional SAN fabric topology, using the FC worksheet from the NetApp guide *SANtricity Express Configuration for Linux*.

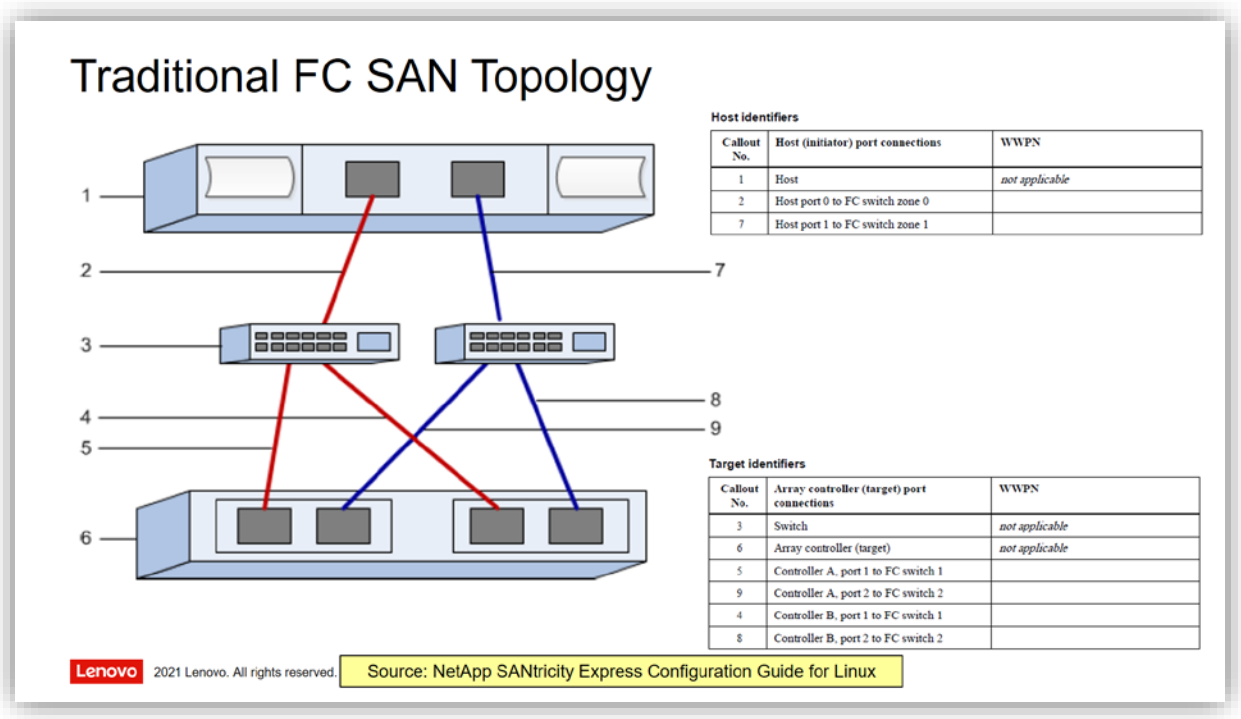

Figure 6: Traditional FC SAN Topology.

InfiniBand switches can be used to build a storage topology that corresponds exactly to these traditional Fibre Channel SAN topologies. In fact, the above NetApp guide recommends this topology for the configuration of the *iSER over Infiniband*, *SRP over InfiniBand*, and *NVMe over Fibre Channel* software stacks.

As an alternative to the fabric-based storage topologies, *direct attached* topologies are typically used with *SAS* host ports (they could also be used with Fibre Channel). This topology is shown in Figure 7. As there is no fabric, the number of hosts that can be connected to a storage system is limited by the number of host ports on each of the two controllers ("A" and "B") of the storage system.

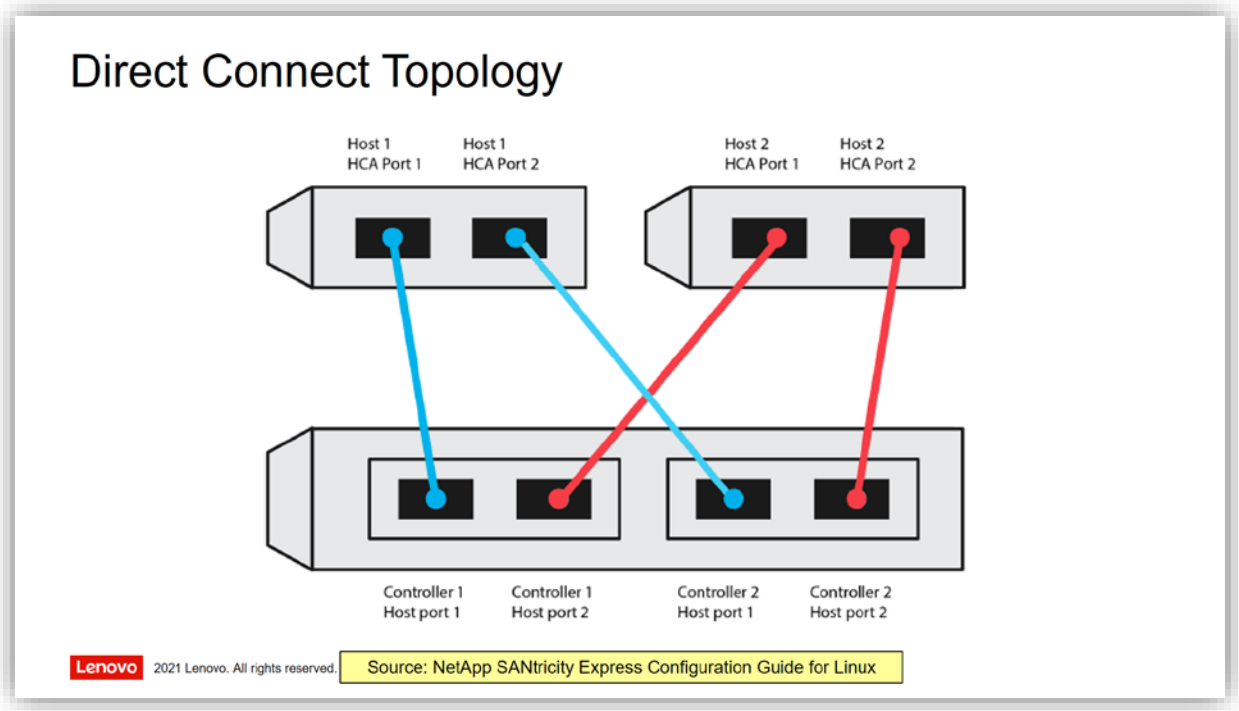

Figure 7: Direct Connect Topology.

The NetApp guide *Express Configuration for Linux* also recommends this direct attached topology for the *NVMe over InfiniBand* software stack, which is the subject of this document. While it is possible to do direct attach topologies with Infiniband, the NVMe-over-InfiniBand technology is not limited to direct attachment. In addition to the two-fabric designs that mimic traditional FC SAN fabrics, there are many other Infiniband switch-based topologies that are used in HPC and AI environments, and those topologies can also be used for NVMe-over-Fabrics.

Figure 8 shows a typical InfiniBand topology for HPC clusters: An InfiniBand "Clos" network topology is built by either using director-class InfiniBand switches, or by using 1U switches as spine switches and as leaf switches. Those InfiniBand topologies can support thousands of endpoints, and are deployed in many HPC clusters. In terms of storage attachment, it makes sense to attach the host ports of the EF600 to multiple InfiniBand leaf switches. This ensures high availability if an InfiniBand leaf switch goes down. Note that the color coding in Figure 8 does not imply any fabric separation … blue and red simply show the two "failure domains" given by the two IB leaf switches to which the EF600 is connected. If the EF600 has eight InfiniBand host ports, and the fabric is large enough, it is possible (and recommended) to spread out the EF600 host ports to four leaf switches, which reduces the impact of losing a single leaf switch even further.

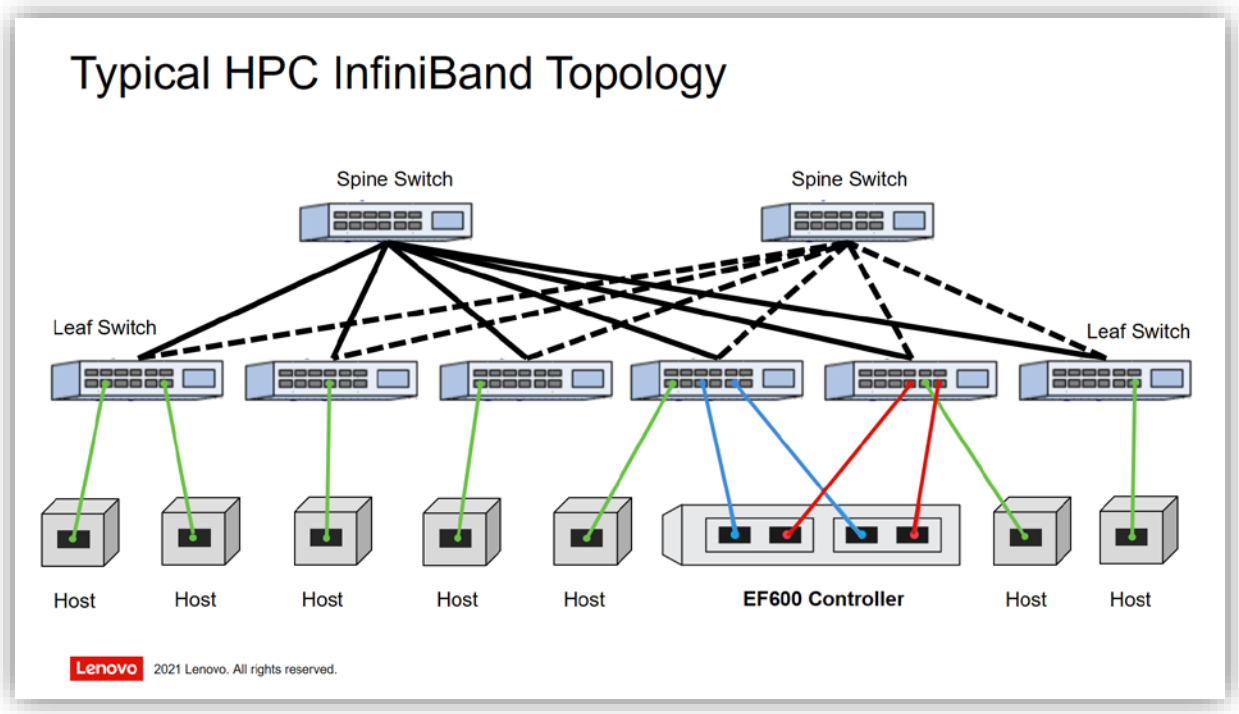

Figure 8: Typical HPC InfiniBand topology.

Note: Contrary to traditional enterprise SANs with mission-critical servers, compute nodes in HPC clusters are typically using only a single InfiniBand host port, as individual compute nodes are not considered to be mission-critical. Critical infrastructure components like storage systems are typically connected with multiple InfiniBand ports, as high availability is important for those systems.

The Lenovo HPC benchmarking center uses an InfiniBand topology comparable to Figure 8. This is the basis for the EF600 configuration steps that are described in this document: All endpoints on this fabric are visible to all other endpoints. Depending on where the two participants of a point-to-point communication are attached, the switching will happen either locally on the same IB leaf switch, or it will go through the spine layer of Infiniband switches to communicate between two different IB leaf switches. The additional latencies that are incurred by those extra hops are well below a microsecond. For large I/O requests this can be neglected, for small I/O on very fast storage media a small increase in latencies may be visible on the application side.

NetApp EF600 Storage Configuration

Like all NetApp E-Series and Lenovo DE-Series storage arrays, the configuration of the EF600 can be performed through the GUI, through a REST API, or through the `SMcli` command-line tool. For Spectrum Scale deployments, which are typically scale-out environments with several identical building blocks, our preferred choice to configure the storage arrays is the `SMcli` command-line tool.

The first step is to set a password for the "admin" user, and to set the IP addresses of the two management ports to their site-specific values. This can be done by physically connecting a laptop to the management port of the "A" and the "B" controllers, opening the GUI, and manually setting the IPs. The default IP addresses of the "A" and "B" management ports of the EF600 are 169.254.128.101 and 169.254.128.102, with a subnet mask of 255.255.0.0. The rest of the configuration can be performed from a Linux host that has connectivity into the management LAN (the `SMcli` tool can be downloaded using the GUI). Running

```
$ SMcli de0704a -k -u admin -p $Secret -f command-file.smcli
```

will connect to the "A" controller of our EF600 storage system using the `admin` user and `$Secret` password, and will then run the sequence of CLI commands that is saved in the `command-file.smcli` file. It is also possible to use tools like Ansible to perform the configuration.

Basic EF600 Storage Subsystem Configuration

The `SMcli` commands to perform basic storage subsystem configuration are shown in Figure 9. For parallel filesystem usage, it is **absolutely essential** that the `autoLoadBalancingEnable` configuration setting is set to **`false`**. This feature monitors controller load and dynamically assigns the controller ownership of LUNs based on current load. In a parallel filesystem where all LUNs have roughly equal load, this feature would cause a lot of random "path flapping" that would destroy the performance of the storage subsystem.

```
show "Setting storageArray configuration...";
set storageArray userLabel="de0704-ef600";
set storageArray time; // better: set up NTP
set storageArray cacheBlockSize=32;
set storageArray cacheFlushStart=50;
set storageArray mediaScanRate=30;
set storageArray autoLoadBalancingEnable=false;
set storageArray hostConnectivityReporting disable;
set storageArray defaultHostType=28; // Linux DM-MP (Kernel 3.10 or later)

show "Setting controller NVMe-over-Fabrics hostports IPs...";
set controller [a] hostPort ["1a"] IPV4Address=172.30.57.107;
set controller [a] hostPort ["1b"] IPV4Address=172.30.57.108;
set controller [a] hostPort ["2a"] IPV4Address=172.30.57.109;
set controller [a] hostPort ["2b"] IPV4Address=172.30.57.110;
set controller [b] hostPort ["1a"] IPV4Address=172.30.57.111;
set controller [b] hostPort ["1b"] IPV4Address=172.30.57.112;
set controller [b] hostPort ["2a"] IPV4Address=172.30.57.113;
set controller [b] hostPort ["2b"] IPV4Address=172.30.57.114;

show "Done.";
```

Figure 9: SMcli commands for basic storage subsystem configuration.

The script also sets the default host type to Linux, and finally it sets the IP addresses of the eight EDR controller host ports to the addresses shown in Figure 5.

Configuring the EF600 Storage Volumes

Similar to other NetApp E-Series and Lenovo DE-Series storage arrays, the EF600 supports both Dynamic Disk Pools (DDP) and classical RAID volumes (where RAID6 and RAID1 are typically used with Spectrum Scale for data and for metadata).

- **DDP** is a declustered RAID6 (8+2P) scheme on disk pools of 12 or more physical disks. It provides the best flexibility in terms of number of drives in the EF600. Its declustered RAID enables faster rebuild times in case of NVMe disk failures than classical RAID rebuilds. DDP pools provide excellent *read* bandwidth. But because DDP cannot use *Full-Stripe Write Acceleration (FSWA)*, using DDP severely limits the achievable *write* bandwidth to 12.5 GByte/s (CME), roughly half of the 24 GByte/s that are possible with FSWA.
- Classical **RAID6** (8+2P) volumes support FSWA and can therefore provide the maximum write bandwidth, but are inflexible in terms of the number of drives. Because the Spectrum Scale filesystem blocksize should always be equal to (or an integer multiple of) the RAID6 stripe width, a classical RAID6 volume group should always contain 8 data disks and 2 parity disks, so the choices for the EF600 are either *ten* or *twenty* disks. (The remaining disk slots can be used for other purposes, like additional RAID1 volumes for metadataOnly usage or hotSpare disks.)

Given these advantages and disadvantages, DDP is likely the best solution when the workload is predominantly read-oriented. However, when write bandwidth is important, then configuring the EF600 with only *20 NVMe disks* and two classical RAID6 (8+2P) volume groups is preferred over DDP.

When designing the pool or volume group layout, another important aspect to consider is the internal *PCIe bus structure* of the EF600: There are two PCIe buses that connect the controllers to the drives. One PCIe bus serves drive slots 0 to 11, and the second PCIe bus serves drive slots 12 to 23. In order to achieve the optimal performance, all pools and volume groups should have an equal number of member drives on each of these two PCIe buses. As outlined in the NetApp TR-4800, this can be achieved by either populating the drives from the center and moving outwards, or by populating the drives from both outer sides of the chassis and moving inwards. See the following subsections for specific examples.

To make sure that all PCIe paths between the controllers and the NVMe drives are utilized, we also recommend to always create *pairs* of equally sized volumes on a pool or a volume group, and assign one of them to controller “A” and the other to controller “B”. If this is not done, it will be difficult to achieve the peak bandwidth of the controller. For example, the achievable read bandwidth of the EF600 may drop from ~44 GB/s to only ~38 GB/s when not all PCIe paths are utilized.

Configuring Volumes on Dynamic Disk Pools

Dynamic Disk Pools (DDP) are NetApp's implementation of declustered RAID. For DDP with its minimum pool size of 12 drives, and a maximum pool size of all 24 drives in the EF600, it makes sense to put all populated drives into a single DDP pool. To achieve the goal to have an equal number of member drives on each of the two PCIe buses, an even-numbered drive count from 12 to 24 drives should be used. Figure 10 shows the recommended layout for DDP12, DDP20 and DDP24 pools. Figure 11 shows the `SMcli` commands to perform the `create diskPool` and `create Volume` operations for the DDP24 pool, plus the recommended volume settings for the usage of these volumes with Spectrum Scale.

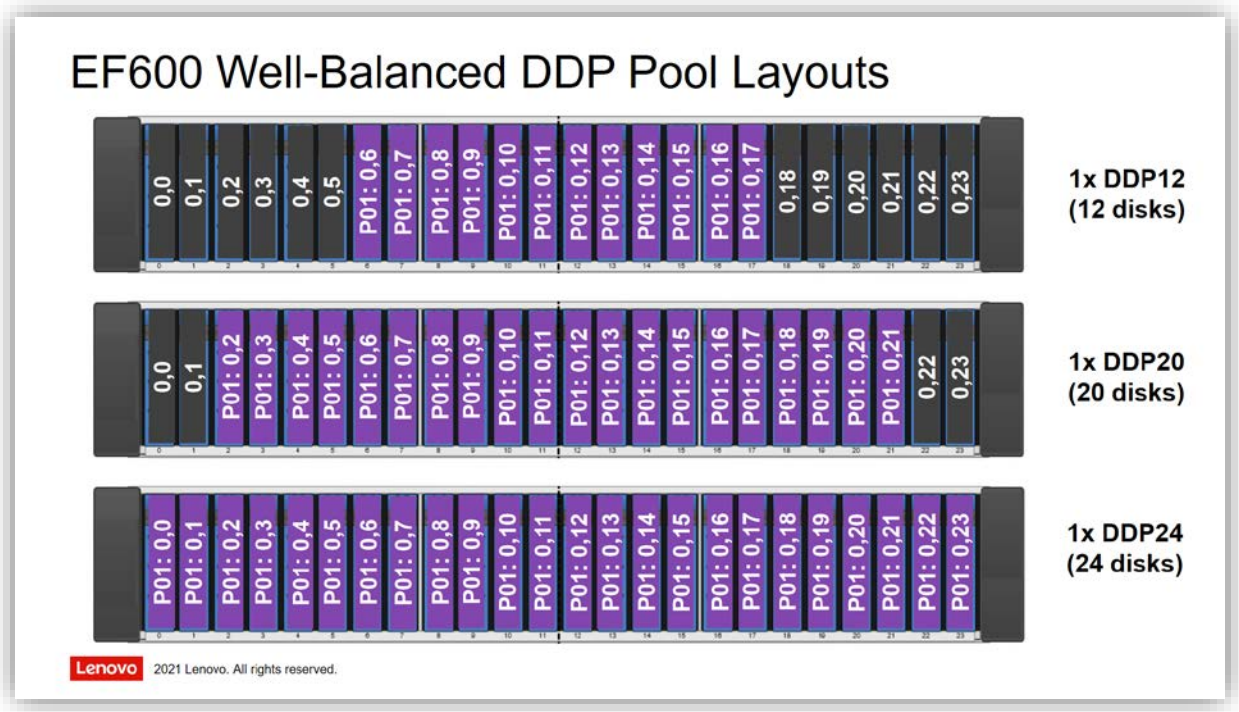

Figure 10: Well-balanced DDP Pool Layouts.

Note that we are creating *four* volumes in the *single* `"de0704p01"` disk pool. The reason for this is load balancing: For the NetApp E-Series and Lenovo DE-Series storage arrays, controller ownership of a volume is always active-passive: At any one time, either the "A" controller or the "B" controller serves a given volume. For load balancing across the two controllers, it is therefore recommended to always create an even number of volumes and set their controller ownership accordingly. In the Spectrum Scale SAN mode, *two* volumes per EF600 system could therefore be enough. However, we have seen better performance for some workloads if a total of four volumes are used, two on each controller. (In the NSD Client/Server model, as NSD server ownership is also active/passive, it is always recommended to create volumes in multiples of *four* to be able to evenly balance all paths between both controllers and both NSD servers.)

As the details are workload dependent, it may be beneficial to generate some performance baseline data with a varying number of volumes for your specific workload mix to decide which (even-numbered) number of volumes works best.


```
on error stop;

show "Creating DDP24 disk pool...";
create diskPool drives=(
  0,0  0,1  0,2  0,3  0,4  0,5  0,6  0,7  0,8  0,9  0,10 0,11
  0,12 0,13 0,14 0,15 0,16 0,17 0,18 0,19 0,20 0,21 0,22 0,23)
  userLabel="de0704p01" trayLossProtect=false
  drawerLossProtect=false criticalThreshold=0
  securityType=none dataAssurance=none;

show "Creating volumes on DDP disk pools...";
create volume diskPool="de0704p01" userLabel="de0704p01v1"
  capacity=4096GB owner=A thinProvisioned=false
  cacheReadPrefetch=false dataAssurance=none;
create volume diskPool="de0704p01" userLabel="de0704p01v2"
  capacity=4096GB owner=B thinProvisioned=false
  cacheReadPrefetch=false dataAssurance=none;
create volume diskPool="de0704p01" userLabel="de0704p01v3"
  capacity=4096GB owner=A thinProvisioned=false
  cacheReadPrefetch=false dataAssurance=none;
create volume diskPool="de0704p01" userLabel="de0704p01v4"
  capacity=4096GB owner=B thinProvisioned=false
  cacheReadPrefetch=false dataAssurance=none;

show "Setting volume redundancy checks...";
set volume ["de0704p01v1"] redundancyCheckEnabled=true;
set volume ["de0704p01v2"] redundancyCheckEnabled=true;
set volume ["de0704p01v3"] redundancyCheckEnabled=true;
set volume ["de0704p01v4"] redundancyCheckEnabled=true;

show "Setting allVolumes configuration...";
set allVolumes cacheFlushModifier=10;
set allVolumes cacheWithoutBatteryEnabled=false; // true==DANGER!
set allVolumes mediaScanEnabled=true;
set allVolumes mirrorCacheEnabled=true;
set allVolumes readCacheEnabled=true;
set allVolumes writeCacheEnabled=true;
set allVolumes cacheReadPrefetch=false; // let GPFS do the prefetch

show "Done.";
```

Figure 11: SMcli commands for DDP pool and volume configuration.

Configuring Volumes on RAID Volume Groups

To use the NetApp E-Series or Lenovo DE-Series storage arrays' performance feature of *Full Stripe Write Acceleration (FSWA)*, the best configuration choice is to use RAID6 (8+2P) volume groups with *ten* drives each. Figure 12 shows the recommended layout for one or two RAID6 (8+2P) volume groups in the EF600. Also shown at the bottom is the option to utilize the remaining four free disk slots for two additional RAID1 volume groups – this may be beneficial for some workloads, for example for dedicated metadata LUNs.

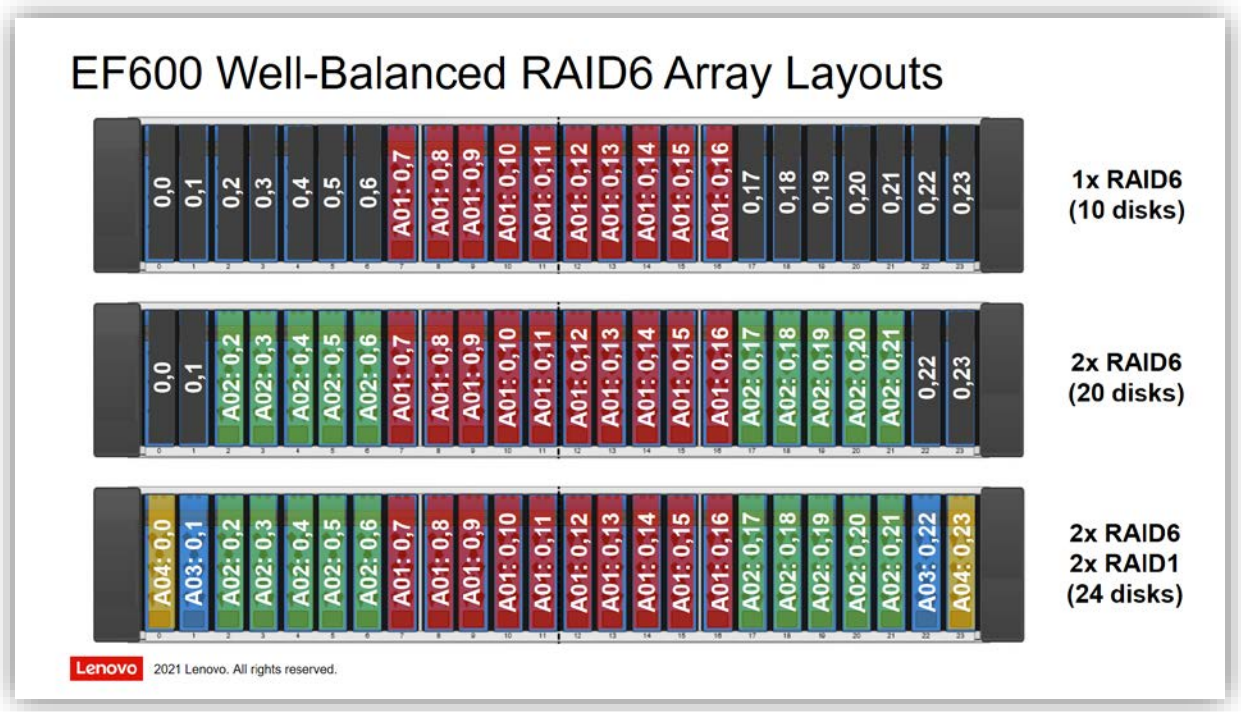

Figure 12: Well-balanced RAID Volume Group Layouts.

To use FSWA, the full stripe width of the RAID6 (8+2P) volume groups must be 2 MiB or smaller (corresponding to a strip width or `segmentSize` of 256 kiB or smaller). FSWA does not work with 4 MiB stripes (corresponding to a strip width or `segmentSize` of 512 kiB).

When chosing a RAID6 stripe width, it is important to make sure that the Linux hosts can actually send I/O requests of that size to the storage subsystem. This is mainly governed by the **`max_sectors_kb`** setting of the block devices (and the multipathing devices that are sitting on top of the individual block devices). If this setting is smaller than the RAID6 stripe width, the driver will split up an application I/O request to chunks of size `max_sectors_kb` or smaller. The storage system will then not see full-stripe writes, and will typically not be able to use the FSWA feature to speed up those write requests.

When using the EF600 with NVMe-over-Fabrics over InfiniBand, the largest supported value for `max_sectors_kb` may depend on the Linux hosts' operating system version. For RHEL 8.2, the default (and maximum supported) setting for NVMe-over-Fabrics devices is 1024 kiB. With eight data disks and two parity disks, a full stripe width of 1024 kiB corresponds to a `segmentSize` of 128 kiB.

Figure 13 shows the `SMcli` commands to perform the `create volumeGroup` and `create Volume` operations for two RAID6 (8+2P) volume groups, plus the recommended volume settings for the usage of these volumes with Spectrum Scale. In this example we use a `segmentSize` of 128 kiB which is the maximum that's possible on our RHEL 8.2 Linux hosts.

As in the DDP case, we are creating a total of four volumes – two volumes on each of the two RAID6 volume groups. To optimally use all internal PCIe paths to the NVMe drives, it is important to serve one volume in a volume group by controller "A", and the other volume by controller "B". If this best practice is not followed, the maximum achievable read performance will be significantly lower (as not all PCIe paths will be used).


```
on error stop;

show "Creating RAID6 volumes...";
create volumeGroup
  drives=(0,7  0,8  0,9  0,10 0,11 0,12 0,13 0,14 0,15 0,16)
  raidLevel=6 userLabel="de0704a01"
  drawerLossProtect=false trayLossProtect=false
  securityType=none dataAssurance=none;
create volumeGroup
  drives=(0,2  0,3  0,4  0,5  0,6  0,17 0,18 0,19 0,20 0,21)
  raidLevel=6 userLabel="de0704a02"
  drawerLossProtect=false trayLossProtect=false
  securityType=none dataAssurance=none;

show "Creating volumes on RAID6 volumeGroup...";
create volume volumeGroup="de0704a01" userLabel="de0704a01v1"
  capacity=4992GB owner=A segmentSize=128 cacheReadPrefetch=false
  dssPreAllocate=false securityType=none dataAssurance=none;
create volume volumeGroup="de0704a01" userLabel="de0704a01v2"
  capacity=4992GB owner=B segmentSize=128 cacheReadPrefetch=false
  dssPreAllocate=false securityType=none dataAssurance=none;
create volume volumeGroup="de0704a02" userLabel="de0704a02v1"
  capacity=4992GB owner=A segmentSize=128 cacheReadPrefetch=false
  dssPreAllocate=false securityType=none dataAssurance=none;
create volume volumeGroup="de0704a02" userLabel="de0704a02v2"
  capacity=4992GB owner=B segmentSize=128 cacheReadPrefetch=false
  dssPreAllocate=false securityType=none dataAssurance=none;

show "Setting volume redundancy checks...";
set volume ["de0704a01v1"] redundancyCheckEnabled=true;
set volume ["de0704a01v2"] redundancyCheckEnabled=true;
set volume ["de0704a02v1"] redundancyCheckEnabled=true;
set volume ["de0704a02v2"] redundancyCheckEnabled=true;

show "Setting allVolumes configuration...";
set allVolumes cacheFlushModifier=10;
set allVolumes cacheWithoutBatteryEnabled=false; // true==DANGER!
set allVolumes mediaScanEnabled=true;
set allVolumes mirrorCacheEnabled=true;
set allVolumes readCacheEnabled=true;
set allVolumes writeCacheEnabled=true;
set allVolumes cacheReadPrefetch=false; // let GPFS do the prefetch

show "Done.";
```

Figure 13: SMcli commands for RAID6 volume group and volume configuration.

Note: Should you need to clean up the volume definitions on the storage array, for example to experiment with other settings for the volumes, this can be done either through the GUI or through `SMcli` with the `"clear storageArray configuration volumeGroups;"` command, followed by `"clear storageArray recoveryMode;"`. This removes all volumes and volume groups/pools, but keeps all other configuration information including the host topology.

Offline and Online Volume Formatting

Note that as soon as a volume has been created, the storage system will start a formatting operation for that volume in the background. This formatting happens on the controller ("A" or "B") that "owns" the volume. On the EF600, the "A" controller and the "B" controller can both format a single volume at a time. If they own more than one volume (in the same pool/volume group, or in separate pools/volume groups), the formatting of those volumes will happen sequentially.

Formatting is a time consuming operation, and it is important to understand that there is a big performance difference between *offline* and *online* formatting: As long as a volume has not been mapped to a host group (see "Mapping the Volumes" on page 23), no host can access it. In this case the formatting can be done in *offline* mode, which is significantly faster. When the volume gets mapped to a host group, the controller falls back to a much slower *online* formatting. While it is tempting to proceed with the configuration work while the online formatting is running in the background, we recommend to wait for the completion of the *offline* formatting before mapping the volumes. In many deployments, the initial bring-up is followed by performance testing to validate the baseline performance. The application performance will be significantly impacted while the volumes are being formatted, so this step cannot be done before the formatting has completed. This means that the overall deployment including the performance testing will likely complete faster when the configuration work is paused here until offline formatting has completed. Figure 14 shows an example, where offline formatting only takes around 10 minutes, while online formatting of the same volumes takes over one hour (per volume on each of the controllers).

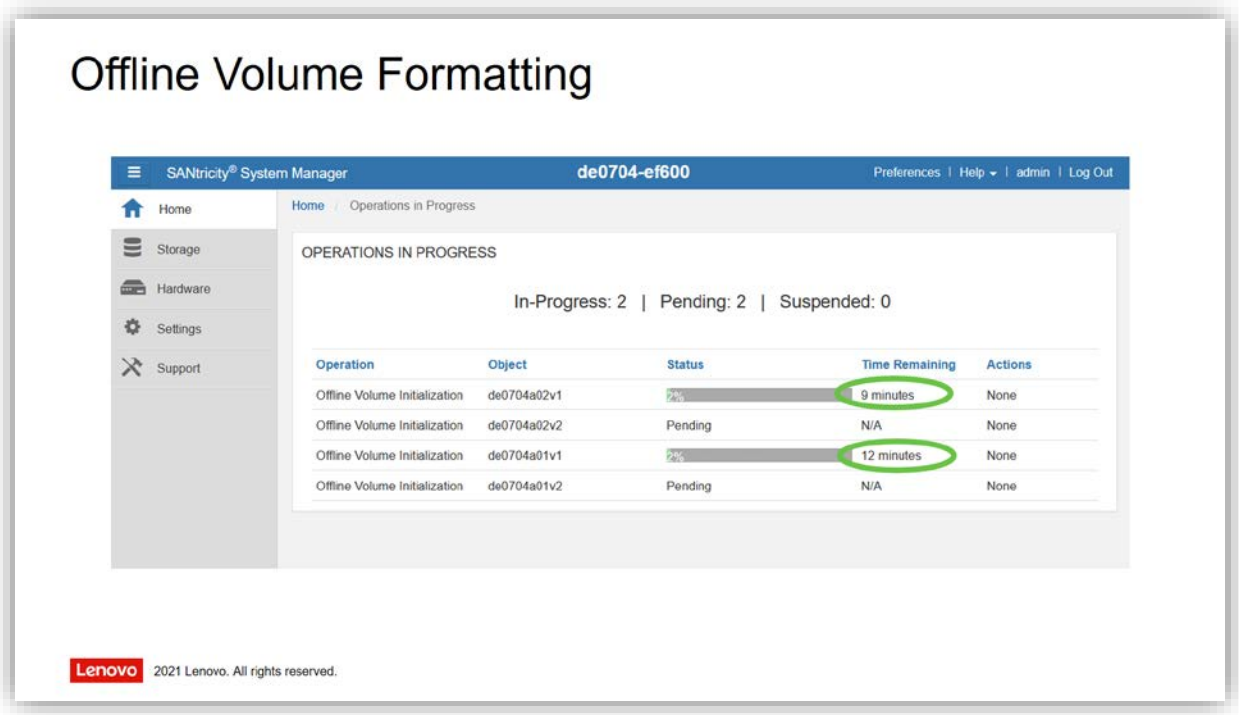

Figure 14: EF600 offline volume formatting.

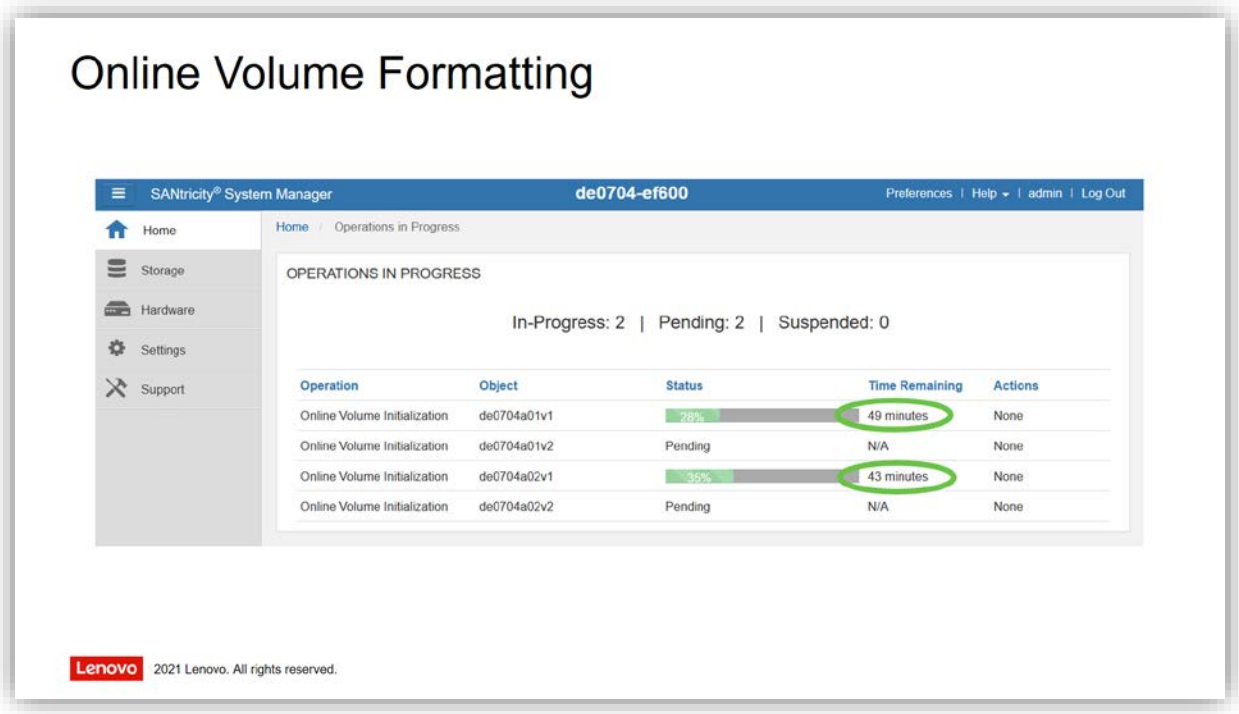

Figure 15: EF600 online volume formatting.

Note: The EF600 host groups *can* be created in parallel to the offline volume formatting, as described in “Creating the Host Topology for NVMe-over-Fabrics” on page 22. *Creating* the host group(s) does not affect the formatting process, as long as the volumes are not yet *mapped* to the host group(s).

Creating the Host Topology for NVMe-over-Fabrics

Creating the host topology on a Lenovo DE-Series or NetApp E-Series storage subsystem consists of three steps: First a named host group is created with the `create hostGroup` command, then hosts are created (and associated with a host group) using the `create host` command, and finally the storage connections of those hosts are defined. For SAS and FC host connections, the `SMcli` command for this third step is `create hostPort`, whereas for NVMe-over-Fabric connections the `create initiator` command with `interfaceType=nvmeof` is used.

An `SMcli` script to set up the host topology for one of the compute node racks in the Lenovo HPC Benchmarking Center (nodes cmp2501 to cmp2554) is shown in Figure 16:

```
on error stop;

show "Creating hostGroup...";
create hostGroup userLabel="de0704_hg1";

show "Creating hosts...";
create host userLabel="cmp2501" hostType=28 hostGroup="de0704_hg1";
create host userLabel="cmp2502" hostType=28 hostGroup="de0704_hg1";
create host userLabel="cmp2503" hostType=28 hostGroup="de0704_hg1";
...
create host userLabel="cmp2554" hostType=28 hostGroup="de0704_hg1";

show "Creating NVMe-over-Fabrics initiators...";
create initiator
  identifier="nqn.2020-12.com.lenovo.eu.hpc:nvme:cmp2501-ib0"
  userLabel="cmp2501-ib0" host="cmp2501" interfaceType=nvmeof;
create initiator
  identifier="nqn.2020-12.com.lenovo.eu.hpc:nvme:cmp2502-ib0"
  userLabel="cmp2502-ib0" host="cmp2502" interfaceType=nvmeof;
create initiator
  identifier="nqn.2020-12.com.lenovo.eu.hpc:nvme:cmp2503-ib0"
  userLabel="cmp2503-ib0" host="cmp2503" interfaceType=nvmeof;
...
create initiator
  identifier="nqn.2020-12.com.lenovo.eu.hpc:nvme:cmp2554-ib0"
  userLabel="cmp2554-ib0" host="cmp2554" interfaceType=nvmeof;

show "Done.";
```

Figure 16: SMcli commands to create the NVMe-over-Fabrics host topology.

In a SAS or FC fabric, the `identifier` that is used to identify a `hostPort` is the World-Wide Name (WWN) of the SAS or FC port of the host channel adapter (HCA) in the server. The storage array also has a unique WWN that can be used by the hosts to address the storage array.

In NVMe-over-Fabrics environments, the property that loosely corresponds to the SAS or FC WWN is the NVM Subsystem *NVMe Qualified Name* (NQN). This NQN is used in the `identifier` of the host port in the `create initiator` command.

Section 7.9 of the *NVM Express base specification, revision 1.4b,* defines the format and encoding of NVMe Qualified Names (NQNs). Two different formats are supported. The first format uses a UUID as a unique identifier and hast the following structure:

nqn.2014-08.org.nvmexpress:uuid:783f3338-eda8-46d9-bb27-dd14cdcb4a1b

This NQN format is always available in an NVMe environment. One big disadvantage of this format is that the UUID that is created by the NVMe subsystem on a Linux host is not persistent. At each reboot, a new UUID is generated and consequently any host port definitions that may have been used on a storage subsystem like the EF600 will no longer work. The UUID format also does not provide an easy way to identify the host to which it belongs in a human-readable format.

The NVM Express specification therefore defines an alternative format, which can be used by organizations that own a domain name. This type of NQN is made unique by using the per-organization domain name (in reverse order and with a "date code" prefix in "yyyy-mm" format), and combining it with a human-readable text string that must be unique within the organization. These NQN identifiers are maintained by the organization that owns the domain, and they are persistent by design.

For the EF600 setup, we are using this second format of NQN as we need persistence to define the host topology. For example, the NQN identifier of our first compute node in Figure 16 is this:

nqn.2020-12.com.lenovo.eu.hpc:nvme:cmp2501-ib0

We are using the IP name of the "ib0" interface of the node as the unique text string after the ":nvme:" separator string. As those IP names are unique within our environment, the resulting NQN is unique, too. See below for details how those host NQN identifiers are used on the Linux hosts.

Mapping the Volumes

With the host topology defined, the final step to make the volumes available to the hosts on the Infiniband fabric is to map the volumes to the host group. This is done with the `set volume` command, and is done in the exact same way as for SAS or FC host ports. Figure 17 shows the necessary commands, using the four RAID6 volumes from above as an example.

```
show "Creating volume mappings...";

set volume ["de0704a01v1"] logicalUnitNumber=1  hostGroup="de0704_hg1";
set volume ["de0704a01v2"] logicalUnitNumber=2  hostGroup="de0704_hg1";
set volume ["de0704a02v1"] logicalUnitNumber=3  hostGroup="de0704_hg1";
set volume ["de0704a02v2"] logicalUnitNumber=4  hostGroup="de0704_hg1";

show "Done.";
```

Figure 17: SMcli commands to map volumes to host groups.

Note: With FC or SAS host ports, the NetApp E-Series and Lenovo DE-Series storage arrays create an "access volume" for in-band management as soon as the first volume is mapped to a host group. As we only perform out-of-band management of the storage array through the Ethernet management ports, we usually remove that "access volume" (and free up the LUN number that it consumed). No "access volume" is created For NVMe-over-Fabrics host ports, and that cleanup step is not necessary.

NVMe Configuration on the Linux Hosts

NVMe-over-Fabrics support in the operating system is still relatively new. Care must be taken to only use supported levels of the various software components. The work for the current document has been performed on Linux hosts with the RHEL 7 operating system, and the same testing has been repeated on Linux hosts with RHEL 8. Based on this testing, our strong recommendation is: **Do not mix RHEL 7 and RHEL 8** in a Spectrum Scale "SAN Mode" cluster on the same NVMe-over-Fabrics environment. The NVMe-over-Fabrics implementations, as well as the multipathing layer on top of the individual NVMe-over-Fabrics block devices, are radically different in the two operating system versions. If a mix of RHEL 7 and RHEL 8 hosts need to be served, it is best to place those hosts into two separate Spectrum Scale "SAN Mode" clusters to avoid any issues that may be caused by the different NVMe-over-Fabrics implementations.

The Lenovo Scalable Infrastructure (LeSI) cluster support is provided based on a "best recipe" of software and firmware levels, whose interoperability has been extensively tested. The NetApp EF600 is available from Lenovo through the Vendor Logo Hardware (VLH) path, and the EF600 storage software is not yet part of the LeSI best recipe. Nevertheless, it is recommended to stick to a supported LeSI best recipe for the host software, and to also validate those levels with the latest NetApp Interoperability Matrix Tool (IMT).

Installing MOFED with NVMe-over-Fabrics Support

The Lenovo LeSI best recipe mandates to use the Mellanox OFED (MOFED) stack for Infiniband support, not just the Linux inbox OFED. All work for the current document has been performed on Linux hosts with MLNX_OFED_LINUX-5.0-2.1.8.0 (OFED-5.0-2.1.8). Note that to enable the NVMe-over-Fabrics support in MOFED, it has to be built with NVMe-over-Fabrics support:

```
$ ./mlnxofedinstall --add-kernel-support --with-nvmf
```

The InfiniBand interface status can be checked with `ibstat`. The IP interface needs to be configured with the host's "`ib0`" IP address and IP name, typically by setting up the `ifcfg-ib0` configuration file in `/etc/sysconfig/network-scripts`. After bringing up the interface, check its status:

```
$ ibstat
$ vi /etc/sysconfig/network-scripts/ifcfg-ib0
$ ifup ib0
$ ip addr show ib0
```

Finally, the `nvme-rdma` module needs to be loaded:

```
$ echo "nvme-rdma" >> /etc/modules-load.d/nvme-rdma.conf
$ modprobe nvme-rdma
$ lsmod | grep nvme
```

With these preparations, the Mellanox OFED stack should be ready for the setup of the actual NVMe-over-Fabrics functionality.

See also: https://community.mellanox.com/s/article/howto-configure-nvme-over-fabrics

Setting up DM-Multipath for the EF600

The volumes on the EF600 are visible to the application nodes through all (four or eight) InfiniBand host ports of the EF600. In RHEL 7, the Linux DM-Multipath infrastructure will be used to manage the access to the volumes through those paths. In RHEL 8 the multipathing is managed by the NVMe-over-Fabrics layer, but it is still recommended to configure DM-Multipath to properly detect report the EF600 paths. If not already installed, the `device-mapper-multipath` software needs to be installed, and a configuration file for the EF600 needs to be created before starting the service. These steps are shown in Figure 18.

```
$ yum install -y device-mapper-multipath

$ cat > /etc/multipath.conf <<E_O_F
#
# NetApp EF600 NVMe-over-Fabrics devices:
#
devices {
  device {
    vendor               "NVME"
    product              "NetApp E-Series*"
    path_grouping_policy "group_by_prio"
    failback             "immediate"
    no_path_retry        30
  }
}
E_O_F

$ systemctl enable multipathd
$ systemctl start  multipathd
```

Figure 18: DM-Multipath installation and configuration for the EF600.

Discovering and Connecting to the EF600

Like locally installed NVMe devices, NVMe-over-Fabrics devices are managed by the `nvme` command. If not already installed, the `nvme-cli` package needs to be installed on each application node:

```
$ yum install -y nvme-cli
```

To gain access to the volumes on the EF600, a two-step process is needed. First, an `nvme discover` operation for each host port of the EF600 is needed to discover that path. After all paths into the EF600 storage array have been discovered, an `nvme connect` operation for each of the host ports of the EF600 will make the NVMe-over-Fabrics devices visible. Those two operations are not persistent across reboots, so they need to be placed into a startup script that is called either at system boot time, or as part of a job prolog.

In the Lenovo HPC benchmarking center, this NVMe-over-Fabrics setup and the subsequent Spectrum Scale filesystem mount operation is managed through a Slurm job prolog/epilog mechanism. In a larger

cluster, this has the advantage that only nodes that actually need access to the EF600 storage will connect to it, keeping the overhead low.

```
$ man nvme-discover

$ # do this for each of the eight host ports:

$ # Attention: Must use an IP address; IP names do not work for -a
$ nvme discover -t rdma -a 172.30.57.107

Discovery Log Number of Records 8, Generation counter 0
=====Discovery Log Entry 0======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified
portid:  0
trsvcid: 4420
subnqn:  nqn.1992-08.com.netapp:6000.6d039ea0003ef5100000000059729fff
traddr:  172.30.57.107
rdma_prtype: infiniband
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
=====Discovery Log Entry 1======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified
portid:  3
trsvcid: 4420
subnqn:  nqn.1992-08.com.netapp:6000.6d039ea0003ef5100000000059729fff
traddr:  172.30.57.110
rdma_prtype: infiniband
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
=====Discovery Log Entry 2======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified
portid:  1
trsvcid: 4420
subnqn:  nqn.1992-08.com.netapp:6000.6d039ea0003ef5100000000059729fff
traddr:  172.30.57.108
rdma_prtype: infiniband
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
=====Discovery Log Entry 3======
...
```

Figure 19: nvme discover command for per-port discovery of the EF600 host ports.

Figure 19 shows the `nvme discover` operation for one of the host port IP addresses. This has to be repeated for each of the host ports. In the output for a single host port, one section for *each* of the eight host ports is returned. These should all be identical (including the `subnqn` NQN identifier of the EF600 storage array), except for the `portid` and `traddr`, which is the port number and IP address of the host ports. The order in which the output for the individual host ports is returned is not deterministic, so do not expect any particular order.

Note: Instead of running this command individually for each of the host ports, it is possible to create an `/etc/nvme/discovery.conf` configuration file, and then run a single `nvme discover` command to act on all (four or eight) host ports at once. This is discussed below in conjunction with the `nvme connect-all` command.

After the `nvme discover` step, the `nvme connect` operation can be used to create a transport connection for an individual host port of the EF600. Like the `nvme discover` operation, this needs to be performed for each of the (four or eight) NVMe host ports of the EF600.

While the `nvme discover` only needed the IP address of an EF600 host port, the `nvme connect` requires the EF600 NQN as well. In the example in Figure 20, we just hardcode it to the NQN value of the EF600 in our benchmark center. In a production environment, it is recommended to obtain the EF600 NQN dynamically from the output of the `nvme discover` operation.

The host's NQN also has to be passed in, unless the system default host NQN is used. As explained above, we prefer to use persistent NQN identifiers that are defined by the organization, and thus we need to pass in this custom host NQN as an argument to the `nvme connect` command.

The other settings for this command are based on NetApp recommendations for the EF600. As always, best practices may depend on the size of the solution as well as the workload. So it is a good idea to do some baseline testing with different settings for those tunables.

```
$ man nvme-connect

$ EF600_NQN="nqn.1992-08.com.netapp:6000.6d039ea0003ef5100000000059729fff"
$ HOST_NQN="nqn.2020-12.com.lenovo.eu.hpc:nvme:`hostname -s`-ib0"

$ queue_size=1024    # default is 128
$ ctrl_loss_tmo=3600 # default is 600

$ # do this for each of the eight host ports:

$ nvme connect -t rdma -n $EF600_NQN -a 172.30.57.107 \
   -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
```

Figure 20: nvme connect command for per-port connection of the EF600 host ports.

It is good to understand the individual per-port `nvme discover` and `nvme connect` commands, as described above. However, in production deployments it is more convenient to store all the required configuration information in an `/etc/nvme/discovery.conf` file, and then use a single invocation of the `nvme connect-all` command to perform the discover and connect operations for all (four or eight) host ports of the EF600 in a single sweep. This is shown in Figure 21. Note that the EF600 NQN is not needed in this case, as the `nvme connect-all` command will also perform the discovery step and will automatically derive the EF600 NQN from that discovery.


```
$ man nvme-connect-all

$ HOST_NQN="nqn.2020-12.com.lenovo.eu.hpc:nvme:`hostname -s`-ib0"

$ queue_size=1024    # default is 128
$ ctrl_loss_tmo=3600 # default is 600

$ cat > /etc/nvme/discovery.conf <<E_O_F
# NVMe-over-Fabrics discovery of de0704 (NetApp EF600)
-t rdma -a 172.30.57.107 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.108 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.109 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.110 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.111 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.112 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.113 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
-t rdma -a 172.30.57.114 -q $HOST_NQN -Q $queue_size -l $ctrl_loss_tmo
E_O_F

$ # do this only once for the EF600 storage array:

$ nvme connect-all
```

Figure 21: nvme connect-all command to connect to all EF600 host ports.

Note: After the `nvme connect-all` has been run, it usually takes a few seconds for all the NVMe-over-Fabrics devices to become visible on the node. If this operation is performed as part of a start-up script, it is a good idea to add a small waiting time (or check if the number of visible paths matches the expected number of paths) before proceeding to the next steps.

Note: When connecting a large number of application nodes, it is advisable to spread out the load of the connect operations by introducing a random delay across all the nodes. Otherwise the EF600 storage system may experience some timeouts, and those connect operations would need to be repeated to ensure that all the expected paths are visible on all application nodes.

Listing the NVMe-over-Fabrics devices (RHEL 7)

Note: This section applies to Linux hosts running RHEL 7. For information about RHEL 8, see "Listing the NVMe-over-Fabrics devices (RHEL 8)" on page 32.

The `nvme` command can be used to list the NVMe-over-Fabrics devices. Two different views are important: The generic `nvme list` command can be used to obtain basic information, and the vendor-specific `nvme netapp smdevices` command provides more detailed information.

In RHEL 7, the `nvme list` command will show one device for each path to each volume on the EF600 storage array. It does provide basic information about those devices, like the capacity and the type of formatting. Other than showing NetApp in the Model field, it does not show any details about the EF600 configuration. Figure 22 shows the output for a setup with eight EF600 EDR host ports and four mapped volumes. Each host port is manifested as an `nvme[0-7]` device, while each volume is manifested as an `n[1-4]` namespace. Some blank lines have been inserted to improve readability.

```
$ nvme list

Node          SN            Model            Namespace     Usage         Format       FW Rev
------------- ------------- ---------------- ------- ----------------- ------------ --------
/dev/nvme0n1  021910045419  NetApp E-Series  1       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme0n2  021910045419  NetApp E-Series  2       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme0n3  021910045419  NetApp E-Series  3       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme0n4  021910045419  NetApp E-Series  4       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700

/dev/nvme1n1  021910045419  NetApp E-Series  1       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme1n2  021910045419  NetApp E-Series  2       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700

...

/dev/nvme6n3  021910045419  NetApp E-Series  3       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme6n4  021910045419  NetApp E-Series  4       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700

/dev/nvme7n1  021910045419  NetApp E-Series  1       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme7n2  021910045419  NetApp E-Series  2       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme7n3  021910045419  NetApp E-Series  3       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
/dev/nvme7n4  021910045419  NetApp E-Series  4       2.66 TB / 2.66 TB  4 KiB + 0 B  98710700
```

Figure 22: nvme list output (RHEL 7).

This command is good for a first overview, and corresponds to the way local NVMe SSDs would be listed. But it does not show any information about the EF600 specifics. In particular, it is not evident from this output how the local nvmeXnY devices relate to the EF600 volumes and their paths through the EF600 host ports.

To get more information about the EF600 configuration information, a NetApp vendor plugin of the nvme command can be called. This provides the `smdevices` command, which will also list the EF600 volume name, the volume ID, the active controller for the volume, and the EF600's descriptive "Array Name" for each of the NVMe devices.

Figure 22 shows the output for the same setup as above, with eight EF600 host ports and four mapped volumes. Again, each EF600 host port (A1, A2, … , B4) is manifested as an `nvme[0-7]` device while each volume is manifested as an `n[1-4]` namespace (in the device name, as well as in the NSID field). Note that the long output lines are wrapping into two lines here, and some blank lines have been inserted to improve readability.

```
$ nvme netapp smdevices

/dev/nvme0n1, Array Name de0704-ef600, Volume Name de0704a01v1, NSID 1,
  Volume ID 0000127e5bacc864d039ea00003ef510, Controller A, Access State unknown, 2.66TB
/dev/nvme0n2, Array Name de0704-ef600, Volume Name de0704a01v2, NSID 2,
  Volume ID 00000eb35bacd37cd039ea00003ef1fd, Controller A, Access State unknown, 2.66TB
/dev/nvme0n3, Array Name de0704-ef600, Volume Name de0704a01v3, NSID 3,
  Volume ID 000012805bacc86cd039ea00003ef510, Controller A, Access State unknown, 2.66TB
/dev/nvme0n4, Array Name de0704-ef600, Volume Name de0704a01v4, NSID 4,
  Volume ID 00000eb55bacd37fd039ea00003ef1fd, Controller A, Access State unknown, 2.66TB

/dev/nvme1n1, Array Name de0704-ef600, Volume Name de0704a01v1, NSID 1,
  Volume ID 0000127e5bacc864d039ea00003ef510, Controller A, Access State unknown, 2.66TB
/dev/nvme1n2, Array Name de0704-ef600, Volume Name de0704a01v2, NSID 2,
  Volume ID 00000eb35bacd37cd039ea00003ef1fd, Controller A, Access State unknown, 2.66TB

...

/dev/nvme6n3, Array Name de0704-ef600, Volume Name de0704a01v3, NSID 3,
  Volume ID 000012805bacc86cd039ea00003ef510, Controller B, Access State unknown, 2.66TB
/dev/nvme6n4, Array Name de0704-ef600, Volume Name de0704a01v4, NSID 4,
  Volume ID 00000eb55bacd37fd039ea00003ef1fd, Controller B, Access State unknown, 2.66TB

/dev/nvme7n1, Array Name de0704-ef600, Volume Name de0704a01v1, NSID 1,
  Volume ID 0000127e5bacc864d039ea00003ef510, Controller B, Access State unknown, 2.66TB
/dev/nvme7n2, Array Name de0704-ef600, Volume Name de0704a01v2, NSID 2,
  Volume ID 00000eb35bacd37cd039ea00003ef1fd, Controller B, Access State unknown, 2.66TB
/dev/nvme7n3, Array Name de0704-ef600, Volume Name de0704a01v3, NSID 3,
  Volume ID 000012805bacc86cd039ea00003ef510, Controller B, Access State unknown, 2.66TB
/dev/nvme7n4, Array Name de0704-ef600, Volume Name de0704a01v4, NSID 4,
  Volume ID 00000eb55bacd37fd039ea00003ef1fd, Controller B, Access State unknown, 2.66TB
```

Figure 23: nvme netapp smdevices output (RHEL 7).

With the correct `/etc/multipath.conf` configuration file installed, it is now possible to list the DM-Multipath devices and their slave devices that represent the different paths through the EF600 host ports.


```
$ multipath -ll
eui.00000eb55bacd37fd039ea00003ef1fd dm-3 NVME,NetApp E-Series
size=2.4T features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| |- 4:0:4:0 nvme4n4 259:70  active ready running
| |- 5:0:4:0 nvme5n4 259:86  active ready running
| |- 6:0:4:0 nvme6n4 259:102 active ready running
| `- 7:0:4:0 nvme7n4 259:118 active ready running
`-+- policy='service-time 0' prio=10 status=enabled
  |- 0:0:4:0 nvme0n4 259:6   active ready running
  |- 1:0:4:0 nvme1n4 259:22  active ready running
  |- 2:0:4:0 nvme2n4 259:38  active ready running
  `- 3:0:4:0 nvme3n4 259:54  active ready running
eui.00000eb75bacd383d039ea00003ef1fd dm-5 NVME,NetApp E-Series
size=2.4T features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| |- 4:0:6:0 nvme4n6 259:74  active ready running
| |- 5:0:6:0 nvme5n6 259:90  active ready running
| |- 6:0:6:0 nvme6n6 259:106 active ready running
| `- 7:0:6:0 nvme7n6 259:122 active ready running
`-+- policy='service-time 0' prio=10 status=enabled
  |- 0:0:6:0 nvme0n6 259:10  active ready running
  |- 1:0:6:0 nvme1n6 259:26  active ready running
  |- 3:0:6:0 nvme3n6 259:58  active ready running
  `- 2:0:6:0 nvme2n6 259:42  active ready running
eui.000012825bacc86fd039ea00003ef510 dm-6 NVME,NetApp E-Series
size=2.4T features='1 queue_if_no_path' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| |- 4:0:5:0 nvme4n5 259:72  active ready running
| |- 5:0:5:0 nvme5n5 259:88  active ready running
| |- 6:0:5:0 nvme6n5 259:104 active ready running
| `- 7:0:5:0 nvme7n5 259:120 active ready running
`-+- policy='service-time 0' prio=10 status=enabled
  |- 0:0:5:0 nvme0n5 259:8   active ready running
  |- 1:0:5:0 nvme1n5 259:24  active ready running
  |- 3:0:5:0 nvme3n5 259:56  active ready running
  `- 2:0:5:0 nvme2n5 259:40  active ready running
eui.00000eb35bacd37cd039ea00003ef1fd dm-0 NVME,NetApp E-Series
size=2.4T features='1 queue_if_no_path' hwhandler='0' wp=rw

...
```

Figure 24: Output of multipath -ll (RHEL 7).

In the output of `multipath -ll` there should be one `dm-N` multipath device per mapped EF600 volume. On an EF600 with eight connected EDR InfiniBand ports, each of those multipathing devices should have eight paths to every volume, visible as eight different `nvmeXnY` devices for the NVMe namespace Y that corresponds to the `dm-N` multipath device.

The four paths that represent the connections to the active controller of the EF600 for that volume have a higher `prio=50`, while the four paths that represent the connections to the “backup” controller for that volume have a lower `prio=10`. Within a priority group, the DM-Multipathing driver usually selects a path in a round-robin manner.

These `dm-N` multipath devices will be used below to create the Spectrum Scale NSDs.

Listing the NVMe-over-Fabrics devices (RHEL 8)

Note: This section applies to Linux hosts running RHEL 8. For information about RHEL 7, see "Listing the NVMe-over-Fabrics devices (RHEL 8)" on page 32.

The `nvme` command can be used to list the NVMe-over-Fabrics devices. Two different views are important: The generic `nvme list` command can be used to obtain basic information, and the `nvme netapp smdevices` command provides more detailed information.

In RHEL 8, the `nvme list` command will show one device for each volume on the EF600 storage array – it does *not* show individual device paths like on RHEL 7. The output provides some basic information about those devices, like the capacity and the type of formatting. Other than showing NetApp in the Model field, it does not show any details about the EF600 configuration. Figure 25 shows the output for an EF600 with four mapped volumes. Each volume is manifested as an `n[1-4]` namespace.

```
$ nvme list

Node          SN            Model             Namespace    Usage                 Format          FW Rev
------------  ------------  ---------------   -----  -----------------  ------------  --------
/dev/nvme0n1  021910045419  NetApp E-Series   1      5.36 TB / 5.36 TB  4 KiB +  0 B  98711200
/dev/nvme0n2  021910045419  NetApp E-Series   2      5.36 TB / 5.36 TB  4 KiB +  0 B  98711200
/dev/nvme0n3  021910045419  NetApp E-Series   3      5.36 TB / 5.36 TB  4 KiB +  0 B  98711200
/dev/nvme0n4  021910045419  NetApp E-Series   4      5.36 TB / 5.36 TB  4 KiB +  0 B  98711200
```

Figure 25: nvme list output (RHEL 8).

To get more information about the EF600 configuration information, a NetApp vendor plugin of the nvme command can be called. This provides the `smdevices` command, which will also list the EF600 volume name, the volume ID, the active controller for the volume, and the EF600's descriptive "Array Name" for each of the NVMe devices.

Figure 26 shows the output for the same setup as above. Again, in RHEL 8 the EF600 host ports are *not* shown, but the Controller A/B ownership *is* show. As in the generic list command, each volume is manifested as an `n[1-4]` namespace (in the device name, as well as in the NSID field). Note that the long output lines are wrapping into two lines.

```
$ nvme netapp smdevices

/dev/nvme0n1, Array Name de0704-ef600, Volume Name de0704a01v1, NSID 1,
  Volume ID 00004edf5c0b4a01d039ea00003ef510, Controller A, Access State unknown, 5.36TB

/dev/nvme0n2, Array Name de0704-ef600, Volume Name de0704a01v2, NSID 2,
  Volume ID 00004ee15c0b4a07d039ea00003ef510, Controller B, Access State unknown, 5.36TB

/dev/nvme0n3, Array Name de0704-ef600, Volume Name de0704a02v1, NSID 3,
  Volume ID 00004de15c0b5539d039ea00003ef1fd, Controller A, Access State unknown, 5.36TB

/dev/nvme0n4, Array Name de0704-ef600, Volume Name de0704a02v2, NSID 4,
  Volume ID 00004de35c0b553bd039ea00003ef1fd, Controller B, Access State unknown, 5.36TB
```

Figure 26: nvme netapp smdevices output (RHEL 8).

With the correct `/etc/multipath.conf` configuration file installed, we can now list the NVMe-over-Fabrics devices and their slave devices that represent the paths through the EF600 host ports:

```
$ multipath -ll
eui.00004edf5c0b4a01d039ea00003ef510 [nvme]:nvme0n1 NVMe,NetApp E-Series,98711200
size=10468982784 features='n/a' hwhandler='ANA' wp=rw
|-+- policy='n/a' prio=50 status=optimized
| `- 0:0:1 nvme0c0n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:1:1 nvme0c1n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:2:1 nvme0c2n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:3:1 nvme0c3n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:4:1 nvme0c4n1 0:0 n/a non-optimized live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:5:1 nvme0c5n1 0:0 n/a non-optimized live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:6:1 nvme0c6n1 0:0 n/a non-optimized live
`-+- policy='n/a' prio=10 status=non-optimized
  `- 0:7:1 nvme0c7n1 0:0 n/a non-optimized live
eui.00004ee15c0b4a07d039ea00003ef510 [nvme]:nvme0n2 NVMe,NetApp E-Series,98711200
size=10468982784 features='n/a' hwhandler='ANA' wp=rw
|-+- policy='n/a' prio=50 status=optimized
| `- 0:0:1 nvme0c0n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:1:1 nvme0c1n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:2:1 nvme0c2n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=50 status=optimized
| `- 0:3:1 nvme0c3n1 0:0 n/a optimized     live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:4:1 nvme0c4n1 0:0 n/a non-optimized live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:5:1 nvme0c5n1 0:0 n/a non-optimized live
|-+- policy='n/a' prio=10 status=non-optimized
| `- 0:6:1 nvme0c6n1 0:0 n/a non-optimized live
`-+- policy='n/a' prio=10 status=non-optimized
  `- 0:7:1 nvme0c7n1 0:0 n/a non-optimized live
eui.00004de15c0b5539d039ea00003ef1fd [nvme]:nvme0n3 NVMe,NetApp E-Series,98711200
...
```

Figure 27: Output of multipath -ll (RHEL 8).

In RHEL 8, the output of `multipath -ll` shows one multipath device per mapped EF600 volume, identified as an nvme0nN device with its namespace ID N just like in the nvme command output. On an EF600 with eight connected EDR InfiniBand ports, each of those devices should have eight paths X to every volume, visible as eight different nvme0cXnY devices.

Note: Figure 27 reveals a software bug, as the slave paths to *all* 4 namespaces are shown as "n1".

The four paths that represent the connections to the "active" controller of the EF600 for that volume have a higher `prio=50` and are shown with `status=optimized`. The four paths that represent the connections to the "backup" controller for that volume have a lower `prio=10` and `status=non-optimized`. Within a priority group, the multipathing layer selects a path in a round-robin manner.

These nvme0nN devices will be used below to create the Spectrum Scale NSDs.

Disconnecting the NVMe-over-Fabrics devices

Should it be necessary to disconnect the NVMe-over-Fabrics devices, for example because they are only connected for the duration of a Slurm batch job run, the `nvme disconnect-all` command can be used. Make sure that all higher-level applications like the Spectrum Scale filesystem that may be using the NVMe-over-Fabrics devices have been quiesced before disconnecting.

```
$ # Quiesce the application before disconnecting:
$ FS="ef600_mlx"
$ mmumount $FS
$ mmlsmount $FS -L

$ multipath -ll
$ nvme disconnect-all
$ multipath -ll
```

Figure 28: nvme disconnect-all for the NVMe-over-Fabrics volumes.

Should there be more NVMe-over-Fabrics subsystems than just a single EF600, and the goal is to selectively disconnect from only some of those subsystems, it is also possible to disconnect only specific devices. See `man nvme-disconnect` for details.

Spectrum Scale "SAN Mode" Configuration

When using Spectrum Scale "SAN Mode" to make all Spectrum Scale Network Shared Disks (NSDs) visible on all Spectrum Scale nodes, most Spectrum Scale installation and configuration steps are similar to the NSD Client/Server model, and should already be familiar to Spectrum Scale administrators. For "SAN Mode" there are a few important differences to Spectrum Scale environments that are based on the NSD Client/Server model, these are described in this section.

No Spectrum Scale Multi-Cluster Support

One high-level difference between the Spectrum Scale SAN Mode and the NSD Client/Server Model is that SAN Mode does *not* support Spectrum Scale Multi-Cluster. The ability to mount a Spectrum Scale filesystem in multiple Spectrum Scale "client" clusters relies on the usage of the NSD Client/Server model, where the "owning" cluster provides access to "client" clusters through its NSD servers. In SAN Mode, only a single Spectrum Scale cluster is allowed to *directly* access the NVMe-over-Fabrics volumes.

Attention: While it is technically possible to make the NVMe-over-Fabrics volumes visible to nodes that reside on the same InfiniBand fabric, but belong to two different Spectrum Scale clusters, trying to access those volumes as "directly attached" NSDs from both clusters is not supported and must be avoided. Such concurrent access will cause the corruption of the NSDs.

Note: It *is* possible to define dedicated NSD Servers for all the NVMe-over-Fabric volumes, and then use those NSD severs to provide *indirect* access to those volumes to other Spectrum Scale clusters, just like with other filesystems that use the NSD Client/Server model.

Basic Spectrum Scale Cluster Setup

Before creating NSDs on the NVMe-over-Fabrics volumes, the application nodes need to be members of a Spectrum Scale cluster, which can be set up like any other Spectrum Scale "client" cluster. Use the `mmcrcluster`, `mmchlicense`, `mmaddnode`, `mmchconfig` and `mmstartup` commands like in other Spectrum Scale environments.

Note: In a cluster with mostly stateless compute nodes, it is important to dedicate one or more stateful nodes as manager nodes. It is recommended to always perform the NSD and filesystem configuration on those stateful nodes. As there are no dedicated NSD servers, there is no "natural" node to perform the NSD and filesystem configuration. It helps with the overall operation and manageability of the Spectrum Scale environment if the nodes on which those tasks are performed are clearly identified.

Figure 29 shows the cluster configuration of the Infiniband-based Spectrum Scale "client cluster" in Lenovo's HPC Benchmarking Center. Note that there are no tunables related to NSD servers (like `nsdbufspace`, `nsdMinWorkerThreads`, or `nsdMaxWorkerThreads`), as the "SAN Mode" does not use the NSD Client/Server layer. (There are some NSD client tunables, because this cluster is also accessing other Spectrum Scale file systems through NSD servers that are located in other Spectrum Scale clusters.)


```
$ mmlsconfig -Y | sort
mmlsconfig::0:1:::adminMode:central::
mmlsconfig::0:1:::autoload:yes::
mmlsconfig::0:1:::avoidDirFragments:yes::
mmlsconfig::0:1:::cipherList:AUTHONLY::
mmlsconfig::0:1:::clusterId:11407465450739028732::
mmlsconfig::0:1:::clusterName:compute-mlx.hpc.eu.lenovo.com::
mmlsconfig::0:1:::dioSmallSeqWriteBatching:yes::
mmlsconfig::0:1:::dmapiFileHandleSize:32::
mmlsconfig::0:1:::ignorePrefetchLUNCount:yes::
mmlsconfig::0:1:::ioHistorySize:4k::
mmlsconfig::0:1:::maxblocksize:16m::
mmlsconfig::0:1:::maxBufferDescs:2m::
mmlsconfig::0:1:::maxFilesToCache:1m::
mmlsconfig::0:1:::maxInodeDeallocHistory:0::
mmlsconfig::0:1:::maxMBpS:20000::
mmlsconfig::0:1:::maxStatCache:1m::
mmlsconfig::0:1:::minReleaseLevel:5.1.0.0::
mmlsconfig::0:1:::nsdClientCksumTypeLocal:ck64::
mmlsconfig::0:1:::nsdClientCksumTypeRemote:ck64::
mmlsconfig::0:1:::numaMemoryInterleave:yes::
mmlsconfig::0:1:::pagepool:4g::
mmlsconfig::0:1:::prefetchPct:50::
mmlsconfig::0:1:::scatterBufferSize:256K::
mmlsconfig::0:1:::verbsPorts:mlx5_0/1/1::
mmlsconfig::0:1:::verbsRdma:enable::
mmlsconfig::0:1:::verbsRdmaMinBytes:32k::
mmlsconfig::0:1:::verbsRdmaSend:yes::
mmlsconfig::0:1:::workerThreads:512::
mmlsconfig::HEADER:version:reserved:reserved:configParameter:value:nodeList:
```

Figure 29: Spectrum Scale cluster configuration settings.

Note: If the EF600 in "SAN Mode" is the only Spectrum Scale storage that is used in this Spectrum Scale cluster, the RDMA setup with the `verbs*` tunables is not strictly necessary, as all data access to the NSDs will be performed through the NVMe-over-Fabrics layer. However, we recommend to set up the Spectrum Scale RDMA configuration for InfiniBand just like with every other Spectrum Scale cluster.

Creating NSDs for the EF600 Volumes

To format the EF600 volumes as Spectrum Scale NSDs, a stanza file needs to be created that is then used as input to the `mmcrnsd` command. In these `%nsd` stanzas, we recommend to use the names of the EF600 volumes (that have been set as the `userLabel` in the `create volume` commands in Figure 11 and Figure 13) as the NSD `name`. This ensures that the relationship of Spectrum Scale NSDs to EF600 volumes is explicitly visible in Spectrum Scale.

The `device` entries for the `%nsd` stanzas depend on the operating system version. For RHEL 7 they should be the *DM-Multipath* device names like `/dev/dm-3`. For RHEL 8 where the multipathing is managed by the NVMe-over-Fabrics layer, the device names have the form `/dev/nvme0n1`.

The default Spectrum Scale `mmdevdiscover` mechanism automatically detects the DM-Multipath devices that are used with RHEL 7. For RHEL 8 Linux hosts, we need to manually install a `/var/mmfs/etc/nsddiscover` script to discover the NVMe-over-Fabrics multipathing devices. Figure 30 shows a simple example script to achieve this:

```
cat /var/mmfs/etc/nsddevices
#!/bin/ksh

lsblk | grep ^nvme | cut -d' ' -f1 | while read DEV
do
  echo "$DEV generic"
done

return 0 # To continue with the GPFS disk discovery steps, return 1
```

Figure 30: nsddevices script to discover NVMe-over-Fabrics devices with RHEL 8.

Mapping the device names to EF600 volume names is a two-step process, using the output of `multipath -ll` as well as the output of the `nvme netapp smdevices` command. Figure 31 shows the RHEL 7 based helper script to create the NSD stanzas with this method.

```
$ cat ef600-nsdstanza-rhel7.sh
#!/bin/bash

TRIM="no"  # in the future, set this to "nvme" to enable
SM_TABLE="/tmp/smdevices.$$"

nvme netapp smdevices | cut -d"," -f 1,3 | cut -d" " -f1,4 \
  | sed "s/,//g" | while read DEV NSD
do
  echo "$DEV $NSD"
done > $SM_TABLE

FOUND_PATH=0
multipath -ll | while read LINE
do
  if [[ "$LINE" =~ "NetApp E-Series" ]]; then
    DM_DEV=`echo $LINE | cut -d" " –f2`
    FOUND_PATH=0
  fi
  if [[ "$LINE" =~ "  nvme" && $FOUND_PATH -eq 0 ]]; then
    NVME_DEV=`echo $LINE|cut -d":" -f4|cut -d" " -f2`
    FOUND_PATH=1
    NSD_NAME=`grep $NVME_DEV $SM_TABLE | cut -d" " -f2`
    echo "%nsd: device=$DM_DEV nsd=$NSD_NAME usage=dataAndMetadata
              failureGroup=704 pool=system thinDiskType=$TRIM"
  fi
done

rm $SM_TABLE
```

Figure 31: Helper script to create NSD stanzas for the EF600 volumes (RHEL 7).

The corresponding helper script for RHEL 8 is shown in Figure 32.

```
$ cat ef600-nsdstanza-rhel8.sh
#!/bin/bash

TRIM="no"  # in the future, set this to "nvme" to enable
SM_TABLE="/tmp/smdevices.$$"

nvme netapp smdevices | cut -d"," -f 1,3 | cut -d" " -f1,4 \
  | sed "s/,//g" | while read DEV NSD
do
  echo "$DEV $NSD"
done > $SM_TABLE

FOUND_PATH=0
multipath -ll | while read LINE
do
  if [[ "$LINE" =~ "NetApp E-Series" ]]; then
    DM_DEV=`echo $LINE | cut -d" " -f2 | cut -d":" -f2`
    FOUND_PATH=0
  fi
  if [[ "$LINE" =~ " nvme" && $FOUND_PATH -eq 0 ]]; then
    NVME_DEV=$DM_DEV
    FOUND_PATH=1
    NSD_NAME=`grep $NVME_DEV $SM_TABLE | cut -d" " -f2`
    echo "%nsd: device=$DM_DEV nsd=$NSD_NAME usage=dataAndMetadata
             failureGroup=704 pool=system thinDiskType=$TRIM"
  fi
done

rm $SM_TABLE
```

Figure 32: Helper script to create NSD stanzas for the EF600 volumes (RHEL 8).

Note: We are already preparing the setup of the NSDs for *TRIM support,* by specifying the `thinDiskType` attribute for the NSDs. However, the 11.70.1 release of the NetApp EF600 firmware does *not* yet support TRIM, so the Spectrum Scale NSDs' `thinDiskType` must be set to "no".

NetApp may support TRIM on the EF600 with a later EF600 firmware release, in which case the `thinDiskType` attribute can be set to "`nvme`". TRIM will be discussed further in "Managing NAND Flash: Over-Provisioning and TRIM Support" on page 40.

Using these helper scripts, the NSDs can then be created (using the RHEL8 example; the RHEL7 case is exactly the same except for the device names).


```
# ./ef600-nsdstanza-rhel8.sh | tee ef600_mlx.stanza
%nsd: device=nvme0n1 nsd=de0704a01v1 usage=dataAndMetadata
  failureGroup=704 pool=system thinDiskType=no
%nsd: device=nvme0n2 nsd=de0704a01v2 usage=dataAndMetadata
  failureGroup=704 pool=system thinDiskType=no
%nsd: device=nvme0n3 nsd=de0704a02v1 usage=dataAndMetadata
  failureGroup=704 pool=system thinDiskType=no
%nsd: device=nvme0n4 nsd=de0704a02v2 usage=dataAndMetadata
  failureGroup=704 pool=system thinDiskType=no

$ mmcrnsd –F ef600_mlx.stanza
mmcrnsd: Processing disk nvme0n4
mmcrnsd: Processing disk nvme0n3
mmcrnsd: Processing disk nvme0n2
mmcrnsd: Processing disk nvme0n1
mmcrnsd: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.

$ mmlsnsd
 File system   Disk name       NSD servers
---------------------------------------------------------------
 (free disk)   de0704a01v1     (directly attached)
 (free disk)   de0704a01v2     (directly attached)
 (free disk)   de0704a02v1     (directly attached)
 (free disk)   de0704a02v2     (directly attached)

$ mmlsnsd –m
Disk name    NSD volume ID     Device     Node name or Class  Remarks
-------------------------------------------------------------------------
 de0704a01v1  AC1E030460630B76  /dev/nvme0n1  mgt2304
 de0704a01v2  AC1E030460630B78  /dev/nvme0n2  mgt2304
 de0704a02v1  AC1E030460630B79  /dev/nvme0n3  mgt2304
 de0704a02v2  AC1E030460630B77  /dev/nvme0n4  mgt2304
```

Figure 33: Creating and listing directly attached NSDs (RHEL 8).

Creating the Spectrum Scale Filesystem

After the “directly attached” NSDs have been created, a Spectrum Scale filesystem can be created on these NSDs in the same way as in the NSD Client/server model, using the `mmcrfs` command. Of course, all of the Spectrum Scale nodes that will mount the file systems need to have performed the NVMe-over-Fabrics connect operation before running `mmmount`, so they can access the volumes. This also applies to all Spectrum Scale manager nodes, which may perform internal mounts at any time.

```
$ mmcrfs ef600_mlx -F ef600_mlx.stanza -T /gpfs/ef600_mlx -B 1m \
    -E no -S relatime -K whenpossible \
    -m 1 -M 2 -r 1 -R 2   -n 137   -A no \
    --inode-limit 100000000:50000000
```

Figure 34: Creating the Spectrum Scale filesystem with mmcrfs.

Managing NAND Flash: Over-Provisioning and TRIM Support

All storage media that are based on NAND flash technology are facing two fundamental challenges:

1. It is not possible to overwrite individual sectors on a NAND flash device “in-place”. In order to re-write NAND storage, it needs to be erased in large chunks that are typically Megabyte-sized. Clearing those large “erasure blocks” is a very slow operation compared to the normal read and write operations of the SSDs.
2. NAND flash media have a limited endurance: The NAND flash cells can only be written a finite number of times. After that limit has been reached, the device will change into read-only mode and data has to be moved to a fresh device.

Both of these challenges are rooted in the physical properties of the NAND flash storage technology. The controllers on the NAND flash SSDs, the storage subsystem like the EF600 that uses such SSDs, as well as the application layer that is accessing the volumes provisioned on NAND flash SSD storage, can all employ a number of technologies to minimize the impact of these two NAND flash properties on the performance and endurance of the storage solution. Two techniques that are universally used with NAND flash media are:

1. **Over-provisioning**. In order to allow the SSD to perform normally even in the presence of a lot of write operations that require sectors to be copied to freshly erased areas of the NAND media, all NAND flash devices perform some level of over-provisioning. The actual internal storage capacity of the SSD is bigger than the advertised capacity, and the controller on the SSD can use that extra storage capacity to perform the erasure operations in the background. Note that MLC and TLC flash SSDs are often using the same physical media, but the SSDs qualified as MLC drives are reserving a larger amount of overprovisioning capacity. This enables the MLC drives to guarantee a larger Drive Write per Day (DWD) endurance over a specified lifetime (5 years for Lenovo SSDs) than the corresponding TLC drives.
2. Background **garbage collection**: Because the clearing of a large erasure block is much slower than a sector write operation (sometimes up to 10x slower), all SSD controllers perform background garbage collection in order to hide the latency of those erasure operations. If most of the sectors in an erasure block could already be erased (for example because new versions of those sectors have already been written somewhere else), but some sectors are still containing valid data, those sectors are copied into a new location to allow the SSD controller to perform the erase operation on the “almost free” erasure block. For many workloads, this allows the controller to keep up with the write operations. But for heavy irregular writes the SSD may not be able to sustain its write performance, if it needs to allocate fresh sectors quicker than the controller can clear out invalidated areas on the NAND flash media.

One negative effect of the background garbage collection is known as *write amplification*: The additional copy operations within the SSD are adding to the wear-out of the NAND flash cells, reducing the remaining endurance of the media. So even with a large over-provisioning capacity, there is always a trade-off between high endurance (less aggressive garbage collection, risking a performance drop under heavy write loads) and high performance (more aggressive garbage collection, causing lower remaining endurance).

EF600 Optimization Capacity

The EF600 storage array includes a feature called *Optimization Capacity* that allows the storage administrator to set aside a percentage of the capacity of the NVMe SSDs as an additional over-provisioning capacity (on top of what the NVMe SSDs are already reserving internally). This is especially useful to deploy TLC flash media in enterprise environments: Higher endurance MLC flash media have a higher cost per TB than TLC drives, and are "locking up" a higher percentage of over-provisioning capacity within the SSD drives than TLC drives. That capacity will never be exposed to the application. With the EF600 optimization capacity feature, it is possible to deploy TLC flash media with a much better cost per TB, and the storage administrator can then selectively allocate or reserve a percentage of the TLC capacity (depending on the expected workloads). The NetApp Technical Report TR-4800 contains recommendations on the over-provisioning capacity for various NVMe SSD capacities.

Note that the NetApp GUI will automatically determine and apply an optimization capacity when a DDP pool or a RAID volume is created in the GUI. The CLI does not configure optimization capacity automatically. The original REST API also does not automatically reserve optimization capacity, but newer versions of the REST API can be used to model the behavior of the GUI.

Note: The `SMcli` examples in this document do not use the optimization capacity feature, but are provisioning volumes that are not using 100% of the available capacity of the DDP pool or volume group. This is not fully equivalent: The available but unallocated capacity will *not* be TRIMmed (see below), whereas the optimization capacity *will* be TRIMmed by the EF600 controller at the time the DDP pool or volume group is created.

Spectrum Scale and EF600 TRIM Support

The controller on a NAND flash SSD has only very limited information about the usage of the data sectors of the SSD, which makes it hard to optimize the garbage collection. The only way that the controller can know that the data in a sector is no longer in use is if it has received a write operation for that sector, and has thus copied the sector from its original location to a freshly erased area of the NAND flash media. But the controller is not aware of any other changes on the application level. For example, if a file has been deleted on the file system layer, this invalidates all data blocks that belonged to that file. If a new file system is been created on a number of volumes, this invalidates all data blocks on all volumes that are used by the file system.

It is the purpose of the ATA "TRIM" command (or SCSI "unmap" command, or NVMe "deallocate" command) to communicate such information from the application level to the SSD controller. Those commands can notify the SSD controller of ranges of LBA addresses that are no longer valid. The controller's garbage collection algorithm then no longer needs to copy those sectors to fresh areas of the NAND flash media before performing an erase operation. This is a big help to avoid excessive write amplification, and to maintain high performance.

Spectrum Scale has introduced initial support for the ATA TRIM command (or SCSI "unmap" command, or NVMe "deallocate" command) with the Spectrum Scale 5.0.4 modification level. The Spectrum Scale TRIM support is based on two components:

- The Spectrum Scale NSDs need to be created with the `thinDiskType=nvme` attribute (for NVMe volumes). This advertises the underlying disks' (or volumes') capability to handle the TRIM (or unmap, or deallocate) operation. The helper script in Figure 31 on page 37 explicitly sets this attribute (the default is "no"). Figure 33 shows the resulting `%nsd` stanzas.
- The Spectrum Scale `mmreclaimspace` command can be used for a filesystem to send TRIM commands to the NSDs, if at least one NSD in the filesystem support it. Note that in Spectrum Scale Version 5.0 and 5.1.0, no automatic TRIM operations are performed by Spectrum Scale: The `mmreclaimspace` command needs to be explicitly called by the storage administrator.

IBM has initially tested TRIM support with Intel-branded internal NVMe drives, and has also enabled TRIM support for Spectrum Scale RAID (which is used with IBM ESS, Lenovo DSS-G, and the Spectrum Scale Erasure Code Edition).

Supporting the TRIM functionality with hardware RAID controllers is more involved than the support of individual NVMe drives: The RAID controller also needs to correctly interpret the TRIM commands, and has to translate them to TRIM operations for the corresponding RAID segments on the member disks of the DDP pool or volume group. The NetApp EF600 does not yet support TRIM in the 11.70.1 firmware level. We strongly recommend to update to the latest available firmware level, because NetApp may enable TRIM support in a later firmware release. The NetApp Technical Report TR-4800 contains more details on the TRIM features in the EF600, as well as a list of current limitations.

Figure 35 shows how to run the `mmreclaimspace` command on a filesystem with TRIM-enabled NSDs. (For this test, a development snapshot of EF600 firmware was used that does support TRIM.)

The `mmreclaimspace` command can be run when the filesystem is mounted, or when the filesystem is not mounted. If the filesystem contains multiple storage pools, it can also be invoked on a single storage pool by using the `-P` option. The `--reclaim-threshold` argument is mandatory: It specifies the minimum threshold of reclaimable space that needs to exist within a Spectrum Scale allocation segment before space in that segment will be reclaimed. To reclaim all space, use the value "0" as shown in Figure 35. As this is an I/O intensive and potentially long-running operation, using a higher value reduces the load on the cluster during the reclaim operation. When the command completes, it reports NSD usage statistics similar to the output of `mmdf`, with an additional column that reports the reclaimed space in kiB.


```
$ mmreclaimspace -?
  mmreclaimspace Device [-Y] [-P PoolName] [--qos QosClass]
    --reclaim-threshold percentage | --emergency-reclaim

$ mmreclaimspace ef600_mlx --reclaim-threshold 0

disk            disk size  failure holds    holds       free in KB     free in KB    reclaimed in KB
name                in KB    group metadata data    in full blocks   in fragments       in subblocks
------------ ------------ -------- -------- ----- ----------------- -------------- -----------------
Disks in storage pool: system (Maximum disk size allowed is 39.47 TB)
de0704a01v1    5234491392      704 Yes      Yes   5182926848 ( 99%)    11832 ( 0%)  5182938680 (99%)
de0704a02v2    5234491392      704 Yes      Yes   5182918656 ( 99%)    11896 ( 0%)  5182930552 (99%)
de0704a02v1    5234491392      704 Yes      Yes   5182918656 ( 99%)     8056 ( 0%)  5182926712 (99%)
de0704a01v2    5234491392      704 Yes      Yes   5183062016 ( 99%)    11032 ( 0%)  5183073048 (99%)
             -------------                        ------------------ -------------- -----------------
(pool total)  20937965568                         20731826176 ( 99%)    42816 ( 0%) 20731868992 (99%)

             =============                        ================== ============== =================
(total)       20937965568                         20731826176 ( 99%)    42816 ( 0%) 20731868992 (99%)
```

Figure 35: Running the mmreclaimspace command.

More information on the Spectrum Scale support for TRIM can be found in „IBM Spectrum Scale with data reduction storage devices” in the *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*. See also „Chapter 18. File system format changes between versions of IBM Spectrum Scale“ in the *IBM Spectrum Scale: Administration Guide*.

Capacity Sizing

Sizing the capacity of an EF600 based Spectrum Scale solution is a two-step process. First, the configuration options for a single EF600 storage subsystem need to be evaluated to determine if the usable capacity requirements can be fulfilled with a single EF600. If the desired usable capacity is bigger than what can be achieved with a single EF600 storage system, then a scale-out solution with multiple EF600 storage systems can be used. Ideally, all EF600 storage systems in a scale-out solution should be configured identically. The total usable capacity is then scaling linearly with the number of EF600 systems in the solution.

To derive the usable capacity of an EF600 configuration from its raw capacity, the following factors need to be considered:

- Individual NVMe disk sizes range from 1.92 TB to 15.36 TB, and a maximum of 24 NVMe disks are supported in an EF600 storage system. All NVMe disks in a pool or volume group should have the same size.
- On each physical disk, a small amount of capacity is used by the storage subsystem to store internal configuration data. This capacity has to be subtracted from the raw disk capacity.
- When translating raw capacity to usable capacity, the main overhead is the RAID overhead of the chosen RAID level (e.g. RAID1, RAID6, or DDP).
- Spare capacity can be provisioned to provide capacity for automatic rebuilds.
- Some capacity can or should be set aside as *optimisation capacity*, to sustain the write performance and endurance of the NAND flash media.

These factors are discussed in more detail below. Putting everything together, Table 1 summarizes typical capacity sizing scenarios for the EF600 when using RAID6 volumes, and Table 2 shows the sizing for DDP volumes.

Table 1: EF600 Usable Capacity for RAID6 volumes.

Disk Capacity [TB]	Disk Quantity	Gross Capacity [TB]	[TiB]	Reserved for Config Data [GB/disk]	RAID Protection Scheme	Usable [%]	Spare Capacity [#disks]	Optimisation Capacity [%]	Usable Capacity [TB]	[TiB]
1,92	10	19,2	17,5	5,5	RAID6	80%	0	28%	11,0	10,0
1,92	20	38,4	34,9	5,5	RAID6	80%	0	28%	22,1	20,1
3,84	10	38,4	34,9	5,5	RAID6	80%	0	14%	26,4	24,0
3,84	20	76,8	69,8	5,5	RAID6	80%	0	14%	52,8	48,0
7,68	10	76,8	69,8	5,5	RAID6	80%	0	10%	55,3	50,3
7,68	20	153,6	139,7	5,5	RAID6	80%	0	10%	110,5	100,5
15,36	10	153,6	139,7	5,5	RAID6	80%	0	4%	117,9	107,2
15,36	20	307,2	279,4	5,5	RAID6	80%	0	4%	235,8	214,5

Note: Table 1 does not include any hot spare drives. If desired, one or two can be added.

Table 2: EF600 Usable Capacity for DDP volumes.

Disk Capacity [TB]	Disk Quantity	Gross Capacity [TB]	[TiB]	Reserved for Config Data [GB/disk]	RAID Protection Scheme	Usable [%]	Spare Capacity [#disks]	Optimisation Capacity [%]	Usable Capacity [TB]	[TiB]
1,92	12	23,0	21,0	5,5	DDP	80%	2	28%	11,0	10,0
1,92	16	30,7	27,9	5,5	DDP	80%	2	28%	15,4	14,0
1,92	20	38,4	34,9	5,5	DDP	80%	2	28%	19,8	18,1
1,92	24	46,1	41,9	5,5	DDP	80%	2	28%	24,3	22,1
3,84	12	46,1	41,9	5,5	DDP	80%	2	14%	26,4	24,0
3,84	16	61,4	55,9	5,5	DDP	80%	2	14%	36,9	33,6
3,84	20	76,8	69,8	5,5	DDP	80%	2	14%	47,5	43,2
3,84	24	92,2	83,8	5,5	DDP	80%	2	14%	58,0	52,8
7,68	12	92,2	83,8	5,5	DDP	80%	2	10%	55,3	50,3
7,68	16	122,9	111,8	5,5	DDP	80%	2	10%	77,4	70,4
7,68	20	153,6	139,7	5,5	DDP	80%	2	10%	99,5	90,5
7,68	24	184,3	167,6	5,5	DDP	80%	2	10%	121,6	110,6
15,36	12	184,3	167,6	5,5	DDP	80%	2	4%	117,9	107,2
15,36	16	245,8	223,5	5,5	DDP	80%	2	4%	165,1	150,1
15,36	20	307,2	279,4	5,5	DDP	80%	2	4%	212,3	193,0
15,36	24	368,6	335,3	5,5	DDP	80%	2	4%	259,4	235,9

Note: The usable capacity in Table 1 and Table 2 is the "LUN" capacity that is available for Spectrum Scale NSDs. In the typical deployment scenario with RAID6 or DDP volumes, Spectrum Scale data and metadata are usually shared on all NSDs. In this case, the usable capacity from the EF600 "LUN" perspective includes the capacity that will be consumed by the Spectrum Scale metadata. Please refer to the Spectrum Scale documentation, and the metadata presentations on the Spectrum Scale User Group website, for guidance regarding Spectrum Scale metadata sizing. The usable "filesystem" capacity (as shown by commands like `df`) will be smaller than the "LUN" usable capacity, and the exact amount of the difference depends mainly on the metadata needs (block allocation maps, inode maps, additional metadata space for replication and snapshots, extended attributes, etc.).

Per-Disk Configuration Data

Each physical disk in an EF600 that is part of a volumeGroup or diskPool will be used to store internal configuration data. This consumes about 5.5 GB on each of the physical disks.

RAID Overhead

The RAID overhead of the selected data protection scheme is usually the biggest contributor to the difference between gross capacity and usable capacity. The most common cases are:

- **DDP**: A DDP `diskPool` uses a declustered RAID6 (8+2P) scheme, which results in 80% usable space *including* spare space. (see below for a discussion of spare capacity).

- **RAID6**: A classical `volumeGroup` with RAID6 (8+2P) volumes also results in 80% usable space. If configured, spare capacity will be in dedicated *additional* drives, so those choices will not affect the usable capacity.
- **RAID1**: A classical `volumeGroup` with RAID1 (mirroring) volumes has 50% usable space. With Spectrum Scale, RAID1 would only be used for dedicated metadata NSDs. If configured, spare capacity will be in dedicated *additional* drives, so those choices will not affect the usable capacity.
- **RAID10** (mirroring and striping) is *not* recommended with Spectrum Scale. It is preferable to perform only the mirroring (RAID1) in the storage subsystem, and let the file system layer control the striping across multiple RAID1 NSDs.

All of these data protection schemes incur a small overhead for internal configuration data. This is ignored in the above calculations.

Hot Spare Capacity

Spare capacity is managed differently for DDP and non-DDP volumes.

Hot spare drives for classical volume groups

For a classical `volumeGroup` with RAID6 (8+2P) or RAID1 (mirrored) volumes, dedicated **`hotSpare` drives** can be defined. These drives will be idle in the error-free state, and will only become active in the event of a drive failure.

- When a drive failure happens in a RAID array, a hot-spare will be used instead of the failed drive, and **rebuild** will start automatically.
- Once the failed drive has been replaced, a **copy-back** operation from the hot-spare to the replacement drive is needed.

As a best practice for RAID6 (8+2P) volumes, we recommend to **not** define hot spare drives but instead have one or two replacement drives on site as **cold** spares. This avoids the additional copy-back operation. Because the RAID6 protection is 2-fault-tolerant, an (unlikely) second drive failure in the same RAID6 `volumeGroup` before the replacement of the first failed drive could still be tolerated.

For RAID1 (mirrored) volumes with two drives, a second drive failure before the replacement of the first failed drive would lead to data loss of the volumes on that RAID1 array. So it is prudent to include hot spare drives for RAID1 volumes. There is one important exception: If there is an additional replication on the software level (like Spectrum Scale filesystem-level replication), then it is not strictly necessary to use hot spare drives for RAID1. As in the RAID6 case, onsite **cold** spares could be used to ensure a timely replacement of a failed drive. In case of an (unlikely) second failure in the same RAID1 `volumeGroup`, the Spectrum Scale level would still ensure the filesystem integrity.

Preservation Capacity for DDP

For volumes in a DDP `diskPool`, **preservation capacity** is automatically included in the declustered RAID setup. This spare capacity is distributed across all physical drives in the `diskPool`.

- Preservation capacity is measured in an equivalent number of disks. The defaults depend on the `diskPool` size. For the minimum DDP `diskPool` size of 11 drives, the equivalent of one

disk will be set aside as preservation capacity. For DDP pools with 12 to 24 drives, a capacity equivalent to two disks will be used.

- The default can be changed with the `SMcli` command by specifying the desired capacity in equivalent number of drives: `set diskPool` *`diskPoolName`* `reservedDriveCount=`*`N`*

We recommend to keep the defaults for DDP preservation capacity.

Optimisation Capacity for NAND Flash Overprovisioning

The GUI of the EF600 allows the storage administrator to set aside some *optimisation capacity* on a pool or volume group. This capacity will then not be available for volumes, but can be used by the controllers of the NVMe disks to optimize the NVMe disks' internal operation. Having this dedicated capacity available helps help with the optimisation of the NAND Flash garbage collection, as well as the sustained write performance. This is an optional setting: If the workload is predominantly read-oriented, or if a higher usable capacity is valued higher than sustained write bandwidth, the storage administrator can choose to not provision any optimisation capacity.

The maximum optimisation capacity that is recommended by NetApp depends on the individual NVMe disks' capacity. Figure 36 summarizes the NetApp recommendations. More details on this subject can be found in the NetApp Technical Report TR-4800.

Drive Capacity	Recommended Optimization Capacity (System Manager Defaults)	Approximate Effective Over-provisioning
1.92TB	28%	49%
3.84TB	14%	24%
7.68TB	10%	19%
15.36TB	4%	12%

Figure 36: NetApp-recommended optimisation capacity.

Performance Sizing

In this section we investigate two performance metrics for a single EF600 storage system that is deployed with Spectrum Scale in "SAN Mode": Sequential bandwidth (which is often the predominant sizing metric for HPC&AI storage solutions), and the IO500 suite of performance tests.

Spectrum Scale provides excellent scale-out properties, so many performance metrics (including the aggregate sequential bandwidth) are expected to scale almost linearly as the number of EF600 storage systems in the storage solution is scaled up. Other performance metrics are more difficult to project for multiple EF600 storage systems. If in doubt, it is always advisable to request a storage performance benchmark or proof-of-concept activity through your Lenovo account team.

EF600 Bandwidth

The sequential bandwidth of a single client of the EF600 is shown in Figure 37, as a function of the number of tasks that perform I/O on the client. The achievable bandwidth is limited by the network bandwidth (here: 1x EDR). As with other Spectrum Scale configurations, around four tasks are needed to fully saturate the 100Gbps network link.

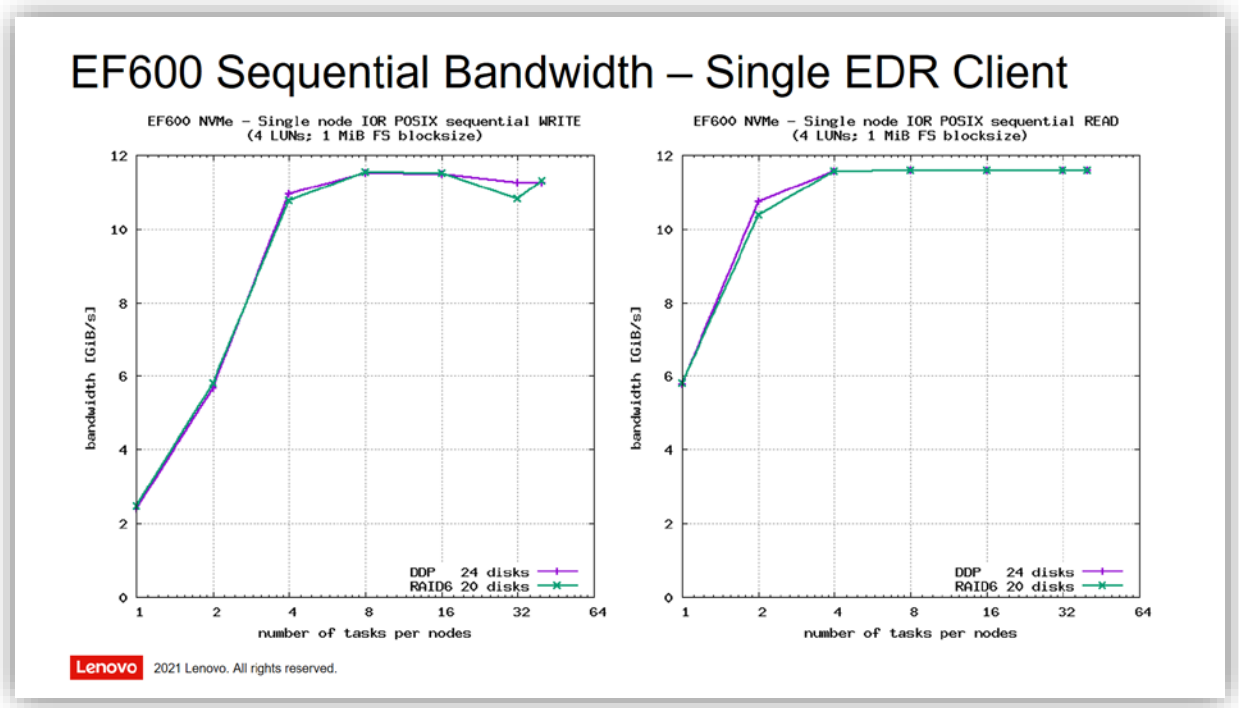

Figure 37: IOR sequential bandwidth: Single EDR client performance.

As the number of clients is increased, the achievable aggregate bandwidth per EF600 depends on both the aggregate bandwidth of the participating NVMe disks, and the bandwidth limits that the EF600 controller itself imposes. In our benchmarking setup, the NVMe disks are dual-ported Samsung PM1725b 1.92 TB drives (PCIe gen3, product ID MZWLL1T9HAJQ-0G5). The peak write speed of these drives is ~2.0 GB/s, peak read speed is about 3.5~GB/s per drive.

Figure 38 shows the scaling curves for DDP volumes as the number of clients is increased. DDP pools from the minimum of 12 disks up to the maximum of all 24 disks have been tested, and runs have been performed with both a single task per node and with 40 tasks/node (one task per physical core). In each case, four volumes with alternating "A" and "B" ownership have been created in the pool to guarantee that all PCIe connections within the EF600 get fully utilized.

- The write bandwidth is clearly limited by the controller's *CME write* bandwidth limit of ~12.5 GB/s (~11.6 GiB/s). The aggregate device bandwidth of even the smallest pool of 12 drives is substantially higher than this limit.
- For *read* bandwidth, the controller limit of ~44 GB/s (~41 GiB/s) is hit by all DDP pool sizes, except for the smallest pool with 12 NVMe drives. Twelve drives have a peak read bandwidth of around 42 GB/s (~39 GiB/s); the achieved read bandwidth is slightly lower than this.

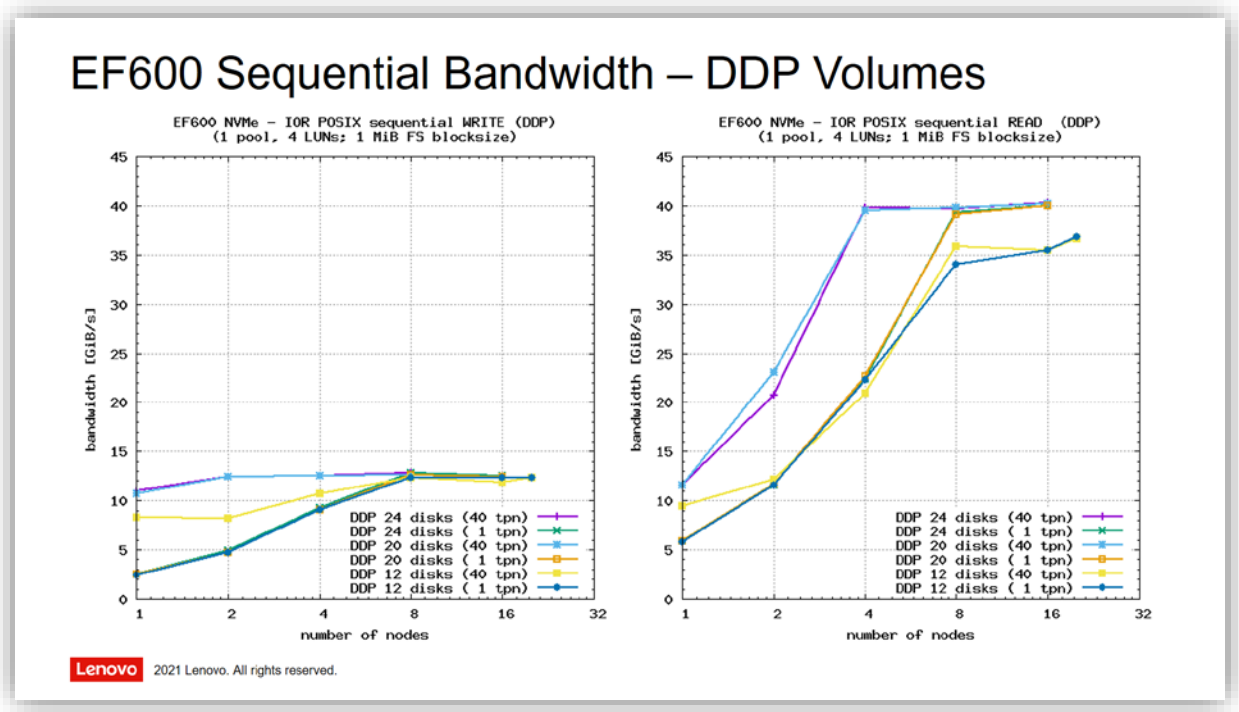

Figure 38: IOR sequential bandwidth: Client scale-out with DDP volumes.

Overall, it is clear that populating an EF600 in DDP mode with more than the minimum of 12 drives provides no increase in write bandwidth, and only a minimal increase in read bandwidth. Adding more drives does of course provide more capacity, and other performance metrics may also benefit from the additional drives.

When RAID6 volumes are used and full-stripe write acceleration can be used, the bandwidth scaling behavior is as shown in Figure 39:

- A *single* RAID6 (8+2P) volume group with 10 drives achieves about 15 GiB/s of *write* bandwidth. This is in line with the 10 devices' peak write bandwidth, reduced by 20% due to the RAID6 overhead.
- When provisioning *two* RAID6 volume groups with a total of 20 drives, the limiting factor is the controller's *FSWA write* bandwidth limit of ~24 GB/s (~22.4 GiB/s). Our testing has achieved slightly better numbers than the published controller maximum.
- For the 100% *read* workload, *ten* of the Samsung NVMe disks in a RAID6 array provide an aggregate bandwidth of ~35 GB/s (~33 GiB/s), and we have been able to achieve this.
- Doubling up the number of disks to 20 disks (in two RAID6 volume groups) exposes the controller's peak *read* performance limit of ~44 GB/s (~41 GiB/s).

As in the DDP case, we have always used four volumes in total, with alternating "A" and "B" ownership, to guarantee that all PCIe connections within the EF600 get fully utilized.

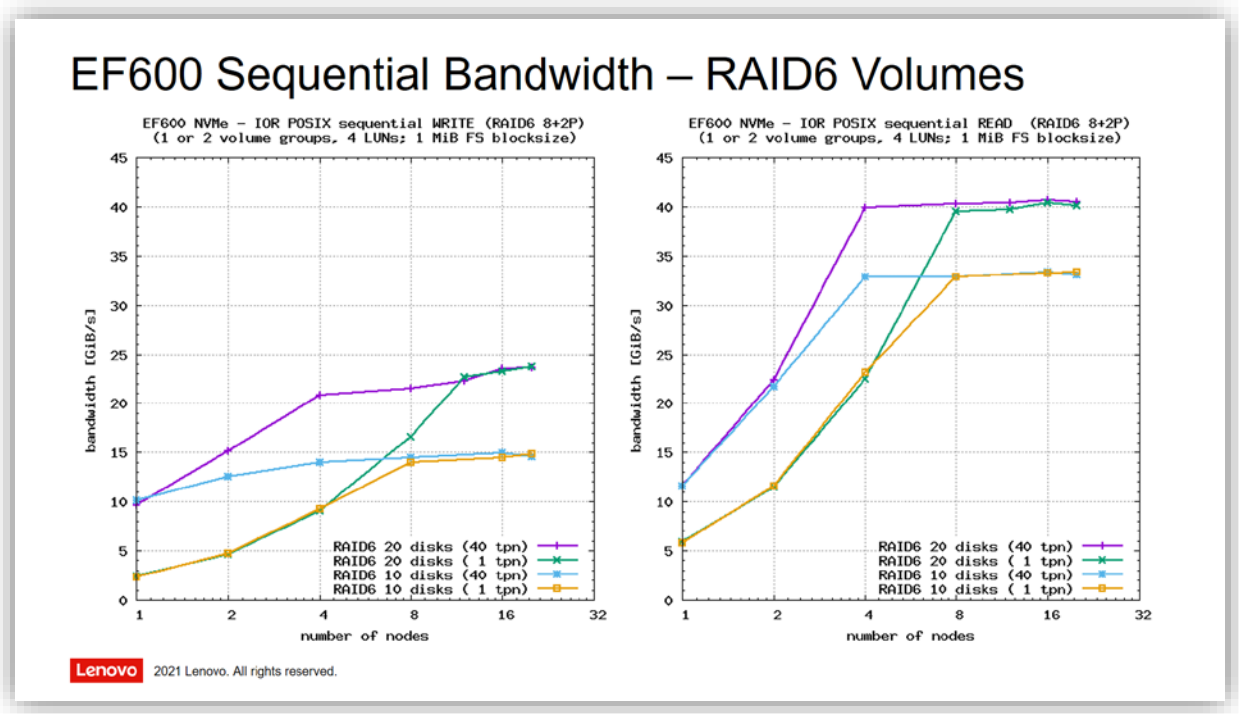

Figure 39: IOR sequential bandwidth: Client scale-out with RAID6 volumes.

These performance results re-emphasize the fact that it is beneficial to use RAID6 (8+2P) volume groups when write bandwidth is important. The FSWA feature of the NetApp controllers enables much higher write bandwidth than DDP. The drawbacks of RAID6 are the lower flexibility in the number of populated drives, and longer rebuild times in case of a drive failure (at least the bigger DDP pools provide better rebuild times; the DDP12 rebuild time is very similar to RAID6).

IO500 Performance of a single EF600

The IO500 HPC storage benchmark suite (https://www.vi4io.org/io500/about/start) contains twelve different benchmarks that measure the bandwidth of large sequential I/O as well as small strided I/O patterns, various metadata workloads, and a "find" directory traversal. The overall IO500 "SCORE" (which is calculated as the geometric mean of all twelve individual measurements) may or may not be useful as a measurement of overall system balance, as typically not all of those test cases are equally important in a production workload. But it is certainly instructive to compare the twelve individual IO500 benchmarks for different parallel file systems and other HPC storage solutions.

Figure 40 and Figure 41 show the IO500 performance results of a single EF600 storage system, configured with DDP volumes on 24 NVMe disks and with RAID6 (8+2P) volumes on 20 NVMe disks. In both runs, 10 client nodes have been used with 40 tasks/node (one per physical core). The sequential bandwidth results that correspond to Figure 38 and Figure 39 are highlighted in green. As expected, the `ior-easy-write` bandwidth depends on the type of volumes (showing the CME limit for DDP volumes, and FSWA for RAID6 volumes). Most of the other tests show very similar performance.

```
IO500 version io500-sc20_v3
[RESULT]       ior-easy-write       11.999765 GiB/s : time 330.250 seconds
[RESULT]    mdtest-easy-write       72.624633 kIOPS : time 314.061 seconds
[RESULT]       ior-hard-write        1.992876 GiB/s : time 405.601 seconds
[RESULT]    mdtest-hard-write        9.352404 kIOPS : time 553.050 seconds
[RESULT]                 find      545.445741 kIOPS : time  50.920 seconds
[RESULT]        ior-easy-read       39.807237 GiB/s : time  99.542 seconds
[RESULT]     mdtest-easy-stat       65.934547 kIOPS : time 342.713 seconds
[RESULT]        ior-hard-read        7.310141 GiB/s : time 110.581 seconds
[RESULT]     mdtest-hard-stat       47.405206 kIOPS : time 108.760 seconds
[RESULT]   mdtest-easy-delete       28.052824 kIOPS : time 807.691 seconds
[RESULT]     mdtest-hard-read       19.955459 kIOPS : time 258.359 seconds
[RESULT]   mdtest-hard-delete        8.455678 kIOPS : time 609.816 seconds
[SCORE] Bandwidth 9.133457 GiB/s : IOPS 39.116552 kiops : TOTAL 18.901570
```

Figure 40: EF600 IO500 performance: DDP with 24 disks, using 10 EDR clients and 40 tasks/node.

```
IO500 version io500-sc20_v3
[RESULT]       ior-easy-write       20.648374 GiB/s : time 358.194 seconds
[RESULT]    mdtest-easy-write       75.592898 kIOPS : time 311.601 seconds
[RESULT]       ior-hard-write        1.975688 GiB/s : time 438.134 seconds
[RESULT]    mdtest-hard-write        9.652048 kIOPS : time 557.228 seconds
[RESULT]                 find      530.515748 kIOPS : time  54.089 seconds
[RESULT]        ior-easy-read       39.873936 GiB/s : time 185.480 seconds
[RESULT]     mdtest-easy-stat       67.903995 kIOPS : time 343.323 seconds
[RESULT]        ior-hard-read        7.693757 GiB/s : time 112.494 seconds
[RESULT]     mdtest-hard-stat       54.439904 kIOPS : time  98.516 seconds
[RESULT]   mdtest-easy-delete       28.956244 kIOPS : time 807.231 seconds
[RESULT]     mdtest-hard-read       20.099673 kIOPS : time 266.694 seconds
[RESULT]   mdtest-hard-delete       10.099587 kIOPS : time 539.940 seconds
[SCORE] Bandwidth 10.576890 GiB/s : IOPS 41.267274 kiops : TOTAL 20.892089
```

Figure 41: EF600 IO500 performance: RAID6 (8+2P) with 20 disks, using 10 EDR clients and 40 tasks/node.

It should be noted that one such IO500 run performs a *single iteration* of each of the twelve tests. In the examples above, the complete IO500 run has a wall clock run time of about one hour. Because only a single iteration of each test is measured, run-to-run variations of the same IO500 suite on the same client and server setup should be expected – especially for the "hard" test cases. The result shown here should not be interpreted as a performance commitment, but only as a rough guideline to indicate the capabilities of an EF600 storage system with Spectrum Scale.

Also note that some of the IO500 metrics depend heavily on the number of clients (number of nodes, number of tasks per node, or both). So it is always good to perform these benchmarks with a client count that best represents the expected workload on the system. We have chosen 10 clients here simply because there is an IO500 "10-Client Challenge" list, which may make comparison of different storage solutions easier than with other numbers of clients.

Lenovo Professional Services

Lenovo offers a variety of professional services in conjunction with its HPC and AI solutions. Please refer to the LeSI Product Guide for general HPC & AI Cluster services. As a general guideline, we recommend to include three Lenovo Professional Services (LPS) Service Units as part of a Spectrum Scale engagement with NetApp EF600 storage, to get customers up and running quickly.

Table 3: Lenovo Professional Services (LPS) Service Unit Part Numbers.

PN or FC	Description	Quantity
5MS7A85672	Professional Service Unit	3

Services are tailored to the customer needs, and typically include:

- Conduct a preparation and planning call
- Verify, and update if needed, firmware on the application nodes
- Verify, and update if needed, firmware on the EF600 storage system(s)
- Configure the network settings specific to the customer environment
- Install the Mellanox OFED networking stack on the application nodes
- Configure the EF600 storage system(s), including storage layout and host topology setup
- Install and configure Spectrum Scale on the EF600 volumes, and validate successful access
- Provide skills transfer to customer personnel
- Develop post-installation documentation describing the specifics of the firmware/software versions, network configuration, and storage system configuration work that was done

The sizing of a Lenovo Professional Services engagement for a Spectrum Scale deployment depends on the size of the storage environment, the number of client nodes/clusters, as well as the complexity of the required integration work. A detailed Statement of Work (SOW) and sizing for specific projects can be provided by the LPS team.

Appendix: Conversion of Decimal and Binary Units

When measuring capacity and bandwidth of high-performance storage systems, the numerical differences between base-10 units and base-2 units are significant. For example, 1000 Byte are one kilo-Byte, with the well-known decimal prefixes of the international SI System. On the other hand, 1024 Byte are one kibi-Byte, with the less well-known binary prefixes (which were first defined in IEC 60027-2).

The effect of this difference is compounding with every order of magnitude, and at Petascale it already results in a difference of over 11%: One Peta-Byte is only 0,888 Pebi-Byte, and one Pebi-Byte equals 1,126 Peta-Byte.

Table 4: Storage capacity measured in base-10 units and base-2 units.

Base-10 Units				Base-2 Units				Base-10 / Base-2 Ratio		Base-2 / Base-10 Ratio	
prefix		value		prefix		value					
kilo	k	10 **	3	kibi	ki	2 **	10	0,976563	-2,34%	1,024000	2,40%
mega	M	10 **	6	mebi	Mi	2 **	20	0,953674	-4,63%	1,048576	4,86%
giga	G	10 **	9	gibi	Gi	2 **	30	0,931323	-6,87%	1,073742	7,37%
tera	T	10 **	12	tebi	Ti	2 **	40	0,909495	-9,05%	1,099512	9,95%
peta	P	10 **	15	pebi	Pi	2 **	50	0,888178	-11,18%	1,125900	12,59%
exa	E	10 **	18	exbi	Ei	2 **	60	0,867362	-13,26%	1,152922	15,29%

The industry standard is to quote *disk storage capacities* in base-10 units, and *memory capacities* in base-2 units. For *bandwidth* (which is capacity transferred per unit of time), there is no clear industry standard. In this document we always use the correct prefix notation (for example, GiB versus GB), and convert all base-10 numbers to base-2 numbers when quoting solution-level capacity and bandwidth figures.

Additional Resources

IBM Spectrum Scale 5.1.0 Product Documentation:

- Knowledge Center
  https://www.ibm.com/support/knowledgecenter/STXKQY_5.1.0/ibmspectrumscale510_welcome.html
  - Installation Guide
  - Administration Guide
  - Command and Programming Reference
- Spectrum Scale FAQ
  https://www.ibm.com/support/knowledgecenter/STXKQY/gpfsclustersfaq.pdf
- *Spectrum Scale: Metadata secrets and sizing revealed* presentation on the Spectrum Scale User Group website: http://files.gpfsug.org/presentations/2016/south-bank/D2_P2_A_spectrum_scale_metadata_dark_V2a.pdf

NetApp EF600 Product Documentation:

- Technical Report TR-4800: Introduction to NetApp EF600 Array
  https://www.netapp.com/pdf.html?item=/media/17009-tr4800pdf.pdf
- Datasheet: NetApp EF600 All-Flash NVMe Array
  https://www.netapp.com/pdf.html?item=/media/8116-ds-4002.pdf
- EF600 Online Documentation:
  https://mysupport.netapp.com/documentation/docweb/index.html?productID=62963
- SANtricity Express Configuration Guide for Linux
  http://docs.netapp.com/ess-11/topic/com.netapp.doc.ssm-exp-ic-lin/Linux%20express%20configuration.pdf
- NettApp Interoperability Matrix Tool (IMT)
  http://mysupport.netapp.com/matrix (NetApp support login required)

NVM Express Standards:

- NVM Express™ Base Specification Revision 1.4b, September 21, 2020
  http://nvmexpress.org
- NVM Express™ over Fabrics, Revision 1.1, October 22, 2019
  http://nvmexpress.org

Lenovo Press Guides:

- Lenovo Scalable Infrastructure (LeSI) Solutions
  https://lenovopress.com/lp0900
- Lenovo Distributed Storage Solution for IBM Spectrum Scale (DSS-G)
  https://lenovopress.com/lp0837
- Lenovo SAN Storage
  https://lenovopress.com/storage/san/lenovo

About the Author

Michael Hennecke is Lenovo's Chief Technologist for HPC Storage and Networking. He has over 28 years of experience in High Performance Computing and HPC Storage, and is currently focusing on emerging storage solutions like NVMe-over-Fabrics. Michael holds a Master of Science degree in physics from Ruhr-Universität Bochum (Germany), and a "Distinguished IT Specialist" certification from The Open Group.

Thanks to the following people for their contributions to this project:

- Mark Regester (NetApp)
- Jason Knauff (NetApp)
- Chris Hepner (NetApp)
- Abdel Sadek (NetApp)
- Dan Bennett (NetApp)
- Adam Cook (NetApp)
- Kuok Hoe Tan (NetApp)
- Florian Zillner (Lenovo)
- Sigrun Eggerling (Lenovo)
- Taylor Allison (Lenovo)
- David Watts (Lenovo)

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. A current list of Lenovo trademarks is available on the Web at https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:

Lenovo®
System x®
ThinkSystem

The following terms are trademarks of other companies:

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc.

IBM, Spectrum Scale, and the marks listed at http://www.ibm.com/legal/copytrade.shtml are trademarks or registered trademarks of International Business Machines Corp.

Intel®, Intel Optane™, and Xeon® are trademarks of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

