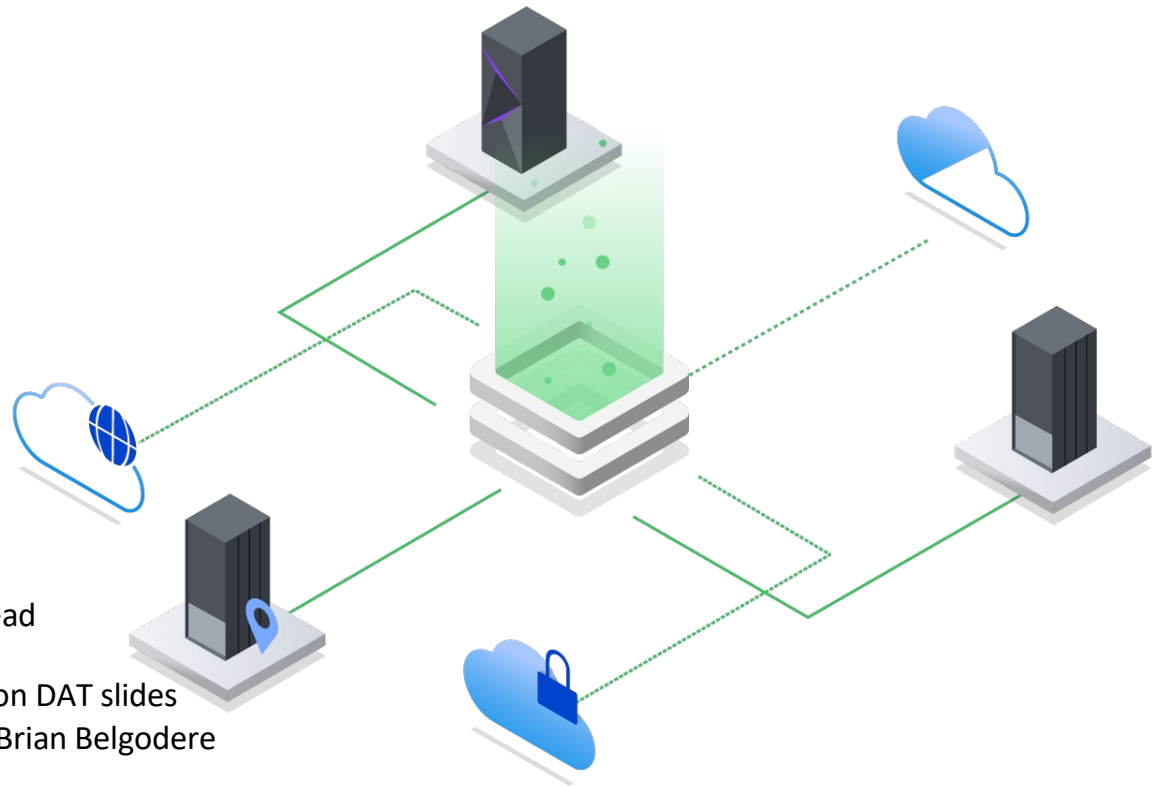


IBM Storage Scale Performance Update

User Group Meeting, SC2025



John Lewars
IBM Storage Scale Development Performance Lead

Input/help from John Divirgilio and Olaf Weiser on DAT slides
Blue Vela io500 runs done in collaboration with Brian Belgodere



Agenda

- Update on DAT
- Details on IBM's Storage Scale submissions for io500
- Container Performance Issue with Direct I/O and Improvement in 5.2.3

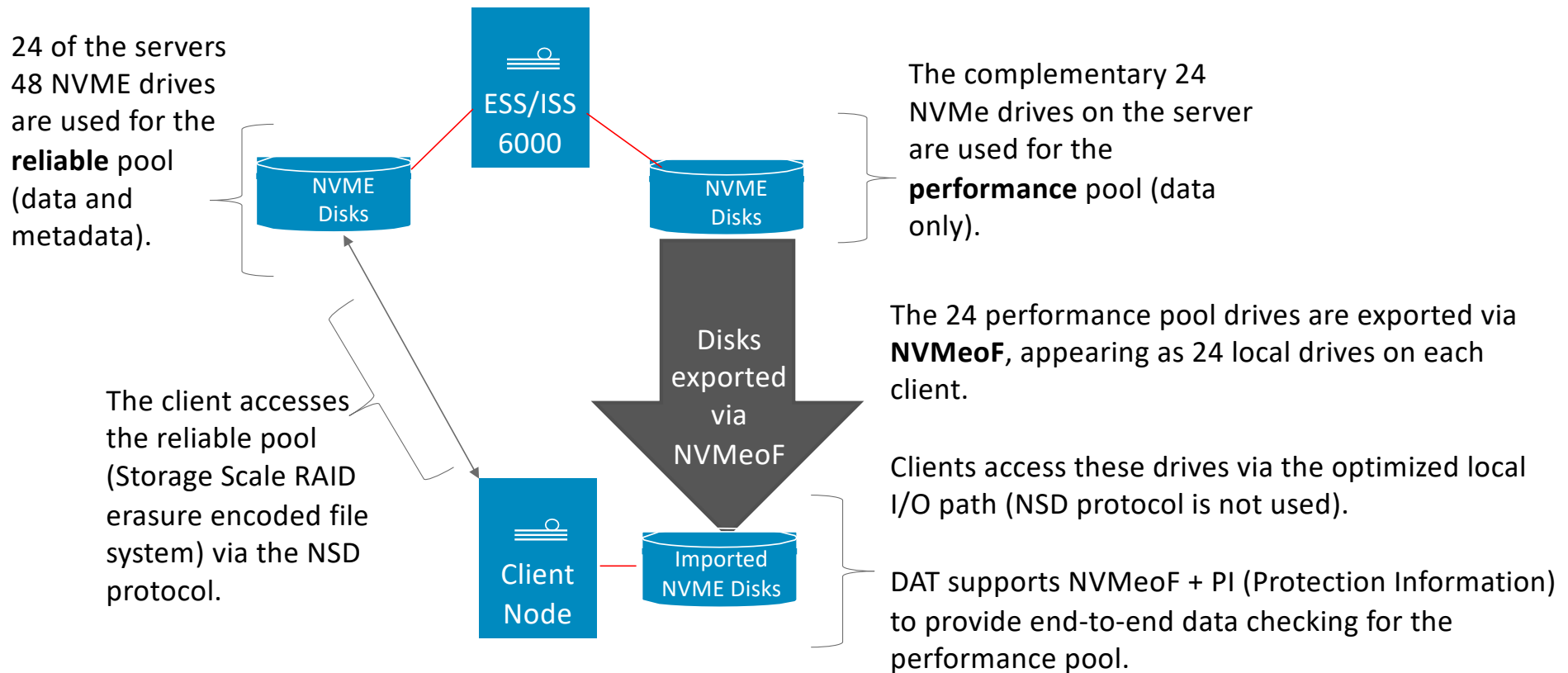
DAT / Asymmetric Data Replication

- GPFS maintains the consistency of the reliable and performance replicas in a filesystem, just like it does for multiple data replicas.
- DAT filesystem reads are from the performance replica for optimal performance.
- Data and metadata are stored in the reliable pool while the performance pool is for data only.
- The decentralized DAT deployment enables policy and mmchattr enhancements to affinitized performance replicas created on (or moved to) clients, to allow the clients to operate on local data without traversing the network

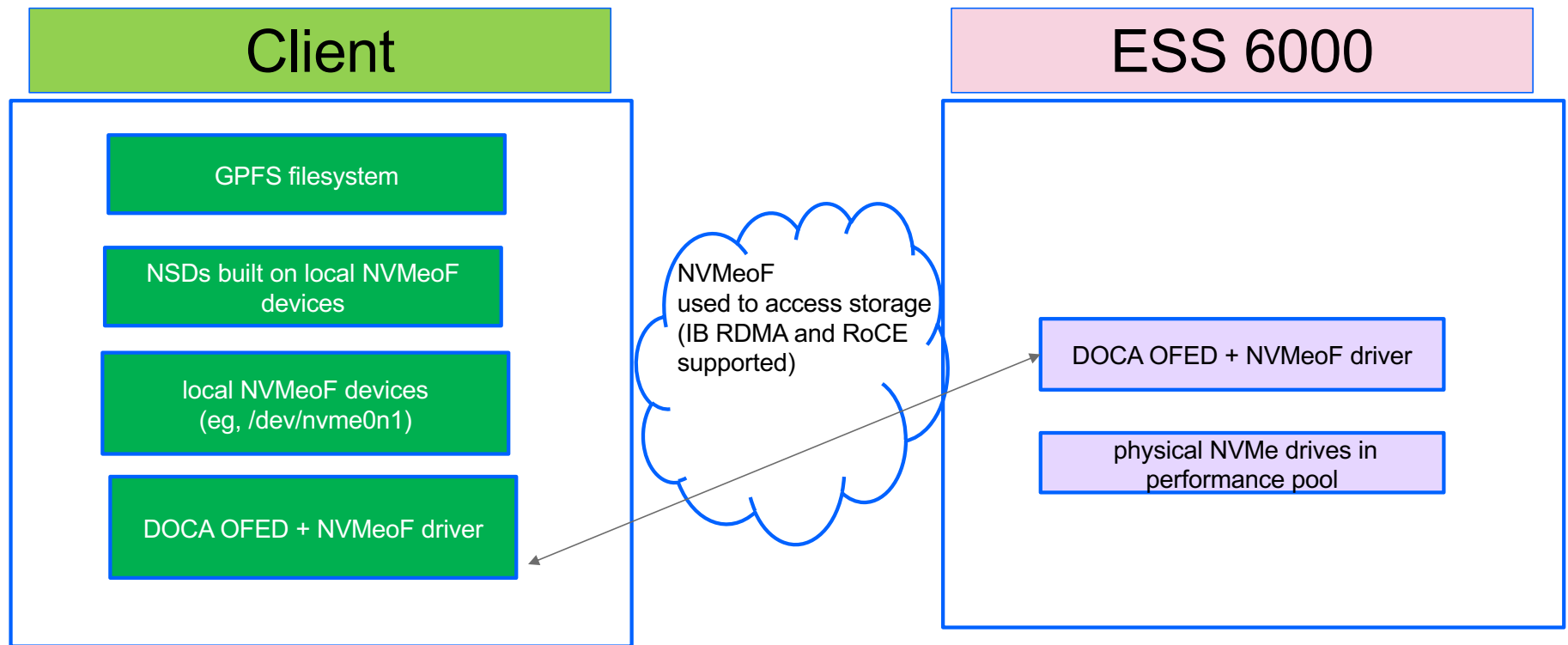
Update on DAT status added in Jan. 2026:

- Support for decentralized DAT (local drive solutions) deployments was delivered in the Storage Scale IBM Storage Scale System (ESS) 7.0.0.0 release
- IBM intends to release support for centralized DAT later in 2026

Centralized DAT Deployment (aka, DAT-NVMeoF)



Centralized DAT Performance Pool Block Diagram



Centralized DAT performance

IBM intends to advertise up to **28 million** 4K random read IOPs aggregate with centralized DAT deployments that do **not** enable PI (up to 30 million IOPS have been measured in the lab).

IBM intends to advertise up to **25 million** 4K random read IOPs aggregate with centralized DAT deployments that **do** enable PI data verification (up to 27 million IOPS have been measured in the lab).

This is over a 10X improvement over the IOPs we can deliver now using one ESS 6000 with a standard ESS/GNR filesystem.

Test configuration:

Clients: 20 (24 AMD cores per client)

Network: 400 Gb/s RoCE

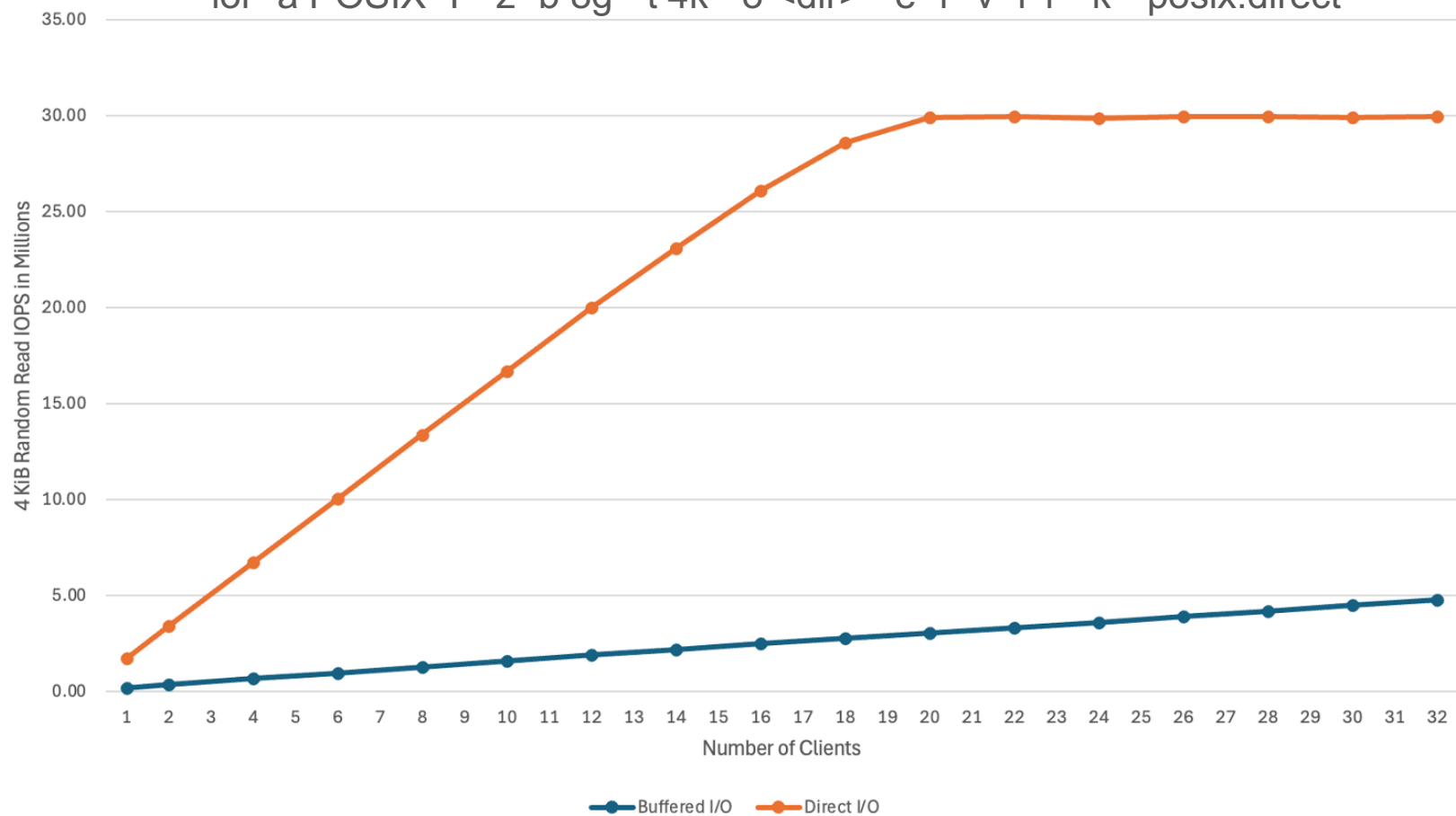
ESS 6000:

NVMe: 48 x 3.84TB NVMe (Gen 5)

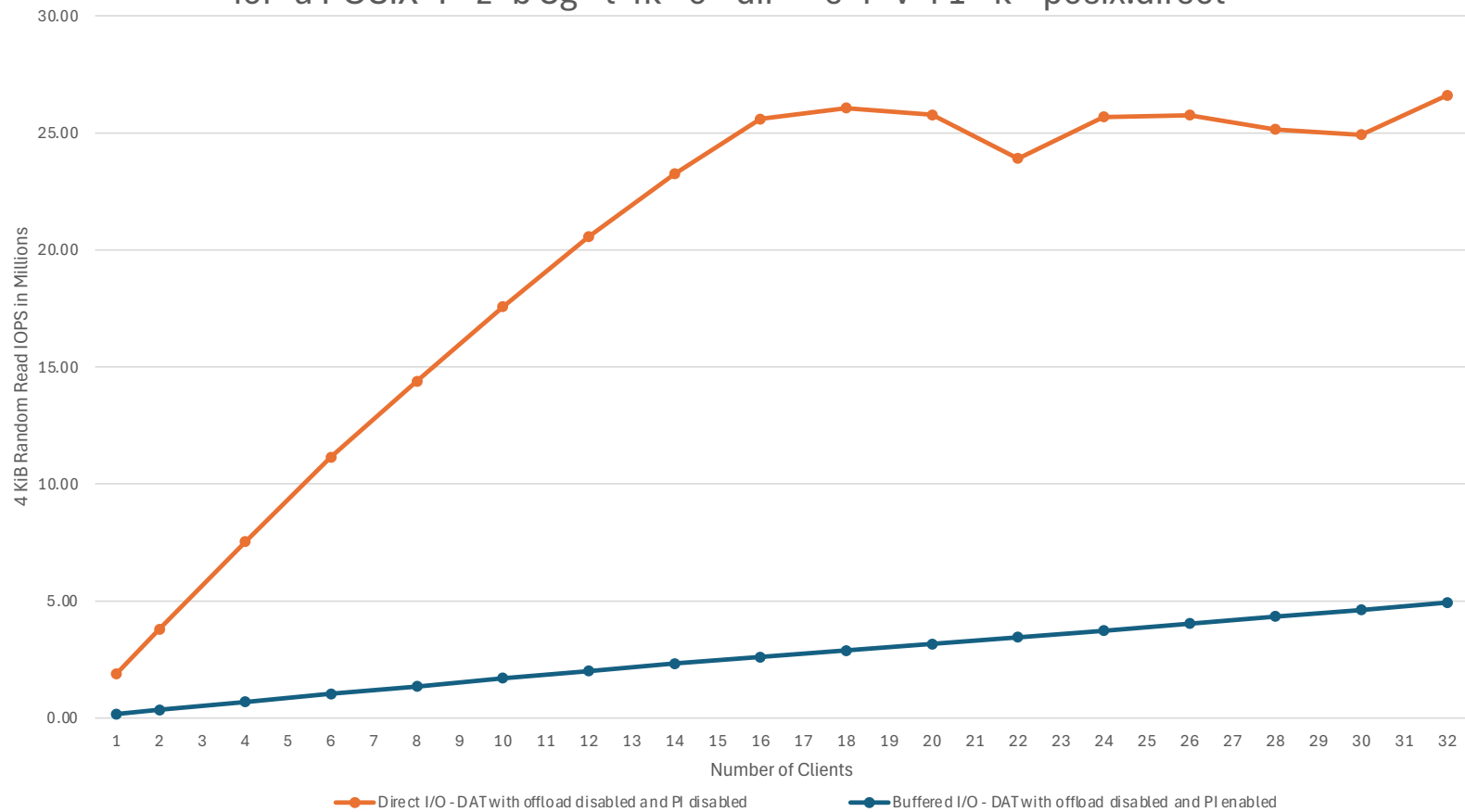
IB adapters per canister: 8 x 200Gb/s ports (CX-7 connections to SN5600 switch configured for RoCE)

NVMeoF target offload: enabled for non-PI measurements and disabled for PI measurements

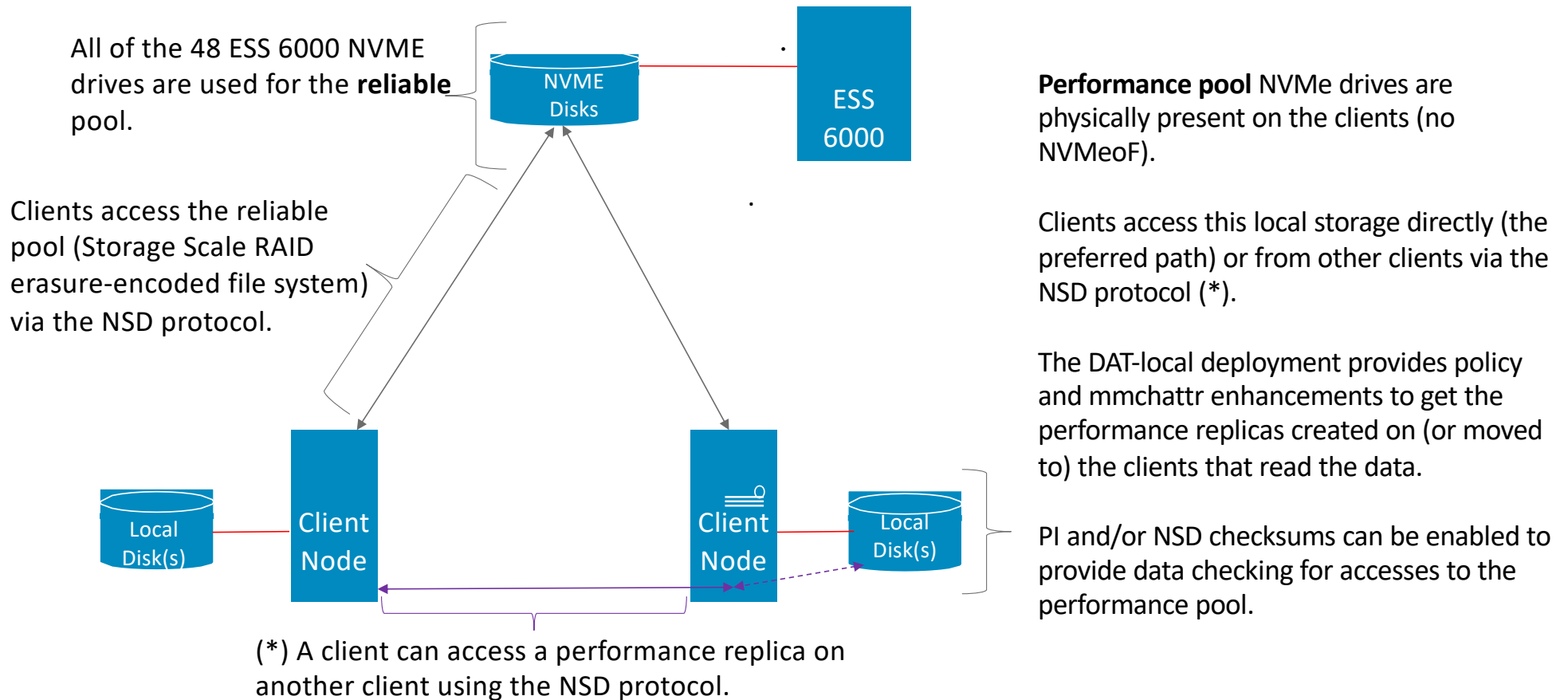
4 KiB DIO Random Read with Centralized DAT on SSS 6000
NVMeoF **offload is Enabled and PI is Disabled**
(average of 5 three-minute runs – each thread reads an 8 GB data file)
`ior -a POSIX -F -z -b 8g -t 4k -o <dir> -e -r -v -i 1 -k --posix.direct`



4 KiB DIO Random Read with Centralized DAT on SSS 6000
NVMeoF **offload is Disabled** and **PI is enabled**
(average of 5 three-minute runs - each task has its own 8 GB data file)
`ior -a POSIX -F -z -b 8g -t 4k -o <dir> -e -r -v -i 1 -k --posix.direct`



Decentralized DAT Deployment (aka, DAT-Local)



Decentralized DAT Deployment Performance

Performance is dependent on the number of clients as well as the NVMe and compute capabilities of the clients.

IBM lab results show between **1.0 - 1.4 million** 4KiB random read IOPs per client on systems when reading local data

As the client count is increased, we observe **linear** scaling because the performance replicas reside local to the clients.

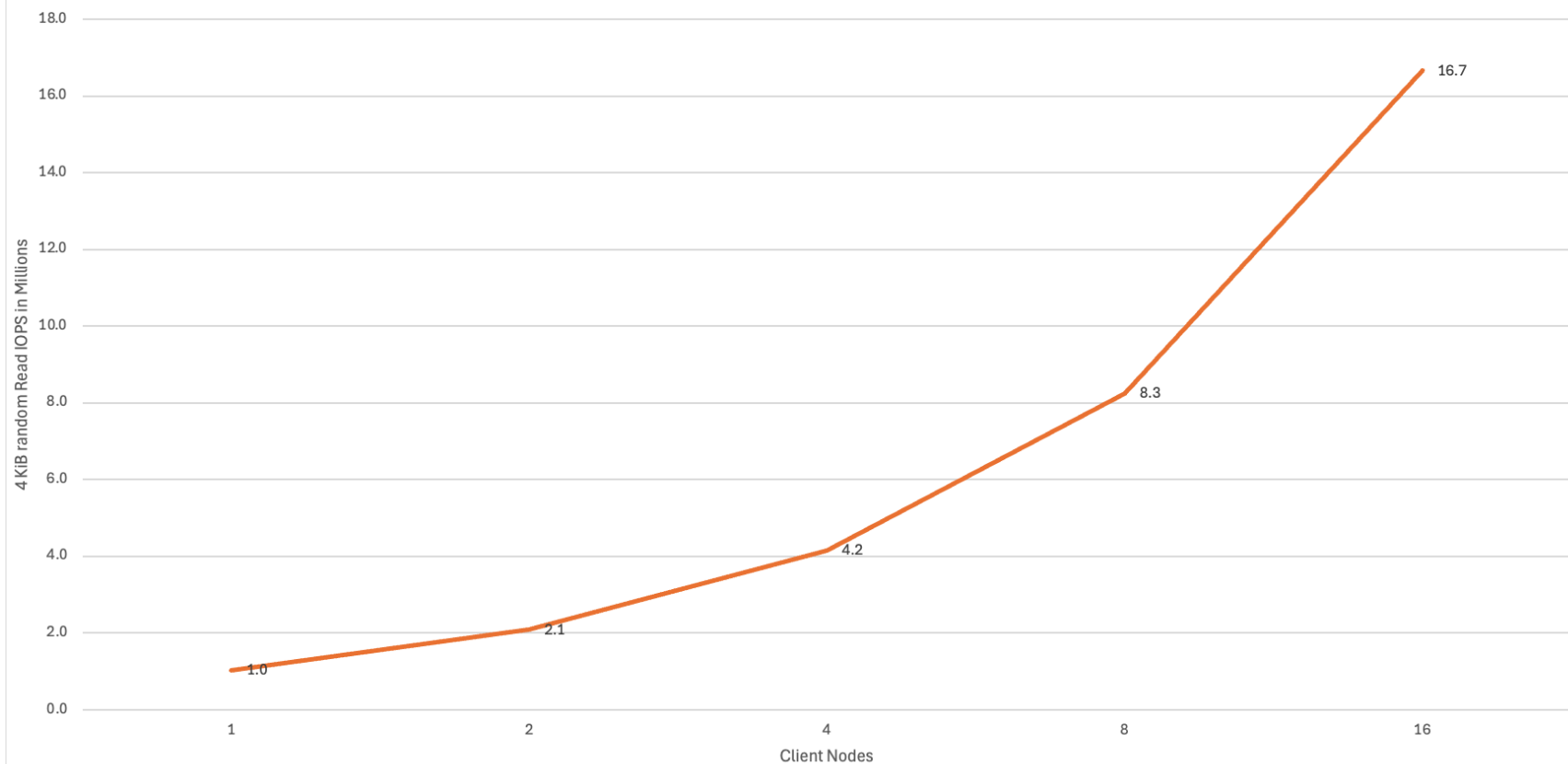
Test configuration for Scaling runs:

Clients: 10, each client has 48 AMD cores and 2 NVMe drives

ESS 6000 (reliable pool)

4 KiB Random Read IOPS Performance with Decentralized DAT (Local Drives in Client Nodes)

IOPS Ehningen bb8 ISS 6000 + 2 NVME drives per client - 4 KiB Random Read IOPS (**Millions**)
Local Drive Variant of DAT - 32 clients qdepth=256 njobs=32 - Scaling Up Number of Clients



Asymmetric Data Replication Value Proposition: Exploiting local storage

Local storage configurations (particularly NVMe drives in client nodes) have become more common and IBM Storage Scale provides multiple solutions that leverage local storage:

Existing solutions that use local storage to create a filesystem:

- SNC (Shared-nothing cluster)
- FPO (File Placement Optimizer: SNC with write-affinity)
- ECE (Erasure Code Edition)
- DAT (Data Acceleration Tier)

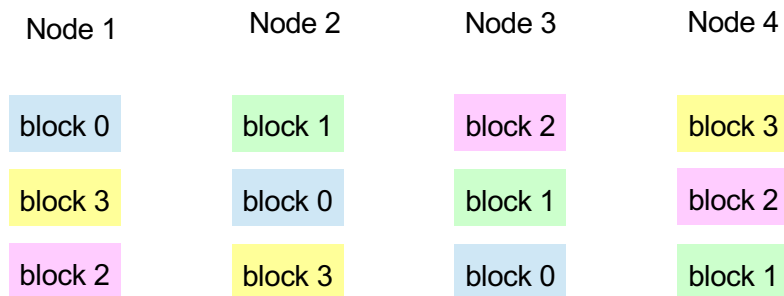
Existing solution that uses local storage to cache data for existing file systems:

- LROC (Local Read-Only Cache)

Using Asymmetric Data Replication, DAT combines key features of these existing local storage solutions to create a filesystem and cache data in a persistent, non-redundant manner.

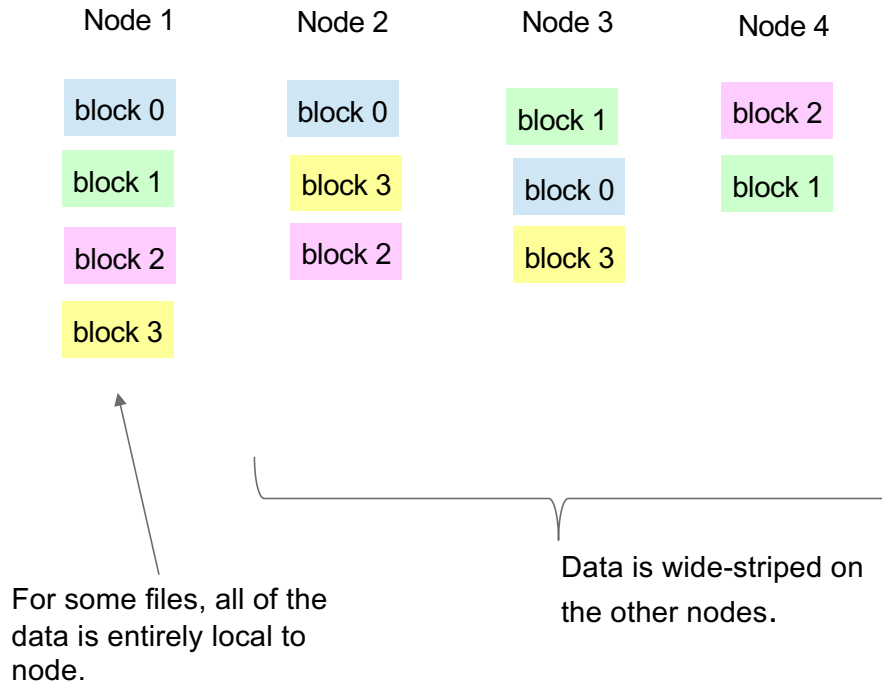
Workloads on local storage with SNC (Shared Nothing Cluster)

- Data Access: Workloads that use data-aware schedulers (e.g., Hadoop) can take advantage of locally stored data.
- Data Reliability/Availability: 3-way replication. Metadata scan needed after a failed node rejoins the cluster.
- Cluster Resources: Each unit of user-consumable storage uses **3x** units of filesystem storage due to replication overhead.



Workloads on local storage with FPO (File Placement Optimizer)

- Data access: FPO provides a method to place data so a wider range of applications can exploit data locality.
- Data reliability/availability: 3-way replication. Metadata scan needed after a failed node rejoins the cluster.
- Cluster resources: Each unit of user-consumable storage uses **3x** units of storage due to replication overhead.

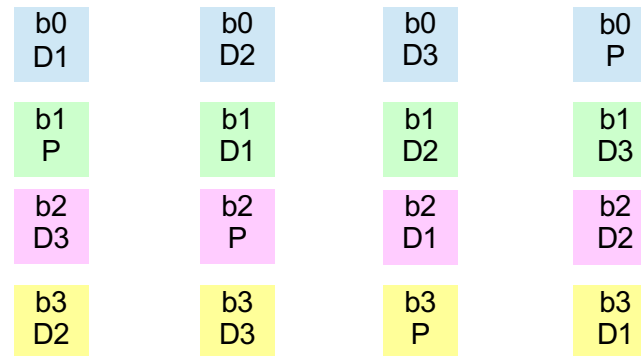


Workloads with ECE (Erasure Code Edition) on Storage-Rich Servers

Employs Storage Scale RAID software across local storage in the cluster. Each user data block is split into data and parity strips, and these strips are spread across ECE server nodes.

- Data Access: Most data is not local and there is no control over data placement.

A read typically requires gathering strips from other ECE servers (multiple network hops).



- Data Reliability/Availability:

Uses erasure encoding and parity strips to provide data reliability, and this more efficient than using traditional replication.

- Cluster Resources: Each unit of user-consumable storage uses 1.25x of storage resources due to parity overhead (assuming 8+2P).

Workloads on local storage with an Asymmetric Data Replication File System

Maintain two copies (replicas) of each data block in two different pools with asymmetric data replication.

- “reliable copy”: protected by software erasure encoding
- “performance copy”: stored on a client-local NVMe drive or an NVMeoF drive, providing fast reads

The performance copy is a persistent, non-redundant cache.

Data Access: **The performance copy (replica) is local to the client (NVMe drive or NVMeoF drive).**

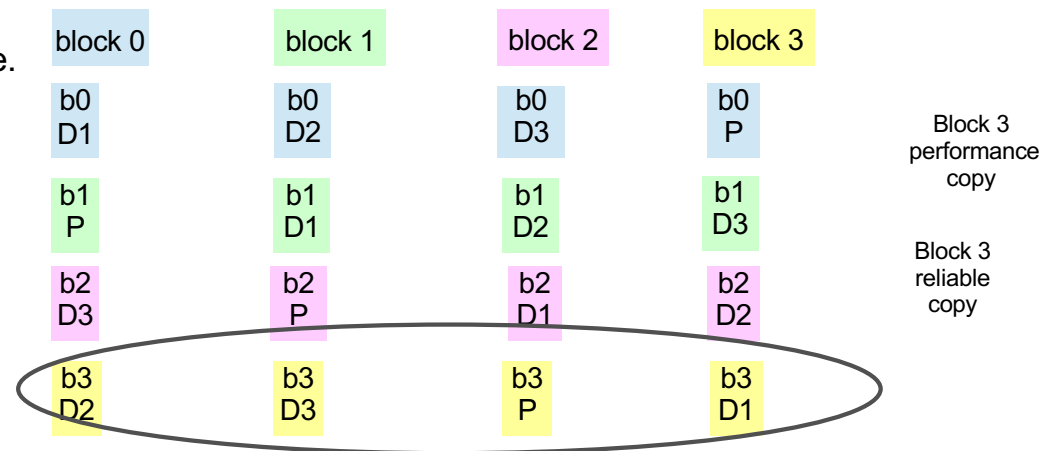
Data Reliability/Availability:

The performance copy is maintained with a feature called “lazy consistency”, where missed updates are repaired (initiated) on the next read after the disk is back online.

A full metadata scan is not needed when a performance disk is brought back online.

Also, when a missing update is repaired, all missed updates in the same indirect block are fixed.

Cluster resources: Each unit of user-consumable storage uses 2.25x of storage resources, lower than 3x FPO.



Workloads on local storage with LROC

LROC uses local storage (typically SSD/NVMe) to support caching.

Asymmetric Data Replication/DAT has advantages over LROC:

- LROC cache is potentially redundant because a data replica can be cached on multiple nodes. No more than a single performance replica exists in a DAT filesystem.
- LROC is not persistent because a reload is needed after a failed node is back up. With DAT, the performance copy is persistent.
- LROC is a true cache, so it doesn't help Direct I/O workloads. DAT provides a high-performance tier that accelerates both Direct I/O and buffered I/O workloads.

Creating an ESS 6000 Decentralized DAT (DAT-local) filesystem

1. Install ($\geq$) Scale 6.0.0.1 and ESS 7.0.0.0 software and create a file system on the SSS 6000 using the standard deployment approach
2. Run “mmchconfig release=LATEST” to upgrade the daemon version. *
3. Run “mmchfs -V full” to upgrade the filesystem format version. *
4. Add the performance pool on client-local NVMe drives using “mmcrnsd” and “mmvdisk”.

* If cluster is upgraded BEFORE creating file system, these steps aren't required

DAT configurations for the Fall 2025 GA

An ESS 6000 supports one DAT filesystem.

A DAT filesystem may not use more than one ESS 6000.
(IBM intends to remove this restriction in a future release.)

DAT filesystems use the fully-configured ESS 6000 performance model with NVMe drives.

DAT filesystems do not support a remote storage cluster.
(IBM intends to remove this restriction in a future release.)

Replica Placement in FPO

Details on FPO Entities and Policies

The mechanisms used to control the placement of files described in the “File Placement Optimizer Documentation” can be applied to control how files are placed when using the Decentralized DAT feature. See: <https://www.ibm.com/docs/en/storage-scale/6.0.0?topic=administering-file-placement-optimizer>

Definitions for FPO allowWriteAffinity and 'Chunks'

allowWriteAffinity: This storage pool configuration option defines if a given pool is considered to be :

- 1) FPO mode (allowWriteAffinity=yes . . . *deprecated for pure local disk configs in Scale 6.0*) OR:
- 2) replica-based sharing nothing mode (allowWriteAffinity=no)

Chunks: Chunks allow for configuring logical groups of blocks, such that each chunk behaves like a single large block size (this is useful for applications that need high sequential bandwidth, e.g. many Hadoop applications). Each chunk size is defined by: **file system block size * blockGroupFactor** where the blockGroupFactor is defined to be between 1-1024 (defaulting to 1) and is defined according to one of following methods (only applicable to FPO mode and DAT performance storage pools):

- 1) Using the **blockGroupFactor** storage pool config parameter
- 2) Via the **setBGF** argument to the mmchpolicy and mmappypolicy commands
- 3) On a per file basis via the **mmchattr** command:
mmchattr --block-group-factor <desired_chunk_size> <filename>

Extended Failure Groups Defined (1/3)

Extended failure groups : A failure group is defined as a set of disks that share a common point of failure that might cause them all to become simultaneously unavailable. Originally, all IBM Storage Scale failure groups were identified by single integers (-1 is the default failure group and can be used for all disks that don't share a common point of failure).

Extended failure groups apply only to affinitized storage pools and are specified as a vector of three comma-separated numbers. This vector conveys topology information that Scale exploits when making data placement decisions. In general, a topology vector is a way for the user to specify which disks are closer together and which are farther away. The three elements of the failure group topology vector are typically referred to as representing:

The rack number ,

Rack position

(0 = the top of rack ;

1 = the bottom of rack) ,

Node number within the rack

Three rack example shown:

Rack 1	Rack 2	Rack 3
1, 0, 1	2, 0, 1	3, 0, 1
1, 0, 2	2, 0, 2	3, 0, 2
1, 0, 3	2, 0, 3	3, 0, 3
1, 0, 4	2, 0, 4	3, 0, 4
1, 1, 5	2, 1, 5	3, 1, 5
1, 1, 6	2, 1, 6	3, 1, 6
1, 1, 7	2, 1, 7	3, 1, 7
1, 1, 8	2, 1, 8	3, 1, 8

Extended Failure Groups Defined (2/3)

Extended failure groups cont.: The first two elements of the failure group represent the failure group ID and the three elements together represent the locality group ID. For example, for the failure group 2,1,0:

2,1 is the failure group ID and
2,1,0 is the locality group ID

IBM Storage Scale makes **data** block placement decisions about the disk selection for data replicas based on the Failure Group definitions. **Note that metadata cannot be affinitized through extended failure group definitions.**

The FPO docs note that:

- Disks that differ in the first of the three numbers are farthest apart (as they are in different racks).
- Disks that have the same first number but differ in the second number are closer (as they are in the same rack, but in different halves).
- Disks that differ only in the third number reside in different nodes in the same half of the same rack.
- Only disks that have all three numbers in common reside in the same node (all the disks that reside in a single node must have the same failure group number)

Decentralized DAT, however, allows for only **one** performance replica which can be affinitized to local storage devices, and therefore cannot optimize the placement of **multiple** replicas according to the rack number and position within the rack. For this reason, creating locality group IDs that differ only in the third element of the vector, as per the example on the following chart.

Extended Failure Groups Defined (3/3)

Extended failure groups cont.: To specify the topology vector that identifies a failure group, you use the failureGroup attribute in an NSD stanza that is passed to the **mmcrnsd** command as per the following example (note that affinitized pools only support layoutMap=cluster):

For example, we could create an NSD stanza file ([as per the example below, nsdStanza](#)) for a performance pool and create the performance NSDs via “**mmcrnsd -F nsdStanza -v no**”

```
%pool:
  pool=perf
  allowWriteAffinity=yes
  performancePool=yes
  layoutMap=cluster
  blockSize=16M
```

```
%nsd:
  device=/dev/nvme0n1
  server=saguarolib
  nsd=saguarol_NSD1
  usage=dataOnly
  pool=perf
  failureGroup=1,0,1
```

[...]

```
%nsd:
  device=/dev/nvme6n1
  server=saguarol0ib
  nsd=saguarol0_NSD6
  usage=dataOnly
  pool=perf
  failureGroup=1,0,10
```

Write Affinity Depth Defined

Write affinity depth is a policy that allows the application to determine the layout of a file in the cluster to optimize for typical access patterns.

The write affinity is specified by a depth that indicates the number of localized copies (as opposed to wide striped). It can be specified at the storage pool or file level.

The enabling of Write affinity depth indicates that the first replica is being written on the node where the writing is triggered. In FPO deployment (no longer supported starting with Storage Scale 6.0) It also managed how the second and third replica (if any) are being written on the other node disks.

To specify write affinity depth, you use the **writeAffinityDepth** attribute in a storage pool stanza (as input to the `mmadddisk` or `mmcrfs` command) or the **--write-affinity-depth** argument of the **mmchattr** command.

You can use **--block-group-factor** argument of the `mmchpool` command to change a storage pool's block group factor.

You can also specify the write affinity depth for file by the `setWAD` directive of the `mmchpolicy` and `mmappypolicy` commands. However, starting with IBM Storage Scale 6.0 release, write affinity depth is only supported in IBM Decentralized DAT configurations and the `setWADFGP` directive replaces `setWADFG` to affinitize data in a DAT performance pool.

Write Affinity Depth – WAD=0 (1/2)

WAD=0 is defined in the [FPO docs](#) as follows:

A write affinity depth of 0 indicates that each replica is to be striped across the disks in a cyclical fashion with the restriction that no two disks are in the same failure group. By default, the unit of striping is a block; however, if the block group factor is specified in order to exploit chunks, the unit of striping is a chunk.

Based on WAD=0 tests below is an example of the first 3 chunks allocations (marked: replica_no. – chunk no.)

Below each box represents a disk, with two disks per each (color-coded) node. Each rack has 4 nodes and 8 disks in total.

The **extended failure group number** (unique per node, with four nodes per rack in this example) is listed in **brown**.

The **disk ID number** (unique per disk, with two disks per node) is shown in **red**. The **ingesting node** is circled in **purple**.

	Rack 1	Rack 2	Rack 3	Rack 4	Rack 5
0-Chunk No.	Replica 0 striped on a per 'chunk' basis (each chunk numbered)				
1-Chunk No.	Replica 1 striped on a per 'chunk' basis (each chunk numbered)				
2-Chunk No.	Replica 2 striped on a per 'chunk' basis (each chunk numbered)				
	0-0 1,0,1-1	0-1 2,0,1-9	1-0 3,0,1-17	1-2 4,0,1-25	5,0,1-33
	1,0,1-2	2,0,1-10	3,0,1-18	4,0,1-26	5,0,1-34
	1,0,2-3	2,0,2-11	3,0,2-19	4,0,2-27	2-1 5,0,2-35
	1,0,3-4	2,0,2-12	3,0,2-20	4,0,2-28	5,0,2-36
	0-0 1,1,3-5	2,0,3-13	1-1 3,1,3-21	4,1,3-29	5,1,3-37
	1,1,3-6	2,0,3-14	3,1,3-22	4,1,3-30	5,1,3-38
	1,1,4-7	0-2 2,0,4-15	4,1,4-23	2-0 4,1,4-31	2-2 5,1,4-39
	1,1,4-8	2,0,4-16	4,1,4-24	4,1,4-32	5,1,4-40

Write Affinity Depth – WAD=0 (2/2)

WAD=0 Performance Replica Placement Example

Based on WAD=0 tests (from [here](#)) the previous example of replica allocation is extended and only replica 0 is shown.

Below each box represents a disk, with two disks per each (color-coded) node. Each rack has 4 nodes and 8 disks in total.

The **extended failure group number** (unique per node, with four nodes per rack in this example) is listed in **brown**.

The **disk ID number** (unique per disk, with two disks per node in this example) is shown in **red**. The **ingesting node** is circled in **purple**.

With WAD=0, a DAT performance replica is striped across the all the disks in the system, cycling on a half rack basis, starting with a node in the next half rack from the ingesting node. The picture below represents the allocation of the first eighteen chunks of a file ingested on the first node (rack 1, node 1, failure group number: **1,0,1**). Note that chunks are striped on a half rack basis (one chunk assigned per node before moving to next half rack) but it's possible to define all nodes in the system as being in a single rack and all in the upper or lower rack.

Rack 1	Rack 2	Rack 3	Rack 4	Rack 5
1,0,1-1	2,0,1-9 11	3,0,1-17 13	4,0,1-25 15	5,0,1-33 17
1,0,1-2	2,0,1-10	3,0,1-18	4,0,1-26	5,0,1-34
1,0,2-3 9	1 2,0,2-11	3 3,0,2-19	5 4,0,2-27	7 5,0,2-35
1,0,2-4	2,0,2-12	3,0,2-20	4,0,3-28	5,0,2-36
0 1,1,3-5	2,1,3-13 12	4 3,1,3-21	4,1,3-29 16	5,1,3-37
1,1,3-6	2,1,3-14	3,1,3-22	4,1,3-30	5,1,3-38
1,1,4-7 10	2 2,1,4-15	3,1,4-23 14	6 4,1,4-31	8 5,1,4-39
1,1,4-8	2,1,4-16	3,1,4-24	4,1,4-32	5,1,4-40

Replica chunks are shown in order of allocation: **0 1 2 3** etc (chunk 0 is the first (2MB) chunk of the file).

Write Affinity Depth – WAD=1 (1/2)

WAD=1 Defined in the FPO Docs as per below:

A write affinity depth of 1 indicates that the first copy is written to the writer node. The second copy is written to a different rack. The third copy is written to the same rack as the second copy, but on a different half (which can be composed of several nodes). *

Based on WAD=1 tests below is an example how 3 chunks allocations (marked: replica_no. – chunk no.) work with 3 replicas.

Below each box represents a disk, with two disks per each (color-coded) node. Each rack has 4 nodes and 8 disks in total.

The **extended failure group number** (unique per node, with four nodes per rack in this example) is listed in **brown**.

The **disk ID number** (unique per disk, with two disks per node) is shown in **red**. The **ingesting node** is circled in **purple**.

	Rack 1	Rack 2	Rack 3	Rack 4	Rack 5
0-Chunk No.	0-0 1,0,1-1 0-2 0-1 1,0,1-2	1-1 2,0,1-9 1-0 2,0,1-10	3,0,1-17	4,0,1-25	5,0,1-33
1-Chunk No.	1,0,2-3 1,0,3-4 1,1,3-5 1,1,3-6	2-2 2,0,2-11 2,0,2-12 2,0,3-13 1-2 2,0,3-14	3,0,2-19 3,0,2-20 3,1,3-21 3,1,3-22	4,0,2-27 4,0,2-28 4,1,3-29 4,1,3-30	5,0,2-35 5,0,2-36 5,1,3-37 5,1,3-38
2-Chunk No.	1,1,4-7 1,1,4-8	2-0 2,0,4-15 2-1 2,0,4-16	4,1,4-23 4,1,4-24	4,1,4-31 4,1,4-32	5,1,4-39 5,1,4-40

***note: the documentation defines where the first chunk is written but it should note chunks are striped**

Write Affinity Depth – WAD=1 (2/2)

WAD=1 Performance Replicas Placement Example

Based on WAD=1 tests in the lab, the previous example of replica allocation has been extended and only replica 0 is shown.

Below each box represents a disk, with each (color-coded) node having two disks.

The **extended failure group number** (unique per node, with four nodes per rack in this example) is listed in **brown**.

The **disk ID number** (unique per disk, with two disks per node in this example) is shown in **red**.

The **ingesting node** is circled in **purple**.

With WAD=1, the DAT performance replica is striped across the disks in only the node that ingests the data. The picture below represents the allocation of the first eight chunks of a file ingested on the first node (rack 1, node 1, failure group number: **1,0,1**). The chunks alternate in placement across the disks local to node (two disks in this example). (Each chunk size = (File System Blocksize)*(Block Group Factor) = (1MB) * (2) = 2MB)

Rack 1	Rack 2	Rack 3	Rack 4	Rack 5
0 2 4 6 1,0,1-1	2,0,1-9	3,0,1-17	4,0,1-25	5,0,1-33
1 3 5 7 1,0,1-2	2,0,1-10	3,0,1-18	4,0,1-26	5,0,1-34
1,0,2-3	2,0,2-11	3,0,2-19	4,0,2-27	5,0,2-35
1,0,2-4	2,0,2-12	3,0,2-20	4,0,3-28	5,0,2-36
1,1,3-5	2,1,3-13	3,1,3-21	4,1,3-29	5,1,3-37
1,1,3-6	2,1,3-14	3,1,3-22	4,1,3-30	5,1,3-38
1,1,4-7	2,1,4-15	3,1,4-23	4,1,4-31	5,1,4-39
1,1,4-8	2,1,4-16	3,1,4-24	4,1,4-32	5,1,4-40

Replica chunks are shown in order of allocation: **0 1 2 3** etc. (chunk 0 is the first (2 MB) chunk of the file).

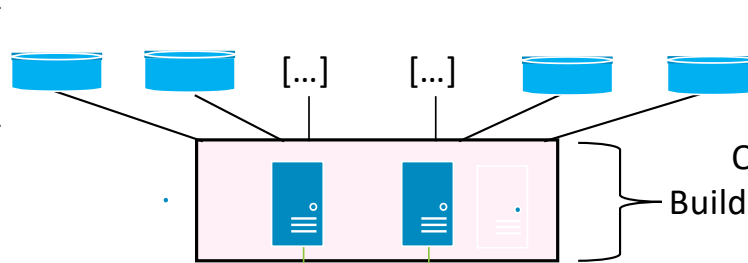
Example Policy from Saguaro SC25 io500 Submission

Below is an example of a policy used to place ior-easy file for the io500 benchmark to allow reads of the file to be to local disk:

```
RULE 'ior_file_easy.0-24' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000000' AND name <=
'ior_file_easy.00000024') AND (setWADFGP(1,0,10))
RULE 'ior_file_easy.25-49' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000025' AND name <=
'ior_file_easy.00000049') AND (setWADFGP(1,0,1))
RULE 'ior_file_easy.50-74' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000050' AND name <=
'ior_file_easy.00000074') AND (setWADFGP(1,0,2))
RULE 'ior_file_easy.75-99' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000075' AND name <=
'ior_file_easy.00000099') AND (setWADFGP(1,0,3))
RULE 'ior_file_easy.100-124' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000100' AND name <=
'ior_file_easy.00000124') AND (setWADFGP(1,0,4))
RULE 'ior_file_easy.125-149' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000125' AND name <=
'ior_file_easy.00000149') AND (setWADFGP(1,0,5))
RULE 'ior_file_easy.150-174' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000150' AND name <=
'ior_file_easy.00000174') AND (setWADFGP(1,0,6))
RULE 'ior_file_easy.175-199' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000175' AND name <=
'ior_file_easy.00000199') AND (setWADFGP(1,0,7))
RULE 'ior_file_easy.200-224' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000200' AND name <=
'ior_file_easy.00000224') AND (setWADFGP(1,0,8))
RULE 'ior_file_easy.225-249' SET POOL 'system' REPLICATE(0:1) WHERE (name >= 'ior_file_easy.00000225' AND name <=
'ior_file_easy.00000249') AND (setWADFGP(1,0,9))
RULE 'default' SET POOL 'system'
```

Saguaro System Storage Details

IBM Storage Scale RAID Pool (46 twin-tailed NVMe Drives, connected to Storage Scale System 6000 servers - 2 failed drives)

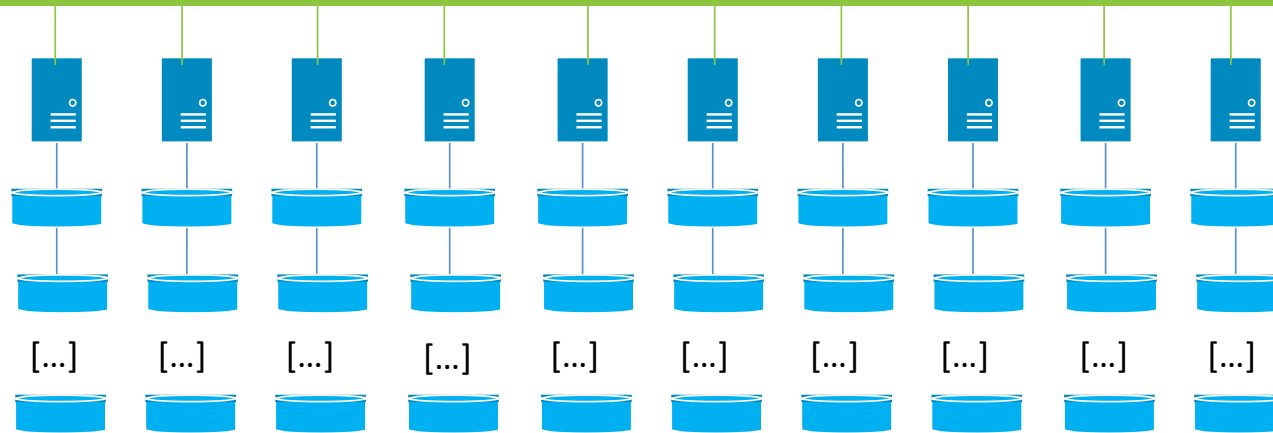


One Storage Scale System 6000 Building Block has two canisters (server nodes)

NDR400 InfiniBand Network with RDMA Enabled

Ten Saguaro Client Nodes

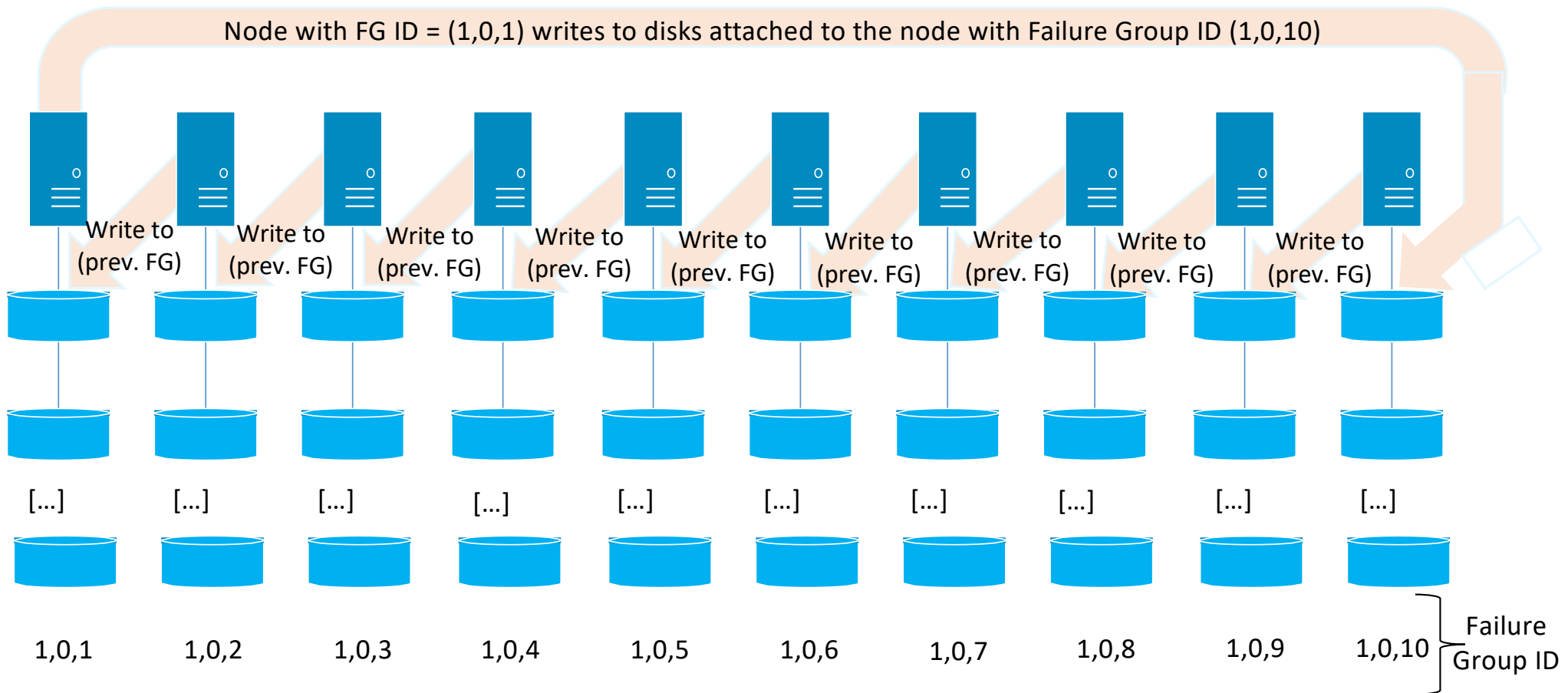
Decentralized DAT Pool
(uses 7 of the 8 client NVMe drives available in each client)



Saguaro System uses Policy to Facilitate Local Disk Reads for ior-easy Files

- Each client writes to decentralized DAT pool disks that are local to the node where the data will later be read

Node with FG ID = (1,0,1) writes to disks attached to the node with Failure Group ID (1,0,10)



SC25 io500 Submission 784 (Saguaro) Results

	Saguaro SC24	Saguaro SC25	
	IBM Storage Scale 5.2.2-0 (early pre-GA build) with ESS 6.2.1-0	IBM Storage Scale 6.0.0.0 with ESS 7.0.0.0 (early pre- GA build)	
Benchmark/Metric			Percent Improvement for SC25
ior-easy-write	151.1	238.7	58
mdtest-easy-write	191.2	223.6	17
ior-hard-write	28.3	39.7	41
mdtest-hard-write	24.4	23.9	-2
find	7516.7	4803.1	-36
ior-easy-read	284.2	422.2	49
mdtest-easy-stat	571.3	546.6	-4
ior-hard-read	107.5	142.4	32
mdtest-hard-stat	272.7	292.6	7
mdtest-easy-delete	165.3	218.9	32
mdtest-hard-read	321.6	365.3	14
mdtest-hard-delete	29.4	28.0	-5
ior-rnd4K-easy-read	N/A	13.0	N/A
Bandwidth (GiB/s)	106.9	154.5	45
IOPS (kiops)	232.5	234.6	1
Total	157.7	190.4	21

Details on IBM's SC25 Saguaro System io500 Submission (1/4)

Hardware Details and Tuning Regarding IBM's Saguaro SSS6000 Cluster io500 submissions for SC24:

- **1x ESS 6000 Building Block**, 2 servers/canisters per BB running Scale 6.0.0 with Storage Scale Systems 7.0.0 (early pre-GA build)
 - 16 MB file system block size
 - (default) NPS-4 (Numa Nodes Per Socket) BIOS Tuning, plus default ESS PCI Tuning, enabling ESS default PCI relaxed ordering and `INT_LOG_MAX_PAYLOAD_SIZE = _4KiB(12)` ←
 - Eight CX-7 links per server/canister running 200 Gb/s Infiniband
 - Dual AMD EPYC 9454 48-Core Emb Processors
 - **46x** Samsung 1733a NVMe Drives (2 of 48 drives had failed due to system issue) twin-tailed
- **10x Lenovo AMD clients:**
 - One NDR400-Infiniband connection used per client with PCI relaxed ordering enabled
 - 768 GB memory (pagepool=300 GiB) + Dual EPYC 7302P 16-Core sockets per client (SMT off and only the first socket used by io500 and Storage Scale)
 - Seven of the eight client 3.2 TB Intel INTEL SSDPF2KE032T10 NVMe drives used for decentralized DAT performance pool

```
#INT_LOG_MAX_PAYLOAD:  
#using default tuning for the SSS 6000:  
for i in `ls /sys/class/infiniband/`; do  
    echo y | mlxconfig -d $i -e s  
    INT_LOG_MAX_PAYLOAD_SIZE=12  
done
```

```
#NPS BIOS Setting is under "AMD  
#CBS" (Core and Bus Settings)
```

Details on SC25 Saguaro System io500 Submission (2/4)

Additional Configuration Details:

- Manager node role assigned to all clients (including 10 clients not running io500) and both SSS 6000 storage servers
- Manager node role assigned to all clients and two servers (only 10 of the clients and the two SSS 6000 servers described above are directly running io500)
- Kernel command line option “mitigations=off” on all the servers and clients running io500

MPI-Level Changes only for Ehningen Run:

- `mpi-args: --bind-to numa:1`

(Binding to the same socket as the single CX-7 adapter used for the io500 run)

Benchmark Configuration:

- Unlike the previous Saguaro io500 submission the internal io500 version of pfind is run instead of an external script
- Hints used: `--posix.gpfs.finegrainwritesharing`, `--posix.gpfs.finegrainreadsharing`, and `--posix.gpfs.createsharing`
- IOR was built with assyc. I/O support

Details on SC25 Saguaro System io500 Submission (3/4)

Servers' Relevant mmchconfig Tuning:

- dataShipServerBufferPct 50

On both clients and servers, the following mmchconfig tuning was done:

- RDMA enabled on all clients and servers (verbsRdmaSend + verbsRdma enabled)
- preferDesignatedMnode yes
- maxMBpS 50000
- maxStatCache 131072
- maxFilesToCache 131072
- nBucketGroups 1024
- preferDesignatedMnode yes
- fsynclsGlobal no
- autoCompactDir no
- fgdITokenBatchAcquire yes

Details on SC25 Saguaro System io500 Submission (4/4)

Additional Saguaro client-only mmchconfig Tuning:

- dataShipClientBuffersPerServer 50
- dataShipClientBufferPct 50
- pagepool 300G
- nsdSmallThreadRatio 2
- nsdThreadsPerQueue 16
- nsdbufspace 70
- numactlOptioni 0 # Binding mmfsd memory to same socket (0) as the CX-7 network adapter used for io500
- numactlOptionN 0 # Binding mmfsd execution to same socket (0) as the CX-7 network adapter used for io500
- verbsPorts mlx5_0/1 # only use the CX-7 network adapter in first NUMA domain (socket 0) for RDMA

Notes on the Blue Vela io500 Submission

- Blue Vela is a production system run by IBM Research for internal workloads, comprising 768 H100-based Dell XE9680s, with a production file system served by four building blocks of IBM Storage Scale System 6000
- The primary workload for Blue Vela is internal AI model development and testing, along with a growing quantum HPC workload used by Researchers, AI and Data Engineers, and Software Developers.
- Compute is accessed via an LSF job scheduler for mission-critical AI workloads. While the system is comprised of 768 compute nodes, the io500 run was performed on an allocation of 20 nodes made available for MPI job access outside LSF, and additional tuning was performed in collaboration between IBM Storage Scale Development and IBM Research.
- During the io500 runs, which occupied 2.6% of the cluster's compute capacity, the remaining 97% was fully occupied by various IBM Research workloads (shared system usage - **not** a dedicated system run)



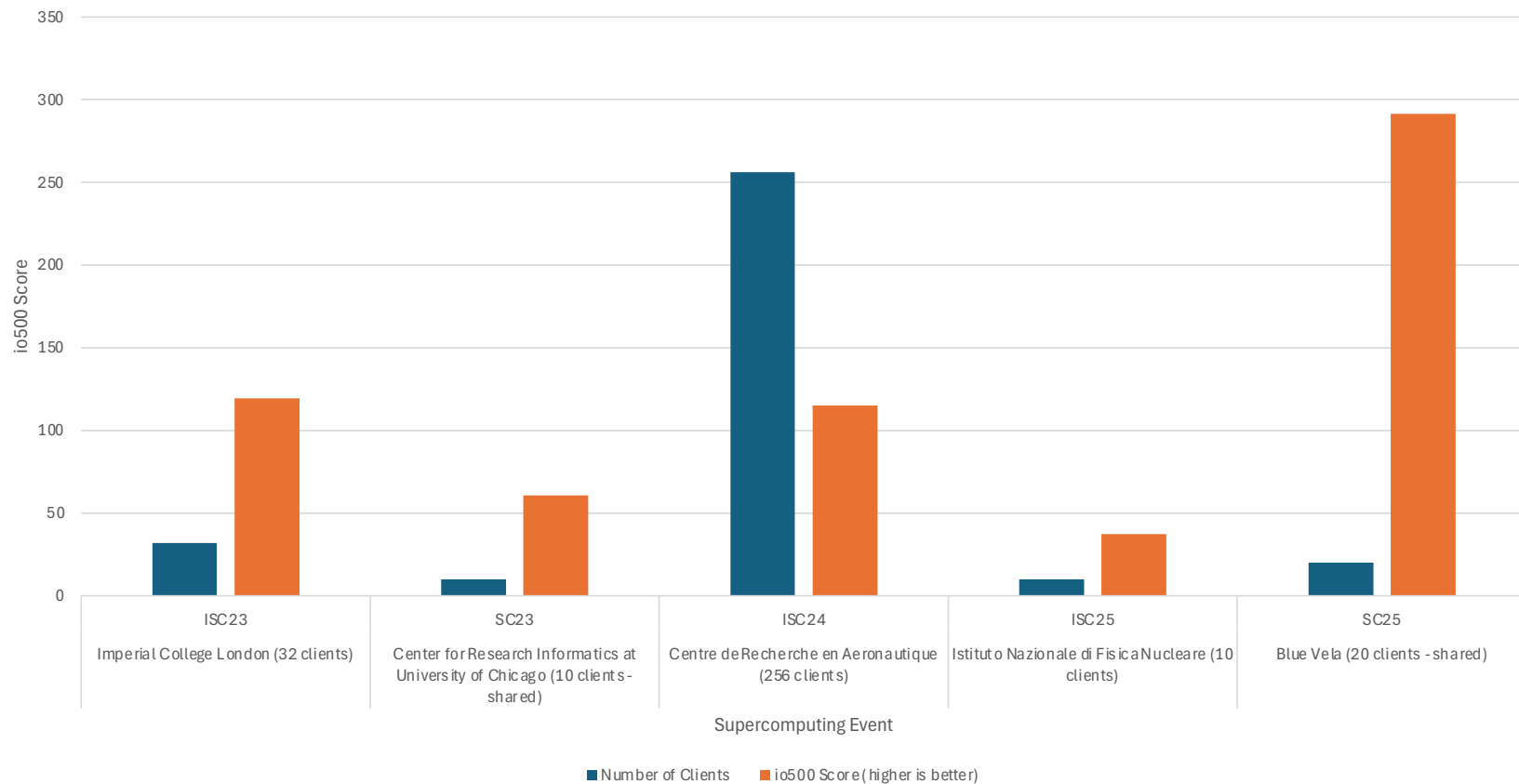
Blue Vela io500 Submission

	Saguaro SC25	Blue Vela SC25	
	IBM Storage Scale 6.0.0.0 with ESS 7.0.0.0 (early pre- GA build)	IBM Storage Scale 5.2.2.1 with ESS 6.2.2.0	
Benchmark/Metric			Percent Improvement for BV
ior-easy-write	238.7	436.6	83
mdtest-easy-write	223.6	506.8	127
ior-hard-write	39.7	107.2	170
mdtest-hard-write	23.9	40.3	69
find	4803.1	1922.5	-60
ior-easy-read	422.2	615.7	46
mdtest-easy-stat	546.6	559.6	2
ior-hard-read	142.4	172.1	21
mdtest-hard-stat	292.6	560.1	91
mdtest-easy-delete	218.9	321.6	47
mdtest-hard-read	365.3	547.5	50
mdtest-hard-delete	28.0	51.3	83
ior-rnd4K-easy-read	13.0	5.6	-57
Bandwidth (GiB/s)	154.5	265.4	72
IOPS (kiops)	234.6	320.4	37
Total	190.4	291.6	53

IBM Storage Scale io500 **Production** Submissions

Blue Vela Submission ID 785

(categorized as production for SC25)



Details on IBM's SC25 Blue Vela System io500 Submission (1/4)

Hardware Details and Tuning Regarding IBM's Blue Vela SSS6000 Cluster io500 submission for SC25:

- **4x ESS 6000 Building Blocks** used, 2 servers/canisters per BB running Storage Scale 5.2.2.1 with Storage Scale Systems (ESS) 6.2.2.0:
 - 16MB file system block size
 - (default) NPS-4 (Numa Nodes Per Socket) BIOS Tuning, plus default ESS PCI Tuning, enabling ESS default PCI relaxed ordering and INT_LOG_MAX_PAYLOAD_SIZE = _4KiB(12)
 - Four CX-7 links per server/canister running 400 Gb/s InfiniBand (NDR400)
 - Dual AMD EPYC 9454 48-Core Emb Processors
 - 48x Samsung 1733a NVMe drives in all four building blocks
- **20x Dell XE9680 clients** used, out of a total 768 clients – the rest are running Blue Vela production:
 - Two NDR400-Infiniband connections used on client for communication with storage servers
 - 2 TiB memory memory (pagepool=1024 GiB) + Dual Intel(R) Xeon(R) Platinum 8468 sockets per client (SMT on)
 - Clients have additional NVMe drives that are not exploited by this SC25 submission (we plan to enable the IBM Storage Scale decentralized DAT solution on BV when it becomes generally available)



Details on SC25 Blue Vela System io500 Submission (2/4)

Additional Configuration Details:

- Manager node role assigned to all eight Storage Scale System 6000 I/O servers along with two additional Storage Scale System 3500 servers (the 3500 servers serve a cold storage tier that was not used for the io500 run)

Benchmark Configuration:

- Unlike the most recent io500 submissions made by IBM Storage Scale Development, the internal io500 version of pfind is run instead of an external script
- Hints used: --posix.gpfs.finegrainwritesharing, --posix.gpfs.finegrainreadsharing, and --posix.gpfs.createsharing

Details on SC25 Blue Vela System io500 Submission (3/4)

Servers' Relevant mmchconfig Tuning:

- dataShipServerBufferPct 50

On both clients and servers, the following mmchconfig tuning was done:

- RDMA enabled on all clients and servers (verbsRdmaSend + verbsRdma enabled – all links on Storage network enabled for RDMA)
- preferDesignatedMnode yes
- maxMBpS 50000
- maxStatCache 131072
- maxFilesToCache 131072
- nBucketGroups 1024
- preferDesignatedMnode yes
- fsynclsGlobal no
- autoCompactDir no

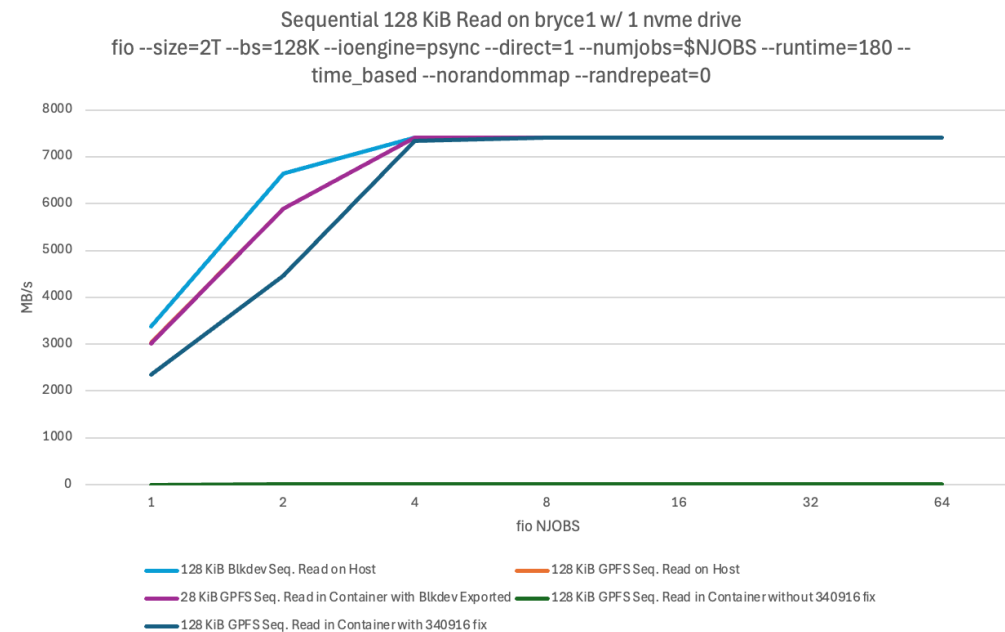
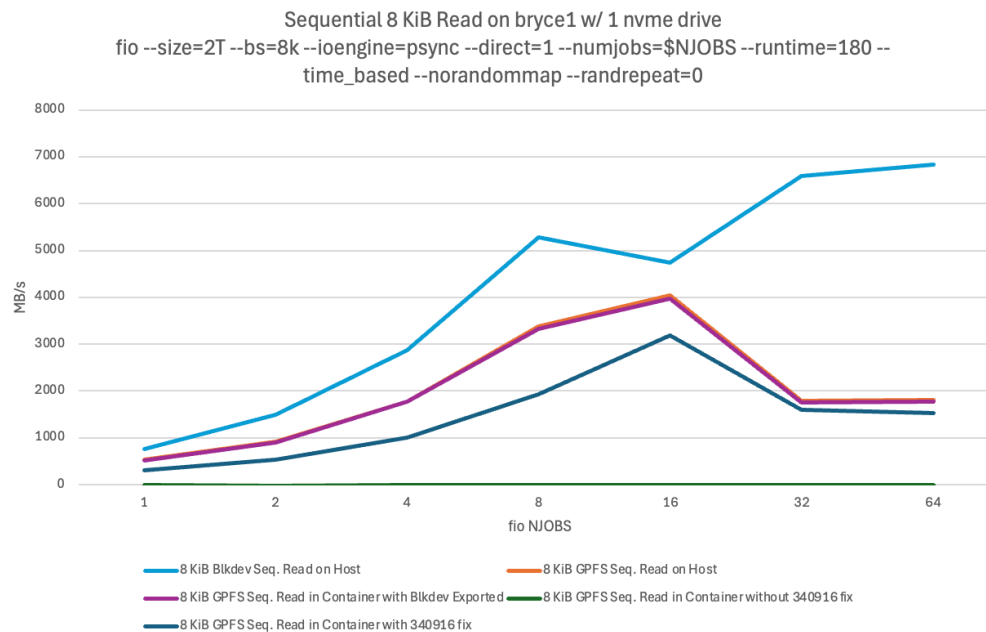
Additional, client-only, mmchconfig Tuning:

- dataShipClientBuffersPerServer 50
- dataShipClientBufferPct 50
- pagepool 1024G

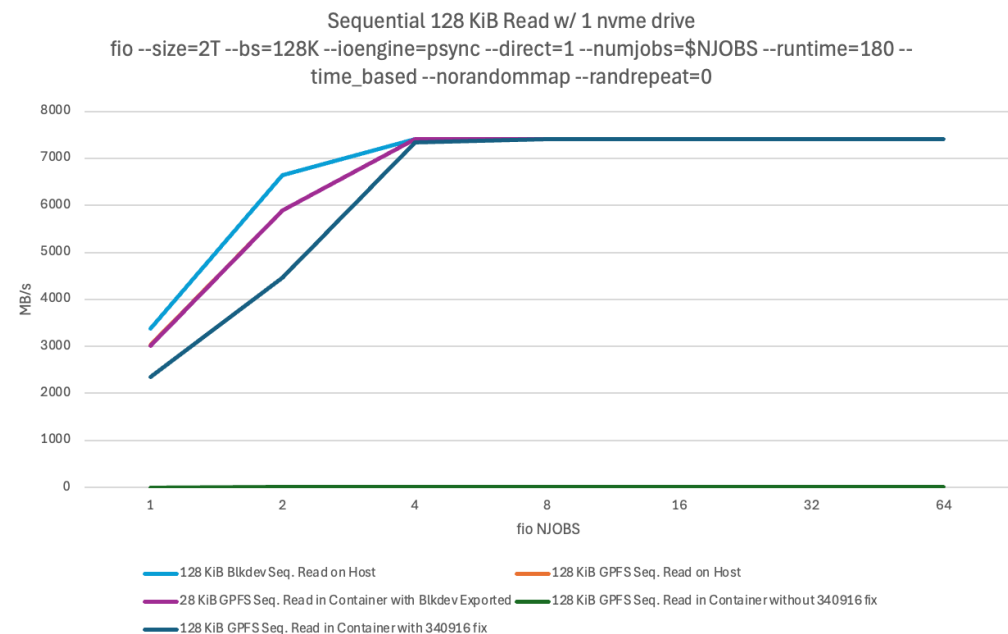
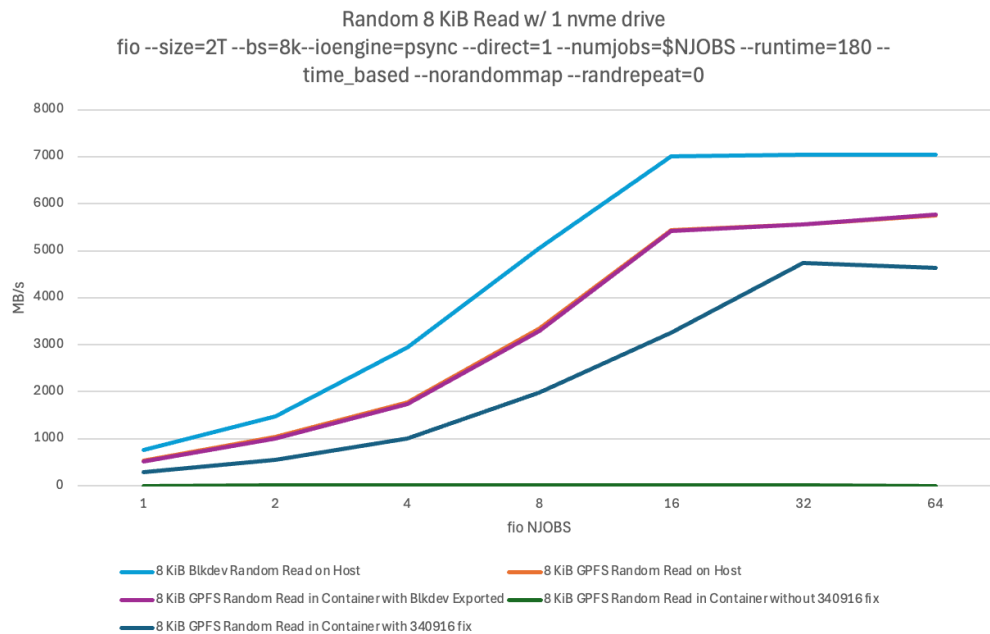
Container Performance Issue with Direct I/O and Improvement in 5.2.3

- Starting in RHEL 9, when IBM Storage Scale runs in an unprivileged container it will get an EPERM error when calling `gpfs_blkdev_get_by_dev()` to access any block device that is directly attached (either because they are local devices, or the devices are made to appear local, e.g., via an iSCSI or NVMeoF attachment).
- The exposure is really related to a cgroups permissions check and may also be seen in other cgroups enabled configurations, including VMs (outside of container)
- This EPERM error will cause Storage Scale to drop out of Direct I/O mode, even if all the other conditions for Direct I/O are met, and this overhead of switching in and out of Direct I/O will have a severe performance impact.
- A code fix in IBM Storage Scale 5.2.3 avoids dropping out of Direct I/O mode on getting this EPERM error, and instead retries the I/O through `mmfsd`, which should not be running in a cgroup that is not exposed to this permissions check (e.g., a privileged container in case of containerized CNSA deployments)
- Additional Storage Scale improvements are being investigated but the EPERM performance issue can be eliminated by giving the container/cgroup permissions to access Scale block devices, e.g.: <https://github.com/ocp-power-demos/power-dev-plugin>

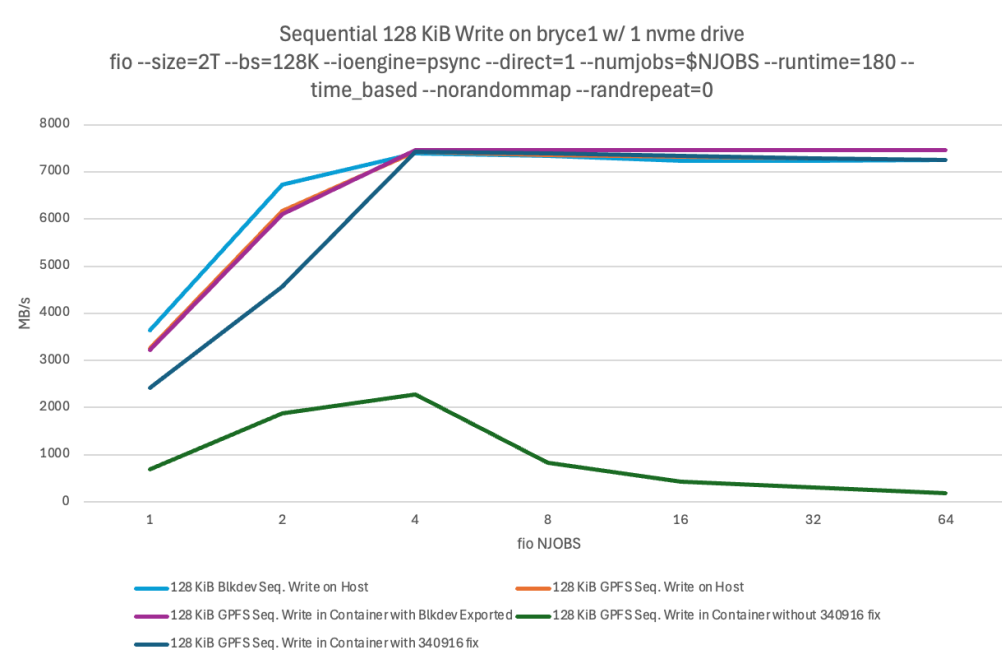
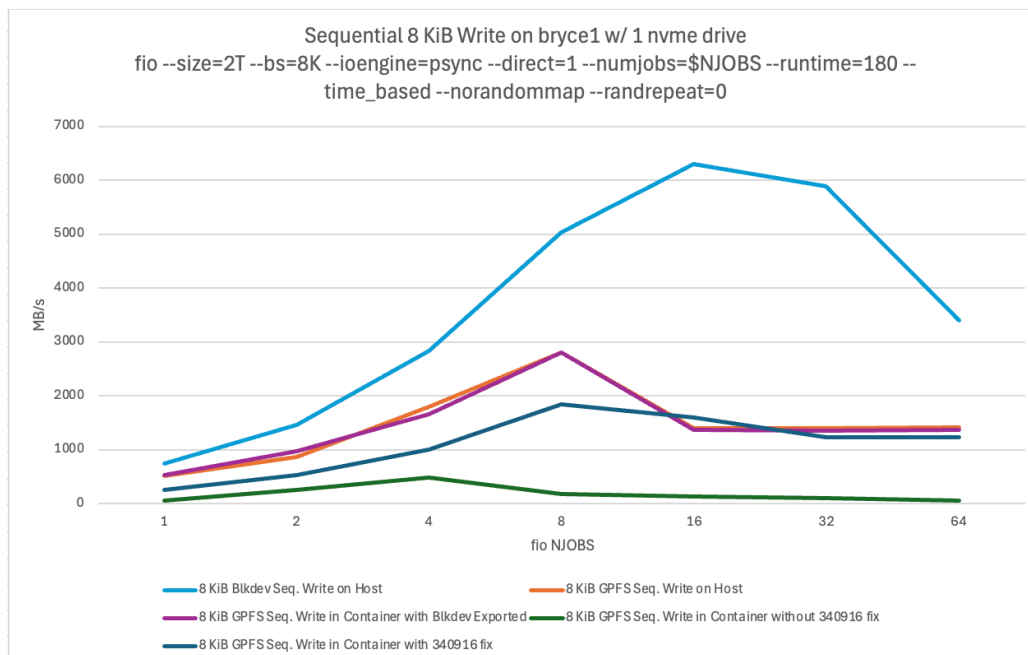
Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)



Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)



Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)



Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)

