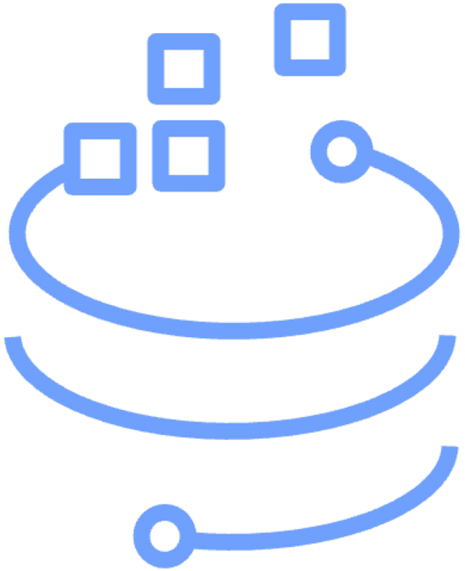
Storage Scale Performance Update

May 13, 2025

Storage Scale London User Group



Storage Scale

John Lewars (Storage Scale Development Performance Architect)
(some slides on ior-hard-read improvements are from Olaf Weiser)

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- ✓ **Data Acceleration Tier (DAT) Performance**
- Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9
- Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3
- Storage Scale 5.2.2 MMAP Write Performance Improvements
- Details on Storage Scale Configuration and Tuning Parameters

Data Acceleration Tier (DAT) +
Based on Asymmetric Data Replication*

Accelerates AI and Analytics by allowing data to be stored in two pools at once, enabling current use of:
(1) a Storage Scale RAID (formerly known as GPFS Native RAID, or GNR) encoded reliable copy
AND a performance copy, which could be:
(2a) a copy of data that is as close as possible to the computing client (where it's operated on) OR
(2b) a copy of the data that leverages the small I/O performance benefits of NVMeoF.

In the first release of asymmetric data replication we intend to support both the ESS/ISS DAT offering AND the use of local disks for a performance pool, though both may not be configured concurrently together.

NVIDIA DGX SuperPOD with IBM ESS/ISS NVIDIA.

* Asymmetric Data Replication has been referred to in past presentations by its IBM internal name, 'Ustore'.

Specifications	
GPU	8x NVIDIA H100 Tensor Core GPUs
GPU memory	640GB total
Performance	32 petaFLOPS FP8
NVIDIA® NVSwitch™	4x
System power usage	10.2kW max
CPU	Dual Intel® Xeon® Platinum 8480C Processors 112 Cores total, 2.00 GHz (Base), 3.80 GHz (Max Boost)
System memory	2TB
Networking	4x OSFP ports serving 8x single-port NVIDIA ConnectX-7 VPI > Up to 400Gb/s InfiniBand/Ethernet 2x dual-port QSFP112 NVIDIA ConnectX-7 VPI > Up to 400Gb/s InfiniBand/Ethernet
Management network	10Gb/s onboard NIC with RJ45 100Gb/s Ethernet NIC Host baseboard management controller (BMC) with RJ45
Storage	OS: 2x 1.92TB NVMe M.2
Internal storage:	8x 3.84TB NVMe U.2

Where Can Data Be In an Asymmetric Data Replication File System?

- Data exists in one of three states in any Asymmetric Data Replication file system.
- For the first release of the DAT (Data Acceleration Tier) Pool, we will support only the following options:

1) a single reliable pool replica only

2) a single performance pool replica only (aka/'burst buffer' when used for write workloads)

3) both a single reliable pool replica and a single performance pool replica, with both instances existing concurrently.

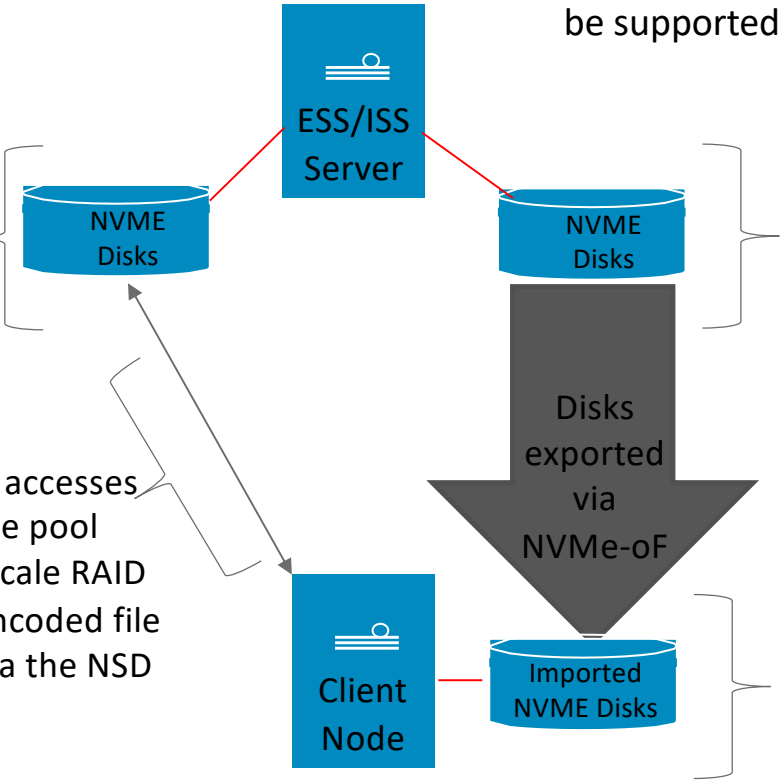
- The above three states can be managed using the policy feature, or the mmchattr command (these approaches, along with the Write Affinity Failure Group--WADFG-- feature, can also control the affinity of the performance copy in solutions that exploit local storage)

- Policies can automate movement of data between pools, and we intend to add additional enhancements to the 'burst buffer' mode, e.g., automatic copying of performance data to the reliable pool, after a file is closed (this will transition the data from option (2) to option (3) above)

Asymmetric Data Replication File System use case: Exploiting shared storage via NVMeoF

The first planned offering of this shared storage use case is called the **Data Acceleration Tool (DAT)**, planned to be supported on the ISS 6000.

A subset of server's NVME drives are used for the **reliable** pool (Storage Scale RAID encoded).

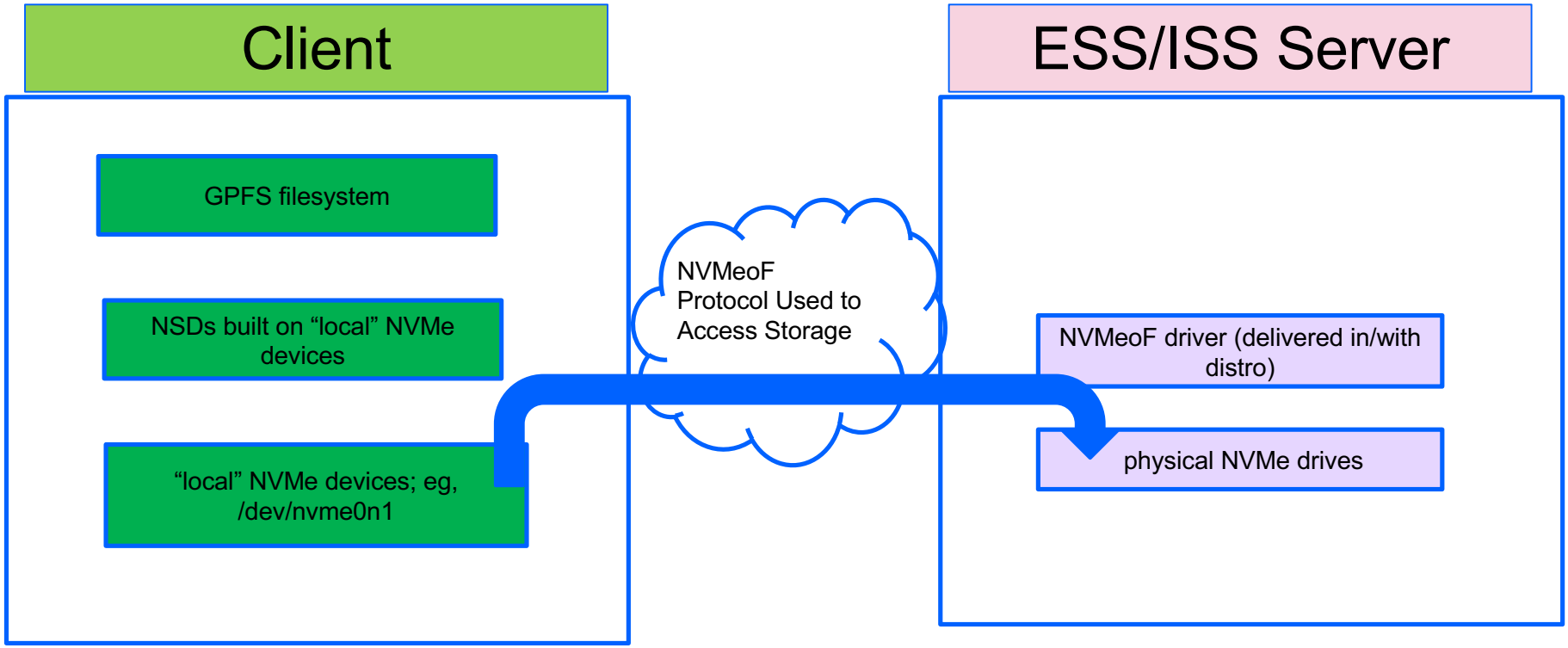


The complementary subset of the server's drives are used for the **performance** pool.

The client accesses the reliable pool (Storage Scale RAID erasure encoded file system) via the NSD protocol.

The Performance pool drives are exported to the client(s) via NVMeoF, appearing like local drives to the client (e.g /dev/nvmeXn1). A client accesses the drives via the local I/O path (not NSD protocol) and there is no erasure encoding or checksum provided by Storage Scale RAID.

Use of NVMeoF In Data Acceleration Tier (DAT) Solution for ISS 6000



The Data Acceleration Tier uses the NVMeoF protocol to share drives from the servers to the clients. The clients see all the servers’ drives used for the performance pool, sharing these drives, which basically implements a SAN solution over the network instead of using fibre channel cables.

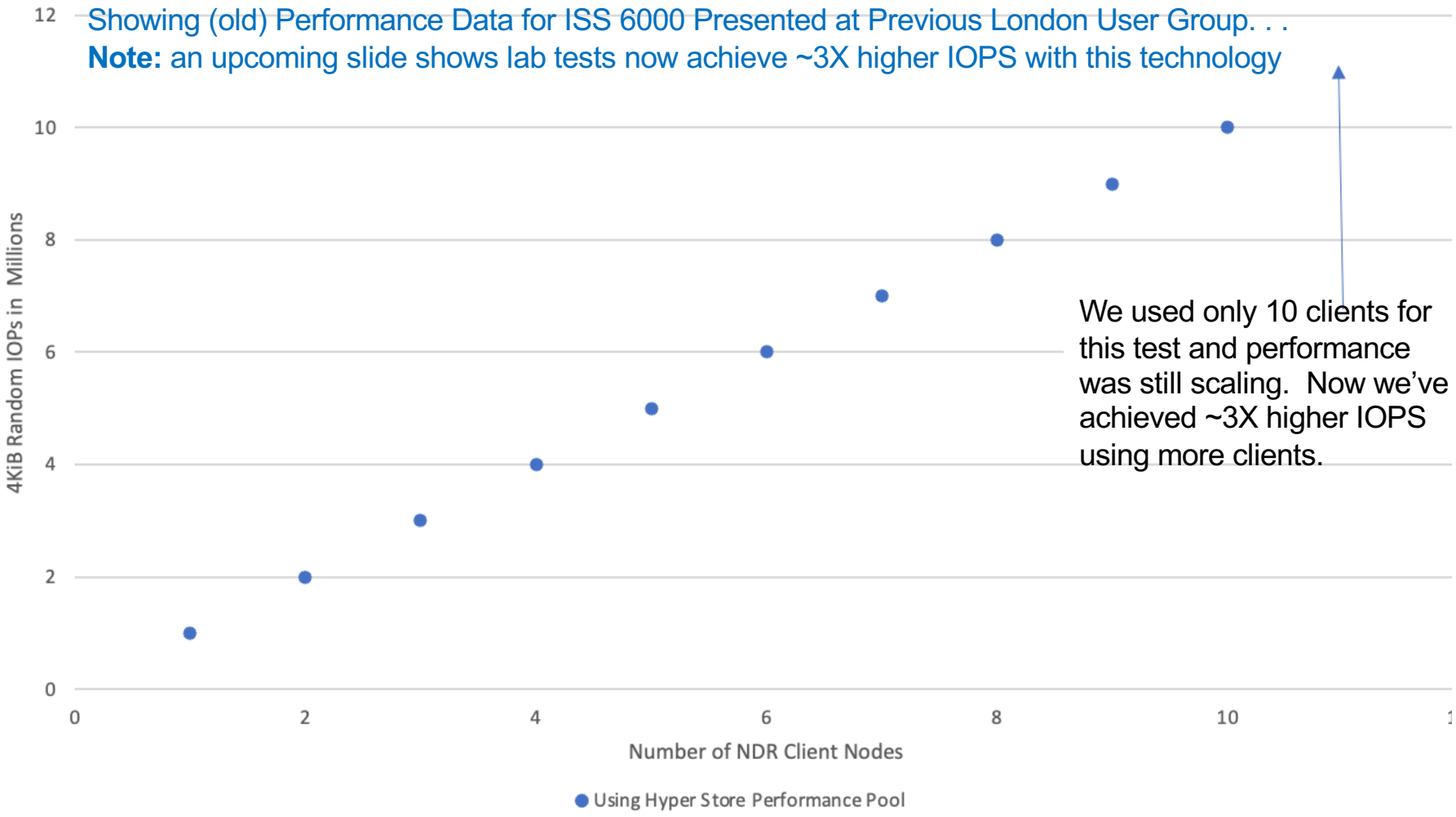
Configuration Details for Performance Measurements

InfiniBand Connected ESS/ISS 6000 Configuration Evaluated	
NVMe Drives	48x 3.84TB NVMe Tier-1 Flash (Gen 4)
Memory per canister	1536 GB (24x 64GB dimms)
IB adapters per canister	4x CX7 adapters, each with 2x 200Gb ports
NVMeoF Offload	NVMeoF Hardware Offload Enabled
Clients	10x clients each with 2 cpu chips/48 cores and 2x CX7 400Gb/s IB 1-port adapters
Switch	NDR 400Gib MQM9700 - Infiniband

4 KiB Random Read IOPS on 10 NDR clients and 1 SSS 6000
Default Config compared to Data Acceleration Tier

Showing (old) Performance Data for ISS 6000 Presented at Previous London User Group. . .

Note: an upcoming slide shows lab tests now achieve ~3X higher IOPS with this technology

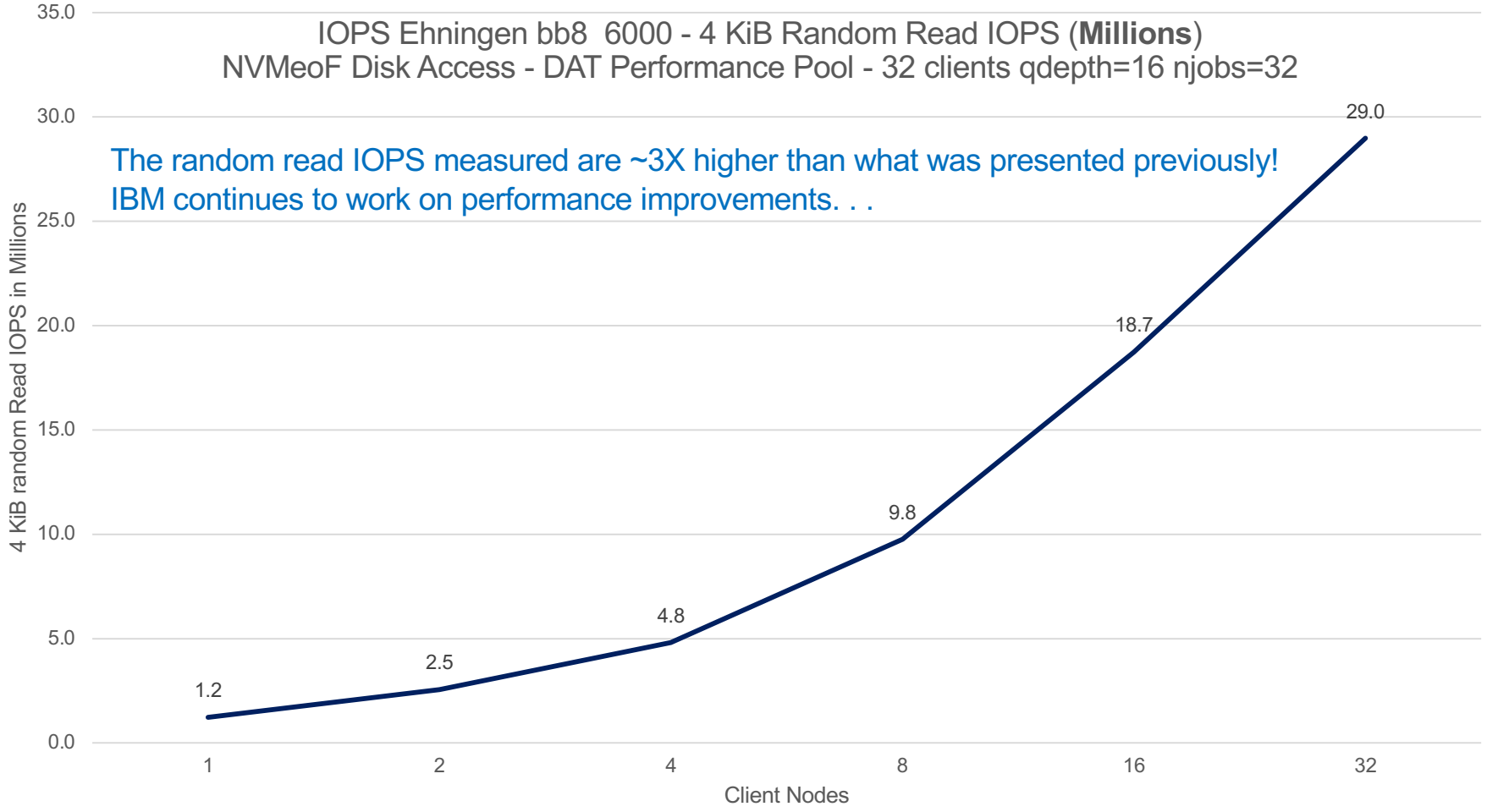


We used only 10 clients for this test and performance was still scaling. Now we've achieved ~3X higher IOPS using more clients.

Configuration Details for Performance Measurements

Ethernet Connected ISS 6000 Configuration Evaluated	
NVMe Drives	48x 3.84TB NVMe Tier-1 Flash(Gen 5)
Memory per canister	768GB (24x 32GB dimms)
IB adapters per canister	4x CX7 adapters, each with 2x 200Gb ports
NVMeoF Offload	NVMeoF Hardware Offload **Not** Enabled
Clients	32x clients each with 1 cpu chip/24 cores (AMD EPYC 9274F) 400 Gb/s CX7 RoCE (half nodes have 1 port adapters and the other half use 2 port adapters)
Switch	400Gib SN 5600 - Ethernet

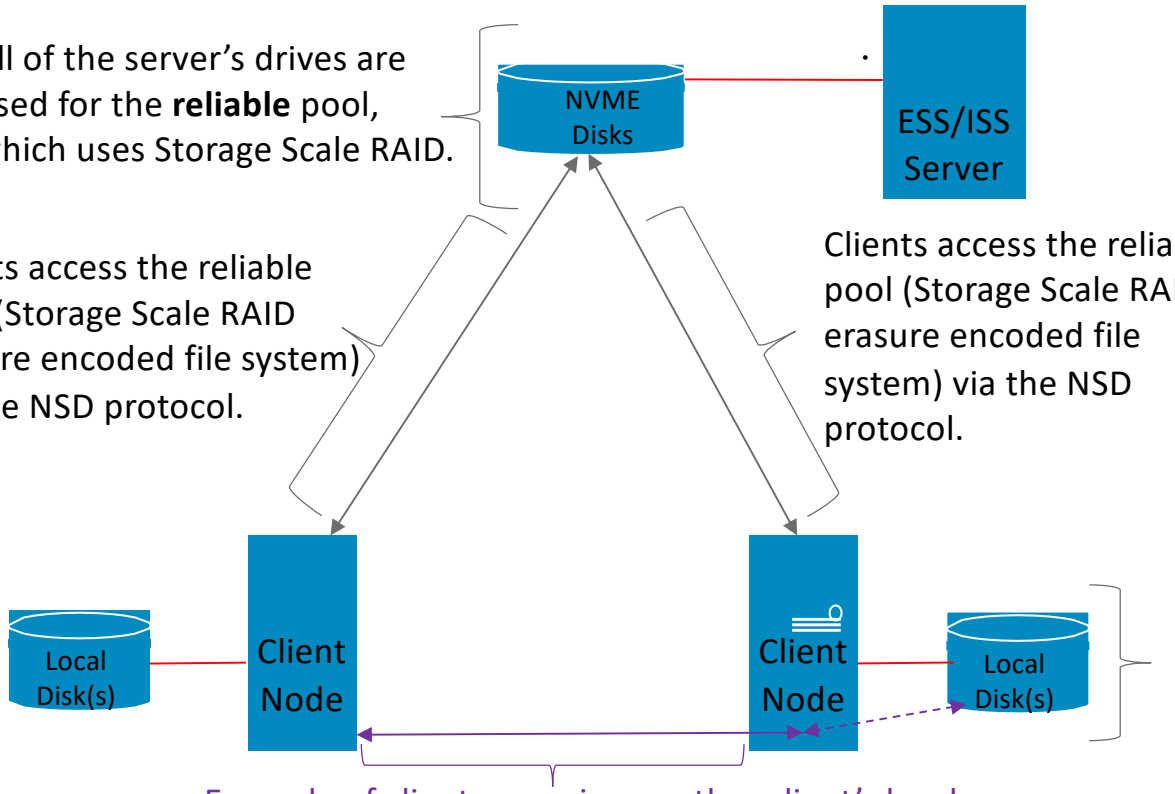
DAT 4 KiB Rand. Read on RoCE Configuration (3 min. runs)



Example iostat Data From 30M IOPS Run

```
05/12/2025 01:37:47 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.11    0.00   71.32    0.01    0.00   28.56

Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz     d/s     dkB/s   drqm/s
%drqm d_await dareq-sz     f/s f_await  aqu-sz  %util
md123            0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
md124            0.00      0.00     0.00   0.00    0.00     0.00    6.80     27.20     0.00   0.00    0.10     4.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.04
md125            0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
md126            0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
md127            0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
nvme0c0n1        0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
nvme0n1          0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00    0.00   0.00
nvme10c10n1 652815.70 2611262.80     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   63.38 100.01
nvme10n1    652772.30 2611089.20     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   62.93 100.00
nvme11c11n1 652854.60 2611418.40     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   62.92 100.01
nvme11n1    652881.50 2611526.00     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   62.55 100.01
{…]
nvme14c14n1 652639.90 2610560.00     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   63.09 100.01
nvme14n1    652635.90 2610543.60     0.00   0.00    0.10     4.00    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00
0.00    0.00     0.00    0.00    0.00   62.67 100.01
[…]
```

Asymmetric Data Replication Use Case: Exploiting local storage



The depicted **local** storage use case is not yet supported. This example shows how the solution might use an ESS/ISS server but there are no plans to limit the local solution to using ESS/ISS hardware.

Storage physically present on the client (not exported via NVMeoF) comprises the performance pool. Clients access this local storage directly or can access local storage from other clients via the NSD protocol.

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:

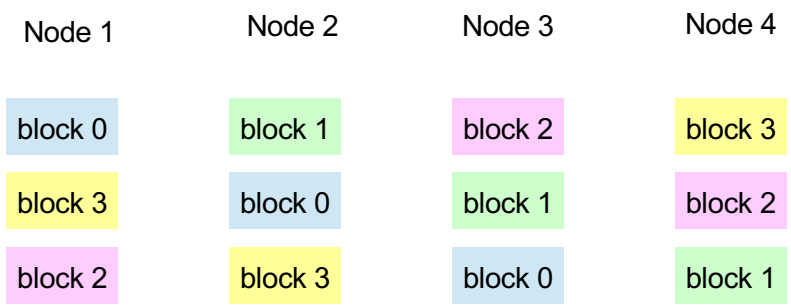
- SNC (Shared-nothing cluster)
- FPO (File Placement Optimizer: SNC with write-affinity)
- ECE (Erasure Code Edition)
- DAT (Data Acceleration Tier)
  Using Asymmetric Data Replication, DAT combines key features of these existing local storage solutions to create a filesystem and cache data in a persistent, non-redundant manner.

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

- LROC (Local Read-Only Cache)

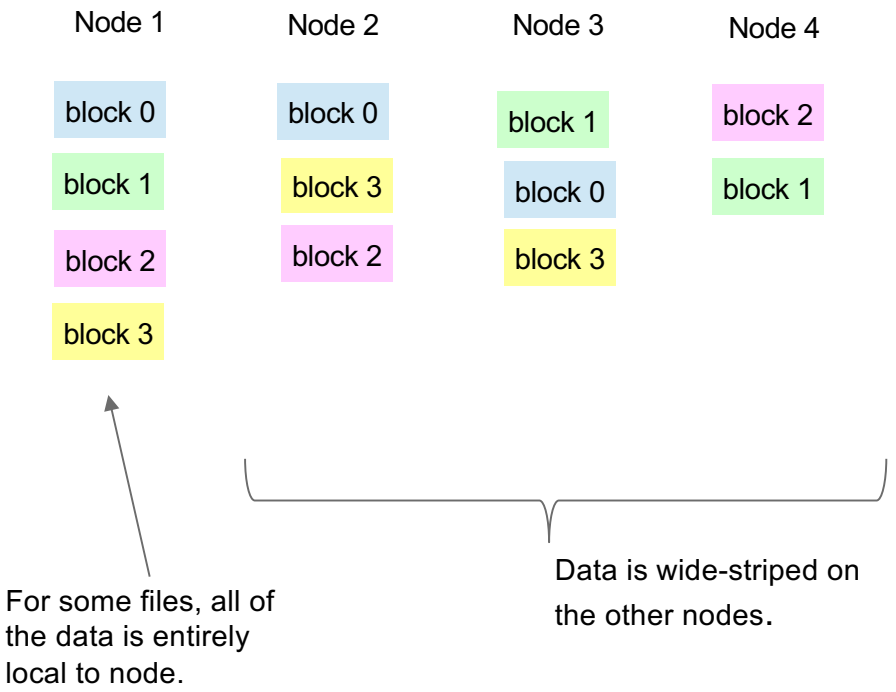
Workloads on local storage with SNC (Shared Nothing Cluster)

- 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 storage for replication overhead.



Workloads on local storage with FPO (File Placement Optimizer)

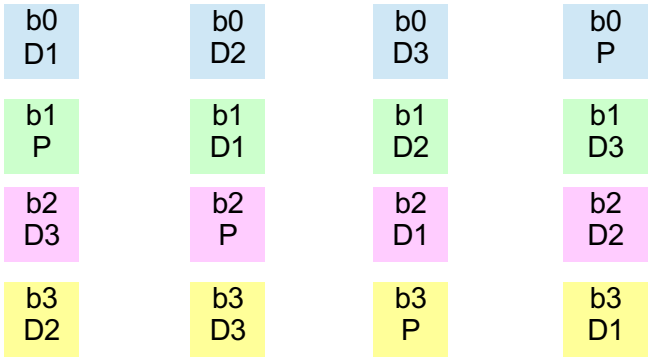
- Data access: Provides a method to affinitize data so a wider range of applications can exploit data locality; e.g., databases can ensure that one replica of a target fileset is entirely stored on a given node.
- Data reliability/availability: 3-way replication. Metadata scan needed after a failed node rejoins the cluster.
- Cluster resources: Each unit of user-consumable storage uses 3x units of storage for replication overhead.



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

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

- Data Access: Most data not local and there is no control over data placement (data chunks too small for schedulers like Hadoop to manage). 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.

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

Workloads on local storage with an Asymmetric Data Replication File System

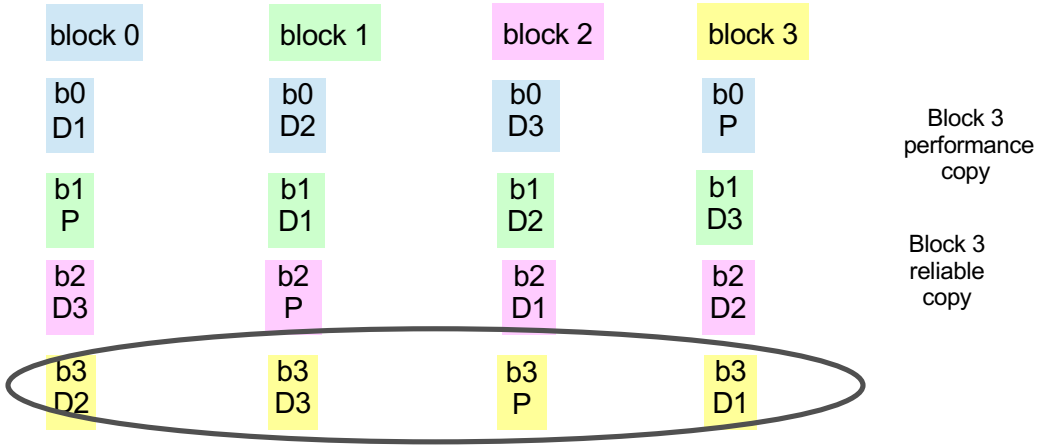
Maintain two copies of each data block in two different pools with “asymmetric replication”.

- “reliable copy”: protected by ECE erasure encoding
- “performance copy”: stored entirely (as one unit) on a single node, providing fast reads for workloads that leverage data locality

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

Data Access: The performance copy of data is entirely local to a client (as with FPO); the reliable copy is erasure-encoded.

Data Reliability/Availability:
The reliable copy is erasure-encoded. The long-term plan is to maintain the performance copy using “**lazy consistency**”, meaning any missed updates due to a disk being offline will be corrected during the next read after the disk is back online.
The team is currently working on changes to the direct I/O flow for DAT configurations to fully enable lazy consistency.



Cluster resources: Each unit of user-consumable storage uses 2.25x of storage resources, which is higher than 1.25x ECE but lower than 3x FPO.

Workloads on local storage with LROC

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

Asymmetric Data Replication has 2 advantages over LROC:

- LROC cache is potentially redundant because the same data can be cached on multiple nodes, but no more than a single copy of the data will be contained in the Asymmetric Data Replication file system performance pool.

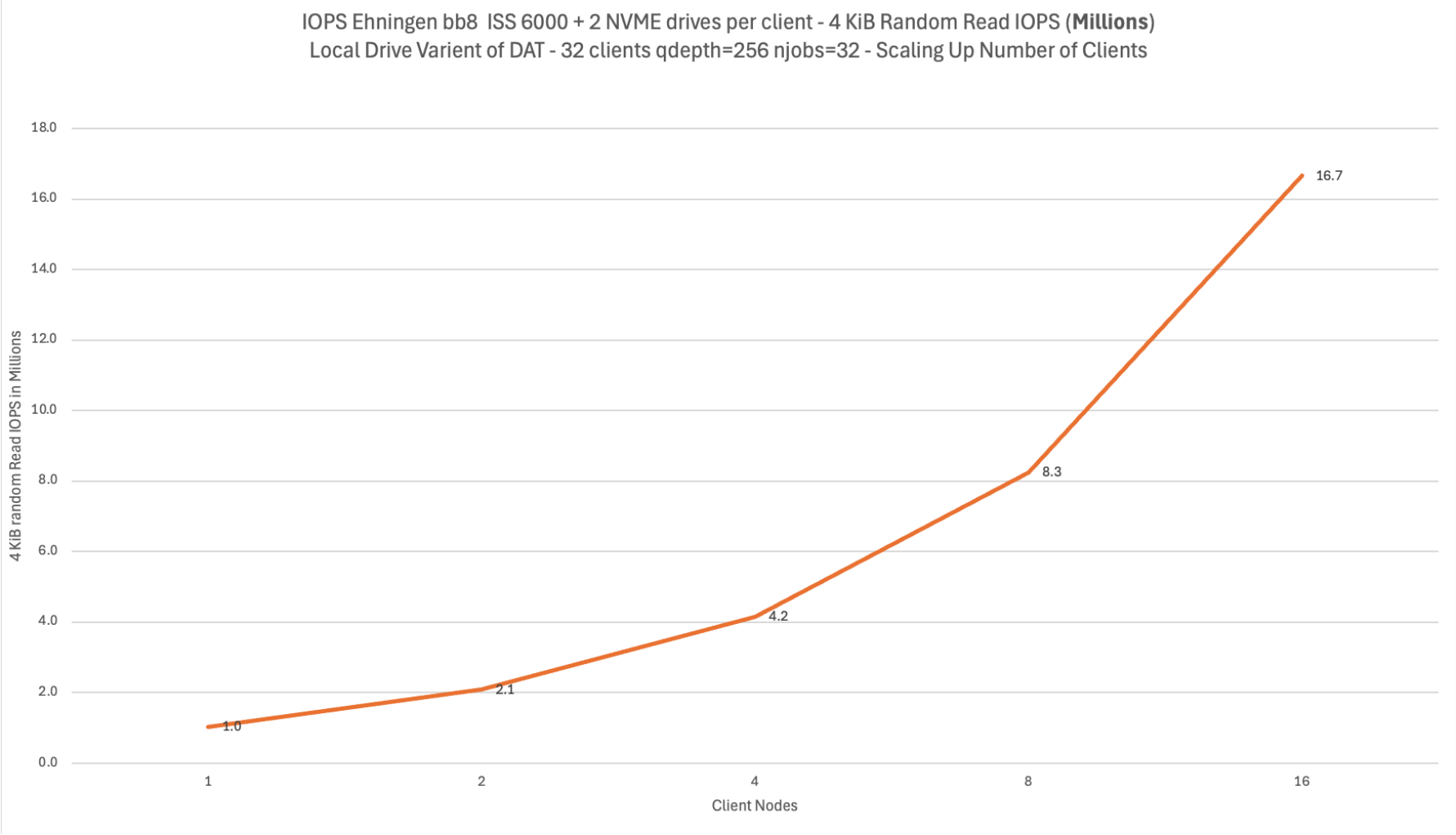
- LROC is not a persistent cache because a reload is needed after a failed node is back up. With Asymmetric Data Replication, the performance copy is persistent except when a drive goes down (in case of drive failures, the reliable copy is used until the performance copy is available and any missing updates are repaired).

- LROC is a true cache, which means it won't help Direct I/O workloads, while Asymmetric Data Replication provides a high-performance tier that can accelerate Direct I/O workloads,

Asymmetric Data Replication has an advantage over LROC:

- Multiple copies of the same data can be cached in LROC (like how the page pool works).

4 KiB Random Read IOPS (M) Performance with Local Drive Solution



Agenda

- Data Acceleration Tier (DAT) Performance
- ✓ **Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9**
- Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3
- Storage Scale 5.2.2 MMAP Write Performance Improvements
- Details on Storage Scale Configuration and Tuning Parameters

Container Performance Issue with Direct I/O Addressed in 5.2.3

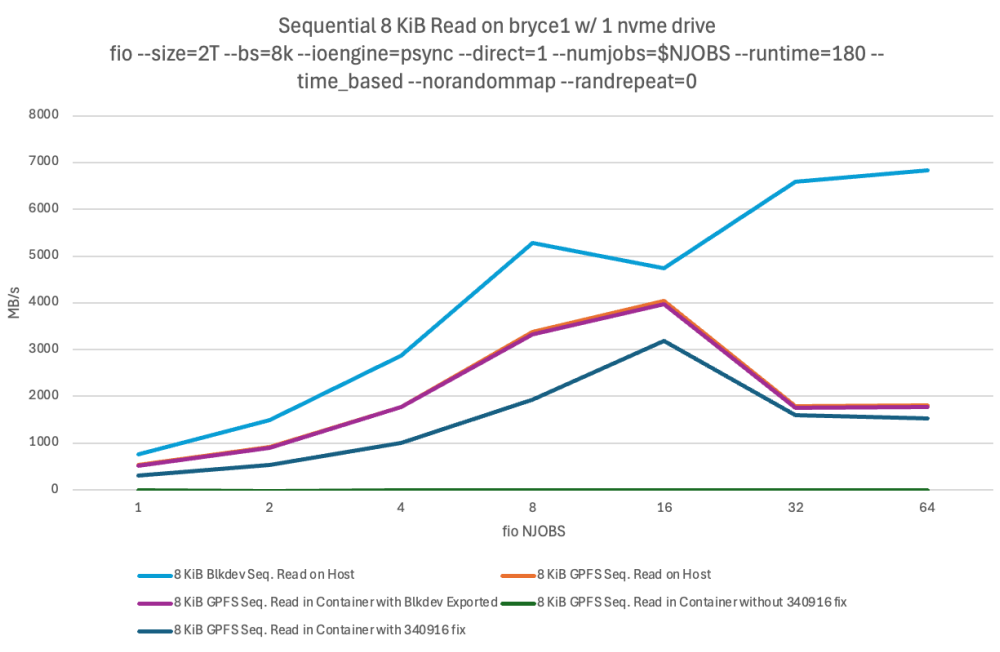
- Starting in RHEL 9, when IBM Storage Scale runs in an unprivileged container it will get an EPERM error when calling gpfs_blkdev_get_by_dev() to access any block device that is directly attached (either because they are local devices, or the devices are made to appear local, e.g., via an iSCSI or NVMeoF attachment).

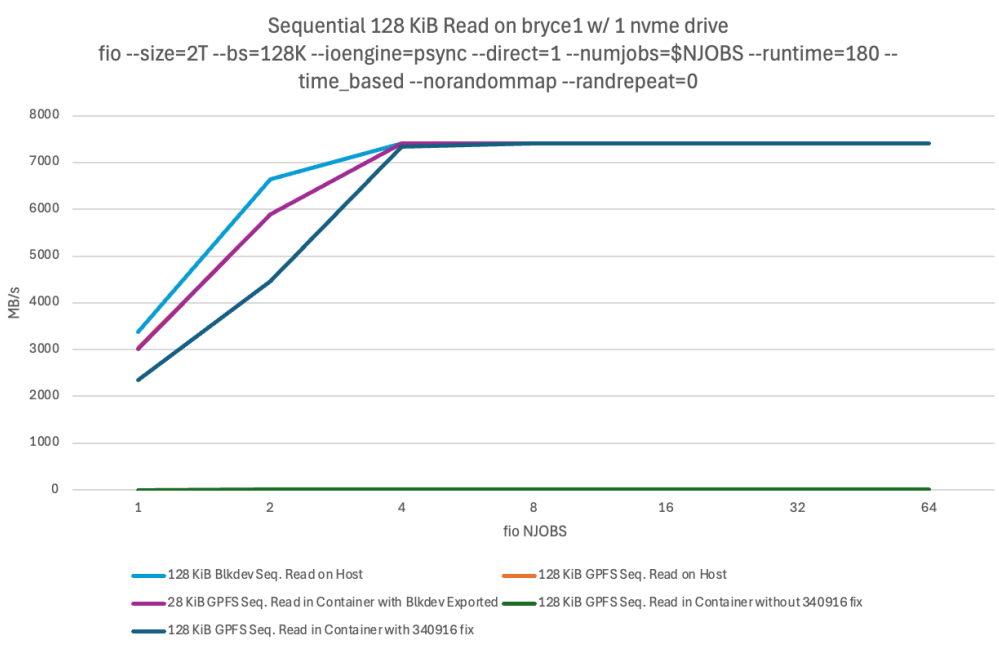
- The exposure is really related to a cgroups permissions check and may also be seen in other cgroups enabled configurations, including VMs (outside of container)

- 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.

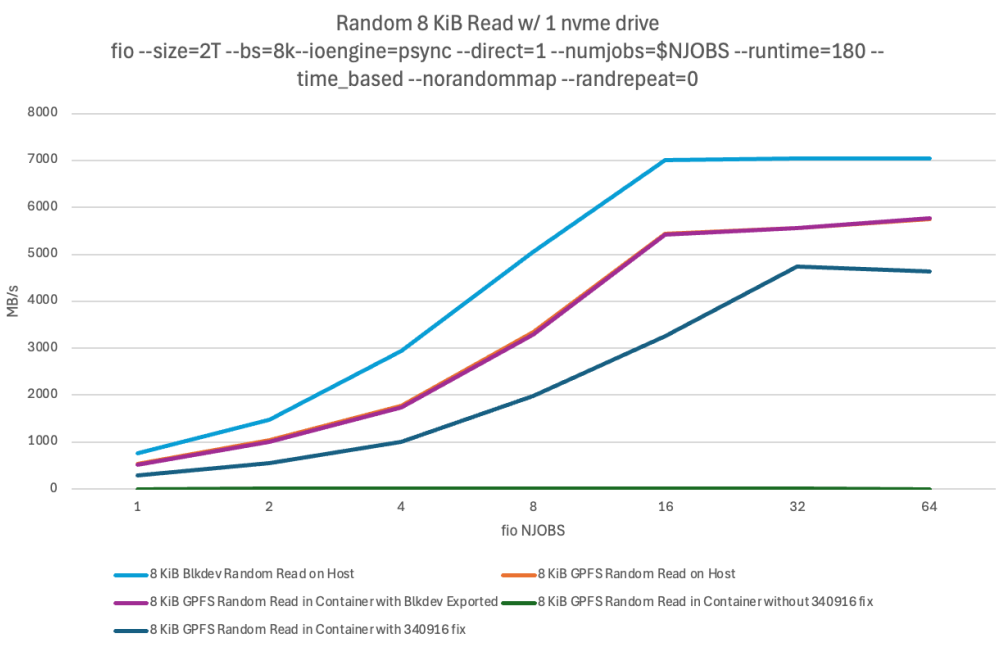
- A code fix in IBM Storage Scale 5.2.3 avoids dropping out of Direct I/O mode on getting this EPERM error, and instead retries the I/O through mmfsd, which should not be running in a cgroup that is not exposed to this permissions check (e.g., a privileged container in case of containerized CNSA deployments)

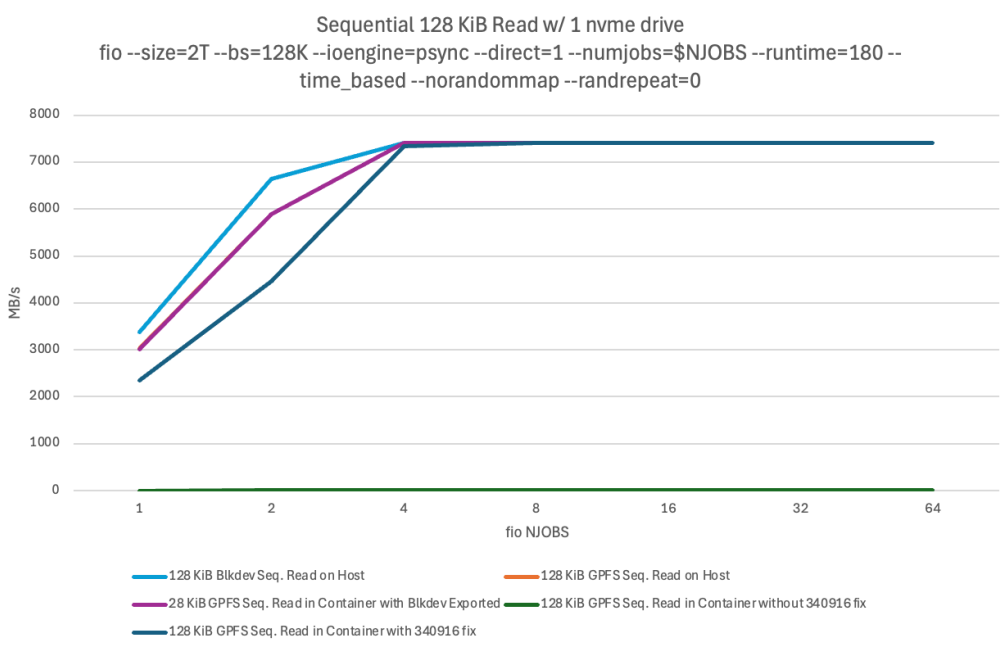
Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)



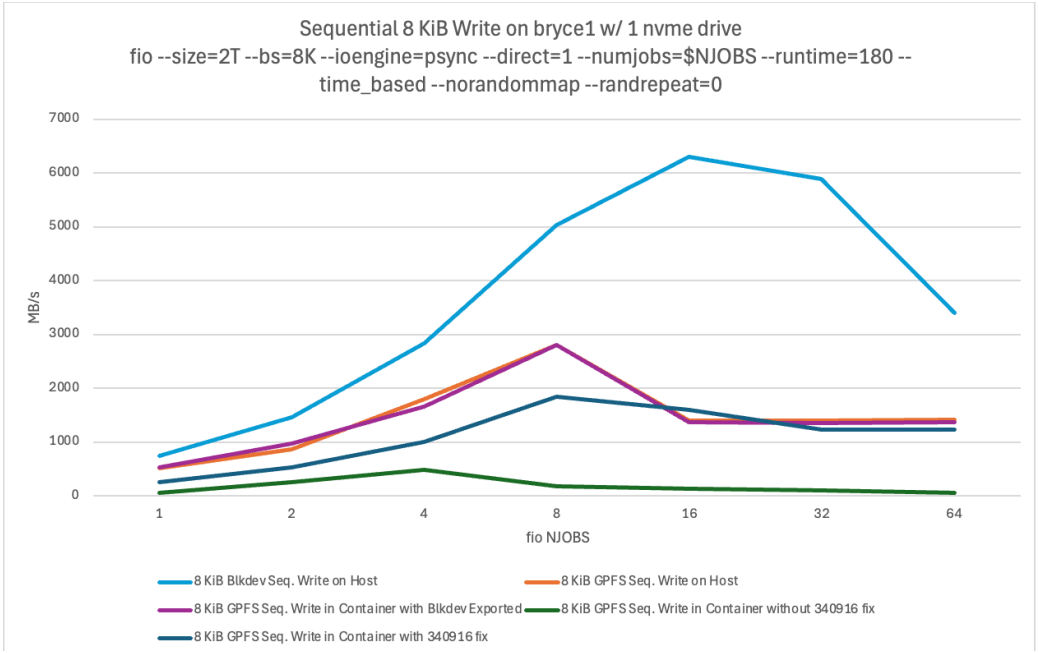


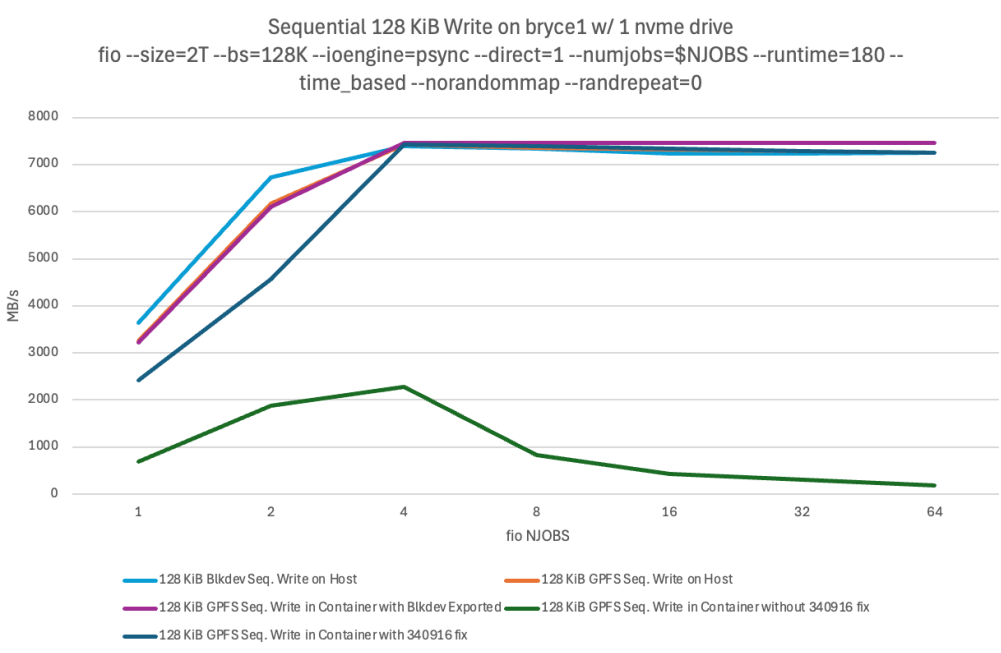
Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916



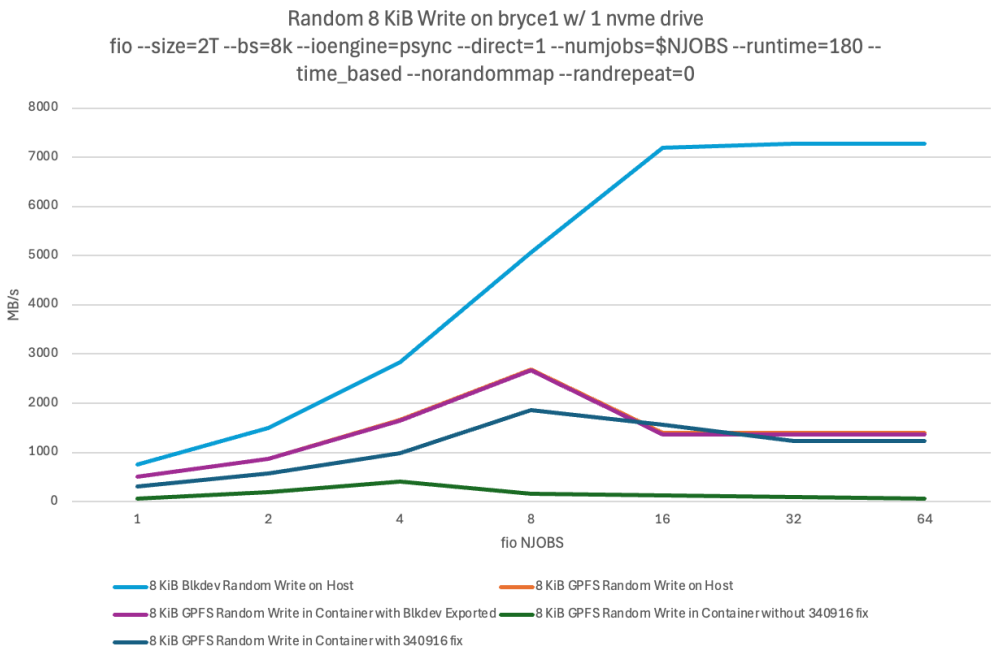


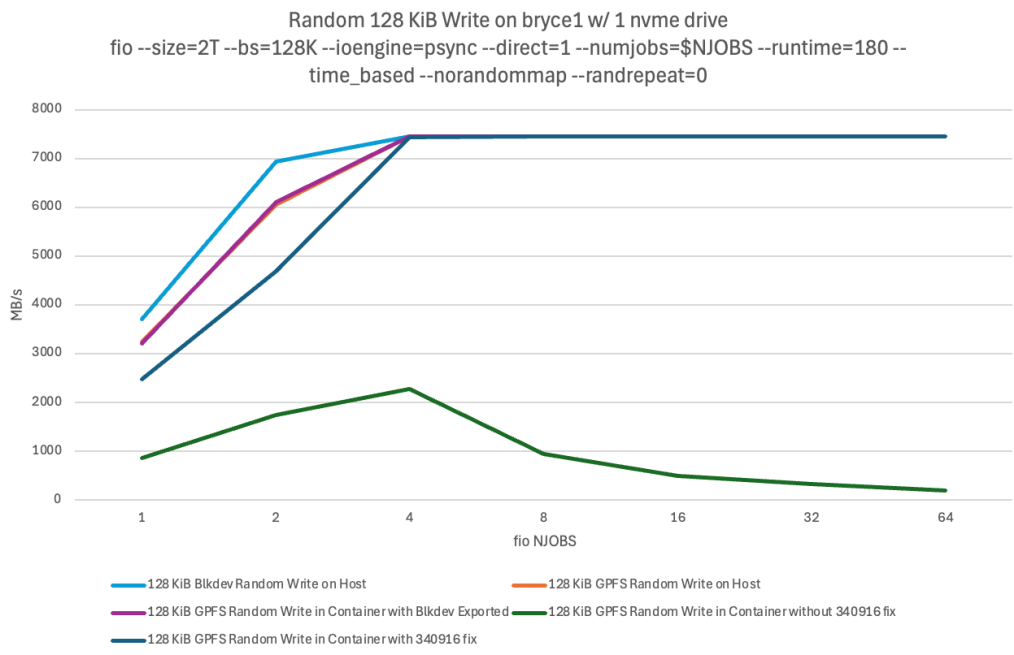
Performance Improvement in 5.2.3 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)





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





Agenda

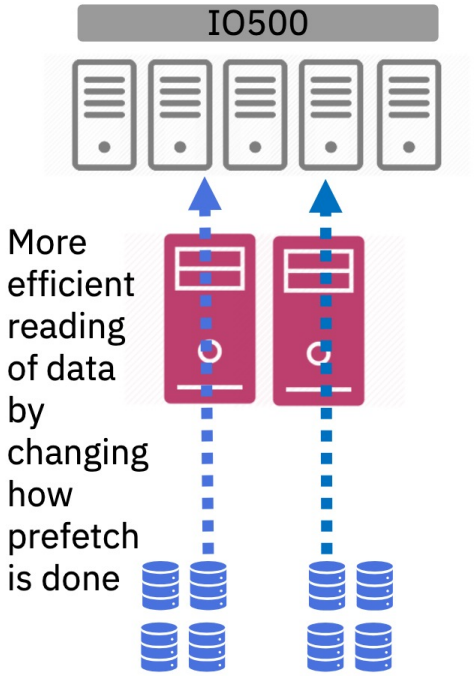
- Data Acceleration Tier (DAT) Performance
- Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9
- ✓ **Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3**
- Storage Scale 5.2.2 MMAP Write Performance Improvements
- Details on Storage Scale Configuration and Tuning Parameters

The ior-read-hard Benchmark Challenges and Action Plan

- Originally presented in an Storage Scale Expert talk in July 2022

- **What makes ior hard read so hard?**
  - Shared file reads currently causes aggressive prefetching leading to client nodes reading data that will not be consumed.
  - Small read request size limits efficiencies of network transfers if prefetch isn't done correctly.
  - The benchmark ensure that all tasks read data written by another task, which means there are token considerations (can be addressed by MPI hint to release tokens).
  - Like ior hard write, a 47008 byte read request size is used, which prevents Direct IO from being used.

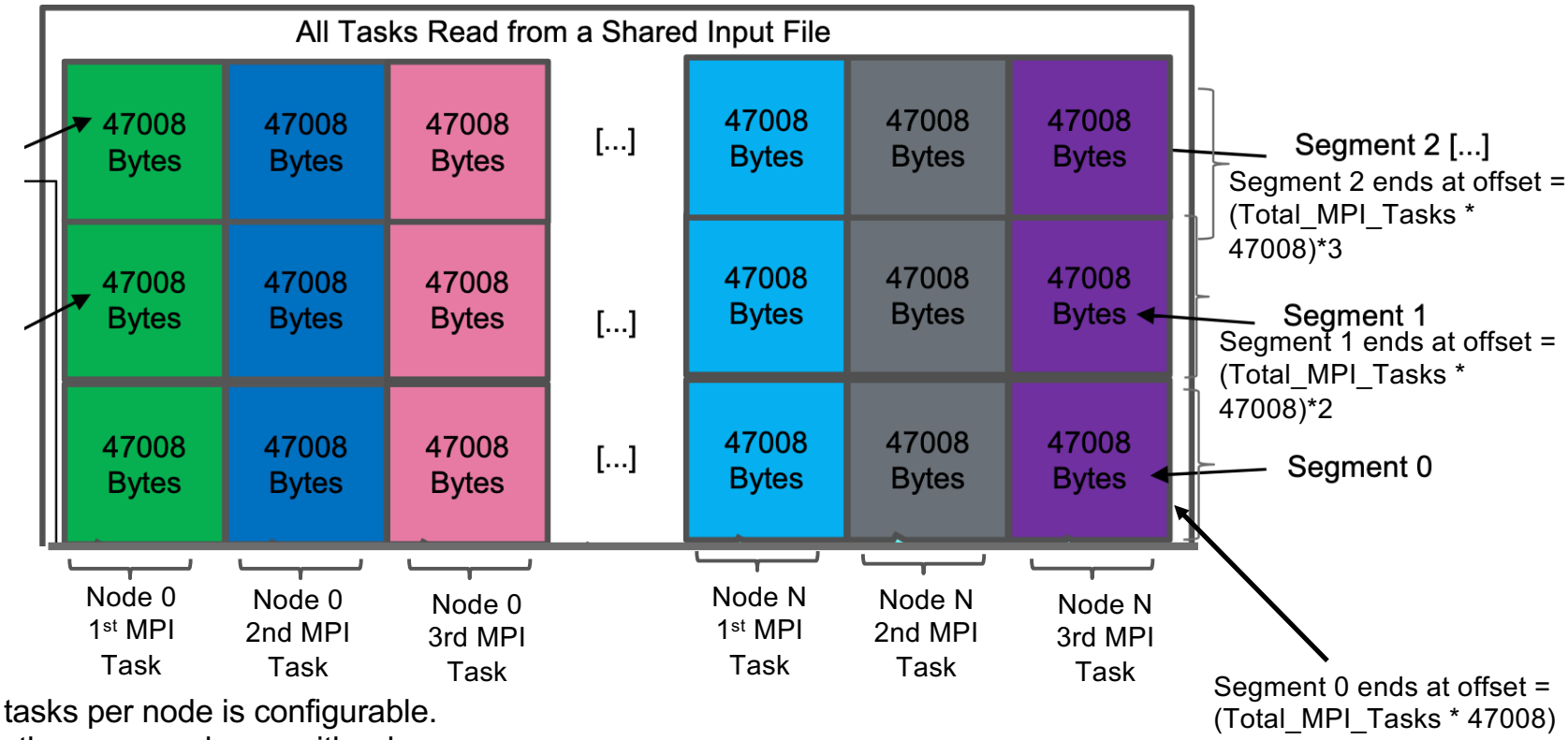
- **Action plan:**
  - Multiple design changes are being made to prefetch with optimizations controlled via fcntl hint. A fineGrainReadSharing hint has been added to IOR via this commit: https://github.com/hpc/ior/issues/390



Access Pattern for ior-hard-read

The ior-hard-read benchmark shifts the mapping of task IDs across nodes to avoid nodes from reading cached data they may have previously written.

47008 bytes written by the first MPI task running on node N. With the exception of the first node (which reads data written by Node N) all nodes read the data written by the previous node in the hostlist (so Node 1 reads data written by Node 0, Node 2 reads data written by node 1, etc.)



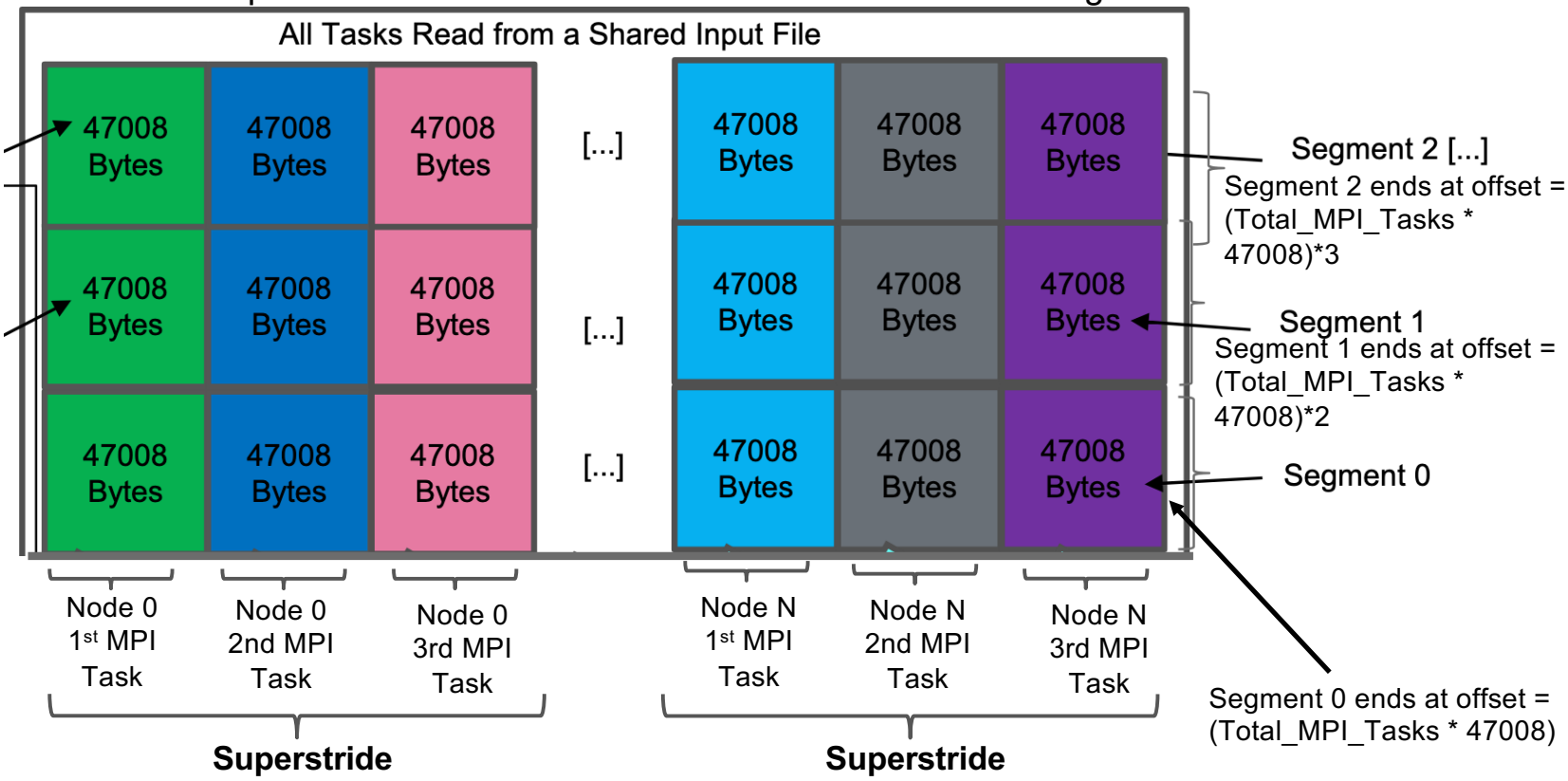
Number of tasks per node is configurable.
All tasks on the same node can either be:
1. Contiguous (in terms of MPI task IDs) as depicted in this slide (which gives the opportunity to coalesce prefetches) **or**
2. Round-robin'ed (e.g., with 4 tasks per node and 8 nodes, the first node has tasks 0, 8, 16, and 24, etc.)

ior-hard-read and Design Changes to Prefetch

The ior-hard-read benchmark shifts the mapping of task IDs across nodes to avoid nodes from reading cached data they may have previously written.

47008 bytes written by the first MPI task running on node N. With the exception of the first node (which reads data written by Node N) all nodes read the data written by the previous node in the hostlist (so Node 1 reads data written by Node 0, Node 2 reads data written by node 1, etc.)

- In Storage Scale 5.2.2, when using the fine-grain-read-sharing hint, prefetching becomes aware of the interaction between multiple instances/threads on the same node doing strided reads. The performance of this feature is enhanced in the Storage Scale 5.2.3 release



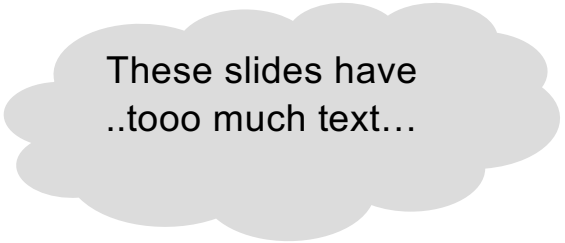
Superstride prefetching coalesce prefetches across instances doing strided reads, improving efficiency.

MPI task/job mapping When Using Open-MPI

Specifying Host Nodes https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php

Host nodes can be identified on the mpirun command line with the -host option or in a hostfile.

For example,
mpirun -H aa,aa,bb ./a.out
launches two processes on node aa and one on bb.



Or, consider the hostfile
% cat myhostfile
aa slots=2
bb slots=2
cc slots=2
Here, we list both the host names (aa, bb, and cc) but also how many slots there are for each.
mpirun -hostfile myhostfile ./a.out
will launch two processes on each of the three nodes.
mpirun -hostfile myhostfile -host aa ./a.out
will launch two processes, both on node aa.
mpirun -hostfile myhostfile -host dd ./a.out
will find no hosts to run on and abort with an error. That is, the specified host dd is not in the specified hostfile.
When running under resource managers (e.g., SLURM, Torque, etc.), Open MPI will obtain both the hostnames and the number of slots directly from the resource manger.

MPI Task/Job Mapping
How to Allocate MPI Tasks to Achieve the Best Strided Read Performance

- We can use a simple program to show how mpi allocate task IDs:

```
// ------------------------------------------------- MAIN(z) -----------------------
int main(int argc,char *argv[]){

//time
 struct timespec start, end;
// hostname
char hostname[256];
struct hostent *host_entry;
int hostint;
hostint = gethostname(hostname, sizeof(hostname));

// MPI stuff
MPI_Init( &argc, &argv );
int rank, mpitask;

MPI_Comm_rank( MPI_COMM_WORLD, &rank );                    // thats me
MPI_Comm_size( MPI_COMM_WORLD, &mpitask );                 // total job numbers


printf("[%s]: mpitask  [%d] rank [%d] \n",hostname, mpitask, rank );


MPI_Barrier(MPI_COMM_WORLD);
// master rank calcluates the time/bw
}
```

MPI Task/Job Mapping

How to Allocate MPI Tasks to Achieve the Best Strided Read Performance

mpirun -H aa,bb -np 8 ./a.out
launches 8 processes. Since only two hosts are specified, after the first two processes are mapped,
  one to aa and one to bb,
the remaining processes oversubscribe the specified hosts.

[root@fscc-sr665-8 ~]# mpiexec -n 10 -hostfile hostlist /gpfs/bb1nvme/a.out | sort -nk 1
[fscc-sr665-11]: mpitask [10] rank [0]
[fscc-sr665-11]: mpitask [10] rank [5]
[fscc-sr665-12]: mpitask [10] rank [1]
[fscc-sr665-12]: mpitask [10] rank [6]
[fscc-sr665-13]: mpitask [10] rank [2]
[fscc-sr665-13]: mpitask [10] rank [7]
[fscc-sr665-14]: mpitask [10] rank [3]
[fscc-sr665-14]: mpitask [10] rank [8]
[fscc-sr665-15]: mpitask [10] rank [4]
[fscc-sr665-15]: mpitask [10] rank [9]

MPI Task/Job Mapping
How to Allocate MPI Tasks to Achieve the Best Strided Read Performance

mpirun -H aa,bb -np 8 ./a.out
launches 8 processes. Since only two hosts are specified, after the first two processes are mapped,
  one to aa and one to bb,
the remaining processes oversubscribe the specified hosts.

```
[root@fscc-sr665-8 ~]# mpiexec -n 10 -hostfile hostlist /gpfs/bb1nvme/a.out | sort -nk 1
[fscc-sr665-11]: mpitask  [10] rank [0]
[fscc-sr665-11]: mpitask  [10] rank [5]
[fscc-sr665-12]: mpitask  [10] rank [1]
[fscc-sr665-12]: mpitask  [10] rank [6]
[fscc-sr665-13]: mpitask  [10] rank [2]
[fscc-sr665-13]: mpitask  [10] rank [7]
[fscc-sr665-14]: mpitask  [10] rank [3]
[fscc-sr665-14]: mpitask  [10] rank [8]
[fscc-sr665-15]: mpitask  [10] rank [4]
[fscc-sr665-15]: mpitask  [10] rank [9]
```

```
[root@fscc-sr665-8 ~]# mpiexec -n 10 -ppn 2 -hostfile hostlist /gpfs/bb1nvme/a.out | sort -nk 1
[fscc-sr665-11]: mpitask  [10] rank [0]
[fscc-sr665-11]: mpitask  [10] rank [1]
[fscc-sr665-12]: mpitask  [10] rank [2]
[fscc-sr665-12]: mpitask  [10] rank [3]
[fscc-sr665-13]: mpitask  [10] rank [4]
[fscc-sr665-13]: mpitask  [10] rank [5]
[fscc-sr665-14]: mpitask  [10] rank [6]
[fscc-sr665-14]: mpitask  [10] rank [7]
[fscc-sr665-15]: mpitask  [10] rank [8]
[fscc-sr665-15]: mpitask  [10] rank [9]
[root@fscc-sr665-8 ~]#
```

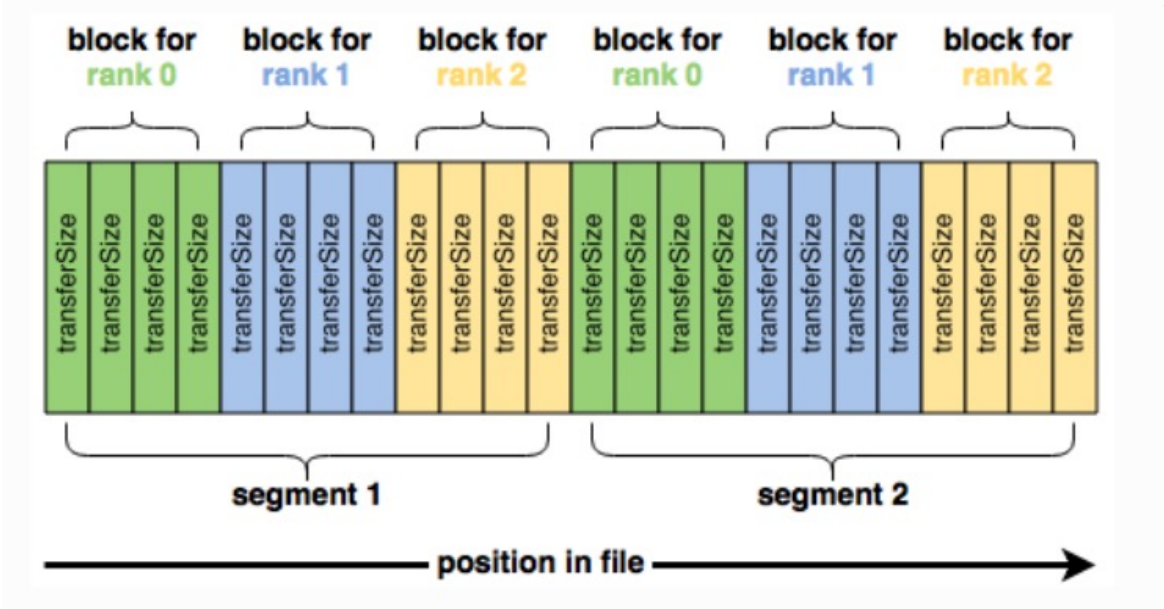
Storage Scale IOR Task/Job Mapping w/MPICH –Bringing the Two Patterns Together

default mapping

[fscc-sr665-11]: mpitask [10] rank [0]
[fscc-sr665-11]: mpitask [10] rank [5]
[fscc-sr665-12]: mpitask [10] rank [1]
[fscc-sr665-12]: mpitask [10] rank [6]
[fscc-sr665-13]: mpitask [10] rank [2]
[fscc-sr665-13]: mpitask [10] rank [7]
[fscc-sr665-14]: mpitask [10] rank [3]
[fscc-sr665-14]: mpitask [10] rank [8]
[fscc-sr665-15]: mpitask [10] rank [4]
[fscc-sr665-15]: mpitask [10] rank [9]

-ppn mapping

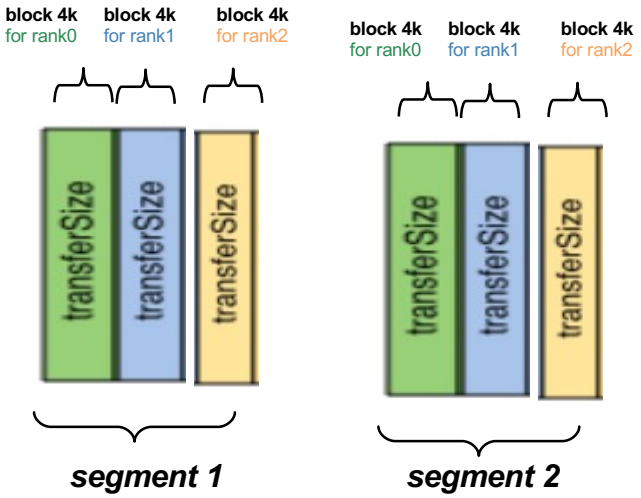
[fscc-sr665-11]: mpitask [10] rank [0]
[fscc-sr665-11]: mpitask [10] rank [1]
[fscc-sr665-12]: mpitask [10] rank [2]
[fscc-sr665-12]: mpitask [10] rank [3]
[fscc-sr665-13]: mpitask [10] rank [4]
[fscc-sr665-13]: mpitask [10] rank [5]
[fscc-sr665-14]: mpitask [10] rank [6]
[fscc-sr665-14]: mpitask [10] rank [7]
[fscc-sr665-15]: mpitask [10] rank [8]
[fscc-sr665-15]: mpitask [10] rank [9]



IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (1/7)

mpiexec -n 320 $PATH/ior -C -Q1 -g -t 4k -b 4k -s 10000000

one shared file




```
task offset          : 1
nodes                : 16
tasks                : 320
clients per node     : 20
memoryBuffer         : CPU
dataAccess           : CPU
GPUDirect            : 0
repetitions          : 1
xfersize             : 4096 bytes
blocksize            : 4096 bytes
aggregate filesize   : 11.92 TiB
```

SegmentSize= total task * blocksize

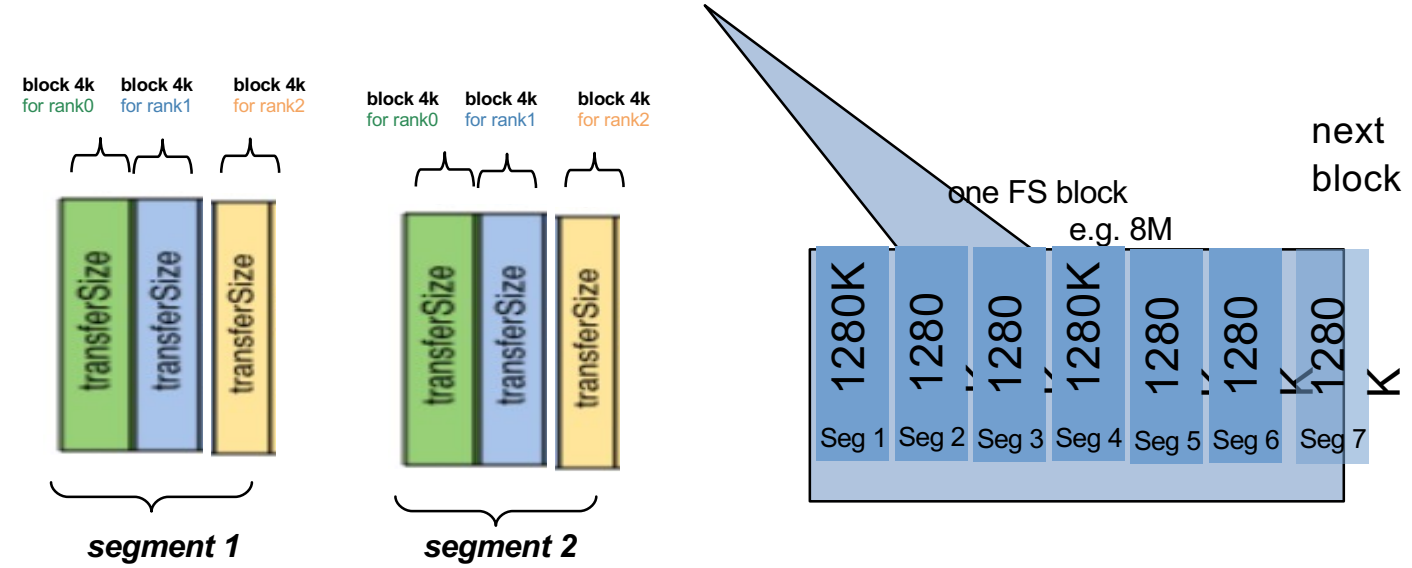
segSize= 320 * 4k = 1280 K

1280K * 10000000 = 11.92 TB

IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (2/7)

mpiexec -n 320 $PATH/ior -C -Q1 -g -t 4k -b 4k -s 10000000

one shared file



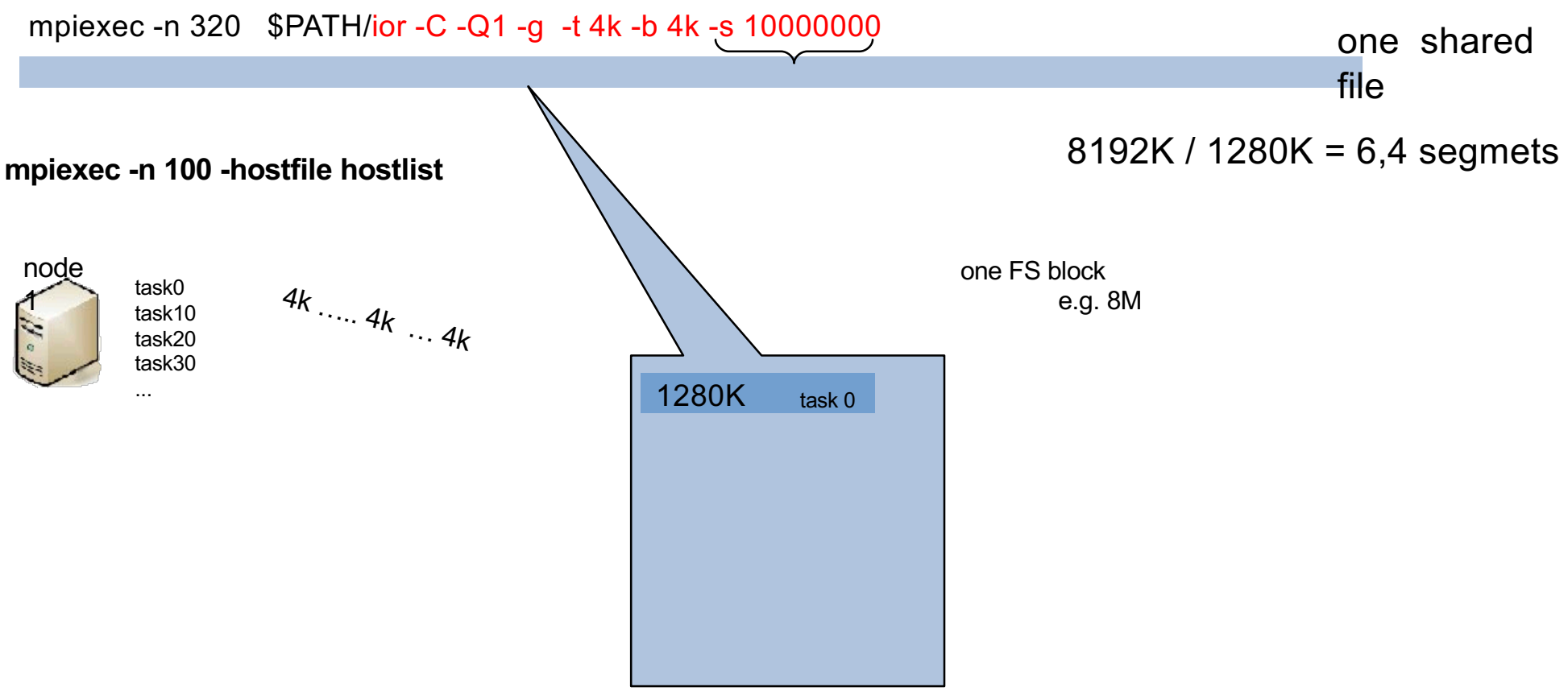

```
task offset          : 1
nodes                : 16
tasks                : 320
clients per node     : 20
memoryBuffer         : CPU
dataAccess           : CPU
GPUDirect            : 0
repetitions          : 1
xfersize             : 4096 bytes
blocksize            : 4096 bytes
aggregate filesize   : 11.92 TiB
```

SegmentSize= total task * blocksize

segSize= 320 * 4k = 1280 K

1280K * 10000000 = 11.92 TB

IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (3/7)



IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (4/7)

mpiexec -n 320 $PATH/ior -C -Q1 -g -t 4k -b 4k -s 10000000

one shared file

8192K / 1280K = 6,4 segmets

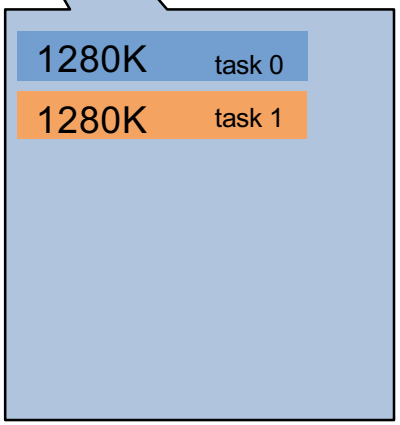
mpiexec -n 100 -hostfile hostlist

task0
task10
task20
task30
...

4k ….. 4k … 4k

task1
task11
task21
task31
...

4k ….. 4k … 4k



one FS block
e.g. 8M

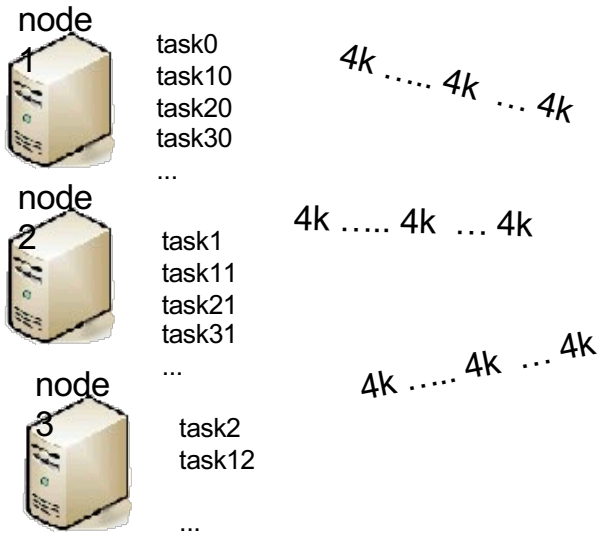
IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (5/7)

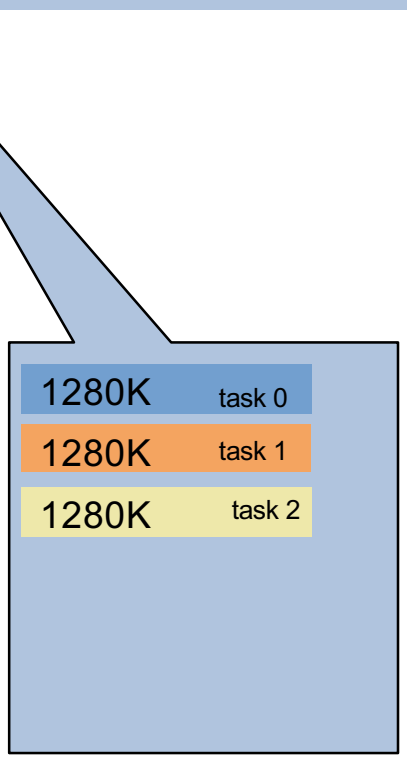
mpiexec -n 320 $PATH/ior -C -Q1 -g -t 4k -b 4k -s 10000000

one shared file

8192K / 1280K = 6,4 segmets

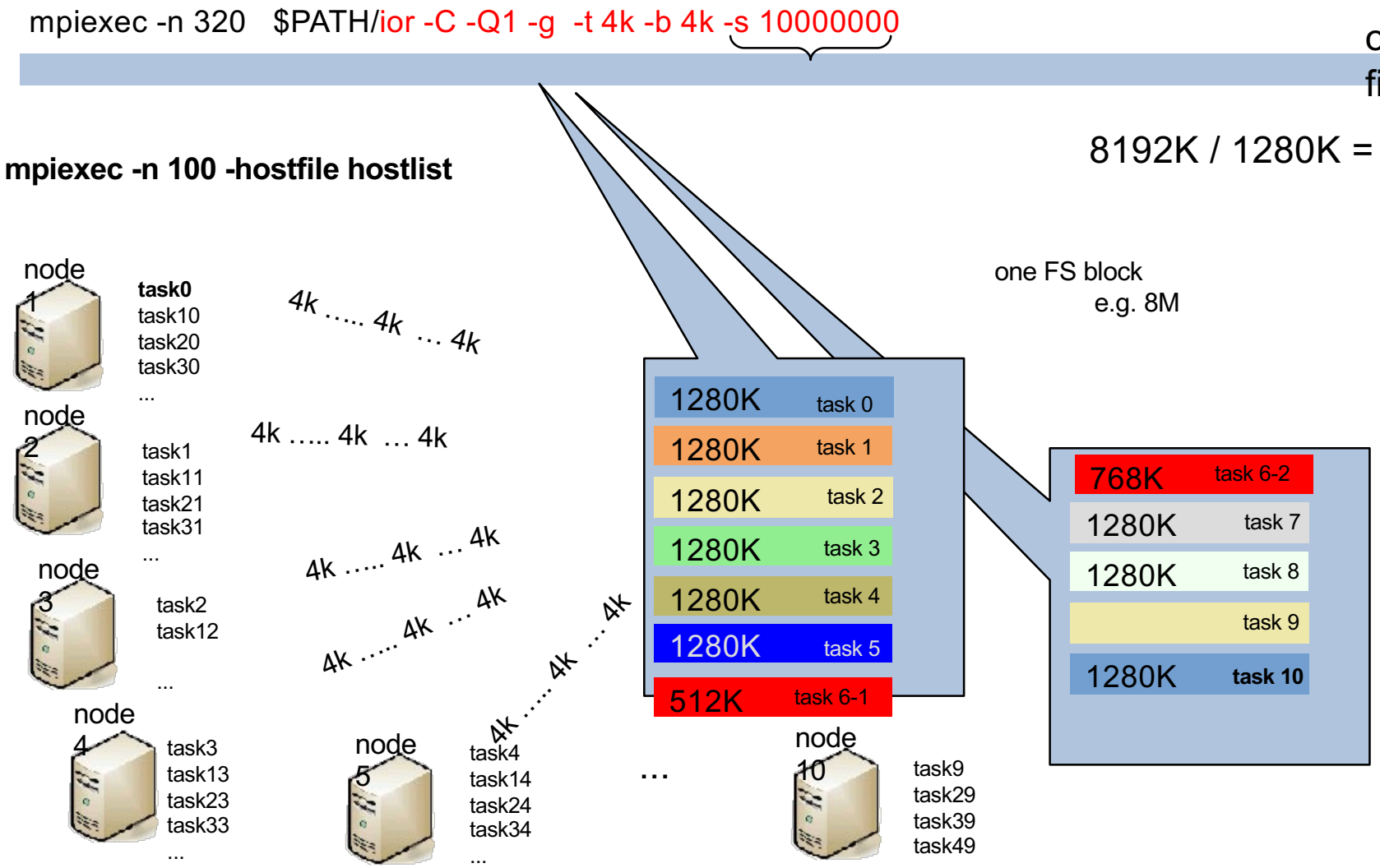
mpiexec -n 100 -hostfile hostlist



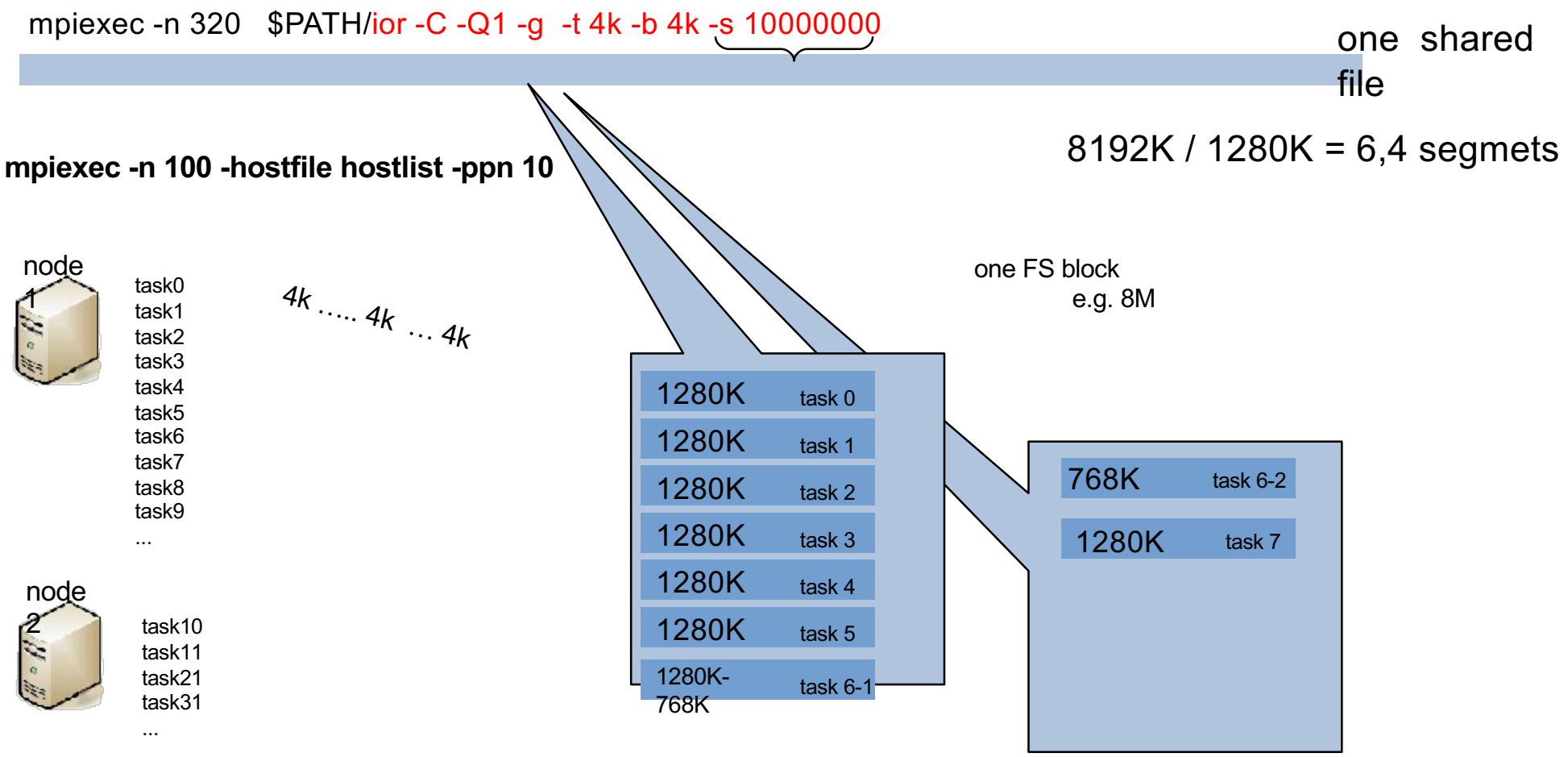


one FS block
e.g. 8M

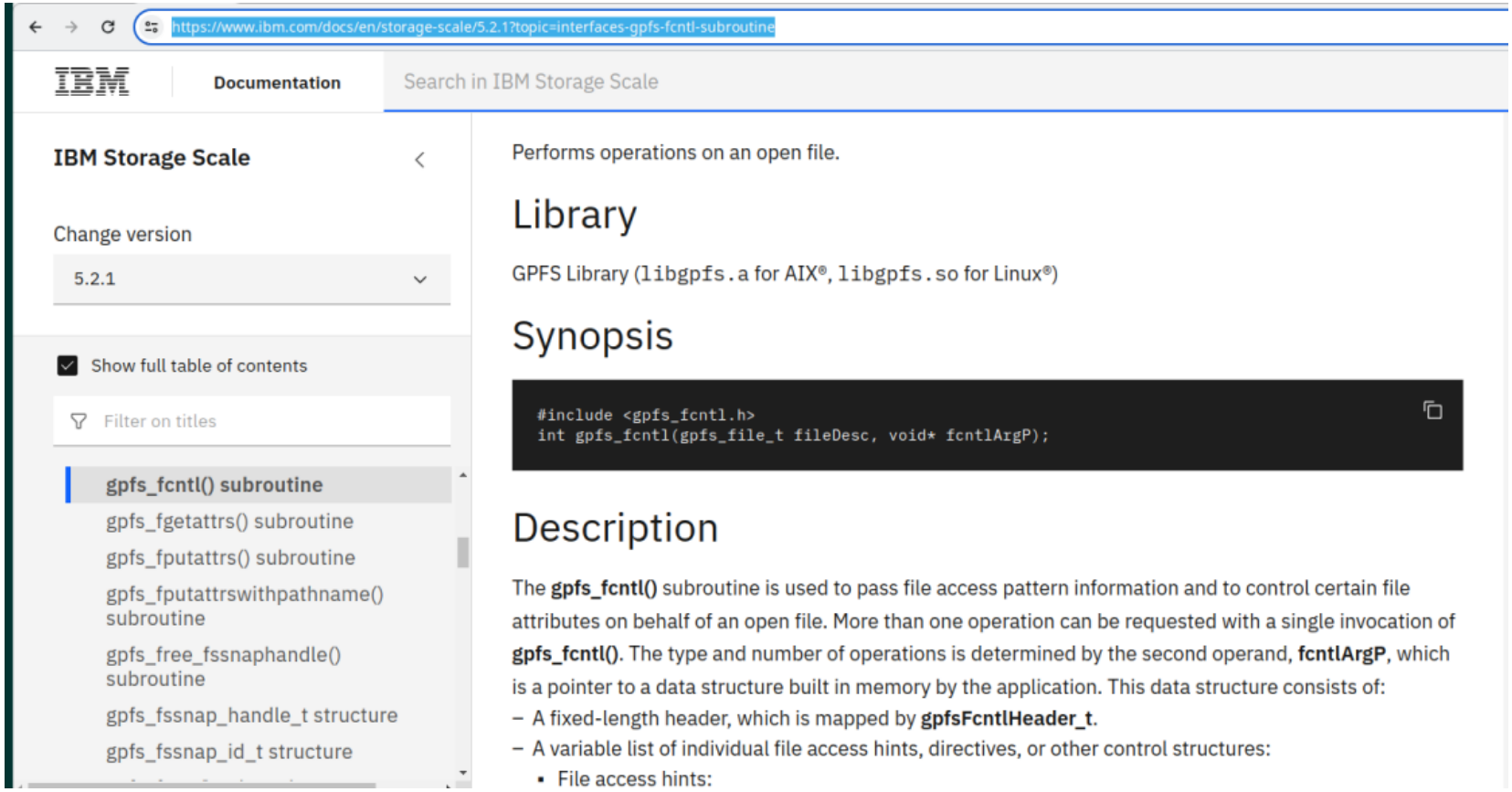
IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (6/7)



IOR Tasks for FZJulich's Version of ior-hard-read (-t 4k -b 4k . . .) – Zoom on I/O Pattern (7/7)



Code Enhancement for Strided Workloads – Hint Documentation



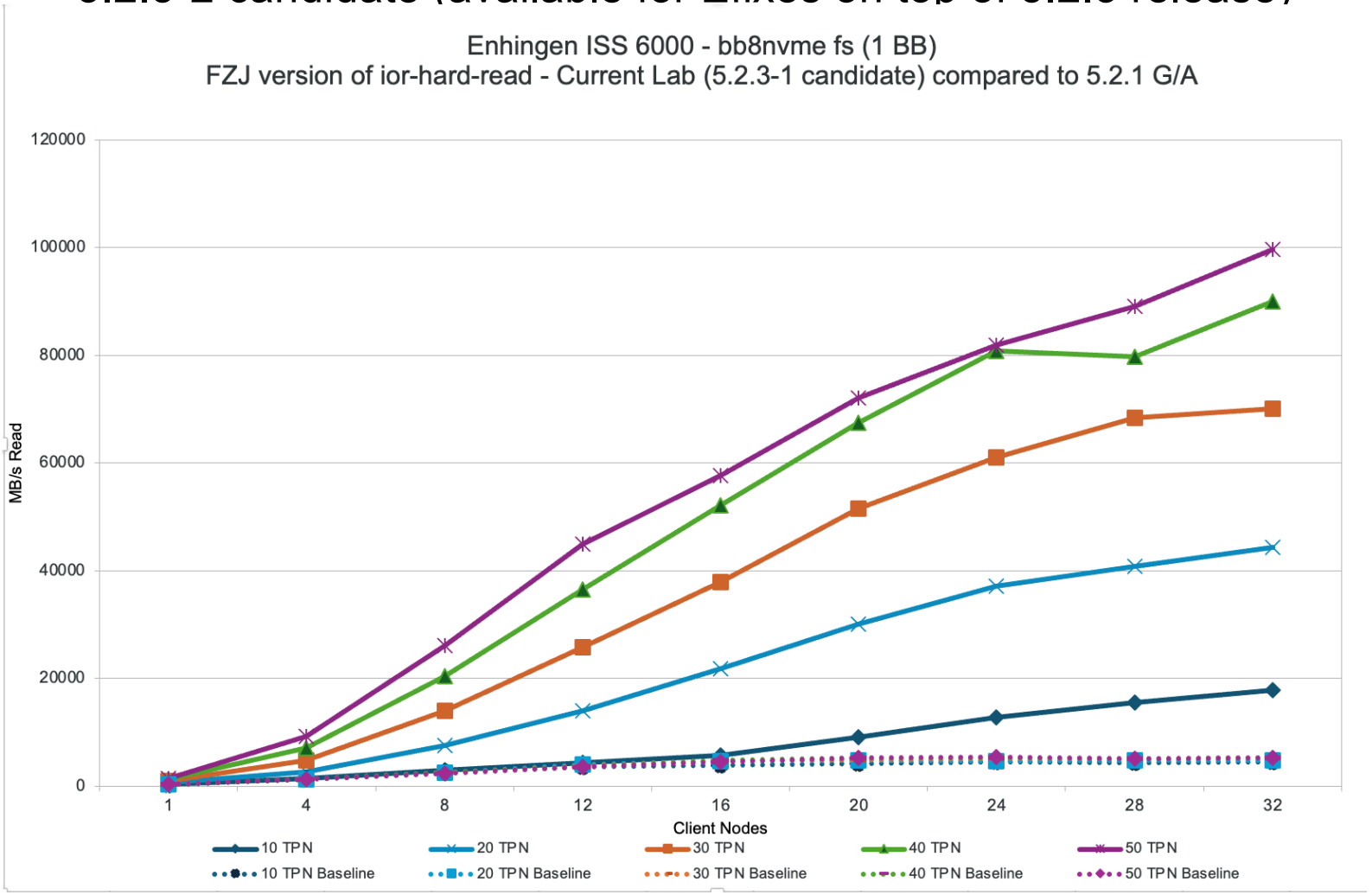
https://www.ibm.com/docs/en/storage-scale/5.2.1?topic=interfaces-gpfs-fcntl-subroutine

ior-hard-read, Example Waiters when Not using fine-grain-read-sharing

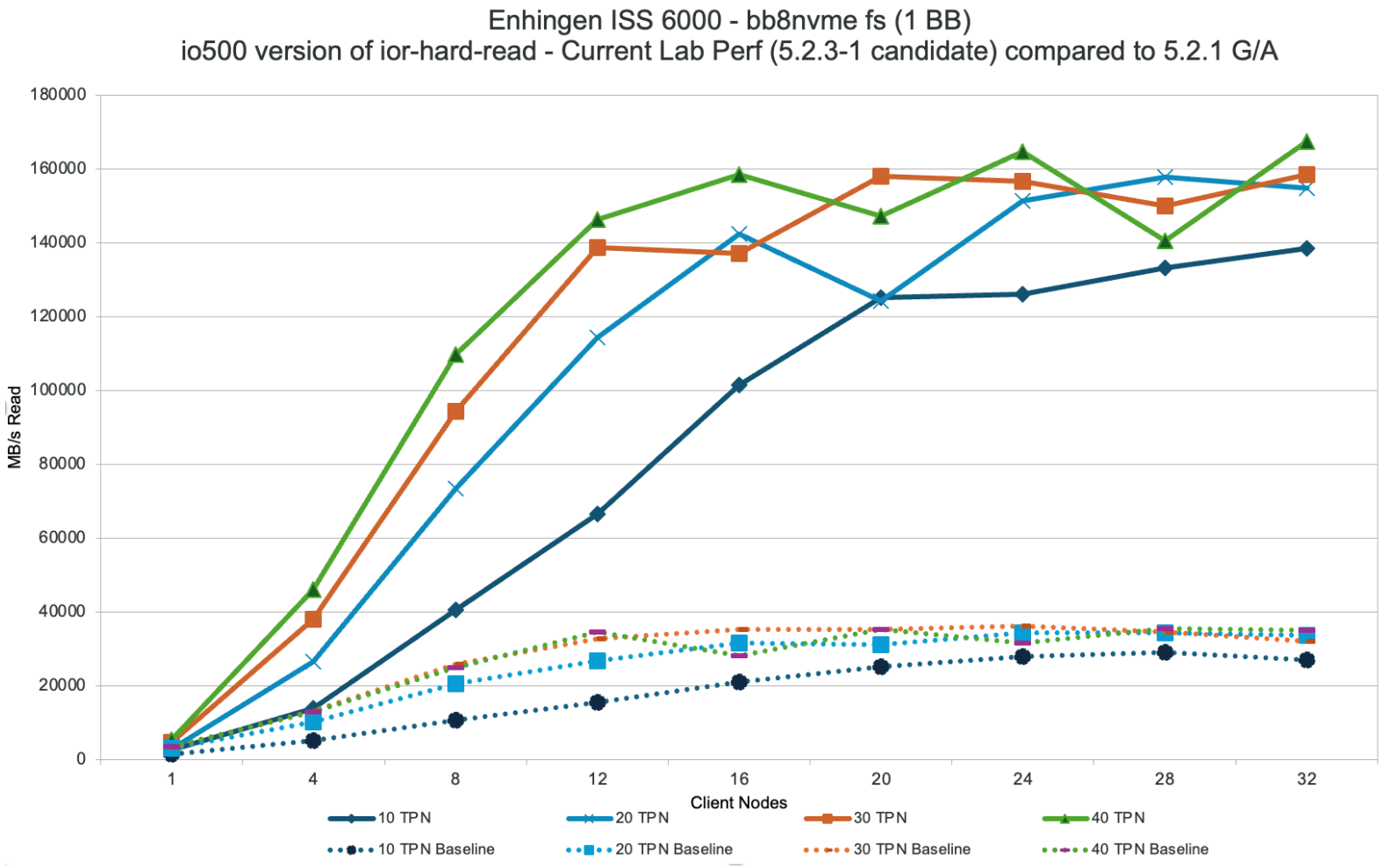
These kinds of waiters are a major bottleneck if fine-grain-sharing is not used for ior-hard-read:

```
[root@fscc-sr655v3-16 ~]# mmfsadm dump waiters
Waiting 0.0181 sec since 2024-09-10_11:00:23, monitored, thread 37228 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0177 sec since 2024-09-10_11:00:23, monitored, thread 40174 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0174 sec since 2024-09-10_11:00:23, monitored, thread 31938 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0136 sec since 2024-09-10_11:00:23, monitored, thread 40170 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0136 sec since 2024-09-10_11:00:23, monitored, thread 40188 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0136 sec since 2024-09-10_11:00:23, monitored, thread 40175 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0134 sec since 2024-09-10_11:00:23, monitored, thread 40173 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40177 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40169 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 33373 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 38095 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40165 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 32333 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40166 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40167 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40178 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0133 sec since 2024-09-10_11:00:23, monitored, thread 40168 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0132 sec since 2024-09-10_11:00:23, monitored, thread 39202 PrefetchWorkerThread: on ThCond 0x18024BA4AA0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0131 sec since 2024-09-10_11:00:23, monitored, thread 40176 PrefetchWorkerThread: on ThCond 0x18024BA30A0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
Waiting 0.0131 sec since 2024-09-10_11:00:23, monitored, thread 40164 PrefetchWorkerThread: on ThCond 0x18024BA30A0 (SyncPairCondvar), reason 'waiting for weak exclusive ThSXLock'
```

Current Lab Performance of Customer Variant of ior-hard-read on ISS 6000 5.2.3-2 candidate (available for Efixes on top of 5.2.3 release)



Current Lab Performance of io500-version of ior-hard-read on ISS 6000 5.2.3-2 candidate (available for Efixes on top of 5.2.3 release)



Agenda

- Data Acceleration Tier (DAT) Performance
- Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9
- Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3
- ✓ **Storage Scale 5.2.2 MMAP Write Performance Improvements**
- Details on Storage Scale Configuration and Tuning Parameters

Mmap: The Basics

- The mmap() facility allows a process to create a memory-mapped region for a file, enabling direct memory access to the file's contents.

- Instead of using traditional read() and write() system calls, a mapped file can be accessed and modified as if it were part of the program's memory.

- This feature provides a convenient way to manipulate file data using pointers, offering an alternative flexible I/O method to using POSIX calls.

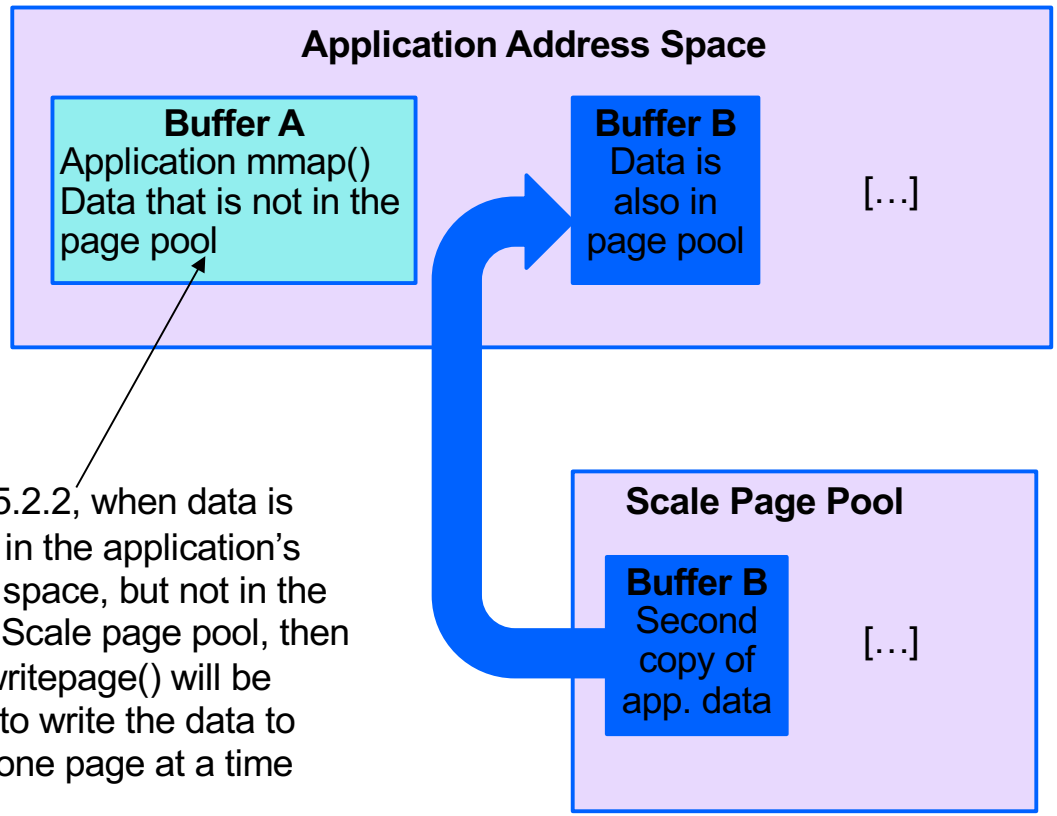
    - Side-note: According to POSIX (http://www.unix.org/single_unix_specification), the behavior of a file system is undefined if normal read or write system calls are issued against a file that is at the same time being accessed through mmap: The application must ensure correct synchronization when using mmap() in conjunction with any other file access method, such as read() and write(), standard input/output, and shmat().

- Changes made to the mapped memory are automatically reflected in the file (for writeable mappings), while reading from the mapped memory fetches data directly from the file.

Details on Improvement for MMAP Writes in 5.2.2 cont.

- When a user modifies memory that has been mmaped the data isn't necessarily written to storage immediately. Writes to storage can occur for multiple reasons, including:
    - Periodic writebacks of data
    - Memory pressure
    - An explicit request from the user via msync()
    - An munmap() call to end the mapping (this is implicit in the exit flow for a process on Linux)
    - Scale will flush dirty mmap pages in response to operations such as handling token revokes and initiating the snapshot quiesce flow
- Prior to 5.2.2, the flushing of dirty mmap data is done via Direct I/O one page at a time if the data to be written is not in the page pool
- In 5.2.2, a new internal gpfs_i_writepages() function is slated to be added to provide better write performance for the case in which mmaped data to be written is not available in the page pool -- Storage Scale will keep a list of virtually contiguous mapped pages and attempt to write up to a full file system block of data in a single I/O operation when the pages are not in the page pool
    - We've seen mmap write performance improvements in the range of 2x to **8x** for such cases

Details on Improvement for MMAP Writes in 5.2.2 cont.



- Prior to 5.2.2, when data is mapped in the application's address space, but not in the Storage Scale page pool, then gpfs_i_writepage() will be invoked to write the data to storage one page at a time

- Prior to 5.2.2, when data is in both in the page pool and the application's address space, optimal performance will be obtained writing this data to storage because we can write larger chunks (typically all the virtually contiguous data up to a full block), but the down-side is that this path requires maintaining two copies of the data

- In 5.2.2, a new gpfs_i_writepage**s**() function is added so that multiple pages can be flushed to storage in a single operation, when the data is not in the pagepool (e.g. Buffer A in the shown diagram)

- Writing data in larger chunks makes more efficient use of storage and network bandwidth

Example mmap Sequential Write Performance Test with iozone with Scale 5.2.2

```
iozone -r 4096k -s 320g -i 0 -C -B -+u -+m client.list.iozone -+n -w -t 16
        Iozone: Performance Test of File I/O
                Version $Revision: 3.506 $
[…]

        Record Size 4096 kB
        File size set to 335544320 kB
        Using mmap files
        CPU utilization Resolution = 0.000 seconds.
        CPU utilization Excel chart enabled
        Network distribution mode enabled.
        No retest option selected
        Setting no_unlink
        Command line used: iozone -r 4096k -s 320g -i 0 -C -B -+u -+m
client.list.iozone -+n -w -t 16
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
        Throughput test with 16 processes
        Each process writes a 335544320 kByte file in 4096 kByte records
```

Example mmap Sequential Write Performance Test with iozone with Scale 5.2.2

```
        Test running:
        Children see throughput for 16 initial writers  = 3434808.53 kB/sec
        Min throughput per process                      =  172065.73 kB/sec
        Max throughput per process                      =  242590.41 kB/sec
        Avg throughput per process                      =  214675.53 kB/sec
        Min xfer                                        = 237998080.00 kB
        CPU Utilization: Wall time 1846.320    CPU time 8531.498    CPU
utilization 462.08 %

        Child[0] xfer count = 296591360.00 kB, Throughput =  214420.53 kB/sec,
wall=1536.026, cpu=532.873, %= 34.69
        Child[1] xfer count = 324034560.00 kB, Throughput =  234266.62 kB/sec,
wall=1420.051, cpu=517.188, %= 36.42
Child[15] xfer count = 253612032.00 kB, Throughput =  183180.95 kB/sec,
wall=1802.350, cpu=576.398, %= 31.98

iozone test complete.
```

cat client.list.iozone

```
fscc-sr655v3-1 /gpfs/bb8nvme /usr/local/bin/iozone
[…]
fscc-sr655v3-16 /gpfs/bb8nvme /usr/local/bin/iozone
```

Example mmap Sequential Write Performance Test with iozone with Scale 5.2.2

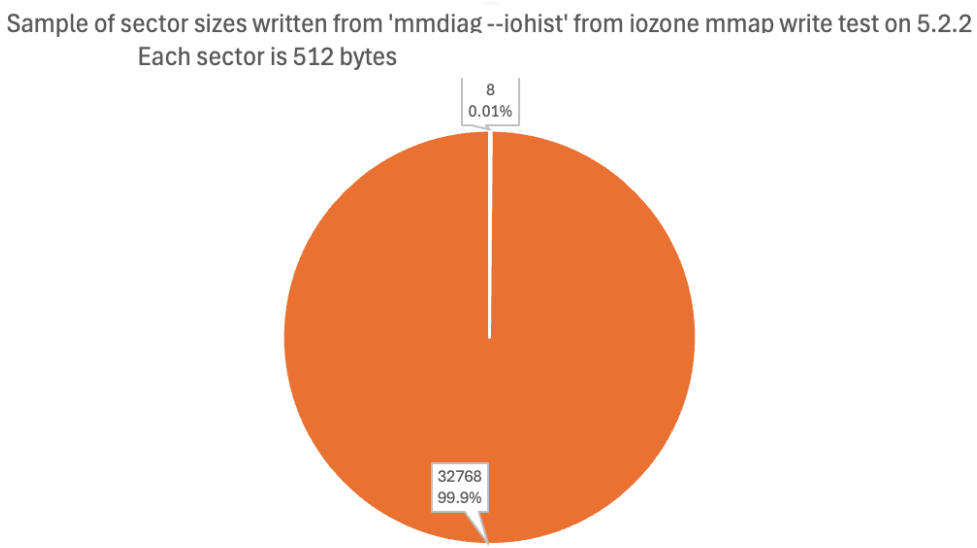
Test running:

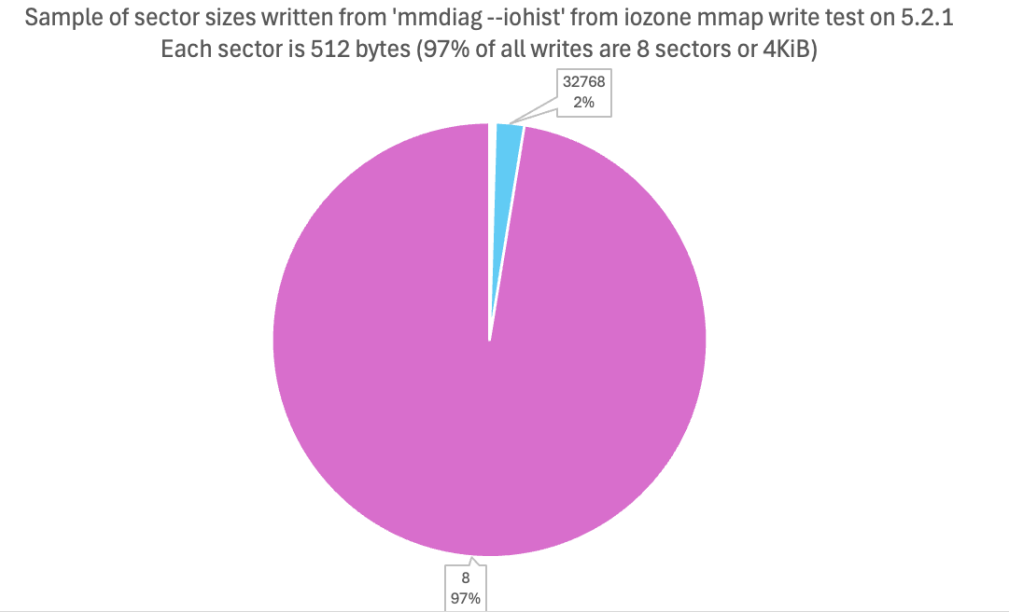
```
        Children see throughput for 16 initial writers  = 8742036.52 kB/sec
        Min throughput per process                      =  188138.77 kB/sec
        Max throughput per process                      =  749017.94 kB/sec
        Avg throughput per process                      =  546377.28 kB/sec
 CPU Utilization: Wall time 1671.646    CPU time 6887.430    CPU utilization
412.01 %

        Child[0] xfer count = 335212544.00 kB, Throughput =  748271.31 kB/sec,
wall=448.362, cpu=362.127, %= 80.77
        Child[1] xfer count = 332029952.00 kB, Throughput =  741161.50 kB/sec,
wall=452.867, cpu=365.989, %= 80.82
[…]
Child[15] xfer count = 85319680.00 kB, Throughput =  190288.11 kB/sec,
wall=1635.845, cpu=573.107, %= 35.03
```

For the same example, the Scale GA 5.2.2 code get 8.7 GB/s write throughput
the Scale GA 5.2.1 code got 3.4 GB/s write throughput

Comparing Application Write I/O Sizes for mmap Write pre-GA Scale 5.2.2 vs 5.2.1

- Sampling ‘mmdiag –iostat for the IO requests submitted by the clients:
  - almost all the write I/Os are full block I/Os (32768 sectors = 16M) in the case of the GA 5.2.2 run
  - almost all the write I/Os are 4KiB in the Scale 5.2.1 run





Agenda

- Data Acceleration Tier (DAT) Performance
- Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9
- Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3
- Storage Scale 5.2.2 MMAP Write Performance Improvements
- ✓ **Details on Storage Scale Configuration and Tuning Parameters**

Storage Scale Configuration Tuning Parameters

Some starting **mmchconfig** parameter point tuning suggestions (details on what these parameters do is provided later in these slides)

Starting Point Client Settings:

```
pagepool= <Set the page pool to 25% of real memory with a maximum size of 64G>
numaMemoryInterleave=yes # make sure numactl RPM is installed
ignorePrefetchLUNCount=yes
workerThreads=1024
maxFilesToCache=128K
maxStatCache=128K
maxblocksize=16M
maxMBpS=<NW-Bandwidth-in-MBpS*2> # e.g. 2048 for a 1024 MiB/s (1 GiB/s) ethernet
prefetchLargeBlockThreshold=<file system blocksize> # consider this setting, unless you're
                                                    # running sequential I/O benchmarks
(If the configuration supports RDMA, enable verbsRdma, verbsRdmaSend, and verbsPorts
settings.)
```

In Storage Scale 5.2.0, we implemented the following default changes:

```
pagepool=increased from 1G to 4G
numaMemoryInterleave=changed from 'no' to 'yes'
ignorePrefetchLUNCount=changed from 'no' to 'yes'
workerThreads=increased from 48 to 512
```

Storage Scale Configuration Tuning Parameters (cont.)

Some starting point server **mmchconfig** tuning suggestions (for GNR servers, described later in these slides, the default settings should be OK but check if RDMA can be enabled)

Non-GNR NSD Server Settings:

```
pagepool=<Set the page pool to 25% of real memory with a max. size of 32G
        #generally, a pagepool > 32G isn't advised, even for servers with > 128G>
numaMemoryInterleave=yes # make sure numactl RPM is installed
ignorePrefetchLUNCount=yes
workerThreads=1024
nsdMinWorkerThreads=3842
nsdMaxWorkerThreads=3842
nsdSmallThreadRatio=2
nsdThreadsPerQueue=16
nsdbufspace=70
(If the configuration supports RDMA, enable verbsRdma, verbsRdmaSend, and verbsPorts
settings as described below.)
```

tuning recommendations that are usually associated with client workloads but are commonly set on servers:

```
maxFilesToCache=128K
maxStatCache=128K
maxMBpS=<NW-Bandwidth-in-MBpS*2>
```

Storage Scale Configuration Tuning Parameters (cont.)

mmchconfig Parameter Tuning Recommendations

- **Pagepool**
  - Defines the amount of memory to be used for caching file system data and metadata. Also used in some non-caching operations, e.g., buffers allocated for encryption buffers and DMA transfers for DIO data
  - NSD servers that are not running IBM Storage Scale RAID (GNR) do not cache data in the page pool that has been used to satisfy client requests. For this reason, these servers typically benefit less from higher page pool settings than IBM Storage Scale (GNR_ servers
  - On client nodes, increasing the pagepool beyond a given size (needed for good sequential performance) is most beneficial for (non-direct I/O) workloads that benefit from caching – e.g., workloads that reread the same data, when a good portion of the data can be cached in the GPFS page pool
  - IBM Storage Scale RAID servers can cache data used to satisfy client requests.
  - In GNR configurations, the mmchconfig option nsdRAIDBufferPoolSizePct defines the percentage of the GPFS page pool that is dedicated for GNR use – this defaults to 80% in ESS/SS configurations; in ECE configurations, particularly converged use cases, lower values for nsdRAIDBufferPoolSizePct may be more appropriate
  - The mmchconfig option nsdRAIDNonStealableBufPct defaults to half of the GNR page pool memory, defining the maximum amount of non-stealable memory used for GNR metadata allocations, e.g., non master reserved memory, vtrack map, nspd disk memory, and any remaining GNR page pool memory is available for caching

Storage Scale Configuration Tuning Parameters (cont.)

- **maxFilesToCache (MFTC)**
  - The total number of different files that can be cached at one time. Every entry in the file cache requires some shared segment space for control related data structures and the inode. This is in addition to any of the file's data and indirect blocks that might be cached in the page pool.
  - Adjust for workloads on specific nodes (e.g. login, gateway, protocol) that will access the data from many files, which may benefit from caching
  - Also adjust for any manager nodes that may be acting as the metanode in these cases:
    a) in a multi-cluster configuration in which multiple remote clusters access the same file-- in such cases the metanode will be forced to one of the manager nodes in the storage cluster
    b) if the metanode is forced to a manager node
- **maxStatCache (MSC)**
  - Specifies the number of inodes to keep in the stat cache. The stat cache maintains only the inode information needed to resolve a stat() call for the associated inode.
  - Adjust for workloads on specific nodes (e.g. login, gateway, protocol) that performs many stat operations (e.g. ls -l)
- Often **MFTC** and **MSC** limits will be defined by memory limits on the clients or the token servers – in tuning both of these variables ensure there is sufficient memory on both client and server nodes to handle these caching requirements (you need to look at memory usage on both the clients and token servers to determine how to optimize these tunables)

Storage Scale Configuration Tuning Parameters (cont.)

This chart provides more details on the memory allocations associated with increasing more file and stat cache entries as discussed in the Spectrum Scale Memory Usage charts on www.spectrumscaleug.org:

- **Memory Allocations for maxStatCache and maxFilesToCache on the manager nodes:**
  - The total amount of memory required for token management:
    ServerBRTreeNode = 56b (memory used per each byte range (BR) lock managed by tokens on a token server)
    TokenMemPerFile ~= 464b (total token memory per cached file, independent of number of BR locks)

    ClusterBRTokens = NoOfHotBlocksRandomAccess * ServerBRTreeNode
    NoOfHotBlocksRandomAccess is the total number of unique ServerBRTreeNode range locks held

    BasicTokenMem = (MFTC+MSC)* NoOfNodes * TokenMemPerFile
    TotalTokenMemory = BasicTokenMem + ClusterBRTokens

- **Additional Shared Segment Allocations (client memory allocations -- not Token Memory on manager nodes):**
  Each MFTC entry =~ 10k
  Each MSC entry =~ 480b

  Since the size of a MSC entry is smaller than the MFTC entry, if we need to cache only enough information to preform a stat() call, it's preferrable to use a MSC entry rather than a MFTC entry (tune **maxStatCache** and **maxFilesToCache** accordingly for optimal caching of both stat() calls and file accesses).

Storage Scale Configuration Tuning Parameters (cont.)

- **workerThreads**
  - Controls a set of configuration parameters related to parallelism (parallelWorkerThreads, logWrapThreads, logBufferCount, maxBackgroundDeletionThreads, maxBufferCleaners, maxFileCleaners, syncBackgroundThreads, syncWorkerThreads, sync1WorkerThreads, sync2WorkerThreads, maxInodeDeallocPrefetch, flushedDataTarget, flushedInodeTarget, maxAllocRegionsPerNode, maxGeneralThreads, worker3Threads, and prefetchThreads)
  - Generally setting workerThreads to 512 or 1024 is a good starting point.
  - Values that are too low will limit how many requests can be initiated in parallel and values that are too high will reduce performance as a result of mutex and lock contention. You can use traces (formatted with the trcio tool), and/or 'mmfsadm dump iocounters', to assess mutex and lock contention.

- **ignorePrefetchLUNCount**
  - IBM typically recommend this be set to 'yes', though it's more important in cases in which the LUNs presented to GPFS are made up of a large numbers of physical disks (e.g. ESS/ISS vdisks).

Storage Scale Configuration Tuning Parameters (cont.)

- **numaMemoryInterleave**
  - Set to “yes” on Linux systems to ensure that GPFS allocates memory across all the available NUMA domains with system memory. Also ensure that the ‘numactl-’ rpm is installed (e.g. numactl-2.0.14-9.el9.x86_64)
- **maxMBpS**
  - Defines the rate at which GPFS submits I/O (controls how prefetching and write-behind occur)
  - Set MBpS to twice the network bandwidth to GPFS in MBpS (default of **2048** is appropriate for a 1 Gigabit ethernet)
- **verbsRdma**
  - Set to “enable” to allow GPFS to use RDMA via the verbs API for data transfers between an NSD clients and servers.
- **verbsRdmaSend**
  - Set to “yes” to enable the use of verbs send operations for RPCs between GPFS daemons
- **verbsPorts**
  - Defines the list of interfaces to use for verbs requests (as defined by **verbsRdma** and **verbsRdmaSend** settings)

Storage Scale Configuration Tuning Parameters (cont.)

Prefetch Tuning

- In some cases, GPFS prefetching may be excessive, resulting in performance issues, such as:

reduced application performance for I/O patterns that don't benefit from prefetch. If not consumed by the application, the prefetched I/Os compete for network, disk and CPU without adding any benefit to the application performance.

(a) On the client side, all any prefetched I/Os that are not consumed by the application will use client CPU and network bandwidth.

(b) On the server side, all any prefetched I/Os that are not consumed by the application will use server CPU, network, and storage bandwidth, which can lead to what is sometimes described as a 'noisy neighbor effect') to other applications running on the system.

Both (a) and (b) can be masked on some sometimes if the clients and servers are not reaching the limits of the throughputs they can deliver.

- the 'normal' full prefetching flows are tied to full block reads so increasing the block size of the filesystem will increase the rate of prefetch (smaller file system block sizes may mitigate the interference effect)

- Introduced in 5.1.3, **prefetchLargeBlockThreshold** can be enabled to cause GPFS to prefetch less aggressively:

`mmchconfig prefetchLargeBlockThreshold =<FS_blocksize> –i`

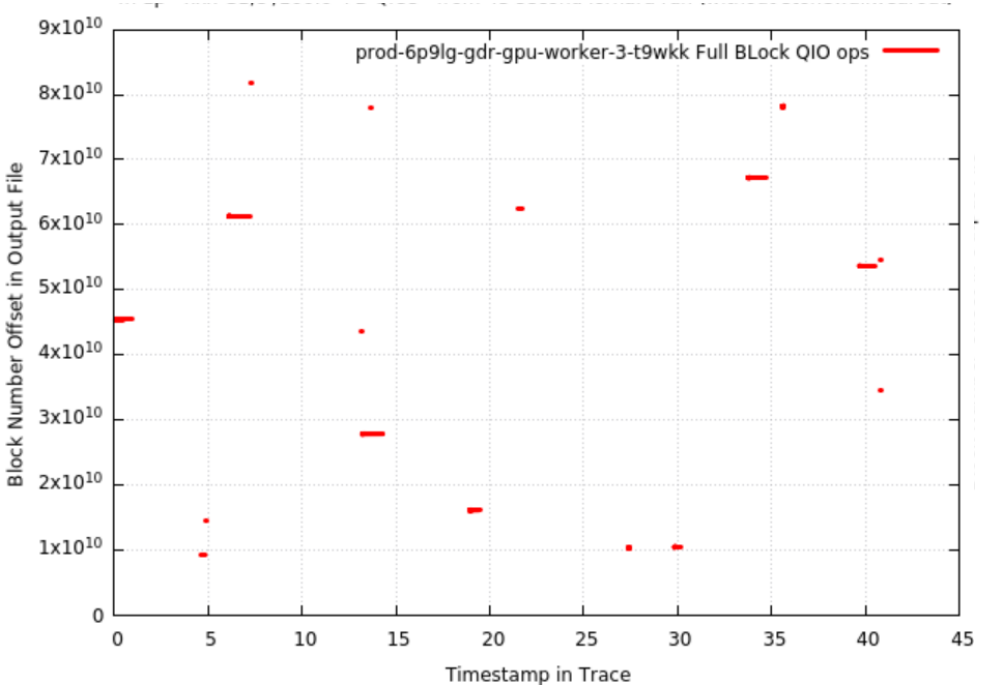
For example, to enable reduced prefetching for any file system with a blocksize of 4MiB or lower, set prefetchLargeBlockThreshold= 4194304 (with the mmchconfig '-i' this takes effect without restarting GPFS)

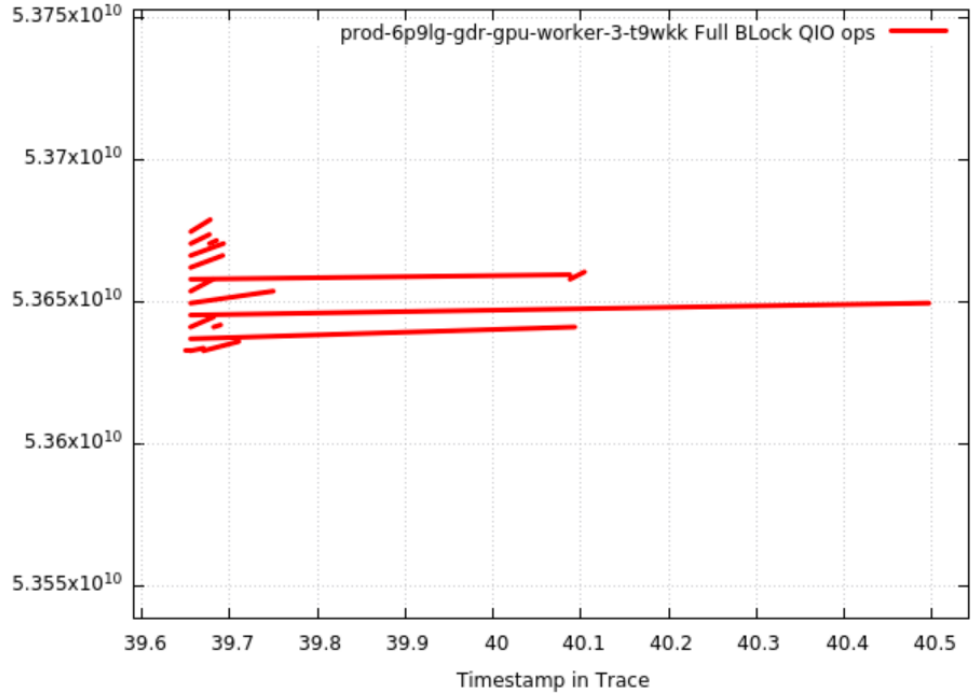
Storage Scale Configuration Tuning Parameters (cont.)

Any application that frequently does short series of sequential reads, followed by seeking to a new location and reading again, may perform better with the **prefetchLargeBlockThreshold** setting.

When monitoring the system, much more I/O traffic than expected will be observed. With the default prefetching (not **prefetchLargeBlockThreshold**) two consecutive sequential reads, (even small, 4 KiB reads) will cause full blocks to be prefetched.

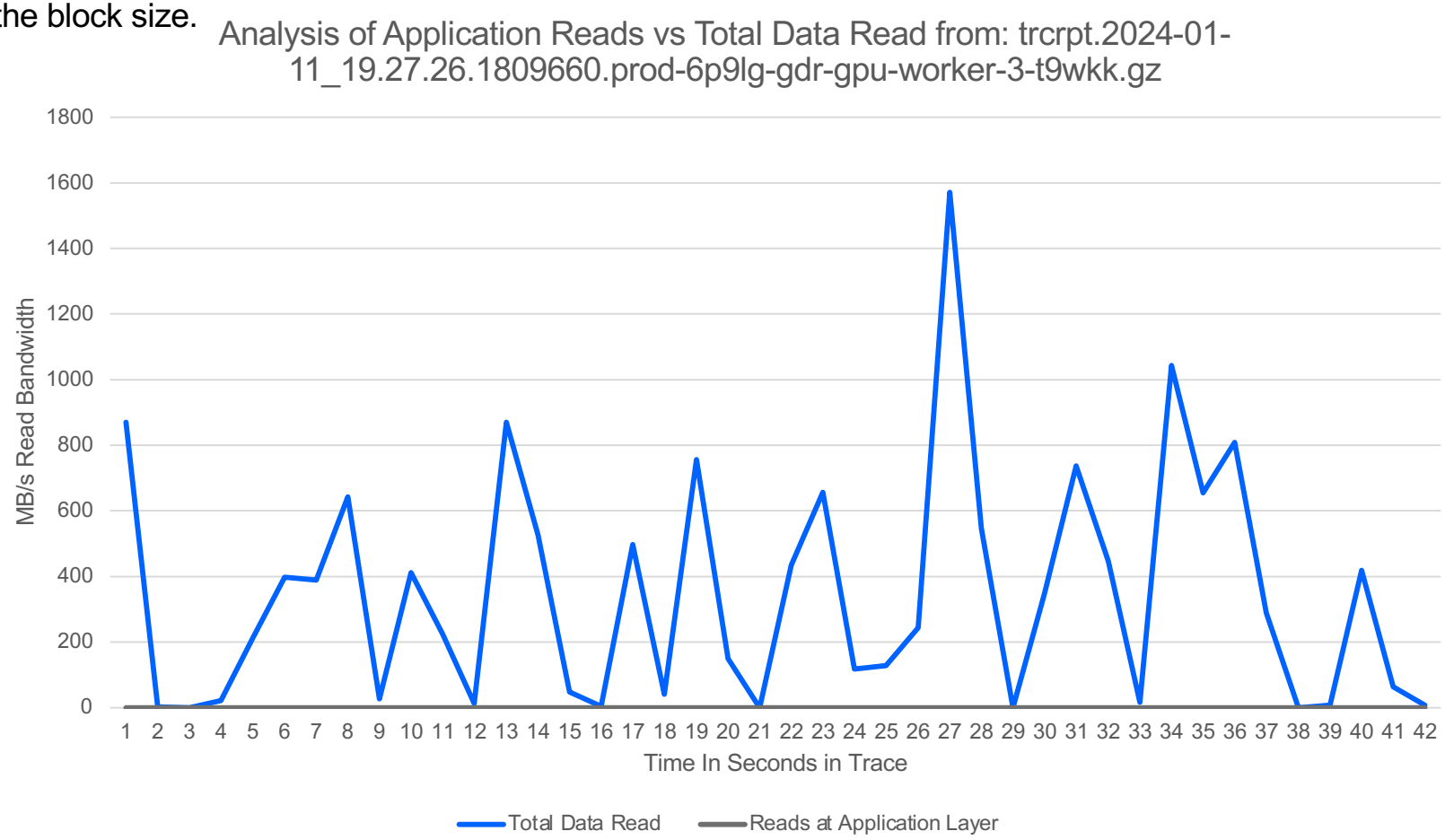
We first observed the default (full block) prefetching may be inefficient analyzing a customer application from New York Genome (KittyHawk) and we've since observed this behavior with many other workloads (e.g. AI workloads as per the example below from the Vela system):





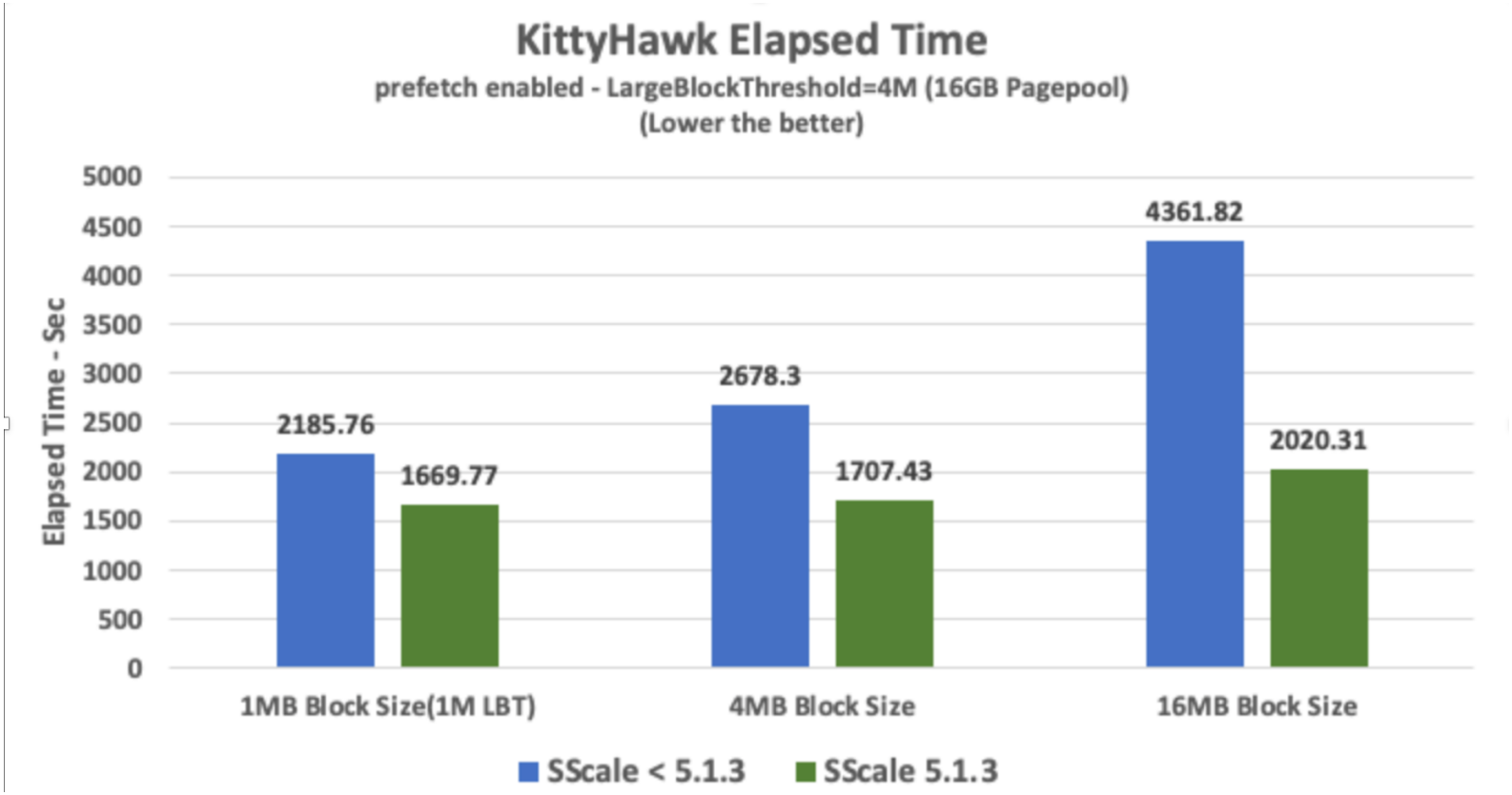
Storage Scale Configuration Tuning Parameters (cont.)

- In the plot below, the amount of data being requested for read at the application layer blends with the x-axis.
- The excessive prefetching shown below on Vela was eliminated by setting **prefetchLargeBlockThreshold** to match the block size.



Storage Scale Configuration Tuning Parameters (cont.)

Results of Performance Tuning prefetchLargeBlockThreshold Using Customer Provided Testcase (KittyHawk)



Storage Scale Configuration Tuning Parameters (cont.)

Prefetch Tuning cont.

- Another issue that can occur with prefetching is excessive mutex contention across the threads that are doing prefetch and write behind.
- This mutex contention can be mitigated by enabling the **prefetchLargeBlockThreshold** option and/or by tuning the **prefetchPartitions** mmchconfig option, documented in 5.2.0
- The contention comes from too many prefetch threads contending for a mutex, and tuning **prefetchPartitions** splits the mutex into a number of shards equal to the value specified (the default is 1)
- The mmchconfig man page describes a procedure for using ‘mmfsadm dump iocounters’ to look for mutex contention and shows this example of “PrefetchListMutex” contention from a dump:

```
                        Call Count Aggregate Seconds   Average Seconds   Maximum Seconds      nvcsw     nivcsw  ID MutexName
  ---------------------- ---------- ----------------- ----------------- ----------------- ---------- ---------- --- --------------------------------
              dAcquireMutex:      1036721        54.955000412           0.000053008           0.007209259     2083108          35   44 PrefetchListMutex
[...]
              kAcquireMutex:       162468         5.575387236           0.000034317           0.007656795      168757          16   44 PrefetchListMutex
              kAcquireMutex:            1         0.000000791           0.000000791           0.000000791           0           0   10 MutexListMute
```

(Along with ‘mmfsadm dump iocounters” analysis, a similar procedure can be done with traces and the trcio tool.)

- The down-side to sharding the mutex by increasing **prefetchPartitions** is that fewer prefetch threads will be available for workloads that use fewer application threads (this translates to the number of ‘instances’ that Scale sees) and the long-term plan is to address this issue is to enable a currently undocumented option that dynamically rebalances prefetch threads (every 5 seconds) across instances

Storage Scale Configuration Tuning Parameters (cont.)

- Beyond just **mmchconfig** parameter tuning, additional system tuning can be done for various system parameters, such as:
  - sysctl parameters (e.g. to optimize TCP/IP network settings),
  - network adapters (e.g. ethernet device specific settings, such as RX/TX ring buffers)
  - storage devices (e.g. udev settings for disks),
  - CPU settings (e.g. governor, cstate and pstate tuning – typically done through tuned profiles).

    For optimal client performance (particularly for RDMA) consider preventing low power CPU states, e.g.: `cpupower frequency-set -g performance`

- These settings are managed on ESS/ISS systems and, to some degree, ECE systems, but should be tuned according to the overall system workload on other (non-Storage Scale RAID/ non-GNR) systems.
  - Some starting point recommendations for these parameters can be found in the **IBM Storage Scale and IBM Elastic Storage System Network Guide:**
    - section 3.2 (Ethernet Network Adaptor configuration settings)
    - section 3.3 (TCP/IP Settings)

Thanks!