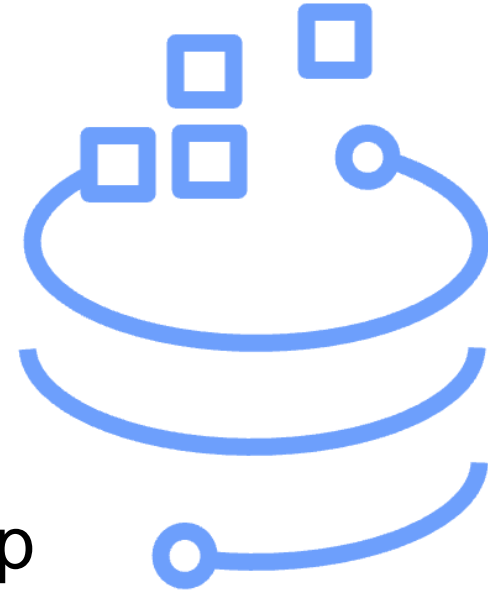


Storage Scale Performance Update

Aug 6, 2025

Storage Scale West Coast User Group



Storage Scale

John Lewars (Storage Scale Performance Architect)

John Divirgilio (Storage Scale Performance)

Olaf Weiser (Storage Scale Performance)



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 introduced in Storage Scale 6.0.0**
- 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

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 concurrent use of:

a Storage Scale RAID (formerly known as GNR) encoded **reliable** copy (replica)

AND

a **performance** copy (replica) which can be:

- (a) a replica physically local to the compute client
- or -
- (b) a replica stored on an NVMeoF drive

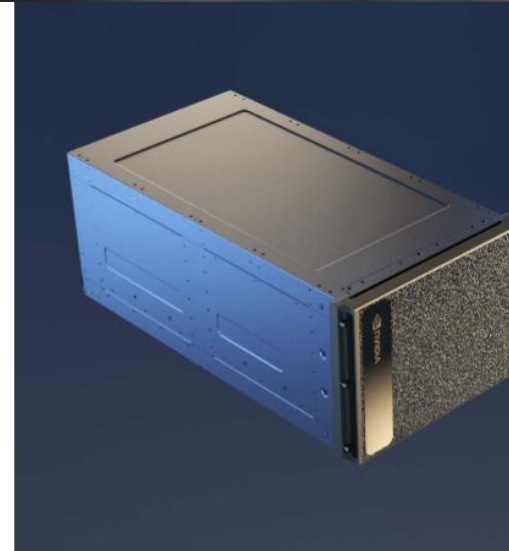
This provides high performance without compromising reliability. IOPs are significantly improved over existing offerings.

In the Fall 2025 release, IBM intends to support the NVMeoF-based and local-storage offerings (note these offerings may not be configured at the same time).

NVIDIA DGX SuperPOD with IBM ESS/ISS



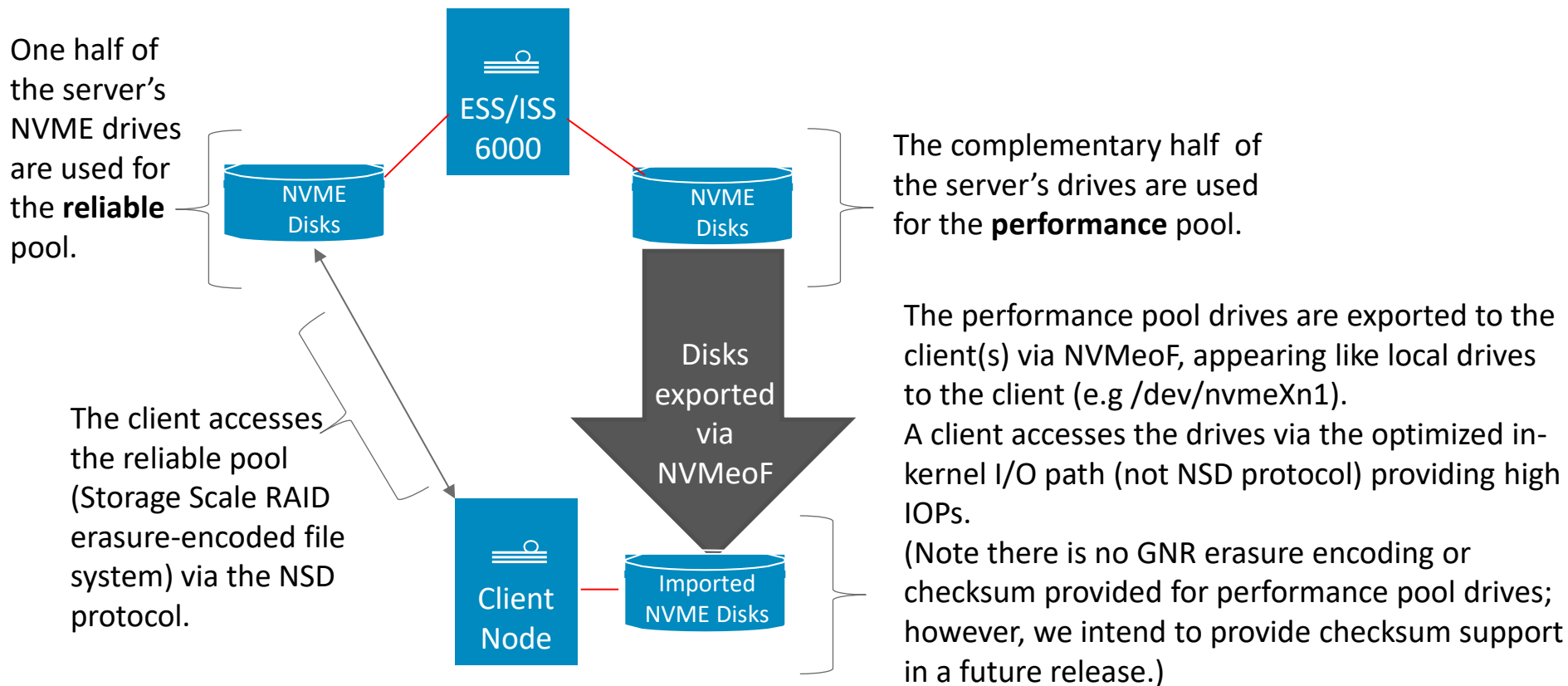
* Asymmetric Data Replication has been referred to in past presentations by its IBM internal name, 'Ustore'.



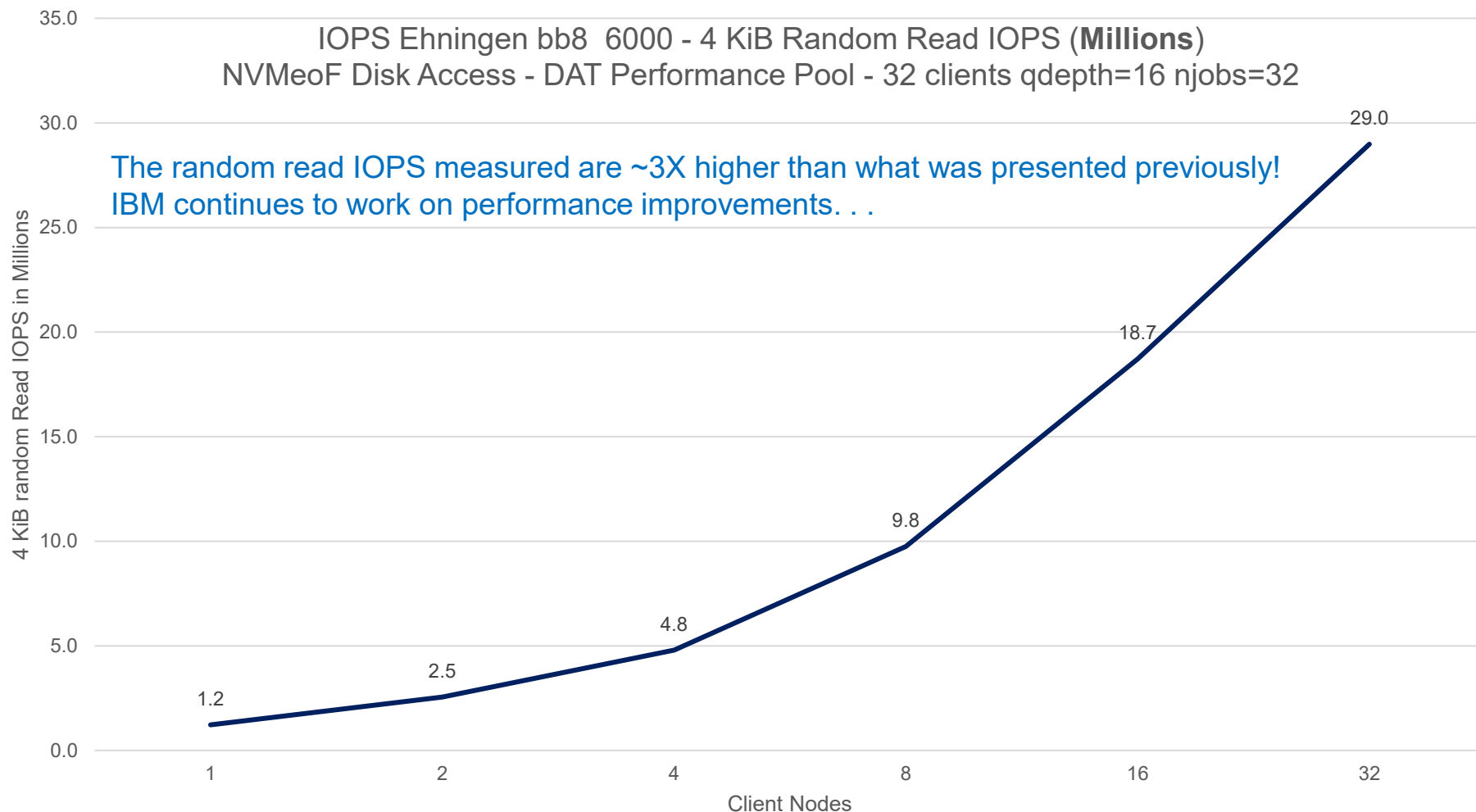
Specifications

GPU	8x NVIDIA H100 Tensor Core GPUs
GPU memory	640GB total
Performance	32 petaFLOPS FP8
NVIDIA® NVSwitch™	4x

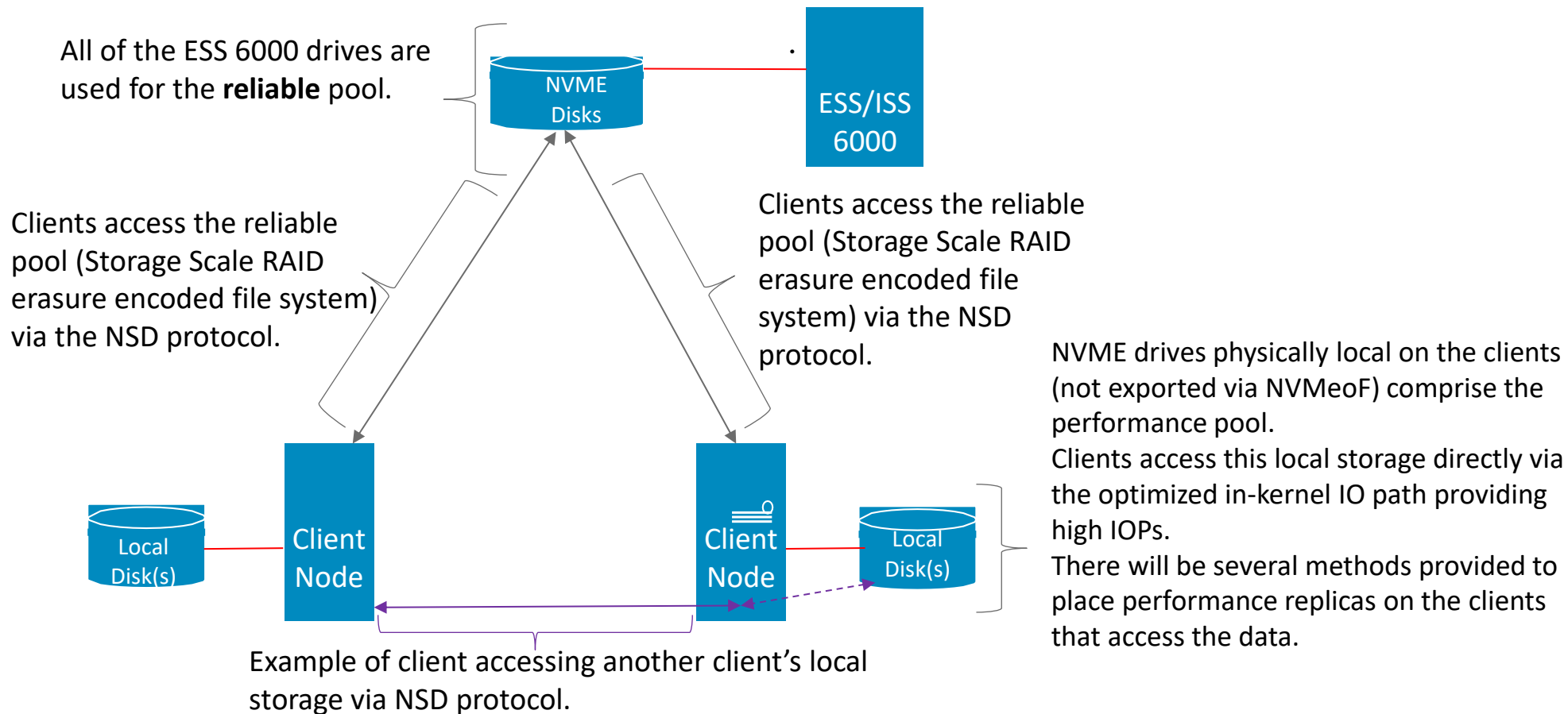
Asymmetric Data Replication File System use case: DAT exploiting shared storage via NVMeoF



DAT with NVMeoF (RoCE): 4 KiB Rand. Read: **29 million IOPS** at 32 nodes

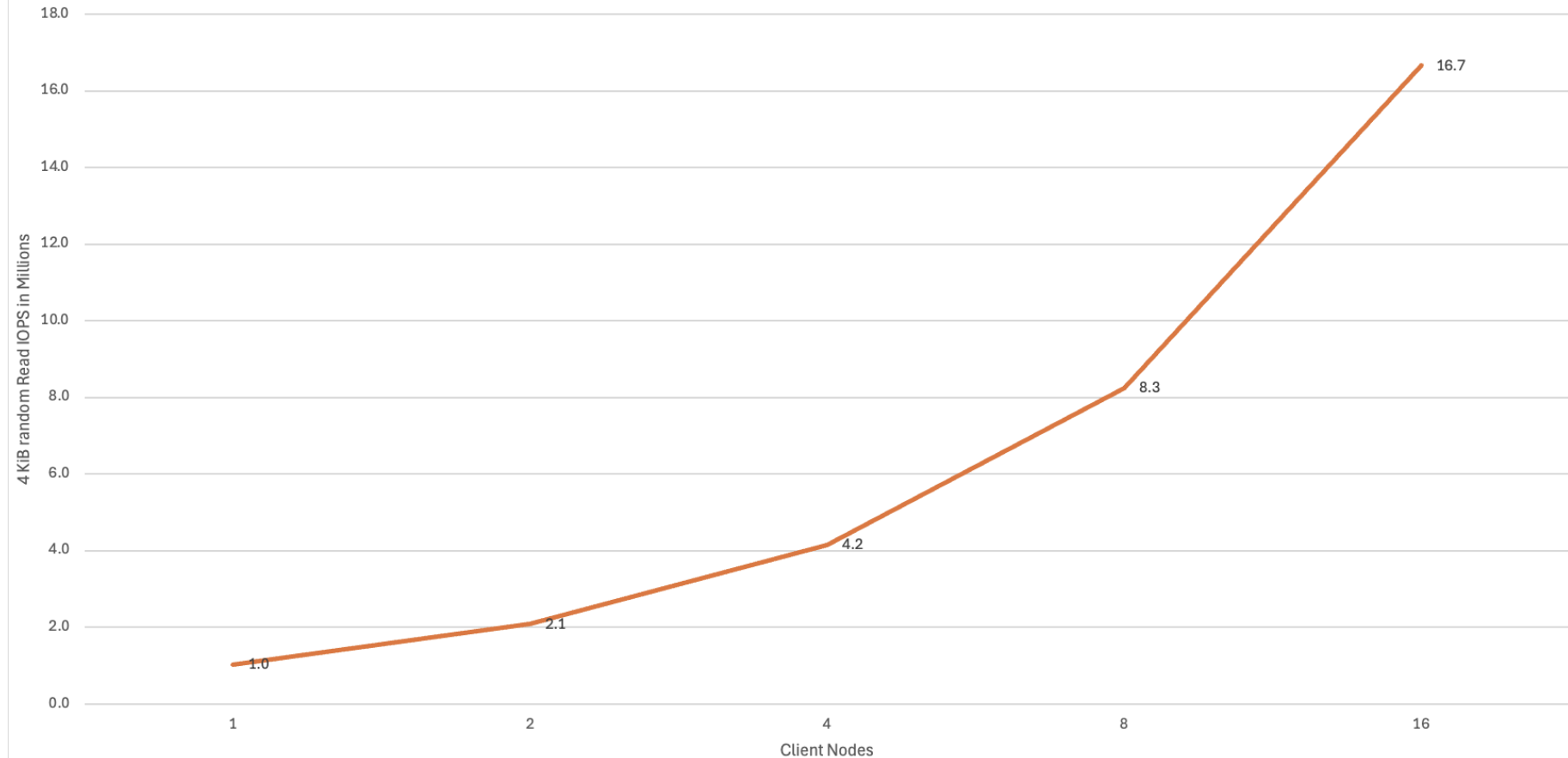


Asymmetric Data Replication Use Case: DAT exploiting client-local storage



DAT with client-local storage: 4 KiB Rand Read: 16 million IOPs at 16 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



Agenda

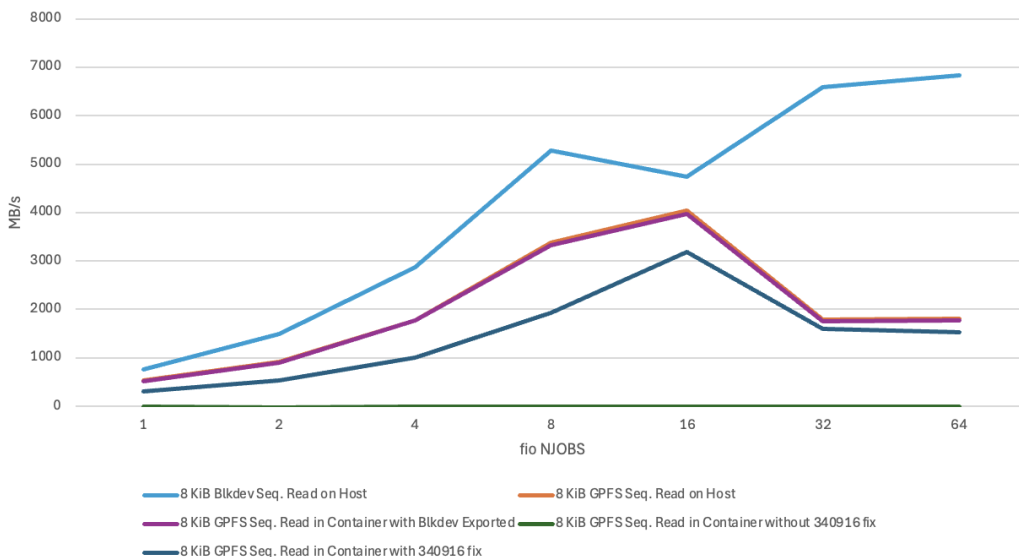
- Data Acceleration Tier (DAT) Performance
- ✓ **Storage Scale Performance 5.2.3 Fix for Cgroups Permissions Changes Introduced in RHEL 9**
- Performance Improvements for Strided-Read (e.g. ior-hard-read) in Storage Scale 5.2.2 and 5.2.3
- Storage Scale 5.2.2 MMAP Write Performance Improvements

Container Performance Issue with Direct I/O fixed in 5.2.2

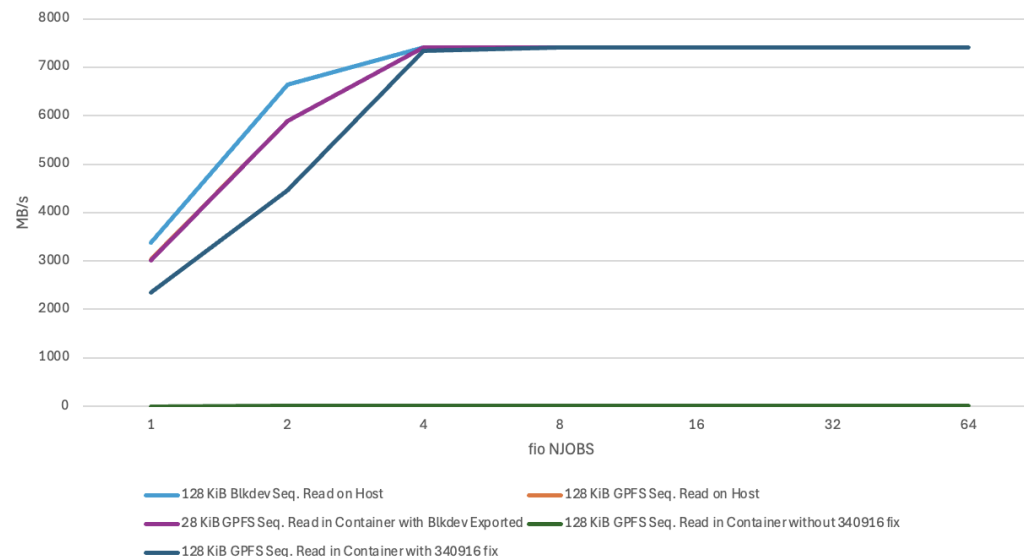
- Starting in RHEL 9, when IBM Storage Scale runs in an unprivileged container it will get an EPERM error when calling `gpfs_blkdev_get_by_dev()` to access any block device that is directly attached (either because they are local devices, or the devices are made to appear local, e.g., via an iSCSI or NVMeoF attachment).
- The exposure is due to a cgroups permissions check and may also be seen in other cgroups enabled configurations, including VMs (outside of containers).
- 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 (eg, token swapping) has a severe performance impact.
- A code fix in IBM Storage Scale 5.2.2 avoids dropping out of Direct I/O mode. On an EPERM error, the I/O is retried through `mmfsd` (Storage Scale daemon) which runs 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 5.2.2 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)

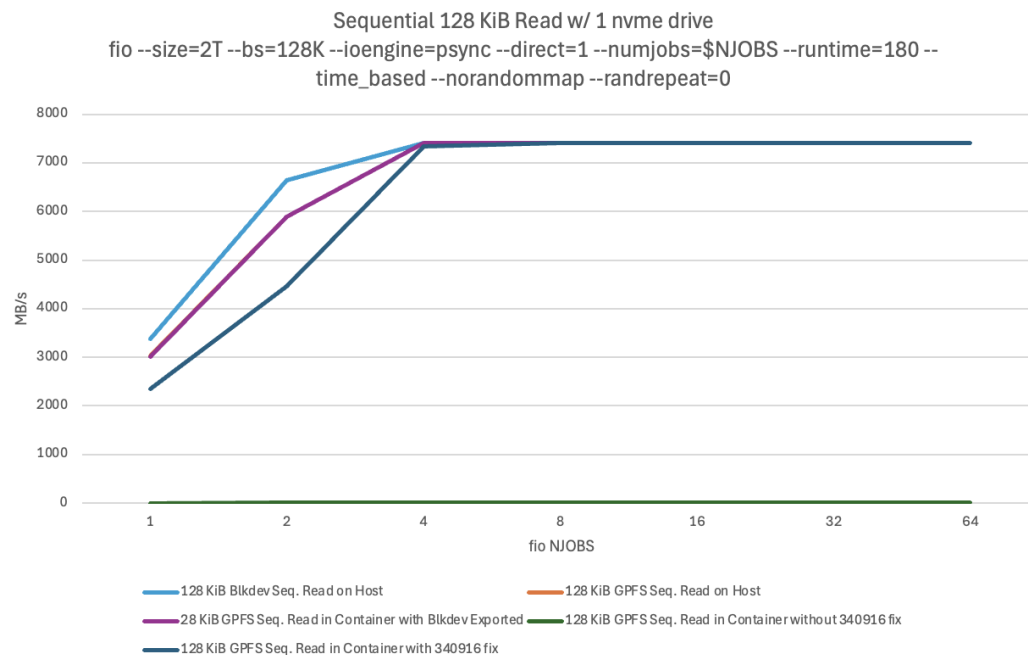
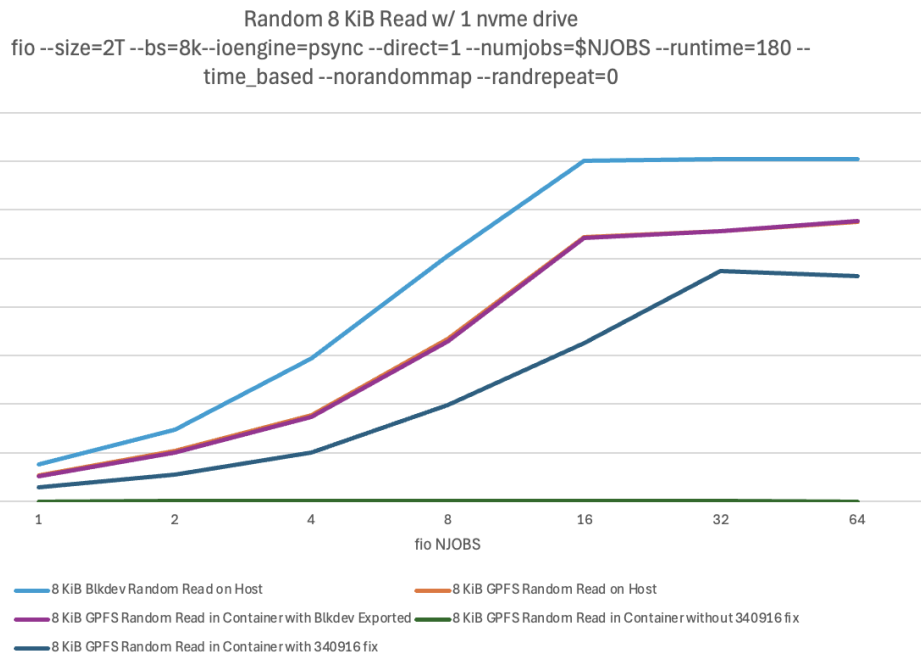
Sequential 8 KiB Read on bryce1 w/ 1 nvme drive
fio --size=2T --bs=8k --ioengine=psync --direct=1 --numjobs=\$NJOBS --runtime=180 --time_based --norandommap --randrepeat=0



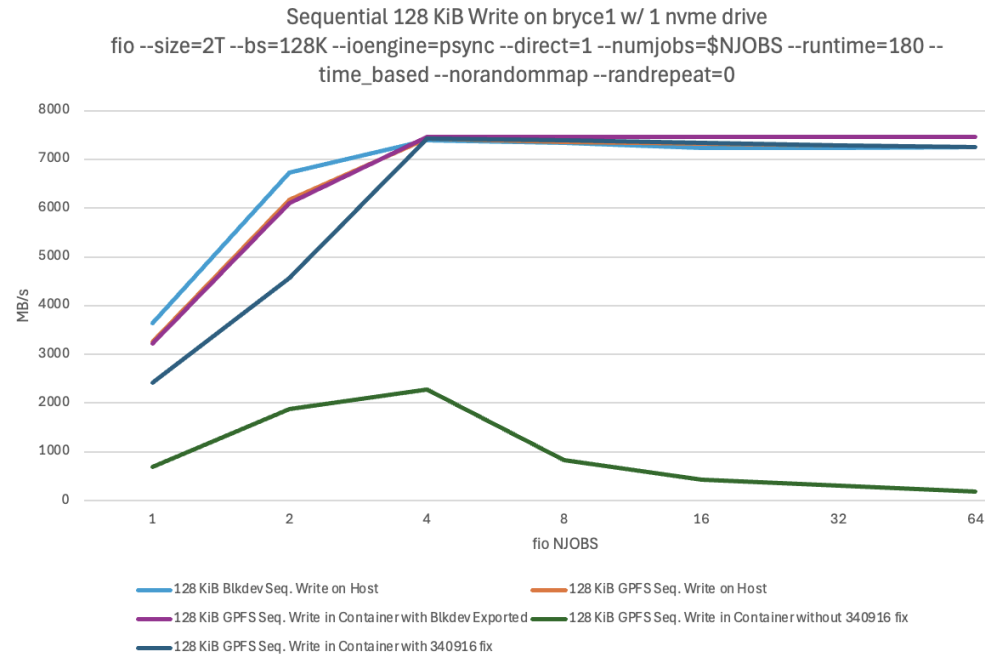
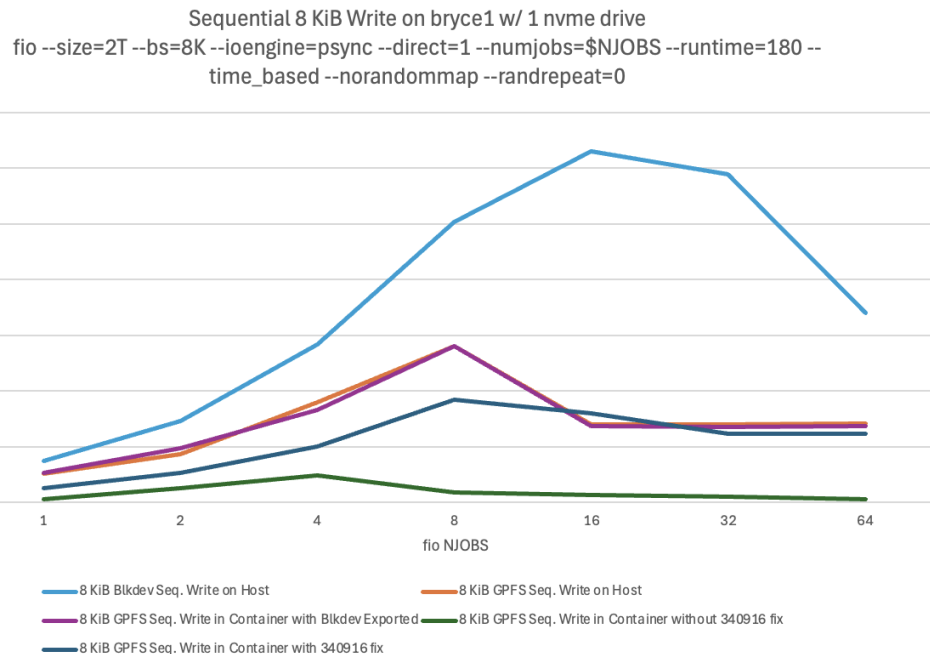
Sequential 128 KiB Read on bryce1 w/ 1 nvme drive
fio --size=2T --bs=128K --ioengine=psync --direct=1 --numjobs=\$NJOBS --runtime=180 --time_based --norandommap --randrepeat=0



Performance Improvement in 5.2.2 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)

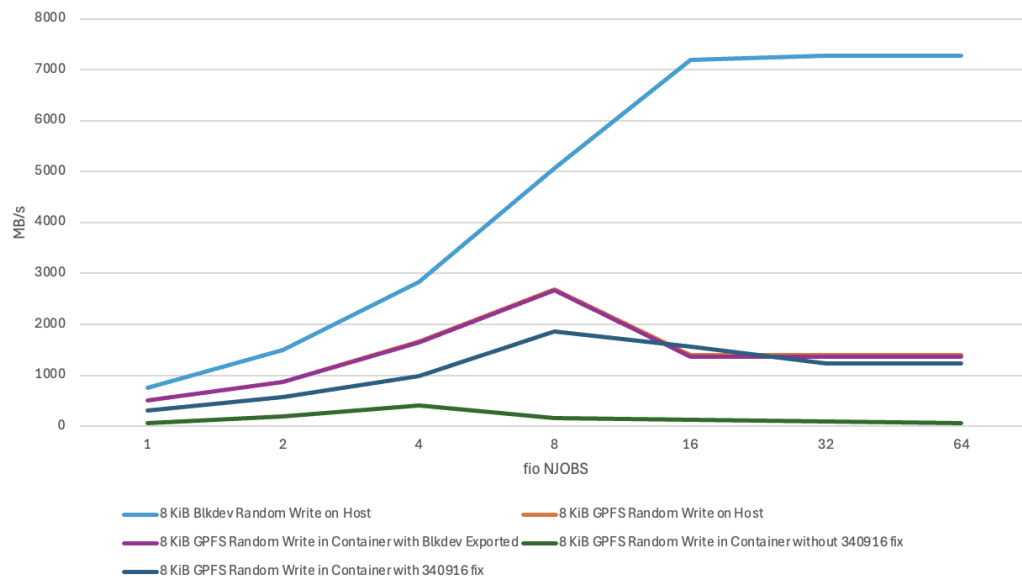


Performance Improvement in 5.2.2 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)

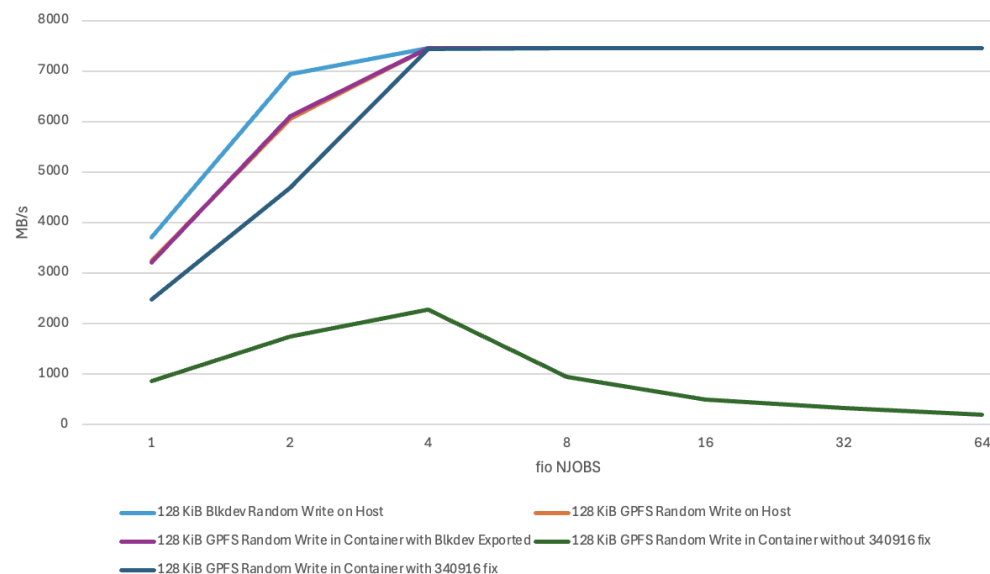


Performance Improvement in 5.2.2 to Address Cgroups Permissions Issues (Containers and VMs – Story 340916)

Random 8 KiB Write on bryce1 w/ 1 nvme drive
fio --size=2T --bs=8k --ioengine=psync --direct=1 --numjobs=\$NJOBS --runtime=180 --
time_based --norandommap --randrepeat=0



Random 128 KiB Write on bryce1 w/ 1 nvme drive
fio --size=2T --bs=128K --ioengine=psync --direct=1 --numjobs=\$NJOBS --runtime=180 --
time_based --norandommap --randrepeat=0



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

The ior-read-hard Benchmark Challenges and Action Plan

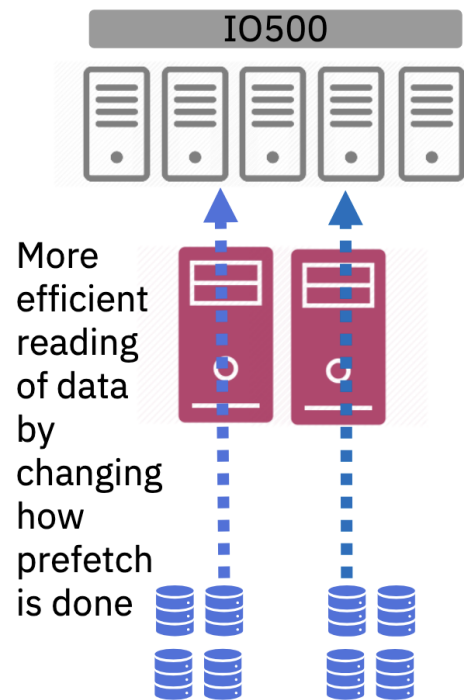
- **What makes ior hard read so hard?**

- 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:

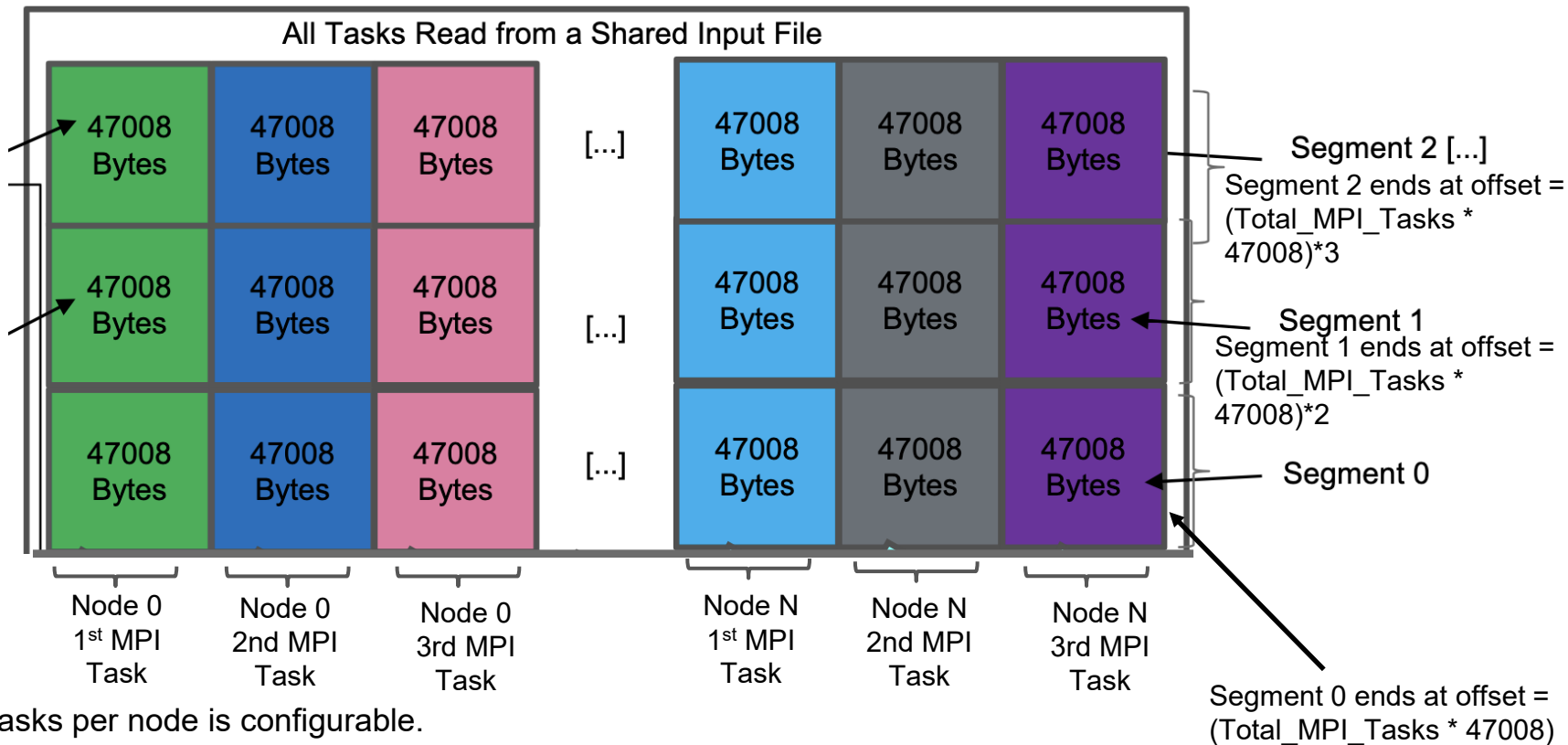
<https://github.com/hpc/ior/issues/390>



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:

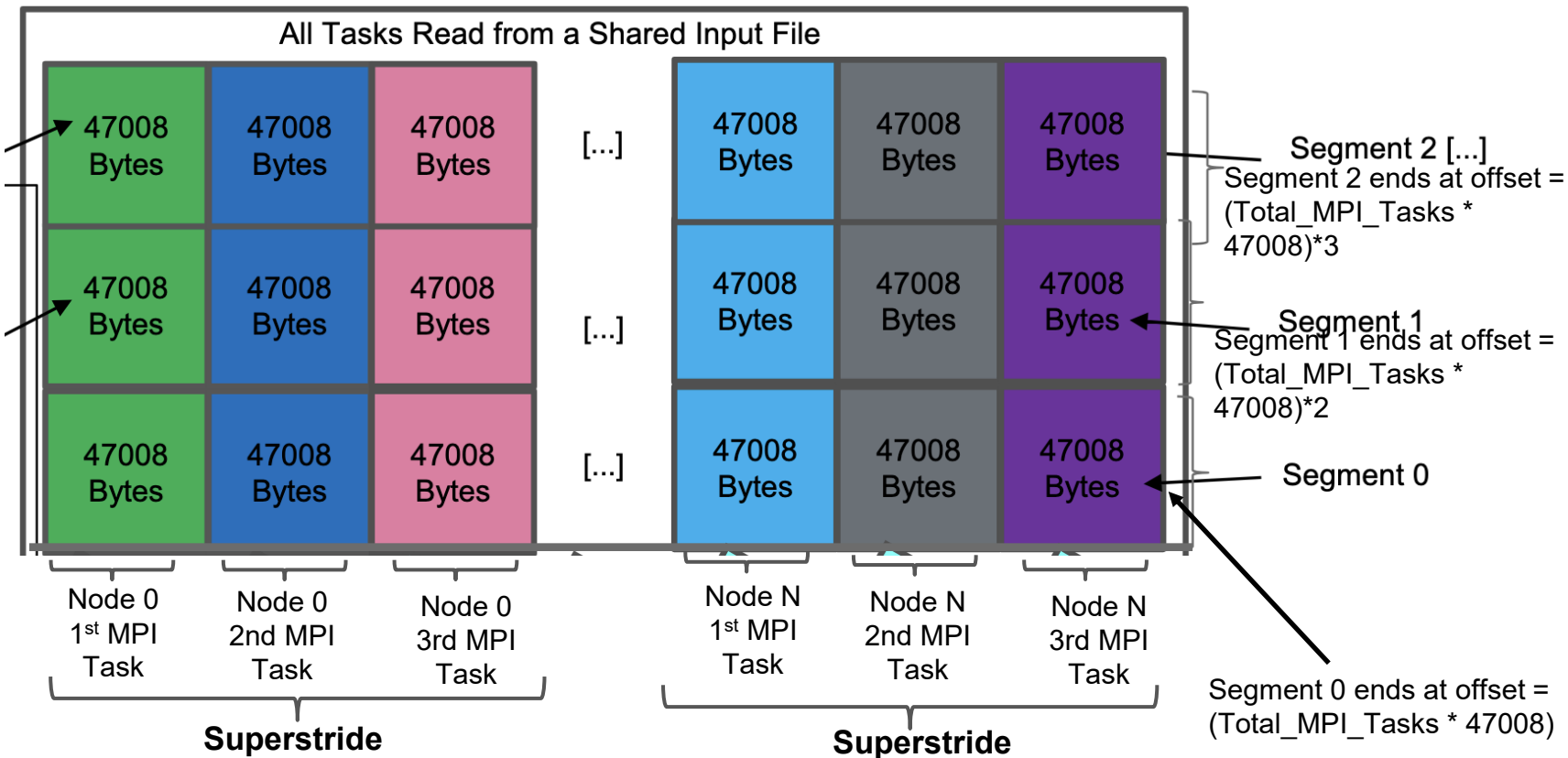
1. **Contiguous** (in terms of MPI task IDs) as depicted in this slide (which gives the opportunity to coalesce prefetches) **or**
2. **Round-robin** (e.g., with 4 tasks per node and 8 nodes, the first node has tasks 0, 8, 16, and 24, etc.)

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**” coalesces prefetches across instances doing strided reads, providing more efficient use of network bandwidth.

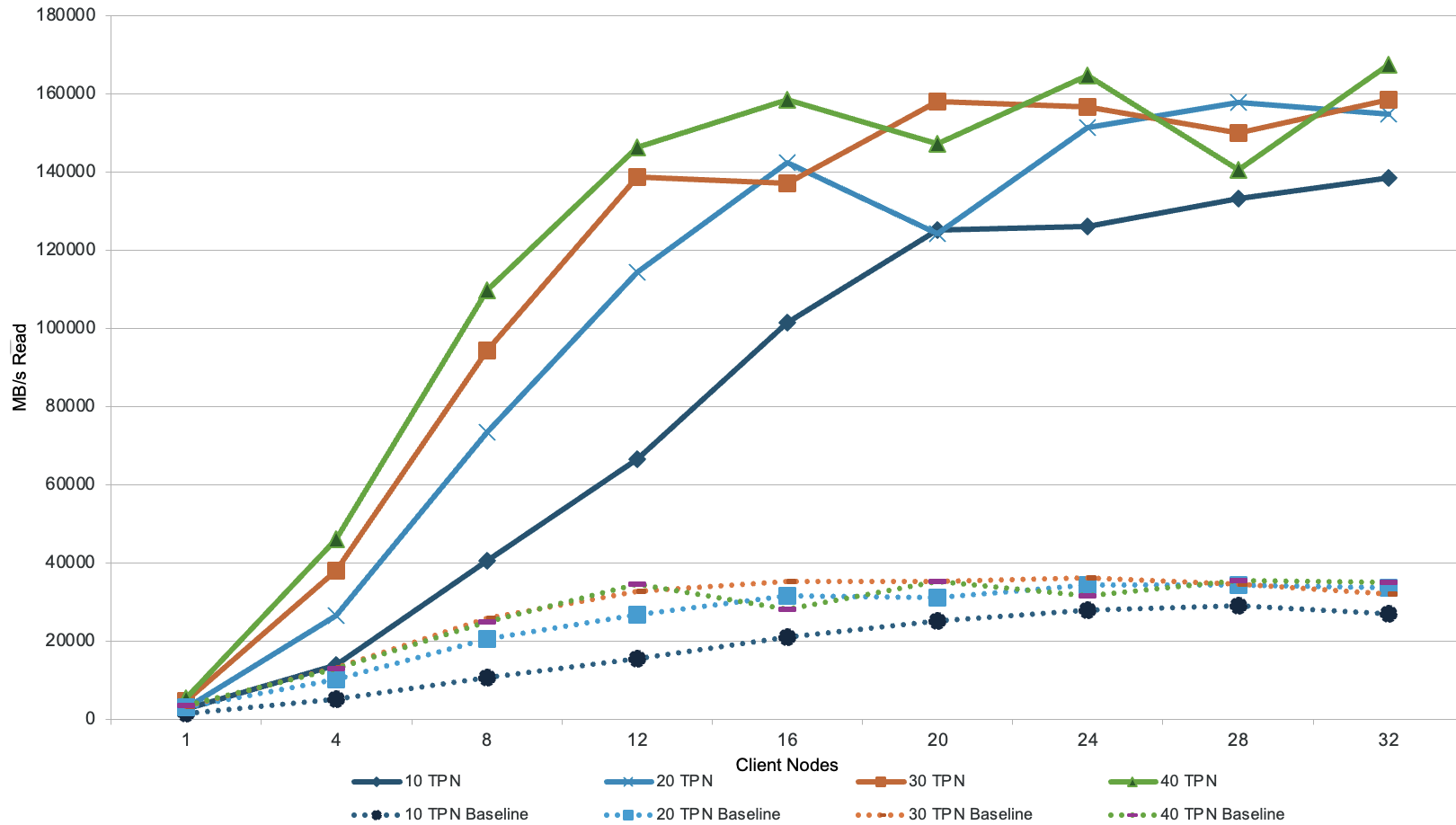
Contiguous allocation of MPI Tasks required to enable superstride prefetching

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

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

ior-hard-read improvement with superstride prefetching

Enhingen ISS 6000 - bb8nvme fs (1 BB)
io500 version of ior-hard-read - Current Lab Perf (5.2.3-1 candidate) compared to 5.2.1 G/A



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**

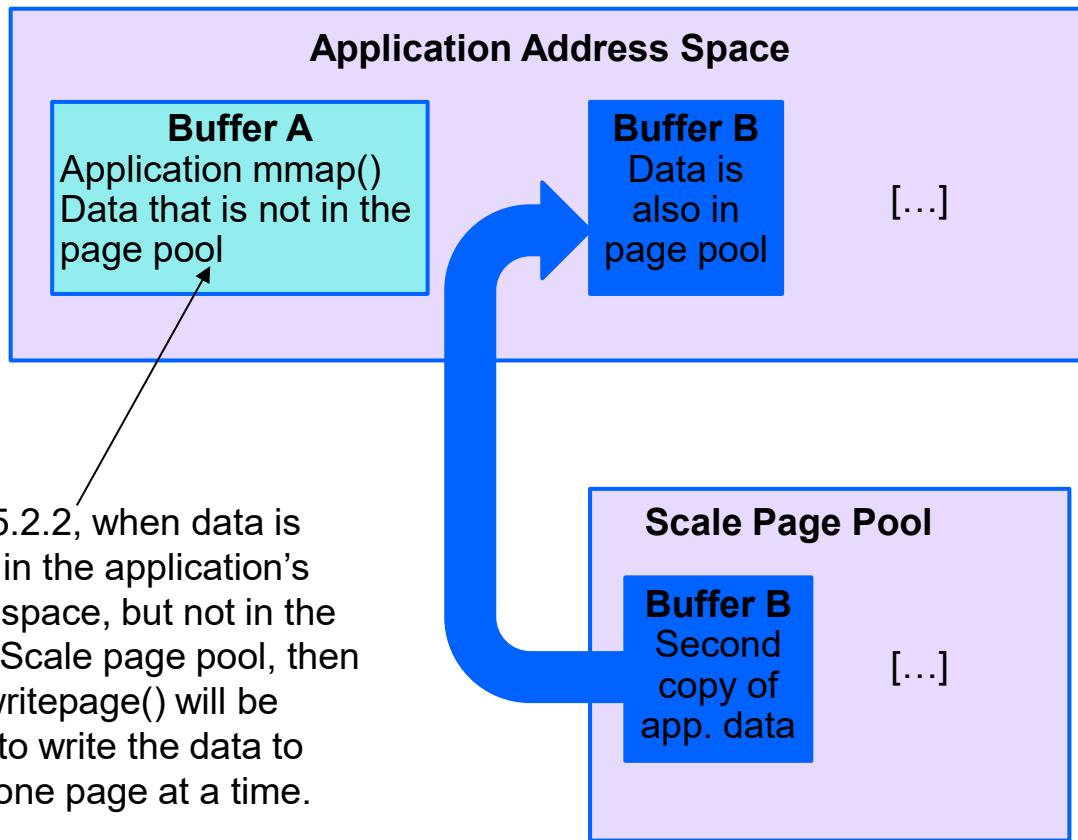
Mmap: The Basics

- 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 added to provide better write performance when mmap-ed data to be written is not in the page pool: Storage Scale will keep a list of virtually contiguous mapped pages and attempt to write up to a full filesystem block in a single I/O operation.
- We 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 write performance can be obtained because we can write larger chunks (ideally full filesystem blocks) from the page pool.
- In 5.2.2, a new `gpfs_i_writepages()` 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.

Example mmap Sequential Write Performance Test with iotest with Scale 5.2.2

```
iotest -r 4096k -s 320g -i 0 -C -B -+u -+m client.list.iotest -+n -w -t 16
```

```
Iotest: 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: iotest -r 4096k -s 320g -i 0 -C -B -+u -+m  
client.list.iotest -+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 iotop 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
```

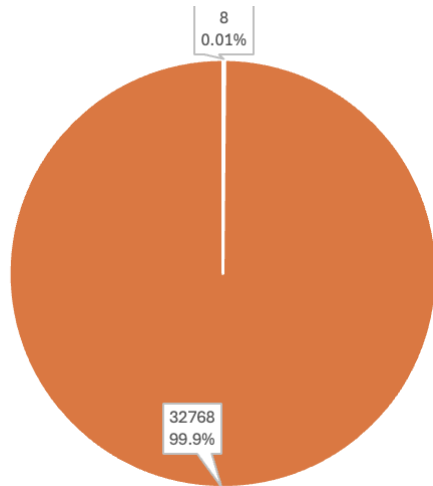
Scale GA 5.2.2: 8.7 GB/s write throughput

Scale GA 5.2.1: 3.4 GB/s write throughput

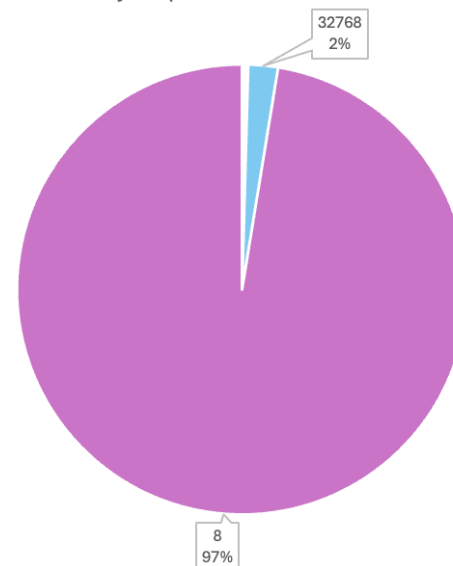
Comparing Application Write I/O Sizes for mmap Write Scale 5.2.2 vs 5.2.1

- Sampling 'mmdia -iohist' for the IO requests submitted by the clients:
 - 5.2.2 (left-side pie chart): effectively all writes are full block I/Os (32768 sectors = 16M)
 - 5.2.1 (right-side pie chart): effectively all writes are 4KiB

Sample of sector sizes written from 'mmdia --iohist' from iозone mmap write test on 5.2.2
Each sector is 512 bytes



Sample of sector sizes written from 'mmdia --iohist' from iозone mmap write test on 5.2.1
Each sector is 512 bytes (97% of all writes are 8 sectors or 4KiB)



Thanks!