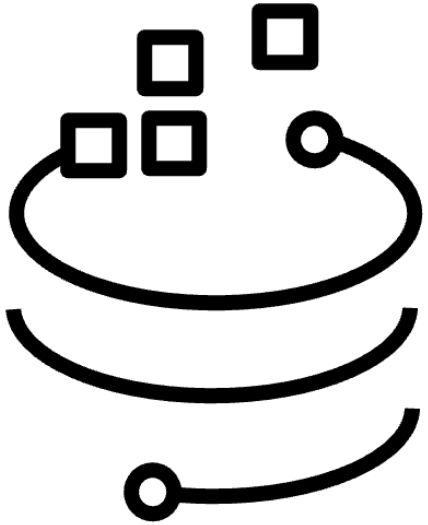
SC23 Scale Performance Update



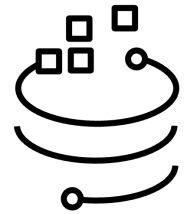
Storage Scale

Storage Scale User Group Meeting @ SC23
Denver, CO – Nov 12, 2023

Presented by: John Lewars (IBM)

Additional Slides Contributions by: John Divirgilio, Pidad D’Souza, Felipe Knop, Hai Zhong Zhou

Disclaimer



IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice 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. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

IBM reserves the right to change product specifications and offerings at any time without notice. This publication could include technical inaccuracies or typographical errors. References herein to IBM products and services do not imply that IBM intends to make them available in all countries.

Agenda

- ESS6000 Update
- Client-Side GNR and first stage of related improvements on the ESS3500
- Update on io500 work – including new features used for io500 runs
- New COW-related performance improvements

IBM Storage Scale System 6000

The simplest and fastest way to deploy a global data platform for AI and Hybrid Cloud workloads

Manage next generation and traditional workloads with simultaneous high-performance file and object data access services to the same data

Optimize local and remote access and simplify DR with global hybrid cloud data services

Speed access to critical data with Intelligent and automated data management services

Protect against cyber threats with Cyber-secure data services for unstructured data including end to end encryption and identification to recovery

Lower RTO times with proven data protection and data resiliency services

Scale-up to PBs and scale-out to YBs for GB/s+ performance and capacity to manage your entire data ecosystem with lower cost and enterprise security and resiliency your business requires

Scale from 1 to 1000s of system

Lab Measurements:

up to 310 GB/s seq. read* per system (building block) - ~2.5X ESS3500

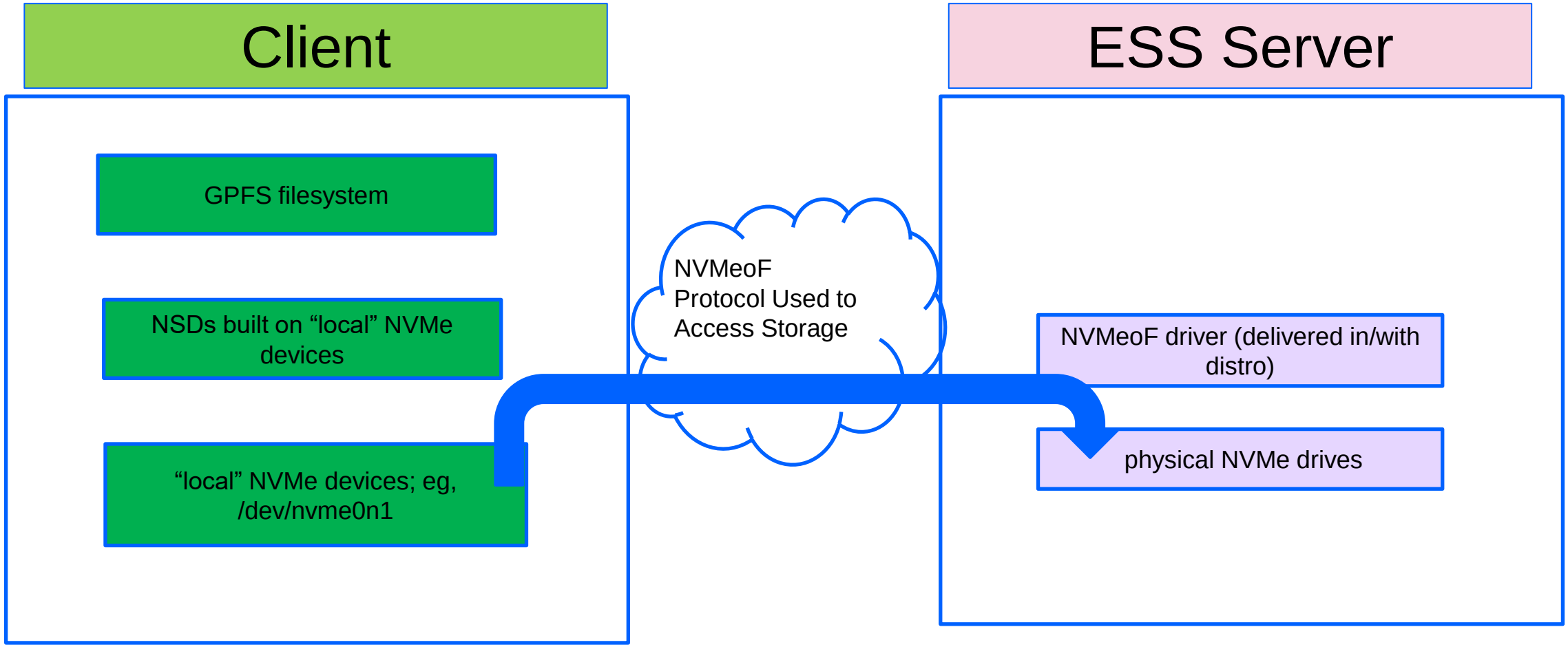
up to 155 GB/s seq. write per system (building block) - ~2.6X ESS3500

up to 13M 4KiB random read IOPS** per system (building block)

* Update: Chart was updated to reflect current lab read performance, as of Dec. 4/23, an improvement over what was measured as of Nov. 12/23.

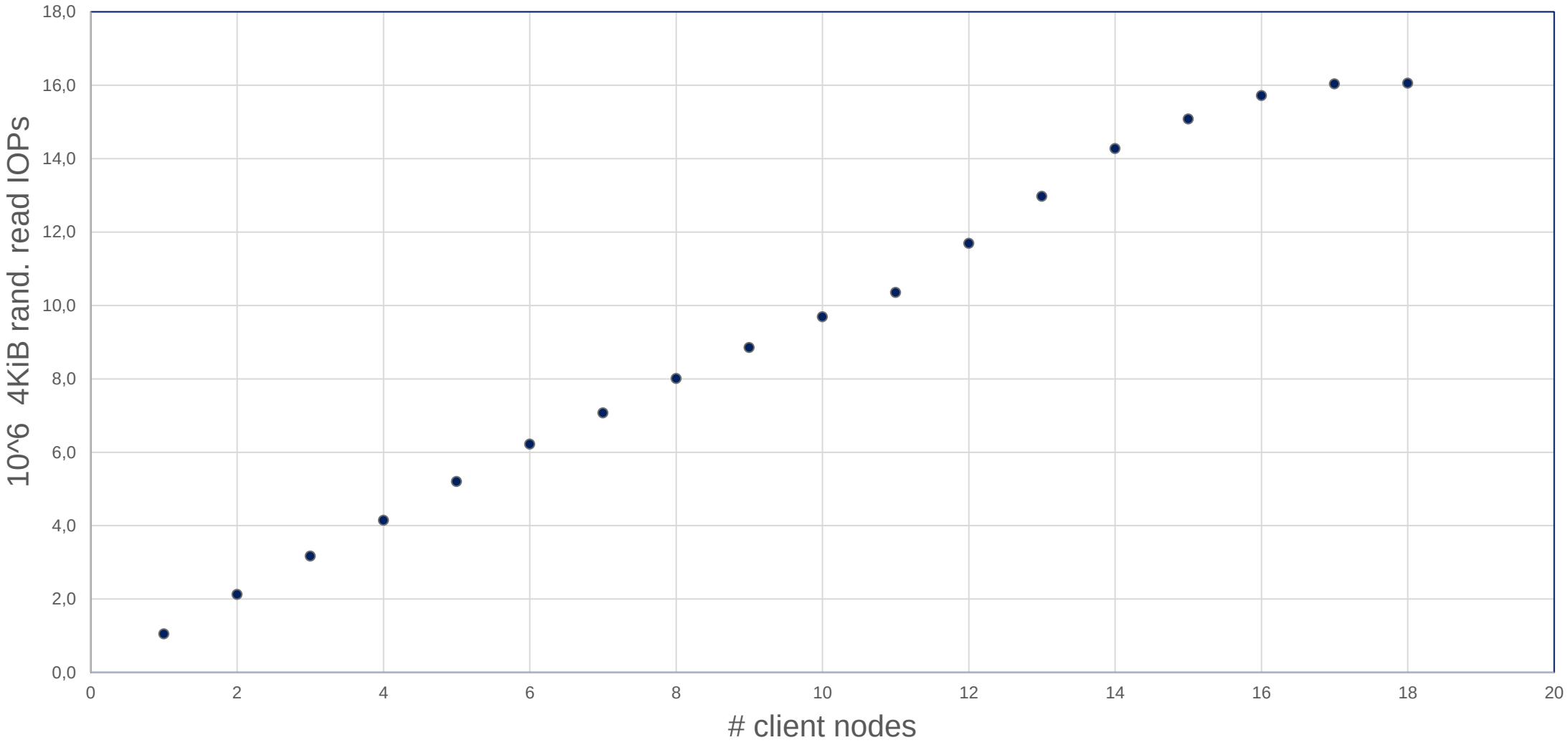
**13 M IOPS with 10 clients and 24 of the BB's 48 NVMe drives, using the NVMeoF protected tier capability supported via special request as described in the Storage Scale System 6000 Data Sheet

Upcoming NVMeoF Solution for ESS



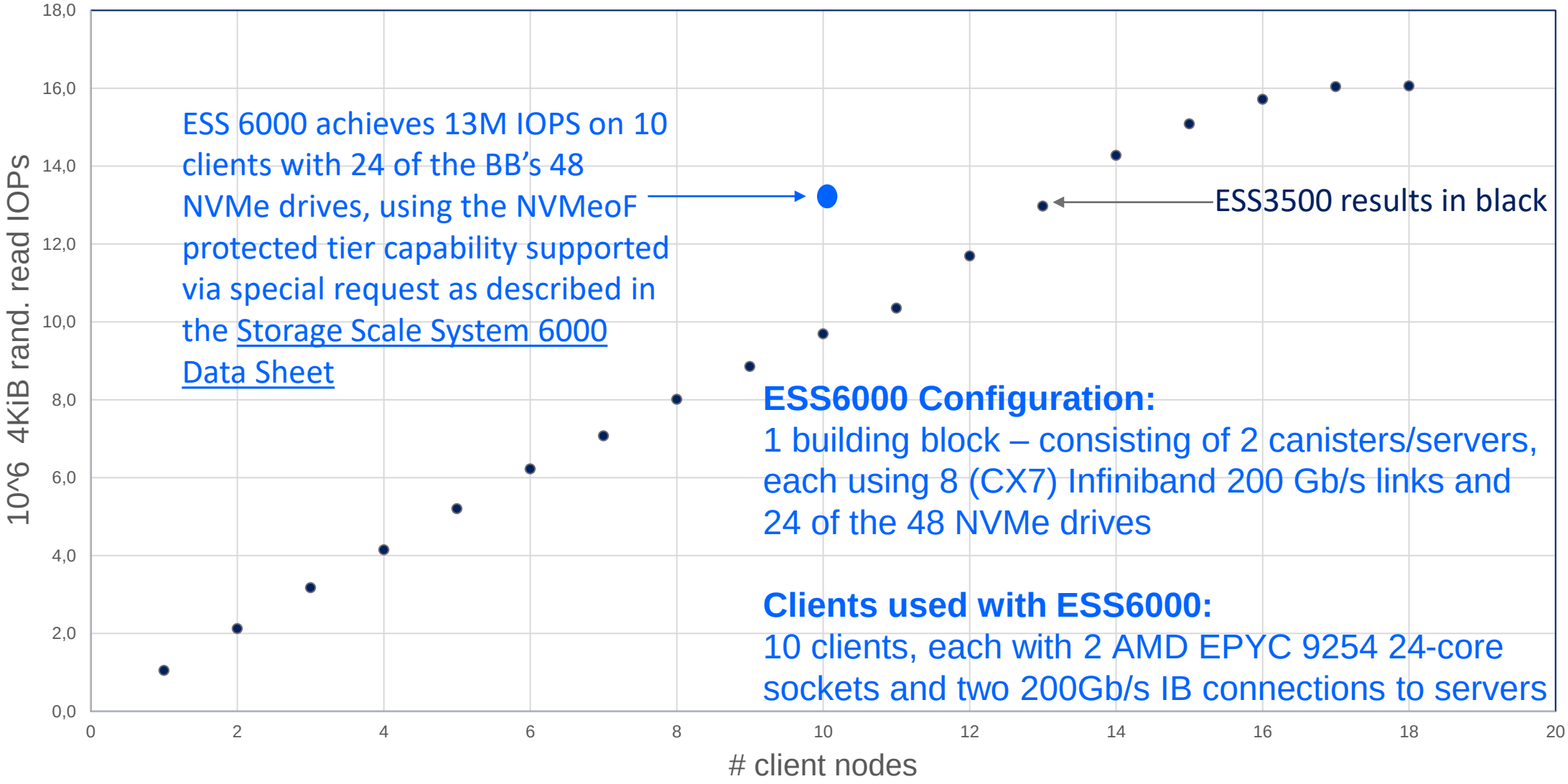
4KiB random read IOPs (Direct IO) Using NVMeoF Exported Drives

ESS 3500 Configuration: 24 NVMe drives, 4 CX6/canister (1 port used per adapter)
Clients used with ESS3500 : 1 CX6 nr-io-queues=3, native MP with iopolicy=roundrobin



4KiB random read IOPs (Direct IO) Using NVMeoF Exported Drives

ESS 3500 Configuration: 24 NVMe drives, 4 CX6/canister (1 port used per adapter)
Clients used with ESS3500 : 1 CX6 nr-io-queues=3, native MP with iopolicy=roundrobin



Client-side GNR Architecture

Next generation ECE / ESS technology for high performance exa-scale disaggregated storage

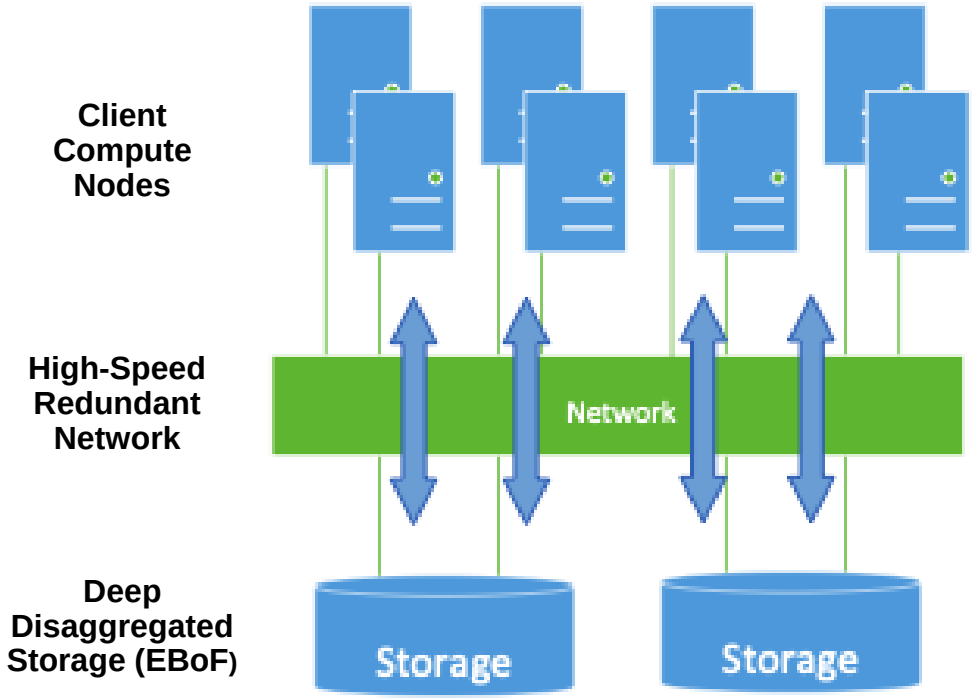
- **Improved performance:** Client node fetches data directly with zero-copy RDMA / GPU Direct Storage. Efficient lock free structures
- **Improved availability:** Direct client access to **all** data. Erasure encoding across racks or cloud availability zones.
- **Reduced client impact:** Limit on threads & memory. Optional DPU offload or dedicated server nodes.
- **Reduced cost**: Deep cost-effective storage density. Increased storage capacity per node

Client node provides erasure encoding across storage nodes

Storage may be converged or disaggregated

All Clients can directly read and write all storage

Client caches mapping of virtual disks to physical location



Storage nodes export aggregated disks spanning local devices

Checksum encodes data version to detect stale mappings

End-to-end checksum reliability

IBM Elastic Storage System 3500

The simplest and fastest way to deploy a global data platform for AI and Hybrid Cloud workloads

Manage next generation and traditional workloads with simultaneous high-performance file and object data access services to the same data

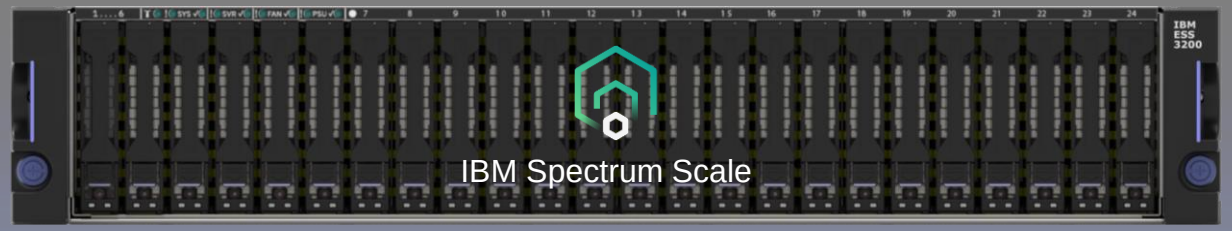
Optimize local and remote access and simplify DR with global hybrid cloud data services

Speed access to critical data with Intelligent and automated data management services

Protect against cyber threats with Cyber-secure data services for unstructured data including end to end encryption and identification to recovery

Lower RTO times with proven data protection and data resiliency services

IBM Breaks Storage Performance Barriers for AI and Hybrid Cloud Workloads and Accelerates Recovery Times for Cyber Threats



up to **500+YB** per cluster
up to **30M IOPS** per rack
up to **126 GB/s** per building block
up to **1.8 TB+**/s per rack

Read bandwidth with 4 HDR200 adapters per canister improved from 91 GB/s to 126 GB/s in ESS 6.1.8.
Performance improved to 117 GB/s with 3 HDR200 adapters, and 100 GB/s with 2 HDR200 adapters.

IBM Elastic Storage System 3500 – Memory Bandwidth and CPU Usage Measurements from Development Builds

- Below are measurements the single checksum feature, showing how enabling this feature changes CPU usage and memory bandwidth usage on both the client and server side.
- These measurements were done with nine HDR200 connected clients

		5.1.5.1 GA	trailerchecksum** 5.1.5.1
IOR Seq Read BW		90 GB/s	125 GB/s
Client	CPU Idle	86%	80%
	Mem BW	35 GB/s	50 GB/s
	Mem Fact	~3.5	~3.5
Server	CPU Idle	70%	88%
	Mem BW	144 GB/s	134 GB/s
	Mem Fact	~3.2	~2.2

**These measurements were done originally as part of work on client-side-GNR in the 5.1.5 time frame. The final shipping code on ESS3500 delivers 126 GB/s sequential read throughput.

io500 Work and Plan

- The io500 benchmark suite has received an increasing amount of focus in recent years and now provides an important set of performance metrics that the Spectrum Scale Research and Development teams are working on.

- One of the goals of io500 is to measure ‘hard’ workloads to determine the worst possible performance that may be achieved across all possible I/O patterns.

- By improving the performance of the ‘hard’ io500 benchmarks, we expect to improve the performance of challenging modern workloads, following this plan:

  1. Focus on the low performing *or degraded benchmarks* first, determine bottlenecks, and then apply existing tuning parameters to improve performance.
  2. Develop new tuning parameters/hints that allow us to target focus workloads.
  3. Improve heuristics so that we can automatically adapt to workloads without specific tuning parameters or hints so that future runs of the benchmark are able to achieve optimal performance without explicit hints/tuning.

IO500 submissions from IBM for SC23 (all with IBM Storage Scale 5.1.9.0)

Benchmark Storage Scale 5.1.9.0	IBM Tucson(EPYC 7302P 16-Core, 10 clients, 8MB Blocksize, Infiniband) – ESS3500 2BBs	IBM Ehningen (EPYC 9274F 24-Core 10 clients (4MB Blocksize, Infiniband) – ESS3500 3BBs	IBM Cloud HPC (Xeon Platinum 8260 48-core, 199 nodes, 4MB Blocksize, Ethernet 100Gib/s) - SNC
ior-easy-write	106.6	140.6	757.9
mdtest-easy-write	231.5	288.5	3163.9
ior-hard-write	51.3	64.8	367.3
mdtest-hard-write	34.3	33.5	28.3
find	4965.6	3912.5	13502.2
ior-easy-read	205.0	227.1	787.1
mdtest-easy-stat	561.4	824.5	3558.7
ior-hard-read	27.0	27.9	172.6
mdtest-hard-stat	303.4	700.6	1993.1
mdtest-easy-delete	190.4	316.7	1640.1
mdtest-hard-read	373.5	464.9	998.9
mdtest-hard-delete	32.7	31.9	23.1
Bandwidth GiB/s	74.2	87.1	441.0
kIOPS	250.6	317.2	868.6
TOTAL Score	136.4	166.3	618.9

IO500 submission – Improvements on the IBM Tucson System

Benchmark Storage Scale 5.1.9.0	IBM Tucson(EPYC 7302P 16-Core, 10 clients, 8MB Blocksize, Infiniband) – ESS3500 2BB – ISC23	IBM Tucson(EPYC 7302P 16-Core, 10 clients, 8MB Blocksize, Infiniband) – ESS3500 2BB – SC23	Percent Improvement
ior-easy-write	107.8	106.6	-1.1
mdtest-easy-write	184.5	231.5	25.5
ior-hard-write	49.9	51.3	2.8
mdtest-hard-write	33.6	34.2	1.8
find	4123.1	4965.6	20.4
ior-easy-read	204	205.0	0.5
mdtest-easy-stat	384.3	561.4	46.1
ior-hard-read	27	27.0	0.0
mdtest-hard-stat	297.5	303.4	2.0
mdtest-easy-delete	190.9	190.4	-0.3
mdtest-hard-read	399.8	373.5	-6.6
mdtest-hard-delete	26.4	32.7	23.9
Bandwidth GiB/s	73.8	74.2	0.5
kIOPS	222	250.6	12.9
TOTAL Score	128	136.4	6.1

Performance gains result from the following:

- Code changes/tuning that reduce the mutex contention (OpenFileHashTabMutex) seen in mdtest-easy-stat benchmark
- Code changes/tuning to improve how efficiently tokens are managed for mdtest-hard-delete benchmark

IO500 submission – Improvements on the IBM Ehningen Relative to IBM Tucson System

Benchmark Storage Scale 5.1.9.0	IBM Tucson(EPYC 7302P 16-Core, 10 clients, 8MB Blocksize, Infiniband) – ESS3500 2BB – ISC23	IBM Ehningen (EPYC 9274F 24-Core 10 clients (4MB Blocksize, Infiniband) – ESS3500 3BB	Percent Improvement
ior-easy-write	106.6	140.5	31.8
mdtest-easy-write	231.5	288.5	24.6
ior-hard-write	51.3	64.8	26.3
mdtest-hard-write	34.2	33.5	-2.3
find	4965.6	3912.5	-21.2
ior-easy-read	205.0	227.1	10.8
mdtest-easy-stat	561.4	824.5	46.9
ior-hard-read	27.0	27.9	3.2
mdtest-hard-stat	303.4	700.6	130.9
mdtest-easy-delete	190.4	316.7	66.3
mdtest-hard-read	373.5	464.9	24.5
mdtest-hard-delete	32.7	31.9	-2.4
Bandwidth GiB/s	74.2	87.1	17.5
kIOPS	250.6	317.2	26.5
TOTAL Score	136.4	166.3	21.9

Performance gains, relative to the IBM Tucson submission, result from:

- 3 ESS3500 building blocks used instead of 2 building blocks
- The Ehningen client nodes with EPYC9274F chips (November 2022 launch) are more powerful than the Tucson EPYC7302P (August 2019 launch) nodes
- Just like the IBM Tucson submission, (OpenFileHashTabMutex) mutex contention in mdtest-easy-stat is mitigated through code changes/tuning

IBM Cloud HPC Submission Details

IBM Cloud HPC system for HPC solution Development, Test, and enhancements. IBM Research and IBM Systems EDA teams use for Chip Design, Verification and Simulation works. IBM Storage Scale team uses it for Development and Performance engineering and Scaling works. In addition, system is also used for Customer POCs and co-development activities.

Bare Metal Server configuration: 199 nodes: Intel® Xeon® 8260 (2 Sockets), NVMe Drives

Filesystem: IBM Storage Scale 5.1.9.0

```
maxStatCache 128K
nsdMinWorkerThreads 3842
nsdMaxWorkerThreads 3842
pagepool 100G
nBucketGroups 1024
preferDesignatedMnode yes
fsyncIsGlobal no
numactlOptioni 0   <= Binding mmfsd to same socket (0) as adapter
numactlOptionN 0
dataShipClientBuffersPerServer 10  <- For ior-hard-write
dataShipClientBufferPct 50
dataShipServerBufferPct 10
autoCompactDir 0          <- For mdtest-hard-delete
fgdlTokenBatchAcquire 1
```

IO500 – SC23 results

Benchmark	SCORE
ior-easy-write	757.9
mdtest-easy-write	3163.8
ior-hard-write	367.3
mdtest-hard-write	28.3
find	13502.2
ior-easy-read	787.1
mdtest-easy-stat	3558.7
ior-hard-read	172.6
mdtest-hard-stat	1993.1
mdtest-easy-delete	1640.1
mdtest-hard-read	998.9
mdtest-hard-delete	23.1
Bandwidth GiB/s	**441.0**
kIOPS	**868.6**
TOTAL Score	**618.9**

10 Tasks Per Node

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

* Binding to the same socket as adapter is located

Details on IBM's Recent ESS io500 Submissions

Details Regarding IBM's ESS3500 Cluster io500 submission for ISC23:

2x ESS 3500 Building Blocks, 2 servers/canisters per BB with 8MB Blocksize File System:
Four HDR-Infiniband links per canister
Single socket 48-core processor per canister
24x Samsung NVMe Drives per building block, shared across both canisters in each BB

10x Lenovo AMD clients:
One HDR200-Infiniband connection per client (Tucson) / two HDR200 connections (Ehningen)
Single socket - AMD EPYC 7302P 16-Core Processor per client (Tucson) / 24 core 9274F HDR200 (Ehningen)

256GB Memory per client (Tucson) / 192 GB/s memory per client (Ehningen) – page pool is 48GB / 50GB

Clients' mmchconfig Tuning
/usr/lpp/mmfs/samples/gss/gssClientConfig.sh -M 65536
maxStatCache 131072
nBucketGroups 1024
preferDesignatedMnode yes
fsyncIsGlobal no
dataShipClientBuffersPerServer 50
dataShipClientBufferPct 50
dataShipServerBufferPct 50
autoCompactDir 0
fgdlTokenBatchAcquire 1 # not used for IBM Ehningen submission because it requires gpfs.base 5.1.9.0

IBM Tucson: Manager node role assigned to all 10 clients and all four storage nodes
ESS 6.1.8.2 (RHEL 8.6) + Spectrum Scale upgraded to 5.1.9.0 on all canisters

IBM Ehningen: ESS 6.1.8.2 (RHEL 8.6) with Spectrum Scale left at 5.1.8.1 on all canisters (Ehningen)
All storage nodes and clients not used for io500 assigned manager node role

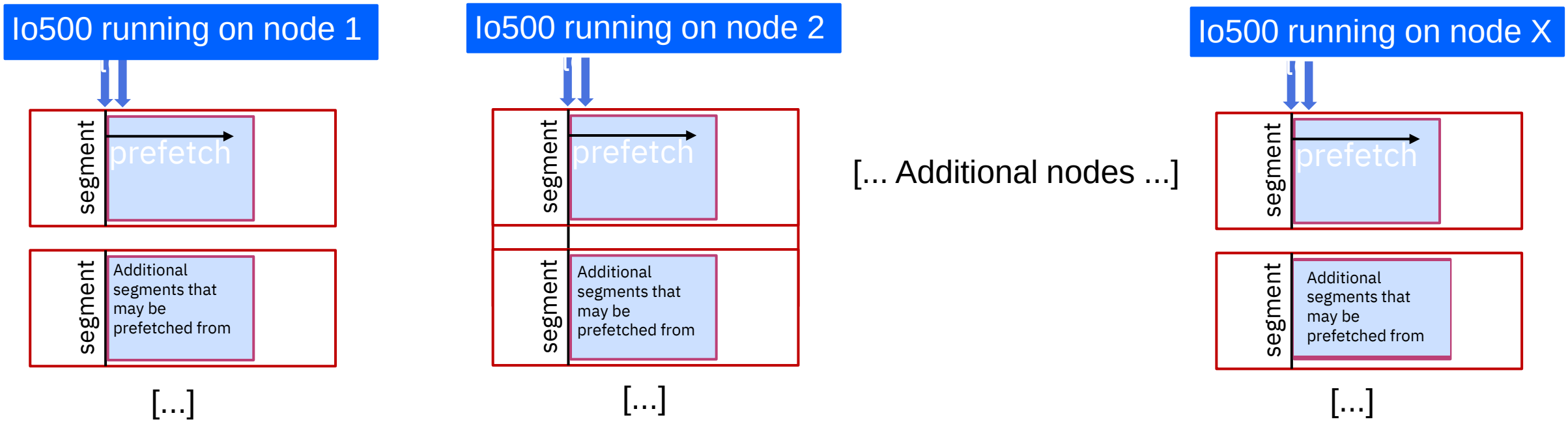
mdtest Hard Deletes - Prefetch Inode Lookups for Deletes (1/3)

- mdtest workload characterization
  - mdtest Hard consists of four phases: 1) write, 2) stat, 3) read, 4) **delete**
  - Write phase creates files of size 3901 bytes in a single shared directory, which the **delete** phase will remove
- Analysis
  - Linux serializes updates on a directory. A single client node can only do one delete at a time for a given directory
  - On each node, the stat, read, and delete phases operate on the set of files that another node operated on in the previous phase; also the pfind benchmark will access these files after mdtest-hard-write runs
  - Since the task mapping rotates between phases, each file delete generally needs to revoke the inode token and read the inode (unless the job is only run on a single node)

- Proposed approach – **Metadata (inode lookup) Prefetch for Deletes**
  - Observation: since the files being deleted should all have been created on one node, the inodes should have been allocated from the same (or small set of) segment(s) of the inode allocation map
  - We can predict which files will be deleted next when we observe files from the same segment of the allocation map being deleted
  - This predictive approach can be used to prefetch required tokens and read inodes in parallel, which mitigates the serialization of lookups that occurs deleting files in the same directory on a node in Linux

- This predictive lookup Implementation was enabled in 5.1.3 and lab measurements showed up to a 2X improvement
  - mmchconfig option **enableIASegPrefetch** (enabled, set to 1, by default)

mdtest Hard Deletes - Prefetch Inode Lookups for Deletes (2/3)



Multiple segments may be dedicated to each node (assuming sufficient inodes are free), but, since mdtest-hard-delete operates on only one directory, only one segment is typically used at a time during the file creation phase (the total number of segments used depends on how many files are created, the number of inodes per segment, and whether sufficient inodes are available to assign all clients dedicated segments)

To get the performance improvements we've observed in the lab, here are two recommendations:

1. There must be sufficient free inode segments so that the (mdtest-write-hard) create phase of the benchmark is able to allocate sufficient dedicated inode segments to all clients involved
2. Designated metanode function must be enabled: **echo 999 | mmchconfig preferDesignatedMnode=yes**

mdtest Hard Deletes - Prefetch Inode Lookups for Deletes (3/3)

Metadata (inode lookup) prefetch for deletes can be limited by serialization in FineGrainDirectoryLocking (FGDL) token acquisition flows. Storage Scale 5.1.9 adds an undocumented token batching technique to efficiently acquire a wider range of FGDL ByteRange(BR) tokens. Such batched requests are sent in a single RPC to the token manager. The token manager then grants this wider range of non-conflicting FGDL BR tokens.

- Speeds up FGDL BR token acquisition
- Minimizes the RPC traffic on the cluster

Set **“echo 999 | mmchconfig fgdlTokenBatchAcquire=1”** (defaults to 0) on both clients and IO servers to enable the token batching function (in the future we intend to add a new hint for this feature, or enable it by default)

A large number of file deletions can drive directory compaction, which merges the directory blocks. As a result of compaction, we may end up with directory blocks containing inodes from multiple segments. Token batching performs optimally when a minimal number of segment numbers exist for the inodes in the block.

Set “**echo 999 | mmchconfig autoCompactDir=0”** (defaults to 1, which enables compaction) to avoid the merging of directory blocks. We intend to optimize compaction during intensive file deletion in the future.

In our lab measurements, we have observed **an improvement** of **~15-20% to mdtest-hard-delete phase** during io500 runs by making these two configuration changes.

Details on nBucketGroups Configuration Option

GPFS maintains both stat cache and file cache entries in an array of hash table entries, with the number of array elements defined by the undocumented mmchconfig option **nBucketGroups**. With more array elements, (OpenFileHashTabMutex) mutex contention serializing access to cached entries can be reduced. In Storage Scale 5.1.9, the limit on nBucketGroups is increased from 128 (the default) to 1024.

The limits on the caching of objects are defined by:
1. mmchconfig option: **maxStatCache**, which limits stat cache entries (limited to 100000000)

2. mmchconfig option: **maxFilesToCache**, which limits cached open file`s (limited to 100000000)

3. Shared segment memory for caching stat cache and file cache memory (see details here)

4. Token server memory needed to manage cached entries (see details here in a past Scale presentation)

Example of stat cache entries in array of hash tables

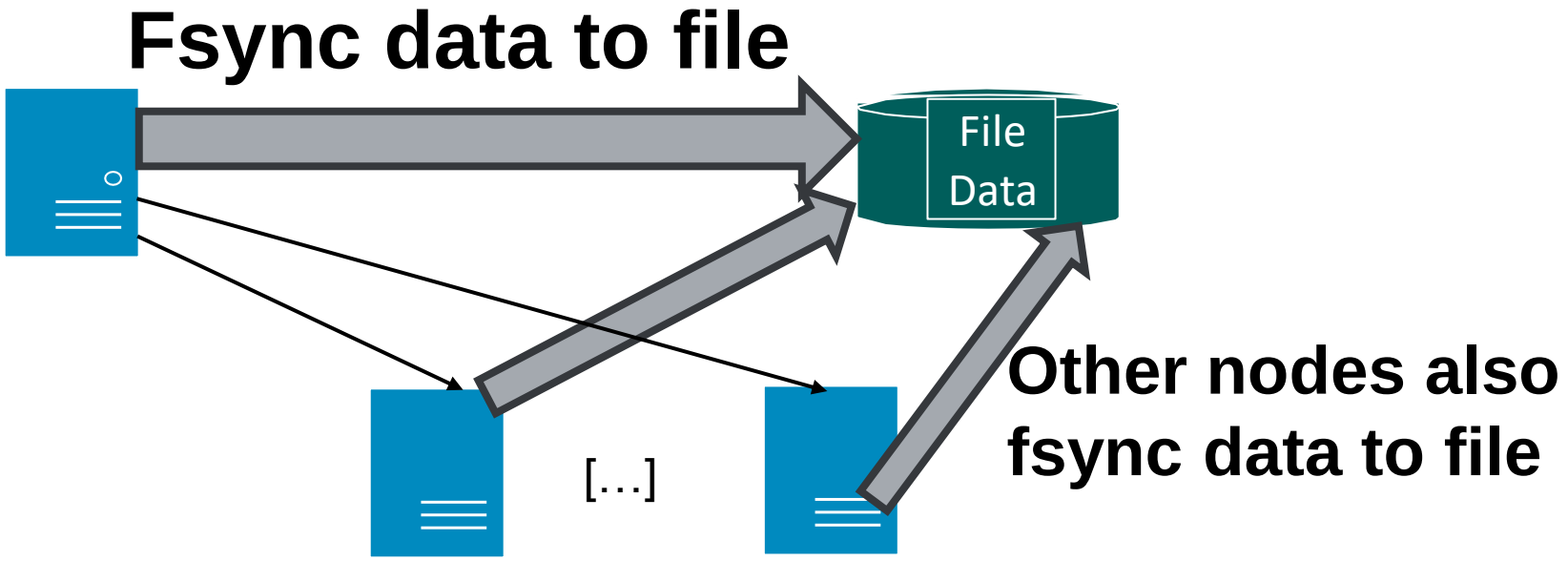
Bucket Group Number	**Hash Tables with Stat Cache Entries**			
BUCKETGROUP 0	Stat Cache Entry	Stat Cache Entry	Stat Cache Entry	[…]
BUCKETGROUP 1	Stat Cache Entry	Stat Cache Entry	Stat Cache Entry	[…]
[…]	[…]			
BUCKETGROUP N-1	Stat Cache Entry	Stat Cache Entry	Stat Cache Entry	[…]

New fsyncIsGlobal option added in Scale 5.1.7 (This option is Documented in 5.1.9 release)

What happens when fsync() is called to persist a file that resides on Storage Scale file system?

1. The node calling fsync() will flush any pending dirty data the calling thread has for the target file.

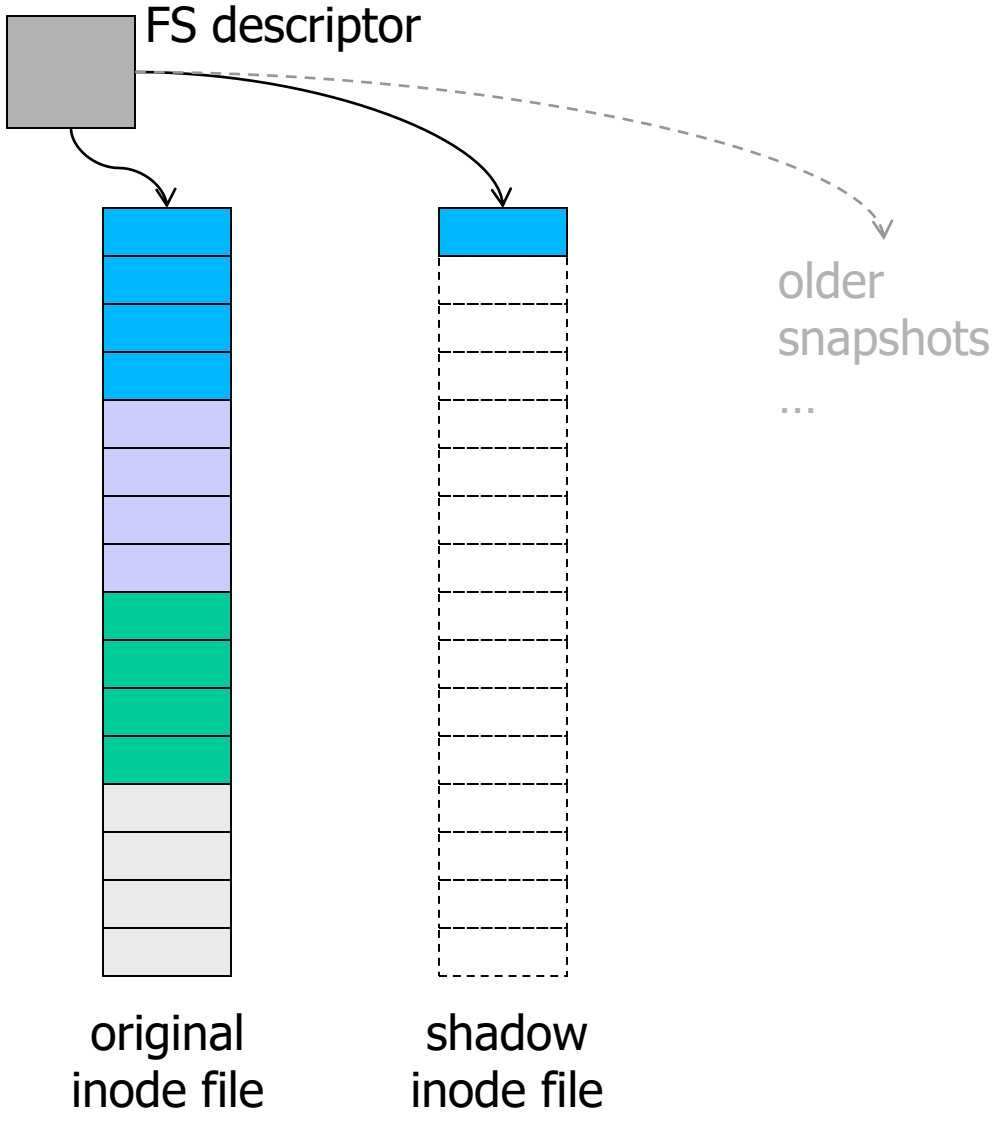
2. The standard for fsync() in a parallel file system is not defined, by default, Storage Scale takes a conservative approach and flushes any pending dirty data for the target file.



Since the standard for fsync() in a parallel file system is not defined, Storage Scale takes a conservative approach and flushes any pending dirty data for the target file, but this behavior (the second step depicted here) can be disabled as of 5.1.7.0 by setting:
mmchconfig fsyncIsGlobal=no –i # no mmfsd restart required

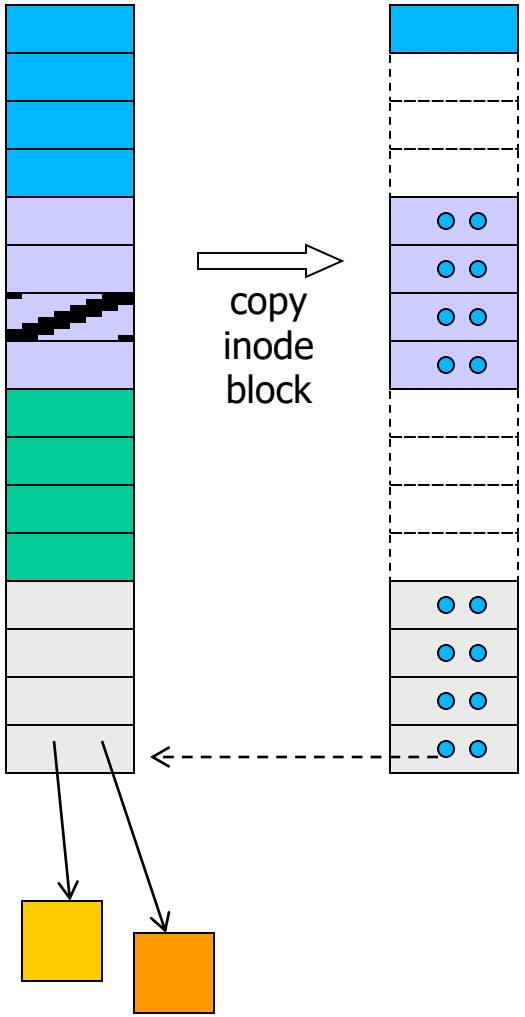
Details on new 5.1.9 COW Improvement

Snapshot Creation Details



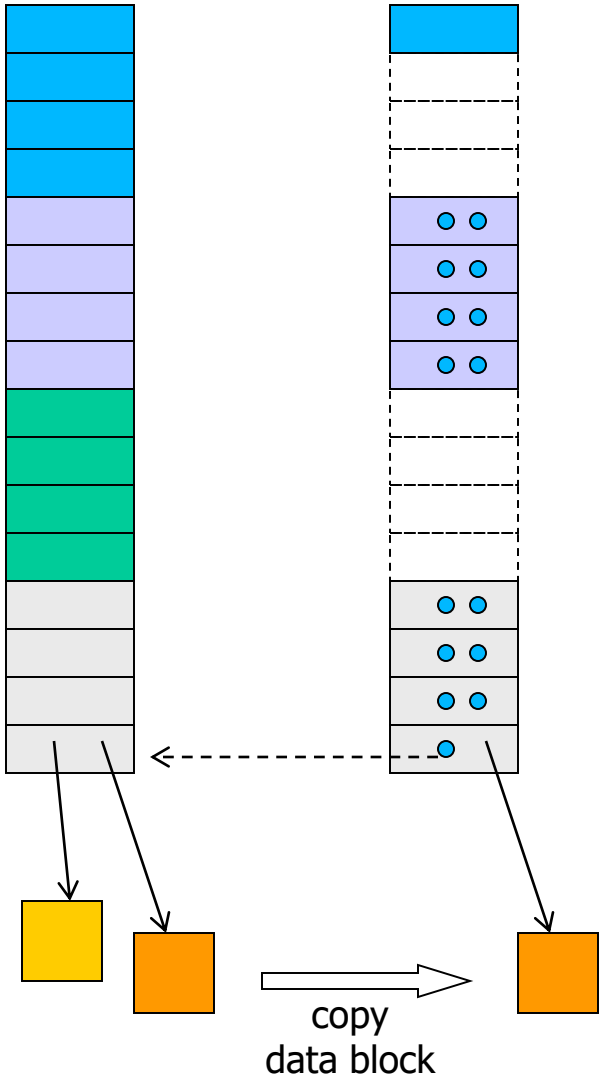
- Flush dirty data
- Quiesce FS operations
- Flush dirty data
- Create sparse shadow inode file
- Add entry to Fileset Metadata File

Snapshot: inode copy-on-write



- First update to an inode copies the inode block into the shadow inode file
- Data/indirect block addresses in the snapshot inode are replaced with a special “ditto” disk address
- Ditto is a logical back pointer: when a reading file in the snapshot, a ditto means: *This data block has not changed since the snapshot was created; read the data from the original file instead* (“ditto resolution”)
- A hole in the shadow inode file is like an inode full of dittos

Snapshot: data copy-on-write



- First update to a data block copies the data block into the snapshot file
- Ditto replaced with the address of a new data block
- New data block must be flushed to disk before old data block can be updated

Snapshot Performance Improvements

- Inode Copy-On-Write (COW) performance improvement
  - Details on COW-related performance issue hit by some customers using snapshots, resulting in **llMsgCopyInodeBlock** and **llMsgMultiBlockUpdate** waiters, e.g.:

```
node_name_changed: Waiting 8.8900 sec since 2023-06-10_21:09:37, monitored, thread
1690813 MsgHandler@llMsgCopyInodeBlock: on ThCon d 0x7F6E3C051318
(MsgRecordCondvar), reason 'RPC wait' for llMsgMultiBlockUpdate
```

- A Snapshot is used to capture a point-in-time copy of file system or fileset. Whenever a file is changed, created or deleted, its inode is copied to the latest snapshot to record that point-in-time state of the file before updating it.
- Inode COW is based on the inode block granularity: a one-time copy needs to copy all inodes in that block
  - Only one thread performs the copy operation for an inode block.
- A one-time inode COW is needed for an inode block if no changes have been made to any of the inodes since the last snapshot was taken.
- With lots of file creations, deletions, and changes to files, frequent inode COW requests are triggered.

Snapshot Performance Improvements

❑Inode Copy-On-Write (COW) performance improvement

➢Background for the performance issue hit by some customers (cont'd)

- With many client nodes (e.g., 100s or 1000s of nodes in the cluster), there can be many concurrent inode COW requests coming from different nodes.
- In addition, for each inode block COW, the updates from COW requests need to be broadcast to all the client nodes who have mounted the file system.
- Broadcasting the inode COW updates to all client nodes takes longer if some clients have network issues.
- File operations that trigger inode COW will get their replies only after the broadcasts described above complete

Snapshot Performance Improvements

➢Solution to improve the inode COW performance

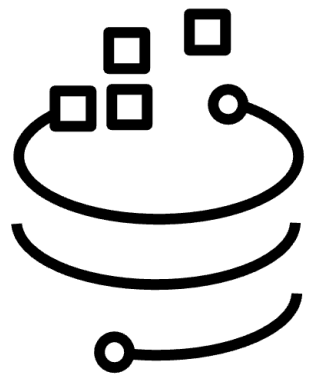
- Change the broadcasting for the inode COW updates from sync mode to the async mode
- A reply can be sent to the node initiating the inode COW request without waiting for the broadcast completion
- The file operations can proceed forward more quickly.
- With this approach, regardless of the time taken for the broadcast for the inode COW (e.g., either because of a scaling issue with how many client node must be broadcast to, or network issues with some client nodes), it won't slow down the file operations that initiated the inode COW requests. In summary:
  - If there are no network problems on the FS manager node, changing the broadcast to be asynchronous can provide a big performance improvement for the case in which a subset of clients might otherwise delay the broadcast operations
  - For example, for the original case in which some clients drop packets for a synchronous broadcast, the entire broadcast operation will be limited by the performance of these slow clients, but, after the design change to switch to asynchronous mode, the broadcast flow is changed to no longer be limited by the slow performance of clients experiencing packet loss

Snapshot Performance Improvements

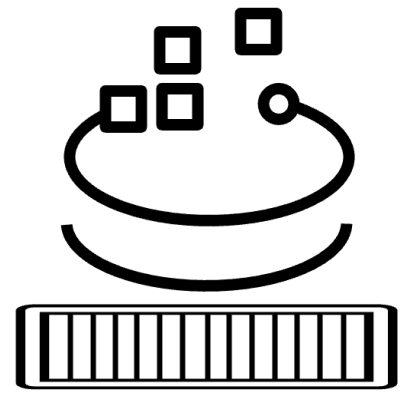
➢Solution to improve the inode COW performance (cont'd)

- For any client node *without* this improvement, the reply for inode COW request is still sent only *after* the broadcast above completes:
- The performance of async broadcasting may still impact the overall system performance because of the above
- Another improvement is also made to *batch* the async broadcast requests before broadcasting messages:
  - Fewer RPC messages are broadcasted
  - This change mitigates the impact of not having the new code fix applied to all client nodes, though it must be applied to the file system manager node to have an impact

Thank you for using



Storage Scale



Storage Scale System