



NVIDIA Magnum IO GPUDirect Storage

Installation and Troubleshooting Guide

Table of Contents

Chapter 1. Introduction.....	1
Chapter 2. Installing GPUDirect Storage.....	2
2.1. Before You Install GDS.....	2
2.2. Installing GDS.....	3
2.2.1. Configuring File System Settings for GDS.....	3
2.2.2. Verifying a Successful GDS Installation.....	4
2.3. Installed GDS Libraries and Tools.....	5
2.4. Uninstalling GPUDirect Storage.....	6
2.5. Environment Variables Used by GPUDirect Storage.....	6
2.6. JSON Config Parameters Used by GPUDirect Storage.....	7
2.7. GDS Configuration File Changes to Support Dynamic Routing.....	8
2.8. Determining Which Version of GDS is Installed.....	8
2.9. Experimental Repos for Network Install of GDS Packages for DGX Systems.....	9
Chapter 3. API Errors.....	10
3.1. CU_FILE_DRIVER_NOT_INITIALIZED.....	10
3.2. CU_FILE_DEVICE_NOT_SUPPORTED.....	10
3.3. CU_FILE_IO_NOT_SUPPORTED.....	10
3.4. CU_FILE_CUDA_MEMORY_TYPE_INVALID.....	11
Chapter 4. Basic Troubleshooting.....	12
4.1. Log Files for the GDS Library.....	12
4.2. Enabling a Different cufile.log File for Each Application.....	12
4.3. Enabling Tracing GDS Library API Calls.....	13
4.4. cuFileHandleRegister Error.....	13
4.5. Troubleshooting Applications that Return cuFile Errors.....	14
4.6. cuFile-* Errors with No Activity in GPUDirect Storage Statistics.....	14
4.7. CUDA Runtime and Driver Mismatch with Error Code 35.....	14
4.8. CUDA API Errors when Running the cuFile-* APIs.....	15
4.9. Finding GDS Driver Statistics.....	15
4.10. Tracking IO Activity that Goes Through the GDS Driver.....	15
4.11. Read/Write Bandwidth and Latency Numbers in GDS Stats.....	15
4.12. Tracking Registration and Deregistration of GPU Buffers.....	16
4.13. Enabling RDMA-specific Logging for Userspace File Systems.....	16
4.14. CUDA_ERROR_SYSTEM_NOT_READY After Installation.....	16
4.15. Adding udev Rules for RAID Volumes.....	17
4.16. When You Observe "Incomplete write" on NVME Drives.....	17

4.17. CUFILe async I/O is failing.....	18
Chapter 5. Advanced Troubleshooting.....	19
5.1. Resolving Hung cuFile* APIs with No Response.....	19
5.2. Sending Relevant Data to Customer Support.....	19
5.3. Resolving an IO Failure with EIO and Stack Trace Warning.....	21
5.4. Controlling GPU BAR Memory Usage.....	21
5.5. Determining the Amount of Cache to Set Aside.....	22
5.6. Monitoring BAR Memory Usage.....	22
5.7. Resolving an ENOMEM Error Code.....	22
5.8. GDS and Compatibility Mode.....	23
5.9. Enabling Compatibility Mode.....	23
5.10. Tracking the IO After Enabling Compatibility Mode.....	23
5.11. Bypassing GPUDirect Storage.....	24
5.12. GDS Does Not Work for a Mount.....	24
5.13. Simultaneously Running the GPUDirect Storage IO and POSIX IO on the Same File.....	25
5.14. Running Data Verification Tests Using GPUDirect Storage.....	25
Chapter 6. Troubleshooting Performance.....	26
6.1. Running Performance Benchmarks with GDS.....	26
6.2. Tracking Whether GPUDirect Storage is Using an Internal Cache.....	27
6.3. Tracking when IO Crosses the PCIe Root Complex and Impacts Performance.....	27
6.4. Using GPUDirect Statistics to Monitor CPU Activity.....	27
6.5. Monitoring Performance and Tracing with cuFile-* APIs.....	28
6.6. Example: Using Linux Tracing Tools.....	28
6.7. Tracing the cuFile-* APIs.....	29
6.8. Improving Performance using Dynamic Routing.....	30
Chapter 7. Troubleshooting IO Activity.....	32
7.1. Managing Coherency of Data in the Page Cache and on Disk.....	32
Chapter 8. EXAScaler Filesystem LNet Troubleshooting.....	33
8.1. Determining the EXAScaler Filesystem Client Module Version.....	33
8.2. Checking the LNet Network Setup on a Client.....	34
8.3. Checking the Health of the Peers.....	34
8.4. Checking for Multi-Rail Support.....	34
8.5. Checking GDS Peer Affinity.....	35
8.6. Checking for LNet-Level Errors.....	37
8.7. Resolving LNet NIDs Health Degradation from Timeouts.....	41
8.8. Configuring LNet Networks with Multiple OSTs for Optimal Peer Selection.....	41

Chapter 9. Understanding EXAScaler Filesystem Performance.....	45
9.1. osc Tuning Performance Parameters.....	45
9.2. Miscellaneous Commands for osc, mdc, and stripesize.....	46
9.3. Getting the Number of Configured Object-Based Disks.....	47
9.4. Getting Additional Statistics related to the EXAScaler Filesystem.....	48
9.5. Getting Metadata Statistics.....	48
9.6. Checking for an Existing Mount.....	48
9.7. Unmounting an EXAScaler Filesystem Cluster.....	48
9.8. Getting a Summary of EXAScaler Filesystem Statistics.....	49
9.9. Using GPUDirect Storage in Poll Mode.....	49
Chapter 10. Troubleshooting and FAQ for the WekaIO Filesystem.....	50
10.1. Downloading the WekaIO Client Package.....	50
10.2. Determining Whether the WekaIO Version is Ready for GDS.....	50
10.3. Mounting a WekaIO File System Cluster.....	51
10.4. Resolving a Failing Mount.....	51
10.5. Resolving 100% Usage for WekaIO for Two Cores.....	52
10.6. Checking for an Existing Mount in the Weka File System.....	52
10.7. Checking for a Summary of the WekaIO Filesystem Status.....	52
10.8. Displaying the Summary of the WekaIO Filesystem Statistics.....	53
10.9. Why WekaIO Writes Go Through POSIX.....	54
10.10. Checking for nvidia-fs.ko Support for Memory Peer Direct.....	54
10.11. Checking Memory Peer Direct Stats.....	55
10.12. Checking for Relevant nvidia-fs Statistics for the WekaIO Filesystem.....	55
10.13. Conducting a Basic WekaIO Filesystem Test.....	56
10.14. Unmounting a WekaIO File System Cluster.....	56
10.15. Verify the Installed Libraries for the WekaIO Filesystem.....	57
10.16. GDS Configuration File Changes to Support the WekaIO Filesystem.....	57
10.17. Check for Relevant User-Space Statistics for the WekaIO Filesystem.....	58
10.18. Check for WekaFS Support.....	58
Chapter 11. Enabling IBM Spectrum Scale Support with GDS.....	59
11.1. IBM Spectrum Scale Limitations with GDS.....	59
11.2. Checking nvidia-fs.ko Support for Mellanox PeerDirect.....	59
11.3. Verifying Installed Libraries for IBM Spectrum Scale.....	60
11.4. Checking PeerDirect Stats.....	61
11.5. Checking for Relevant nvidia-fs Stats with IBM Spectrum Scale.....	61
11.6. GDS User Space Stats for IBM Spectrum Scale for Each Process.....	62
11.7. GDS Configuration to Support IBM Spectrum Scale.....	63
11.8. Scenarios for Falling Back to Compatibility Mode.....	64

11.9. GDS Limitations with IBM Spectrum Scale.....	64
Chapter 12. NetApp E-series BeeGFS with GDS Solution Deployment.....	65
12.1. Netapp BeeGFS/GPUDirect Storage and Package Requirements.....	65
12.2. BeeGFS Client Configuration for GDS.....	65
12.3. GPU/HCA Topology on the Client - DGX-A100 and OSS servers Client Server.....	66
12.4. Verify the Setup.....	66
12.4.1. List the Management Node.....	66
12.4.2. List the Metadata Nodes.....	66
12.4.3. List the Storage Nodes.....	67
12.4.4. List the Client Nodes.....	67
12.4.5. Display Client Connections.....	67
12.4.6. Verify Connectivity to the Different Services.....	68
12.4.7. List Storage Pools.....	68
12.4.8. Display the Free Space and inodes on the Storage and Metadata Targets.....	68
12.5. Testing.....	69
12.5.1. Verifying Integration is Working.....	69
12.5.2. Conducting a Basic NetApp BeeGFS Filesystem Test.....	70
Chapter 13. Setting Up and Troubleshooting VAST Data (NFSoverRDMA+MultiPath)..	71
13.1. Installing MLNX_OFED and VAST NFSoverRDMA+Multipath Packages.....	71
13.1.1. Client Software Requirements.....	71
13.1.2. Install the VAST Multipath Package.....	72
13.2. Set Up the Networking.....	72
13.2.1. VAST Network Configuration.....	73
13.2.2. Client Network Configuration.....	73
13.2.3. Verify Network Connectivity.....	75
13.3. Mount VAST NFS.....	76
13.4. Debugging and Monitoring VAST Data.....	76
Chapter 14. Troubleshooting and FAQ for NVMe and NVMeOF Support.....	78
14.1. MLNX_OFED Requirements and Installation.....	78
14.2. Determining Whether the NVMe device is Supported for GDS.....	78
14.3. RAID Support in GDS.....	79
14.4. Mounting a Local Filesystem for GDS.....	79
14.5. Check for an Existing EXT4 Mount.....	79
14.6. Check for IO Statistics with Block Device Mount.....	79
14.7. RAID Group Configuration for GPU Affinity.....	80
14.8. Conduct a Basic EXT4 Filesystem Test.....	80
14.9. Unmount a EXT4 Filesystem.....	80
14.10. Udev Device Naming for a Block Device.....	80

14.11. BATCH I/O Performance.....	81
Chapter 15. Displaying GDS NVIDIA FS Driver Statistics.....	82
15.1. nvidia-fs Statistics.....	83
15.2. Analyze Statistics for each GPU.....	87
15.3. Resetting the nvidia-fs Statistics.....	87
15.4. Checking Peer Affinity Stats for a Kernel Filesystem and Storage Drivers.....	87
15.5. Checking the Peer Affinity Usage for a Kernel File System and Storage Drivers..	89
15.6. Display the GPU-to-Peer Distance Table.....	91
15.7. The GDSIO Tool.....	92
15.8. Tabulated Fields.....	94
15.9. The GDSCHECK Tool.....	96
15.10. NFS Support with GPUDirect Storage.....	97
15.10.1. Install Linux NFS server with RDMA Support on MLNX_OFED 5.3 or Later....	97
15.10.2. Install GPUDirect Storage Support for the NFS Client.....	98
15.11. NFS GPUDirect Storage Statistics and Debugging.....	98
15.12. GPUDirect Storage IO Behavior.....	98
15.12.1. Read/Write Atomicity Consistency with GPUDirect Storage Direct IO.....	99
15.12.2. Write with File a Opened in O_APPEND Mode (cuFileWrite).....	99
15.12.3. GPU to NIC Peer Affinity.....	99
15.12.4. Compatible Mode with Unregistered Buffers.....	99
15.12.5. Unaligned writes with Non-Registered Buffers.....	99
15.12.6. Process Hang with NFS.....	99
15.12.7. Tools Support Limitations for CUDA 9 and Earlier.....	100
15.13. GDS Statistics for Dynamic Routing.....	100
15.13.1. Peer Affinity Dynamic Routing.....	101
15.13.2. cuFile Log Related to Dynamic Routing.....	103
Chapter 16. GDS Library Tracing.....	105
16.1. Example: Display Tracepoints.....	105
16.1.1. Example: Tracepoint Arguments.....	105
16.2. Example: Track the IO Activity of a Process that Issues cuFileRead/ cuFileWrite..	110
16.3. Example: Display the IO Pattern of all the IOs that Go Through GDS.....	111
16.4. Understand the IO Pattern of a Process.....	111
16.5. IO Pattern of a Process with the File Descriptor on Different GPUs.....	112
16.6. Determine the IOPS and Bandwidth for a Process in a GPU.....	113
16.7. Display the Frequency of Reads by Processes that Issue cuFileRead.....	113
16.8. Display the Frequency of Reads when cuFileRead Takes More than 0.1 ms.....	114
16.9. Displaying the Latency of cuFileRead for Each Process.....	115
16.10. Example: Tracking the Processes that Issue cuFileBufRegister.....	116

16.11. Example: Tracking Whether the Process is Constant when Invoking cuFileBufRegister.....	116
16.12. Example: Monitoring IOs that are Going Through the Bounce Buffer.....	117
16.13. Example: Tracing cuFileRead and cuFileWrite Failures, Print, Error Codes, and Time of Failure.....	117
16.14. Example: User-Space Statistics for Each GDS Process.....	118
16.15. Example: Viewing GDS User-Level Statistics for a Process.....	119
16.16. Example: Displaying Sample User-Level Statistics for each GDS Process.....	119
Chapter 17. User-Space Counters in GPUDirect Storage.....	121
17.1. Distribution of IO Usage in Each GPU.....	123
17.2. User-space Statistics for Dynamic Routing.....	124
Chapter 18. User-Space RDMA Counters in GPUDirect Storage.....	125
18.1. cuFile RDMA IO Counters (PER_GPU RDMA STATS).....	126
18.2. cuFile RDMA Memory Registration Counters (RDMA MRSTATS).....	126
Chapter 19. Cheat Sheet for Diagnosing Problems.....	128

Chapter 1. Introduction

This guide describes how to debug and isolate the NVIDIA® Magnum IO GPUDirect® Storage (GDS) related performance and functional problems and is intended for systems administrators and developers.

GDS enables a direct data path for direct memory access (DMA) transfers between GPU memory and storage, which avoids a bounce buffer through the CPU. This direct path increases system bandwidth and decreases the latency and utilization load on the CPU.

Creating this direct path involves distributed filesystems such as NFSoRDMA, DDN EXAScaler® parallel filesystem solutions (based on the Lustre filesystem) and WekaFS, so the GDS environment is composed of multiple software and hardware components. This guide addresses questions related to the GDS installation and helps you triage functionality and performance issues. For non-GDS issues, contact the respective OEM or filesystems vendor to understand and debug the issue.

Chapter 2. Installing GPUDirect Storage

This section includes GDS installation, uninstallation, configuration information, and using experimental repos.



Note: For NVAIE and vGPU environments, please follow steps from their respective documents.

2.1. Before You Install GDS

To install GDS on a non-DGX platform, complete the following steps:

1. Run the following command to check the current status of IOMMU.

```
$ dmesg | grep -i iommu
```

On x86_64 based platforms, if IOMMU is enabled, complete step 2 to disable it, otherwise continue to step 3.

2. Disable IOMMU.



Note: In our experience, `iommu=off` works the best in terms of functionality and performance. On certain platforms such as DGX A100 and DGX-2, `iommu=pt` is supported. `iommu=on` is not guaranteed to work functionally or in a performant way.

- a). Run the following command:

```
$ sudo vi /etc/default/grub
```

- b). Add one of the following options to the `GRUB_CMDLINE_LINUX_DEFAULT` option.

- ▶ If you have an AMD CPU, add `amd_iommu=off`.
- ▶ If you have an Intel CPU, add `intel_iommu=off`.

If there are already other options, enter a space to separate the options, for example, `GRUB_CMDLINE_LINUX_DEFAULT="console=tty0 amd_iommu=off`

- c). Run the following commands:

```
$ sudo update-grub  
$ sudo reboot
```

d). After the system reboots, to verify that the change took effect, run the following command:

```
$ cat /proc/cmdline
```

It should show the options which have been added to the grub file.

3. Before you run the instructions, please make sure to read the Notes section. Use the instructions in [MLNX_OFED Requirements and Installation](#) to install MLNX_OFED.

Notes:

- ▶ This step is required ONLY IF you need to enable support for NVMe, NVMeF, NFSoRDMA
- ▶ This step is not required for DGX OS 6.x or later.

2.2. Installing GDS

GDS installation is supported in two ways:

- ▶ using package managers such as Debian and RPMs

For installation on DGX platforms, refer to:

- ▶ [DGX-OS](#)
- ▶ [RHEL 8](#)

For installation on non-DGX platforms, refer to [here](#).



Note:

For CUDA 11.5.1 and later, if you plan to use Weka FS or IBM SpectrumScale then you must run:

```
modprobe nvidia_peermem
```

This will load the module that supports PeerDirect capabilities. It is necessary to run this command after reboot of the system.

In order to load the module automatically after every reboot, run the following command:

```
echo "nvidia-peermem" | sudo tee /etc/modules-load.d/nvidia-peermem.conf
```

Throughout this document, in `cuda-<x>.<y>`, `x` refers to the CUDA major version and `y` refers to the minor version.

2.2.1. Configuring File System Settings for GDS

Before proceeding, please refer to the File System specific section in this document for necessary configurations needed to support GDS:

- ▶ Lustre-Lnet: [Configuring LNet Networks with Multiple OSTs for Optimal Peer Selection](#)

- ▶ WekaIO: [GDS Configuration File Changes to Support the WekaIO Filesystem](#)
- ▶ IBM Spectrum Scale: [GDS Configuration to Support IBM Spectrum Scale](#)
- ▶ BeeGFS: [BeeGFS Client Configuration for GDS](#)
- ▶ VAST: [Set Up the Networking](#)
- ▶ NVMe: [RAID Group Configuration for GPU Affinity](#)



Note: This step can be skipped for local file systems such as Ext4/XFS.

2.2.2. Verifying a Successful GDS Installation

To verify that GDS installation was successful, run `gdscheck`:

```
$ /usr/local/cuda-<x>.<y>/gds/tools/gdscheck.py -p
```



Note: The `gdscheck` command expects `python3` to be present on the system. If it fails because of `python3` not being available, then you can invoke the command with the explicit path to where `python` (i.e. `python2`) is installed. For example:

```
$ /usr/bin/python /usr/local/cuda-<x>.<y>/gds/tools/gdscheck.py -p
```

The output of this command shows whether a supported filesystem or device installed on the system supports GDS. The output also shows whether PCIe ACS is enabled on any of the PCI switches.



Note: For best GDS performance, disable PCIe ACS.

Sample output:

```
GDS release version: 1.0.0.80
nvidia_fs version: 2.7 libcufile version: 2.4
=====
ENVIRONMENT:
=====
=====
DRIVER CONFIGURATION:
=====
NVMe                : Unsupported
NVMeOF              : Unsupported
SCSI                 : Unsupported
ScaleFlux CSD       : Unsupported
NVMesh              : Unsupported
DDN EXAScaler       : Unsupported
IBM Spectrum Scale  : Unsupported
NFS                  : Supported
WekaFS               : Unsupported
Userspace RDMA      : Unsupported
--Mellanox PeerDirect : Enabled
--rdma library       : Not Loaded (libcufile_rdma.so)
--rdma devices       : Not configured
--rdma_device_status : Up: 0 Down: 0
=====
CUFILE CONFIGURATION:
=====
properties.use_compat_mode : false
properties.gds_rdma_write_support : true
properties.use_poll_mode   : false
```

```

properties.poll_mode_max_size_kb : 4
properties.max_batch_io_timeout_msecs : 5
properties.max_direct_io_size_kb : 16384
properties.max_device_cache_size_kb : 131072
properties.max_device_pinned_mem_size_kb : 33554432
properties.posix_pool_slab_size_kb : 4 1024 16384
properties.posix_pool_slab_count : 128 64 32
properties.rdma_peer_affinity_policy : RoundRobin
properties.rdma_dynamic_routing : 0
fs.generic.posix_unaligned_writes : false
fs.lustre.posix_gds_min_kb: 0
fs.weka.rdma_write_support: false
profile.nvtx : false
profile.cufile_stats : 0
miscellaneous.api_check_aggressive : false
=====
GPU INFO:
=====
GPU index 0 A100-PCIE-40GB bar:1 bar size (MiB):65536 supports GDS
=====
PLATFORM INFO:
=====
IOMMU: disabled
Platform verification succeeded

```

**Note:**

There are READMEs provided in `/usr/local/cuda-<x>.<y>/gds/tools` and `/usr/local/cuda-<x>.<y>/gds/samples` to show usage.

2.3. Installed GDS Libraries and Tools

GPUDirect Storage userspace libraries are located in the `/usr/local/cuda-<X>.<Y>/targets/x86_64-linux/lib/` directory.



Note: GPUDirect Storage packages are installed at `/usr/local/cuda-X.Y/gds`, where X is the major version of the CUDA toolkit, and Y is the minor version.

```

$ ls -l /usr/local/cuda-X.Y/targets/x86_64-linux/lib/*cufile*
cufile.h
libcufile.so
libcufile.so.0
libcufile.so.1.0.0
libcufile_rdma.so
libcufile_rdma.so.0
libcufile_rdma.so.1.0.0

```

GPUDirect Storage tools and samples are located in the `/usr/local/cuda-X.Y/gds` directory.

```

$ ls -lh /usr/local/cuda-X.Y/gds/
total 20K
-rw-r--r-- 1 root root 8.4K Mar 15 13:01 README
drwxr-xr-x 2 root root 4.0K Mar 19 12:29 samples
drwxr-xr-x 2 root root 4.0K Mar 19 10:28 tools

```

For this release, GPUDirect Storage is providing an additional `libcufile-dev` package (cuFile library developers package). This is primarily intended for the developer's environment. Essentially the `libcufile-dev` package contains a static version of cuFile

library (`libcufile_static.a`, `libcufile_rdma_static.a`) and `cufile.h` header file which may be required by the applications that use cuFile library APIs.

2.4. Uninstalling GPUDirect Storage

To uninstall GDS from Ubuntu and DGX OS:

```
$ sudo apt-get remove --purge "*libcufile*" "*gds-tools*" "*nvidia-fs*"
```

To uninstall from RHEL:

```
$ sudo dnf remove "nvidia-gds*"
```

2.5. Environment Variables Used by GPUDirect Storage

GDS uses the following environment variables.

Table 1. GDS Environment Variables

CUFILE_ENV Variable	Description
<code>CUFILE_CQ_DEPTH</code>	Completion queue depth for the DC target.
<code>CUFILE_ENV_EXPERIMENTAL_FS=1</code>	Controls whether cufile checks for supporting filesystems. When set to 1, allows testing with new filesystems that are not yet officially enabled with cuFile.
<code>CUFILE_ENV_PATH_JSON=/home/user/cufile.json</code>	Controls the path where the cuFile library reads the configuration variables from. This can be used for container environments and applications that require different configuration settings from system default configuration at <code>/etc/cufile.json</code> .
<code>CUFILE_ETH_SL</code>	Sets QOS level on RoCEv2 device QP for userspace RDMA targets (WekaFS and GPFS).
<code>CUFILE_IB_SL=[0-15]</code>	Sets QOS level on IB device QP for userspace RDMA targets (WekaFS and GPFS).
<code>CUFILE_LOGFILE_PATH=/etc/log/cufile_\$\$log</code>	Controls the path for cuFile log information. Specifies the default log path, which is the current working directory of the application. Useful for containers or logging.

CUFILE_ENV Variable	Description
CUFILE_LOGGING_LEVEL=TRACE	Controls the tracing level and can override the trace level for a specific application without requiring a new configuration file.
CUFILE_MIN_RNR_TIMER	Minimum RNR value for QP after which the QP will error out with RNR timeout if no Work Request is posted on the remote end. Default value is 16 (2.56ms).
CUFILE_NVTX=true	Enables NVTX tracing for use with Nsight systems.
CUFILE_RDMA_DC_KEY="OXABABCDEF"	Controls the DC_KEY for userspace RDMA DC targets for WekaFS and GPFS.
CUFILE_RDMA_HOP_LIMIT	Maximum number of hops before the packet is discarded on the network. Prevents indefinite looping of the packet. Default is 64.
CUFILE_RDMA_PKEY_INDEX	Partition key index.
CUFILE_RDMA_SR_MAX_WR	Maximum number of Work requests supported by the Shared Request Queue.
CUFILE_RDMA_SR_MAX_SGE	Maximum number of Scatter Gather Entries supported per Work Request.
CUFILE_SKIP_TOPOLOGY_DETECTION	Setting this environment variable to true will skip topology detection in compat mode. This will reduce the high startup latency seen in compat mode on systems with multiple PCI devices.
CUFILE_FORCE_COMPAT_MODE	Overrides cufile.json settings and forces I/O to go through compatible mode instead of GDS mode.
CUFILE_ALLOW_COMPAT_MODE	This does exactly what the <code>allow_compat_mode</code> tag in <code>cufile.json</code> file does.

2.6. JSON Config Parameters Used by GPUDirect Storage

Refer to [GPUDirect Storage Parameters](#) for details about the JSON Config parameters used by GDS

Consider `compat_mode` for systems or mounts that are not yet set up with GDS support. To learn more about Compatibility Mode, refer to [cuFile Compatibility Mode](#).

2.7. GDS Configuration File Changes to Support Dynamic Routing

For dynamic routing to support multiple file systems and mount points, configure the global per file system `rdma_dev_addr_list` property for a single mount or the `rdma_dev_addr_list` property for a per file system mount table.

```
"fs": {
  "lustre": {
    // if using a single lustre mount, provide the ip addresses
    // here (use : sudo lnetctl net show)
    //"rdma_dev_addr_list" : []

    // if using multiple lustre mounts, provide ip addresses
    // used by respective mount here
    //"mount_table" : {
    //  "/lustre/ai200_01/client" : {
    //    "rdma_dev_addr_list" : ["172.172.1.40",
    //                          "172.172.1.42"]
    //  },
    //  "/lustre/ai200_02/client" : {
    //    "rdma_dev_addr_list" : ["172.172.2.40",
    //                          "172.172.2.42"]
    //  }
    //}
  },

  "nfs": {
    //"rdma_dev_addr_list" : []

    //"mount_table" : {
    //  "/mnt/nfsrdma_01/" : {
    //    "rdma_dev_addr_list" : []
    //  },
    //  "/mnt/nfsrdma_02/" : {
    //    "rdma_dev_addr_list" : []
    //  }
    //}
  },
},
```

2.8. Determining Which Version of GDS is Installed

To determine which version of GDS you have, run the following command:

```
$ gdscheck.py -v
```

Example output:

```
GDS release version: 1.0.0.78
nvidia_fs version: 2.7 libcufile version: 2.4
```


2.9. Experimental Repos for Network Install of GDS Packages for DGX Systems

GDS 1.0.0 and MLNX_OFED packages can be installed by enabling the preview repository on supported DGX platforms using the following steps.

For Ubuntu 18.04/20.04 distributions:

GDS 1.0.0, NVSM and MLNX_OFED packages can be installed via network using the preview network repository.

For Ubuntu 20.04 distributions:

```
$ sudo apt-key adv --fetch-keys https://repo.download.nvidia.com/baseos/GPG-KEY-dgx-cosmos-support  
  
$ sudo add-apt-repository "deb https://repo.download.nvidia.com/baseos/ubuntu/focal/x86_64/ focal-updates preview"  
  
$ sudo apt update
```

Chapter 3. API Errors

This section provides information about the common API errors you might get when using GDS.

3.1. CU_FILE_DRIVER_NOT_INITIALIZED

If the `cuFileDriverOpen` API is not called, errors encountered in the implicit call to driver initialization are reported as cuFile errors encountered when calling `cuFileBufRegister` or `cuFileHandleRegister`.

3.2. CU_FILE_DEVICE_NOT_SUPPORTED

GDS is supported only on NVIDIA graphics processing units (GPU) Tesla[®] or Quadro[®] models that support compute mode, and a compute major capability greater than or equal to 6.



Note: This includes V100 and T4 cards.

3.3. CU_FILE_IO_NOT_SUPPORTED

If the file descriptor is from a local filesystem, or a mount that is not GDS ready, the API returns the `CU_FILE_IO_NOT_SUPPORTED` error.

See [Before You Install GDS](#) for a list of the supported filesystems.

Common reasons for this error include:

- ▶ The file descriptor belongs to an unsupported filesystem.
- ▶ The specified `fd` is not a regular UNIX file.
- ▶ Any combination of encryption, and compression, compliance settings on the `fd` are set.

For example, `FS_COMPR_FL | FS_ENCRYPT_FL | FS_APPEND_FL | FS_IMMUTABLE_FL`.



Note: These settings are allowed when `compat_mode` is set to `true`.

- ▶ Any combination of unsupported file modes are specified in the open call for the `fd`. For example,

`O_APPEND | O_NOCTTY | O_NONBLOCK | O_DIRECTORY | O_NOFOLLOW | O_TMPFILE`

3.4. CU_FILE_CUDA_MEMORY_TYPE_INVALID

Physical memory for `cudaMallocManaged` memory is allocated dynamically at the first use. Currently, it does not provide a mechanism to expose physical memory or Base Address Register (BAR) memory to pin for use in GDS. However, GDS indirectly supports `cudaMallocManaged` memory when the memory is used as an unregistered buffer with `cuFileWrite` and `cuFileRead`.

Chapter 4. Basic Troubleshooting

4.1. Log Files for the GDS Library

A `cufile.log` file is created in the same location where the application binaries are located. Currently the maximum log file size is 32MB. If the log file size increases to greater than 32MB, the log file is truncated and logging is resumed on the same file.

4.2. Enabling a Different `cufile.log` File for Each Application

You can enable a different `cufile.log` file for each application.

There are several relevant cases:

- ▶ If the `logging:dir` property in the default `/etc/cufile.json` file is not set, by default, the `cufile.log` file is generated in the current working directory of the application.
- ▶ If the `logging:dir` property is set in the default `/etc/cufile.json` file, the log file is created in the specified directory path.



Note: This is usually not recommended for scenarios where multiple applications use the `libcufile.so` library.

For example:

```
"logging": {
  // log directory, if not enabled
  // will create log file under current working
  // directory
  "dir": "/opt/gdslogs/",
}
```

The `cufile.log` will be created as a `/opt/gdslogs/cufile.log` file.

If the application needs to enable a different `cufile.log` for different applications, the application can override the default JSON path by doing the following steps:

1. Export `CUFILE_ENV_PATH_JSON="/opt/myapp/cufile.json"`.
2. Edit the `/opt/myapp/cufile.json` file.

```
"logging": {
  // log directory, if not enabled
  // will create log file under current working
  // directory
  "dir": "/opt/myapp",
}
```

3. Run the application.
4. To check for logs, run:

```
$ ls -l /opt/myapp/cufile.log
```

4.3. Enabling Tracing GDS Library API Calls

There are different logging levels, which can be enabled in the `/etc/cufile.json` file. By default, logging level is set to `ERROR`. Logging will have performance impact as we increase the verbosity levels like `INFO`, `DEBUG`, and `TRACE`, and should be enabled only to debug field issues.

Configure tracing and run the following:

```
"logging": {
  // log directory, if not enabled
  // will create log file under local directory
  //"dir": "/home/<xxxx>",

  // ERROR|WARN|INFO|DEBUG|TRACE (in decreasing order of priority)
  "level": "ERROR"
},
```

4.4. cuFileHandleRegister Error

If you see the `cuFileHandleRegister` error on the `cufile.log` file when an IO is issued:

```
"cuFileHandleRegister error: GPUDirect Storage not supported on current file."
```

Here are some reasons why this error might occur:

- ▶ The filesystem is not supported by GDS.
 - See [CU_FILE_DEVICE_NOT_SUPPORTED](#) for more information.
- ▶ `DIRECT_IO` functionality is not supported for the mount on which the file resides.

For more information, enable tracing in the `/etc/cufile.json` file.

4.5. Troubleshooting Applications that Return cuFile Errors

To troubleshoot cuFile errors:

1. See the `cufile.h` file for more information about errors that are returned by the API.
2. If the IO was submitted to the GDS driver, check whether there are any errors in GDS stats.

If the IO fails, the error stats should provide information about the type of error.

See [Finding the GDS Driver Statistics](#) for more information.

3. Enable GDS library tracing and monitor the `cufile.log` file.
4. Enable GDS Driver debugging:

```
$ echo 1 >/sys/module/nvidia_fs/parameters/dbg_enabled
```

After the driver debug logs are enabled, you might get more information about the error.

4.6. cuFile-* Errors with No Activity in GPUDirect Storage Statistics

If there are cuFile errors in the GDS statistics, this means that the API failed in the GDS library. You can enable tracing by setting the appropriate logging level in the `/etc/cufile.json` file to get more information about the failure in `cufile.log`.

4.7. CUDA Runtime and Driver Mismatch with Error Code 35

Error code 35 from the CUDA documentation points to `cudaErrorInsufficientDriver`, which indicates that the installed NVIDIA CUDA driver is older than the CUDA runtime library. This is not a supported configuration. For the application to run, you must update the NVIDIA display driver.



Note: cuFile tools depend on CUDA runtime 10.1 and later. You must ensure that the installed CUDA runtime is compatible with the installed CUDA driver and is at the recommended version.

4.8. CUDA API Errors when Running the cuFile-* APIs

The GDS library uses the CUDA driver APIs.

If you observe CUDA API errors, you will observe an error code. Refer to the error codes in the [CUDA Libraries documentation](#) for more information.

4.9. Finding GDS Driver Statistics

To find the GDS Driver Statistics, run the following command:

```
$ cat /proc/driver/nvidia-fs/stats
```

GDS Driver kernel statistics for `READ/WRITE` are available for all file systems except for Weka. For Weka file system statistics, refer to [Troubleshooting and FAQ for the WekaIO Filesystem](#) for more information about `READ/WRITE`.

4.10. Tracking IO Activity that Goes Through the GDS Driver

In GDS Driver statistics, the `ops` row shows the active IO operation. The `Read` and `Write` fields show the current active operation in flight. This information should provide an idea of how many total IOs are in flight across all applications in the kernel. If there is a bottleneck in the userspace, the number of active IOs will be less than the number of threads that are submitting the IO. Additionally, to get more details about the `Read` and `Write` bandwidth numbers, look out for counters in the `Read/Write` rows.

4.11. Read/Write Bandwidth and Latency Numbers in GDS Stats

Measured latencies begin when the IO is submitted and end when the IO completion is received by the GDS kernel driver. Userspace latencies are not reported. This should provide an idea whether the user space is bottlenecked or whether the IO is bottlenecked on the backend disks/fabric.



Note: The WekaIO filesystem reads do not go through the `nvidia-fs` driver, so `Read/Write` bandwidth stats are not available for WekaIO filesystem by using this interface.

Refer to the [Troubleshooting and FAQ for the WekaIO Filesystem](#) for more information.

4.12. Tracking Registration and Deregistration of GPU Buffers

In GDS Driver stats, look for the active field in BAR1-map stats row.

The pinning and unpinning of GPU memory through `cuFileBufRegister` and `cuFileBufDeregister` is an expensive operation. If you notice a large number of registrations (`n`) and deregistration (`free`) in the `nvidia-fs` stats, it can hurt performance. Refer to the [GPUDirect Storage Best Practices Guide](#) for more information about using the `cuFileBufRegister` API.

4.13. Enabling RDMA-specific Logging for Userspace File Systems

In order to troubleshoot RDMA related issues for userspace file systems, ensure that the `CUFILE_LOGGING_LEVEL` environment variable is set to `INFO`, `DEBUG`, or `TRACE` prior to running the application. However, for this to work, `cufile.json` logging level also should be set to `TRACE/DEBUG/INFO` level.

For example:

```
$ export CUFILE_LOGGING_LEVEL=INFO
This is an example to set log level to INFO via the environment variable.

$ cat /etc/cufile.json
....
"logging": {
  // log directory, if not enabled will create log file
  // under current working directory
  // "dir": "/home/<xxxx>",
  // ERROR|WARN|INFO|DEBUG|TRACE (in decreasing order of priority)
  "level": "DEBUG"
},
....
This is an example on how to set log level to DEBUG via cufile.json.
```

4.14. CUDA_ERROR_SYSTEM_NOT_READY After Installation

On systems with NVSwitch, if you notice the `CUDA_ERROR_SYSTEM_NOT_READY` error being reported, then make sure that you install the same version of Fabric Manager as the CUDA driver.

For example, if you use:

```
$ sudo apt install nvidia-driver-460-server -y
```

then use:

```
$ apt-get install nvidia-fabricmanager-460
```


Make sure to restart the Fabric Manager service using:

```
$ sudo service nvidia-fabricmanager start
```

4.15. Adding udev Rules for RAID Volumes

To add udev rules for RAID volumes:

As a sudo user, change the following line in `/lib/udev/rules.d/63-md-raid-arrays.rules`:

```
IMPORT{program}="/usr/sbin/mdadm --detail --export $devnode"
```

Reboot the node or restart the `mdadm`.

4.16. When You Observe "Incomplete write" on NVME Drives

During GDS mode writes, you may receive error messages similar to the following:

```
Tid: 0 incomplete Write, done = 0 issued = 1048576
```

GPUDirect storage in P2P mode does not support NVMe end to end data protection features. To support GDS in P2P mode, the NVMe must be formatted with Protection Information - Metadata Size is set to zero bytes.

Confirm that the drive has data-integrity mode enabled:

```
$ sudo nvme id-ns /dev/nvme0n1 -H
```

```
...
LBA Format 0 : Metadata Size: 0 bytes - Data Size: 512 bytes - Relative
Performance: 0x1 Better
LBA Format 1 : Metadata Size: 8 bytes - Data Size: 512 bytes - Relative
Performance: 0x3 Degraded (in use)
LBA Format 2 : Metadata Size: 0 bytes - Data Size: 4096 bytes - Relative
Performance: 0 Best
LBA Format 3 : Metadata Size: 8 bytes - Data Size: 4096 bytes - Relative
Performance: 0x2 Good
LBA Format 4 : Metadata Size: 64 bytes - Data Size: 4096 bytes - Relative
Performance: 0x3 Degraded
```

Note in the preceding example, the metadata size of the drive (`nvme0n1`) is set to non-zero.

You can set the LBA format to 0 or 2 to disable the protection feature on the drive:

```
$ sudo nvme format /dev/nvme0n1 -l 2
```

```
$ sudo nvme id-ns /dev/nvme0n1 -H
```

```
...
LBA Format 0 : Metadata Size: 0 bytes - Data Size: 512 bytes - Relative
Performance: 0x1 Better
LBA Format 1 : Metadata Size: 8 bytes - Data Size: 512 bytes - Relative
Performance: 0x3 Degraded
LBA Format 2 : Metadata Size: 0 bytes - Data Size: 4096 bytes - Relative
Performance: 0 Best (in use)
LBA Format 3 : Metadata Size: 8 bytes - Data Size: 4096 bytes - Relative
Performance: 0x2 Good
```

```
LBA Format 4 : Metadata Size: 64 bytes - Data Size: 4096 bytes - Relative  
Performance: 0x3 Degraded
```

4.17. CUFILE async I/O is failing

There could be many reasons for which stream based async I/O can fail. This will be logged in `cufiler.log`. One of the common reasons could be that the internal thread pool is not enabled. Refer to `cufiler.json` "execution" section on how to enable it.

Chapter 5. Advanced Troubleshooting

This section provides information about troubleshooting some advanced issues.

5.1. Resolving Hung cuFile* APIs with No Response

To resolve hung cuFile APIs:

1. Check whether there are any kernel panics/warnings in `dmesg`:

```
$ dmesg > warnings.txt. less warnings.txt
```

2. Check whether the application process is in the `'D'` (uninterruptible) state).

3. If the process is in the `'D'` state:

- a). Get the PID of the process by running the following command:

```
$ ps axf | grep 'D'
```

- b). As a root user, get the backtrace of the `'D'` state process:

```
$ su root  
$ cat /proc/<pid>/stack
```

4. Verify whether the threads are stuck in the kernel or in user space.

For more information, review the backtrace of the `'D'` state threads.

5. Check whether any threads are showing heavy CPU usage.

- a). The `htop` and `mpstat` tools should show CPU usage per core.

- b). Get the call graph of where the CPUs are being used.

The following code snippet should narrow down whether the threads are hung in user space or in the kernel:

```
$ perf top -g
```

5.2. Sending Relevant Data to Customer Support

This section describes how to resolve a kernel panic with stack traces using NVSM or the GDS Log Collection tool.

DGX OS:

For DGX BaseOS with the preview network repo enabled and NVSM installed:

```
$ sudo apt-get install nvsm
$ sudo nvsm dump health
```

For more details on running NVSM commands, refer to [NVIDIA System Management User Guide](#).

Non DGX:

The GDS Log Collection tool, `gds_log_collection.py`, may be run by GDS users to collect relevant debugging information from the system when issues with GDS IO are seen.

Some of the important information that this tool captures is highlighted below:

- ▶ dmesg Output and relevant kernel log files.
- ▶ System map files and vmlinux image
- ▶ modinfo output for relevant modules
- ▶ `/proc/cmdline` output
- ▶ IB devices info like `ibdev2net` and `ibstatus`
- ▶ OS distribution information
- ▶ Cpuinfo, meminfo
- ▶ `nvidia-fs` stats
- ▶ Per process information like `cufile.log`, `cufile.json`, `gds_stats`, `stack pointers`
- ▶ Any user specified files

To use the log collection tool:

```
$ sudo /usr/local/cuda/gds//tools/gdstools/gds_log_collection.py -h
```

This tool is used to collect logs from the system that are relevant for debugging.

It collects logs such as OS and kernel info, `nvidia-fs` stats, `dmesg` logs, `syslogs`, system map files and per process logs such as `cufile.json`, `cufile.log`, `gdsstats`, process stack, and so on.

Usage:

```
./gds_log_collection.py [options]
```

Options:

`-h help`

`-f file1,file2,..` (Note: there should be no spaces between ',')

These files could be any relevant files apart from the one's being collected (such as crash files).

Usage examples:

```
sudo ./gds_log_collection.py - Collects all the relevant logs.
```

```
sudo ./gds_log_collection.py -f file1,file2 - Collects all the relevant files as well as the user specified files.
```

5.3. Resolving an IO Failure with EIO and Stack Trace Warning

You might see an IO failure with EIO and a warning with a stack trace with an `nvfs_mgroup_check_and_set` function in the trace.

This could mean that the EXAScaler filesystem did not honor `O_DIRECT` and fell back to page cache mode. GDS tracks this information in the driver and returns EIO.



Note: The WARNING stack trace is observed only once during the lifetime of the kernel module. You will get an `Error: Input/Output (EIO)`, but the trace message will be printed only once. If you consistently experience this issue, contact support.

5.4. Controlling GPU BAR Memory Usage

1. To show how much BAR Memory is available per GPU, run the following command:

```
$ /usr/local/cuda-x.y/gds/tools/gdscheck
```

2. Review the output, for example:

```
GPU INFO:
GPU Index: 0 bar:1 bar size (MB):32768
GPU Index: 1 bar:1 bar size (MB):32768
```

GDS uses BAR memory in the following cases:

- ▶ When the process invokes `cuFileBufRegister`.
- ▶ When GDS uses the cache internally to allocate bounce buffers per GPU.



Note: There is no per-GPU configuration for cache and BAR memory usage.

Each process can control the usage of BAR memory via the configurable property in the `/etc/cufile.json` file:

```
"properties": {
  // device memory size for reserving bounce buffers for the entire GPU (in KB)
  "max_device_cache_size" : 131072,
  // limit on maximum memory that can be pinned for a given process (in KB)
  "max_device_pinned_mem_size" : 33554432
}
```



Note: This configuration is per process, and the configuration is set across all GPUs.

5.5. Determining the Amount of Cache to Set Aside

By default, 128 MB of cache is set in the configurable `max_device_cache_size` property. However, this does not mean that GDS pre-allocates 128 MB of memory per GPU up front. Memory allocation is done on the fly and is based on need. After the allocation is complete, there is no purging of the cache.

By default, since 128 MB is set, the cache can grow up to 128 MB. Setting the cache is application specific and depends on workload. Refer to the [GPUDirect Storage Best Practices Guide](#) to understand the need of cache and how to set the limit based on guidance in the guide.

5.6. Monitoring BAR Memory Usage

There is no way to monitor the BAR memory usage per process. However, GDS Stats tracks the global BAR usage across all processes. For more information, see the following stat output from `/proc/driver/nvidia_fs/stats` for the GPU with `B:D:F` `0000:34:00.0`:

```
GPU 0000:34:00.0  uid:12a86a5e-3002-108f-ee49-4b51266cdc07 : Registered_MB=32
Cache_MB=10
```

`Registered_MB` tracks how much BAR memory is used when applications are explicitly using the `cuFileBufRegister` API.

`Cache_MB` tracks GDS usage of BAR memory for internal cache.

5.7. Resolving an ENOMEM Error Code

-12 ENOMEM error code.

Each GPU has some BAR memory reserved. The `cuFileBufRegister` function makes the pages that underlie a range of GPU virtual memory accessible to a third-party device. This process is completed by pinning the GPU device memory in BAR space by using the `nvidia_p2p_get_pages` API. If the application tries to pin memory beyond the available BAR space, the `nvidia_p2p_get_pages` API returns a -12 (ENOMEM) error code.

To avoid running out of BAR memory, developers should use this output to manage how much memory is pinned by application. Administrators can use this output to investigate how to limit the pinned memory for different applications.

5.8. GDS and Compatibility Mode

To determine the GDS compatibility mode, complete the following:

1. In the `/etc/cufile.json` file, verify that `allow_compat_mode` is set to `true`.
2. `gdscheck -p` displays whether the `allow_compat_mode` property is set to `true`.
3. Check the `cufile.log` file for the `cufile IO mode: POSIX` message.

This message is in the hot IO path, where logging each instance significantly impacts performance, so the message is only logged when `logging:level` is explicitly set to the `TRACE` mode in the `/etc/cufile.json` file.

5.9. Enabling Compatibility Mode

Compatibility mode can be used by application developers to test the applications with cuFile-enabled libraries under the following conditions:

- ▶ When there is no support for GDS for a specific filesystem.
- ▶ The `nvidia-fs.ko` driver is not enabled in the system by the administrator.

To enable compatibility mode:

1. Remove the `nvidia-fs` kernel driver:


```
$ rmmmod nvidia-fs
```
2. In the `/etc/cufile.json` file, set `compat-mode` to `true`.
3. Set the `CUFILE_FORCE_COMPAT_MODE` environment variable to `true`.

The IO through `cuFileRead/cuFileWrite` will now fall back to the CPU path.

5.10. Tracking the IO After Enabling Compatibility Mode

When GDS is used in compatibility mode, and `cufile_stats` is enabled in the `/etc/cufile.json` file, you can use `gds_stats` or another standard Linux tools, such as `strace`, `iostat`, `iotop`, `SAR`, `ftrace`, and `perf`. You can also use the BPF compiler collection tools to track and monitor the IO.

When compatibility mode is enabled, internally, `cuFileRead` and `cuFileWrite` use POSIX `pread` and `pwrite` system calls, respectively.

5.11. Bypassing GPUDirect Storage

There are some scenarios in which you can bypass GDS.

There are some tunables where GDS IO and POSIX IO can go through simultaneously. The following are cases where GDS can be bypassed without having to remove the GDS driver:

- ▶ On supported filesystems and block devices.

In the `/etc/cufile.json` file, if the `posix_unaligned_writes` config property is set to `true`, the unaligned writes will fall back to the compatibility mode and will not go through GDS. Refer to [Before You Install GDS](#) for a list of supported file systems.

- ▶ On an EXAScaler filesystem

In the `/etc/cufile.json` file, if the `posix_gds_min_kb` config property is set to a certain value (in KB), the IO for which the size is less than or equal to the set value, will fall back to POSIX mode. For example, if `posix_gds_min_kb` is set to 8KB, IOs with a size that is less than or equal to 8KB, will fall back to the POSIX mode.

- ▶ On a WekaIO filesystem:



Note: Currently, `cuFileWrite` will always fallback to the POSIX mode.

In the `/etc/cufile.json` file, if the `allow_compat_mode` config property is set to `true`:

- ▶ If RDMA connections and/or memory registrations cannot be established, `cuFileRead` will fall back to the POSIX mode.
- ▶ `cuFileRead` fails to allocate an internal bounce buffer for non-4K aligned GPU VA addresses.

Refer to the [GPUDirect Storage Best Practices Guide](#) for more information.

5.12. GDS Does Not Work for a Mount

GDS will not be used for a mount in the following cases:

- ▶ When the necessary GDS drivers are not loaded on the system.
- ▶ The filesystem associated with that mount is not supported by GDS.
- ▶ The mount point is denylisted in the `/etc/cufile.json` file.

5.13. Simultaneously Running the GPUDirect Storage IO and POSIX IO on the Same File

Since a file is opened in `O_DIRECT` mode for GDS, applications should avoid mixing `O_DIRECT` and normal I/O to the same file and to overlapping byte regions in the same file.

Even when the filesystem correctly handles the coherency issues in this situation, overall I/O throughput might be slower than using either mode alone. Similarly, applications should avoid mixing `mmap(2)` of files with direct I/O to the same files. Refer to the filesystem-specific documentation for information about additional `O_DIRECT` limitations.

5.14. Running Data Verification Tests Using GPUDirect Storage

GDS has an internal data verification utility, `gdsio_verify`, which is used to test data integrity of reads and writes. Run `gdsio_verify -h` for detailed usage information.

For example:

```
$ /usr/local/cuda-11.2/gds/tools/gdsio_verify -f /mnt/ai200/fio-seq-writes-1 -d 0 -o 0
-s 1G -n 1 -m 1
```

Sample output:

```
gpu index :0, file :/mnt/ai200/fio-seq-writes-1, RING buffer size :0,
gpu buffer alignment :0, gpu buffer offset :0, file offset :0,
io_requested :1073741824, bufregister :true, sync :1, nr ios :1,
fsync :0,
address = 0x560d32c17000
Data Verification Success
```



Note: This test completes data verification of reads and writes through GDS.

Chapter 6. Troubleshooting Performance

This section covers issues related to performance.

6.1. Running Performance Benchmarks with GDS

You can run performance benchmarks with GDS and compare the results with CPU numbers.

GDS has a homegrown benchmarking utility, `/usr/local/cuda-x.y/gds/tools/gdsio`, which helps you compare GDS IO throughput numbers with CPU IO throughput. Run `gdsio -h` for detailed usage information.

Here are some examples:

GDS: Storage --> GPU Memory

```
$ /usr/local/cuda-x.y/tools/gdsio -f /mnt/ai200/fio-seq-writes-1 -d 0 -w 4 -s 10G -i 1M -I 0 -x 0
```

Storage --> CPU Memory

```
$ /usr/local/cuda-x.y/tools/gdsio -f /mnt/ai200/fio-seq-writes-1 -d 0 -w 4 -s 10G -i 1M -I 0 -x 1
```

Storage --> CPU Memory --> GPU Memory

```
$ /usr/local/cuda-x.y/tool/gdsio -f /mnt/ai200/fio-seq-writes-1 -d 0 -w 4 -s 10G -i 1M -I 0 -x 2
```

Storage --> GPU Memory using batch mode

```
$ /usr/local/cuda-x.y/tool/gdsio -f /mnt/ai200/fio-seq-read-1 -d 0 -w 4 -s 10G -i 1M -I 0 -x 6
```

Storage --> GPU Memory using async stream mode

```
$ /usr/local/cuda-x.y/tool/gdsio -f /mnt/ai200/fio-seq-read-1 -d 0 -w 4 -s 10G -i 1M -I 0 -x 5
```

6.2. Tracking Whether GPUDirect Storage is Using an Internal Cache

You can determine whether GDS is using an internal cache.

Prerequisite: Before you start, read the [GPUDirect Storage Best Practices Guide](#).

GDS Stats has per-GPU stats, and each piece of the GPU bus device function (BDF) information is displayed. If the `cache_MB` field is active on a GPU, GDS is using the cache internally to complete the IO.

GDS might use the internal cache when one of the following conditions are true:

- ▶ The `file_offset` that was issued in `cuFileRead/cuFileWrite` is not 4K aligned.
- ▶ The size in `cuFileRead/cuFileWrite` calls are not 4K aligned.
- ▶ The `devPtr_base` that was issued in `cuFileRead/cuFileWrite` is not 4K aligned.
- ▶ The `devPtr_base+devPtr_offset` that was issued in `cuFileRead/cuFileWrite` is not 4K aligned.

6.3. Tracking when IO Crosses the PCIe Root Complex and Impacts Performance

You can track when the IO crosses the PCIe root complex and affects performance.

Refer to [Review Peer Affinity Stats for a Kernel Filesystem and Storage Devices](#) for more information.

6.4. Using GPUDirect Statistics to Monitor CPU Activity

Although you cannot use GDS statistics to monitor CPU activity, you can use the following Linux tools to complete this task:

- ▶ `htop`
- ▶ `perf`
- ▶ `mpstat`

6.5. Monitoring Performance and Tracing with cuFile-* APIs

You can monitor performance and tracing with the cuFile-* APIs.

You can use the FTrace, the Perf, or the BCC-BPF tools to monitor performance and tracing. Ensure that you have the symbols that you can use to track and monitor the performance with a standard Linux IO tool.

6.6. Example: Using Linux Tracing Tools

The `cuFileBufRegister` function makes the pages that underlie a range of GPU virtual memory accessible to a third-party device. This process is completed by pinning the GPU device memory in the BAR space, which is an expensive operation and can take up to a few milliseconds.

You can use the BCC/BPF tool to trace the `cuFileBufRegister` API, understand what is happening in the Linux kernel, and understand why this process is expensive.

Scenario

1. You are running a workload with 8 threads where each thread is issuing `cuFileBufRegister` to pin to the GPU memory.

```
$ ./gdsio -f /mnt/ai200/seq-writes-1 -d 0 -w 8 -s 10G -i 1M -I 0 -x 0
```

2. When IO is in progress, use a tracing tool to understand what is going on with `cuFileBufRegister`:

```
$ /usr/share/bcc/tools# ./funccount -Ti 1 nvfs_mgroup_pin_shadow_pages
```

3. Review the sample output:

```
15:04:56
FUNC                                COUNT
nvfs_mgroup_pin_shadow_pages        8
```

As you can see, the `nvfs_mgroup_pin_shadow_pages` function has been invoked 8 times in one per thread.

4. To see the latency for that function, run:

```
$ /usr/share/bcc/tools# ./funclatency -i 1 nvfs_mgroup_pin_shadow_pages
```

5. Review the output:

```
Tracing 1 functions for "nvfs_mgroup_pin_shadow_pages"... Hit Ctrl-C to end.
```

nsecs	: count	distribution
0 -> 1	: 0	
2 -> 3	: 0	
4 -> 7	: 0	
8 -> 15	: 0	
16 -> 31	: 0	
32 -> 63	: 0	
64 -> 127	: 0	

```

128 -> 255      : 0      |
256 -> 511      : 0      |
512 -> 1023     : 0      |
1024 -> 2047    : 0      |
2048 -> 4095    : 0      |
4096 -> 8191    : 0      |
8192 -> 16383   : 1      |*****
16384 -> 32767  : 7      |*****

```

Seven calls of the `nvfs_mgroup_pin_shadow_pages` function took about 16-32 microseconds. This is probably coming from the Linux kernel `get_user_pages_fast` that is used to pin shadow pages.

`cuFileBufRegister` invokes `nvidia_p2p_get_pages` NVIDIA driver function to pin GPU device memory in the BAR space. This information is obtained by running `$ perf top -g` and getting the call graph of `cuFileBufRegister`.

The following example the overhead of the `nvidia_p2p_get_pages`:

```

$ /usr/share/bcc/tools# ./funclatency -Ti 1 nvidia_p2p_get_pages

15:45:19
nsecs      : count      distribution
 0 -> 1      : 0      |
 2 -> 3      : 0      |
 4 -> 7      : 0      |
 8 -> 15     : 0      |
16 -> 31     : 0      |
32 -> 63     : 0      |
64 -> 127    : 0      |
128 -> 255   : 0      |
256 -> 511   : 0      |
512 -> 1023  : 0      |
1024 -> 2047 : 0      |
2048 -> 4095 : 0      |
4096 -> 8191 : 0      |
8192 -> 16383 : 0      |
16384 -> 32767 : 0      |
32768 -> 65535 : 0      |
65536 -> 131071 : 0      |
131072 -> 262143 : 0      |
262144 -> 524287 : 2      |*****
524288 -> 1048575 : 6      |*****

```

6.7. Tracing the cuFile-* APIs

You can use `nvprof`/`NVIDIA Nsight` to trace the `cuFile-*` APIs.

NVTX static tracepoints are available for public interface in the `libcufile.so` library. After these static tracepoints are enabled, you can view these traces in `NVIDIA Nsight` just like any other `CUDA®` symbols.

You can enable the NVTX tracing using the JSON configuration at `/etc/cufile.json`:

```

"profile": {
    // nvtx profiling on(true)/off(false)
    "nvtx": true,
},

```

6.8. Improving Performance using Dynamic Routing

On platforms where the IO transfers between GPU(s) and the storage NICs involve PCIe traffic across PCIe-host bridge, GPUDirect Storage IO may not see a great throughput especially for writes. Also, certain chipsets may support only P2P read traffic for host bridge traffic. In such cases, the dynamic routing feature can be enabled to debug and identify what routing policy is deemed best for such platforms. This can be illustrated with a single GPU write test with the `gdsio` tool, where there is one Storage NIC and 10 GPUs with NVLINKs access enabled between the GPUS. With dynamic routing enabled, even though the GPU and NIC might be on different sockets, GDS can still achieve the maximum possible write throughput.

```
$ cat /etc/cufile.json | grep rdma_dev
  "rdma_dev_addr_list": [ "192.168.0.19" ],
```

Dynamic Routing OFF:

```
$ cat /etc/cufile.json | grep routing
  "rdma_dynamic_routing": false
$ for i in 0 1 2 3 4 5 6 7 8 9 10;
do
./gdsio -f /mnt/nfs/file1 -d $i -n 0 -w 4 -s 1G -i 1M -x 0 -I 1 -p -T 15 ;
done
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 45792256/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.873560 GiB/sec, Avg_Latency: 1359.280174 usecs ops: 44719
total_time 15.197491 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 45603840/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.867613 GiB/sec, Avg_Latency: 1363.891220 usecs ops: 44535
total_time 15.166344 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 42013696/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.848411 GiB/sec, Avg_Latency: 1373.154082 usecs ops: 41029
total_time 14.066573 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 43517952/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.880763 GiB/sec, Avg_Latency: 1358.207427 usecs ops: 42498
total_time 14.406582 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 34889728/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.341907 GiB/sec, Avg_Latency: 1669.108902 usecs ops: 34072
total_time 14.207836 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 36955136/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.325239 GiB/sec, Avg_Latency: 1680.001220 usecs ops: 36089
total_time 15.156790 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 37075968/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.351491 GiB/sec, Avg_Latency: 1661.198487 usecs ops: 36207
total_time 15.036584 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 35066880/4194304(KiB) IOSize:
1024(KiB) Throughput: 2.235654 GiB/sec, Avg_Latency: 1748.638950 usecs ops: 34245
total_time 14.958656 secs
  url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 134095872/4194304(KiB) IOSize:
1024(KiB) Throughput: 8.940253 GiB/sec, Avg_Latency: 436.982682 usecs ops: 130953
total_time 14.304269 secs
```

```

url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 135974912/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.932070 GiB/sec, Avg_Latency: 437.334849 usecs ops: 132788
total_time 14.517998 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 174486528/4194304 (KiB) IOSize:
1024(KiB) Throughput: 11.238476 GiB/sec, Avg_Latency: 347.603610 usecs ops: 170397
total_time 14.806573 secs

```

Dynamic Routing ON (nvlincs enabled):

```

$ cat /etc/cufile.json | grep routing
    "rdma_dynamic_routing": true
    "rdma_dynamic_routing_order": [ "GPU_MEM_NVLINKS" ]

$ for i in 0 1 2 3 4 5 6 7 8 9 10;
do
./gdsio -f /mnt/nfs/file1 -d $i -n 0 -w 4 -s 1G -i 1M -x 0 -I 1 -p -T 15 ;
done
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 134479872/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.885214 GiB/sec, Avg_Latency: 437.942083 usecs ops: 131328
total_time 14.434092 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 138331136/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.891407 GiB/sec, Avg_Latency: 437.668104 usecs ops: 135089
total_time 14.837118 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 133800960/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.897250 GiB/sec, Avg_Latency: 437.305565 usecs ops: 130665
total_time 14.341795 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 133990400/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.888714 GiB/sec, Avg_Latency: 437.751327 usecs ops: 130850
total_time 14.375893 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 141934592/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.905190 GiB/sec, Avg_Latency: 437.032919 usecs ops: 138608
total_time 15.200055 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 133379072/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.892493 GiB/sec, Avg_Latency: 437.488259 usecs ops: 130253
total_time 14.304222 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 142271488/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.892426 GiB/sec, Avg_Latency: 437.660016 usecs ops: 138937
total_time 15.258004 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 134951936/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.890496 GiB/sec, Avg_Latency: 437.661177 usecs ops: 131789
total_time 14.476154 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 132667392/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.930203 GiB/sec, Avg_Latency: 437.420830 usecs ops: 129558
total_time 14.167817 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 137982976/4194304 (KiB) IOSize:
1024(KiB) Throughput: 8.936189 GiB/sec, Avg_Latency: 437.123356 usecs ops: 134749
total_time 14.725608 secs
url index :0, urlname :192.168.0.2 urlport :18515
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 170469376/4194304 (KiB) IOSize:
1024(KiB) Throughput: 11.231479 GiB/sec, Avg_Latency: 347.818052 usecs ops: 166474
total_time 14.474698 secs

```

Chapter 7. Troubleshooting IO Activity

This section covers issues that are related to IO activity and the interactions with the rest of Linux.

7.1. Managing Coherency of Data in the Page Cache and on Disk

When using GDS, files are often opened with the `O_DIRECT` mode. When IO is complete, in the context of DIRECT IO, it bypasses the page cache.

Starting with CUDA toolkit 12.2 (GDS version 1.7.x) files can also be opened with non-`O_DIRECT` mode. Even in such a case, whenever the library software deems fit, it will follow the GDS enabled `O_DIRECT` path. This conserves coherency by default.

- ▶ On EXAScaler filesystem:
 - ▶ For reads, IO bypasses the page cache and fetches the data directly from backend storage.
 - ▶ When writes are issued, the `nvidia-fs` drivers will try to flush the data in the page cache for the range of offset-length before issuing writes to the VFS subsystem.
 - ▶ The stats that track this information are:
 - ▶ `pg_cache`
 - ▶ `pg_cache_fail`
 - ▶ `pg_cache_eio`
- ▶ On WekaIO filesystem:
 - ▶ For reads, IO bypasses the page cache and fetches the data directly from backend storage.

Chapter 8. EXAScaler Filesystem LNet Troubleshooting

This section describes how to troubleshoot issues with the EXAScaler Filesystem.

8.1. Determining the EXAScaler Filesystem Client Module Version

To check the EXAScaler filesystem Client version, check dmesg after you install the EXAScaler filesystem.



Note: The EXAScaler server version should be EXA-5.2.

This table provides a list of the client kernel module versions that have been tested with DDN AI200 and DDN AI400 systems:

Table 2. Tested Kernel Module Versions

DDN Client Version	Kernel Version	MLNX_OFED version
2.12.3_ddn28	4.15.0	MLNX_OFED 4.7
2.12.3_ddn29	4.15.0	MLNX_OFED 4.7
2.12.3_ddn39	4.15.0	MLNX_OFED 5.1
2.12.5_ddn4	5.4.0	MLNX_OFED 5.1
2.12.6_ddn19	5.4.0	MLNX_OFED 5.3

To verify the client version, run the following command:

```
$ sudo lctl get_param version
```

Sample output:

```
Lustre version: 2.12.3_ddn39
```

8.2. Checking the LNet Network Setup on a Client

To check the LNet network setup on the client:

1. Run the following command.

```
$ sudo lnetctl net show:
```

2. Review the output, for example:

```
net:
- net type: lo
```

8.3. Checking the Health of the Peers

An Lnet health value of 1000 is the best possible value that can be reported for a network interface. Anything less than 1000 indicates that the interface is running in a degraded mode and has encountered some errors.

1. Run the following command ;

```
$ sudo lnetctl net show -v 3 | grep health
```

2. Review the output, for example:

```
health stats:
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
  health stats:
    health value: 1000
```

8.4. Checking for Multi-Rail Support

To verify whether multi-rail is supported:

1. Run the following command:

```
$ sudo lnetctl peer show | grep -i Multi-Rail:
```

2. Review the output, for example:

```
Multi-Rail: True
```

8.5. Checking GDS Peer Affinity

For peer affinity, you need to check whether the expected interfaces are being used for the associated GPUs.

The code snippet below is a description of a test that runs load on a specific GPU. The test validates whether the interface that is performing the send and receive is the interface that is the closest, and is correctly mapped, to the GPU. See [Resetting the nvidia-fs Statistics](#) and [Reviewing Peer Affinity Stats for a Kernel File System and Storage Drivers](#) for more information about the metrics that are used to check peer affinity.

You can run a `gdsio` test for the tools section and monitor the LNET stats. See the readme file for more information. In the `gdsio` test, a write test has been completed on GPU 0. The expected NIC interface for GPU 0 is `ib0` on the NVIDIA DGX-2™ platform. The `lnetctl net show` statistics were previously captured, and after the `gdsio` test, you can see that the RPC send and receive have happened over the `IB0`.

1. Run the `gdsio` test.
2. Review the output, for example:

```
$ sudo lustre_rmmmod
$ sudo mount -t lustre 192.168.1.61@o2ib,192.168.1.62@o2ib:/ai200 /mnt/ai200/
$ sudo lnetctl net show -v 3 | grep health
    health stats:
      health value: 0
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000
    health stats:
      health value: 1000

$ sudo lnetctl net show -v 3 | grep -B 2 -i 'send_count\|recv_count'
    status: up
    statistics:
      send_count: 0
      recv_count: 0
  --
    0: ib0
    statistics:
      send_count: 3
      recv_count: 3
  --
    0: ib2
    statistics:
      send_count: 3
      recv_count: 3
  --
```

```

    0: ib3
    statistics:
      send_count: 2
      recv_count: 2
--
    0: ib4
    statistics:
      send_count: 13
      recv_count: 13
--
    0: ib5
    statistics:
      send_count: 12
      recv_count: 12
--
    0: ib6
    statistics:
      send_count: 12
      recv_count: 12
--
    0: ib7
    statistics:
      send_count: 11
      recv_count: 11

$ echo 1 > /sys/module/nvidia_fs/parameters/peer_stats_enabled

$ /usr/local/cuda-x.y/tools/gdsio -f /mnt/ai200/test -d 0 -n 0 -w 1 -s 1G -i 4K -
x 0 -I 1
IoType: WRITE XferType: GPUD Threads: 1 DataSetSize: 1073741824/1073741824
IOSize: 4(KB),Throughput: 0.004727 GB/sec, Avg_Latency: 807.026154 usecs ops:
262144 total_time 211562847.000000 usecs

$ sudo lnetctl net show -v 3 | grep -B 2 -i 'send_count\|recv_count'

    status: up
    statistics:
      send_count: 0
      recv_count: 0
--
    0: ib0
    statistics:
      send_count: 262149
      recv_count: 524293
--
    0: ib2
    statistics:
      send_count: 6
      recv_count: 6
--
    0: ib3
    statistics:
      send_count: 6
      recv_count: 6
--
    0: ib4
    statistics:
      send_count: 33
      recv_count: 33
--
    0: ib5
    statistics:
      send_count: 32
      recv_count: 32
--
    0: ib6
    statistics:

```

```

          send_count: 32
          recv_count: 32
---
          0: ib7
    statistics:
      send_count: 32
      recv_count: 32

$ cat /proc/driver/nvidia-fs/peer_affinity

GPU P2P DMA distribution based on pci-distance

(last column indicates p2p via root complex)
GPU :0000:be:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:3b:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:e7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:e5:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:e0:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:57:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:39:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:36:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:e2:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:59:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:b7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:b9:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:bc:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:34:00.0 :0 0 23872512 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
GPU :0000:5e:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:5c:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

```

8.6. Checking for LNet-Level Errors

The errors impact the health of individual NICs and affect how the EXAScaler filesystem selects the best peer, which impacts GDS performance.



Note: To run these commands, you must have sudo privileges.

1. Run the following command:

```
$ cat /proc/driver/nvidia-fs/peer_affinity
```

2. Review the output, for example:

```

GPU P2P DMA distribution based on pci-distance
(last column indicates p2p via root complex)
GPU :0000:be:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 GPU :0000:3b:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

```

0 0 0 0 GPU :0000:e7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 GPU :0000:e5:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 GPU :0000:e0:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 GPU :0000:57:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1276417

```

(Note : if peer traffic goes over Root-Port, one of the reasons might be that health of nearest NIC might be affected)

```

GPU :0000:39:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:36:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:e2:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:59:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:b7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:b9:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:bc:00.0 :0 0 7056141 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
GPU :0000:34:00.0 :0 0 8356175 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
GPU :0000:5e:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
GPU :0000:5c:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

```

```
$ sudo lnetctl stats show
```

```

statistics:
  msgs_alloc: 1
  msgs_max: 126
  rst_alloc: 25
  errors: 0
  send_count: 243901
  resend_count: 1
  response_timeout_count: 1935
  local_interrupt_count: 0
  local_dropped_count: 208
  local_aborted_count: 0
  local_no_route_count: 0
  local_timeout_count: 1730
  local_error_count: 0
  remote_dropped_count: 0
  remote_error_count: 0
  remote_timeout_count: 0
  network_timeout_count: 0
  recv_count: 564436
  route_count: 0
  drop_count: 0
  send_length: 336176013248
  recv_length: 95073248
  route_length: 0
  drop_length: 0
lnetctl net show -v 4

```

```
net:
```

```

- net type: o2ib
  local NI(s):
    - nid: 192.168.1.71@o2ib
      status: up
      interfaces:
        0: ib0
      statistics:
        send_count: 171621
        recv_count: 459717

```

```

    drop_count: 0
sent_stats:
  put: 119492
  get: 52129
  reply: 0
  ack: 0
  hello: 0
received_stats:
  put: 119492
  get: 0
  reply: 340225
  ack: 0
  hello: 0
dropped_stats:
  put: 0
  get: 0
  reply: 0
  ack: 0
  hello: 0
health stats:
  health value: 1000
  interrupts: 0
  dropped: 0
  aborted: 0
  no route: 0
  timeouts: 0
  error: 0
tunables:
  peer_timeout: 180
  peer_credits: 32
  peer_buffer_credits: 0
  credits: 256
  peercredits_hiw: 16
  map_on_demand: 1
  concurrent_sends: 64
  fmr_pool_size: 512
  fmr_flush_trigger: 384
  fmr_cache: 1
  ntx: 512
  conns_per_peer: 1
lnd tunables:
  dev cpt: 0
  tcp bonding: 0
CPT: "[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]"
- nid: 192.168.2.71@o2ib
  status: up
  interfaces:
    0: ib1
  statistics:
    send_count: 79
    rcv_count: 79
    drop_count: 0
sent_stats:
  put: 78
  get: 1
  reply: 0
  ack: 0
  hello: 0
received_stats:
  put: 78
  get: 0
  reply: 1
  ack: 0
  hello: 0
dropped_stats:
  put: 0
  get: 0

```

```

    reply: 0
    ack: 0
    hello: 0
health stats:
    health value: 979
    interrupts: 0
    dropped: 0
    aborted: 0
    no route: 0
    timeouts: 1
    error: 0
tunables:
    peer_timeout: 180
    peer_credits: 32
    peer_buffer_credits: 0
    credits: 256
    peercredits_hiw: 16
    map_on_demand: 1
    concurrent_sends: 64
    fmr_pool_size: 512
    fmr_flush_trigger: 384
    fmr_cache: 1
    ntx: 512
    conns_per_peer: 1
lnd tunables:
dev cpt: 0
tcp bonding: 0
CPT: "[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]"
- nid: 192.168.2.72@o2ib
  status: up
  interfaces:
    0: ib3
  statistics:
    send_count: 52154
    rcv_count: 52154
    drop_count: 0
  sent_stats:
    put: 25
    get: 52129
    reply: 0
    ack: 0
    hello: 0
  received_stats:
    put: 25
    get: 52129
    reply: 0
    ack: 0
    hello: 0
  dropped_stats:
    put: 0
    get: 0
    reply: 0
    ack: 0
    hello: 0
  health stats:
    health value: 66
    interrupts: 0
    dropped: 208
    aborted: 0
    no route: 0
    timeouts: 1735
    error: 0
  tunables:
    peer_timeout: 180
    peer_credits: 32
    peer_buffer_credits: 0
    credits: 256

```



```

peercredits_hiw: 16
map_on_demand: 1
concurrent_sends: 64
fmr_pool_size: 512
fmr_flush_trigger: 384
fmr_cache: 1
ntx: 512
conns_per_peer: 1
lnd tunables:
dev cpt: 0
tcp bonding: 0
CPT: "[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]"

```

If you see incrementing error stats, capture the net logging and provide this information for debugging:

```

$ lctl set_param debug=+net
# reproduce the problem
$ lctl dk > logfile.dk

```

8.7. Resolving LNet NIDs Health Degradation from Timeouts

With large machines, such as DGX™ that have multiple interfaces, if Linux routing is not correctly set up, there might be connection failures and other unexpected behavior.

A typical network setting that is used to resolve local connection timeouts is:

```
sysctl -w net.ipv4.conf.all.accept_local=1
```

There are also generic pointers for resolving LNet Network issues. Refer to [MR Cluster Setup](#) for more information.

8.8. Configuring LNet Networks with Multiple OSTs for Optimal Peer Selection

When there are multiple OSTs (Object Storage Targets), and each OST is dual interface, to need to have one interface on each of the LNet for which the client is configured.

For example, you have the following two LNet Subnets on the client side:

- ▶ o2ib
- ▶ o2ib1

The server has only one Lnet subnet, o2ib. In this situation, the routing is not optimal, because you are restricting the ib selection logic to a set of devices, which may not be closest to the GPU. There is no way to reach OST2 except over the LNet to which it is connected.

The traffic that goes to this OST will never be optimal, and this configuration might affect overall throughput and latency. If, however, you configure the server to use two networks, `o2ib0` and `o2ib1`, then OST1 and OST2 can be reached over both networks. When the selection algorithm runs, it will determine that the best path is, for example, OST2 over `o2ib1`.

1. To configure the client-side LNET, run the following command:

```
$ sudo lnetctl net show
```

2. Review the output, for example:

```
net:
- net type: lo
  local NI(s):
  - nid: 0@lo
    status: up
- net type: o2ib
  local NI(s):
  - nid: 192.168.1.71@o2ib
    status: up
    interfaces:
      0: ib0
  - nid: 192.168.1.72@o2ib
    status: up
    interfaces:
      0: ib2
  - nid: 192.168.1.73@o2ib
    status: up
    interfaces:
      0: ib4
  - nid: 192.168.1.74@o2ib
    status: up
    interfaces:
      0: ib6
- net type: o2ib1
  local NI(s):
  - nid: 192.168.2.71@o2ib1
    status: up
    interfaces:
      0: ib1
  - nid: 192.168.2.72@o2ib1
    status: up
    interfaces:
      0: ib3
  - nid: 192.168.2.73@o2ib1
    status: up
    interfaces:
      0: ib5
  - nid: 192.168.2.74@o2ib1
    status: up
    interfaces:
      0: ib7
```

For an optimal configuration, the LNet peer should show two LNet subnets.

In this case, the primary `nid` is only one `o2ib`:

```
$ sudo lnetctl peer show
```

Sample output:

```
peer:
- primary nid: 192.168.1.62@o2ib
  Multi-Rail: True
  peer ni:
  - nid: 192.168.1.62@o2ib
    state: NA
```

```

- nid: 192.168.2.62@o2ib1
  state: NA
- primary nid: 192.168.1.61@o2ib
Multi-Rail: True
peer ni:
- nid: 192.168.1.61@o2ib
  state: NA
- nid: 192.168.2.61@o2ib1
  state: NA

```

From the server side, here is an example of sub-optimal LNet configuration:

```

[root@ai200-090a-vm01 ~]# lnetctl net show
net:
- net type: lo
  local NI(s):
  - nid: 0@lo
    status: up
- net type: o2ib (o2ib1 is not present)
  local NI(s):
  - nid: 192.168.1.62@o2ib
    status: up
    interfaces:
      0: ib0
  - nid: 192.168.2.62@o2ib
    status: up
    interfaces:
      0: ib1

```

Here is an example of an IB configuration for a non-optimal case, where a file is stripped over two OSTs, and there are sequential reads:

```
$ ibdev2netdev -v
```

```

0000:b8:00.1 mlx5_13 (MT4123 - MCX653106A-ECAT) ConnectX-6 VPI adapter card, 100Gb/s
(HDR100, EDR IB and 100GbE), dual-port QSFP56
fw 20.26.4012 port 1
(ACTIVE) ==> ib4 (Up) (o2ib)

```

```
ib4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 2044
```

```
inet 192.168.1.73 netmask 255.255.255.0 broadcast 192.168.1.255
```

```

0000:bd:00.1 mlx5_15 (MT4123 - MCX653106A-ECAT) ConnectX-6 VPI adapter card, 100Gb/s
(HDR100, EDR IB and 100GbE), dual-port QSFP56
fw 20.26.4012 port 1
(ACTIVE) ==> ib5 (Up) (o2ib1)

```

```
ib5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 2044
```

```
inet 192.168.2.73 netmask 255.255.255.0 broadcast 192.168.2.255
```

```
$ cat /proc/driver/nvidia-fs/peer_distance | grep 0000:be:00.0 | grep network
```

```

0000:be:00.0 0000:58:00.1 138 0 network
0000:be:00.0 0000:58:00.0 138 0 network
0000:be:00.0 0000:86:00.1 134 0 network
0000:be:00.0 0000:35:00.0 138 0 network
0000:be:00.0 0000:5d:00.0 138 0 network
0000:be:00.0 0000:bd:00.0 3 0 network
0000:be:00.0 0000:b8:00.1 7 30210269 network (ib4) (chosen peer)
0000:be:00.0 0000:06:00.0 134 0 network
0000:be:00.0 0000:0c:00.1 134 0 network
0000:be:00.0 0000:e6:00.0 138 0 network
0000:be:00.0 0000:3a:00.1 138 0 network
0000:be:00.0 0000:e1:00.0 138 0 network

```

```

0000:be:00.0    0000:bd:00.1    3      4082933 network (ib5) (best peer)
0000:be:00.0    0000:e6:00.1    138    0      network
0000:be:00.0    0000:86:00.0    134    0      network
0000:be:00.0    0000:35:00.1    138    0      network
0000:be:00.0    0000:e1:00.1    138    0      network
0000:be:00.0    0000:0c:00.0    134    0      network
0000:be:00.0    0000:b8:00.0    7      0      network
0000:be:00.0    0000:5d:00.1    138    0      network
0000:be:00.0    0000:3a:00.0    138    0      network

```

Here is an example of an optimal LNet configuration:

```

[root@ai200-090a-vm00 ~]# lnetctl net show
net:
- net type: lo
  local NI(s):
  - nid: 0@lo
    status: up
- net type: o2ib
  local NI(s):
  - nid: 192.168.1.61@o2ib
    status: up
    interfaces:
      0: ib0
- net type: o2ib1
  local NI(s):
  - nid: 192.168.2.61@o2ib1
    status: up
    interfaces:
      0: ib1

```

Chapter 9. Understanding EXAScaler Filesystem Performance

Depending on the type of host channel adapter (HCA), commonly known as a NIC, there are mod parameters that can be tuned for LNet. The NICs that you select should be up and healthy.

To verify the health by mounting and running some basic tests, use `lnetctl health` statistics, and run the following command:

```
$ cat /etc/modprobe.d/lustre.conf
```

Example output:

```
options libcfs cpu_npartitions=24 cpu_pattern=""
options lnet networks="o2ib0(ib1,ib2,ib3,ib4,ib6,ib7,ib8,ib9)"

options ko2iblnd peer_credits=32 concurrent_sends=64 peer_credits_hiw=16
map_on_demand=0
```

9.1. osc Tuning Performance Parameters

The following is information about tuning filesystem parameters.



Note: To maximize the throughput, you can tune the following EXAScaler® filesystem client parameters, based on the network.

1. Run the following command:

```
$ lctl get_param osc.*.max* osc.*.checksums
```

2. Review the output, for example:

```
$ lctl get_param osc.*.max* osc.*.checksums

osc.ai400-OST0024-osc-ffff916f6533a000.max_pages_per_rpc=4096
osc.ai400-OST0024-osc-ffff916f6533a000.max_dirty_mb=512
osc.ai400-OST0024-osc-ffff916f6533a000.max_rpcs_in_flight=32
osc.ai400-OST0024-osc-ffff916f6533a000.checksums=0
```


To check llite parameters, run `$ lctl get_param llite.*.*`.

9.2. Miscellaneous Commands for osc, mdc, and stripesize

If the tuning parameters are set correctly, you can use these parameters to observe.

1. To get an overall EXAScaler filesystem client side statistics, run the following command:

```
$ lctl get_param osc.*.import
```

 Note: The command includes `rpc` information.

2. Review the output, for example:

```
$ watch -d 'lctl get_param osc.*.import | grep -B 1 inflight'
  rpcs:
    inflight: 5
  rpcs:
    inflight: 33
```

3. To get the maximum number of pages that can be transferred per `rpc` in a EXAScaler filesystem client, run the following command:

```
$ lctl get_param osc.*.max_pages_per_rpc
```

4. To get the overall `rpc` statistics from a EXAScaler filesystem client, run the following command:

```
$ lctl set_param osc.*.rpc_stats=clear (to reset osc stats)
$ lctl get_param osc.*.rpc_stats
```

5. Review the output, for example:

```
osc.ai200-OST0000-osc-ffff8e0b47c73800.rpc_stats=
snapshot_time:      1589919461.185215594 (secs.nsecs)
read RPCs in flight: 0
write RPCs in flight: 0
pending write pages: 0
pending read pages: 0
```

	read				write		
pages per rpc	rpcs	%	cum %		rpcs	%	cum %
1:	14222350	77	77		0	0	0
2:	0	0	77		0	0	0
4:	0	0	77		0	0	0
8:	0	0	77		0	0	0
16:	0	0	77		0	0	0
32:	0	0	77		0	0	0
64:	0	0	77		0	0	0
128:	0	0	77		0	0	0
256:	4130365	22	100		0	0	0

	read				write		
rpcs in flight	rpcs	%	cum %		rpcs	%	cum %
0:	0	0	0		0	0	0
1:	3236263	17	17		0	0	0
2:	117001	0	18		0	0	0
3:	168119	0	19		0	0	0
4:	153295	0	20		0	0	0
5:	91598	0	20		0	0	0

6:	42476	0	20		0	0	0
7:	17578	0	20		0	0	0
8:	9454	0	20		0	0	0
9:	7611	0	20		0	0	0
10:	7772	0	20		0	0	0
11:	8914	0	21		0	0	0
12:	9350	0	21		0	0	0
13:	8559	0	21		0	0	0
14:	8734	0	21		0	0	0
15:	10784	0	21		0	0	0
16:	11386	0	21		0	0	0
17:	13148	0	21		0	0	0
18:	15473	0	21		0	0	0
19:	17619	0	21		0	0	0
20:	18851	0	21		0	0	0
21:	21853	0	21		0	0	0
22:	21236	0	21		0	0	0
23:	21588	0	22		0	0	0
24:	23859	0	22		0	0	0
25:	24049	0	22		0	0	0
26:	26232	0	22		0	0	0
27:	29853	0	22		0	0	0
28:	31992	0	22		0	0	0
29:	43626	0	22		0	0	0
30:	116116	0	23		0	0	0
31:	14018326	76	100		0	0	0

To get statistics that are related to client metadata operations, run the following command:



Note: MetaDataClient (MDC) is the client side counterpart of MetaData Server (MDS).

```
$ lctl get_param mdc.*.md_stats
```

To get the stripe layout of the file on the EXAScaler filesystem, run the following command:

```
$ lfs getstripe /mnt/ai200
```

9.3. Getting the Number of Configured Object-Based Disks

To get the number of configured object-based disks:

1. Run the following command:

```
$ lctl get_param lov.*.target_obd
```

2. Review the output, for example:

```
0: ai200-OST0000_UUID ACTIVE
1: ai200-OST0001_UUID ACTIVE
```

9.4. Getting Additional Statistics related to the EXAScaler Filesystem

You can get additional statistics that are related to the EXAScaler Filesystem. Refer to the [Lustre Monitoring and Statistics Guide](#) for more information.

9.5. Getting Metadata Statistics

To get metadata statistics:

1. Run the following command:

```
$ lctl get_param lmv.*.md_stats
```

2. Review the output, for example:

```
snapshot_time      1571271931.653827773 secs.nsecs
close              8 samples [reqs]
create             1 samples [reqs]
getattr           1 samples [reqs]
intent_lock       81 samples [reqs]
read_page         3 samples [reqs]
revalidate_lock   1 samples [reqs]
```

9.6. Checking for an Existing Mount

To check for an existing mount in the EXAScaler Filesystem:

1. Run the following command:

```
$ mount | grep lustre
```

2. Review the output, for example:

```
192.168.1.61@o2ib,192.168.1.62@o2ib1:/ai200 on /mnt/ai200 type lustre
(rw,flock,lazystatfs)
```

9.7. Unmounting an EXAScaler Filesystem Cluster

To unmount an EXAScaler filesystem cluster:

Run the following command.

```
$ sudo umount /mnt/ai200
```


9.8. Getting a Summary of EXAScaler Filesystem Statistics

You can get a summary of statistics for the EXAScaler filesystem.

Refer to the [Lustre Monitoring and Statistics Guide](#) for more information.

9.9. Using GPUDirect Storage in Poll Mode

This section describes how to use GDS in Poll Mode with EXAScaler filesystem files that have a Stripe Count greater than 1.

Currently, if poll mode is enabled, `cuFileReads` or `cuFileWrites` might return bytes that are less than the bytes that were requested. This behavior is POSIX compliant and is observed with files that have a stripe count that is greater than the count in their layout. If behavior occurs, we recommend that the application checks for returned bytes and continues until all of the data is consumed. You can also set the corresponding `properties.poll_mode_max_size_kb`, (say 1024(KB)) value to the lowest possible stripe size in the directory. This ensures that IO sizes that exceed this limit are not polled.

1. To check EXAScaler filesystem file layout, run the following command.

```
$ lfs getstripe <file-path>
```

2. Review the output, for example:

```
lfs getstripe /mnt/ai200/single_stripe/md1.0.0
/mnt/ai200/single_stripe/md1.0.0
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 0
      objidx          objid          objid          group
          0             6146          0x1802           0
```

Chapter 10. Troubleshooting and FAQ for the WekaIO Filesystem

This section provides troubleshooting and FAQ information about the WekaIO file system.

10.1. Downloading the WekaIO Client Package

To download the WekaIO client package:

Run the following command:

```
$ curl http://<IP of one of the WekaIO hosts' IB interface>:14000/dist/v1/install | sh
```

For example, `$ curl http://172.16.8.1:14000/dist/v1/install | sh.`

10.2. Determining Whether the WekaIO Version is Ready for GDS

To determine whether the WekaIO version is ready for GDS:

1. Run the following command:

```
$ weka version
```

2. Review the output, for example:

```
* 3.6.2.5-rdma-beta
```



Note: Currently, the only WekaIO FS version that supports GDS is `* 3.6.2.5-rdma-beta`

10.3. Mounting a WekaIO File System Cluster

The WekaIO filesystem can take a parameter to reserve a fixed number of cores for the user space process.

1. To mount a server_ip 172.16.8.1 with two dedicated cores, run the following command:

```
$ mkdir -p /mnt/weka
$ sudo mount -t wekafs -o num_cores=2 -o
  net=ib0,net=ib1,net=ib2,net=ib3,net=ib4,net=ib5,net=ib6,net=ib7
172.16.8.1/fs01 /mnt/weka
```

2. Review the output, for example:

```
Mounting 172.16.8.1/fs01 on /mnt/weka
Creating weka container
Starting container
Waiting for container to join cluster
Container "client" is ready (pid = 47740)
Calling the mount command
Mount completed successfully
```

10.4. Resolving a Failing Mount

1. Before you use the IB interfaces in the mount options, verify that the interfaces are set up for net=<interface>:

```
$ sudo mount -t wekafs -o num_cores=2 -o
net=ib0,net=ib1,net=ib2,net=ib3,net=ib4,net=ib5,net=ib6,net=ib7
172.16.8.1/fs01 /mnt/weka
```

2. Review the output, for example:

```
Mounting 172.16.8.1/fs01 on /mnt/weka
Creating weka container
Starting container
Waiting for container to join cluster
error: Container "client" has run into an error: Resources
assignment failed: IB/MLNX network devices should have
pre-configured IPs and ib4 has none
```

3. Remove interfaces that do not have network connectivity from the mount options.

```
$ ibdev2netdev

mlx5_0 port 1 ==> ib0 (Up)
mlx5_1 port 1 ==> ib1 (Up)
mlx5_2 port 1 ==> ib2 (Up)
mlx5_3 port 1 ==> ib3 (Up)
mlx5_4 port 1 ==> ib4 (Down)
mlx5_5 port 1 ==> ib5 (Down)
mlx5_6 port 1 ==> ib6 (Up)
mlx5_7 port 1 ==> ib7 (Up)
mlx5_8 port 1 ==> ib8 (Up)
mlx5_9 port 1 ==> ib9 (Up)
```

10.5. Resolving 100% Usage for WekaIO for Two Cores

If you have two cores, and you are experiencing 100% CPU usage:

1. Run the following command.

```
$ top
```

2. Review the output, for example:

```
PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
54816 root        20   0 11.639g 1.452g 392440 R   94.4   0.1 781:06.06 wekanode
54825 root        20   0 11.639g 1.452g 392440 R   94.4   0.1 782:00.32 wekanode
```

When the `num_cores=2` parameter is specified, two cores are used for the user mode poll driver for WekaIO FE networking. This process improves the latency and performance. Refer to the [WekaIO documentation](#) for more information.

10.6. Checking for an Existing Mount in the Weka File System

To check for an existing mount in the WekaIO file system:

1. Run the following command:

```
$ mount | grep wekafs
```

2. Review the output, for example:

```
172.16.8.1/fs01 on /mnt/weka type wekafs (
rw,relatime,writecache,inode_bits=auto,dentry_max_age_positive=1000,
dentry_max_age_negative=0)
```

10.7. Checking for a Summary of the WekaIO Filesystem Status

To check for a summary of the WekaIO file system status.

1. Run the following command:

```
$ weka status
```

2. Review the output, for example:

```
WekaIO v3.6.2.5-rdma-beta (CLI build 3.6.2.5-rdma-beta)
  cluster: Nvidia (e4a4e227-41d0-47e5-aa70-b50688b31f40)
  status: OK (12 backends UP, 72 drives UP)
  protection: 8+2
  hot spare: 2 failure domains (62.84 TiB)
  drive storage: 62.84 TiB total, 819.19 MiB unprovisioned
  cloud: connected
  license: Unlicensed

  io status: STARTED 1 day ago (1584 buckets UP, 228 io-nodes UP)
```

```
link layer: InfiniBand
clients: 1 connected
reads: 61.54 GiB/s (63019 IO/s)
writes: 0 B/s (0 IO/s)
operations: 63019 ops/s
alerts: 3 active alerts, use `Wekaalerts` to list them
```

10.8. Displaying the Summary of the WekaIO Filesystem Statistics

To display a summary of the status of the WekaIO filesystem:

1. Run the following command.

```
$ cat /proc/wekafs/stat
```

2. Review the output, for example:

IO type:	UM Average	UM Longest	KM Average	KM Longest
IO count				
total:	812 us	563448 us	9398 ns	10125660 ns
718319292 (63260 IOPS, 0 MB/sec)				
lookup:	117 us	3105 us	6485 ns	436709 ns
4079 (12041)				
readdir:	0 us	0 us	0 ns	0 ns
0				
mknod:	231 us	453 us	3970 ns	6337 ns
96				
open:	0 us	0 us	0 ns	0 ns
0 (3232)				
release:	0 us	0 us	0 ns	0 ns
0 (2720)				
read:	0 us	0 us	0 ns	0 ns
0				
write:	18957 us	563448 us	495291 ns	920127 ns
983137 (983041)				
getattr:	10 us	10 us	6771 ns	6771 ns
1 (9271)				
setattr:	245 us	424 us	4991 ns	48222 ns
96				
rmdir:	0 us	0 us	0 ns	0 ns
0				
unlink:	0 us	0 us	0 ns	0 ns
0				
rename:	0 us	0 us	0 ns	0 ns
0				
symlink:	0 us	0 us	0 ns	0 ns
0				
readlink:	0 us	0 us	0 ns	0 ns
0				
hardlink:	0 us	0 us	0 ns	0 ns
0				
statfs:	4664 us	5072 us	38947 ns	59618 ns
7				
SG_release:	0 us	0 us	0 ns	0 ns
0				
SG_allocate:	1042 us	7118 us	2161 ns	110282 ns
983072				
falloc:	349 us	472 us	4184 ns	10239 ns
96				
atomic_open:	0 us	0 us	0 ns	0 ns
0				

flock:	0	0 us	0 us	0 ns	0 ns
backcomm:	0	0 us	0 us	0 ns	0 ns
getroot:	19701	19701 us	19701 us	57853 ns	57853 ns
trace:	1	0 us	0 us	0 ns	0 ns
jumbo alloc:	0	0 us	0 us	0 ns	0 ns
jumbo release:	0	0 us	0 us	0 ns	0 ns
jumbo write:	0	0 us	0 us	0 ns	0 ns
jumbo read:	0	0 us	0 us	0 ns	0 ns
keepalive:	184255	46 us	1639968 us	1462 ns	38996 ns
ioctl:	717328710	787 us	50631 us	8732 ns	10125660 ns
setxattr:	0	0 us	0 us	0 ns	0 ns
getxattr:	0	0 us	0 us	0 ns	0 ns
listxattr:	0	0 us	0 us	0 ns	0 ns
removexattr:	0	0 us	0 us	0 ns	0 ns
setfileaccess:	3072	130 us	3437 us	6440 ns	71036 ns
umount:	0	0 us	0 us	0 ns	0 ns

10.9. Why WekaIO Writes Go Through POSIX

For the WekaIO filesystem, GDS supports RDMA based reads and writes. You can use the `fs:weka:rdma_write_support` JSON property to enable writes on supported Weka filesystems. This option is disabled by default. If this option is set to false, writes will be internally staged through system memory, and the cuFile library will use `pwrite` POSIX calls internally for writes.

10.10. Checking for nvidia-fs.ko Support for Memory Peer Direct

To check for `nvidia-fs.ko` support for memory peer direct:

1. Run the following command:

```
$ lsmod | grep nvidia_fs | grep ib_core && echo "Ready for Memory Peer Direct"
```

2. Review the output, for example:

```
ib_core          319488  16
rdma_cm,ib_ipoib,mlx4_ib,ib_srp,iw_cm,nvidia_fs,ib_iser,ib_umad,
rdma_ucm,ib_uverbs,mlx5_ib,ib_cm,ib_ucm
```

```
"Ready for Memory Peer Direct"
```

10.11. Checking Memory Peer Direct Stats

To to check memory peer statistics:

1. Run the following script, which shows the counter for memroy peer direct statistics:

```
list=`ls /sys/kernel/mm/memory_peers/nvidia-fs/`. for stat in $list .
do echo "$stat value: " $(cat /sys/kernel/mm/memory_peers/nvidia-fs/$stat). done
```

2. Review the output.

```
num_alloc_mrs value: 1288
num_dealloc_mrs value: 1288
num_dereg_bytes value: 1350565888
num_dereg_pages value: 329728
num_free_callbacks value: 0
num_reg_bytes value: 1350565888
num_reg_pages value: 329728
version value: 1.0
```

10.12. Checking for Relevant nvidia-fs Statistics for the WekaIO Filesystem

To check for relevant nvida-fs statistics for the WekaIO file system:

1. Run the following command:

```
$ cat /proc/driver/nvidia-fs/stats | egrep -v 'Reads|Writes|Ops|Error'
```

2. Review the output, for example:

```
GDS Version: 1.0.0.80
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2.7.49)

Active Shadow-Buffer (MB): 256
Active Process: 1
Mmap          : n=2088 ok=2088 err=0 munmap=1832
Bar1-map      : n=2088 ok=2088 err=0 free=1826 callbacks=6 active=256
GPU 0000:34:00.0  uuid:12a86a5e-3002-108f-ee49-4b51266cdc07 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=1977
GPU 0000:e5:00.0  uuid:4c2c6b1c-27ac-8bed-8e88-9e59a5e348b5 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
GPU 0000:b7:00.0  uuid:b224ba5e-96d2-f793-3dfd-9caf6d4c31d8 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
GPU 0000:39:00.0  uuid:e8fac7f5-d85d-7353-8d76-330628508052 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
GPU 0000:5c:00.0  uuid:2b13ed25-f0ab-aedb-1f5c-326745b85176 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
GPU 0000:e0:00.0  uuid:df46743a-9b22-30ce-6ea0-62562efaf0a2 : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
GPU 0000:bc:00.0  uuid:c4136168-2a1d-1f3f-534c-7dd725fedbff : Registered_MB=32
  Cache_MB=0 max_pinned_MB=32
```

```
GPU 0000:57:00.0  uuid:54e472f2-e4ee-18dc-f2a1-3595fa8f3d33 : Registered_MB=32
Cache_MB=0 max_pinned_MB=32
```



Note: Reads, Writes, Ops, and Error counters are not available through this interface for the WekaIO filesystem, so the value will be zero. See [Displaying the Summary of the WekaIO Filesystem Statistics](#) about using the Weka status for reads and writes.

10.13. Conducting a Basic WekaIO Filesystem Test

To conduct a basic WekaIO file system test:

1. Run the following command:

```
$ /usr/local/cuda-x.y/tools/gdsio_verify -f /mnt/weka/gdstest/tests/reg1G
-n 1 -m 0 -s 1024 -o 0 -d 0 -t 0 -S -g 4K
```

2. Review the output, for example:

```
gpu index :0, file :/mnt/weka/gdstest/tests/reg1G, RING buffer size :0,
gpu buffer alignment :4096, gpu buffer offset :0, file offset :0,
io_requested :1024, bufregister :false, sync :0, nr ios :1, fsync :0,
address = 0x564ffc5e76c0
Data Verification Success
```

10.14. Unmounting a WekaIO File System Cluster

To unmount a WekaIO file system cluster:

1. Run the following command.

```
$ sudo umount /mnt/weka
```

2. Review the output, for example:

```
Unmounting /mnt/weka
Calling the umount command
umount successful, stopping and deleting client container
Umount completed successfully
```


10.15. Verify the Installed Libraries for the WekaIO Filesystem

The following table summarizes the tasks and command output for verifying the installed libraries for the WekaIO filesystems.

Table 3. Verifying the Installed Libraries for WekaIO Filesystems

Task	Output
Check the WekaIO version.	<pre>\$ weka status WekaIO v3.6.2.5-rdma-beta (CLI build 3.6.2.5-rdma-beta)</pre>
Check whether GDS support for WekaFS is present.	<pre>\$ gdscheck -p [...] WekaFS: Supported Userspace RDMA: Supported [...]</pre>
Check for MLNX_OFED information.	<p>Check with <code>ofed_info -s</code></p> <p>Currently supported with:</p> <p>MLNX_OFED_LINUX-5.1-0.6.6.0</p> <pre>\$ ofed_info -s MLNX_OFED_LINUX-5.1-0.6.6.0:</pre>
Check for the <code>nvidia-fs.ko</code> driver.	<pre>\$ lsmod grep nvidia_fs grep ib_core && echo "Ready for Memory Peer Direct"</pre>
Check for <code>libibverbs.so</code>	<pre>\$ dpkg -s libibverbs-dev Package: libibverbs-dev Status: install ok installed Priority: optional Section: libdevel Installed-Size: 1151 Maintainer: Linux RDMA Mailing List <linux-rdma@vger.kernel.org> Architecture: amd64 Multi-Arch: same Source: rdma-core Version: 47mlnx1-1.47329</pre>

10.16. GDS Configuration File Changes to Support the WekaIO Filesystem

By default, the configuration for Weka RDMA-based writes is disabled.

```
"fs": {
  "weka": {
    // enable/disable WekaFs rdma write
    "rdma_write_support" : false
  }
}
```

To support the WekaIO Filesystem, change the configuration to add a new property, `rdma_dev_addr_list`:

```
"properties": {
  // allow compat mode,
  // this will enable use of cufile posix read/writes
  //"allow_compat_mode": true,

  "rdma_dev_addr_list": [
    "172.16.8.88" , "172.16.8.89",
    "172.16.8.90" , "172.16.8.91",
    "172.16.8.92" , "172.16.8.93",
    "172.16.8.94" , "172.16.8.95"
  ]
}
```

10.17. Check for Relevant User-Space Statistics for the WekaIO Filesystem

To check for relevant user-space statistics for the WekaIO filesystem, issue the following command:

```
$ ./gds_stats -p <pid> -l 3 | grep GPU
```

Refer to [GDS User-Space RDMA Counters](#) for more information about statistics.

10.18. Check for WekaFS Support

If WekaFS support does not exist, the following issues are possible:

Table 4. Weka Filesystem Support Issues

Issue	Action
MLNX_OFED peer direct is not enabled.	<p>Check whether MLNX_OFED is installed (<code>ofed_info -s</code>).</p> <p>This issue can occur if the <code>nvidia-fs</code> Debian package was installed before MLNX_OFED was installed. When this issue occurs, <code>uninstall</code> and <code>reinstall</code> the <code>nvidia-fs</code> package.</p>
RDMA devices are not populated in the <code>/etc/cufile.json</code> file.	Add IP addresses to <code>properties.rdma_dev_addr_list</code> . Currently only IPv4 addresses are supported.
None of the configured RDMA devices are UP.	Check IB connectivity for the interfaces.

Chapter 11. Enabling IBM Spectrum Scale Support with GDS

GDS is supported starting with IBM Spectrum Scale 5.1.2.

After reviewing the NVIDIA GDS documentation, refer to IBM Spectrum Scale 5.1.2. Please see especially the GDS sections in the Planning and Installation guides.

- Planning: <https://www.ibm.com/docs/en/spectrum-scale/5.1.2?topic=considerations-planning-gpudirect-storage>

- Installing: <https://www.ibm.com/docs/en/spectrum-scale/5.1.2?topic=installing-gpudirect-storage-spectrum-scale>

For troubleshooting see <https://www.ibm.com/docs/en/spectrum-scale/5.1.2?topic=troubleshooting-gpudirect-storage-issues>.

11.1. IBM Spectrum Scale Limitations with GDS

Refer to the following documentation for IBM Spectrum Scale Limitations with GDS:

<https://www.ibm.com/docs/en/spectrum-scale/5.1.2?topic=architecture-gpudirect-storage-support-spectrum-scale>

11.2. Checking nvidia-fs.ko Support for Mellanox PeerDirect

Use the following command to check support for memory peer direct.

```
$ cat /proc/driver/nvidia-fs/stats | grep -i "Mellanox PeerDirect Supported"
Mellanox PeerDirect Supported: True
```

In the above example, False means that MLNX_OFED was not installed with GPUDirect Storage support prior to installing nvidia-fs.

The other option to check for Mellanox PeerDirect Support is via “gdscheck -p” output. If it’s enabled, you should be able to see something as below.

```
--Mellanox PeerDirect : Enabled
```

11.3. Verifying Installed Libraries for IBM Spectrum Scale

The following tasks, shown with sample output, can be performed to show installed libraries for IBM Spectrum Scale:

- ▶ Check whether GDS support for IBM Spectrum Scale is present:

```
[~]# /usr/local/cuda/gds/tools/gdscheck -p | egrep -e "Spectrum Scale|PeerDirect|rdma_device_status"
IBM Spectrum Scale : Supported
--Mellanox PeerDirect : Enabled
--rdma_device_status : Up: 2 Down: 0
```

- ▶ Check for MLNX_OFED information:

```
$ ofed_info -s
MLNX_OFED_LINUX-5.4-1.0.3.0:
```

- ▶ Check for nvidia-fs.ko driver:

```
[~]# cat /proc/driver/nvidia-fs/stats
GDS Version: 1.0.0.82
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2.7.49)
Mellanox PeerDirect Supported: True
IO stats: Disabled, peer IO stats: Disabled
Logging level: info

Active Shadow-Buffer (MiB): 0
Active Process: 0
Reads                               : err=0 io_state_err=0
Sparse Reads                         : n=230 io=0 holes=0 pages=0
Writes                               : err=0 io_state_err=237 pg-cache=0 pg-cache-
fail=0 pg-cache-eio=0
Mmap                                 : n=27 ok=27 err=0 munmap=27
Bar1-map                            : n=27 ok=27 err=0 free=27 callbacks=0 active=0
Error                               : cpu-gpu-pages=0 sg-ext=0 dma-map=0 dma-ref=0
Ops                                  : Read=0 Write=0
GPU 0000:2f:00.0  uid:621f7d17-5e7d-8f79-be27-d2f4256ddd88 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=2
```

- ▶ To check for libibverbs.so on Ubuntu:

```
$ dpkg -s libibverbs-dev
root@fscs-sr650-59:~# dpkg -s libibverbs-dev
Package: libibverbs-dev
Status: install ok installed
Priority: optional
Section: libdevel
Installed-Size: 1428
Maintainer: Linux RDMA Mailing List <linux-rdma@vger.kernel.org>
Architecture: amd64
Multi-Arch: same
Source: rdma-core
Version: 54mlnx1-1.54103
```

To check for libibverbs.so on RHEL:

```
[~]# rpm -qi libibverbs
Name       : libibverbs
Version    : 54mlnx1
Release    : 1.54103
```

```

Architecture: x86_64
Install Date: Tue 13 Jul 2021 10:21:18 AM CEST
Group        : System Environment/Libraries
Size         : 535489
License      : GPLv2 or BSD
Signature    : DSA/SHA1, Fri 02 Jul 2021 08:14:44 PM CEST, Key ID
c5ed83e26224c050
Source RPM   : rdma-core-54mlnx1-1.54103.src.rpm
Build Date  : Fri 02 Jul 2021 06:59:01 PM CEST
Build Host   : c-141-24-1-005.mtl.labs.mlnx
Relocations  : (not relocatable)
URL          : https://github.com/linux-rdma/rdma-core
Summary     : A library and drivers for direct userspace use of RDMA
(InfiniBand/iWARP/RoCE) hardware
Description  :
libibverbs is a library that allows userspace processes to use RDMA
"verbs" as described in the InfiniBand Architecture Specification and
the RDMA Protocol Verbs Specification. This includes direct hardware
access from userspace to InfiniBand/iWARP adapters (kernel bypass) for
fast path operations.

Device-specific plug-in ibverbs userspace drivers are included:

- libmlx5: Mellanox ConnectX-4+ InfiniBand HCA

```

11.4. Checking PeerDirect Stats

To check memory peer statistics, run the following script:

```
list=`ls /sys/kernel/mm/memory_peers/nvidia-fs/`; for stat in $list;do echo "$stat
value: " $(cat /sys/kernel/mm/memory_peers/nvidia-fs/$stat); done
```

Sample output:

```

num_alloc_mrs value: 1288
num_dealloc_mrs value: 1288
num_dereg_bytes value: 1350565888
num_dereg_pages value: 329728
num_free_callbacks value: 0
num_reg_bytes value: 1350565888
num_reg_pages value: 32972
version value: 1.0

```

11.5. Checking for Relevant nvidia-fs Stats with IBM Spectrum Scale

Use the following steps to check for relevant `nvidia-fs` statistics for the IBM Spectrum Scale file system.

1. Enable `nvidia-fs` statistics:

```
# echo 1 > /sys/module/nvidia_fs/parameters/rw_stats_enabled
```

2. \$ `cat /proc/driver/nvidia-fs/stats`

3. Review the output:

```
[~]# cat /proc/driver/nvidia-fs/stats
GDS Version: 1.0.0.82
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2.7.49)
Mellanox PeerDirect Supported: True
```

```

IO stats: Disabled, peer IO stats: Disabled
Logging level: info

Active Shadow-Buffer (MiB): 0
Active Process: 0
Reads                          : err=0 io_state_err=0
Sparse Reads                    : n=230 io=0 holes=0 pages=0
Writes                          : err=0 io_state_err=237 pg-cache=0 pg-cache-
fail=0 pg-cache-eio=0
Mmap                            : n=27 ok=27 err=0 munmap=27
Bar1-map                        : n=27 ok=27 err=0 free=27 callbacks=0 active=0
Error                           : cpu-gpu-pages=0 sg-ext=0 dma-map=0 dma-ref=0
Ops                             : Read=0 Write=0
GPU 0000:2f:00.0  uuid:621f7d17-5e7d-8f79-be27-d2f4256ddd88 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=2

```

11.6. GDS User Space Stats for IBM Spectrum Scale for Each Process

To check GDS user space level stats, make sure the “cufile_stats” property in cufile.json is set to 3. Run the following command to check the user space stats for a specific process:

```

$ /usr/local/cuda-<x>.<y>/gds/tools/gds_stats -p <pid> -l 3
cuFile STATS VERSION : 4
GLOBAL STATS:
Total Files: 1
Total Read Errors : 0
Total Read Size (MiB): 7302
Read BandWidth (GiB/s): 0.691406
Avg Read Latency (us): 6486
Total Write Errors : 0
Total Write Size (MiB): 0
Write BandWidth (GiB/s): 0
Avg Write Latency (us): 0
READ-WRITE SIZE HISTOGRAM :
0-4(KiB): 0 0
4-8(KiB): 0 0
8-16(KiB): 0 0
16-32(KiB): 0 0
32-64(KiB): 0 0
64-128(KiB): 0 0
128-256(KiB): 0 0
256-512(KiB): 0 0
512-1024(KiB): 0 0
1024-2048(KiB): 0 0
2048-4096(KiB): 3651 0
4096-8192(KiB): 0 0
8192-16384(KiB): 0 0
16384-32768(KiB): 0 0
32768-65536(KiB): 0 0
65536-...(KiB): 0 0
PER_GPU STATS:
GPU_0 Read: bw=0.690716 util(%)=199 n=3651 posix=0 unalign=0 dr=0 r_sparse=0
r_inline=0 err=0 MiB=7302 Write: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 err=0
MiB=0 BufRegister: n=2 err=0 free=0 MiB=4
PER_GPU POOL BUFFER STATS:
PER_GPU POSIX POOL BUFFER STATS:

PER_GPU RDMA STATS:
GPU 0000:43:00.0 : mlx5_0(130:64):Reads: 3594 Writes: 0 mlx5_1(130:64):Reads:
3708 Writes: 0

```

```
RDMA MRSTATS:
peer name    nr_mrs      mr_size (MiB)
mlx5_0       1              2
mlx5_1       1              2
```

In the example above, 3954 MiB of IBM Spectrum Scale Read-IO went through `mlx5_0` and 3708 MiB of IBM Spectrum Scale Read went through `mlx5_1`. The `RDMA MRSTATS` value shows the number of RDMA memory registrations and size of those registrations.

11.7. GDS Configuration to Support IBM Spectrum Scale

1. Configure the DC key.

The DC key for the IBM Spectrum Scale client can be configured in the following ways:

- ▶ Set the environment variable `CUFILE_RDMA_DC_KEY`. This should be set to a 32-bit hex value. This can be set as shown in the following example.

```
export CUFILE_RDMA_DC_KEY = 0x11223344
```

- ▶ Set the property `rdma_dc_key` in `cufile.json`. This property is a 32-bit value and can be set as shown in the following example.

```
"rdma_dc_key": "0xffeeddcc",
```

In case both the environment variable and the `cufile.json` have the property set, the environment variable `CUFILE_RDMA_DC_KEY` will take precedence over the `rdma_dc_key` property set in `cufile.json`.

In case none of the above is set, the default DC Key configured is `0xffeeddcc`.

2. Configure the IP addresses in `cufile.json`.

The `>rdma_dev_addr_list` property should be set in `cufile.json` with the IP address of the RDMA devices to be used for IO.

```
"properties": {
  -----
  "rdma_dev_addr_list": [
    "172.16.8.88" , "172.16.8.89",
    "172.16.8.90" , "172.16.8.91",
    "172.16.8.92" , "172.16.8.93",
    "172.16.8.94" , "172.16.8.95" ]
  }
  -----
}
```

3. Configure the `max_direct_io_size_kb` property in `cufile.json`.

You can change the IO size with the following property:

```
"properties": {
  -----
  "max_direct_io_size_kb" : 1024
  -----
}
```

4. Configure the `rdma_access_mask` property in `cufile.json`.

This property is a performance tunable. Refer to IBM Spectrum Scale documentation in <https://www.ibm.com/docs/en/spectrum-scale/5.1.2?topic=configuring-gpudirect-storage-spectrum-scale> for optimal configuration of this property.

```
"properties": {
  -----
  "rdma_access_mask": "0x1f",
  -----
}
```

11.8. Scenarios for Falling Back to Compatibility Mode

There are couple of scenarios that will cause the IBM Spectrum Scale IOs to go through compatibility mode, irrespective of the `allow_compat_mode` property's value in `cufile.json`. For a full list of these cases please refer to <http://www.ibm.com/support/pages/node/6444075>.

11.9. GDS Limitations with IBM Spectrum Scale

The current maximum of RDMA memory registrations for a GPU buffer is 16. Hence, the maximum size of memory that can be registered with RDMA per GPU buffer is $16 * \text{max_direct_to_size_kb}$ (set in `cufile.json`). Any GDS IO with IBM Spectrum Scale beyond this offset will go through bounce buffers and might have a performance impact.

Chapter 12. NetApp E-series BeeGFS with GDS Solution Deployment

NetApp supports BeeGFS High Availability.

Refer to the BeeGFS with Netapp E-Series Technical Report on how to deploy the BeeGFS parallel file system: [Netapp BeeGFS Deployment](#). For deployments requiring high availability, refer to [BeeGFS High Availability with NetApp E-Series](#).

12.1. Netapp BeeGFS/GPUDirect Storage and Package Requirements

BeeGFS client and storage with GDS:

CUDA and GDS are only required on the beegfs-client hosts. There are no CUDA or GPUDirect Storage requirements for the BeeGFS server hosts.

12.2. BeeGFS Client Configuration for GDS

After installing beegfs-client, the client build needs to be configured for RDMA and GDS.

1. Edit `/etc/beegfs/beegfs-client-autobuild.conf`. Change line 57 of the file to:

```
buildArgs=-j8 NVFS_H_PATH=/usr/src/mlnx-ofed-kernel-5.4/drivers/nvme/host  
OFED_INCLUDE_PATH=/usr/src/ofa_kernel/default/include
```

This should all be on the same line,

2. Rebuild beegfs-client:

```
sudo /etc/init.d/beegfs-client rebuild
```

12.3. GPU/HCA Topology on the Client - DGX-A100 and OSS servers Client Server

Client Server:

ibdev	netdev	IP	GPU	Numa	OSS	Target	Mount Point
mlx5_4	ibp97s0f0	10.10.0.177	0,2,3	0	meta 1	5,6,7,8	/mnt/ beegfs/
mlx5_5	ibp97s0f1	10.10.1.177	0,2,3	0	meta 1	5,6,7,8	/mnt/ beegfs/
mlx5_10	ibp225s0f0	10.10.2.157	4,5,6,7	4	meta 2	1,2,3,4	/mnt/ beegfs/
mlx5_11	ibp225s0f1	10.10.3.157	4,5,6,7	4	meta 2	1,2,3,4	/mnt/ beegfs/

OSS Servers:

OSS	ID	IP	Numa
meta01- numa0-1	1001	10.10.0.131:8003	0
meta01- numa1-2	1002	10.10.1.131:8004	1
meta02- numa0-1	2001	10.10.2.132:8003	0
meta02- numa1-2	2002	10.10.3.132:8004	1

12.4. Verify the Setup

To verify the setup, run the following commands on any client:

12.4.1. List the Management Node

```
root@dgxa100-b:/sys/class# beegfs-ctl --listnodes --nodetype=management --details
meta-02.cpoc.local [ID: 1]
Ports: UDP: 8008; TCP: 8008
Interfaces: em3(TCP)
```

12.4.2. List the Metadata Nodes

```
root@dgxa100-b:/sys/class# beegfs-ctl --listnodes --nodetype=meta -details
meta01- numa0-1-meta [ID: 1101]
  Ports: UDP: 8005; TCP: 8005
  Interfaces: ib0:net1(RDMA) ib0:net1(TCP)
meta01- numa1-2-meta [ID: 1102]
  Ports: UDP: 8006; TCP: 8006
  Interfaces: ib2:net3(RDMA) ib2:net3(TCP)
```

```
meta02-numa0-1-meta [ID: 2101]
  Ports: UDP: 8005; TCP: 8005
  Interfaces: ib0:net0(RDMA) ib0:net0(TCP)
meta02-numa1-2-meta [ID: 2102]
  Ports: UDP: 8006; TCP: 8006
  Interfaces: ib2:net2(RDMA) ib2:net2(TCP)
Number of nodes: 4
Root: 2101
```

12.4.3. List the Storage Nodes

```
root@dgxa100-b:/sys/class# beegfs-ctl --listnodes --nodetype=storage -details

meta01-numa0-1 [ID: 1001]
  Ports: UDP: 8003; TCP: 8003
  Interfaces: ib0:net1(RDMA) ib0:net1(TCP)
meta01-numa1-2 [ID: 1002]
  Ports: UDP: 8004; TCP: 8004
  Interfaces: ib2:net3(RDMA) ib2:net3(TCP)
meta02-numa0-1 [ID: 2001]
  Ports: UDP: 8003; TCP: 8003
  Interfaces: ib0:net0(RDMA) ib0:net0(TCP)
meta02-numa1-2 [ID: 2002]
  Ports: UDP: 8004; TCP: 8004
  Interfaces: ib2:net2(RDMA) ib2:net2(TCP)
Number of nodes: 4
```

12.4.4. List the Client Nodes

```
root@dgxa100-b:/sys/class# beegfs-ctl --listnodes --nodetype=client -details
B4330-6161F689-dgxa100-b [ID: 11]
  Ports: UDP: 8004; TCP: 0
  Interfaces: ibp97s0f0(RDMA) ibp97s0f0(TCP) ibp97s0f1(TCP) ibp97s0f1(RDMA)
  ibp225s0f0(TCP)
  ibp225s0f0(RDMA) ibp225s0f1(TCP) ibp225s0f1(RDMA)
```

12.4.5. Display Client Connections

```
root@dgxa100-b:/sys/class# beegfs-net

mgmt_nodes
=====
meta-02.cpoc.local [ID: 1]
  Connections: TCP: 1 (192.168.0.132:8008);

meta_nodes
=====
meta01-numa0-1-meta [ID: 1101]
  Connections: RDMA: 1 (10.10.1.131:8005);
meta01-numa1-2-meta [ID: 1102]
  Connections: RDMA: 1 (10.10.3.131:8006);
meta02-numa0-1-meta [ID: 2101]
  Connections: RDMA: 1 (10.10.0.132:8005);
meta02-numa1-2-meta [ID: 2102]
  Connections: RDMA: 1 (10.10.2.132:8006);

storage_nodes
=====
meta01-numa0-1 [ID: 1001]
  Connections: RDMA: 8 (10.10.1.131:8003);
meta01-numa1-2 [ID: 1002]
  Connections: RDMA: 8 (10.10.3.131:8004);
meta02-numa0-1 [ID: 2001]
  Connections: RDMA: 16 (10.10.0.132:8003);
meta02-numa1-2 [ID: 2002]
```

```
Connections: RDMA: 8 (10.10.2.132:8004);
```

12.4.6. Verify Connectivity to the Different Services

```
root@dgxa100-b:/sys/class# beegfs-check-servers

Management
=====
meta-02.cproc.local [ID: 1]: reachable at 192.168.0.132:8008 (protocol: TCP)

Metadata
=====
meta01-numa0-1-meta [ID: 1101]: reachable at 10.10.1.131:8005 (protocol: TCP)
meta01-numa1-2-meta [ID: 1102]: reachable at 10.10.3.131:8006 (protocol: TCP)
meta02-numa0-1-meta [ID: 2101]: reachable at 10.10.0.132:8005 (protocol: TCP)
meta02-numa1-2-meta [ID: 2102]: reachable at 10.10.2.132:8006 (protocol: TCP)

Storage
=====
meta01-numa0-1 [ID: 1001]: reachable at 10.10.1.131:8003 (protocol: TCP)
meta01-numa1-2 [ID: 1002]: reachable at 10.10.3.131:8004 (protocol: TCP)
meta02-numa0-1 [ID: 2001]: reachable at 10.10.0.132:8003 (protocol: TCP)
meta02-numa1-2 [ID: 2002]: reachable at 10.10.2.132:8004 (protocol: TCP)
```

12.4.7. List Storage Pools

In this example we used the default mounting point:

```
root@dgxa100-b:/sys/class# sudo beegfs-ctl -liststoragepools
```

Pool ID	Pool Description	Targets	Buddy Groups
1	Default	1,2,3,4,5,6,7,8	

12.4.8. Display the Free Space and inodes on the Storage and Metadata Targets

```
root@dgxa100-b:/sys/class# beegfs-df
```

METADATA SERVERS:								
TargetID	Cap. Pool	Total	Free	%	ITotal	IFree	%	
1101	normal	573.3GiB	572.9GiB	100%	401.1M	401.0M	100%	
1102	normal	573.3GiB	572.9GiB	100%	401.1M	401.0M	100%	
2101	normal	573.3GiB	572.9GiB	100%	401.1M	401.0M	100%	
2102	normal	573.3GiB	572.9GiB	100%	401.1M	401.0M	100%	

STORAGE TARGETS:								
TargetID	Cap. Pool	Total	Free	%	ITotal	IFree	%	
1	normal	2574.7GiB	1470.8GiB	57%	270.1M	270.1M	100%	
2	normal	2574.7GiB	1404.0GiB	55%	270.1M	270.1M	100%	
3	normal	2574.7GiB	1265.5GiB	49%	270.1M	270.1M	100%	
4	normal	2574.7GiB	1278.5GiB	50%	270.1M	270.1M	100%	
5	normal	2574.7GiB	1274.0GiB	49%	270.1M	270.1M	100%	
6	normal	2574.7GiB	1342.6GiB	52%	270.1M	270.1M	100%	
7	normal	2574.7GiB	1485.3GiB	58%	270.1M	270.1M	100%	
8	normal	2574.7GiB	1481.7GiB	58%	270.1M	270.1M	100%	

12.5. Testing

12.5.1. Verifying Integration is Working

Once beegfs-client has been started with GDS support, a basic test can be performed to verify that the integration is working:

```
root@dgxa100-b:/usr/local/cuda-11.4/gds/tools# ./gdscheck.py -p
GDS release version: 1.1.1.14
nvidia_fs version: 2.7 libcufile version: 2.9
=====
ENVIRONMENT:
=====
=====
DRIVER CONFIGURATION:
=====
NVMe           : Supported
NVMeOF         : Unsupported
SCSI           : Unsupported
ScaleFlux CSD  : Unsupported
NVMesh         : Unsupported
DDN EXAScaler  : Unsupported
IBM Spectrum Scale : Unsupported
NFS            : Unsupported
BEEGFS        : Supported
WekaFS         : Unsupported
Userspace RDMA : Unsupported
--Mellanox PeerDirect : Disabled
--rdma library   : Not Loaded (libcufile_rdma.so)
--rdma devices   : Not configured
--rdma_device_status : Up: 0 Down: 0
=====
CUFILE CONFIGURATION:
=====
properties.use_compat_mode : true
properties.gds_rdma_write_support : false
properties.use_poll_mode : false
properties.poll_mode_max_size_kb : 4
properties.max_batch_io_timeout_msecs : 5
properties.max_direct_io_size_kb : 16384
properties.max_device_cache_size_kb : 131072
properties.max_device_pinned_mem_size_kb : 33554432
properties.posix_pool_slab_size_kb : 4 1024 16384
properties.posix_pool_slab_count : 128 64 32
properties.rdma_peer_affinity_policy : RoundRobin
properties.rdma_dynamic_routing : 0
fs.generic.posix_unaligned_writes : false
fs.lustre.posix_gds_min_kb: 0
fs.beegfs.posix_gds_min_kb: 0
fs.weka.rdma_write_support: false
profile.nvtx : false
profile.cufile_stats : 0
miscellaneous.api_check_aggressive : false
=====
IOMMU: disabled
Platform verification succeeded
```

12.5.2. Conducting a Basic NetApp BeeGFS Filesystem Test

```
/usr/local/cuda/gds/tools/gdsio_verify -f /mnt/beegfs/file 1g -d 0 -o 0 -s 1G -n 1 -  
m 1  
gpu index :0, file :/mnt/beegfs/file 1g, gpu buffer alignment :0, gpu buffer  
offset :0, gpu devptr offset :0, file offset :0, io_requested :1073741824,  
io_chunk_size :1073741824, bufregister :true, sync :1, nr ios :1, fsync :0,  
Data Verification Success
```

Chapter 13. Setting Up and Troubleshooting VAST Data (NFSoRDMA +MultiPath)

This section provides information about how to set up and troubleshoot VAST data (NFSoRDMA+MultiPath).

13.1. Installing MLNX_OFED and VAST NFSoRDMA+Multipath Packages

13.1.1. Client Software Requirements

The following table lists the minimum client software requirements for using MLNX_OFED and VAST NFSoRDMA+Multipath packages.

Table 5. Minimum Client Requirements

NFS Connection Type	Linux Kernel	MLNX_OFED
NFSoRDMA + Multipath	The following kernel versions are supported: <ul style="list-style-type: none">▶ 4.15▶ 4.18▶ 5.4	The following MLNX_OFED versions are supported: <ul style="list-style-type: none">▶ 4.6▶ 4.7▶ 5.0▶ 5.1▶ 5.3

For the most up to date supportability matrix and client configuration steps and package downloads, refer to: <https://support.vastdata.com/hc/en-us/articles/360016813140-NFSoRDMA-with-Multipath>.

MLNX_OFED must be installed for the VAST NFSoRDMA+Multipath package to function optimally. It is also important to download the correct VAST software packages to match your kernel+MLNX_OFED version combination. Refer to [Troubleshooting and FAQ for NVMe and NVMeOF](#) support for information about how to install MLNX_OFED with GDS support.

- ▶ To verify the current version of MLNX_OFED, issue the following command:

```
$ ofed_info -s
MLNX_OFED_LINUX-5.3-0.6.6.01:
```

- ▶ To verify the currently installed Linux kernel version, issue the following command:

```
$ uname -r -v
```

After you verify that your system has the correct combination of kernel and MLNX_OFED, you can install the VAST Multipath package.

13.1.2. Install the VAST Multipath Package

Although the VAST Multipath with NFSoRDMA package has been submitted upstream for inclusion in a future kernel release, it is currently only available as a download from: <https://support.vastdata.com/hc/en-us/articles/360016813140-NFSoRDMA-with-Multipath>.

Be sure to download the correct .deb file that is based on your kernel and MLNX_OFED version.

1. Install the VAST NFSoRDMA+Multipath package.

```
$ sudo apt-get install mlnx-nfsrdma-*.deb
```


2. Generate a new initramfs image.

```
$ sudo update-initramfs -u -k `uname -r`
```

3. Verify that the package is installed, and the version is the number that you expected.

```
$ dpkg -l | grep mlnx-nfsrdma
ii  mlnx-nfsrdma-dkms      5.3-OFED.5.1.0.6.6.0      all  DKMS support for NFS
RDMA kernel module
```

4. Reboot the host and run the following commands to verify that the correct version is loaded.

 Note: The versions shown by each command should match.

```
$ cat /sys/module/sunrpc/srcversion
4CC8389C7889F82F5A59269
$ modinfo sunrpc | grep srcversion
srcversion:      4CC8389C7889F82F5A59269
```

13.2. Set Up the Networking

This section provides information about how to set up client networking for VAST for GDS.

To ensure optimal GPU-to-storage performance while leveraging GDS, you need to configure VAST and client networking in a balanced manner.

13.2.1. VAST Network Configuration

VAST is a multi-node architecture. Each node has multiple high-speed (IB-HDR100 or 100GbE) interfaces, which can host-client-facing Virtual IPs. Refer to [VAST-Managing Virtual IP \(VIP\) Pools](#) for more information.

Here is the typical workflow:

1. Multiply the number of VAST-Nodes * 2 (one per Interface).
2. Create a VIP Pool with the resulting IP count.
3. Place the VAST-VIP Pool on the same IP-subnet as the client.

13.2.2. Client Network Configuration

The following is information about client network configuration.

Typically, GPU optimized clients (such as the NVIDIA DGX-2 and DGX-A100) are configured with multiple high speed network interface cards (NICs). In the following example, the system contains 8 separate NICs that were selected for optimal balance for NIC -->GPU and NIC -->CPU bandwidth.

```
$ sudo ibdev2netdev
mlx5_0 port 1 ==> ibp12s0 (Up)
mlx5_1 port 1 ==> ibp18s0 (Up)
mlx5_10 port 1 ==> ibp225s0f0 (Down)
mlx5_11 port 1 ==> ibp225s0f1 (Down)
mlx5_2 port 1 ==> ibp75s0 (Up)
mlx5_3 port 1 ==> ibp84s0 (Up)
mlx5_4 port 1 ==> ibp97s0f0 (Down)
mlx5_5 port 1 ==> ibp97s0f1 (Down)
mlx5_6 port 1 ==> ibp141s0 (Up)
mlx5_7 port 1 ==> ibp148s0 (Up)
mlx5_8 port 1 ==> ibp186s0 (Up)
mlx5_9 port 1 ==> ibp202s0 (Up)
```

Not all interfaces are connected, and this is to ensure optimal bandwidth.

When using the aforementioned VAST NFSoRDAM+Multipath package, it is recommended to assign static IP's to each interface on the same subnet, which should also match the subnet configured on the VAST VIP Pool. If using GDS with NVIDIA DGX-A100's, a simplistic netplan is all that is required, for example:

```
ibp12s0:
  addresses: [172.16.0.17/24]
  dhcp4: no
ibp141s0:
  addresses: [172.16.0.18/24]
  dhcp4: no
ibp148s0:
  addresses: [172.16.0.19/24]
  dhcp4: no
```

However, if you are using other systems, or non-GDS code, you need to apply the following code to ensure that the proper interfaces are used to traverse from Client-->VAST.



Note: See the `routes` section for each interface in the following sample.

```
$ cat /etc/netplan/01-netcfg.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    enp226s0:
      dhcp4: yes
    ibp12s0:
      addresses: [172.16.0.25/24]
      dhcp6: no
      routes:
        - to: 172.16.0.0/24
          via: 172.16.0.25
          table: 101
      routing-policy:
        - from: 172.16.0.25
          table: 101
    ibp18s0:
      addresses: [172.16.0.26/24]
      dhcp4: no
      routes:
        - to: 172.16.0.0/24
          via: 172.16.0.26
          table: 102
      routing-policy:
        - from: 172.16.0.26
          table: 102
    ibp75s0:
      addresses: [172.16.0.27/24]
      dhcp4: no
      routes:
        - to: 172.16.0.0/24
          via: 172.16.0.27
          table: 103
      routing-policy:
        - from: 172.16.0.27
          table: 103
    ibp84s0:
      addresses: [172.16.0.28/24]
      dhcp4: no
      routes:
        - to: 172.16.0.0/24
          via: 172.16.0.28
          table: 104
      routing-policy:
        - from: 172.16.0.28
          table: 104
    ibp141s0:
      addresses: [172.16.0.29/24]
      dhcp4: no
      routes:
        - to: 172.16.0.0/24
          via: 172.16.0.29
          table: 105
      routing-policy:
        - from: 172.16.0.29
          table: 105
    ibp148s0:
```

```

addresses: [172.16.0.30/24]
dhcp4: no
routes:
  - to: 172.16.0.0/24
    via: 172.16.0.30
    table: 106
routing-policy:
  - from: 172.16.0.30
    table: 106
ibp186s0:
addresses: [172.16.0.31/24]
dhcp4: no
routes:
  - to: 172.16.0.0/24
    via: 172.16.0.31
    table: 107
routing-policy:
  - from: 172.16.0.31
    table: 107
ibp202s0:
addresses: [172.16.0.32/24]
dhcp4: no
routes:
  - to: 172.16.0.0/24
    via: 172.16.0.32
    table: 108
routing-policy:
  - from: 172.16.0.32
    table: 108

```

After making changes to the netplan, before issuing the following command, ensure that you have a IPMI/console connection to the client:

```
$ sudo netplan apply
```

13.2.3. Verify Network Connectivity

Once the proper netplan is applied, verify connectivity between all client interfaces and all VAST-VIPs with a ping loop:

```

# Replace with appropriate interface names
$ export IFACES="ibp12s0 ibp18s0 ibp75s0 ibp84s0 ibp141s0 ibp148s0 ibp186s0
ibp202s0"
# replace with appropriate VAST-VIPs
$ export VIPS=$(echo 172.16.0.{101..116})
$ echo "starting pingtest" > pingtest.log
$ for i in $IFACES;do for v in $VIPS; do echo $i >> pingtest.log; ping -c 1 $v -W
0.2 -I $i|grep loss >> pingtest.log;done;done;
# Verify no failures:
$ grep '100%' pingtest.log

```

You should also verify that one of the following conditions are met:

- ▶ All client interfaces are directly cabled to the same IB switches as VAST.
- ▶ There are sufficient InterSwitch Links (ISLs) between client-switches, and switches to which VAST is connected.

To verify the current IB switch topology, issue the following command:

```
$ sudo ibnetdiscover
<output trimmed>
```

```
[37] "H-b8599f0300c3f4cb"[1](b8599f0300c3f4cb) # "vastraplab-cn1 HCA-2" lid 55 2xHDR
# <-- example of Vast-Node

[43] "S-b8599f0300e361f2"[43] # "MF0;RL-QM87-C20-U33:MQM8700/U1" lid 1
4xHDR # <-- example of ISL

[67] "H-1c34da030073c27e"[1](1c34da030073c27e) # "rl-dgxa-c21-u19 mlx5_9" lid 23
4xHDR # <-- example of client
```

13.3. Mount VAST NFS

To fully utilize available VAST VIPs, you must mount the filesystem by issuing the following command:

```
$ sudo mount -o proto=rdma,port=20049,vers=3 \
-o noidlexprt,nconnect=40 \
-o localports=172.16.0.25-172.16.0.32 \
-o remoteports=172.16.0.101-172.16.0.140 \
172.16.0.101:/mnt/vast
```

The options are:

proto

RDMA must be specified.

port=20049

Must be specified, this is RDMA control port.

noidlexprt

Do not disconnect idle connections. This is to detect and recover failing connections when there are no pending I/O's.

nconnect

Number of concurrent connections. Should be divisible evenly by the number of remoteports specified below for best balance.

localports

A list of IPv4 addresses for the local ports to bind.

Remoteports

A list of NFS server IPv4 ports to bind.

For both localports and remoteports you can specify an inclusive range with the -delimiter, for example, *FIRST-LAST*. Multiple ranges or individual IP addresses can be separated by ~ (a tilde)

13.4. Debugging and Monitoring VAST Data

Typically, `mountstats` under `/proc` shows `xprt` statistics. However, instead of modifying it in a non-compatible way with the `nfsstat` utility, the VAST Multipath package extends

mountstats with extra state reporting, to be exclusively accessed from /sys/kernel/debug.

The stats node was added for each RPC client, and the RPC client 0 shows the mount that is completed:

```
$ sudo cat /sys/kernel/debug/sunrpc/rpc_clnt/0/stats
```

The added information is multipath IP address information per xprt and xprt state in string format.

For example:

```
xprt: rdma 0 0 1 0 24 3 3 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0
      172.25.1.101 -> 172.25.1.1, state: CONNECTED BOUND
xprt: rdma 0 0 1 0 24 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0
      172.25.1.102 -> 172.25.1.2, state: CONNECTED BOUND
xprt: rdma 0 0 1 0 23 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0
      172.25.1.103 -> 172.25.1.3, state: CONNECTED BOUND
xprt: rdma 0 0 1 0 22 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0
      172.25.1.104 -> 172.25.1.4, state: CONNECTED BOUND
xprt: rdma 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      172.25.1.101 -> 172.25.1.5, state: BOUND
xprt: rdma 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      172.25.1.102 -> 172.25.1.6, state: BOUND
xprt: rdma 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      172.25.1.103 -> 172.25.1.7, state: BOUND
xprt: rdma 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      172.25.1.104 -> 172.25.1.8, state: BOUND
```

Chapter 14. Troubleshooting and FAQ for NVMe and NVMeOF Support

This section provides troubleshooting information for NVMe and NVMeOF support.

14.1. MLNX_OFED Requirements and Installation

- ▶ To enable GDS support for NVMe and NVMeOF, you need to install at least MLNX_OFED 5.3 or later.
- ▶ You must install MLNX_OFED with support for GDS.

After installation is complete, for the changes to take effect, use `update -initramfs` and reboot. The Linux kernel version that was tested with MLNX_OFED 5.3-1.0.5.01 is 4.15.0-x and 5.4.0-x. Issue the following command:

```
$ sudo ./mlnxofedinstall --with-nvme --with-nfsrdma --enable-gds --add-kernel-support --dkms
```



Note: With MLNX_OFED 5.3 onwards, the `--enable-gds` flag is no longer necessary.

```
$ sudo update-initramfs -u -k `uname -r`  
$ reboot
```

14.2. Determining Whether the NVMe device is Supported for GDS

NVMe devices must be compatible with GDS; the device cannot have the block device integrity capability.

For device integrity, the Linux block layer completes the metadata processing based on the payload in the host memory. This is a deviation from the standard GDS IO path and, as a result, cannot accommodate these devices. The cuFile file registration will fail when

this type of underlying device is detected with appropriate error log in the `cufile.log` file.

```
$ cat /sys/block/<nvme>/integrity/device_is_integrity_capable
```

14.3. RAID Support in GDS

Currently, GDS only supports RAID 0.

14.4. Mounting a Local Filesystem for GDS

Currently, EXT4 and XFS are the only block device based filesystem that GDS supports. Because of Direct IO semantics, the ext4 file system must be mounted with the journaling mode set to `data=ordered`. This has to be explicitly part of the mount options so that the library can recognize it:

```
$ sudo mount -o data=ordered /dev/nvme0n1 /mnt
```

If the EXT4 journaling mode is not in the expected mode, the `cuFileHandleRegister` will fail, and an appropriate error message will be logged in the log file. For instance, in the following case, `/mnt1` is mounted with writeback, and GDS returns an error:

```
$ mount | grep /mnt1
/dev/nvme0n1p2 on /mnt1 type ext4 (rw,relatime,data=writeback)

$ ./cufile_sample_001 /mnt1/foo 0
opening file /mnt1/foo
file register error:GPUDirect Storage not supported on current file
```

14.5. Check for an Existing EXT4 Mount

To check for an existing EXT4 mount:

```
$ mount | grep ext4
/dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/nvme1n1 on /mnt type ext4 (rw,relatime,data=ordered)
/dev/nvme0n1p2 on /mnt1 type ext4 (rw,relatime,data=writeback)
```



Note: A similar check can be used to check for an existing XFS mount, for example:

```
mount | grep xfs
```

14.6. Check for IO Statistics with Block Device Mount

The following command and partial log show you how to obtain the IO statistics:

```
$ sudo iotop
Actual DISK READ:      0.00 B/s | Actual DISK WRITE:      193.98 K/s
```

TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
881	be/3	root	0.00 B/s	15.52 K/s	0.00 %	0.01 %	[jbd2/sda2-8]
1	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	init splash

14.7. RAID Group Configuration for GPU Affinity

Creating one RAID group from the available NVMe devices might not be optimal for GDS performance. You might need to create RAID groups that consist of devices that have a pci-affinity with the specified GPU. This is required to prevent and cross-node P2P traffic between the GPU and the NVMe devices.

If affinity is not enforced, GDS will use an internal mechanism of device bounce buffers to copy data from the NVMe devices to an intermediate device that is closest to the drives and copy the data back to the actual GPU. If NVLink is enabled, this will speed up these transfers.

14.8. Conduct a Basic EXT4 Filesystem Test

To conduct a basic EXT4 filesystem test, issue the following command:

```
$ /usr/local/cuda-x.y/gds/tools/gdsio_verify -f /mnt/nvme/gdstest/tests/reg1G -n 1
-m 0 -s 1024 -o 0 -d 0 -t 0 -S -g 4K
```

Sample output:

```
gpu index :0, file :/mnt/weka/gdstest/tests/reg1G, RING buffer size :0, gpu buffer
alignment :4096, gpu buffer offset :0, file offset :0, io_requested :1024,
bufregister :false, sync :0, nr ios :1, fsync :0,
address = 0x564ffc5e76c0
Data Verification Success
```

14.9. Unmount a EXT4 Filesystem

To unmount an EXT4 filesystem, issue the following command:

```
$ sudo umount /mnt/
```

14.10. Udev Device Naming for a Block Device

The library has a limitation when identifying the NVMe-based block devices in that it expects device names to have the `nvme` prefix as part of the naming convention.

14.11. BATCH I/O Performance

It has been observed that m separate batches with n/m entries each, showed better performance than 1 batch with n entries especially in case of NVMe based storage.

Chapter 15. Displaying GDS NVIDIA FS Driver Statistics

GDS exposes the IO statistics information on the `procfs` filesystem.

1. To display driver statistics, run the following command.

```
$ cat /proc/driver/nvidia-fs/stat
```

2. Review the output, for example:

```
GDS Version: 1.0.0.71
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2:7:47)
Mellanox PeerDirect Supported: True
IO stats: Enabled, peer IO stats: Enabled
Logging level: info

Active Shadow-Buffer (MiB): 0
Active Process: 0
Reads                : n=0 ok=0 err=0 readMiB=0 io_state_err=0
Reads                : Bandwidth(MiB/s)=0 Avg-Latency(usec)=0
Sparse Reads : n=6 io=0 holes=0 pages=0
Writes               : n=0 ok=0 err=0 writeMiB=0 io_state_err=0
                   pg-cache=0 pg-cache-fail=0 pg-cache-eio=0
Writes               : Bandwidth(MiB/s)=0 Avg-Latency(usec)=0
Mmap : n=183 ok=183 err=0 munmap=183
Bar1-map             : n=183 ok=183 err=0 free=165 callbacks=18
                   active=0
Error                : cpu-gpu-pages=0 sg-ext=0 dma-map=0 dma-ref=0
Ops : Read=0 Write=0
GPU 0000:be:00.0  uuid:87e5c586-88ed-583b-df45-fcee0f1e7917 : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e7:00.0  uuid:029faa3b-cb0d-2718-259c-6dc650c636eb : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:5e:00.0  uuid:39eeb04b-1c52-81cc-d76e-53d03eb6ed32 : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:57:00.0  uuid:a99a7a93-7801-5711-258b-c6aca4fe6d85 : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:39:00.0  uuid:d22b0bc4-cdb1-65ac-7495-3570e5860fda : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:34:00.0  uuid:e11b33d9-60f7-a721-220a-d14e5b15a52c : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=128 cross_root_port(%)=0
GPU 0000:b7:00.0  uuid:e8630cd2-5cb7-cab7-ef2e-66c25507c119 : Registered_MiB=0
Cache_MiB=0
                   max_pinned_MiB=1 cross_root_port(%)=0
```

```
GPU 0000:e5:00.0  uuid:b3d46477-d54f-c23f-dc12-4eb5ea172af6 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e0:00.0  uuid:7a10c7bd-07e0-971b-a19c-61e7c185a82c : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:bc:00.0  uuid:bb96783c-5a46-233a-cbce-071aeb308083 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e2:00.0  uuid:b6565ee8-2100-7009-bcc6-a3809905620d : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=2 cross_root_port(%)=0
GPU 0000:5c:00.0  uuid:5527d7fb-a560-ab42-d027-20aeb5512197 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:59:00.0  uuid:bb734f6b-24ad-2f83-86c3-6ab179bce131 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:3b:00.0  uuid:0ef0b9ee-bb8f-cdae-4535-c0d790b2c663 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:b9:00.0  uuid:ad59f685-5836-c2ea-2c79-3c95bea23f0d : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:36:00.0  uuid:fda65234-707b-960a-d577-18c519301848 : Registered_MiB=0
Cache_MiB=0
    max_pinned_MiB=1 cross_root_port(%)=0
```

15.1. nvidia-fs Statistics

The following table describes `nvidia-fs` statistics.

Table 6. NVIDIA-FS Statistics

Type	Statistics	Description
Reads	n	Total number of read requests.
	ok	Total number of successful read requests.
	err	Total number of read errors.
	Readmb (mb)	Total data read into the GPUs.
	io_state_err	Read errors that were seen. Some pages might have been in the page cache.
Reads	Bandwidth (MB/s)	Active Read Bandwidth when IO is in flight. This is the period from when IO was submitted to the GDS kernel driver until the IO completion was

Type	Statistics	Description
		received by the GDS kernel driver. There was no userspace involved.
	Avg-Latency (usec)	Active Read latency when IO is in flight. This is from the period from when IO was submitted to the GDS kernel driver until the IO completion is received by the GDS kernel driver. There was no userspace involved.
Sparse Reads	n	Total number of sparse read requests.
	holes	Total number of holes that were observed during reads.
	pages	Total number of pages that span the holes.
Writes	n	Total number of write requests.
	ok	Total number of successful write requests.
	err	Total number of write errors.
	Writemb (mb)	Total data that was written from the GPUs to the disk.
	io_state_err	Write errors that were seen. Some pages might have been in the page cache.
	pg-cache	Total number of write requests that were found in the page cache.
	pg-cache-fail	Total number of write requests that were found in the page cache but could not be flushed.

Type	Statistics	Description
	pg-cache-eio	Total number of write requests that were found in the page-cache, but could not be flushed after multiple retries, and IO failed with EIO.
Writes	Bandwidth (MB/s)	Active Write Bandwidth when IO is in flight. This is the period from when IO is submitted to the GDS kernel driver until the IO completion is received by the GDS kernel driver. There was no userspace involved.
	Avg-Latency (usec)	Active Write latency when IO is in flight. This is the period from when IO is submitted to the GDS kernel driver until the IO completion is received by the GDS kernel driver. There was no userspace involved.
Mmap	n	Total number of mmap system calls that were issued.
	ok	Total number of successful mmap system calls.
	err	Errors that were observed through the mmap system call.
	munmap	Total number of munmap that were issued.
Bar-map	n	Total number of times the GPU BAR memory was pinned.
	ok	Total number of times the successful GPU BAR memory was pinned.
	err	Total errors that were observed during the BAR1 pinning.

Type	Statistics	Description
	<code>free</code>	Total number of times the BAR1 memory was unpinned.
	<code>callbacks</code>	Total number of times the NVIDIA kernel driver invoked callback to the GDS driver. This is invoked on the following instances: <ul style="list-style-type: none"> ▶ When the process crashes or was abruptly killed. ▶ When <code>cudaFree</code> is invoked on memory, which is pinned through <code>cuFileBufRegister</code>, but <code>cuFileBufDeregister</code> is not invoked.
	<code>active</code>	Active number of BAR1 memory that was pinned. (This value is the total number and not the total memory.)
Error	<code>cpu-gpu-pages</code>	Number of IO requests that had a mix of CPU-GPU pages when <code>nvfs_dma_map_sg_attrs</code> is invoked.
	<code>sg-ext</code>	Scatterlist that could not be expanded because the number of GPU pages is greater than <code>blk_nq_nr_phys_segments</code> .
	<code>dma-map</code>	A DMA map error.
ops	Read	Total number of Active Read IO in flight.
	Write	Total number of Active Write IO in flight.

15.2. Analyze Statistics for each GPU

You can analyze the statistics for each GPU to better understand what is happening in that GPU.

Consider the following example output:

```
GPU 0000:5e:00:0   uuid:dc87fe99-4d68-247b-b5d2-63f96d2adab1 : pinned_MB=0 cache_MB=0
max_pinned_MB=79
GPU 0000:b7:00:0   uuid:b3a6a195-d08c-09d1-bf8f-a5423c277c04 : pinned_MB=0 cache_MB=0
max_pinned_MB=76
GPU 0000:e7:00:0   uuid:7c432aed-a612-5b18-76e7-402bb48f21db : pinned_MB=0 cache_MB=0
max_pinned_MB=80
GPU 0000:57:00:0   uuid:aa871613-ee53-9a0c-a546-851d1afe4140 : pinned_MB=0 cache_MB=0
max_pinned_MB=80
```

In this sample output, `0000:5e:00:0`, is the PCI BDF of the GPU with the `Dc87fe99-4d68-247b-b5d2-63f96d2adab1` UUID. This is the same UUID that can be used to observe `nvidia-smi` statistics for this GPU.

Here is some additional information about the statistics:

- ▶ `pinned-MB` shows the active GPU memory that is pinned by using `nvidia_p2p_get_pages` from the GDS driver in MB across all active processes.
- ▶ `cache_MB` shows the active GPU memory that is pinned by using `nvidia_p2p_get_pages`, but this memory is used as the internal cache by GDS across all active processes.
- ▶ `max_pinned_MB` shows the max GPU memory that is pinned by GDS at any point in time on this GPU across multiple processes.

This value indicates that the max BAR size and administrator can be used for system sizing purposes.

15.3. Resetting the nvidia-fs Statistics

To reset the `nvidia-fs` statistics, run the following commands:

```
$ sudo bash
$ echo 1 >/proc/driver/nvidia-fs/stats
```

15.4. Checking Peer Affinity Stats for a Kernel Filesystem and Storage Drivers

The following `proc` files contain information about peer affinity DMA statistics via `nvidia-fs` callbacks:

- ▶ `nvidia-fs/stats`
- ▶ `nvidia-fs/peer_affinity`

► `nvidia-fs/peer_distance`

To enable the statistics, run the following command:

```
$ sudo bash
$ echo 1 > /sys/module/nvidia_fs/parameters/peer_stats_enabled
```

To view consolidated statistics as a regular user, run the following command:

```
$ cat /proc/driver/nvidia-fs/stats
```

Sample output:

```
GDS Version: 1.0.0.71
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2:7:47)
Mellanox PeerDirect Supported: True
IO stats: Enabled, peer IO stats: Enabled
Logging level: info

Active Shadow-Buffer (MiB): 0
Active Process: 0
Reads                               : n=0 ok=0 err=0 readMiB=0 io_state_err=0
Reads                               : Bandwidth(MiB/s)=0 Avg-Latency(usec)=0
Sparse Reads                         : n=6 io=0 holes=0 pages=0
Writes                               : n=0 ok=0 err=0 writeMiB=0 io_state_err=0 pg-
cache=0 pg-cache-fail=0 pg-cache-eio=0
Writes                               : Bandwidth(MiB/s)=0 Avg-Latency(usec)=0
Mmap                                 : n=183 ok=183 err=0 munmap=183
Bar1-map                             : n=183 ok=183 err=0 free=165 callbacks=18 active=0
Error                                : cpu-gpu-pages=0 sg-ext=0 dma-map=0 dma-ref=0
Ops                                  : Read=0 Write=0
GPU 0000:be:00.0   uuid:87e5c586-88ed-583b-df45-fcee0fle7917 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e7:00.0   uuid:029faa3b-cb0d-2718-259c-6dc650c636eb : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:5e:00.0   uuid:39eeb04b-1c52-81cc-d76e-53d03eb6ed32 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:57:00.0   uuid:a99a7a93-7801-5711-258b-c6aca4fe6d85 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:39:00.0   uuid:d22b0bc4-cdb1-65ac-7495-3570e5860fda : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:34:00.0   uuid:e11b33d9-60f7-a721-220a-d14e5b15a52c : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=128 cross_root_port(%)=0
GPU 0000:b7:00.0   uuid:e8630cd2-5cb7-cab7-ef2e-66c25507c119 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e5:00.0   uuid:b3d46477-d54f-c23f-dc12-4eb5ea172af6 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e0:00.0   uuid:7a10c7bd-07e0-971b-a19c-61e7c185a82c : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:bc:00.0   uuid:bb96783c-5a46-233a-cbce-071aeb308083 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:e2:00.0   uuid:b6565ee8-2100-7009-bcc6-a3809905620d : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=2 cross_root_port(%)=0
GPU 0000:5c:00.0   uuid:5527d7fb-a560-ab42-d027-20aeb5512197 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:59:00.0   uuid:bb734f6b-24ad-2f83-86c3-6ab179bce131 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:3b:00.0   uuid:0ef0b9ee-bb8f-cdae-4535-c0d790b2c663 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:b9:00.0   uuid:ad59f685-5836-c2ea-2c79-3c95bea23f0d : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
GPU 0000:36:00.0   uuid:fd65234-707b-960a-d577-18c519301848 : Registered_MiB=0
Cache_MiB=0 max_pinned_MiB=1 cross_root_port(%)=0
```

The `cross_root_port (%)` port is the percentage of total DMA traffic through `nvidia-fs` callbacks, and this value spans across PCIe root ports between GPU and its peers such as HCA.

For example, the row with GPU 0000:34:00.0 has 2621440 IO operations through the peer with rank 3. Non-zero values in the last column indicate that the IO is routed through the root complex.

Here are some examples:

Run the following command:

```
$ /usr/local/cuda-x.y/gds/samples /mnt/lustre/test 0
$ cat /proc/driver/nvidia-fs/stats
```

Here is the output:

```
GDS Version: 1.0.0.71
NVFS statistics(ver: 4.0)
NVFS Driver(version: 2:7:47)
Mellanox PeerDirect Supported: True
IO stats: Enabled, peer IO stats: Enabled
Logging level: info

Active Shadow-Buffer (MB): 0
Active Process: 0
Reads                : n=0 ok=0 err=0 readmb=0 io_state_err=0
Reads                : Bandwidth(MB/s)=0 Avg-Latency(usec)=0
Sparse Reads        : n=0 io=0 holes=0 pages=0
Writes              : n=1 ok=1 err=0 writemb=0 io_state_err=0 pg-cache=0
  pg-cache-fail=0
  pg-cache-eio=0
Writes              : Bandwidth(MB/s)=0 Avg-Latency(usec)=0
Mmap                 : n=1 ok=1 err=0 munmap=1
Bar1-map             : n=1 ok=1 err=0 free=1 callbacks=0 active=0
Error                : cpu-gpu-pages=0 sg-ext=0 dma-map=0
Ops                  : Read=0 Write=0
GPU 0000:34:00:0    uuid:98bb4b5c-4576-b996-3d84-4a5d778fa970 : pinned_MB=0 cache_MB=0
  max_pinned_MB=0 cross_root_port(%)=100
```

Run the following command:

```
$ cat /proc/driver/nvidia-fs/peer_affinity
```

Here is the output:

```
GPU P2P DMA distribution based on pci-distance
(last column indicates p2p via root complex)

GPU :0000:b7:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:b9:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:bc:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:be:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e0:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e2:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e5:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e7:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:34:00:0 :0 0 0 0 0 0 0 0 0 0 0 2
GPU :0000:36:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:39:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:3b:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:57:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:59:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5c:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5e:00:0 :0 0 0 0 0 0 0 0 0 0 0 0
```

In the above example, there are DMA transactions between the GPU (34:00.0) and one of its peers. The peer device has the highest possible rank which indicates it is farthest away from the respective GPU pci-distance wise.

To check the percentage of traffic, check the `cross_root_port %` in `/proc/driver/nvidia-fs/stats`. In the third example above, this value is 100%, which means that the peer-to-peer-traffic is happening over QPI links.

15.6. Display the GPU-to-Peer Distance Table

The `peer_distance` table displays the device-wise IO distribution for each peer with its rank for the specified GPU, and it complements the rank-based stats.

The `peer_distance` table displays the device-wise IO distribution for each peer with its rank for the specified GPU. It complements the rank-based stats.

The ranking is done in the following order:

1. Primary priority given to p2p distance (upper 2 bytes).
2. Secondary priority is given to the device bandwidth (lower 2 bytes)

For peer paths that cross the root port, a fixed cost for p2p distance (127) is added. This is done to induce a preference for paths under one CPU root port relative to paths that cross the CPU root ports.

Issue the following command:

```
$ cat /proc/driver/nvidia-fs/peer_distance
```

Sample output:

gpu	class	peer	peerrank	p2pdist	np2p	link	gen
0000:af:00.0	network	0000:86:00.0	0x820088	0x82	0	0x8	0x3
0000:af:00.0	network	0000:18:00.0	0x820088	0x82	0	0x8	0x3
0000:af:00.0	nvme	0000:86:00.1	0x820088	0x82	0	0x8	0x3
0000:af:00.0	network	0000:19:00.1	0x820088	0x82	0	0x8	0x3
0000:af:00.0	network	0000:87:00.0	0x820088	0x82	0	0x8	0x3
0000:af:00.0	nvme	0000:19:00.0	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	network	0000:86:00.0	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	network	0000:18:00.0	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	nvme	0000:86:00.1	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	network	0000:19:00.1	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	network	0000:87:00.0	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	nvme	0000:19:00.0	0x820088	0x82	0	0x8	0x3
0000:3b:00.0	network	0000:86:00.0	0x820088	0x82	0	0x8	0x3
0000:5e:00.0	network	0000:18:00.0	0x820088	0x82	0	0x8	0x3
0000:5e:00.0	nvme	0000:86:00.1	0x820088	0x82	0	0x8	0x3
0000:5e:00.0	network						

0000:5e:00.0 network	0000:19:00.1	0x820088	0x82	0	0x8	0x3
0000:5e:00.0 nvme	0000:87:00.0	0x820088	0x82	0	0x8	0x3
0000:5e:00.0 network	0000:19:00.0	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 network	0000:86:00.0	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 nvme	0000:18:00.0	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 network	0000:86:00.1	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 network	0000:19:00.1	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 nvme	0000:87:00.0	0x820088	0x82	0	0x8	0x3
0000:d8:00.0 network	0000:19:00.0	0x820088	0x82	0	0x8	0x3

15.7. The GDSIO Tool

GDSIO is a synthetic IO benchmarking tool that uses `cufile` APIs for IO. The tool can be found in the `/usr/local/cuda-x.y/tools` directory. For more information about how to use this tool, run `$ /usr/local/cuda-x.y/tools/gdsio -h` or review the `gdsio` section in the `/usr/local/cuda-x.y/tools/README` file. In the examples below, the files are created on an `ext4` file system.

Issue the following command:

```
# ./gdsio -f /root/sg/test -d 0 -w 4 -s 1M -x 0 -i 4K:32K:1K -I 1
```

Sample output:

```
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 671/1024(KiB)
IOSize: 4-32-1(KiB) Throughput: 0.044269 GiB/sec, Avg_Latency:
996.094925 usecs ops: 60 total_time 0.014455 secs
```

This command does a write IO (`-I 1`) on a file named `test` of size 1MiB (`-s 1M`) with an IO size that varies between 4KiB to 32 KiB in steps of 1KiB (`-i 4K:32K:1K`). The transfer is performed using GDS (`-x 0`) using 4 threads (`-w 4`) on GPU 0 (`-d 0`).

Some additional features of the tool are:

- Support for read/write at random offsets in a file.

The `gdsio` tool provides options to perform a read and write to a file at random offsets.

- Using `-I 2` and `-I 3` options does a file read and write operation at random offset respectively but the random offsets are always 4KiB aligned.

```
# ./gdsio -f /root/sg/test -d 0 -w 4 -s 1M -x 0 -i 4K:32K:1K -I 3
IoType: RANDWRITE XferType: GPUD Threads: 4 DataSetSize: 706/1024(KiB) IOSize:
4-32-1(KiB) Throughput: 0.079718 GiB/sec, Avg_Latency: 590.853274 usecs ops:
44 total_time 0.008446 secs
```

- To perform a random read and write at unaligned 4KiB offsets, the `-U` option can be used with `-I 0` or `-I 1` for read and write, respectively.

```
# ./gdsio -f /root/sg/test -d 0 -w 4 -s 1M -x 0 -i 4K:32K:1K -I 1 -U
```

```
IoType: RANDWRITE XferType: GPUD Threads: 4 DataSetSize: 825/1024(KiB) IOSize:
4-32-1(KiB) Throughput: 0.055666 GiB/sec, Avg_Latency: 919.112500 usecs ops:
49 total_time 0.014134 secs
```

- ▶ Random buffer fill for dedupe and compression.

Using the '-R' option fills the io size buffer (-i) with random data. This random data is then written to the file onto different file offsets.

```
# ./gdsio -f /root/sg/test -d 0 -w 4 -s 1M -x 0 -i 4K:32K:1K -I 1 -R
```

```
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 841/1024(KiB) IOSize:
4-32-1(KiB) Throughput: 0.059126 GiB/sec, Avg_Latency: 788.884580 usecs ops:
69 total_time 0.013565 secs
```

- ▶ Using the -F option will fill the entire file with random data.

```
# ./gdsio -f /root/sg/test -d 0 -w 4 -s 1M -x 0 -i 4K:32K:1K -I 1 -F
```

```
IoType: WRITE XferType: GPUD Threads: 4 DataSetSize: 922/1024(KiB) IOSize:
4-32-1(KiB) Throughput: 0.024376 GiB/sec, Avg_Latency: 1321.104532 usecs ops:
73 total_time 0.036072 secs
```

This is useful for file systems that use dedupe and compression algorithms to minimize disk access. Using random data increases the probability that these file systems will hit the backend disk more often.

- ▶ Variable block size.

To perform a read or a write on a file, you can specify the block size (-i), which says that IO would be performed in chunks of block sized lengths. To check the stats for what block sizes are used use the gds_stats tool. Ensure the the /etc/cufile.json file has cufile_stats is set to 3:

```
# ./gds_stats -p <pid of the gdsio process> -l 3
```

Sample output:

```
0-4 (KiB): 0 0
4-8 (KiB): 0 17205
8-16 (KiB): 0 45859
16-32 (KiB): 0 40125
32-64 (KiB): 0 0
64-128 (KiB): 0 0
128-256 (KiB): 0 0
256-512 (KiB): 0 0
512-1024 (KiB): 0 0
1024-2048 (KiB): 0 0
2048-4096 (KiB): 0 0
4096-8192 (KiB): 0 0
8192-16384 (KiB): 0 0
16384-32768 (KiB): 0 0
32768-65536 (KiB): 0 0
65536-... (KiB): 0 0
```

The highlighted counters show that, for the command above, the block sizes that are used for file IO are in the 4-32 KiB range.

- ▶ Verification mode usage and limitations.

To ensure data integrity, there is an option to perform IO in a Write and Read in verify mode using the -v option. Here is an example:

```
# ./gdsio -V -f /root/sg/test -d 0 -w 1 -s 2G -o 0 -x 0 -k 0 -i 4K:32K:1K -I 1
IoType: WRITE XferType: GPUD Threads: 1 DataSetSize: 2097144/2097152(KiB) IOSize:
4-32-1(KiB) Throughput: 0.074048 GiB/sec, Avg_Latency: 231.812570 usecs ops:
116513 total_time 27.009349 secs
Verifying data
```

```
IoType: READ XferType: GPUD Threads: 1 DataSetSize: 2097144/2097152(KiB) IOSize:
4-32-1(KiB) Throughput: 0.103465 GiB/sec, Avg_Latency: 165.900663 usecs ops:
116513 total_time 19.330184 secs
```

The command mentioned above will perform a write followed by a read verify test.

While using the verify mode, remember the following points:

- ▶ read test (-l 0) with verify option (-v) should be used with files written (-l 1) with the -v option
- ▶ read test (-l 2) with verify option (-v) should be used with files written (-l 3) with the -v option and using same random seed (-k) using same number of threads, offset, and data size
- ▶ write test (-l 1/3) with verify option (-v) will perform writes followed by read.
- ▶ Verify mode cannot be used in timed mode (-T option).
If Verify mode is used in a timed mode, it will be ignored.
- ▶ The configuration file

GDSIO has an option to configure the parameters that are needed to perform an IO in a configuration file and run the IO using those configurations. The configuration file gives the option of performing multiple jobs, where each job has some different configurations.

The configuration file has global parameters and job specific parameter support. For example, with a configuration file, you can configure each job to perform on a GPU and with a different number of threads. The global parameters, such as IO Size and transfer mode, remain the same for each job. For more information, refer to /usr/local/cuda-x.y/tools/README and /usr/local/cuda-x.y/tools/rw-sample.gdsio files. After configuring the parameters, to perform the IO operation using the configuration file, run the following command:

```
# ./gdsio <config file name>
```

See [Tabulated Fields](#) for a list of the tabulated fields.

15.8. Tabulated Fields

The following table describes the tabulated fields in the output of the #./gdsio <config file name> command.

Table 7. Tabulated Fields

Global Option	Description
xfer_type	GDSIO Transfer types: <ul style="list-style-type: none"> ▶ 0 : Storage->GPU ▶ 1 : Storage->CPU ▶ 2 : Storage->CPU->GPU ▶ 3 : Storage->CPU->GPU_ASYNC

Global Option	Description
	<ul style="list-style-type: none"> ▶ 4 : Storage->PAGE_CACHE->CPU->GPU ▶ 5 : Storage->GPU_ASYNC_STREAM ▶ 6: Storage->GPU_BATCH ▶ 7: Storage->GPU_BATCH_STREAM
rw	IO type, rw=read, rw=write, rw=randread, rw=randwrite
bs	block size, for example, bs=1M, for variable block size can specify range, for example, bs=1M:4M:1M, (1M : start block size, 4M : end block size, 1M :steps in which size is varied).
size	File-size, for example, size=2G.
runtime	Duration in seconds.
do_verify	Use 1 for enabling verification
skip_bufregister	Skip cufile buffer registration, ignored in cpu mode.
enable_nvlinks	Set up NVlinks. This field is recommended if p2p traffic is cross node.
random_seed	Use random seed, for example, 1234.
refill_buffer	Refill io buffer after every write.
fill_random	Fill request buffer with random data.
unaligned_random	Use random offsets which are not page-aligned.
start_offset	File offset to start read/write from.
Per-Job Options	Description
numa_node	NUMA node.
gpu_dev_id	GPU device index (check nvidia-smi).
num_threads	Number of IO Threads per job.
directory	Directory name where files are present. Each thread will work on a per file basis.
filename	Filename for single file mode, where threads share the same file. (Note: directory mode and filemode should not be used in a mixed manner across jobs).

Global Option	Description
mem_type	Memory types to be used. Supported values: 0 - (cudaMalloc), 1 - (cuMem), 2 - (cudaMallocHost) 3 - malloc 4 - mmap.
fd_type	File Descriptor mode. 0 - O_DIRECT (default) 1 - non-O_DIRECT

15.9. The GDSCHECK Tool

The `/usr/local/cuda-x.y/tools/gdscheck.py` tool is used to perform a GDS platform check and has other options that can be found by using `-h` option.

```
$ ./gdscheck.py -h
usage: gdscheck.py [-h] [-p] [-f FILE] [-v] [-V]
GPUDirectStorage platform checker
optional arguments:
  -h, --help  show this help message and exit
  -p          gds platform check
  -f FILE     gds file check
  -v          gds version checks
  -V          gds fs checks
```

To perform a GDS platform check, issue the following command and expect the output in the following format:

```
# ./gdscheck.py -p
GDS release version: 1.0.0.78
nvidia_fs version: 2.7 libcufile version: 2.4
=====
ENVIRONMENT:
=====
=====
DRIVER CONFIGURATION:
=====
NVMe                : Supported
NVMeOF              : Unsupported
SCSI                 : Unsupported
ScaleFlux CSD       : Unsupported
NVMesh              : Unsupported
DDN EXAScaler       : Supported
IBM Spectrum Scale  : Unsupported
NFS                  : Unsupported
WekaFS               : Unsupported
Userspace RDMA      : Unsupported
--Mellanox PeerDirect : Enabled
--rdma library       : Not Loaded (libcufile_rdma.so)
--rdma devices       : Not configured
--rdma_device_status : Up: 0 Down: 0
=====
CUFILE CONFIGURATION:
=====
properties.use_compat_mode : true
properties.gds_rdma_write_support : true
properties.use_poll_mode : false
properties.poll_mode_max_size_kb : 4
properties.max_batch_io_timeout_msecs : 5
properties.max_direct_io_size_kb : 16384
```



```

properties.max_device_cache_size_kb : 131072
properties.max_device_pinned_mem_size_kb : 33554432
properties.posix_pool_slab_size_kb : 4 1024 16384
properties.posix_pool_slab_count : 128 64 32
properties.rdma_peer_affinity_policy : RoundRobin
properties.rdma_dynamic_routing : 0
fs.generic.posix_unaligned_writes : false
fs.lustre.posix_gds_min_kb: 0
fs.weka.rdma_write_support: false
profile.nvtx : false
profile.cufile_stats : 0
miscellaneous.api_check_aggressive : false
=====
GPU INFO:
=====
GPU index 0 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 1 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 2 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 3 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 4 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 5 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 6 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 7 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 8 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 9 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 10 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 11 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 12 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 13 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 14 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
GPU index 15 Tesla V100-SXM3-32GB bar:1 bar size (MiB):32768 supports GDS
=====
PLATFORM INFO:
=====
IOMMU: disabled
Platform verification succeeded

```

15.10. NFS Support with GPUDirect Storage

This section provides information about NFS support with GDS.

15.10.1. Install Linux NFS server with RDMA Support on MLNX_OFED 5.3 or Later

To install a standard Linux kernel-based NFS server with RDMA support, complete the following steps:



Note: The server must have a Mellanox connect-X4/5 NIC with MLNX_OFED 5.3 or later installed.

1. Issue the following command:

```
$ ofed_info -s MLNX_OFED_LINUX-5.3-1.0.5.1:
```

2. Review the output to ensure that the server was installed.

```
$ sudo apt-get install nfs-kernel-server
```

```
$ mkfs.ext4 /dev/nvme0n1
$ mount -o data=ordered /dev/nvme0n1 /mnt/nvme
$ cat /etc/exports
/mnt/nvme *(rw,async,insecure,no_root_squash,no_subtree_check)
$ service nfs-kernel-server restart
$ modprobe rprdma
$ echo rdma 20049 > /proc/fs/nfsd/portlist
```

15.10.2. Install GPUDirect Storage Support for the NFS Client

To install a NFS client with GDS support complete the following steps:



Note: The client must have a Mellanox connect-X4/5 NIC with MLNX_OFED 5.3 or later installed.

1. Issue the following command:

```
$ ofed_info -s MLNX_OFED_LINUX-5.3-1.0.5.0:
```

2. Review the output to ensure that the support exists.

```
$ sudo apt-get install nfs-common
$ modprobe rprdma
$ mkdir -p /mnt/nfs_rdma_gds
$ sudo mount -v -o proto=rdma,port=20049,vers=3 172.16.0.101:/ /mnt/nfs_rdma_gds
To mount with nconnect using VAST nfs client package:
Eg: client IB interfaces 172.16.0.17 , 172.16.0.18, 172.16.0.19, 172.16.0.20,
172.16.0.21,172.16.0.22,172.16.0.23 172.16.0.24
$ sudo mount -v -o
proto=rdma,port=20049,vers=3,nconnect=20,localports=172.16.0.17-172.16.0.24,remoteports=172.16.0.17-172.16.0.24 172.16.0.101:/ /mnt/nfs_rdma_gds
```

15.11. NFS GPUDirect Storage Statistics and Debugging

NFS IO can be observed using regular Linux tools that are used for monitoring IO, such as `iotop` and `nfsstat`.

- ▶ To enable NFS RPC stats debugging, run the following command.

```
$ rpcdebug -v
```

- ▶ To observe GDS-related IO stats, run the following command.

```
$ cat /proc/driver/nvidia-fs/stats
```

- ▶ To determine GDS statistics per process, run the following command.

```
$ /usr/local/cuda-x.y/tools/gds_stats -p <PID> -l 3
```

15.12. GPUDirect Storage IO Behavior

This section provides information about IO behavior in GDS.

15.12.1. Read/Write Atomicity Consistency with GPUDirect Storage Direct IO

In GDS, the `max_direct_io_size_kb` property controls the IO unit size in which the limitation is issued to the underlying file system. By default, this value is 16MB. This implies that from a Linux VFS perspective, the atomicity of size is limited to the `max_direct_io_size_kb` size and not the original request size. This limitation exists in the standard GDS path and in compatible mode.

15.12.2. Write with File a Opened in O_APPEND Mode (cuFileWrite)

For a file that is opened in O_APPEND mode with concurrent writers, if the IO size that is used is larger than the `max_direct_io_size_kb` property, because of the write atomicity limitations, the file might have interleaved data from multiple writers. This cannot be prevented even if the underlying file-system has locking guarantees.

15.12.3. GPU to NIC Peer Affinity

The library maintains a peer affinity table that is a pci-distance-based ranking for a GPU and the available NICs in the platform for RDMA. Currently, the limitation in the ranking does not consider NUMA attributes for the NICs. For a NIC that does not share a common root port with a GPU, the P2P traffic might get routed cross socket over QPI links even if there is a NIC that resides on the same CPU socket as the GPU.

15.12.4. Compatible Mode with Unregistered Buffers

Currently, in compatible mode, the IO path with non-registered buffers does not have optimal performance and does buffer allocation and deallocation in every `cuFileRead` or `cuFileWrite`.

15.12.5. Unaligned writes with Non-Registered Buffers

For unaligned writes, using unregistered buffers performance may not be optimal as compared to registered buffers.

15.12.6. Process Hang with NFS

A process hang is observed in NFS environments when the application crashes.

15.12.7. Tools Support Limitations for CUDA 9 and Earlier

The gdsio binary has been built against CUDA runtime 10.1 and has a dependency on the CUDA runtime environment to be equal to version 10.1 or later. Otherwise, a driver dependency error will be reported by the tool.

15.13. GDS Statistics for Dynamic Routing

Dynamic Routing decisions are performed at I/O operation granularity. The GDS User-space Statistics contain a per-GPU counter to indicate the number of I/Os that have been routed using Dynamic Routing.

Table 8. cuFile Dynamic Routing Counter

Entry	Description
dr	Number of cuFileRead/cuFileWrite for which I/O was routed using Dynamic Routing for a given GPU.

There are existing counters in the `PER_GPU POOL BUFFER STATS` and `PER_GPU POSIX POOL BUFFER STATS` from which a user can infer the GPUs that are chosen by dynamic routing for use as the bounce buffers.

a) Platform has GPUs (0 and 1) not sharing the same PCIe host bridge as the NICs:

```
"rdma_dev_addr_list": [ "192.168.0.12", "192.168.1.12" ],
"rdma_dynamic_routing": true,
"rdma_dynamic_routing_order": [ "GPU_MEM_NVLINKS", "GPU_MEM", "SYS_MEM" ]

$ gds_stats -p <process id> -l 3
GPU 0 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 r_sparse=0 r_inline=0 err=0
MiB=0 Write: bw=3.37598 util(%)=532 n=6629 posix=0 unalign=0 dr=6629 err=0 MiB=6629
BufRegister: n=4 err=0 free=0 MiB=4
GPU 1 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 r_sparse=0 r_inline=0 err=0
MiB=0 Write: bw=3.29297 util(%)=523 n=6637 posix=0 unalign=0 dr=6637 err=0 MiB=6637
BufRegister: n=4 err=0 free=0 MiB=4

PER_GPU POOL BUFFER STATS:
GPU : 6 pool_size_MiB : 7 usage : 1/7 used_MiB : 1
GPU : 7 pool_size_MiB : 7 usage : 0/7 used_MiB : 0
GPU : 8 pool_size_MiB : 7 usage : 2/7 used_MiB : 2
GPU : 9 pool_size_MiB : 7 usage : 2/7 used_MiB : 2

PER_GPU POSIX POOL BUFFER STATS:

PER_GPU RDMA STATS:
GPU 0000:34:00.0 : mlx5_3(138:48):0 mlx5_6(265:48):0
GPU 0000:36:00.0 : mlx5_3(138:48):0 mlx5_6(265:48):0
GPU 0000:39:00.0 : mlx5_3(138:48):0 mlx5_6(265:48):0
GPU 0000:3b:00.0 : mlx5_3(138:48):0 mlx5_6(265:48):0
GPU 0000:57:00.0 : mlx5_3(7:48):0 mlx5_6(265:48):0
GPU 0000:59:00.0 : mlx5_3(7:48):0 mlx5_6(265:48):0
GPU 0000:5c:00.0 : mlx5_3(3:48):3318 mlx5_6(265:48):0
GPU 0000:5e:00.0 : mlx5_3(3:48):3318 mlx5_6(265:48):0
```

```
GPU 0000:b7:00.0 : mlx5_6(3:48):3316 mlx5_3(265:48):0
GPU 0000:b9:00.0 : mlx5_6(3:48):3317 mlx5_3(265:48):0
GPU 0000:bc:00.0 : mlx5_6(7:48):0 mlx5_3(265:48):0
GPU 0000:be:00.0 : mlx5_6(7:48):0 mlx5_3(265:48):0
GPU 0000:e0:00.0 : mlx5_6(138:48):0 mlx5_3(265:48):0
GPU 0000:e2:00.0 : mlx5_6(138:48):0 mlx5_3(265:48):0
GPU 0000:e5:00.0 : mlx5_6(138:48):0 mlx5_3(265:48):0
GPU 0000:e7:00.0 : mlx5_6(138:48):0 mlx5_3(265:48):0
```

b) Platform configuration that has no GPUs sharing the same PCIe host bridge as the NICs and no NVLinks between the GPUs. For such configurations, an admin can set a policy to use system memory other than the default P2P policy.

```
"rdma_dev_addr_list": [ "192.168.0.12", "192.168.1.12" ],
"rdma_dynamic_routing": true,
"rdma_dynamic_routing_order": [ "SYS_MEM" ]
PER_GPU STATS:
GPU_4 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 r_sparse=0 r_inline=0 err=0 MiB=0
Write: bw=1.11GiB util(%)=0 n=1023 posix=1023 unalign=1023 dr=1023 err=0 MiB=1023
BufRegister: n=0 err=0 free=0 MiB=0
GPU_8 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 r_sparse=0 r_inline=0 err=0 MiB=0
Write: bw=1.11GiB util(%)=0 n=1023 posix=1023 unalign=1023 dr=1023 err=0 MiB=1023
BufRegister: n=0 err=0 free=0 MiB=0
PER_GPU POSIX POOL BUFFER STATS:
GPU_4 4(KiB) :0/0 1024(KiB) :0/1 16384(KiB) :0/0
GPU_8 4(KiB) :0/0 1024(KiB) :1/1 16384(KiB) :0/0
```

15.13.1. Peer Affinity Dynamic Routing

Dynamic Routing decisions are performed at I/O operation granularity. The GDS User-space Statistics contain a per-GPU counter to indicate the number of I/Os that have been routed using Dynamic Routing.

Table 9. cuFile Dynamic Routing Counter

Entry	Description
dr	Number of cuFileRead/cuFileWrite for which I/O was routed using Dynamic Routing for a given GPU.

There are existing counters in the PER_GPU POOL BUFFER STATS and PER_GPU POSIX POOL BUFFER STATS from which a user can infer the GPUs that are chosen by dynamic routing for use as the bounce buffers.

```
// "rdma_dev_addr_list": [ "192.168.4.12", "192.168.5.12", "192.168.6.12",
"192.168.7.12" ],
cufile.log:
-----
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:133 Computing
GPU->NIC affinity table:
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:34:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:36:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:39:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:3b:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:57:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
```

```

23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:59:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:5c:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:5e:00.0 RDMA dev: mlx5_6 mlx5_8 mlx5_7 mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:b7:00.0 RDMA dev: mlx5_6
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:b9:00.0 RDMA dev: mlx5_6
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:bc:00.0 RDMA dev: mlx5_7
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:be:00.0 RDMA dev: mlx5_7
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:e0:00.0 RDMA dev: mlx5_8
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:e2:00.0 RDMA dev: mlx5_8
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:e5:00.0 RDMA dev: mlx5_9
23-02-2021 10:17:49:641 [pid=22436 tid=22436] INFO curdma-ldbal:139 GPU:
0000:e7:00.0 RDMA dev: mlx5_9

```

A sample from `gds_stats` showing the GPU to NIC binding during a sample IO test:

```

PER_GPU_RDMA_STATS:
GPU 0000:34:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:36:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:39:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:3b:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:57:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:59:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:5c:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:5e:00.0 : mlx5_6(265:48):0 mlx5_8(265:48):0 mlx5_7(265:48):0
mlx5_9(265:48):0
GPU 0000:b7:00.0 : mlx5_6(3:48):22918 mlx5_7(7:48):0 mlx5_8(138:48):0
mlx5_9(138:48):0
GPU 0000:b9:00.0 : mlx5_6(3:48):22949 mlx5_7(7:48):0 mlx5_8(138:48):0
mlx5_9(138:48):0
GPU 0000:bc:00.0 : mlx5_7(3:48):22945 mlx5_6(7:48):0 mlx5_8(138:48):0
mlx5_9(138:48):0
GPU 0000:be:00.0 : mlx5_7(3:48):22942 mlx5_6(7:48):0 mlx5_8(138:48):0
mlx5_9(138:48):0
GPU 0000:e0:00.0 : mlx5_8(3:48):22937 mlx5_9(7:48):0 mlx5_6(138:48):0
mlx5_7(138:48):0
GPU 0000:e2:00.0 : mlx5_8(3:48):22930 mlx5_9(7:48):0 mlx5_6(138:48):0
mlx5_7(138:48):0
GPU 0000:e5:00.0 : mlx5_9(3:48):22922 mlx5_8(7:48):0 mlx5_6(138:48):0
mlx5_7(138:48):0
GPU 0000:e7:00.0 : mlx5_9(3:48):22920 mlx5_8(7:48):0 mlx5_6(138:48):0
mlx5_7(138:48):0

```

- For kernel-based DFS, DDN-Lustre and VAST-NFS, `nvidia-fs` driver provides a callback to determine the best NIC given a target GPU. The `nvidia-fs` `peer_affinity` can be used to track end-to-end IO affinity behavior.

For example, with a routing policy of “GPU_MEM_NVLINK”, one should not see cross-port traffic as shown in the statistics snippet below:

```
$ cat /proc/driver/nvidia-fs/peer_affinity
GPU P2P DMA distribution based on pci-distance

(last column indicates p2p via root complex)
GPU :0000:bc:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e0:00.0 :0 0 205305577 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0
GPU :0000:e5:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:57:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:59:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:be:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:34:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:b7:00.0 :0 0 205279892 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0
GPU :0000:36:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:3b:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:39:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:b9:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5c:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e2:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5e:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

With routing policy of P2P, one can expect to see cross-port traffic as shown in the following statistics snippet:

```
dgxuser@e155j-dgx2-c6-u04:~/ssens$ cat /proc/driver/nvidia-fs/peer_affinity
GPU P2P DMA distribution based on pci-distance

(last column indicates p2p via root complex)
GPU :0000:bc:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e0:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e5:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:57:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:59:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:be:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:34:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 9186359
GPU :0000:e7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:b7:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:36:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 9191164
GPU :0000:3b:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 9194318
GPU :0000:39:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 9188836
GPU :0000:b9:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5c:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:e2:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GPU :0000:5e:00.0 :0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

15.13.2. cuFile Log Related to Dynamic Routing

The following log shows the routing table with possible GPUS to be used for IP addresses:

```
/"rdma_dev_addr_list": [ "192.168.0.12", "192.168.1.12", "192.168.2.12",
"192.168.3.12", "192.168.4.12", "192.168.5.12", "192.168.6.12", "192.168.7.12" ],
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:141 Computing NIC->
GPU affinity table for rdma devices available in config:
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib3
bdf:0000:5d:00.0 ip: 192.168.3.12 best gpus: 6 7 4 5
```

```
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib9
bdf:0000:e6:00.0 ip: 192.168.5.12 best gpus: 14 15 12 13
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib2
bdf:0000:58:00.0 ip: 192.168.2.12 best gpus: 4 5 6 7
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib6
bdf:0000:b8:00.0 ip: 192.168.6.12 best gpus: 8 9 10 11
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib1
bdf:0000:3a:00.0 ip: 192.168.1.12 best gpus: 3 2 0 1
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib8
bdf:0000:e1:00.0 ip: 192.168.4.12 best gpus: 12 13 14 15
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib0
bdf:0000:35:00.0 ip: 192.168.0.12 best gpus: 0 1 3 2
22-02-2021 21:16:27:776 [pid=90794 tid=90794] INFO    cufio-route:156 netdev:ib7
bdf:0000:bd:00.0 ip: 192.168.7.12 best gpus: 10 11 8 9

22-02-2021 21:16:27:776 [pid=90794 tid=90794] DEBUG   cufio:1218 target gpu: 4 best
gpu: 4 selected based on dynamic routing
```

Chapter 16. GDS Library Tracing

The GDS Library has USDT (static tracepoints), which can be used with Linux tools such as `lttng`, `bcc/bpf`, `perf`. This section assumes familiarity with these tools.

The examples in this section show tracing by using the `bcc/bpf` tracing facility. GDS does not ship these tracing tools. Refer to [Installing BCC](#) for more information about installing `bcc/bpf` tools. Users must have root privileges to install.



Note: The user must also have `sudo` access to use these tools.

16.1. Example: Display Tracepoints

1. To display tracepoints, run the following command:

```
# ./tplist -l /usr/local/gds/lib/libcufile.so
```

2. Review the output, for example:

```
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_px_read  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_rdma_read  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_gds_read  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_gds_read_async  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_px_write  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_gds_write  
/usr/local/cuda-x.y/lib/libcufile.so cufio:cufio_gds_write_async  
/usr/local/cuda-x.y/lib/libcufile.so cufio-internal:cufio-internal-write-bb  
/usr/local/cuda-x.y/lib/libcufile.so cufio-internal:cufio-internal-read-bb  
/usr/local/cuda-x.y/lib/libcufile.so cufio-internal:cufio-internal-bb-done  
/usr/local/cuda-x.y/lib/libcufile.so cufio-internal:cufio-internal-io-done  
/usr/local/cuda-x.y/lib/libcufile.so cufio-internal:cufio-internal-map
```

16.1.1. Example: Tracepoint Arguments

Here are examples of tracepoint arguments.

`cufio_px_read`

This tracepoint tracks POSIX IO reads and takes the following arguments:

- ▶ Arg 1: File descriptor
- ▶ Arg 2: File offset
- ▶ Arg 3: Read size
- ▶ Arg 4: GPU Buffer offset

- ▶ Arg 5: Return value
- ▶ Arg 6: GPU ID for which IO is done

`cufile_rdma_read`

This tracepoint tracks IO reads for through WEKA filesystem and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Read size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufile_rdma_write`

This tracepoint tracks IO reads for through WEKA filesystem and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Write size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufile_gds_read`

This tracepoint tracks IO reads going through the GDS kernel drive and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Read size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufio_gds_read_async`

This tracepoint tracks iO reads going through the GDS kernel driver and poll mode is set and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Read size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufio_px_write`

This tracepoint tracks POSIX IO writes and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg 2: File offset
- ▶ Arg 3: Write size
- ▶ Arg 4: GPU Buffer offset
- ▶ Arg 5: Return value
- ▶ Arg 6: GPU ID for which IO is done

`cufio_gds_write`

This tracepoint tracks IO writes going through the GDS kernel driver and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Write size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufio_gds_unaligned_write`

This tracepoint tracks IO writes going through the GDS kernel driver if the IO was unaligned and takes the following arguments:

- ▶ Arg1: File descriptor

- ▶ Arg2: File offset
- ▶ Arg3: Write size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufile_gds_write_async`

This tracepoint tracks IO writes going through the GDS kernel driver, and poll mode is set and takes the following arguments:

- ▶ Arg1: File descriptor
- ▶ Arg2: File offset
- ▶ Arg3: Write size
- ▶ Arg4: GPU Buffer offset
- ▶ Arg5: Return value
- ▶ Arg6: GPU ID for which IO is done
- ▶ Arg7: Is the IO done to GPU Bounce buffer

`cufile-internal-write-bb`

This tracepoint tracks IO writes going through internal GPU Bounce buffers and is specific to the EXAScaler[®] filesystem and block device-based filesystems. This tracepoint is in hot IO-path tracking in every IO and takes the following arguments:

- ▶ Arg1: Application GPU (GPU ID)
- ▶ Arg2: GPU Bounce buffer (GPU ID)
- ▶ Arg3: File descriptor
- ▶ Arg4: File offset
- ▶ Arg5: Write size
- ▶ Arg6: Application GPU Buffer offset
- ▶ Arg7: Size in bytes transferred from application GPU buffer to target GPU bounce buffer.
- ▶ Arg8: Total Size in bytes transferred so far through bounce buffer.
- ▶ Arg9: Pending IO count in this transaction

`cufile-internal-read-bb`

This tracepoint tracks IO reads going through internal GPU Bounce buffers and is specific to the EXAScaler[®] filesystem and block device-based filesystems. This tracepoint is in hot IO-path tracking every IO and takes the following arguments:

- ▶ Arg1: Application GPU (GPU ID)
- ▶ Arg2: GPU bounce buffer (GPU ID)
- ▶ Arg3: File descriptor
- ▶ Arg4: File offset
- ▶ Arg5: Read size
- ▶ Arg6: Application GPU Buffer offset
- ▶ Arg7: Size in bytes transferred from the GPU bounce buffer to application GPU buffer.
- ▶ Arg8: Total Size in bytes transferred so far through bounce buffer.
- ▶ Arg9: Pending IO count in this transaction.

`cufile-internal-bb-done`

This tracepoint tracks all IO going through bounce buffers and is invoked when IO is completed through bounce buffers. The tracepoint can be used to track all IO going through bounce buffers and takes the following arguments:

- ▶ Arg1: IO-type READ - 0, WRITE - 1
- ▶ Arg2: Application GPU (GPU ID)
- ▶ Arg3: GPU Bounce buffer (GPU ID)
- ▶ Arg4: File descriptor
- ▶ Arg5: File offset
- ▶ Arg6: Read/Write size
- ▶ Arg7: GPU buffer offset
- ▶ Arg8: IO is unaligned (1 - True, 0 - False)
- ▶ Arg9: Buffer is registered (1 - True, 0 - False)

`cufile-internal-io-done`

This tracepoint tracks all IO going through the GDS kernel driver. This tracepoint is invoked when the IO is completed and takes the following arguments:

- ▶ Arg1: IO-type READ - 0, WRITE - 1
- ▶ Arg2: GPU ID for which IO is done
- ▶ Arg3: File descriptor
- ▶ Arg4: File offset
- ▶ Arg5: Total bytes transferred

`cufile-internal-map`

This tracepoint tracks GPU buffer registration using `cuFileBufRegister` and takes the following arguments:

- ▶ Arg1: GPU ID
- ▶ Arg2: GPU Buffer size for which registration is done
- ▶ Arg3: max_direct_io_size that was used for this buffer.
The shadow memory size is set in the /etc/cufile.json file.
- ▶ Arg4: boolean value indicating whether buffer is pinned.
- ▶ Arg5: boolean value indicating whether this buffer is a GPU bounce buffer.
- ▶ Arg6: GPU offset.

The data type of each argument in these tracepoints can be found by running the following command:

```
# ./tplist -l /usr/local/cuda-x.y/lib/libcufile.so -vvv | grep cufile_px_read -A 7
cufile:cufile_px_read [sema 0x0]
```

Here is the output:

```
# ./tplist -l /usr/local/cuda-x.y/lib/libcufile.so -vvv | grep cufile_px_read -A 7
cufile:cufile_px_read [sema 0x0]
  location #1 7usr/local/cuda-x.y/lib/libcufile.so 0x16437c
    argument #1 4 signed bytes @ dx
    argument #2 8 signed bytes @ cx
    argument #3 8 unsigned bytes @ si
    argument #4 8 signed bytes @ di
    argument #5 8 signed bytes @ r8
    argument #6 4 signed bytes @ ax
```

16.2. Example: Track the IO Activity of a Process that Issues cuFileRead/cuFileWrite

This example provides information about how you can track the IO activity of a process that issues the `cuFileRead` or the `cuFileWrite` API.

1. Run the following command.

```
# ./funccount u:/usr/local/cuda-x.y/lib/libcufile.so:cufile_* -i 1 -T -p 59467
Tracing 7 functions for "u:/usr/local/cuda-x.y/lib/libcufile.so:cufile_*"... Hit
Ctrl-C to end.
```

2. Review the output, for example:

```
cufile_gds_write          1891

16:21:13
FUNC                      COUNT
cufile_gds_write          1852

16:21:14
FUNC                      COUNT
cufile_gds_write          1865
^C
16:21:14
FUNC                      COUNT
cufile_gds_write          1138
Detaching...
```

16.3. Example: Display the IO Pattern of all the IOs that Go Through GDS

This example provides information about how you can display and understand the IO pattern of all IOs that go through GDS.

1. Run the following command:

```
# ./argdist -C 'u:/usr/local/cuda-x.y/lib/
libcufile.so:cufio_gds_read():size_t:arg3# Size Distribution'
```

2. Review the output, for example:

```
[16:38:22]
IO Size Distribution
COUNT      EVENT
4654         arg3 = 1048576
7480         arg3 = 131072
9029         arg3 = 65536
13561        arg3 = 8192
14200        arg3 = 4096
[16:38:23]
IO Size Distribution
COUNT      EVENT
4682         arg3 = 1048576
7459         arg3 = 131072
9049         arg3 = 65536
13556        arg3 = 8192
14085        arg3 = 4096
[16:38:24]
IO Size Distribution
COUNT      EVENT
4678         arg3 = 1048576
7416         arg3 = 131072
9018         arg3 = 65536
13536        arg3 = 8192
14082        arg3 = 4096
```

The 1M, 128K, 64K, 8K, and 4K IOs are all completing reads through GDS.

16.4. Understand the IO Pattern of a Process

You can review the output to understand the IO pattern of a process.

1. Run the following command.

```
# ./argdist -C 'u:/usr/local/cuda-x.y/lib/
libcufile.so:cufio_gds_read():size_t:arg3#IO
Size Distribution' -p 59702
```

2. Review the output.

```
[16:40:46]
IO Size Distribution
COUNT      EVENT
20774        arg3 = 4096
[16:40:47]
```

```
IO Size Distribution
COUNT      EVENT
20727       arg3 = 4096
[16:40:48]
IO Size Distribution
COUNT      EVENT
20713       arg3 = 4096
```

Process 59702 issues 4K IOs.

16.5. IO Pattern of a Process with the File Descriptor on Different GPUs

1. Run the following command.

```
# ./argdist -C
'u:/usr/local/cuda-x.y/lib/libcufile.so:cufio_gds_read():int,int,size:arg1,
arg6,arg3#IO Size Distribution arg1=fd, arg6=GPU# arg3=IOSize' -p `pgrep -n
gdsio`
```

2. Review the output, for example:

```
[17:00:03]
u:/usr/local/cuda-x.y/lib/
libcufile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3#IO Size Distribution
arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
5482         arg1 = 87, arg6 = 2, arg3 = 131072
7361         arg1 = 88, arg6 = 1, arg3 = 65536
9797         arg1 = 89, arg6 = 0, arg3 = 8192
11145        arg1 = 74, arg6 = 3, arg3 = 4096
[17:00:04]
u:/usr/local/cuda-x.y/lib/
libcufile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3#IO Size Distribution
arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
5471         arg1 = 87, arg6 = 2, arg3 = 131072
7409         arg1 = 88, arg6 = 1, arg3 = 65536
9862         arg1 = 89, arg6 = 0, arg3 = 8192
11079        arg1 = 74, arg6 = 3, arg3 = 4096
[17:00:05]
u:/usr/local/cuda-x.y/lib/
libcufile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3#IO Size Distribution
arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
5490         arg1 = 87, arg6 = 2, arg3 = 131072
7402         arg1 = 88, arg6 = 1, arg3 = 65536
9827         arg1 = 89, arg6 = 0, arg3 = 8192
11131        arg1 = 74, arg6 = 3, arg3 = 4096
```

`gdsio` issues READS to 4 files with `fd=87, 88, 89, 74` to GPU 2, 1, 0, and 3 and with `IO-SIZE` of 128K, 64K, 8K, and 4K.

16.6. Determine the IOPS and Bandwidth for a Process in a GPU

You can determine the IOPS and bandwidth for each process in a GPU.

1. Run the following command.

```
#!/argdist -C
'u:/usr/local/cuda-x.y/lib/libcufiofile.so:cufio_gds_read():int,int,size_t:arg1,
arg6,arg3:arg6==0||arg6==3#IO Size Distribution arg1=fd, arg6=GPU#
arg3=IOSize' -p `pgrep -n gdsio`
```

2. Review the output.

```
[17:49:33]
u:/usr/local/cuda-x.y/lib/
libcufiofile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3:arg6==0||arg6==3#IO
Size Distribution arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
9826         arg1 = 89, arg6 = 0, arg3 = 8192
11168        arg1 = 86, arg6 = 3, arg3 = 4096
[17:49:34]
u:/usr/local/cuda-x.y/lib/
libcufiofile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3:arg6==0||arg6==3#IO
Size Distribution arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
9815         arg1 = 89, arg6 = 0, arg3 = 8192
11141        arg1 = 86, arg6 = 3, arg3 = 4096
[17:49:35]
u:/usr/local/cuda-x.y/lib/
libcufiofile.so:cufio_gds_read():int,int,size_t:arg1,arg6,arg3:arg6==0||arg6==3#IO
Size Distribution arg1=fd, arg6=GPU# arg3=IOSize
COUNT      EVENT
9914         arg1 = 89, arg6 = 0, arg3 = 8192
11194        arg1 = 86, arg6 = 3, arg3 = 4096
```

- ▶ gdsio is doing IO on all 4 GPUs, and the output is filtered for GPU 0 and GPU 3.
- ▶ Bandwidth per GPU is GPU 0 - 9826 IOPS of 8K block size, and the bandwidth = ~80MB/s .

16.7. Display the Frequency of Reads by Processes that Issue cuFileRead

You can display information about the frequency of reads by process that issue the cuFileRead API.

1. Run the following command.

```
#!/argdist -C 'r:/usr/local/cuda-x.y/lib/libcufiofile.so:cuFileRead():u32:$PID'
```

2. Review the output, for example:

```
[17:58:01]
r:/usr/local/cuda-x.y/lib/libcufiofile.so:cuFileRead():u32:$PID
COUNT      EVENT
31191        $PID = 60492
31281        $PID = 60593
```

```
[17:58:02]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID
COUNT    EVENT
11741     $PID = 60669
30447     $PID = 60593
30670     $PID = 60492
[17:58:03]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID
COUNT    EVENT
29887     $PID = 60593
29974     $PID = 60669
30017     $PID = 60492
[17:58:04]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID
COUNT    EVENT
29972     $PID = 60593
30062     $PID = 60492
30068     $PID = 60669
```

16.8. Display the Frequency of Reads when cuFileRead Takes More than 0.1 ms

You can display the frequency of reads when the `cuFileRead` API takes more than 0.1 ms.

1. Run the following command.

```
#!/argdist -C 'r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000'
```

2. Review the output, for example:

```
[18:07:35]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000
COUNT    EVENT
17755     $PID = 60772
[18:07:36]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000
COUNT    EVENT
17884     $PID = 60772
[18:07:37]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000
COUNT    EVENT
17748     $PID = 60772
[18:07:38]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000
COUNT    EVENT
17898     $PID = 60772
[18:07:39]
r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead():u32:$PID:$latency > 100000
COUNT    EVENT
17811     $PID = 60772
```

16.9. Displaying the Latency of cuFileRead for Each Process

You can display the latency of the the `cuFileRead` API for each process.

1. Run the following command.

```
#!/funclatency /usr/local/cuda-x.y/lib/libcufile.so:cuFileRead -i 1 -T -u
```

2. Review the output, for example:

```
Tracing 1 functions for
"/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead"... Hit Ctrl-C to end.
```

Here are two process with PID 60999 and PID 60894 that are issuing `cuFileRead`:

```
18:12:11
Function = cuFileRead [60999]
  usecs      : count      distribution
    0 -> 1      : 0          |
    2 -> 3      : 0          |
    4 -> 7      : 0          |
    8 -> 15     : 0          |
   16 -> 31     : 0          |
   32 -> 63     : 0          |
   64 -> 127    : 17973     |*****|
  128 -> 255    : 13383     |*****|
  256 -> 511    : 27         |
Function = cuFileRead [60894]
  usecs      : count      distribution
    0 -> 1      : 0          |
    2 -> 3      : 0          |
    4 -> 7      : 0          |
    8 -> 15     : 0          |
   16 -> 31     : 0          |
   32 -> 63     : 0          |
   64 -> 127    : 17990     |*****|
  128 -> 255    : 13329     |*****|
  256 -> 511    : 19         |

18:12:12
Function = cuFileRead [60999]
  usecs      : count      distribution
    0 -> 1      : 0          |
    2 -> 3      : 0          |
    4 -> 7      : 0          |
    8 -> 15     : 0          |
   16 -> 31     : 0          |
   32 -> 63     : 0          |
   64 -> 127    : 18209     |*****|
  128 -> 255    : 13047     |*****|
  256 -> 511    : 58         |
Function = cuFileRead [60894]
  usecs      : count      distribution
    0 -> 1      : 0          |
    2 -> 3      : 0          |
    4 -> 7      : 0          |
    8 -> 15     : 0          |
   16 -> 31     : 0          |
   32 -> 63     : 0          |
   64 -> 127    : 18199     |*****|
  128 -> 255    : 13015     |*****|
  256 -> 511    : 46         |
```

```
512 -> 1023 : 1 |
```

16.10. Example: Tracking the Processes that Issue cuFileBufRegister

This example shows you can track processes that issue the `cuFileBufRegister` API.

1. Run the following command:

```
# ./trace 'u:/usr/local/cuda-x.y/lib/libcufio.so:cufio-internal-map "GPU %d Size %d Bounce-Buffer %d",arg1,arg2,arg5'
```

2. Review the output, for example:

PID	TID	COMM	FUNC	-
62624	62624	gdsio_verify	cufio-internal-map	GPU 0 Size 1048576 Bounce-Buffer 1
62659	62726	fio	cufio-internal-map	GPU 0 Size 8192 Bounce-Buffer 0
62659	62728	fio	cufio-internal-map	GPU 2 Size 131072 Bounce-Buffer 0
62659	62727	fio	cufio-internal-map	GPU 1 Size 65536 Bounce-Buffer 0
62659	62725	fio	cufio-internal-map	GPU 3 Size 4096 Bounce-Buffer 0

`gdsio_verify` issued an IO, but it did not register GPU memory using `cuFileBufRegister`. As a result, the GDS library pinned 1M of a bounce buffer on GPU 0. `FIO`, on the other hand, issued a `cuFileBufRegister` of 128K on GPU 2.

16.11. Example: Tracking Whether the Process is Constant when Invoking cuFileBufRegister

You can track whether the process is constant when invoking the `cuFileBufRegister` API.

1. Run the following command:

```
# ./trace 'u:/usr/local/cuda-x.y/lib/libcufio.so:cufio-internal-map (arg5 == 0) "GPU %d Size %d",arg1,arg2'
```

2. Review the output, for example:

PID	TID	COMM	FUNC	-
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576
444	472	cufile_sample_0	cufio-internal-map	GPU 0 Size 1048576

```

444      472      cufile_sample_0 cufio-internal-map GPU 0 Size 1048576
444      472      cufile_sample_0 cufio-internal-map GPU 0 Size 1048576
444      472      cufile_sample_0 cufio-internal-map GPU 0 Size 1048576

```

As seen in this example, there is one thread in a process that continuously issues 1M of `cuFileBufRegister` on GPU 0. This might mean that the API is called in a loop and might impact performance.



Note: `cuFileBufRegister` involves pinning GPU memory, which is an expensive operation.

16.12. Example: Monitoring IOs that are Going Through the Bounce Buffer

This example shows how you can monitor whether IOs are going through the bounce buffer.

1. Run the following command:

```

# ./trace 'u:/usr/local/cuda-x.y/lib/libcufile.so:cufio-internal-bb-done
"Application GPU %d Bounce-Buffer GPU %d Transfer Size %d Unaligned %d Registered
%d",
arg2,arg3,arg8,arg9,arg10'

```

2. Review the output, for example:

```

PID TID COMM  FUNC
1013 1041 gdsio App-GPU 0 Bounce-Buffer GPU 0 Transfer Size 1048576 Unaligned 1
Registered 0
1013 1042 gdsio App-GPU 3 Bounce-Buffer GPU 3 Transfer Size 1048576 Unaligned 1
Registered 0
1013 1041 gdsio App-GPU 0 Bounce-Buffer GPU 0 Transfer Size 1048576 Unaligned 1
Registered 0
1013 1042 gdsio App-GPU 3 Bounce-Buffer GPU 3 Transfer Size 1048576 Unaligned 1
Registered 0

```

The `gdsio` app has 2 threads and both are doing unaligned IO on GPU 0 and GPU 3. Since the IO is unaligned, bounce buffers are also from the same application GPU.

16.13. Example: Tracing `cuFileRead` and `cuFileWrite` Failures, Print, Error Codes, and Time of Failure

This example shows you how to trace the `cuFileRead` and `cuFileWrite` failures, print, error codes, and time of failure.

1. Run the following command:

```

# ./trace 'r:/usr/local/cuda-x.y/lib/libcufile.so:cuFileRead ((int)retval < 0)
"cuFileRead failed: %d", retval' 'r:/usr/local/cuda-x.y/lib/
libcufile.so:cuFileWrite ((int)retval < 0)
"cuFileWrite failed: %d", retval' -T

```

2. Review the output, for example:

TIME	PID	TID	COMM	FUNC	-
23:22:16	4201	4229	gdsio	cuFileRead	cuFileRead failed: -5
23:22:42	4237	4265	gdsio	cuFileWrite	cuFileWrite failed: -5

In this example, two failures were observed with EIO (-5) as the return code with the timestamp.

16.14. Example: User-Space Statistics for Each GDS Process

The `cuFile` library exports user-level statistics in the form of API level counters for each process. In addition to the regular GDS IO path, there are paths for user-space file-systems and IO compatibility modes that use POSIX read/writes, which do not go through the `nvidia-fs` driver. The user-level statistics are more useful in these scenarios.

There is a verbosity level for the counters which users can specify using JSON configuration file to enable and set the level. The following describes various verbosity levels.

Table 10. User-Space Statistics for Each GDS Process

Level	Description
Level 0	cuFile stats will be disabled.
Level 1	cuFile stats will report only Global Counters like overall throughput, average latency and error counts.
Level 2	With the Global Counters, an IO Size histogram will be reported for information on access patterns.
Level 3	At this level, per GPU counters are reported and also live usage of cuFile internal pool buffers.

The following is the JSON configuration key to enable GDS statistics by using the `/etc/cufile.json` file:

```
"profile": {
    // cufile stats level(0-3)
    "cufile_stats": 3
},
```

16.15. Example: Viewing GDS User-Level Statistics for a Process

This example shows how you can use the `gds_stats` tool to display user-level statistics for a process.

Prerequisite: Before you run the tool, ensure that the IO application is active, and the `gds_stats` has the same user permissions as the application.

The `gds_stats` tool can be used to read statistics that are exported by `libcufile.so`.

The output of the statistics is displayed in the standard output. If the user permissions are not the same, there might not be sufficient privilege to view the stats. A future version of `gds_stats` will integrate `nvdi-fs` kernel level statistics into this tool.

To use the tool, run the following command:

```
$ /usr/local/cuda-x.y/tools/gds_stats -p <pidof application> -l <stats_level(1-3)>
```

When specifying the statistics level, ensure that the corresponding level (`profile.cufile_stats`) is also enabled in the `/etc/cufile.json` file.

The GDS user level statistics are logged once to `cufile.log` file when the library is shut down, or the `cuFileDriverClose` API is run. To view statistics in the log file, set the log level to INFO.

16.16. Example: Displaying Sample User-Level Statistics for each GDS Process

This example shows how to display sample user-level statistics for each GDS process.

1. Run the following command:

```
$ ./gds_stats -p 23198 -l 3
```

2. Review the output, for example:

```
cuFile STATS VERSION : 4
GLOBAL STATS:
Total Files: 1
Total Read Errors : 0
Total Read Size (MiB): 7302
Read BandWidth (GiB/s): 0.691406
Avg Read Latency (us): 6486
Total Write Errors : 0
Total Write Size (MiB): 0
Write BandWidth (GiB/s): 0
Avg Write Latency (us): 0
READ-WRITE SIZE HISTOGRAM :
0-4 (KiB): 0 0
4-8 (KiB): 0 0
8-16 (KiB): 0 0
16-32 (KiB): 0 0
```

```

32-64(KiB): 0 0
64-128(KiB): 0 0
128-256(KiB): 0 0
256-512(KiB): 0 0
512-1024(KiB): 0 0
1024-2048(KiB): 0 0
2048-4096(KiB): 3651 0
4096-8192(KiB): 0 0
8192-16384(KiB): 0 0
16384-32768(KiB): 0 0
32768-65536(KiB): 0 0
65536-...(KiB): 0 0
PER_GPU STATS:
GPU_0 Read: bw=0.690716 util(%)=199 n=3651 posix=0 unalign=0 dr=0 r_sparse=0
  r_inline=0 err=0 MiB=7302 Write: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 err=0
  MiB=0 BufRegister: n=2 err=0 free=0 MiB=4
PER_GPU POOL BUFFER STATS:
PER_GPU POSIX POOL BUFFER STATS:

PER_GPU RDMA STATS:
GPU 0000:43:00.0 :   mlx5_0(130:64):Reads: 3594 Writes: 0   mlx5_1(130:64):Reads:
  3708 Writes: 0
RDMA MRSTATS:
peer name   nr_mrs      mr_size(MiB)
mlx5_0      1           2
mlx5_1      1           2

```

Chapter 17. User-Space Counters in GPUDirect Storage

The following tables provide information about user-space counters in GDS.

Table 11. Global cuFile Counters

Counter Name	Description
Total Files	Total number of files registered successfully with <code>cuFileHandleRegister</code> . This is a cumulative counter. <code>cuFileHandleDeregister</code> does not change this counter.
Total Read Errors	Total number of <code>cuFileRead</code> errors.
Total Read Size	Total number of bytes read in MB using <code>cuFileRead</code> .
Read Bandwidth	Average overall read throughput in GiB/s over one second time period.
Avg Read Latency	Overall average read latency in microseconds over one second time period.
Total Write Errors	Total number of <code>cuFileWrite</code> errors.
Total Write Size	Total number of bytes written in MB using <code>cuFileWrite</code> .
Write Bandwidth	Overall average write throughput in GiB/s over one second time period.
Avg Write Latency	Overall average read latency in microseconds over one second time period.

Table 12. IO-Size Histogram

Counter Name	Description
Read	Distribution of number of <code>cuFileRead</code> requests based on IO size. Bin Size uses a 4K log scale.
Write	Distribution of number of <code>cuFileWrite</code> requests based on IO size. Bin Size uses a 4K log scale.

Table 13. Per-GPU Counters

Counter Name	Description
Read.bw/Write.bw	Average GPU read/write bandwidth in GiB/s per GPU.
Read.util/Write.util	Average per GPU read/write utilization in %. If A is the total length of time the resource was busy in a time interval T, then utilization is defined as A/T. Here the utilization is reported over one second period.
Read.n/Write.n	Number of <code>cuFileRead/cuFileWrite</code> requests per GPU.
Read.posix/Write.posix	Number of <code>cuFileRead/cuFileWrite</code> using POSIX read/write APIs per GPU.
Read.dr/Write.dr	Number of <code>cuFileRead/cuFileWrites</code> for a GPU have been issued using dynamic routing. If the routing policy uses <code>SYS_MEM</code> , GPU posix counters for read/writes will be incrementing in addition to the <code>dr</code> counter. Note: This counter does not tell which GPU was actually being used for routing the IO. For the latter information, one needs to observe the <code>PER_GPU POOL BUFFER STATS/PER_GPU POSIX POOL BUFFER STATS</code> .
Read.unalign/Write.unalign	Number of <code>cuFileRead/cuFileWrite</code> per GPU which have at least one IO parameter not 4K aligned. This can be either size, file offset or device pointer.
Read.error/Write.error	Number of <code>cuFileRead/cuFileWrite</code> errors per GPU.

Counter Name	Description
Read.mb/Write.mb	Total number of bytes in MB read/written using <code>cuFileRead/cuFileWrite</code> per GPU.
BufRegister.n	Total number of <code>cuFileBufRegister</code> calls per GPU.
BufRegister.err	Total number of errors per GPU seen with <code>cuFileBufRegister</code> .
BufRegister.free	Total number of <code>cuFileBufRegister</code> calls per GPU.
BufRegister.mb	Total number of bytes in MB currently registered per GPU.

Table 14. Bounce Buffer Counters Per GPU

Counter Name	Description
pool_size_mb	Total size of buffers allocated for per GPU bounce buffers in MB.
used_mb	Total size of buffers currently used per GPU for bounce buffer based IO.
usage	Fraction of bounce buffers used currently.

17.1. Distribution of IO Usage in Each GPU

The `cuFile` library has a metric for IO utilization per GPU by application. This metric indicates the amount of time, in percentage, that the `cuFile` resource was busy in IO.

To run a single-threaded `gdsio` test, run the following command:

```
./gdsio -f /mnt/md1/test -d 0 -n 0 -w 1 -s 10G -i 4K -x 0 -I 1
```

Here is the sample output:

```
PER_GPU STATS
GPU_0 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 err=0 mb=0 Write: bw=0.154598
util(%)=89 n=510588 posix=0 unalign=0 err=0 mb=1994 BufRegister: n=1 err=0 free=0
mb=0
```

The `util` metric says that the application was completing IO on GPU 0 89% of the time.

To run a `gdsio` test using two-threads, run the following command:

```
./gdsio -f /mnt/md1/test -d 0 -n 0 -w 2 -s 10G -i 4K -x 0 -I 1
```

Here is the sample output:

```
PER_GPU STATS
```

```
GPU 0 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 err=0 mb=0 Write:
bw=0.164967 util(%)=186 n=140854 posix=0 unalign=0 err=0 mb=550 BufRegister: n=2
err=0 free=0 mb=0
```

Now the utilization is ~186%, which indicates the amount of parallelism in the way each GPU is used for IO.

17.2. User-space Statistics for Dynamic Routing

The `PER_GPU` section of `gds_stats` has a `dr` counter which indicates how many `cuFileRead/cuFileWrites` for a GPU have been issued using dynamic routing.

```
$ ./gds_stats -p <pidof application> -l 3

GPU 0 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 r_sparse=0 r_inline=0
err=0 MiB=0 Write: bw=3.37598 util(%)=532 n=6629 posix=0 unalign=0 dr=6629 err=0
MiB=6629 BufRegister: n=4 err=0 free=0 MiB=4
GPU 1 Read: bw=0 util(%)=0 n=0 posix=0 unalign=0 dr=0 r_sparse=0 r_inline=0
err=0 MiB=0 Write: bw=3.29297 util(%)=523 n=6637 posix=0 unalign=0 dr=6637 err=0
MiB=6637 BufRegister: n=4 err=0 free=0 MiB=4
```

Chapter 18. User-Space RDMA Counters in GPUDirect Storage

The library provides counters to monitor the RDMA traffic at a per-GPU level and requires that cuFile starts verbosity with a value of 3.

Table [14-1](#) provides the following information:

- ▶ Each column stores the total number of bytes that are sent/received between a GPU and a NIC.
- ▶ Each row shows the distribution of RDMA load with regards to a GPU across all NICs.
- ▶ Each row reflects the order of affinity that a GPU has with a NIC.

Ideally, all traffic should be routed through the NIC with the best affinity or is closest to the GPU as shown in *Example 1* in [cuFile RDMA IO Counters \(PER_GPU_RDMA_STATS\)](#).

In the annotation of each NIC entry in the table, the major number is the pci-distance in terms of the number of hops between the GPU and the NIC, and the minor number indicates the current bandwidth of the NIC (link_width multiplied by pci-generation). The NICs that the GPUs use for RDMA are loaded from the `rdma_dev_addr_list` `cufile.json` property:

```
"rdma_dev_addr_list": [  
  "172.172.1.240",  
  "172.172.1.241",  
  "172.172.1.242",  
  "172.172.1.243",  
  "172.172.1.244",  
  "172.172.1.245",  
  "172.172.1.246",  
  "172.172.1.247" ],
```

Each IP address corresponds to an IB device that appear as column entries in the RDMA counter table.

18.1. cuFile RDMA IO Counters (PER_GPU RDMA STATS)

The following tables list cuFile RDMA IO counters.

Table 15. cuFile RDMA IO Counters (PER_GPU RDMA STATS)

Entry	Description
GPU	Bus device function
NIC	+) Bus device function +) Device Attributes ++) pci-distance between GPU and NIC ++) device bandwidth indicator +) Send/Receive bytes

Table 16. Example 1

GPU 0000:34:	mlx5_3 (3:48):6	mlx5_5 (7:48):0	mlx5_15 (138:48):	mlx5_15 (138:48):	mlx5_17 (138:48):	mlx5_9 (138:48):	mlx5_13 (138:48):	mlx5_7 (138:12):	:0
GPU 0000:36:	mlx5_3 (3:48):6	mlx5_5 (7:48):1	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_17 (138:48):	mlx5_9 (138:48):	mlx5_13 (138:48):	mlx5_7 (138:12):	:0
GPU 0000:3b:	mlx5_5 (3:48):5	mlx5_3 (7:48):0	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_17 (138:48):	mlx5_9 (138:48):	mlx5_13 (138:48):	mlx5_7 (138:12):	:0
GPU 0000:57:	mlx5_7 (3:12):4	mlx5_9 (7:48):0	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_5 (138:48):	mlx5_17 (138:48):	mlx5_13 (138:48):	mlx5_3 (138:48):	:0
GPU 0000:59:	mlx5_7 (3:12):4	mlx5_9 (7:48):1	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_5 (138:48):	mlx5_17 (138:48):	mlx5_13 (138:48):	mlx5_3 (138:48):	:0
GPU 0000:5c:	mlx5_9 (3:48):4	mlx5_7 (7:12):0	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_5 (138:48):	mlx5_17 (138:48):	mlx5_13 (138:48):	mlx5_3 (138:48):	:0
GPU 0000:5e:	mlx5_9 (3:48):4	mlx5_7 (7:12):0	mlx5_15 (138:48):	mlx5_19 (138:48):	mlx5_5 (138:48):	mlx5_17 (138:48):	mlx5_13 (138:48):	mlx5_3 (138:48):	:0

18.2. cuFile RDMA Memory Registration Counters (RDMA MRSTATS)

The following tables list cuFile RDMA memory registration counters.

Table 17. cuFile RDMA IO Counters (PER_GPU RDMA STATS)

Entry	Description
peer name	System name of the NIC.
nr_mrs	Count of active memory registration per NIC.

Entry	Description
mr_size(mb)	Total size

Table 18. Example 2

peer name	nr_ms	mr_size (mb)
mlx5_3	128	128
mlx5_5	128	128
mlx5_11	0	0
mlx5_1	0	0
mlx5_15	128	128
mlx5_19	128	128
mlx5_17	128	128
mlx5_9	128	128
mlx5_13	128	128
mlx5_7	128	128


Chapter 19. Cheat Sheet for Diagnosing Problems

The following tables can help users diagnose GDS problems.

Make sure to go through following variables and observe if performance is where it needs to be.

Variable impacting performance	Description	Steps to take enable/disable the functionality ("How to")
Compat mode	Disable compat mode in cufile.json	Set allow_compat_mode: false in cufile.json. Or Set CUFIL_FORCE_COMPAT_MODE environment variable to false.
Log level	Set log level to ERROR in cufile.json	Following setting in cufile.json will set the logging level to ERROR. <pre>"logging": { // log directory, if not enabled will create log file under current working directory //"dir": "/home/ <xxxx>", // NOTICE ERROR WARN INFO DEBUG TRACE (in decreasing order of severity) "level": "ERROR" },</pre>

Variable impacting performance	Description	Steps to take enable/disable the functionality ("How to")
Nvidia-fs stats	Make sure nvidia-fs read/write stats are disabled. These can have a performance impact for small IO sizes. By default these are disabled.	<p>To check the current state of the stats, use the following command.</p> <pre>#cat /sys/module/nvidia_fs/parameters/rw_stats_enabled</pre> <p>0 - Disabled 1 - Enabled</p> <p>To disable them,</p> <pre>echo 0 > /sys/module/nvidia_fs/parameters/rw_stats_enabled</pre>
GDR stats/RDMA stats (CQE errors)		
Relax Ordering	<p>For distributed file systems, make sure NICs have relax ordering is enabled</p> <p>set "MAX_ACC_OUT_READ=44" on CX-6.</p> <p>Set "MAX_ACC_OUT_READ=128" for CX-7.</p>	<pre>sudo mlxconfig -y -d <NIC> set ADVANCED_PCI_SETTINGS=1</pre> <pre>sudo mlxconfig -y -d <NIC> set MAX_ACC_OUT_READ=44 128</pre> <pre>sudo reboot</pre>
Persistent mode	Enable persistent mode	
Clock speed	Set clock speed to maximum	
BAR size	Make sure BAR size is enabled to maximum possible value	
Numa affinity	Set numa affinity of the process where NIC-GPU are in the same switch	
MRRS	<p>Sets the PCIe Max Read Request Size for the NIC/ NVMe</p> <p>Specifying Max Read request size enables the Requestor (NIC/NVME) to read data from the GPU memory upto the specified size to improve Writes from GPU to storage performance.</p>	<p>Check the setting using</p> <pre># lspci -vvv -s <B:D.F> grep -i MaxReadReq</pre> <p>Read the current value</p> <pre>#setpci -v -s <B:D.F> cap_exp+8.w</pre> <p>To set to 4K</p> <pre>#setpci -v -s <B:D.F> cap_exp+8.w=5000:7000</pre>

Variable impacting performance	Description	Steps to take enable/disable the functionality ("How to")
		<p>To set to 512 bytes</p> <pre>#setpci -v -s <B:D.F> cap_exp+8.w=2000:7000</pre> <p>The acceptable values are: 0 - 128B, 1 - 256B, 2 - 512B, 3 - 1024B, 4 - 2048B and 5 - 4096B.</p> <div data-bbox="1036 604 1422 789" style="border: 1px solid gray; background-color: #f0f0f0; padding: 5px;"> <p> Note: Specifying selector indexes outside this range might cause the system to crash.</p> </div>

For ROCE setups, consider additional following items:

CPU Governor	Performance	#cpupower frequency-set -g performance
RX/TX ring	Set them to maximum	#ethtool -G \$adapter rx \$(ethtool -g \$adapter awk '/RX:/ {print \$NF; exit}')#ethtool -G \$adapter tx \$(ethtool -g \$adapter awk '/TX:/ {print \$NF; exit}')
RX/TX channels	Set to max allowed	#ethtool -L \$adapter combined 15
LRO	Turn on large receive offload	#ethtool -K \$adapter lro on
IRQ affinity	Set IRQ affinity to the affine NUMA node	
IRQ balance	Turn off IRQ balancer	#systemctl stop irqbalance
TX queue length	Increase TX queue length	#ifconfig \$adapter \$addr/\$netmask mtu 9000 txqueuelen 20000 up

If the above steps do not help, collect the following information and share it with us.

Measure GDR performance	For RDMA connectivity and performance issues run ib_read and ib_write tests with cuda and GPU enabled	<p>Please follow instructions at https://github.com/linux-rdma/perftest</p> <p>In tests use the option</p> <pre>--use_cuda=<gpu index></pre>
Use Nsight systems	For RDMA connectivity and performance issues run ib_read and ib_write tests with cuda and GPU enabled	<pre>cat /etc/cufile.json grep nvtx // nvtx profiling on/off "nvtx": true, /usr/local/cuda/bin/nsys profile <command></pre>

Describe env	Virtual (Docker or actual VM) or BM	<pre># printenv # lsb_release -a # dmidecode # docker info # docker container inspect <container id></pre>
Collect gds logs	Collect cufile.log /usr/local/cuda/gds/tools/ gds_log_collection.py	<pre># /usr/local/cuda/gds/tools/ gds_log_collection.py</pre>

Debugging:

Dmesg errors?	Check for kernel errors	<pre># dmesg # /var/log/kern.log</pre>
MiG mode enabled?	Check if MIG is enabled	<pre># nvidia-smi mig -lgi</pre>
FM enabled or not	For NVSwitch based systems. Check if fabric manager is running and active without any errors	<pre># systemctl status nvidia fabricmanager</pre>

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, DGX, DGX-1, DGX-2, DGX-A100, Tesla, and Quadro are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020-2024 NVIDIA Corporation and affiliates. All rights reserved.

