AMD Instinct Hub

Release 0.1.0

Advanced Micro Devices, Inc.

Mar 31, 2026

HOW TO

When running HPC and AI applications in a cluster network environment, performance is only as fast as the slowest individual node in the network. To achieve optimal performance, each server must be configured for maximum data transfer rates and bandwidth utilization based on the available hardware. It is crucial to validate both host and device performance in single-node and multi-node environments using the appropriate benchmarking tools.

Refer to the relevant networking guides for step-by-step instructions on validating network configurations in single-node and multi-node environments. These guides cover system settings, device configurations, networking tools, and performance tests to ensure AMD Instinct™-powered GPU clusters achieve optimal speed and bandwidth during operation.

How to

- *Enabling MPI*
- *Single-node network configuration*
- *Multi-node network configuration*
- *RoCE network configuration*
- Multi-node inference load balancing

Reference

- *Hardware support*
- *Cluster design*

Note

AMD Instinct systems vary in form and configuration, and cluster design adds additional layers of complexity. The guidelines in this documentation are written at a high level for broad applicability across diverse environments. While certain scenarios may include specific hardware examples, your setup will likely differ in terms of GPUs and CPUs per server, firmware versions, and network interconnects. Adjustments might be necessary to align with your particular configuration.

CHAPTER

ONE

GPU-ENABLED MESSAGE PASSING INTERFACE

The Message Passing Interface (MPI) is a standard API for distributed and parallel application development that can scale to multi-node clusters. To facilitate the porting of applications to clusters with GPUs, ROCm enables various technologies. You can use these technologies add GPU pointers to MPI calls and enable ROCm-aware MPI libraries to deliver optimal performance for both intra-node and inter-node GPU-to-GPU communication.

The AMD kernel driver exposes remote direct memory access (RDMA) through *PeerDirect* interfaces. This allows network interface cards (NICs) to directly read and write to RDMA-capable GPU device memory, resulting in high-speed direct memory access (DMA) transfers between GPU and NIC. These interfaces are used to optimize inter-node MPI message communication.

The Open MPI project is an open source implementation of the MPI. It's developed and maintained by a consortium of academic, research, and industry partners. To compile Open MPI with ROCm support, refer to the following sections:

- *ROCm-aware Open MPI on InfiniBand and RoCE networks using UCX*
- *ROCm-aware Open MPI using libfabric*

1.1 ROCm-aware Open MPI on InfiniBand and RoCE networks using UCX

The Unified Communication Framework (UCX), is an open source, cross-platform framework designed to provide a common set of communication interfaces for various network programming models and interfaces. UCX uses ROCm technologies to implement various network operation primitives. UCX is the standard communication library for InfiniBand and RDMA over Converged Ethernet (RoCE) network interconnect. To optimize data transfer operations, many MPI libraries, including Open MPI, can leverage UCX internally.

UCX and Open MPI have a compile option to enable ROCm support. To install and configure UCX to compile Open MPI for ROCm, use the following instructions.

1. Set environment variables to install all software components in the same base directory. We use the home directory in our example, but you can specify a different location if you want.

```
export INSTALL_DIR=$HOME/ompi_for_gpu
export BUILD_DIR=/tmp/ompi_for_gpu_build
mkdir -p $BUILD_DIR
```

2. Install UCX. To view UCX and ROCm version compatibility, refer to the communication libraries tables

```
export UCX_DIR=$INSTALL_DIR/ucx
cd $BUILD_DIR
git clone https://github.com/openucx/ucx.git -b v1.15.x
cd ucx
```

(continues on next page)

(continued from previous page)

```
./autogen.sh
mkdir build
cd build
../configure -prefix=$UCX_DIR \
    --with-rocm=/opt/rocm
make -j $(nproc)
make -j $(nproc) install
```

3. Install Open MPI.

```
export OMPI_DIR=$INSTALL_DIR/ompi
cd $BUILD_DIR
wget https://download.open-mpi.org/release/open-mpi/v5.0/openmpi-5.0.7.tar.gz
tar zxvf openmpi-5.0.7.tar.gz
cd openmpi-5.0.7
mkdir build
cd build
../configure --prefix=$OMPI_DIR --with-ucx=$UCX_DIR \
    --with-rocm=/opt/rocm
make -j $(nproc)
make install
```

1.1.1 ROCm-enabled OSU benchmarks

You can use OSU Micro Benchmarks (OMB) to evaluate the performance of various primitives on ROCm-supported AMD GPUs. The `--enable-rocm` option exposes this functionality.

```
export OSU_DIR=$INSTALL_DIR/osu
cd $BUILD_DIR
wget http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-7.2.tar.gz
tar xfz osu-micro-benchmarks-7.2.tar.gz
cd osu-micro-benchmarks-7.2
./configure --enable-rocm \
    --with-rocm=/opt/rocm \
    CC=$OMPI_DIR/bin/mpicc CXX=$OMPI_DIR/bin/mpicxx \
    LDFLAGS="-L$OMPI_DIR/lib/ -lmpi -L/opt/rocm/lib/ -lamdhip64" \
    CFLAGS="$(hipconfig -C | tr -d '\n')"
    CXXFLAGS="-std=c++11"
make -j $(nproc)
```

1.1.2 Intra-node run

Before running an Open MPI job, you must set the following environment variables to ensure that you're using the correct versions of Open MPI and UCX.

```
export LD_LIBRARY_PATH=$OMPI_DIR/lib:$UCX_DIR/lib:/opt/rocm/lib
export PATH=$OMPI_DIR/bin:$PATH
```

To run the OSU bandwidth benchmark between the first two GPU devices (`GPU 0` and `GPU 1`) inside the same node, use the following code.

```
$OMPI_DIR/bin/mpirun -np 2 \
-x UCX_TLS=sm,self,rocm \
--mca pml ucx \
./c/mpi/pt2pt/standard/osu_bw D D
```

This measures the unidirectional bandwidth from the first device (`GPU 0`) to the second device (`GPU 1`). To select specific devices, for example `GPU 2` and `GPU 3`, include the following command:

```
export HIP_VISIBLE_DEVICES=2,3
```

To force using a copy kernel instead of a DMA engine for the data transfer, use the following command:

```
export HSA_ENABLE_SDMA=0
```

The following output shows the effective transfer bandwidth measured for inter-die data transfer between `GPU 2` and `GPU 3` on a system with MI250 GPUs. For messages larger than 67 MB, an effective utilization of about 150 GB/sec is achieved:

```
# OSU MPI-ROCM Bandwidth Test v5.9
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
2                    1.01
4                    2.03
8                    4.04
16                   7.77
32                  10.43
64                  22.47
128                 44.26
256                 88.10
512                114.57
1024               211.02
2048               290.50
4096               335.14
8192               389.36
16384             1826.78
32768             3484.68
65536             6773.21
131072           12619.03
262144           22524.74
524288           40163.84
1048576          60298.85
2097152          83916.57
4194304         118524.13
8388608         133420.64
16777216        143527.03
33554432        147779.60
67108864        151540.93
134217728       151317.96
268435456       150834.76
```

1.1.3 Collective operations

Collective operations on GPU buffers are best handled through the Unified Collective Communication (UCC) library component in Open MPI. To accomplish this, you must configure and compile the UCC library with ROCm support.

> **Note**
>
> You can verify UCC and ROCm version compatibility using the communication libraries tables

```
export UCC_DIR=$INSTALL_DIR/ucc
git clone https://github.com/openucx/ucc.git -b v1.2.x
cd ucc
./autogen.sh
./configure --with-rocm=/opt/rocm \
            --with-ucx=$UCX_DIR   \
            --prefix=$UCC_DIR
make -j && make install

# Configure and compile Open MPI with UCX, UCC, and ROCm support
cd ompi
./configure --with-rocm=/opt/rocm  \
            --with-ucx=$UCX_DIR    \
            --with-ucc=$UCC_DIR
            --prefix=$OMPI_DIR
```

To use the UCC component with an MPI application, you must set additional parameters:

```
mpirun --mca pml ucx --mca osc ucx \
   --mca coll_ucc_enable 1     \
   --mca coll_ucc_priority 100 -np 64 ./my_mpi_app
```

1.2 ROCm-aware Open MPI using libfabric

For network interconnects that are not covered in the previous category, such as HPE Slingshot, ROCm-aware communication can often be achieved through the libfabric library. For more information, refer to the libfabric documentation.

Note

When using Open MPI v5.0.x with libfabric support, shared memory communication between processes on the same node goes through the *ob1/sm* component. This component has fundamental support for GPU memory that is, accomplished by using a staging host buffer Consequently, the performance of device-to-device shared memory communication is lower than the theoretical peak performance allowed by the GPU-to-GPU interconnect.

1. Install libfabric. Note that libfabric is often pre-installed. To determine if it's already installed, run:

   ```
   module avail libfabric
   ```

   Alternatively, you can download and compile libfabric with ROCm support. Note that not all components required to support some networks (e.g., HPE Slingshot) are available in the open source repository. Therefore, using a pre-installed libfabric library is strongly recommended over compiling libfabric manually.

   If a pre-compiled libfabric library is available on your system, you can skip the following step.

2. Compile libfabric with ROCm support.

   ```
   export OFI_DIR=$INSTALL_DIR/ofi
   cd $BUILD_DIR
   git clone https://github.com/ofiwg/libfabric.git -b v1.19.x
   cd libfabric
   ./autogen.sh
   ./configure --prefix=$OFI_DIR   \
   ```

(continues on next page)

(continued from previous page)

```
            --with-rocr=/opt/rocm
make -j $(nproc)
make install
```

1.2.1 Installing Open MPI with libfabric support

To build Open MPI with libfabric, use the following code:

```
export OMPI_DIR=$INSTALL_DIR/ompi
cd $BUILD_DIR
git clone --recursive https://github.com/open-mpi/ompi.git \
    -b v5.0.x
cd ompi
./autogen.pl
mkdir build
cd build
../configure --prefix=$OMPI_DIR --with-ofi=$OFI_DIR \
                --with-rocm=/opt/rocm
make -j $(nproc)
make install
```

1.2.2 ROCm-aware OSU with Open MPI and libfabric

Compiling a ROCm-aware version of OSU benchmarks with Open MPI and libfabric uses the same process described in *ROCm-enabled OSU benchmarks*.

To run an OSU benchmark using multiple nodes, use the following code:

```
export LD_LIBRARY_PATH=$OMPI_DIR/lib:$OFI_DIR/lib64:/opt/rocm/lib
$OMPI_DIR/bin/mpirun --mca pml ob1 --mca btl_ofi_mode 2 -np 2 \
./c/mpi/pt2pt/standard/osu_bw D D
```

CHAPTER

TWO

SINGLE-NODE NETWORK CONFIGURATION FOR AMD INSTINCT ACCELERATORS

This section explains setting up a testing environment on a single accelerator node and running benchmarks to simulate an AI or HPC workload.

2.1 Prerequisites

Before following the steps in the following sections, ensure you have completed these prerequisites.

1. Install GPU and network hardware. Refer to the *hardware support matrix*.
2. Install OS and required GPU and network software on each node:
   - Install the ROCm software stack.
   - Install network drivers for NICs. If using InfiniBand, also install OpenSM.

   > **Note**
   >
   > For MI325X, *ROCm version 6.3.1 or higher* is required.

3. Ensure network settings are correctly configured for your hardware.
4. Configure system BIOS and OS settings according to System optimization for your architecture (MI300, MI200, and so on).
5. Disable NUMA balancing.
   a. Run `sudo sh -c 'echo 0 > /proc/sys/kernel/numa_balancing'`.
   b. To verify NUMA balancing is disabled, run `cat /proc/sys/kernel/numa_balancing` and confirm that `0` is returned.

   This disables NUMA balancing during the current system session and does not persist through reboot. To permanently disable NUMA balancing, use the grub boot loader settings.
   a. Open `/etc/default/grub` for editing.
   b. Locate the line that starts with `GRUB_CMDLINE_LINUX_DEFAULT` and add `numa_balancing=disable` inside the double quotes, separated from the other values by a space.

   ```
   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash numa_balancing=disable"
   ```

   c. Depending on your system OS, run `sudo update-grub`, `sudo grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg`, or `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` then reboot.

d. Run `sudo cat /proc/cmdline` to verify grub command line settings.

It's a common practice to first disable NUMA balancing on a temporary basis , then make it a permanent grub setting once you verify the setting contributes to better peformance. See Disable NUMA auto-balancing for more information.

6. Disable PCI ACS (access control services). Run the disable-acs-script on all PCIe devices supporting it. This must be done after each reboot.

> **Note**
>
> Some systems can disable ACS in BIOS. Observations of this feature show it does not always impact the operating system ACS configuration. Therefore, the disable ACS script must still be run before any workloads after a server has been rebooted.

7. Configure IOMMU settings.
   a. Add `iommu=pt`, `pci=realloc=off`, and `pci=bfsort` to the `GRUB_CMDLINE_LINUX_DEFAULT` entry in `/etc/default/grub`.
   b. Depending on your system OS, run `sudo update-grub`, `sudo grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg`, or `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` then reboot.
   c. Run `sudo cat /proc/cmdline` to verify grub command line settings.
   d. See GRUB settings and Issue #5: Application hangs on Multi-GPU systems for more information.
8. Verify group permissions.
   a. Ensure the user belongs to the `render` and `video` groups.
   b. Refer to Configuring permissions for GPU access for guidance.

2.1.1 Best practices for software consistency

To ensure consistent software configurations across systems:

- Use a shared NFS (network file system) mount. Install the necessary software on a common NFS mount accessible to all systems.
- Create a system image with all the software installed. Re-image when software changes are made.

2.2 Validate PCIe performance

Checking that your relevant PCIe devices (GPUs, NICs, and internal switches) are using the maximum available transfer speed and width in their respective bus keeps you from having to troubleshoot any related issues in subsequent testing where it may not be obvious.

> **Tip**
>
> Gather all the PCIe addresses for your GPUs, NICs, and switches in advance and take note of them so you have them on hand for next steps.

2.2.1 Check PCIe device speed and width

1. From the command line of your host, run `lspci` to retrieve a list of PCIe devices and locate your GPU and network devices.
2. Run `sudo lspci -s <PCI address> -vvv | grep Speed` to review the speed and width of your device. This example shows the speed and width for a GPU at the address `02:00.0`.

   Shell output

   ```
   $ sudo lspci -s 02:00.0 -vvv | grep Speed

   LnkCap: Port #0, Speed 32GT/s, Width x16, ASPM L0s L1, Exit Latency L0s <64ns, L1
   ↪<1us
   LnkSta: Speed 32GT/s (ok), Width x16 (ok)
   ```

   Commands

   ```
   sudo lspci -s 02:00.0 -vvv | grep Speed
   ```

   The maximum supported speed of the GPU is reported in `LnkCap` along with a width of x16. Current status is shown in `LnkSta`–both speed and width are aligned. Your values may differ depending on your hardware.
3. Query and validate all GPUs in your node with the previous steps.
4. Gather the PCI addresses for your NICs and validate them next. See this example from a NIC running at `05:00.0`:

   Shell output

   ```
   $ sudo lspci -s 05:00.0 -vvv | grep Speed

   LnkCap: Port #0, Speed 16GT/s, Width x16, ASPM not supported
   LnkSta: Speed 16GT/s (ok), Width x16 (ok)
   ```

   Commands

   ```
   sudo lspci -s 05:00.0 -vvv | grep Speed
   ```

   Here, the NIC is running at a speed of 16GT/s. However, because the NIC configuration only supports PCIe Gen4 speeds, this is an expected value.

Once you verify all GPUs and NICs are running at maximum supported speeds and widths, then proceed to the next section.

> **Note**
>
> If you're running a cloud instance, hardware passthrough to your guest OS might not be accurate. Verify your `lspci` results with your cloud provider.

2.2.2 Check PCIe switch speed and width

Now, check the PCIe switches to ensure they are operating at the maximum speed and width for the `LnkSta` (Link Status).

1. Run `lspci -vv` and `lspci -tv` to identify PCIe switch locations on the server.
2. Run `lspci -vvv <PCI address> | grep Speed` to verify speed and width as previously demonstrated.

2.2.3 Check max payload size and max read request

The `MaxPayload` and `MaxReadReq` attributes define the maximum size of PCIe packets and the number of simultaneous read requests, respectively. For optimal bandwidth, ensure that all GPUs and NICs are configured to use the maximum values for both attributes.

1. Run `sudo lspci -vvv -s <PCI address> | grep DevCtl: -C 2` to review max payload size and max read request. Here is an example using the same NIC as before.

Shell output

```
$ sudo lspci -vvv -s 05:00.0 | grep DevCtl: -C 2

DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s <4us, L1 <64us
         ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset+ SlotPowerLimit 40.000W
DevCtl: CorrErr+ NonFatalErr+ FatalErr+ UnsupReq-
         RlxdOrd+ ExtTag+ PhantFunc- AuxPwr+ NoSnoop+ FLReset-
         MaxPayload 512 bytes, MaxReadReq 4096 bytes
```

Commands

```
sudo lspci -vvv -s 05:00.0 | grep DevCtl: -C 2
```

2. `MaxReadRequest` is unique because it can be changed during runtime with the `setpci` command. If your value here is lower than expected, you can correct it as follows:

Shell output

```
$ sudo lspci -vvvs a1:00.0 | grep axReadReq

MaxPayload 512 bytes, MaxReadReq 512 bytes

$ sudo setpci -s a1:00.0 68.w

295e

$ sudo setpci -s a1:00.0 68.w=595e

$ sudo lspci -vvvs a1:00.0 | grep axReadReq

MaxPayload 512 bytes, MaxReadReq 4096 bytes
```

Commands

```
sudo lspci -vvvs a1:00.0 | grep axReadReq

sudo setpci -s a1:00.0 68.w

sudo setpci -s a1:00.0 68.w=595e

sudo lspci -vvvs a1:00.0 | grep axReadReq
```

Note

Changes made with `setpci` are not persistent across reboots. This example uses a single NIC for simplicity, but in practice you must run the change for each NIC in the node.

2.3 Validate NIC configuration

After you've verified optimal PCIe speeds for all devices, configure your NICs according to best practices in the manufacturer or vendor documentation. This might already include some of the pre-assessment steps outlined in this guide and provide more hardware-specific tuning optimizations.

2.3.1 Vendor-specific NIC tuning

Your NICs may require tuning if it has not already been done. Some steps differ based on the type of NIC you're deploying (InfiniBand or RoCE).

- Ensure ACS is disabled manually or by running the ACS disable script.

Note

ACS may need to be disabled in BIOS settings as well, if supported by your hardware.

- For Mellanox NICs (InfiniBand or RoCE): Enable PCIe relaxed ordering, increase max read requests, enable advanced PCI settings.

```
sudo mst start

sudo mst status

sudo mlxconfig -d /dev/mst/mt4123_pciconf0 s ADVANCED_PCI_SETTINGS=1

sudo mlxconfig -d /dev/mst/mt4123_pciconf0 s MAX_ACC_OUT_READ=128

sudo mlxconfig -d /dev/mst/mt4123_pciconf0 s PCI_WR_ORDERING=1

reboot
```

- For Broadcom NICs: enable PCIe relaxed ordering, RDMA support, RoCE performance profile, set speed mask exclude to 200G, and consider disabling any unused ports. For assistance, use the scripts available on the cluster networking github for each configuration setting.
- Ensure relaxed ordering is enabled in the PCIe settings for your system BIOS as well.

> **Note**
>
> All instructions for RoCE networks in this guide and additional guides are based on the v2 protocol.

2.3.2 Check NIC link speed

Verify the NICs in your servers are reporting the correct speeds. Several commands and utilities are available to measure speed based on your network type.

- **RoCE / Ethernet**
  - `sudo ethtool <interface> | grep -i speed`
  - `cat /sys/class/net/<interface>/speed`

> **Note**
>
> An ethtool script is available for configuring a desired speed (default target 200G) on all available nodes.

- **InfiniBand**
  - `ibdiagnet` provides an output of the entire fabric in the default log files. You can verify link speeds here.
  - `ibstat` or `ibstatus` tells you if the link is up and the speed at which it is running for all HCAs in the server.

2.3.3 Verify Mellanox OFED and firmware installation

> **Note**
>
> This step is only necessary for InfiniBand networks.

Download the latest version of Mellanox OFED (MLNX_OFED) from NVIDIA. Run the installer and flint tools to verify the latest version of MLNX_OFED and firmware is on the HCAs.

2.4 Set up a GPU testing environment

Next, create a testing environment to gather performance data for your GPUs. This requires installation of ROCm Validation Suite (RVS), TransferBench, and ROCm Bandwidth Test.

1. Connect to the CLI of your GPU node.
2. Install ROCm Validation Suite following the directions at Installing ROCm Validation Suite.
   - Once installed, RVS is located in `/opt/rocm/`.
3. Install TransferBench. Refer to Installing TransferBench for details.

```
$ git clone https://github.com/ROCm/TransferBench.git

$ cd TransferBench

$ sudo make
```

(continues on next page)

(continued from previous page)

```
# Running make without sudo seems to cause runtime issues
# If this doesn't work, install math libraries manually using https://github.com/
↪ROCm/ROCm/issues/1843

$ sudo apt install libstdc++-12-dev
```

4. Install ROCm Bandwidth Test. Refer to Installing ROCm Bandwidth Test for details.

```
$ sudo apt install rocm-bandwidth-test
```

2.4.1 Run ROCm Validation Suite (RVS)

RVS contains many different tests, otherwise referred to as modules. The relevant tests for this guide are as follows:

- P2P Benchmark and Qualification Tool (PBQT)
- ROCm Configuration Qualification Tool (RCQT)
- PCI Express Bandwidth Benchmark (PEBB)
- GPU Properties (GPUP)
- GPU Stress test (GST)

You can run multiple tests at once with `sudo /opt/rocm/rvs/rvs -d 3`, which runs all tests set in `/opt/rocm/share/rocm-validation-suite/rvs.conf` at verbosity level 3. The default tests are GPUP, PEQT, PEBB, and PBQT, but you can modify the config file to add your preferred tests. The RVS documentation has more information on how to modify `rvs.conf` and helpful command line options.

Tip

When you identify a problem, use `rvs -g` to understand what the GPU ID is referring to.

GPU numbering in RVS does not have the same order as in `rocm-smi`. To map the GPU order listed in `rvs-g` to the rocm output, run `rocm-smi --showbus` and match each GPU by bus ID.

You can run a specific RVS test by calling its configuration file with `sudo /opt/rocm/bin/rvs -c /opt/rocm/share/rocm-validation-suite/conf/<test name>.conf`. The following shell examples demonstrate what the commands and outputs look like for some of these tests.

2.4.1.1 Example of GPU stress tests with the GST module

Shell output

```
$ sudo /opt/rocm/bin/rvs -c /opt/rocm/share/rocm-validation-suite/conf/gst_single.conf

[RESULT] [508635.659800] Action name :gpustress-9000-sgemm-false
[RESULT] [508635.660582] Module name :gst
[RESULT] [508642.648770] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508643.652155] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508644.657965] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
```

(continues on next page)

(continued from previous page)

```
[RESULT] [508646.633979] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508647.641379] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508648.649070] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508649.657010] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508650.665296] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output>
[RESULT] [508655.632843] [gpustress-9000-sgemm-false] gst <GPU ID> GFLOPS <performance␣
↪output> Target stress : <stress value> met :TRUE
```

Commands

```
sudo /opt/rocm/bin/rvs -c /opt/rocm/share/rocm-validation-suite/conf/gst_single.conf
```

2.4.1.2 Example of PCIe bandwidth benchmarks with the PBQT module

Shell output

```
$ sudo /opt/rocm/rvs/rvs -c /opt/rocm/share/rocm-validation-suite/conf/pbqt_single.conf -
↪d 3

[RESULT] [1148200.536604] Action name :action_1

            Discovered Nodes
=============================================

Node Name                                                                Node Type        ␣
↪       Index      GPU ID
=============================================================================================================================
<CPU1>                                                                      CPU           ␣
↪       0          N/A

<CPU2>                                                                      CPU           ␣
↪       1          N/A

<CPU3>                                                                      CPU           ␣
↪       2          N/A

<CPU4>                                                                      CPU           ␣
↪       3          N/A

<GPU1>                                                                      GPU           ␣
↪       4          <GPU1-ID>

<GPU2>                                                                      GPU           ␣
↪       5          <GPU2-ID>
=============================================================================================================================
[RESULT] [1148200.576371] Module name :pbqt
[INFO  ] [1148200.576394] Missing 'device_index' key.
```

(continues on next page)

(continued from previous page)

```
[RESULT] [1148200.576498] [action_1] p2p <GPU1> <GPU2> peers:true distance:72 PCIe:72
[RESULT] [1148205.576740] [action_1] p2p-bandwidth  [1/1] <GPU1> <GPU2>  bidirectional:␣
↪true  <result> GBps  duration: <result> sec
[RESULT] [1148205.577850] Action name :action_2
[RESULT] [1148205.577862] Module name :pbqt
[INFO  ] [1148205.577883] Missing 'device_index' key.
[RESULT] [1148205.578085] [action_2] p2p <GPU1> <GPU2> peers:true distance:72 PCIe:72
[INFO  ] [1148216.581794] [action_2] p2p-bandwidth  [1/1] <GPU1> <GPU2>  bidirectional:␣
↪true  <result> GBps
[INFO  ] [1148217.581371] [action_2] p2p-bandwidth  [1/1] <GPU1> <GPU2>  bidirectional:␣
↪true  <result> GBps
[INFO  ] [1148218.580844] [action_2] p2p-bandwidth  [1/1] <GPU1> <GPU2>  bidirectional:␣
↪true  <result> GBps
[INFO  ] [1148219.580909] [action_2] p2p-bandwidth  [1/1] <GPU1> <GPU2>  bidirectional:␣
↪true  <result> GBps
```

Commands

```
sudo /opt/rocm/rvs/rvs -c /opt/rocm/share/rocm-validation-suite/conf/pbqt_single.conf -d␣
↪3
```

2.4.2 Run TransferBench

TransferBench is a benchmarking tool designed to measure simultaneous data transfers between CPU and GPU devices. To use it, first navigate to the TransferBench installation directory. Then, execute the following command to display available commands, flags, and an overview of your system's CPU/GPU topology as detected by TransferBench:

```
./TransferBench
```

Like RVS, TransferBench operates based on configuration files. You can either choose from several preset configuration files or create a custom configuration to suit your testing needs. A commonly recommended test is the `p2p` (peer-to-peer) test, which measures unidirectional and bidirectional transfer rates across all CPUs and GPUs detected by the tool. The following example shows the output of a `p2p` test on a system with 2 CPUs and 8 GPUs, using 4 MB transfer packets.

Shell output

```
$ ./TransferBench p2p 4M

TransferBench v1.50
========================================================================
[Common]
ALWAYS_VALIDATE      =            0 : Validating after all iterations
<SNIP>......
Bytes Per Direction 4194304
Unidirectional copy peak bandwidth GB/s [Local read / Remote write] (GPU-Executor: GFX)
   SRC+EXE\DST    CPU 00    CPU 01       GPU 00    GPU 01    GPU 02    GPU 03    GPU 04 ␣
↪  GPU 05    GPU 06    GPU 07
   CPU 00  ->     24.37     25.62        17.32     16.97     17.33     17.47     16.77  ␣
↪  17.12     16.91     16.96
   CPU 01  ->     18.83     19.62        14.84     15.47     15.16     15.13     16.11  ␣
```

(continues on next page)

(continued from previous page)

```
↪  16.13     16.01     15.91

   GPU 00  ->     23.83     23.40       108.95     64.58     31.56     28.39     28.44  ␣
↪  26.99     47.46     39.97
   GPU 01  ->     24.05     23.93        66.52    109.18     29.07     32.53     27.80  ␣
↪  31.73     40.79     36.42
   GPU 02  ->     23.83     23.47        31.48     28.58    109.45     65.11     47.40  ␣
↪  40.11     28.45     27.46
   GPU 03  ->     24.35     23.93        28.65     32.00     65.68    108.68     39.85  ␣
↪  36.08     27.08     31.49
   GPU 04  ->     23.30     23.84        28.57     26.93     47.36     39.77    110.94  ␣
↪  64.66     31.14     28.15
   GPU 05  ->     23.39     24.08        27.19     31.26     39.85     35.49     64.98  ␣
↪ 110.10     28.57     31.43
   GPU 06  ->     23.43     24.03        47.58     39.22     28.97     26.93     31.48  ␣
↪  28.41    109.78     64.98
   GPU 07  ->     23.45     23.94        39.70     35.50     27.08     31.25     28.14  ␣
↪  32.19     65.00    110.47
                             CPU->CPU  CPU->GPU  GPU->CPU  GPU->GPU
   Averages (During UniDir):     22.23     16.35     23.77     37.74

Bidirectional copy peak bandwidth GB/s [Local read / Remote write] (GPU-Executor: GFX)
   SRC\DST    CPU 00    CPU 01       GPU 00    GPU 01    GPU 02    GPU 03    GPU 04  ␣
↪GPU 05    GPU 06    GPU 07
   CPU 00  ->       N/A     17.07        16.90     17.09     15.39     17.07     16.62  ␣
↪  16.65     16.40     16.32
   CPU 00 <-        N/A     13.90        24.06     24.03     24.00     24.21     23.09  ␣
↪  23.14     22.11     22.15
   CPU 00 <->       N/A     30.97        40.96     41.12     39.39     41.28     39.71  ␣
↪  39.80     38.51     38.47

   CPU 01  ->     12.85       N/A        15.29     15.14     15.03     15.16     15.95  ␣
↪  15.62     16.06     15.85
   CPU 01 <-      17.34       N/A        22.95     23.18     22.98     22.92     23.86  ␣
↪  24.05     23.94     23.94
   CPU 01 <->     30.19       N/A        38.24     38.32     38.01     38.08     39.80  ␣
↪  39.67     40.00     39.79


   GPU 00  ->     23.99     22.94          N/A     62.40     30.30     25.15     25.00  ␣
↪  25.20     46.58     37.99
   GPU 00 <-      16.87     14.75          N/A     65.21     31.10     25.91     25.53  ␣
↪  25.48     47.34     38.17
   GPU 00 <->     40.85     37.69          N/A    127.61     61.40     51.06     50.53  ␣
↪  50.68     93.91     76.16

   GPU 01  ->     24.11     23.20        65.10       N/A     25.88     31.74     25.66  ␣
↪  31.01     39.37     34.75
   GPU 01 <-      17.00     14.08        61.91       N/A     26.09     31.90     25.73  ␣
↪  31.34     38.97     34.76
   GPU 01 <->     41.11     37.29       127.01       N/A     51.97     63.64     51.39  ␣
↪  62.35     78.35     69.51
```

(continues on next page)

(continued from previous page)

```
   GPU 02  ->     23.89     22.78        30.94     26.39       N/A     62.22     45.73  ␣
↪  38.40     25.95     25.26
   GPU 02 <-      16.59     13.91        30.47     26.54       N/A     63.63     47.42  ␣
↪  38.68     26.29     25.64
   GPU 02 <->     40.48     36.69        61.42     52.93       N/A    125.85     93.15  ␣
↪  77.08     52.24     50.90

   GPU 03  ->     24.15     22.98        25.84     31.69     64.03       N/A     38.82  ␣
↪  35.12     25.46     30.82
   GPU 03 <-      17.22     14.19        25.28     31.16     61.90       N/A     38.16  ␣
↪  34.85     25.81     30.97
   GPU 03 <->     41.37     37.16        51.12     62.84    125.93       N/A     76.99  ␣
↪  69.97     51.27     61.79

   GPU 04  ->     23.12     23.73        25.50     25.40     47.04     38.29       N/A  ␣
↪  62.44     30.56     25.15
   GPU 04 <-      16.15     12.86        25.13     25.63     46.38     38.65       N/A  ␣
↪  63.89     30.88     25.74
   GPU 04 <->     39.27     36.58        50.63     51.03     93.42     76.94       N/A  ␣
↪ 126.34     61.43     50.89

   GPU 05  ->     23.09     24.04        25.61     31.29     38.82     34.96     63.55  ␣
↪    N/A     25.87     30.35
   GPU 05 <-      13.65     15.46        25.26     30.87     38.51     34.70     61.57  ␣
↪    N/A     26.34     31.47
   GPU 05 <->     36.75     39.50        50.87     62.16     77.32     69.66    125.12  ␣
↪    N/A     52.21     61.82

   GPU 06  ->     22.09     23.73        47.51     38.56     26.15     25.59     31.32  ␣
↪  25.98       N/A     62.34
   GPU 06 <-      16.31     15.40        46.22     39.16     25.63     25.17     30.44  ␣
↪  25.58       N/A     63.88
   GPU 06 <->     38.39     39.13        93.72     77.72     51.78     50.76     61.76  ␣
↪  51.56       N/A    126.22

   GPU 07  ->     22.31     23.88        38.68     34.96     25.54     30.96     25.79  ␣
↪  31.28     63.69       N/A
   GPU 07 <-      16.27     15.89        38.39     35.06     25.27     30.62     25.25  ␣
↪  30.91     62.36       N/A
   GPU 07 <->     38.58     39.77        77.07     70.02     50.81     61.58     51.05  ␣
↪  62.20    126.04       N/A
                            CPU->CPU  CPU->GPU  GPU->CPU  GPU->GPU
Averages (During  BiDir):     15.29     19.72     19.39     36.17
```

Commands

```
./TransferBench p2p 4M
```

If you want to define your own configuration file, run `cat ~/TransferBench/examples/example.cfg` to view an example configuration file with information on commands and arguments to run more granular testing. Running DMA tests between single pairs of devices is one helpful and common use case for custom configuration files. See the

TransferBench documentation for more information.

2.4.3 Run ROCm Bandwidth Test (RBT)

ROCm Bandwidth Test lets you identify performance characteristics for host-to-device (H2D), device-to-host (D2H), and device-to-device (D2D) buffer copies on a ROCm platform. This assists when looking for abnormalities and tuning performance.

Run `/opt/rocm/bin/rocm-bandwidth-test -h` to get a help screen with available commands.

```
$ /opt/rocm/bin/rocm-bandwidth-test -h

Supported arguments:

        -h    Prints the help screen
        -q    Query version of the test
        -v    Run the test in validation mode
        -l    Run test to collect Latency data
        -c    Time the operation using CPU Timers
        -e    Prints the list of ROCm devices enabled on platform
        -i    Initialize copy buffer with specified 'long double' pattern
        -t    Prints system topology and allocatable memory info
        -m    List of buffer sizes to use, specified in Megabytes
        -b    List devices to use in bidirectional copy operations
        -s    List of source devices to use in copy unidirectional operations
        -d    List of destination devices to use in unidirectional copy operations
        -a    Perform Unidirectional Copy involving all device combinations
        -A    Perform Bidirectional Copy involving all device combinations

        NOTE: Mixing following options is illegal/unsupported
              Case 1: rocm_bandwidth_test -a with {lm}{1,}
              Case 2: rocm_bandwidth_test -b with {clv}{1,}
              Case 3: rocm_bandwidth_test -A with {clmv}{1,}
              Case 4: rocm_bandwidth_test -s x -d y with {lmv}{2,}
```

The default behavior of `/opt/rocm/bin/rocm-bandwidth-test` without any flags runs unilateral and bilateral benchmarks (flags `-a` and `-A`) on all available combinations of device. Review the following for examples of common commands and output.

Getting a list of all ROCm-detected devices:

Shell output

```
$ /opt/rocm/bin/rocm-bandwidth-test -e

RocmBandwidthTest Version: 2.6.0

   Launch Command is: /opt/rocm/bin/rocm-bandwidth-test -e


   Device Index:                             0
   Device Type:                            CPU
   Device Name:                            <CPU Name>
      Allocatable Memory Size (KB):         1044325060
```

(continues on next page)

(continued from previous page)

```
  Device Index:                             1
  Device Type:                              CPU
  Device Name:                              <CPU Name>
     Allocatable Memory Size (KB):          1056868156

  Device Index:                             2
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480

  Device Index:                             3
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480

  Device Index:                             4
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480

  Device Index:                             5
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480

  Device Index:                             6
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480

  Device Index:                             7
  Device Type:                              GPU
  Device Name:                              <GPU Name>
  Device  BDF:                              XX:0.0
  Device UUID:                              GPU-0000
     Allocatable Memory Size (KB):          67092480
     Allocatable Memory Size (KB):          67092480
```

(continues on next page)

(continued from previous page)

```
   Device Index:                             8
   Device Type:                            GPU
   Device Name:                            <GPU Name>
   Device  BDF:                            XX:0.0
   Device UUID:                            GPU-0000
      Allocatable Memory Size (KB):         67092480
      Allocatable Memory Size (KB):         67092480

   Device Index:                             9
   Device Type:                            GPU
   Device Name:                            <GPU Name>
   Device  BDF:                            XX:0.0
   Device UUID:                            GPU-0000
      Allocatable Memory Size (KB):         67092480
      Allocatable Memory Size (KB):         67092480
```

Commands

```
/opt/rocm/bin/rocm-bandwidth-test -e
```

Running a unidirectional benchmark between devices 0 (CPU) and 4 (GPU):

Shell output

```
$ /opt/rocm/bin/rocm-bandwidth-test -s 0 -d 4
.....................................
         RocmBandwidthTest Version: 2.6.0

         Launch Command is: /opt/rocm/bin/rocm-bandwidth-test -s 0 -d 4


================    Unidirectional Benchmark Result    ================
================ Src Device Id: 0 Src Device Type: Cpu ================
================ Dst Device Id: 4 Dst Device Type: Gpu ================

Data Size      Avg Time(us)   Avg BW(GB/s)   Min Time(us)   Peak BW(GB/s)
1 KB           5.400          0.190          5.280          0.194
2 KB           5.360          0.382          5.280          0.388
4 KB           5.440          0.753          5.440          0.753
8 KB           5.440          1.506          5.440          1.506
16 KB          5.880          2.786          5.760          2.844
32 KB          6.400          5.120          6.400          5.120
64 KB          7.520          8.715          7.520          8.715
128 KB         9.920          13.213         9.920          13.213
256 KB         14.520         18.054         14.400         18.204
512 KB         23.560         22.253         23.520         22.291
1 MB           41.880         25.038         41.760         25.110
2 MB           78.400         26.749         78.400         26.749
4 MB           153.201        27.378         152.641        27.478
8 MB           299.641        27.996         299.521        28.007
16 MB          592.002        28.340         592.002        28.340
```

(continues on next page)

(continued from previous page)

```
32 MB          1176.925       28.510         1176.805       28.513
64 MB          2346.730       28.597         2346.730       28.597
128 MB         4686.180       28.641         4686.100       28.642
256 MB         9365.280       28.663         9365.160       28.663
512 MB         18722.762      28.675         18722.482      28.675
```

Commands

```
/opt/rocm/bin/rocm-bandwidth-test -s 0 -d 4
```

Running a bidirectional benchmark on all available device combinations:

Shell output

```
$ /opt/rocm/bin/rocm-bandwidth-test -A

<SNIP>......
Bidirectional copy peak bandwidth GB/s

     D/D       0           1           2           3           4           5           ␣
↪6           7           8           9

     0         N/A         N/A         47.703      47.679      47.619      47.586      ␣
↪38.106      38.160      36.771      36.773

     1         N/A         N/A         38.351      38.395      36.488      36.454      ␣
↪47.495      47.512      47.525      47.471

     2         47.703      38.351      N/A         101.458     80.902      81.300      ␣
↪81.387      79.279      101.526     101.106

     3         47.679      38.395      101.458     N/A         81.278      80.488      ␣
↪79.535      79.907      101.615     101.618

     4         47.619      36.488      80.902      81.278      N/A         101.643     ␣
↪101.089     101.693     81.336      79.232

     5         47.586      36.454      81.300      80.488      101.643     N/A         ␣
↪101.217     101.478     79.460      79.922

     6         38.106      47.495      81.387      79.535      101.089     101.217     ␣
↪N/A         101.506     80.497      81.302

     7         38.160      47.512      79.279      79.907      101.693     101.478     ␣
↪101.506     N/A         81.301      80.501

     8         36.771      47.525      101.526     101.615     81.336      79.460      ␣
↪80.497      81.301      N/A         100.908

     9         36.773      47.471      101.106     101.618     79.232      79.922      ␣
↪81.302      80.501      100.908     N/A
```

Commands

```
/opt/rocm/bin/rocm-bandwidth-test -A
```

For a more detailed explanation of different ways to run ROCm Bandwidth Test, see the ROCm Bandwidth Test user guide.

CHAPTER

THREE

MULTI-NODE NETWORK CONFIGURATION FOR AMD INSTINCT ACCELERATORS

After single node configuration testing has been completed and verified, validate network connections in node pairs. All the tests described in this topic must be run between two nodes in a client-server relationship. Both nodes must be configured and verified according to *Single-node network configuration for AMD Instinct accelerators* before running any node-to-node performance tests.

3.1 Prerequisites

Before following the steps in this guide, complete the following prerequisites.

- Install all required software for MPI in *GPU-enabled Message Passing Interface*.
  - Specifically, follow the installation instructions for Open MPI, OSU benchmarks, and collective operations.
- Install Slurm Workload Manager (if applicable). Refer to the Slurm Workload Manager documentation.
- Implement passwordless SSH.

3.1.1 Evaluate platform-specific BIOS tunings

Check your BIOS settings to make sure they are optimized for AMD GPUs. See the AMD Instinct system optimization guides for more information.

- Enable large bar addressing in the BIOS to support peer to peer GPU memory access.
- Verify SR-IOV is enabled, if needed.
- Disable ACS (ACS forces P2P transactions through the PCIe root complex).

Note

If using virtual devices, AER and ACS should be enabled.

Important

You must still run the disable ACS script prior to running any workloads, as the BIOS setting (if present) may not disable all ACS configurations at the OS level.

3.1.2 Single tier switch configuration

Take these actions on each single tier (leaf/edge) switch you plan to include in network testing.

1. Configure remote access to the switch management console.
2. Verify the switch sees all hosts and ports are active.
3. For an InfiniBand switch, configure Fabric Manager on the switch or start OpenSM on a host in the network if a subnet manager isn't already in place.
4. For an ethernet switch, configure MTU size and priority flow control (PFC) and ECN support as needed.
5. Clear all port counters after the switch is ready to use.

3.2 OFED perftest installation and benchmarking

Install and run the OFED performance tests for host to host (H2H) testing. Loopback is implemented in the tests to remove the switch from benchmark results. Remember to install OFED perftests on both nodes you plan to use in this section. Commands may require `sudo` depending on user privileges.

1. From the CLI of your host, clone the perftest repository.

```
git clone https://github.com/linux-rdma/perftest.git
```

2. Navigate to the installation directory and build the tests.

```
cd perftest
./autogen.sh
./configure --prefix=$PWD/install --enable-rocm --with-rocm=/opt/rocm
```

3. Locate and open `Makefile` in your editor of choice, then append `-D__HIP_PLATFORM_AMD__` to `CFLAGS` and `CXXFLAGS`. This is required to compile the code correctly for this guide.
4. Run `make && make install`.
5. Repeat these steps on a second node connected to the same switch.

3.3 Run host-based (CPU) performance tests

Once installed, there are six main modules available with OFED perftests:

- `ib_write_bw` - Test bandwidth with RDMA write transactions.
- `ib_write_lat` - Test latency with RDMA write transactions.
- `ib_read_bw` - Test bandwidth with RDMA read transactions.
- `ib_read_lat` - Test latency with RDMA read transactions.
- `ib_send_bw` - Test bandwidth with send transactions.
- `ib_send_lat` - Test latency with send transactions.

The examples in this section use the `ib_send_bw` tool, but you can achieve similar results with other benchmarking tools, depending on your requirements. The primary objective of these tests is to verify high-speed Host-to-Host (H2H) data transfer rates between nodes before introducing GPU traffic–as a result, the `use_rocm` flag is intentionally omitted from all commands.

3.3.1 Run H2H RDMA benchmark

To run the OFED perftest, establish an SSH connection to both nodes you installed the OFED perftests on.

1. Initiate a server connection on the first node:

```
$ cd perftest   #if not already in directory

$ numactl -C 1 ./ib_send_bw -a -F -d <IB/RoCE interface>

************************************
* Waiting for client to connect... *
************************************
```

2. Initiate a client connection on the second node:

```
$ cd perftest   #if not already in directory

$ numactl -C 1 ./ib_send_bw <node1 IP> -a -F -d <IB/RoCE interface>
```

3. Test should run and complete in several moments.

> **Note**
>
> The use of `numactl` or `taskset` commands makes sure NUMA domains are not crossed when communicating, which can create overhead and latency. When running tests you must ensure you use cores local to the network device.

Consult this table for an explanation of flags used in the `numactl` examples and other optional flags that may be useful for you.

-d <IB/RoCE interface> Specifies a NIC to use. Ensure you use a NIC that is both adjacent to a GPU and not crossing NUMA domains or otherwise needing pass traffic between CPUs before egressing from the host. Tools like `rocm-smi --showtopo` and `lstopo` can help define which NICs are adjacent to which GPUs.

-p <port #> Assign a port number to the server/client. Each instance must run on a different port when executed simultaneously.

--report_gbits Reports in Gb/s instead of Mb/s.

-m <mtu> Set MTU size.

-b Bidirectional runs.

-a Runs messages in all sizes.

-n <number> Provides the number of iterations.

-F Do not show warning if cpufreq_ondemand is loaded.

--use_rocm=<rocm_device_number> This is for device testing, allows you to specify which GPU to use. Zero-based numbering.

--perform_warm_up Runs several iterations before benchmarking to warm up memory cache.

As servers typically have one NIC per GPU, you must change the device location frequently as you iterate through tests.

3.3.2 Run multithreaded H2H RDMA benchmark

To perform a multithreaded RDMA benchmark using the OFED perftest, run it concurrently on each NIC in the server. Use the `taskset` command to assign a CPU core within the same NUMA domain as the NICs. While testing the XGMI/Infinity Fabric link between CPUs is not required at this stage, it can be an optional test if desired.

3.3.3 Run extended multithreaded H2H RDMA benchmark

Repeat the multithreaded RDMA benchmark, but loop the test and run it continuously for at least 8 hours. This extended test is designed to stress the I/O network fabric over a prolonged period to assess stability and performance under sustained load.

3.4 Run device-based (GPU) OFED performance tests

After confirming Host-to-Host (H2H) performance, proceed to run Device-to-Device (D2D) OFED perftests, which include GPU traffic. This will evaluate RDMA performance between GPUs.

3.4.1 Run D2D RDMA benchmark

To run a D2D RDMA benchmark, use the following example setup to test GPU pairs–for example, GPU0 to GPU1, GPU2 to GPU3.

> **Note**
>
> If you have Mellanox or NVIDIA NICs, be aware that the default OFED perftest installation doesn't include ROCm support. Follow the *installation instructions* if you haven't done so already.

In this example, `localhost` is used by the client to call the server. You may use a specific IP address to ensure the network is tested.

```
$ (ib_write_bw -b -a -d <RDMA-NIC-1> --report_gbits -F -use_rocm=0 >> /dev/null &);␣
↪sleep 1; ib_write_bw -b -a -d <RDMA-NIC-2> --report_gbits -use_rocm=0 -F localhost
---------------------------------------------------------------------------------------
                 RDMA_Write Bidirectional BW Test
Dual-port       : OFF          Device         : <RDMA-NIC-2>
Number of qps   : 1            Transport type : IB
Connection type : RC           Using SRQ      : OFF
PCIe relax order: ON
ibv_wr* API     : OFF
TX depth        : 128
CQ Moderation   : 100
Mtu             : 4096[B]
Link type       : Ethernet
GID index       : 3
Max inline data : 0[B]
rdma_cm QPs     : OFF
Data ex. method : Ethernet
---------------------------------------------------------------------------------------
local address: LID 0000 QPN 0x0901 PSN 0x5e30c8 RKey 0x2000201 VAddr 0x007fe663d20000
GID: 00:00:00:00:00:00:00:00:00:00:255:255:01:01:101:45
remote address: LID 0000 QPN 0x0901 PSN 0xf40c3c RKey 0x2000201 VAddr 0x007f282a06e000
GID: 00:00:00:00:00:00:00:00:00:00:255:255:01:01:101:35
```

(continues on next page)

(continued from previous page)

```
---------------------------------------------------------------------------------------
#bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
2          5000           0.142947            0.012281            0.767588
4          5000             0.28              0.26                8.255475
8          5000             0.55              0.54                8.471791
16         5000             1.16              1.16                9.025968
32         5000             2.31              2.27                8.865877
64         5000             4.49              4.43                8.647051
128        5000             8.98              8.96                8.745890
256        5000             17.57             16.32               7.969287
512        5000             34.63             34.41               8.400441
1024       5000             67.22             66.92               8.168969
2048       5000             129.04            126.20              7.702863
4096       5000             188.76            188.56              5.754307
8192       5000             194.79            192.62              2.939080
16384      5000             195.32            195.21              1.489355
32768      5000             203.15            203.13              0.774887
65536      5000             204.12            203.85              0.388818
131072     5000             204.44            204.43              0.194964
262144     5000             204.51            204.51              0.097517
524288     5000             204.56            204.56              0.048770
1048576    5000             204.57            204.57              0.024387
2097152    5000             204.59            204.59              0.012194
4194304    5000             204.59            204.59              0.006097
8388608    5000             204.59            204.59              0.003049
---------------------------------------------------------------------------------------
```

Note

If you run the test with different values for `--use_rocm=#` on the server and the client, the output will show results from whichever GPU is local to the node you're looking at. The tool is unable to show server and client simultaneously.

3.4.2 Run H2D/D2H RDMA benchmark

This is similar to the D2D test, but also includes the CPU on either the server or client side of the test-case scenarios.

For a 2-CPU/8-GPU node you would have 32 test scenarios per pairs of server.

Table 1: H2D/D2H Benchmark with Server-Side CPUs

Client	GPU 0	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7
Server	CPU 0	CPU 1						

Table 2: H2D/D2H Benchmark with Client-Side CPUs

Server	GPU 0	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7
Client	CPU 0	CPU 1						

To run this test, use a command similar to the example in the D2D benchmark, but only add the `--use_rocm` flag on

either the server or client side so that one node communicates with the GPUs while the other does so with CPUs. Then, run the test a second time with the `use_rocm` flag on the other side. Continue to use the most adjacent NIC to the GPU or CPU being tested so that communication doesn't run between intra-node CPUs (testing the internal CPU-CPU fabric isn't a goal now).

3.4.3 D2D RDMA multithread benchmark

For this test you must run the previous D2D benchmark simultaneously on all GPUs. Scripting is required to accomplish this, but the command input should resemble something like the following image with regard to your RDMA device naming scheme.

```
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16001  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16002  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16003  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16004  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16005  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16006  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16007  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
taskset -c 0-63,128-191 /home/AMD/perftest/ib_write_bw -S 0 -p 16008  --report_gbits -d mlx5_0 -m 4096 -n 10000 -x 3  --use_rocm=0
```

Important OFED perftest flags for this effort include:

-p <port#> Lets you assign specific ports for server/client combinations. Each pair needs an independent port number so you don't inadvertently use the wrong server.

-n <# of iterations> Default is 1000, you can increase this to have the test run longer.

For bandwidth tests only:

-D <seconds> Defines how long the test runs for.

--run_infinitely Requires user to break the runtime, otherwise runs indefinitely.

3.4.4 D2D RDMA multithread extended benchmark

Perform the D2D RDMA multithread benchmark again but set the duration for a minimum of 8 hours.

3.5 Build collective tests

This section guides you through setting up the remaining tools necessary to simulate an AI workload on your GPU nodes after they have been sufficiently traffic-tested. Per the *prerequisites*, UCX, UCC, MPI and the OSU benchmarks must already be installed.

3.5.1 Install RCCL

RCCL is likely already installed as part of ROCm on your compute nodes. Sometimes newer features and fixes might be available in the latest version of RCCL, which you can build from source at https://github.com/ROCm/rccl.

3.5.2 Build RCCL collective tests

To more easily build and run the RCCL collective tests, review and implement the rccl test install script. Otherwise, you can follow the steps to manually install at https://github.com/ROCm/rccl-tests.

3.6 Run RCCL benchmarks

ROCm Communication Collectives Library (RCCL) is a set of collective operations that perform multi-GPU and multi-node communication over a network. These operations are `AllReduce`, `AllGather`, `AlltoAll`, `Broadcast`, `ReduceScatter`, `Reduce`, `Scatter`, and `Gather`, implemented as ring or tree algorithms. The **collective** descriptor

for these operations means they can support multiple devices (GPUs) in a single run. As RCCL is specifically optimized for AMD GPUs, it's the standard by which performance can be tested and measured on cluster deployments.

Communication between GPUs is handled over PCIe and Infinity Fabric (XGMI) interconnects within an individual node, while communication from node to node can run on RoCE, InfiniBand, or TCP/IP cluster networks.

Although a version of RCCL is included with all ROCm installations, it is a standalone library that can be installed apart from ROCm as well.

> **Note**
>
> The paths for the MPI and RCCL commands in this section presume both are installed in the `/opt` directory. Installation paths for your environment may be different and should be updated accordingly.

3.6.1 Using MPI to run RCCL-test

You can use `mpirun` to initiate a RCCL operation from the command line. The command structure is described as follows:

`/path/to/mpirun <MPI parameters> /path/to/rccl-operation <RCCL parameters>`

To apply this model to an `AllReduce` run on a single node with 8 GPUs:

```
/opt/ompi/bin/mpirun -np 8 --bind-to numa /opt/rccl-tests/build/all_reduce_perf -b 8 -e␣
↪16G -f 2 -g 1
```

This command runs 8 MPI processes (`np -8`), binding each process to a unique GPU (`--bind-to numa`, `-g 1`) and scanning from 8 bytes to 16 gigabytes (`-b 8 -e 16G`).

Further examples in this guide continue to use `AllReduce`, but you can run any of the other operations by editing the path to point at your desired RCCL test.

3.6.2 Multi-node RCCL operations

> **Note**
>
> To successfully run multi-node RCCL, all nodes you plan to test must be configured with passwordless SSH, otherwise the runs fail.

To run a RCCL test between two nodes, adjust the previous command as follows:

```
/opt/ompi/bin/mpirun -host <node01>:8,<node02>:8 -np 16 -x NCCL_IB_HCA=<nic01>,<nic02>,
↪<nic03>,<nic04>,<nic05>,<nic06>,<nic07>,<nic08> /opt/rccl-tests/build/all_reduce_perf -
↪b 8 -e 16G -f 2 -g 1
```

The `host` parameter defines nodes to include in the run, where `<node01>` and `<node02>` are the respective IP or DNS addresses for those nodes, and `:8` represents the number of mpi processes (defined by `np`) allocated to each node. The value for `np` is equal to the total number of GPUs included in the RCCL run across all nodes, which should also be equal to the total number of allocated processes per node (assuming each node has 8 GPUs). NICs included in the run are defined in `NCCL_IB_HCA`, where `<nic01>,<nic02>..` are the RDMA device names for each NIC (`bnxt_re0, bnxt_re1..` for Broadcom devices and `mlx5_0,mlx5_2,..` for Mellanox devices).

You can scale this command to more nodes by incrementing the values for `host` and `np` accordingly. Alternatively, you can create a file with a list of hosts and use the `--hostfile` option in place of `host` if you have a larger number of nodes to test. For an explanation of how to format and invoke a hostfile, refer to the Open MPI Documentation.

4-node RCCL test

```
/opt/ompi/bin/mpirun -host <node01>:8,<node02>:8,<node03>:8,<node04>:8 -np 32 -x NCCL_IB_
↪HCA=<nic01>,<nic02>,<nic03>,<nic04>,<nic05>,<nic06>,<nic07>,<nic08> /opt/rccl-tests/
↪build/all_reduce_perf -b 8 -e 16G -f 2 -g 1
```

8-node RCCL test

```
/opt/ompi/bin/mpirun -host <node01>:8,<node02>:8,<node03>:8,<node04>:8,<node05>:8,
↪<node06>:8,<node07>:8,<node08>:8 -np 64 -x NCCL_IB_HCA=<nic01>,<nic02>,<nic03>,<nic04>,
↪<nic05>,<nic06>,<nic07>,<nic08> /opt/rccl-tests/build/all_reduce_perf -b 8 -e 16G -f 2␣
↪-g 1
```

16-node RCCL test

```
/opt/ompi/bin/mpirun -host <node01>:8,<node02>:8,<node03>:8,<node04>:8,<node05>:8,
↪<node06>:8,<node07>:8,<node08>:8,<node09>:8,<node10>:8,<node11>:8,<node12>:8,<node13>
↪:8,<node14>:8,<node15>:8,<node16>:8 -np 128 -x NCCL_IB_HCA=<nic01>,<nic02>,<nic03>,
↪<nic04>,<nic05>,<nic06>,<nic07>,<nic08> /opt/rccl-tests/build/all_reduce_perf -b 8 -e␣
↪16G -f 2 -g 1
```

3.6.3 Additional command parameters

To optimize RCCL performance across nodes, most systems require additional `mpirun` parameters in tandem with system-specific tuning. This section provides a description of parameters that may be helpful for improving performance depending on the design features of your cluster (network topology, NICs, OS, and so on). These parameters can be provided at the command line or used in a pre-designed RCCL configuration file.

3.6.3.1 oob_tcp_if_exclude

MCA parameter that instructs Open MPI to exclude a network interface when searching for out-of-band (OOB) TCP communications during the initiation of `mpirun`. Include this parameter if your system has interfaces that shouldn't be involved in RCCL operations. Multiple interfaces may be included as comma-separated values.

Example

```
-mca oob_btl_if_exclude=<interface1>,<interface2>
```

3.6.3.2 oob_btl_if_exclude

Provides the same function as `oob_tcp_if_exclude`, but for BTL OOB communications. If excluding any interfaces, use both parameters.

Example

```
-mca oob_btl_if_exclude=<interface1>,<interface2>
```

A common scenario for using these options together is when a node has Docker and Loopback (lo) interfaces that interfere with Open MPI internal communications and can cause a hang when running operations.

```
$ ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen␣
↪1000
    ...
```

(continues on next page)

(continued from previous page)

```
12: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group␣
↪default
    ...
```

You can exclude these interfaces from an Open MPI run by adding these options to your command string:

```
-mca oob_tcp_if_exclude docker,lo -mca btl_tcp_if_exclude docker,lo
```

3.6.3.3 NCCL_NET_GDR_LEVEL

Used to define the maximum level of distance between a GPU and NIC at which GPU Direct RDMA/PeerDirect should be used. The GDR value is detected automatically based on PCI device topology, so setting this manually isn't typically necessary but may be useful when debugging low performance.There are several accepted string values:

- `LOC` - Never use RDMA (always disabled).
- `PIX` - Use RDMA when GPU and NIC are connected to the same PCI switch.
- `PXB` - Use RDMA when GPU and NIC are connected through different PCI switches (potentially multiple hops).
- `PHB` - Use RDMA when GPU and NIC are on the same NUMA node. Traffic will go through the CPU.
- `SYS` - Use RDMA even across the SMP interconnect between NUMA nodes (e.g., QPI/UPI) (always enabled).

For most configurations, `PHB` is the recommended value.

Example

```
NCCL_NET_GDR_LEVEL=PHB
```

3.6.3.4 NCCL_DEBUG

Including this parameter displays various levels of information for debugging. Useful values include `VERSION` to display the RCCL version, linked ROCm version, and the RCCL git tag, while `INFO` displays debugging information that can be useful when trying to diagnose reasons for hangs or other errors while running RCCL tests. It can also be used with `NCCL_DEBUG_SUBSYS` to view subsystem information.

Example

```
NCCL_DEBUG=VERSION
```

3.6.3.5 NCCL_DEBUG_SUBSYS

Used in conjunction with `NCCL_DEBUG=INFO` to filter information based on subsystem. Value is a comma separated list of subsystems to include in debugging.

Accepted values are INIT (initialization, default value), COLL (collectives), P2P (peer-to-peer), SHM (shared memory), NET (network), GRAPH (topology detection and graph search), TUNING (algorithm/protocol tuning), ENV (environment settings), ALLOC (memory allocations), and ALL (includes all subsystems).

Example

```
NCCL_DEBUG=INFO NCCL_DEBUG_SUBSYS=INIT,GRAPH,NET,COLL
```

3.6.3.6 NCCL_ALGO

Sets the algorithm for a collective operation. Value may be either `Ring` or `Tree`. A default value is set at each message size and differs between collectives (that is, a collective may use a tree algorithm at smaller message sizes and transition to a ring algorithm as the message size becomes larger). Defining `NCCL_ALGO` as a parameter forces the selected algorithm for all message sizes.

Example

```
NCCL_ALGO=Ring
```

3.6.3.7 NCCL_TOPO_FILE

Loads a pre-existing topology XML file derived from a system with AMD GPUs before detecting node topology. Value is the path to the XML file.

Example

```
NCCL_TOPO_FILE=/path/to/topology-file.xml
```

> **Note**
>
> If you are working in a virtual environment, the topology file **must** declare `<system version="2">` at the start of the file, or the RCCL test will fail. This is because RCCL defines its own `NCCL_TOPO_XML_VERSION` to accommodate additional fields present in RCCL topology files.

3.6.3.8 NCCL_TOPO_DUMP_FILE

Creates a post-detection topology XML file in a user-defined location. Value is the path to where the file will be created or overwritten.

Example

```
NCCL_TOPO_DUMP_FILE=/path/to/topology-file.xml
```

3.6.3.9 NCCL_IB_GID_INDEX

Sets the Global ID index (GID) for a RoCE device. In most cases for RoCEv2, the value should be set to `3`, but you can verify this with the `show_gids` script on a Mellanox NIC and `ibv_devinfo -vvv` on a Broadcom NIC. Unnecessary for InfiniBand networks.

Example

```
NCCL_IB_GID_INDEX=3
```

3.6.3.10 NCCL_IB_QPS_PER_CONNECTION

Defines the amount of queue pairs (QPs) to use per InfiniBand/RoCE connection. Default value is 1. Increasing the value can have performance impact as more connections require more memory.

Example

```
NCCL_IB_QPS_PER_CONNECTION=4
```

3.6.3.11 NCCL_IB_PCI_RELAXED_ORDERING

Determines the usage of relaxed ordering. 2 is the default value and uses relaxed ordering when available while setting a value of 1 forces relaxed ordering, and 0 disables it.

Example

```
NCCL_IB_PCI_RELAXED_ORDERING=1
```

3.6.3.12 NCCL_SOCKET_IFNAME

Explicitly defines the network interface for RCCL to use when initiating communications. This is useful to define when multiple interfaces present to ensure RCCL is using the intended interface. Multiple interfaces may be provided as comma separated values.

Example

```
NCCL_SOCKET_IFNAME=<interface>
```

3.6.3.13 RCCL_ENABLE_INTRANET

Enables use of local intranet during single-node RCCL testing. Use this if you want to include the NICs and leaf switch while testing on single-node. Can deliver performance improvements as the CPU is assisted by the switch. To activate, set value to 1 (Default value is `0`, or deactivated).

Example

```
RCCL_ENABLE_INTRANET=1
```

3.7 Run OSU Micro Benchmarks

Running the OSU Micro Benchmarks (OMB) with MPI simulates conditions similar to an AI/HPC workload over your cluster network and serves as a good back-up or secondary test to run and compare with RCCL results. As with RCCL, passwordless SSH and fingerprinting is required between all server pairs that will be tested.

OMB supports both point to point (pt2pt) operations between one discrete component on a server (host or device) to another, and the same collective operations already seen in the RCCL section.

In a typical use case, you start with a pair of nodes and run the pt2pt benchmarks then move on to collectives.

Commands in the table below must run on two nodes with RoCE or InfiniBand interconnect from Host to Host (CPU to CPU). You can invoke the command from either node, but directories must mirror one another or the tests will hang.

> **Note**
>
> The paths for the MPI and OMB commands in this section presume both are installed in the `/opt` directory. Installation paths for your environment may be different and should be updated accordingly.

Command	Usage
osu_bw	`$OMPI_DIR/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host <node1-IP>,<node2-IP> -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc $OSU_DIR/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -d rocm`
osu_bibw	`$OMPI_DIR/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host <node1-IP>,<node2-IP> -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc $OSU_DIR/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bibw -d rocm`
osu_mbw_mr	`$OMPI_DIR/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host <node1-IP>,<node2-IP> -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc $OSU_DIR/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_mbw_mr -d rocm`
osu_latency	`/$OMPI_DIR/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host <node1-IP>,<node2-IP> -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc $OSU_DIR/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_latency -d rocm`
osu_multi_lat	`$OMPI_DIR/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host <node1-IP>,<node2-IP> -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc $OSU_DIR/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_multi_lat -d rocm`

You can change communications mode by appending `D D` to the end of command for D2D, or `D H` for D2H (and vice-versa).

For more information on MCA parameter options, refer to the Module Component Architecture (MCA) documentation for Open MPI.

3.7.1 Collective OSU benchmarks

Command	Usage
osu_allreduce	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host 10.1.10.110,10.1.10.72 -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_allreduce -d rocm D D`
osu_allreduce 2N 16Proc	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 16 -hostfile ./hostfile -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_allreduce -d rocm D D`
osu_alltoall	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host 10.1.10.110,10.1.10.72 -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_alltoall -d rocm D D`
osu_alltoall 2N 16Proc	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 16 -hostfile ./hostfile -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_alltoall -d rocm D D`
osu_allgather	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 2 -host 10.1.10.110,10.1.10.72 -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_allgather -d rocm D D`
osu_allgather 2N 16Proc	`/opt/ompi/bin/mpirun --mca pml ucx --mca osc ucx --mca spml ucx --mca btl ^self,vader,openib --mca coll_hcoll_enable 0 --bind-to none -np 16 -hostfile ./hostfile -x UCX_TLS=all -x HIP_VISIBLE_DEVICES=1 numactl --localalloc /opt/osu-7.3/libexec/osu-micro-benchmarks/mpi/collective/osu_allgather -d rocm D D`

CHAPTER

FOUR

MULTI-NODE INFERENCE LOAD BALANCING

This guide describes how to set up a scalable, high-performance multi-node LLM inference cluster using AMD GPUs, supporting efficient horizontal scaling and highly available deployments.

4.1 Architecture overview

This solution implements a distributed LLM inference system with three main components:

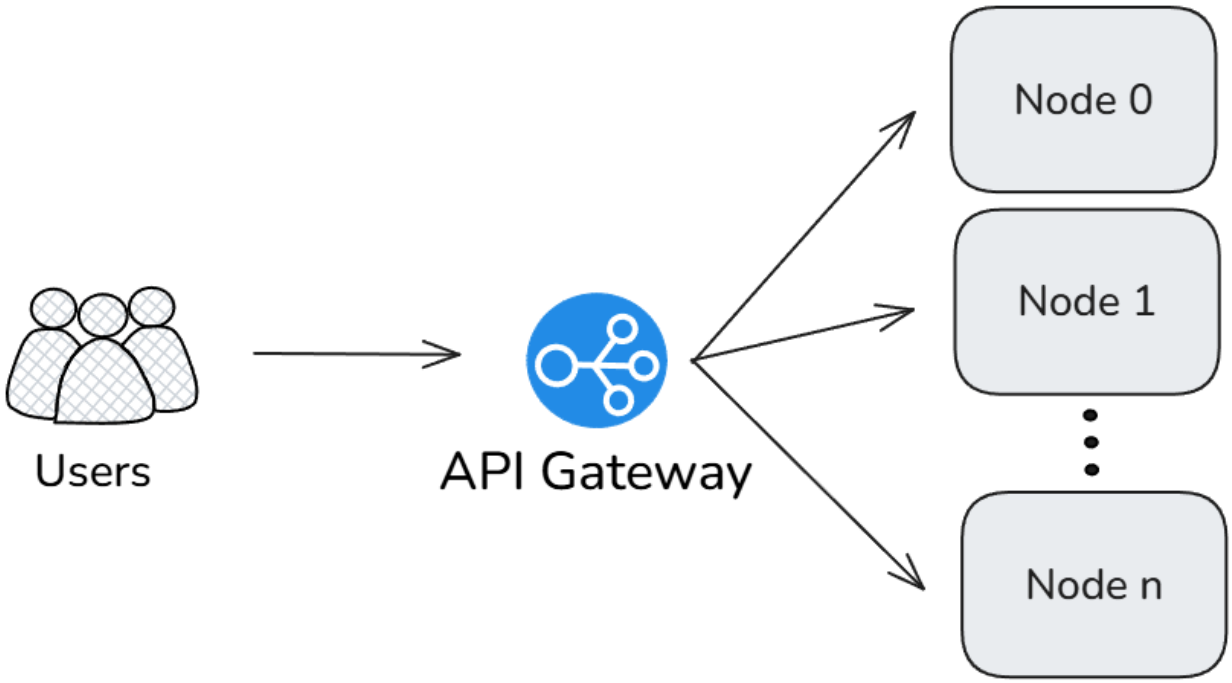
- **Inference pool**: Multiple inference nodes running vLLM or SGLang servers on AMD GPUs using tensor parallelism.
- **API gateway layer**: A unified entry point that distributes requests across the inference pool. This guide demonstrates two options:
  - A LiteLLM-based load balancer - optimized for LLM workloads with built-in observability.
  - An Nginx-based load balancer - a production-grade reverse proxy with high performance.
- **Monitoring layer**: Prometheus and Grafana for comprehensive metrics collection and visualization, with additional load testing tools.

This architecture allows horizontal scaling by adding more inference nodes while maintaining a single API endpoint for client applications. This architecture supports various model sizes:

- **Small models**: Can run efficiently on a single GPU.
- **Medium models**: Typically require two or more GPUs with tensor parallelism.
- **Large models**: Requires multi-node deployments for high availability and performance.

Tensor Parallelism distributes model layers across multiple GPUs, allowing inference of models too large to fit in a single GPU's memory. The `--tensor-parallel-size` (`-tp`) parameter determines how many GPUs will share the model weights.

4.1.1 Logical diagram



4.2 Prerequisites

- Multiple ROCm-compatible nodes with AMD GPUs.
- Docker and Docker Compose installed on all nodes.
- Network connectivity between nodes.
- Models downloaded to a shared or local storage location.

4.2.1 NUMA configuration

For optimal performance, disable automatic NUMA balancing on each node before starting the inference servers:

```
# Disable automatic NUMA balancing
sudo sh -c 'echo 0 > /proc/sys/kernel/numa_balancing'

# Verify NUMA balancing is disabled (should return 0)
cat /proc/sys/kernel/numa_balancing
```

4.3 Deployment

This section provides step-by-step instructions for deploying components for multi-node inference load balancing.

4.3.1 Project structure

```
/llm-cluster/
├── nodes/                  # Inference node files
│   ├── docker-compose.yml
├── gateway/                # API Gateway/Load Balancer files
│   ├── litellm
│   │   ├── config.yaml
│   │   └── docker-compose.yml
│   └── nginx
│       ├── docker-compose.yml
│       └── nginx.conf
├── monitoring/             # Monitoring stack files
│   ├── docker-compose.yml
│   ├── grafana/
│   │   ├── datasources.yml
│   │   ├── Instinct_Dashboard.json
│   │   └── vLLM_Dashboard.json
│   ├── influxdb/
│   ├── prometheus/
│   │   └── prometheus.yml
│   └── scripts/
│       ├── chat-completions-test.js
│       ├── helpers/
│       │   └── openaiGeneric.js
│       ├── prompt-length-test.js
│       ├── ramp-up-test.js
│       └── stress-test.js
```

4.3.2 Inference pool setup

Perform these actions on each inference node.

1. Create the directory structure:

```
mkdir -p ~/llm-cluster/nodes
cd ~/llm-cluster/nodes
```

2. Create a `.env` file in the `nodes/` folder the with appropriate configuration for your environment:

```
NODE_ID=node1               # Unique identifier for this node
MODEL_PATH=/path/to/models  # Path to local or shared model storage
MODEL_NAME=Llama-3.1-8B-Instruct  # Model to deploy
TP_SIZE=4                   # Tensor parallelism degree (number of GPUs to use)
GPU_DEVICES=0,1,2,3         # GPU devices to use
PORT=8000                   # Port to expose the inference API
SHM_SIZE=32GB               # Shared memory size for container
```

3. Create a `docker-compose.yml` file for the inference nodes. Two options are provided below for different inference backends.

vLLM example

```
services:
  vllm:
```

(continues on next page)

(continued from previous page)

```
    image: rocm/vllm:instinct_main
    container_name: vllm_${NODE_ID:-node1}
    shm_size: ${SHM_SIZE:-32GB}
    ipc: host
    network_mode: host
    devices:
      - /dev/kfd
      - /dev/dri
    group_add:
      - video
    security_opt:
      - seccomp=unconfined
    volumes:
      - ${MODEL_PATH}:/data/models
    environment:
      - ROCR_VISIBLE_DEVICES=${GPU_DEVICES:-0,1,2,3}
    command: >
      vllm serve /data/models/${MODEL_NAME}
      --dtype float16
      --tensor-parallel-size ${TP_SIZE:-4}
      --port ${PORT:-8000}
    restart: unless-stopped
```

SGLang example

```
services:
  sglang:
    image: lmsysorg/sglang:v0.4.6.post2-rocm630
    container_name: sglang_${NODE_ID:-node1}
    shm_size: ${SHM_SIZE:-32GB}
    ipc: host
    network_mode: host
    devices:
      - /dev/kfd
      - /dev/dri
    group_add:
      - video
    security_opt:
      - seccomp=unconfined
    volumes:
      - ${MODEL_PATH}:/data/models
    environment:
      - ROCR_VISIBLE_DEVICES=${GPU_DEVICES:-0,1,2,3}
      - RCCL_MSCCL_ENABLE=0
      - CK_MOE=1
      - HSA_NO_SCRATCH_RECLAIM=1
    command: >
      python3 -m sglang.launch_server
      --model /data/models/${MODEL_NAME}
      --tp ${TP_SIZE:-4}
      --trust-remote-code
      --port ${PORT:-8000}
```

(continues on next page)

(continued from previous page)

```
      --enable-metrics
    restart: unless-stopped
```

4. Start the inference services:

```
docker compose up -d
```

4.3.3 API gateway setup

On the API gateway node, create the gateway directory structure:

```
mkdir -p ~/llm-cluster/gateway
cd ~/llm-cluster/gateway
```

Choose one of the following gateway options based on your requirements.

4.3.3.1 Option 1: LiteLLM-based load balancer

LiteLLM provides specialized routing, load balancing, and observability for LLM API calls, supporting multiple LLM providers and models through a unified OpenAI-compatible interface.

1. Create `docker-compose.yml` for LiteLLM:

```
services:
  litellm:
    image: ghcr.io/berriai/litellm:main-stable
    container_name: litellm_gateway
    network_mode: host
    volumes:
      - ./config.yaml:/app/config.yaml
    command: ["--config", "/app/config.yaml", "--port", "4000", "--num_workers", "8
↪"]
    environment:
      LITELLM_MASTER_KEY: "${LITELLM_MASTER_KEY}"
    env_file: .env
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

2. Create `config.yaml` to define the model routing configuration:

```
model_list:
- model_name: DeepSeek-R1
   litellm_params:
      model: openai/deepseek-ai/DeepSeek-R1
      api_base: http://node0:8000/v1

- model_name: DeepSeek-R1
   litellm_params:
      model: openai/deepseek-ai/DeepSeek-R1
```

(continues on next page)

(continued from previous page)

```
      api_base: http://node1:8000/v1

# Add additional nodes as needed
# - model_name: DeepSeek-R1
#   litellm_params:
#     model: openai/deepseek-ai/DeepSeek-R1
#     api_base: http://nodeN:8000/v1

# Configure load balancing
router_settings:
routing_strategy: least-busy  # Distributes requests to least busy nodes
num_retries: 3                # Number of retries if a request fails
timeout: 300                  # Request timeout in seconds
```

3. Create `.env` file with your API key:

```
LITELLM_MASTER_KEY=sk-1234
```

> **Note**
>
> For production environments, replace the default key with a strong, randomized value.

4. Start the LiteLLM gateway:

```
docker compose up -d
```

5. Verify that all LLM endpoints are healthy:

```
curl -X 'GET' \
'http://localhost:4000/health' \
-H 'accept: application/json' \
-H 'Authorization: Bearer sk-1234' | jq
```

Expected output


```
{
"healthy_endpoints": [
   {
      "model": "openai/deepseek-ai/DeepSeek-R1",
      "api_base": "http://node0:8000/v1"
   },
   {
      "model": "openai/deepseek-ai/DeepSeek-R1",
      "api_base": "http://node1:8000/v1"
   },
   {
      "model": "openai/deepseek-ai/DeepSeek-R1",
      "api_base": "http://node2:8000/v1"
   },
   {
      "model": "openai/deepseek-ai/DeepSeek-R1",
```


(continues on next page)

(continued from previous page)

```
      "api_base": "http://node3:8000/v1"
   }
],
"unhealthy_endpoints": [],
"healthy_count": 4,
"unhealthy_count": 0
}
```

4.3.3.1.1 LiteLLM monitoring options

LiteLLM provides several monitoring and observability options:

- **Basic logging**: Available in the open source version, provides request/response logging and basic metrics
- **Callback integrations**: LiteLLM supports custom callbacks for advanced monitoring with tools like:
  - LangFuse
  - Helicone
  - LangSmith
  - Custom callback handlers

This guide uses the open source version of LiteLLM with an internal Prometheus/Grafana stack for system-level monitoring. If you need LLM-specific tracing and observability, consider exploring the callback integrations.

4.3.3.2 Option 2: Nginx-based load balancer

Nginx provides a high-performance, scalable HTTP server and reverse proxy that can efficiently distribute traffic across multiple inference nodes.

1. Create `nginx.conf` with the following configuration:

```
worker_processes auto;
worker_rlimit_nofile 65535;
events {
   worker_connections 65535;
}

http {
   include       mime.types;
   default_type  application/octet-stream;
   sendfile      on;
   keepalive_timeout 65;

   # Define upstream server group
   upstream vllm_pool {
      # Use least_conn for distributing traffic based on least number of current␣
↪connections
      least_conn;

      # Add inference server entries - update with your node hostnames/IPs
      server node0:8000;
      server node1:8000;
      # Add additional nodes as needed
```

(continues on next page)

(continued from previous page)

```
        # server nodeN:8000;

        keepalive 32;
    }

    server {
        listen 80;

        # Health check endpoint
        location /health {
            return 200 'healthy\n';
            add_header Content-Type text/plain;
        }

        # API endpoint for frontend clients
        location / {
            proxy_pass http://vllm_pool;
            proxy_http_version 1.1;
            proxy_set_header Connection "";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

            # Timeouts for long-running inference requests
            proxy_connect_timeout 300s;
            proxy_read_timeout 300s;
            proxy_send_timeout 300s;

            # Buffer settings for large responses
            proxy_buffer_size 16k;
            proxy_buffers 8 16k;
            proxy_busy_buffers_size 32k;
        }
    }
}
```

2. Create `docker-compose.yml` for Nginx:

```
services:
nginx:
    image: nginx:latest
    container_name: nginx_gateway
    network_mode: host
    volumes:
        - ./nginx.conf:/etc/nginx/nginx.conf:ro
    restart: unless-stopped
    logging:
        driver: "json-file"
        options:
        max-size: "10m"
        max-file: "3"
```

3. Start the Nginx gateway:

```
docker compose up -d
```

4.3.3.2.1 Monitoring Nginx gateway

To enable monitoring for your Nginx gateway, add the `nginx-prometheus-exporter`:

1. Update `docker-compose.yml` to include the exporter:

```
services:
nginx:
   # ...existing nginx configuration...

nginx-exporter:
   image: nginx/nginx-prometheus-exporter:latest
   container_name: nginx_exporter
   command:
      - --nginx.scrape-uri=http://localhost/stub_status
   network_mode: host
   restart: unless-stopped
   depends_on:
      - nginx
```

2. Add a status endpoint to `nginx.conf` inside the server block:

```
location /metrics {
   stub_status on;
   access_log off;
   allow 127.0.0.1;
   deny all;
}
```

4.4 Monitoring stack setup

Perform these steps on the monitoring node.

1. Create the monitoring directory structure:

```
mkdir -p ~/llm-cluster/monitoring/{prometheus,grafana,influxdb}
cd ~/llm-cluster/monitoring
```

2. Set appropriate permissions for Grafana and InfluxDB data directories:

```
# Set permissions to allow container processes to write data
chmod 777 ~/llm-cluster/monitoring/grafana
chmod 777 ~/llm-cluster/monitoring/influxdb
```

3. Create `docker-compose.yml` for the monitoring stack:

```
services:
# Check https://hub.docker.com/r/rocm/device-metrics-exporter/tags for the latest␣
↪version
device-metrics-exporter:
   image: rocm/device-metrics-exporter:v1.3.0-beta.1
```

(continues on next page)

(continued from previous page)

```
    container_name: device-metrics-exporter
    restart: unless-stopped
    group_add:
      - video
    volumes:
      - ./config.json:/etc/metrics/config.json
    devices:
      - /dev/kfd
      - /dev/dri
    ports:
      - "5000:5000"

prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
    ports:
      - "9090:9090"
    restart: unless-stopped

influxdb:
    image: influxdb:1.11.8
    container_name: influxdb
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=k6
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=admin
    volumes:
      - ./influxdb:/var/lib/influxdb

grafana:
    image: grafana/grafana:latest
    container_name: grafana
    volumes:
      - ./grafana/datasources.yml:/etc/grafana/provisioning/datasources/datasources.
↪yml
      - ./grafana:/var/lib/grafana
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_ADMIN_PASSWORD:-admin}
    ports:
      - "3000:3000"
    depends_on:
      - prometheus
    restart: unless-stopped
```

4. Create `prometheus/prometheus.yml` to configure metrics collection:

```yaml
global:
scrape_interval: 15s

scrape_configs:
# Host OS metrics
- job_name: 'node'
   static_configs:
   - targets: ['localhost:9100']

# Inference servers
- job_name: 'vllm'
   metrics_path: /metrics
   scrape_interval: 15s
   static_configs:
      - targets: ['node0:8000', 'node1:8000'] # Add additional nodes as needed
      labels:
         service: 'vllm'

# Nginx Gateway metrics (if using Nginx with nginx-prometheus-exporter)
- job_name: 'nginx'
   scrape_interval: 15s
   metrics_path: /metrics
   static_configs:
      - targets: ['localhost:9113']
   relabel_configs:
      - source_labels: [__address__]
      target_label: instance
      replacement: 'nginx-gateway'

# AMD GPU device metrics
- job_name: 'amd_gpu_metrics'
   scrape_interval: 5s
   metrics_path: /metrics
   static_configs:
      - targets: ['node0:5000', 'node1:5000']
      labels:
         service: 'amd_gpu_metrics'
```

> **Note**
>
> Replace `node0` and `node1` with the actual hostnames or IP addresses of your inference nodes. When running Prometheus in a docker container, change instances of `localhost` to `host.docker.internal`.

5. Create `grafana/datasources.yml` to configure the Prometheus data source:

```yaml
apiVersion: 1

datasources:
- name: Prometheus
   type: prometheus
   access: proxy
```

(continues on next page)

(continued from previous page)

```
    url: http://prometheus:9091
    isDefault: true

- name: InfluxDB
    type: influxdb
    access: proxy
    url: http://influxdb:8086
    database: k6
    user: admin
    password: admin
    editable: true
```

6. Start the monitoring services:

```
docker compose up -d
```

4.5 Testing and performance evaluation

Once your multi-node inference system is deployed, you can validate its functionality and evaluate its performance.

4.5.1 Testing with LiteLLM gateway

Send a test request to the LiteLLM endpoint:

```
curl http://localhost:4000/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer sk-1234" \
  -d '{"model": "DeepSeek-R1", "prompt": "What is AMD Instinct?", "max_tokens": 256,
↪"temperature": 0.0}'
```

4.5.2 Testing with Nginx gateway

Send a test request to the Nginx endpoint:

```
curl http://localhost/v1/completions \
  -H "Content-Type: application/json" \
  -d '{"model": "DeepSeek-R1", "prompt": "What is AMD Instinct?", "max_tokens": 256,
↪"temperature": 0.0}'
```

Expected output format (content may vary):

```
{
  "text": [
    "What is AMD Instinct? AMD Instinct is a line of high-performance computing (HPC)␣
↪and
    artificial intelligence (AI) accelerators designed for datacenter and cloud␣
↪computing
    applications. It is based on AMDs Radeon Instinct architecture, which is optimized␣
↪for HPC
    and AI workloads. AMD Instinct accelerators are designed to provide high-performance
    computing and AI acceleration for a wide range of applications, including scientific␣
```

(continues on next page)

(continued from previous page)

```
↪simulations,
    data analytics, machine learning, and deep learning.

    AMD Instinct accelerators are based on AMDs Radeon Instinct architecture, which is␣
↪designed
    to provide high-performance computing and AI acceleration. They are built on a 7nm␣
↪process node
    and feature a high-performance GPU core, as well as a large amount of memory and␣
↪bandwidth to
    support high-performance computing and AI workloads.

    AMD Instinct accelerators are designed to be used in a variety of applications,␣
↪including:
    Scientific simulations: AMD Instinct accelerators can be used to accelerate complex␣
↪scientific
    simulations, such as weather forecasting, fluid dynamics, and molecular dynamics.
    Data analytics: AMD Instinct accelerators can be used to accelerate data analytics␣
↪workloads,
    such as data compression, data encryption, and data mining.
    Machine learning: AMD Instinct accelerators can be used to accelerate machine␣
↪learning workloads"
  ]
}
```

4.5.3 Performance testing with Apache Bench

Apache Bench (`ab`) is a lightweight tool for benchmarking HTTP servers, ideal for quick performance evaluation.

4.5.3.1 Installation options

Option 1: Install Apache Bench Locally

```
sudo apt-get update
sudo apt-get install apache2-utils
```

Option 2: Run Apache Bench in a Container

```
docker run -it --rm \
  --shm-size=8GB \
  --ipc=host \
  --network=host \
  --entrypoint bash \
  ubuntu/apache2:2.4-22.04_beta
```

4.5.3.2 Running Apache Bench tests

1. Create a request payload file:

   ```
   cat > postdata << EOF
   {"model": "DeepSeek-R1", "prompt": "What is AMD Instinct?", "max_tokens": 256,
   ↪"temperature": 0.0}
   EOF
   ```

2. Run the benchmark with desired concurrency and request count:

```
ab -n 1000 -c 100 -T application/json -p postdata -H "Authorization: Bearer sk-1234
↪" http://localhost:4000/v1/completions
```

Key parameters:

- `-n 1000`: Total number of requests to perform
- `-c 100`: Number of concurrent requests
- `-T application/json`: Content-type header for POST data
- `-p postdata`: File containing data to POST
- `-H`: Additional header for authentication

4.5.3.3 Sample performance test commands

Here are examples of commands to test different models and configurations:

```
# Test with Llama-3.1-8B-Instruct
ab -n 20000 -c 2000 -T application/json -p postdata http://localhost:80/v1/completions

# Test with Llama-3.1-405B-Instruct
ab -n 20000 -c 2000 -T application/json -p postdata http://localhost:80/v1/completions

# Test with DeepSeek-R1
ab -n 20000 -c 2000 -T application/json -p postdata http://localhost:80/v1/completions
```

4.5.4 Advanced load testing with k6

For more sophisticated load testing scenarios, Grafana k6 offers enhanced capabilities including detailed metrics collection and realistic user simulation. The test scripts used in this section are available to download from https://github.com/ROCm/gpu-cluster-networking/tree/develop/examples/llm-cluster/monitoring/scripts

4.5.4.1 Installing k6

Option 1: Install k6 locally

```
apt install -y k6
```

For additional installation options, refer to the official k6 installation guide.

Option 2: Run k6 in a Container

```
docker run --rm -i \
  --network=host \
  -v ${PWD}/scripts:/scripts \
  -e "OPENAI_URL=http://localhost:4000" \
  -e "API_KEY=sk-1234" \
  -e "MODEL_NAME=DeepSeek-R1" \
  grafana/k6 run /scripts/chat-completions-test.js
```

4.5.4.2 Setting up k6

Configure environment variables for the test scripts:

```
cd ~/llm-cluster/monitoring/scripts
cat > .env << EOL
export OPENAI_URL=http://localhost:4000  # Use your LiteLLM or Nginx endpoint
export API_KEY=sk-1234                   # API key if required by your gateway
export MODEL_NAME=DeepSeek-R1            # Your deployed model name
EOL

source .env
```

4.5.4.3 Running k6 test scripts

The repository includes several specialized test scripts for different testing scenarios:

4.5.4.3.1 Chat completions test

```
k6 run --out influxdb=http://localhost:8086/k6 chat-completions-test.js
```

4.5.4.3.2 Ramp-up test

```
k6 run --out influxdb=http://localhost:8086/k6 ramp-up-test.js
```

4.5.4.3.3 Stress test

```
k6 run --out influxdb=http://localhost:8086/k6 stress-test.js
```

4.5.4.3.4 Prompt length test

```
k6 run --out influxdb=http://localhost:8086/k6 prompt-length-test.js
```

On completion, k6 will provide a summary similar to this:

```
$ k6 run --out influxdb=http://localhost:8086 scripts/chat-completions-test.js

         /\      Grafana   //
    /\  /  \     |\  __   /  /
   /  \/    \    | |/ /  /   \
  /          \   |   (  |  ()  |
 / __________ \  |_|\_\  \_____/

     execution: local
        script: scripts/chat-completions-test.js
        output: InfluxDBv1 (http://localhost:8086)

     scenarios: (100.00%) 1 scenario, 5 max VUs, 1m30s max duration (incl. graceful␣
↪stop):
              * default: 5 looping VUs for 1m0s (gracefulStop: 30s)
```

(continues on next page)

(continued from previous page)

```
   THRESHOLDS

    http_req_duration
    ✓ 'p(95)<5000' p(95)=1.66s

    http_req_failed
    ✓ 'rate<0.01' rate=0.00%


   TOTAL RESULTS

    checks_total.....................: 170     2.64314/s
    checks_succeeded...................: 100.00% 170 out of 170
    checks_failed.....................: 0.00%   0 out of 170

    ✓ is status 200
    ✓ has valid JSON response

    CUSTOM
    completion_tokens...................: avg=100       min=100      med=100      ␣
↪max=100       p(90)=100       p(95)=100
    prompt_tokens.....................: avg=26        min=26       med=26       ␣
↪max=26        p(90)=26        p(95)=26
    tokens_per_second.....................: avg=83.173393 min=57.87037 med=87.565674␣
↪max=91.324201 p(90)=90.546921 p(95)=90.810037
    total_tokens.....................: avg=126       min=126      med=126      ␣
↪max=126       p(90)=126       p(95)=126

    HTTP
    http_req_duration...................: avg=1.21s     min=1.09s    med=1.14s     max=1.
↪72s     p(90)=1.44s     p(95)=1.66s
      { expected_response:true }.....: avg=1.21s     min=1.09s    med=1.14s     max=1.
↪72s     p(90)=1.44s     p(95)=1.66s
    http_req_failed.....................: 0.00% 0 out of 85
    http_reqs.....................: 85    1.32157/s

    EXECUTION
    iteration_duration...................: avg=3.67s     min=2.16s    med=3.62s     max=5.
↪35s     p(90)=4.82s     p(95)=4.96s
    iterations.....................: 85    1.32157/s
    vus.....................: 1     min=1       max=5
    vus_max.....................: 5     min=5       max=5

    NETWORK
    data_received.....................: 87 kB 1.3 kB/s
    data_sent.....................: 31 kB 477 B/s

running (1m04.3s), 0/5 VUs, 85 complete and 0 interrupted iterations
default ✓ [======================================] 5 VUs  1m0s
```

4.5.4.4 Viewing k6 test results

After running the tests, you can view the results in Grafana:

1. Open Grafana at `http://<your-monitoring-node-ip>:3000`
2. Log in with your credentials (default: admin/admin, unless changed via `GRAFANA_ADMIN_PASSWORD` environment variable)
3. Access the k6 dashboard by importing the dashboard ID `14801` or by navigating to the pre-configured dashboard if available. The dashboard can be found at: https://grafana.com/grafana/dashboards/14801-k6-dashboard/

The k6 dashboard provides detailed metrics about request rates, response times, errors, and other performance indicators that help you understand your system's behavior under load.

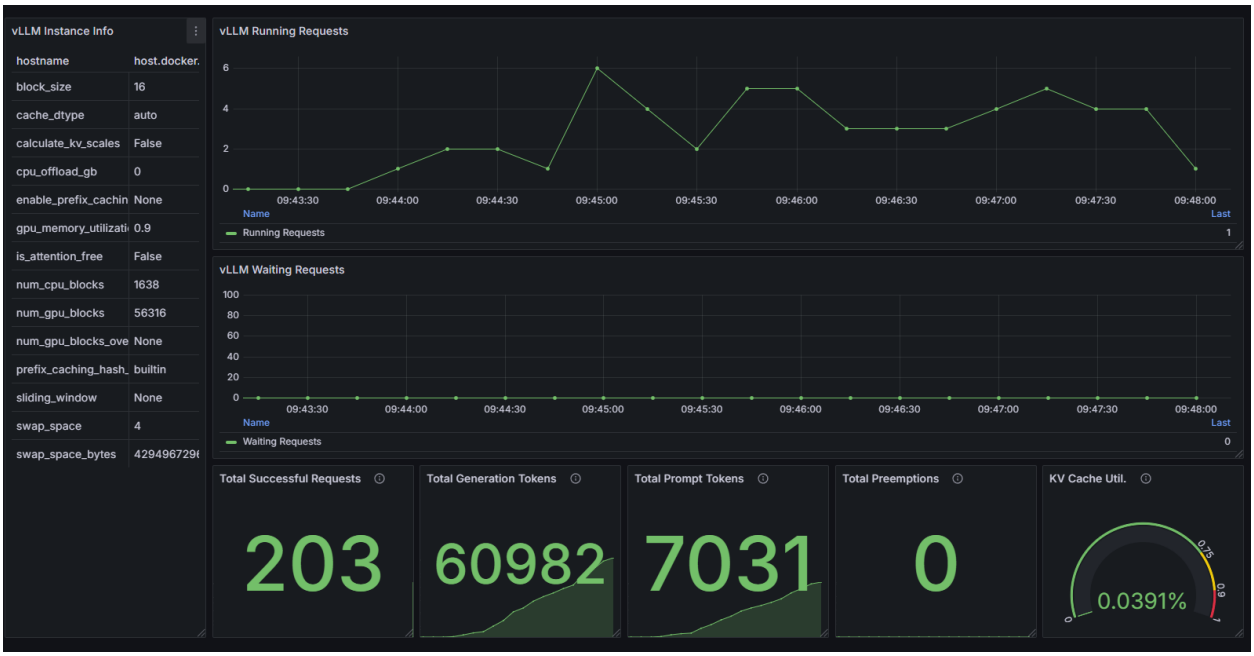
4.6 Monitoring and visualization

4.6.1 Available dashboards

The monitoring stack includes pre-configured Grafana dashboards for comprehensive system visibility. These dashboards are provided in the repository's `examples/llm-cluster/monitoring/grafana` directory:

AMD Instinct dashboard (`Instinct_Dashboard.json`): Monitors GPU performance metrics including temperature, utilization, memory usage, and power consumption. Also available at AMD Instinct Single Node Dashboard.

vLLM dashboard (`vLLM_Dashboard.json`): Provides insights into vLLM server performance, including request throughput, latency metrics, and queue statistics.



Additional recommended dashboards for comprehensive monitoring:

- **k6 dashboard**: Visualizes load test results with detailed performance metrics. Available for import into Grafana with ID `14801` or at https://grafana.com/grafana/dashboards/14801-k6-dashboard/.
- **vLLM reference dashboard**: Official dashboard from the vLLM project for detailed inference metrics. Available at https://github.com/vllm-project/vllm/blob/main/examples/online_serving/prometheus_grafana/grafana.json.
- **NGINX dashboard**: Official dashboard for the NGINX Prometheus exporter. https://grafana.com/grafana/dashboards/12767-nginx/

For instructions on importing dashboards into your Grafana instance, follow the official Grafana Dashboard Import Guide.

4.7 Performance optimization recommendations

Achieving optimal performance for your multi-node inference deployment requires experimentation and continuous monitoring. This section provides recommendations for tuning your setup based on your specific workload characteristics.

4.7.1 Compare different configurations

To identify the optimal setup for your specific use case, systematically test different configurations.

- Load balancer options
  - **LiteLLM**: Generally provides better handling of LLM-specific requirements like streaming responses and specialized routing
  - **Nginx**: Often delivers higher raw throughput for simple completion requests and offers more configuration flexibility
- Inference servers
  - **vLLM**: https://docs.vllm.ai/

  - **SGLang**: https://docs.sglang.ai/
  - **TGI**: https://huggingface.co/docs/text-generation-inference/index
- Inference configuration
  - Test different tensor parallel sizes to find the optimal balance between throughput and latency.
  - Experiment with batch sizes (`--max-batch-size` in vLLM) to increase throughput for concurrent requests.
  - Try different quantization options to improve memory efficiency.

4.7.2 Using historical performance data

You can use the monitoring setup in this guide to review stored historical performance data and track changes over time:

- Establish performance baselines
  - Run benchmark tests after initial setup to establish baseline performance metrics.
  - Document key metrics like tokens per second, request latency, and GPU utilization.
- Track performance trends
  - Set up Grafana dashboards with time series views of key metrics.
  - Create alerts for significant deviations from established baselines.

4.7.3 System-level optimizations

Beyond the application components themselves, consider these system-level optimizations:

- Network configuration
  - Ensure nodes have sufficient network bandwidth for model weight synchronization.
  - Consider using dedicated network interfaces for inter-node communication.
- Host OS tuning
  - Adjust kernel parameters related to networking and memory management.
  - The NUMA configuration mentioned earlier in this guide is just one example.

You can find more information on system optimization at these links:

- System optimization guides: https://rocm.docs.amd.com/en/latest/how-to/system-optimization/index.html
- Performance guides: https://rocm.docs.amd.com/en/latest/how-to/gpu-performance/mi300x.html
- Instinct single node networking: https://instinct.docs.amd.com/projects/gpu-cluster-networking/en/latest/how-to/single-node-config.html
- Instinct multi-node networking: https://instinct.docs.amd.com/projects/gpu-cluster-networking/en/latest/how-to/multi-node-config.html

4.7.4 Cost-performance balance

When scaling your cluster, consider both performance and resource utilization:

- Right-sizing
  - Use Grafana dashboards to identify under-utilized resources.
  - Scale the number of nodes based on actual usage patterns and SLAs.

- Workload scheduling
  - Consider dedicating specific nodes to different models based on usage patterns.
  - Use metrics to identify peak usage times and scale accordingly.

By systematically testing configurations and leveraging the monitoring data, you can continuously optimize your multi-node inference setup to achieve the best balance of performance, reliability, and resource efficiency.

4.8 Repository resources

All configuration files, scripts, and dashboards referenced in this guide are available in the ROCm GPU Cluster Networking GitHub repository:

https://github.com/ROCm/gpu-cluster-networking/examples/llm-cluster

The repository includes:

- Docker Compose files for inference nodes (vLLM and SGLang examples)
- API Gateway configurations (LiteLLM and Nginx examples)
- Monitoring stack setup with Prometheus, Grafana, and InfluxDB
- Grafana dashboards for AMD Instinct GPUs and vLLM
- Benchmark scripts for Apache Bench and k6
- Example configuration files and setup scripts

CHAPTER

FIVE

ROCE CLUSTER NETWORK CONFIGURATION GUIDE FOR AMD INSTINCT ACCELERATORS

RDMA over Converged Ethernet (RoCE) is a network protocol can deliver speeds comparable to InfiniBand when running AI/HPC workloads, and offer lower cost than InfiniBand due their compatibility with standard ethernet architecture.

This guide contains instructions for optimizing the performance of a RoCE cluster network at the network interface card (NIC) and switch level with proper configuration, as well as routing directions to mitigate issues like MAC address mismatch (ARP flux) that can occur establishing RDMA sessions on nodes with multiple NICs.

5.1 RoCE configuration for NICs

The specific steps to configure your NIC for RoCE support differ based on the NIC manufacturer. As this guide cannot provide steps for every potential manufacturer, this section provides some high-level recommendations of what to look for when setting up RoCE NICs according to the most common manufacturers, but always defers to manufacturer documentation for complete setup.

5.1.1 Install NIC firmware and driver

NIC drivers typically include a regular ethernet driver, a RoCE driver, and a peer-mem (aka GPU direct RDMA), depending on the vendor.

Specific installation steps vary from vendor to vendor and may differ between NIC models from the same vendor. Here are a few examples of public vendor documentation for installing RoCE drivers:

- Installing RoCE Drivers for Broadcom NICs
- RDMA Support for Intel NICs

Always consult vendor-specific instructions in addition to this guide when configuring your NIC. You may need to reach out directly to the vendor if instructions are not publicly available.

Note

The latest driver for a NIC may require a Linux kernel that is not yet supported by the AMD ROCm/amdgpu software stack. Before updating, review the supported operating systems for ROCm and verify the driver kernel is one supported by the version of ROCm you have installed.

5.1.2 Set static NIC speed

Most high-speed network adapters support multiple speeds and are often configured with a default "auto-negotiation" feature that dynamically sets the speed based on network conditions. It's a best practice to disable this feature and configure a single static network speed instead. This avoids unexpected changes in speed and simplified debugging if you encounter performance issues.

5.1.3 Enable RoCE support mode

Most RoCE-capable NICs have a feature flag that must be set before they can communicate through RDMA. As the default setting for this feature differs by NIC vendor and model, you must verify all NICs are configured to support RoCE before running tests.

As an example, Broadcom NICs use the `support_rdma` flag to govern this feature. You can check the status with the NICCLI configuration tool:

Shell output

```
$ sudo niccli -i 3 nvm -getoption support_rdma -scope 0

support_rdma = False
```

Command

```
sudo niccli -i <NIC index> nvm -getoption support_rdma -scope <scope index>
```

In this case, the NIC is not configured to support RoCE, so run `nvm -setoption` to enable it:

Shell output

```
$ sudo niccli -i 3 nvm -setoption support_rdma -value 1 -scope 0

support_rdma is set successfully
```

Command

```
sudo niccli -i <NIC index> nvm -setoption support_rdma -value <value> -scope <scope␣
↪index>
```

Other vendors use different utilities and flags to control this setting; refer to vendor-specific documentation in these scenarios. You can also refer to the NICCLI configuration scripts provided in the cluster networking github to review and configure RDMA support in bulk on each NIC in a node.

5.1.4 Enable PCIe relaxed ordering

Configuring relaxed ordering for your NICs can offer performance improvement by changing the ordering rules that govern data transfers in the base PCIe specification. As with RoCE support, how to enable this feature differs based on vendor and NIC model, but examples from Broadcom NICs are provided in this guide as a starting framework.

> **Note**
>
> NIC configuration is only one part of enabling PCIe relaxed ordering. You must also ensure your server architecture supports relaxed ordering and that it is enabled in BIOS on each node in your cluster.

To check relaxed ordering on a Broadcom NIC, use NICCLI:

Shell output

```
$ sudo niccli -i 3 nvm -getoption pcie_relaxed_ordering

pcie_relaxed_ordering = Enabled
```

Command

```
sudo niccli -i <NIC index> nvm -getoption pcie_relaxed_ordering
```

If `pcie_relaxed_ordering` shows a `disabled` value, you can enable it with this command:

Shell output

```
$ sudo niccli -i 3 nvm -setoption pcie_relaxed_ordering -value 1

pcie_relaxed_ordering is set successfully
Please reboot the system to apply the configuration
```

Command

```
sudo niccli -i <NIC index> nvm -setoption pcie_relaxed_ordering -value <value>
```

For other vendors, refer to vendor-specific documentation for information about how to verify and enable this setting. You can also refer to the NICCLI configuration scripts provided in the cluster networking github to review and configure relaxed ordering in bulk on each NIC in a node.

5.1.5 Disable ACS and set IOMMU passthrough

On any nodes hosting GPUs, ensure you disable ACS and configure IOMMU passthrough to ensure peer to peer transfer between NICs and GPUs functions as expected.

To disable ACS, use the disable ACS script provided on the cluster networking github. To set IOMMU passthrough on a Linux system, add `iommu=pt` to the `GRUB_CMDLINE_LINUX_DEFAULT` entry in `/etc/default/grub`, then run `sudo update-grub`. You can see a more detailed flow at GRUB settings and Issue #5: Application hangs on Multi-GPU systems.

5.1.6 Enable DCQCN through QoS configuration

Data Center Quantized Congestion Notification (DCQCN) is a traffic control method achieved by enabling two features, Explicit Congestion Notification (ECN) and Priority Flow Control (PFC), to support end-to-end lossless ethernet in a data center environment.

In communication between NICs, ECN detects congestion in PCIe switch buffers and alerts the endpoint (receiving NIC) through packet ECN bits. The receiving NIC then transmits a congestion notification package (CNP) to the sending NIC to reduce the transfer rate. If congestion is still too high for a specific traffic class with ECN in effect, PFC pauses traffic for that class until congestion is resolved.

ECN and PFC are configured per NIC through quality of service (QoS) parameters. Refer to your vendor-specific documentation on how to set the following parameters for your NICs:

- RoCE priority class.
- Enable PFC on RoCE priority class.

- Set CNP priority class (usually the highest priority of 7).

Example of default QoS configuration on a Broadcom Thor2 NIC

Shell output

```
# RoCE v2 packets are marked with a  DSCP value 26 and use Priority 3 internally
# CNP packets are marked with a DSCP value 48 and use Priority 7 internally
# PFC is enabled for Priority 3 traffic
# Three Traffic classes are set up, TC0 for non RoCE traffic, TC1 for RoCE traffic, and␣
↪TC2 for CNP traffic
# RoCE and non-RoCE traffic share ETS bandwidth of 50% each. The ETS bandwidth share␣
↪applies only when the actual traffic is available to use the bandwidth share. In the␣
↪absence of non-RoCE traffic, all the available bandwidth will be used by RoCE and vice-
↪versa.
# CNP traffic is treated as ETS Strict Priority

$ sudo niccli -dev 1 get_qos

IEEE 8021QAZ ETS Configuration TLV:
         PRIO_MAP: 0:0 1:0 2:0 3:1 4:0 5:0 6:0 7:2
         TC Bandwidth: 50% 50% 0%
         TSA_MAP: 0:ets 1:ets 2:strict
IEEE 8021QAZ PFC TLV:
         PFC enabled: 3
IEEE 8021QAZ APP TLV:
         APP#0:
         Priority: 7
         Sel: 5
         DSCP: 48

         APP#1:
         Priority: 3
         Sel: 5
         DSCP: 26

         APP#2:
         Priority: 3
         Sel: 3
         UDP or DCCP: 4791

TC Rate Limit: 100% 100% 100% 0% 0% 0% 0% 0%

$ sudo niccli -dev 1 dump pri2cos

Base Queue is 0 for port 0
----------------------------
Priority   TC   Queue ID
------------------------
0          0    4
1          0    4
2          0    4
3          1    0
4          0    4
```

(continues on next page)

(continued from previous page)

```
5          0    4
6          0    4
7          2    5

$ sudo niccli -dev 1 get_dscp2prio

dscp2prio mapping:
         priority:7  dscp: 48
         priority:3  dscp: 26
```

Commands

```
sudo niccli -dev <NIC index> get_qos

sudo niccli -dev <NIC index> dump pri2cos

sudo niccli -dev <NIC index> get_dscp2prio
```

5.1.6.1 NIC QoS troubleshooting

Sometimes, the default QoS on a NIC may differ significantly from that recommended by Broadcom in the BCM957608 Ethernet Networking Guide for AMD Instinct MI300X GPU Clusters (pages 38-42), even if the RoCE profile has been set in NVM.

If you determine this is the case for any of your NICs, follow these steps to resolve:

1. Run the `install.sh` script provided in the Broadcom NIC release package.

   ```
   $ cd bcm5760x_<version.x.y.z>/utils/linux_installer/

   $ bash install.sh -i <interface_name> -o ECNPFC -f -b 50 -w
   ```

   The flags can be understood as follows:

   - `-o ECNPFC` - Enables PFC and sets traffic priority, sets DSCP values for RoCE traffic and CNP traffic.
   - `-b 50` - Sets RoCE to occupy a minimum 50% of bandwidth.
   - `-w` - Assumes proper firmware is already installed and skips it.

2. Once the script completes, run `sudo reboot` to prevent the RoCE driver warning about unmatched DSCP values.
3. Verify the QoS is now correct. Run `sudo niccli -dev 1 getqos` and ensure it matches the example in the previous section, paying particular attention to PFC state, traffic classes, and DSCP values.

5.2 RoCE configuration for network switches

You will need direct or remote access to switches in your cluster to configure them for optimal data transfer over a RoCE network. This guide provides instructions for Dell and Arista switches using SONiC and Arista EOS respectively.

5.2.1 Switch authentication and configuration terminal access

The first step is to log in to the switch and elevate your permissions so that you can change configurations.

Dell

1. Access your switch CLI with ssh.
2. Run `sonic-cli` as a command.
3. Run `configure` or `configure terminal` as a command to enter configuration mode.
4. Run `exit` as a command at any time to leave configuration mode.

Arista

1. Access your switch CLI with ssh.
2. Run `enable` as a command to receive elevated privileges.
3. Run `configure terminal` as a command to enter configuration mode.
4. Run `exit` as a command at any time to leave configuration mode.

Juniper

1. Access your switch CLI with ssh.
2. Run `edit` or `configure` as a command to enter configuration mode.
3. Run `exit` as a command at any time to leave configuration mode.

Cisco

1. Access your switch CLI with ssh.
2. Run `configure` as a command to enter configuration mode.
3. Run `exit` as a command at any time to leave configuration mode.

5.2.2 Enable RoCE support

To enable RoCE support on your switch, you may need to enable a specific RoCE mode or set up QoS policies to prioritize RoCE traffic. The exact steps depend on the switch vendor and model, but general instructions for Dell, Arista, Juniper, and Cisco switches are provided below.

Dell

1. While in configuration mode, run `roce enable` as a command.
2. Reboot the switch when or if prompted.

Arista

Arista EOS supports RoCE communication by default. Instead, ensure the PFC for the RoCE traffic class is enabled on each port that handles RoCE traffic.

Juniper

To support RoCE communications in JunOS, enter configuration mode from the command line and set the following configuration statements:

1. From the CLI, run `start shell`.
2. Create a custom configuration file in `/var/tmp`: `vi /var/tmp/<file-name>.conf`.
3. Add the following policy statements to the file:

```
policy-options {
    policy-statement lb-perpacket {
        then {
            load-balance per-packet;
        }
    }
}
chassis {
    maximum-ecmp 128;
    fpc 0 {
        traffic-manager {
            buffer-monitor-enable;
        }
    }
}
routing-options {
    maximum-ecmp 128;
    forwarding-table {
        export lb-perpacket;
        ecmp-fast-reroute;
    }
}
forwarding-options {
    hash-key {
        family inet {
            layer-3;
            layer-4;
        }
    }
    enhanced-hash-key {
        ecmp-dlb {
            flowlet {
                inactivity-interval 256;
                flowset-table-size 2048;
                reassignment {
                    prob-threshold 3;
                    quality-delta 6;
                }
            }
            ether-type {
                ipv4;
                ipv6;
            }
            sampling-rate 1000000;
        }
    }
}
```

4. Save the file, then run `exit` to leave shell mode.
5. Run `configure` to enter configuration mode.
6. Run `load merge /var/tmp/<file-name>.conf` to load the configuration file.
7. Run `show | compare` to verify your changes, then `commit` to submit them.

Cisco

Cisco NX-OS supports RoCE communication by default. Instead, ensure the PFC for the RoCE traffic class is enabled on each port that handles RoCE traffic.

5.2.3 Implement standard extended naming for switch interfaces

Dell recommends what is referred to as the "standard" or "standard extended" naming convention for switch interfaces. The naming scheme is understood as `Eth<line_card_id>/<port_id>/[breakout_port_id]`. In a fixed switch this naming scheme simplifies matching each port to its front-panel label, where `Eth1/16/[x]` corresponds to port 16 as physically labeled on the switch. The line card ID also remains 1 since there is a single line card.

However, if multiple line cards are present in a modular switch like the Arista 7388X5 series, additional effort is required to match the port name to its physical label.

Dell

1. While in configuration mode, run `interface-naming standard extended` as a command.
2. Run `write memory` as a command.
3. Log out of the switch, then log back in to view the change in interface names.

```
Interface Name Vendor Part No. Serial No. QSA Adapter Qualified
-------------------------------------------------------------------------------------
↪-------------------------------------------------
Eth1/1 QSFP56-DD 400GBASE-SR8-AEC-3.0M DELL EMC DH11M CN0F9KR711I0060 N/A True
Eth1/2 QSFP56-DD 400GBASE-SR8-AEC-3.0M DELL EMC DH11M CN0F9KR711I0042 N/A True
Eth1/3 QSFP56-DD 400GBASE-SR8-AEC-3.0M DELL EMC DH11M CN0F9KR70CO0095 N/A True
...
Eth1/64      QSFP56-DD 400GBASE-SR8-AEC-3.0M         DELL EMC          DH11M       ␣
↪     CN0F9KR70CM0041   N/A               True
Eth1/65      N/A                                     N/A               N/A         ␣
↪     N/A               N/A               False
Eth1/66      N/A                                     N/A               N/A         ␣
↪     N/A               N/A               False
```

Arista

Arista switches are pre-configured to use the standard extended naming convention, no additional action is required.

Juniper

Juniper switches are pre-configured to use the standard extended naming convention, no additional action is required.

Cisco

Cisco switches are pre-configured to use the standard extended naming convention, no additional action is required.

5.2.4 Verify all connected transceivers are detected

Once all physical cluster cabling is complete, check that your switch transceivers are detected and online.

Dell

1. While in configuration mode, run `show interface transceiver summary | no-more`.
2. Verify all transceivers appear in the interface list.

```
--------------------------------------------------------------------------------
↪-------------------------------------------------
Interface    Name                                    Vendor            Part No.  ␣
↪     Serial No.        QSA Adapter       Qualified
--------------------------------------------------------------------------------
↪-------------------------------------------------
Eth1/1       QSFP56-DD 400GBASE-SR8-AEC-3.0M         DELL EMC          DH11M     ␣
↪     CN0F9KR711I0060   N/A               True
Eth1/2       QSFP56-DD 400GBASE-SR8-AEC-3.0M         DELL EMC          DH11M     ␣
↪     CN0F9KR711I0042   N/A               True
Eth1/3       QSFP56-DD 400GBASE-SR8-AEC-3.0M         DELL EMC          DH11M     ␣
↪     CN0F9KR70CO0095   N/A               True
Eth1/4       QSFP56-DD 400GBASE-SR8-AEC-3.0M         DELL EMC          DH11M     ␣
↪     CN0F9KR70CQ0026   N/A               True
...
```

Arista

1. While in configuration mode, run `show inventory`.
2. Verify all transceivers appear in the interface list.

```
System has 54 switched transceiver slots
  Port Manufacturer     Model            Serial Number    Rev
  ---- ---------------- ---------------- ---------------- ----
  1    Arista Networks  DCS-7050TX-72Q
  2    Arista Networks  DCS-7050TX-72Q
  3    Arista Networks  DCS-7050TX-72Q
  4    Arista Networks  DCS-7050TX-72Q
  5    Arista Networks  DCS-7050TX-72Q
```

Juniper

1. While in configuration mode, run `show chassis hardware` or `show interfaces diagnostics optics`.
2. Verify all transceivers appear in the interface list.

Cisco

1. From the command line, run `show interface transceiver | include Eth|type`.
2. Verify all transceivers appear in the interface list.

```
Ethernet1/1/1
Ethernet1/1/2
Ethernet1/2/1
Ethernet1/2/2
Ethernet1/3/1
....
```

5.2.5 Configure switch links

Link training is used to calibrate the network signal between two devices over a physical, copper-based ethernet cable. This is typically a requirement when running a direct access cable (DAC) but discouraged for optics (Dell SmartFabric OS10 User Guide Release 10.5.6).

If you require link training, enable it on both your NIC and switch OS.

Broadcom NIC

You can run `niccli` to enable link training on your NICs:

```
niccli -dev 1 nvm -setoption link_training -value [0|1] -scope 0
```

Dell

If your switch ports are connected to non-DAC cables you should disable link training:

1. While in configuration mode, run `interface range` as a command to select an interface range such as `Eth 1/1-1/32`.
2. Run `no shutdown` as a command .
3. Run `no standalone-link-training` as a command.

```
$ (config)# interface range Eth 1/1-1/32

%Info: Configuring only existing interfaces in range

$ (config-if-range-eth**)# no shutdown

$ (config-if-range-eth**)# no standalone-link-training
```

For switch ports connected to DAC cables:

1. While in configuration mode, run `interface range` as a command to select an interface range such as `Eth 1/1-1/32`.
2. Run `no shutdown` as a command.
3. Run `standalone-link-training` as a command.

```
$ (config-if-range-eth**)# interface range Eth 1/33-1/64

%Info: Configuring only existing interfaces in range

$ (config-if-range-eth**)# no shutdown

$ (config-if-range-eth**)# standalone-link-training
```

Arista

1. While in configuration mode, run `interface Ethernet` as a command to select an interface range such as `1-32`.
2. Run `no shutdown` as a command.

```
$ (config)# interface Ethernet 1-32

$ (config-if-Et1-32)# no shutdown
```

Juniper

To enable link training in JunOS, enter configuration mode from the command line and set the following configuration statements:

1. From the CLI, run `start shell`.
2. Create a custom configuration file in `/var/tmp`: `vi /var/tmp/<file-name>.conf`.
3. Add the following policy statements to the file:

```
interfaces {
    et-0/0/0 {
        number-of-sub-ports 2;
        speed 400g;
        mtu 9216;
    }
    et-0/0/0:0 {
        mtu 9216;
        ether-options {
            no-flow-control;
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members 200;
                }
            }
        }
    }
    et-0/0/0:1 {
        mtu 9216;
        ether-options {
            no-flow-control;
        }
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members 200;
                }
            }
        }
    }
```

4. Save the file, then run `exit` to leave shell mode.
5. Run `configure` to enter configuration mode.
6. Run `load merge /var/tmp/<file-name>.conf` to load the configuration file.
7. Run `show | compare` to verify your changes, then `commit` to submit them.

Cisco

If connecting a 400 gbps NIC to an 800 gbps switch interface, create a breakout of the switch interface to 2 x 400 gbps mode using the `interface breakout module` NX-OS command while in configuration mode.

1. Run `configure` to enter configuration mode.
2. Run `interface breakout module 1 port <port range> map 400g-2x` to split the 800G interface across two 400G interfaces for the defined port range, such as `1-64`.
3. Run `int eth <int range>` to select the interface range, such as `1/1-64`.
4. Run `no shutdown`.

Important

Some Arista switches are observed to not support autonegotiation or standalone link training on the edge ports (eth1, eth2, eth31-34, eth63, eth64) when running older versions of Arista EOS. This causes a situation where you must either only use DACs in the switch ports that can enable link training or disable link training on all ports and the NIC to allow full usage of the switch.

Since neither approach is ideal, the preferred solution is to update Arista EOS to version 4.33.0F or later, which should allow standalone link training and autonegotiation on all ports.

5.2.5.1 Link training support matrix

Refer to the table below as a reference for whether standalone link training should be enabled or not based on your switch OS and cable type.

Switch OS	cable type	port speed	link training	BRCM NIC link training
Arista EOS >= 4.33.0F	optics	400 - no autoneg	off	off
Arista EOS >= 4.33.0F	DAC	400 - no autoneg	on	on
JunOS	optics	400 - no autoneg	off	off
JunOS	DAC	400 - no autoneg	on	on
Dell Sonic	optics	400 - no autoneg	off	off
Dell Sonic	DAC	400 - no autoneg	on	on
Dell OS10	optics	400 - no autoneg	N/A	off
Dell OS10	DAC	400 - no autoneg	N/A	on
NX-OS 10.6(1)	optics	400 - no autoneg	N/A	off
NX-OS 10.6(1)	DAC	400 - no autoneg	N/A	off

> **Note**
>
> If you are using a Cisco switch running NX-OS with DAC cables, leave link training disabled on NIC and switch initially. If you find links aren't coming up, enable link training using the instructions provided in the previous section.

5.2.6 Match switch QoS configuration to NIC for DCQCN

It's critical that the configuration you set up on a NIC be matched by your switch. Refer to the BCM957608 Ethernet Networking Guide for AMD Instinct MI300X GPU Clusters (pages 38-42) as a reference; the configuration detailed there is a good baseline for most switches and NICs.

DCQCN in Arista EOS can be summarized as:

- Enable PFC on all ports.
- RoCE and CNP DSCP values match those set on the NIC.
- Enable ECN and configure ECN thresholds.

Example - DCQCN configuration on an Arista 7388 switch

```
!!! Map traffic RoCE and CNP classes (TC) to their DSCP to match NIC config
qos map traffic-class 3 to dscp 26
qos map traffic-class 7 to dscp 48

!!! Define PFC settings and ECN buffer thresholds for RoCE traffic. Note that ECN buffer␣
↪thresholds can be optimized later depending on the workload running on the cluster.
qos profile QOS_ROCE_DCQCN
   qos trust dscp
   priority-flow-control on
   priority-flow-control priority 3 no-drop
   !
   uc-tx-queue 3
      random-detect ecn minimum-threshold 2000 segments maximum-threshold 10000 segments␣
↪max-mark-probability 20 weight 0
      random-detect ecn count
!

!!! Sample switch port configuration. Note speed was set to 200GbE because the NICs on␣
↪nodes had a 200GbE speed.
interface Ethernet2/5/1
   load-interval 2
   mtu 9214
   speed 200g-4
   error-correction encoding reed-solomon
   ip address 1.1.122.14/31
   phy link training
   service-profile QOS_ROCE_DCQCN
!
interface Ethernet2/5/5
   load-interval 2
   mtu 9214
   speed 200g-4
```

(continues on next page)

(continued from previous page)

```
   error-correction encoding reed-solomon
   ip address 1.1.122.24/31
   phy link training
   service-profile QOS_ROCE_DCQCN
!
```

For SONIC running on Dell switches, most of the configuration below will be auto generated when the command `enable roce` is run. Just make sure that the QoS configuration generated on the switch match those on the NICs.

Example - DCQCN configuration on Dell Z9664f-O64 switch using sonic-cli

```
!!! Configure ECN buffer thresholds for RoCE traffic. Thresholds can be adjusted later␣
↪depending on network performance.
!
qos wred-policy ROCE
green minimum-threshold 2048 maximum-threshold 12480 drop-probability 15
ecn green
!
qos scheduler-policy ROCE
!
queue 0
type dwrr
weight 50
!
queue 3
type dwrr
weight 50
!
queue 4
type dwrr
weight 50
!
queue 6
type strict
!
qos map dscp-tc ROCE
dscp 0-3,5-23,25,27-47,49-63 traffic-class 0
dscp 24,26 traffic-class 3
dscp 4 traffic-class 4
dscp 48 traffic-class 6
!
qos map dot1p-tc ROCE
dot1p 0-2,5-7 traffic-class 0
dot1p 3 traffic-class 3
dot1p 4 traffic-class 4
!
qos map tc-queue ROCE
traffic-class 0 queue 0
traffic-class 1 queue 1
traffic-class 2 queue 2
traffic-class 3 queue 3
traffic-class 4 queue 4
```

(continues on next page)

(continued from previous page)

```
traffic-class 5 queue 5
traffic-class 6 queue 6
traffic-class 7 queue 7
!
qos map tc-pg ROCE
traffic-class 3 priority-group 3
traffic-class 4 priority-group 4
traffic-class 0-2,5-7 priority-group 7
!
qos map pfc-priority-queue ROCE
pfc-priority 0 queue 0
pfc-priority 1 queue 1
pfc-priority 2 queue 2
pfc-priority 3 queue 3
pfc-priority 4 queue 4
pfc-priority 5 queue 5
pfc-priority 6 queue 6
pfc-priority 7 queue 7
!
qos map pfc-priority-pg ROCE
pfc-priority 0 pg 0
pfc-priority 1 pg 1
pfc-priority 2 pg 2
pfc-priority 3 pg 3
pfc-priority 4 pg 4
pfc-priority 5 pg 5
pfc-priority 6 pg 6
pfc-priority 7 pg 7
!
hardware
!
access-list
counters per-entry
!
tcam
!
line vty
service-policy type qos in oob-qos-policy
!
interface Loopback 0
ip address 192.168.0.1/32
!
interface Eth1/1
description Spine-Eth1/1
mtu 9216
speed 400000
fec RS
unreliable-los auto
no shutdown
ipv6 enable
ars bind port_pro
queue 3 wred-policy ROCE
```

(continues on next page)

(continued from previous page)

```
queue 4 wred-policy ROCE
scheduler-policy ROCE
qos-map dscp-tc ROCE
qos-map dot1p-tc ROCE
qos-map tc-queue ROCE
qos-map tc-pg ROCE
qos-map pfc-priority-queue ROCE
qos-map pfc-priority-pg ROCE
priority-flow-control priority 3
priority-flow-control priority 4
priority-flow-control watchdog action drop
priority-flow-control watchdog on detect-time 200
priority-flow-control watchdog restore-time 400
!
interface Eth1/2
description Spine-Eth1/2
mtu 9216
speed 400000
fec RS
unreliable-los auto
no shutdown
ipv6 enable
ars bind port_pro
queue 3 wred-policy ROCE
queue 4 wred-policy ROCE
scheduler-policy ROCE
qos-map dscp-tc ROCE
qos-map dot1p-tc ROCE
qos-map tc-queue ROCE
qos-map tc-pg ROCE
qos-map pfc-priority-queue ROCE
qos-map pfc-priority-pg ROCE
priority-flow-control priority 3
priority-flow-control priority 4
priority-flow-control watchdog action drop
priority-flow-control watchdog on detect-time 200
priority-flow-control watchdog restore-time 400
!
interface Eth1/3
description Spine-Eth1/3
mtu 9216
speed 400000
fec RS
unreliable-los auto
no shutdown
ipv6 enable
ars bind port_pro
queue 3 wred-policy ROCE
queue 4 wred-policy ROCE
scheduler-policy ROCE
qos-map dscp-tc ROCE
qos-map dot1p-tc ROCE
```

(continues on next page)

(continued from previous page)

```
qos-map tc-queue ROCE
qos-map tc-pg ROCE
qos-map pfc-priority-queue ROCE
qos-map pfc-priority-pg ROCE
priority-flow-control priority 3
priority-flow-control priority 4
priority-flow-control watchdog action drop
priority-flow-control watchdog on detect-time 200
priority-flow-control watchdog restore-time 400
!
```

For JunOS on Juniper switches, you can set the following configuration statements to implement the necessary QoS.

Example - DCQCN configuration for a Juniper QFX5240 switch using JunOS

```
classifiers {
    dscp mydscp {
        forwarding-class CNP {
            loss-priority low code-points 110000;
        }
        forwarding-class NO-LOSS {
            loss-priority low code-points 011010;
        }
    }
}

drop-profiles {
    dp1 {
        interpolate {
            fill-level [ 45 90 ];
            drop-probability [ 0 100 ];
        }
    }
}

shared-buffer {
    ingress {
        buffer-partition lossless {
            percent 80;
        }
        buffer-partition lossless-headroom {
            percent 10;
        }
        buffer-partition lossy {
            percent 10;
        }
    }
    egress {
        buffer-partition lossless {
            percent 80;
        }
        buffer-partition lossy {
```

(continues on next page)

(continued from previous page)

```
            percent 10;
        }
    }
}

forwarding-classes {
    class CNP queue-num 3;
    class NO-LOSS queue-num 4 no-loss pfc-priority 3;
}

congestion-notification-profile {
    cnp {
        input {
            dscp {
                code-point 011010 {
                    pfc;
                }
            }
        }
        output {
            ieee-802.1 {
                code-point 011 {
                    flow-control-queue 4;
                }
            }
        }
    }
}
interfaces {
    et-* {
        congestion-notification-profile cnp;
        scheduler-map sm1;
        unit * {
            classifiers {
                dscp mydscp;
            }
        }
    }
}
scheduler-maps {
    sm1 {
        forwarding-class CNP scheduler s2-cnp;
        forwarding-class NO-LOSS scheduler s1;
    }
}
schedulers {
    s1 {
        drop-profile-map loss-priority any protocol any drop-profile dp1;
        explicit-congestion-notification;
    }
    s2-cnp {
        transmit-rate percent 5;
```

(continues on next page)

(continued from previous page)

```
        priority strict-high;
    }
}
```

For NX-OS on Cisco switches, you can set the following configuration statements to implement the necessary QoS.

Example - DCQCN configuration for a Cisco N9364E-SG2-O switch using NX-OS

```
!
policy-map type network-qos qos_network
  class type network-qos c-8q-nq3
    mtu 9216
    pause pfc-cos 3
  class type network-qos c-8q-nq-default
    mtu 9216
!
class-map type qos match-any CNP
  match dscp 48
class-map type qos match-any ROCEv2
  match dscp 26
policy-map type qos QOS_CLASSIFICATION
  class ROCEv2
    set qos-group 3
  class CNP
    set qos-group 7
  class class-default
    set qos-group 0
!
policy-map type queuing QOS_EGRESS_PORT
  class type queuing c-out-8q-q6
    bandwidth remaining percent 0
  class type queuing c-out-8q-q5
    bandwidth remaining percent 0
  class type queuing c-out-8q-q4
    bandwidth remaining percent 0
  class type queuing c-out-8q-q3
    bandwidth remaining percent 50
    random-detect minimum-threshold 950 kbytes maximum-threshold 3000 kbytes drop-
↪probability 7 weight 0 ecn
  class type queuing c-out-8q-q2
    bandwidth remaining percent 0
  class type queuing c-out-8q-q1
    bandwidth remaining percent 0
  class type queuing c-out-8q-q-default
    bandwidth remaining percent 50
  class type queuing c-out-8q-q7
    priority level 1
system qos
  service-policy type network-qos qos_network
  service-policy type queuing output QOS_EGRESS_PORT
!
interface Ethernet1/1/1 - 2
```

(continues on next page)

(continued from previous page)

```
  mtu 9216
  priority-flow-control mode on
  priority-flow-control watch-dog-interval on
  service-policy type qos input QOS_CLASSIFICATION
  ip address 10.1.0.1/31
  no shutdown
!
```

5.3 Backend network routing methods for preventing ARP flux

ARP flux occurs when an IP address is mapped to an incorrect MAC address in the ARP table. This is a known problem in Linux hosts with multiple network interfaces on the same subnet, as any ARP request for an IP address to a host will be answered by every available interface on that host.

For an HPC/AI cluster, an incorrect MAC address in the ARP table can have several impacts on RDMA traffic:

- Communication may fail if the interface corresponding to the returned (incorrect) MAC address has no open RDMA session.
- Multiple IP addresses map to the same MAC address resulting in one NIC receiving excessive traffic while other NICs are idle, causing a performance bottleneck.

This section discusses two methods for mitigating the effects for ARP flux: IPV4 configuration at the host level or VLAN/L3 routing at the switch level.

5.3.1 Preventing ARP Flux with Linux Host IPV4 Configuration

You can set the IPV4 `sysctl` parameters for individual Linux hosts to prevent ARP flux. This method is most effective when systems across the network are stable and do not frequently change OS.

To temporarily force only the correct NIC to respond to ARP, run the following commands:

```
$ sysctl -w net.ipv4.conf.all.arp_announce=1 # Ignore NICs not on subnet

$ sysctl -w net.ipv4.conf.all.arp_ignore=2 # ignore NICs not matching exact IP addr
```

To make the change permanent, add these lines to `/etc/sysctl.conf` and reboot:

```
net.ipv4.conf.all.arp_announce = 1
net.ipv4.conf.all.arp_ignore = 2
```

5.3.2 Preventing ARP Flux with individual subnets and L3 routing

Instead of configuring the host's IPV4 parameters, you can leverage your network switches to isolate each NIC on a unique subnet. ARP requests can then be sent through inter-VLAN or point-to-point routing to only reach one NIC at a time.

5.3.2.1 Backend network routing with VLAN

Routing with VLANs ensures any two backend network NICs can communicate with one another while preventing ARP flux.

The requirements for inter-VLAN routing are as follows:

- The number of VLANs must equal the number of backend NICs per host.

- For each host, a NIC is routed to only one switch VLAN. NIC1 on each host is routed to VLAN2, NIC2 on each host to VLAN2, and so on.
- If using SONIC as the switch OS, each VLAN is assigned an IP address on the switch side. On the server side, the VLAN IP address is specified as the gateway of the interface.

Switch VLAN-based routing - Host 1 example with 8 NICs

```
network:
  ethernets:
    eth1:
      mtu: 9000
      addresses:
      - 192.168.2.1/24  # Unique subnet 192.168.2.X/24
      routing-policy:
        - from: 192.168.2.1
          table: 102
      routes:   # Everything from this interface routes to VLAN with IP address 192.168.
↪2.254
        - to: 0.0.0.0/0
          via: 192.168.2.254    # VLAN IP address specified in the switch
          table: 102
    eth2:
      mtu: 9000
      addresses:
      - 192.168.3.1/24
      routing-policy:
        - from: 192.168.3.1
          table: 103
      routes:
        - to: 0.0.0.0/0
          via: 192.168.3.254
          table: 103
    eth3:
      mtu: 9000
      addresses:
      - 192.168.4.1/24
      routing-policy:
        - from: 192.168.4.1
          table: 104
      routes:
        - to: 0.0.0.0/0
          via: 192.168.4.254
          table: 104
    eth4:
      mtu: 9000
      addresses:
      - 192.168.5.1/24
      routing-policy:
        - from: 192.168.5.1
          table: 105
      routes:
        - to: 0.0.0.0/0
          via: 192.168.5.254
```

(continues on next page)

(continued from previous page)

```
        table: 105
    eth5:
      mtu: 9000
      addresses:
      - 192.168.6.1/24
      routing-policy:
        - from: 192.168.6.1
          table: 106
      routes:
        - to: 0.0.0.0/0
          via: 192.168.6.254
          table: 106
    eth6:
      mtu: 9000
      addresses:
      - 192.168.7.1/24
      routing-policy:
        - from: 192.168.7.1
          table: 107
      routes:
        - to: 0.0.0.0/0
          via: 192.168.7.254
          table: 107
    eth7:
      mtu: 9000
      addresses:
      - 192.168.8.1/24
      routing-policy:
        - from: 192.168.8.1
          table: 108
      routes:
        - to: 0.0.0.0/0
          via: 192.168.8.254
          table: 108
    eth8:
      mtu: 9000
      addresses:
      - 192.168.9.1/24
      routing-policy:
        - from: 192.168.9.1
          table: 109
      routes:
        - to: 0.0.0.0/0
          via: 192.168.9.254
          table: 109
  version: 2
```

Switch VLAN-based routing - Host 2 example with 8 NICs

```
network:
  ethernets:
```

(continues on next page)

(continued from previous page)

```
    eth1:
      mtu: 9000
      addresses:
      - 192.168.2.2/24
      routing-policy:
        - from: 192.168.2.2
          table: 102
      routes:
        - to: 0.0.0.0/0
          via: 192.168.2.254
          table: 102
    eth2:
      mtu: 9000
      addresses:
      - 192.168.3.2/24
      routing-policy:
        - from: 192.168.3.2
          table: 103
      routes:
        - to: 0.0.0.0/0
          via: 192.168.3.254
          table: 103
    eth3:
      mtu: 9000
      addresses:
      - 192.168.4.2/24
      routing-policy:
        - from: 192.168.4.2
          table: 104
      routes:
        - to: 0.0.0.0/0
          via: 192.168.4.254
          table: 104
    eth4:
      mtu: 9000
      addresses:
      - 192.168.5.2/24
      routing-policy:
        - from: 192.168.5.2
          table: 105
      routes:
        - to: 0.0.0.0/0
          via: 192.168.5.254
          table: 105
    eth5:
      mtu: 9000
      addresses:
      - 192.168.6.2/24
      routing-policy:
        - from: 192.168.6.2
          table: 106
      routes:
```

(continues on next page)

(continued from previous page)

```
      - to: 0.0.0.0/0
        via: 192.168.6.254
        table: 106
  eth6:
    mtu: 9000
    addresses:
    - 192.168.7.2/24
    routing-policy:
      - from: 192.168.7.2
        table: 107
    routes:
      - to: 0.0.0.0/0
        via: 192.168.7.254
        table: 107
  eth7:
    mtu: 9000
    addresses:
    - 192.168.8.2/24
    routing-policy:
      - from: 192.168.8.2
        table: 108
    routes:
      - to: 0.0.0.0/0
        via: 192.168.8.254
        table: 108
  eth8:
    mtu: 9000
    addresses:
    - 192.168.9.2/24
    routing-policy:
      - from: 192.168.9.2
        table: 109
    routes:
      - to: 0.0.0.0/0
        via: 192.168.9.254
        table: 109
version: 2
```

Example - Sonic switch configuration with VLAN definitions

```
interface Vlan1
 description nic1_vlan
 ip address 192.168.2.254/24
!
interface Vlan2
 description nic2_vlan
 ip address 192.168.3.254/24
!
interface Vlan3
 description nic3_vlan
 ip address 192.168.4.254/24
```

(continues on next page)

(continued from previous page)

```
!
interface Vlan4
 description nic4_vlan
 ip address 192.168.5.254/24
!
interface Vlan5
 description nic5_vlan
 ip address 192.168.6.254/24
!
interface Vlan6
 description nic6_vlan
 ip address 192.168.7.254/24
!
interface Vlan7
 description nic7_vlan
 ip address 192.168.8.254/24
!
interface Vlan8
 description nic8_vlan
 ip address 192.168.9.254/24
!

interface Eth1/1
 description "Node1 nic1"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 1
!
interface Eth1/2
 description "Node1 nic2"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 2
!
interface Eth1/3
 description "Node1 nic3"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 3
!
```

(continues on next page)

(continued from previous page)

```
interface Eth1/4
 description "Node1 nic4"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 4
!
interface Eth1/5
 description "Node1 nic5"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 5
!
interface Eth1/6
 description "Node1 nic6"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 6
!
interface Eth1/7
 description "Node1 nic7"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 7
!
interface Eth1/8
 description "Node1 nic8"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 8
!
interface Eth1/9
 description "Node2 nic1"
```

(continues on next page)

(continued from previous page)

```
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 1
!
interface Eth1/10
 description "Node2 nic2"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 2
!
interface Eth1/11
 description "Node2 nic3"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 3
!
interface Eth1/12
 description "Node2 nic4"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 4
!
interface Eth1/13
 description "Node2 nic5"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 5
!
interface Eth1/14
 description "Node2 nic6"
 mtu 9100
 speed 400000
```

(continues on next page)

(continued from previous page)

```
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 6
!
interface Eth1/15
 description "Node2 nic7"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 7
!
interface Eth1/16
 description "Node2 nic8"
 mtu 9100
 speed 400000
 fec RS
 standalone-link-training
 unreliable-los auto
 no shutdown
 switchport access Vlan 8
!
```

5.3.2.2 Backend network routing with /31 subnet point-to-point routing

Since /31 subnets allow only two hosts (one for network and one for broadcast), they can be leveraged to prevent ARP flux in a way similar to VLANs.

The requirements for point-to-point routing are:

- Each NIC on a host must have a /31 network mask (for example, 192.168.131.X/31).
- Each connected backend switch port must have an IP address that the NIC interface can use as a gateway.

Example - point-to-point /31 IPV4 routing host netplan file

```
network:
  ethernets:
    eth1:
      mtu: 9000
      addresses:
      - 192.168.1.1/31
      routing-policy:
      - from: 192.168.1.1
        table: 101
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.0
        table: 101
```

(continues on next page)

(continued from previous page)

```
    eth2:
      mtu: 9000
      addresses:
      - 192.168.1.3/31
      routing-policy:
      - from: 192.168.1.3
        table: 102
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.2
        table: 102
    eth3:
      mtu: 9000
      addresses:
      - 192.168.1.5/31
      routing-policy:
      - from: 192.168.1.5
        table: 103
        routes:
      - to: 0.0.0.0/0
        via: 192.168.1.4
        table: 103
    eth4:
      mtu: 9000
      addresses:
      - 192.168.1.7/31
      routing-policy:
      - from: 192.168.1.7
        table: 104
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.6
        table: 104
    eth5:
      mtu: 9000
      addresses:
      - 192.168.1.9/31
      routing-policy:
      - from: 192.168.1.9
        table: 105
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.8
        table: 105
    eth6:
      mtu: 9000
      addresses:
      - 192.168.1.11/31
      routing-policy:
      - from: 192.168.1.11
        table: 106
      routes:
```

(continues on next page)

(continued from previous page)

```
      - to: 0.0.0.0/0
        via: 192.168.1.10
        table: 106
    eth7:
      mtu: 9000
      addresses:
      - 192.168.1.13/31
      routing-policy:
      - from: 192.168.1.13
        table: 107
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.12
        table: 107
    eth8:
      mtu: 9000
      addresses:
      - 192.168.1.15/31
      routing-policy:
      - from: 192.168.1.15
        table: 108
      routes:
      - to: 0.0.0.0/0
        via: 192.168.1.14
        table: 108
  version: 2
```

Example - Switch configuration for point-to-point /31 IPV4 routing (applicable for Sonic, EOS, NX-OS, and others)

```
!
interface Eth1/1
 description node1-eth1
 ip address 192.168.1.0/31
!
interface Eth1/2
 description node1-eth2
 ip address 192.168.1.2/31
!
interface Eth1/3
 description node1-eth3
 ip address 192.168.1.4/31
!
interface Eth1/4
 description node1-eth4
 ip address 192.168.1.6/31
!
interface Eth1/5
 description node1-eth5
 ip address 192.168.1.8/31
!
```

(continues on next page)

(continued from previous page)

```
interface Eth1/6
 description node1-eth6
 ip address 192.168.1.10/31
!
interface Eth1/7
 description node1-eth7
 ip address 192.168.1.12/31
!
interface Eth1/8
 description node1-eth8
 ip address 192.168.1.14/31
```

CHAPTER

SIX

TROUBLESHOOTING FOR COMMON GPU CLUSTER NETWORKING ISSUES

Despite best efforts, you may run into situations where a cluster is not performing as expected after following the configuration steps outlined in the other guides. Diagnosing these problems can be challenging due to the nature of the data flow in cluster communications. There are many hardware components in the data path where a failure may hay occurred, including but not limited to storage, host memory, host CPU, PCI switches, GPU, network cards, transceivers, network cables, network switches, and more. Many of these hardware components have dedicated firmware or software to control or use them.

The combination of hardware, firmware, and software makes it difficult to isolate some issues. Therefore, the aim of this guide is not to provide an exhaustive list of all the issues a GPU cluster might run into, but provide the you with a general guidance and intuition on where to look if you detect a certain functional or performance issue.

6.1 RCCL Errors

Performance runs with rccl-tests have multiple points of failure due to their interaction with different software components (RDMA libraries and drivers, UCX, MPI, ROCm, and so on). Since several of these components are open-source, error messages often are not RCCL-specific and may be challenging to decipher in the context of a failed run. The table in this section provides a list of common errors and guidance on how to resolve them.

Should you encounter an error that's not covered in this section, run the RDMA perftest `ib_write_bw` benchmark to get at the root of the issue.

Error / Behavior	Solution
System hangs after initiating RCCL run	There are multiple reasons RCCL could hang, including resource limit issues (ulimit), MPI attempting to initialize or run across non-viable interfaces (loopback, docker, virtual machine), or network connectivity. In the case of network issues, the system may hang before the software stack can report anything. • Check and properly set *resource limits*. • Use *mca parameters* to exclude undesired MPI interfaces. `mpirun ... -mca -oob_tcp_if_exclude=docker,lo -mca btl_tcp_if_exclude=docker,lo ...` • *Disable firewall*, if enabled. • Check and resolve any *connectivity issues*. Check RDMA ping, routing for RoCE, subnet manager working for IB setup. • Verify the interface name passed to *NCCL_SOCKET_IFNAME* exists on all the servers. • Check if *ACS is disabled* with `sudo lspci -vvv | grep -i "acsctl"`. If you see `SrcValid+` among any of the ouputs, then ACS isn't disabled. Run the disable ACS script as sudo/root to resolve.
Low performance	Multiple possible causes, check the following: • On baremetal, *amdgpu driver is not loaded*. For virtualized environments, ACS can be enabled, but make sure ATS is as well. • Check *PCIe link status* on all the devices in the CPU-to-NIC packet datapath. This includes host bridges, PCI bridges, NICs, and GPUs. • Check the *system BIOS settings for MI300X*. `XGMI force link width` should be set to 2. Memory interleaving should be set to `Auto`. • Do a GPU subsystem health check using AMD-provided tools like AFHGC. • Update to a Linux kernel that has symbol `ib_register_peer_memory_client` or install `ib_peer_mem` for Broadcom NICs. • Use vendor PCI switch tools to ensure that P2P is enabled in the firmware. Otherwise, reach out to the server OEM to rebuild the PCIe switch firmware with P2P support. • Use *NCCL_IB_HCA* to specify the interface RCCL should run on.
UCX errors	Challenges with UCX often root source at network connectivity problems or RCCL failing to locate the bootstrap interface on a node. • *Disable firewall* if enabled. • Check and resolve any *connectivity issues*. Check RDMA ping, routing for RoCE, subnet manager working for IB setup. • Verify the interface name passed to *NCCL_SOCKET_IFNAME* exists on all the servers.

6.2 RDMA Perftest errors

Error / Behavior	Solution
Couldn't connect to <server_ip>:<port>	Causes include the network port already being in use and general connectivity problems. • Kill any perftest `ib_write|read|send*` processes that may be running. • *Disable firewall*, if enabled. • Check and resolve any *connectivity issues*. Check RDMA ping, routing for RoCE, subnet manager working for IB setup.
Failed to create QP	Occurs due to resource limits (ulimit), no RoCE/InfiniBand driver loaded, or no route to peer. • Check and properly set *resource limits*. • If you're using ROCm memory, verify you enabled it during compilation. • Verify RoCE/InfiniBand driver is loaded. • *Disable firewall*, if enabled. • Ensure all *links are online*. • Check and resolve any *connectivity issues*. Check RDMA ping, routing for RoCE, subnet manager working for IB setup.
Unsupported memory type	Occurs when perftest is compiled without ROCm support. Review and recompile RDMA perftest using *Multi-node Networking Guide instructions*.
libibverbs: Warning: Driver <version> does not support the kernel ABI	The *inbox Linux libraries* are conflicting with the proprietary vendor (Broadcom, Nvidia) RDMA drivers. • Reinstall the RDMA drivers according to vendor instructions.
"Completion with error <x>" on message sizes greater than 1024 bytes	Occurs when *MTU is set to 1500 bytes* on the host and/or switch side. • Set an MTU of 9000 on the host. This supports both RDMA jumbo frames (4096) and TCP/IP jumbo frames (9000). • Set MTU to the maximum allowed value on the switch. This is usually a value slightly above 9000 for most modern switches. You may need to consult vendor documentation for the specific value.
Couldn't initialize ROCm device	Occurs when ROCm and amdgpu are installed but the *amdgpu driver is not loaded*, or the current user/login name has not been added to the video and render groups. • Run `sudo modprobe amdgpu` • Run `sudo usermod -a -G video,render $LOGNAME`, exit the shell, and log in again. Some systems may require a reboot after running these commands.
Host memory RDMA low performance	Investigate PCIe links for *downgraded speed/width* or *BIOS misconfiguration*-particularly XGMI width and memory interleaving. • Check the *system BIOS settings for MI300X*. `XGMI force link width` should be set to 2. Memory interleaving should be set to `Auto`.
GPU RDMA low performance	Multiple possible causes, check the following.

6.3 Causes and resolution for common network failures

6.3.1 Network connectivity issues

When your error reports a network issue and indicates no other hardware or software component, run the RDMA ping utility `rping` to ensure there are no RDMA connectivity issues. Note that some HPC libraries or versions will hang and fail to report a specific error. You should also run `rping` in those scenarios to rule out connectivity issues.

Example commands to run RDMA ping between all backend network paths on 2 servers

```
# Sample scripts to run RDMA ping on all the possible 64 network paths between 2 servers,
# each with 8 NICs connected over a switch or multiple switches.
#
# rping format
# on host1: rping -s -a <host1_nic_ip_addr> -v -C <number_of_pings>
# on host2: rping -c -a <host1_nic_ip_addr> -I <host2_nic_ip_addr> -v -C <number_of_
↪pings>

# host 1 script
# ======================================================
host1_nics="192.168.0.1 192.168.1.1 192.168.2.1 192.168.3.1 192.168.4.1 192.168.5.1 192.
↪168.6.1 192.168.7.1"
host2_nics="192.168.0.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2 192.
↪168.6.2 192.168.7.2"
for server in ${host1_nics}; do
    for client in ${host2_nics}; do
        echo "rping: server: ${server}. Expected client: ${client}"
        rping -s -a ${server} -v -C 4
    done
done

# host2 script (runs after host1 script)
# ======================================================
host1_nics="192.168.0.1 192.168.1.1 192.168.2.1 192.168.3.1 192.168.4.1 192.168.5.1 192.
↪168.6.1 192.168.7.1"
host2_nics="192.168.0.2 192.168.1.2 192.168.2.2 192.168.3.2 192.168.4.2 192.168.5.2 192.
↪168.6.2 192.168.7.2"
for server in ${host1_nics}; do # each NIC on host1 has a pending rping server process
    for client in ${host2_nics}; do # Each NIC on host2 spins a client to respond to the␣
↪rping server on host1
        rping -c -a ${server} -I ${client} -v -C 4
    done
done
```

If RDMA ping uncovers a network connectivity issue, then the next step is to look into NICs that are down, RDMA configuration issues, routing misconfiguration, cabling issues or even bad switch ports.

6.3.1.1 Firewall enabled

When the firewall is enabled, distributed MPI/RCCL jobs hang because the firewall blocks incoming traffic used for MPI initialization and rank discovery. Even if MPI initialization were successful, the job might still fail when the firewall blocks RCCL collectives from receiving incoming data through the backend interfaces.

You can observe by attempting to run an MPI/RCCL job with the firewall active. Even a simple mpi job like `mpirun -np2 --hostfile hosts <hostname>` is likely to hang.

To resolve, disable the firewall with the following commands:

- Ubuntu: `sudo ufw disable`
- RHEL: `sudo systemctl disable firewalld --now`

6.3.1.2 Link status

Due to environmental factors such as temperature, network cable quality, and hardware degradation, links may either go down or alternate between down and up states (flapping). Commands you can use to discover link issues include `ip link show`, `rdma link show`, and `ibstat`.

Links may also go down due to driver and firmware issues. For those cases, run `dmesg` to see if the driver logged any errors.

6.3.1.3 ARP flux and routing misconfiguration

For servers with multiple NICs, having NICs in the same subnet often leads to ARP flux issues, where one interface responds to an ARP request dedicated to another interface on the same host. If the interface that responds doesn't have an RDMA stack (such as a frontend storage NIC), jobs and applications will fail due to RDMA packets getting dropped.

Even if the NIC that responds can process RDMA traffic, there is a risk of the router associating multiple IP addresses to the same NIC and causing a traffic bottleneck while other NICs are idle.

You can observe this behavior by running the RDMA `ib_write_bw` performance test and getting an error on completion. Low bandwidth on RCCL tests are also a potential indicator.

Methods to resolve ARP flux include the following:

- Isolate NICs by placing them in different subnets (example: 192.168.2.1/24, 192.168.3.1/24, and so on).
- Isolate NICs by using point-to-point routing with the /31 netmask.
- Configuring `arp_ignore` and `arp_announce` sysctl settings.

You can review a more detailed explanation for each of these methods in the section on *preventing ARP flux* from the RoCE cluster network configuration guide.

6.3.2 Resource limit restrictions

RCCL and other HPC applications often open numerous files and demand significant pinned memory for each process. In certain scenarios, the default limits the operating system places on open file descriptors (nofile) or memory locked by a process (memlock) may be insufficient for RCCL's requirements.

Signs that process limit resources are too small include:

- `ib_write_bw` runs may return an error: `failed to create QP`.
- RCCL tests hang.
- RCCL experiences segmentation errors.
- `hipMalloc` fails.

You can resolve this by editing `/etc/security/limits.conf` and appending the following lines:

```
* soft memlock unlimited
* hard memlock unlimited
* soft nofile 1048576
* hard nofile 1048576
```

Once saved, log out of the Linux shell and log back in.

6.3.3 Conflicting NIC vendor and Linux inbox RDMA packages

Sometimes a system may have had NIC drivers correctly installed according to vendor instructions (Broadcom, Nvidia), but Linux inbox drivers or libraries were introduced with later packages. This can cause conflicting drivers or libraries that interfere with the normal operation of RDMA applications.

RDMA drive conflicts can be identified by the error `libibverbs: Warning: Driver <x.y.z> does not support the kernel ABI`.

To resolve, reinstall the RoCE drivers according to the vendor instructions.

6.3.4 Low MTU setting

On many Linux distributions the default ethernet MTU is 1500 bytes. This will also be the MTU size for RoCE interfaces unless changed.

An MTU of 1500 is a performance limiter for HPC applications due to aggressive data segmentation. For the best performance, MTU should be set to 9000 on the host and the maximum allowable MTU on the switch, which is greater than 9000 on most high-performance switches. You may need to check your switch documentation for the specific maximum value.

You can identify this error by running `ib_write_bw -a` from RDMA Perftests. The run completed the error message `Completion with error <x>` when the message size is greater than 1500. Reduced performance on RCCL `all_reduce` runs can further corroborate the problem.

6.3.5 AMD drivers not loaded

At the time of publication, it's recommended to manually load AMD drivers after the OS has fully booted for system running MI200 and MI300 series GPUs. If you try to run applications with ROCm without loading the drives, you'll get the following errors:

- `no ROCm-capable device is detected` when running anything from rccl-tests.
- `Couldn't initialize ROCm device` when running `ib_write_bw --use_rocm=n` commands.

To resolve, run `sudo modprobe amdgpu` to load the drivers.

6.3.6 RCCL bootstrap interface mismatch

RCCL needs a bootstrapping interface for management, and requires this interface have an identical name across all nodes. This can cause a problem with RCCL runs if a cluster has been misconfigured with inconsistent interface names and `NCCL_SOCKET_IFNAME` parameter is set to an interface that's not available on some nodes. You may see this issue manifest as failed RCCL runs with an MPI/RCCL error or an indefinite system hang.

One way to diagnose this error is to include the `NCCL_DEBUG=WARN` parameter with RCCL runs. The run returns a `NCCL WARN Bootstrap : no socket interface found` error if there's a problem with the bootstrap interface.

To resolve, ensure the `NCCL_SOCKET_IFNAME` parameter is included in your RCCL commands and that it is assigned an interface that exists on all nodes in the cluster.

6.3.7 Misconfigured LD_LIBRARY_PATH

GPU distributed jobs depend on a deep software stack and the shared libraries of each individual component in the stack must be accessible through the `LD_LIBRARY_PATH` environment variable. Otherwise, jobs will fail because they cannot find OpenMPI, UCX, RCCL, or higher-level application libraries. The default RCCL shared object should be added to `LD_LIBRARY_PATH` when ROCm is installed, but if you download and manually compile a custom version RCCL, you must specify the path to the RCCL library.

You can diagnose this problem through the following error messages:

- `librccl.so not found` or `librccl-net.so not found`
- `libmpi.so not found` or `libprrte.so not found`
- `libuc*.so not found`

To resolve, provide an updated `LD_LIBRARY_PATH` value as a RCCL parameter:

```
mpirun ... -x LD_LIBRARY_PATH=<path_to_ompi>/ompi/lib:<path_to_rocm>/rocm-x.y.z/lib:
↪<path_to_ucx>/ucx-x.y.z/lib:$LD_LIBRARY_PATH ...
```

6.3.8 MPI traffic across loopback, Docker, or VM interface

If there is virtualization or Docker software installed on a Linux system, Open MPI often defaults to using the docker or virtual interface for initialization. This cause the job to fail since the Docker or virtual interface cannot communicate with the others nodes in the cluster.

Methods to diagnose this issue include:

- Depending on the software stack, an Open MPI job may hang indefinitely.
- Depending on the software stack, an Open MPI may return the error message: `send() to socket failed: Connection refused`

To resolve, always exclude Docker or virtual interfaces from jobs when they are present on a node. The parameters `-mca -oob_tcp_if_exclude=virbr0,docker,lo` and `-mca btl_tcp_if_exclude=virbr0,docker,lo` exclude the interfaces from both out-of-band communication and message passing communication. While loopback is typically excluded from Open MPI by default, it should be added to the flags as a best a practice.

6.3.9 BIOS misconfiguration

The default settings in your system BIOS may not be optimal for network performance. For example, if memory interleaving is disabled as a default option in your BIOS you may see notably lower performance in RDMA operations.

Low performance is the most observable indicator of BIOS misconfiguration, but can have multiple possible causes. The best approach to this issue one of prevention by ensuring your system BIOS is in alignment with AMD's optimization guides:

- For MI3XX systems - AMD Instinct MI300X system optimization
- For MI2XX systems - AMD Instinct MI200 system optimization

6.3.10 ACS enabled on baremetal systems

PCIe ACS is a security feature that enforces isolation between PCIe devices by routing all incoming traffic through the PCIe root-complex first as a security checkpoint. For GPU RDMA this is a significant performance bottleneck as each data transfer between the NIC and GPU gains additional latency when passing through the root complex.

When diagnosing low performance, ensuring ACS is disabled for all PCIe devices is a standard practice along with checking BIOS settings and PCIe speeds. You can verify the status of ACS your devices by running `sudo lspci -vvv | grep -i "acsctl"` from the command line:

```
$ sudo lspci -vvv | grep -i "acsctl"

ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid+ TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid+ TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
```

(continues on next page)

(continued from previous page)

```
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl- DirectTrans-
```

In this example, `SrcValid+` indicates a device still has ACS enabled. AMD provides a disable ACS script that you can run on your nodes to systematically disable ACS for all PCIe devices.

Note

Some systems offer a BIOS option to disable PCIe ACS, but before using it you should verify it disables ACS on every PCIe endpoint and bridge. In many cases, the BIOS only disables ACS on a subset of PCIe devices. To disable ACS on all devices, the best practice is to run `setpci` commands in the OS as demonstrated in the disable ACS script.

6.3.11 Kernel NUMA balancing enabled

ROCm uses `hipHostMalloc` to manage NUMA (Non-Uniform Memory Access) pinning, automatically allocating memory from the NUMA node nearest to the GPU and minimizing host-to-GPU transfer times. Kernel NUMA balancing must therefore be disabled to avoid any additional overhead from migrating the memory utilized by ROCm and ensure optimal performance.

There are two ways to disable kernel NUMA balancing:

1. You can temporarily disable kernel NUMA balancing by running `sudo sh -c 'echo 0 > /proc/sys/kernel/numa_balancing'`
2. To permanently disable kernel NUMA balancing, edit `/etc/default/grub` and add the string `numa_balancing=0` to the `GRUB_CMDLINE_LINUX_DEFAULT` line.
3. Run `sudo update-grub && sudo reboot`.

6.3.12 Downgraded PCIe link

Low results from `ib_write_bw` and rccl-tests can occur when a PCIe link in the data path is in a downgraded state, meaning the speed and/or width is lower than it ought to be. Review the *Single-node networking guide instructions* and ensure all PCI links are operating at sufficient capacity.

6.3.13 RCCL traffic going through frontend NICs

When the NICs that carry GPU traffic are not specified, RCCL's default behavior is to use all available RDMA interfaces. This becomes a problem if RDMA interface are being used for frontend services like storage, since the frontend NICs tend to have lower speed than the GPU-connected backend NICs and likely use a different switch as well, which can cause data transfers to make additional network hops or become unroutable to the backend switches.

A general indicator of this issue is lower-than-expected bandwidth on RCCL tests, but you can get more specific by including the `NCCL_DEBUG=info` parameter on jobs and see if frontend NICs are being used to transfer data.

To resolve, always specify the backend NICs by using the `NCCL_IB_HCA` parameter. Usage is detailed in *Multi-node RCCL operations*.

6.3.14 Dynamic load balancing is disabled

In leaf-spine network topologies, relying solely on ECMP (Equal-Cost Multi-Path) with statically hashed paths can result in hotspots, depending on the application. Hotspots occur when specific switch ports become over-utilized during peak traffic periods. To address this issue, dynamic load balancing (DLB) improves ECMP routing by continuously monitoring transmit buffer occupancy and link utilization. This proactive approach enables DLB to efficiently redirect flows to alternative paths, significantly alleviating congestion.

If DLB is not enabled, you may notice that nodes on the same leaf switches show high rccl-tests allreduce bandwidth, but nodes on different leaf switches show lower allreduce bandwidth when traffic crosses the spine switches.

To resolve, review your switch user guide for specific steps to enable DLB.

CHAPTER

SEVEN

HARDWARE SUPPORT MATRIX

This page provides information on hardware that has been verified to work as expected with the directions for single and multi-node network validation defined elsewhere on this site.

7.1 GPU Architecture

The GPU workloads and benchmarks specified in the *Single Node* and *Multi-node* networking guides should work as written on the following microarchitectures, presuming a supported version of ROCm.

7.1.1 MI355X

MI355X GPUs require ROCm 7.0.1 or higher when running GPU applications as this is the earliest version of ROCm where MI355X has certified support.

7.1.2 MI350X

MI350X GPUs require ROCm 7.0.1 or higher when running GPU applications as this is the earliest version of ROCm where MI350X has certified support.

7.1.3 MI325X

MI325X GPUs requires ROCm 6.3.1 or higher when running GPU applications, as this is the earliest version of ROCm where MI325X has certified support.

7.1.4 MI300X

MI300X GPUs can run all specified workloads and benchmarks in the networking guides presuming a supported version of ROCm.

7.1.5 Pre-MI300X

MI200X and MI100X series GPUs can run all specified workloads and benchmarks in the networking guides presuming a supported version of ROCm.

7.2 Supported NICs

When deploying ROCm, compatibility between the accelerators and NICs is critical for ensuring optimized data transfer in high-performance computing environments. The following NICs have been validated for use with AMD Instinct MI300X, MI200, and MI100 accelerators, supporting high-speed interconnects like RoCE v2 (RDMA over Converged Ethernet) and InfiniBand for low-latency, high-throughput communication.

The processes detailed in these guides are validated to run on the following hardware with AMD Instinct™ accelerators:

7.2.1 NICs for AMD Instinct MI355X

Product name	Speed (GB/s)	Interconnect
Pollara 400 AI NIC	400	RoCE v2
Broadcom P2200G	400	RoCE v2
Broadcom P1400GD	400	RoCE v2
Broadcom N1400GD	400	RoCE v2
Broadcom N2200G	400	RoCE v2

7.2.2 NICs for AMD Instinct MI350X

Product name	Speed (GB/s)	Interconnect
Pollara 400 AI NIC	400	RoCE v2
Broadcom P2200G	400	RoCE v2
Broadcom P1400GD	400	RoCE v2
Broadcom N1400GD	400	RoCE v2
Broadcom N2200G	400	RoCE v2

7.2.3 NICs for AMD Instinct MI325X

Product name	Speed (GB/s)	Interconnect
Pollara 400 AI NIC	400	RoCE v2
Broadcom P2200G	400	RoCE v2
Broadcom P1400GD	400	RoCE v2
Broadcom N1400GD	400	RoCE v2
Broadcom N2200G	400	RoCE v2
NVIDIA ConnectX-7 series	400	RoCE v2 / InfiniBand

7.2.4 NICs for AMD Instinct MI300X

Product name	Speed (GB/s)	Interconnect
Pollara 400 AI NIC	400	RoCE v2
Broadcom P2200G	400	RoCE v2
Broadcom P1400GD	400	RoCE v2
Broadcom N1400GD	400	RoCE v2
Broadcom N2200G	400	RoCE v2
NVIDIA ConnectX-7 series	400	RoCE v2 / InfiniBand

7.2.5 NICs for AMD Instinct MI200 and MI100 series

Product name	Speed (GB/s)	Interconnect
Broadcom N2100G	200	RoCE v2
Broadcom N1200G	200	RoCE v2
Broadcom P2100G	200	RoCE v2
Broadcom P1200G	200	RoCE v2
NVIDIA ConnectX-6 series	200	RoCE v2 / InfiniBand

When deploying ROCm, consult the ROCm compatibility matrix to ensure compatibility, and install the latest version appropriate for your operating system and driver support.

Refer to the Broadcom Ethernet Network Adapter User Guide for installation, configuration, and tuning documentation for Broadcom devices.

CHAPTER

EIGHT

CLUSTER ARCHITECTURE AND NETWORK DESIGN

Reference Cluster Design Architecture MI300X/MI325

- 128-1028 GPU Reference Cluster Design
- 1024-8192 GPU Reference Cluster Design

Reference Network Design Architecture 3XX

- 3XX-2K Reference Network Design
- 3XX-4K Reference Network Design
- 3XX-6K Reference Network Design
- 3XX-8K Reference Network Design
- 355-4MW Reference Network Design