MPI4Dask 0.3 User Guide

HIGH-PERFORMANCE BIG DATA TEAM
http://hibd.cse.ohio-state.edu

NETWORK-BASED COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE OHIO STATE UNIVERSITY

Last revised: February 8, 2023

Contents

1 Overview of the MPI4Dask Project

MPI4Dask is a custom version of the Dask Distributed library that contains high-performance MPI communication backend for Dask. The MPI backend in MPI4Dask uses mpi4py over an MVAPICH2 library and targets modern HPC clusters. The current version of MPI4Dask supports both CPU and GPU-based Dask applications. MPI4Dask embodies enhancements to the Dask Distributed package.

MPI4Dask provides point-to-point asynchronous I/O communication *coroutines*, which are non-blocking concurrent operations defined using the `async`/`await` keywords from the Python's `asyncio` framework. If there are any questions, comments or feedback regarding this software package, please post them to (mvapich-discuss@lists.osu.edu).

2 Features

High-level features of MPI4Dask are listed below. New features and enhancements are marked as (NEW).

- (NEW) Based on Dask Distributed 2022.8.1
- Compliant with user-level Dask APIs and packages
- Support for MPI-based communication for GPU-based Dask applications
- Implements point-to-point communication co-routines
- Efficient chunking mechanism implemented for large messages
- Support for MPI-based communication for CPU-based Dask applications
- Built on top of mpi4py over MVAPICH2, MVAPICH2-X, and MVAPICH2-GDR libraries
- Supports starting execution of Dask programs using Dask-MPI
- Tested with
  - (NEW) Mellanox InfiniBand adapters (FDR, EDR, and HDR)
  - (NEW) Various multi-core platforms
  - (NEW) Various benchmarks used by the community (MatMul, Slicing, Sum Transpose, cuDF Merge, etc.)
  - (NEW) Scaling out to hundreds of workers (processes)
  - (NEW) NVIDIA V100 and A100 GPUs

3 Setup Instructions

3.1 Installation Pre-requisites

3.1.1 Miniconda

1. Install Miniconda, which is a free minimal installer for the conda package manager. The following commands download and install latest version of Miniconda3 for Python 3.x:

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

$ chmod a+x Miniconda3-latest-Linux-x86_64.sh

$ ./Miniconda3-latest-Linux-x86_64.sh
```

1. Create a directory for MPI4Dask.

```
$ mkdir root-directory/mpi4dask
```

3.1.2 Modules/Libraries for GPU-based Dask Applications

1. Make sure that relevant/required modules are loaded in the environment. One of the required module is the CUDA toolkit (11.2 in this case). A list of all installed modules on the system can be seen as follows:

```
$ module list
Currently Loaded Modulefiles:
 1) cuda/11.2
```

2. Install pre-requisite libraries including build tools, Python (version 3.8 here), Cython, CUDA tookit (version 11.2 here), Dask packages, RAPIDS libraries, and array packages. Users may install specific versions of these packages. To avoid potential conda environment compatibility check issues, we recommend using the following commands to install:

```
$ cd root-directory/mpi4dask
$ git clone https://github.com/OSU-Nowlab/MPI4Dask-applications.git
$ cd MPI4Dask-applications
All required packages and configuration are included in mpi4dask_cuda.yml.
Default env name is mpi4dask_cuda, which can be modified in the .yml 'name' space.
$ conda env create -f mpi4dask_cuda.yml
```

3.1.3 Modules/Libraries for CPU-based Dask Applications

1. Install pre-requisite libraries including build tools, Python (version 3.8 here), Cython, Dask packages, and array packages. Users may install specific versions of these packages. Also, these packages are installed from three conda channels, including conda-forge, rapidsai, and nvidia, as follows:

```
$ cd root-directory/mpi4dask
$ git clone https://github.com/OSU-Nowlab/MPI4Dask-applications.git
$ cd MPI4Dask-applications
All required packages and configuration are included in mpi4dask.yml.
Default env name is mpi4dask, which can be modified in the .yml 'name' space.
$ conda env create -f mpi4dask.yml
```

3.1.4 Install the MVAPICH2 Library

1. Download and install an appropriate flavor of the MVAPICH2 library from http://mvapich.cse.ohio-state.edu. For running GPU-based Dask applications, the MVAPICH2-GDR library is recommended. For CPU-based Dask applications, either the MVAPICH2 or the MVAPICH2-X library can be used. The MVAPICH2-X software provides advanced features and better performance and, hence, is recommended for CPU-based Dask applications. Please become familiar with advanced configurations of the MVAPICH2 software—this is important to get maximum performance from the hardware system.

   MVAPICH2 user-guides can be seen here:

   - MVAPICH2: https://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-quickstart.pdf
   - MVAPICH2-GDR: https://mvapich.cse.ohio-state.edu/userguide/gdr/
   - MVAPICH2-X: https://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-x-userguide.pdf

2. This guide assumes that an environment variable $MPILIB points to the installation directory of the MVAPICH2 software.

   ```
   $ export MPILIB=/path/to/MVAPICH2-GDR/install/directory
   ```

3. It is important to make sure that the MVAPICH2-GDR is the MPI library used with MPI4Dask. Other installations of MPI libraries can result in errors. In order to avoid such situations, it is important to make sure that the correct version of the MVAPICH2-GDR library is available on the $PATH variable. This can be checked as follows:

   ```
   $ $MPILIB/bin/mpiname -a
   ```

   If the $PATH is set correctly, the output of the following command should match the previous command output.

   ```
   $ mpiname -a
   ```

3.1.5 Install the mpi4py Library

1. Download and install the mpi4py library. Details about mpi4py can be seen at https://mpi4py.readthedocs.io. Download the software as follows:

```
$ cd root-directory/mpi4dask

$ git clone https://github.com/mpi4py/mpi4py.git

$ cd mpi4py
```

2. As part of the installation, the configuration file (mpi.cfg) in the root directory of mpi4py software needs to be updated to point to the MVAPICH2 library. It is recommended to make a new section—in the mpi.cfg file—for MVAPICH2 as follows:

```
# MVAPICH2
# ------------
[MVAPICH2]
mpi_dir = /path/to/MVAPICH2-GDR/install/directory
mpicc = %(mpi_dir)s/bin/mpicc
mpicxx = %(mpi_dir)s/bin/mpicxx
include_dirs = %(mpi_dir)s/include
library_dirs = %(mpi_dir)s/lib64
runtime_library_dirs = %(library_dirs)s
```

3. After updating the mpi.cfg configuration file, install mpi4py with the following commands:

```
$ python setup.py build --mpi=MVAPICH2

$ pip install .
```

3.1.6 Install the dask-mpi library

Download and Install dask-mpi package from the github repository. MPI4Dask relies on Dask-MPI to start execution of Dask scheduler, client, and workers. Details about Dask-MPI package can be seen at http://mpi.dask.org. It is recommended not to install Dask-MPI through the conda package manager as it installs some conflicting packages as dependencies.

```
$ cd root-directory/mpi4dask

$ git clone https://github.com/dask/dask-mpi.git

$ cd dask-mpi

$ python setup.py build

$ pip install .
```

3.2 Install MPI4Dask distributed

As part of this step, an enhanced version of the Dask Distributed library with support for MPI communication-backend is installed using the following commands:

1. Download the integrated MPI4Dask distributed:

```
$ cd root-directory/mpi4dask
$ git clone https://github.com/OSU-Nowlab/distributed.git
```

2. Change directory to the custom Dask Distributed library:

```
$ cd distributed
```

3. Install the custom Dask Distributed package with the following commands:

```
$ python setup.py build

$ pip install .
```

4. At this time, MPI4Dask with all its dependencies should be installed. Verify this by running the following commands:

```
$ conda list

$ conda list | grep distributed

$ conda list | grep dask
```

4 Basic Usage Instructions

The MPI4Dask package contains two sample applications that can be used to test and benchmark performance of MPI4Dask. These applications include: 1) cuPy Array Sum and Transpose, 2) cuPy Array Dot production, 3) cuPy Array Slicing, and 4) cuDF Merge.

Notice that we highly recommend users always do `import mpi4py` before `import dask` or `import distributed`. This is due to the `Dask-distributed` will interfere with the initialization of MPI communication after a certain scale. An example is:

```
import mpi4py
from mpi4py import MPI
import mpi4py.rc
mpi4py.rc.threads = True
from dask_mpi import initialize
import dask
from dask.distributed import *
```

And specifically, when users call `initialize()` function from `dask-mpi`, please do the following to make sure that MPI Comm world is created before initializing dask scheduler/workers.

```
initialize(
      interface='ib0',
      protocol='mpi,
      nthreads=8, # user can specify this accordingly
      comm=MPI.COMM_WORLD, # MPI comm has to be created first
      local_directory=os.environ.get("TMPDIR", None)
)
```

4.1 cuPy Array Sum and Tranpose

This benchmark creates a cuPy array and distributes its chunks across Dask workers. The benchmark adds these distributed chunks to their transpose, forcing the GPU data to move around over the network. This benchmark is originally taken from the dask-cuda package repository https://github.com/rapidsai/dask-cuda/tree/branch-0.18/dask_cuda/benchmarks.

Write the hosts files. This file contains names of the compute nodes, with GPUs, that will execute the parallel Dask program.

```
$ git clone https://github.com/OSU-Nowlab/MPI4Dask-applications.git
$ cd MPI4Dask-applications/dask_apps

$ vim hosts
.. write name of the compute nodes ..

$ cat hosts
machine1
machine2
machine3
machine4
```

Note that users need to customize the hosts file as per their environment. This benchmark can be executed as follows:

```
$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1
    MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python cupy_sum_mpi.py
```

This command is starting the execution of the Dask application using MPI—this is enabled by the Dask-MPI package. A total of 4 MPI processes are started with process 0 and 1 assuming the role of Dask scheduler and client respectively. All other processes with ranks greater than 1 becoming Dask workers. Here processes 2 and 3 are Dask workers. The `LD_PRELOAD` environment variable is required for Python applications to run correctly with the MVAPICH2-GDR library.

Details on various options, passed to the MVAPICH2-GDR library, are as follows:

- `-np 4` — specifies the total number of parallel processes started by the MPI library
- `-hostfile hosts` — specifies the names of machines where to start parallel processes in the "hosts" file
- `MV2_USE_CUDA=1` — ensures that MVAPICH2-GDR library supports GPU-to-GPU communication
- `MV2_USE_GDRCOPY=1` — turns on the GDRCopy protocol
- `MV2_GPUDIRECT_GDRCOPY_LIB` — specifies path to the GDRCopy dynamic library
- `MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER` — selects a socket-level scatter process binding policy

If users are deploying MPI4Dask on nodes that each have multiple GPUs, please specify the CPU mapping explicitly to improve the performance and avoid potential issues. The standard procedure is as follows:

1. Check the Nvidia device topology.

```
$ nvidia-smi topo -m
$
    GPU0 GPU1 mlx5_0 mlx5_1 CPU Affinity NUMA Affinity
GPU0 X SYS NODE SYS 0-63 0
GPU1 SYS X SYS NODE 64-127 1
mlx5_0 NODE SYS X SYS
mlx5_1 SYS NODE SYS X

Legend:

 X = Self
 SYS = Connection traversing PCIe as well as the SMP interconnect between NUMA
     nodes (e.g., QPI/UPI)
 NODE = Connection traversing PCIe as well as the interconnect between PCIe Host
     Bridges within a NUMA node
 PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
     CPU)
 PXB = Connection traversing multiple PCIe bridges (without traversing the PCIe
     Host Bridge)
 PIX = Connection traversing at most a single PCIe bridge
 NV# = Connection traversing a bonded set of # NVLinks
```

2. For example, the above info is about a node with 2 Nvidia A100 GPUs, with 2 AMD EPYC 7713 64-Core Processers (w. 128 HyperThreading). Check the 'CPU Affinity' column, 'GPU0' is mapped to Core 0-63, 'GPU1' is mapped to Core 64-127.

3. If users launch two dask-workers (two MPI processes) on this node, first, remove two parameters from the above command

```
   MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER
```

4. And specify three MPI runtime parameters. Here is an example where each dask-worker uses 8 threads.

```
$ OMP_NUM_THREADS=8 MV2_THREADS_PER_PROCESS=8
    MV2_CPU_MAPPING=0,1,2,3,4,5,6,7:64,65,66,67,68,69,70,71
```

5. Therefore, the overall command to run will be:

```
$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4
    -hostfile hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1
    MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    OMP_NUM_THREADS=8 MV2_THREADS_PER_PROCESS=8
    MV2_CPU_MAPPING=0,1,2,3,4,5,6,7:64,65,66,67,68,69,70,71 python cupy_sum_mpi.py
```

This is not an exhaustive list of options that can be passed to the MVAPICH2-GDR library. Details on these—and other flags for the MVAPICH2-GDR library—can be seen at: http://mvapich.

cse.ohio-state.edu/userguide/gdr/. As mentioned earlier, it is important to configure the MVAPICH2-GDR installation correctly to ensure optimal performance.

The `cupy_sum_mpi.py` script has some configuration parameters that can be modified, in the source-code, by the users. These include:

- `DASK_PROTOCOL = 'mpi'` — specifies the protocol used by the Dask Distributed library. Options include 'mpi', 'tcp', 'ucx'
- `DASK_INTERFACE = 'ib0'` — specifies the interface used by the Dask Distributed library
- `GPUS_PER_NODE = 1` — specifies the number of GPUs in the system
- `THREADS_PER_NODE = 8` — specifies the number of threads (cores) in the system

4.2 cuPy Array Dot Product

This benchmark creates a cuPy array and distributes its chunks across Dask workers. The benchmark performs multiple sub-matrix dot production on these distributed chunks, and combines them together, forcing the GPU data to move around over the network. This benchmark is originally taken from the dask-cuda package repository https://github.com/rapidsai/dask-cuda/tree/branch-0.18/dask_cuda/benchmarks. Please refer to the previous section for a detailed configuration. The benchmark can be executed as follows:

```
$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1
    MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python cupy_dot_mpi.py
```

4.3 cuPy Array Slicing

This benchmark creates a cuPy array and distributes its chunks across Dask workers. The benchmark performs a slicing operation with an interval of 3, forcing the GPU data to move around over the network. This benchmark is originally taken from the dask-cuda package repository https://github.com/rapidsai/dask-cuda/tree/branch-0.18/dask_cuda/benchmarks. Please refer to the previous section for a detailed configuration. The benchmark can be executed as follows:

```
$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1
    MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python
    cupy_slice_mpi.py
```

4.4 cuDF Merge

cuDF `DataFrames` are table-like data-structure that are stored in the GPU memory. As part of this application, the merge operation is carried out for multiple cuDF data frames. This benchmark is origi-

nally taken from the dask-cuda package repository https://github.com/rapidsai/dask-cuda/tree/branch-0.18/dask_cuda/benchmarks. This application can be executed as follows:

```
$ git clone https://github.com/OSU-Nowlab/MPI4Dask-applications.git
$ cd MPI4Dask-applications/dask_apps

$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1
    MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python
    cudf_merge_mpi.py
```

The `cudf_merge_mpi.py` script accepts some runtime arguments. Some of the important ones include:

- `--type gpu` — specifies that the data frame is GPU-based. Currently only GPU has been tested
- `--protocol mpi` — specifies the communication backend used by Dask Distributed. Options include 'mpi', 'tcp', 'ucx'
- `--runs 20` — specifies the number of repetitions for the experiment
- `--chunk-size 1_000_000_00` — specifies the chunk size for the data frame

In addition, the `cudf_merge_mpi.py` script has some configuration parameters that can be modified by the users. These include:

- `DASK_INTERFACE = 'ib0'` — specifies the interface used by the Dask Distributed library
- `GPUS_PER_NODE = 1` — specifies the number of GPUs in the system
- `THREADS_PER_NODE = 8` — specifies the number of threads (cores) in the system

If users are deploying MPI4Dask on nodes that each have multiple GPUs, please specify the CPU mapping explicitly to improve the performance and avoid potential issues. Please follow the same standard procedure as in $4.1.

5 Troubleshooting with MPI4Dask

If you are experiencing any problems with MPI4Dask, please feel free to contact us by sending an email to (mvapich-discuss@lists.osu.edu).