MPI4Dask 0.2 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: June 29, 2021

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).

- Based on Dask Distributed 2021.01.0
- 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
- (NEW) Support for MPI-based communication for CPU-based Dask applications
- (NEW) 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) CPU-based Dask applications using numPy and Pandas data frames
  - (NEW) GPU-based Dask applications using cuPy and cuDF
  - Mellanox InfiniBand adapters (FDR and EDR)
  - Various multi-core platforms
  - NVIDIA V100 and Quadro RTX 5000 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
```

2. After installing the conda package manager, create a new conda environment named mpi4dask (suggested name). A tutorial on managing conda environments can be seen here.

```
$ conda create --name mpi4dask

$ conda activate 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 (10.2 in this case). A list of all installed modules on the system can be seen as follows:

```
$ module list
Currently Loaded Modulefiles:
 1) cuda/10.2
```

2. Install pre-requisite libraries including build tools, Python (version 3.8 here), Cython, CUDA tookit (version 10.2 here), Dask packages, RAPIDS libraries, 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:

```
$ conda install -c conda-forge -c rapidsai -c nvidia automake make libtool
    pkg-config libhwloc psutil python=3.8 setuptools cython cudatoolkit=10.2 cupy
    dask-cudf dask==2021.1.1 distributed numpy rmm
```

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:

```
$ conda install -c conda-forge -c rapidsai -c nvidia automake make libtool
    pkg-config libhwloc psutil python=3.8 setuptools cython dask==2021.1.1
    distributed=2021.1.1 numpy
```

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: http://mvapich.cse.ohio-state.edu/userguide/gdr/
   - MVAPICH2-GDR: http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3.5-userguide.html
   - MVAPICH2-X: http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-x-2.3-userguide.html

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:

```
$ 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. Note that depending on the installation of the MVAPICH2 library, the `library_dirs` can either point to `%(mpi_dir)s/lib64` or `%(mpi_dir)s/lib` folder. Users may check if their installation has `%(mpi_dir)s/lib64` or `%(mpi_dir)s/lib` folder and adjust `library_dirs` accordingly in the configuration file (mpi.cfg).

```
# 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 following commands:

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

$ pip install .
```

3.1.6 Dask-MPI

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.

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

$ cd dask-mpi

$ python setup.py build

$ pip install .
```

3.2 Download MPI4Dask

The latest version of MPI4Dask package can be downloaded from http://hibd.cse.ohio-state.edu/download/hibd/dask/mpi4dask-0.2.tar.gz. This contains two items:

1. the distributed folder that is an enhanced version of the the Dask Distributed library with MPI communication backend—this is what we refer to as MPI4Dask
2. the dask-apps folder that contains representative CPU and GPU Dask test/benchmarking applications

3.3 Install MPI4Dask

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. Unzip the integrated MPI4Dask distribution tarball:

```
$ wget http://hibd.cse.ohio-state.edu/download/hibd/dask/mpi4dask-0.2.tar.gz

$ tar -xzvf mpi4dask-0.2.tar.gz
```

2. Change directory to the custom Dask Distributed library inside MPI4Dask:

```
$ cd mpi4dask-0.2/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 Running GPU-based Dask Applications

The MPI4Dask package contains two sample GPU-based Dask applications that can be used to test and benchmark performance of MPI4Dask. These applications include: 1) Sum of cuPy Array and its Transpose, and 2) cuDF Merge.

For performance reasons, we recommend running GPU-based Dask applications using the MVAPICH2-GDR library.

4.1 Writing the host file

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

```
$ cd mpi4dask-0.2/dask-apps/gpu

$ 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.

4.2 Sum of cuPy Array and its Transpose

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.

This benchmark can be executed as follows:

```
$ cd mpi4dask-0.2/dask-apps/gpu

$ LD_PRELOAD=$MPILIB/lib64/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. Depending on the installation of the MVAPICH2 library, the `LD_PRELOAD` can either point to `$MPILIB/lib64/libmpi.so` or `$MPILIB/lib/libmpi.so`. Users are requested to check if their installation has `$MPILIB/lib64` or `$MPILIB/lib` folder and adjust `LD_PRELOAD` accordingly.

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

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 = 28` — specifies the number of threads (cores) in the system

4.3 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 originally taken from the dask-cuda package repository. This application can be executed as follows:

```
$ cd mpi4dask-0.2/dask-apps/gpu

$ LD_PRELOAD=$MPILIB/lib64/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 --type gpu --protocol mpi --runs 20 --chunk-size 1_000_000_00
```

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 = 28` — specifies the number of threads (cores) in the system

5 Running CPU-based Dask Applications

The MPI4Dask package contains three sample CPU-based Dask applications that can be used to test and benchmark performance of MPI4Dask. These applications include: 1) Sum of numPy Array and its Transpose, 2) Sum of Pandas data frame, and 3) Singular Value Decomposition (SVD).

For performance reasons, we recommend running CPU-based Dask applications using the MVAPICH2-X library. The MVAPICH2 library can also be used for executing CPU-based Dask applications.

5.1 Writing the host file

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

```
$ cd mpi4dask-0.2/dask-apps/cpu

$ 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.

5.2 Sum of numPy Array and its Transpose

This benchmark creates a NumPy array and distributes its chunks across Dask workers. The benchmark adds these distributed chunks to their transpose.

This benchmark can be executed as follows:

```
$ cd mpi4dask-0.2/dask-apps/cpu

$ LD_PRELOAD=$MPILIB/lib64/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python
   numpy_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 library. Depending on the installation of the MVAPICH2 library, the `LD_PRELOAD` can either point to `$MPILIB/lib64/libmpi.so` or `$MPILIB/lib/libmpi.so`. Users are requested to check if their installation has `$MPILIB/lib64` or `$MPILIB/lib` folder and adjust `LD_PRELOAD` accordingly.

Details on various options, passed to the MVAPICH2 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_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER` — selects a socket-level scatter process binding policy

This is not an exhaustive list of options that can be passed to the MVAPICH2 library. Details on these and other flags, for the MVAPICH2 and MVAPICH2-X libraries, can be seen at MVAPICH2-X and MVAPICH2 userguides, respectively.

The `numpy_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
- `THREADS_PER_NODE = 28` — specifies the number of threads (cores) in the system

5.3 Sum of Pandas Data Frame

This benchmark creates a Pandas data frame, partitions it and stores chunks of it on parallel workers. Later a sum operation is executed on the distributed data frame.

This benchmark can be executed as follows:

```
$ cd mpi4dask-0.2/dask-apps/cpu

$ LD_PRELOAD=$MPILIB/lib64/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python
   df_sum_mpi.py
```

Details on various options, passed to the MVAPICH2 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_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER` — selects a socket-level scatter process binding policy

In addition, the `df_sum_mpi.py` script has some configuration parameters that can be modified 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
- `THREADS_PER_NODE = 28` — specifies the number of threads (cores) in the system

5.4 SVD

This benchmark computes SVD for a tall-and-skinny matrix, which has has many more rows than columns. The benchmark stores this matrix as a `dask.array`, which is split across all workers. Later, the SVD operation is invoked by the client using the `da.linalg.svd(X)` function.

```
$ cd mpi4dask-0.2/dask-apps/cpu

$ LD_PRELOAD=$MPILIB/lib64/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile
    hosts MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python
   svd_mpi.py
```

Details on various options, passed to the MVAPICH2 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_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER` — selects a socket-level scatter process binding policy

In addition, the `svd_mpi.py` script has some configuration parameters that can be modified 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
- `THREADS_PER_NODE = 28` — specifies the number of threads (cores) in the system

6 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).