Dask-mpi Documentation

Release 0+untagged.121.g6a0493b

['Dask-MPI Development Team']

Dec 27, 2018

Getting Started

1 Example: 3

2 Example: 5

Easily deploy Dask using MPI

The Dask-MPI project makes it easy to deploy Dask from within an existing MPI environment, such as one created with the common MPI command-line launchers `mpirun` or `mpiexec`. Such environments are commonly found in high performance supercomputers, academic research institutions, and other clusters where MPI has already been installed. Dask-MPI provides a convenient interface for launching your cluster either from within a batch script or directly from the command-line.

CHAPTER 1

Example:

You can launch a Dask cluster directly from the command-line using the `dask-mpi` command and specifying a scheduler JSON file.

```
mpirun -np 4 dask-mpi --scheduler-file /path/to/scheduler.json
```

You can then access this cluster from a batch script or an interactive session (such as a Jupyter Notebook) by referencing the scheduler file.

```
from dask.distributed import Client
client = Client(scheduler_file='/path/to/scheduler.json')
```

CHAPTER 2

Example:

Alternatively, you can turn your batch Python script into an MPI executable simply by using the `initialize` function.

```
from dask_mpi import initialize
initialize()

from dask.distributed import Client
client = Client()  # Connect this local process to remote workers
```

which makes your Python script launchable directly with `mpirun` or `mpiexec`.

```
mpirun -np 4 python my_client_script.py
```

2.1 Installing

You can install Dask-MPI with `pip`, `conda`, or by installing from source.

2.1.1 Pip

Pip can be used to install both Dask-MPI and its dependencies (e.g. dask, distributed, NumPy, Pandas, etc.) that are necessary for different workloads.:

```
pip install dask_mpi --upgrade   # Install everything from last released version
```

2.1.2 Conda

To install the latest version of Dask-MPI from the conda-forge repository using conda:

```
conda install dask-mpi -c conda-forge
```

2.1.3 Install from Source

To install Dask-MPI from source, clone the repository from github:

```
git clone https://github.com/dask/dask-mpi.git
cd dask-mpi
python setup.py install
```

or use `pip` locally if you want to install all dependencies as well:

```
pip install -e .
```

You can also install directly from git master branch:

```
pip install git+https://github.com/dask/dask-mpi
```

2.1.4 Test

Test Dask-MPI with `pytest`:

```
git clone https://github.com/dask/dask-mpi.git
cd dask-mpi
pytest dask_mpi
```

2.2 Dask-MPI with Interactive Jobs

Dask-MPI can be used to easily launch an entire Dask cluster in an existing MPI environment, and attach a client to that cluster in an interactive session.

In this scenario, you would launch the Dask cluster using the Dask-MPI command-line interface (CLI) `dask-mpi`.

```
mpirun -np 4 dask-mpi --scheduler-file scheduler.json
```

In this example, the above code will use MPI to launch the Dask Scheduler on MPI rank 0 and Dask Workers (or Nannies) on all remaining MPI ranks.

It is advisable, as shown in the previous example, to use the `--scheduler-file` option when using the `dask-mpi` CLI. The `--scheduler-file` option saves the location of the Dask Scheduler to a file that can be referenced later in your interactive session. For example, the following code would create a Dask Client and connect it to the Scheduler using the scheduler JSON file.

```
from distributed import Client
client = Client(scheduler_file='/path/to/scheduler.json')
```

As long as your interactive session has access to the same filesystem where the scheduler JSON file is saved, this procedure will let you run your interactive session easily attach to your separate `dask-mpi` job.

After a Dask cluster has been created, the `dask-mpi` CLI can be used to add more workers to the cluster by using the `--no-scheduler` option.

```
mpirun -n 5 dask-mpi --scheduler-file scheduler.json --no-scheduler
```

In this example (above), 5 more workers will be created and they will be registered with the Scheduler (whose address is in the scheduler JSON file).

Tip: Running with a Job Scheduler

In High-Performance Computing environments, job schedulers, such as LSF, PBS, or SLURM, are commonly used to request the necessary resources needed for an MPI job, such as the number of CPU cores, the total memory needed, and/or the number of nodes over which to spread out the MPI job. In such a case, it is advisable that the user place the `mpirun ... dask-mpi ...` command in a job submission script, with the number of MPI ranks (e.g., `-np 4`) matches the number of cores requested from the job scheduler.

Warning: MPI Jobs and Dask Nannies

It is many times useful to launch your Dask-MPI cluster (using `dask-mpi`) with Dask Nannies (i.e., with the `--nanny` option), rather than strictly with Dask Workers. This is because the Dask Nannies can relaunch a worker when a failure occurs. However, in some MPI environments, Dask Nannies will not be able to work as expected. This is because some installations of MPI may restrict the number of actual running processes from exceeding the number of MPI ranks requested. When using Dask Nannies, the Nanny process is executed and runs in the background after forking a Worker process. Hence, one Worker process will exist for each Nanny process. Some MPI installations will kill any forked process, and you will see many errors arising from the Worker processes being killed. If this happens, disable the use of Nannies with the `--no-nanny` option to `dask-mpi`.

For more details on how to use the `dask-mpi` command, see the *Command-Line Interface (CLI)*.

2.3 Dask-MPI with Batch Jobs

Dask, with Dask Distributed, is an incredibly powerful engine behind interactive sessions (see *Dask-MPI with Interactive Jobs*). However, there are many scenarios where your work is pre-defined and you do not need an interactive session to execute your tasks. In these cases, running in *batch-mode* is best.

Dask-MPI makes running in batch-mode in an MPI environment easy by providing an API to the same functionality created for the `dask-mpi` *Command-Line Interface (CLI)*. However, in batch mode, you need the script running your Dask Client to run in the same environment in which your Dask cluster is constructed, and you want your Dask cluster to shut down after your Client script has executed.

To make this functionality possible, Dask-MPI provides the `initialize()` method as part of its *Application Program Interface (API)*. The `initialize()` function, when run from within an MPI environment (i.e., created by the use of `mpirun` or `mpiexec`), launches the Dask Scheduler on MPI rank 0 and the Dask Workers on MPI ranks 2 and above. On MPI rank 1, the `initialize()` function "passes through" to the Client script, running the Dask-based Client code the user wishes to execute.

For example, if you have a Dask-based script named `myscript.py`, you would be able to run this script in parallel, using Dask, with the following command.

```
mpirun -np 4 python myscript.py
```

This will run the Dask Scheduler on MPI rank 0, the user's Client code on MPI rank 1, and 2 workers on MPI rank 2 and MPI rank 3. To make this work, the `myscript.py` script must have (presumably near the top of the script) the following code in it.

```
from dask_mpi import initialize
initialize()

from distributed import Client
client = Client()
```

The Dask Client will automatically detect the location of the Dask Scheduler running on MPI rank 0 and connect to it.

When the Client code is finished executing, the Dask Scheduler and Workers (and, possibly, Nannies) will be terminated.

Tip: Running Batch Jobs with Job Schedulers

It is common in High-Performance Computing (HPC) environments to request the necessary computing resources with a job scheduler, such LSF, PBS, or SLURM. In such environments, is is advised that the `mpirun ... python myscript.py` command be placed in a job submission script such that the resources requested from the job scheduler match the resources used by the `mpirun` command.

For more details on the `initialize()` method, see the *Application Program Interface (API)*.

2.4 Command-Line Interface (CLI)

2.4.1 dask-mpi

```
dask-mpi [OPTIONS]
```

Options

--scheduler-file <scheduler_file>
: Filename to JSON encoded scheduler information.

--interface <interface>
: Network interface like 'eth0' or 'ib0'

--nthreads <nthreads>
: Number of threads per worker.

--memory-limit <memory_limit>
: Number of bytes before spilling data to disk. This can be an integer (nbytes) float (fraction of total memory) or 'auto'

--local-directory <local_directory>
: Directory to place worker files

--scheduler, --no-scheduler
: Whether or not to include a scheduler. Use –no-scheduler to increase an existing dask cluster

--nanny, --no-nanny
: Start workers in nanny process for management

--bokeh, --no-bokeh
: Enable Bokeh visual diagnostics

--bokeh-port <bokeh_port>
Bokeh port for visual diagnostics

--bokeh-worker-port <bokeh_worker_port>
Worker's Bokeh port for visual diagnostics

--bokeh-prefix <bokeh_prefix>
Prefix for the bokeh app

2.5 Application Program Interface (API)

initialize([interface, nthreads, ...])	Initialize a Dask cluster using mpi4py

2.5.1 dask_mpi.core.initialize

dask_mpi.core.**initialize**(*interface=None*, *nthreads=1*, *local_directory=''*, *memory_limit='auto'*, *nanny=False*, *bokeh=True*, *bokeh_port=8787*, *bokeh_prefix=None*, *bokeh_worker_port=8789*)

Initialize a Dask cluster using mpi4py

Using mpi4py, MPI rank 0 launches the Scheduler, MPI rank 1 passes through to the client script, and all other MPI ranks launch workers. All MPI ranks other than MPI rank 1 block while their event loops run and exit once shut down.

Parameters

interface [str] Network interface like 'eth0' or 'ib0'

nthreads [int] Number of threads per worker

local_directory [str] Directory to place worker files

memory_limit [int, float, or 'auto'] Number of bytes before spilling data to disk. This can be an integer (nbytes), float (fraction of total memory), or 'auto'.

nanny [bool] Start workers in nanny process for management

bokeh [bool] Enable Bokeh visual diagnostics

bokeh_port [int] Bokeh port for visual diagnostics

bokeh_prefix [str] Prefix for the bokeh app

bokeh_worker_port [int] Worker's Bokeh port for visual diagnostics

2.6 How Dask-MPI Works

Dask-MPI works by using the `mpi4py` package and using MPI to selectively run different code on different MPI ranks. Hence, like any other application of the `mpi4py` package, it requires creating the appropriate MPI environment through the running of the `mpirun` or `mpiexec` commands.

```
mpirun -np 8 dask-mpi --no-nannies --scheduler-file ~/scheduler.json
```

or

```
mpirun -np 8 python my_dask_script.py
```

2.6.1 Using the Dask-MPI CLI

By convention, Dask-MPI always launches the Scheduler on MPI rank 0. When using the CLI (`dask-mpi`), Dask-MPI launches the Workers (or Nannies and Workers) on the remaining MPI ranks (MPI ranks 1 and above). On each MPI rank, a `tornado` event loop is started after the Scheduler and Workers are created. These event loops continue until a kill signal is sent to one of the MPI processes, and then the entire Dask cluster (all MPI ranks) is shut down.

When using the `--no-scheduler` option of the Dask-MPI CLI, more workers can be added to an existing Dask cluster. Since these two runs will be in separate `mpirun` or `mpiexec` executions, they will only be tied to each other through the scheduler. If a worker in the new cluster crashes and takes down the entire MPI environment, it will not have anything to do with the first (original) Dask cluster. Similarly, if the first cluster is taken down, the new workers will wait for the Scheduler to reactivate so they can re-connect.

2.6.2 Using the Dask-MPI API

Again, Dask-MPI always launches the Scheduler on MPI rank 0. When using the `initialize()` method, Dask-MPI runs the Client script on MPI rank 1 and launches the Workers on the remaining MPI ranks (MPI ranks 2 and above). The Dask Scheduler and Workers start their `tornado` event loops once they are created on their given MPI ranks, and these event loops run until the Client process (MPI rank 1) sends the termination signal to the Scheduler. Once the Scheduler receives the termination signal, it will shut down the Workers, too.

2.7 Development Guidelines

This repository is part of the Dask projects. General development guidelines including where to ask for help, a layout of repositories, testing practices, and documentation and style standards are available at the Dask developer guidelines in the main documentation.

2.7.1 Install

After setting up an environment as described in the Dask developer guidelines you can clone this repository with git:

```
git clone git@github.com:dask/dask-mpi.git
```

and install it from source:

```
cd dask-mpi
python setup.py install
```

2.7.2 Test

Test using `pytest`:

```
py.test dask_mpi --verbose
```

2.8 History

This package came out of the Dask_Distributed project with help from the Pangeo collaboration. The original code was contained in the `distributed.cli.dask_mpi` module and the original tests were contained in

the `distributed.cli.tests.test_dask_mpi` module. The impetus for pulling Dask-MPI out of Dask-Distributed was provided by feedback on the Dask Distributted Issue 2402.

Development history for these original files was preserved.

Index

Symbols

-bokeh, -no-bokeh
    dask-mpi command line option, 8
-bokeh-port <bokeh_port>
    dask-mpi command line option, 8
-bokeh-prefix <bokeh_prefix>
    dask-mpi command line option, 9
-bokeh-worker-port <bokeh_worker_port>
    dask-mpi command line option, 9
-interface <interface>
    dask-mpi command line option, 8
-local-directory <local_directory>
    dask-mpi command line option, 8
-memory-limit <memory_limit>
    dask-mpi command line option, 8
-nanny, -no-nanny
    dask-mpi command line option, 8
-nthreads <nthreads>
    dask-mpi command line option, 8
-scheduler, -no-scheduler
    dask-mpi command line option, 8
-scheduler-file <scheduler_file>
    dask-mpi command line option, 8

D

dask-mpi command line option
    -bokeh, -no-bokeh, 8
    -bokeh-port <bokeh_port>, 8
    -bokeh-prefix <bokeh_prefix>, 9
    -bokeh-worker-port
        <bokeh_worker_port>, 9
    -interface <interface>, 8
    -local-directory <local_directory>,
        8
    -memory-limit <memory_limit>, 8
    -nanny, -no-nanny, 8
    -nthreads <nthreads>, 8
    -scheduler, -no-scheduler, 8
    -scheduler-file <scheduler_file>, 8

I

initialize() (*in module dask_mpi.core*), 9