Distributed communication package - torch.distributed

Note

Please refer to PyTorch Distributed Overview for a brief introduction to all features related to distributed training.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, **line 11**)

Unknown directive type "automodule".

```
.. automodule:: torch.distributed
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, **line 12**)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.distributed
```

Backends

`torch.distributed` supports three built-in backends, each with different capabilities. The table below shows which functions are available for use with CPU / CUDA tensors. MPI supports CUDA only if the implementation used to build PyTorch supports it.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, **line 23**)

Malformed table.

```
+----------------+-----------+-----------+-----------+
| Backend        | ``gloo``  | ``mpi``   | ``nccl``  |
+----------------+-----+-----+-----+-----+-----+-----+
| Device         | CPU | GPU | CPU | GPU | CPU | GPU |
+================+=====+=====+=====+=====+=====+=====+
| send           | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| recv           | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| broadcast      | âœ“   | âœ“   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| all_reduce     | âœ“   | âœ“   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| reduce         | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| all_gather     | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| gather         | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| scatter        | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ˜   |
+----------------+-----+-----+-----+-----+-----+-----+
| reduce_scatter | âœ˜   | âœ˜   | âœ˜   | âœ˜   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| all_to_all     | âœ˜   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
| barrier        | âœ“   | âœ˜   | âœ“   | ?   | âœ˜   | âœ“   |
+----------------+-----+-----+-----+-----+-----+-----+
```

Backends that come with PyTorch

PyTorch distributed package supports Linux (stable), MacOS (stable), and Windows (prototype). By default for Linux, the Gloo and NCCL backends are built and included in PyTorch distributed (NCCL only when building with CUDA). MPI is an optional backend that can only be included if you build PyTorch from source. (e.g. building PyTorch on a host that has MPI installed.)

Note

As of PyTorch v1.8, Windows supports all collective communications backend but NCCL, If the *init_method* argument of :func:`init_process_group` points to a file it must adhere to the following schema:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, **line 62**); ***backlink***

Unknown interpreted text role "func".

- Local file system, `init_method="file:///d:/tmp/some_file"`
- Shared file system, `init_method="file://////{machine_name}/{share_folder_name}/some_file"`

Same as on Linux platform, you can enable TcpStore by setting environment variables, MASTER_ADDR and MASTER_PORT.

Which backend to use?

In the past, we were often asked: "which backend should I use?".

- Rule of thumb
  - Use the NCCL backend for distributed **GPU** training
  - Use the Gloo backend for distributed **CPU** training.
- GPU hosts with InfiniBand interconnect
  - Use NCCL, since it's the only backend that currently supports InfiniBand and GPUDirect.
- GPU hosts with Ethernet interconnect
  - Use NCCL, since it currently provides the best distributed GPU training performance, especially for multiprocess single-node or multi-node distributed training. If you encounter any problem with NCCL, use Gloo as the fallback option. (Note that Gloo currently runs slower than NCCL for GPUs.)
- CPU hosts with InfiniBand interconnect
  - If your InfiniBand has enabled IP over IB, use Gloo, otherwise, use MPI instead. We are planning on adding InfiniBand support for Gloo in the upcoming releases.
- CPU hosts with Ethernet interconnect
  - Use Gloo, unless you have specific reasons to use MPI.

Common environment variables

Choosing the network interface to use

By default, both the NCCL and Gloo backends will try to find the right network interface to use. If the automatically detected interface is not correct, you can override it using the following environment variables (applicable to the respective backend):

- **NCCL_SOCKET_IFNAME**, for example `export NCCL_SOCKET_IFNAME=eth0`
- **GLOO_SOCKET_IFNAME**, for example `export GLOO_SOCKET_IFNAME=eth0`

If you're using the Gloo backend, you can specify multiple interfaces by separating them by a comma, like this: `export GLOO_SOCKET_IFNAME=eth0,eth1,eth2,eth3`. The backend will dispatch operations in a round-robin fashion across these interfaces. It is imperative that all processes specify the same number of interfaces in this variable.

Other NCCL environment variables

Debugging - in case of NCCL failure, you can set `NCCL_DEBUG=INFO` to print an explicit warning message as well as basic NCCL initialization information.

You may also use `NCCL_DEBUG_SUBSYS` to get more details about a specific aspect of NCCL. For example, `NCCL_DEBUG_SUBSYS=COLL` would print logs of collective calls, which may be helpful when debugging hangs, especially those caused by collective type or message size mismatch. In case of topology detection failure, it would be helpful to set `NCCL_DEBUG_SUBSYS=GRAPH` to inspect the detailed detection result and save as reference if further help from NCCL team is needed.

Performance tuning - NCCL performs automatic tuning based on its topology detection to save users' tuning effort. On some socket-based systems, users may still try tuning `NCCL_SOCKET_NTHREADS` and `NCCL_NSOCKS_PERTHREAD` to increase socket network bandwidth. These two environment variables have been pre-tuned by NCCL for some cloud providers, such as AWS or GCP.

For a full list of NCCL environment variables, please refer to NVIDIA NCCL's official documentation

Basics

The *torch.distributed* package provides PyTorch support and communication primitives for multiprocess parallelism across several computation nodes running on one or more machines. The class :func:`torch.nn.parallel.DistributedDataParallel` builds on this functionality to provide synchronous distributed training as a wrapper around any PyTorch model. This differs from the kinds of parallelism provided by :doc:`multiprocessing` and :func:`torch.nn.DataParallel` in that it supports multiple network-connected machines and in that the user must explicitly launch a separate copy of the main training script for each process.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 152); ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 152); ***backlink***
>
> Unknown interpreted text role "doc".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 152); ***backlink***
>
> Unknown interpreted text role "func".

In the single-machine synchronous case, *torch.distributed* or the :func:`torch.nn.parallel.DistributedDataParallel` wrapper may still have advantages over other approaches to data-parallelism, including :func:`torch.nn.DataParallel`:

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 161); ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 161); ***backlink***
>
> Unknown interpreted text role "func".

- Each process maintains its own optimizer and performs a complete optimization step with each iteration. While this may appear redundant, since the gradients have already been gathered together and averaged across processes and are thus the same for every process, this means that no parameter broadcast step is needed, reducing time spent transferring tensors between nodes.
- Each process contains an independent Python interpreter, eliminating the extra interpreter overhead and "GIL-thrashing" that comes from driving several execution threads, model replicas, or GPUs from a single Python process. This is especially important for models that make heavy use of the Python runtime, including models with recurrent layers or many small components.

Initialization

The package needs to be initialized using the :func:`torch.distributed.init_process_group` function before calling any other methods. This blocks until all processes have joined.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 179); ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 183)
>
> Unknown directive type "autofunction".
>
> ```
> .. autofunction:: is_available
> ```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 185)
>
> Unknown directive type "autofunction".
>
> ```
> .. autofunction:: init_process_group
> ```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 187)
>
> Unknown directive type "autofunction".
>
> ```
> .. autofunction:: is_initialized
> ```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 189)
>
> Unknown directive type "autofunction".
>
> ```
> .. autofunction:: is_mpi_available
> ```

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-

master\docs\source\[pytorch-master][docs][source]distributed.rst, line 191)

Unknown directive type "autofunction".

```
.. autofunction:: is_nccl_available
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 193)

Unknown directive type "autofunction".

```
.. autofunction:: is_torchelastic_launched
```

Currently three initialization methods are supported:

TCP initialization

There are two ways to initialize using TCP, both requiring a network address reachable from all processes and a desired `world_size`. The first way requires specifying an address that belongs to the rank 0 process. This initialization method requires that all processes have manually specified ranks.

Note that multicast address is not supported anymore in the latest distributed package. `group_name` is deprecated as well.

```
import torch.distributed as dist

# Use address of one of the machines
dist.init_process_group(backend, init_method='tcp://10.1.1.20:23456',
                        rank=args.rank, world_size=4)
```

Shared file-system initialization

Another initialization method makes use of a file system that is shared and visible from all machines in a group, along with a desired `world_size`. The URL should start with `file://` and contain a path to a non-existent file (in an existing directory) on a shared file system. File-system initialization will automatically create that file if it doesn't exist, but will not delete the file. Therefore, it is your responsibility to make sure that the file is cleaned up before the next :func:`init_process_group` call on the same file path/name.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 221); *backlink*

Unknown interpreted text role "func".

Note that automatic rank assignment is not supported anymore in the latest distributed package and `group_name` is deprecated as well.

Warning

This method assumes that the file system supports locking using `fcntl` - most local systems and NFS support it.

Warning

This method will always create the file and try its best to clean up and remove the file at the end of the program. In other words, each initialization with the file init method will need a brand new empty file in order for the initialization to succeed. If the same file used by the previous initialization (which happens not to get cleaned up) is used again, this is unexpected behavior and can often cause deadlocks and failures. Therefore, even though this method will try its best to clean up the file, if the auto-delete happens to be unsuccessful, it is your responsibility to ensure that the file is removed at the end of the training to prevent the same file to be reused again during the next time. This is especially important if you plan to call :func:`init_process_group` multiple times on the same file name. In other words, if the file is not removed/cleaned up and you call :func:`init_process_group` again on that file, failures are expected. The rule of thumb here is that, make sure that the file is non-existent or empty every time :func:`init_process_group` is called.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 237); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 237); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 237); *backlink*

Unknown interpreted text role "func".

```
import torch.distributed as dist

# rank should always be specified
dist.init_process_group(backend, init_method='file:///mnt/nfs/sharedfile',
                        world_size=4, rank=args.rank)
```

Environment variable initialization

This method will read the configuration from environment variables, allowing one to fully customize how the information is obtained. The variables to be set are:

- `MASTER_PORT` - required; has to be a free port on machine with rank 0
- `MASTER_ADDR` - required (except for rank 0); address of rank 0 node
- `WORLD_SIZE` - required; can be set either here, or in a call to init function
- `RANK` - required; can be set either here, or in a call to init function

The machine with rank 0 will be used to set up all connections.

This is the default method, meaning that `init_method` does not have to be specified (or can be `env://`).

Post-Initialization

Once :func:`torch.distributed.init_process_group` was run, the following functions can be used. To check whether the process group has already been initialized use :func:`torch.distributed.is_initialized`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 280); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 280); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 283)

Unknown directive type "autoclass".

```
.. autoclass:: Backend
    :members:
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 286)

Unknown directive type "autofunction".

```
.. autofunction:: get_backend
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 288)

Unknown directive type "autofunction".

```
.. autofunction:: get_rank
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 290)

Unknown directive type "autofunction".

```
.. autofunction:: get_world_size
```

Distributed Key-Value Store

The distributed package comes with a distributed key-value store, which can be used to share information between processes in the group as well as to initialize the distributed package in :func:`torch.distributed.init_process_group` (by explicitly creating the store as an alternative to specifying `init_method`.) There are 3 choices for Key-Value Stores: :class:`~torch.distributed.TCPStore`, :class:`~torch.distributed.FileStore`, and :class:`~torch.distributed.HashStore`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 297); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 297); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 297); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 297); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 305)

Unknown directive type "autoclass".

```
.. autoclass:: Store
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 306)

Unknown directive type "autoclass".

```
.. autoclass:: TCPStore
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 307)

Unknown directive type "autoclass".

```
.. autoclass:: HashStore
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 308)

Unknown directive type "autoclass".

```
.. autoclass:: FileStore
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 309)

Unknown directive type "autoclass".

```
.. autoclass:: PrefixStore
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 311)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.set
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 312)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.get
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 313)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.add
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 314)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.compare_set
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 315)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.wait
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 316)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.num_keys
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 317)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.delete_key
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 318)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.Store.set_timeout
```

Groups

By default collectives operate on the default group (also called the world) and require all processes to enter the distributed function call. However, some workloads can benefit from more fine-grained communication. This is where distributed groups come into play. :func:`~torch.distributed.new_group` function can be used to create new groups, with arbitrary subsets of all processes. It returns an opaque group handle that can be given as a `group` argument to all collectives (collectives are distributed functions to exchange information in certain well-known programming patterns).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 323); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 332)

Unknown directive type "autofunction".

```
.. autofunction:: new_group
```

Point-to-point communication

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 337)

Unknown directive type "autofunction".

```
.. autofunction:: send
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 339)

Unknown directive type "autofunction".

```
.. autofunction:: recv
```

:func:`~torch.distributed.isend` and :func:`~torch.distributed.irecv` return distributed request objects when used. In general, the type of this object is unspecified as they should never be created manually, but they are guaranteed to support two methods:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 341); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 341); *backlink*

Unknown interpreted text role "func".

- `is_completed()` - returns True if the operation has finished
- `wait()` - will block the process until the operation is finished. `is_completed()` is guaranteed to return True once it returns.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 349)

Unknown directive type "autofunction".

```
.. autofunction:: isend
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 351)

Unknown directive type "autofunction".

```
.. autofunction:: irecv
```

Synchronous and asynchronous collective operations

Every collective operation function supports the following two kinds of operations, depending on the setting of the `async_op` flag passed into the collective:

Synchronous operation - the default mode, when `async_op` is set to `False`. When the function returns, it is guaranteed that the collective operation is performed. In the case of CUDA operations, it is not guaranteed that the CUDA operation is completed, since CUDA operations are asynchronous. For CPU collectives, any further function calls utilizing the output of the collective call will behave as expected. For CUDA collectives, function calls utilizing the output on the same CUDA stream will behave as expected. Users must take care of synchronization under the scenario of running under different streams. For details on CUDA semantics such as stream synchronization, see CUDA Semantics. See the below script to see examples of differences in these semantics for CPU and CUDA operations.

Asynchronous operation - when `async_op` is set to True. The collective operation function returns a distributed request object. In general, you don't need to create it manually and it is guaranteed to support two methods:

- `is_completed()` - in the case of CPU collectives, returns `True` if completed. In the case of CUDA operations, returns `True` if the operation has been successfully enqueued onto a CUDA stream and the output can be utilized on the default stream without further synchronization.
- `wait()` - in the case of CPU collectives, will block the process until the operation is completed. In the case of CUDA collectives, will block until the operation has been successfully enqueued onto a CUDA stream and the output can be utilized on the default stream without further synchronization.
- `get_future()` - returns `torch._C.Future` object. Supported for NCCL, also supported for most operations on GLOO and MPI, except for peer to peer operations. Note: as we continue adopting Futures and merging APIs, `get_future()` call might become redundant.

Example

The following code can serve as a reference regarding semantics for CUDA operations when using distributed collectives. It shows the explicit need to synchronize when using collective outputs on different CUDA streams:

```
# Code runs on each rank.
dist.init_process_group("nccl", rank=rank, world_size=2)
output = torch.tensor([rank]).cuda(rank)
s = torch.cuda.Stream()
handle = dist.all_reduce(output, async_op=True)
# Wait ensures the operation is enqueued, but not necessarily complete.
handle.wait()
# Using result on non-default stream.
with torch.cuda.stream(s):
    s.wait_stream(torch.cuda.default_stream())
    output.add_(100)
if rank == 0:
    # if the explicit call to wait_stream was omitted, the output below will be
    # non-deterministically 1 or 101, depending on whether the allreduce overwrote
    # the value after the add completed.
    print(output)
```

Collective functions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 410)

Unknown directive type "autofunction".

```
.. autofunction:: broadcast
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 412)

Unknown directive type "autofunction".

```
.. autofunction:: broadcast_object_list
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 414)

Unknown directive type "autofunction".

```
.. autofunction:: all_reduce
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 416)

Unknown directive type "autofunction".

```
.. autofunction:: reduce
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 418)

Unknown directive type "autofunction".

```
.. autofunction:: all_gather
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 420)

Unknown directive type "autofunction".

```
.. autofunction:: all_gather_object
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst, line 422)

Unknown directive type "autofunction".

```
.. autofunction:: gather
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 424)

Unknown directive type "autofunction".

```
.. autofunction:: gather_object
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 426)

Unknown directive type "autofunction".

```
.. autofunction:: scatter
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 428)

Unknown directive type "autofunction".

```
.. autofunction:: scatter_object_list
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 430)

Unknown directive type "autofunction".

```
.. autofunction:: reduce_scatter
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 432)

Unknown directive type "autofunction".

```
.. autofunction:: all_to_all
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 434)

Unknown directive type "autofunction".

```
.. autofunction:: barrier
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 436)

Unknown directive type "autofunction".

```
.. autofunction:: monitored_barrier
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 438)

Unknown directive type "autoclass".

```
.. autoclass:: ReduceOp
```

Deprecated enum-like class for reduction operations: `SUM`, `PRODUCT`, `MIN`, and `MAX`.

:class:`~torch.distributed.ReduceOp` is recommended to use instead.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 445); ***backlink***

Unknown interpreted text role "class".

Profiling Collective Communication

Note that you can use `torch.profiler` (recommended, only available after 1.8.1) or `torch.autograd.profiler` to profile collective communication and point-to-point communication APIs mentioned here. All out-of-the-box backends (`gloo`, `nccl`, `mpi`) are supported and collective communication usage will be rendered as expected in profiling output/traces. Profiling your code is the same as any regular torch operator:

```
import torch
import torch.distributed as dist
with torch.profiler():
    tensor = torch.randn(20, 10)
    dist.all_reduce(tensor)
```

Please refer to the profiler documentation for a full overview of profiler features.

Multi-GPU collective functions

If you have more than one GPU on each node, when using the NCCL and Gloo backend, :func:`~torch.distributed.broadcast_multigpu` :func:`~torch.distributed.all_reduce_multigpu` :func:`~torch.distributed.reduce_multigpu` :func:`~torch.distributed.all_gather_multigpu` and :func:`~torch.distributed.reduce_scatter_multigpu` support distributed collective operations among multiple GPUs within each node. These functions can potentially improve the overall distributed training performance and be easily used by passing a list of tensors. Each Tensor in the passed tensor list needs to be on a separate GPU device of the host where the function is called. Note that the length of the tensor list needs to be identical among all the distributed processes. Also note that currently the multi-GPU collective functions are only supported by the NCCL backend.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 467); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 467); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 467); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 467); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 467); *backlink*

Unknown interpreted text role "func".

For example, if the system we use for distributed training has 2 nodes, each of which has 8 GPUs. On each of the 16 GPUs, there is a tensor that we would like to all-reduce. The following code can serve as a reference:

Code running on Node 0

```
import torch
import torch.distributed as dist

dist.init_process_group(backend="nccl",
                        init_method="file:///distributed_test",
                        world_size=2,
                        rank=0)
tensor_list = []
for dev_idx in range(torch.cuda.device_count()):
    tensor_list.append(torch.FloatTensor([1]).cuda(dev_idx))

dist.all_reduce_multigpu(tensor_list)
```

Code running on Node 1

```
import torch
import torch.distributed as dist

dist.init_process_group(backend="nccl",
                        init_method="file:///distributed_test",
                        world_size=2,
                        rank=1)
tensor_list = []
for dev_idx in range(torch.cuda.device_count()):
    tensor_list.append(torch.FloatTensor([1]).cuda(dev_idx))

dist.all_reduce_multigpu(tensor_list)
```

After the call, all 16 tensors on the two nodes will have the all-reduced value of 16

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 522)

Unknown directive type "autofunction".

```
.. autofunction:: broadcast_multigpu
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 524)

Unknown directive type "autofunction".

```
.. autofunction:: all_reduce_multigpu
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 526)

Unknown directive type "autofunction".

```
.. autofunction:: reduce_multigpu
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 528)

Unknown directive type "autofunction".

```
.. autofunction:: all_gather_multigpu
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 530)

Unknown directive type "autofunction".

```
.. autofunction:: reduce_scatter_multigpu
```

Third-party backends

Besides the builtin GLOO/MPI/NCCL backends, PyTorch distributed supports third-party backends through a run-time register mechanism. For references on how to develop a third-party backend through C++ Extension, please refer to Tutorials - Custom C++ and CUDA Extensions and `test/cpp_extensions/cpp_c10d_extension.cpp`. The capability of third-party backends are decided by their own implementations.

The new backend derives from `c10d::ProcessGroup` and registers the backend name and the instantiating interface through :func:`torch.distributed.Backend.register_backend` when imported.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 546); *backlink*

Unknown interpreted text role "func".

When manually importing this backend and invoking :func:`torch.distributed.init_process_group` with the corresponding backend name, the `torch.distributed` package runs on the new backend.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 550); *backlink*

Unknown interpreted text role "func".

Warning

The support of third-party backend is experimental and subject to change.

Launch utility

The *torch.distributed* package also provides a launch utility in *torch.distributed.launch*. This helper utility can be used to launch multiple processes per node for distributed training.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 565)

Unknown directive type "automodule".

```
.. automodule:: torch.distributed.launch
```

Spawn utility

The :ref:`multiprocessing-doc` package also provides a `spawn` function in :func:`torch.multiprocessing.spawn`. This helper function can be used to spawn multiple processes. It works by passing in the function that you want to run and spawns N processes to run it. This can be used for multiprocess distributed training as well.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 571); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 571); *backlink*

Unknown interpreted text role "func".

For references on how to use it, please refer to PyTorch example - ImageNet implementation

Note that this function requires Python 3.4 or higher.

Debugging `torch.distributed` applications

Debugging distributed applications can be challenging due to hard to understand hangs, crashes, or inconsistent behavior across ranks. `torch.distributed` provides a suite of tools to help debug training applications in a self-serve fashion:

Monitored Barrier

As of v1.10, :func:`torch.distributed.monitored_barrier` exists as an alternative to :func:`torch.distributed.barrier` which fails with helpful information about which rank may be faulty when crashing, i.e. not all ranks calling into :func:`torch.distributed.monitored_barrier` within the provided timeout. :func:`torch.distributed.monitored_barrier` implements a host-side barrier using `send`/`recv` communication primitives in a process similar to acknowledgements, allowing rank 0 to report which rank(s) failed to acknowledge the barrier in time. As an example, consider the following function where rank 1 fails to call into :func:`torch.distributed.monitored_barrier` (in practice this could be due to an application bug or hang in a previous collective):

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 591); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 591); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 591); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 591); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 591); *backlink*

Unknown interpreted text role "func".

```
import os
from datetime import timedelta

import torch
import torch.distributed as dist
import torch.multiprocessing as mp


def worker(rank):
    dist.init_process_group("nccl", rank=rank, world_size=2)
    # monitored barrier requires gloo process group to perform host-side sync.
    group_gloo = dist.new_group(backend="gloo")
    if rank not in [1]:
        dist.monitored_barrier(group=group_gloo, timeout=timedelta(seconds=2))


if __name__ == "__main__":
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "29501"
    mp.spawn(worker, nprocs=2, args=())
```

The following error message is produced on rank 0, allowing the user to determine which rank(s) may be faulty and investigate further:

```
RuntimeError: Rank 1 failed to pass monitoredBarrier in 2000 ms
 Original exception:
[gloo/transport/tcp/pair.cc:598] Connection closed by peer [2401:db00:eef0:1100:3560:0:1c05:25d]:8594
```

`TORCH_DISTRIBUTED_DEBUG`

Next, the environment variable `TORCH_DISTRIBUTED_DEBUG` can be used to trigger additional useful logging and collective synchronization checks to ensure all ranks are synchronized appropriately. `TORCH_DISTRIBUTED_DEBUG` can be set to either `OFF` (default), `INFO`, or `DETAIL` depending on the debugging level required. Please note that the most verbose option, `DETAIL` may impact the application performance and thus should only be used when debugging issues.

Setting `TORCH_DISTRIBUTED_DEBUG=INFO` will result in additional debug logging when models trained with :func:`torch.nn.parallel.DistributedDataParallel` are initialized, and `TORCH_DISTRIBUTED_DEBUG=DETAIL` will additionally log runtime performance statistics a select number of iterations. These runtime statistics include data such as forward time, backward time, gradient communication time, etc. As an example, given the following application:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 636); *backlink*

Unknown interpreted text role "func".

```
import os

import torch
import torch.distributed as dist
import torch.multiprocessing as mp


class TwoLinLayerNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.a = torch.nn.Linear(10, 10, bias=False)
        self.b = torch.nn.Linear(10, 1, bias=False)

    def forward(self, x):
        a = self.a(x)
        b = self.b(x)
        return (a, b)


def worker(rank):
    dist.init_process_group("nccl", rank=rank, world_size=2)
    torch.cuda.set_device(rank)
    print("init model")
    model = TwoLinLayerNet().cuda()
    print("init ddp")
    ddp_model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[rank])

    inp = torch.randn(10, 10).cuda()
    print("train")

    for _ in range(20):
        output = ddp_model(inp)
        loss = output[0] + output[1]
        loss.sum().backward()


if __name__ == "__main__":
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "29501"
    os.environ[
        "TORCH_DISTRIBUTED_DEBUG"
    ] = "DETAIL"  # set to DETAIL for runtime logging.
    mp.spawn(worker, nprocs=2, args=())
```

The following logs are rendered at initialization time:

```
I0607 16:10:35.739390 515217 logger.cpp:173] [Rank 0]: DDP Initialized with:
broadcast_buffers: 1
bucket_cap_bytes: 26214400
find_unused_parameters: 0
gradient_as_bucket_view: 0
is_multi_device_module: 0
iteration: 0
num_parameter_tensors: 2
output_device: 0
rank: 0
total_parameter_size_bytes: 440
world_size: 2
backend_name: nccl
bucket_sizes: 440
cuda_visible_devices: N/A
device_ids: 0
dtypes: float
master_addr: localhost
master_port: 29501
module_name: TwoLinLayerNet
nccl_async_error_handling: N/A
nccl_blocking_wait: N/A
nccl_debug: WARN
nccl_ib_timeout: N/A
nccl_nthreads: N/A
nccl_socket_ifname: N/A
torch_distributed_debug: INFO
```

The following logs are rendered during runtime (when `TORCH_DISTRIBUTED_DEBUG=DETAIL` is set):

```
I0607 16:18:58.085681 544067 logger.cpp:344] [Rank 1 / 2] Training TwoLinLayerNet unused_parameter_size=0
 Avg forward compute time: 40838608
 Avg backward compute time: 5983335
Avg backward comm. time: 4326421
 Avg backward comm/comp overlap time: 4207652
I0607 16:18:58.085693 544066 logger.cpp:344] [Rank 0 / 2] Training TwoLinLayerNet unused_parameter_size=0
 Avg forward compute time: 42850427
 Avg backward compute time: 3885553
Avg backward comm. time: 2357981
 Avg backward comm/comp overlap time: 2234674
```

In addition, `TORCH_DISTRIBUTED_DEBUG=INFO` enhances crash logging in :func:`torch.nn.parallel.DistributedDataParallel` due to unused parameters in the model. Currently, `find_unused_parameters=True` must be passed into :func:`torch.nn.parallel.DistributedDataParallel` initialization if there are parameters that may be unused in the forward pass, and as of v1.10, all model outputs are required to be used in loss computation as :func:`torch.nn.parallel.DistributedDataParallel` does not support unused parameters in the backwards pass. These constraints are challenging especially for larger models, thus when crashing with an error, :func:`torch.nn.parallel.DistributedDataParallel` will log the fully qualified name of all parameters that went unused. For example, in the above application, if we modify `loss` to be instead computed as `loss = output[1]`, then `TwoLinLayerNet.a` does not receive a gradient in the backwards pass, and thus results in `DDP` failing. On a crash, the user is passed information about parameters which went unused, which may be challenging to manually find for large models:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 735); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 735); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 735); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 735); *backlink*

Unknown interpreted text role "func".

```
RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one. This error indicates that your module
 the keyword argument `find_unused_parameters=True` to `torch.nn.parallel.DistributedDataParallel`, and by
making sure all `forward` function outputs participate in calculating loss.
If you already have done the above, then the distributed data parallel module wasn't able to locate the output tensors in the return valu
lue of `forward` of your module when reporting this issue (e.g. list, dict, iterable).
Parameters which did not receive grad for rank 0: a.weight
Parameter indices which did not receive grad for rank 0: 0
```

Setting `TORCH_DISTRIBUTED_DEBUG=DETAIL` will trigger additional consistency and synchronization checks on every collective call issued by the user either directly or indirectly (such as DDP `allreduce`). This is done by creating a wrapper process group that wraps all process groups returned by :func:`torch.distributed.init_process_group` and :func:`torch.distributed.new_group` APIs. As a result, these APIs will return a wrapper process group that can be used exactly like a regular process group, but performs consistency checks before dispatching the collective to an underlying process group. Currently, these checks include a :func:`torch.distributed.monitored_barrier`, which ensures all ranks complete their outstanding collective calls and reports ranks which are stuck. Next, the collective itself is checked for consistency by ensuring all collective functions match and are called with consistent tensor shapes. If this is not the case, a detailed error report is included when the application crashes, rather than a hang or uninformative error message. As an example, consider the following function which has mismatched input shapes into :func:`torch.distributed.all_reduce`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 754); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 754); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 754); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 754); *backlink*

Unknown interpreted text role "func".

```
import torch
import torch.distributed as dist
import torch.multiprocessing as mp


def worker(rank):
    dist.init_process_group("nccl", rank=rank, world_size=2)
    torch.cuda.set_device(rank)
    tensor = torch.randn(10 if rank == 0 else 20).cuda()
    dist.all_reduce(tensor)
    torch.cuda.synchronize(device=rank)


if __name__ == "__main__":
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "29501"
    os.environ["TORCH_DISTRIBUTED_DEBUG"] = "DETAIL"
    mp.spawn(worker, nprocs=2, args=())
```

With the `NCCL` backend, such an application would likely result in a hang which can be challenging to root-cause in nontrivial scenarios. If the user enables `TORCH_DISTRIBUTED_DEBUG=DETAIL` and reruns the application, the following error message reveals the root cause:

```
work = default_pg.allreduce([tensor], opts)
RuntimeError: Error when verifying shape tensors for collective ALLREDUCE on rank 0. This likely indicates that input shapes into the col
[ torch.LongTensor{1} ]
```

Note

For fine-grained control of the debug level during runtime the functions :func:`torch.distributed.set_debug_level`, :func:`torch.distributed.set_debug_level_from_env`, and :func:`torch.distributed.get_debug_level` can also be used.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 794); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 794); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 794); *backlink*

Unknown interpreted text role "func".

In addition, *TORCH_DISTRIBUTED_DEBUG=DETAIL* can be used in conjunction with *TORCH_SHOW_CPP_STACKTRACES=1* to log the entire callstack when a collective desynchronization is detected. These collective desynchronization checks will work for all applications that use `c10d` collective calls backed by process groups created with the :func:`torch.distributed.init_process_group` and :func:`torch.distributed.new_group` APIs.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 797); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 797); *backlink*

Unknown interpreted text role "func".

Logging

In addition to explicit debugging support via :func:`torch.distributed.monitored_barrier` and `TORCH_DISTRIBUTED_DEBUG`, the underlying C++ library of `torch.distributed` also outputs log messages at various levels. These messages can be helpful to understand the execution state of a distributed training job and to troubleshoot problems such as network connection failures. The following matrix shows how the log level can be adjusted via the combination of `TORCH_CPP_LOG_LEVEL` and `TORCH_DISTRIBUTED_DEBUG` environment variables.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 804); *backlink*

Unknown interpreted text role "func".

TORCH_CPP_LOG_LEVEL	TORCH_DISTRIBUTED_DEBUG	Effective Log Level
ERROR	ignored	Error
WARNING	ignored	Warning
INFO	ignored	Info
INFO	INFO	Debug
INFO	DETAIL	Trace (a.k.a. All)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 825)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.algorithms
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 826)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.algorithms.ddp_comm_hooks
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 827)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.algorithms.model_averaging
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 828)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.elastic
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 829)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.elastic.utils
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 830)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.elastic.utils.data
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 831)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.launcher
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 832)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.nn
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 833)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.nn.api
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 834)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.nn.jit
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 835)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.nn.jit.templates
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 836)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.pipeline
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 837)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.pipeline.sync
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]distributed.rst`, line 838)

Unknown directive type "py:module".

```
.. py:module:: torch.distributed.pipeline.sync.skip
```