Distributed RPC Framework

The distributed RPC framework provides mechanisms for multi-machine model training through a set of primitives to allow for remote communication, and a higher-level API to automatically differentiate models split across several machines.

> **Warning**
>
> APIs in the RPC package are stable. There are multiple ongoing work items to improve performance and error handling, which will ship in future releases.

> **Warning**
>
> CUDA support was introduced in PyTorch 1.9 and is still a **beta** feature. Not all features of the RPC package are yet compatible with CUDA support and thus their use is discouraged. These unsupported features include: RRefs, JIT compatibility, dist autograd and dist optimizier, and profiling. These shortcomings will be addressed in future releases.

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

Basics

The distributed RPC framework makes it easy to run functions remotely, supports referencing remote objects without copying the real data around, and provides autograd and optimizer APIs to transparently run backward and update parameters across RPC boundaries. These features can be categorized into four sets of APIs.

1. **Remote Procedure Call (RPC)** supports running a function on the specified destination worker with the given arguments and getting the return value back or creating a reference to the return value. There are three main RPC APIs: :meth:`~torch.distributed.rpc.rpc_sync` (synchronous), :meth:`~torch.distributed.rpc.rpc_async` (asynchronous), and :meth:`~torch.distributed.rpc.remote` (asynchronous and returns a reference to the remote return value). Use the synchronous API if the user code cannot proceed without the return value. Otherwise, use the asynchronous API to get a future, and wait on the future when the return value is needed on the caller. The :meth:`~torch.distributed.rpc.remote` API is useful when the requirement is to create something remotely but never need to fetch it to the caller. Imagine the case that a driver process is setting up a parameter server and a trainer. The driver can create an embedding table on the parameter server and then share the reference to the embedding table with the trainer, but itself will never use the embedding table locally. In this case, :meth:`~torch.distributed.rpc.rpc_sync` and :meth:`~torch.distributed.rpc.rpc_async` are no longer appropriate, as they always imply that the return value will be returned to the caller immediately or in the future.

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, **line 34**); ***backlink***
   >
   > Unknown interpreted text role "meth".

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, **line 34**); ***backlink***
   >
   > Unknown interpreted text role "meth".

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, **line 34**); ***backlink***
   >
   > Unknown interpreted text role "meth".

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, **line 34**); ***backlink***
   >
   > Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 34); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 34); *backlink*

Unknown interpreted text role "meth".

2. **Remote Reference (RRef)** serves as a distributed shared pointer to a local or remote object. It can be shared with other workers and reference counting will be handled transparently. Each RRef only has one owner and the object only lives on that owner. Non-owner workers holding RRefs can get copies of the object from the owner by explicitly requesting it. This is useful when a worker needs to access some data object, but itself is neither the creator (the caller of :meth:`~torch.distributed.rpc.remote`) or the owner of the object. The distributed optimizer, as we will discuss below, is one example of such use cases.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 53); *backlink*

Unknown interpreted text role "meth".

3. **Distributed Autograd** stitches together local autograd engines on all the workers involved in the forward pass, and automatically reach out to them during the backward pass to compute gradients. This is especially helpful if the forward pass needs to span multiple machines when conducting, e.g., distributed model parallel training, parameter-server training, etc. With this feature, user code no longer needs to worry about how to send gradients across RPC boundaries and in which order should the local autograd engines be launched, which can become quite complicated where there are nested and inter-dependent RPC calls in the forward pass.
4. **Distributed Optimizer**'s constructor takes a :meth:`~torch.optim.Optimizer` (e.g., :meth:`~torch.optim.SGD`, :meth:`~torch.optim.Adagrad`, etc.) and a list of parameter RRefs, creates an :meth:`~torch.optim.Optimizer` instance on each distinct RRef owner, and updates parameters accordingly when running `step()`. When you have distributed forward and backward passes, parameters and gradients will be scattered across multiple workers, and hence it requires an optimizer on each of the involved workers. Distributed Optimizer wraps all those local optimizers into one, and provides a concise constructor and `step()` API.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 71); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 71); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 71); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst`, line 71); *backlink*

Unknown interpreted text role "meth".

RPC

Before using RPC and distributed autograd primitives, initialization must take place. To initialize the RPC framework we need to use :meth:`~torch.distributed.rpc.init_rpc` which would initialize the RPC framework, RRef framework and distributed autograd.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 87); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 92)

Unknown directive type "automodule".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 93)

Unknown directive type "autofunction".

```
.. autofunction:: init_rpc
```

The following APIs allow users to remotely execute functions as well as create references (RRefs) to remote data objects. In these APIs, when passing a `Tensor` as an argument or a return value, the destination worker will try to create a `Tensor` with the same meta (i.e., shape, stride, etc.). We intentionally disallow transmitting CUDA tensors because it might crash if the device lists on source and destination workers do not match. In such cases, applications can always explicitly move the input tensors to CPU on the caller and move it to the desired devices on the callee if necessary.

Warning

TorchScript support in RPC is a prototype feature and subject to change. Since v1.5.0, `torch.distributed.rpc` supports calling TorchScript functions as RPC target functions, and this will help improve parallelism on the callee side as executing TorchScript functions does not require GIL.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 111)

Unknown directive type "autofunction".

```
.. autofunction:: rpc_sync
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 112)

Unknown directive type "autofunction".

```
.. autofunction:: rpc_async
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 113)

Unknown directive type "autofunction".

```
.. autofunction:: remote
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 114)

Unknown directive type "autofunction".

```
.. autofunction:: get_worker_info
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 115)

Unknown directive type "autofunction".

```
.. autofunction:: shutdown
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 116)

Unknown directive type "autoclass".

```
.. autoclass:: WorkerInfo
    :members:
```

The RPC package also provides decorators which allow applications to specify how a given function should be treated on the callee side.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 124)

Unknown directive type "autofunction".

```
.. autofunction:: torch.distributed.rpc.functions.async_execution
```

Backends

The RPC module can leverage different backends to perform the communication between the nodes. The backend to be used can be specified in the :func:`~torch.distributed.rpc.init_rpc` function, by passing a certain value of the :class:`~torch.distributed.rpc.BackendType` enum. Regardless of what backend is used, the rest of the RPC API won't change. Each backend also defines its own subclass of the :class:`~torch.distributed.rpc.RpcBackendOptions` class, an instance of which can also be passed to :func:`~torch.distributed.rpc.init_rpc` to configure the backend's behavior.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 132); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 132); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 132); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 141)

Unknown directive type "autoclass".

```
.. autoclass:: BackendType
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 143)

Unknown directive type "autoclass".

```
.. autoclass:: RpcBackendOptions
```

```
    :members:
```

TensorPipe Backend

The TensorPipe agent, which is the default, leverages the TensorPipe library, which provides a natively point-to-point communication primitive specifically suited for machine learning that fundamentally addresses some of the limitations of Gloo. Compared to Gloo, it has the advantage of being asynchronous, which allows a large number of transfers to occur simultaneously, each at their own speed, without blocking each other. It will only open pipes between pairs of nodes when needed, on demand, and when one node fails only its incident pipes will be closed, while all other ones will keep working as normal. In addition, it is able to support multiple different transports (TCP, of course, but also shared memory, NVLink, InfiniBand, ...) and can automatically detect their availability and negotiate the best transport to use for each pipe.

The TensorPipe backend has been introduced in PyTorch v1.6 and is being actively developed. At the moment, it only supports CPU tensors, with GPU support coming soon. It comes with a TCP-based transport, just like Gloo. It is also able to automatically chunk and multiplex large tensors over multiple sockets and threads in order to achieve very high bandwidths. The agent will be able to pick the best transport on its own, with no intervention required.

Example:

```
>>> import os
>>> from torch.distributed import rpc
>>> os.environ['MASTER_ADDR'] = 'localhost'
>>> os.environ['MASTER_PORT'] = '29500'
>>>
>>> rpc.init_rpc(
>>>     "worker1",
>>>     rank=0,
>>>     world_size=2,
>>>     rpc_backend_options=rpc.TensorPipeRpcBackendOptions(
>>>         num_worker_threads=8,
>>>         rpc_timeout=20 # 20 second timeout
>>>     )
>>> )
>>>
>>> # omitting init_rpc invocation on worker2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 189)

Unknown directive type "autoclass".

```
.. autoclass:: TensorPipeRpcBackendOptions
    :members:
    :inherited-members:
```

Note

The RPC framework does not automatically retry any :meth:`~torch.distributed.rpc.rpc_sync`, :meth:`~torch.distributed.rpc.rpc_async` and :meth:`~torch.distributed.rpc.remote` calls. The reason being that there is no way the RPC framework can determine whether an operation is idempotent or not and whether it is safe to retry. As a result, it is the application's responsibility to deal with failures and retry if necessary. RPC communication is based on TCP and as a result failures could happen due to network failures or intermittent network connectivity issues. In such scenarios, the application needs to retry appropriately with reasonable backoffs to ensure the network isn't overwhelmed by aggressive retries.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 194); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 194); *backlink*

Unknown interpreted text role "meth".

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

line 194); *backlink*

Unknown interpreted text role "meth".

RRef

Warning

RRefs are not currently supported when using CUDA tensors

An `RRef` (Remote REFerence) is a reference to a value of some type `T` (e.g. `Tensor`) on a remote worker. This handle keeps the referenced remote value alive on the owner, but there is no implication that the value will be transferred to the local worker in the future. RRefs can be used in multi-machine training by holding references to nn.Modules that exist on other workers, and calling the appropriate functions to retrieve or modify their parameters during training. See :ref:`remote-reference-protocol` for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 214); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 224)

Unknown directive type "autoclass".

```
.. autoclass:: RRef
    :members:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 228)

Unknown directive type "toctree".

```
.. toctree::
    :caption: More Information about RRef

    rpc/rref
```

RemoteModule

Warning

RemoteModule is not currently supported when using CUDA tensors

`RemoteModule` is an easy way to create an nn.Module remotely on a different process. The actual module resides on a remote host, but the local host has a handle to this module and invoke this module similar to a regular nn.Module. The invocation however incurs RPC calls to the remote end and can be performed asynchronously if needed via additional APIs supported by RemoteModule.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 247)

Unknown directive type "autoclass".

```
.. autoclass:: torch.distributed.nn.api.remote_module.RemoteModule
    :members: remote_parameters, get_module_rref
```

Distributed Autograd Framework

Warning

Distributed autograd is not currently supported when using CUDA tensors

This module provides an RPC-based distributed autograd framework that can be used for applications such as model parallel training. In short, applications may send and receive gradient recording tensors over RPC. In the forward pass, we record when gradient recording tensors are sent over RPC and during the backward pass we use this information to perform a distributed backward pass using RPC. For more details see :ref:`distributed-autograd-design`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 257); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 264)

Unknown directive type "automodule".

```
.. automodule:: torch.distributed.autograd
    :members: context, backward, get_gradients
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 267)

Unknown directive type "toctree".

```
.. toctree::
    :caption: More Information about RPC Autograd

    rpc/distributed_autograd
```

Distributed Optimizer

See the torch.distributed.optim page for documentation on distributed optimizers.

Design Notes

The distributed autograd design note covers the design of the RPC-based distributed autograd framework that is useful for applications such as model parallel training.

- :ref:`distributed-autograd-design`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 282); *backlink*

Unknown interpreted text role "ref".

The RRef design note covers the design of the :ref:`rref` (Remote REFerence) protocol used to refer to values on remote workers by the framework.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]rpc.rst, line 284); *backlink*

Unknown interpreted text role "ref".

- :ref:`remote-reference-protocol`

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

Unknown interpreted text role "ref".

Tutorials

The RPC tutorials introduce users to the RPC framework, provide several example applications using :ref:`torch.distributed.rpc<distributed-rpc-framework>` APIs, and demonstrate how to use the profiler to profile RPC-based

workloads.

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

Unknown interpreted text role "ref".

- Getting started with Distributed RPC Framework
- Implementing a Parameter Server using Distributed RPC Framework
- Combining Distributed DataParallel with Distributed RPC Framework (covers **RemoteModule** as well)
- Profiling RPC-based Workloads
- Implementing batch RPC processing
- Distributed Pipeline Parallel