Contributing to PyTorch Distributed

Please go through PyTorch's top level Contributing Guide before proceeding with this guide.

PyTorch Distributed Overview is a great starting point with a lot of tutorials, documentation and design docs covering PyTorch Distributed. We would highly recommend going through some of that material before you start working on PyTorch Distributed.

In this document, we mostly focus on some of the code structure for PyTorch distributed and implementation details.

C10D and DistributedDataParallel

The figure below demonstrates building blocks of the c10d and DDP package and shows how typically an application is layered on top. Most parts of the distributed package are implemented in C++ and then bound to the Python frontend (see c10d/init.cpp).

C10D_ARCH

Process Groups

Process groups (PG) take care of communications across processes. It is up to users to decide how to place processes, e.g., on the same machine or across machines. PG exposes a set of communication APIs, e.g., send, recv, broadcast, allgather, allreduce, etc.

Source Code: ProcessGroup.cpp and ProcessGroup.hpp

Process Group Backends We currently offer three backends for Process Groups: ProcessGroupGloo.hpp, ProcessGroupMPI.hpp and ProcessGroup-NCCL.hpp

Store Processes discover each other through a rendezvous process on a common Store (See Store.hpp for the interface and FileStore.hpp, TCPStore.hpp and PrefixStore.hpp for implementations.)

Distributed Data Parallel

DDP is implemented as a module in distributed.py with some of the core functions implemented in reducer.cpp and comm.cpp. Gradients synchronizations occur in backward pass, triggered as autograd hooks.

Onboarding Tasks

A list of onboarding tasks can be found here and here.

RPC Framework

The figure below demonstrates the overall architecture of the RPC framework.

RPC_ARCH

The top level APIs for the RPC framework can found in rpc/api.py and majority of the code is actually written in C++. The pybind entrypoints can be found in rpc/init.cpp.

The RPC framework consists of several additional components:

RPC Agents

The core C++ interface of the RPC framework can be found in rpc_agent.h and the TensorPipe and ProcessGroupGloo implementations can be found at process_group_agent.h and tensorpipe_agent.h respectively.

request_callback.h and request_callback_impl.h deal with how to handle RPC calls on remote servers.

Remote Reference (RRef)

Most of the APIs for RRefs can be found in rpc/api.py. The C++ interface can be found in rref_interface.h and implementations in rref_impl.h and rref_context.h.

Distributed Autograd

The top level APIs for distributed autograd can be found in distributed/autograd/init.py and distributed/autograd/init.cpp.

The core engine for executing a distributed backward pass can be found in dist_engine.h

Distributed Optimizer

The distributed optimizer is completely written in Python and can be found at optimizer.py

Onboarding Tasks

A list of onboarding tasks can be found here.

Running unit tests

All the unit tests can be found under the test/distributed directory and RPC tests in particular are under test/distributed/rpc. A few examples on how to run unit tests:

```
# Run the c10d unit tests.
python test/distributed/test_c10d_common.py
python test/distributed/test_c10d_gloo.py
python test/distributed/test_c10d_nccl.py

# Run the Store tests.
python test/distributed/test_store.py

# Run Process Group Wrapper tests.
python test/distributed/test_pg_wrapper.py

# Run distributed tests, including tests for Distributed Data Parallel.
python test/run_test.py --verbose -i distributed/test_distributed_spawn

# Run a single test in the test_distributed_spawn test suite.
touch /tmp/barrier && TEMP_DIR="/tmp" BACKEND="nccl" WORLD_SIZE="2" python test/distributed/

# Run the RPC test suite for the TensorPipeAgent.
python test/distributed/rpc/test_tensorpipe_agent.py
python test/distributed/rpc/cuda/test_tensorpipe_agent.py

# Run the RPC test suite for the FaultyAgent
python test/distributed/rpc/test_faulty_agent.py

# Run a specific test method.
pytest -k test_self_add test/distributed/rpc/test_process_group_agent.py
```

Note that the RPC framework is by default only tested with filesystem initialization. To run tests with TCP initialization, set the environment variable `RPC_INIT_WITH_TCP=1` before running your test command.