MapReduce Subnet Documentation

Release 1.0.0

December 08, 2023

Welcome! MapReduce Subnet's documentation is split into 2 main parts:

- Chapter 1
- Chapter 2

For information on using the MapReduce Subnet, see Chapter 1.
For information on which functions in MapReduce Subnet, see Chapter 2.

Chapter 1

Bittensor Map Reduce

Error opening image file: cannot identify image file '/__w/map-reduce-doc/map-reduce-doc/docs/_build/rinoh/.doctrees/images/1ea27c6fc493edc1e34966685a29dd3b1209aa84/308323056592486420.svg'

Error opening image file: cannot identify image file '/__w/map-reduce-doc/map-reduce-doc/docs/_build/rinoh/.doctrees/images/520f473109738a62720092529c38de6bb2bf7b78/License-MIT-yellow.svg'

The Incentivized Internet

Discord • Network • Research

1.1 Introduction

The Bittensor Subnet 10 (Map Reduce Subnet) incentivizes miners by offering rewards for contributing network bandwidth and memory resources.

A broadcast subnet leverages the bandwidth of multiple peers to transfer large data from point to A to multiple point Bs without needing to leverage large quantities of your own upload. The concept is simple, a large file D can be split into multiple chunks and sent to N intermediate peers (usually with redundancy) and then forwarded onward to B additional endpoints in an N by B full bipartite fashion. The inverse operation is also valuable, where data DxB large data files can be aggregated from B peers by leveraging the bandwidth of N intermediaries.

In the forward 'map' operation a file D is broken into chunks and split across the N peers each of whom then forwards their chunk to B endpoints allowing each downloading peer to recieve the full file of size D with the sending peer needing an upload of only size D. The backward operation, 'reduce', acts in reverse, the K receiving peers fan out their response data D in chunks to the N intermediary peers who then aggregate the chunks from each other and finally send the sum of total chunks back to the sending peer A.

The map-reduce cycle is essential for reducing the bandwidth by a factor of K on the running peers which is essential for the training of machine learning models in a distributed setting. This template is a protoype for incentivizing the speed at which this operation can take place by validating both the consistency and operation speed of a map-reduce.

1.2 How it works

Error opening image file: cannot identify image file '/__w/map-reduce-doc/map-reduce-doc/docs/map_reduce.svg'

Figure 1.1. Map Reduce Diagram

The diagram illustrates the workflow of a distributed map-reduce system with an integrated validation mechanism:

Peer Gradient Splitting:

Peers do machine training and generate unique gradients. These gradients are divided into segments (Seg1, Seg2, Seg3 in the diagram), which are then distributed among the miners (Miner1, Miner2, Miner3 in the diagram).

Miner Gradient Processing:

Each miner receives segments from the peers. The miners perform computations on these segments, which could involve averaging, sum or other forms of data processing. After processing, each miner holds an averaged gradient segment, denoted as g^ (1) for Miner1, g^ (2) for Miner2, and g^ (3) for Miner3.

Gradient Broadcasting and Aggregation:

The miners then broadcast their processed gradient segments back to all peers. Each peer collects these averaged gradient segments, reconstructing the full set of averaged gradients, which could then be used for further computations or iterations within a larger algorithm.

Validation:

A validator independently samples small subsets of data from both the peers and the miners. The validator's role is to confirm that the miners' computations are accurate and that the integrity of the data remains intact through the process. This system ensures the distributed processing of data with a check-and-balance system provided by the validator. This validation step is crucial for maintaining the reliability of the distributed computation, especially in decentralized or trustless environments where the computation's correctness cannot be taken for granted.

1.3 Installation

This repository requires python3.8 or higher. To install, simply clone this repository and install the requirements.

1.3.1 Install Dependencies

```
git clone https://github.com/dream-well/map-reduce-subnet
cd map-reduce-subnet
python3 -m pip install -e .
```

1.4 Running Miner

1.4.1 Prerequisites

For running a miner, you need enough resources. The minimal requirements for running a miner are

- Public IP address
- Network bandwidth: 1Gbps
- RAM: 10GB

Recommended hardware requirement: - Network bandwidth: 10Gbps - RAM: 32GB

Note: Higher network bandwidth and RAM can lead to more rewards.

1.4.2 Running Miner Script

Run the miner using the following script:

```
# To run the miner
python3 neurons/miner.py
    --netuid 10  # The subnet id you want to connect to
    --subtensor.network finney  # blockchain endpoint you want to connect
    --wallet.name <your miner wallet> # name of your wallet
    --wallet.hotkey <your miner hotkey> # hotkey name of your wallet
    --logging.debug # Run in debug mode, alternatively --logging.trace for trace
mode
```

Important Note: Operating multiple miners from a single machine (using the same IP address) may result in reduced rewards. For optimal performance and reward maximization, it is recommended to run each miner on a separate machine.

1.5 Running Validator

Validators oversee data transfer processes and ensure the accuracy and integrity of data transfers.

1.5.1 Prerequisites

1. (Optional) I recommend to run subtensor instance locally

```
git clone https://github.com/opentensor/subtensor.git
cd subtensor
docker compose up --detach
```

2. For validating, you need to setup benchmark bots first.

Setup Benchmark Bots

Validators should set up at least 3 benchmark bots (more bots lead to faster validation). You can install one of the bots directly on your validator machine. Each bot should run on a different machine with a minimum of 4GB RAM and 1Gbps network bandwidth.

Minimum hardware requirement: - RAM: 4GB - Network: 1Gbps

Recommended hardware requirement: - RAM: 8GB - Network: 10Gbps

```
# To run the benchmark bot
python3 neurons/benchmark.py
    --subtensor.network local  # blockchain endpoint you want to connect
    --wallet.name <your benchmark wallet> # name of your wallet
    --wallet.hotkey <your benchmark hotkey> # hotkey name of your wallet, you can
create a new wallet for benchmark and register in validator.config.json
    --validator.uid <your validator uid> # your validator uid
```

1.5.2 Configuration with validator.config.json

Modify validator.config.json with appropriate settings, including hotkeys for benchmark bots.

```
cp example.validator.config.json validator.config.json
```

```
{
    "subtensor.network": "local",
    "netuid": 10,
    "wallet.name": "your wallet name",
    "wallet.hotkey": "your validator hotkey",
    "max_bandwidth": 10737418240,
    "auto_update": "patch",
    "benchmark_hotkeys": [
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        ""
    ]
}
```

- max_bandwidth: The maximum bandwidth of your benchmark bots. (default 10 Gb)
- benchmark_hotkeys: The hotkeys of your benchmark bots. (default 5 bots)

1.5.3 Running Validator Script

```
cd neurons
# To run the validator
python3 validator.py
    --netuid 10 # The subnet id you want to connect to
    --subtensor.network local # blockchain endpoint you want to connect
    --wallet.name <your validator wallet>  # name of your wallet
    --wallet.hotkey <your validator hotkey> # hotkey name of your wallet
    --logging.debug # Run in debug mode, alternatively --logging.trace for trace
mode
```

1.6 Tutorial

1.6.1 User Guide

```
git clone https://github.com/dream-well/map-reduce-subnet
cd map-reduce-subnet
python3 -m pip install -e .
```

```
import torch
import time
from peer.peer import Peer
```

```
import bittensor as bt
from argparse import ArgumentParser

parser = ArgumentParser()
parser.add_argument('--validator.uid', type = int, default= 0, help='Validator
UID')
parser.add_argument('--netuid', type = int, default=10, help='Map Reduce Subnet
NetUID')
parser.add_argument('--rank', type = int, default=1, help='Rank of the peer')
parser.add_argument('--count', type = int, default=1, help='Number of peers')

config = bt.config(
    parser=parser
)

wallet = bt.wallet(config=config)

# size for testing, set to 100 MB
test_size = 100 * 1024 * 1024

def train(rank, peer_count, bandwidth, wallet, validator_uid, netuid, network ):
    bt.logging.info(f"?{rank} netuid: {netuid}")
    # Initialize Peer instance
    peer = Peer(rank, peer_count, bandwidth, wallet, validator_uid, netuid,
network)

    # Initialize process group with the fetched configuration
    peer.init_process_group()

    weights = None

    if rank -- 1: # if it is the first peer
        weights = torch.rand((int(test_size / 4), 1), dtype=torch.float32)
        peer.broadcast(weights)
    else:
        weights = peer.broadcast(weights)

    epoch = 2

    # Your training loop here
    bt.logging.info(f"Peer {rank} is training...")
    for i in range(epoch):

        bt.logging.success(f"?{i}")
        # Replace this with actual training code
        time.sleep(5)

        # After calculating gradients
        gradients = torch.ones((int(test_size / 4), 1), dtype=torch.float32)
        if rank -- 1:
            gradients = torch.ones((int(test_size / 4), 1), dtype=torch.float32) *
3

        # All-reducing the gradients
        gradients = peer.all_reduce(gradients)

    peer.destroy_process_group()
    print(f"Peer {rank} has finished training.")

if __name__ -- '__main__':
    train(config.rank, config.peer_count, test_size, wallet, config.validator.uid,
config.netuid, config.subtensor.network)
```

User also can test the map-reduce subnet by running test.py

```
python3 test/test.py --subtensor.network local --wallet.name <wallet name>
--wallet.hotkey <hotkey name> --validator.uid <validator uid>
```

1.6.2 License

This repository is licensed under the MIT License.

```
# The MIT License (MIT)
# Copyright © 2023 ChainDude

# Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated
# documentation files (the "Software"), to deal in the Software without
restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software,
# and to permit persons to whom the Software is furnished to do so, subject to
the following conditions:

# The above copyright notice and this permission notice shall be included in all
copies or substantial portions of
# the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO
# THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
```

peer.py

class mapreduce.peer.**Peer** (*rank*, *peer_count*, *bandwidth*, *wallet: bittensor.wallet*, *validator_uid: int = -1*, *network='finney'*, *port_range='9000:9010'*, *benchmark_max_size=0*)

Bases: object

Constructor

rank (int, required): Rank of the current process (it should be a number between 1 and peer_count). peer_count(int, required): Number of peers participating in the job. bandwidth(int, required): Maximum size of the job in bytes. wallet(bt.wallet, required): Wallet object containing the hotkey and coldkey. validator_uid: Validator UID to connect to. network: The subtensor network flag. The likely choices are:

– finney (main network) – test (test network) – local (local running network) If this option is set it overloads subtensor.chain_endpoint with an entry point node from that network.

port_range: Port range for gloo, should be allowed in firewall. benchmark_max_size: Maximum size of the benchmark in bytes.

init_process_group ( )

Connect to validator and get job details, initialize process group.

destroy_process_group ( )

Send exit signal to validator, destroy process group

broadcast (*tensor: torch.Tensor*)

Rank1 broadcasts the tensor to the all peers

Args:

tensor (Tensor): Data to be sent if current process is rank1, and tensor to be used to save received data otherwise.

all_reduce (*tensor: torch.Tensor*)

Reduces the tensor data across all peers in such a way that all get the final result. (average) After the call tensor is going to be bitwise identical in all processes. Complex tensors are supported.

Args

tensor (Tensor): Input and output of the collective. The function operates in-place.

request_benchmark ( )

Request benchmark to validator

benchmark ( )

Benchmark function, only benchmark bots will run this function

- genindex
- modindex

- search

m