计算加速套件 TACO Kit

TACO Train AI 训练加速引擎

腾讯云

文档目录

TACO Train AI 训练加速引擎
TACO Train 概述

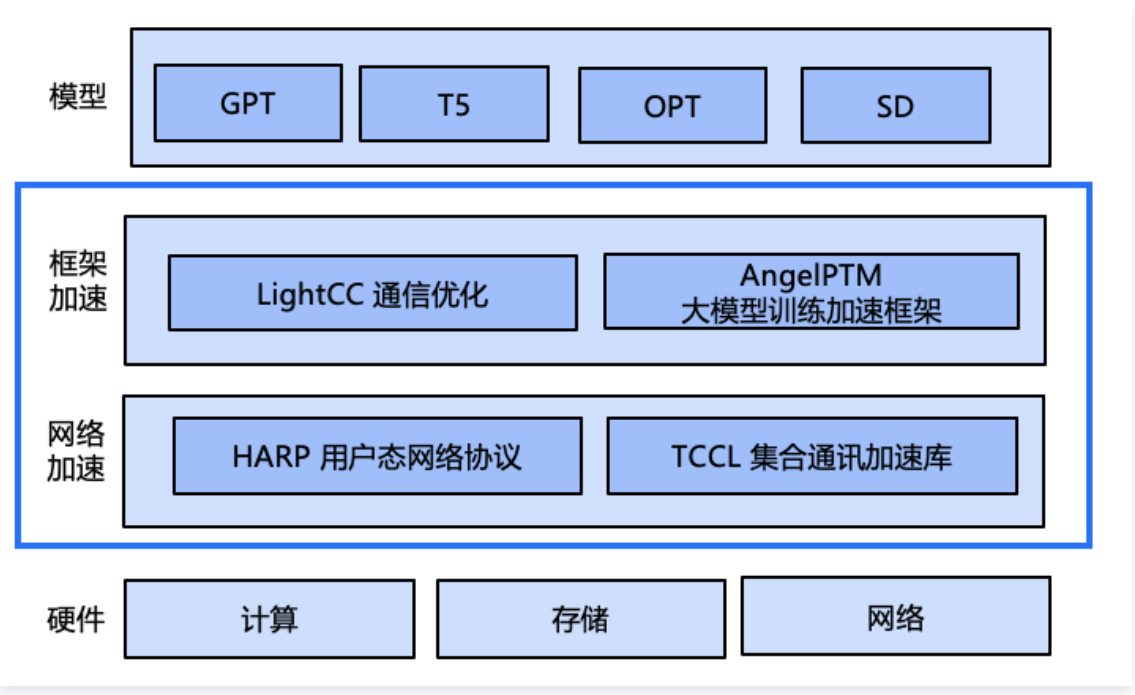
最近更新时间：2024-11-14 11:41:02

背景信息

近几年随着 AI 模型参数的倍增及训练数据的日益增长，用户对模型迭代效率的需求也随之增长，单个 GPU 的算力和显存资源已无法满足大部分业务场景，使用单机多卡或多机多卡训练已成为趋势。单机多卡训练场景的参数同步借助 NVIDIA NVLINK 技术，基本可以获得较高的线性扩展比，但多机多卡训练场景严重依赖多机之间的网络互联技术。网卡厂商提供了高速互联技术 Infiniband 或 RoCE，虽使多机通信效率大幅提升，但成本也大幅增加。如何在普通甚至低速网络环境下提升分布式系统的训练效率，已成为用户关注的焦点。

TACO Train 简介

目前业内已有一些成熟的分布式训练加速技术，例如多级通信、多流通信、梯度融合、压缩通信等，TACO Train 也引入了类似的加速技术。同时，TACO Train 推出了自定义的用户态协议栈 HARP，有效解决普通网络环境下的多机网络通信问题。
TACO Train 是腾讯云**异构计算团队**基于 IaaS 资源推出的 AI 训练加速引擎，为用户提供开箱即用的 AI 训练套件。TACO Train 基于腾讯内部丰富的 AI 业务场景，提供自底向上的网络通信、分布式策略及训练框架等多层级的优化，是一套全生态的训练加速方案。



目前 TACO Train 提供了4个训练加速组件：

- **AngelPTM**：大模型预训练框架，支持 GPT/T5/BERT 等大模型。
- **TCCL**：针对腾讯云星脉网络架构的高性能定制加速通信库，为 AI 大模型训练提供更高效的网络通信性能。
- **HARP**：自研用户态网络协议栈，加速普通网络环境的多机通信效率。
- **LightCC**：分布式训练框架，支持 Horovod， Ray 或者 pytorch DDP 的分布式训练加速。

AngelPTM

AngelPTM 是基于 DeepSpeed 和 Megatron 深度定制开发的大模型训练框架，支持 NLP/ 多模态 /AIGC 等多类预训练任务。由于大模型的参数规模巨大，对硬件存储资源提出了挑战，AngelPTM 支持以更少的资源和更快的速度训练大模型，兼容社区方案 API ，支持业务快速接入，

- **ZeRO Cache 策略**：基于 ZeRO 策略把内存作为二级存储 offload 参数、梯度、优化器状态到 CPU 内存（也支持 SSD 作为第三级存储）。ZeRO-Cache 为了最大化利用内存和显存进行模型状态的缓存，引入了显存内存统一存储视角，将存储容量的上界由内存扩容到内存+显存总和。同时将多流异步化做到了高效，在 GPU 计算的同时进行数据 IO 和 NCCL 通信，使用异构流水线均衡设备间的负载，最大化提升整个系统的吞吐。ZeRO-Cache 将 GPU 显存、CPU 内存统一视角管理，击破了异构存储的壁垒，减少了冗余存储和内存碎片，增加了内存的利用率，极大扩充了模型存储可用空间。
- **自动流水并行**：3D 并行是 Megatron 的原始能力，但是 Megatron 的流水并行只支持 block 级别（Transformer Layer），且需要依靠专家经验人工指定 stage 切分以确保 stage 间负载均衡；自动流水并行可以做到 op 级别，且自动 profile op 性能，自动搜索负载均衡的切分方案后执行流水并行训练。
- **高性能 MoE 组件**：基于拓扑感知的 AlltoAll 通信加速专家并行，提供高性能定制算子，支持计算通信流水提高迭代效率。

TCCL

TCCL（Tencent Collective Communication Library）是一款针对腾讯云星脉网络架构的高性能定制加速通信库。主要功能是依托星脉网络硬件架构，为 AI 大模型训练提供更高效的网络通信性能，同时具备网络故障快速感知与自愈的智能运维能力。TCCL 基于开源的 NCCL 代码做了扩展优化，完全兼容 NCCL 的功能与使用方法。TCCL 目前支持主要特性包括：

- 双网口动态聚合优化，发挥 bonding 设备的性能极限。
- 全局 Hash 路由（Global Hash Routing），负载均衡，避免拥塞。
- 拓扑亲和性流量调度，最小化流量绕行。

HARP

随着网络硬件技术的发展，网络带宽从 10Gbps 增长到 100Gbps 甚至更高，在数据中心大量部署使用。但目前普遍使用的内核网络协议栈存在着一些必要的开销，使其不能很好地利用高速网络设备。为解决该问题，腾讯云自研了用户态网络协议栈 HARP，可以以 Plug-in 的方式集成到 NCCL 中，无需任何业务改动，加速云上分布式训练性能。在 VPC 的环境下，相比传统的内核协议栈，HARP 提供了以下的能力：

- 支持全链路内存零拷贝，HARP 协议栈提供特定的 buffer 给应用，使应用的数据经过 HARP 协议栈处理后由网卡直接进行收发，消除内核协议栈中耗时及占用 CPU 较高的多次内存拷贝操作。
- 支持协议栈多实例隔离，即应用可以在多个 CPU core 上创建特定协议栈实例处理网络报文，每个实例间相互隔离，保证性能线性增长。
- 数据平面无锁设计，HARP 协议栈内部保证网络 session 的数据仅在创建该 session 的 CPU core 上，使用特定的协议栈实例处理。减少了内核中同步锁的开销，也降低了 CPU 的 Cache Miss 率，大幅提升网络数据的处理性能。

LightCC

LightCC 对社区分布式方案的通信策略进行了深度定制优化，完全兼容 Horovod ，Ray 或者 pytorch DDP API，支持业务快速接入。主要包括的优化能力如下：

- 2D AllReduce 充分利用通信带宽。
- 高效的梯度融合方式。
- TOPK/FP16 压缩通信，降低通信量，提高传输效率。
- 简单高效地管理和分配分布式训练任务，具有较强的扩展性和可靠性。

TTF

TensorFlow 是深度学习领域中应用最广泛的开源框架之一，但在很多业务场景下，开源 Tensorflow 有其特定的限制。为了解决实际业务中遇到的问题，Tencent Tensorflow（以下简称 TTF）提供了以下能力：

- 相比原始的静态 Embedding，高维稀疏动态 Embedding 帮助用户在不需要重新训练的条件下，动态添加和删除特征，按需使用内存，避免 Hash 冲突，同时保留原始 TF 的 API 设计风格。
- 混合精度在原有实现的基础上增加了调整精度的策略，根据 loss 的状态自动在全精度和半精度之间切换，避免精度损失。
- 针对特定业务场景的 XLA，Grappler 图优化，以及自适应编译框架解决冗余编译的问题。

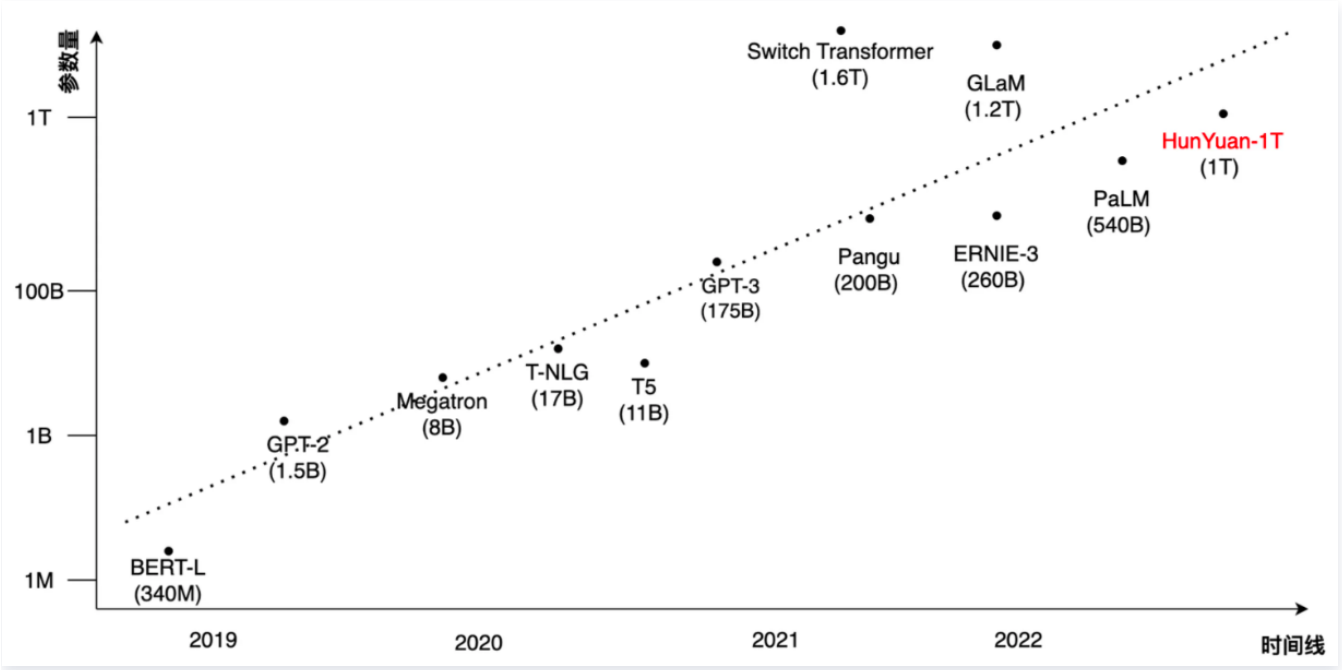
- 开源 TF 1版本不再提供对 Ampere GPU 的支持，但考虑到较多用户仍在使用 TF 1.15版本的问题，TTF 添加了对 CUDA 11的支持，让用户可以使用 A100来进行模型训练。

使用 TACO Train 部署分布式集群大模型预训练最佳实践

最近更新时间：2024-11-25 15:16:52

背景信息

最近 ChatGPT 凭借其强大的语言理解能力、文本生成能力、对话能力等在多个领域均取得了巨大成功，掀起了新一轮的人工智能浪潮。ChatGPT，GPT3，以及 GPT3.5 都是基于 Transformer 架构堆叠而成，研究发现随着训练数据量和模型容量的增加可以持续提高模型的泛化能力和表达能力，研究大模型成为了近两年的趋势。国内外头部科技公司均有布局，发布了若干千亿规模以上的大模型，如下图所示。



近几年 NLP 预训练模型规模的发展，模型已经从亿级发展到了万亿级参数规模。具体来说，2018 年 BERT 模型最大参数量为3.4亿，2019 年 GPT-2 为十亿级参数的模型。2020 年发布的百亿级规模有 T5 和 T-NLG，以及千亿参数规模的 GPT-3。在 2021 年末，Google 发布了 SwitchTransformer，首次将模型规模提升至万亿。然而GPU硬件发展的速度难以满足 Transformer 模型规模发展的需求。近四年中，模型参数量增长了十万倍，但 GPU 的显存仅增长了4倍。

举例来说，万亿模型的模型训练仅参数和优化器状态便需要 1.7TB 以上的存储空间，至少需要 425 张 A100-40G，这还不包括训练过程中产生的激活值所需的存储。在这样的背景下，大模型训练不仅受限于海量的算力，更受限于巨大的存储需求。
业内大模型预训练的解决方案主要包括微软的 DeepSpeed 和英伟达的 Megatron-LM。DeepSpeed 引入 ZeRO（Zero Redundancy Optimizer ）优化器，将模型参数、梯度、优化器状态按需分配到不同的训练卡上，满足大模型对存储的要求；Megatron-LM 基于3D并行（张量并行、流水并行、数据并行）将模型参数进行切分，满足大模型在有限显存资源下的训练诉求。

腾讯内部也有非常多的大模型预训练业务，为了以最小的成本和最快的性能训练大模型，太极机器学习平台对 DeepSpeed 和 Megatron-LM 进行了深度定制优化，推出了AngelPTM 训练框架。2022 年 4 月腾讯发布的混元（ HunYuan-1T ）AI大模型便是基于 AngelPTM 框架训练而来。鉴于最近大模型的火热趋势，我们决定将内部成熟落地的 AngelPTM 框架推广给广大公有云用户，帮助业务降本增效。

AngelPTM简介

AngelPTM 基于 ZERO策略，将模型的参数、梯度、优化器状态以模型并行的方式切分到所有GPU，并自研ZeRO-Cache框架把内存作为二级存储offload参数、梯度、优化器状态到CPU内存，同时也支持把SSD作为第三级存储。ZeRO-Cache为了最大化最优化的利用内存和显存进行模型状态的缓存，引入了显存内存统一存储视角，将存储容量的上界由内存扩容到内存+显存总和。同时结合多流异步化能力，在GPU计算的同时进行数据IO和NCCL通信，使用异构流水线均衡设备间的负载，最大化提升整个系统的吞吐。ZeRO-Cache将GPU显存、CPU内存统一视角管理，减少了冗余存储和内存碎片，增加了内存的利用率。

部署步骤

本文指导用户基于腾讯云高性能计算集群和TACO AngelPTM训练框架，搭建大模型训练环境，展示AngelPTM的强大能力。

购买高性能计算实例

购买实例，其中实例、存储及镜像请参考以下信息选择，其余配置请参见 通过购买页创建实例 按需选择。

- **实例**：选择 GPU 型 HCCPNV4h 。
- **镜像**：建议选择**公共镜像**，公共镜像当中已安装 RDMA 网卡驱动，且支持自动安装 GPU 驱动。
  - 操作系统请使用 CentOS 7.6、Ubuntu 18.04 或 TencentOS 2.4（TK4）。
  - 若您选择**公共镜像**，则请勾选“后台自动安装GPU驱动”，实例将在系统启动后预装对应版本驱动。如下图所示：

若您选择**自定义镜像**，则需要自行安装 RDMA 网卡驱动和 GPU 驱动，请通过 联系我们 获取腾讯云售后支持。

安装docker和NVIDIA docker

1. 参考 使用标准登录方式登录 Linux 实例 ，登录实例。
2. 执行以下命令，安装 docker。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-docker.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令。

3. 执行以下命令，安装 nvidia-docker2。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参考 NVIDIA 官方文档 Installation Guide & mdash 进行安装。

启动AngelPTM运行环境

```
#!/bin/bash

docker run \
    -itd \
    --gpus all \
    --privileged --cap-add=IPC_LOCK \
    --ulimit memlock=-1 --ulimit stack=67108864 \
    --net=host \
    --ipc=host \
    --name=ptm \
    ccr.ccs.tencentyun.com/qcloud/taco-train:torch112-cu116-bm-0.6.1

docker exec -it ptm bash
```

该镜像包含的版本信息如下：

- OS：Ubuntu 20.04.5 LTS
- python：3.8.10
- CUDA toolkit: V11.6.124
- cuDNN: 8.4.0
- tencent-lightcc: 3.1.1
- pytorch: 1.12.1+cu116
- AngelPTM DeepSpeed: 0.6.1+93ef832c
- AngelPTM Megatron-DeepSpeed:1.1.5
- AngelPTM ptm: 0.1

开始测试

28B GPT模型

社区版本 Zero Stage3，单机 A100 40G 所能容纳的最大模型规模。

```
cd /workspace/examples/ptm
bash start.sh 28
```

55B GPT模型

AngelPTM Zero Cache，单机 A100 40G 所能容纳的最大模型规模。

```
cd /workspace/examples/ptm
bash start.sh 55
```

评测其他模型规模

可以通过修改start.sh添加其他的模型规模配置：

```
if [ $MODEL_SIZE == "28" ]; then
        NUM_LAYERS=26
        HIDDEN_SIZE=8192
        NUM_ATTN_HEADS=128
```

```
        FFN_HIDDEN_SIZE=32768
        ATTN_HEAD_SIZE=128
        BATCH_SIZE=38
        CONFIG_PREFETCH_CACHE_SUB_GROUP_RATE=0.0
elif [ $MODEL_SIZE == "55" ]; then
        NUM_LAYERS=68
        HIDDEN_SIZE=8192
        NUM_ATTN_HEADS=64
        FFN_HIDDEN_SIZE=32768
        ATTN_HEAD_SIZE=128
        BATCH_SIZE=10
        CONFIG_PREFETCH_CACHE_SUB_GROUP_RATE=0.1
else
        echo "ERROR: Please supplement new model configuration to test!"
        exit -1
fi
```

配置参数说明

相比原始的 ds config，AngelPTM提供了一些额外的控制参数（参考/workspace/examples/ptm/ds_config_gpt_zero3_maxscale.json），其功能说明说下：

"pipeline_optimizer": true,	打开模型更新过程中流水操作，默认打开，无需配置
"prefetch_cache_sub_group_rate": 0.0	值范围为0.0~1.0，表示放置模型状态到 GPU 的比例： 0.1表示10%的模型状态存储到显存，这样可以支持更大的模型尺寸； 0.0表示模型状态不占用显存资源，这样可以支持更大的 batch，获取更好的性能
"max_contigous_params_size": 2e9,	ZeRO Cache 管理的显存大小，和 stage3_prefetch_bucket_size 以及 stage3_max_live_parameters 配合使用，如果预取参数变大，其值也需要调大，在显存比较充足的情况下可以考虑关闭，设置为-1
"max_param_reduce_events": 0,	梯度 reduce_scatter 同时进行的 event 个数，建议1或者0，配置太大会导致显存 OOM
"is_communication_time_profiling": false,	调试相关，用于打印通信时间，默认关闭
"save_large_model_multi_slice": true,	分片模型保存，避免内存 OOM，默认打开，无需配置

说明：

- 模型测试脚本来自Megatron-DeepSpeed 训练脚本 。
- 第一次测试启动较慢主要是由于数据集下载和预处理。
- 如果想要终止测试，启动新一轮测试之前建议先`ps`确认上次测试程序已经完全退出（由于AngelPTM会使用大量的内存资源cache模型参数和优化器状态，cache的释放需要一段时间才能完成，通常几分钟就好）。

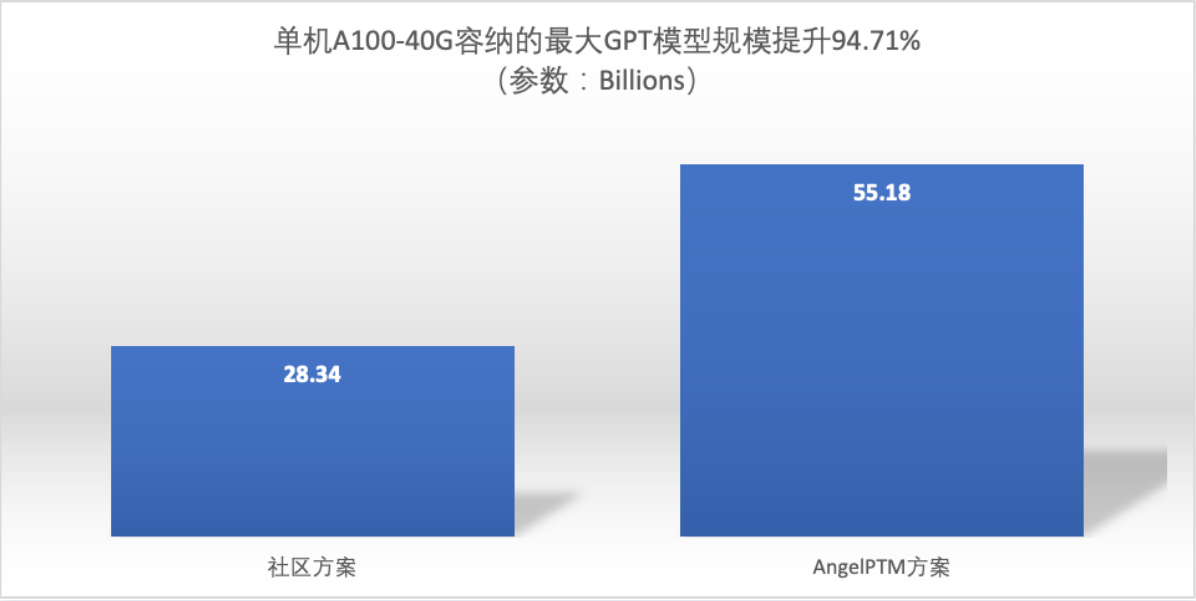
性能结果

方案	DeepSpeed	Megatron-DeepSpeed
社区方案	0.8.1+258d2831	7212b58

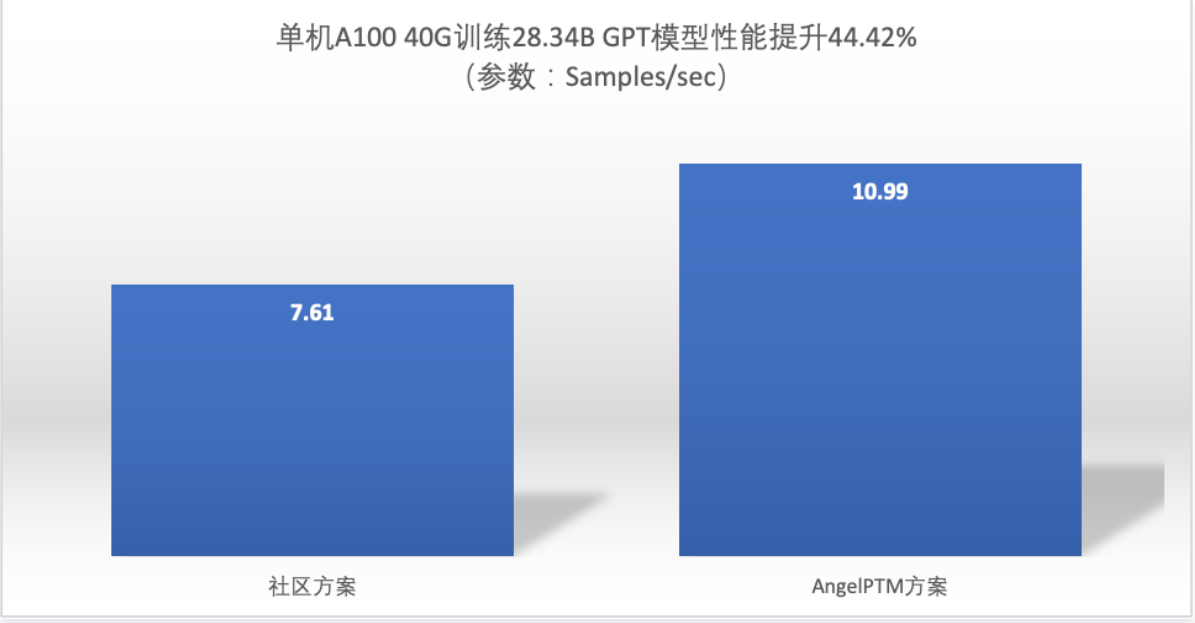
AngelPTM方案	0.6.1+474caa20	c5808e0

注意：其他环境，例如OS/python/CUDA/cuDNN/pytorch等版本二者一致。

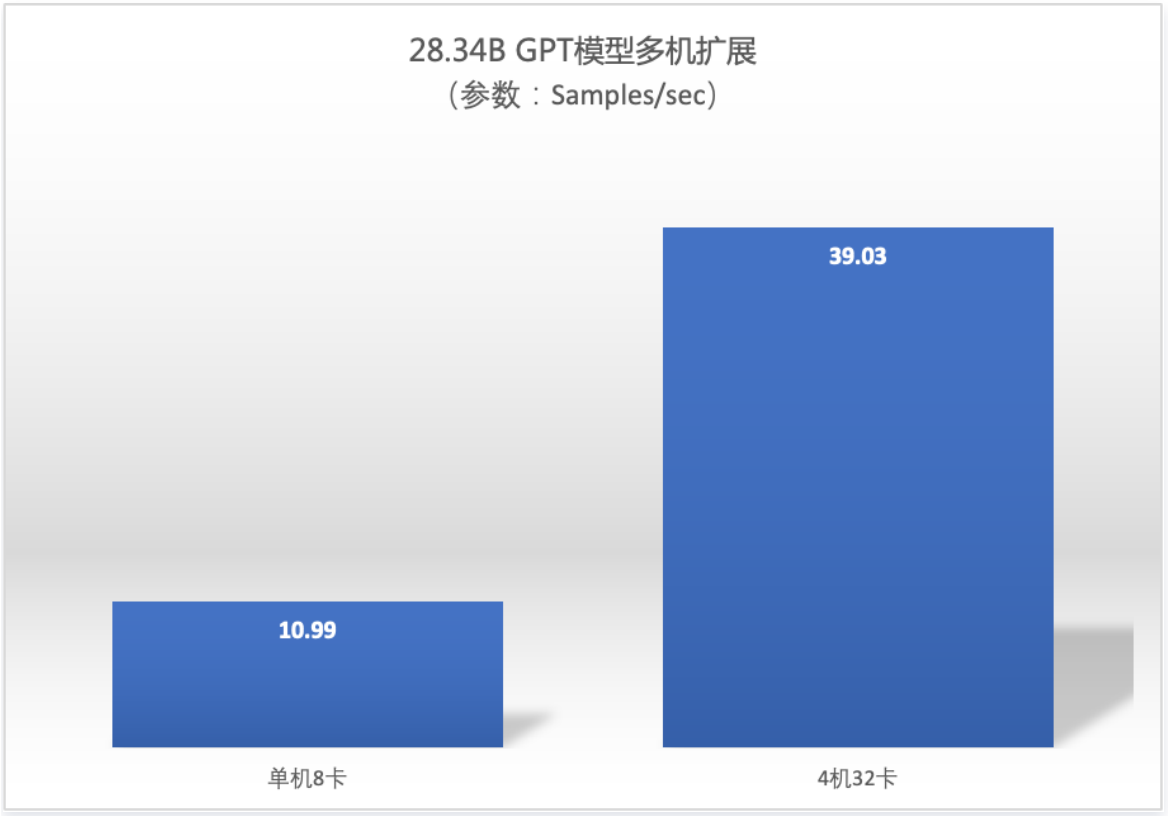
最大容纳模型规模



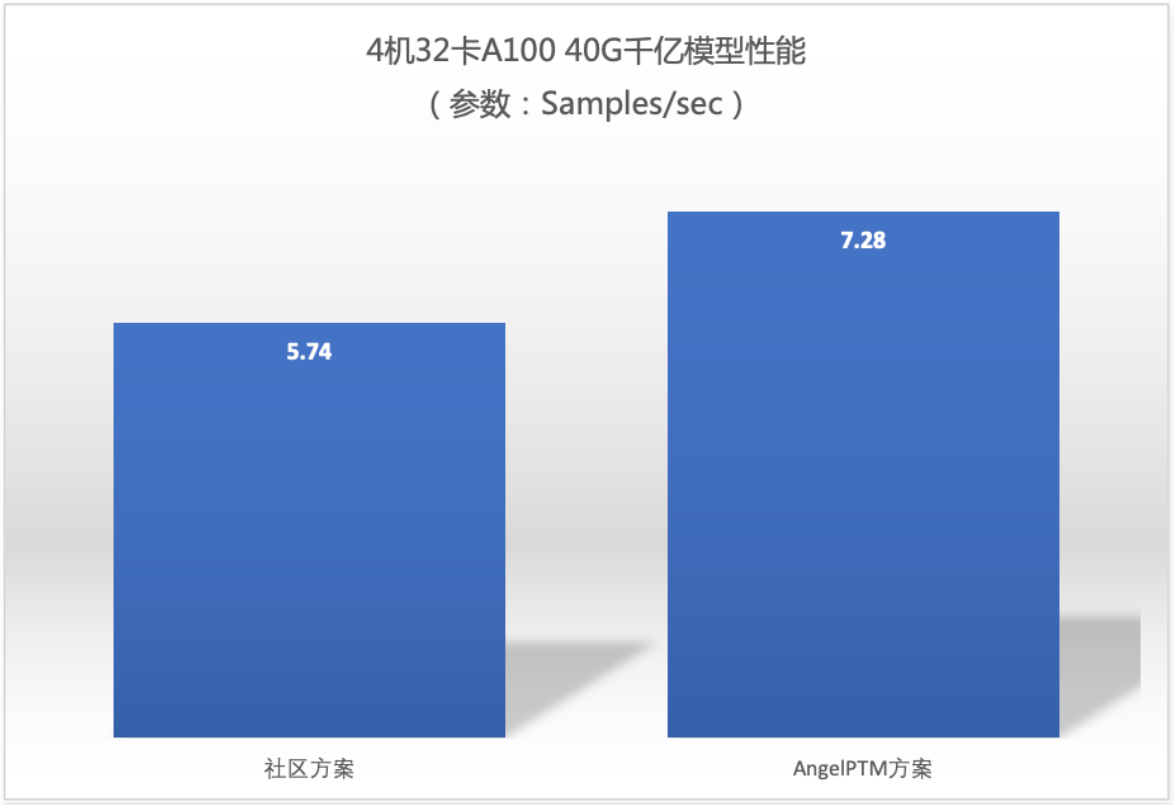
同等模型规模训练性能



多机扩展比



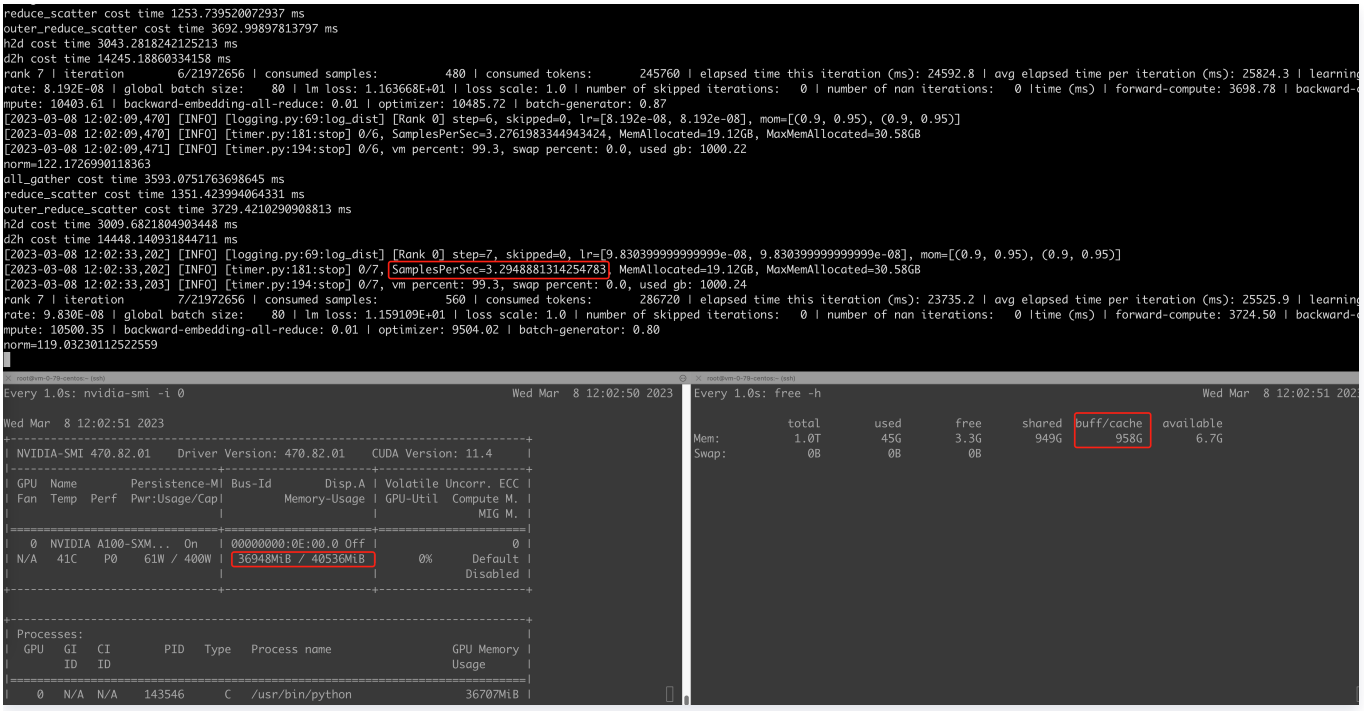
千亿模型性能



注意：4机32卡训练性能提升比例（26.8%）相比单机有所下降主要是由于网络带宽限制。这里使用的是100Gbps RDMA 网络，未来腾讯云会推出更高带宽高性能计算集群，预期性能提升会与单机接近。

监控信息

下图是单机 A100 40G 训练550亿参数的 GPU 和内存监控信息



结论

本文简单介绍了 AngelPTM 产生的背景和技术能力，然后指导用户如何在腾讯云高性能计算集群下搭建 AngelPTM 训练环境。可以看出，相比社区方案：

- AngelPTM 将单机 A100 40G 容纳的模型规模提升了94.71%
- 基于社区方案能容纳的最大模型规模，AngelPTM 性能提升了44.42%
- 千亿模型规模下，AngelPTM 多机扩展比接近线性

在 CVM 上部署 TensorFlow 分布式训练集群

最近更新时间：2025-06-09 18:02:52

操作场景

本文介绍如何基于云服务器 CVM 搭建 Tensorflow+Taco Train 分布式训练集群。

操作步骤

购买实例

购买实例，其中实例、存储及镜像请参考以下信息选择，其余配置请参见 通过购买页创建实例 按需选择。

- **实例**：选择 计算型 GN10Xp 或 GT4 。
- **系统盘**：配置容量不小于50GB的云硬盘。您也可在创建实例后使用文件存储，详情参见 在 Linux 客户端上使用 CFS 文件系统 。
- **镜像**：建议选择**公共镜像**，您也可选择**自定义镜像**。
- 操作系统请使用 TencentOS 3.1/TencentOS 2.4/CentOS 8.0/CentOS 7.8/Ubuntu 20.04/Ubuntu 18.04。
- 若您选择**公共镜像**，则请勾选**后台自动安装 GPU 驱动**，实例将在系统启动后预装对应版本驱动。如下图所示：

> **说明**
> 选择**公共镜像**并自动安装 GPU 驱动的实例，创建成功后，请登录实例等待约20分钟后重启实例，使配置生效。

配置实例环境

验证 GPU 驱动

1. 参见 使用标准登录方式登录 Linux 实例 ，登录实例。
2. 执行以下命令，验证 GPU 驱动是否安装成功。

```
nvidia-smi
```

查看输出结果是否为 GPU 状态：

- 是，代表 GPU 驱动安装成功。
- 否，请参见 NVIDIA Driver Installation Quickstart Guide 进行安装。

配置 HARP 分布式训练环境

1. 参见 配置 HARP 分布式训练环境，配置所需环境。
2. 配置完成后，执行以下命令进行验证，若配置文件存在，则表示已配置成功。

```
ls /usr/local/tfabric/tools/config/ztcp*.conf
```

安装 docker 和 nvidia docker

1. 执行以下命令，安装 docker。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-docker.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参考 Docker 官方文档进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
+ sh -c 'yum install -y -q docker-ce-rootless-extras'
Package docker-ce-rootless-extras-20.10.16-3.el7.x86_64 already installed and latest version

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

2. 执行以下命令，安装 nvidia-docker2。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参考 NVIDIA 官方文档 Installation Guide & mdash 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
Running transaction
  Installing : libnvidia-container1-1.9.0-1.x86_64
  Installing : libnvidia-container-tools-1.9.0-1.x86_64
  Installing : nvidia-container-toolkit-1.9.0-1.x86_64
  Installing : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container-tools-1.9.0-1.x86_64
  Verifying  : nvidia-container-toolkit-1.9.0-1.x86_64
  Verifying  : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container1-1.9.0-1.x86_64

Installed:
  nvidia-docker2.noarch 0:2.10.0-1

Dependency Installed:
  libnvidia-container-tools.x86_64 0:1.9.0-1        libnvidia-container1.x86_64 0:1.9.0-1        nvidia-container-toolkit.x86_64 0:1.9.0-1

Complete!
```

下载 docker 镜像

执行以下命令，下载 docker 镜像。

```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-cvm-0.4.1
```

该镜像包含的软件版本信息如下：

- OS：18.04.5
- python：3.6.9
- cuda toolkits：V11.2.152
- cudnn library：8.1.1
- nccl library：2.8.4
- **tencent-lightcc：3.1.1**
- **HARP library：v1.3**
- ttensorflow：1.15.5

其中：

- LightCC 是腾讯云提供的基于 Horovod 深度定制优化的通信组件，完全兼容 Horovod API，不需要任何业务适配。
- HARP 是腾讯云提供的用户态协议栈，致力于提高 VPC 网络下的分布式训练的通信效率。以动态库的形式提供，官方 NCCL 初始化过程中会自动加载，不需要任何业务适配。
- ttensorflow 是腾讯云基于开源 tensorflow 1.15.5添加了 CUDA 11的支持，同时集成了 TFRA，用来支持动态 embedding 的特性。如需了解更多信息，请参见 TTensorflow 使用说明 。

启动 docker 镜像

执行以下命令，启动 docker 镜像。

```
docker run -it --rm --gpus all --privileged --net=host -v /sys:/sys -v
/dev/hugepages:/dev/hugepages -v /usr/local/tfabric/tools:/usr/local/tfabric/tools
ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-cvm-0.4.1
```

> ⚠ **注意：**
> `/dev/hugepages` 和 `/usr/local/tfabric/tools` 包含了 HARP 运行所需要的大页内存和配置文件。

分布式训练 benchmark 测试

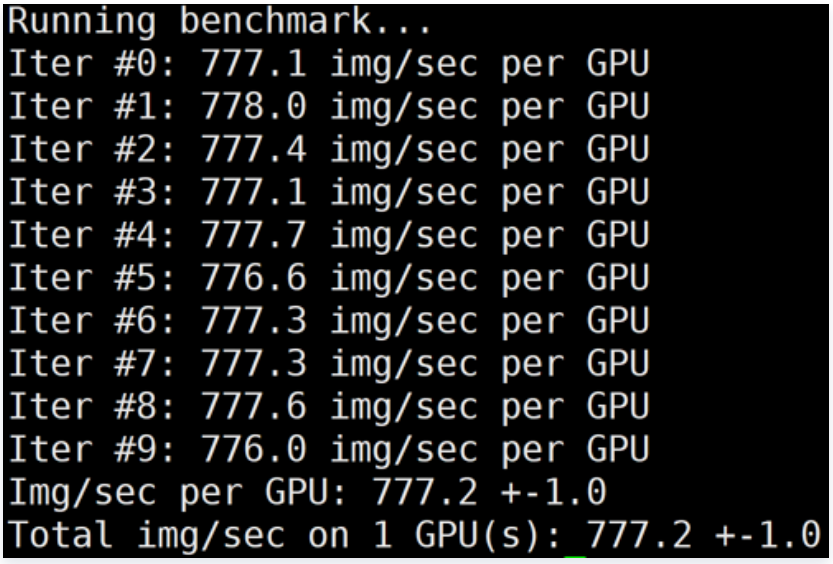
> **说明**
> docker 镜像中的文件 `/mnt/tensorflow_synthetic_benchmark.py` 来自 horovod example 。

单卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 1 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3
/mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

下图为 GT4/A100的单卡 benchmark 结果：

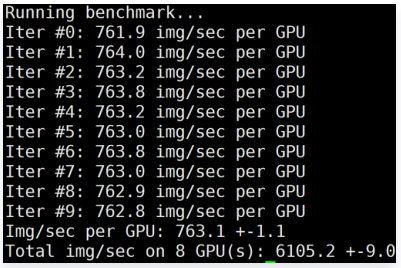


单机多卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 8 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3
/mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

下图为 GT4/A100的单机8卡 benchmark 结果：



多机多卡

1. 参见 购买实例 – 启动 docker 镜像 步骤，购买和配置多台训练机器。
2. 配置多台服务器 docker 间相互免密访问，详情请参见 配置容器 SSH 免密访问 。
3. 执行以下命令，使用 TACO Train 进行多机训练加速。

```
/usr/local/openmpi/bin/mpirun -np 8 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3
/mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

下图为 GT4/A100 的2机16卡 benchmark 结果：

```
Running benchmark...
Iter #0: 700.6 img/sec per GPU
Iter #1: 703.7 img/sec per GPU
Iter #2: 697.4 img/sec per GPU
Iter #3: 695.0 img/sec per GPU
Iter #4: 694.3 img/sec per GPU
Iter #5: 702.8 img/sec per GPU
Iter #6: 701.7 img/sec per GPU
Iter #7: 699.3 img/sec per GPU
Iter #8: 692.1 img/sec per GPU
Iter #9: 696.8 img/sec per GPU
Img/sec per GPU: 698.4 +-7.2
Total img/sec on 16 GPU(s): 11173.9 +-115.8
```

LightCC 的环境变量说明如下表：

环境变量	默认值	说明
LIGHT_2D_ALLREDUCE	0	是否使用2D-Allreduce 算法
LIGHT_INTRA_SIZE	8	2D-Allreduce 组内 GPU 数
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	2D-Allreduce 的阈值，单位是字节，小于等于该阈值的数据才使用2D-Allreduce
LIGHT_TOPK_ALLREDUCE	0	是否使用 TOPK 压缩通信
LIGHT_TOPK_RATIO	0.01	使用 TOPK 压缩的比例
LIGHT_TOPK_THRESHOLD	1048576	TOPK 压缩的阈值，单位是字节，大于等于该阈值的数据才使用 TOPK 压缩通信
LIGHT_TOPK_FP16	0	压缩通信的 value 是否转成 FP16

4. 执行以下命令，关闭 TACO LightCC 加速进行测试。

```
# 修改环境变量，使用Horovod进行多机Allreduce
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_ALGO=RING -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -
mca btl_tcp_if_include eth0 python3 /mnt/tensorflow_synthetic_benchmark.py --
model=ResNet50 --batch-size=256
```

下图为 GT4/A100的2机16卡，关闭 LightCC 之后的 benchmark 结果：

```
Running benchmark...
Iter #0: 494.7 img/sec per GPU
Iter #1: 496.1 img/sec per GPU
Iter #2: 492.7 img/sec per GPU
Iter #3: 490.2 img/sec per GPU
Iter #4: 488.7 img/sec per GPU
Iter #5: 487.8 img/sec per GPU
Iter #6: 494.6 img/sec per GPU
Iter #7: 487.3 img/sec per GPU
Iter #8: 488.9 img/sec per GPU
Iter #9: 490.1 img/sec per GPU
Img/sec per GPU: 491.1 +-5.9
Total img/sec on 16 GPU(s): 7857.7 +-94.1
```

5. 执行以下命令，同时关闭 LightCC 和 HARP 加速进行测试。

```
# 将HARP加速库rename为bak.libnccl-net.so即可关闭HARP加速。
/usr/local/openmpi/bin/mpirun -np 2 -H gpu1:1,gpu2:1 --allow-run-as-root -bind-to
none -map-by slot mv /usr/lib/x86_64-linux-gnu/libnccl-net.so /usr/lib/x86_64-linux-
gnu/bak.libnccl-net.so

# 修改环境变量，使用Horovod进行多机Allreduce
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_ALGO=RING -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -
mca btl_tcp_if_include eth0 python3 /mnt/tensorflow_synthetic_benchmark.py --
model=ResNet50 --batch-size=256
```

下图为 GT4/A100的2机16卡，同时关闭 LightCC 和 HARP 之后的 benchmark 结果：

```
Running benchmark...
Iter #0: 328.6 img/sec per GPU
Iter #1: 338.9 img/sec per GPU
Iter #2: 335.0 img/sec per GPU
Iter #3: 351.9 img/sec per GPU
Iter #4: 356.1 img/sec per GPU
Iter #5: 344.0 img/sec per GPU
Iter #6: 355.1 img/sec per GPU
Iter #7: 348.2 img/sec per GPU
Iter #8: 336.6 img/sec per GPU
Iter #9: 345.7 img/sec per GPU
Img/sec per GPU: 344.0 +-17.0
Total img/sec on 16 GPU(s): 5504.3 +-271.7
```

> **注意：**
> 测试完如需恢复 HARP 加速能力，只需要把所有机器上的 bak.libnccl-net.so 重新命名为 libnccl-net.so 即可。

总结

本文测试数据如下：

机器：GT4（A100 * 8）+ 50G VPC
容器：ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-cvm-0.4.1
网络模型：ResNet50Batch：256
数据：synthetic data

机型	#GPUs	Horovod+TCP		Horovod+HARP		LightCC+HARP	
		性能（img/sec）	线性加速比	性能（img/sec）	线性加速比	性能（img/sec）	线性加速比
GT4/A100	1	777	-	777	-	777	-
	8	6105	98.21%	6105	98.21%	6105	98.21%
	16	5504	44.27%	7857	63.20%	11173	89.87%

说明如下：

- **对于 GT4，相比开源方案，使用 TACO 分布式训练加速组件之后，16卡A100的线性加速比从44.27%提升到89.87%，效果非常显著。**
- **LightCC 和 HARP 只在多机分布式训练当中才有加速效果，单机8卡场景由于 NVLink 的高速带宽存在，一般不需要额外的加速就能达到比较高的线性加速比。**

- 上述 benchmark 脚本也支持除 ResNet50之外的其他模型，ModelName 请参见 Keras Applications 。
- 上述 docker 镜像仅用于 demo，若您具备开发或者部署环境，请提供 OS/python/CUDA/tensorflow 版本信息，并联系腾讯云售后提供特定版本的 TACO 加速组件。

在裸金属服务器上部署 TensorFlow 分布式训练集群

最近更新时间：2024-11-25 15:16:52

操作场景

本文介绍如何基于裸金属服务器搭建 Tensorflow+Taco Train 分布式训练集群。

操作步骤

购买实例

购买实例，其中实例、存储及镜像请参考以下信息选择，其余配置请参见 通过购买页创建实例 按需选择。

- **实例**：选择 GPU 型 HCCG5v 、GPU 型 HCCG5vm 或 GPU 型 HCCPNV4h 。
- **系统盘**：配置容量不小于50GB的云硬盘。您也可在创建实例后使用文件存储，详情参见 在 Linux 客户端上使用 CFS 文件系统 。
- **镜像**：建议选择**公共镜像**，公共镜像当中已安装 RDMA 网卡驱动，且支持自动安装 GPU 驱动。若您选择**自定义镜像**，则需要自行安装 RDMA 网卡驱动和 GPU 驱动，请通过 联系我们 获取腾讯云售后支持。
- 操作系统请使用 CentOS 7.6。
- 若您选择**公共镜像**，则请勾选“后台自动安装GPU驱动”，实例将在系统启动后预装对应版本驱动。如下图所示：

安装 nv_peer_mem（可选）

多机通信的过程中，GPU 显存中的数据需要首先拷贝到内存中，然后通过网卡发出。通过 GPU Direct RDMA 协议，可利用 GPU 和网卡直接通过 PCIe 进行 Peer2Peer 的数据交换这条更快速的路径，无需借助内存来进行数据的传递。
如需使用 GDR 进行数据传输，请在实例中执行以下命令，安装如下驱动。

```
git clone https://github.com/Mellanox/nv_peer_memory.git
cd ./nv_peer_memory/ && git checkout 1.0-9
make && insmod ./nv_peer_mem.ko
// 如果服务器发生了重启，nv_peer_mem驱动需要重新insmod
```

安装 docker 和 nvidia docker

1. 参见 使用标准登录方式登录 Linux 实例，登录实例。
2. 执行以下命令，安装 docker。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-docker.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参考 Docker 官方文档 Install Docker Engine 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
+ sh -c 'yum install -y -q docker-ce-rootless-extras'
Package docker-ce-rootless-extras-20.10.16-3.el7.x86_64 already installed and latest version

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

3. 执行以下命令，安装 nvidia-docker2。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参考 NVIDIA 官方文档 Installation Guide & mdash 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
Running transaction
  Installing : libnvidia-container1-1.9.0-1.x86_64
  Installing : libnvidia-container-tools-1.9.0-1.x86_64
  Installing : nvidia-container-toolkit-1.9.0-1.x86_64
  Installing : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container-tools-1.9.0-1.x86_64
  Verifying  : nvidia-container-toolkit-1.9.0-1.x86_64
  Verifying  : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container1-1.9.0-1.x86_64

Installed:
  nvidia-docker2.noarch 0:2.10.0-1

Dependency Installed:
  libnvidia-container-tools.x86_64 0:1.9.0-1        libnvidia-container1.x86_64 0:1.9.0-1

Complete!
```

下载 docker 镜像

执行以下命令，下载 docker 镜像。

```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-bm-0.4.2
```

该镜像包含的软件版本信息如下：

- OS：18.04.5
- ofed: MLNX_OFED_LINUX-5.1-2.5.8.0
- python：3.6.9
- cuda toolkits：V11.2.152
- cudnn library：8.1.1
- nccl library：2.8.4
- **tencent-lightcc：3.1.1**
- ttensorflow：1.15.5

其中：

- LightCC 是腾讯云提供的基于 Horovod 深度定制优化的通信组件，完全兼容 Horovod API，不需要任何业务适配。
- ttensorflow 是腾讯云基于开源 tensorflow 1.15.5添加了 CUDA 11的支持，同时集成了 TFRA，用来支持动态 embedding 的特性。如需了解更多信息，请参见 产品概述 。

启动 docker 镜像

执行以下命令，启动 docker 镜像。

```
docker run -itd --rm --gpus all --shm-size=32g --ulimit memlock=-1 --ulimit stack=67108864
--net=host --privileged ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-bm-0.4.2
```

注意：

`--privileged` 选项使容器能够访问主机上的 RDMA 设备。

分布式训练 benchmark 测试

说明：

docker 镜像中的文件 `/mnt/tensorflow_synthetic_benchmark.py` 来自 horovod example 。

单卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 1 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x NCCL_SOCKET_IFNAME=bond0 -x
NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x LD_LIBRARY_PATH -x PATH -mca pml ob1
-mca btl_tcp_if_include bond0 -mca btl ^openib python3
/mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

下图为 HCCPNV4h/A100的单卡 benchmark 结果：

```
Running benchmark...
Iter #0: 778.4 img/sec per GPU
Iter #1: 778.5 img/sec per GPU
Iter #2: 778.3 img/sec per GPU
Iter #3: 778.4 img/sec per GPU
Iter #4: 778.3 img/sec per GPU
Iter #5: 778.5 img/sec per GPU
Iter #6: 778.5 img/sec per GPU
Iter #7: 778.5 img/sec per GPU
Iter #8: 778.4 img/sec per GPU
Iter #9: 778.6 img/sec per GPU
Img/sec per GPU: 778.4 +-0.2
Total img/sec on 1 GPU(s): 778.4 +-0.2
```

单机多卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 8 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x NCCL_SOCKET_IFNAME=bond0 -x
NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x LD_LIBRARY_PATH -x PATH -mca pml ob1
-mca btl_tcp_if_include bond0 -mca btl ^openib python3
/mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

下图为 HCCPNV4h/A100的8卡 benchmark 结果：

```
Running benchmark...
Iter #0: 763.6 img/sec per GPU
Iter #1: 761.9 img/sec per GPU
Iter #2: 763.5 img/sec per GPU
Iter #3: 763.3 img/sec per GPU
Iter #4: 761.4 img/sec per GPU
Iter #5: 764.0 img/sec per GPU
Iter #6: 763.7 img/sec per GPU
Iter #7: 763.5 img/sec per GPU
Iter #8: 762.9 img/sec per GPU
Iter #9: 762.7 img/sec per GPU
Img/sec per GPU: 763.0 +-1.6
Total img/sec on 8 GPU(s): 6104.4 +-12.6
```

多机多卡

1. 参见 购买实例 – 启动 docker 镜像 步骤，购买和配置多台训练机器。
2. 配置多台服务器 docker 间相互免密访问，详情请参见 配置容器 SSH 免密访问 。
3. 执行以下命令，使用 TACO Train 进行多机训练加速。

```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-
to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x
NCCL_SOCKET_IFNAME=bond0 -x NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x
```

```
HOROVOD_FUSION_THRESHOLD=0  -x HOROVOD_CYCLE_TIME=0 -x LIGHT_INTRA_SIZE=8 -x
LIGHT_2D_ALLREDUCE=1 -x LIGHT_TOPK_ALLREDUCE=1 -x LIGHT_TOPK_THRESHOLD=2097152 -x
LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl_tcp_if_include bond0 -mca btl
^openib python3 /mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-
size=256
```

下图为 HCCPNV4h/A100 2机16卡 benchmark 结果：

```
Running benchmark...
Iter #0: 736.0 img/sec per GPU
Iter #1: 735.0 img/sec per GPU
Iter #2: 735.3 img/sec per GPU
Iter #3: 736.9 img/sec per GPU
Iter #4: 739.5 img/sec per GPU
Iter #5: 737.5 img/sec per GPU
Iter #6: 735.6 img/sec per GPU
Iter #7: 737.1 img/sec per GPU
Iter #8: 733.9 img/sec per GPU
Iter #9: 735.2 img/sec per GPU
Img/sec per GPU: 736.2 +-3.0
Total img/sec on 16 GPU(s): 11779.1 +-47.6
```

LightCC 的环境变量说明如下表：

环境变量	默认值	说明
LIGHT_2D_ALLREDUCE	0	是否使用2D-Allreduce 算法
LIGHT_INTRA_SIZE	8	2D-Allreduce 组内 GPU 数
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	2D-Allreduce 的阈值，单位是字节，小于等于该阈值的数据才使用2D-Allreduce
LIGHT_TOPK_ALLREDUCE	0	是否使用 TOPK 压缩通信
LIGHT_TOPK_RATIO	0.01	使用 TOPK 压缩的比例
LIGHT_TOPK_THRESHOLD	1048576	TOPK 压缩的阈值，单位是字节，大于等于该阈值的数据才使用 TOPK 压缩通信
LIGHT_TOPK_FP16	0	压缩通信的 value 是否转成 FP16

4. 执行以下命令，关闭 TACO LightCC 加速进行测试。

```
# 去掉LIGHT_xx的环境变量，即可使用Horovod进行多机Allreduce
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-
to none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x
NCCL_SOCKET_IFNAME=bond0 -x NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x
LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl_tcp_if_include bond0 -mca btl
^openib python3 /mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-
size=256
```

下图为 HCCPNV4h/A100 2机16卡，关闭 LightCC 之后的 benchmark 结果：

```
Running benchmark...
Iter #0: 673.6 img/sec per GPU
Iter #1: 678.8 img/sec per GPU
Iter #2: 669.6 img/sec per GPU
Iter #3: 677.4 img/sec per GPU
Iter #4: 671.3 img/sec per GPU
Iter #5: 672.9 img/sec per GPU
Iter #6: 676.1 img/sec per GPU
Iter #7: 680.1 img/sec per GPU
Iter #8: 674.2 img/sec per GPU
Iter #9: 670.0 img/sec per GPU
Img/sec per GPU: 674.4 +-6.8
Total img/sec on 16 GPU(s): 10790.4 +-108.2
```

多机多卡 GDR

执行以下命令，使用 GDR 进行测试。

> ⚠ **注意：**
> 使用 GDR 需安装 nv_peer_mem，详情请参见 安装 nv_peer_mem 。

```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x
NCCL_SOCKET_IFNAME=bond0 -x NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=2 -x
HOROVOD_FUSION_THRESHOLD=0  -x HOROVOD_CYCLE_TIME=0 -x LIGHT_INTRA_SIZE=8 -x
LIGHT_2D_ALLREDUCE=1 -x LIGHT_TOPK_ALLREDUCE=1 -x LIGHT_TOPK_THRESHOLD=2097152 -x
LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl_tcp_if_include bond0 -mca btl ^openib
python3 /mnt/tensorflow_synthetic_benchmark.py --model=ResNet50 --batch-size=256
```

GDR 通常在大模型或者集群规模较大时有显著的加速效果，测试结果如下图所示：

```
Running benchmark...
Iter #0: 733.5 img/sec per GPU
Iter #1: 731.3 img/sec per GPU
Iter #2: 735.2 img/sec per GPU
Iter #3: 734.1 img/sec per GPU
Iter #4: 733.7 img/sec per GPU
Iter #5: 734.8 img/sec per GPU
Iter #6: 734.5 img/sec per GPU
Iter #7: 736.3 img/sec per GPU
Iter #8: 733.8 img/sec per GPU
Iter #9: 733.7 img/sec per GPU
Img/sec per GPU: 734.1 +-2.4
Total img/sec on 16 GPU(s): 11745.3 +-38.6
```

总结

本文使用环境及测试数据如下：

机器：HCCPNV4h（A100 * 8）+ 100G RDMA + 25G VPC
容器：ccr.ccs.tencentyun.com/qcloud/taco-train:ttf115-cu112-bm-0.4.2
网络模型：ResNet50Batch：256
数据：synthetic data

机型	#GPUs	Horovod+RDMA		LightCC+RDMA	
		性能（img/sec）	线性加速比	性能（img/sec）	线性加速比
HCCPNV4h A100	1	778	-	778	-
	8	6104	98.07%	6104	98.07%
	16	10790	86.68%	11779	94.63%

说明如下：

- **对于 HCCPNV4h/A100，相比开源方案，2机16卡通过 LightCC 可以将线性加速比从86.68%提升到94.63%。**
- **上述 benchmark 也支持除 ResNet50之外的其他模型，ModelName 请参见 Keras Applications 。**
- **上述 docker 镜像仅用于 demo，若您需使用自己的 docker 开发环境，请参见 容器安装用户态 RDMA 驱动 安装网卡驱动。**
- **如需特定 OS/python/CUDA/tensorflow 版本的 LightCC 加速组件，请通过 联系我们 联系腾讯云售后获取。**

在裸金属服务器上部署 PyTorch 分布式训练集群

最近更新时间：2024-11-25 15:16:52

操作场景

本文介绍如何基于裸金属服务器搭建 torch+Taco Train 分布式训练集群。

操作步骤

购买实例

购买实例，其中实例、存储及镜像请参考以下信息选择，其余配置请参见 通过购买页创建实例 按需选择。

- **实例**：选择 GPU 型 HCCG5v 、GPU 型 HCCG5vm 或 GPU 型 HCCPNV4h 。
- **系统盘**：配置容量不小于50GB的云硬盘。您也可在创建实例后使用文件存储，详情参见 在 Linux 客户端上使用 CFS 文件系统 。
- **镜像**：建议选择**公共镜像**，公共镜像当中已安装 RDMA 网卡驱动，且支持自动安装 GPU 驱动。若您选择**自定义镜像**，则需要自行安装 RDMA 网卡驱动和 GPU 驱动，请通过 联系我们 获取腾讯云售后支持。
- 操作系统请使用 CentOS 7.6。
- 若您选择**公共镜像**，则请勾选“后台自动安装GPU驱动”，实例将在系统启动后预装对应版本驱动。如下图所示：

安装 nv_peer_mem（可选）

多机通信的过程中，GPU 显存中的数据需要首先拷贝到内存中，然后通过网卡发出。通过 GPU Direct RDMA 协议，可利用 GPU 和网卡直接通过 PCIe 进行 Peer2Peer 的数据交换这条更快速的路径，无需借助内存来进行数据的传递。
如需使用 GDR 进行数据传输，请在实例中执行以下命令，安装如下驱动。

```
git clone https://github.com/Mellanox/nv_peer_memory.git
cd ./nv_peer_memory/ && git checkout 1.0-9
make && insmod ./nv_peer_mem.ko
// 如果服务器发生了重启，nv_peer_mem驱动需要重新insmod
```

安装 docker 和 nvidia docker

1. 参见 使用标准登录方式登录 Linux 实例，登录实例。
2. 执行以下命令，安装 docker。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-docker.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参见 Docker 官方文档 Install Docker Engine 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
+ sh -c 'yum install -y -q docker-ce-rootless-extras'
Package docker-ce-rootless-extras-20.10.16-3.el7.x86_64 already installed and latest version

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

3. 执行以下命令，安装 nvidia-docker2。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参见 NVIDIA 官方文档 Installation Guide & mdash 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
Running transaction
  Installing : libnvidia-container1-1.9.0-1.x86_64
  Installing : libnvidia-container-tools-1.9.0-1.x86_64
  Installing : nvidia-container-toolkit-1.9.0-1.x86_64
  Installing : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container-tools-1.9.0-1.x86_64
  Verifying  : nvidia-container-toolkit-1.9.0-1.x86_64
  Verifying  : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container1-1.9.0-1.x86_64

Installed:
  nvidia-docker2.noarch 0:2.10.0-1

Dependency Installed:
  libnvidia-container-tools.x86_64 0:1.9.0-1       libnvidia-container1.x86_64 0:1.9.0-1       nvidia-container-toolkit.x86_64 0:1.9.0-1

Complete!
```

下载 docker 镜像

执行以下命令，下载 docker 镜像。

```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-bm-0.4.1
```

该镜像包含的软件版本信息如下：

- OS：Ubuntu 20.04.4 LTS
- ofed: MLNX_OFED_LINUX-5.4-3.1.0.0
- python：3.8.10
- cuda toolkits：V11.3.109
- cudnn library：8.2.0
- nccl library：2.9.9
- **tencent-lightcc：3.1.1**
- **torch：1.11.0+cu113**

其中：

- LightCC 是腾讯云提供的基于 Horovod 深度定制优化的通信组件，完全兼容 Horovod API，不需要任何业务适配。
- torch 为官方版本。

启动 docker 镜像

执行以下命令，启动 docker 镜像。

```
docker run -itd --rm --gpus all --shm-size=32g --ulimit memlock=-1 --ulimit stack=67108864
--net=host --privileged ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-bm-0.4.1
```

> **注意：**
> `--privileged` 选项使容器能够访问主机上的 RDMA 设备。

分布式训练 benchmark 测试

> **说明：**
> docker 镜像中的文件 `/mnt/pytorch_synthetic_benchmark.py` 来自 horovod example 。

单卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 1 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH -mca
btl_tcp_if_include eth0 python3 /mnt/pytorch_synthetic_benchmark.py --model=vgg16 --
batch-size=128
```

下图为 GN10Xp/V100的单卡 benchmark 结果：

```
Model: resnet50
Batch size: 256
Number of GPUs: 1
Running warmup...
Running benchmark...
Iter #0: 819.7 img/sec per GPU
Iter #1: 819.6 img/sec per GPU
Iter #2: 819.6 img/sec per GPU
Iter #3: 819.6 img/sec per GPU
Iter #4: 819.6 img/sec per GPU
Iter #5: 819.6 img/sec per GPU
Iter #6: 819.6 img/sec per GPU
Iter #7: 819.6 img/sec per GPU
Iter #8: 819.5 img/sec per GPU
Iter #9: 819.7 img/sec per GPU
Img/sec per GPU: 819.6 +-0.1
Total img/sec on 1 GPU(s): 819.6 +-0.1
```

单机多卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 8 --allow-run-as-root -bind-to none -map-by slot -
x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x NCCL_SOCKET_IFNAME=bond0 -x
NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x LD_LIBRARY_PATH -x PATH -mca pml ob1
-mca btl_tcp_if_include bond0 -mca btl ^openib python3
/mnt/pytorch_synthetic_benchmark.py --model resnet50 --batch-size=256
```

下图为 HCCPNV4h/A100的8卡 benchmark 结果：

```
Model: resnet50
Batch size: 256
Number of GPUs: 8
Running warmup...
Running benchmark...
Iter #0: 808.6 img/sec per GPU
Iter #1: 808.8 img/sec per GPU
Iter #2: 808.9 img/sec per GPU
Iter #3: 809.1 img/sec per GPU
Iter #4: 808.7 img/sec per GPU
Iter #5: 808.4 img/sec per GPU
Iter #6: 809.2 img/sec per GPU
Iter #7: 808.0 img/sec per GPU
Iter #8: 809.1 img/sec per GPU
Iter #9: 808.5 img/sec per GPU
Img/sec per GPU: 808.7 +-0.7
Total img/sec on 8 GPU(s): 6469.8 +-5.2
```

多机多卡

1. 参见 购买实例 – 启动 docker 镜像 步骤，购买和配置多台训练机器。
2. 配置多台服务器 docker 间相互免密访问，详情请参见 配置容器 SSH 免密访问 。
3. 执行以下命令，使用 TACO Train 进行多机训练加速。

```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x
NCCL_SOCKET_IFNAME=bond0 -x NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x
HOROVOD_FUSION_THRESHOLD=0  -x HOROVOD_CYCLE_TIME=0 -x LIGHT_INTRA_SIZE=8 -x
LIGHT_2D_ALLREDUCE=1 -x LIGHT_TOPK_ALLREDUCE=1 -x LIGHT_TOPK_THRESHOLD=2097152 -x
LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl_tcp_if_include bond0 -mca btl ^openib
python3 /mnt/pytorch_synthetic_benchmark.py --model resnet50 --batch-size=256
```

下图为 HCCPNV4h/A100 2机16卡 benchmark 结果：

```
Running benchmark...
Iter #0: 781.5 img/sec per GPU
Iter #1: 784.1 img/sec per GPU
Iter #2: 782.9 img/sec per GPU
Iter #3: 782.8 img/sec per GPU
Iter #4: 783.4 img/sec per GPU
Iter #5: 784.3 img/sec per GPU
Iter #6: 783.2 img/sec per GPU
Iter #7: 782.9 img/sec per GPU
Iter #8: 783.9 img/sec per GPU
Iter #9: 783.6 img/sec per GPU
Img/sec per GPU: 783.3 +-1.5
Total img/sec on 16 GPU(s): 12532.5 +-24.4
```

LightCC 的环境变量说明如下表：

环境变量	默认值	说明
LIGHT_2D_ALLREDUCE	0	是否使用2D-Allreduce 算法
LIGHT_INTRA_SIZE	8	2D-Allreduce 组内 GPU 数
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	2D-Allreduce 的阈值，单位是字节，小于等于该阈值的数据才使用2D-Allreduce
LIGHT_TOPK_ALLREDUCE	0	是否使用 TOPK 压缩通信
LIGHT_TOPK_RATIO	0.01	使用 TOPK 压缩的比例
LIGHT_TOPK_THRESHOLD	1048576	TOPK 压缩的阈值，单位是字节，大于等于该阈值的数据才使用 TOPK 压缩通信
LIGHT_TOPK_FP16	0	压缩通信的 value 是否转成 FP16

4. 执行以下命令，关闭 TACO LightCC 加速进行测试。

```
# 去掉LIGHT_xx的环境变量，即可使用Horovod进行多机Allreduce/usr/local/openmpi/bin/mpirun -
np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to none -map-by slot -x
```

```
NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x NCCL_SOCKET_IFNAME=bond0 -x
NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=0 -x LD_LIBRARY_PATH -x PATH -mca pml ob1
-mca btl_tcp_if_include bond0 -mca btl ^openib python3
/mnt/pytorch_synthetic_benchmark.py --model resnet50 --batch-size=256
```

下图为 HCCPNV4h/A100 2机16卡，关闭 LightCC 之后的 benchmark 结果：

```
Model: resnet50
Batch size: 256
Number of GPUs: 16
Running warmup...
Running benchmark...
Iter #0: 772.3 img/sec per GPU
Iter #1: 766.2 img/sec per GPU
Iter #2: 769.4 img/sec per GPU
Iter #3: 766.5 img/sec per GPU
Iter #4: 767.0 img/sec per GPU
Iter #5: 767.3 img/sec per GPU
Iter #6: 768.9 img/sec per GPU
Iter #7: 772.0 img/sec per GPU
Iter #8: 770.6 img/sec per GPU
Iter #9: 767.2 img/sec per GPU
Img/sec per GPU: 768.7 +-4.2
Total img/sec on 16 GPU(s): 12299.7 +-67.6
```

多机多卡 GDR

执行以下命令，使用 GDR 进行测试。

> ⚠ **注意：**
> 使用 GDR 需安装 nv_peer_mem，详情请参见 安装 nv_peer_mem 。

```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_IB_DISABLE=0 -x
NCCL_SOCKET_IFNAME=bond0 -x NCCL_IB_GID_INDEX=3 -x NCCL_NET_GDR_LEVEL=2 -x
HOROVOD_FUSION_THRESHOLD=0  -x HOROVOD_CYCLE_TIME=0 -x LIGHT_INTRA_SIZE=8 -x
LIGHT_2D_ALLREDUCE=1 -x LIGHT_TOPK_ALLREDUCE=1 -x LIGHT_TOPK_THRESHOLD=2097152 -x
LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl_tcp_if_include bond0 -mca btl ^openib
python3 /mnt/pytorch_synthetic_benchmark.py --model resnet50 --batch-size=256
```

GDR 通常在大模型或者集群规模较大时有显著的加速效果，测试结果如下图所示：

```
Running benchmark...
Iter #0: 782.0 img/sec per GPU
Iter #1: 781.1 img/sec per GPU
Iter #2: 782.0 img/sec per GPU
Iter #3: 783.1 img/sec per GPU
Iter #4: 782.1 img/sec per GPU
Iter #5: 780.7 img/sec per GPU
Iter #6: 781.6 img/sec per GPU
Iter #7: 780.4 img/sec per GPU
Iter #8: 783.0 img/sec per GPU
Iter #9: 784.3 img/sec per GPU
Img/sec per GPU: 782.0 +-2.2
Total img/sec on 16 GPU(s): 12512.6 +-35.7
```

总结

本文使用环境及测试数据如下：

机器：HCCPNV4h（A100 * 8）+ 100G RDMA + 25G VPC
容器：ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-bm-0.4.1
网络模型：ResNet50Batch：256
数据：synthetic data

机型	#GPUs	Horovod+RDMA		LightCC+RDMA	
		性能（img/sec）	线性加速比	性能（img/sec）	线性加速比
HCCPNV4h A100	1	819	-	819	-
	8	6469	98.73%	6469	98.73%
	16	12299	93.85%	12532	95.63%

说明如下：

- 对于 HCCPNV4h/A100，相比开源方案，2机16卡通过 LightCC 可将线性加速比从93.85%提升到95.63%。
- 上述 benchmark 也支持除 ResNet50之外的其他模型，ModelName 请参见 Keras Applications 。
- 上述 docker 镜像仅用于 demo，若您需使用自己的 docker 开发环境，请参见 容器安装用户态 RDMA 驱动 安装网卡驱动。
- 如需特定 OS/python/CUDA/tensorflow 版本的 LightCC 加速组件，请联系腾讯云售后获取。

组件配置和使用
TCCL 使用说明

最近更新时间：2024-10-29 11:31:42

操作场景

本文介绍如何在腾讯云环境中配置 TCCL 加速通信库，实现您在腾讯云 RDMA 环境中多机多卡通信的性能提升。在大模型训练场景，对比开源的 NCCL 方案，TCCL 预计约可以提升 50% 带宽利用率。

操作步骤

准备环境

1、创建 GPU 型 HCCPNV4sne 或 GPU 型 HCCPNV4sn 高性能计算集群实例，分别支持 1.6Tbps 和 800Gbps RDMA 网络。
2、为 GPU 型实例 安装 GPU 驱动 和 nvidia-fabricmanager 服务 。

> **注意：**
> TCCL 运行软件环境要求 glibc 版本 2.17 以上，CUDA 版本 10.0 以上。

选择安装方式

TCCL目前支持使用三种方式安装，您可以根据需要选择适合业务场景的安装方式。

- TCCL 通信库 + 编译安装 Pytorch
- TCCL 通信库 + Pytorch 通信插件
- NCCL 插件 + 排序的 IP 列表

> **说明：**
> 由于当前大模型训练基本都基于 Pytorch 框架，所以主要以 Pytorch 为例进行说明，

TCCL的三种接入方案对比如下表：

安装方式	方法一：编译安装 Pytorch	方法二：安装 Pytorch 通信插件	（推荐）方法三：安装 NCCL 通信插件
使用步骤	• 安装 TCCL • 重新编译安装 Pytorch	• 安装 Pytorch 通信插件 • 修改分布式通信后端	• 安装 NCCL 插件 • 修改启动脚本
优点	对业务代码无入侵	安装方便	安装方便
缺点	需要重新编译安装 Pytorch 对软件环境有要求	需要修改业务代码 对软件环境有要求	集群节点扩充之后，需要更新排序列表
软件环境依赖	对应 NCCL 版本 2.12 要求 glibc 版本 2.17 以上 要求 CUDA 版本 10.0 以上	当前安装包仅支持 Pytorch 1.12 要求 glibc 版本 2.17 以上 要求 CUDA 版本 10.0 以上	安装 NCCL 即可

如果您的机器资源和模型训练场景相对比较固定，推荐使用方法三，兼容不同的 NCCL 版本和 CUDA 版本，安装使用方便，不需要修改业务代码或者重新编译 Pytorch。

如果您的资源需要提供给不同的业务团队，或者经常有扩容的需求，推荐使用前两种方法，不需要算法人员或者调度框架刻意去感知机器的网络拓扑信息。
如果您不希望对业务代码做适配，那么可以使用方法1，只需要重新编译 Pytorch 框架。

配置 TCCL 环境并验证

方法一： 编译安装 Pytorch

由于社区 Pytorch 默认采用静态方式连接 NCCL 通信库，所以无法通过替换共享库的方式使用 TCCL。

1、安装 TCCL

以 Ubuntu 20.04 为例，您可以使用以下命令安装，安装之后 TCCL 位于 `/opt/tencent/tccl` 目录。

```
# 卸载已有tccl版本和nccl插件
dpkg -r tccl && dpkg -r nccl-rdma-sharp-plugins

# 下载安装tccl v1.5版本
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/tccl/TCCL_1.5-
ubuntu.20.04.5_amd64.deb && dpkg -i TCCL_1.5-ubuntu.20.04.5_amd64.deb && rm -f
TCCL_1.5-ubuntu.20.04.5_amd64.deb
```

如果您使用 CentOS 或 TencentOS，参考以下步骤安装：

```
# 卸载已有tccl版本和nccl插件
rpm -e tccl && rpm -e nccl-rdma-sharp-plugins-1.0-1.x86_64

# 下载tccl v1.5版本
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/tccl/tccl-1.5-
1.tl2.x86_64.rpm && rpm -ivh --nodeps --force tccl-1.5-1.tl2.x86_64.rpm && rm -f tccl-
1.5-1.tl2.x86_64.rpm
```

2、重新编译安装 Pytorch

以下为 Pytorch 源码安装示例，详情请参见 官网 Pytorch 安装说明 。

```
#!/bin/bash

# 卸载当前版本
pip uninstall -y torch

# 下载pytorch源码
git clone --recursive https://github.com/pytorch/pytorch
cd pytorch

# <!重要> 配置TCCL的安装路径
export USE_SYSTEM_NCCL=1
export NCCL_INCLUDE_DIR="/opt/tencent/tccl/include"
export NCCL_LIB_DIR="/opt/tencent/tccl/lib"

# 参考官网添加其他编译选项
```

```
# 安装开发环境
python setup.py develop
```

3、配置 TCCL 环境变量

```
export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=eth0
export NCCL_IB_GID_INDEX=3
export NCCL_IB_DISABLE=0
export
NCCL_IB_HCA=mlx5_bond_0,mlx5_bond_1,mlx5_bond_2,mlx5_bond_3,mlx5_bond_4,mlx5_bond_5,mlx
5_bond_6,mlx5_bond_7
export NCCL_NET_GDR_LEVEL=2
export NCCL_IB_QPS_PER_CONNECTION=4
export NCCL_IB_TC=160
export NCCL_IB_TIMEOUT=22
export NCCL_PXN_DISABLE=0
export TCCL_TOPO_AFFINITY=4
```

> **注意：**
> 需要通过 `TCCL_TOPO_AFFINITY=4` 开启网络拓扑感知特性。

4、验证 Pytorch

运行单机多卡或者多机多卡训练过程中打印出如下信息（ `export NCCL_DEBUG=INFO` ）：

```
vm-3-17-centos:74350:74350 [0] NCCL INFO Bootstrap : Using eth0:10.100.3.17<0>
vm-3-17-centos:74350:74350 [0] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
vm-3-17-centos:74350:74350 [0] NCCL INFO NET/IB : Using [0]mlx5_bond_0:1/RoCE [RO]; OOB eth0:10.100.3.17<0>
vm-3-17-centos:74350:74350 [0] NCCL INFO Using network IB
NCCL version 2.12.12_TCCL_v1.5+cuda11.6
vm-3-17-centos:74352:74352 [2] NCCL INFO Bootstrap : Using eth0:10.100.3.17<0>
vm-3-17-centos:74352:74352 [2] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
vm-3-17-centos:74352:74352 [2] NCCL INFO NET/IB : Using [0]mlx5_bond_0:1/RoCE [RO]; OOB eth0:10.100.3.17<0>
vm-3-17-centos:74352:74352 [2] NCCL INFO Using network IB
```

5、验证 nccl-tests

运行 nccl-tests 之前需要 export 对应的 TCCL 路径：

```
export LD_LIBRARY_PATH=/opt/tencent/tccl/lib:$LD_LIBRARY_PATH
```

6、软件版本支持

目前 TCCL 对应 NCCL 版本 2.12 ，要求 glibc 版本 2.17 以上，CUDA 版本 10.0 以上。其他 CUDA 版本支持请联系您的售前经理获取支持。

方法二：安装 Pytorch 通信插件

Pytorch支持通过插件的方式接入第三方通信后端，所以在不重新编译 Pytorch 的前提下，用户可以使用 TCCL 通信后端，API 与 NCCL 完全兼容。详情可参考 Pytorch 现有通信后端介绍 。

1、安装 Pytorch 通信插件

```
# 卸载现有的tccl和NCCL插件
dpkg -r tccl && dpkg -r nccl-rdma-sharp-plugins

# 卸载torch_tccl
pip uninstall -y torch-tccl

# 安装torch_tccl 0.0.2版本
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/tccl/torch_tccl-0.0.2_pt1.12-
py3-none-any.whl && pip install torch_tccl-0.0.2_pt1.12-py3-none-any.whl && rm -f
torch_tccl-0.0.2_pt1.12-py3-none-any.whl
```

2、修改业务代码

```
import torch_tccl

    #args.dist_backend = "nccl"
    args.dist_backend = "tccl"
    torch.distributed.init_process_group(
        backend=args.dist_backend,
        init_method=args.dist_url,
        world_size=args.world_size, rank=args.rank
    )
```

3、配置 TCCL 环境变量

```
export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=eth0
export NCCL_IB_GID_INDEX=3
export NCCL_IB_DISABLE=0
export
NCCL_IB_HCA=mlx5_bond_0,mlx5_bond_1,mlx5_bond_2,mlx5_bond_3,mlx5_bond_4,mlx5_bond_5,mlx
5_bond_6,mlx5_bond_7
export NCCL_NET_GDR_LEVEL=2
export NCCL_IB_QPS_PER_CONNECTION=4
export NCCL_IB_TC=160
export NCCL_IB_TIMEOUT=22
export NCCL_PXN_DISABLE=0
export TCCL_TOPO_AFFINITY=4
```

> ⚠ **注意：**
> 需要通过 `TCCL_TOPO_AFFINITY=4` 开启网络拓扑感知特性。

4、验证 Pytorch

您在执行分布式训练业务时，出现如下提示可确认通信后端被正确加载。

```
vm-3-17-centos:35915:35915 [0] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
vm-3-17-centos:35915:35915 [0] NCCL INFO NET/IB : Using [0]mlx5_bond_0:1/RoCE [RO]; OOB eth0:10.100.3.17<0>
vm-3-17-centos:35915:35915 [0] NCCL INFO Using network IB
NCCL version 2.12.12_TCCL_v1.5+cuda11.6
vm-3-17-centos:35919:35919 [4] NCCL INFO Bootstrap : Using eth0:10.100.3.17<0>
vm-3-17-centos:35919:35919 [4] NCCL INFO NET/Plugin : No plugin found (libnccl-net.so), using internal implementation
vm-3-17-centos:35921:35921 [6] NCCL INFO Bootstrap : Using eth0:10.100.3.17<0>
vm-3-17-centos:35920:35920 [5] NCCL INFO Bootstrap : Using eth0:10.100.3.17<0>
```

5、软件版本限制

当前安装包仅支持 Pytorch 1.12，其他 Pytorch 和 CUDA 版本支持请联系您的售前经理获取支持。

> **说明：**
> 如果运行 nccl-tests 或者其他需要动态链接通信库的场景，请使用方法一安装 TCCL。

（推荐）方法三：安装 NCCL 插件

如果您已经安装了 NCCL ，也可以使用 NCCL 插件的方式使用 TCCL 加速能力。

1、安装 NCCL 插件

以 Ubuntu 20.04 为例，您可以使用以下命令安装插件。

```
# 卸载现有的tccl和nccl插件
dpkg -r tccl && dpkg -r nccl-rdma-sharp-plugins

# 下载安装nccl 1.2插件
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/nccl/nccl-rdma-sharp-plugins_1.2_amd64.deb && dpkg -i nccl-rdma-sharp-plugins_1.2_amd64.deb

# 请确保集群内使用nccl插件版本一致，以下为nccl 1.0版本下载安装命令，推荐使用稳定性更优的nccl 1.2版本
# wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/nccl/nccl-rdma-sharp-plugins_1.0_amd64.deb && dpkg -i nccl-rdma-sharp-plugins_1.0_amd64.deb && rm -f nccl-rdma-sharp-plugins_1.0_amd64.deb
```

如果您使用 CentOS 或 TencentOS，参考以下步骤安装：

```
# 卸载现有的nccl插件
rpm -e nccl-rdma-sharp-plugins-1.0-1.x86_64

# 下载安装nccl 1.2插件
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/nccl/nccl-rdma-sharp-plugins-1.2-1.x86_64.rpm && rpm -ivh --nodeps --force nccl-rdma-sharp-plugins-1.2-1.x86_64.rpm

# 请确保集群内使用nccl插件版本一致，以下为nccl 1.0版本下载安装命令，推荐使用稳定性更优的nccl 1.2版本
# wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/nccl/nccl-rdma-sharp-plugins-1.0-1.x86_64.rpm && rpm -ivh --nodeps --force nccl-rdma-sharp-plugins-1.0-1.x86_64.rpm && rm -f nccl-rdma-sharp-plugins-1.0-1.x86_64.rpm
```

```
```

2、获取拓扑排序的 IP 列表

NCCL 插件不需要依赖文件可提供 bonding 口动态聚合和全局 hash 路由两种优化。如果需要支持网络拓扑的亲和性感知，用户可以通过排序的 IP 列表来实现。
IP 排序可以按照如下方式完成：

a. 准备 IP 列表文件

VPC IP 地址通过 `ifconfig eth0` 获取，每行1个节点 IP，格式如下：

```
root@VM-125-10-tencentos:/workspace# cat ip_eth0.txt
172.16.177.28
172.16.176.11
172.16.177.25
172.16.177.12
```

b. 执行排序

```
wget https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/tccl/get_rdma_order_by_ip.sh && bash get_rdma_order_by_ip.sh
ip_eth0.txt
```

> ⚠ **注意：**
> - 所有节点都安装了 curl 工具（比如对于 Ubuntu，通过 `apt install curl` 安装）。
> - 执行脚本的节点可以 ssh 免密访问其他所有节点。

c. 查看排序后的 IP 列表文件

```
root@VM-125-10-tencentos:/workspace# cat hostfile.txt
172.16.176.11
172.16.177.12
172.16.177.25
172.16.177.28
```

3、配置 TCCL 环境变量

```
export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=eth0
export NCCL_IB_GID_INDEX=3
export NCCL_IB_DISABLE=0
export
NCCL_IB_HCA=mlx5_bond_0,mlx5_bond_1,mlx5_bond_2,mlx5_bond_3,mlx5_bond_4,mlx5_bond_5,
mlx5_bond_6,mlx5_bond_7
export NCCL_NET_GDR_LEVEL=2
export NCCL_IB_QPS_PER_CONNECTION=4
export NCCL_IB_TC=160
export NCCL_IB_TIMEOUT=22
export NCCL_PXN_DISABLE=0
```

```
# 机器 IP 手动排序之后，就不需要添加如下变量了
# export TCCL_TOPO_AFFINITY=4
```

4、修改启动脚本

您需要在启动分布式训练时修改启动脚本。例如，如果使用 deepspeed launcher 启动训练进程，拿到排序后的 IP 列表之后，将对应的 IP 列表写入 hostfile，再启动训练进程。

```
root@vm-3-17-centos:/workspace/ptm/gpt# cat hostfile
172.16.176.11 slots=8
172.16.177.12 slots=8
172.16.177.25 slots=8
172.16.177.28 slots=8

deepspeed --hostfile ./hostfile --master_addr 172.16.176.11 train.py
```

如果使用 torchrun 启动训练进程，通过 `--node_rank` 指定对应的节点顺序，

```
// on 172.16.176.11
torchrun --nnodes=4 --nproc_per_node=8 --node_rank=0 --master_addr=172.16.176.11
train.py ...
// on 172.16.176.12
torchrun --nnodes=4 --nproc_per_node=8 --node_rank=1 --master_addr=172.16.176.11
train.py ...
// on 172.16.176.25
torchrun --nnodes=4 --nproc_per_node=8 --node_rank=2 --master_addr=172.16.176.11
train.py ...
// on 172.16.176.28
torchrun --nnodes=4 --nproc_per_node=8 --node_rank=3 --master_addr=172.16.176.11
train.py ...
```

如果使用 mpirun 启动训练进程，按照顺序排列 IP 即可。

```
mpirun \
-np 64 \
-H 172.16.176.11:8,172.16.177.12:8,172.16.177.25:8,172.16.177.28:8 \
--allow-run-as-root \
-bind-to none -map-by slot \
-x NCCL_DEBUG=INFO
-x NCCL_IB_GID_INDEX=3 \
-x NCCL_IB_DISABLE=0 \
-x NCCL_SOCKET_IFNAME=eth0 \
-x
NCCL_IB_HCA=mlx5_bond_0,mlx5_bond_1,mlx5_bond_2,mlx5_bond_3,mlx5_bond_4,mlx5_bond_5,
mlx5_bond_6,mlx5_bond_7 \
-x NCCL_NET_GDR_LEVEL=2 \
-x NCCL_IB_QPS_PER_CONNECTION=4 \
-x NCCL_IB_TC=160 \
-x NCCL_IB_TIMEOUT=22 \
```

```
-x NCCL_PXN_DISABLE=0 \
-x LD_LIBRARY_PATH -x PATH \
-mca coll_hcoll_enable 0 \
-mca pml ob1 \
-mca btl_tcp_if_include eth0 \
-mca btl ^openib \
all_reduce_perf -b 1G -e 1G -n 1000 -g 1
```

配置 HARP 分布式训练环境

最近更新时间：2024-09-24 10:58:01

操作场景

本文介绍如何通过云服务器控制台，为实例配置 HARP 分布式训练环境。

操作步骤

绑定弹性网卡

弹性网卡数量等于 GPU 卡的数量，例如8卡训练机器则需要绑定8张弹性网卡（加主网卡共9张网卡）。具体步骤如下：

1. 登录 云服务器控制台，选择实例 ID 进入详情页面。
2. 在实例详情页中，选择**弹性网卡**页签，并单击**绑定弹性网卡**。如下图所示：

3. 在弹出的**绑定弹性网卡**窗口中，选择弹性网卡，单击**确认**即可。

配置并验证环境

1. 参见 使用标准登录方式登录 Linux 实例，登录实例。
2. 执行以下命令，执行配置脚本。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/taco_setup.sh | sudo bash
```

返回结果如下图所示：

```
[Thu 12 May 2022 08:36:50 PM CST] ========== Start configuring environment for TACO-Training ==========
[Thu 12 May 2022 08:36:50 PM CST] Check whether assistant nics are attached and enough
[Thu 12 May 2022 08:36:50 PM CST] Modify /etc/rc.local to auto re-configure harp after each reboot
[Thu 12 May 2022 08:36:50 PM CST] Bind assistant nic(s) to kernel at first
[Thu 12 May 2022 08:36:51 PM CST] Remove existing harp configurations
[Thu 12 May 2022 08:36:51 PM CST] Start downloading tools package ...
[Thu 12 May 2022 08:36:51 PM CST] Start loading the uio module for "Ubuntu" ...
[Thu 12 May 2022 08:36:51 PM CST] Start generating harp config files ...
[Thu 12 May 2022 08:36:53 PM CST] Set up HARP successfully
```

3. 执行以下命令，重启实例。

```
sudo reboot
```

4. 依次执行以下命令，检查是否配置成功。
   - 检查大页内存是否配置成功：

     ```
     cat /proc/meminfo | grep HugePages_Total
     ```

     返回如下结果，表示配置成功。

     ```
     HugePages_Total:      50
     ```

   - 检查是否产生了配置文件：

     ```
     ls -l /usr/local/tfabric/tools/config/ztcp*.conf
     ```

     返回结果如下图所示，表示已产生配置文件。

```
(base) ubuntu@VM-18-223-ubuntu:~$ ls -l /usr/local/tfabric/tools/config/ztcp*.conf
-rw-rw-rw- 1 root root 898 May 25 10:36 /usr/local/tfabric/tools/config/ztcp1.conf
-rw-rw-rw- 1 root root 900 May 25 10:36 /usr/local/tfabric/tools/config/ztcp2.conf
-rw-rw-rw- 1 root root 901 May 25 10:36 /usr/local/tfabric/tools/config/ztcp3.conf
-rw-rw-rw- 1 root root 905 May 25 10:36 /usr/local/tfabric/tools/config/ztcp4.conf
-rw-rw-rw- 1 root root 906 May 25 10:36 /usr/local/tfabric/tools/config/ztcp5.conf
-rw-rw-rw- 1 root root 911 May 25 10:36 /usr/local/tfabric/tools/config/ztcp6.conf
-rw-rw-rw- 1 root root 912 May 25 10:36 /usr/local/tfabric/tools/config/ztcp7.conf
-rw-rw-rw- 1 root root 894 May 25 10:36 /usr/local/tfabric/tools/config/ztcp.conf
```

配置容器 SSH 免密访问

最近更新时间：2024-09-24 10:58:01

操作场景

本文介绍如何配置服务器间的容器 SSH 免密访问。

操作步骤

> **说明：**
> 本文以两台机器间容器 SSH 免密访问为例，配置步骤需在两台机器上同步。

1. 参见 使用标准登录方式登录 Linux 实例，登录实例。
2. 执行以下命令，允许 root 使用 ssh 服务，并启动服务（默认端口：22）。

```
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
```

3. 依次执行以下命令，修改容器内 ssh 默认端口为2222，防止与 host 所使用的22端口冲突。

```
sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config
```

```
service ssh restart && netstat -tulpn
```

4. 执行以下命令，设置 root 密码。

```
passwd root
```

5. 执行以下命令，产生 SSH Key。

```
ssh-keygen
```

6. 创建 `~/.ssh/config`，并添加以下内容后，保存并退出，完成 host alias 配置。

```
# ！注意：
# 如果是CVM机型，则ip是两台机器`ifconfig eth0`显示的ip
# 如果是黑石RDMA机型，则ip是两台机器`ifconfig bond0`显示的ip
Host gpu1
 hostname 10.0.2.8
 port 2222
Host gpu2
 hostname 10.0.2.9
 port 2222
```

7. 使用 `ssh-copy-id` 将 SSH key 拷贝到对应的机器，使两台机器互相免密，同时本机对自身进行免密。

```
ssh-copy-id gpu1
```

```
ssh-copy-id gpu2
```

容器安装用户态 RDMA 驱动

最近更新时间：2024-09-20 21:56:01

操作场景

本文介绍如何为容器安装用户态 RDMA 驱动。

操作步骤

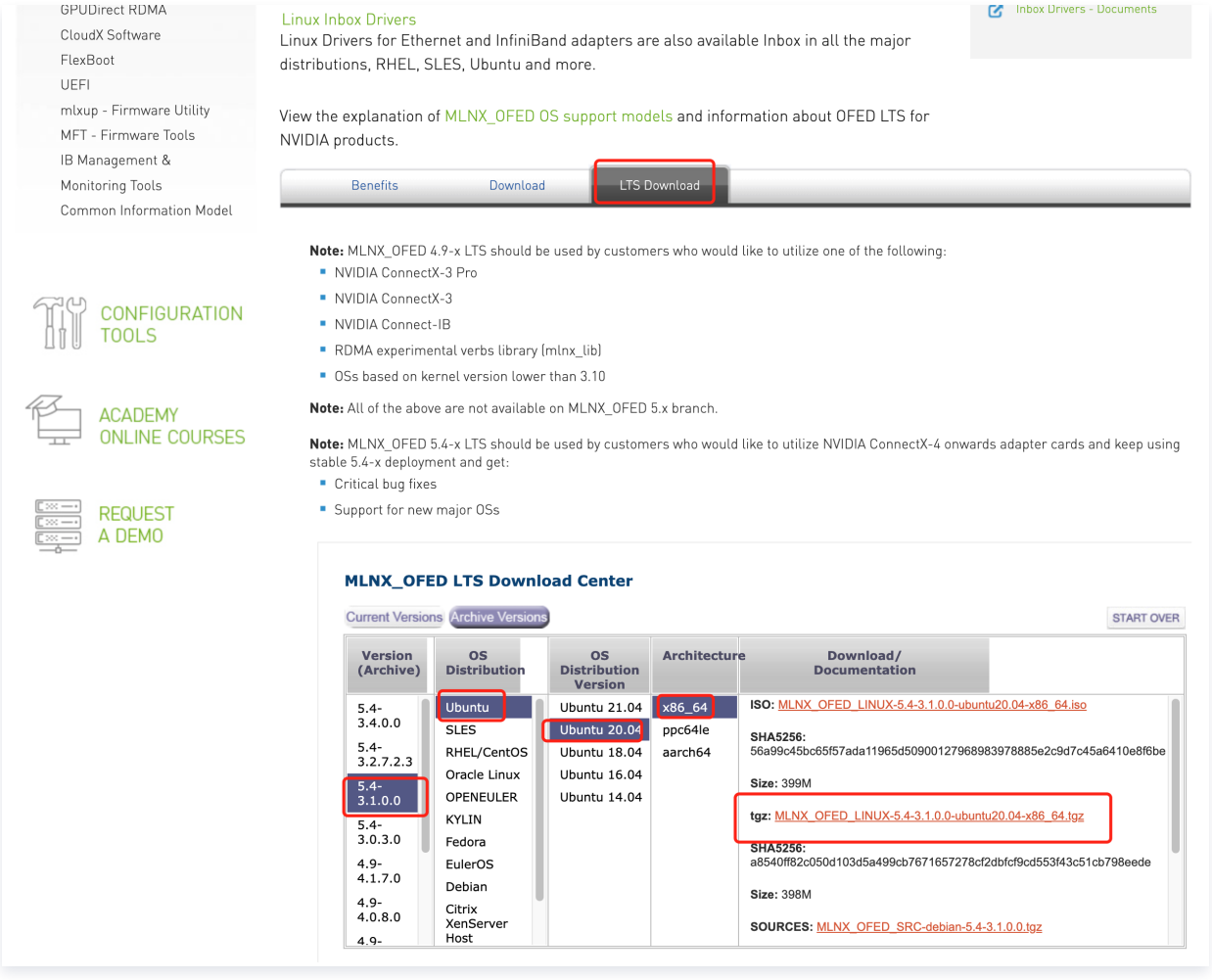
> **说明：**
> 本文以 Ubuntu 20.04 操作系统的机器为例。

1. 执行以下命令，下载对应容器中的 OS 版本的 MLNX OFED 驱动。

```
wget https://www.mellanox.com/downloads/ofed/MLNX_OFED-5.4-3.1.0.0/MLNX_OFED_LINUX-5.4-3.1.0.0-ubuntu20.04-x86_64.tgz
```

若您使用了其他版本操作系统，则请访问 Linux InfiniBand Drivers 下载对应的版本。选择步骤如下图所示：

> **注意：**
> OFED 版本选择 `5.4-3.1.0.0` 。



2. 依次执行以下命令，进行解压及安装。

```
tar xf MLNX_OFED_LINUX-5.4-3.1.0.0-ubuntu20.04-x86_64.tgz
```

```
cd MLNX_OFED_LINUX-5.4-3.1.0.0-ubuntu20.04-x86_64
```

```
./mlnxofedinstall --user-space-only --without-fw-update --force
```

安装过程中出现的红色 warning 信息可忽略，直至页面出现 **Installation passed successfully** 绿色字样，表示安装成功。

相关操作

若您在安装过程中出现如下图所示错误：

```
Installing mlnx-iproute2-5.6.0...
Installing neohost-backend-1.5.0...
Failed to install neohost-backend DEB
Collecting debug info...
See /tmp/MLNX_OFED_LINUX.7204.logs/neohost-backend.debinstall.log
```

请参考以下步骤处理：

1. 由于 neohost 需要依赖 python2，执行以下命令，修改系统默认的 python 版本。

```
ln -sf /usr/bin/python2.7 /usr/bin/python
```

2. 执行以下命令，确认 python 版本。

```
python --version
```

如果提示找不到 python 命令，则需要安装 python2.7。

3. 执行以下命令，重新安装 ofed。

```
./mlnxofedinstall --user-space-only --without-fw-update --force
```

4. 执行以下命令，恢复 python3 作为默认 python 版本。

```
update-alternatives --install /usr/bin/python python /usr/bin/python3 1
```