**预训练分布式环境搭建**

**一. Docker**

1. **安装docker**

|  |
| --- |
| Shell  # 由于Ubuntu里apt官方库里的docker版本可能比较低，因此先用下面的命令行卸载旧版本  apt-get remove docker docker-engine docker-ce docker.io    # 更新apt包索引  apt-get update    # 执行下列命令行，使apt可以通过HTTPS协议去使用存储库  apt-get install -y apt-transport-https ca-certificates curl software-properties-common    # 添加Docker官方提供的GPG密钥  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -    # 设置stable存储库  add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"    # 再次更新apt包索引  apt-get update    # 安装最新版本的docker-ce  apt-get install -y docker-ce |

1. **Docker 换源**

|  |
| --- |
| Shell  # <https://yours.mirror.aliyuncs.com>为自己的docker源仓库  mkdir -p /etc/docker  tee /etc/docker/daemon.json <<-'EOF'  {  "registry-mirrors": ["https://yours.mirror.aliyuncs.com"]  }  EOF |

|  |
| --- |
| Shell  systemctl daemon-reload  systemctl restart docker |

1. **安装显卡驱动（如已装可跳过）**

|  |
| --- |
| Shell  # 检验是否村存在Nvidia驱动  dpkg --list | grep nvidia-\*  # 或者执行cat /proc/driver/nvidia/version    # 如果不存在Nvidia驱动，则需要安装  # 执行ubuntu-drivers devices，查看推荐驱动版本  ubuntu-drivers devices  # 如果“Command 'ubuntu-drivers' not found”，执行apt-get install ubuntu-drivers-common    # 安装推荐的驱动版本  apt-get install nvidia-driver-版本号    # 检验是否安装成功  nvidia-smi  # 如果出现"NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver. Make sure that the latest NVIDIA driver is installed and running."错误  # 执行apt install dkms  # 查看版本号ls /usr/src | grep nvidia  # dkms install -m nvidia -v + 版本号  # 注意：安装完成后，可能需要重新启动服务器 |

1. **配置nvidia-docker源：**

|  |
| --- |
| Shell  # 添加源  distribution=$(. /etc/os-release;echo $ID$VERSION\_ID)  curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -  curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list    # 安装nvidia-docker2和依赖，安装的过程中，选择“默认”  apt-get update  apt-get install -y nvidia-docker2 |

|  |
| --- |
| Plain Text  修改/etc/docker/daemon.json，添加相关信息  "runtimes": {  "nvidia": {  "path": "/usr/bin/nvidia-container-runtime",  "runtimeArgs": []  }  }    /etc/docker/daemon.json最终内容  {  "registry-mirrors": ["[https://](https://obxxb74v.mirror.aliyuncs.com/)xxxxxx"],  "runtimes": {  "nvidia": {  "path": "/usr/bin/nvidia-container-runtime",  "runtimeArgs": []  }  }  } |

|  |
| --- |
| Shell  # 重启docker服务  systemctl daemon-reload  systemctl restart docker |

1. **制作dockerfile**
2. 拉取nvidia 基础镜像, 创建临时文件夹（容器内，镜像创建完成后，删除）

|  |
| --- |
| Dockerfile  # pull base image  FROM nvidia/cuda:10.2-devel-ubuntu18.04  # maintainer  MAINTAINER deepspeed <gqwang@baai.ac.cn>    ##############################################################################  # Temporary Installation Directory  ##############################################################################  ENV STAGE\_DIR=/tmp  RUN mkdir -p ${STAGE\_DIR} |

1. 配置apt 安装源，并安装一些linux 系统常用基础包

|  |
| --- |
| Dockerfile  ##############################################################################  # Installation/Basic Utilities  ##############################################################################  RUN sed -i s@/[archive.ubuntu.com/@/mirrors.tuna.tsinghua.edu.cn/@g](http://archive.ubuntu.com/@/mirrors.tuna.tsinghua.edu.cn/@g) /etc/apt/sources.list  RUN sed -i s@/[security.ubuntu.com/@/mirrors.tuna.tsinghua.edu.cn/@g](http://security.ubuntu.com/@/mirrors.tuna.tsinghua.edu.cn/@g) /etc/apt/sources.list  RUN apt-get update && \  DEBIAN\_FRONTEND="noninteractive" apt-get install -y --no-install-recommends \  software-properties-common build-essential autotools-dev \  nfs-common pdsh \  cmake g++ gcc \  curl wget vim tmux emacs less unzip \  htop iftop iotop ca-certificates openssh-client openssh-server \  rsync iputils-ping net-tools sudo \  llvm-9-dev libsndfile-dev \  libcupti-dev \  libjpeg-dev \  libpng-dev \  screen jq psmisc dnsutils lsof musl-dev systemd |

1. 安装最新版git(创建镜像clone 安装包)

|  |
| --- |
| Dockerfile  ##############################################################################  # Installation Latest Git  ##############################################################################  RUN add-apt-repository ppa:git-core/ppa -y && \  apt-get update && \  apt-get install -y git && \  git --version |

1. 安装 Mellanox OFED, 由于网络问题，推荐安装包下到本地后，再执行dockerfile

|  |
| --- |
| Dockerfile  ##############################################################################  # install Mellanox OFED  # dwonload from <https://www.mellanox.com/downloads/ofed/MLNX_OFED-5.1-2.5.8.0/MLNX_OFED_LINUX-5.1-2.5.8.0-ubuntu18.04-x86_64.tgz>  ##############################################################################  RUN apt-get install -y libnuma-dev libnuma-dev libcap2  ENV MLNX\_OFED\_VERSION=5.1-2.5.8.0  COPY MLNX\_OFED\_LINUX-${MLNX\_OFED\_VERSION}-ubuntu18.04-x86\_64.tgz ${STAGE\_DIR}  RUN cd ${STAGE\_DIR} && \  tar xvfz MLNX\_OFED\_LINUX-${MLNX\_OFED\_VERSION}-ubuntu18.04-x86\_64.tgz && \  cd MLNX\_OFED\_LINUX-${MLNX\_OFED\_VERSION}-ubuntu18.04-x86\_64 && \  PATH=/usr/bin:$PATH ./mlnxofedinstall --user-space-only --without-fw-update --umad-dev-rw --all -q && \  cd ${STAGE\_DIR} && \  rm -rf ${STAGE\_DIR}/MLNX\_OFED\_LINUX-${MLNX\_OFED\_VERSION}-ubuntu18.04-x86\_64\* |

1. 安装 nv\_peer\_mem

|  |
| --- |
| Dockerfile  ##############################################################################  # Install nv\_peer\_mem  ##############################################################################  #COPY nv\_peer\_memory ${STAGE\_DIR}/nv\_peer\_memory (without net)  ############try for more times #################    ENV NV\_PEER\_MEM\_VERSION=1.1  ENV NV\_PEER\_MEM\_TAG=1.1-0  RUN git clone <https://github.com/Mellanox/nv_peer_memory.git> --branch ${NV\_PEER\_MEM\_TAG} ${STAGE\_DIR}/nv\_peer\_memory  RUN cd ${STAGE\_DIR}/nv\_peer\_memory && \  ./build\_module.sh && \  cd ${STAGE\_DIR} && \  tar xzf ${STAGE\_DIR}/nvidia-peer-memory\_${NV\_PEER\_MEM\_VERSION}.orig.tar.gz && \  cd ${STAGE\_DIR}/nvidia-peer-memory-${NV\_PEER\_MEM\_VERSION} && \  apt-get update && \  apt-get install -y dkms && \  dpkg-buildpackage -us -uc && \  dpkg -i ${STAGE\_DIR}/nvidia-peer-memory\_${NV\_PEER\_MEM\_TAG}\_all.deb |

1. 安装openmpi, 需先安装libevent 依赖包

|  |
| --- |
| Dockerfile  ###########################################################################  # Install libevent && OPENMPI  # <https://www.open-mpi.org/software/ompi/v4.0/>  ##############################################################################  ENV OPENMPI\_BASEVERSION=4.0  ENV OPENMPI\_VERSION=${OPENMPI\_BASEVERSION}.5  COPY openmpi-4.0.5.tar.gz ${STAGE\_DIR}  COPY libevent-2.0.22-stable.tar.gz ${STAGE\_DIR}  RUN cd ${STAGE\_DIR} && \  tar zxvf libevent-2.0.22-stable.tar.gz && \  cd libevent-2.0.22-stable && \  ./configure --prefix=/usr && \  make && make install  RUN cd ${STAGE\_DIR} && \  tar --no-same-owner -xzf openmpi-4.0.5.tar.gz && \  cd openmpi-${OPENMPI\_VERSION} && \  ./configure --prefix=/usr/local/openmpi-${OPENMPI\_VERSION} && \  make -j"$(nproc)" install && \  ln -s /usr/local/openmpi-${OPENMPI\_VERSION} /usr/local/mpi && \  # Sanity check:  test -f /usr/local/mpi/bin/mpic++ && \  cd ${STAGE\_DIR} && \  rm -r ${STAGE\_DIR}/openmpi-${OPENMPI\_VERSION}  ENV PATH=/usr/local/mpi/bin:${PATH} \  LD\_LIBRARY\_PATH=/usr/local/lib:/usr/local/mpi/lib:/usr/local/mpi/lib64:${LD\_LIBRARY\_PATH}  # Create a wrapper for OpenMPI to allow running as root by default  RUN mv /usr/local/mpi/bin/mpirun /usr/local/mpi/bin/mpirun.real && \  echo '#!/bin/bash' > /usr/local/mpi/bin/mpirun && \  echo 'mpirun.real --allow-run-as-root --prefix /usr/local/mpi "$@"' >> /usr/local/mpi/bin/mpirun && \  chmod a+x /usr/local/mpi/bin/mpirun |

1. 安装 python

|  |
| --- |
| Dockerfile  ###########################################################################  # Install python  ##############################################################################  ARG PYTHON\_VERSION=3.8  RUN curl -o ~/miniconda.sh <https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-latest-Linux-x86_64.sh> && \  chmod +x ~/miniconda.sh && \  ~/miniconda.sh -b -p /opt/conda && \  rm ~/miniconda.sh && \  /opt/conda/bin/conda install -y python=$PYTHON\_VERSION numpy pyyaml scipy ipython mkl mkl-include ninja cython typing |

1. 安装 magma-cuda

|  |
| --- |
| Dockerfile  ###########################################################################  # Install magma-cuda  ##############################################################################  COPY magma-cuda102-2.5.2-1.tar.bz2 ${STAGE\_DIR}  RUN cd ${STAGE\_DIR} && \  /opt/conda/bin/conda install -y -c pytorch --use-local magma-cuda102-2.5.2-1.tar.bz2 && \  /opt/conda/bin/conda clean -ya  ####optional#####  #RUN /opt/conda/bin/conda install -y -c pytorch magma-cuda102 && \  # /opt/conda/bin/conda clean -ya |

1. 配置路径

|  |
| --- |
| Dockerfile  ###########################################################################  # Export path  ##############################################################################  ENV PATH /opt/conda/bin:$PATH  RUN echo "export PATH=/opt/conda/bin:\$PATH" >> /root/.bashrc  RUN pip install --upgrade pip setuptools  RUN wget <https://tuna.moe/oh-my-tuna/oh-my-tuna.py> && python oh-my-tuna.py |

1. 安装一些pip 包

|  |
| --- |
| Dockerfile  ###########################################################################  # Install some Packages  ##############################################################################  RUN pip install psutil \  yappi \  cffi \  ipdb \  h5py \  pandas \  matplotlib \  py3nvml \  pyarrow \  graphviz \  astor \  boto3 \  tqdm \  sentencepiece \  msgpack \  requests \  pandas \  sphinx \  sphinx\_rtd\_theme \  sklearn \  scikit-learn \  nvidia-ml-py3 \  nltk \  rouge \  filelock \  fasttext \  rouge\_score \  cupy-cuda102\  setuptools==60.0.3 |

1. 安装mpi4py （需下到本地安装，pip 安装容易版本兼容问题报错）

|  |
| --- |
| Dockerfile  ##############################################################################  # Install mpi4py  ##############################################################################  RUN apt-get update && \  apt-get install -y mpich  COPY mpi4py-3.1.3.tar.gz ${STAGE\_DIR}  RUN cd ${STAGE\_DIR} && tar zxvf mpi4py-3.1.3.tar.gz && \  cd mpi4py-3.1.3 &&\  python setup.py build && python setup.py install |

1. 安装pytorch, 版本可替换， 需先下载项目到本地，国内安装容易因为网速原因，造成终止, pytorch git clone 过程中可能有些子包下载过程中会终止。可以多 git clone 几次

|  |
| --- |
| Dockerfile  ##############################################################################  # PyTorch  # clone (may be time out because of the network problem)  #RUN git clone --recursive <https://github.com/pytorch/pytorch> --branch v1.8.1 /opt/pytorch  #RUN cd /opt/pytorch && git checkout -f v1.8.1 && \  # git submodule sync && git submodule update -f --init --recursive  ##############################################################################  ENV TORCH\_CUDA\_ARCH\_LIST="6.0 6.1 7.0+PTX"    COPY opt/pytorch /opt/pytorch  ENV NCCL\_LIBRARY=/usr/lib/x86\_64-linux-gnu  ENV NCCL\_INCLUDE\_DIR=/usr/include  RUN cd /opt/pytorch && TORCH\_NVCC\_FLAGS="-Xfatbin -compress-all" \  CMAKE\_PREFIX\_PATH="$(dirname $(which conda))/../" USE\_SYSTEM\_NCCL=1 \  pip install -v . && rm -rf /opt/pytorch        ##############################################################################  # Install vision  #RUN git clone <https://github.com/pytorch/vision.git> /opt/vision  ##############################################################################  COPY vision /opt/vision  RUN cd /opt/vision && pip install -v . && rm -rf /opt/vision  ENV TENSORBOARDX\_VERSION=1.8  RUN pip install tensorboardX==${TENSORBOARDX\_VERSION} |

1. 安装apex

|  |
| --- |
| Dockerfile  ###########################################################################  # Install apex  ###########################################################################  #RUN git clone <https://github.com/NVIDIA/apex> ${STAGE\_DIR}/apex  COPY apex ${STAGE\_DIR}/apex  RUN cd ${STAGE\_DIR}/apex && pip install -v --no-cache-dir --global-option="--cpp\_ext" --global-option="--cuda\_ext" ./ \  && rm -rf ${STAGE\_DIR}/apex |

1. 安装deepspeed

|  |
| --- |
| Dockerfile  ############################################################################  # Install deepSpeed  #############################################################################  RUN pip install py-cpuinfo  RUN apt-get install -y libaio-dev  ENV TORCH\_CUDA\_ARCH\_LIST="6.0 6.1 7.0+PTX"  RUN git clone https://github.com/microsoft/DeepSpeed.git ${STAGE\_DIR}/DeepSpeed  #COPY DeepSpeed ${STAGE\_DIR}/DeepSpeed  RUN cd ${STAGE\_DIR}/DeepSpeed && \  git checkout . && \  DS\_BUILD\_OPS=1 ./install.sh -r  RUN rm -rf ${STAGE\_DIR}/DeepSpeed  RUN python -c "import deepspeed; print(deepspeed.\_\_version\_\_)" |

1. 安装NCCL(可选，pytorch 已自带)

|  |
| --- |
| Dockerfile  ############################################################################  # Install nccl  #############################################################################    #COPY nccl-local-repo-ubuntu1804-2.9.6-cuda10.2\_1.0-1\_amd64.deb ${STAGE\_DIR}  #RUN cd ${STAGE\_DIR} &&\  # sudo dpkg -i nccl-local-repo-ubuntu1804-2.9.6-cuda10.2\_1.0-1\_amd64.deb &&\  # sudo apt install -y libnccl2 libnccl-dev  #RUN apt install -y --allow-downgrades --no-install-recommends --allow-change-held-packages libnccl2=2.9.6-1+cuda10.2 libnccl-dev=2.9.6-1+cuda10.2  #ENV NCCL\_VERSION=2.9.6 |

1. 配置网络端口、公钥和ssh

|  |
| --- |
| Dockerfile  ############################################################################  # Set SSH Config  #############################################################################  RUN apt-get install openssh-server    ARG SSH\_PORT=6001  # RUN echo 'root:NdjeS+-4gEPmq}D' | chpasswd  # Client Liveness & Uncomment Port 22 for SSH Daemon  RUN echo "ClientAliveInterval 30" >> /etc/ssh/sshd\_config  RUN mkdir -p /var/run/sshd && cp /etc/ssh/sshd\_config ${STAGE\_DIR}/sshd\_config && \  sed "0,/^#Port 22/s//Port 22/" ${STAGE\_DIR}/sshd\_config > /etc/ssh/sshd\_config  RUN cat /etc/ssh/sshd\_config > ${STAGE\_DIR}/sshd\_config && \  sed "0,/^Port 22/s//Port ${SSH\_PORT}/" ${STAGE\_DIR}/sshd\_config > /etc/ssh/sshd\_config && \  sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/g' /etc/ssh/sshd\_config  EXPOSE ${SSH\_PORT}  # Set SSH KEY  RUN printf "#StrictHostKeyChecking no\n#UserKnownHostsFile /dev/null" >> /etc/ssh/ssh\_config && \  ssh-keygen -t rsa -f ~/.ssh/id\_rsa -N "" && cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys && \  chmod og-wx ~/.ssh/authorized\_keys  CMD service ssh start |

1. **构建docker 镜像**
2. 方式一. pull 镜像

|  |
| --- |
| Shell  # 远程拉取  docker pull deepspeed/cuda102  # 本地读取  docker load --input deepspeed-cuda102.tar.gz |

1. 方式二. 构建镜像

|  |
| --- |
| Shell  docker build -f cuda102.dockerfile -t deepspeed/cuda102:1221 .  # cuda102.dockerfile 参考 dockerfile 文件制作流程 |

**二. 在每个机器节点构建容器**

|  |
| --- |
| Shell  # 创建容器（nvidia-docker），hostname, 容器内部host 名称，network= host 与数组机共享，ipc=host, 集群训练时，需按此设置。shm\_size 共享内存，name 容器外部名，--gpus 指定gpu，多数据卷：-v 本地文件夹:容器内文件夹 -v 本地文件夹:容器内文件夹 -v 本地文件夹:容器内文件夹 deepspeed/cuda102:1221 镜像名：tag  nvidia-docker run -id --hostname=glm\_dist16 --network=host --ipc=host --shm-size=16gb --name=glm\_dist16 --gpus '"device=0,1,2,3"' -v /data1/docker/containers:/data deepspeed/cuda102:1221 |

相关命令

|  |
| --- |
| Shell  # 拉取镜像  docker pull nvidia/cuda:cuda版本-runtime-ubuntu版本  # 拉取镜像举例  docker pull nvidia/cuda:10.1-runtime-ubuntu18.04  # 创建容器（普通docker），比如多端口号：-p 22:22 -p 80:80 -p 8080:8080，多数据卷：-v 文件夹:文件夹 -v 文件夹:文件夹 -v 文件夹:文件夹  docker run -id --name=容器名 -p 宿主机端口号:容器内端口号 -e TZ=Asia/Shanghai -v 宿主机文件夹:容器内文件夹 镜像:版本号  # 创建容器（普通docker）举例  docker run -id --name=test -p 80:80 -e TZ=Asia/Shanghai -v /data:/data mysql:5.7  # 创建容器（nvidia-docker），支持多映射，比如多端口号：-p 22:22 -p 80:80 -p 8080:8080，多数据卷：-v 文件夹:文件夹 -v 文件夹:文件夹 -v 文件夹:文件夹  nvidia-docker run -id --name=容器名 -p 宿主机端口号:容器内端口号 -e TZ=Asia/Shanghai --shm-size=大小 -v 宿主机文件夹:容器内文件夹 镜像:版本号  # 创建容器（nvidia-docker）举例  docker run -id --name=test -p 80:80 -e TZ=Asia/Shanghai --shm-size=8gb -v /data:/data nvidia/cuda:10.1-runtime-ubuntu18.04  # 进入容器（普通docker）  docker exec -it 容器名 /bin/bash  # 进入容器（nvidia-docker）  nvidia-docker exec -it 容器名 /bin/bash  # 查看已有镜像  docker images  # 删除镜像  docker rmi 镜像名/镜像id  # 查看正在运行的容器  docker ps  # 查看历史容器，包含正在运行和已经关闭的  docker ps -a  # 停止正在运行的容器  docker stop 容器名/容器id  # 删除容器，如果待删除的容器正在运行，需要先停止再删除  docker rm 容器名/容器id |

**三. 互信机制设置**

1. 公钥生成默认docker 镜像创建时已生成，如不存在，则在shell 端输入

|  |
| --- |
| Shell  ssh-keygen -t rsa -C "example@com.cn"） |

1. 将各节点容器生成的公钥文件

~/.ssh/id\_rsa.pub

中的内容收集，并同步到各机器的文件

~/.ssh/authorized\_keys

1. 免密登陆，

配置各节点容器port : vi /etc/ssh/sshd\_config , 将port 注释取消，并设值，同一节点不同容器port 需要不一样

如下配置各节点 host 文件 ：vi ~/.ssh/config 复制各节点host 登陆信息, 并同步到各节点

|  |
| --- |
| Plain Text  Host V100-1  Hostname 172.31.32.29  Port 6001  User root  Host V100-2  Hostname 172.31.32.40  Port 6002  User root |

1. 测试

ssh V100-1

**四. 分布式训练测试**

a.配置hostfile（hostfile 中的V100-1 与~/.ssh/config 对应）:

|  |
| --- |
| Plain Text  V100-1 slots=4  V100-2 slots=4  V100-3 slots=4  ..... |

b. 配置glm 文件，各节点配置code 和数据，要求路径相同（也可共同访问云端共享文件）

c. cmd

|  |
| --- |
| Shell  bash config/start\_scripts/generate\_block.sh config/config\_tasks/model\_blocklm\_large\_chinese.sh |