

分布式机器学习笔记

Ruichen Wang

March 15, 2019

Abstract

Contents

1	基本知识	1
1.1	ps vs ringAllReduce ?	2
2	pytorch distirbuted	3
3	tensorflow distributed	3
4	docker + tensorflow gpu/cpu	5
4.1	docker 是什么?	5
4.2	安装docker	5
5	k8s	6
5.1	安装k8s	6
5.2	简介	6

1 基本知识

Communication Backends 主要有四种:

- TCP
 - 适用范围广，大部分系统和机器都提供支持。支持cpu上的p2p和collective functions。但是不支持GPU，优化程度也不高。
- MPI
 - 全称The Message Passing Interface，是高性能计算的一个标准工具。高度的并行化，在大规模集群上优化程度较高。pytorch使用MPI需要自己重新手动编译
- Gloo

- 优化collective function, 同时支持gpu 与cpu .使用GPUDirect技术, 在gpu间通信不需要通过cpu.

- NCCL

- 全称The NVIDIA Collective Communications Library, 主要针对Nvidia多机多卡做了对应优化.

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✗
recv	✓	✗	✓	?	✗	✗
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✗
scatter	✓	✗	✓	?	✗	✗
barrier	✓	✗	✓	?	✗	✓

Figure 1: backends

参考: https://pytorch.org/tutorials/intermediate/dist_tuto.html#our-own-ring-allreduce

1.1 ps vs ringAllReduce ?

主流分布式机器学习采用的是ps架构。如tensorflow。ps全称Parameter Server Architecture 也就是参数服务器。

在Parameter server架构（PS架构）中，集群中的节点被分为两类：parameter server和worker。其中parameter server存放模型的参数，而worker负责计算参数的梯度。在每个迭代过程，worker从parameter sever中获得参数，然后将计算的梯度返回给parameter server，parameter server聚合从worker传回的梯度，然后更新参数，并将新的参数广播给worker。

pytorch 采用的是Uber Horovod的形式，也是baidu开源的ringAllReduce算法。采用PS计算模型的分布式，通常会遇到网络的问题，随着worker数量的增加，其加速比会迅速的恶化，例如resnet50这样的模型，目前的TF在10几台机器的時候，加速比已经开始恶化的不可接受了。因此，经常要上RDMA、InfiniBand等技术，并且还带来了一波网卡的升级，有些大厂直接上了100GBs的网卡，有

钱任性。而Uber的Horovod，采用的RingAllReduce的计算方案，其特点是网络通信量不随着worker（GPU）的增加而增加，是一个恒定值。简单看下图理解下，GPU 集群被组织成一个逻辑环，每个GPU有一个左邻居、一个右邻居，每个GPU只从左邻居接受数据、并发送数据给右邻居。即每次梯度每个gpu只获得部分梯度更新，等一个完整的Ring完成，每个GPU都获得了完整的参数。

2 pytorch distirbuted

pytorch 提供distribute包。

```
def init_processes(rank, size, fn, backend='nccl'):
    """ Initialize the distributed environment. """
    os.environ['MASTER_ADDR'] = '127.0.0.1'
    os.environ['MASTER_PORT'] = '29500'
    dist.init_process_group(backend, rank=rank, world_size=size)
    fn(rank, size)

if __name__ == "__main__":
    size = 4
    processes = []
    for rank in range(size):
        p = Process(target=init_processes, args=(rank, size, run))
        p.start()
        processes.append(p)]
```

Figure 2: backends

3 tensorflow distributed

tensorflow 提供tf.train.ClusterSpec来创建相关cluter集群。

```

cluster = tf.train.ClusterSpec({'ps': ps_spec, 'worker': worker_spec})
server = tf.train.Server(cluster, job_name=FLAGS.job_name, task_index=FLAGS.task_index)
if FLAGS.job_name == 'ps':
    server.join()

is_chief = (FLAGS.task_index == 0)
with tf.device(tf.train.replica_device_setter(
    cluster=cluster
)):
    global_step = tf.Variable(0, name='global_step', trainable=False)
    hid_w = tf.Variable(tf.truncated_normal([IMAGE_PIXELS * IMAGE_PIXELS, FLAGS.hidden_units],
        stddev=1.0 / IMAGE_PIXELS), name='hid_w')
    hid_b = tf.Variable(tf.zeros([FLAGS.hidden_units]), name='hid_b')
    sm_w = tf.Variable(tf.truncated_normal([FLAGS.hidden_units, 10],
        stddev=1.0 / math.sqrt(FLAGS.hidden_units)), name='sm_w')
    sm_b = tf.Variable(tf.zeros([10]), name='sm_b')
    x = tf.placeholder(tf.float32, [None, IMAGE_PIXELS * IMAGE_PIXELS])
    y = tf.placeholder(tf.float32, [None, 10])
    hid_lin = tf.nn.xw_plus_b(x, hid_w, hid_b)
    hid = tf.nn.relu(hid_lin)
    y = tf.nn.softmax(tf.nn.xw_plus_b(hid, sm_w, sm_b))
    cross_entropy = -tf.reduce_sum(y * tf.log(tf.clip_by_value(y, 1e-10, 1.0)))
    opt = tf.train.AdamOptimizer(FLAGS.learning_rate)

```

Figure 3: backends

分别在对应的ps,worker机器上执行相关脚本,等待所有节点加入后, 训练开始

```

wangrc@sha04:~$ cat /dev/null >> /dev/null
552383116.479922: Worker 0: training step 1797 done (global step:7596)
552383116.494266: Worker 0: training step 1798 done (global step:7598)
552383116.508687: Worker 0: training step 1799 done (global step:7600)
552383116.522407: Worker 0: training step 1800 done (global step:7602)
552383116.537765: Worker 0: training step 1801 done (global step:7604)
552383116.554577: Worker 0: training step 1802 done (global step:7606)
552383116.574095: Worker 0: training step 1803 done (global step:7608)
552383116.590281: Worker 0: training step 1804 done (global step:7610)
552383116.604561: Worker 0: training step 1805 done (global step:7612)
552383116.620999: Worker 0: training step 1806 done (global step:7614)
552383116.637168: Worker 0: training step 1807 done (global step:7616)
552383116.653346: Worker 0: training step 1808 done (global step:7618)
552383116.668836: Worker 0: training step 1809 done (global step:7620)
552383116.683025: Worker 0: training step 1810 done (global step:7622)
552383116.698552: Worker 0: training step 1811 done (global step:7624)
552383116.714826: Worker 0: training step 1812 done (global step:7626)
552383116.730834: Worker 0: training step 1813 done (global step:7628)
552383116.746829: Worker 0: training step 1814 done (global step:7630)
552383116.761880: Worker 0: training step 1815 done (global step:7632)
552383116.775290: Worker 0: training step 1816 done (global step:7634)
552383116.789999: Worker 0: training step 1817 done (global step:7636)
552383116.806842: Worker 0: training step 1818 done (global step:7638)
552383116.822391: Worker 0: training step 1819 done (global step:7640)
552383116.838144: Worker 0: training step 1820 done (global step:7642)
552383116.852750: Worker 0: training step 1821 done (global step:7644)
552383116.868844: Worker 0: training step 1822 done (global step:7646)
552383116.884383: Worker 0: training step 1823 done (global step:7648)
552383116.899455: Worker 0: training step 1824 done (global step:7650)
552383116.915388: Worker 0: training step 1825 done (global step:7652)
552383116.931080: Worker 0: training step 1826 done (global step:7654)
552383116.945950: Worker 0: training step 1827 done (global step:7656)
552383116.960686: Worker 0: training step 1828 done (global step:7658)
552383116.976164: Worker 0: training step 1829 done (global step:7660)
552383116.992065: Worker 0: training step 1830 done (global step:7662)
552383117.007880: Worker 0: training step 1831 done (global step:7664)
552383117.023551: Worker 0: training step 1832 done (global step:7666)
552383117.039157: Worker 0: training step 1833 done (global step:7668)
552383117.054744: Worker 0: training step 1834 done (global step:7670)
552383117.069590: Worker 0: training step 1835 done (global step:7672)
552383117.085140: Worker 0: training step 1836 done (global step:7674)
552383117.100342: Worker 0: training step 1837 done (global step:7676)
552383117.116053: Worker 0: training step 1838 done (global step:7678)
552383117.131318: Worker 0: training step 1839 done (global step:7680)
552383117.146462: Worker 0: training step 1840 done (global step:7682)
552383117.159993: Worker 0: training step 1841 done (global step:7684)
552383117.175481: Worker 0: training step 1842 done (global step:7686)
552383117.192599: Worker 0: training step 1843 done (global step:7688)
552383117.209801: Worker 0: training step 1844 done (global step:7690)
552383117.226391: Worker 0: training step 1845 done (global step:7692)
552383117.236560: Worker 0: training step 1846 done (global step:7694)
552383117.246675: Worker 0: training step 1847 done (global step:7696)
552383117.256668: Worker 0: training step 1848 done (global step:7698)
552383117.266945: Worker 0: training step 1849 done (global step:7700)
552383117.277337: Worker 0: training step 1850 done (global step:7702)

```

Figure 4: distribute training using tensorflow gpu

4 docker + tensorflow gpu/cpu

4.1 docker 是什么？

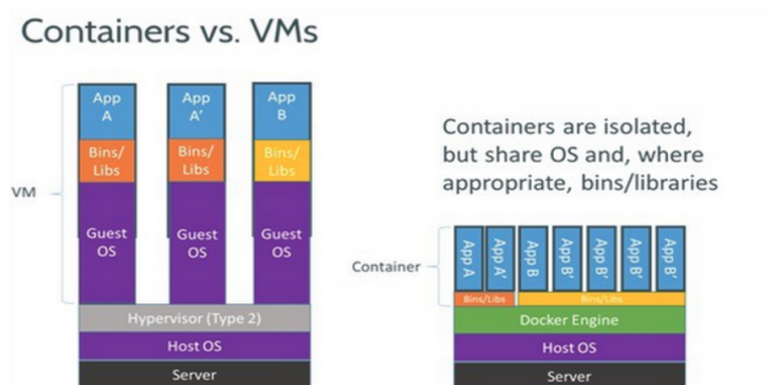


Figure 5: docker

4.2 安装docker

安装教程: <https://docs.docker.com/install/linux/docker-ce/ubuntu/>

- 查看镜像
docker images
- 修改docker镜像源

```
{
  "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
}
```
- 安装tensorflow docker cpu, 可以使用-tag选择想要的tf版本和python版本
docker pull tensorflow/tensorflow:1.12.0-py3
- 安装tensorflow docker gpu
 1. 首先安装nvidia-docker
(<https://github.com/NVIDIA/nvidia-docker>)
 2. 确认nvidia docker 安装完成
docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi

```
(base) wangrc@wangrc:~$ docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
Thu Mar 14 06:05:26 2019
If you are not using the official docker-ce package, please refer to the NVIDIA documentation for more details.

+-----+
| NVIDIA-SMI 415.13      | Driver Version: 415.13      | CUDA Version: 10.0      |
+-----+-----+
| GPU Name               | Persistence-M| Bus-Id        | Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0  GeForce GTX 108...  Off  | 00000000:01:00:00  On  |           N/A       |
| 29%   30C   P8      9W / 250W | 912MiB / 11175MiB |           0%      Default |
+-----+-----+

Processes:
GPU      PID    Type   Process name                      GPU Memory
=====+=====+

```

Figure 6: distribute training using tensorflow gpu

- 3. 安装tensorflow gpu
- docker pull tensorflow/tensorflow:1.12.0-gpu-py3
- 4. 启动支持nvidia的docker
- docker run --runtime=nvidia -it --rm tensorflow/tensorflow:1.12.0-gpu-py3 bash

```
2019-03-14 06:06:38.622977: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1080 Ti Major: 6 Minor: 1 MemoryLocation(GR2): 1.62
pciBusId: 0000:01:00:0
totalMemory: 10.50GiB freeMemory: 9.86GiB
2019-03-14 06:06:38.622975: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-03-14 06:06:31.574876: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-03-14 06:06:31.574865: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2019-03-14 06:06:31.574894: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1342] 0:  0
2019-03-14 06:06:31.574841: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1315] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 9532 MB memory) -> physical GPU (device: 0, name:
GeForce GTX 1080 Ti, pci bus id: 0000:01:00:0, compute capability: 6.1)
INFO:tensorflow:Mounting local unit op.
INFO:tensorflow:Mounting local unit op.
INFO:tensorflow:Starting standard services.
WARNING:tensorflow:Standard services need a 'logdir' passed to the SessionManager
INFO:tensorflow:Starting queue runners.
0.5870687 [0.3985862]
4.3641443 [1.499764]
7.797088 [1.412553]
19.803888 [1.781145]
13.711188 [1.173286]
16.259883 [7.443175]
18.547088 [1.39254]
20.425198 [9.436292]
22.583788 [18.589611]
24.202912 [11.429549]

CentOS 7 (docker-ce), RHEL 7.47.5 (docker-ce), Amazon Linux 1/2

If you are not using the official docker-ce package on CentOS/RHEL, use the next section.

If you have trouble installing Docker on CentOS/RHEL, see the next section.
https://docs.docker.com/install/linux/docker-ce/centos/
If you are not using the official docker-ce package on CentOS/RHEL, use the next section.
```

Figure 7: training using docker tensorflow gpu

5 k8s

5.1 安装k8s

<https://kubernetes.io/docs/setup/>

5.2 简介

随着docker、容器的日渐成熟，容器编排的问题就凸显出来，大量的容器怎么去管理，怎么调度，怎么启停都成了迫切需要解决的问题。单纯地使

用docker没有办法限制cpu or gpu资源. k8s 支持为每个请求分配cpu与gpu资源. k8s优点这里就不详细列举了。

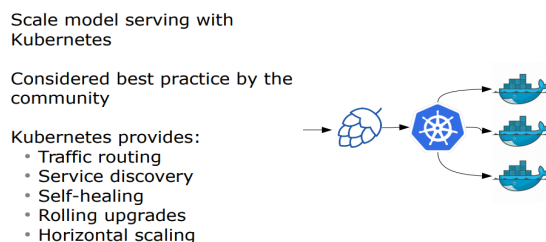


Figure 8: k8s and docker

	物理机	k8s
更改系统配置	需要每台节点修改	更改dockerfile, 全部生效
增加节点	需要安装相同环境	修改pod数量即可
节点宕机	此节点上业务停止	自动在迁移至其他节点

Figure 9: k8s 2

典型的Kubernetes 集群包含一个master 和很多node。Master 是控制集群的中心，node 是提供CPU、内存和存储资源的节点。Master 上运行着多个进程，包括面向用户的API 服务、负责维护集群状态的Controller Manager、负责调度任务的Scheduler 等。每个node 上运行着维护node 状态并和master 通信的kubelet，以及实现集群网络服务的kube-proxy。

Kubernetes 中部署的最小单位是pod，而不是Docker 容器。实时上Kubernetes 是不依赖于Docker 的，完全可以使用其他的容器引擎在Kubernetes 管理的集群中替代Docker。在与Docker 结合使用时，一个pod 中可以包含一个或多个Docker 容器。但除了有紧密耦合的情况下，通常一个pod 中只有一个容器，这样方便不同的服务各自独立地扩展。

k8s 为pod分配gpu资源时可以指定gpu个数和类型：

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
    - name: cuda-vector-add
      # https://github.com/kubernetes/kubernetes/blob/v1.7.11/test/images/nvidia-cuda/Dockerfile
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    accelerator: nvidia-tesla-p100 # or nvidia-tesla-k80 etc.
```

Figure 10: k8s gpu schedule

整体架构:

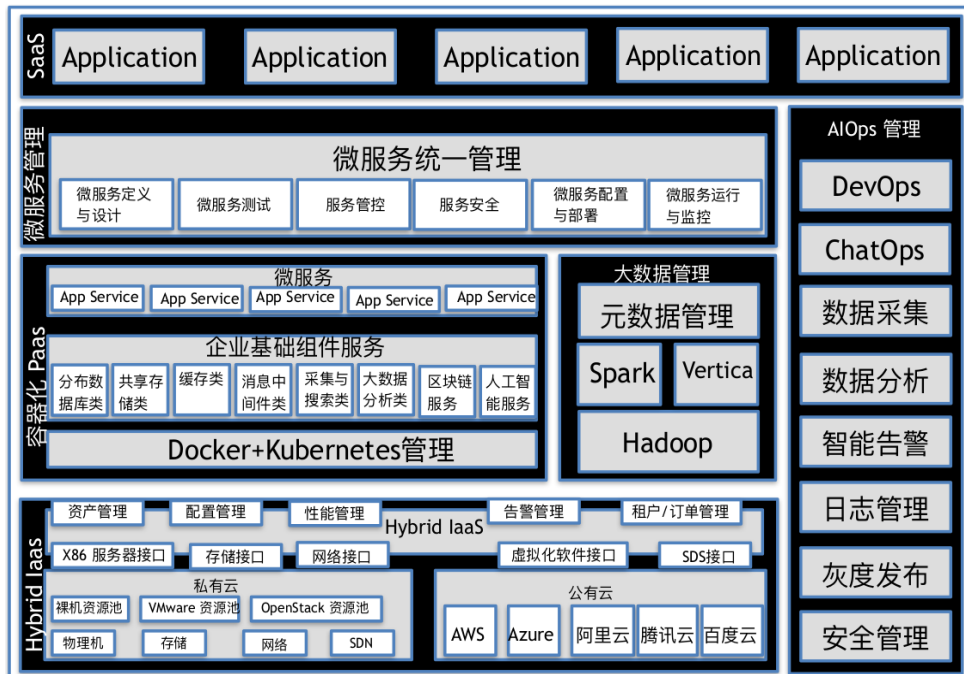


Figure 11: k8s+docker 整体架构

技术构架:

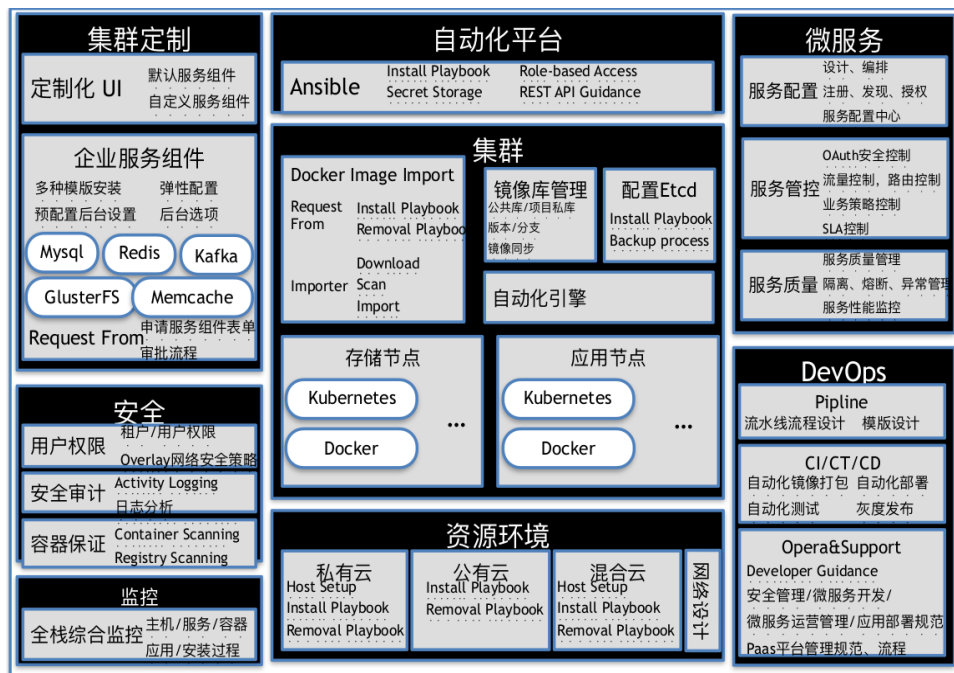


Figure 12: k8s+docker 技术架构