

Kubeflow 安利：在 Kubernetes 上进行机器学习

by 高策 — on 机器学习 (/Blog/tags/index.html#机器学习) 01 Feb 2018

这篇文章主要介绍了 Kubeflow (<https://github.com/kubeflow/kubeflow>) 的使用，以及未来的计划，面向人群为对在 Kubernetes 上运行机器学习负载感兴趣的同学。

问题背景

Kubernetes 本来是一个用来管理无状态应用的容器平台，但是在近两年，有越来越多的公司用它来运行各种各样的工作负载，尤其是机器学习炼丹。各种 AI 公司或者互联网公司的 AI 部门都会尝试在 Kubernetes 上运行 TensorFlow, Caffe, MXNet 等等分布式学习的任务，这为 Kubernetes 带来了新的挑战。

首先，分布式的机器学习任务一般会涉及参数服务器（以下称为 PS）和工作节点（以下成为 worker）两种不同的工作类型。而且不同领域的学习任务对 PS 和 worker 有不同的需求，这体现在 Kubernetes 中就是配置难的问题。以 TensorFlow 为例，TensorFlow 的分布式学习任务 (<https://www.tensorflow.org/deploy/distributed>) 通常会启动多个 PS 和多个 worker，而且在 TensorFlow 提供的最佳实践中，每个 worker 和 PS 要求传入不同的命令行参数。举例说明：

```
# On ps0.example.com:
$ python trainer.py \
  --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
  --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
  --job_name=ps --task_index=0
# On ps1.example.com:
$ python trainer.py \
  --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
  --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
  --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
  --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
  --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
  --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
  --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
  --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
  --job_name=worker --task_index=1
```

其中需要的参数有四个，一个是所有的 PS 的网络地址（主机名-端口），以及所有的 worker 的网络地址。另外是 job 的类型，分为 PS 与 worker 两种。最后是任务的 index，从 0 开始递增。因此在此例中，用户需要写至少四个 pod 的配置文件，以及四个 service 的配置文件，使得 PS 跟 worker 可以互相访问，况且这只是一个机器学习任务。如果大规模地在 Kubernetes 上运行 TensorFlow 分布式任务，可以预见繁杂的配置将成为机器学习工程师们新的负担。

其次，Kubernetes 默认的调度器对于机器学习任务的调度并不友好。如果说之前的问题只是在应用与部署阶段比较麻烦，那调度引发的资源利用率低，或者机器学习任务效率下降的问题，就格外值得关注。机器学习任务对于计算和网络的要求相对较高，一般而言所有的 worker 都会使用 GPU 进行训练，而且为了能够得到一个较好的网络支持，尽可能地同一个机器学习任务的 PS 和 worker 放在同一台机器或者网络较好的相邻机器上会降低训练所需的时间。

Hello, Kubeflow

针对这些问题，Kubeflow (<https://github.com/kubeflow/kubeflow>) 项目应运而生，它以 TensorFlow 作为第一个支持的框架，在 Kubernetes 上定义了一个新的资源类型：TFJob，即 TensorFlow Job 的缩写。通过这样一个资源类型，使用 TensorFlow 进行机器学习训练的工程师们不再需要编写繁杂的配置，只需要按照他们对业务的理解，确定 PS 与 worker 的个数以及数据与日志的输入输出，就可以进行一次训练任务。在本节中，我们将从零开始搭建一个 Kubernetes 集群，并且将 Kubeflow 运行在其上，最后利用其进行一次完整的学习任务运行。

首先，我们需要有一个正在运行的 Kubernetes 集群，而且集群的版本要大于等于 1.8。在这一步里，个人推荐以下两种方式创建一个单节点的本地 Kubernetes 集群：

使用 Kubernetes 里的 local-up-cluster.sh 脚本

- (<https://github.com/kubernetes/kubernetes/blob/master/hack/local-up-cluster.sh>)
- 使用 minikube 项目 (<https://github.com/kubernetes/minikube>)

其中前者会在本地创建一个 native 的 Kubernetes 集群，而后者则会在本地的虚拟机里创建出 Kubernetes 集群。因为本文侧重点不在此，因此整个过程不再赘述。

如果你已经成功地创建了一个 Kubernetes 集群，那么接下来就是在这集群上创建 Kubeflow 所有的组件，这一步需要用到 ksonnet (<https://github.com/ksonnet/ksonnet>)，一个简化应用在 Kubernetes 上的分发与部署的命令行工具，它会帮助你创建 Kubeflow 所需组件。在安装了 ksonnet (<https://github.com/ksonnet/ksonnet>) 后，接下来就是一片坦途了，只需要运行下面的命令，就可以完成 Kubeflow 的部署。

```
# Initialize a ksonnet APP
APP_NAME=my-kubeflow
ks init ${APP_NAME}
cd ${APP_NAME}

# Install Kubeflow components
ks registry add kubeflow github.com/kubeflow/kubeflow/tree/master/kubeflow
ks pkg install kubeflow/core
ks pkg install kubeflow/tf-serving
ks pkg install kubeflow/tf-job

# Deploy Kubeflow
NAMESPACE=default
kubectl create namespace ${NAMESPACE}
ks generate core kubeflow-core --name=kubeflow-core --namespace=${NAMESPACE}
ks apply default -c kubeflow-core
```

Kubeflow 的部署会附带一个 JupyterHub 但笔者并不知道如何使用它，因此下面的操作是用 Docker 打包训练数据和代码，用 kubectl 在 Kubernetes 上启动一次训练任务的。

示例代码可见 tf_smoke.py (https://github.com/tensorflow/k8s/blob/master/examples/tf_sample/tf_sample/tf_smoke.py)，与正常的训练代码类似，只不过 clusterspec 的传递方式是遵循了

Cloud ML (https://cloud.google.com/ml-engine/docs/trainer-considerations#use_tf_config) 的 TF_CONFIG 的方式。Kubeflow 已经根据这一训练文件打好了一个 Docker 镜像：gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff，在这里直接使用就好：

```
kubectl create -f https://raw.githubusercontent.com/tensorflow/k8s/master/examples/tf_job.yaml
```

Kubeflow 实现介绍

本部分主要涉及对 Kubeflow 内部实现的介绍和未来可能的开发计划，如果不感兴趣可以就此打住 :)

对分布式训练任务的支持

为了解决配置困难的问题，Kubeflow 以 TensorFlow 作为第一个支持的框架，为其实现了一个在 Kubernetes 上的 operator (<https://coreos.com/operators/>)：tensorflow/k8s (<https://github.com/tensorflow/k8s>)。由于在 Kubernetes 上内置的资源类型，如 deployment, replicaset，或者是 pod 等，都很难能够简练而清晰地描述一个分布式机器学习的任务，因此我们利用 Kubernetes 的 Custom Resource Definition (<https://kubernetes.io/docs/concepts/api-extension/custom-resources/>) 特性，定义了一个新的资源类型：TFJob，即 TensorFlow Job 的缩写。一个 TFJob 配置示例如下所示：

```
apiVersion: "kubeflow.org/v1alpha1"
kind: "TFJob"
metadata:
  name: "example-job"
spec:
  replicaSpecs:
    - replicas: 1
      tfReplicaType: MASTER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              restartPolicy: OnFailure
    - replicas: 1
      tfReplicaType: WORKER
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              restartPolicy: OnFailure
    - replicas: 2
      tfReplicaType: PS
      template:
        spec:
          containers:
            - image: gcr.io/tf-on-k8s-dogfood/tf_sample:dc944ff
              name: tensorflow
              restartPolicy: OnFailure
```

其中每个字段就不多介绍了，这里主要是说一下实现。任何一个 PS 或者 worker，都由两个资源组成，分别是 job 和 service。其中 job 负责创建出 PS 或者 worker 的 pod，而 service 负责将其暴露出来。这里社区目前也在重新考虑选型，目前希望可以直接创建 pod 而非 job，而用 headless service 替代 service，因为 PS worker 不需要暴露给除了该分布式学习任务外的其他服务。

TFJob operator 的实现早期是从 etcd-operator (<https://github.com/coreos/etcd-operator/>) 复制来的，因此整体的架构在最初是完全仿照其改写而成。在最初的实现中，当有一个 TFJob 被创建时，在 operator 内都会有一个新的 goroutine，以轮询的方式获取 TFJob 的状态，然后基于此状态做出相应的操作，相当于是 operator 内部维护了一个状态机。这样的方式会有一些缺点：

- 这样的架构使得 operator 是有状态的，使得状态很难横向扩展
- 维护基于 Phase 的状态机是 Kubernetes 社区不推崇的一种方式

基于这些问题，operator 的架构正在往事件驱动重构，这部分工作由 @caicloud (<https://github.com/caicloud>) 在推进。重构之后，operator 会在 Kubernetes 的一些资源上注册 informer 的事件回调，比较现在的状态与理想状态的不同而采取相应的操作。比如当有一个新的 TFJob 被创建时，理想状态是所有对应的 PS，worker 都被创建好，而当下的状态则是没有任何 pod 和 service 被创建，此时 operator 会创建出对应的 PS，worker 的 pod 和 service，以达到理想状态，这也是 Kubernetes 社区对于 operator/controller 的最佳实践。

对分布式学习任务效率的关注

目前社区还停留在如何对 AI 工程师更友好，更好地维护上面提到的 operator 这一步，在效率方面考虑地较少。目前有利用 kube-arbitrator (<https://github.com/kubernetes-incubator/kube-arbitrator>) 来进行 gang scheduling 的探索，目前还没有尝试过因此不好评价。但是整体来说 Kubeflow 的性能提高还有很大的空间。

因为机器学习任务根据模型的不同，其输入数据的规模，特征，模型的大小等等都有很大不同。比如 CV 领域与推荐领域的学习模型就有完全不同的特点，因此 TensorFlow 的分布式模型提供了极强的灵活性。而对于 Kubernetes 而言，如何能够在保持灵活性的基础上，同时也保证任务在较高的性能下运行，同时集群的利用率也相对较高，是一个值得研究的问题。

对其他机器学习框架的支持

目前 Kubeflow 主要关注 TensorFlow，而其他机器学习框架的支持将于之后展开，目前有一些第三方实现的 operator，比如 MXNet operator (<https://github.com/deepinsight/mxnet-operator>)，但是质量难以保证。

开发情况与未来展望

目前 Kubeflow 有来自 Google，Caicloud，RedHat 等公司的积极参与，短期的目标有这么几个：

- operator 方面
 - 使用 pod 替换 job tensorflow/k8s#325 (<https://github.com/tensorflow/k8s/issues/325>)
 - 使用 headless service 替换 service tensorflow/k8s#40 (<https://github.com/tensorflow/k8s/issues/40>)

- 由 etcd operator 主动轮询的方式改为事件驱动
tensorflow/k8s#314 (<https://github.com/tensorflow/k8s/issues/314>)
- 分离对 TensorBoard 的支持 tensorflow/k8s#347 (<https://github.com/tensorflow/k8s/issues/347>)
- 支持细粒度的任务状态 tensorflow/k8s#333 (<https://github.com/tensorflow/k8s/issues/333>)
- 模型服务方面
 - GPU 支持 kubeflow/kubeflow#64 (<https://github.com/kubeflow/kubeflow/issues/64>)
 - 监控支持 kubeflow/kubeflow#64 (<https://github.com/kubeflow/kubeflow/issues/64>)
 - 多框架支持下的统一 API 支持 kubeflow/kubeflow#102 (<https://github.com/kubeflow/kubeflow/issues/102>)
- UI 方面
 - 为各个部件支持统一的 UI kubeflow/kubeflow#199 (<https://github.com/kubeflow/kubeflow/issues/199>)

目前 Kubeflow 在 GitHub 上有 2400 多个 star，有 40 个左右的贡献者。其长期的目标是成为 CNCF 的一个项目，目前实现仍存在很多问题，窃以为也并不是 production ready 的状态，但它仍然值得一试。

关于作者

@gaocegege (<https://github.com/gaocegege>)，上海交通大学软件学院研究生在读，
Kubeflow (<https://github.com/kubeflow/kubeflow>) Member，寻找 2018 暑期实习。

License

- This article is licensed under CC BY-NC-SA 3.0 (<https://creativecommons.org/licenses/by-nc-sa/3.0/>).
- Please contact me for commercial use.

 (<http://widget.renren.com/dialog/share?link=http://gaocegege.com/Blog/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/kubeflow&u=/Blog/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/kubeflow>)
 (<https://www.facebook.com/sharer/sharer.php?url=/Blog/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/kubeflow>)
 (<https://plus.google.com/share?url=/Blog/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/kubeflow>)

评论



Author






高策

gaocegege@hotmail.com (<mailto:gaocegege@hotmail.com>)

Friends

女朋友肉肉 (<http://blog.xuruowei.com/>) | 刁宝乐 (<http://blog.weiyiminhengqiang.com/>) | 学帅 | 飞龙 (<http://www.flygon.net/>) | Azard (<http://azard.me/blog>) | hcz (<http://hczhcz.github.io/>) | 姜轶 (<http://zhangwenli.com/blog/>) | zhou (<http://s684373.github.io/>) | consiii (<http://consiii.me/>) | arrowrowe (<http://arrowrowe.me/#/home>) | at15 (<http://at15.github.io/pvquo/#/>) | Sha Li (<https://raspberrice.github.io/>) | 政子哥哥 (<http://blog.zhengzi.me/>) | 易极 (<http://yicodes.com/>) | codeworm96 (<http://codeworm96.github.io/>) | 考拉 (<http://ikoala.net/>)

All content copyright 高策 ([/Blog/about](#)) © 2020 • All rights reserved.
Proudly published with Jekyll (<https://jekyllrb.com/>)

 (<http://www.renren.com/325460067/profile>)  (<http://weibo.com/constructcece>)  (<https://plus.google.com/u/0/109707369864071328013>) 
(<http://github.com/gaocegege>)  (<http://facebook.com/gaocegege>)