基于 Kubernetes 的分布式 TensorFlow 平台的设计与实现

余昌发 程学林 杨小虎
(浙江大学软件学院 杭州 310027)

摘 要 文中介绍了基于 Kubernetes 的分布式 TensorFlow 平台的设计与实现,针对分布式 TensorFlow 存在的环境配置复杂、底层物理资源分布不均、训练效率过低、模型研发周期长等问题,提出了一种容器化 TensorFlow 的方法,并基于 Kubernetes 容器 PaaS 平台来统一调度管理 TensorFlow 容器。文中将 Kubernetes 和 TensorFlow 的优点相结合,由 Kubernetes 提供可靠、稳定的计算环境,以充分发挥 TensorFlow 异构的优势,极大地降低了大规模使用的难度,同时建立了一个敏捷的管理平台,实现了分布式 TensorFlow 资源的快速分配、一键部署、秒级启动、动态伸缩、高效训练等。

关键词 TensorFlow,Kubernetes,Docker,深度学习
中图法分类号 TP311 **文献标识码** A

Design and Implementation of Distributed TensorFlow Platform Based on Kubernetes

YU Chang-fa CHEN Xue-lin YANG Xiao-hu
(School of Software Technology, Zhejiang University, Hangzhou 310027, China)

Abstract This paper designed and implemented a distributed deep learning platform based on Kubernetes. In order to solve the propblems of complex environment configuration of distributed TensorFlow, uneven distribution of underlying physical resources, low efficiency of training model and long development cycle, a method of containerized TensorFlow based on Kubernetes was proposed. By combining the advantages of Kubernetes and TensorFlow, Kubernetes provides a stable and reliable computing environment and gives full play to the advantages of heterogeneous TensorFlow, which greatly reduces the difficulty in large-scale use. Meanwhile, an agile management platform is established, which realizes the fast distribution of distributed TensorFlow resources, one key deployment, second level running, dynamic expansion, efficient training and so on.

Keywords TensorFlow, Kubernetes, Docker, Deep learning

1 引言

随着人工智能的快速发展及兴起,深度学习被广泛应用到日常生活中。这使得很多企业在构建深度学习模型时越来越关注计算资源的快速分配及模型的快速训练,从而提高自身的资源利用率和训练质量。而目前使用 TensorFlow 的门槛较高,尤其是对于分布式 TensorFlow 的计算[1],繁琐且复杂的配置给很多非研发人员带来了巨大的挑战。

Kubernetes 是基于容器技术的分布式云平台架构[2],它是目前最为流行的平台即服务(Platform-as-a-Service, PaaS)平台之一,支持目前最为流行的 Docker 技术[3]。Kubernetes 具有资源管理自动化、资源利用率最大化、微服务架构等优点,与传统的虚拟化技术相比,性能更高,速度更快,成本更低。

TensorFlow 是一款优秀的深度学习框架,异构设备分布式计算是其一大亮点。TensorFlow 支持卷积神经网络、循环神经网络、Docker 运行、自动微分及大多数算法库等特性[4]。

本文针对目前企业在大规模使用分布式 TensorFlow 时无法实现对底层资源的快速分配、简化复杂的 TensorFlow 配置、模型的快速构建等问题,提出了容器化 TensorFlow 并将其运行在 Kubernetes PaaS 云平台上的解决方案。本文还详细介绍了基于 Kubernetes 的分布式 TensorFlow 平台的设计实现,通过该平台实现了分布式 TensorFlow 环境的快速构建,统一分配有限的物理资源,TensorFlow 容器的秒级部署,极大地提高了模型开发的效率,降低了使用分布式 TensorFlow 的门槛,彻底隔离了平台用户与底层环境的交互,最终实现了模型的快速构建、模型的高效训练、模型的一键托管等特性。

2 架构设计

2.1 系统的整体架构

基于 Kubernetes 的分布式深度学习平台系统的整体架

本文受中央高校基本科研业务费专项资金,国家科技支撑计划:公共文化科技服务能力建设与绩效评估技术研究与示范(2015BAK26B00)资助。
余昌发(1992—),男,硕士,主要研究方向为容器云与深度学习,E-mail:yu_changfa@126.com;**程学林**(1966—),男,硕士,高级工程师,主要研究方向为数据可视化分析,E-mail:cxlin@zju.edu.cn;**杨小虎**(1976—),男,博士,研究员,主要研究方向为软件工程、云计算、金融信息技术,E-mail:yangxh@zju.edu.cn。

构设计如图 1 所示。

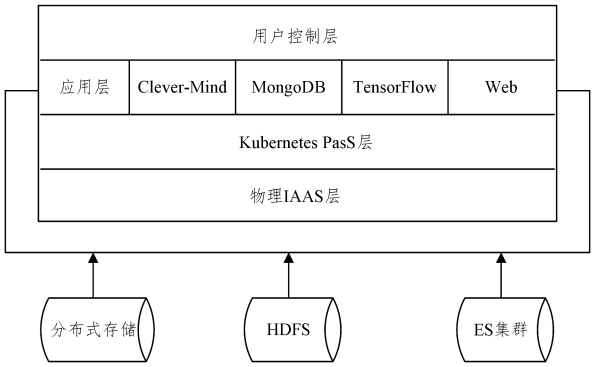

图 1 平台系统架构设计图

系统整体分为物理资源 IaaS 层、Kubernetes PaaS 层、应用层以及用户层。

物理资源 IaaS 层主要是底层的机器资源，这些资源包括物理机、虚拟机、云主机等。通常企业都会拥有自己的 IaaS 层，目前较为主流的 IaaS 层平台有 Openstack，Vmware 等。

Kubernetes PaaS 层主要是由 Kubernetes 构建的 PaaS 云平台，PaaS 平台将会掌控所有应用层的应用，如镜像仓库(harbor)、Clever-Mind(分布式 TensorFlow 平台服务)、TensorFlow、MongoDB、TensorFlow 深度学习平台的 Web 前端等应用都会被容器化并且部署在 PaaS 平台上，应用的生命周期、资源分配等也都由平台统一控制管理。

应用层主要是部署平台的应用程序，当然也可以部署非深度学习平台相关的其他应用，只要该应用已经被容器化并且已被上传(Push)到镜像仓库中，就可以被部署到 PaaS 平台上。

用户控制层主要是对用户的权限控制，提供一个统一的登入注册服务，用户在使用整个平台时只需要验证一次即可使用 PaaS 平台的所有服务，包括分布式 TensorFlow 深度学习平台的服务 Clever-Mind。Clever-Mind 是本文的一个核心服务，它主要对外提供创建及删除 TensorFlow 容器、训练日志、TensorBoard、TensorFlow Serving、TensorFlow 任务详情等服务。创建的 TensorFlow 任务类型有参数服务器(简称 PS)和工作节点(简称 Worker)[5]。

为了保证 Kubernetes 集群的高可用性，采用了 ETCD 集群[6]，ETCD 是一款基于键值对的高性能的存储系统，由于 Kubernetes 会将所有的元数据信息存入 ETCD 中，若是单节点的 ETCD 则无法满足主备需求。采用 ETCD 集群可以有效地防止由于 ETCD 宕机所带来的影响。API-Server 是调用 Kubernetes API 的核心组件，这对 Clever-Mind 来说是相当重要的，因为 Clever-Mind 会调用 Kubernetes 中的 API-Server 来创建 TensorFlow，TensorBoard，TensorFlow Serving 等服务[7]。分布式镜像仓库也部署在 Kubernetes 平台上，Kubernetes 是整个平台的控制中枢。

2.2 Kubernetes 架构

本文采用了 Kubernetes1.7 的版本，并且构建了两个 Kubernetes 集群，在 Kubernetes 平台上运行了大量的容器，效果图如图 2 所示。Kubernetes 架构图如图 3 所示，其中 Pod 是它的基本运行单位，每个 Pod 中可以包含多个 Docker 容器，它们之间共享资源，但是在本文中每个 Pod 只会运行一个 Docker。

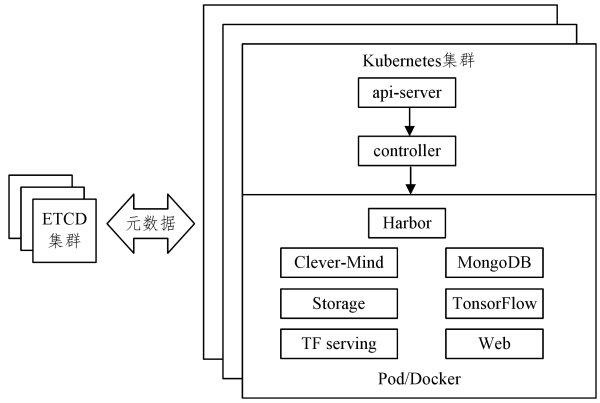

图 2 Kubernetes 集群图

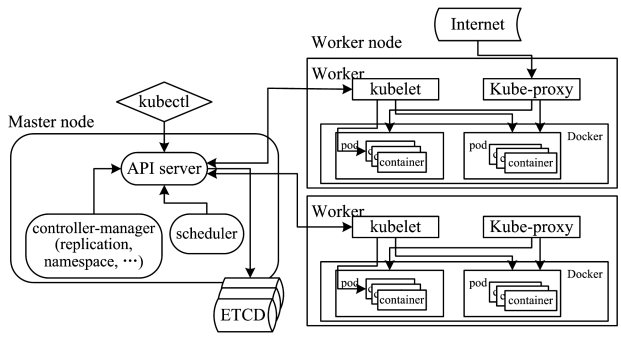

图 3 Kubernetes 架构图

Job 是 Kubernetes 的一种资源对象，它主要用来控制批任务，使用 Job 可以保证这个 Pod 一直处于运行状态，中间如果遇到失败会自己重启，从而保证 Pod 的 Running 状态。

DaemonSet 可以保证在每台 Kubernetes 机器的工作节点上至少运行一个这类任务的 Pod，运行节点也可以通过 label 也即 nodeSelector 来指定特定的机器节点，这样可以保证一些核心的服务在特定的节点上都运行。

在 Kubernetes1.5 版本后将 PetSet 更名为 StatefulSet，这主要是为了提供一些有状态的服务，因为之前提到的 Replication Controller，Replica Sets 都是无状态的，这些 Pod 的名字都是由 Kubernetes 随机生成的，Pod 每次重启都会改变，并且这类 Pod 一般都不挂载存储系统。可以说 StatefulSet 是针对有状态的服务而设计的，由 StatefulSet 启动的 Pod 都会保持同样的名字，不管 Pod 重启了多少次都不会改变，这个特性可以更好地保证服务状态的连续性，从而提高服务的高可用性。

Federation 是 Kubernetes 跨集群的一个重要特性，中文称为集群联邦。通常情况下，主机会分布在不同的地域、不同的可用区、不同的云服务提供商或者一些本地的机器。Kubernetes 目前也在逐渐完善这个特性，而跨集群需要良好的网络，这样才能满足 Kubernetes 快速地分配调度，从而满足存储和计算性能的需求。目前，Kubernetes 也为 Federation 提供了专属的 API，很多企业也正在大胆尝试这一特性。本文所构建的两个 Kubernetes 集群均在同网段，并未使用到 Federation 这一特性。

2.3 Clever-Mind 功能模块的总体设计

Clever-Mind 是用户与 TensorFlow 交互的唯一渠道，平台用户通过平台门户 Web 前端提交相应的任务详情，Clever-Mind 会根据用户提供的信息去调用 Kubernetes 中的 API-

Server，并将整个创建的过程及数据信息存储在 MongoDB 中，为用户实时返回任务信息，调用图如图 4 所示。

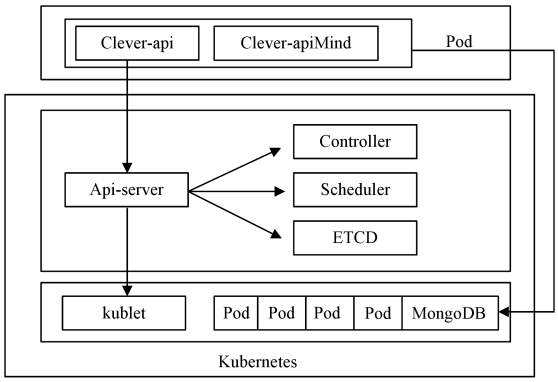

图 4　Clever-Mind 调用图

详细过程如下：Clever-Mind 会被容器化并部署到 Kubernetes PaaS 平台中，它在创建或删除 TensorFlow 任务时会调用 Kubernetes 的 API-Server；然后由 Controller Manager 和 Scheduler 通过 Kubelet 来调度编排 TensorFlow 容器，Clever-Mind 中的 Clever-API 将提供创建/删除分布式 TensorFlow，获取指定用户所有分布式 TensorFlow 列表和某个分布式 TensorFlow 的详细信息；查看分布式 TensorFlow 的状态，获取分布式 TensorFlow 指定 Worker 的日志信息等。

2.4　Clever-Mind API 的设计及开发

目前在软件开发领域，主流的 API 开发都是符合 Restful 规范的，所有 Clever-Mind API 的开发也是 Restful 风格。整个平台的开发语言为 Golang，Golang 是 Google 公司开源的优秀编程语言，超高的并发性是其一大亮点，目前在云计算及大数据领域都有广泛的应用，Docker 及 Kubernetes 的开发语言也都是由 Golang 来实现的。使用 Golang 还可以更好地将 Clever-Mind 分布式深度学习平台与 Kubernetes 相结合，方便底层接口的调用。

实现 Restful API 时所采用的框架为 emicklei/go-restful，这是一款开源的框架，代码在 Github 上托管[8]。设计的 API 如表 1 所列。

表 1　Clever-Mind API 设计详情

资源	方法
创建分布式 TensorFlow 应用/api/v1/tfApps?uid={UID}	POST
获取指定用户所有分布式 TensorFlow 应用列表/api/v1/tfApps?uid={UID}&cid={CID}&start={start}&limit={limit}	GET DELETE
获取某个分布式 TensorFlow 应用详情/api/v1/tfApps/{ID}	GET
分布式 TensorFlow 应用的生命周期/api/1/tfApps/{ID}/stage	GET
查询分布式 TensorFlow 任务指定的 worker 日志信息/api/v1/tfApps/{ID}/workers/{worker_addr}/logs	GET
删除分布式 TensorFlow 应用/api/v1/tfApps/{ID}	DELETE
TensorFlow 任务状态实时信息/api/v1/tfTraining/{userId}/jobs/{jobId}/status	POST

2.5　平台生态

由图 1 可以清晰地了解整个平台的架构，在平台外还有 Hadoop File System[9]（简 DFS）、Elastic Search[10]（ES）、高可用分布式存储等。HDFS 及 ES 都是分布式 TensorFlow 平台重要的数据来源，只需要打通各集群与 Kubernetes 集群之间的网络就可实现在 PaaS 平台上使用这些外部数据。而高可用的分布式存储（Storage）是 PaaS 平台提供给平台用户的私有存储，其主要作用是使用户上传 TensorFlow 程序代码、TensorFlow 模型的导入导出、训练日志的存储等，这里对 Storage 的实现采用了 GlusterFS[11]，是一款高性能的可靠存储系统，是一套开源的文件存储系统，具有良好的扩展性和优秀的存储能力，并且 Kubernetes 对其有着较好的兼容性。GlusterFS 可以将底层闲置的存储空间都集中起来统一提供存储服务，提高了存储资源的利用率，可通过 TCP 网络协议的方式来管理这些存储资源。Kubernetes 能够较好地兼容 GlusterFS。Heketi 是一个基于 RESTful API 的 GlusterFS 卷管理框架[12]。因为 Heketi 本身提供了 Rest 风格的 API，所以 Kubernetes 可以很方便地调用该服务，即使是不同的 GlusterFS 集群也能胜任，它会将存储副本均匀地分布到不同的 GlusterFS 存储集群上，提高了存储的可用性。相比于 NFS[13]，GlusterFS 在容器方案中更加稳定，NFS 在 Kubernetes 中会经常出现无法挂载的问题，该问题在社区中也是热烈讨论的话题之一。经过实践，在 GlusterFS 及 NFS 两套方案中最终选择了 GlusterFS。

2.6　Clever-Mind 平台规范的设计实现

制定平台的统一标准是十分有必要的，如果不这么做，将无法控制用户的行为，无法定位到用户的代码路径，这对于很多用户导入的模块将存在异常，这也违背了平台设计的初衷。为了避免用户每次提交代码时都构建一个新的镜像，减少维护镜像的成本，平台需要制定相应的规范，在一个整体框架中去实现 TensorFlow 应用。这样在代码上虽然会对用户产生一定的限制，但是整体代码的实现逻辑并没有改变，只是将其细分为几个框架中的模块，便于平台的统一管理，也就是说平台的 TensorFlow 镜像是基于不同版本的 TensorFlow 构建的，这些不同版本的镜像能够满足用户在不同 TensorFlow 版本上代码的需求，并且适用于平台的所有用户，最终用户只是上传已编写好的 TensorFlow 代码就可以在平台上进行高效的训练。

目前最流行的 TensorFlow 编程语言是 Python，用 Python 来实现一个 TensorFlow 应用显得十分简单，对此，平台提供了一个统一的 Python 入口。在平台的这个基类中会定义一些系统的 FLAG，但系统的 FLAG 会比较复杂，这样做是为了避免与用户定义的 FLAG 冲突。为了友好地展示 TensorFlow 任务的实时状态，需要调用表 1 中的 tfTraining API，在基类中每隔一段时间将当前最新的信息发回 Clever-Mind 中，这些数据包含开始训练时间、当前训练时间、当前步数、总步数、当前状态，Clever-Mind 会根据接收到的数据来动态地计算任务预计完成的时间，并且计算出当前运行的百分比，前端 Web 服务可以很友好地向用户展示这些信息。对于用户而言，用户的代码只需要继承平台提供的基类即可，这里会有一些基本的运行参数，如训练步长、是否使用 GPU、同步方式还是异步方式、日志存储目录、模型 checkpoints 的时间等。用户在重写代码时只需要重写模型构建（build_model）、模型训练（train）、模型预测或其他训练后的工作（after_train）这三大基本模块，其他将由平台来完成，用户无需关心。

2.7　模型托管（TensorFlow Serving）

模型托管是基于 TensorFlow Serving 来实现的，它也是由 Google 主导的一个开源项目，最终是将训练好的模型以

gRPC 的形式对外提供服务，这也是 TensorFlow 模型 Serving 的一种方式，它可以减少整个研发上线的周期，并且支持多种模型的 Serving 功能，不仅仅局限于 TensorFlow 模型，用户也可以将模型导出，根据自己的业务需求来构建自己的模型服务。

模型开发人员通过 TensorFlow 进行模型的构建工作，通过层层的神经网络训练后得到模型。经验证后，该训练模型便会被发布到 Kubernetes 上进行模型托管，也即通过 TensorFlow Serving，用户只需要自己编写客户端，通过 gRPC 来调用该模型服务即可获取相应的结果[14]。

整个 Serving 的工作也都将以容器的形式通过 Kubernetes PaaS 平台来为用户提供模型服务，容器化的过程如下：以 Ubuntu16.04 为基础镜像，安装必要的环境，有 Bazel、python、编译所需要的依赖包等，在构建过程中需要去外网拉一些依赖包。为用户构建模型服务时还需要将用户训练好的模型打包至镜像中，最后用 Kubernetes 创建 Serving 服务，为用户提供 gRPC 的服务。

2.8 Clever-Mind 服务

Clever-Mind 最终会被容器化并部署在 Kubernetes 平台上，为用户提供一个 Web 管理界面，数据均通过 Clever-Mind 的服务来提供，流程如下：若平台用户没有创建存储，则通过 Web 管理界面来创建自己的私有存储；上传已编写好的 TensorFlow 代码，选择一些环境参数，如 CPU、GPU、内存、日志路径、代码路径、自定义 FLAG 等；最后创建任务。在模型训练结束后还可以选择 TensorFlow Serving（模型托管）服务，平台会为每个模型服务提供一个专用的平台 IP，用户只需要自己编写客户端通过 gRPC 来进行调用即可。

3 系统运行效率分析

3.1 Kubernetes PaaS 平台效率分析

系统使用的环境如表 2 所列。

表 2 环境版本列表

名称	版本
Kubernetes	v1.7
Docker	17.09.0-ce
Golang	1.8.5
TensorFlow	r1.0，r1.1，r1.2，r1.3，r1.4

在系统开发完成后，首先对 Kubernetes PaaS 平台进行使用评估。目前的平台由两个 Kubernetes 集群组成，一个是由 20 台虚拟机所构成的 CPU 集群，另一个是由 3 台物理机所构成的 GPU 集群，系统均为 CentOS7.2，两个集群的机器配置如表 3 所列。

表 3 Kubernetes 集群配置表

硬件资源	CPU 集群	GPU 集群
CPU	8 core	20 core，Xeon E5-2640(v4) 2.4GHz
内存	32GB	256GB
磁盘	1T	2 * 800G SSD，6 * 4T
GPU	无	NVIDIA Tesla P100(16GB) * 2

由于测试过程中未使用 GPU，因此本文的数据都是基于虚拟机的 CPU 集群。这里首先测试 Kubernetes 集群的容器启动速度，测试的镜像均来自公有云上的官方版本，选取了其中几个较为热门的镜像作为测试镜像，单一容器测试列表如表 4 所列。

表 4 单一容器启动速度测试表

（单位：s）

镜像名称	耗时
MongoDB:3.6.0	2.3
Nginx:1.13.7	2.2
Tomcat:8.5.24-jre8	2.3
Mysql:5.7	2.6

表 4 列出了随机启动容器的时间花费，每次启动可能会略有波动，测试的容器都没有 ResourceQuota（资源配额）的限制，Kubernetes 集群上没有运行其他非测试容器。启动时间是指容器由 Penning 状态转变为 Running 状态所花费的时间。由表 4 可知，5 个测试容器都正常，达到预期结果，可以进行正常的读写操作。接下来把表 4 中的每个镜像都启动 100 个容器，容器的所有参数都相同，只是容器的名称不同，这些名称都由一个测试的脚本随机生成，测试结果如表 5 所列。

表 5 100 个容器启动速度测试表

（单位：s）

镜像名称	最短耗时	最长耗时	平均耗时
MongoDB:3.6.0	1.8	2.8	2.3
Nginx 1.13.7	1.5	2.6	2.1
Tomcat:8.5.24-jre8	1.7	2.6	2.3
Mysql:5.7	1.7	2.9	2.4

表 5 中测试的方式没有直接将 Replica 的数量调整至 100，而是用脚本生成不同 name 的 yaml 文件，这是因为深度学习平台没有使用 Replica 这个方式来实现分布式 TensorFlow。为了保证测试与深度学习平台的高度统一，此处测试方式选用的也是创建基于同一镜像的不同容器。通过表 4 和表 5 中的结果可以发现，随着容器数量的增加，容器的启动速度并没有下降，整体的平均时间都控制在 3s 以内，真正实现了秒级部署，高效的并行创建速度为分布式 TensorFlow 提供了良好的技术支撑。

接下来进一步进行压测，之前测试的反馈情况相当优异，因此这次将直接把容器数量提升至 1000，也就是说在表 5 测试数量的基础上再增加 10 倍。操作过程和表 5 保持一致，唯一不同的是增加了生成的数量，但遗憾的是此次测试未能成功，有一部分容器没能成功 Running，查看日志发现抛出了 OOM 异常，通过 IaaS 云平台查看 20 台虚拟机的硬件状态，发现可用内存几乎为零，达到了硬件瓶颈，本轮测试结束。

3.2 Clever-Mind 创建 TensorFlow 的效率分析

通过 Clever-Mind 前端可视化 Web 界面进行任务的创建，与表 4 一样，通过一个程序来记录容器从创建 Pending 到 Running 所花费的时间，这里使用的 TensorFlow 镜像版本均为 r1.0，第一轮仅测试一个容器，前端页面选择资源为 512M，1 核，PS 和 worker 的数量均为一个，选择 Mnist 样例，Mnist 是一个著名的手写图片数据集[15]，TensorFlow 官方公开了对该数据集的模型实现，Clever-Mind 平台改写了官方的代码，大部分与官方一致，只是用平台的基类复写了一遍，其他参数均保持默认，测试结果如表 6 所列。

表 6 Clever-Mind 创建单一任务测试表

（单位：s）

TensorFlow 任务类型	耗时
PS	4.3
Worker	4.1

由表 6 中的测试数据可知，Clever-Mind 创建单一任务测试的时间与表 4 中 Kubernetes 创建容器的时间相比要长得多。为了找出耗时增加的原因，这里直接通过 Kubernetes 来创建相同的 TensorFlow 任务，经过多次测试发现，直接通过 Kubernetes 创建 TensorFlow 任务比 Clever-Mind 要快大约 700 ms，其实这是合理的，因为用户的请求需要经过 Clever-Mind 处理后才调用 Kubernetes API-Server 来创建任务，但是这与表 4 中的数据还是有较大出入，因此需要深入挖掘和分析。在创建任务时与表 4 中的唯一区别就是挂载了用户的 Storage，为了排除这个干扰项，在通过 Kubernetes 创建任务时不挂载任何目录，因为样例代码本身就存在于已构建好的 TensorFlow 镜像中，所以这里不挂载目录也不会有任何影响。在经过多次创建任务的测试后发现创建任务的速度有所提升，提升大约为 800 ms，虽然与表 4 中的结果还存在一点差距，但已相当接近。

在下一轮测试中，将 PS 和 Worker 的数量提升至 100，采用的方法很简单，只需要改变前端页面 PS 和 Worker 的数量即可，其他参数保持默认，测试结果如表 7 所列。

表 7　Clever-Mind 创建 100 个任务测试表

（单位：s）

TensorFlow 任务类型	最短耗时	最长耗时	平均耗时
PS	2.4	6.2	4.5
Worker	2.5	6.6	4.7

表 7 的数据表明，即使在创建 100 个容器的情况下，整体的耗时都是非常可观的。对于平台用户来说，简便地操作分布式 TensorFlow 是十分重要的，而快速分配资源、创建 TensorFlow 实例、高速训练则是 Kubernetes 分布式 TensorFlow 深度学习平台的优势所在。

结束语　本文主要围绕 Kubernetes 及 TensorFlow 这两个新兴的云计算及大数据框架展开研究，针对目前企业在使用分布式 TensorFlow 时无法高效地利用物理资源，并且工作效率过低，造成企业成本上升等问题，提出了容器技术的解决方案。传统的深度学习方式已经无法满足目前人工智能浪潮所带来的经济效益，高效、精确的构建及训练模型是深度学习最重要的核心步骤。因此，在对 TensorFlow 的使用现状进行了深入的分析及研究后总结得出，Kubernetes PaaS 云平台可以较好地帮助企业充分利用底层资源；而容器化 TensorFlow 可以使 TensorFlow 运行在 PaaS 平台上，最终实现在数秒内为开发人员提供上百个 TensorFlow 的分布式集群，从而达到快速构建深度学习环境的目的，极大提高了开发人员的效率。

最终测试创建 TensorFlow 容器所消耗的时间依然还有很大的优化空间，下一步将优化存储与 Kubernetes 集群之间的网络通信，并且尝试其他高性能存储系统，减少挂载带来的性能影响。

参 考 文 献

[1] ABADI M，AGARWAL A，BARHAM P，et al. TensorFlow：Large-Scale Machine Learning on Heterogeneous Distributed System[J]. arXiv：1603.04467v2，2016.

[2] 龚正，吴治辉，王伟，等. Kubernetes 权威指南：从 Docker 到 Kubernetes 实践全接触（纪念版）[M]. 北京：电子工业版社，2017：1-42.

[3] 浙江大学 SEL 实验室. Docker 容器与容器云[M]. 北京：人民邮电出版社，2016：1-27.

[4] 李航. 统计学习方法 [M]. 北京：清华大学出版社，2012：1-24.

[5] 李嘉璇. TensorFlow 技术解析与实战[M]. 北京：人民邮电出版社，2017：218-224.

[6] PEINL R，HOLZSCHUHER A F，PFITZER F. Docker Cluster Management for the Cloud-Survey Results and Own Solution [J]. Grid Computing，2016，14：265-282.

[7] Serving a TensorFlow Model[EB/OL]. https://www.tensorflow.org/serving/serving_basic.

[8] go-restful[EB/OL]. https://github.com/emicklei/go-restful.

[9] CHANG F，DEAN J，GHEMAWAT S，et al. Gruber. Bigtable：A Distributed Storage System for Structured Data[J]. ACM Transactions on Computer Systems (TOCS)，2008，26(2)：1-26.

[10] 朱林. Elasticsearch 技术解析与实战[M]. 北京：机械工业出版社，2017：6-10.

[11] https://github.com/kubernetes/examples/blob/master/staging/volumes/glusterfs/README.md.

[12] https://github.com/heketi/heketi.

[13] https://en.wikipedia.org/wiki/Network_File_System.

[14] SEYMOUR K，NAKADA H，MATSUOKA S，et al. Overview of GridRPC：A Remote Procedure Call API for Grid Computing [J]. Grid Computing，2002，2536：274-278.

[15] http://yann.lecun.com/exdb/mnist.

（上接第 504 页）

[5] BAJTA M E，IDRI A，FERNÁNDEZ-ALEMÁN J L，et al. Software cost estimation for global software development a systematic map and review study[C]// International Conference on Evaluation of Novel Approaches To Software Engineering. IEEE，2015：197-206.

[6] 杨抒，王业，乌尔柯西，等. 基于 C&S-PSO 的软件成本估算类比法特征权重优化[J]. 计算机系统应用，2015，24(7)：99-103.

[7] PAPATHEOCHAROUS E，ANDREOU A S. On the Problem of Attribute Selection for Software Cost Estimation：Input Backward Elimination Using Artificial Neural Networks[C]// Artificial Intelligence Applications and Innovations，Ifip Wg 12.5 International Conference. Springer Berlin Heidelberg，2010：287-294.

[8] 吴登生，李建平，蔡晨. 软件成本估算的粒子群算法类比模型及自助法推断[J]. 管理科学，2010，23(3)：113-120.

[9] DIZAJI Z A，KHALILPOUR K. Particle Swarm Optimization and Chaos Theory Based Approach for Software Cost Estimation[J]. International Journal of Academic Research，2014，6(3)：130.

[10] 王振丽. 基于加权 MP 马氏距离的 GS 方法研究[D]. 南京：南京理工大学，2016.

[11] KENNEDY J，EBERHART R. Particle swarm optimization[C]// IEEE International Conference on Neural Networks. IEEE，1995：1942-1948.

[12] 牛利勇，张帝，王晓峰，等. 基于自适应变异粒子群算法的电动出租车充电引导[J]. 电网技术，2015，39(1)：63-68.

[13] DESHARNAIS J M. Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction[D]. Quebec：University of Montreal，1989.

[14] 中华人民共和国工业和信息化部. 软件研发成本度量规范：SJ/T 11463-2013[S]. 2013.