文章编号：2096-1472(2021)-05-22-04
DOI:10.19644/j.cnki.issn2096-1472.2021.05.006

基于微服务的深度学习模型服务系统

江宁远，张华熊

(浙江理工大学信息学院，浙江 杭州 310018)
✉240662137@qq.com; zhxhz@zstu.edu.cn

摘　要：Tensorflow Serving是Google开源的一个服务系统，针对Tensorflow Serving单体应用吞吐量低、服务调用烦琐、模型生命周期管理不完善等问题，本文设计了一种基于Tensorflow Serving的微服务软件架构方案，在部署Tensorflow Serving的Docker(开源的应用容器引擎)容器里添加本文研发的监控程序，该监控程序根据各个实例模型加载情况，将可用模型服务主动注册到微服务架构中的注册中心以实现对模型的编排管理。实验结果表明：采用本文的微服务架构方案，有效提升了Tensorflow Serving服务的吞吐量，降低了服务响应时间，简化了模型调用流程，从而满足Tensorflow Serving在生产环境中部署和运维的实际需求。

关键词：Tensorflow Serving；微服务；第三方注册
中图分类号：TP311　**文献标识码**：A

Deep Learning Model Service System based on Microservice

JIANG Ningyuan, ZHANG Huaxiong

(*School of Information*, *Zhejiang Sci-Tech University*, *Hangzhou* 310018, *China*)
✉240662137@qq.com; zhxhz@zstu.edu.cn

Abstract: Tensorflow Serving, an open source service system of Google, has problems of low throughput of Tensorflow Serving single application, cumbersome service invocation, and imperfect model lifecycle management. Aiming at these problems, this paper proposes to design a microservice software architecture solution based on Tensorflow Serving. Monitoring program developed in this research is added to Docker (open source application container engine) container where Tensorflow Serving is deployed. The monitoring program actively registers available model services to the registry of the microservice architecture according to loading status of each instance model, so to realize orchestration management of the model. Experimental results show that the proposed microservice architecture solution effectively improves throughput of Tensorflow Serving services, reduces service response time, and simplifies model invocation process. It can meet actual needs of Tensorflow Serving deployment, operation, and maintenance in production environment.

Keywords: Tensorflow Serving; microservice; third-party registration

1 引言(Introduction)

TensorFlow[1]是目前主流的开源机器学习[2]平台，它拥有一个包含各种工具、库和社区资源的全面灵活的生态系统。为解决神经网络模型部署问题，TensorFlow推出在线原生模型服务系统Tensorflow Serving[3]。但随着神经网络的发展，网络的深度逐渐加深，进行模型推理所需要的计算资源也在不断扩大。在线上的生产环境中，往往需要对请求进行即时响应，Tensorflow Serving这种单体架构已经无法满足企业对系统的需求，应当对其进行相应的架构调整。

微服务[4]是一种软件架构风格，利用模组化的方式组合出复杂的大型应用程序。本文将Tensorflow Serving作为一个服务模块嵌入微服务架构中，使可用模型服务主动注册到微服务架构中的注册中心。通过对模型的生命周期进行管理，使整个系统拥有更好的可扩展性、易维护性和资源隔离性。微

基金项目：浙江省基础公益研究计划项目(LGG20F020016)；浙江省重点研发计划项目(2020C03104)．

服务架构下多实例化也能满足高并发时的快速响应，大幅提高系统的性能。

2 相关技术简介(Related technology introduction)

2.1 Tensorflow Serving简介

2016年2月，TensorFlow团队在GitHub开源一个名为“Tensorflow Serving”的项目。Tensorflow Serving是一种灵活、高性能的机器学习模型服务系统，适用于生产环境，通过GRPC或HTTP接口对外提供服务，HTTP接口如表1所示。Tensorflow Serving 使得Tensorflow深度学习模型开箱即用，也可以轻松扩展服务于其他类型的模型和数据。

表1 Tensorflow Serving HTTP接口API

Tab.1 Tensorflow Serving HTTP interface API

名称	描述
Monitoring	收集所有由Tensorflow核心捕获的指标
Model status API	返回模型的服务状态的接口
Model Metadata API	返回模型的元数据的接口
Classify and Regress API	进行分类和回归推理的接口
Predict API	进行预测推理的接口

Tensorflow Serving加载的标准模型格式为pb模型，pb模型文件实际上是使用了protocol buffer格式存储的模型图和模型参数文件。如图1所示，pb模型文件可以保存整个网络模型(计算图+参数)，并将所有的变量固化为常量。

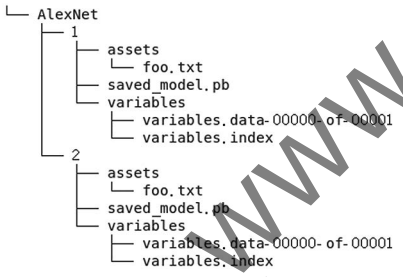

图1 Pb文件格式

Fig.1 Pb file format

此外，Tensorflow Serving可以通过配置文件自定义模型版本的加载策略，管理模型迭代。在部署方面，Tensorflow Serving可以使用apt二级制安装、源码编译和Docker[5]容器(官方推荐)部署，支持调用CPU和NVIDIA GPU资源。

2.2 服务注册和发现简介

服务注册和发现[6]是微服务架构运行的基础。服务注册通常是将服务提供者的网络信息发送至注册中心[7]，而后服务消费者通过服务发现从注册中心获取部署服务提供者所在的网络地址。在这一过程中，注册中心需要支持服务动态上下线并能够实时推送网络信息列表的变化。注册中心主要维护服务实例名称及其相应的网络地址信息，此外我们往往要对不健康或特定的实例进行筛选，那么注册中心还需存储实例的权重、健康状态等数据。

自注册和第三方注册[8]是目前主要的两种服务注册方式。其中，自注册是由服务实例本身负责在服务注册表中注册、注销以及维持心跳。而在第三方注册中，由一个第三方独立的服务registrar通过轮询部署环境或订阅事件等方式去跟踪运行中实例的变化，负责实例的注册与注销。

3 系统设计(System design)

3.1 系统设计原则

(1)扩展性。本系统是面向微服务设计的，开发的各组件需能够独立运行和部署，根据业务的需求，易于扩展，节省硬件成本。

(2)低开销。嵌入原生Tensorflow Serving容器中的第三方监控收集数据的业务逻辑应当简单，没有复杂的依赖，对服务器资源占用也较少，对被监控对象的影响也要尽可能小。

(3)稳定性。在生产环境中，网络的波动、模型版本的迭代会经常发生，系统需要保证模型服务的可用性、稳定性。

(4)时效性。当模型服务新上线或不可用时，需要及时更新模型注册列表，确保模型服务能被成功调用。

3.2 系统总体设计

系统总体架构如图2所示，Tensorflow Serving作为模型服务的提供者，在微服务架构中要被服务消费者调用，消费者需要从注册中心获取服务提供者的服务名、IP及端口等信息。本系统在部署了Tensorflow Serving的Docker端里构建了一个第三方监控程序，该监控程序将获取Tensorflow Serving的IP、端口、成功加载的模型名及模型版本号。获得这些信息后，监控程序通过HTTP接口以JSON的形式向注册中心注册模型信息。

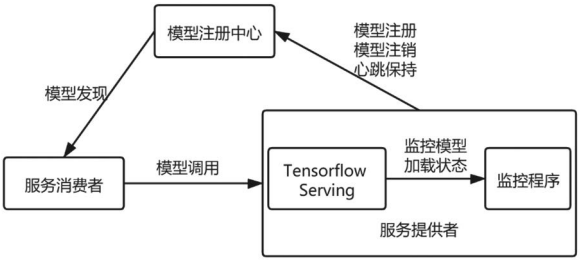

图2 系统总体架构

Fig.2 The overall structure of system

模型注册中心是整个系统中最基础的组件，它维护各模型的信息，信息包括：模型名、模型版本号、模型健康状态、模型服务IP地址及端口号等。其中模型健康状态是由心跳检测维持的，心跳的周期默认是5 s，若服务注册中心在15 s内没收到某实例的心跳信息，就将该实例设置为不健康；若在30 s内没收到实例的心跳信息，就将该实例摘除。

如图3所示，服务的调用可分为五个步骤：(1)第三方监控程序获取成功被Tensorflow Serving加载的模型相关信息。

(2)第三方监控程序向模型注册中心进行模型注册或注销，并维持心跳。(3)模型注册中心更新模型注册表。(4)服务消费者从模型注册中心获取模型的调用地址信息。(5)服务消费者根据模型网络地址，调用模型服务。

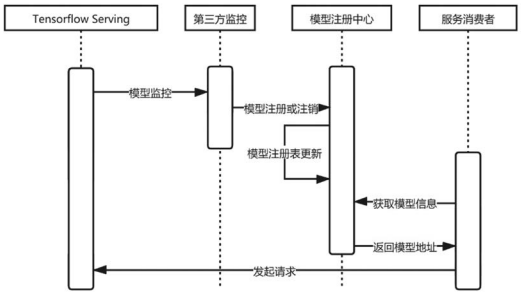

图3 服务调用时序图

Fig.3 Service call sequence diagram

4 系统实现及分析(System implementation and analysis)

4.1 监控程序实现

第三方监控程序基于Spring Boot开发，使用@EnableSchedul开启定时任务，实时监测Tensorflow Serving状态的变化。通过Tensorflow Serving已有的HTTP接口获取模型及其版本的状态，监控流程如图4所示。

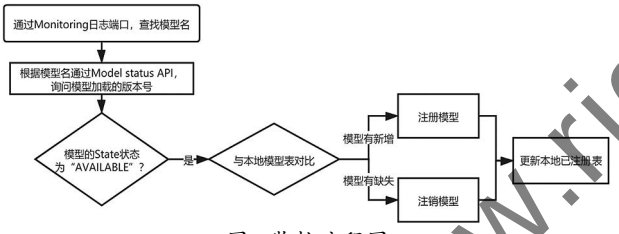

图4 监控流程图

Fig.4 Monitoring flow chart

(1)信息的获取：通过Monitoring日志信息获取Tensorflow Serving加载的模型名；再根据模型名，通过Model status API获取该模型的状态信息，如表2所示。

表2 Model status API范例

Tab.2 Model status API example

操作方式	API信息
范例输入	查询ResNet模型信息
访问方式	GET
范例URL	http://localhost:8501/v1/models/ResNet
范例输出	{ "model_version_status": { "version": "1", "state": "AVAILABLE", "status": { "error_code": "OK", "error_message": "" } } }

(2)分析：根据Model status API返回的信息，同监控程序本地模型表对比，判断是否存在新增或异常的模型。

(3)执行：向模型注册中心进行模型注册或注销，并更新监控程序本地模型表。

(4)此外，监控程序还需定时向注册中心发送心跳信息。

4.2 模型注册中心实现

模型注册中心基于Nacos，Nacos是阿里巴巴在2018年7月发布的一个易于构建云原生应用的动态服务发现、配置管理和服务管理平台。Nacos与当前主流微服务框架Spring Cloud和Dubbo紧密融合，可通过NacosSync与Zookeeper、Eureka、Consul等主流注册中心进行数据同步。在Nacos中，以{ip#端口#集群名#分组名@服务名}设定一个实例ID，本文模型注册中心使用{ip#端口#集群名#模型名@版本号}来确定一个模型实例。本系统实现的模型注册中心界面如图5所示。

图5 模型注册中心界面

Fig.5 Model registration center interface

4.3 Docker端改造

第三方监控程序需要和Tensorflow Serving在同一个Docker容器中运行，查阅Tensorflow Serving原生dockerfile文件，启动命令为：

ENTRYPOINT ["/usr/bin/tf_serving_entrypoint.sh"]

那么只需在tf_serving_entrypoint.sh文档中添加执行第三方监控程序的命令，如图6所示。

```
#!/bin/bash
# 第三方监控程序执行命令
java -jar /app.jar "$@" &
# 原有的TensorflowServing执行命令
tensorflow_model_server --port=8500 --rest_api_port=8501 --model_name=${MODEL_NAME} --
model_base_path=${MODEL_BASE_PATH}/${MODEL_NAME} "$@"
```

图6 程序启动命令

Fig.6 Program start command

4.4 实验分析

为了评价第三方监控程序对原生Tensorflow Serving服务的性能影响，在进行监控和不进行监控的情况下对服务的

平均响应时间、吞吐量进行对比，得出第三方监控程序本身的执行代价。服务器采用四台物理机(操作系统Windows 7、Intel i5 4590@3.3 GHz CPU、8 GB内存、10 M/s带宽)，其中一台作为消费者使用性能测试工具ContiPerf，另外三台分别部署Docker容器并加载ResNet模型提供模型调用服务。实验对单节点有监控和无监控、多节点负载均衡(采用三台相同配置的节点，并使用本系统的模型注册中心，采用轮询的负载均衡调用方式发送请求)三种情况进行对比，记录不同请求并发数下服务平均响应时间和服务吞吐量的数值，实验结果如图7和图8所示。

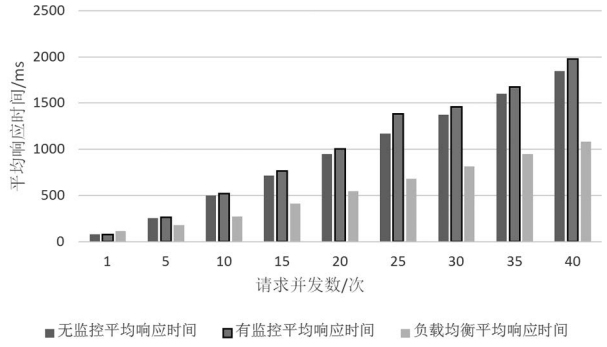

图7 单节点有监控和无监控、多节点负载均衡情况下服务平均响应时间

Fig.7 Average response time of service under single node with and without monitoring, multi-node load balancing

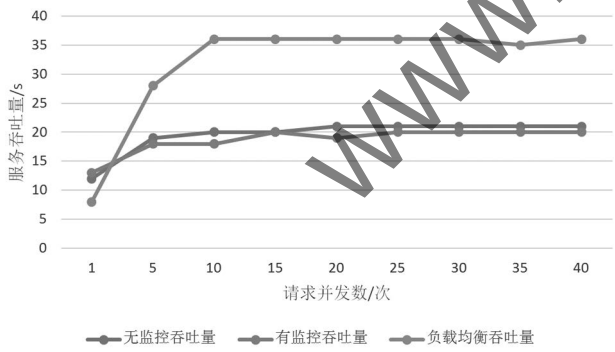

图8 单节点有监控和无监控、多节点负载均衡情况下的服务吞吐量

Fig.8 Service throughput of single node with and without monitoring, multi-node load balancing

由图7和图8可知，随并发请求数的增加，模型服务的平均响应时间也逐渐增加，且两者基本成正比的关系。对比单节点有监控和无监控情况下服务响应时间及吞吐量的关系，计算得出由第三方监控造成的服务响应延迟时间的代价为10—100 ms，对吞吐量几乎不构成影响。同时当请求并发数达到20时，平均响应时间已达到1,000 ms，有明显的响应延迟现象。当由1 个服务节点扩充至3 个服务节点并采用负载均衡的请求方式后，服务的吞吐量大幅提升，服务响应时间明显下降。综上所述，说明本文的基于微服务架构的深度学习模型服务系统在性能方面有明显的优势。

5 结论(Conclusion)

本文改造Tensorflow Serving原生Docker端，在Docker容器中以较小的侵入性添加第三方监控程序，并使用第三方注册的模式同已有的微服务框架进行整合。仿真实验结果表明，本文的基于微服务的深度学习模型服务系统设计，一定程度上简化了模型的部署、调用；另一方面，实验测试了多实例负载均衡的调用方式，实验结果表明本文的设计符合高吞吐、高并发、高可用的性能要求。

参考文献(References)

[1] 贵宁,张浩然.TensorFlow架构与实现机制的研究[J].计算机技术与发展,2019,029(009):31-34.

[2] HAO X, ZHANG G, MA S. Deep learning[J]. International Journal of Semantic Computing, 2016, 10(03):417-439.

[3] OLSTON C, FIEDEL N, GOROVOY K, et al. TensorFlow-Serving: Flexible, high-performance ML serving[J/OL]. ArXiv Preprint ArXiv:1712.06139v2, 2017-12-27[2021-04-08]. https://arxiv.org/pdf/1712.06139.pdf.

[4] LARRUCEA X, SANTAMARIA I, COLOMO-PALACIOS R, et al. Microservices[J].IEEE Software, 2018, 35(3):96-100.

[5] 林跃,冯薇桦,孙源泽.基于Docker的容器虚拟化技术[J].中国新通信,2020,22(09):68.

[6] 段丽君.Web服务发现研究现状分析[J].计算机科学与应用,2017,7(12):1270-1277.

[7] 张子龙,毛新军,尹俊文,等.面向自主Web服务的注册中心模型及其实现技术[J].计算机科学,2014,041(011):118-123.

[8] RICHARDSON C. Microservice patterns[M].Greenwich: Manning, 2017:80-85.

作者简介：

江宁远(1996-)，男，硕士生.研究领域：软件开发，信息研究.

张华熊(1971-)，男，博士，教授.研究领域：软件开发，信息研究.