计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(5):83−91 [doi: 10.15888/j.cnki.csa.007900]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于微服务架构的众包图像数据集标注系统①

袁晓晨[1], 张卫山[1], 高绍姝[1], 时　斌[2], 赵永俊[2], 王　治[3], 安云云[4]

[1](中国石油大学 (华东) 计算机科学与技术学院, 青岛 266580)
[2](青岛海尔空调电子有限公司 电控模块开发部, 青岛 266101)
[3](解放军 9144 部队, 青岛 266102)
[4](国网山东省电力公司 青岛市黄岛区供电公司, 青岛 266500)
通讯作者: 张卫山, E-mail: zhangws@upc.edu.cn

摘　要: 深度学习在图像识别领域凸显出了优势, 而在深度学习图像识别模型训练的准备阶段, 制备图像数据集需要人工将图片上的信息进行标注. 这一准备过程往往需要耗费大量人力成本与时间成本. 为了提升数据制备阶段的工作效率, 从而加速深度学习模型的生成与迭代, 提出了一种基于微服务架构的多人协作众包式图像数据集标注系统. 通过将繁重的标注任务划分为不同的小任务, 使更多的人能够参与并协同完成数据标定. 通过引入对象存储机制并采用微服务架构, 提升了系统性能, 在开发阶段使用了基于 Gitlab 的持续集成与持续部署, 实现了系统的快速迭代与部署, 提升了微服务系统在开发过程中的集成效率.

关键词: 微服务; Spring Cloud; 持续集成; 持续部署; 图像标注

引用格式: 袁晓晨,张卫山,高绍姝,时斌,赵永俊,王冶,安云云.基于微服务架构的众包图像数据集标注系统.计算机系统应用,2021,30(5):83–91. http://www.c-s-a.org.cn/1003-3254/7900.html

Image Dataset Annotation System in Crowdsourcing Based on Microservice Architecture

YUAN Xiao-Chen[1], ZHANG Wei-Shan[1], GAO Shao-Shu[1], SHI Bin[2], ZHAO Yong-Jun[2], WANG Ye[3], AN Yun-Yun[4]

[1](College of Computer Science and Technology, China University of Petroleum, Qingdao 266580, China)
[2](Electronic Control Module Development Department, Haier Air Conditioning Electronic Ltd. (Qingdao), Qingdao 266101, China)
[3](No. 9144 Troops of PLA, Qingdao 266102, China)
[4](Qingdao Huangdao District Power Supply Company, State Grid Shandong Electric Power Company, Qingdao 266500, China)

Abstract: Deep learning has shown visible advantages in the artificial intelligence-based image classification. It usually costs plenty of time on manual information annotation for preparing image datasets. Then this study proposes an online collaborative system for image dataset annotation based on microservice architecture to improve the efficiency of annotating datasets and thus to accelerate the generation and iteration of deep learning models and applications. More users can join for image annotation after the heavy annotation task is divided into smaller ones. Besides, the system performance has been improved by introducing an object storage system and microservice architecture, and the integration efficiency of the system in the development progress has been enhanced by continuous integration and deployment.

Key words: microservice; Spring Cloud; continuous integration; continuous deployment; image annotation

① 基金项目: 国家自然科学基金 (62072469); 国家重点科研计划 (2018YFE0116700); 山东省自然科学基金 (ZR2019MF049); 中央高校基础研究基金 (2015020031); 西海岸人工智能技术创新中心建设专项 (2019-1-5, 2019-1-6); 上海可信工业控制平台开放项目 (TICPSH202003015-ZC)
Foundation item: National Natural Science Foundation of China (62072469); National Key R&D Program (2018yfe0116700); Shandong Natural Science Foundation (ZR2019MF049); Basic Research Fund of Central University (2015020031); West Coast Artificial Intelligence Technology Innovation Center (2019-1-5, 2019-1-6); the Opening Project of Shanghai Trusted Industrial Control Platform (TICPSH202003015-ZC)
收稿时间: 2020-09-10; 修改时间: 2020-10-09; 采用时间: 2020-10-13; csa 在线出版时间: 2021-04-28

计算机视觉图像识别是人工智能的重要应用, 广泛应用在工业、医学、军事、教育、商业、体育、安防检测等行业与领域中. 机器学习, 尤其是深度学习展现出了针对图像识别领域优秀的识别性能. 而机器学习本身需要建立在大量的带有指导意义的既有数据集基础之上. 在进行深度学习模型训练流程中, 往往需要针对海量图片进行人工数据标注, 繁重的图像标注任务增添了大量时间成本.

传统的图像标注工具, 如表 1 所示, 大多以单机运行的传统单体式系统架构为主, 运行在单机之上, 同一时间同一系统运行实例上只允许一个用户对本地资源进行图像标定. 当使用传统图像标注工具进行协作标定时, 需要用户手动进行图像集的分组并拷贝至协作组员的工作站. 协作组员各自完成任务后仍需要用户自行合并制作数据集. 由于缺少集中化的图像与工作流, 导致传统单机标注工具在协作场景下存在大量的文件与数据传输, 在海量图像文件的压缩、打包与传输过程中产生了大量的时间与人工成本.

表 1 传统图像标注工具对比

图像标注工具实例	系统架构	兼容系统	开源与付费模式	协作模式	支持的图像标注类型	导出格式	网址
LabelImg	单机(PyQt5)	全平台	开源免费可独立部署	手动资源分组打包后独立运行协作标注	矩形	PASCAL、VOC、ImageNet	https://github.com/tzutalin/labelImg
LabelMe	单机(PyQt5)	全平台	开源免费		多边形、矩形、圆形、线标注、关键点	COCO	https://github.com/wkentaro/labelme
VIA-VGG Image Annotator	Web本地静态页面	全平台	开源免费		多边形、矩形	JSON、CSV、COCO	https://rectlabel.com/
PixelAnnotaion	单机(Qt5)	Windows、MacOS、Linux(自行编译)	开源免费		填充区域	JSON	https://github.com/abreheret/PixelAnnotationTool
RectLabel	单机	MacOS	不开源付费$55		多边形、填充区域、矩形	PASCAL、VOC、YOLO、COCO、CSV	https://rectlabel.com/
OpenCV/CVAT	Web本地部署(Django)	全平台	开源免费	单实例不支持直接协作, 需要手动对资源分组打包独立部署多实例	图像分类、多边形	工具预定义XML格式	https://github.com/opencv/cvat
VoTT	Web本地部署(NodeJs)	全平台	开源免费		矩形	Azure自有格式、CNTK、Pascal、VOC、TFRecords、VOTT(JSON)、CSV	https://github.com/microsoft/VoTT
LabelBox	纯前端库	全平台	不开源需要引用公网库资源	需要额外系统支持	图像分类、多边形、矩形	CSV、JSON(由官方后端提供)	https://github.com/Labelbox/Labelbox)

为了解决机器学习图像识别训练的各种前置准备工作费时费力的问题, 本文提出并设计实现了一种基于 Spring Cloud、面向机器学习模型训练的协作式图像数据管理与标注平台, 通过众包任务的方式, 优化图像标注流程, 设计实现了面向海量图像的存储、标定集管理、图像标定任务管理等功能模块. 系统采用微服务架构, 将系统各个部分进行解耦[1], 实现服务注册与发现、负载均衡、容错处理, 提升了系统的高可用性、可维护性与可扩展性. 基于 GitLab-CICD 实现了微服务的增量更新、持续部署与灰度更新. 通过将海量图像标定数据进行分组划分, 简化数据标注过程中的操作, 降低了数据标注人员的时间成本, 提升了数据标注工作效率.

1 概述

1.1 众包任务模式

众包任务模式指的是将本应当由单一机构或个人执行的工作内容在公开或非公开网络上以公开的方式外包给特定或非特定用户[2]. 通过采用众包任务的模式, 将庞大的图像标注任务切分为小的任务, 能够有效提

升标定的速度与质量, 从而加速深度学习模型训练流程.

1.2 微服务架构

微服务[3] 是一系列功能简单、互相之间采用轻量级通信协议协同工作的功能区块. 每个小的功能区块具有高内聚、低耦合的特点, 能够独立自主地运行.

微服务架构是一种架构模式, 它是 SOA 架构 (Service Of Application) 的进一步发扬, 通过将庞大的系统根据业务边界细粒度地拆分为小的服务模块. 每个小的服务模块具有高内聚、低耦合的特点, 均能够独立地运行. 每个模块可以由不同的团队进行开发[4].

微服务架构拥有着许多传统单体式应用所不具备的优势[5]:

① 复杂度可控: 通过将复杂的系统细粒度地拆分为小体积、业务简单的模块, 降低了每个模块的开发难度与复杂度. 各个模块间互相解耦[6], 当需求变更时, 只需要修改对应的模块即可实现系统整体功能上的变更, 无需考虑其他模块可能因为接口实现的变更而无法正常工作的问题.

② 团队协作效率高: 当确定系统各个微服务接口定义后即可分组并行开发不同的微服务模块, 从而提升了整体开发效率. 同时不同微服务模块无需集成为一个整体, 只需要能够互相调用即可保证系统的正常运行, 避免了单体式应用集成过程中出现的组件源码不兼容, 降低了集成成本.

③ 独立部署: 每个小模块都能够独立地运行, 不会因为其他服务故障而受到影响. 通过微服务调度系统的支持, 能够实现快速部署在单机或集群上, 从而充分利用计算性能.

④ 维护难度低: 配合持续集成与持续部署 (CI/CD), 即可实现灰度发布与服务热更新, 无需系统全部停机或重启即可完成功能更新.

⑤ 多种技术允许共存: 微服务独立部署使得微服务间只需要约定技术、语言无关的 RPC 协议即可实现不同技术、语言实现的微服务共存并配合工作[7].

⑥ 系统稳定性高: 微服务架构的系统拥有熔断机制, 当系统部分组件失效时能够及时阻止故障船体, 从而避免系统发生雪崩式功能失效, 提升了系统的稳定性.

1.3 对象存储系统

基于对象的云存储即对象存储 (object storage) 是近几年逐渐流行的一个新兴且切实可行的大规模存储方案[8], 使用较为简便的方法实现数据存储, 即充分利用已有的存储组件、网络技术和处理技术, 使系统拥有较好的可扩展性以及高吞吐量[9]. 对象存储思想的核心为对象, 每个对象都有唯一的标识[10]. 对象存储将文件划分为一个个对象, 为用户提供了统一的存储空间, 从而能更好的对文件进行访问控制和存储管理[11]. 这些对象被分布在整个集群之中, 为保障数据安全、防止数据丢失将每一个对象多重备份复制到多个设备上. 对象存储系统将数据块列表映射为对象列表, 将各类数据块简化成为一个个对象来进行管理[12], 极大改善了系统的可伸缩性, 可以轻易实现海量数据的管理.

2 系统设计

2.1 需求分析

传统的数据集标注往往在单终端中进行, 大量的图像需要由一个部门或者个人完成. 使用传统的标注工具进行多人协作式图像标注时, 在图像整理上浪费了大量时间. 协作前需要人工将大量图像打包分组, 协作后需要将图像与标定数据回收合并为同一个数据集合. 常见的文件系统针对海量小文件的处理性能往往不及少量大文件, 导致人工打包也是一项耗时的工作. 针对以上痛点, 图像标注系统主要有以下需求:

① 图像数据托管: 为避免文件系统直接频繁操作大量小文件, 从而节省任务分配数据打包的时间, 图像数据应当由专门的机制进行托管, 必要时以图像为基本单位提供图像及其元数据检索服务.

② 任务划分与管理: 为实现众包模式的图像标注, 需要将包含大量图像的任务拆分为小任务, 交由不同的个人进行标注.

③ 支持多种标注模式: 系统应当支持机器视觉训练常用的关键点、矩形、多边形标注模式.

④ 支持多种导出格式: 针对主流深度学习源码所接受的格式, 系统应当具备良好的扩展性以支持更多的深度学习框架.

⑤ 数据统计: 任务发起者应当能够看到子任务的标注进度、标签数量等统计信息, 从而有针对性地决定是否需要增加或者调整图像组成.

⑥ 支撑模块需求: 除此之外, 系统应当具有基本的用户与权限控制功能.

2.2 微服务划分与系统架构设计

① 系统微服务划分

根据系统需求分析结果, 将系统划分为 4 个功能

性微服务和 3 个架构支撑服务: 图像对象存储服务、图像集合管理服务、标定集合管理服务、数据输入输出服务、用户及授权管理和服务网关.

图像对象存储服务用于面向海量图片的对象存储, 将图片统一以相同规则生成的不重复定位符作为索引, 避免图片重复存储与图片文件名重复冲突的问题.

图像集合管理服务用于将离散的图片在逻辑上组成一个集合, 作为系统中图片操作的基本单位.

标定集合管理服务用于管理和存储图像集对应的标定数据, 同时提供标定任务的划分与分配.

数据导入导出服务主要用于图像、标签数据的解包导入与打包导出. 提供常见的压缩与视频格式的解析与常见数据集格式的导出等功能.

② 系统架构设计

系统整体采用分层式结构, 如图 1 所示. 为了开发过程中能够更明确的分工, 其中服务层按照微服务的思想进行拆分. 各层主要包含内容如下:

持久化层: 为了方便使用微服务编排框架进行部署, 所有有状态的服务均从整体架构中分离整合在持久化层中. 其中包括用于对象数据存储的 NoSQL 数据库; 用于关系型信息数据存储的 SQL 数据库; 用于全局数据缓存的内存 NoSQL 数据库; 用于全局消息同步的消息队列中间件.

微服务层: 包含了业务逻辑微服务群和架构支撑服务群两部分. 架构支撑服务群中包含了微服务架构必需的注册发现中心、日志监控、配置中心等基础微服务. 业务逻辑微服务群提供了实现业务逻辑的相关微服务, 包括图像对象存储微服务、图像集管理微服务、标定管理微服务和数据导入导出微服务.

网关层: 网关层包含了由 Spring Gateway 实现的 API 网关微服务、前端站点微服务和 Nginx 总代理微服务, 提供了系统接入入口.

CI/CD 支撑部分: 系统源代码采用了 Gitlab 私服进行托管, Gitlab 也提供了对持续集成的支持, 故直接采用 Gitlab-CICD 实现系统开发过程中的持续集成与部署.

系统采用现有完整的微服务方案, 使用服务发现实现松散的服务间耦合, 采用声明式 RPC 客户端实现微服务间的互相调用, 使用统一网关代理微服务作为系统入口, 添加负载均衡机制以扩展系统负载容量, 使用 OAuth2 开放认证协议作为认证机制.

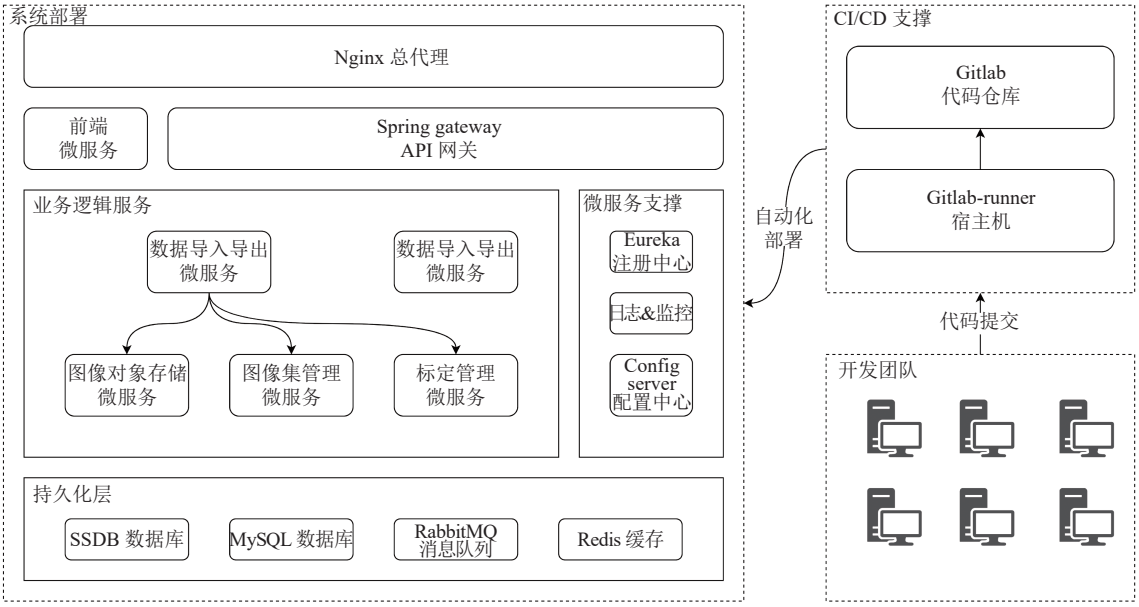

图 1 图像标注系统架构

3 持续集成、持续部署与微服务架构实现

3.1 微服务注册与发现

在微服务架构的系统中, 为了实现微服务间既有松散的耦合度, 又能够互相访问, 微服务注册与发现机制是一种常用方法[13]. 通过由微服务自行将自身的信息主动注册至注册中心的方式, 使得其他微服务可以

通过查询注册中心注册记录的方式间接地发现其存在并获得访问相关参数. 注册中心是注册与发现中最重要的一部分, 在 Spring Cloud 微服务套件中, 提供了注册中心的一种原生实现和两种接入实现:

① Eureka Server: 由 Netflix 开发的基于原生 Java 的开源注册中心实现, Spring Cloud 套件给出了基于 Spring Boot 快速构建注册中心的方案.

② Spring Cloud Consul: Consul 是由 HashiCorp 公司开发的一种服务网格解决方案, 提供具有服务发现, 配置和分段功能的全功能控制平面[14]. 通过添加 org.springframework.cloud.spring-cloud-starter-consul-all 依赖, 同样可以实现类似的注册发现功能.

Spring Cloud Zookeeper: Apache ZooKeeper 是一项集中式服务, 用于维护配置信息, 命名, 提供分布式同步和提供组服务[15]. 基于 Spring Cloud 的微服务可以通过添加 org.springframework.cloud.spring-cloud-starter-zookeeper-all 依赖实现基于 ZooKeeper 的注册发现.

由于后两种注册发现的方案需要单独部署第三方的应用实例来支撑微服务注册与发现的功能, 故选择可自行构建的 Eureka Server 方案.

Eureka 的注册与发现方案整体流程如图 2 所示, 主要分为两部分.

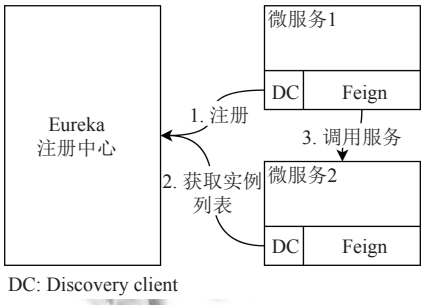

图 2 基于 Eureka 的注册发现实现流程

第一部分为 Eureka 注册中心. Spring 提供了一套项目生成工具 Spring Initializer[16], 允许开发人员直接通过可视化配置直接生成 Spring Boot 项目, 这里借用 Spring Initializer 可以直接生成 Eureka Server 的项目.

如图 3 所示, 在 Spring Initializer 中输入项目信息并选中 Eureka Server 依赖, 点击 Generate 即可得到初始项目模板. 在 Spring Boot 启动类上添加@EnableEurekaServer 参数即可开启项目依赖中的 Eureka Server.

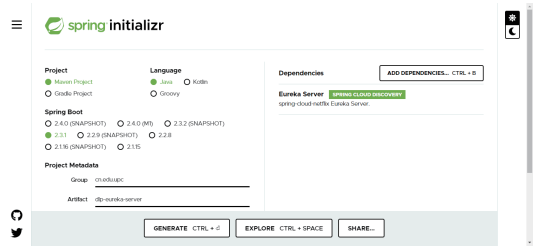

图 3 使用 Spring Initializer 生成 Eureka Server 实例

第二部分即业务微服务中的 Discovery Client. 在需要通过服务发现感知其他服务实例的微服务中, 增加 org.springframework.cloud.spring-cloud-starter-netflix-eureka-client 依赖, 并在项目引导类上添加@EnableDiscoveryClient 注解, 即可为 Feign 客户端添加服务发现的功能.

完成其他业务微服务开发后, 同时启动各个微服务, 可通过登录 Eureka 监控页面看到各个微服务的注册情况, 如图 4 所示.

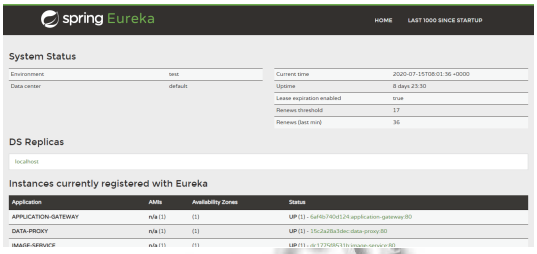

图 4 通过 Eureka Server 查看服务发现注册状态

3.2 微服务接口网关与负载均衡

相较于传统的单体式应用, 微服务架构的应用更接近于单体式应用站群. 微服务架构应用为了实现从外部表现为与单体式应用类似的形式, 需要一个微服务作为接口网关, 同时需要提供负载均衡的特性.

对于 API 网关, Spring Cloud 提供了两套方案:

① Zuul: Zuul 来源于 Netflix 开源的微服务架构套件. Zuul 底层采用了 Tomcat Embeded 版本作为 HTTP 支撑层.

② Spring Cloud Gateway: Spring Cloud Gateway 是由 Spring 项目组基于其自研 Web 框架 WebFlux 实现的 API 网关. Spring Cloud Gateway 是 Spring Cloud 去 Netflix 进程中非常重要的一个项目, 旨在替代 Zuul 成为未来的 Spring Cloud 框架下的 API 网关组件.

WebFlux 底层为直接使用 Netty 等高性能非阻塞

服务器, 相较于采用 Servlet 架构的 Tomcat(Embeded)性能上略胜一筹 [17]. 考虑到未来 Spring 项目组开发的方向与性能预期, 采用 Spring Cloud Gateway 作为系统 API 网关.

Spring Cloud Gateway 同样可以使用 Spring Initializer 进行项目初始化, 初始化完成后可以通过在 application.yml 中的 spring.cloud.gateway.route 字段中配置接口路径与微服务的映射.

代码 1. Spring Cloud Gateway 路由配置

```
spring:
  application:
    name: application-gateway
  cloud:
    gateway:
      routes:
        # 用户服务
        - id: user-service
          uri: lb://user-service
          predicates:
            - >
              Path=
              /oauth/**,
              /user/**,
              /role/**
        # 其他服务
        # …
```

对于负载均衡, Spring Cloud 框架提供了 Ribbon 组件. Ribbon 是一款客户端侧负载均衡器, 它可以自动从 Discovery Client 中获取微服务实力列表, 应用常见的负载均衡算法实现在同一服务的多个冗余实例上的负载均衡[18]. 通过添加 org.springframework.cloud.spring-cloud-starter-netflix-ribbon 依赖即可在服务调用侧引入负载均衡组件.

3.3 微服务熔断机制

微服务之间往往避免不了互相调用对方的服务. 当进行同步接口调用时, 被调用方宕机、接口调用超时往往会引起调用方的异常. 为了避免因微服务之间的依赖关系而出现大面积故障, 调用方服务应当感知被调用方的异常并作出防御性动作, 防止故障继续的蔓延.

在 Spring Cloud 框架中, 提供了 Hystrix 组件. Hystrix 能够在被调用微服务出现异常时及时熔断, 触发调用方的异常处理流程, 防止调用方产生异常. Hystrix 组件可以通过在微服务中添加 org.springframework.cloud.spring-cloud-starter-netflix-hystrix 依赖来添加至微服务中. 在微服务启动类上添加@EnableHystrix 注解即可启用熔断器.

3.4 基于 SSDB 的图像对象存储系统

在深度学习图像识别模型训练过程中往往需要准备大量图像, 而实现存储并快速索引图像数据就是标注系统性能提升的关键点.

SSDB 是一个高性能的支持丰富数据结构的 NoSQL 数据库, 其底层实现为 Google 的高性能键值对数据库 LevelDB.

首先是一个高性能 SSDB 集群, SSDB 采用的是与 Redis 相同的网络通信实现, 故可以采用 Redis 集群常用的 TwemProxy 代理实现如图 5 所示的对象存储架构.

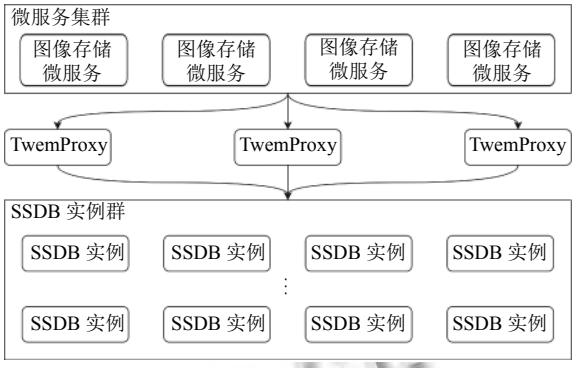

图 5 SSDB 集群架构

图 5 上部分是图像对象存储微服务. 图像存储微服务使用图像原始数据经哈希运算得到的索引号作为图像在对象系统中的唯一索引. 每一张图像采用一个哈希表进行存储, 在哈希表中额外增加有关图像内容相关的元信息, 对于重复上传的内容采取增加元信息中记录的索引数量而不重复存储.

3.5 基于 Gitlab-CICD 的持续集成与部署

在传统的单体式应用中, 只需要编译一次即可得到可运行的产物, 而在微服务架构应用中, 因为划分为了多个实例, 这种类似于“站群”的系统往往需要编译多个“单体式应用”并封装为 Docker 镜像进行部署. 为了减少编译部署阶段的工作量, 使用 Gitlab-CICD 实现全自动化的编译、测试与部署.

首先是准备一个 Gitlab 实例, 可以使用 Gitlab 官方站点或者建立开源的 Gitlab-CE 实例, 本文不再赘述

有关 Gitlab 实例搭建的内容.

系统采用了 Docker Swarm 作为微服务编排框架, 使用 Harbor 作为 Docker Swarm 集群的私有镜像源.

Gitlab-CICD 中另一个重要组成部分便是 Gitlab-Runner. Gitlab-Runner 是实际执行持续集成任务的主体. Gitlab-Runner 支持多种部署方式, 本项目选择了基于裸机系统的编译环境, 方便做一些特殊的环境配置.

Gitlab-CICD 执行流程如图 6 所示, 主要环节包括:

① 代码提交: 在各个微服务中建立 Git 仓库, 并将仓库托管至 Gitlab. 当微服务代码得到更新并推送提交至 Gitlab 时, Gitlab 会自动检查仓库目标分支中的.gitlab-ci.yml 文件中的配置, 并在流水线 (pipeline) 中添加持续集成的任务.

② CICD 任务的执行: Gitlab-Runner 定时向 Gitlab 的流水线队列请求任务, 当 Gitlab-Runner 得到任务后在其宿主机上执行配置文件中定义的脚本. 脚本中定义的动作完成微服务的编译、测试、镜像打包与提交、灰度更新.

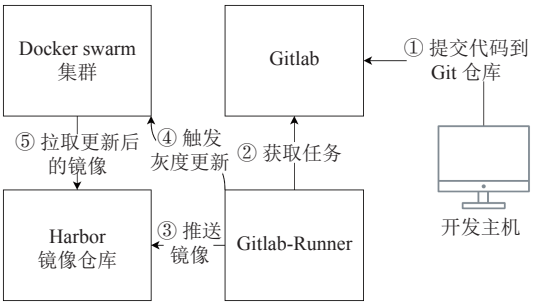

图 6 Gitlab-CICD 执行流程

以图像管理服务为例, 编写了如代码 2 所示的 gitlab-CICD.yml 配置文件.

代码 2. Gitlab 持续集成与持续部署配置

```
stages:
  - 测试
  - 构建
  - 部署
测试:
  stage: 测试
  only:
    - master
  tags:
    - dlp
  script:
    - mvn test
构建:
  stage: 构建
  only:
    - master
  tags:
    - dlp
  script:
    - mvn package -Dmaven.test.skip=true
    - docker build -t ${DOCKER_REGISTRY}/dlp/${SERVICE_NAME}:latest .
    - docker push ${DOCKER_REGISTRY}/dlp/${SERVICE_NAME}:latest
部署:
  stage: 部署
  only:
    - master
  tags:
    - dlp
  script:
    - docker -H ${DOCKER_SWARM} service update
  --image ${DOCKER_REGISTRY}/dlp/${SERVICE_NAME}:latest${DOCKER_SWARM_STACK_NAME}_${SERVICE_NAME}
```

如代码 2 所示, CICD 一次任务将分为 3 个阶段: 构建、测试与部署. 其中测试阶段由 Maven 进行编译并执行系统中的单元测试; 构建阶段直接使用 Maven 进行服务器端编译, 随后使用 Gitlab-Runner 宿主机上的 docker 构建镜像并推送至 Harbor; 部署阶段通过暴露 Docker Swarm 中的 Manager 节点上 dockerd 的 2375 端口, 实现从 Gitlab-Runner 宿主机直接控制集群并触发微服务镜像更新. 运行效果如图 7 所示.

通过在各个微服务实例中分别修改相同逻辑后分别以手动部署与持续集成在多节点集群上部署并统计用时, 得到逻辑更新至生产环境的耗时如表 2 所示.

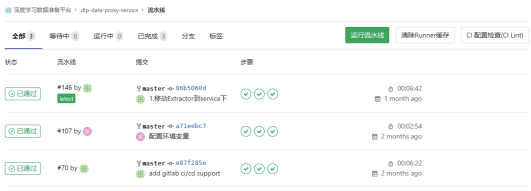

图 7 Gitlab-CICD 执行效果

表 2 手动部署与 CICD 部署效率对比

微服务实例	手动部署耗时	CICD耗时
图像存储(5实例)	30 min 12 s	3 min 8 s
图像集管理(3实例)	21 min 05 s	1 min 26 s
标定集管理(3实例)	20 min 40 s	1 min 6 s
数据导入导出(1实例)	13 min 23 s	6 min 42 s
其他微服务支撑服务	16 min 41 s	1 min 5 s

由表 2 可知, CICD 有效减少了系统集成过程中的集成与部署时间成本, 提升了系统逻辑变更同步至生产环境的效率.

4 系统测试

4.1 系统后端接口测试

项目采用了微服务系统常见的前后端分离结构, 前端在接口确定后采用桩服务器 (mock-server) 与后端并行开发. 后端系统测试过程中使用了 postman 进行接口的测试, 如图 8 所示.

Postman[19] 是一个用于 API 接口开发的协作平台. 它提供了一套 API 接口开发工具, 包括 API 客户端、文档生成、自动化测试、API 接口监控、API 设计与桩服务器与接口文档协作.

Postman 内嵌了 OAuth2 的认证模型, 通过配置即可实现一系列共用同认证服务的 API 同时获得登录认证的功能.

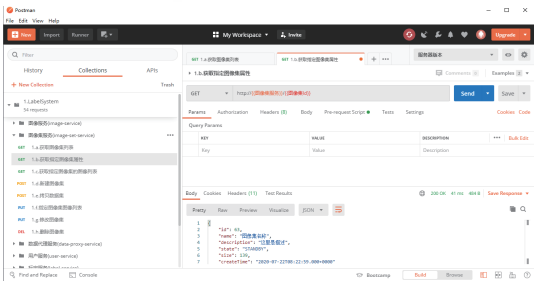

图 8 使用 postman 对后端接口进行测试

4.2 系统功能测试

以深度学习图像数据制备流程为例, 对系统中图像导入、任务分配与标定、数据导出流程进行功能测试.

首先是图像导入. 图像标注系统支持图像、视频、PDF、OpenDocument 等格式混合打包上传, 上传界面如图 9 所示. 上传完成数据传输后转入异步解压处理流程, 异步解压完成后前端显示实际图像集大小.

在图像集中创建出标定集后即可向系统中其他用户分配标定任务进行图像标定的团队协作. 分配任务过程如图 10 所示.

切换至任务执行用户, 可以在“我的任务”页面中看到由任务发起用户分配的任务, 点击“开始”按顺序对图像进行标定. 进入标定流程后, 如图 11 所示, 通过在图像上拖拽创建图像区域, 完成对图像内容的标记.

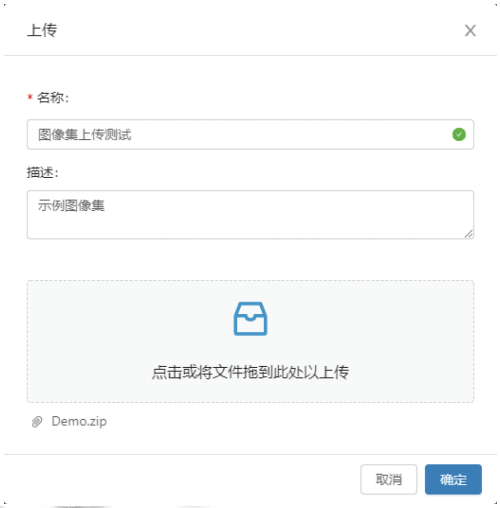

图 9 创建图像集

图 10 分配标定任务

标注过程中发起任务的用户能够看到各个任务执行用户的标定进度与当前已标定图像的标签统计信息. 当图像集标定完成后, 任务发起者可以对图像进行最终修正并导出图像集合与标定集合开展深度学习模型训练. 以导出的 VOC XML 格式为例, 最终导出的标签集如图 12 所示.

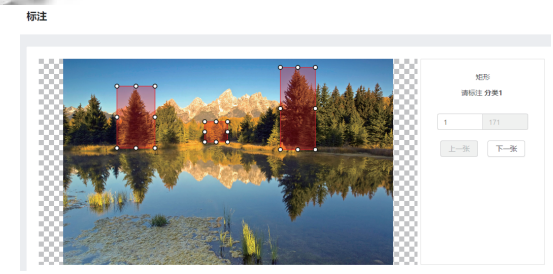

图 11 图像标注过程

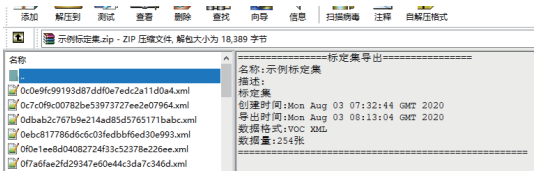

图 12 标签集合导出

4.3 系统性能测试

本文采用开源 Apache Benchmark 对系统中图像对象存储系统进行多并发场景下的性能测试, 测试环境如表 3 所示.

如表 4 所示为模拟多用户进行每个终端进行 10 次访问 (总计访问次数=并发数量×10) 的资源响应速度的测试的测试结果.

由表 4 可知, 采用微服务架构有效提升了系统容量, 允许更多的用户参与到众包图像标定工作中.

表 3 性能测试节点环境配置

环境属性	客户端	微服务集群(x5)
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz
RAM	16 GB	64 GB
磁盘	–	3×HDD RAID5
网络环境	1 Gb/s	10 Gb/s

表 4 图像存储服务性能测试结果 (单位: ms)

并发数量	单实例	多实例
1	17	18
10	32	30
100	311	275
1000	3198	2799
10000	27530	17553

5 结束语

本文设计实现了面向深度学习的图像标注系统, 与传统的本地标注工具不同, 本系统引入了众包协作的思想与多种图像格式存储支持, 将繁重的数据标注任务进行合作完成, 并提供了多种导出格式, 避免了传统工具协作标注工作流中大量图像数据的分包与标签数据合并操作, 简化了协作标定流程, 有效提升了数据准备阶段的工作效率. 同时, 系统架构层面采用了微服务架构, 开发阶段引入基于 Gitlab-CICD 的持续集成与持续部署, 加速了系统开发到部署的更新迭代.

参考文献

1 李春阳, 刘迪, 崔蔚, 等. 基于微服务架构的统一应用开发平台. 计算机系统应用, 2017, 26(4): 43–48. [doi: 10.15888/j.cnki.csa.005757]

2 Howe J. The rise of crowdsourcing. Wired Magazine, 2006, 14(6): 1–5.

3 Richardson C. Introduction to microservices. https://www.nginx.com/blog/introduction-to-microservices/, (2015-05-19).

4 Lewis J, Fowler M. Microservices. http://martinfowler.com/articles/microservices.html. (2014-03-25).

5 周建丁. 七牛技术总监肖勤: 微服务架构实践经验分享. http://www.csdn.net/article/2015-08-07/2825412. (2015-08-12).

6 黄嘉诚, 董晶. 基于微服务的智能档案服务系统设计与实现. 电子设计工程, 2018, 26(2): 26–30. [doi: 10.3969/j.issn.1674-6236.2018.02.007]

7 蒋勇. 基于微服务架构的基础设施设计. 软件, 2016, 37(5): 93–97. [doi: 10.3969/j.issn.1003-6970.2016.05.024]

8 Wang F, Brandt SA, Miller EL, *et al*. Obfs: A file system for object-based storage devices. Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies. Greenbelt, ML, USA. 2004. 283–300.

9 Mesnier M, Ganger G, Riedel E. Object-based storage: Pushing more functionality into storage. IEEE Potentials, 2005, 24(2): 31–34.

10 Sobe P. Reconfiguration of RAID-like data layouts in distributed storage system. Proceedings of the 18th International Parallel and Distributed Processing Symposium. Santa Fe, NM, USA. 2004. 2132–2139.

11 Schall D, Härder T. Dynamic physiological partitioning on a shared-nothing database cluster. Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul, Republic of Korea. 2015. 1095–1106.

12 SCSI Object-Based Storage Device Command (OSD). Technical Report ISO/IEC 14776, INCITS Technical Committee T10. Revision 10, 2004.

13 张晶, 黄小锋. 一种基于微服务的应用框架. 计算机系统应用, 2016, 25(9): 265–270. [doi: 10.15888/j.cnki.csa.005347l]

14 Consul. Introduction to Consul. https: //www.consul.io/intro. (2020-04-10)[2020-08-01].

15 Apache. Zookeeper. https: //zookeeper.apache.org/. (2020-05-13)[2020-08-02]

16 Spring. Spring initialize. https: //start.spring.io/. (2020-07-05)[2020-08-02]

17 Spring. Spring WebFlux documentation. https: //docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html. (2020-06-20)[2020-08-02]

18 马雄. 基于微服务架构的系统设计与开发 [硕士学位论文]. 南京: 南京邮电大学, 2017.

19 Sroka A. POSTMAN–Powerful API testing tool. Diwebsity. https: //www.diwebsity.com/2016/04/21/postman/. (2016-04-21)[2020-08-02].