计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(7):80−86 [doi: 10.15888/j.cnki.csa.007997]

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

面向深度学习的分布式任务执行系统①

高国樑[1], 陈雷放[2], 刘一鸣[3]

[1](中国石油大学 (华东) 计算机科学与技术学院, 青岛 266580)
[2](青岛农业大学 理学与信息科学学院, 青岛 266109)
[3](华北电力大学 (保定) 电气与电子工程学院, 保定 071003)
通讯作者: 高国樑, E-mail: 2725373994@qq.com

摘　要: 深度学习全流程托管平台提供了深度学习实验任务的网页端解决方案, 加速了深度学习技术在生产生活中的应用. 为了解决网页端深度学习平台进行图像识别模型训练的问题, 本文设计实现了面向深度学习实验任务的分布式任务执行系统. 系统由资源监控、任务调度、任务执行、日志管理 4 大模块组成, 将任务依据资源使用率等策略进行调度, 采用 Docker 容器技术进行执行, 并对产生的日志信息进行了实时收集. 经过测试, 分布式任务执行系统不仅保证了正常的功能需求, 在可靠性、稳定性等指标上也都达到了预期的要求, 将其集成到平台中可减少 20% 左右的训练时间.

关键词: 分布式; 任务调度; 任务执行; 日志; 资源监控

引用格式: 高国樑,陈雷放,刘一鸣.面向深度学习的分布式任务执行系统.计算机系统应用,2021,30(7):80–86. http://www.c-s-a.org.cn/1003-3254/7997.html

Distributed Task Execution System for Deep Learning

GAO Guo-Liang[1], CHEN Lei-Fang[2], LIU Yi-Ming[3]

[1](College of Computer Science and Technology, China University of Petroleum, Qingdao 266580, China)
[2](College of Science and Information, Qingdao Agricultural University, Qingdao 266109, China)
[3](School of Electrical and Electronic Engineering, North China Electric Power University (Baoding), Baoding 071003, China)

Abstract: The whole lifecycle hosting platform of deep learning offers a web solution to experimental tasks and boosts the application of deep learning technology in production and life. To address the problem of training image recognition models by the platform, this study designs and implements a distributed task execution system for experimental tasks. The system is composed of modules for resource monitoring, task scheduling, task execution, and log management. It schedules tasks according to indicators, such as resource utilization, executes tasks in Docker containers and collects generated log data in real time. The test results demonstrate that the system fulfils the normal functional requirements, achieving the desired targets regarding reliability and stability while reducing about 20% of training time after being integrated into the deep learning platform.

Key words: distribution; task scheduling; task execution; log; resource monitoring

近年来, 深度学习技术广泛应用在图像处理、语音识别等领域, 极大地促进了人工智能的进步, 对人类生活的改善有着重要意义. 然而, 深度学习实践并非一件易事, 往往需要花费大量的时间和精力, 因此, 使深度学习变得尽可能“简单”显得尤为重要. 深度学习托管平台可为深度学习实验任务提供全流程的管理服务, 给深度学习实践带来了积极影响. 结合深度学习平台项目的实际需求, 本文设计和实现了分布式任务执行

① 收稿时间: 2020-11-03; 修改时间: 2020-12-02; 采用时间: 2020-12-18; csa 在线出版时间: 2021-06-30

系统 DTES (Distributed Task Execution System). DTES 参照任务执行集群的实时监控数据选出当前压力最小的节点并向其发送执行请求, 节点根据请求信息去获取任务所需数据, 随后创建 Docker 容器来执行该任务, 同时回收任务执行期间的日志信息. 此外, DTES 基于 Spring Boot 框架实现, 具有良好的稳定性和可扩展性.

1 研究背景

深度学习全流程托管平台是一个管理深度学习实验任务的网页应用系统. 该平台提供了数据集管理、数据集标注、模型训练以及模型部署等服务, 它将深度学习实验的所有流程全部集中到网页端进行, 有效提升了研究效率. 研究者首先将数据集上传到平台, 接着对数据集进行标注, 之后在平台中编写程序训练模型, 最后将符合要求的模型导出部署.

本文结合深度学习平台的具体需要和深度学习实验的特性[1–3], 提出了一种面向深度学习模型训练的分布式任务执行系统.

2 需求分析

需求分析是系统开发的基础, 关系到开发工程的成败和软件产品的质量. 本文提出的分布式任务执行系统主要应用于网页端深度学习模型训练, 系统的主要功能有监控、任务调度、任务执行、日志管理等, 具体如下.

(1) 监控: 包括对执行器节点的监控以及对任务的监控, 其中节点监控主要是监控各个节点执行器的 CPU、GPU、内存、磁盘、网络等情况, 任务监控主要是监控任务的当前状态;

(2) 任务调度: 调度器首先获取各个节点执行器的内存、CPU 等资源的占用情况, 以确定可以继续执行任务的执行器, 然后向执行器发送任务执行请求;

(3) 任务执行: 调度器发出的执行请求进入执行器的任务队列, 执行器按照优先级顺序执行任务, 并返回执行结果;

(4) 日志管理: 记录系统在任务执行过程中产生的日志.

DTES 作为一个完善的系统, 除了满足上述功能性需求外, 还应该满足一定的非功能性需求[4], 如安全性、兼容性、健壮性、可扩展性等.

系统流程图如图 1 所示.

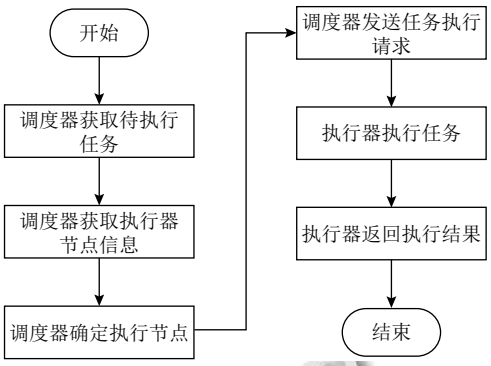

图 1 DTES 流程图

3 系统设计与实现

3.1 总体设计

3.1.1 系统架构设计

深度学习全流程托管平台采用微服务架构[5,6]实现, 分布式任务执行系统 DTES 最终会作为一个微服务集成到平台中. DTES 选用 Spring Boot 作为框架, 通过 Spring Cloud 提供的注册中心组件 Eureka 将 DTES 作为一个微服务注册到深度学习平台的 Eureka 服务端, 实现分布式任务执行系统和其他微服务之间的相互访问, 通过 Ribbon 组件实现 HTTP 负载均衡, 通过 Hystrix 实现容错处理.

DTES 采用模块化设计, 根据系统的需求分析, 将系统划分为监控、任务调度、任务执行、日志管理 4 大功能模块, 其中每个模块又包含相应的子模块. 系统的功能模块分解图如图 2 所示.

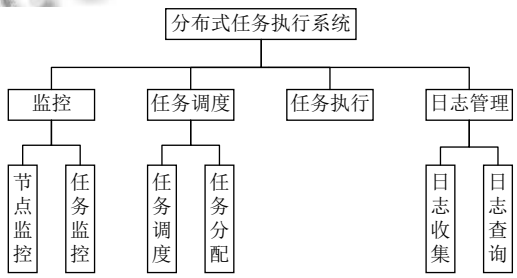

图 2 DTES 功能模块图

DTES 采用分层模式, 系统整体分为数据层、业务层和接口层 3 个层级. 层级之间相互配合, 共同实现任务执行功能. 系统架构图如图 3 所示.

(1) 数据层: 数据层包括 MySQL 数据库、Redis 数据库和 RabbitMQ 消息中间件. 其中 MySQL 主要用于存储任务的基本信息, 如任务的状态、工作目录、环

境变量等; Redis 用于全局的数据缓存; RabbitMQ 用于消息同步和存放任务执行结果, 供深度学习平台的其他微服务进行消费.

(2) 业务层: 业务层包括系统的业务逻辑实现. 其中监控模块负责监控执行器节点的资源使用情况和任务的运行状态; 任务调度模块接收提交的任务并添加到任务资源库, 根据系统中的调度策略从资源库中读取任务的基本信息并依此确定执行器节点, 最后将任务发送给执行器; 任务执行模块接收执行请求并执行具体的任务, 将执行结果返回给任务调度模块; 日志管理模块则主要负责收集任务执行过程中产生的日志信息, 以便于后续对任务进行分析.

(3) 接口层: 系统使用 REST (REpresentational State Transfer, 表述性状态转移) 协议对外提供接口服务, 使用 JSON 进行数据交互. 系统对外开放的接口主要有任务提交接口和日志查询接口.

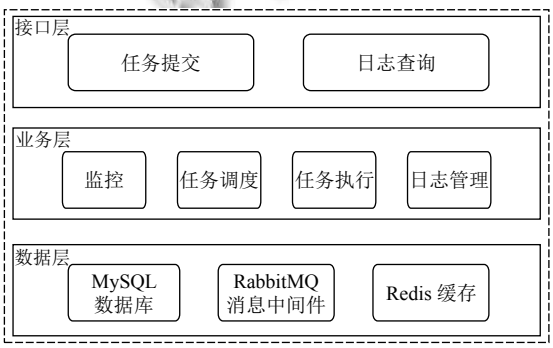

图 3 DTES 架构图

3.1.2 数据库设计

数据库设计是设计阶段的重要一环, 分布式任务执行系统选用 MySQL 数据库实现数据的持久化. 根据系统的业务功能设计, 系统共涉及任务、日志、执行器节点等多个实体, 现将系统中重要的数据表描述如表 1 和表 2 所示.

表 1 任务数据表

字段名	数据类型	描述
id	varchar (64)	主键
priority	int	优先级
status	varchar (16)	状态
image	varchar (256)	镜像名
directory	varchar (256)	工作目录
volume	varchar (256)	数据卷
command	varchar (1024)	启动命令
environment	varchar (1024)	环境变量

表 2 日志数据表

字段名	数据类型	描述
id	int	主键, 自动增长
time	bigint	产生时间
type	varchar (8)	类型
content	longtext	内容

3.2 详细设计

3.2.1 任务调度模块设计与实现

任务调度是 DTES 的重要功能之一, 整个系统的有序稳定运行得益于系统合理有效的调度策略[7–11]. 除了常见的任务队列和随机分配方法, 系统还实现了依据执行器资源利用率进行调度的策略. DTES 为监控模块[12–14]的核心类 GlobalHardwareMonitor 的 update 方法添加 Spring 的心跳机制@Scheduled (fixedDelay = 5000) 实现对各个节点执行器信息的实时更新. @Scheduled 的参数定义了 update 方法执行的时间规则, 如 fixedDelay = 5000 表示该方法从上一次执行完开始计时, 经过固定的延迟时间 5000 ms 后执行下一次. 在 update 方法中订阅执行器资源利用信息, 并依据利用率进行任务调度.

任务调度模块的核心类图如图 4 所示. 在 ScheduleThreadSupport 类中调用 TaskManager 接口的 schedule 方法进行任务调度, schedule 方法通过上行链路服务 UpstreamService 发送 HTTP GET 请求至下行链路控制器 ClientController 加载任务信息. 此外, UpstreamService 中注入了硬件监控服务 GlobalHardwareMonitor, 依据任务信息和硬件资源利用率确定执行节点, 并将结果返回给 TaskManager.

3.2.2 任务执行模块设计与实现

顾名思义, 任务执行模块[15–17] 负责任务的具体执行. 考虑到深度学习平台的实际需求以及深度学习实验的特性, DTES 现阶段的任务执行主要基于 Docker 容器[18] 的方式. 用户根据具体的任务自定义 Docker 镜像并将镜像上传至深度学习平台, 深度学习平台的其他微服务可以接收镜像并保存, 由于不是本文的重点, 具体过程不再详细陈述.

任务执行模块的核心类图如图 5 所示. 接口 TaskManager 中声明了任务执行的必需方法, 它有两个实现类, 分别是 ThreadPoolTaskManager 和 AbstractDockerTaskManager, 后者是对 Docker 容器任务管理器的抽象, 它提供了用于获取任务镜像的方法 pullImage. 若后续需要支持其他类型的任务, 只要实现该接口进行扩

展即可. DefaultTaskManagerImpl 是容器任务管理的具体实现类, UpstreamService 使用 Spring 的@Autowired 注入进来, 实现了启动方法 start 和终止方法 kill 等任务处理方法.

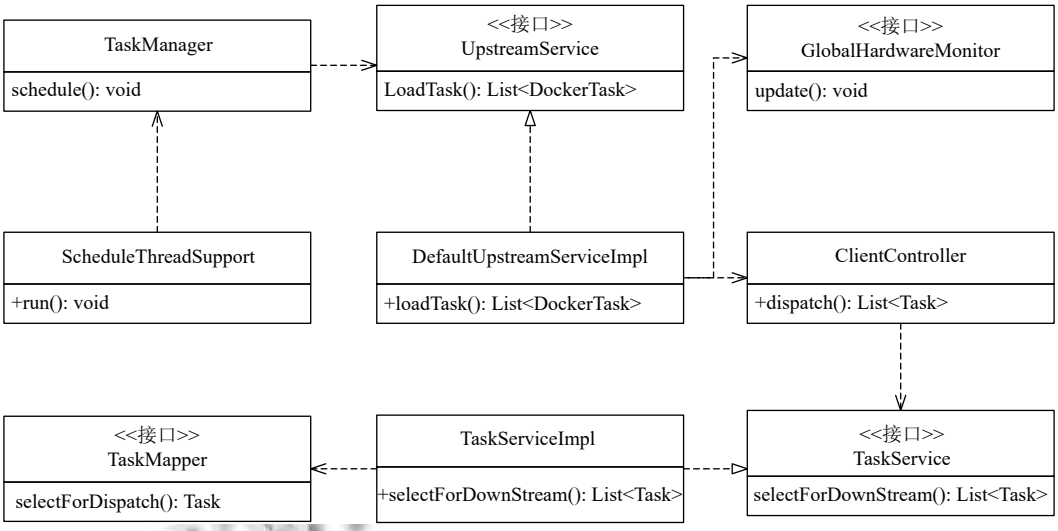

图 4 DTES 任务调度核心类图

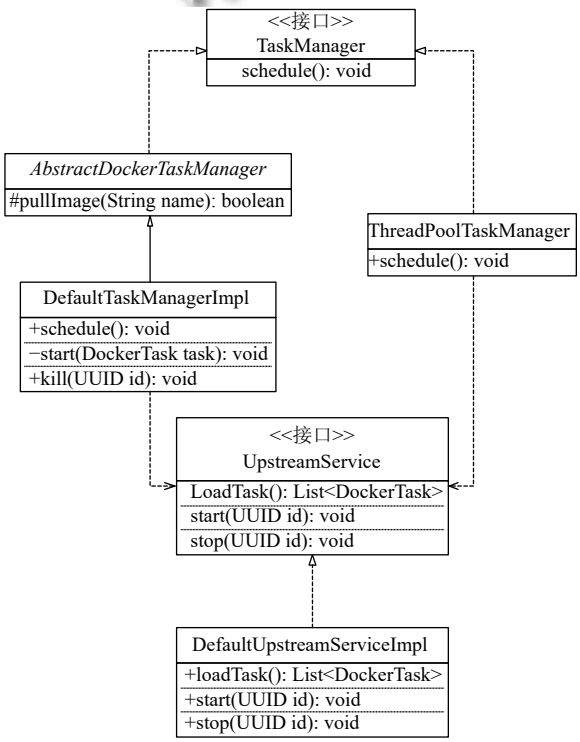

图 5 任务执行核心类图

任务执行的具体流程如下:

(1) TaskManager 调用 UpstreamService 的 loadTask 方法加载任务, loadTask 使用 Spring 的 RestTemplate 方式发送 HTTP GET 请求至下行链路控制器 ClientController, ClientController 调用 TaskService 查询数据库将结果返回;

(2) TaskManager 调用 UpstreamService 的 prepare 方法更新任务的当前状态, prepare 使用 Spring 的 RestTemplate 方式发送 HTTP POST 请求至任务控制器 TaskController, TaskController 将任务的状态由 ACCEPTED(被执行器接受) 改为 PREPARING(任务数据准备中);

(3) 检查任务所需镜像是否存在, 若不存在则进行拉取;

(4) 设置数据卷等信息;

(5) 创建 Docker 容器;

(6) 开启日志收集;

(7) 启动 Docker 容器, 执行任务.

3.2.3 日志管理模块设计与实现

作为系统的支撑模块, 日志管理模块在分布式任务执行系统中也起着重要的作用[19–21]. 日志管理模块收集了任务执行期间产生的日志信息, 给用户对任务结果的分析总结带来极大的便利.

日志管理模块的核心类图如图 6 所示. TaskLogReceiver 类用于收集任务日志, 当有当异步事件结果产生时就会调用该类的 onNext 方法, onNext 方法通过 Builder 模式实例化一个 LogRecord 对象, 记录下日志产生的时间、日志级别、日志具体内容等信息. DTES 的日志级别分为 3 种, 分别是标准输出 STDOUT、标准

错误 STDERR 和纯日志输出 RAW. 实例化的 LogRecord 对象通过日志服务 LogService 传到上行链路服务 UpstreamService, 之后用 RestTemplate 方式通过 HTTP POST 请求转发给 LogController, LogController 调用日志存储服务 LogStorageService 将任务日志保存到 MySQL 数据库中. LogController 对外开放了日志查询 API, 深度学习全流程托管平台的其他相关微服务可以调用该 API 将日志信息展示给用户.

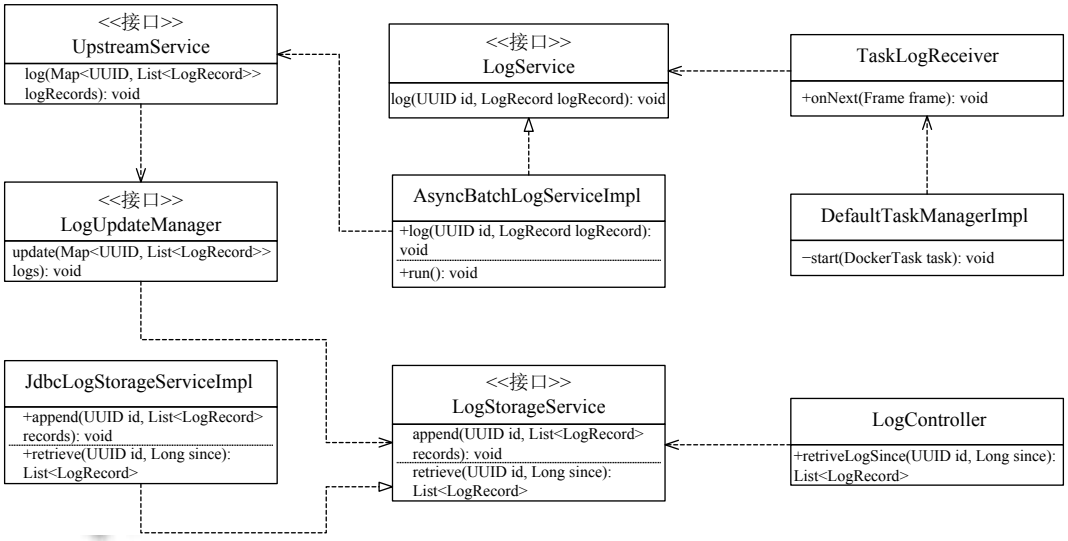

图 6 DTES 日志管理核心类图

代码 1. 任务执行核心代码

```
public void start(DockerTask task)
    {DockerTask task = upstreamService.loadTask();
    upstreamService.prepare(task.getTaskId());
    TaskContext.Builder contextBuilder = TaskContext.Builder.
    aTaskContext();
    contextBuilder.withDockerTask(task);
    prepareImage(task.getImage());
    List<Bind> volumeBinds = new LinkedList<>();
    List<String> temporaryVolumes = new LinkedList<>();
    prepareVolume(volumeBinds, temporaryVolumes);
    contextBuilder.withTemporaryVolumes(temporaryVolumes);
    Map<String, String> labels = new Hashtable<>();
    labels.put(TAG_TYPE, VAL_TASK_CONTAINER);
    labels.put(TAG_TASK_ID, task.getTaskId().toString());
    CreateContainerResponse containerResponse = dockerClient.
    createContainerCmd(task.getImage())
        .withWorkingDir(task.getDirectory())
        .withCmd(task.getCommand())
        .withEnv(task.getEnvironment())
        .withAttachStdout(true).withAttachStderr(true)
        .withLabels(labels)
        .withHostConfig(
            HostConfig.newHostConfig()
                .withBinds(volumeBinds)
                .withAutoRemove(true)
    )
    .exec();
    contextBuilder.withContainerId(containerResponse.getId());
    TaskLogReceiver logReceiver = applicationContext.getBean
    (TaskLogReceiver.class, task.getTaskId());
    dockerClient.attachContainerCmd(containerResponse.getId())
        .withStdOut(true)
        .withStdErr(true)
        .withFollowStream(true)
        .exec(logReceiver);
    contextBuilder.withLogAdapter(logReceiver);
    this.taskContextHolder.put(task.getTaskId(), contextBuilder.build());
    dockerClient.startContainerCmd(containerResponse.getId()).exec();
}
```

4 系统测试

4.1 接口测试

接口测试用来测试系统与外界之间以及系统内部各个模块之间的交互, 主要测试系统的依赖关系以及数据传递等. 使用 Postman 对 DTES 任务提交接口的测试如图 7 所示.

4.2 功能测试

DTES 作为子系统测试通过后, 集成到整个深度学习平台中. 深度学习平台部署在由 3 台物理机构成的集群中, 每台机器的硬件配置为 CPU Intel Core i7@2.40 GHz, 内存 8 GB, 硬盘 100 GB, 操作系统 Ubuntu 16.04 LTS.

本节以深度学习模型训练任务为例, 对 DTES 的功能进行测试.

如图 8 所示, 用户进入工作台页面, 在左侧的目录树中将任务需要使用的图片数据和标签数据挂载进来, 接着创建程序文件并在右侧区域进行代码编码. 编码完成后, 鼠标右键点击左侧项目的根目录选择“创建训练任务”, 创建成功后, DTES 在后台进行调度执行. 切换到任务详情页面, 可以看到任务的实时日志信息, 如图 9 所示.

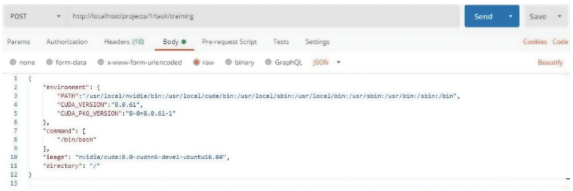

图 7 接口测试

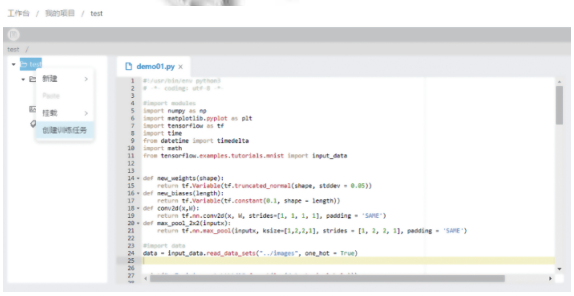

图 8 创建任务

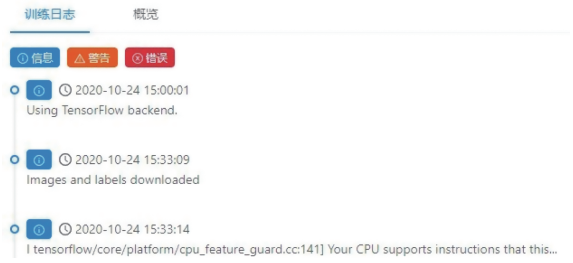

图 9 日志查询

4.3 性能测试

为测试 DTES 的性能, 本节使用 TensorFlow 框架在已集成 DTES 的深度学习平台与未集成 DTES 的第一代平台中多次训练手写数字 MNIST 数据集, 并对完成同一训练任务的平均用时进行比较. 实验数据如表 3 所示.

通过对比可以看出, 对于同样的深度学习训练任务, DTES 可以节约大约 23.8% 的时间, 具有较高的应用价值.

表 3 不同平台完成同一训练任务的平均耗时 (单位: s)

平台	单隐藏层神经网络	双隐藏层神经网络	三隐藏层神经网络
使用DTES	786	2424	4309
未使用DTES	1031	3217	5610

5 总结与展望

使用深度学习平台在网页端进行深度学习模型训练可以给实验研究人员带来极大的便利, 本文结合平台的实际需要设计并实现了分布式任务执行系统 DTES. DTES 基于 Spring Boot 框架实现任务调度、任务执行、日志管理等功能, 可以作为一个微服务快速集成到深度学习平台中. DTES 将接收的任务按照既定的调度策略进行调度并创建 Docker 容器进行执行, 同时将任务执行期间产生的日志信息反馈给实验研究人员. 经过系统测试, DTES 已经达到预期目标, 但仍然存在待完善之处, 接下来要继续对该系统进行扩展, 使其可以支持更多类型的任务.

参考文献

1 张琦, 张荣梅, 陈彬. 基于深度学习的图像识别技术研究综述. 河北省科学院学报, 2019, 36(3): 28–36. [doi: 10.16191/j.cnki.hbkx.2019.03.004]

2 殷琪林, 王金伟. 深度学习在图像处理领域中的应用综述. 高教学刊, 2018, (9): 72–74.

3 陈勇涛, 郭晓颖, 陶慧杰. 基于深度学习的图像识别模型研究综述. 电子世界, 2018, (4): 65–66. [doi: 10.19353/j.cnki.dzsj.2018.04.018]

4 林文. 实验室图形化深度学习开发平台. 福建电脑, 2020, 36(8): 107–109. [doi: 10.16707/j.cnki.fjpc.2020.08.032]

5 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]

6 刘从军, 刘毅. 基于微服务的维修资金管理系统. 计算机系统应用, 2019, 28(4): 52–60. [doi: 10.15888/j.cnki.csa.006843]

7 Yao Y, Gao H, Wang JY, *et al*. New scheduling algorithms for improving performance and resource utilization in Hadoop YARN clusters. IEEE Transactions on Cloud Computing. [doi: 10.1109/TCC.2019.2894779]

8 Saleh H, Nashaat H, Saber W, *et al*. IPSO task scheduling

algorithm for large scale data in cloud computing environment. IEEE Access, 2019, 7: 5412–5420. [doi: 10.1109/ACCESS.2018.2890067]

9 Mirzayi S, Rafe V. A hybrid heuristic workflow scheduling algorithm for cloud computing environments. Journal of Experimental & Theoretical Artificial Intelligence, 2015, 27(6): 721–735. [doi: 10.1080/0952813X.2015.1020524]

10 樊程, 苏若凡. 基于负载均衡的任务调度优化算法. 计算机工程与设计, 2017, 38(6): 1532–1535. [doi: 10.16208/j.issn1000-7024.2017.06.024]

11 葛维春, 叶波. 负载均衡优先的改进优先级表调度算法. 沈阳工业大学学报, 2017, 39(3): 241–247. [doi: 10.7688/j.issn.1000-1646.2017.03.01]

12 彭振华, 苟伟强, 张兰英. 基于 Nagios 的气象设备监控系统. 计算机系统应用, 2017, 26(8): 60–65. [doi: 10.15888/j.cnki.csa.005916]

13 朱瑞斌. 服务器集群监控系统的设计与实现 [硕士学位论文]. 北京: 北京交通大学, 2015. [doi: 10.7666/d.Y2917557]

14 杨杰, 曾凌波, 彭运勇, 等. 面向大规模集群的自动化监控系统. 计算机工程与科学, 2020, 42(10): 1801–1806. [doi: 10.3969/j.issn.1007-130X.2020.10.012]

15 王昆. 高可用分布式任务调度与执行系统设计与实现 [硕士学位论文]. 西安: 西安电子科技大学, 2019. [doi: 10.27389/d.cnki.gxadu.2019.001426]

16 林萌. 高性能分布式一致性协调服务系统 [硕士学位论文]. 成都: 电子科技大学, 2016.

17 凌少虎. 面向视频分析的任务管理与执行系统 [硕士学位论文]. 广州: 华南理工大学, 2015.

18 刘熙, 胡志勇. 基于 Docker 容器的 Web 集群设计与实现. 电子设计工程, 2016, 24(8): 117–119. [doi: 10.14022/j.cnki.dzsjgc.2016.08.033]

19 翟雅荣, 于金刚. 基于 Filebeat 自动收集 Kubernetes 日志的分析系统. 计算机系统应用, 2018, 27(9): 81–86. [doi: 10.15888/j.cnki.csa.006528]

20 屈国庆. 基于 Storm 的实时日志分析系统的设计与实现 [硕士学位论文]. 南京: 南京大学, 2016.

21 王军利, 杨卫中. 基于 ELK 架构的日志分析系统研究与实践. 中国信息化, 2020, (9): 50–51. [doi: 10.3969/j.issn.1672-5158.2020.09.022]