(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 113687989 B
(45)授权公告日 2024.08.16

(21)申请号 202110908593.1
(22)申请日 2021.08.09
(65)同一申请的已公布的文献号
申请公布号 CN 113687989 A
(43)申请公布日 2021.11.23
(73)专利权人 华东师范大学
地址 200241 上海市闵行区东川路500号
(72)发明人 陈铭松 周亮 黄红兵 焦阳
马言悦 戴静安
(74)专利代理机构 上海德禾翰通律师事务所 31319
专利代理师 夏思秋
(51)Int.Cl.
G06F 11/36(2006.01)
G06F 18/214(2023.01)
G06F 18/2433(2023.01)
G06N 5/04(2023.01)
G16Y 30/00(2020.01)
G16Y 40/10(2020.01)

(56)对比文件
CN 111818155 A,2020.10.23
US 2017102978 A1,2017.04.13

审查员 李宇文

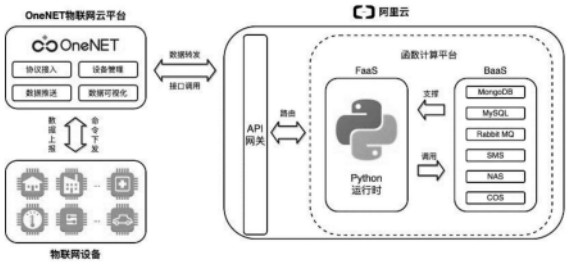
权利要求书2页 说明书16页 附图4页

(54)发明名称

一种基于无服务器架构的物联网数据异常检测方法及系统

(57)摘要

本发明公开了一种基于无服务器架构的物联网数据异常检测方法,所述方法包括:采用第三方物联网云平台作为物联网设备的接入端,所有物联网设备通过特定协议接入第三方物联网平台,并将采集的数据上报到第三方物联网平台;用户为接入系统的任一物联网设备选择合适的异常检测算法并配置相应的参数;采用无服务器架构实现数据异常检测端构建,根据算法和参数为设备自动生成和部署异常检测用的训练云函数和推理云函数。训练云函数训练异常检测模型并将其供推理云函数调用。部署完异常检测算法的设备上报的新数据点转发至推理云函数进行异常检测,并返回检测结果。本发明还提出了一种基于无服务器架构的物联网数据异常检测系统。



CN 113687989 B

1.一种基于无服务器架构的物联网数据异常检测方法,其特征在于,所述方法包括如下步骤:

步骤1:使用第三方物联网云平台作为物联网设备的接入端,所有物联网设备通过特定协议接入第三方物联网云平台,并将采集的数据上报到第三方物联网云平台;

步骤2:用户为接入系统的任一物联网设备选择合适的异常检测算法并配置相应的参数;

步骤3:采用无服务器架构实现数据异常检测端的构建,根据算法和参数为设备自动生成和部署异常检测用的训练云函数和推理云函数;部署完毕后,训练云函数会根据设备的历史数据训练异常检测模型并将其保存在NAS文件存储服务中供推理云函数调用;

步骤4:部署完异常检测算法的设备上报的新数据点都会通过第三方物联网云平台数据推送服务转发至推理云函数进行异常检测,推理云函数从NAS文件存储服务中读取训练好的异常检测模型对新数据进行异常检测,并返回检测结果。

2.如权利要求1所述的基于无服务器架构的物联网数据异常检测方法,其特征在于,所述第三方物联网云平台是指中国移动OneNET、阿里云IoT、AmazonIoT;所述设备接入端采用第三方物联网云平台实现,所述数据异常检测端采用阿里云函数计算平台实现。

3.如权利要求1所述的基于无服务器架构的物联网数据异常检测方法,其特征在于,步骤1中,所述特定协议具体是指通信协议,包括MQTT、Modbus、HTTP;所述采集的数据是物联网设备在不同场景下采集到的各类数值型数据。

4.如权利要求1所述的基于无服务器架构的物联网数据异常检测方法,其特征在于,步骤2中,所述异常检测算法包括IForest、CBLOF、OCSVM、AutoEncoder、DBSCAN;所述配置的参数包括数据污染率及各异常检测算法所需的训练参数。

5.如权利要求1所述的基于无服务器架构的物联网数据异常检测方法,其特征在于,步骤3中,所述检测模型的训练和推理都通过无服务器架构中的云函数实现,所述云函数按调用次数计费。

6.如权利要求1所述的基于无服务器架构的物联网数据异常检测方法,其特征在于,所述训练云函数和推理云函数都是根据模板自动生成并部署到无服务器计算平台上的;所述模板以文本文件的方式预置在系统内,其中的具体参数以模板字符串的方式占位;所述训练云函数和推理云函数是在算法部署时再根据前端传递的具体参数替换模板中的占位符自动生成的,通过阿里云函数计算平台提供的SDK实现自动部署。

7.一种实现如权利要求1-6之任一项所述方法的系统,其特征在于,所述系统包括数据转存模块,算法部署模块,云函数自动生成与部署模块以及训练和推理模块。

8.如权利要求7所述的系统,其特征在于,所述数据转存模块负责接收第三方物联网云平台推送过来的物联网设备数据,并存储到数据库中供后续部署算法和训练模型使用;

所述算法部署模块以Web页面形式为系统用户提供一个部署异常检测算法的窗口,系统用户在其中修改或输入需要选用的异常检测算法及相应参数;

所述云函数自动生成与部署模块负责根据模板和前端传递的参数自动生成训练云函数和推理云函数的代码,并通过函数计算平台提供的SDK自动部署至云端;

所述训练和推理模块以训练云函数和推理云函数的形式部署在阿里云函数计算平台中,分别负责训练异常检测模型和调用模型进行异常检测。

9.一种实现如权利要求1-6之任一项所述方法的硬件系统,其特征在于,所述硬件系统包括:存储器和处理器;所述存储器上存储有计算机程序,当所述计算机程序被所述处理器执行时,实现如权利要求1-6任一项所述的方法。

10.一种计算机可读存储介质,其上存储有计算机程序,其特征在于,所述计算机程序被处理器执行时,实现如权利要求1-6任一项所述的方法。

一种基于无服务器架构的物联网数据异常检测方法及系统

技术领域

[0001] 本发明属于计算机技术领域,涉及一种基于无服务器架构的物联网数据异常检测方法及系统。

背景技术

[0002] 随着科学技术与经济建设的不断推进,物联网技术正蓬勃发展。一些专家学者提出了人工智能物联网的概念,旨在通过人工智能技术赋能传统物联网产业。物联网数据的异常检测是物联网与人工智能结合应用的领域之一。物联网系统由海量资源受限的终端节点构成,这些节点实时地将物理信息转换为数字信息并发送至云端或者边缘端存储以供人工智能应用分析使用。然而,在实际场景中由于外部环境、网络通信、设备故障或者恶意攻击等因素的影响,物联网设备采集到的数据中往往包含着异常——即与正常模式中的数据存在明显偏差的数据值。物联网场景中的异常检测是十分重要的,异常能提示外部环境突变、传感器节点故障、设备被入侵或攻击等有价值事件的发生,这对于提升物联网系统整体的安全性、稳定性,以及所分析数据的准确性都有重要意义。

[0003] 尽管传统的物联网云平台在设备接入和管理、数据采集、指令下发等功能上相当成熟,但是它们都不支持机器学习算法的部署。机器学习、深度学习技术是目前大数据处理和分析的重要方式之一,但以OneNET为代表的传统物联网云平台只支持通过SQL查询的方式对设备数据进行查询和处理,难以满足如今物联网应用的需求。因此,如何将机器学习、深度学习等新技术与传统物联网云平台相结合用于异常检测任务,是本发明要解决的一个问题。

[0004] 传统的物联网数据异常检测系统通常部署在IaaS虚拟机或PaaS应用平台上,开发人员在部署异常检测程序前需要根据预估系统的请求量,根据预估的请求预先购买一定量的云计算资源(计算资源、存储资源、网络资源等)用于支撑系统的运行。然而,这种方式存在以下问题:首先,物联网设备通常以固定频率周期性地上报数据,采用上述方式部署的异常检测程序需要7*24小时运行在服务器后台,在没有数据上报的时间段内会造成计算资源的极大浪费,产生“闲时计费”问题;其次,根据预估请求量购买的云计算资源在请求高峰时刻可能无法满足服务需要,在请求低谷时刻也会造成资源的浪费。如何避免上述问题是本发明要解决的另一个问题。

发明内容

[0005] 为了解决现有技术存在的不足,本发明的目的是提供一种基于无服务器架构的物联网数据异常检测方法及系统。

[0006] 本发明提供的方法包括如下步骤:

[0007] 步骤1:使用第三方物联网云平台作为物联网设备的接入端,所有物联网设备通过特定协议接入第三方物联网云平台,并将采集的数据上报到第三方物联网云平台;

[0008] 步骤2:用户为接入系统的任一物联网设备选择合适的异常检测算法并配置相应

的参数;

[0009] 步骤3:采用无服务器架构实现数据异常检测端的构建,根据算法和参数为设备自动生成和部署异常检测用的训练云函数和推理云函数;部署完毕后,训练云函数会根据设备的历史数据训练异常检测模型并将其保存在NAS文件存储服务中供推理云函数调用;

[0010] 步骤4:部署完异常检测算法的设备上报的新数据点都会通过第三方物联网云平台数据推送服务转发至推理云函数进行异常检测,推理云函数从NAS文件存储服务中读取训练好的异常检测模型对新数据进行异常检测,并返回检测结果。

[0011] 所述物联网云平台是指中国移动OneNET、阿里云IoT、Amazon IoT等第三方物联网云平台;所述设备接入端采用第三方物联网云平台实现,所述数据异常检测端采用阿里云函数计算平台实现。

[0012] 步骤1中,所述特定协议具体是指MQTT、Modbus、HTTP等通信协议;所述采集的数据是物联网设备在不同场景下采集到的各类数值型数据。

[0013] 步骤2中,所述异常检测算法包括IForest、CBLOF、OCSVM、AutoEncoder、DBSCAN等;所述配置的参数包括数据污染率及各异常检测算法所需的训练参数等。

[0014] 步骤3中,所述检测模型的训练和推理都通过无服务器架构中的云函数实现,所述云函数按调用次数计费。

[0015] 所述训练云函数和推理云函数都是根据模板自动生成并部署到无服务器计算平台上的;所述模板以文本文件的方式预置在系统内,其中的具体参数以模板字符串的方式占位;所述训练云函数和推理云函数是在算法部署时再根据前端传递的具体参数替换模板中的占位符自动生成的,通过阿里云函数计算平台提供的SDK实现自动部署。

[0016] a)基于传统物联网云平台的设备接入端实现方法:

[0017] 本发明采用中国移动OneNET物联网云平台作为设备的接入端,所有物联网设备均采用特定协议接入OneNET云平台,所有物联网设备采集的数据均报送至OneNET云平台,通过OneNET云平台对设备进行管理,通过OneNET云平台的数据推送功能将数据推送至异常检测端进行异常检测。OneNET是中移物联推出的物联网应用开发平台,旨在为物联网应用程序提供安全可靠的设备连接通信能力,支持MQTT、TCP、EDP、NB-IoT、Modbus等多种设备协议,为物联网场景中的各类硬件终端提供了设备接入和管理、数据采集、命令下发等服务,并为应用开发提供了丰富的API和SDK。

[0018] b)基于无服务器架构的异常检测端实现方法:

[0019] 为了将机器学习、深度学习等技术与OneNET云平台这样的传统物联网云平台相结合,进一步降低系统的部署和运行成本,本发明采用无服务器架构实现系统的异常检测端。无服务器并不是指不再需要后端服务器,而是指开发者可以摆脱开发应用程序所需要的服务器设置和管理工作,将这些操作交由云服务商来完成,开发人员自己则专注于具体业务的开发。无服务器架构是一种以事件驱动的云计算模式,通过在云上部署一系列的“云函数”并绑定相应的事件源,将其组织为微服务来支撑应用程序。云函数的每次触发执行都会被调度到一个新的临时容器中,执行完毕之后一段时间内该运行环境会被云服务商回收,因此云函数本身是无状态的。无状态特性使得云函数在请求高峰时可以迅速在多个容器中被实例化,从而支撑大规模并发请求。在没有请求时云函数不会被调度执行,当有突发的大量请求时云服务商会创建足够的运行实例来满足需求。由于云函数的调度执行是毫秒级别

的,因此无服务器架构的自动缩扩容特性可以近乎完美地贴合负载变化。云函数一般按照调用次数或调用时间计费,未被调用时不产生任何费用,解决了传统云服务“闲时计费”的问题,大大节约了使用成本。本发明实现的异常检测系统部署在阿里云函数计算平台上,阿里云函数计算平台是阿里云提供的无服务器架构云服务。

[0020] c)异常检测系统中训练云函数与推理云函数自动生成部署的方法:

[0021] 由于采用了无服务器架构,本发明实现的异常检测系统的异常检测端均以云函数的方式部署在阿里云函数计算平台上(云函数的结构如图2所示),在为物联网设备部署完异常检测算法之后,会在阿里云函数计算平台上为每台设备部署一个训练云函数与推理云函数,分别用于异常检测模型的训练,以及调用训练好的异常检测模型对新数据进行异常检测。本发明实现了训练云函数与推理云函数的自动生成与部署,本发明实现的异常检测系统中支持的所有算法都包含训练云函数模板与推理云函数模板两份模板,在部署算法时会将相应的算法参数填充到模板中生成对应的训练云函数代码和推理云函数代码,再通过阿里云函数计算平台提供的SDK进行自动部署,如图5所示。

[0022] 本发明采用OneNET物联网云平台作为物联网设备的接入端,所有物联网设备通过特定协议与OneNET云平台相连,并报送数据至OneNET平台,通过OneNET平台的数据推送功能转发至无服务器异常检测端进行异常检测。

[0023] 本发明采用阿里云函数计算平台提供的无服务器架构云服务实现数据的异常检测端,主要由与设备一一对应的训练云函数和推理云函数进行异常检测模型的训练与调用。无服务器架构中的云函数根据调用次数收费,不被调用时不计费,大大降低了成本。

[0024] 本发明支持多种异常检测算法的部署,所有基于Python实现的异常检测算法程序都可以从训练代码和推理代码中抽取出算法主体保存为模板,在模型部署时再选择具体算法,将具体算法的参数填充到模板中生成训练云函数代码和推理云函数代码。

[0025] 本发明支持训练云函数与推理云函数的自动部署,由模板生成的训练云函数代码和推理云函数代码能通过阿里云函数计算平台提供的SDK自动部署至云端,不需要使用人员手动部署。

[0026] 本发明中基于无服务器架构的物联网数据异常检测系统的时序如图3所示,包括:1.系统用户创建虚拟设备,并设置消息推送;2.第三方物联网平台反馈设置成功;3.物联网设备通过特定协议接入平台;4.物联网设备上传一定量的设备数据;5.第三方物联网云平台转发数据到阿里云FaaS;6.阿里云FaaS反馈接收成功;7.阿里云FaaS储存数据到阿里云BaaS;8.系统用户为设备配置并部署异常检测算法;9.阿里云FaaS保存设备的算法配置;10.阿里云FaaS请求设备的历史数据;11.阿里云BaaS返回设备的历史数据;12.阿里云FaaS训练并导出检测模型;13.物联网设备上传新数据到第三方物联网云平台;14.第三方物联网云平台转发数据并触发检测;15.阿里云FaaS反馈接收成功;16.阿里云BaaS加载检测模型;17.阿里云FaaS进行异常告警;18.第三方物联网云平台对物联网设备进行命令下发;19.第三方物联网云平台对系统用户进行异常告警。

[0027] 本发明还提供了一种实现上述方法的系统,所述系统包括数据转存模块,算法部署模块,云函数自动生成与部署模块以及训练和推理模块。

[0028] 所述数据转存模块负责接收第三方物联网云平台推送过来的物联网设备数据,并存储到数据库中供后续部署算法和训练模型使用;

[0029] 所述算法部署模块以Web页面形式为系统用户提供一个部署异常检测算法的窗口,系统用户可以在其中修改或输入需要选用的异常检测算法及相应参数;

[0030] 所述云函数自动生成与部署模块负责根据模板和前端传递的参数自动生成训练云函数和推理云函数的代码,并通过函数计算平台提供的SDK自动部署至云端;

[0031] 所述训练和推理模块以训练云函数和推理云函数的形式部署在阿里云函数计算平台中,分别负责训练异常检测模型和调用模型进行异常检测。

[0032] 本发明还提出了一种实现上述方法的硬件系统,包括:存储器和处理器;所述存储器上存储有计算机程序,当所述计算机程序被所述处理器执行时,实现上述方法。

[0033] 本发明还提出了一种计算机可读存储介质,其上存储有计算机程序,所述计算机程序被处理器执行时,实现上述方法。

[0034] 本发明的有益效果包括:本发明结合传统的物联网云平台和新兴的无服务器架构,通过采用传统物联网云平台作为设备接入端降低了设备接入与管理的成本,通过无服务器架构构建异常检测端大幅降低了系统的部署和运行成本。无服务器架构中的云函数按需执行,不像传统云服务器一样有“闲时计费”问题,非常适合物联网这样请求呈现低频、周期性特点的场景。且无服务器架构在并发性能、响应速度上相比传统云计算模式更有优势,能更快地检测出物联网设备上报的异常数据。在现有的技术中,尚没有通过引入无服务器架构来降低物联网数据异常检测系统的成本、提高系统检测速度的。

附图说明

[0035] 图1是本发明基于无服务器架构的物联网数据异常检测系统的架构图。

[0036] 图2是本发明基于无服务器架构的物联网数据异常检测系统中云函数的结构图。

[0037] 图3是本发明基于无服务器架构的物联网数据异常检测系统的时序图。

[0038] 图4是本发明设备算法部署模块的操作界面示意图。

[0039] 图5是本发明训练与推理云函数的自动生成部署的示意图。

具体实施方式

[0040] 结合以下具体实施例和附图,对本发明作进一步的详细说明。实施本发明的过程、条件、实验方法等,除以下专门提及的内容之外,均为本领域的普遍知识和公知常识,本发明没有特别限制内容。

[0041] OneNET平台是中移物联网有限公司基于物联网技术和产业特点打造的物联网开放平台,支持各类传感器和智能硬件的快速接入和大数据服务,提供了丰富的API和应用模板以支持各类行业应用和智能硬件的开发,能够有效降低物联网应用开发和部署成本。设备端可以通过多种传输协议将数据上传至云端。

[0042] 因此,采用中国移动OneNET物联网云平台作为设备的接入端,所有物联网设备均采用特定协议接入OneNET云平台,所有物联网设备采集的数据均报送至OneNET云平台,通过OneNET云平台对设备进行管理,通过OneNET云平台的数据推送功能将数据推送至异常检测端进行异常检测。

[0043] 本发明中基于无服务器架构的物联网数据异常检测方法及系统,包括以下内容:

[0044] 1、采用传统物联网云平台实现设备接入端:

[0045] 本发明采用中国移动OneNET物联网云平台作为设备的接入端,所有物联网设备均采用特定协议接入OneNET云平台,所有物联网设备采集的数据均报送至OneNET云平台,通过OneNET云平台对设备进行管理,通过OneNET云平台的数据推送功能将数据推送至异常检测端进行异常检测。OneNET是中移物联推出的物联网应用开发平台,旨在为物联网应用程序提供安全可靠的设备连接通信能力,支持MQTT、TCP、EDP、NB-IoT、Modbus等多种设备协议,为物联网场景中的各类硬件终端提供了设备接入和管理、数据采集、命令下发等服务,并为应用开发提供了丰富的API和SDK。

[0046] 2、采用无服务器架构实现异常检测端:

[0047] 为了将机器学习、深度学习等技术与OneNET云平台这样的传统物联网云平台相结合,进一步降低系统的部署和运行成本,本发明采用无服务器架构实现系统的异常检测端。无服务器并不是指不再需要后端服务器,而是指开发者可以摆脱开发应用程序所需要的服务器设置和管理工作,将这些操作交由云服务商来完成,开发人员自己则专注于具体业务的开发。无服务器架构是一种以事件驱动的云计算模式,通过在云上部署一系列的“云函数”并绑定相应的事件源,将其组织为微服务来支撑应用程序。云函数的每次触发执行都会被调度到一个新的临时容器中,执行完毕之后一段时间内该运行环境会被云服务商回收,因此云函数本身是无状态的。无状态特性使得云函数在请求高峰时可以迅速在多个容器中被实例化,从而支撑大规模并发请求。在没有请求时云函数不会被调度执行,当有突发的大量请求时云服务商会创建足够的运行实例来满足需求。由于云函数的调度执行是毫秒级别的,因此无服务器架构的自动缩扩容特性可以近乎完美地贴合负载变化。云函数一般按照调用次数或调用时间计费,未被调用时不产生任何费用,解决了传统云服务“闲时计费”的问题,大大节约了使用成本。本发明实现的异常检测系统部署在阿里云函数计算平台上,阿里云函数计算平台是阿里云提供的无服务器架构云服务。

[0048] 3、支持训练云函数与推理云函数的自动生成部署:

[0049] 由于采用了无服务器架构,本发明实现的异常检测系统的异常检测端均以云函数的方式部署在阿里云函数计算平台上,在为物联网设备部署完异常检测算法之后,会在阿里云函数计算平台上为每台设备部署一个训练云函数与推理云函数,分别用于异常检测模型的训练,以及调用训练好的异常检测模型对新数据进行异常检测。本发明实现了训练云函数与推理云函数的自动生成与部署,本发明实现的异常检测系统中支持的所有算法都包含训练云函数模板与推理云函数模板两份模板,在部署算法时会将相应的算法参数填充到模板中生成对应的训练云函数代码和推理云函数代码,再通过阿里云函数计算平台提供的SDK进行自动部署。

[0050] 实施例

[0051] 本发明提出了一种基于无服务器架构的物联网数据异常检测系统实现方法,以下是其代码实现部分(截取重要):

[0052] 如图4所示,用户可以为接入系统中的任一物联网设备选择合适的异常检测算法并配置相应的算法参数,之后系统能自动根据算法和参数为其生成训练云函数和推理云函数的代码并自动部署到阿里云函数计算平台上。

[0053] 如下述代码1所示,这部分包括了自动按算法模板生成训练云函数和推理云函数的代码的逻辑:

[0054] 代码1

```
def generate_event_train_py(clf_name, device_id, params): # 生成训练云函数代码的函数
    model_template_path = config.get('code', 'model_template_path') + "train_event/" #模板路径
    output_path = config.get('code', 'output_path') + str(device_id) + "/" # 输出路径
    template_filename = "train_event_" + clf_name + "_template" + ".txt" # 模板名
    output_dirname = "train_event_" + clf_name + "_" + str(device_id) # 输出目录
    params_str = str(params)
    if not os.path.exists(model_template_path + template_filename): # 检查模板是否存在
        logger.warning("模板文件%s 不存在！" % (model_template_path + template_filename))
        return False
    # 获取模型的实例化语句（根据 params 生成）
    try:
```

[0055]

```
        clf_init_sentence = get_clf_init_sentence(clf_name, params)
    except:
        traceback.print_exc()
        logger.warning("生成算法%s 的模型初始化语句失败" % clf_name)
        return False
    print("clf_init_sentence:", clf_init_sentence)
    # 读取模板文件
    with codecs.open(model_template_path + template_filename, "rb", "UTF-8") as f:
        template_str = f.read()
    if not template_str:
        return False
    # 替换模板中的元素
    try:
```

```
            tmp = Template(template_str)
            train_py_str = tmp.substitute(db_host=MySQL_CONFIG["host"],
        db_port=MySQL_CONFIG["port"], db_user=MySQL_CONFIG["user"],
        db_pwd=MySQL_CONFIG["pwd"], db_area=MySQL_CONFIG["db"],
        db_charset=MySQL_CONFIG["charset"], clf_init_sentence=clf_init_sentence,
        device_id=device_id,    fc_endpoint=FC_CONFIG['endpoint'],
        fc_accessKeyID=FC_CONFIG['accessKeyID'],
        fc_accessKeySecret=FC_CONFIG['accessKeySecret'],
        csv_path=CSV_PATH, model_path=MODEL_PATH, area=AREA,
        params_str=params_str,)
        except:
            traceback.print_exc()
            logger.warning("替换模版失败！ ")
            return False
        # 在本地创建出目录并写入目标文件
[0056]  try:
            directory = os.path.exists(output_path + output_dirname)
            if not directory:
                os.makedirs(output_path + output_dirname)
            # 写入目标文件(要使用 fc-sdk 上传部署的函数源码必须命名为 index.py)
            with codecs.open(output_path + output_dirname + '/index.py', "wb", "UTF-8") as f:
                f.write(train_py_str)
                f.flush()
        except:
            traceback.print_exc()
            logger.warning("创建云函数代码文件失败！ ")
            return False
        return output_path + output_dirname + '/index.py'


    def generate_event_predict_py(clf_name, device_id, params): # 生成推理云函数代码的函数
        model_template_path = config.get('code', 'model_template_path') + "predict_event/" #模板
```

路径

output_path = config.get('code', 'output_path') + str(device_id) + "/" #输出路径

template_filename = "predict_event_" + clf_name + "_template" + ".txt" # 模板名

output_dirname = "predict_event_" + clf_name + "_" + str(device_id)# 输出目录

clf_init_sentence = get_clf_init_sentence(clf_name, params) # 实例化检测器

if not os.path.exists(model_template_path + template_filename): #检查模板是否存在

logger.warning("模板文件%s 不存在！" % (model_template_path + template_filename))

return False

read template file

with codecs.open(model_template_path + template_filename, "rb", "UTF-8") as f:

template_str = f.read()

if not template_str:

return False

try:

[0057] # 替换模板中的元素

tmp = Template(template_str)

if clf_name in [DL_ALGO_LIST]:

predict_py_str = tmp.substitute(db_host=MySQL_CONFIG["host"], db_port=MySQL_CONFIG["port"], db_user=MySQL_CONFIG["user"], db_pwd=MySQL_CONFIG["pwd"], db_area=MySQL_CONFIG["db"], db_charset=MySQL_CONFIG["charset"], device_id=device_id, model_path=MODEL_PATH, clf_init_sentence=clf_init_sentence)

else:

predict_py_str = tmp.substitute(db_host=MySQL_CONFIG["host"], db_port=MySQL_CONFIG["port"], db_user=MySQL_CONFIG["user"], db_pwd=MySQL_CONFIG["pwd"], db_area=MySQL_CONFIG["db"], db_charset=MySQL_CONFIG["charset"], device_id=device_id, model_path=MODEL_PATH)

创建目标目录

directory = os.path.exists(output_path + output_dirname)

if not directory:

```
                os.makedirs(output_path + output_dirname)
            # 写入目标文件
            with codecs.open(output_path + output_dirname + '/index.py', "wb", "UTF-8") as f:
                f.write(predict_py_str)
                f.flush()
```

[0058]

```
        except:
            traceback.print_exc()
            logger.warning("创建云函数代码文件失败！ ")
            return False
        return output_path + output_dirname + '/index.py'
```

[0059] 其中generate_http_train_py函数用于生成事件触发的训练云函数代码，generate_http_predict_py函数用于生成事件触发的推理云函数代码。其中函数参数clf_name,device_id,params分别表示算法名称、物联网设备在OneNET云平台中的设备id和算法参数。

[0060] 如下述代码2所示,这部分包含了云函数自动部署的实现逻辑：

[0061] 代码2

```
    def deploy_ fc (clf_name, source_file_path, service_name, cron_exp): #自动部署云函数用的
    函数
        client = get_fc_client() # 与阿里云函数计算平台建立连接
        source_code_dir = config.get('code', 'output_path') # 云函数代码路径
        device_id_str = source_file_path[source_file_path.find(source_code_dir) +
    len(source_code_dir):source_file_path.find('/',source_file_path.find(source_code_dir) +
    len(source_code_dir))] # 提取设备id
```

[0062]

```
        function_name = source_file_path[source_file_path.find(device_id_str) +
    len(device_id_str) + 1:source_file_path.rfind('/')] # 云函数名称
        function_dir = source_code_dir + device_id_str + '/' + function_name # 云函数路径
        if not os.path.exists(source_file_path): # 检查云函数代码文件是否存在
            logger.warning("函数代码文件%s不存在！ " % source_file_path)
            return False
        try:
            # 创建云函数
```

```
            if clf_name in [DL_ALGO_LIST]:
                client.create_function(
                    service_name, function_name, 'python3', 'index.handler',
                    codeDir=function_dir, memorySize=3072, timeout=600,
                    initializationTimeout=300,
                    environmentVariables={
                        'PYTHONPATH': '/mnt/tf/python/lib/python2.7/site-packages:'
                                      '/mnt/tf/python/lib/python3.6/site-packages'},
                )
            else:
                client.create_function(
                    service_name, function_name, 'python3', 'index.handler',
                    codeDir=function_dir, memorySize=1024, timeout=600,
                    initializationTimeout=300,
                    environmentVariables={
```

[0063]

```
                        'PYTHONPATH': '/mnt/auto/python/lib/python2.7/site-packages:'
                                      '/mnt/auto/python/lib/python3.6/site-packages'},
                )
        except:
            traceback.print_exc()
            logger.warning("在服务%s中创建函数%s失败" % (service_name, function_name))
            return False
        try:
            # 创建并绑定Timer Trigger
            create_timer_trigger(service_name, function_name, cron_exp)
        except:
            traceback.print_exc()
            logger.warning("为创建函数%s创建定时触发器失败" % function_name)
            return False
        # 返回创建好的函数名称
        return function_name
```

[0064] 在代码2中,描述了本发明实现的系统是如何将生成的训练云函数和预测云函数部署到阿里云函数计算平台上的。

[0065] 如下述代码3所示,给出了训练云函数代码的模板,这部分包含了训练云函数的执行逻辑。训练云函数根据设备的历史数据训练和传入的算法参数训练异常检测模型,训练好的异常检测模型保存到NAS文件存储服务中供推理云函数调用。

[0066] 代码3

```
def train_model(device_id, csv_dir_path, output_dir_path): # 异常检测模型训练函数
    start = time.time()
    csv_file_path = csv_dir_path + str(device_id) + ".csv"  # 训练数据的csv文件位置
    if os.path.exists(csv_file_path) == False: # 检查csv文件是否存在
        print("%s文件不存在，无法进行训练！" % csv_file_path)
        res_code = 555
        res_info = "Training file %s does not exist." % csv_file_path
        return [res_code, res_info]
    df = pd.read_csv(csv_file_path)
    X_train = df.iloc[:, 4:].values  # csv文件的前四列是index、timestamp、device_id、label，
不作为训练特征
```

[0067]

```
    training_count = len(X_train)
    feature_count = df.shape[1] - 4
    clf_name = '${algo_name}' # 异常检测算法名字
    params_str = "${params_str}"
    try:
        clf = ${clf_init_sentence} # 实例化异常检测器
        clf.fit(X_train)
        unix_timestamp = int(time.time())
        # export model
        model_name = '{}_{}_{}.pkl'.format(str(device_id), clf_name, unix_timestamp)
        with open(os.path.join(output_dir_path, model_name), 'wb') as out:
            pickle.dump(clf, out, protocol=pickle.HIGHEST_PROTOCOL)
    except Exception as e:
```

```
                traceback.print_exc() # 打印异常信息
                # 训练失败
                fc_timestamp = int(round(time.time() * 1000))
                finish_time = datetime.datetime.fromtimestamp((fc_timestamp + 28800000) /
1000).strftime("%Y-%m-%d %H:%M:%S")
                end = time.time()
                insert_training_log(device_id, 0, finish_time, '', 0, end-start)  # 写入失败记录
                print("training for device=%d failed!" % (device_id))
                res_code = 555
                res_info = "Failed to train model"
                return [res_code, res_info]
            else:
                # 训练成功
                fc_timestamp = int(round(time.time() * 1000))
                finish_time = datetime.datetime.fromtimestamp((fc_timestamp + 28800000) /
1000).strftime("%Y-%m-%d %H:%M:%S")
```

[0068]

```
                update_devices_table(device_id, finish_time ,training_count)  # 若训练成功则修
改devices表的last_training和last_train_index字段
                update_settings_table(device_id, model_name)  # 若训练成功则修改settings表的
latest_model_name字段
                inset_models_table(unix_timestamp, device_id, model_name, finish_time,
training_count, feature_count, clf_name, params_str)  # 若训练成功则将模型记录插入models
表中
                end = time.time()
                insert_training_log(device_id, training_count, finish_time, model_name, 1, end-start)
# 写入成功记录
                print("training for device=%d success, time=%fs, finish_time=%s,
model_name=%s" % (device_id, end-start, finish_time, model_name))
                return [200, 'ok']

    async def request_archive2csv_async(area, device_id): # 使用aiohtt库，向训练云函数发出异
```

步http请求

```
payload = {'device': device_id}
url = ' ${server_url}' + area + '/'
async with aiohttp.ClientSession() as session:
    async with session.post(url, params=payload) as res:
        print(res.status)
        print(await res.text())


def handler(event, context): # 云函数入口函数
    # 获取基本信息
    area = "${area}"
    device_id = ${device_id}
    csv_dir_path = "${csv_path}"
    output_dir_path = "${model_path}"
```

[0069]

```
    # 1、请求归档云函数对该设备的数据点归档（aiohttp异步请求）
    start_archive = time.time()
    loop = asyncio.get_event_loop()
    task = loop.create_task(request_archive2csv_async(area, device_id))
    loop.run_until_complete(task)
    end_archive = time.time()
    print("archive time:", end_archive-start_archive)

    # 2、调用训练函数对该设备归档好的csv文件进行训练
    start_train = time.time()
    res_code, res_info = train_model(device_id, csv_dir_path, output_dir_path)
    end_train = time.time()
    print("train time:", end_train-start_train)
    return json.dumps({'info': res_info, 'code': res_code})
```

[0070] 如下述代码4所示,给出了推理云函数代码的模板,这部分包含了推理云函数的执行逻辑。推理云函数从NAS文件存储中加载训练好的异常检测模型,并对新报送的数据进行

异常检测。

[0071] 代码4

```
def predict_with_model(model_path, x_test, device_id, model_name): # 调用模型进行异常检测的函数
    # 加载模型：
    try:
        with open(model_path, 'rb') as inp:
            clf = pickle.load(inp)
    except:
        res_code = 555
        res_info = "Failed to load model."
        return [res_code, res_info, 0]

    # 使用模型检测
    try:
        y_pred =  clf.predict(x_test)
        y_scores = clf.decision_function(x_test)  # outlier scores
    except:
        traceback.print_exc()
        res_code = 555
        print("Failed to call detection function.")
        res_info = 'Failed to call detection function.'
        return [res_code, res_info, 0]
    else:
        print("x_test:", x_test, "\t y_pred:", y_pred, "\t y_scores:", y_scores)
        if y_pred == [1]:  # 若是异常
            res_result = 1
            # 获取时间戳
            fc_timestamp = int(round(time.time() * 1000))
            predict_time = datetime.datetime.fromtimestamp((fc_timestamp + 28800000) /
```

[0072]

```
1000).strftime("%Y-%m-%d %H:%M:%S")
                    x_anomaly = x_test[0].tolist()
                    insert_anomaly_log(device_id, str(x_anomaly), 1, y_scores[0], model_name,
predict_time)
                    insert_view_bulletin(device_id, str(x_anomaly), predict_time)
                    msg = "设备%d的新数据点[%s]疑似异常，请及时处理" % (device_id,
x_anomaly)
                    send_msg_to_ftqq(msg)
                else:
                    res_result = 0
                # 将异常得分y_scores保留2位小数作为info返回，将异常检测结果y_pred作为
result返回
                return [200, str(format(y_scores[0], '.2f')), res_result]


    def handler(event, context): # 云函数入口函数
[0073]          device_id = ${device_id}
                # 查表获得特征数目
                feature_count = query_feature_count(device_id)
                # 从event中获取输入的参数，并按字母顺序重新调整参数顺序，保证顺序与csv中一
致
                evt_dict = json.loads(event)
                sorted_key_list = sorted(evt_dict)
                sorted_param_dict = { key: evt_dict[key] for key in sorted_key_list}
                # 检查输入的特征值是否都是数字
                try:
                    sorted_param_dict = { k:float(v) for (k,v) in sorted_param_dict.items()}
                except:
                    print("Your input is not a number")
                    return json.dumps({'info': "Your input is not a number.", 'code': 555, 'result': 0})
                # 检查维度是否匹配
                if len(sorted_param_dict) != feature_count:
```

[0074]

```
                print("The input data does not match the feature dimension of the model.")
                return json.dumps({'info': "The input data does not match the feature dimension of the model.", 'code': 555, 'result': 0})
            # 转为numpy array
            x_test = [sorted_param_dict[key] for key in sorted_key_list]
            x_test = np.array([x_test])
            model_name = query_model_name(device_id) # 获得要使用的模型名称
            if model_name == False or model_name == None:
                print("Failed to get model name.")
                return json.dumps({'info': "Failed to get model name.", 'code': 555, 'result': 0})
            model_path = os.path.join("${model_path}", model_name)
            # 调用预测函数进行异常检测
            res_code, res_info, res_result = predict_with_model(model_path, x_test, device_id, model_name)
            return json.dumps({'model_name': model_name, 'info': res_info, 'code': res_code, 'result': res_result})
```

[0075] 本发明结合传统的物联网云平台和新兴的无服务器架构,通过采用传统物联网云平台作为设备接入端降低了设备接入与管理的成本,通过无服务器架构构建异常检测端大幅降低了系统的部署和运行成本。无服务器架构中的云函数按需执行,不像传统云服务器一样有“闲时计费”问题,非常适合物联网这样请求呈现低频、周期性特点的场景。且无服务器架构在并发性能、响应速度上相比传统云计算模式更有优势,能更快地检测出物联网设备上报的异常数据。在现有的技术中,尚没有通过引入无服务器架构来降低物联网数据异常检测系统的成本、提高系统检测速度的。

[0076] 本发明的保护内容不局限于以上实施例。在不背离本发明构思的精神和范围下,本领域技术人员能够想到的变化和优点都被包括在本发明中,并且以所附的权利要求书为保护范围。

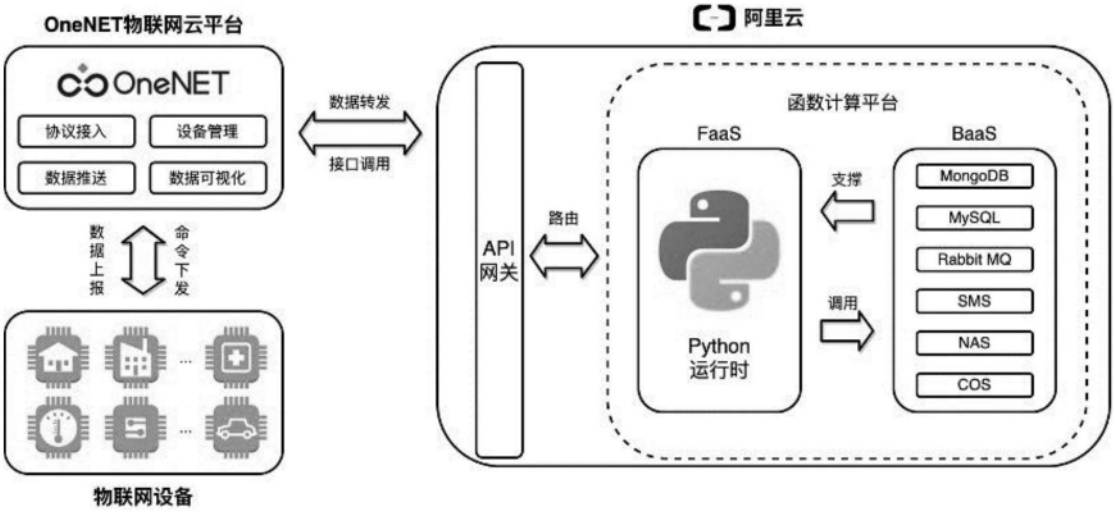

OneNET物联网云平台
OneNET
协议接入
设备管理
数据推送
数据可视化
数据转发
接口调用
数据上报
命令下发
物联网设备
阿里云
API网关
路由
函数计算平台
FaaS
Python运行时
支撑
调用
BaaS
MongoDB
MySQL
Rabbit MQ
SMS
NAS
COS

图1

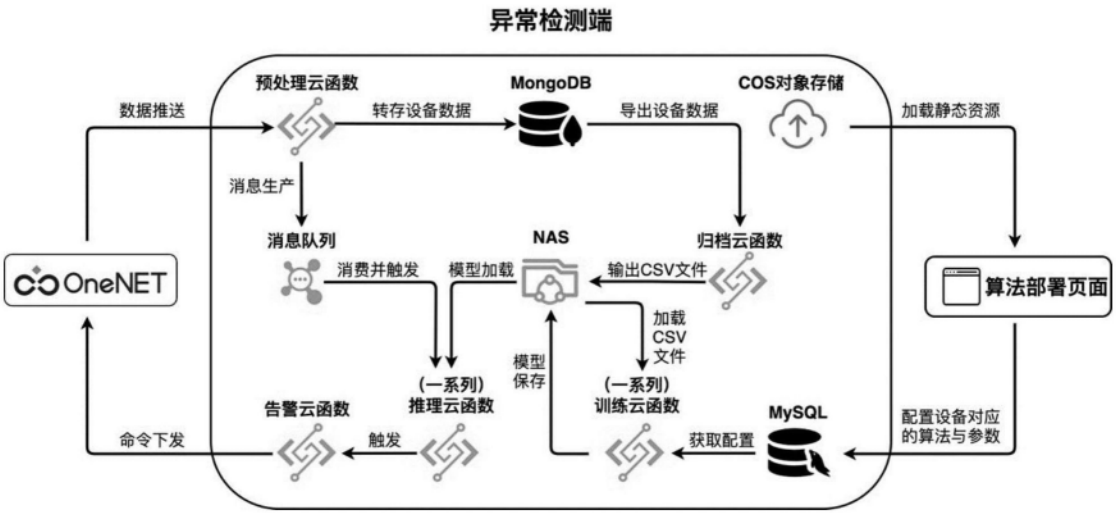

异常检测端
预处理云函数
MongoDB
COS对象存储
数据推送
转存设备数据
导出设备数据
加载静态资源
消息生产
消息队列
NAS
归档云函数
OneNET
消费并触发
模型加载
输出CSV文件
算法部署页面
加载CSV文件
模型保存
(一系列)推理云函数
(一系列)训练云函数
告警云函数
MySQL
命令下发
触发
获取配置
配置设备对应的算法与参数

图2

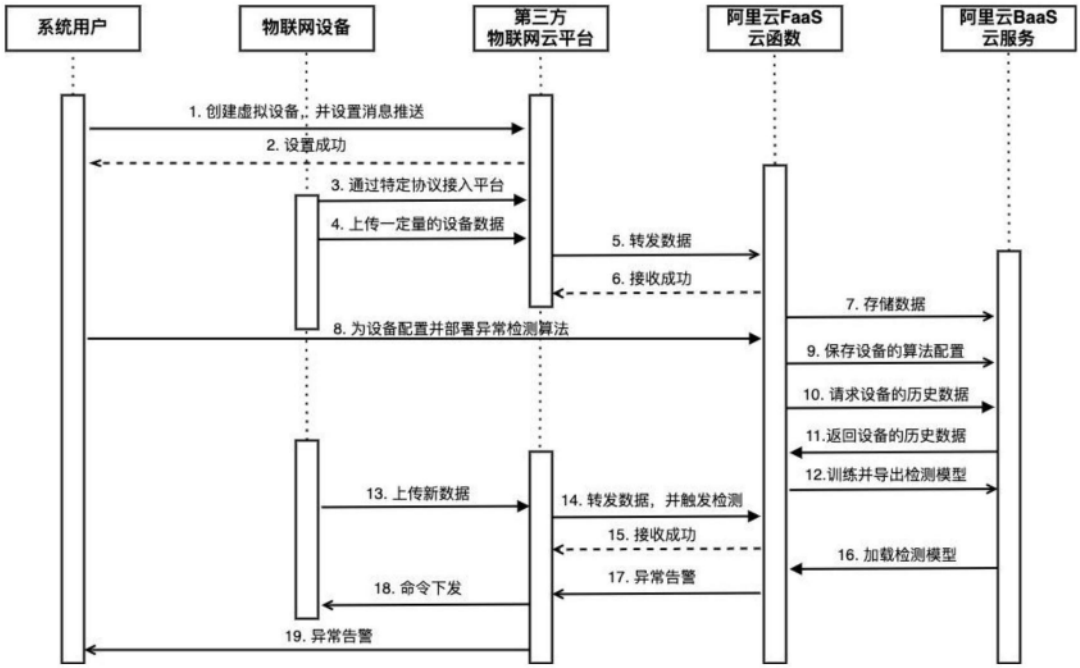

系统用户
物联网设备
第三方
物联网云平台
阿里云FaaS
云函数
阿里云BaaS
云服务
1. 创建虚拟设备，并设置消息推送
2. 设置成功
3. 通过特定协议接入平台
4. 上传一定量的设备数据
5. 转发数据
6. 接收成功
7. 存储数据
8. 为设备配置并部署异常检测算法
9. 保存设备的算法配置
10. 请求设备的历史数据
11.返回设备的历史数据
12.训练并导出检测模型
13. 上传新数据
14. 转发数据，并触发检测
15. 接收成功
16. 加载检测模型
17. 异常告警
18. 命令下发
19. 异常告警

图3


设备名称
pima
设备id
666269874
协议类型
HTTP
创建时间
2021-01-05 14:34:00
设备在OneNET云平台上的创建时间
数据点数目
768
一般地，已有的数据点越多模型越准确
特征维度
8
算法选择
DEC
请选择合适的异常检测算法
算法描述
DEC(Deep Embedded Clustering, 深度嵌入式聚类)算法是一个重要的深度聚类算法，该算法主要包括两个部分，一是对自动编码器进行参数初始化，二是用KL散度迭代优化聚类。本系统基于DEC算法构建了一个基于聚类的异常检测模型。
算法参数
污染率
0.10
污染率是对设备异常数据比例的一个估计值
autoencoder_layers
autoencoder_layers: 自编码器的层次, 请输入各层的维度, 中间用","隔开 , 例如"9, 6, 2". 请注意，首层维度必须与数据特征维度一致, 否则会训练失败.
n_clusters
8
The number of clusters to form as well as the number of centroids to generate.
batch_size
64
batch_size: 一次训练所选取的样本数.
更新模式
周期更新
请选择云端检测模型自动更新的模式
更新配置
时间间隔
请选择使用哪一种定时规则
时间间隔
60
min
只能输入正整数，以分钟作为单位，最小1分钟
部署
取消

图4

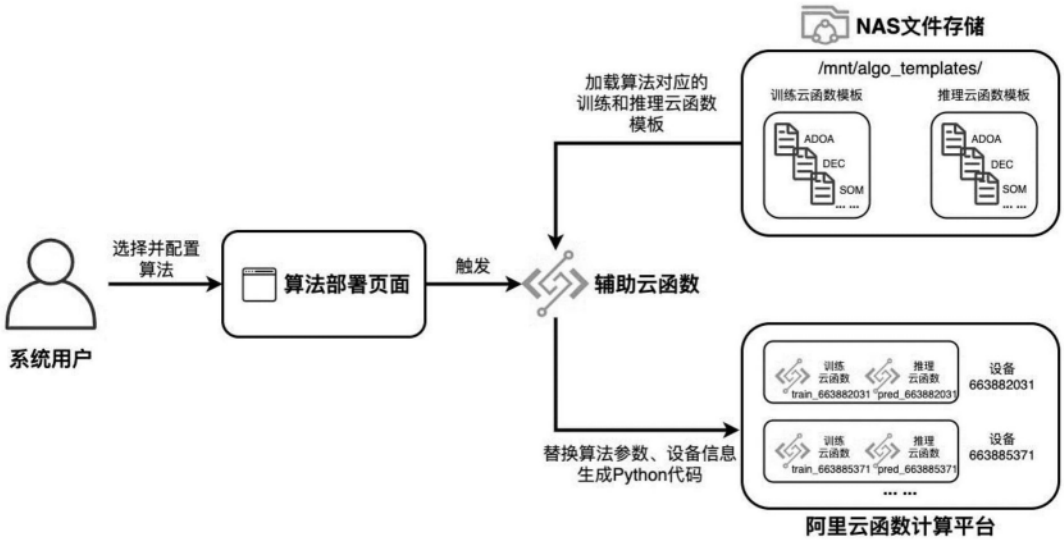

NAS文件存储
/mnt/algo_templates/
训练云函数模板
推理云函数模板
ADOA
DEC
SOM
加载算法对应的
训练和推理云函数
模板
选择并配置
算法
系统用户
算法部署页面
触发
辅助云函数
训练
云函数
推理
云函数
train_663882031
pred_663882031
设备
663882031
train_663885371
pred_663885371
设备
663885371
替换算法参数、设备信息
生成Python代码
阿里云函数计算平台

图5