交互感知应用微云服务框架①

赵伟峰[1,3], 杨秋松[1,2], 李 梅[1,3], 张鸿骏[1,3]

[1](中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院软件研究所 计算机科学国家重点实验室, 北京 100190)
[3](中国科学院大学, 北京 100190)

摘 要: 当前, 开发使用微云服务的交互感知应用时存在开发困难以及重复开发的问题. 为解决这些问题, 设计并实现了为交互感知应用提供微云服务的框架 Bolt. Bolt 服务框架以 ZeroMQ 消息处理队列为基础, 分为三部分: Bolt Client Lib 库、Bolt Service Lib 库以及 Bolt Broker 代理, 构成“Client-Broker-Service”的三级架构. Bolt Client Lib 和 Bolt Service Lib 分别为移动端应用和微云端处理引擎分别提供 API, 使应用与处理引擎的开发得以分离, 降低了开发难度并减少重复开发的情况. 实验结果表明, 与 Gabriel 等应用相比, Bolt 服务框架使用的通信端口降低到 2 个, 同时, 将系统开销降低到 1ms 左右的时间.

关键词: 可穿戴设备; 交互感知应用; 云计算; 微云; 服务框架

Service Framework for Interactive Perception Applications on Cloudlets

ZHAO Wei-Feng[1,3], YANG Qiu-Song[1,2], LI Mei[1,3], ZHANG Hong-Jun[1,3]

[1](Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[3](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Recently, developing interactive perception applications using cloudlet’s service always traps in hardship or duplication. To resolve this problem, Bolt service framework is designed and realized for creating interactive perception applications on cloudlet. Bolt is based on ZeroMQ, which is a kind of message queue, and could be divided into three parts: Bolt Client Lib, Bolt Service Lib and Bolt Broker. These make up a “Client-Broker-Service” architecture. Bolt Client Lib and Bolt Service Lib can respectively provide API for clients on mobile device side and cognitive engines on cloudlet side, reducing the difficulty and redundant works dues to the separate developing. The experiments show that Bolt service framework has only a 1ms system overhead and using just 2 socket ports meanwhile compared to applications such as Gabreil.

Key words: wearable device; intaeractive perception application; cloud computing; cloudlet;service framework

当下, 可穿戴设备的增多导致交互感知应用开始涌现. BaiduEye[1]展示了利用百度开发的头戴式设备为使用者提供视野内物体相关信息的使用场景. 交互感知应用[2]通过处理摄像头、话筒、传感器采集的图像、视频、音频和各种传感数据, 为使用者提供物体识别、人脸识别、手势识别等服务. 例如, 物体识别可以在商场内为购物者提供商品的产地、材质、其它用户评论等相关信息; 人脸识别可以为使用者快速找到人群中的目标; 手势识别使用户有了新的控制设备和发出指令的手段.

由于交互感知应用采集和处理的是图像、视频、音频等多媒体信息, 需要的计算能力和存储能力都相对较高. 例如, 当前语音识别、图像识别等模仿人脑的研究领域使用的深度学习技术, 甚至需要多个 GPU 进行并行计算[3], 对大量的数据进行训练. 受限于体积小、重量轻、外观优雅的设计要求, 可穿戴设备暂时无法满足这些需求.

将计算任务迁移至云, 利用云的海量计算和存储

① 基金项目:国家自然科学基金重大项目(91318301);中国科学院战略性科技先导专项(XDA06010600)
收稿时间:2016-01-28;收到修改稿时间:2016-03-03 [doi:10.15888/j.cnki.csa.005368]

资源, 是解决可穿戴设备计算和存储资源匮乏的一种有效方法. 然而, 通过广域网连接到云, 存在延迟、抖动、堵塞和失败的问题, 并且短期内互联网的基础不太可能发生大的变革, 这些问题需要很长一段时间才能得到解决[4], 使用云服务无法满足交互感知应用的低响应时间的要求. 因此, CMU 的 Mahadev Satyanarayanan 等人在论文[4]中提出了微云(Cloudlets)的概念, 并在 Elijah[5,6]中得到了实现. 微云邻近可穿戴设备, 可以有效的降低网络延时. 其融合了移动计算和云计算的特点, 代表了“移动设备-微云-云”三级架构的中间层. 微云的目标是利用逻辑上邻近移动设备的小型云计算系统, 为移动设备提供类似云服务的功能. 逻辑邻近是指移动端到微云之间的网络连接只经过一次或几次路由, 可以提供低延时和高带宽的网络连接[6]. Elijah 着重于微云平台的基础设施技术研究, 对现有的 OpenStack 云计算系统进行了扩展以适用于微云环境中, 提出了虚拟机动态合成等方法. GigaSight[7]、QuiltView[8]、Gabriel[9,10]项目就是基于 Elijah 开发的使用 Google Glass 头戴式设备的交互感知应用.

目前, 开发使用微云服务的交互感知应用时, 存在以下两个问题:

① 使用传统套接字进行开发, 而传统套接字的开发难度较大, 且存在端口资源浪费的情况.

② 应用各自使用了单独的通信协议、引擎封装方法, 无法复用.

例如, 在 Gabriel 中, 需要为每一个传感器使用一个通信端口, 另外还有控制、结果转发等需要的通信端口, 使用了多达 6 个 TCP 端口, 造成了端口资源的浪费; Gabriel 使用的处理引擎封装方法只能用于其项目中, 而不适用于其它设备和应用.

针对现有使用微云服务的交互感知应用开发时存在的问题, 本文在当前研究的基础上, 设计和实现了一种交互感知应用的微云服务框架 Bolt. Bolt 内部使用了 ZeroMQ 消息处理队列进行套接字通信, 提高了网络性能, 对外提供提供透明的传输通道, 降低网络开发的难度; 提供 Bolt Client Lib 和 Bolt Service Lib 作为移动端应用和微云端处理引擎的开发 API, 封装了开发过程中的重复工作, 提高各组件的复用性.

1 系统架构

Bolt 服务框架的示意图如图 1 所示.

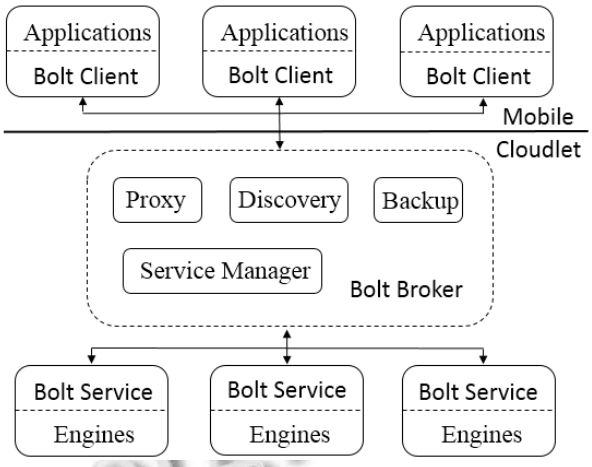

图 1 Bolt 服务框架示意图

Bolt 服务框架分为三部分:

① Bolt Client Lib: 为移动端的交互感知应用提供 API, 方便交互感知应用的开发.

② Bolt Service Lib: 为人脸识别、增强现实等微云上的各种处理引擎提供 API, 简化处理引擎的封装方法.

③ Bolt Broker: 介于两者之间, 充当两者的代理, 形成“Client-Broker-Service”的三级架构.

Bolt 服务框架可以接入多个处理引擎, 并以服务的方式提供给多个移动设备上的多个交互感知应用同时使用.

1.1 Bolt Client Lib

可以使用由 Bolt Client Lib 提供的 API 开发交互感知应用的移动客户端, 开发出的客户端称为 Bolt Client. 与网络协议栈类似, Bolt 服务框架提供透明的数据传输通道, 客户端的开发者不需要了解 Bolt 的内部工作机制, 可以认为和微云上的处理引擎进行直接交互. Bolt Client 与 Bolt Service 交互的模型如图 2 所示.

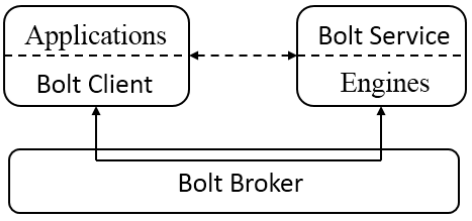

图 2 Bolt Client 和 Bolt Service 直接交互模型

Bolt Client Lib 使用 ZeroMQ 作为消息传输层, 由于 ZeroMQ 可以方便的移植到各种平台, 并提供了各

种开发语言使用的绑定库, 因此, Bolt Client Lib 可以方便的在各种移动平台上实现.

为了充分使用移动端的摄像头、传感器等硬件资源, 可以进一步将这些设备的相关代码封装进 Bolt Client Lib 中并对外提供使用接口. 这样可以进一步降低交互感知应用的开发难度, 提高开发效率并增加硬件资源的利用率.

1.2 Bolt Service Lib

Bolt 服务框架中开发物体识别、增强现实等处理引擎时使用 Bolt Service Lib 库. 处理引擎可以在微云中注册为一个服务, 称为 Bolt Service, 供之后连接到微云的交互感知应用客户端使用. 和移动端类似, 处理引擎的开发者也不需要了解 Bolt 服务框架的内部工作机制.

使用 Bolt Service Lib 开发的处理引擎运行在 Elijah 系统提供的基础设施上. 这些基础设施包括虚拟机及存储、网络、Docker 容器等. 使用虚拟机或者容器技术, 可以忽略处理引擎的操作系统相关性或者开发语言相关性, 提供更一般的支持[10].

使用 Bolt Service Lib 的 API 创建服务, 可以标准化微云上的处理引擎. 这些标准化的处理引擎可以存储在远端的云服务中, 当交互感知应用请求的服务不在微云中时, 可以从远端云服务器中下载并启动服务; 另一方面, 标准化保证从云端下载的处理引擎可以正确接入 Bolt 服务框架而不造成错误. 利用虚拟机动态合成技术, 可以大幅降低处理引擎从云服务器下载至微云的时间[5].

1.3 Bolt Broker

Bolt Broker 的作用主要在于汇总所有已连接的服务, 并对外提供统一的服务接口. Bolt Broker 需要包含以下功能:

① 服务管理　主要包括服务的注册、检测服务是否可用、提供服务的查询等.

② 代理发现　对收到的代理搜索广播消息给出应答, 交互感知应用在收到应答消息后, 才可以得到 Broker 的地址, 进行下一步的连接.

③ 请求调度　为众多交互感知应用的请求选择其目的处理引擎. 调度可以选用不同的策略, 如 LRU、Round-Robin 等.

另外, 采用 Bolt Broker 作为框架的中间层, 容易引起系统的单点故障. 解决方法一是采用备份服务的方式; 另一种可以采用无中间代理的分布式架构设计方法. 对于微云系统来说, 第一种方法更为简单有效, 但会造成系统资源一定程度上的浪费.

2 原型系统的实现

以下从消息通信、Bolt Client Lib 的实现、Bolt Broker 的实现、Bolt Service Lib 的实现、OpenFace 人脸识别引擎、故障转移等方面介绍实现 Bolt 服务框架的细节.

2.1 消息通信

Bolt 框架内包含了众多的组件, 而 Broker 代理起到了连接移动端交互感知应用和微云端处理引擎的作用, 因此, 需要首先确定 Broker 的网络位置. 在移动设备通过无线连接的方式接入到网络后, 启动一个交互感知应用会首先通过 UDP 广播的方式向其所在的局域网内询问 Bolt Broker 的网络地址, 如果该局域网内存在一个可用的 Broker, 或者局域网内存在 Broker 的代理, 则会收到包含 Broker 网络地址的回复消息, 之后就可以连接到 Broker 进行消息通信. 需要说明的是, 此连接过程是由 Bolt Client Lib 执行的, 交互感知应用的开发者不需要了解此过程细节. 处理引擎连接 Broker 的过程与此相同.

Bolt 使用 ZeroMQ 作为整个框架内消息通信的传输层. 相比于使用 Socket 套接字进行网络通信应用的开发, ZeroMQ 屏蔽了 Socket 套接字编程时连接的建立、维护、协议选择、错误处理等细节, 使网络编程更加简单且性能更高[12]. 相比于 Gabriel 中使用的多达 6 个 TCP 通信端口, 使用 ZeroMQ 作为消息通信的手段, 只需要 1 个 TCP 端口即可. 这个端口由 Bolt Broker 绑定并等待 Bolt Client Lib 与 Bolt Service Lib 进行连接. 由此, Bolt 服务框架只需要使用 2 个 socket 端口.

2.2 Bolt Client Lib 的实现

在原型系统中, Bolt Client Lib 使用 Java 语言以 BoltClient 类实现. BoltClient 的成员函数包括创建一个客户端、发送请求命令函数 send、接收到响应数据时的回调函数 on_message、查询可用的服务等. 在 Android 移动端以 ZeroMQ 的 Java 版本 JeroMQ 为基础实现并编译出 Bolt Client Lib 的 Java 包. 在创建其它交互感知应用的 Android 端时, 可以将此 Java 包导入到工程中, 选择从 BoltClient 类派生, 使用提供的 API 完

成应用的开发.

对于 Bolt Client Lib 的其它实现版本, 都应该至少包含以下基本 API 函数: 连接到 Broker、发送数据到 Broker、从 Broker 收到消息时的回调函数等.

2.3 Bolt Broker 的实现

当交互感知应用的处理引擎连接到 Broker 代理时, 首先会发送一条服务注册消息. 服务注册消息中包含此处理引擎提供的服务名称以及对该服务的相关描述. Broker 内部维护一个可用服务表 service_hashtable, 在收到此消息后, 将该服务添加到该表中, 如图 3 所示.

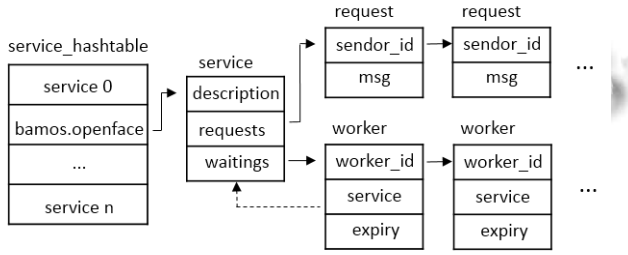

图 3 Broker 内部服务管理示意图

service_hashtable 以哈希表的方式实现, 表的每一项中, 关键字 key 为注册服务的名称, 表项的值是一个 Service 类型的值, 包括 3 项, 分别为对该服务的描述 description、该服务的请求列表 requests、和该服务可用的处理引擎 waitings. Bolt Broker 将请求分发给处理引擎使用的是 Round-Robin 轮询调度策略. 移动端的交互感知应用每发送一个请求, 都会被 ZeroMQ 打包为一个 request 请求, 并添加到 requests 请求列表的末尾; 如果 watings 中包含可用的处理引擎, 则会将 requests 请求列表的队首的请求发送给 waitings 可用列表的队首对应的处理引擎进行处理, 并从 waitings 可用列表中删除, 待任务完成后, 重新添加到 waitings 可用列表的队尾. 在 Broker 的代码中, 负责请求分发策略的是为 dispatch 函数, 如果需要实现其他的请求调度策略, 则需要修改此函数的代码. dispatch 函数核心代码如下:

```
while service.waitings and service.requests:
    msg = service.requests.pop(0)
    worker_id = service.waitings.pop(0)
    self.waitings.remove(worker_id)
      self.send_to_worker(worker_id,  W_REQUEST, msg)
```

当客户端请求的服务不在 service_hashtable 中时, 会检查远端云服务器上此服务是否可用, 如果可用, 则下载该服务的虚拟机或者 docker 容器, 并在微云上启动. 启动后, 该服务被添加进 service_hashtable 中, 使得服务可用. 如果远端云服务器上不存在所请求的服务, 则为客户端返回一个错误.

2.4 Bolt Service Lib

在 Bolt 的原型系统中, Bolt Service Lib 使用 Python 语言以 BoltService 类实现. 在开发处理引擎时, 可以继承和派生 BoltService 类, 在消息处理的部分添加处理引擎对于客户端请求的处理代码. 和 BoltService 类同样功能的代码也可以使用其它语言在其它操作系统上实现, 实现时需要包含以下 API 函数: 连接到 Broker、注册服务、发送消息到 Broker、从 Broker 收到消息时的回调函数、心跳检测函数等.

在创建处理引擎实例时, BoltService 类的初始化函数首先通过 UDP 广播的方式在局域网内寻找 Bolt Broker 或其代理并连接, 连接成功后注册一个服务. 注册服务需要提供服务的名称以及服务的描述信息. 注册成功后, 处理引擎进入一个循环函数 serve_forever. serve_forever 主要负责查看是否接收到消息, 如果有, 则交由 msg_handler 进行处理; 如果接收失败达到一定次数, 则会重新连接. 在没有处理引擎的请求时, Bolt Broker 和处理引擎需要互相定时发送心跳信息, 确保对对方的感知.

在 msg_handler 中, 如果解析到接收到的消息是客户端的请求命令, 则会调用 request_handler 进行处理, 开发处理引擎时, 主要修改的代码就在 request_handler 函数中. request_handler 函数有一个参数 request, 并返回一个值 reply. request 的内容为移动端的请求内容; 返回值 reply 被 msg_handler 发回给交互感知应用客户端.

2.5 OpenFace 人脸识别引擎

为了测试 Bolt 的有效性, 在原型系统中实现了人脸识别处理引擎 OpenFace[11]. 人脸识别是交互感知应用中最常见的任务之一, 且目前技术发展较为成熟. 使用 Python 代码派生了 BoltService 类, 集成了 OpenFace 的相关使用接口, 将 OpenFace 注册为 bamos.openface 服务供交互感知应用客户端使用.

OpenFace 人脸识别引擎基于深度神经网络技术,

是 FaceNet[13]的 Python 和 Torch 实现版本，并且共享给开源社区使用. OpenFace 使用了 CASIA-WebFace 和 FaceScrub 作为数据集，在 Tesla k40 GPU 上训练出 nn4、nn4.small 的模型提供下载使用. OpenFace 提供的 docker 容器包含了安装好的 OpenFace 及 torch、dlib、opencv 等组件，降低了其使用难度.

2.6 故障转移

在“Client-Broker-Service”三级架构中，处于中间的 Broker 代理是整个服务框架中的瓶颈. 如果执行代理任务的主机发生一个致命错误从而在网络中消失，就会造成整个系统的不可用. 在 Bolt 中，使用“主机-备用机”的方式提高系统的可用性[14]. 主机和备用机位于不同的物理服务器上，但运行相同的软件，都可以完成 Broker 代理的功能. 在任何时刻，系统内只存在一个活动的服务器，另一个服务器作为备用机并不接收任何来自客户端的请求. 当主机发生故障时，经过一段时间后，备用机就会接管代理工作，充当活动的服务器.

3 系统验证

为了验证 Bolt 服务框架的性能，使用 Bolt 开发了使用 OpenFace 的实时人脸识别应用. 实验时，使用 Nexus 9 平板电脑作为应用的客户端，用于采集图像，并将处理引擎处理结果返回 Nexus 9 显示；分别使用 Dell Optiplex 9010 和阿里云虚拟机模拟微云以及云提供服务. 各个实验设备的配置如表 1 所示.

表 1 实验设备配置表

实验设备	主要配置、操作系统及版本
Google Nexus 9	CPU: Nvidia Tegra K1, 2.3GHz
	内存: 2 GB
	摄像头: 160 万像素(前置)
	Android 5.1.1
Dell Optiplex 9010	CPU: Intel Core i7-3770 3.40GHz
	内存: 16 GB
	Ubuntu Server 14.04.1, x64
	内核: Linux 3.16.0-30-generic
阿里云虚拟机	CPU: Intel Xeon E5-2650 2.60GHz
	内存: 32GB
	Ubuntu SMP
	内核: 3.13.0-32-generic

3.1 交互感知应用开发过程的简化

Bolt 服务框架的使用使得交互感知应用不必关心底层套接字的使用，直接使用较高层次的通信协议. 例如，使用 Bolt 服务框架开发的部分 echo 客户端代码如下:

```
BoltClient client = new BoltClient();
while(!stop) {
    ZMsg request = new ZMsg();
    request.addString("hello");
    client.send("echo", request);
    ZMsg reply = client.recv();
    if(reply != null) {
    ...
    }
}
```

而使用套接字实现类似功能时，则需要包含创建套接字、设置套接字属性、连接到远端端口、发送并接收数据等操作；并且需要处理连接失败及断开等多种情况. 而使用 Bolt 时，只包含了较高层次的通信方法.

3.2 Bolt 服务框架系统开销

为了测得 Bolt 服务框架带来的系统开销，使用 Bolt 开发了 echo 客户端和 echo 服务. echo 客户端发送简单字符串(“hello”)给 echo 服务；echo 服务在收到任何消息后，回复此消息. 在微云中的不同虚拟机上分别运行 echo 客户端和 echo 服务，记录 echo 客户端发出消息时的时间以及收到回复时的时间，两者的差值即使用 Bolt 服务框架造成的时间开销. 测得的数据如表 2 中第二行所示，第三行是 Gabriel 应用的系统时间开销.

表 2 空载时的时间开销对比

percentile	1%	10%	50%	90%	99%
Bolt(ms)	0.35	0.53	0.66	0.79	1.18
Gabriel(ms)	1.8	2.3	3.4	5.1	6.4

以上测试采用了和 Gabriel 时间开销测试相同的方法，以 1 秒的间隔采集了 3000 个差值数据. 从表 2 的数据可以看出，Bolt 服务框架的系统时间开销远远小于 Gabriel 的系统时间开销. 一方面，是因为 Bolt 采用了 ZeroMQ 作为消息通信的组件，使得通信效率得到大幅提升；另一方面，则是因为 Broker 中间代理的设计，使得 Bolt Client 与 Bolt Service 之间只经过一次网络跳转，减少局域网内的通信时间开销.

3.3 响应时间改进

Bolt 服务框架保持了微云低响应时间的优点. 如

图 4 所示，使用客户端请求 bamos.openface 服务，对 Bolt 和远端阿里云服务器上的服务分别选择 10KB 和 100KB 大小的图片进行了 10000 次测试，获得了响应时间 CDF 曲线. 在使用 10KB 图片进行测试时，Bolt 响应时间的中位数为 261ms，阿里云服务器为 282ms；而当使用 100KB 大小的图片进行测试时，Bolt 响应时间的中位数为 358ms，阿里云服务器为 647ms. 实验结果表明，使用 Bolt 服务框架可以有效降低响应时间.

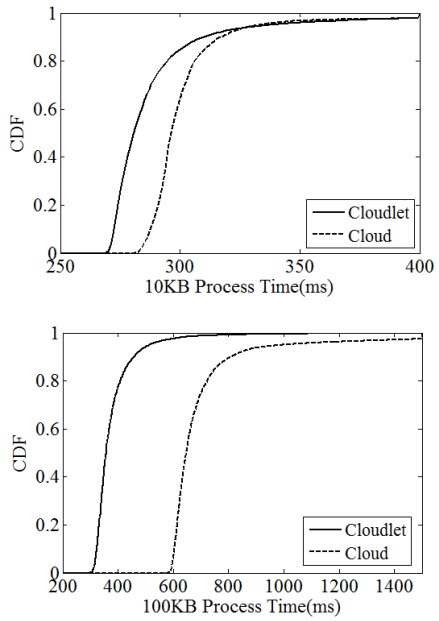

图 4 人脸识别处理时间比较

响应时间的降低取决于两方面的因素：一是局部网络的稳定性好，局部网络的网络延时时间要小于广域网络的延时时间；二是局部网络连接的带宽比广域网络连接要高，这降低了数据量较大的图片、视频等内容的传输时间. 从图 4 中也可以看出，当图片由 10KB 到 100KB 时，响应时间差从 21ms 增加到 289ms，这说明在交互感知应用选择使用较高分辨率的图片或者视频时，使用 Bolt 服务框架可以带来可观的用户体验改进.

4 结语与未来工作

Bolt 服务框架可以简化交互感知应用的开发，且性能良好. 但在当前 Bolt 的原型系统中，依然存在交互感知应用数量少、任务调度模型简单等问题. 在未来的研究工作中，会继续深入研究这些问题，促进交互感知应用的发展.

参考文献

1 百度.BaiduEye_官网.http://baidueye.baidu.com. [2015-11-15].
2 Ra MR, Sheth A, Mummert L, et al. Odessa: Enabling interactive perception applications on mobile devices. International Conference on Mobile Systems. ACM. 2011. 43–56.
3 顾乃杰，赵增,吕亚飞,等.基于多 GPU 的深度神经网络训练算法.小型微型计算机系统,2015,5(5):1042–1046.
4 Satyanarayanan M, Bahl P, Caceres R, et al. The case for VM-based cloudlets in mobile computing.. IEEE Pervasive Computing, 2009, 8(4): 14−23.
5 Ha K, Pillai P, Richter W, et al. Just-in-time provisioning for cyber foraging. Proc. of the 11th Annual International Conference on Mobile Systems, Applications, and Services. ACM. 2013. 153–166.
6 Satyanarayanan M, et al. Elijah Home. http://elijah.cs.cmu.edu/. [2015-10-16].
7 Simoens P, Xiao Y, Pillai P, et al. Scalable crowd-sourcing of video from mobile devices. Proc. of the 11th Annual International Conference on Mobile Systems, Applications, and Services. ACM. 2013. 139–152.
8 Chen Z, Hu W, Ha K, et al. QuiltView: A crowd-sourced video response system. Cs.Cmu.Edu, 2014: 1–6.
9 cmusatyalab/gabriel. gabriel - Cognitive assistance with Google Glass. https://github.com/cmusatyalab/gabriel, [2015-11-10].
10 Ha K, Chen Z, Hu W, et al. Towards wearable cognitive assistance. International Conference on Mobile Systems. ACM. 2014. 68–81.
11 Brandon A, Bartosz L, Jan H,et al. OpenFace: Face recognition with deep neural networks. http://github.com/cmusatyalab/openface. [2016-01-11].
12 薛鹏飞,胡荣贵,胡劲松.基于 ZeroMQ 的分布式系统通信方法.计算机应用,2015,(S2):34–37.
13 Schroff F, Kalenichenko D, Philbin J. FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. 2015. 815–823.
14 Hintjens P. 卢涛,李颖译. ZeroMQ:云时代极速消息通信库. 北京:电子工业出版社,2015.