计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2018.20170867
55(3):572-584, 2018

Web 使能的物端计算系统

彭晓晖[1] 张星洲[1,2] 王一帆[1,2] 朝 鲁[1,2]
[1](中国科学院计算技术研究所 北京 100190)
[2](中国科学院大学 北京 100190)
(pengxiaohui@ict.ac.cn)

Web Enabled Things Computing System

Peng Xiaohui[1], Zhang Xingzhou[1,2], Wang Yifan[1,2], and Chao Lu[1,2]
[1](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)
[2](*University of Chinese Academy of Sciences, Beijing* 100190)

Abstract The rising edge computing paradigm tries to shift some computing tasks from cloud to devices recently, which reduces the computing load of cloud and traffic load of the Internet. The things computing system consists of the devices which are physical world oriented with physical functionalities. It is a great challenge to design a unified system architecture for things computing system because of the system diversity. The architecture of the modern Web system is an efficient solution for the diversity issue. However, due to the resource-constrained feature extending the Web architecture to the things computing system is also very difficult. In this paper, we first introduce the concept of edge computing system and things computing system, and summarize the challenges brought by diversity and resource-constrained features of things computing system. Then, a detailed study of the state-of-the-art technologies, including REST principle, script languages and debugging technique for extending the Web to things computing system, is presented. Most of the related work tried to modify the "Uniform Interface" principle to adapt to edge system. We conclude from the examined literature that things computing system is a massive market, but there is still no unified system architecture which supports both the Web and intelligence. Finally, we present some future research directions for things computing system including the unified system architecture, efficient Web technologies, supporting intelligence and debugging techniques.

Key words things computing system; resource-constrained; Internet of everything; representational state transfer (REST) principles; things device

摘 要 近年来兴起的边缘计算试图将部分计算从云端移到设备端,从而减少云端计算负载和网络传输负载.物端计算系统是边缘计算系统中面向物理世界的终端设备组成的计算系统.由于物端设备具有多样性,设计一个统一的体系结构来支持物端智能应用十分具有挑战.现代 Web 系统的体系结构是解决多样性的有效方案之一,但由于大部分物端设备的资源受限的特性,应用 Web 体系结构十分困难.1)阐述了现代 Web 系统、边缘计算系统和物端计算系统的概念,从组成物端计算系统的设备多样性和资源

收稿日期:2017-11-17;**修回日期**:2017-12-08
基金项目:中国科学院率先行动"百人计划"(Y704061000)
This work was supported by the CAS Pioneer Hundred Talents Program (Y704061000).

受限特性出发分析其面临的挑战;2)针对这些问题和挑战调研了一些基于 REST 的用于边缘计算系统的应用层协议;3)详细调研和评估了 4 个 Web 系统代表性脚本语言,总结了一些试图将这些语言应用于物端设备的工作;4)调研了传统嵌入式系统的调试技术.通过调研得出结论:目前的物端计算系统虽然市场规模巨大,但是仍未形成高效的、统一的体系结构来支撑人工智能应用的大量部署;5)列出了物端计算系统的一些重要研究方向,包括统一的体系结构、高能效 Web、支持物端智能和物端调试技术.

关键词 物端计算系统;资源受限;万物互联;表述性状态转移原理;物端设备

中图法分类号 TP391

随着接入网络的设备日益增加,数据规模也呈爆发式增长[1].现有的物联网采用的是云计算模式,海量数据传输到云端计算加重了网络传输和云端的计算负载.传统的计算模式无法满足海量数据大规模传输和计算、以及即将到来的人工智能时代应用大量爆发的需求[2].为了解决以上问题,研究者提出了边缘计算,它是将云端计算负载部分移到设备端的一种新分布式计算模式[3-4].该模式的计算行为发生在数据源到数据中心路径的任意网络和计算资源上.相比于云计算,边缘计算模式能够提供更加高效的数据访问和计算,减少云服务器计算负载和网络传输负载,同时能提供更加实时的控制和反馈,以及提高边缘计算系统[5]应用程序的服务质量和系统可靠性.

边缘计算也常常被称为雾计算[5-6].文献[5]定义边缘计算为"网络边缘执行计算的一种新型计算模型",其核心理念是如何利用接近数据源的计算资源执行计算任务,并未严格界定所涉及的设备类型,理论上需要考虑从数据生产端到处理端之间(不包括两端)的任一设备.如图 1 所示,从物端到云端完整设备链中有大量类似于传感器、控制器等分布在环境中设备被边缘计算概念所忽视,这类设备面向物理世界,多具备物理功能,我们可称之为物端设备.这类设备多数具有资源受限的特点,但仍具备不可忽视的计算能力,离物理环境和用户近的特点使其具有一定的计算优势.因此,我们在边缘计算的基础上进行相关概念延展和聚焦研究,我们提出了以物端设备为主的分布式计算系统和物理环境,称为物端计算系统.

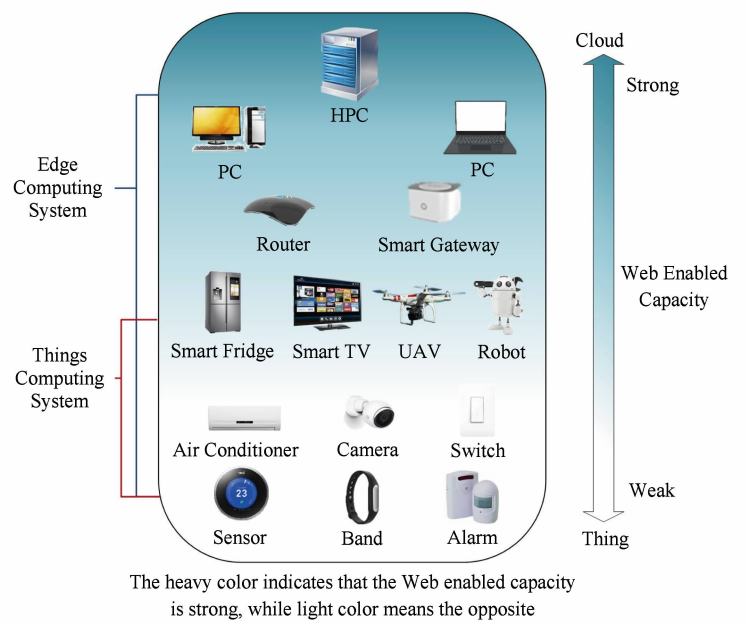

Fig. 1 The relationship between edge computing system and things computing system

图 1 边缘计算系统与物端计算系统的关系示意图

鉴于现代 Web 系统的灵活性和可伸缩性优点,物端计算系统也需要全面的 Web 使能的支持.在本文进一步研究中,我们将设备依照 Web 使能能力进行划分,即强 Web 使能设备和弱 Web 使能设备.强 Web 使能设备包括服务器、PC、手机、智能电视等,可原生运行完整的操作系统及现代 Web 的全栈协

议(TCP/IP 协议族). 而弱 Web 使能设备,如部分传感器、可穿戴设备、智能硬件等,由于资源受限,不能支持完整的原生网络协议栈. 因此,本文将主要研究弱 Web 使能设备为主的物端计算系统,其中,边缘路由等强 Web 使能的物端设备不在调查范围内.

图 1 显示了 Web 使能能力设备、边缘计算系统和物端计算系统的关系. 强 Web 使能区域表示已经成功地应用了 REST Web(基于 REST 基本原则建立起来的现代 Web 体系结构)[7]的服务器、个人计算机(PC 互联网)和智能手机(移动互联网)组成的现代 Web 系统,它们均有相对统一的体系结构. 例如个人计算机有 X86 + Windows/macOS + REST 平台,而智能手机有 ARM + Android/iOS + REST 平台[8]. 弱 Web 使能设备区域,由于系统的多样性和资源受限等特性,其没有统一的体系结构模型,无法适应未来万物互联时代人工智能应用的需求. 物端计算系统需要遵循现代 Web 架构设计原理,继承其体系结构,并针对物端系统面向物理世界的特性做出相应的扩展或改进,以适应灵活的、高动态的、高度异构的计算环境. 主要有 3 个理由:

1) 借鉴云计算和移动互联网的优点. 云计算模式下,用户可以通过移动设备使用 Web 技术随时随地访问云端资源. 物端计算模式下,这种需求仍然存在,并且通过 Web 技术,用户还可以访问和控制端的资源,例如查看环境数据、控制家用电器开关等.

2) 兼容现有的 Web 系统. 采用 Web 体系结构的物端计算系统可以兼容现有的个人计算机和手机互联网,但需要针对物端设备特性和计算环境做相应的改进.

3) 解决物端设备的"昆虫纲悖论"[8]. 物端设备如自然界的"昆虫"一样种类繁多,在硬件平台、操作系统和通信协议等方面均存在多样性的问题,这给物端计算系统应用的开发带来极大挑战. 在传统 PC Web 和移动 Web 领域并不存在碎片化问题,根本原因是它们都应用了表述性状态转移(representational state transfer, REST)[7]架构风格. 因此在物端计算领域,支持 REST 的 Web 技术能够解决"昆虫纲悖论"的问题.

然而,现有的 Web 技术无法直接迁移到物端计算系统中. 1)由于应用场景和成本等因素的限制,物端设备的计算、存储、传输等能力较弱,他们不具备直接支持 Web 应用的能力;2)这些设备在硬件、系统、通信、物理功能等方面均具有多样性的特点. Web 体系架构,需要经过改进才能应用于物端设备. 目前的研究主要集中在修改 Web 的一些实现,增强"统一接口"原则[9-10],来满足一些特有的应用需求. 另外一些研究也关注脚本语言的轻量化[11],并运用在物端设备上. 这些工作通常只关注了物端系统的部分需求. 本文认为有 2 点需要在未来的体系结构设计中得到优化与加强:

1) REST 基本原理[7]. 为了与现有的 Web 系统兼容,我们需要继承已有的 Web 的基本设计原理 REST. 同时,为了适应资源受限的环境,实现面向物理世界的智能感知与控制,需要对当前 Web 的实现进行轻量化的改进.

2) 按需代码[7],即 Scripting 功能. 脚本语言简单易用、可移植性强,适合物端计算系统的人工智能应用. 按需代码在 REST 里是一个可选的原则,但对实现物端设备智能应用来说是一个必须实现的原则.

物端计算系统与现在的 Web 和物联网[12]的本质区别是数据存储、计算和交互(包括感知和控制)更靠近物理设备和用户[13]. 因此,需要加强物端设备的计算能力来承载更多的智能处理,同时需要与现有的 Web 系统融合,组建协同工作的、鲁棒的、本地化的智能分布式系统. 本文对 Web 使能的物端计算系统的概念、现状和发展方向进行了综述性研究,具体贡献有 4 个方面:

1) 首次阐述了物端计算系统概念及适用范围.

2) 比较分析得出:Web 使能技术及脚本语言可有效解决物端设备多样性和资源受限问题.

3) 从 REST 架构设计、脚本语言的应用和调试技术的发展 3 个方面,归纳对比现有应用于物端计算系统的 Web 技术的发展现状.

4) 分析了未来物端计算系统研究的挑战,为物端计算领域指明了值得进一步研究的方向.

1 Web 技术在物端计算系统上的应用

1.1 物端计算系统现状

物端计算系统的主要特征是:多样性和资源受限,不能很好地支持 Web 体系结构. 如图 2 所示,多样性主要包括平台多样性和应用场景的多样性. 接下来我们从硬件平台、操作系统和通信协议、应用场景等方面来描述其多样性.

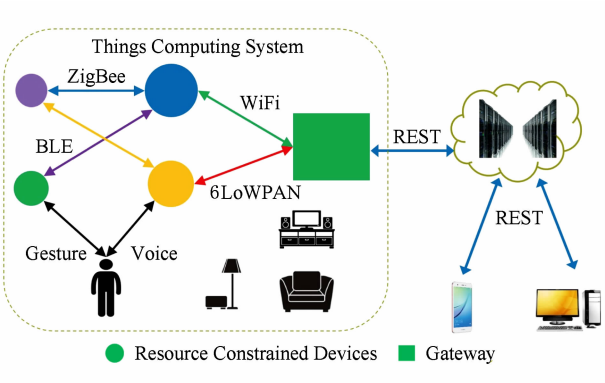

Fig. 2 The diversity of things computing system

图 2 物端计算系统的多样性

1) 硬件平台. 物端设备的硬件通常为专用功能而定制. 它们的处理器/微控制器在数据宽度、指令系统、体系结构等方面有很大的差异性[14]. 这些设备可按照计算、存储和通信等能力的大小进行分类. 如表 1 所示,IETF 将物端设备按照存储能力粗略地分为 3 个类别[15]:Class 0(C0),Class 1(C1)和 Class 2(C2). C0 设备通常指的是资源极端受限的传感器类设备,没有直接连入现有互联网的能力. C1 设备能力相对强大一些,但是它们仍然无法直接使用现有的类 HTTP 协议族与互联网中其他节点通信. C2 设备的资源受限程度相对较小,能支持一些裁剪后的互联网通信协议栈,运行简单的脚本程序.

Table 1 The Classification of IoT Devices Based on Memory of IETF

表 1 IETF 低端物联网设备依内存分类表[15]

Name(Abbreviation)	RAM Size/KB	ROM Size/KB
Class 0(C0)	≪10	≪100
Class 1(C1)	≈10	≈100
Class 2(C2)	≈50	≈250

2)操作系统. 文献[16]指出:当能够为这些高度异构的硬件平台提供统一通用的 API 和 SDK 的一款或几款嵌入式操作系统出现时,物联网应用才能无处不在. 早期嵌入式操作系统通常为特定的硬件平台或应用需求而设计的,平台间相互不能兼容,通用性差[17]. 近年来,一些嵌入式操作系统开始致力于发展成为通用性操作系统,如 TinyOS[18],Contiki[19],T-kernel[20]等,表 2 对它们做了比较,它们大多支持多线程和动态内存,也大多支持 C 语言,内存最小的为 TinyOS,只有 1 KB.

3) 通信协议. 操作系统需要根据硬件平台支持不同的通信协议栈,这些协议栈大多具有低功耗低速率的特点,但他们之间不能够“互通、互操作”[22]. 物端计算系统中常见的无线通信协议有 Z-Wave[23],ZigBee[24], Bluetooth Low Energy (BLE)[25],6LoWPAN[26]等,表 3 对它们在常用模式下的特性做了简单比较. 这些通信协议支持的应用层协议大多数不能与现有的互联网应用层协议(HTTP)兼容,通常需要在应用程序层做协议的相互转换工作. 6LoWPAN,CoAP[27]协议通过运行有 TCP/IP 协议栈的网关节点,让资源受限的设备以兼容 HTTP 的方式连接到现有的互联网系统中. 很多嵌入式操作系统(例如 Contiki[19],TinyOS[18],Micro T-kernel[28],mbed[29]等)均实现了对 6LoWPAN 和 CoAP 的支持. 这种方式将大大降低应用程序开发和部署的复杂度.

4) 应用场景. 物端计算的应用场景十分丰富. 例如智能家居、交通、电网、智慧交通、车联网等. 每一个应用场景对硬件,网络和用户体验的要求各不相同. 然而在智慧城市里,常常需要这些不同的系统之间相互配合来提供更加优质的服务[30]. 现有的物端计算系统没有统一的体系结构,各个应用领域之间基本是相互隔离的,容易形成信息孤岛. 因此,有必要将物端设备计算机化,形成统一体系结构和计算平台.

Table 2 The Comparison of Operation Systems in Things Devices

表 2 物端设备操作系统比较[21]

Operation System	Language	Minimum Memory/KB	Multi-thread	Dynamic Memory
TinyOS	NesC	1	partial	yes
Contiki	C	2	yes	yes
LiteOS	C	4	yes	yes
RiotOS	C/C++	1.5	yes	yes
T-kernel	C	2	yes	yes

Table 3 The Comparison of Communication Protocols of Things Computing System in Normal Operating Mode

表 3 常见物端计算系统通信协议在常用工作模式下的特性比较

Communication Protocol	Transmission Speed/Kbps	Power Consumption	Transmission Coverage/m	Protocol Standard
Z-Wave	100	Extremely Low	90	
ZigBee	250	Low	100	IEEE 802.15.4
Bluetooth Low Energy 4.0	1000	Relatively Low	>100	IEEE 802.15.1
6LoWPAN	250	Low	100	IEEE 802.15.4

物端设备另外一个显著特征是资源受限. 每个设备的功能较为单一,大量部署在各种复杂环境中,要求具有很低的功耗水平. 这些特性限制了它的计算、存储和通信能力,具体表现在 3 个方面:

1) 计算能力. 通常采用嵌入式处理器,其表现形式通常为一个单片机或微控制器(MCU),基本都是单核结构,其主频一般不超过 1 GHz.

2) 存储能力. RAM 和 ROM 空间大多数都在 100 KB 以内,传统计算机的应用程序开发技术和运行环境不适用于这种资源受限的物端计算环境.

3) 通信能力. 它们大多采用无线通信的方式. 现有的基于 PC 和智能手机的通信协议栈无法运行在物端设备上. 学术界和工业界设计了很多低功耗、低成本的无线通信协议[21],但是传输能力通常受到极大限制. 资源受限的特性限制了 PC 互联网和移动互联网的基础体系结构 REST Web 在物端计算系统的应用.

1.2 物端计算系统面临的挑战

1) 统一的体系结构. 为了应对物端系统物端设备多样性问题,同时支持未来人工智能应用,物端系统需要一个通用的软件架构. 文献[20]早在 1987 年提出智能物体互联网的概念,指出分布在全球各地的智能设备以松耦合的方式组成一个超功能分布式系统(highly functionally distributed systems, HFDS)[20]. 随着移动设备性能的增强,PC 互联网的 Web 体系结构成功的拓展到智能手机端,从而带来了移动互联网的繁荣. 然而,物端设备支持现有的 Web 体系结构仍然具有很大的挑战,万物互联的网络很难形成统一的体系架构,从而实现 HFDS 的愿景.

如图 3 所示,Web 体系结构中的 REST 风格 API 可为物端计算系统组成部分之间交互提供统一的接口,设备间通过基于 REST 的应用层协议相互连接. 本文提倡将基于 REST 设计原理的 Web 技术体系拓展至物端设备,统一其体系结构,解决因多样性带来的应用碎片化问题. 物端系统中设备需要高频率地与真实世界交互,甚至一些应用的每次计算都需要感知物理世界的数据或物理世界的反馈作为输入,这种应用场景与传统 Web 系统有着很大的差别,我们有必要针对新的环境,为物端计算系统设计改进的 REST 架构风格,来指导物端系统软件架构的设计. 同时,我们必须从软硬件协同的角度出发,设计高效的、低功耗 Web 体系,使其能够应用在资源受限的物端设备上.

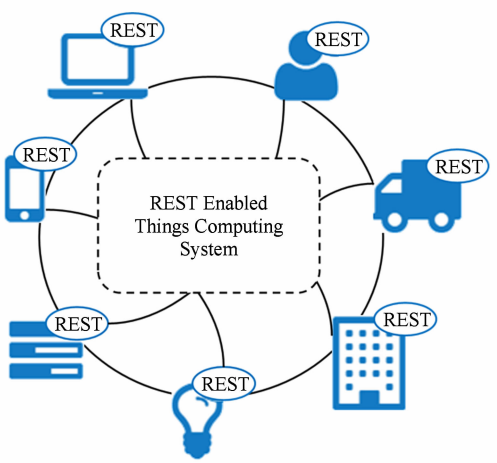

Fig. 3 REST APIs provide uniform interaction interface for things

图 3 REST API 可为物端系统元素间交互提供统一接口

2) 不支持智能. 目前的海量物端设备功能单一,资源受限,只是作为数据收集和命令执行的工具,无法有效地处理智能任务. 然而,物端计算系统很多应用场景需要模型在设备端进行本地化地、实时地、在线学习与进化[31]. 考虑物端设备的多样性和资源受限的特点,我们需要从芯片到应用架构全新的创新和设计才能高效地支持未来的人工智能应用. 例如设计低功耗、高能效、面向物理世界的智能物端处理器,在硬件上原生的支持智能. 在系统上,形成统一的体系架构,支持设备间主动交互与协同. 在应用架构上,将算法的模型分布式化,从而利用物端设备有限的计算能力进行局部学习. 这样可以充分利用海量设备的有限计算能力,同时减轻数据上传到云端计算时造成的网络负载和云端智能计算负载[5].

3) 开发难度高.应用开发人员要想在物端计算系统上开发程序,通常需要具备从芯片、操作系统到高级编程语言等完备的全栈知识.物端系统通常有数十种微处理器架构、上百种嵌入式操作系统[16]、数十种通信协议[21].一个嵌入式程序员通常需要使用汇编语言初始化 SoC 上的各种资源、选择合适的操作系统和通信协议栈来构建整个软硬件系统.这对程序员的能力和经验要求十分高,阻碍了物端系统应用程序的开发和普及.而且大多数应用是为了特定硬件实现特定功能而编写,依赖于特定的硬件资源.为特定硬件平台编译好的应用通常无法运行在另外一个平台上.

物端设备大多数为传统的嵌入式设备,其应用开发使用交叉编译环境,需要配置复杂的编译调试环境和特定的硬件调试器,增加了调试的难度[32].此外,由于物端计算系统通常分布在复杂的居住环境中,需要随时随地的动态交互.开发人员很难预测系统所有的限制和程序执行情境,从而在部署前验证程序所有可能的分支.

综上所述,物端计算系统输入输出与物理环境高度相关,而且未来物端系统会承担更多的智能处理任务.基于 Web 技术和脚本语言的物端计算系统架构能够有效解决物端设备多样性的问题,并且能够支持人工智能应用.接下来,本文将调研基于 REST 的 Web 技术应用于物端计算系统的一些研究工作.

2 REST 架构在物端计算系统中的应用

REST[7]是 Web 系统软件架构的基本风格和原理.其基本原理包括:客户-服务器风格、无状态、缓存、统一接口、分层系统、和按需编码.合理的设计和强大的生命力使其成为现代 Web 系统设计的基本原理.文献[33]是早期倡导通过 REST 原理将物整合到 Web 中的研究之一.文中提出构建物理世界的"智能物体"网络,并将其融入现有的 Web 系统,即 Web of Things[34].其优点是可以利用成熟的 Web 系统的技术(例如 JavaScript,Ruby,HTML 等)和机制快速构建可用的应用.

随着边缘计算的兴起,学术界和工业界出现了许多致力于将 Web 技术扩展到资源受限环境的研究.然而,由于物端计算系统的多样性,这些研究大多数没有考虑万物互联场景下人工智能应用的新需求,只针对特定的场景和问题对 Web 的一些技术进行裁剪和优化,应用在特定的场景.下面将介绍 5 个典型的基于 REST 风格的物端计算系统应用层协议:pREST (pico-REST)[9], ubiREST[10], TinyREST[35], seaHTTP[36], CoAP[27].这些协议都支持客户-服务器、无状态和分层的原则,因此,在表 4 中只列出是否支持缓存、统一接口和按需代码原则.

Table 4 The Representative Application Layer Protocols in Things Computing System Based on REST

表 4 基于 REST 的代表性物端计算系统应用层协议

Protocol	Cache Support	Uniform Interface Support	Code on Demand
pREST(pico-REST)	no	Partially support resource operations, add "SUBSCRIBE" method	unknown
ubiREST(Pervasive REST)	yes	Add "INSPECT" method,support abstract URL	yes
TinyREST	unknown	Partially support resource operations	unknown
seaHTTP	yes	Add "BRANCH,COMBINE" methods, add space-time attributes to URL	unknown
CoAP	yes	yes	yes

从表 4 中可以看出,这类协议大部分支持 REST 的客户-服务器风格、无状态和分层系统的约束.但大部分都不支持缓存,也没有强调按需代码的原则.其中 pREST 简化了统一接口原则(只支持原有的 GET 和 POST 方法),增加了 SUBSCRIBE 方法,允许表达对给定资源的兴趣,并在该资源发生变化时被通知,从而提高上层应用的实时性.ubiREST 是建立在 Pervasive REST (P-REST)[10]架构风格的一个中间件.而 P-REST 通过加强统一接口这一原则来实现对泛在网络环境应用程序的支持.P-REST 定义了一个新的资源操作方法"INSPECT",该方法允许获取目标资源的元数据(Meta-data).P-REST 还通过引入抽象的统一资源标识符(abstract URI, 即 aURI)[10]来支持标识一组资源.seaHTTP 提供了 2 种新的方法(BRANCH 和 COMBINE),用于并发群组多资源请求的拆分和合并[36].同时在标准 URI 中加入时空属性支持动态群组命名和实体资源表述.

CoAP 支持在资源受限的情况下实现简单、低开销和多播的传输,旨在资源极端受限的节点和网络中以适当的形式实现 REST 架构.学术界和工业界也涌现了很多基于 CoAP 的物联网应用体系架构方面的研究,例如 uID-CoAP[37],Lithe[38],jCoAP[39],mjCoAP[40],CoAPthon[41]等. CoAP 是目前应用最为广泛的资源受限环境的基于 REST 的应用层协议,基本上解决了一些物端设备应用层传输协议与现代 Web 的 HTTP 协议的兼容问题.

文献[42]还针对智能互联环境提出了一种可扩展的设备服务发现机制;文献[43]提出了智能家居协议(SHP),使应用可以使用标准的 HTTP 方法来检索和操纵设备的资源;文献[44]提出了一种通过基于云的应用服务器和一致的 REST 风格的编程模型为物端设备提供类似 Web 脚本的架构.

这些研究大多都试图通过修改或加强统一接口这一原则,使 REST 架构更加适合有高动态性、高扩展性、面向物理世界的物端计算系统.然而在万物互联时代,人工智能应用的兴起,对物端计算系统提出了更多的需求.因此,研究者们必须谨慎地改进原有的 REST 设计原理,设计更加灵活的、可扩展的系统架构原则,指导构建统一的物端计算系统体系架构,从而支持未来人工智能应用的大量部署.

3 物端设备上的脚本语言

脚本语言是 REST 按需代码原则的具体实现,相对于 C/C++和 Java 等系统编程语言,其具有更高层次的抽象.它们简单易用、可移植性强,适合具有多样性的物端设备上应用程序的开发,例如 JavaScript[45],Lua[46],Ruby[47],Python[41].本节介绍这 4 种语言和其为适应物端计算系统所做的改进,并从内存占用(memory footprint)角度对它们进行比较.

基于 JavaScript 将脚本编程拓展到资源受限环境的工作主要有:IoT. js[48],Cylon. js[49],Node. js[50],Device. js[51]等. IoT. js 是由 Samsung 公司推出的基于轻量级 JS 引擎 JerryScript[11]的 IoT 应用程序的平台,它针对 IoT 器件在 CPU 性能和内存占用方面限制,做了大量优化工作;Node. js 基于 Google V8 JavaScript 引擎和 libuv 事件 I/O,使得 JavaScript 既可用于客户端,又可用于服务端,实现了客户端-服务器语言的统一;Cylon. js 是面向机器人和物联网的 JavaScript 框架,支持多种物联网硬件平台;Device. js 允许用户在选择和控制各种智能设备时构建 Web 应用程序.

另外 Lua,Ruby,Python 等也经常被裁剪用于资源受限的环境里. mRuby[52]是为 IoT 应用定制的轻量级 Ruby 编程语言;MicroPython[53]是 Python 语言的变体,使用它开发的应用可以做到离线运行,给调试带来很大的便利,它集成了多种网络通信协议,适用于需要多种网络通信协议的物联网领域.

物端设备的内存资源受限,因此编程语言运行时内存占用的大小对物端计算系统十分重要.由于实验环境的限制,无法评估一些语言变种的各项性能,本文测试 Node. js,Lua,Ruby 和 Python 这 4 种语言编写的 Helloword 程序运行时内存占用情况.实验平台为 MacBook Pro,处理器为 2.7 GHz Intel Core i5,操作系统为 OSX10. 12. 6,4 种语言编写的测试程序均输出"Hello World"字符串,使用 top 命令采集内存占用.表 5 是不同语言的内存占用对比.

Table 5 The Comparison of Memory Footprint of Scripts

表 5 脚本语言内存占用比较

Language	Version	Code	Memory Footprint/KB
Node. js	6. 2. 0	console. log("Hello World");	12
Lua	5. 3. 0	print("Hello World");	4
Ruby	2. 0. 0	print("Hello World")	24
Python	2. 7. 0	print "Hello World"	8

此外,本文引用了程序语言测试平台(the computer language benchmarks game)[54]对上述语言做测试结果.其基准测试程序为 fasta 格式的转换,平台处理器为 4 核 2.4 GHz Intel Q6600,操作系统为 Ubuntu™ 17. 04 Linux x64 4. 10. 0-28-generic,测试结果如表 6 所示:

Table 6 The Result From the Computer Language Benchmarks Game

表 6 程序语言测试平台的测试结果

Language	Runtime/s	Memory Footprint/KB	Code Size/B
Node. js	9.79	35 012	1 745
Lua	45.92	2 828	1 052
Ruby	108.36	107 984	973
Python3	110.91	8 024	977

Note: The code size is calculated by the GZip compressed bytes size.

由表 6 可知,Lua 的内存占用量是最少的,Node. js 是运行速度最快的.因此,在存储资源非常

珍贵的物端设备上,Lua 最适合用来改进,作为其应用编程语言.

4 物端调试技术

物端设备大多是传统的嵌入式设备,因此大多采用的是传统嵌入式的交叉编译调试技术[55].程序通常在宿主机上开发,而在目标机上运行.随着嵌入式操作系统和应用程序的功能和规模不断扩大,这种复杂高难度的调试过程逐渐成为这些设备上应用发展的瓶颈.现有的嵌入式调试方法可分为软件模拟、硬件模拟(包括 ROM 仿真和在线仿真 2 种方法)、监视器和在线调试等 4 种方法[56-57],下面分别对其进行介绍.

1) 软件模拟.为了降低调试难度,将调试工具和需要调试的程序均运行在计算资源丰富的宿主机器上,通过调试工具来模拟程序在目标机上行为的方式称为软件模拟调试.然而,软件模拟目标机处理器行为的方式,有可能因为处理器的指令集、体系结构的不同造成执行时间、中断响应等系统重要特征上的巨大差异,无法反应程序在目标机上的实际运行状况.

2) 硬件模拟.分为 ROM 仿真和在线仿真 2 种方式.ROM 仿真器又称为 ROMEmulator[58],通常由 RAM 以及附加电路组成.程序通过串口等通信方式下载到 ROM 仿真器的 RAM 里,模拟在目标机上的运行.这种方式避免了程序调试过程中频繁操作目标机的存储设备.ROM 仿真器通常针对特定 MCU 架构设计的专用仿真硬件,使用这种方式调试会显著增加成本.在线仿真又称为 In-CircuitEmulator (ICE)[59].它可以确保在线调试器在目标系统崩溃后仍然能控制程序的运行,再现程序崩溃"现场".但是,仍需要针对特定的平台定制价格昂贵的硬件仿真器.

3) 监视器.即 Monitor 调试方法[60],由于资源受限,这种方式通常会将传统的调试器拆分,分别运行在目标机和宿主机的操作系统中,使用串口、USB 等常见串行通信方式,遵守相同的远程调试协议来实现通信.这种方式通常无须特定硬件,价格低廉,使用较为方便,能再现程序在目标系统上运行的真实状况.然而这种方式需要目标操作系统和调试模块的支持,同时会增加目标系统资源开销.例如 GdbStub[61]技术是对 MCU 程序进行监控调试的一种方法,该方法为开发嵌入式软件的调试工具提供了技术基础.

4) 在线调试(in-circuit debugging, ICD)[62].在线调试是目前嵌入式开发主要的调试方式.由于硬件仿真器的价格昂贵,且需要针对特定平台定制,使得开发成本十分高.解决方法之一是将调试接口标准化、集成在处理器中.例如 Motorola 的背景调试模式(BDM)[63]接口和联合测试小组的 JTAG 接口[64].其中 JTAG 是一种开放的国际标准测试接口协议,现有的大部分嵌入式器件都支持 JTAG 调试接口.JTAG 在保持强大功能的前提下,大大降低了在线调试的成本.

除了上述 4 个调试方法外,许多研究者使用软硬件相结合的方法实现远程调试.文献[65]提出了一种在协同设计环境下的交叉调试技术 BackC 方法.在软件调试中,可逆调试技术[66-67]是解决复杂程序调试过程中错误复现、错误定位问题的有效技术,近年来成为研究的热点.文献[68]提出了一种基于硬件辅助模块的适用于多核平台的记录-调试方案;文献[69-70]等也分别研究了可逆调试技术;文献[71]通过使用一个程序将串口和 Telnet 协议进行转换,主机通过 Telnet 与目标机通信.表 7 对 4 种主要调试方法进行了简单对比,其中,在线调试方法的系统开销小,功能多,通用性强并且成本低,适合物端系统物端设备上的应用开发调试.

Table 7 The Comparison of Debugging Methods of the Embedded System

表 7 嵌入式系统调试方法比较

Debugging Method	Runtime Overhead	Function Support	Difficulty	Cost	Features
Software Emulator	high	few	simple	low	Unable to simulate timing errors
Hardware Emulator	low	much	hard	high	Customized for special platform
Monitor	high	few	medium	low	Require support of operating system and debug module
In-Circuit Debugger	low	much	medium	medium	Strong generality

5 未来研究方向

物端计算作为新兴计算模式应用前景广阔,涉及传感器、芯片、网络、人工智能等多个研究方向[72].因此也面临众多挑战.本节总结物端计算系统研究的趋势和遇到的挑战,列举了值得进一步研究的方向:体系结构、高效能 Web、支持智能和应用程序调试等.

1) 体系结构.物端系统的多样性阻碍了其统一的体系结构的形成.由于物端设备资源受限的特性,在 PC 互联网和移动互联网取得巨大成功的 Web 技术无法很好地直接应用在物端计算系统.随着越来越多的物端设备连入互联网,统一体系结构的研究与形成变得越来越迫切.如图 2 所示,物联网模式下,设备间的架构极为混乱,计算系统和通信方式十分多样化.

文献[8]提出了可应用于物端计算领域的 SWoT 架构,如图 4 所示,将已经在 PC 互联网和移动互联网上得到验证的 REST 架构拓展到物端计算领域.一个 SWoT 系统由一个或者多个物端控域(ΦZone)构成,物端控域是物理空间的抽象(例如智能家庭).它与云端之间通过网关(ΦPort)连接,ΦZone 内部的多个物端设备通过基于 REST 的协议 ΦREST 交互,ΦPort 可以在 ΦZone 内外自动转换协议,在内部使用 ΦREST,而外部使用 REST.该体系结构将 PC 的 Web 技术拓展至物端设备,让计算尽可能更靠近设备和用户.

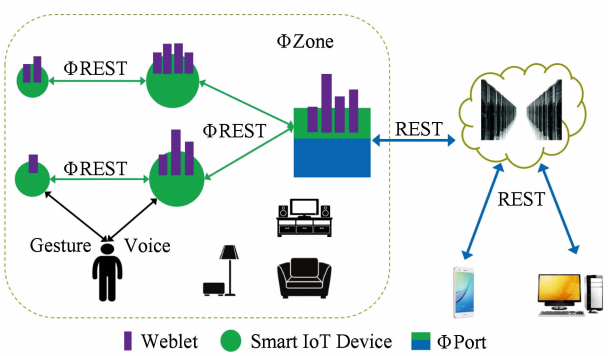

Fig. 4 The architecture of Smart Web of Things (SwoT)

图 4 Smart Web of Things (SwoT)架构[8]

文献[73]为物端计算系统提出了 IoT-Aggregator 的架构,如图 5 所示,其主要思想是为每一个设备分配一个全球唯一的 ID (ucode)[74],云端或边缘上存储该 ID 关联的所有数据,所有计算和访问控制都在云端或边缘上进行.这种架构继承了传统物联网的计算模式,其终端设备主要任务是采集数据和执行命令.

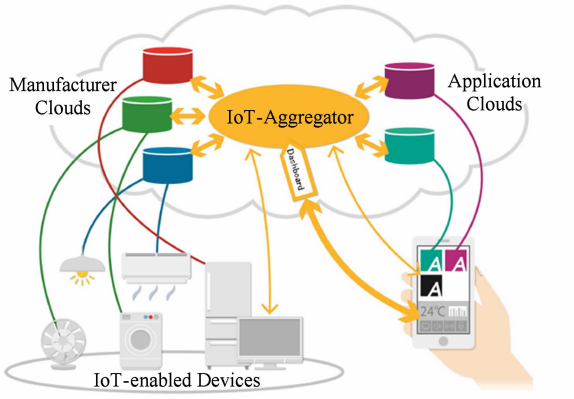

Fig. 5 The architecture of IoT-Aggregator

图 5 IoT-Aggregator 架构[73]

2) 高能效 Web.由于物端设备资源受限,应用于其上的 Web 系统必须是高能效的.这里的高能效指的是轻量级、速度快、功耗低.文献[8]提出了一个从硬件到软件完全开放的技术栈 Φ-Stack,为物端设备提供智能、高效的 Web 支持.如图 6 所示,Φ-Stack 是一个全新的、软硬件协同设计的系统栈,主要模块包括 ΦPU,ΦOS,ΦDK 和应用程序 Weblet.所有模块的设计都优先考虑设备的物理功能.

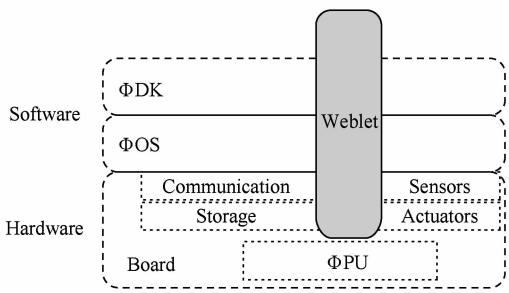

Fig. 6 The structure of Φ-Stack

图 6 Φ-Stack 结构图[8]

其他研究多集中于系统的某一部分实现部分目标.例如在硬件上,很多厂商将计算、通信、传感等全部集成在一个芯片上,并尽可能地减少功耗.操作系统的主要设计目标则是占用更小的内存空间,为应用预留更多计算资源,提供更全面的功能(例如支持 HTTP 协议栈等)[75-76].

3) 支持智能.目前 AI 算法大多需要大量的数据和很强的计算能力,一般运行在云端.而由于资源限制,物端设备无法运行现有的绝大部分智能算法.我们需要在硬件和算法 2 个层面进行研究.在硬件

上,需要设计面向物端设备的人工智能芯片,它应具有速度快、功耗低、面积小并且可重塑的特点,文献[8]介绍了其为物端设备设计的一款基于 RISC-V 指令集的低功耗智能处理器 ΦPU. ΦPU 包含 AI 核、大核(通用核)和用于监视和控制传感器的小核. 其中 AI 核是一个能根据用户需求可重塑的神经网络处理引擎,实现了可重塑 AI 的概念.

在软件上,应该考虑到人的参与因素,具体需要根据物端计算场景设计轻量级智能算法,需要基于小数据集进行实时在线学习、迁移学习,需要提出适用于物端计算场景的计算框架等. 弗罗里达大学提出了一个注重用户满意度的 CNN 推理框架——Pervasive CNN (P-CNN)[77]. P-CNN 在物端计算的场景下满足终端用户的多样性需求,为不同的推理任务提供最佳的用户满意度. P-CNN 框架由跨平台离线编译(cross-platform offline compilation)和运行时间管理(runtime management)两个部分组成,可根据用户的需求生成不同的内核. 该架构在符合可接受准确率的要求下动态地确定一个最快的内核,内核调度器会为每一层神经网络分配出最优的计算资源,并根据用户的反馈选择最优内核.

4) 物端调试技术. 物端计算系统本身受物理环境影响大,而 REST 这种松耦合架构会给系统软件带来更大的不确定性,具体体现在响应延迟上. 这种不确定性给物端系统程序的调试带来尤其巨大的挑战,程序员可能无法使用现有的调试的方法来验证程序的正确性. 为了设计高效、安全、可移植的 Web 底层代码格式,研究者们提出了名为 WebAssembly[78] 的字节码技术. 该技术具有高效率、速度快、安全、可调试等特点,然而 WebAssembly 的设计并没有考虑到物端设备资源相对受限的特性.

6 结束语

信息技术的发展即将进入万物互联时代,联网的每个节点是计算机化、智能化、可重塑的[79],需要通用的软硬件体系架构支撑应用的无缝运行和系统间的自主协作. 本文详细调研了将 Web 技术应用到物端计算系统的一些工作. 我们可以得出结论:物端计算系统市场规模巨大,但设备资源受限、功能弱. 其尚无一个实际的既支持 REST Web 架构,又支持智能的系统平台,即没有一个统一的体系架构. 大多数研究只是针对某一个方面的问题进行优化. 针对物端系统的多样性和资源受限特性,文献[8]提出了一个软硬件协同设计的开放式架构栈 Φ-Stack. 通过"软硬件高度协同设计"和"原生的支持 Web 和智能"2 个核心创新手法来设计和构造高效能的、适合物端设备的 REST Web 体系结构. Φ-Stack 的最终目标是将物端设备计算机化,从而彻底地改造现在的物端计算系统.

本文在第 1 节分析了物端计算系统的现状和存在的问题与挑战. 然后,调研和分析了学术界和工业界为应对这些挑战而做出的努力. 在第 2 节详细分析了 5 个典型的基于 REST 的物端计算系统应用层协议的特性. 这些协议大多数只针对"统一接口"原则进行改进,并不能满足物端计算系统的需求. 第 3 节中,我们详细调研和分析了一些代表性的脚本语言,并对它们的性能做了简单的评估. 其中 Lua 开发出的脚本程序占用存储空间最小,适合用来改进并应用于资源受限的物端计算系统. 第 4 节详细调研了嵌入式系统使用的应用程序调试技术. 最后,本文总结了未来物端计算系统需要致力的 4 个关键问题,包括统一的体系结构、高能效 Web、物端智能和物端调试技术. 随着技术不断进步,物端计算系统、移动互联网、桌面互联网最终会组成一个万物互联网络,成为未来人工智能应用的基础设施. 为"万物互联"设计一个相互兼容的软件架构风格来指导其体系结构的建立,会为即将爆发的人工智能应用的爆发打下坚实的基础.

参 考 文 献

[1] Evans D. The Internet of things: How the next evolution of the Internet is changing everything [R/OL]. San Jose, CA: CISCO, 2011. [2017-11-15]. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] Satyanarayanan M. The emergence of edge computing [J]. Computer, 2017, 50(1): 30-39

[3] Shi W, Dustdar S. The promise of edge computing [J]. Computer, 2016, 49(5): 78-81

[4] Ahmed A, Ahmed E. A survey on mobile edge computing [C] //Proc of the 10th Int Conf on Intelligent Systems and Control. Piscataway, NJ: IEEE, 2016: 1-8

[5] Shi Weisong, Sun Hui, Cao Jie, et al. Edge computing—an emerging computing model for the Internet of everything era [J]. Journal of Computer Research and Development, 2017, 54(5): 907-924 (in Chinese)
(施巍松, 孙辉, 曹杰, 等. 边缘计算: 万物互联时代新型计算模型[J]. 计算机研究与发展, 2017, 54(5): 907-924)

[6] Bonomi F, Milito R, Zhu Jiang, et al. Fog computing and its role in the Internet of things [C] //Proc of the 1st Edition of the MCC Workshop on Mobile Cloud Computing. New York: ACM, 2012: 13-16

[7] Fielding R T, Taylor R N. Principled design of the modern Web arc-hi-tecture [C] //Proc of the 22nd Int Conf on Software Engineering. New York: ACM, 2000: 407-416

[8] Xu Zhiwei, Peng Xiaohui, Zhang Lei, et al. The Φ-stack for smart Web of things [C] //Proc of SmartIoT'17. New York: ACM, 2017: Article No. 10

[9] Drytkiewicz W, Radusch I, Arbanowski S, et al. pREST: A REST-based protocol for pervasive systems [C] //Proc of 2004 IEEE Int Conf on Mobile Ad-hoc and Sensor Systems. Piscataway, NJ: IEEE, 2004: 340-348

[10] Caporuscio M, Funaro M, Ghezzi C, et al. Advanced Web Services: UbiREST-A Restful Service-Oriented Middleware for Ubiquitous Net-Working [M]. Berlin: Springer, 2014: 475-500

[11] Gavrin E, Lee S J, Ayrapetyan R, et al. Ultra lightweight JavaScript engine for Internet of things [C] //Proc of the 2015 ACM SIGPLAN Int Conf on Systems, Programming, Languages and Applications: Software for Humanity. New York: ACM, 2015: 19-20

[12] Xie Kaibin, Chen Haiming, Cui Li. PMDA: A physical model driven software architecture for Internet of things [J]. Journal of Computer Research and Development, 2013, 50 (6): 1185-1197 (in Chinese)
(谢开斌, 陈海明, 崔莉. PMDA:一种物理模型驱动的物联网软件体系结构[J]. 计算机研究与发展, 2013, 50(6): 1185-1197)

[13] Liu Peng, Lance H, Suman B. Lightweight multitenancy at the network's extreme edge [J]. Computer, 2017, 50(10): 50-57

[14] Dey S, Mukherjee A, Paul H S, et al. Challenges of using edge devices in IoT computation grids [C] //Proc of 2013 Int Conf on Parallel and Distributed Systems. Piscataway, NJ: IEEE, 2013: 564-569

[15] Bormann C, Ersue M, Keranen A. Terminology for constrained node networks, RFC7228 [R]. Fremont, CA: IETF, 2014

[16] Hahm O, Baccelli E, Petersen H, et al. Operating systems for low-end devices in the Internet of things: A survey [J]. IEEE Internet of Things Journal, 2016, 3(5): 720-734

[17] Atzori L, Iera A, Morabito G. The Internet of things: A survey [J]. Computer Networks, 2010, 54(15): 2787-2805

[18] Levis P, Madden S, Polastre J, et al. Ambient Intelligence: TinyOS: An operating system for sensor networks [M]. Berlin: Springer, 2005: 115-148

[19] Dunkels A, Grnvall B, Voigt T. Contiki-a lightweight and flexible operating system for tiny networked sensors [C] // Proc of the 38th Annual IEEE Conf on Local Computer Networks. Piscataway, NJ: IEEE, 2004: 455-462

[20] Sakamura K. The Objectives of the TRON Project: TRON Project 1987 Open-Architecture Computer Systems [M]. Berlin: Springer, 1987: 3-16

[21] Al-Fuqaha A, Guizani M, Mohammadi M, et al. Internet of things: A survey on enabling technologies, protocols, and applications [J]. IEEE Communications Surveys & Tutorials, 2015, 17(4): 2347-2376

[22] Heuer J, Hund J, Pfaff O. Toward the Web of things: Applying Web technologies to the physical world [J]. Computer, 2015, 48(5): 34-42

[23] Surhone L M, Timpledon M T, Marseken S F, et al. Z-Wave [J]. Internet of Things Key Applications & Protocols, 2010, 39(4): 620-622

[24] Gill K, Yang S H, Yao F, et al. A zigbee-based home automation system [J]. IEEE Trans on Consumer Electronics, 2009, 55(2): 422-430

[25] Gomez C, Oller J, Paradells J. Overview and evaluation of bluetooth low energy: An emerging low power wireless technology [J]. Sensors, 2012, 12(9): 11734-11753

[26] Kushalnagar N, Montenegro G, Schumacher C. IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals, RFC4919 [R]. Fremont, CA: IETF, 2007

[27] Bormann C, Castellani A P, Shelby Z. CoAP: An application protocol for billions of tiny Internet nodes [J]. IEEE Internet Computing, 2012, 16(2): 62-67

[28] Kamio M, Nakamura K, Kobayashi S, et al. Micro T-Kernel: A low power and small footprint RTOS for networked tiny devices [C] //Proc of the 6th Int Conf on Information Technology: New Generations. Piscataway, NJ: IEEE, 2009: 587-594

[29] ARM Limited. Mbed [EB/OL]. [2017-11-15]. https://www.mbed.com

[30] Ahlgren B, Hidell M, Ngai C H. Internet of things for smart cities: Interoperability and open data [J]. IEEE Internet Computing, 2016, 20(6): 52-56

[31] Shi Weisong, Cao Jie, Zhang Quan, et al. Edge computing: Vision and challenges [J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646

[32] Eugster P, Sundaram V, Zhang Xiangyu. Debugging the Internet of things: The case of wireless sensor networks [J]. IEEE Software, 2015, 32(1): 38-49

[33] Wilde E. Putting things to REST, 2007-015 [R/OL]. Berkeley: School of Information, University of California, Berkeley, 2007. [2017-11-15]. http://dret.net/netdret/docs/wilde-irep07-015-restful-things.pdf

[34] Guinard D, Trifa V, Mattern F, et al. From the Internet of things to the Web of things: Resource-oriented architecture and best practices [G] //Architecting the Internet of Things. Berlin: Springer, 2011: 97-129

[35] Luckenbach T, Gober P, Arbanowski S, et al. TinyREST: A protocol for integrating sensor networks into the Internet [C] //Proc of 2015 the Workshop on Real-World Wireless Sensor Networks. New York: ACM, 2005: 101-105

[36] Hou Chenda, Li Dong, Qiu Jiefa, et al. SeaHttp: A resource-oriented protocol to extend REST style for Web of things [J]. Journal of Computer Science and Technology, 2014, 29(2): 205-215

[37] Yashiro T, Kobayashi S, Koshizuka N, et al. An Internet of things (IoT) architecture for embedded appliances [C] //Proc of 2013 IEEE Region 10 Humanitarian Technology Conf. Piscataway, NJ: IEEE, 2013: 314-319

[38] Raza S, Shafagh H, Hewage K, et al. Lithe: Lightweight secure CoAP for the Internet of things [J]. IEEE Sensors Journal, 2013, 13(10): 3711-3720

[39] Konieczek B, Rethfeldt M, Golatowski F, et al. Real-time communication for the Internet of things using jCoAP [C] // Proc of the 18th IEEE Int Symp on Real-Time Distributed Computing. Piscataway, NJ: IEEE, 2015: 134-141

[40] Cirani S, Picone M, Veltri L. mjCoAP: An Open-source Lightweight Java CoAP Library for Internet of Things Applications [M]. Berlin: Springer, 2015: 118-133

[41] Tanganelli G, Vallati C, Mingozzi E. CoAPthon: Easy development of CoAP-based IoT applications with Python [C] //Proc of the 2nd IEEE World Forum on Internet of Things. Piscataway, NJ: IEEE, 2015: 63-68

[42] Stirbu V. Towards a RESTful plug and play experience in the Web of things [C] //Proc of the 6th IEEE Int Conf on Semantic Computing. Piscataway, NJ: IEEE, 2008: 512-517

[43] Kim S, Hong J Y, Kim S, et al. RESTful design and implementation of smart appliances for smart home [C] // Proc of IEEE UIC-ATC-SCALCOM'14. Piscataway, NJ: IEEE, 2015: 717-722

[44] Kovatsch M, Lanter M, Duquennoy S. Actinium: A RESTful runtime container for scriptable Internet of things applications [C] //Proc of the 3rd Int Conf on the Internet of Things. Piscataway, NJ: IEEE, 2012: 135-142

[45] Jaimini U, Dhaniwala M. JavaScript empowered Internet of things [C] //Proc of the 3rd Int Conf on Computing for Sustainable Global Development. Piscataway, NJ: IEEE, 2016: 2373-2377

[46] Ierusalimschy R, De Figueiredo L H, Celes Filho W. Lua-an extensible extension language [J]. Software Practice and Experience, 1996, 26(6): 635-652

[47] Yamamoto T, Oyama H, Azumi T. Lightweight ruby framework for improving embedded software efficiency [C] // Proc of the 4th IEEE Int Conf on Cyber-Physical Systems, Networks, and Applications. Piscataway, NJ: IEEE, 2016: 71-76

[48] Apache. IoT.js [EB/OL]. [2017-11-15]. http://iotjs.net

[49] The Hybrid Group. Cylon.js [EB/OL]. [2017-11-15]. https://cylonjs.com

[50] Tilkov S, Vinoski S. Node.js: Using JavaScript to build high-performance network programs [J]. IEEE Internet Computing, 2010, 14(6): 80-83

[51] Hemphill E. DeviceJS is Javascript for the physical world [EB/OL]. [2017-11-15]. http://devicejs.org/blog/post/devicejs-is-javascript-for-the-physical-world

[52] Nagumanthri A D, Tanaka K. Internet of things with mruby [C] //Proc of 2016 Int Conf on Information Technology. Piscataway, NJ: IEEE, 2016: 142-147

[53] Khamphroo M, Kwankeo N, Kaemarungsi K, et al. MicroPython-based educational mobile robot for computer coding learning [C] //Proc of 2017 Information and Communication Technology for Embedded Systems. Piscataway, NJ: IEEE, 2017: 131-136

[54] Software in the Public Interest Inc. The computer language benchmarks game [EB/OL]. [2017-11-15]. http://benchmarksgame.alioth.debian.org

[55] Wikipedia. Cross compiler [EB/OL]. [2017-11-15]. https://en.wikipedia.org/wiki/Cross_compiler

[56] MacNamee C, Heffernan D. Emerging on-ship debugging techniques for real-time embedded systems [J]. Computing & Control Engineering Journal, 2000, 11(6): 295-303

[57] Rössler P, Höller R. A novel debug solution for distributed embedded applications and implementation options [C] //Proc of the 37th IEEE Annual Conf on Industrial Electronics Society. Piscataway, NJ: IEEE, 2011: 2796-2801

[58] Wang J R, Yu C-H. ROM emulator: USA, US20060224377 [P/OL]. 2006-10-15. [2017-11-15]. http://www.freepatentsonline.com/20060224377.pdf

[59] Chikao Uchino. In-circuit emulator system: USA, US20040078671A1 [P/OL]. 2004-04-22. [2017-11-15]. http://www.freepatentsonline.com/20040078671.pdf

[60] Sutter E. Embedded Systems Firmware Demystified [M]. Boca Raton, FL: CRC Press, 2002: 247-249

[61] Gatliff B. Embedding with gnu: The gdb remote serial protocol [J]. Embedded Systems Programming, 1999, 12(11): 108-113

[62] Ibrahim D. Microcontroller debugging and testing tools [J]. Electronics World, 2009, 115: 20-23

[63] Huang Hongyan. Analysis and design of debugging technology in embedded system [D]. Hangzhou: Zhejiang University, 2016 (in Chinese)
(黄红燕. 嵌入式系统调试技术的分析与设计[D]. 杭州: 浙江大学, 2016)

[64] Logt L V D, Heyden F V D, Waayers T. An extension to JTAG for at-speed debug on a system [C] //Proc of IEEE Int Test Conf (ITC). Piscataway, NJ: IEEE, 2003: 123-130

[65] Kra Y. A cross-debugging method for hardware/software co-design environments [C] //Proc of the 30th Int Design Automation Conf. New York: ACM, 1993: 673-677

[66] Lee Y-H, Song Y W, Girme R, et al. Replay debugging for multi-threaded embedded software [C] //Proc of the 8th IEEE/IFIP Int Conf on Embedded and Ubiquitous Computing. Piscataway, NJ: IEEE, 2010: 15-22

[67] Dunlap G W, King S T, Cinar S, et al. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay [J]. ACM SIGOPS Operating Systems Review, 2002, 36 (SI): 211-224

[68] Honarmand N. Record and deterministic replay of parallel programs on multiprocessors [D]. Urbana-Champaign, IL: University of Illinois at Urbana-Champaign, 2015

[69] Luo Yan. Research and primitive implementation of backtracking debugging based on process snapshotting in kernel mode [D]. Hangzhou: Zhejiang University, 2008 (in Chinese)
(罗琰. 基于内核模式下进程快照的可回溯调试研究及初步实现[D]. 杭州: 浙江大学, 2008)

[70] Jiang Shan. Research on reversible debugging on embedded multicore [D]. Hangzhou: Zhejiang University, 2016 (in Chinese)
(江山. 嵌入式多核架构可逆调试技术研究[D]. 杭州: 浙江大学, 2016)

[71] Wu Mingqi, Ma Chao. A software-hardware cooperative remote debugging method for embedded system [J]. Microcontrollers and Embedded Systems, 2005(7): 15-16 (in Chinese)
(吴明琪, 马潮. 一种软硬结合的嵌入式系统远程调试方法[J]. 单片机与嵌入式系统应用, 2005(7): 15-16)

[72] Salman O, Elhajj I, Kayssi A, et al. Edge computing enabling the Internet of things [C] //Proc of the 2nd IEEE World Forum on Internet of Things (WF-IoT). Piscataway, NJ: IEEE, 2015: 603-608

[73] Asano S, Yashiro T, Sakamura K. Device collaboration framework in IoT-aggregator for realizing smart environment [C] //Proc of 2016 TRON Symposium. Piscataway, NJ: IEEE, 2016

[74] Koshizuka N, Sakamura K. Ubiquitous ID: Standards for ubiquitous computing and the Internet of things [J]. IEEE Pervasive Computing, 2010, 9(4): 98-101

[75] Choi N, Kim D, Lee S J, et al. A fog operating system for user-oriented IoT services: Challenges and research directions [J]. IEEE Communications Magazine, 2017, 55 (8): 44-51

[76] Cao Jie, Xu Lanyu, Abdallah R, et al. EdgeOS_H: A home operating system for Internet of everything [C] //Proc of the 37th IEEE Int Conf on Distributed Computing Systems. Piscataway, NJ: IEEE, 2017: 1756-1764

[77] Song Mingcong, Hu Yang, Chen Huixiang, et al. Towards pervasive and user satisfactory CNN across GPU microarchitectures [C] //Proc of 2017 IEEE Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2017: 1-12

[78] Haas A, Rossberg A, Schuff D L, et al. Bringing the Web up to speed with WebAssembly [C] //Proc of the 38th ACM Sigplan Conf on Programming Language Design and Implementation. New York: ACM, 2017: 185-200

[79] Shi Hailong, Li Dong, Qiu Jiefan, et al. EasiSHA: A reconfigurable node architecture for IoT based on joint design of software and hardware [J]. Journal of Computer Research and Development, 2014, 51(5): 959-973 (in Chinese)
(石海龙, 李栋, 邱杰凡, 等. EasiSHA: 一种软硬件协同的物联网可重塑终端架构[J]. 计算机研究与发展, 2014, 51 (5): 959-973)

Peng Xiaohui, born in 1984. Assistant professor. Member of CCF. His main research interests include the architecture and computing models of things computing system.

Zhang Xingzhou, born in 1992. PhD candidate. Student member of CCF. His main research interests include things computing system and artificial intelligence.

Wang Yifan, born in 1992. PhD candidate. Student member of CCF. His main research interest is things computing system.

Chao Lu, born in 1989. PhD candidate. Student member of CCF. His main research interests include things computing system and distributed data computing framework.