中国科学: 信息科学 2019 年 第 49 卷 第 9 期: 1119–1137
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
评述

云计算应用服务开发环境: 从代码逻辑到数据流图

刘云浩, 杨启凡, 李振华*

清华大学软件学院, 北京 100084
* 通信作者. E-mail: lizhenhua1983@tsinghua.edu.cn

收稿日期: 2018–10–12; 接受日期: 2019–01–18; 网络出版日期: 2019–09–03

国家高技术研究发展计划 (863) (批准号: 2015AA01A201) 资助项目

摘要 云计算应用服务具备优良的弹性与扩展性, 但其开发则具备较高的难度和复杂性, 要求开发者熟练掌握并运用各项云计算技术, 搭建并维护网络化信息系统. 在国民经济各行各业对云计算应用服务日益旺盛的需求驱动下, 近年来出现了众多支持或辅助云计算应用服务完整或部分生命周期的开发环境. 本文实践调研了多个主流云计算应用服务开发环境, 根据自动化程度的高低将它们分为 3 个类别: 用户自主搭建开发环境、社区自发维护标准构件及托管开发环境, 其中第 3 类进一步分为 4 代: 编辑型、应用级、函数级与集成型. 通过典型案例的深入挖掘, 梳理出云计算应用服务开发环境的 4 个主要发展趋势: 虚拟化、轻量化、智能化与可视化, 并基于我们研发的 Cloud Studio 开发环境探索其可能的具体形态与展现形式, 特别是开发理念从微观层面的代码逻辑转变到宏观层面的数据流图, 期望能够有效降低开发门槛, 加速开发效率, 促进 “中国云” 核心竞争力的形成.

关键词 云计算, 开发环境, 软件工程, 数据流图, 应用开发

1 引言

作为 21 世纪初蓬勃发展的新型计算模式, 云计算 [1] 因其对信息基础设施的集中管理与租售计算资源的规模效应, 为各类计算机应用服务的运行提供了灵活方便、经济可靠的承载平台 [2]. 运行于云计算平台 (如亚马逊 AWS、微软 Azure、谷歌云、阿里云等) 的应用服务通常能够具备优良的弹性与扩展性, 但其开发则具备较高的难度和复杂性, 要求开发者熟练掌握并运用各项云计算技术, 搭建并维护网络化信息系统 [3]. 具体来说, 云计算是利用互联网实现随时随地、按需、便捷地访问共享资源池的计算模式, 而云计算应用服务则是托管在云计算上的具体应用和服务. 为了开发一个普通的云计算应用服务, 开发者通常需要掌握操作系统 (尤其是 Linux)、虚拟化 (特别是虚拟机和容器技术)、对象存储、数据库、分布式系统、网络协议、Web 展现、信息安全和特定编程语言等多项技术, 几乎跨越整个计算机协议栈 [4].

引用格式: 刘云浩, 杨启凡, 李振华. 云计算应用服务开发环境: 从代码逻辑到数据流图. 中国科学: 信息科学, 2019, 49: 1119–1137, doi: 10.1360/N112018-00264
Liu Y H, Yang Q F, Li Z H. Cloud computing development environment: from code logic to dataflow diagram (in Chinese). Sci Sin Inform, 2019, 49: 1119–1137, doi: 10.1360/N112018-00264

为了开发云计算应用服务, 要么需要寻找到稀缺而合格的 “全栈工程师” [5]、要么需要组建起一支完备而互补的默契开发团队, 但两者实际上都十分不易且代价不菲, 尤其是对中小型企业或创业公司来说几乎是难以逾越的人力资源门槛 [6, 7]. 在国民经济各行各业对云计算应用服务日益旺盛的需求驱动下, 为了解决 “云计算系统难搭、云计算人才难求” 的产业困境, 近年来出现了众多支持或辅助云计算应用服务完整或部分生命周期的开发环境, 给云计算应用服务的开发维护带来了一定程度的帮助和改善 (特别是简化编程、降低难度和提升效率), 也促进了云计算模式向全社会的推广和普及.

本文实践调研了国际国内 50 个主流云计算应用服务开发环境 (简称 “云计算开发环境”), 包括亚马逊公司的 AWS Cloud9 [8]、谷歌的 App Engine [9]、阿里云的函数计算平台 [10]、华为的 DevCloud [11]、Coding 公司的 WebIDE [12] 等. 基于广泛调研和深度思考, 我们选择自动化程度作为这些主流云计算应用服务开发环境的核心对比指标, 从而根据自动化程度的高低将它们分为 3 个类别: (1) 用户自主搭建开发环境, (2) 社区自发维护标准构件, (3) 云计算托管开发环境. 其中第 3 类 (托管开发环境) 自动化程度最高且辅助性最强, 我们根据其功能性进一步将它们分为 4 代: (1) 编辑型托管开发环境, (2) 应用级托管开发环境, (3) 函数级托管开发环境, (4) 集成型托管开发环境. 与此 4 代并列, 还存在多个面向特定领域的托管开发环境.

通过对一系列典型案例的横向对照和纵向挖掘, 我们梳理出云计算开发环境当前的 4 个主要发展趋势: (1) 虚拟化, (2) 轻量化, (3) 智能化与 (4) 可视化. 为了探索上述趋势可能的具体形态与展现形式, 我们历时三年多, 研发出 (清华大学) Cloud Studio 图形化集成云计算开发环境 [13]. 我们设计了 (a) 基于事件流图的 Web 构件可视化开发, (b) 封装函数间胶水代码的简化接口, (c) 基于互动令牌机制的代码编辑弱隔离, 以及 (d) 应用适配的容器/虚拟机运行时隔离等关键技术, 实现云计算应用服务的开发理念从微观层面的代码逻辑到宏观层面的数据流图的本质转变, 期望能够有效降低开发门槛, 加速开发效率, 促进 “中国云” 核心竞争力的形成, 更好地支撑我国信息产业和现代服务业建设.

正文部分组织结构如下: 第 2 节, 调研国际国内主流云计算开发环境, 归纳出每一类开发环境的主要技术特征, 并进一步探讨第 3 类 (云计算托管开发环境) 每一代的典型案例. 基于调研结果, 在第 3 节提炼出云计算开发环境的主要发展趋势, 并在第 4 节中介绍我们研发的 Cloud Studio 云计算开发环境. 第 5 节总结全文和展望未来.

2 主流云计算应用服务开发环境调研

从云计算产业大趋势看, 近年来云计算 “刚性” 技术迅猛发展, 工业界已经部署了千万量级的服务器, 支持数十亿量级的终端用户设备 [14]. 与此同时我们也注意到, 云计算 “柔性” 开发环境方面的深度探索刚刚起步, 成为关键的产业瓶颈 [5]. 几乎所有的云计算产业巨头最近几年都推出了自己的开发环境, 也有相当多的新兴创业公司以云计算开发环境作为自身核心业务; 我们将初步的调研结果列在表 1 中. 在实践调研中我们观察到: 云计算开发环境的演变进化过程表现出十分明显的自动化程度从低到高的趋势. 首先, 开发流程覆盖程度不断提高, 从工程团队内部约定、手工流程或是脚本化流程, 到标准的构件模块, 再到覆盖关键流程、特定领域流程的针对性系统, 最终发展到覆盖开发与运维全流程的集成化系统. 同时, 用户操作和监控界面变得越发友好和便捷, 从只能命令行操作, 到提供开发客户端与编程接口, 再到提供 Web 管理与托管服务界面, 最终发展到图形化集成开发环境. 下面首先分 3 类具体介绍现存的主流云计算开发环境, 再分析传统软件工程方法在服务云计算开发时遇到的挑战, 从而对当前云计算开发所面临的挑战与现存的解决方案形成一个较为全面的认知.

表 1 主流云计算应用服务开发环境

Table 1 Mainstream cloud computing development environments

	Classification	Systems (sorted mostly by popularity)
Home-brewed development environments		LAMP [15]
Managed development environments	Spontaneously maintained standard components by communities	Google Kubernetes, Docker, OpenStack, Hadoop, Spark, Gitlab [16], Mesos [17], Grafana [18], Prometheus [19], Jenkins [20], Jaeger [21]
	Editor-style development environments	CodeAnywhere Cloud IDE [22], Jupyter Notebook [23], Coding WebIDE [12], ShiftEdit [24], NeutronDrive [25]
	Application-level development environments	Google App Engine (GAE) [9], AWS Elastic Beanstalk [26], Heroku [27], Google Firebase [28], Red Hat OpenShift [29], Huawei DevCloud [11], Codio [30], Sina App Engine (SAE) [31], Baidu App Engine (BAE) [32], Leancloud [33], AppScale [34]
	Function-level development environments	AWS Lambda [35], Google Cloud Functions [36], Microsoft Azure Functions [37], IBM OpenWhisk [38], Aliyun Function Compute [10], Tencent Cloud Function [39], Auth0 WebTasks [40], Kubeless [41], Fission [42], Spotinst [43]
	Integrated development environments	AWS Cloud9 [8], Tsinghua Cloud Studio [13]
	Domain-specific development environments	Google AutoML Vision [44], IBM Watson IoT [45], 4 Paradigm Prophet [46], Aliyun PAI [47], Amazon Sagemaker [48], JD AI NeuHub [49], Microsoft Azure Bot Service [50], WeChat IOT service [51], Baidu IOT platform [52], Ubidots [53]

2.1 第 1 类: 用户自主搭建开发环境

十年前, 云计算模式刚刚兴起, 应用服务的开发基本上依靠用户自主搭建开发环境, 典型代表是 LAMP, 即 Linux 操作系统 + Apache Web 服务器 + MySQL 数据库 + PHP/Python 网页. 具体地, 开发者需要手工维护的开发流程与环境通常包括: (1) 搭建系统管理开发流程, 如需求管理、系统设计等; (2) 搭建依赖服务, 如数据库、缓存、消息队列、对象存储、块存储、数据仓库等; (3) 手动管理软件依赖; (4) 编写与构建云计算应用服务; (5) 系统测试与集成; (6) 部署与持续迭代.

开发者自行维护一个应用服务的整个生命周期, 对应用服务能够获得最深的理解和最好的掌控, 但是工作量庞大而繁杂、开发难度大、持续时间长, 且技术门槛高. 云计算应用服务的开发本身就比单机应用的开发环节更多更易出错, 加上缺乏成熟开发环境 (单机应用开发能够使用十分成熟的开发环境, 诸如微软 Visual Studio、苹果 Xcode、Apache Eclipse 等) 的支持, 工程师大量的宝贵时间耗费在底层细节和调试, 反而容易忽略高层的架构和设计.

2.2 第 2 类: 社区自发维护标准构件

过去十年, 很多云计算公司及从业人员在长期的开发实践中, 摸索出大量软件复用的设计方法和实现原理, 从而形成一系列开放的云计算社区, 自发维护各种各样的标准构件供其他开发者使用 [54]—— 各种概念、组件、框架如雨后春笋般涌现, 有效降低了应用服务的开发工作量和开发难度, 缩短了开发周期. 标准构件的形成在造福社区的同时, 也给贡献者带来了实际的经济效益, 比如谷歌 Kubernetes

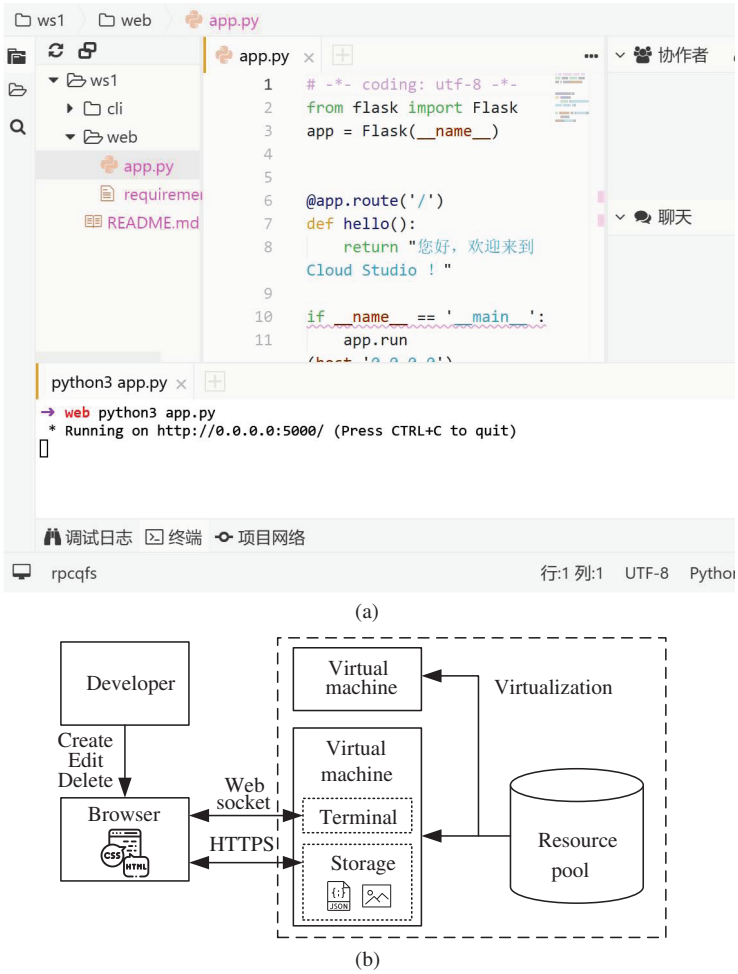

图 1 (网络版彩图) Coding 公司 WebIDE 编辑型托管开发环境. (a) 使用界面截图; (b) 基本架构图

Figure 1 (Color online) Coding WebIDE, an editor-level development environment. (a) Screenshot; (b) basic architecture

集群管理工具在开源社区十分流行, 在企业的应用也非常广泛 [55]; Spark 内存计算平台从学术界走向工业界 [56,57], 所孵化的 Databricks 公司市值已高达几十亿美元; Gitlab 提供免费的公有云开发环境, 对私有云开发环境部署收费.

标准构件通常能够解决云计算应用服务开发或运维过程中某一环节的局部难点和复杂性, 但远远不是全部, 并且存在明显的无组织性和异构性. 它们相当于应用服务开发的 "砖头", 要把砖头砌成 "房子", 开发者还需要做大量的连接、粘贴、整合和配置等 "外围" 工作, 工作量依然不小, 要求依然很高.

2.3 第 3 类: 云计算托管开发环境

为了减少甚至避免上文所说的这些 "外围" 工作, 近年来市场上出现了多家云计算托管开发环境, 按照时间先后及托管程度的高低, 我们进一步将它们分为如下四代.

第 1 代: 编辑型托管开发环境. 以 Coding 公司的 WebIDE 为代表, 如图 1 所示, 其布局与常见的本地代码编辑器相仿: 左侧为项目文件树, 中间为代码编辑区, 下方为终端控制台、日志等调试多功能区, 右侧为在线协作和交流功能区. 它提供云计算应用服务的 Web 在线代码编辑功能, 而应用服

务的编译和运行还是需要程序员通过 Web SSH Shell 终端手工输入命令来执行. Web 编辑器和 Web Shell 终端的结合使用, 是以最为原始的方式满足云计算应用服务开发者们的基本开发需求, 如代码编辑、版本管理、服务器管理和项目部署等, 因此通常用来开发简单的 Web 应用 [58].

编辑型托管开发环境是云计算服务提供商们对云计算开发环境的初期探索, 是以下 3 项底层关键技术结合的产物: 虚拟化、新型 Web 开发语言、新型 Web 通信协议. 首先, 虚拟化技术允许编辑型托管开发环境在有限的物理机上实现多租户共享的云端开发模式, 从而有效降低了运营成本. 其次, 新一代 HTML 标准的推出和 JavaScript 与 CSS 的应用, 允许 Web 开发者设计和实现功能丰富、互动性高的 Web 应用, 这使得带有简单代码高亮、自动缩进、自动补全功能的 Web 编辑器成为可能. 此外, 借助 Web 文件差分同步 [59] 等技术, 编辑型托管开发环境可以将开发者对代码文件的改动在用户浏览器和云端服务器之间快速同步. 最后, 在浏览器上实现 Web Shell 的前端逻辑, 需要将用户在浏览器上的输入流实时地转发到服务器中的虚拟终端, 并将虚拟终端的输出和错误流实时地转发到用户浏览器. 更细节地说, 实现 Web Shell 与 Web 文件编辑器的不同之处在于, Web Shell 要求极低的延时和有状态的长时间连接, 而传统 HTTP 协议主要针对无状态、无连接的场景设计 [60], 并不能很好地服务有状态长连接的应用. 因此, 新一代 WebSocket 协议被采用以支持服务器和客户端之间的长时间全双工通信, 依托相关的 API (应用编程接口), 开发者能够并不困难地实现高效的 Web Shell 输入输出流 [61].

编辑型托管开发环境的主要优点在于实现难度低: 借助于新型的 Web 技术, 高效的 Web 编辑器与 Web Shell 是容易实现的, 并且已经存在较多的社区开源组件可供使用. Web Shell 与 Web 编辑器的组合一般能够满足常用的最简开发需求, 因此编辑型托管开发环境已十分流行, 其所使用的技术也被拓展应用在了如数据库 Web 管理环境等其他专用开发环境上.

编辑型托管开发环境的主要缺点是受制于底层云平台的支持能力. 一般来说, 其托管粒度是比较粗放和模糊的, 应用对于托管环境基本上是一个黑盒的存在, 大多数生命阶段完全由开发者手工管理, 导致开发环境无法做到更好的应用管理与调度优化. 另外, 其界面友好性比较低, 操作门槛比较高, 基本相当于古老的 SSH 远程终端开发, 还没有发展出类似真正 IDE (集成开发环境) 的高阶编辑、提示和调试能力, 如智能联想、断点调试等.

第 2 代: 应用级托管开发环境. 以谷歌 App Engine (GAE) 为代表, 如图 2(a) 所示, 它是一个开发、托管云计算应用的平台, 可让开发者在谷歌的基础架构上运行其云计算应用, 而谷歌底层的基础设施则对开发者是透明的 [62]. 更进一步, 华为 DevCloud 致力于为云计算应用服务打造标准化的构建、部署与发布流水线, 提供全生命周期的流程托管支持, 用户可以通过 Web 编辑器接入, 也可以直接上传代码, 而系统则通常以容器的形式打包用户的应用后再部署上线.

托管的粒度细化到应用级后, 相对统一的应用运行时和标准化的生命周期为应用级托管开发环境带来了进一步优化的可能. 与此同时, 用于编辑型托管开发环境的 Web 编辑器、Web Shell 和后端多租户虚拟机的架构已不再适用, 而需要依据应用级的托管粒度设计新的系统架构. 如图 2(b) 所示, 谷歌 App Engine 针对 Python 应用的基本架构可以分为两部分. 第 1 部分是用户的 Python 应用实例, 也是托管的主体, 由 Python 虚拟机进程和一个只读文件系统组成. Python 虚拟机运行在相互独立、权限受限的沙箱进程中 [63], 没有虚拟化的开销. 只读文件系统既包括 Python 的系统库和第三方库, 也包含用户的应用代码; 只读性保证了托管应用的功能不会被随意篡改. 第 2 部分由谷歌提供的一系列托管服务和无状态编程 API 构成. 托管服务如数据存储、内存缓存等; 无状态编程 API 则是将特权的和公用的编程 API 抽离出用户应用形成的集合, 如访问外部 URL、邮件管理和图片处理.

应用级托管开发环境的出现和普及标志着 PaaS (Platform-as-a-Service) 在云计算应用服务开发环

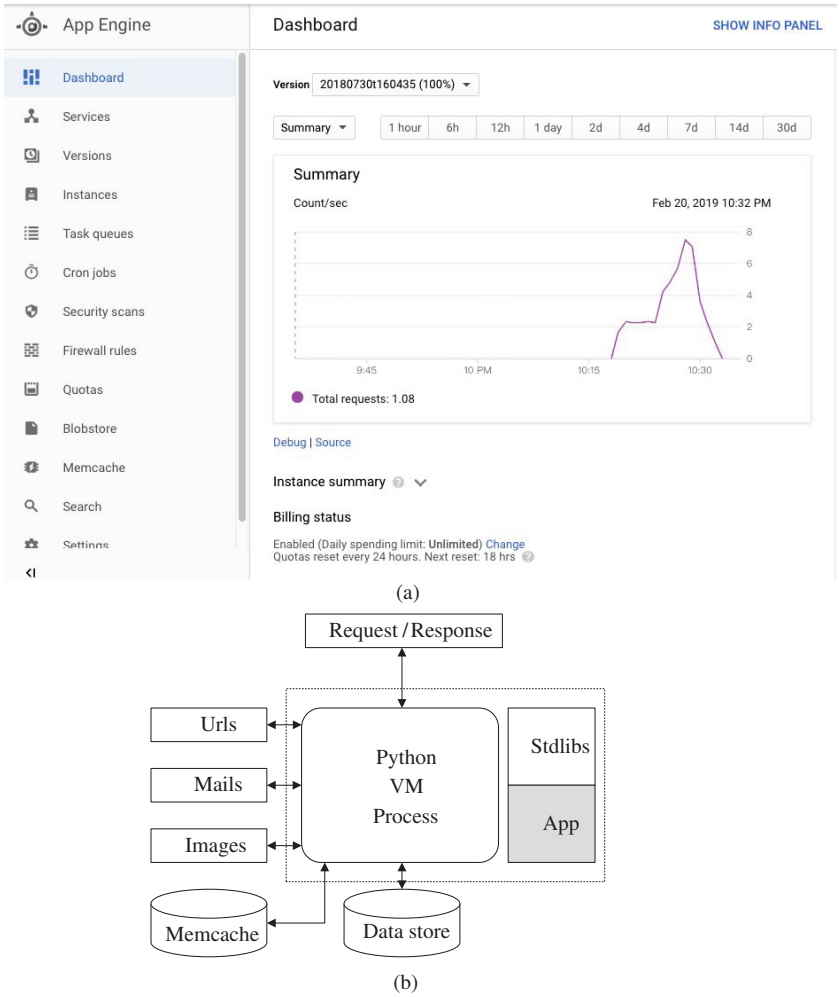

图 2 (网络版彩图) 谷歌 App Engine (GAE) 应用级托管开发环境. (a) 使用界面截图; (b) 基本架构图

Figure 2 (Color online) Google App Engine (GAE), an application-level development environment. (a) Screenshot; (b) basic architecture

境方向上的落地和成熟, 立足于对云计算应用生命周期的宏观建模和整体把握. 以谷歌的 Python App Engine 为例, 它是以下 3 项底层关键技术结合的产物: 定制的应用运行时、基础设施的无状态化服务、在线源代码管理. 首先, 定制化 Python 虚拟机配合使用只读文件系统, 能够在公有云环境下实现更好的资源隔离和权限管控. 其次, 将谷歌内部的基础设施以无状态的外部服务的形式暴露给应用来使用, 一定程度上保障了公有云上应用托管的安全性, 同时降低了云计算应用的编程难度. 最后, 应用级托管开发环境通常都提供用户应用代码的在线版本管理, 其中包含上传和简易的在线编辑功能; 同时提供自动化部署功能, 将用户应用及时更新上线.

应用级托管开发环境的优点主要包括 3 个方面. 首先, 将云计算应用的生命周期拆解开并标准化, 使用统一的框架进行调度执行, 向开发者提供了比较便捷的部署运维管理系统; 相比编辑型托管开发环境, 是理念和工程上的一大进步. 其次, 应用级托管所提供的外部服务组件仅暴露无状态的编程接口而屏蔽底层的分布式细节, 节省了开发者手动部署并维护其他服务的开销. 最后, 应用级托管已经可以实现资源的弹性调度、多租户的安全隔离、粗粒度的实时监控等基本需求, 其实现方式对开发者

透明, 又通常以 Web 管理界面友好地呈现给开发者, 从而极大地缓解了应用部署和维护的后顾之忧.

应用级托管开发环境也存在两个方面的缺点. 首先, 用户应用依赖于大量的外部服务, 如数据库、内存缓存、对象存储等, 仍然需要手工胶合数量众多的 RESTful 或 RPC 接口 [64]; 同时, 还有大量的通用逻辑需要由开发者重复实现, 如过滤 HTTP 包、维护定时任务、维护消息队列、打点内部监控状态等. 另外, 虽然部署与运维的生命周期在应用级托管开发环境得到了重视, 但开发过程依然被轻视 —— 开发者通常在本地调试后才通过版本控制系统上传应用到托管开发环境进行最终测试. 这是由于用户托管的是一个完整的应用, 使用简易的在线编辑器的开发和调试效率并没有显著超越本地调试效率.

第 3 代: 函数级托管开发环境. 以亚马逊 Lambda、阿里云函数计算为代表, 其核心思想是实现 "无服务器计算" [65,66]: 在应用的开发和运维阶段, 开发者就应无需管理服务器等基础设施, 而专注于应用创新. 在这一思想指导下, 云计算应用被解耦为一组细粒度的函数, 作为开发和运维的基本单位. 由函数级托管开发环境支持函数之间的事件触发执行, 并维护函数调用的上下文. 开发者无需手动打包应用, 函数片段随着函数调用上下文被自动送入特定语言环境容器或虚拟机中, 即可执行并产出结果.

如图 3(a) 所示, 依托阿里云函数计算平台, 开发者可以在该界面编辑或上传托管的函数、设定函数的触发器、查看函数执行结果并查询运行日志. 图 3(b) 进一步描绘了阿里云函数计算平台的基本架构. 一个针对某函数的调用请求会首先被负载均衡到特定 API 服务上, 之后的操作分同步和异步两种类型. 如果是同步调用, 首先由该 API 服务通过表格存储获取该函数的元数据, 并通过元数据从对象存储中获取函数片段代码, 再获取计算资源的调度结果, 最后依据调度结果, 同步地调用函数执行引擎执行该函数. 如果是异步调用, API 服务将对该函数的调用写入消息队列, 接着等事件分发器异步地读取到该触发事件后, 再由分发器调用函数执行引擎执行该函数. 函数执行引擎运行在相互隔离的网络环境中, 从而避免多租户环境下可能产生的恶意攻击.

函数级托管开发环境的优点主要包括 4 个方面. 首先, 无状态函数的设定决定了用户的云计算应用容易被托管、负载均衡和弹性伸缩. 其次, 用户基本摆脱了服务器等底层基础设施的管理和维护负担, 可以将精力和资源集中在应用业务创新上 [67]. 再次, 事件触发机制的引入在轻松对接多种应用场景的同时, 降低了开发者的编程复杂度. 最后, 用户只需要为实际使用的资源付费, 如目前函数级托管开发平台通常按照函数的调用次数和公网流量费用综合计费, 计费标准可低至 1.33 元/百万次, 这对初创企业来说是成本容易把控且经济实惠的方案.

函数级托管开发环境也存在 3 个方面的缺点. 首先, 虽然它进一步优化了云计算应用的编写方式, 但应用编写时的可视、集成、调试等周边工具链仍然缺乏. 其次, 使用传统方式编写的云计算应用很难直接迁移到函数级托管开发平台, 原因在于未使用典型框架的云计算应用通常涉及到一组有状态函数的复杂依赖, 目前还难以由程序自动而准确地将它们解耦成无状态函数, 而通常依靠人工重构完成迁移. 最后, 函数级托管开发平台还处于起步阶段, 各云计算巨头虽然都推出了自家的相关开发环境, 但现阶段标准差异很大, 用户面临由标准不通、平台绑定造成的迁移难题 [68].

第 4 代: 集成型托管开发环境. 也是目前最先进的云计算托管开发环境. 典型代表是亚马逊 Cloud9, 如图 4 所示, 其大致界面布局与 Coding WebIDE 相似, 不同的是它显式地在右侧功能区加入了更多专业的云计算应用服务开发辅助功能, 特别是支持 Web 断点调试、变量查看和应用预览等高级功能 [69]. 这些高级功能的加入提升了开发效率, 补齐了在云端开发应用服务相对本地开发的关键短板. 作为云计算提供商对开发者集中展示的平台, 集成型托管开发环境实现了从应用到微服务再到函数计算的多粒度开发支持, 覆盖部署运维的全生命周期.

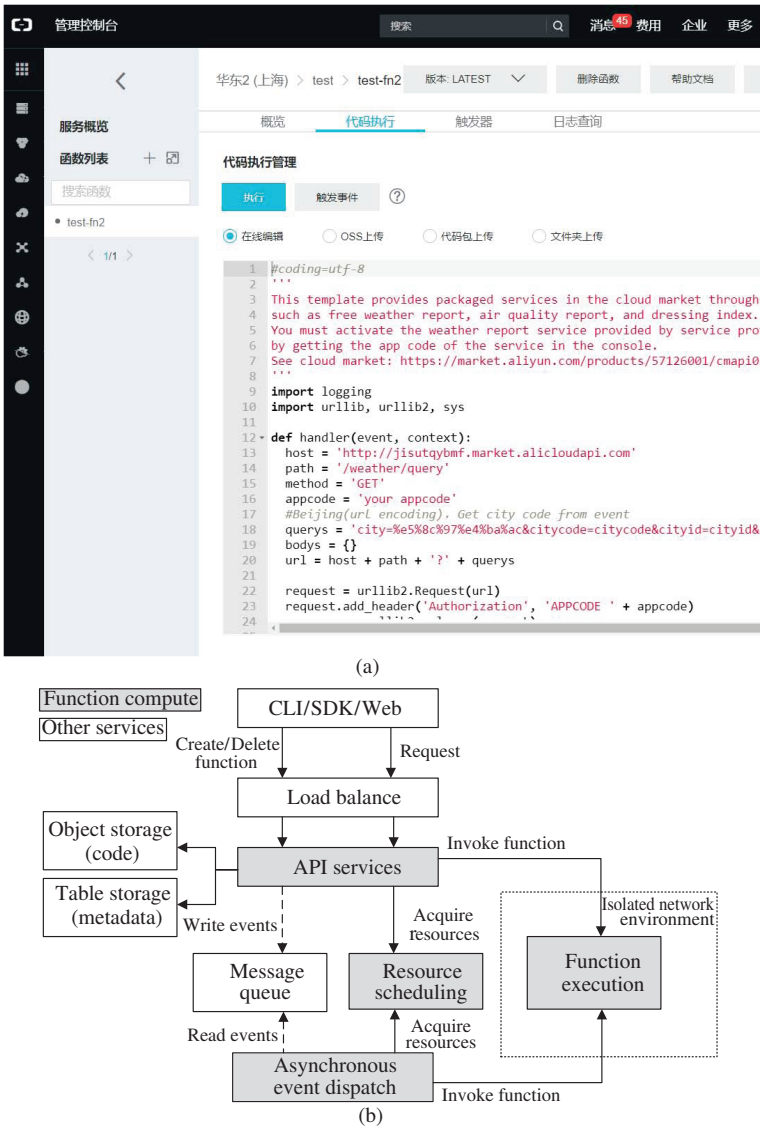

图 3　(网络版彩图) 阿里云函数计算平台. (a) 使用界面截图; (b) 基本架构图

Figure 3　(Color online) Aliyun function compute service. (a) Screenshot; (b) basic architecture

亚马逊的 Lambda 函数[35] 能够直接在 Cloud9 开发环境中调试. Cloud9 在调试时, 首先将 Lambda 的运行环境无缝切换到用户在云端独占的虚拟机中, 此时用户与 Cloud9 的前端连接, 通过本地的 API 网关调用并调试目标 Lambda 函数. 函数的调试信息, 如调用栈、变量值和断点设置都可以从独占的虚拟机中获取到. 上述过程对用户是透明的, 用户所感受到的只是 Lambda 函数可以被云端调试.

集成型托管开发环境具备 4 个方面的优点. 首先, 它提供更加丰富的代码编写时辅助功能, 如更加智能的提示、高亮和多人协作等. 其次, 它集成了应用级和函数级托管开发环境的的优点, 实现了多粒度的开发环境支持. 再次, 它在 Web 前端界面提供了诸多实用的调试功能和应用预览功能, 能有效加速用户的应用开发过程; 针对后端逻辑, 它提供了对远程函数的断点调试功能. 最后, 它与平台提供方的其他解决方案, 如大数据分析、网络安全工具链、机器学习技术栈做了深度整合, 从而方便用户开发功能更加丰富、更加健壮的云端应用.

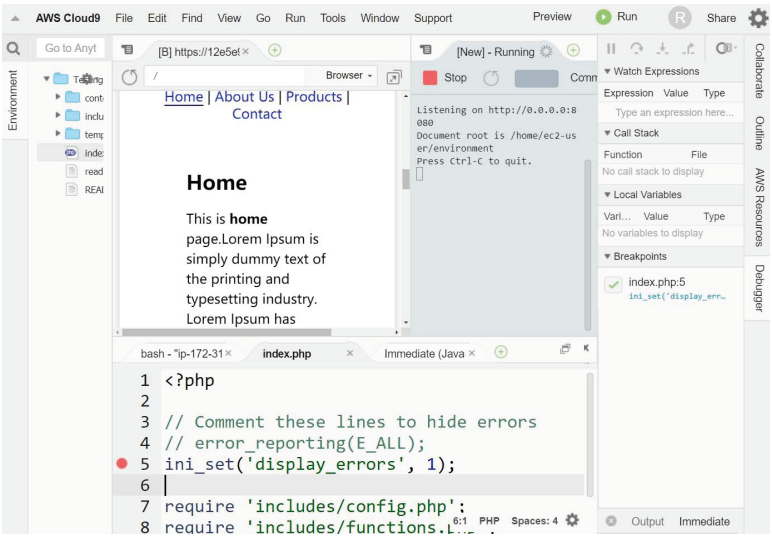

图 4 (网络版彩图) 亚马逊 Cloud9 集成型托管开发环境 (使用截图)

Figure 4 (Color online) Amazon Cloud9, an integrated development environment (screenshot)

集成型托管开发环境通常是一家云计算厂商 (比如亚马逊公司) 综合实力的体现, 开发环境的能力与自身其他产品线的能力息息相关, 这导致集成型托管开发环境的研发门槛极高, 并且不同厂商之间产品的兼容性很低, 常常需要使用者自行权衡和抉择. 另外, 集成型托管开发环境现阶段的知名度和普及度还有待提高, 与开发者也需要有一个磨合适应的长期过程.

作为主线外的重要分支, 近五年还出现了多个面向特定领域的托管开发环境, 主要针对大数据、人工智能、物联网等领域的特殊开发需求, 比如谷歌 Cloud AutoML Vision 平台提供的有监督图像分类开发引擎, 以及 IBM 沃森物联网应用开发平台.

2.4 传统软件开发环境面临的挑战

经典的单机软件开发环境历史悠久并发展成熟, 然而并不能很好地直接适用于云计算应用服务的开发. 以最主流的 3 个单机开发环境为例, 分别是微软的 Visual Studio、苹果的 Xcode 和开源的 Eclipse, 它们在如下 4 个方面很难适应云计算开发的需求. 首先, 云计算开发环境与互联网紧密耦合, 大多数情况下需要联网开发 [70, 71], 而经典开发环境与互联网松散耦合, 一般脱机开发即可. 其次, 云计算应用通常涉及大规模分布式环境和虚拟资源, 而经典开发环境一般运行在单机环境, 直接管理物理硬件资源. 再次, 在系统设计和代码编写方面, 对云计算应用的开发来说, 整体架构设计先于局部功能优化、开发接口高于逻辑流程、系统配置难于代码编写 [72]; 反过来, 普通应用的开发几乎不涉及复杂的系统设计, 主要难度在于代码优化. 最后, 云计算应用开发按需计费, 整个开发过程对费用很敏感 [73], 而经典开发环境几乎不需要考虑应用服务按使用量计费的问题.

用户在传统软件开发过程中可以使用统一建模语言 UML 比较好地刻画一个大型 (面向对象) 软件的组织结构和逻辑模型 [74], 结合这一需要, 文献 [75, 76] 讨论了基于 UML 的集成化软件开发环境的设计思想及其应具备的主要功能, 并介绍了基于该思想开发的可视化建模工具 UML/Designer. Mei 等 [77] 针对以构件为基本单元的软件体系的流行, 归纳出要将软件体系结构引入到软件开发的各个阶段作为系统开发的蓝图, 并进而提出通过自动转换机制使用工具缩小从高层设计到实现的距离, 在构件平台的运行支持下实现自动化的系统组装与生成. Huang 等 [78] 针对 Web 应用开发的兴起, 提

出了使用 Web 搭建分布式仿真开发环境的想法以及组件对象在 Web 环境下的开发和使用方法. 文献 [79~81] 分别针对语义网、基于构件的系统、Web 服务编排, 设计了不同的集成式软件开发环境. Bragdon 等 [82] 针对集成式开发环境基于文件的传统编辑方式, 提出了以 “代码气泡” 为中心的开发环境新交互形式. 上述研究工作对于云计算开发环境的设计与实现都有一定的启发和帮助, 但都不能解决云计算应用服务开发运维所面临的特殊挑战, 因此很难照搬或直接使用.

3 云计算应用服务开发环境的 4 个发展趋势

基于上述 “三类四代” 的调研结果, 我们可以清晰地看到云计算应用服务开发环境的总体发展趋势, 包括虚拟化、轻量化、智能化和可视化. 简单来说, 就是资源消耗越来越少、开发效率越来越高、开发门槛越来越低、用户界面越来越友好. 下面我们分别阐述每一个趋势.

3.1 虚拟化

云计算开发环境为什么要尽量虚拟化? 首先, 是为了提升开发环境的安全性, 尤其是企业安全 (代码、文档、产品等需要受公司审计和监控) 和技术安全 (需要虚拟化技术在物理层面做隔离以防止业务间相互攻击). 其次, 是为了兼顾资源的利用率和开发性能. 如果用户申请的开发机性能不够, 导致开发时出现性能瓶颈, 则开发效率低下、体验较差; 反过来, 如果用户申请的开发机性能过高、导致大多数时间处于 “空转” 状态, 则资源浪费较多、开发环境的运营管理和维护成本较高. 最后, 是为了简化开发环境的搭建流程, 因为在实体机器上搭建环境往往非常繁琐, 容易出错, 不容易被同时更新和维护.

云计算开发环境可以被怎样虚拟化? 在过去的二十多年中, 我们观察到虚拟化技术的大趋势是不断采用新技术平衡性能和安全性, 虽然性能与安全性往往难以同时兼顾, 但技术的革新使这一理想逐渐成为可能. 对于传统虚拟机 (如 VMware [83] 和 VirtualBox [84]), 借助 Intel/AMD 提供的硬件虚拟化技术 (VT [85]), 在保证安全性的同时能够极大提升虚拟化性能. Linux 以共享内核的方式达到高效虚拟化的目的 (如 Xen [86]), 虚拟化开销比传统虚拟机低很多, 但有 Linux 内核版本的限制. 而通过 Linux 内核提供的 Namespace, CGroups [87] 等进程组技术, 在 Docker 和 Rocket 等容器管理程序的帮助下 [55], 可以在隔离 Linux 进程组的同时, 让 Linux 进程以接近原生的性能运行, 当然不可避免的是安全性较差. 最近, 谷歌发布了一个用于提供安全隔离的轻量级容器运行时沙箱 “Google gVisor” [88], 将客户机的内核运行在宿主机的用户态, 提供了强隔离边界, 可以将其视为极致半虚拟化的操作系统, 不过相比于 Docker 和 Rocket 又牺牲了一定的性能和兼容性.

3.2 轻量化

轻量化主要关注云计算开发环境对应用服务的托管粒度. 我们发现: 总的趋势是应用服务被托管的粒度越来越细, 并且被托管的逻辑更加简单和业务导向. 具体来说, 我们将所观察到的主要托管粒度分为如下 4 个级别: (1) 较粗的托管粒度, 如亚马逊 EC2 只负责虚拟机级别的管理和维护; (2) 中等托管粒度, 如谷歌 Kubernetes 管理和调度容器集群 [89]; (3) 较细的托管粒度, 如谷歌 App Engine 或 Heroku 能够托管某个特定语言或框架的应用; (4) 目前最细的托管粒度是托管函数片段, 已经得到亚马逊 AWS、谷歌云、微软 Azure、阿里云和腾讯云等主流云计算平台的广泛支持.

3.3 智能化

云计算开发环境的智能化主要包含 3 个方面 (不仅限于人工智能). 首先是服务对象的智能化, 需

要服务面向特定领域的应用, 如机器学习、语音识别、计算机视觉、物联网等. 其次是由传统单机 IDE 衍生出来的智能化, 例如代码提示、联想和纠错 [90], 甚至跨项目的代码映射、关联和匹配. 此外还有云计算应用服务开发环境自身的智能化, 例如提供服务后如何实现规模效应, 这里智能调度起到关键性作用, 以及用户历史数据驱动 [91] 对于开发环境各菜单和功能键的智能显示与布局.

3.4 可视化

云计算开发环境的可视化是一个非常柔性、较难量化的指标, 和应用服务的刚性性能几乎没有什么关系, 但对用户来说却是最具直观效果的. 高度可视化的开发环境能够提升程序员的工作效率、降低出错概率、降低准入门槛、方便理解与操作, 还能与云计算应用开发的其他流程无缝接合 (例如需求分析、设计和部署等). 可视化的具体实现形式有很多种, 根据我们的调研, 目前主流的可视化形式主要有如下 3 种: (1) 传统类型 IDE 式, 如 Visual Studio 或 Coding WebIDE 所提供的可视化环境; (2) 教育风格的积木式, 如麻省理工学院开发的 Scratch 系统, 很适合青少年编程教学; (3) 数据或计算流图式, 如 Nodered [92] 系统所提供的互动计算流图.

4 Cloud Studio: 集成型托管开发环境的继续探索

集成型托管开发环境 (如亚马逊 Cloud9) 代表了云计算应用服务开发环境当前的最高发展水平, 但在实际使用过程中, 我们发现 (即使亚马逊 Cloud9) 依然存在多个值得创新和优化的地方.

(1) 尚不支持构件拖拽式开发, 也就是说还没有达到 Visual Studio 中 MFC (基础类库可视化编辑器) 的程度, 这主要受限于两方面原因: 第一, 开发界面围绕细节代码逻辑, 而非宏观数据流程; 第二, 构件支持侧重内部功能效用, 而非构件间连接关系.

(2) 提供的开发接口十分丰富但异构复杂, 需要开发者编写很多 “胶水” 代码来使用, 不直观、易出错, 并且重复劳动多.

(3) 用户的开发流程是资源强隔离的, 每个用户都需要分配一台虚拟机, 消耗资源比较多, 从而导致基础设施成本高, 并且由于虚拟机环境加载慢, 明显影响用户体验.

总的来说, 我们在第 3 节总结出来的 4 个发展趋势尚未得到较为满意的体现. 基于对云计算开发环境的综合理解、长期体验及其发展趋势的总体把握, 我们研制出 (清华大学) Cloud Studio 图形化集成开发环境. 针对集成型托管开发环境现存的上述 3 个核心问题, 我们设计并实现了 4 项关键技术. 首先, 针对不支持构件拖拽式开发的问题, 我们实现了 (1) 基于事件流图的 Web 构件可视化开发; 其次, 针对所提供接口过于异构复杂的问题, 我们进行了 (2) 封装函数间胶水代码的简化接口实现; 此外, 针对用户开发环境强隔离导致资源消耗过多的问题, 我们实现了 (3) 基于互动令牌机制的代码编辑弱隔离以及 (4) 应用适配的容器/虚拟机运行时隔离. 每一项关键技术在研制 Cloud Studio 的具体过程中都有明确的应用并取得了切实的效果.

研制 Cloud Studio 的第 1 步, 需要提炼一系列云计算开发的标准构件. 如图 5 所示, 首先是标准输入输出构件, 比如测试时间、HTTP 事件、消息队列事件、定时器事件、调试输出、邮件输出等; 其次是各种标准函数, 实现图像处理、格式转换、流速控制、参数验证、流程切换、身份认证等常用操作; 此外还包括同等重要的第三方服务适配器, 帮助开发者适配各类云服务, 如对象存储、数据库、缓存等.

研制 Cloud Studio 的第 2 步, 需要提炼一系列标准接口, 通过关键技术 (1) 基于事件流图的 Web 构件可视化开发和关键技术 (2) 封装函数间胶水代码的简化接口实现的使用, 实现用户友好的可视化

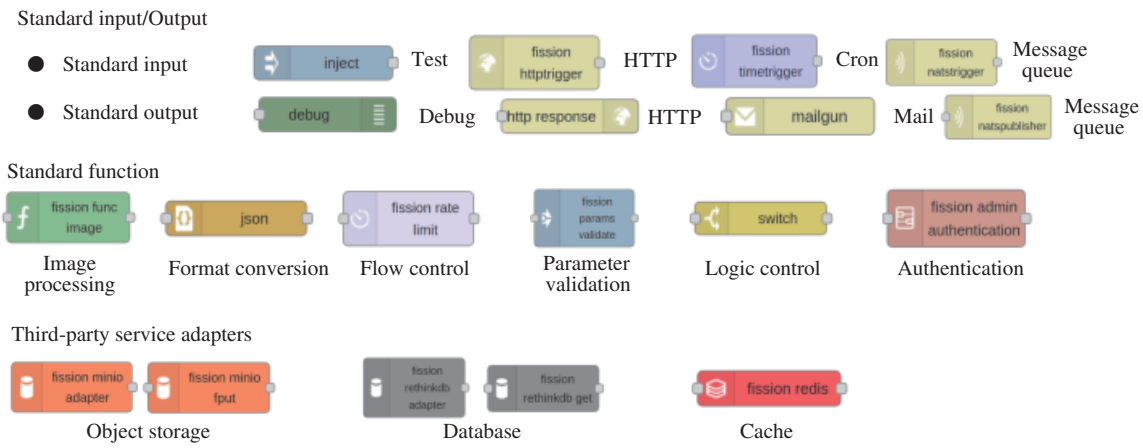

图 5 (网络版彩图) Cloud Studio 标准构件示例

Figure 5 (Color online) Examples of Cloud Studio's basic building blocks

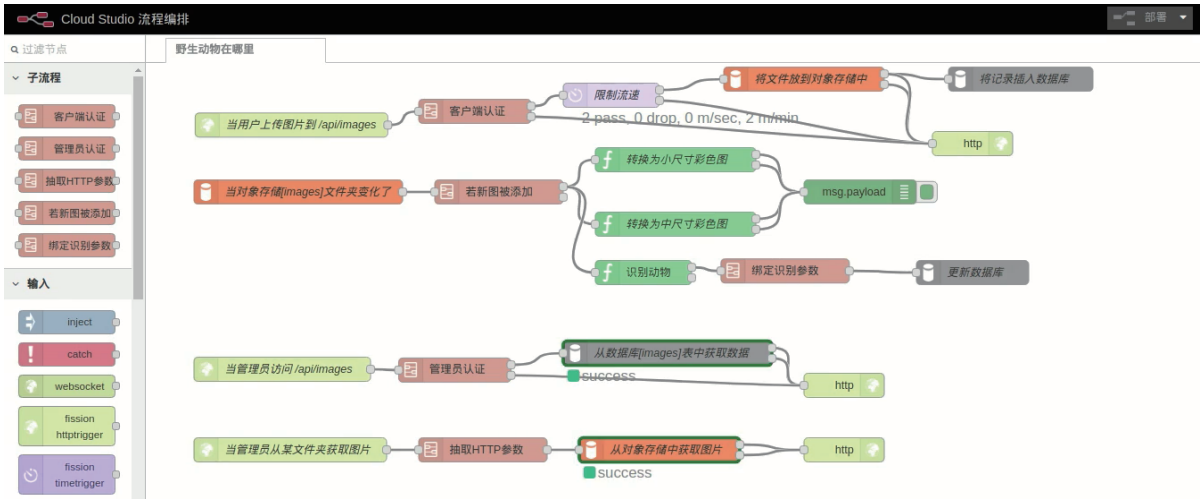

图 6 (网络版彩图) Cloud Studio 流程编排 (使用截图)

Figure 6 (Color online) Process orchestration of Cloud Studio (screenshot)

流程编排 [93]. 如图 6 所示, 让云计算应用服务的开发像我们平时绘制流程图一样简单, 方框表示数据的产生或消耗, 连线表示数据的流动, 程序的顶层逻辑一目了然. 由于每个构件节点的输入输出都有详细定义, Cloud Studio 能用这些定义自动检查接口的有效性, 从而让接口约定有了工具级别的检查保证, 有效避免了后期集成时的困难.

研制 Cloud Studio 的第 3 步, 还需要支持构件或接口代码的深度定制. 云计算应用服务的构件和接口千变万化, 我们提炼和实现的标准构件和标准接口能够提供基本功能, 但不可能面面俱到. 为了帮助用户方便地进行构件和接口的深度优化和个性定制, 我们为用户提供了非常简单的定制方式: 双击构件即可修改函数的实现代码和构件的配置代码, 并且用户修改之后自动执行节点配置的正确性检查. 开发过程中, 版本管理和测试也十分重要, 所以 Cloud Studio 的节点还附带有版本管理以及节点的输入输出自测功能.

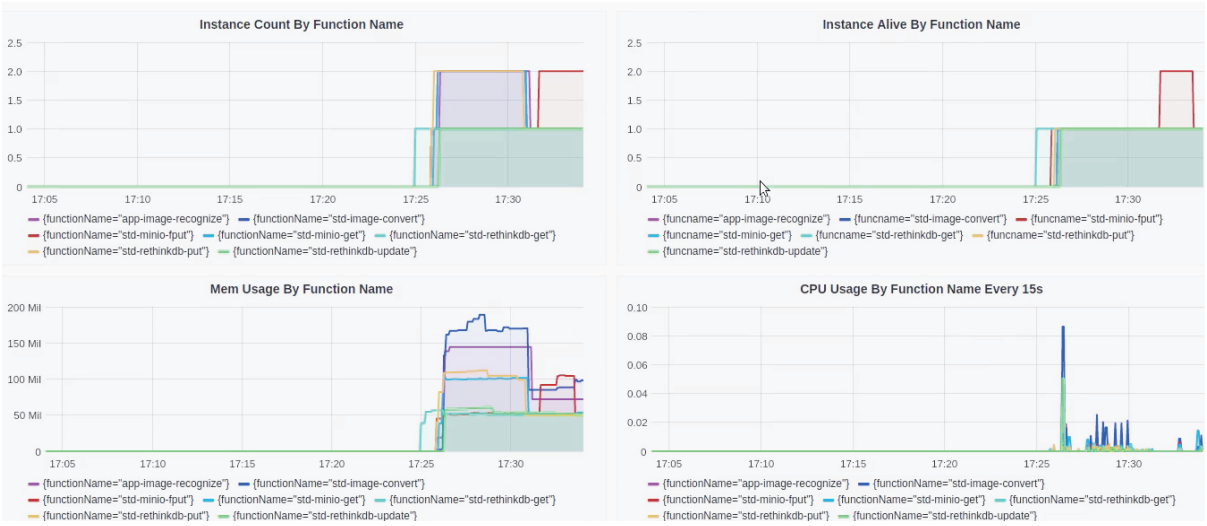

图 7 (网络版彩图) Cloud Studio 自动监控 (使用截图)

Figure 7 (Color online) Automatic system monitoring of Cloud Studio (screenshot)

经过上述三步的努力, 我们将云计算应用服务的开发从聚焦微观层面的代码逻辑转变为聚焦宏观层面的数据流图 [94], 能够显著提升开发效率和用户友好型, 降低开发难度和学习门槛. 这里需要指出的是: 在用户使用 Cloud Studio 开发云计算应用服务的过程中, 需要同时处理数据流和控制流 [95], 但我们希望用户的注意力首先集中在数据流上, 因为在用户绘制数据流有向图的过程中, 控制流通常就已经被隐式地随数据流图表达出来了. 此外, 由于用户大量地使用 Cloud Studio 提供的标准组件, 部分微观层面的数据流和控制流已直接被标准组件集成接管并实现; 而对于用户自定义的函数, 虽然其数据流和控制流是自主实现的, 但是由于他们需要使用 Cloud Studio 定义的标准接口, 其微观层面的代码逻辑就能够被自动地集成在宏观层面的数据流和控制流中. 综上所述, 宏观层面数据流图的确定, 不仅隐式地包含了宏观层面的控制流, 还同时规范了微观层面的代码逻辑.

上面三步我们实现了云计算应用服务的高效开发和快捷部署. 开发部署之后, 运维工作成为最大挑战 [96]. 如图 7 所示, Cloud Studio 可以将开发者应用的运行详细情况, 如调用次数、实例个数、延时分布、资源使用情况等以图表的形式清晰地呈现给开发者, 并自动监控各项指标, 完成资源的调度和报警给开发者需要关注的异常. 从可视化开发到简易化运维, 不需要开发者的额外努力, Cloud Studio 即可自动完成大量运维工作, 包括自动日志收集、自动系统监控, 以及自动线上性能采样追踪.

图 8 是 Cloud Studio 运行时体系架构图, 通过关键技术 (3) 基于互动令牌机制的代码编辑弱隔离和关键技术 (4) 应用适配的容器/虚拟机运行时隔离的使用, 有效降低了单用户在线开发所需消耗的资源, 一台主流的商品化物理服务器能够支撑高达 500 个左右的容器, 而一个用户在线开发通常只需要使用几个容器. 此外, Cloud Studio 还能细粒度监控每个容器或虚拟机的资源消耗, 给云计算用户提供实时准确的计费参考. 更具体地, Cloud Studio 在第三方服务与用户流程的执行中, 通过事件管理架起了一座桥. 它执行开发者之前所绘制的流程图, 所以不需要用户部署代码, 用户所需要的具体业务逻辑由函数计算引擎来执行. 实际上, Cloud Studio 自身就是一个云计算原生应用 [97], 仅由数十个微服务构成 [98], 它自身支持在容器框架上的快速部署和自动伸缩.

除了在线部署, 让注册用户公开使用, 我们还将 Cloud Studio 的项目代码发布到开源社区著名云

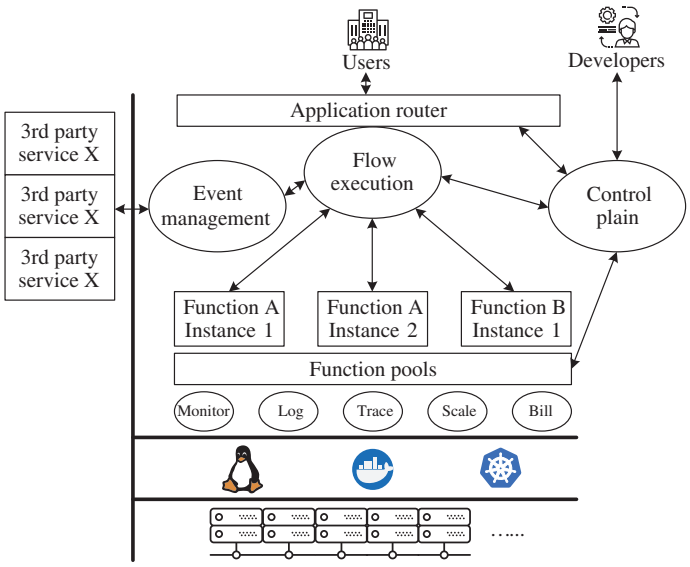

图 8　(网络版彩图) Cloud Studio 运行时体系架构

Figure 8　(Color online) Run-time system architecture of Cloud Studio

计算项目 Fission 与 Nodered 中. 到目前为止, 我们提交的 15 个功能已经被 Fission 正式接受, 包括函数计算性能优化、第三方服务融合、实时监控、自动伸缩、编排运行等.

5　总结与展望

云计算是继 1980 年代全球信息产业从以大型计算机为核心向以低端计算机组成分布式计算系统的大转变之后的又一次计算模式巨变, 虽然它的历史只有十余年 (2006 年以亚马逊 EC2 为工业起点, 2007 年谷歌 101 课程正式给以学术命名), 其发展之蓬勃、扩散之迅速, 可能是人类信息产业史无前例的 —— 几乎所有的发达国家与有实力的发展中国家, 所有的计算机产业知名公司与众多不知名公司, 都以巨大的人力和资本投入到这场计算革命中, 并期望能在这场转变中打造自身实力、争取自身位置.

云计算不是某项单一的技术, 不属于计算机既有的某个特定领域, 它是分布式操作系统、并行计算、效用计算、网络通信、可靠存储、虚拟化、负载均衡等多项技术和理念融合汇聚的产物, 但同时又在改变甚至重塑这些传统技术领域的面貌. 本文所研究的云计算应用服务开发环境, 处在云计算革命发生十余年后, 刚性 (面向机器的) 技术趋于成熟而柔性 (面向用户的) 技术刚刚起步的关键时期 [5], 国际上尚未出现一家独大的垄断局面, 也没有业界公认的技术标准或规范被推出, 留出一个宝贵的窗口期, 是我们国家云计算学术界和工业界实现自主创新、填补产业空白的良好机遇.

在广泛实践调研国际国内各主流云计算开发环境的过程中, 我们可喜地看到: 我国的阿里云、华为云、腾讯云、Coding 等多家公司, 在这一重要领域有长期的投入和丰富的产出并且一直在提升进步, 和国际云计算巨头 (亚马逊 AWS、谷歌云及微软 Azure) 的差距越来越小, 甚至某些方面已经平齐或者做出了自身特色 —— 腾讯云、LeanCloud 针对微信小程序和小游戏等提供了包括通讯、存储、支付等全功能的一站式云开发服务; 百度云、阿里云、第四范式等推出覆盖面向人工智能领域的全行业云计算开发环境; 华为云对于开源 OpenStack 社区的贡献已经连续几年在国际上名列前茅.

随着云计算模式的进一步深化拓展和渗透到国民经济各个领域, 会有越来越多的新特性设备发

明、新应用场景出现、新关键技术聚合、新用户需求适配, 必将给云计算应用服务开发环境的研究提出更多的问题、带来更多的机会、实现更深的意义, 架起底层基础设施 (IaaS) 与上层应用服务 (SaaS) 之间真正人性化、智能化的平台 (PaaS) 桥梁. 具体来说, 在云计算模式出现之前, 新特性设备发明的初期成本极高, 往往只有有实力的大公司才能使用这些设备; 而借助云计算技术, 前沿的非易失性存储介质 (non volatile memory [99]) 技术的实现 —— Intel Optane Memory 已经能在阿里云、腾讯云上租用, 而人工智能加速卡 (如 Google TPU [100]、寒武纪 NPU [101]) 未来可能在多家云平台上随时租用, 云计算开发环境未来应该支持这些设备的灵活共享与经济租用. 其次, 无人车和物联网等新应用场景的出现, 可能需要大量的数据并同时产生大量的数据, 云计算开发环境未来应该能够帮助用户更高效地汇聚与处理海量数据, 并为不同公司间的数据分享提供方便和安全的接口, 加速新应用场景的成熟. 再次, 新一代的无线通讯、芯片制造和量子技术可能将撬动冯 · 诺依曼 (John von Neumann) 体系架构, 未来云端的硬件环境与体系架构可能不断升级与转变, 而客户端则不太可能快速更新, 云计算开发 "软" 环境将能够帮助开发者更好地完成开发方式的渐进式更新. 最后, 为了适配新用户需求, 进一步降低云计算开发的门槛, 我们在 Cloud Studio 中构思设想并已原型实现的 "宏观层面的数据流图式云计算编程" 的基本粒度还可以包含应用、服务和函数, 甚至下降到单个参数, 从而更方便软件复用, 节省开发者在拼接各种不同模块时的心智开销, 最终实现云计算全民编程.

参考文献

1 Forum. Commun ACM, 2008, 51: 9–11
2 Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. Commun ACM, 2010, 53: 50–58
3 Furht B, Escalante A. Handbook of Cloud Computing. Berlin: Springer, 2010
4 Reese G. Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. Sebastopol: O'Reilly Media, Inc., 2009
5 Li Z H, Zhang Y, Liu Y H. Towards a full-stack DevOps environment (platform-as-a-service) for cloud-hosted applications. Tsinghua Sci Tech, 2017, 22: 1–9
6 Hajjat M, Sun X, Eric Sung Y-W, et al. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), New Delhi, 2010. 243–254
7 Khajeh-Hosseini A, Greenwood D, Sommerville I. Cloud migration: a case study of migrating an enterprise IT system to IaaS. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD), Miami, 2010. 450–457
8 Amazon. Amazon AWS Cloud9. 2018. https://aws.amazon.com/cloud9
9 Google. Google App Engine. 2018. https://cloud.google.com/appengine
10 Aliyun. Aliyun Function Compute. 2018. https://www.alibabacloud.com/zh/product/function-compute
11 Huawei. Huawei Devcloud. 2018. https://www.huaweicloud.com/devcloud
12 Coding. Coding Web IDE. 2018. https://ide.coding.net
13 THUcloud group. Tsinghua Cloud Studio. 2018. http://thucloud.com
14 Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg. In: Proceedings of the 10th European Conference on Computer Systems (EuroSys), Bordeaux, 2015. 18
15 Wikipedia. LAMP (Linux, Apache HTTP Server, MySQL and PHP). 2018. https://en.wikipedia.org/wiki/LAMP (software bundle)
16 Gitlab. Gitlab Source Code Management System. 2018. https://gitlab.com
17 Apache. Mesos Distributed Resource Management Framework. 2018. http://mesos.apache.org
18 Grafana. Grafana Monitoring System. 2018. https://grafana.com
19 The Linux Foundation. Prometheus Alarm System. 2018. https://prometheus.io
20 Jenkins. Jenkins Continuous Integration System. 2018. https://jenkins.io
21 Jaeger. Jaeger Distributed Tracking System. 2018. https://www.jaegertracing.io

22 CodeAnywhere. CodeAnywhere cloud development environment. 2018. https://codeanywhere.com
23 Jupyter. Jupyter Notebook. 2018. http://jupyter.org
24 ShiftEdit. ShiftEdit Online Editor. 2018. https://shiftedit.net
25 NeutronDrive. NeutronDrive Editor. 2018. https://super.neutrondrive.com
26 Amazon. Amazon AWS Elastic Beanstalk App Engine. 2018. https://aws.amazon.com/elasticbeanstalk
27 Heroku. Heroku App Engine. 2018. https://www.heroku.com
28 Google. Google Firebase. 2018. https://firebase.google.com
29 Redhat. OpenShift. 2018. https://www.openshift.com
30 Codio. Codio. 2018. https://codio.com
31 Sina. Sina App Engine. 2018. https://sae.sina.com.cn
32 Baidu. Baidu App Engine. 2018. https://cloud.baidu.com/product/bae.html
33 LeanCloud. LeanCloud. 2018. https://leancloud.cn
34 AppScale. AppScale App Engine. 2018. https://www.appscale.com
35 Amazon. Amazon AWS Lambda. 2018. https://aws.amazon.com/lambda
36 Google. Google Cloud Functions. 2018. https://cloud.google.com/functions
37 Microsoft. Microsoft Azure Functions. 2018. https://azure.microsoft.com/en-us/services/functions
38 IBM. IBM OpenWhisk. 2018. https://www.ibm.com/cloud/functions
39 Tencent. Tencent Cloud Function. 2018. https://cloud.tencent.com/product/scf
40 Auth0. Auth0 WebTasks. 2018. https://webtask.io
41 Kubeless. Kubeless Cloud Function. 2018. https://github.com/kubeless/kubeless
42 Platform9. Fission Cloud Function. 2018. https://github.com/fission/fission
43 Spotinst. Spotinst SaaS. 2018. https://spotinst.com
44 Google. Google Cloud AutoML Vision. 2018. https://cloud.google.com/automl
45 IBM. IBM Watson Internet of Things Platform. 2018. https://www.ibm.com/internet-of-things
46 The Fourth Paradigm. Prophet. 2018. https://www.4paradigm.com/product/prophet
47 Aliyun. Aliyun PAI. 2018. https://data.aliyun.com/product/learn
48 Amazon. Amazon AWS SageMaker. 2018. https://aws.amazon.com/sagemaker
49 Jingdong. Jingdong NeuHub. 2018. https://neuhub.jd.com
50 Microsoft. Microsoft Azure Bot Service. 2018. https://azure.microsoft.com/en-us/services/bot-service
51 Tencent. Weixin IoT. 2018. https://iot.weixin.qq.com
52 Baidu. Baidu IoT. 2018. https://cloud.baidu.com/solution/iot/index.html
53 Ubidots. Ubidots IoT. 2018. https://ubidots.com
54 Lin L, Shi W C. Survey of open source software for building cloud computing platforms. Comput Sci, 2012, 39: 1–7 [林利, 石文昌. 构建云计算平台的开源软件综述. 计算机科学, 2012, 39: 1–7]
55 Bernstein D. Containers and cloud: from LXC to docker to kubernetes. IEEE Cloud Comput, 2014, 1: 81–84
56 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), Boston, 2010. 95
57 Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI), San Jose, 2012. 2
58 Christensen J H. Using RESTful web-services and cloud computing to create next generation mobile applications. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA), Orlando, 2009. 627–634
59 Xiao H, Li Z H, Zhai E N, et al. Towards web-based delta synchronization for cloud storage services. In: Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST), Oakland, 2018. 155–168
60 Pautasso C, Zimmermann O, Leymann F. RESTful web services vs. “Big” web services: making the right architectural decision. In: Proceedings of the 17th International Conference on World Wide Web (WWW), Beijing, 2008. 805–814
61 Pimentel V, Nickerson B G. Communicating and displaying real-time data with WebSocket. IEEE Internet Comput, 2012, 16: 45–53
62 Malawski M, Kuzniar M, Wojcik P, et al. How to use Google app engine for free computing. IEEE Internet Comput,

2013, 17: 50–59
63 Wright W, Schroh D, Proulx P, et al. The sandbox for analysis - concepts and methods. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Montréal, 2006. 801–810
64 Han H, Kim S, Jung H, et al. A RESTful approach to the management of cloud infrastructure. In: Proceedings of the 2nd IEEE International Conference on Cloud Computing (CLOUD), Bangalore, 2009. 139–142
65 Adzic G, Chatley R. Serverless computing: economic and architectural impact. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), Paderborn, 2017. 884–889
66 Hendrickson S, Sturdevant S, Harter T, et al. Serverless computation with OpenLambda. In: Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), Denver, 2016
67 Eivy A. Be wary of the economics of "serverless" cloud computing. IEEE Cloud Comput, 2017, 4: 6–12
68 Paraiso F, Haderer N, Merle P, et al. A federated multi-cloud PaaS infrastructure. In: Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD), Honolulu, 2012. 392–399
69 Ducasse S, Lienhard A, Renggli L. Seaside: a flexible environment for building dynamic web applications. IEEE Softw, 2007, 24: 56–63
70 Benson T, Akella A, Shaikh A, et al. CloudNaaS: a cloud networking platform for enterprise applications. In: Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC), Cascais, 2011. 8
71 Sherry J, Hasan S, Scott C, et al. Making middleboxes someone else's problem: network processing as a cloud service. In: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), Helsinki, 2012. 13–24
72 Xu T Y, Zhang J Q, Huang P, et al. Do not blame users for misconfigurations. In: Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), Farminton, 2013. 244–259
73 Wang Q, Ren K, Meng X Q. When cloud meets eBay: towards effective pricing for cloud computing. In: Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM), Orlando, 2012. 936–944
74 Mellor S J, Balcer M, By-Jacoboson I F. Executable UML: a Foundation for Model-driven Architectures. Boston: Addison-Wesley Longman Publishing Inc., 2002
75 Tian L C, Zhang L, Zhou B S. Research and realization of UML-based integrated software development environment. J Beijing Univ Aeronaut Astronaut, 2003, 29: 935–938 [田丽从, 张莉, 周伯生. 基于 UML 的集成化软件开发环境的研究与实现. 北京航空航天大学学报, 2003, 29: 935–938]
76 Fang M Y, Dai X P. Research and realization of UML - base integrated case environment. Comput Tech Develop, 2006, 16: 26–31 [方木云, 戴小平. 基于 UML 的集成化 CASE 平台的研究和实现. 计算机技术与发展, 2006, 16: 26–31]
77 Mei H, Chen F, Feng Y D, et al. ABC: an architecture based, component oriented approach to software development. J Softw, 2003, 14: 721–732 [梅宏, 陈锋, 冯耀东, 等. ABC: 基于体系结构, 面向构件的软件开发方法. 软件学报, 2003, 14: 721–732]
78 Huang G, Wang Q X, Mei H, et al. Research on architecture-based reflective middleware. J Softw, 2003, 14: 1819–1826 [黄罡, 王千祥, 梅宏, 等. 基于软件体系结构的反射式中间件研究. 软件学报, 2003, 14: 1819–1826]
79 Kerrigan M, Mocan A, Tanler M, et al. The web service modeling toolkit - an integrated development environment for semantic web services. In: Proceedings of the 4th European Semantic Web Conference (ESWC), Innsbruck, 2007. 789–798
80 Hatcliff J, Deng X H, Dwyer M B, et al. Cadena: an integrated development, analysis, and verification environment for component-based systems. In: Proceedings of the 25th IEEE International Conference on Software Engineering (ICSE), Portland, 2003. 160–173
81 Chafle G, Das G, Dasgupta K, et al. An integrated development environment for web service composition. In: Proceedings of the IEEE International Conference on Web Services (ICWS), Salt Lake City, 2007. 839–847
82 Bragdon A, Reiss S P, Zeleznik R, et al. Code bubbles: rethinking the user interface paradigm of integrated development environments. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), Cape Town, 2010. 455–464
83 Sugerman J, Venkitachalam G, Lim B-H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. ACM Trans Comput Syst, 2001, 15: 1–14
84 Watson J. VirtualBox: bits and bytes masquerading as machines. Linux J, 2008, 166: 1

85 Uhlig R, Gil N, Dion R, et al. Intel virtualization technology. Computer, 2005, 38: 48–56
86 Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), Bolton Landing, 2003. 164–177
87 Rami R. Linux containers and the future cloud. Linux J, 2014, 240: 86–95
88 Google. Google gVisor container runtime sandbox. 2018. https://github.com/google/gvisor
89 Brewer E A. Kubernetes and the path to cloud native. In: Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC), Kohala Coast, 2015. 167–167
90 Xu T Y, Jin X X, Huang P, et al. Early detection of configuration errors to reduce failure damage. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, 2016. 619–634
91 Zhang Y Q, Prekas G, Fumarola G M, et al. History-based harvesting of spare cycles and storage in large-scale datacenters. In: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI), Savannah, 2016. 755–770
92 JS Foundation. Nodered IoT orchestration tool. 2018. https://nodered.org
93 Peltz C. Web services orchestration and choreography. Computer, 2003, 10: 46–52
94 Chinnici R, Moreau J-J, Ryman A, et al. Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, 2007, 26: 19
95 Curbera F, Duftler M, Khalaf R, et al. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Comput, 2002, 6: 86–93
96 Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables DevOps: migration to a cloud-native architecture. IEEE Softw, 2016, 33: 42–52
97 Balalaie A, Heydarnoori A, Jamshidi P. Migrating to cloud-native architectures using microservices: an experience report. In: Proceedings of the European Conference on Service-Oriented and Cloud Computing (ESOCC), Taormina, 2015. 201–215
98 Kang H, Le M, Tao S. Container and microservice driven design for cloud infrastructure DevOps. In: Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), Berlin, 2016. 202–211
99 Lee E, Bahn H, Noh S H. Unioning of the buffer cache and journaling layers with non-volatile memory. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST), San Jose, 2013. 73–80
100 Abadi M, Barham P, Chen J M, et al. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, 2016. 265–283
101 Liu S L, Du Z D, Tao J H, et al. Cambricon: an instruction set architecture for neural networks. In: Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), Seoul, 2016. 393–405

Cloud computing development environment: from code logic to dataflow diagram

Yunhao LIU, Qifan YANG & Zhenhua LI*

School of Software, Tsinghua University, Beijing 100084*, China*
* Corresponding author. E-mail: lizhenhua1983@tsinghua.edu.cn

Abstract Cloud applications possess excellent flexibility and scalability; however, they often require developers to master various cloud computing techniques and build networked systems from scratch, which is difficult and complex for most developers. Recently, there has been considerable growth in development environments that support the partial or complete life cycle of cloud applications because of the growing demand for cloud applications and services from all sectors of the national economy. On the basis of the study of multiple mainstream development environments for cloud applications, we classify them into three categories: home-brewed development environments by users, spontaneously maintained standard components by communities, and managed development environments by public clouds. The third category (managed environments) is further divided into four generations: editor style, application-level, function-level, and integrated development environments. Through in-depth mining of a series of typical cases, we summarize the four major evolution trends of the development environments for cloud applications: virtualization, lightweight, intelligence, and visualization. Using our implemented development environment, named Cloud Studio, we explore possible forms and paradigms of development environments for cloud applications in the future, especially transformation in terms of core concept: from micro-level code logic to macro-level dataflow diagram. We believe that implementing Cloud Studio can effectively reduce requirements on cloud application developers, accelerate development progress, and promote formation of the core competence of China's cloud computing industry.

Keywords cloud computing, development environment, software engineering, data flow diagram, application development

Yunhao LIU was a Professor at the School of Software, Tsinghua University when this research was conducted. He is presently a part of the faculty of the Department of Computer Science and Engineering in Michigan State University. His research interests include distributed systems, sensor networks/RFID, and Internet of Things (IoT).

Qifan YANG is a master's student at the School of Software, Tsinghua University. He received his B.S. degree from Southeast University in 2016. His research areas mainly consist of cloud computing, distributed systems, and cross-OS platforms.

Zhenhua LI is an Associate Professor at School of Software, Tsinghua University. He received his B.S. and M.S. degrees in Computer Science and Technology from Nanjing University in 2005 and 2008, respectively, and his Ph.D. degree Computer Science and Technology from Peking University in 2013. His research areas mainly consist of cloud computing/storage, big data analysis, content distribution, and mobile internet.