DANCE:一种面向云-端动态集成的服务适配方法

张守利[1),2),3)] 刘 晨[2),3)] 韩燕波[1),2),3)] 李晓红[1)]

[1)](天津大学智能与计算学部 天津 300072)

[2)](北方工业大学大规模流数据集成与分析技术北京市重点实验室 北京 100144)

[3)](北方工业大学计算机学院 北京 100144)

摘 要 边缘计算可以通过将计算移到边缘设备上来提高大型物联网流数据处理质量以及降低网络运行成本.对于流数据处理,边缘设备通常只有有限的计算能力和存储能力,显然不能支持所有的实时流数据查询和处理.本文尝试引入服务并在边缘和云之间灵活地划分服务来实现云-端集成,云服务和端服务之间通过事件机制进行服务适配.物联网动态环境中,云-端服务的动态适配是使云基础设施和端设备间无缝集成的关键.动态集成背景下的服务适配需要把握适配时机来应对端服务适配请求的不确定性和非完全适配等难题.针对这一问题,论文提出了一种面向云-端动态集成的服务适配方法(Dynamic Adaption cloud Services with Edge Services, DANCE).这种方法的主要贡献在于:将云服务实例和端服务实例之间的适配问题建模为二分图顶点之间的动态匹配问题,同时结合排队论中的 M/M/c/∞ 模型对二分图最优匹配 Kuhn-Munkres 算法进行了优化改进,保障适配过程中端服务实例的全局平均请求响应时间最小.最后,基于真实的电能质量监控案例和数据,验证了本文方法的有效性.

关键词 云-端集成;云服务;端服务;流数据处理;服务适配

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2020.00423

DANCE: A Service Adaption Approach for the Dynamic Integration of Cloud and Edge

ZHANG Shou-Li[1),2),3)] LIU Chen[2),3)] HAN Yan-Bo[1),2),3)] LI Xiao-Hong[1)]

[1)](*Division of Intelligence and Computing, Tianjin University, Tianjin* 300072)

[2)](*Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, North China University of Technology, Beijing* 100144)

[3)](*School of Computer Science, North China University of Technology, Beijing* 100144)

Abstract In IoT, large-scale sensor data are continually generated in the form of data stream. These data are usually infinite, noisy, of low-value-density, and orderless on a broad panorama. To provide high-quality services, comprehensive analyses of big stream data based on cloud are essential. Although having powerful potential, cloud computing has faced increasing computational challenges like performance, cost, energy consumption, and service qualities because of the exponential growth of big IoT data. Edge computing may improve the processing quality of big IoT stream data and reduce network operational costs by moving computation onto the edge. While the edge equipment usually has very limited computing power as well as storage ability, and apparently cannot support all the processing of big and real-time stream data. Our previous work has introduced services and provided a flexible division of such services between the cloud

收稿日期:2018-06-06;在线出版日期:2019-07-01.本课题得到国家重点研发计划(2018YFB1402500)、国家自然科学基金面上项目(61672042)资助.**张守利**,博士研究生,主要研究方向为服务计算、大规模流数据集成与分析等.E-mail:zhangshoulia@163.com.**刘 晨**,博士,副研究员,中国计算机学会(CCF)会员,主要研究方向为服务计算、大规模流数据集成与分析等.**韩燕波**,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为分布式系统、互联网服务、业务流程管理和协同、大规模流数据集成与分析等.**李晓红**,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为安全软件工程、可信软件、信息安全领域.

and the edge for the cloud-edge integration. The service adaptation between the cloud service and the edge service can be accomplished through the event mechanism. It is the key to enabling the seamless integration of cloud infrastructure and edge equipment. However, in IoT context, the adaption between cloud services and edge services faces challenges because of the uncertainty of edge service request arrivals as well as incomplete matching. Firstly, compared with the cloud infrastructure, edge equipment can join or quit at any time due to their own flexibility. Secondly, the capability of the edge equipment will constantly change with using which can result in dynamic change of the state of the edge services. Finally, driven by the event, the moment of adaptation request for edge service is uncertain because of the peculiarity of the large-scale streaming data. It is because that the generation and arrival of events on edge equipment are uncertain. Thus, the dynamic service adaption in the context of dynamic integration needs to grasp the right moment of adaption to face the challenges of uncertainty of edge service request arrivals as well as incomplete matching. To solve this problem, this paper puts forward a service adaption approach for the dynamic integration of cloud and edge, called as DANCE. The main contributions include: we transform the dynamic service adaptation problem into the improved maximal weight matching model with a dynamic bipartite graph. At the same time, we modify the M/M/c/∞ model in the queuing theory. The modified M/M/c/∞ model is used to optimize the Kuhn-Munkres algorithm for bipartite graph to minimize the average response time for the request of edge service. DANCE guarantees the minimum global average request response time for the edge service instances during the adaptation process. Finally, based on a real dataset from the case of China's State Power Grid, the effectiveness of the proposed approach is demonstrated by examining a series of simulation experiments. Experimental results show that, DACNE can perform better than us previous work that does not consider dynamic adaption, and the total average response time of DANCE is 24.3% less than us previous work. Experimental results verified the effectiveness and efficiency of our approach.

Keywords cloud-edge integration; cloud service; edge service; stream processing; service adaption

1 引 言

随着物联网的快速发展,大规模的传感设备被部署在各个领域应用中,不间断产生大规模的实时流数据.传统基于云的流数据处理架构通常采用数据集中存储、集中处理的方式[1].然而,原始的传感数据流呈爆炸式增长、价值密度低、网络传输时间长、存储成本高,这些都对云基础设施的数据集中处理模式带来了巨大挑战.近年来,许多终端设备发展迅速,例如智能网关、无线基站等,开始具有一定的计算能力.一种新的计算泛型即边缘计算开始出现并受到广泛关注[2-3],其主要思想是将云端的部分计算任务调度到端(边缘设备)上来执行,以减轻云的负担.

作为分布式系统的一种主流集成方法,服务计算技术以其自身的规范性、灵活性、通用性而受到研究人员的广泛认可[4-5].针对云-端集成问题,当前已有研究工作开始关注如何基于服务的思想和技术来加以实现.通常认为,云基础设施的计算能力强大,可以部署和执行耗费计算资源较多的重量级服务.相对而言,端的计算能力较弱,可以部署和执行计算资源耗费较少的轻量级服务.云服务和端服务之间可以通过消息或者事件机制来进行适配,以使能云和端的最终集成[6-9].

在前期工作[10]中,我们基于主动式服务模型,提出了一种云-端集成方法(Seamless Integration of Cloud and Edge with a Service-based approach, SICES).基于主动式数据服务模型[5],使能其被划分为轻量级和重量级两部分,其中轻量级部分可以被迁移到边缘端执行,重量级的部分仍然在云中心运行.利用事件机制,以及前期工作所提出的"激励-响应"

模式，当边缘端设备上的端服务有事件产生时，则将端服务产生的事件流将返回给相应的云服务实例，驱动云服务实例运行.本文把这一过程称为云-端服务的适配过程.

云-端集成过程中的难点与挑战在于端服务实例与云服务实例之间适配的不确定性.主要表现在：(1)相对于云基础设施，端设备由于其自身的灵活性，可以随时加入或退出.且随着设备的使用损耗，端设备自身的能力会不断的变化，这些变化会贯穿集成过程的始终，其所提供的服务状态在时间和空间上是动态变化的；(2)受大规模流数据的驱动，端设备上事件的产生和到达方式是不确定的.例如，国家电网中风电受天气的影响敏感，其引起电能质量异常事件的时机具有很大的不确定性.此外，当一个地区发生了电能质量扰动事件，与其处于同一母线上的所有地区都会受到影响，极短时间内会引起大量电能质量扰动事件的发生.因此，大规模传感流数据环境下基于事件驱动的云-端服务适配请求的时机是不确定的.云服务与边缘端服务之间建立一种稳定的长期连接是浪费的，且随着边缘设备的增加，这种长连接难以应对不断增加的事件流[11].

服务调度一直以来是服务计算中的研究热点，经典的工作有：根据不同的用户级别提供不同响应程度的服务调度；根据不同的服务截止期提供不同级别的服务调度；根据服务执行的优先级的提供不同级别的服务调度策略[12-14].但是上述已有的服务调度策略无法直接适用于本文所面临的云-端适配问题.首先，典型的服务调度策略所面对的是集群环境中的任务调度或者云环境下的服务调度问题.而本文所面临的是新型的云-端集成环境下的云服务与端服务的适配问题，属于一个开放的环境，其复杂度要高于工作流中的任务调度.且接入的端设备本身是具有自主计算能力，其可以灵活加入或者退出.其次，典型的服务调度策略中服务之间进行交换的是服务的输入输出信息，而本文所面临的是大规模的传感流数据，需要实时持续不断的进行数据的传输.例如，国家电网输电系统中所接入的传感设备，每秒能够产生过亿条传感数据.最后，物联网应用系统要求实时响应边缘端的服务请求.例如，输电网中一旦有电能质量扰动事件发生，会造成大面积的电力故障，此时系统需要及时反映发生电能质量扰动的位置，引起扰动的成因等，以便能够及时采取措施.

本文在前期工作的基础上，提出了一种面向云-端动态集成的服务适配方法 DANCE(an approach for Dynamic Adaption Cloud services with Edge services).该方法在已有的主动式数据服务模型的基础上，通过对适配时机的把握来动态建立云服务实例和端服务实例的适配方案，最大限度地缩短所有端服务实例的平均适配请求响应时间，最大限度地利用云-端的资源.主要贡献有，将云服务实例和端服务实例之间的不确定适配问题建模为二分图节点之间的动态匹配问题，同时结合排队论中的 M/M/c/∞ 模型对二分图最大权完美匹配 Kuhn-Munkres 算法[13]进行了优化改进.

2　问题分析

以国家电网电能质量实时分析场景为例.在电网系统中，电力设备的电能质量受到各类干扰源(如风电场、光伏电站、电气化铁路等)影响而产生电能质量异常.国家电网公司已在全国各地的输电线路上部署了超过的 20 000 个传感设备，对电力传输过程中一些重要设备以及敏感用户的电能质量等进行实时监测.每个传感设备包含 2000 多个电能质量指标(电压、电流、频率等)，单个指标每秒产生超过 1000 条数据记录.7000 余个数据采集终端(Data Acquisition Terminals，DATs)用来定制接入电能质量传感流数据并通过路由器传输流数据到云中心.

图 1 展示了基于我们前期所提出的云-端集成方法所搭建的国家电网电能质量分析平台，其中，DAT 已经被实现为一个如表 1 所示的智能终端，部署在云中心与传感设备之间，被认为是一种典型的端设备，可以承担一些对传感数据的预处理任务.

表 1　DAT 硬件以及软件配置

硬件	32 位嵌入式微处理器
	DSP 芯片
	ARM7 TDMI 核的 LPC2214 处理器
软件	嵌入式 Linux 操作系统
	嵌入式 JDK
通信协议	IEC 61850
	TCP/HTTP

为了检测电能质量扰动事件检测并识别干扰源类型，在图 1 所示的架构中设计并实现了两类服务：数据扰动事件抽取服务 DEE(Disturbance Event Extraction service)和干扰源辨识服务 DSI(Disturbance Source Identification service).其中，DAT 上实现并

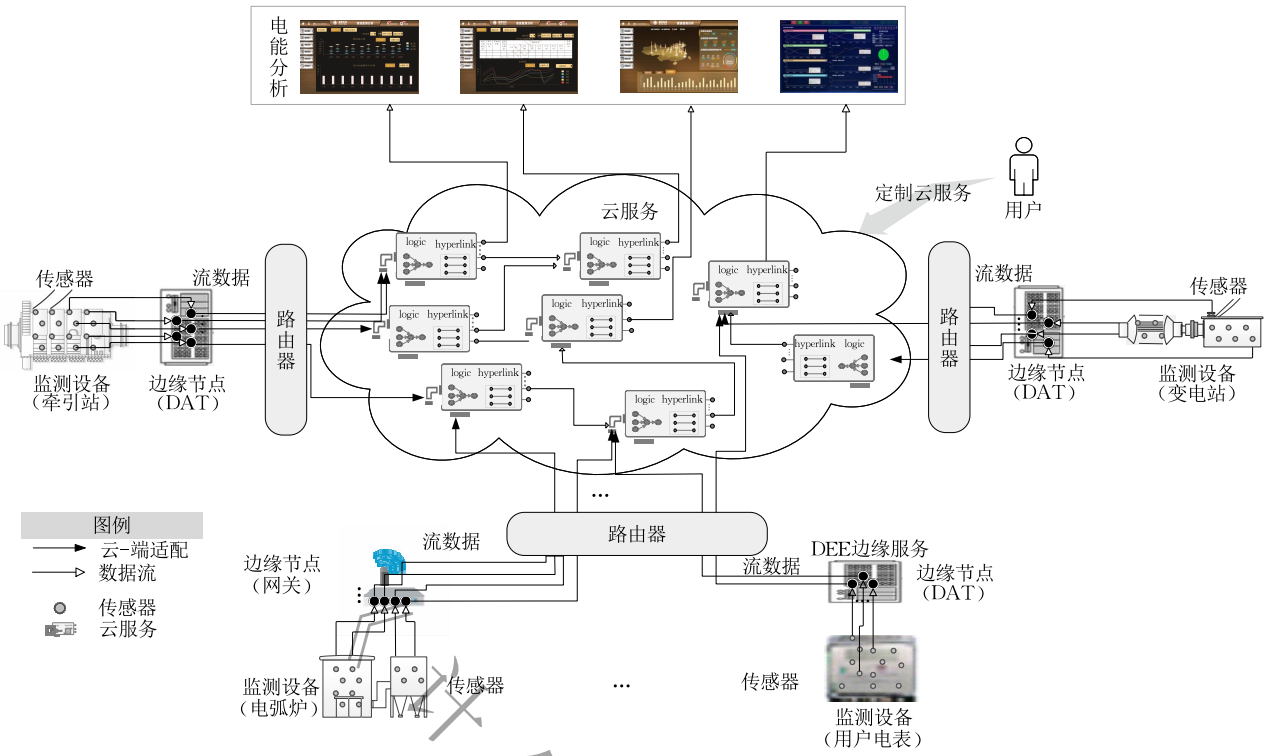

图 1 基于 SICES 的电网电能质量分析平台

部署了轻量级的 DEE，其输入为原始的传感流数据，服务的主要功能是实时检测各类扰动事件（如数据采样值过高、过低等），输出为检测到的事件及事件所包含的数据. 相应的，云基础设施上实现并部署了 DSI 服务. DSI 的输入为 DEE 检测出的扰动事件，其主要的功能是通过学习 DEE 服务发送的扰动事件的趋势变化，来对数据的变化趋势分类，从而判定干扰源的类型和成因（电铁、风电、光伏等）. 该服务依赖于一个巨大递归神经网络.

当 DEE 服务检测到电能质量扰动事件时，DEE 立即通过路由请求与云进行通信，寻找空闲可用的 DSI 服务实例，来接收处理 DEE 输出的扰动事件. 云基础设施确认有可用云服务实例之后，DEE 与 DSI 建立通信通道，DEE 的扰动事件传输到云中心，DSI 接收扰动事件并分析其产生的原因. 这一过程，就是本文所关注所的适配过程，适配的过程是基于事件驱动的.

传统的数据入云集中处理方式每天需要传输并处理超过 PB 级别的流数据，数据规模大，网络和存储压力大，且容易导致数据处理的延迟，无法满足实时性要求. 面向服务的云-端集成方法，可以大大的降低流数据的传输以及存储压力，降低对流数据的处理的响应时间.

然而，由于流数据的固有特性，DAT 上电能质量发生扰动的时机具有很大的不确定性，这也使得云-端的适配时机有了很大的不确定性. 例如，电网中的暂态电能质量问题（脉冲暂态、电压暂降、振荡暂态等）的产生大部分是由于系统内部出错或者电力系统受到了外来的干扰，亦或是人为的错误操作而产生系统冲击，这决定了暂态电能质量扰动事件的发生具有偶然性、随机性等特点. 以风电为例，大规模风电的入网，大大增加了电网中电能质量的不确定因素，风电功率的波动性会导致电压波动事件和电压闪变事件的产生，属于一类常见的干扰源. 而风电受平均风速和脉动风速随机性的影响，使其具体很大的不确定性[15].

此外，电能质量扰动事件往往具有很强的传播性，容易爆发大规模的扰动事件，产生大规模服务适配请求. 例如，在 t_1 = 2017-11-11 04:12:25 时刻，某地区的一回线接地，使得相同线路上的 306 个监测设备产生了共计 1028 个“B 相谐波电压暂降”事件，t_2 = 2017-11-11 04:12:26 时刻（间隔仅 876 ms），受其影响，使得其 109 公里内的 107 个变电站的电能质量数据产生异常，发生了共计 8 种类型的 578 个扰动事件.

扰动事件产生的不确定性使得端服务与云服务之间的适配时机也是不确定的. 因此，实现图 1 所示架构的难点在于如何在运行时有效的调度有限的云

服务实例，动态建立云服务实例和端服务实例的适配方案，来应对边缘端服务不断变化的服务请求时机，保障所有端服务实例的平均适配请求响应时间最小.

3　DANCE 适配方法基本原理

图 2 展示了 DANCE 方法的基本原理，其核心思想是通过云服务和端服务实例之间的动态适配来实现云-端的无缝集成. DANCE 方法基于主动式数据服务模型，在云中心和边缘端上分别部署不同级别的服务[6,10]. 如定义 1 所示，不同于传统的 Web 服务模型[16]，主动式服务的基本思想是在服务的建模过程中融入可配置的流数据处理能力，并将原始低价值密度的传感数据流转变为价值密度高的事件流.

定义 1.　主动式数据服务模型[6]. 主动式数据服务是一个六元组，记为 $ps=\langle APIs, in\_channels, out\_channels, logics, operations, hyperlinks\rangle$.

其中，$APIs$ 是对外提供调用服务的接口；$in\_channels$ 是服务的输入，其用来接收原始传感流数据或者事件流；$out\_channels$ 是服务的输出，输出服务所产生的事件；$logics$ 是服务处理输入数据或事件的逻辑；$operations$ 是 $logics$ 使用的操作集合；$hyperlinks$ 协助服务将产生事件主动抛给其他服务或者应用，类似于路由表.

在 DANCE 方法中，云服务和端服务均基于主动式服务模型实现. 不同的是，云服务是重量级的服务，其封装的是需要消耗大量计算资源与存储资源的计算任务，例如 DSI 服务中所封装的基于循环神经网络的干扰源识别算法. 端服务属于轻量级的服务，其封装的是无需大量计算资源的计算任务，往往可以在智能终端运行，例如图 1 所示的 DEE 服务中所封装的扰动事件抽取操作. 正如前文所提出的，云服务和端服务适配过程的关键在于把握适配的时机. 由于云服务实例有限，而端服务的规模又往往较大，因此很难为每一个端服务实例长期分配一个云服务实例与之适配. 通过对前述场景的观察可以发现，端服务往往在检测到特定的事件产生后，才需要和云服务进行适配，这一现象就给予了对服务适配问题充分的优化空间.

图 2 展示了云-端服务适配的全部过程. 首先，如图 2 中①所示，部署在边缘设备上的端服务实例在运行后将对该边缘设备产生的流数据进行实时分析和处理，对数据中蕴含的关键事件进行检测和捕获（如数据采样值过高、过低等），将原始的数据流转变为事件流. 当第一个事件发生后，端服务实例将向云端发送适配请求，请求对端服务输出的事件流进行处理，如图 2 中②所示.

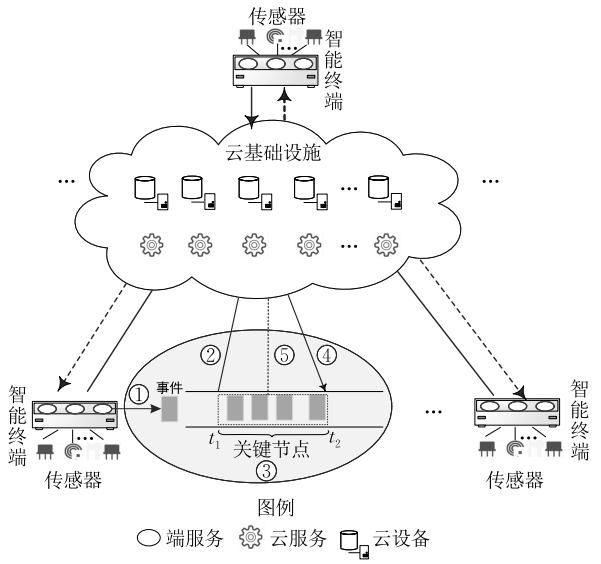

图 2　DANCE 适配方法基本原理

为了便于端服务和云服务实例之间的适配，我们提出了如图 2 中③所示的关键节点这一数据结构. 它标识这端服务实例请求云服务实例适配过程的开始，并在适配过程完成前缓存端服务输出的事件流. 如定义 2 所示，关键节点被定义为一个部署在边缘设备上的、与特定端服务实例绑定的临时事件缓冲区.

定义 2.　关键节点. 关键节点可以表示为 $key\_node=(id, sid, t, E)$，其中，id 是关键节点的编号，作为关键节点的唯一标识；sid 是其所绑定的端服务实例的编号，t 是关键节点产生的时间戳；$E=\{e_1, e_2, e_3, \cdots, e_i, \cdots, e_p\}$ 是缓存的的事件列表.

随后，如图 2 中④所示，端服务实例和云服务实例的动态适配过程开始. 正如前文所分析，端服务实例和云服务实例之间的适配存在不确定性问题，且云服务和端服务实例数量之间可能存在着失配矛盾，也就是说同时存在着大量的端服务实例来请求有限的云服务实例进行适配. 为了解决这一问题，本文在第四节提出了基于排队论和动态二分图的云-端服务实例动态适配算法. 该算法能在保障整体适配方案最优的情况下（所有等待适配的端服务实例的平均延迟响应时间最小），为给定的端服务实例适

《计算机学报》

配适合处理其请求的云服务实例.

最后,如图 2 中⑤所示,在适配算法结束后,端服务实例已经被分配了可以处理其请求的云服务实例,此时两个服务实例之间将建立起事件流的传输通道,端服务关键节点缓存的事件流将传输到云服务中供其处理,并向应用输出最终处理结果.

4 基于动态二分图的服务适配算法

正如图 2 中④所示,端服务和云服务实例适配过程实现的关键是解决端服务实例和云服务实例之间的适配时机的不确定性问题.为了解决这一问题,本节首先将端服务和云服务实例适配问题转化为一个二分图的完美匹配问题.同时,结合排队论中的 M/M/c/∞ 模型来为动态产生的端服务关键节点集合建立排队模型,基于启发式算法优化关键节点的等待时间,减少队列的长度,从而辅助动态二分图计算最大权完美匹配方案,以保障整体端服务实例的平均事件处理时间最小.

4.1 基于动态二分图的云-端服务适配模型

首先给出关于二分图概念.

定义 3. 二分图.给定图 $G=(V,E)$,其顶点集 V 是两个互不相交的非空集合,且每条边 $e_i\in E$ 所在的两个顶点属于互不相交的两个顶点集合,则图 G 为一个二分图.给图中的每条边 $e_i\in E$ 赋值,这个赋值为边的权重,此时图 G 为加权二分图.

定义 4. 匹配.对于加权二分图 G,其每条边的权重为 $\omega_{ij}\geq 0$,如果存在一个匹配 M,使得所有匹配的边的权重的和为最大,则称 M 为 G 的一个最优匹配.若 G 中每个顶点都属于 M 中的顶点,那么 M 是图 G 的完美匹配.加权和最大的完美匹配则是最大权完美匹配(*WMW*).

定义 5. 相等子图.二分图 G,其中两个顶点集合分别为 $X=\{x_1,x_2,x_3,\cdots,x_o\}$, $Y=\{y_1,y_2,y_3,\cdots,y_o\}$,边的权值 $\omega_{ij}=\omega(x_i,y_j)$.设 L 为二分图 G 中顶点集合到实数集的映射,若对此任意的 $x\in X,y\in Y$,均存在 $L(x)+L(y)\geq\omega(xy)$,则称 L 为 G 的可行顶点标记.令 $E_L=\{xy\mid e=xy\in E(G)\}$,且 $L(x)+L(y)=\omega(xy)$,则称 E_L 为图 G 的相等子图.

定理 1. 二分图 G 的等价子图的完美匹配即为 G 的最佳匹配.

定理 2. 二分图 G 存在完美匹配的充分必要条件是:对于任一顶点集合的子集 A,它的相邻点构成的邻集 $X(A)$,都满足 $|X(A)|\geq|A|$.

设 M 是 G 的一个匹配,若顶点 v 是 M 中某条边的端点,则称 v 是 M 饱和的.M 交错路指其边在 $\{E-M\}$ 和 M 中交替出现的路.若一条 M 交错路的起点和终点都是 M 非饱和的,则称其为 M 增广路[17].由 Berge 定理可知,当不能再找到增广轨时,就得到了一个最大的匹配.

二分图匹配是调度与指派问题中的常见优化求解方案[17-18].云-端服务适配问题可以被建模成为一个二分图的匹配问题.DANCE 方法以端服务实例所产生的关键节点为最小适配单元来与云服务实例进行适配.这是因为一个关键节点中缓存了一个时间段内需要云服务实例处理的事件集合,且一个端服务实例可能会不断产生不同的关键节点,需要不同的云服务实例进行处理.

本文将云服务实例与端服务实例关键节点之间的适配问题建模为如图 3 所示的加权二分图模型 $G=(V(G),E(G))$.端服务实例产生的关键节点为二分图的一极顶点集合 K,同时将云服务实例建模为二分图的另一极顶点集合 S,即为 $V(G)=S\cup K$.二分图中的一条边 $e_{ij}=(s_j,k_i)$, $s_j\in S$, $k_i\in K$, $e_{ij}\in E(G)$ 代表一个关键节点和一个云服务实例可以适配.

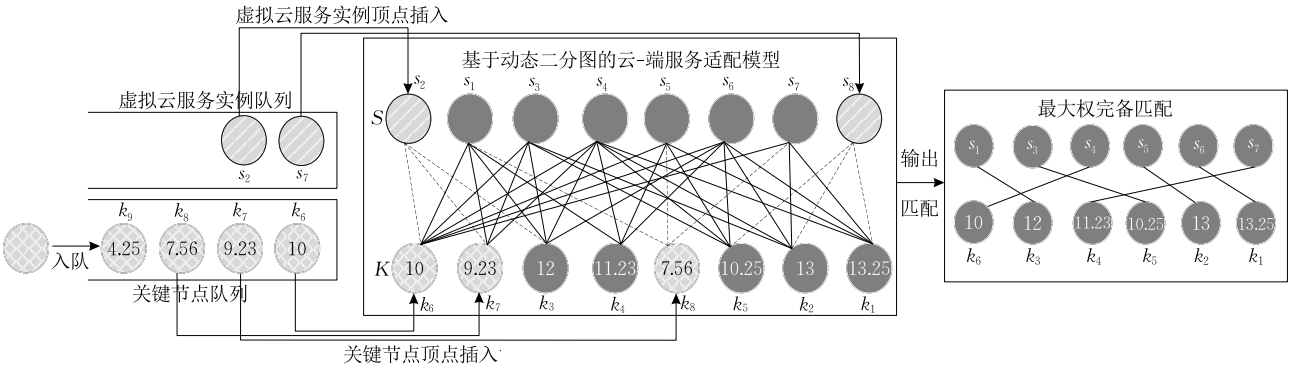

图 3 基于动态二分图的云-端服务实例适配模型

服务适配的过程就是寻找该加权二分图中边的权值和最大的完美匹配. 需要给图 G 的边赋值，为了使得关键节点的平均等待时间最小，本文把到来的关键节点的等待时间赋值给边的权重.

$$\omega_{ij}=TW_{k_i} \tag{1}$$

式(1)中 TW_{k_i} 表示了关键节点 k_i 的等待时间，将由 4.2 节计算得.

对于二分图 G，S 顶点集合是可以提前预知的. 而对于 K 顶点集合，由于其是基于边缘端流数据的驱动下产生的，因此关键节点顶点和相关边可能是动态到达的，对应的加权二分图 G 是动态增长的. 如图 3 所示，边缘端产生的关键节点使得图的 K 的集合是一个开放且动态增长的集合，一旦有新的关键节点顶点插入或一条边的更新可能会导致已有匹配中所有的边的改变. 由于流数据的特性，关键节点的产生不断增加，图的结构规模不断扩大与更新，基于现有的静态算法越来越难以实现对动态二分图匹配的维护. 因此，需要一种动态的服务适配算法来应对本文所面临的难题.

4.2　基于动态二分图的服务适配算法

首先，我们使用排队论中的 M/M/c/∞模型建模二分图中关键节点的产生过程.

对于关键节点来说，其产生服从均值为 λ 的泊松分布，其产生的时间间隔服从参数为 $\gamma=\frac{1}{\lambda}$ 的负指数分布. 处理每个关键节点的时间独立、且服从参数为 μ 的负指数分布. n 表示关键节点集合中表示正在匹配的平均关键节点数. $d=m-|K|$ 表示关键节点集合当前最多可容纳的关键节点数 $d_{\max}=m$.

根据排队论的生灭过程[19]的定义，设 $N(t)$ 表示时刻 t 当前的关键节点数量，则 $\{N(t),t\geq 0\}$ 为一个生灭过程，我们可以得到排队模型的平稳状态的分布，用 $p_n(n=0,1,2,\cdots)$ 表示. 具体的过程为：

根据生灭过程中得队列在任一状态下的稳态方程：

$$\begin{array}{ll}
0 & \mu_1 p_1=\lambda_0 p_0\\
1 & \lambda_0 p_0+\mu_2 p_2=(\lambda_1+\mu_1)p_1\\
2 & \lambda_1 p_1+\mu_3 p_3=(\lambda_2+\mu_2)p_2\\
\vdots & \vdots\\
n & \lambda_{n-1}p_{n-1}+\mu_{n+1}p_{n+1}=(\lambda_n+\mu_n)p_n\\
\vdots & \vdots
\end{array} \tag{2}$$

由上述稳态方程，可求得

$$\begin{array}{ll}
0 & p_1=\frac{\lambda_0 p_0}{\mu_1}\\
1 & p_2=\frac{\lambda_1\lambda_0}{\mu_1\mu_2}p_0\\
2 & p_3=\frac{\lambda_2\lambda_1\lambda_0}{\mu_1\mu_2\mu_3}p_0\\
\vdots & \vdots\\
n & p_{n+1}=\frac{\lambda_n\lambda_{n-1}\cdots\lambda_0}{\mu_1\mu_2\mu_3\cdots\mu_{n+1}}p_0\\
\vdots & \vdots
\end{array}$$

令 $C_n=\frac{\lambda_{n-1}\lambda_{n-2}\cdots\lambda_0}{\mu_1\mu_2\mu_3\cdots\mu_n}$，则队列平稳状态的分布为

$$p_{n+1}=\frac{\lambda_{n-1}\lambda_{n-2}\cdots\lambda_0}{\mu_1\mu_2\mu_3\cdots\mu_n}p_0=C_n p_0 \tag{3}$$

概率分布要求 $\sum_{n=0}^{\infty}p_0=1$，也就是

$$\left[1+\sum_{n=1}^{\infty}C_n\right]p_0=1$$

于是有

$$p_0=\frac{1}{1+\sum_{n=1}^{\infty}C_n} \tag{4}$$

仅当 $\sum_{n=1}^{\infty}C_n<\infty$，才能由式(4)得到排队队列达到平稳状态时的概率分布.

用 $p_n=P\{N=n\}(n=0,1,2,\cdots)$ 表示为队列达到平稳状态后队长 N 的概率分布，对于有 m 个云服务实例的匹配系统有，

$$\begin{aligned}
&\gamma_n=\gamma(n=0,1,2,\cdots),\\
&\mu_n=\begin{cases}n\mu, & n=1,2,\cdots,m\\ m\mu, & n=m,m+1,\cdots\end{cases}
\end{aligned} \tag{5}$$

记 $\rho_s=\frac{\rho}{s}=\frac{\rho}{s\mu}$，则当 $\rho_s<1$ 时，由式(3)和(4)可得

$$C_n=\begin{cases}\frac{(\lambda/\mu)^n}{n!}, & n=1,2,\cdots,m\\ \frac{(\lambda/\mu)^n}{m!m^{n-m}}, & n\geq m\end{cases} \tag{6}$$

则

$$p_n=\begin{cases}\frac{\rho^n}{n!}p_0, & n=1,2,\cdots,m\\ \frac{\rho^n}{m!m^{n-m}}p_0, & n\geq m\end{cases} \tag{7}$$

其中，

$$p_0=\left[\sum_{n=0}^{m-1}\frac{\rho^n}{n!}+\frac{\rho^m}{m!(1-\rho_m)}\right]^{-1} \tag{8}$$

平稳状态下，如果 $n<m$，则新到来的关键节点无需等待就可获取匹配，此时

$$P_{k_i}=1 \tag{9}$$

计算机学报

当 $n \geq m$ 时，即则新到来的关键节点需要等待匹配，此时

$$P_{k_i} = \sum_{n=m}^{\infty} p_n = \frac{\rho^m}{m!(1-\rho_m)} p_n \tag{10}$$

因此，我们可以得到关键节点的平均排队长 L_q，平均队长 L_c，平均等待时间 W_q 和平均处理时间 W_c 为

$$L_q = \frac{p_0 (\lambda/\mu)^c \lambda/(n\mu)}{n![1-\lambda/(n\mu)]^2} \tag{11}$$

$$L_c = L_q + n \tag{12}$$

$$W_c = \frac{L_c}{\lambda} \tag{13}$$

$$W_q = W_c - \frac{1}{\mu} \tag{14}$$

通常来说，队列中的关键节点仅仅按照到达时间进行排队，忽视了关键节点的等待时间以及排队队长，很容易造成排队队列过长[20]. 因此，本文基于粒子群算法(PSO)[21]来优化平均等待时间以及减少队列的长度，对存储在该队列中的关键节点进行批量管理. 首先，计算所有可能的关键节点序列的等待时间，得到最佳或最优的序列，然后返回最小值也就是关键节点在队列中的正确顺序，可以使等待时间最小化，减少队列阻塞.

为了优化等待时间，式(15)表示了目标函数(*objective function*)，表示需要最小化平均等待时间.

$$objective\ function = \min\{W_q\} \tag{15}$$

因此，粒子群算法中用来计算每个粒子的解的适应度函数(*Fitness*)被定义为式(16)所示.

$$Fitness = \min\left\{\frac{1}{k}\sum_{i=1}^{k} WT_{k_i}\right\} \tag{16}$$

其中，WT_{k_i} 是关键节点 k_i 的等待时间，k 是队列中的关键节点的数量. 当 PSO 算法返回最小值时，我们最终得到关键节点的最佳排队顺序，并形成一个新的队列.

优化后的队列，能够使得动态二分图能够在匹配的过程中，维持短时间内的稳定状态. 其实现基本思路是：当有新的关键节点产生时，首先判断当前的二分图 G 中是否已经完成了一次最大权完美匹配，如果存在一个 MWM，则删除二分图中的 MWM 中的饱和顶点，取出队列中关键节点插入到二分图中. 否则，关键节点在队列中继续等待. 如图 3 所示.

本文中，仅当边缘服务有关键节点产生时，请求调度云服务实例，才会创建稳定的二分图 G，根据定理 2，可以得出此时图 G 一定存在一个完美匹配. Kuhn-Munkres(KM)算法[22]适用于图的带权二分图匹配中，可以使得最后得到的权值总和 $Sim(S,K)$ 最大.

$$Sim(S,K) = \max\left\{\sum \omega_{ij}\right\} \tag{17}$$

下面给出 KM 算法的基本实现步骤：

首先根据定义 5 初始化可行顶点标号 L. 求得图 G 的相等子图 G_L. 考虑到本文的关键关节点实际包含一组待处理处理的事件，相当于事件的批到达. 因此，一个云服务实例一次服务一个端服务实例. 为了求取二分图最大权完美匹配，需要满足 $|S| = |K|$. 然而由于 G 中 K 的无限性，使得需要对 KM 算法进行改进：得到服务适配之后，需要删除图 G 中的 MWM 的顶点 $\overline{S}$ 和 $\overline{K}$，表示下次的匹配中，不需要再考虑 $\overline{S}$ 和 $\overline{K}$. 当完成事件的处理之后，$\overline{S}$ 中的服务实例被释放，需要再插入到 G 中进行下一次的匹配. $\overline{S}$ 未被释放之前，则需要为 G 中添加一些虚拟服务实例节点，来表示已经被适配的顶点集合. 虚拟节点的权重设置为 0.

因此基于改进的 KM 算法和排队论，算法 1 (Service-Adaption)实现了动态二分图的服务适配优化算法. 首先初始化队列 Q，关键节点按照产生时间进入排队 Q，利用 PSO 算法对初始队列优化，使得等待时间最优化，得到优化后的队列 Q'. 初始化云端服务实例集合 S 和关键节点集合 K，根据 4.1 节中的二分图模型对 S 和 K 建模为 G，根据式(3)和(14)计算边的权重，如算法 1 行 3 所示. 如果 G 存在一个匹配 M，M 中的顶点是饱和的且满足式(17)，则 M 是要找的一个最大权重最优匹配 WMW，如算法 1 行 2 至行 6 所示. 否则，获取 M 的非饱和顶点，并不断修改顶点的可行标号，执行 KM 算法找到相当子图的完美匹配 M^*，即为 G 最大权重完美匹配 WMW，如算法 1 行 8 到行 15 所示. 提取 MWM 中的云服务实例和端服务实例集合，输出云服务实例到端服务实例的适配方案，如算法 1 行 17 到行 18 所示. 同时删除 G 中 WMW 对应的顶点集合 $\overline{CS_{\text{ins}}}$ 和 $\overline{K}$，也就是删除已经获得云服务实例资源的关键节点和已经被适配了的云服务实例集合，同时如果二分图左侧顶点集合当前允许插入 c 个关键节点，当 $c \neq 0$ 且时，关键节点可被插入到二分图中，否则进入排队等待，直到再次满足条件. 排队队列中等待的关键节点被插入到图 G 中形成新的二分图. 如算法 1 行 19 所示.

算法 1. Service-Adaption 基于动态二分图的服务适配优化算法.

输入：CS_{ins} 可用云服务实例集合；Q 关键节点队列；ES_{ins} 端服务实例集合

输出：$map(CS_{\text{ins}}, ES_{\text{ins}}) = \{cs \rightarrow es, cs \in CS_{\text{ins}}, es \in$

ES_{ins}}云-端调度的适配方案

1. $Q' \leftarrow PSO(Q)$　//优化排队系统的最优排队时间
2. IF($Q \neq \varnothing$){
3. 根据 4.1 节为 CS_{ins} 和 K 建立二分图模型 $G=(CS_{ins} \cup K, E)$，根据式(3)和(14)计算边的权重 ω_{ij}；
4. 初始化一个匹配 $M \subseteq E(G)$；
5. IF($\exists cs \in CS_{ins}$是 M 饱和的且满足式(9)){
6. $WMW=M$;}
7. ELSE {
8. 获取 M 的非饱和点 $Y=\{y, y \in CS_{ins}\}$，$T=Q$；
9. $N_{G_L}(Y)=\{k \mid y \in Y, yk \in G_L\}$；//获取 Y 的邻集
10. IF($\exists N_{G_L}(Y)==T$){
11. 根据式(10)初始化 Y 中的可行顶点标号 L，获取 G 的相等子图 E_L；
12. 令 $\alpha_L=\min\{L(x)+L(y)-we(xy) \mid x \in X, y \in (K-T)\}$，修改每个顶点的可行标号为

$$L'(v)\begin{cases} L(v)-\alpha_L, & v \in S \\ L(v)+\alpha_L, & v \in T \\ L(v) \end{cases}$$

13. 令 $L=L'$，$E_L=E_{L'}$，重新给出 E_L 的一个匹配 M'；
14. $P=(y_k)$ 是 E_L 中 M' 的增广路径，THEN $M^*=M' \oplus P$；//执行匈牙利算法直到找到 E_L 完美匹配 M^*
15. $WMW=M^*$;}
16. }
17. $\overline{CS_{ins}}=\{cs=e.cs, e \in WMW\}$；$\overline{K}=\{k=e.k, e \in WMW\}$；$\overline{ES_{ins}}=\{es=k.sid, k \in \overline{K} \&\& e \in ES_{ins}\}$ //$\overline{CS_{ins}}$和 $\overline{K}$ 分别为 WMW 中云服务实例集合和关键节点结合；$\overline{ES_{ins}}$为端服务集合
18. $map(CS_{ins}, ES_{ins})=\{cs \rightarrow es, cs \in CS_{ins}, es \in ES_{ins}\}$；//云-端服务调度方案
19. $CS_{ins}=CS_{ins}-\overline{CS_{ins}}+\{s'_1, s'_2, \cdots, s'_c\}$，$K=K-\overline{K}+\{k'_1, k'_2, \cdots, k'_c\}$，$\{k'_1, k'_2, \cdots, k'_c\} \in Q$；//更新 CS_{ins}和 K 的顶点集合，使得$|CS_{ins}|=|K|$
20. return $map(CS_{ins}, ES_{ins})$; }

由于 Kuhn-Munkres 算法的时间复杂度为 $O(n^3)$[23]，其中 $n=\max(|S|)$. 当有关键节点插入到 G 中时，只调用改进的 Kuhn-Munkres 算法共 1 次，且关键节点的等待时间为 W_q，则整个动态二分图匹配过程中的时间复杂度为 $O(W_q n^3)$，即为 $O(n^3)$.

5　实　验

5.1　实验环境与数据

实验环境. 本文实验基于一个云基础设施和端环境. 具体的配置如表 2 所示. 20 台虚拟机实例采用1Gbps以太网光纤和交换机搭建云基础设施. 20 台 DATs 作为端接入到所搭建的云环境中. 一台虚拟机用来安装 KEPServerEX① 作为传感设备，基于真实的电网传感数据模拟流数据的产生. DAT 接入 KEPServerEX 的流数据.

表 2　实验环境设置

环境类型	设备	硬件配置	软件配置
云基础设施	20台虚拟机实例	2×32 核 2.4 GHz CPU 128GB 内存 1 TB RAID5	Centos7 Hadoop-2.7 JDK-1.8.5 Spark-2.4.0
流数据模拟器	1台虚拟机	2×32 核 2.4GHz CPU 256GB 内存 1 TB RAID5	Windows 7
端	20 个 DATs	32 位嵌入式微处理器 ARM7TDMI 核的 LPC2214	嵌入式Red Hat JDK 10.0.1

实验数据. 本节基于电网所采集到的真实传感器数据，共有来自某省的 361 个 DATs 的数据，每个 DAT 上接有 2520 个检测指标数据源，用来展示每个监测节点的电能质量情况. 数据采样时间为 2016-07-01 00:00:00 到 2018-01-31 23:59:59.

5.2　实验设置

本文基于前述的扰动事件抽取服务 DEE 和干扰源辨识服务 DSI 服务的实现来验证 DANCE 方法的有效性. 其中，DEE 服务中的逻辑处理部分是基于统计控制图实现对传感流数据数值的监测，实时检测数据采样值过高、过低等扰动事件的发生，其被部署并运行在端(DAT)上，在扰动事件发生后请求适配云端的 DSI 服务实例. DSI 服务是基于一个 24 层，每层 100 个神经元的循环神经网络模型建立的. 每个 DSI 实例运行时需要消耗较多的计算资源. 根据所搭建的云环境，云端可以同时运行 1200 个 DSI 服务的实例. 每个边缘端可以同时运行 200 个 DEE 服务实例，接入的 DATs 组成的边缘端最多可以同时运行 4000 个 DEE 服务实例.

定义 6.　平均响应时间(RS)：表示给定时间段内关键节点从出现到被处理完的平均响应时间.

$$RS=\frac{\sum(k.T_p-k.t)}{l} \tag{18}$$

$k.T_p$为关键节点的处理时间完成时刻，l 表示当前时间段内的关键节点的数目.

(1)实验 1. 与传统流处理方法对比

传统的流数据处理方法基于云计算模型[24]，部署了当前流行的流数据处理框架 Spark. DEE 服务

① https://www.kepware.com/en-us/products/kepserverex/

计算机学报

和 DSI 服务均被部署并运行在 Spark 上. DAT 终端实时采集传感数据,并将数据发送到云端,DEE 和 DSI 服务实例接收并处理来自 DATs 的传感数据.

实验将对比传统流数据处理方法和 DANCE 方法,验证动态环境中本文所提出的 DANCE 方法是否能够有效降低数据传输量,降低扰动事件的平均响应时间. 实验分别模拟实现了以下 3 个动态案例:

① 当接入的端的数量动态变化时,验证其对扰动事件平均响应时间的影响. 具体的,在云端实例化 1200 个 DSI 服务实例. 依次接入 5,10,15,20 个 DATs 时,每个 DATs 的流速率为 100 条/s. 每个 DAT 端启动 200 个服务实例观察不同方法下的数据传输量和事件的平均响应时间.

② 当 DATs 上启动的端服务实例数量动态变化时,验证其对扰动事件平均响应时间的影响. 具体的,在云端实例化 1200 个 DSI 服务实例. 边缘端接入 20 个 DATs. DATs 端依次随机启动 1000,1500,2000,4000 个服务实例. DATs 端的流速为 100 条/s,按照 DATs 的服务实例的数量,观察不同方法下的数据传输量和事件的平均响应时间.

③ 当接入的端的流数据速率动态变化时,验证其对扰动事件平均响应时间的影响. 具体的,在云端实例化 1200 个 DSI 服务实例,边缘端接入 20 个 DATs ,动态改变每个 DATs 的流的速率为 100,200,300,400,500 条/s. 每个 DAT 端启动 200 个服务实例观察不同方法下的数据传输量和事件的平均响应时间.

(2) 实验 2. 与 SICES 云-端集成方法对比

前期工作中的 SICES 云-端集成方法能够把服务划分为两个部分分别在云与端运行,基于二分图建立为端服务的事件处理请求与云服务实例建立的服务匹配模型,用来实现服务的调度,但是其通过简单的图的顶点和边的插入和删除操作来实现服务的动态调度.

实验将对比 SICES 和 DANCE,验证动态环境中 DANCE 是否优化了云-端集成过程中的适配时机不确定性和非完全适配难题,降低扰动事件的平均响应时间. 实验从以下 3 个方面展开:

① 验证接入的端服务的数量对于扰动事件平均响应时间的影响. 具体的,在云端实例化 1200 个 DSI 服务实例,DATs 端依次启动 1000,1500,2000,4000 个服务实例. 保持端上的流速度不变(设置为 1600 条/s),按照 DATs 的数量,观察不同方法的事件平均响应时间;

② 验证接入的 DAT 所产生的流数据速率对于事件平均响应时间影响. 具体的,云端实例化 1200 个 DSIs 服务实例,DATs 端启动 4000 个服务实例,对 DATs 上流速率变化为 100,200,300,400,500 观察不同方法下的事件平均响应时间;

③ 验证云端的服务实例数量对于事件平均响应时间影响. 设定接入 4000 个 DATs 端服务实例,设置云端 DSI 服务实例个数从 200,400,800,1200 依次增加,观察不同方法的事件平均响应时间.

实验 2 中,由于事件传输过程涉及到的数据量很小,且本文所在的环境为内部网络,因此,此阶段不考虑事件的传输时间.

(3) 实验 3. 与经典的调度算法的对比实验

借鉴文献[25]中实验比对方案,可知先到先得(FCFS)是经典的调度算法,FCFS 根据他们的到达时间执行服务请求. 本环节的实验设置与实验 2 中相同. 对比本文方法与基于 FCFS 算法实现的云-端集成架构. 本文从实验数据集中选取了“2017-06-01 00:00:00”到“2017-06-30 23:59:59”时间内的电能质量数据模拟流数据. 为提高实验结果的客观性,所有的实验重复了 10 次,取平均值作为实验结果.

以上所设置的三个实验中的每个实验案例所用的实验数据均从数据集中随机选取 20 个 DATs 的电能质量数据作为数据源,KEPServerEX 按需模拟流数据的发送. 同时选取四类主要干扰源类型下的电能质量扰动事件作为 DEEs 服务所输出的扰动事件. 表 3 给出了其具体的事件类型.

表 3 电能质量扰动事件

干扰源类型	电能质量扰动事件
电铁	负序电压不平衡度越限 2 次谐波电压越高限 7 次谐波电压越高限 三项有功功率越限 频率低越限告警
风电	电压负序不平衡度越限 长时间闪变值越限 2 次谐波电压越低限 电压暂降
光伏	长时间闪变值越限 电压有效值越低限 三项有功功率值越低限 4 次谐波电压越高限 6 次谐波电压越高限
冶炼	电流负序不平衡度越限 3 次谐波电压上偏差越限 短时间闪变值越限

计算机学报

5.3　实验结果与分析

(1) 实验 1 结果及分析

表 4 到表 6 给出了实验 1 的数据传输量的实验结果.

表 4　接入不同 DAT 终端数量时的数据传输量

端个数	数据量	
	传统流处理方法/GB	DANCE/MB
5	29.647	45.215
10	57.294	110.430
15	78.941	165.650
20	104.588	220.860

表 5　接入不同端服务数量时的数据传输量

端服务个数	数据量	
	传统流处理方法/GB	DANCE/MB
1000	26.57	25.85
1500	57.75	69.48
2000	75.45	125.42
4000	98.87	186.12

表 6　不同流数据速率的数据传输量

流度/(条/s)	数据量	
	传统流处理方法/GB	DANCE
100	98.870	186.12 MB
200	166.438	289.48 MB
300	293.258	581.84 MB
400	372.242	789.35 MB
500	474.540	1.36 GB

实验结果表明，不管是改变 DAT 的数量、端服务实例的数量，还是改变 DAT 上的流数据速率，DANCE 所传输的数据量是传统流数据处理的 0.16%，远远低于传统流数据处理方法，而且，随着 DAT 的不断增加，二者之间的差距也不断加大. 主要是因为 DANCE 集成了云与端，不需要在网络中传输大量的原始流数据，只有当 DATs 上运行的 DEEs 服务实例有事件产生时，才会把事件涉及到的部分传感数据在网络中进行传输. 实验结果证明了 DANCE 能够有效提升原始流数据的价值密度，减少无价值的流数据在网络中的传输量，减轻网络的压力.

图 4 给出了接入不同的端服务数量时，不同方法下事件平均响应时间. 图 4 实验结果表明，面对不同的端服务数量时，DANCE 要比传统的流数据处理方法的响应时间平均减少 60.8%.

图 5 给出了不同的 DATs 流数据速率时的事件平均响应时间. 图 5 中的结果表示，不同流速下，DANCE 响应时间要比传统的流数据处理方法的响应时间平均减少 74.2%.

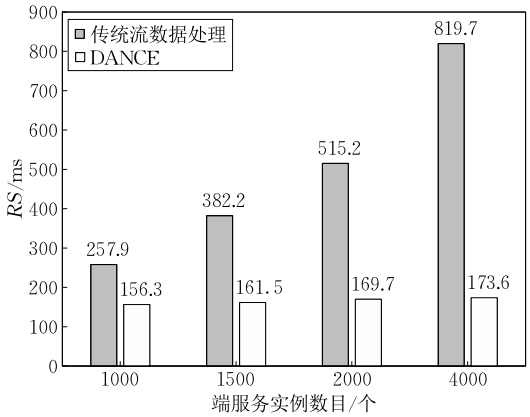

图 4　不同端服务数量的事件平均响应时间

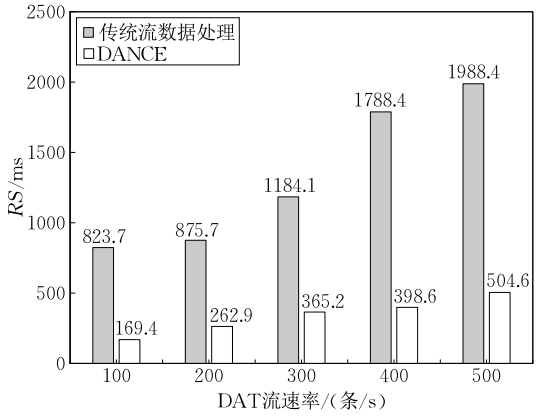

图 5　流数据不同速率时事件平均响应时间

图 6 给出了不同 DATs 数量时的事件平均响应时间. 图 6 表明接入不同 DAT 数量下，DANCE 响应时间要比传统的流数据处理方法的响应时间平均减少 70.6%.

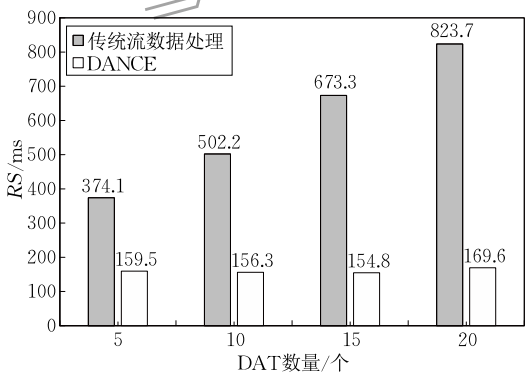

图 6　不同 DAT 数量下的事件平均响应时间

这主要是因为两个方面，一是由于 DANCE 不需要传输大量的数据，因此数据传输对于网络带宽的占用率低，降低网络拥堵的可能性，有助于降低事件的响应时间. 另一方面，主要是 DANCE 中的动态服务适配方法，能够动态调度 DSI 服务实例来处理 DEE 服务实例，提高了 DSI 的利用率，降低了事件

的等待时间.而传统流处理方法需要依赖于云与端之间建立长连接,才能使得流数据不断的被传输到云端,当 DATs 的服务实例超过云端的 DSI 服务实例时,会使得部分的 DATs 端的 DEE 服务无可用的 DSI 服务实例.实验结果表明,面对动态运行环境中,DANCE 能够有效地为不断增加的端服务适配有限的云服务实例,降低事件的平均响应时间.

(2)实验 2 和实验 3 结果及分析

由于实验 2 和实验 3 中得实验对比设置方案一致,因此,其实验结果均统计在图 7 到图 9 中.

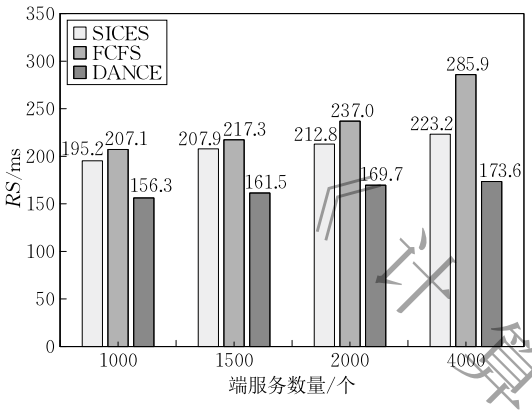

图 7 不同端服务数量的事件平均响应时间

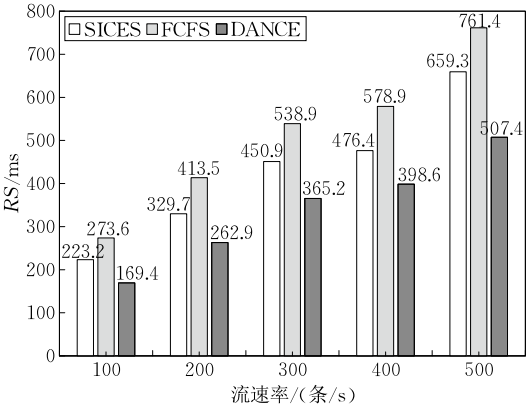

图 8 不同流速时的平均响应时间

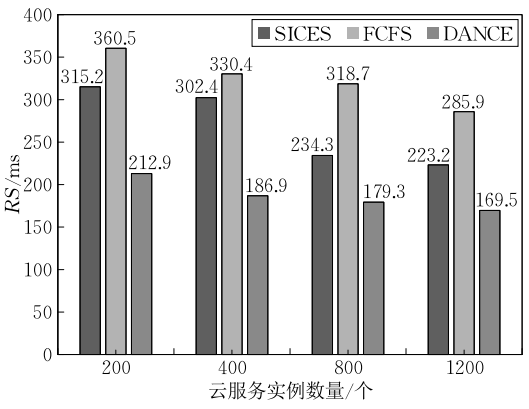

图 9 不同云服务实例数量时的事件平均响应时间

图 7 给出了改变 DATs 端服务实例数量时不同方法的事件平均响应时间.从图中可以看出 DANCE 的平均响应时间比 SICES 的平均响应时间平均降低了 21.2%,比 FCFS 的平均响应时间平均降低了 29.5%.

图 8 给出了改变 DATs 终端的流数据速率时不同方法的事件平均响应时间.从图中可以看出 DANCE 的平均响应时间比 SICES 的平均响应时间平均降低了 20.55%,比 FCFS 的平均响应时间平均降低了 34.25%.

图 9 给出了增加云端的服务实例数量时不同方法的事件平均响应时间.从图中可以看出 DANCE 的平均响应时间比 SICES 的平均响应时间平均降低了 29.55%,FCFS 的平均响应时间平均降低了 42.2%.

图 7 到图 9 的实验结果表明,对于不同设置下的事件平均响应时间,DANCE 都要低于 SICES 和 FCFS.相对于 SICES,DANCE 平均减少 23.51%的事件平均响应时间,最大可减少 38.2%的平均响应时间.相对于 FCFS,DANCE 平均减少 35.23%的事件平均响应时间,最大可减少 43.74%的平均响应时间.这是因为 SICES 在调度云服务实例时,可能会造成端服务实例的请求冲突,影响基于二分图的服务匹配过程,降低匹配效率.DANCE 利用排队论模型,使二分图的匹配保持在一个稳定的状态,从而降低了由于竞争冲突而引起的等待时间.此外利用排队可以预测关键节点的等待时间,提前优化安排服务适配方案,提升匹配的效率.

以上的实验环节中,通过改变端的数量、云服务实例的数量以及流数据产生的速率模拟了动态运行环境下所面临的不确定性问题.本文分析实验的中间结果发现,本文所提出的 DANCE 在动态环境下,能够使得事件的平均响应时间维持在一定的范围内.因此,DANCE 能够有效应对云-端环境下的服务适配不确定性问题,提升流数据的价值密度,降低网络的数据传输压力,降低事件的平均等待时间,提高云-端无缝集成的能力.

实际上,第 2 节中的应用场景和平台是基于 SICES 来设计和实现的,而 DANCE 是对 SICES 的优化改进.因此,为了验证 DANCE 在实际应用中的使用效果,本文利用国家电网的 PSCAD 仿真系统,首先基于 SICES 搭建了第 2 节中的应用场景平台,其次将本文所提出的 DANCE 应用到其上,在仿真

《计算机学报》

平台进行试运行.并根据实验 2 中的实验设置仿真了不同的动态变化情景.

对监测到的电能质量扰动事件的平均响应时间进行统计可知：当改变端服务的数量时，SICES 下的系统平均事件响应时间为 216.4 ms，最大为 229.6 ms，最小为 220.2 ms；DANCE 的扰动事件平均响应时为 164.2 ms，最大为 174.0 ms，最小为 158.8 ms. 当改变流的速率时，SICES 下的系统事件平均响应时间为472.1 ms，最大为729.3 ms，最小为 246.8 ms；DANCE 的事件平均响应时间为 376.9 ms，最大为561.2 ms，最小为187.3 ms. 当改变云服务实例的数量时，SICES 下的系统事件平均响应时间为 296.6 ms，最大为 347.8 ms，最小为 246.3 ms；DANCE 的事件平均响应时间为 193.8 ms，最大为 206.7 ms，最小为 187.1 ms. 通过对中间结果的计算可知，相对于 SICES，应用了 DANCE 之后，仿真系统中对扰动事件的平均响应时间降低了 24.3%，最大能够降低 38.2%. 仿真结果表明，DANCE 具有实际应用价值，能够在一定程度上降低应用系统对扰动事件的响应时间.

6　相关工作

6.1　边缘计算与 IoT 服务

随着 IoT 的快速发展，很多研究者针对 IoT 提出了各自需求引导下的 IoT 服务. Xiao 等人[9]针对 IoT 中的异构数据导致数据共享效率低的难题，提出了一种可互操作的基于服务的边缘端数据信息发布方法，可以根据服务需求进行自适应数据共享来提高服务级互操作性和功能级互操作性. Xu 等人[26]以及 Perera 等人[27]提出 IoT 数据资源作为服务，使得跨平台的数据可以通过 Web 或者 IoT 应用中的 URI 被访问到. 本文所在团队在前期的工作[5,28]中，针对智能交通和工厂设备流数据，提出了主动式数据服务模型，该服务模型能够把 IoT 中的传感设备通过数据接入的方式映射成一个可编程的“软传感”，能够形成更大粒度数据服务. 然而，这些仍然是对数据的封装，仍然需要数据集中式存储，没有考虑把数据的处理迁移到云端执行.

得益于边缘计算的提出与发展，许多研究人员开始尝试从云端迁移部分的计算任务到边缘设备上执行，以期望能够降低数据的网络传输量，从而实现数据分析的低延迟需求. Morabito 等人[29]设计了一个边缘计算平台，利用轻量级的容器实现资源的弹性化. 基于一个独立的平板电脑作为网关来部署和运行某些例如数据压缩和数据处理的功能. Petrolo 等人[30]专注于无线传感器网络(WSN)的网关设计，来优化边缘端的通信和使用. 网关可以管理类似语义的东西，并作为一个边缘终端向用户提供数据. Xu 等人[31]提出了一个可扩展的分析服务 EAaaS，其目标是提供一种基于规则的编程，来应对那些缺乏灵活和统一方法实现特定领域的分析逻辑问题，提升物联网场景中数据处理的实时性和处理效率. 但是它是一种基于规则的分析服务，仍具有一定的局限性，无法应对大规模流数据下的数据驱动下的流数据应用.

上述工作基于边缘计算设计实现了边缘计算的平台或者服务，能够在边缘端完成对数据的预处理，但是没有考虑到如何集成云与端能力的问题.

6.2　云-端集成方法

渗透计算(Osmotic Computing)[32]是一种新的范式，支持在网络边缘有效地执行物联网(物联网)服务和应用程序. 这种范式建立在需要一个整体的分布式系统抽象的基础上，这使得在网络边缘的资源受限的物联网平台上部署轻量级微服务，再加上在大型数据中心运行的更复杂的微服务. 这种模式是由网络边缘的资源能力的显著增加所驱动的，同时支持数据传输协议，使这些资源能够更无缝地与基于数据中心的服务进行交互.

Varghese 等人[33]提出了一个边缘即服务的(EaaS)平台，旨在将网络的边缘整合到计算生态系统中. EaaS 使用树莓派作为边缘节点，基于一种轻量级的发现协议，边缘节点可以在平台中被访问. 同时实现了一个可伸缩的资源供应机制，可以将工作负载从云中卸载到边缘，以满足多个用户请求. Kaur 等人[34]认为任务调度将在集成物联网与边缘计算架构中发挥关键作用，使用轻量级的容器即服务(CoaaS)进行任务选择和调度，利用合作博弈理论来应对任务选择和调度问题. 同时设计了一个多目标函数，通过考虑不同的约束条件，如内存、CPU 和用户的预算，来减少能源消耗. Renart 等人[35]提出一个基于边缘的编程框架，该框架具有位置感知机制，能够获取与客户端和数据源位置相关的信息，并在基础设施的边缘无缝、高效地分配流计算. Wang 等人[36]提出了一种新型的云计算架构，主要包括云部分和边缘部分. 云用于处理大规模、长期的全局数据，这些数据可用于获取决策信息，如特征或规则集. 边缘端用于处理小型的、短期的本地数据，用于显示实时情况. 此外，根据所获得的数据特征、

《计算机学报》

规则集和在边缘端获得的本地高质量数据,同时提供高质量的个性化服务.然而,其没有考虑物联网场景中流数据驱动下动态产生的事件流的处理任务.

上述工作尝试了集成云基础设施与边缘设备端来实现对数据的处理.但是上述工作面对的端与云之间的协作都是静态协作的方法,即提前预定义好端与云的协作方法.不适应于本文中研究场景中云服务于端服务之间的适配时机的动态性的问题.

6.3 服务调度

目前,关于 SOA 服务调度已经有许多的研究工作,近年来的关注主要在于如何通过建立基于获益和感知的调度策略,使得服务调度更加科学[37-40].按照不同的获益目标,也有不同的调度策略.经过分析发现,许多的服务调度相关的工作都采用了排队论的工作.表 7 对现有的服务调度相关工作进行归纳.

表 7 SOA 服务调度相关工作

文献	方法特征	缺点
Leitner 等人[41]	文章指出并不是所有的客户都有相同的协议,也不是所有的请求都需要相同数量的服务资源,因此,其根据用户不同的协议对云中的虚拟计算资源进行最优调度,以使资源支付和因违反服务水平协议而造成的损失之和最小化.在执行调度策略之前,其利用排队队列对请求的到达、处理以及等待时间进行形式化描述.	文章没有考虑用户协议的动态变化性.
Liu 等人[42]	文章研究了云环境下的服务实时调度问题,定义了获益函数和惩罚函数,对于每个任务,只有当获益函数值大于惩罚函数值时,这个任务才能被调度执行.同时,如果任务已经运行的时间越长,则惩罚函数值越大,惩罚函数值超过临界点时将被停止运行.调度的过程中根据到达任务的期望效用,选择高利润的任务作为就绪队列,并且积极地删除那些可能导致巨大损失的任务.	文章未进一步阐述证明其所定义函数的合理性,无法验证其是否能够适用于大规模分布式物联网中的流数据处理.
Chen[43]	文章为解决分布式 SOA 系统中的服务实时调度问题,对比了基于截止日期和基于优先级的服务调度方法.服务调度程序维护三个关键队列:一个简单的队列缓冲来自服务管理器的已批准请求,一个用于实时服务请求的实时队列;以及基于非实时服务请求的优先级队列.	文章没涉及被调度的服务如果需要请求调度其他服务的复杂场景.
Wang 等人[44]	文章提出了一种混合了基于优先级和先到先服务的服务调度方法,任务按照到达顺序进行排队,等待调度策略的处理,解决了单个服务调度策略所存在的资源浪费问题.	文章没有考虑流环境下的排队队长过长的问题.

虽然典型的服务策略根据各自的需求都能够取得一定的获益,但是这些典型的服务调度策略无法直接适用于本文所面临的云服务与端服务之间的适配问题.典型的服务调度策略所面临的是云环境中的服务调度问题,本文所面临的是云-端集成环境下的服务适配问题,其中端设备具有很强的自主计算能力,使得云服务与端服务的适配时机具有更加复杂的不确定性因素.此外,本文还需要考虑在大规模高速流数据以及应用系统实时响应下,如何动态调度有限的云服务实例来应对不断增长的端服务实例的适配请求.

还有部分研究人员提供了一些针对特定需求的调度算法.Zhou 等人[45]提出了一种基于分层 TBTree (hTBTree)的调度数据结构,用于管理面向实时服务架构(RT-SOA)应用程序中的无序服务请求.Chen 等人[46]分析了大规模服务计算系统的节能请求调度和服务管理中面临的挑战:服务请求到达的动态性和难以预测性.共同考虑能效,性能和队列拥塞的指标,将请求调度,服务管理和 DVFS 三种方法结合起来,提出了一种基于李雅普诺夫优化技术的分布式在线调度和管理算法,该算法在限制队列长度的同时可以获得接近最优的系统利润.

随着边缘计算的发展,也有部分的研究者关注移动边缘云计算中的请求调度问题.Chen 等人[47]针对在移动端边缘计算系统(MEC)中服务请求模式的动态特性和不确定性,基于李亚普诺夫优化技术,提出了一个动态服务请求调度(DSRS)算法.Wu 等人[48]为平衡边缘服务器之间的负载以最小化总体响应时间,结合遗传算法和模拟退火算法提出了一种新的启发式方法(GASD),通过联合进行请求调度和服务调度来减少服务请求的总体响应时间.还应用了一种求解组合算法来降低该方法的计算复杂度.Liu 等人[49]提出了一种用于移动边缘计算系统的延迟最优计算任务调度.通过分析每个任务的平均延迟和移动设备的平均功耗,他们提出了一种有效的一维搜索算法来找到最优的任务调度策略.

上述工作的调度算法一定程度上为本文的调度算法的设计与实现提供了有价值的参考.然而,上述工作关注的问题移动边缘计算中如何把计算请求均衡分布在移动边缘设备上,以降低用户的服务请求响应时间.本文关注的是如何动态调度云端的服务来应对不断增加且具有不确定性边缘端服务的适配请求.

7 总　结

本文提出了一种面向云-端动态集成的服务适配方法，来使能云基础设施和端设备间的无缝集成.本文主要解决云-端服务适配时遇到的适配时机不确定以及云-端服务非完全适配难题.基于前期提出的主动式服务模型和云-端无缝集成方法的基础上，将云服务实例和端服务实例之间的适配问题建模为二分图节点之间的动态匹配问题，同时结合排队论中的 M/M/c/∞模型对二分图最大权最优匹配 Kuhn-Munkres 算法进行了优化改进，保障适配过程中端服务实例的全局平均请求响应时间最小.最后，基于真实的电能质量监控案例和数据，验证了本文方法的有效性.

参　考　文　献

[1] He B, Yang M, Guo Z, et al. Comet: Batched stream processing for data intensive distributed computing//Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis, USA, 2010: 63-74

[2] Ahmed A, Ahmed E. A survey on mobile edge computing//Proceedings of the 10th IEEE International Conference on Intelligent Systems and Control. Coimbatore, India, 2016: 1-8

[3] Lopez P G, Montresor A, Epema D, et al. Edge-centric computing: Vision and challenges. ACM SIGCOMM Computer Communication Review, 2015, 45(5): 37-42

[4] Huhns M N, Singh M P. Service-oriented computing: Key concepts and principles. IEEE Internet Computing, 2005, 9(1): 75-81

[5] Han Y, Liu C, Su S, et al. A proactive service model facilitating stream data fusion and correlation. International Journal of Web Services Research, 2017, 14(3): 1-16

[6] Cheng B, Zhu D, Zhao S, et al. Situation-aware IoT services coordination platform using event driven SOA paradigm. IEEE Transactions on Network & Service Management, 2016, 13(2): 1

[7] Ryden M, Oh K, Chandra A, et al. Nebula: Distributed edge cloud for data intensive computing//Proceedings of the 2014 International Conference on Collaboration Technologies and Systems. MN, USA, 2014: 491-492

[8] Cai X, Kuang H, Hu H, et al. Response time aware operator placenrent for complex event processing in edge computing//Proceedings of the 16th International Conference on Service-Oriented Computing, Hangzhou, China, 2018: 264-278

[9] Xiao B, Rahmani R, Li Y, et al. Edge-based interoperable service-driven information distribution for intelligent pervasive services. Pervasive and Mobile Computing, 2017, 40: 359-381

[10] Zhang S, Liu C, Han Y, et al. Seamless integration of cloud and edge with a service-based approach//Proceedings of the 2018 IEEE International Conference on Web Services. San Francisco, USA, 2018: 155-162

[11] Urgaonkr R, Wang S, He T, et al. Dyncanic Service Migration and Workload Scheduling in Edge-Clouds. Performance Evaluation, 2015, 91: 205-228

[12] Lee J H, Park J M. User-centric real time service scheduling for robots//Proceedings of the 2011 IEEE International Conference on Robotics & Biomimetics. Karon Beach, Thailand, 2011: 1024-1028

[13] Dargahi F, Bhuiyan M F H, Wang C, et al. Prioritizing and scheduling service requests under time constraints//Proceedings of the 2014 IEEE International Conference on Services Computing. Anchorage, USA, 2014: 1-8

[14] Lee Z, Wang Y, Zhou W. A dynamic priority scheduling algorithm on service request scheduling in cloud computing//Proceedings of the International Conference on Electronic and Mechanical Engineering and Information Technology. Harbin, China, 2011: 4665-4669

[15] Xue Yu-Sheng, Lei Xing, Xue Feng, et al. A review on impacts of wind power uncertainties on power systems. Proceedings of the Chinese Society for Electrical Engineering, 2014, 34(29): 5029-5040(in Chinese)
(薛禹胜，雷兴，薛峰等. 关于风电不确定性对电力系统影响的评述. 中国电机工程学报，2014, 34(29): 5029-5040)

[16] Daagi M, Ouniy A, Kessentini M, et al. Web service interface decomposition using formal concept analysis//Proceedings of the IEEE International Conference on Web Services. Honolulu, USA, 2017: 172-179

[17] Zu Quan, Zhang Miao-Miao, Liu Jing. Dynamic weight matchings in left weighted convex bipartite graphs. Chinese Journal of Computers, 2016, 39(11): 2388-2402(in Chinese)
(祖佺，张苗苗，刘静. 左侧带权凸二分图动态权值匹配. 计算机学报，2016, 39(11): 2388-2402)

[18] Plaxton C G. Fast scheduling of weighted unit jobs with release times and deadlines//Proceedings of the International Colloquium on Automata, Languages and Programming. Reykjavik, Iceland, 2008: 222-233

[19] Guizani M, Rayes A, Khan B, et al. Network Modeling and Simulation: A Practical Perspective. Chichester, UK: John Wiley & Sons, 2010

[20] Alla H B, Alla S B, Touhafi A, et al. A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment. Cluster Computing, 2018, 21(4): 1797-1820

[21] Marini F, Walczak B. Particle swarm optimization (PSO). A tutorial. Chemometrics and Intelligent Laboratory Systems, 2015, 149(B): 153-165

《计算机学报》

[22] Azad A, Buluc A, Pothen A. A parallel tree grafting algorithm for maximum cardinality matching in bipartite graphs// Proceedings of the Parallel and Distributed Processing Symposium. Hyderabad, India, 2015: 1075-1084

[23] Riesen K, Bunke H. Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing, 2009, 27(7): 950-959

[24] Munshi A A, Mohamed Y A I. Cloud-based visual analytics for smart grids big data//Proceedings of the Innovative Smart Grid Technologies Conference. Minneapolis, USA, 2016: 1-5

[25] Wu H, Deng S, Li W, et al. Revenue-driven service provisioning for resource sharing in mobile cloud computing// Proceedings of the International Conference on Service-Oriented Computing. Malaga, Spain, 2017: 625-640

[26] Xu B, Xu L D, Cai H, et al. Ubiquitous data accessing method in IoT-based information system for emergency medical services. IEEE Transactions on Industrial Informatics, 2014, 10(2): 1578-1586

[27] Perera C, Talagala D S, Liu C H, Estrella J C. Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in IoT clouds. IEEE Transactions on Computational Social Systems, 2015, 2(4): 171-181

[28] Han Y, Wang G, Yu J, et al. A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in China. IEEE Transactions on Intelligent Transportation Systems, 2015, 1(9): 2648-2657

[29] Morabito R, Beijar N. Enabling data processing at the network edge through lightweight virtualization technologies//Proceedings of the IEEE International Conference on Sensing, Communication and Networking. London, UK, 2016: 1-6

[30] Petrolo R, Morabito R, Loscrì V, et al. The design of the gateway for the Cloud of Things. Annals of Telecommunications, 2016, 72(1-2): 1-10

[31] Xu X, Huang S, Feagan L, et al. EAaaS: Edge analytics as a service//Proceedings of the IEEE International Conference on Web Services. Honolulu, USA, 2017: 349-356

[32] Villari M, Fazio M, Dustdar S, et al. Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Computing, 2016, 3(6): 76-83

[33] Varghese B, Wang N, Li J, et al. Edge-as-a-service: Towards distributed cloud architectures. Advances in Parallel Computing, 2017, 32: 784-793

[34] Kaur K, Dhand T, Kumar N, et al. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog Nano data centers. IEEE Wireless Communications, 2017, 24(3): 48-56

[35] Renart E G, Diaz-Montes J, Parashar M. Data-driven stream processing at the edge//Proceedings of the IEEE, International Conference on Fog and Edge Computing. Madrid, Spain, 2017: 31-40

[36] Wang X, Yang L T, Xie X, et al. A cloud-edge computing framework for cyber-physical-social services. IEEE Communications Magazine, 2017, 55(11): 80-85

[37] Guan He-Qing, Zhang Wen-Bo, Wei Jun, et al. An application-aware Web service requests scheduling strategy. Chinese Journal of Computers, 2006, 29(7): 1189-1198(in Chinese) (官荷卿, 张文博, 魏峻等. 一种应用敏感的 Web 服务请求调度策略. 计算机学报, 2006, 29(7): 1189-1198)

[38] Zhou A, Sun Q, Sun L, et al. Maximizing the profits of cloud service providers via dynamic virtual resource renting approach. EURASIP Journal on Wireless Communications & Networking, 2015, 2015(1): 71

[39] Li Z, Ge J, Hu H, et al. Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. IEEE Transactions on Services Computing, 2018, PP(99): 713-726

[40] Mei J, Li K, Ouyang A, et al. A profit maximization scheme with guaranteed quality of service in cloud computing. IEEE Transactions on Computers, 2015, 64(11): 3064-3078

[41] Leitner P, Hummer W, Satzger B, et al. Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud//Proceedings of the IEEE, International Conference on Cloud Computing. Honolulu, USA, 2012: 213-220

[42] Liu S, Quan G, Ren S. On-line scheduling of real-time services for cloud computing//Proceedings of the 2010 6th World Congress on Services. Miami Florida, USA, 2010: 459-464

[43] Chen Y. A service scheduler in a trustworthy system// Proceedings of the Simulation Symposium. Arlington Virginia, USA, 2004: 89

[44] Wang Z, Xing W, Chen B. On-line service scheduling. Journal of Scheduling, 2009, 12(1): 31-43

[45] Zhou S, Lin K J. A flexible service reservation scheme for real-time SOA//Proceedings of the International Conference on E-Business Engineering. Beijing, China, 2011: 215-222

[46] Chen Y, Lin C, Huang J, et al. Energy efficient scheduling and management for large-scale services computing systems. IEEE Transactions on Services Computing, 2017, 10(2): 217-230

[47] Chen Y, Zhang Y, Chen X. Dynamic service request scheduling for mobile edge computing systems. Wireless Communications and Mobile Computing, 2018, 1324897: 1-10

[48] Wu H, Deng S, Li W, et al. Request dispatching for minimizing service response time in edge cloud systems//Proceedings of the 27th International Conference on Computer Communication and Networks. Hangzhou, China, 2018: 1-9

[49] Liu J, Mao Y, Zhang J, et al. Delay-optimal computation task scheduling for mobile-edge computing systems//Proceedings of the IEEE International Symposium on Information Theory. Barcelona, Spain, 2016: 1451-1455

ZHANG Shou-Li, Ph. D. candidate. Her research interests include services computing, streaming data integration and analysis.

LIU Chen, Ph. D., associate professor. His research interests include services computing, streaming data integration and analysis of large-scale stream data.

HAN Yan-Bo, Ph. D., professor, Ph. D. supervisor. His research interests include distributed systems, business process management and collaboration, services computing, streaming data integration and analysis of large-scale stream data.

LI Xiao-Hong, Ph. D., professor, Ph. D. supervisor. Her current research interests include knowledge engineering, trusted computing, and security software engineerin.

Background

In IoT, large-scale sensor data are continually generated in the form of data stream. These data are usually infinite, noisy, of low-value-density, and orderless on a broad panorama. Although having powerful potential, cloud computing has faced increasing computational challenges like performance, cost, energy consumption, and service qualities because of the exponential growth of big IoT data. On the other side, the edge computing has emerged and attracts broad attentions. It stresses sensor data and its processing should be put close to the edge equipment, which can greatly relieve the burdens of the cloud server. But the edge equipment usually has very limited computing power as well as storage ability. To handle the processing of the big real-time IoT data, a key is to seamlessly integrate the power of cloud server as well as large-scale edge equipment.

As the service-oriented paradigm has been a mainstream approach for building large-scale distributed software systems. The service-based integration of cloud and edges has attracted many attentions. However, there are some challenges because of the uncertainty of edge service request arrivals as well as incomplete matching. Firstly, compared with the cloud infrastructure, edge equipment can join or quit at any time due to their own flexibility. Secondly, the capability of the edge equipment will constantly change with use. It will make the state of the edge service change dynamically. Finally, the moment of event-driven adaptation request for edge service is uncertain in the context of the large-scale streaming data, it is because that the generation and arrival of events on edge equipment are uncertain.

Although there are many typical service scheduling algorithms, they cannot be directly applied to the problems faced in this paper. There are three difficulties: (1) the application environment of the typical scheduling algorithm is mostly the workflow scheduling in the local area network or cloud environment, while this paper is faced with the cloud-edge integration environment in the wan. (2) Most of the data involved in a typical scheduling algorithm is the process control information or input/output information of services. However, this paper faces the large-scale streaming data in the physical network. (3) IoT applications require real-time response for the service adaptation request of the edge device.

Thus, the service adaption in the context of dynamic integration needs to grasp the right moment of adaption to face the challenges of uncertainty of edge service request arrivals as well as incomplete matching. Based on the proactive data service model, this paper proposes an event-driven service adaption approach for dynamic adaption for the dynamic integration of cloud and edge. The main contributions include: transforming the uncertain service adaptation problem into the improved maximal weight matching model in a dynamic bipartite graph. The M/M/c/∞ model in the queuing theory is modified to optimize the Kuhn-Munkres algorithm to minimize the average response time of the request of edge service.

This work is supported by "National Key R&D Plan (No. 2018YFB1402500), Research on the Theory and Technology of Internet of Services". And "National Natural Science Foundation of China (No. 61672042), Models and Methodology of Data Services Facilitating Dynamic Correlation of Big Stream Data".

《计算机学报》