基于无线城域网的微云部署及用户任务调度

张建山[1] 林 兵[1,2] 卢 宇[1] 许芙蓉[1]
(福建师范大学物理与能源学院 福州 350117)[1]
(福建省网络计算与智能信息处理重点实验室 福州 350116)[2]

摘 要 移动应用对计算能力的需求越来越大,然而便携式移动设备的计算能力却是有限的。将任务卸载到附近的由计算机组成的微云上,是减小移动设备中程序系统响应时间的有效方法之一。边缘计算使得计算任务在源头附近就能得到及时处理,是减小系统时延的有效方法。微云技术是边缘计算的重要应用。目前,移动微云卸载技术已经有了诸多的研究成果,但是在给定网络中如何部署微云以优化移动应用性能的问题却很少被关注。文中在无线城域网(Wireless Metropolitan Area Network,WMAN)背景下研究微云部署和用户任务调度方案,并设计算法来解决以下问题:在无线城域网中的用户密集区域部署微云,并在各微云负载均衡的条件下调度用户到部署好的微云。最后进行仿真实验,结果表明所提算法是有效、可行的。

关键词 微云部署,边缘计算,用户任务调度,系统响应时间最小化,任务卸载,移动云计算
中图法分类号 TP338 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.06.019

Cloudlet Placement and User Task Scheduling Based on Wireless Metropolitan Area Networks

ZHANG Jian-shan[1] LIN Bing[1,2] LU Yu[1] XU Fu-rong[1]
(College of Physics and Energy,Fujian Normal University,Fuzhou 350117,China)[1]
(Fujian Key Laboratory of Network Computing and Intelligent Information Processing,Fuzhou 350116,China)[2]

Abstract The computing capability requirements of mobile applications are becoming increasingly intensive,while the computing capability of transferable mobile devices is limited. In a mobile device,an effective way to reduce the system response time of an application is offloading its task to nearby cloudlet,which consists of clusters of computers. Edge computing enables computational tasks to be processed in time near the source,which is an effective way to reduce system delay. Cloudlet technology is an important application of edge computing. Although there is a great deal of research in mobile cloudlet offloading technology,there has been very little attention paid to how cloudlets should be placed in a given network to optimize mobile application performance. This paper studied cloudlet placement and mobile user task scheduling to the cloudlet in a wireless metropolitanare network(WMAN). This paper devised an algorithm for the problem,which enables the placement of the cloudlets at user dense regions of the WMAN,and scheduled mobile user to the placed cloudlets which balancing their workload. This paper also conducted experiments through simulation. The simulation results indicate that the proposed algorithm is very promising.

Keywords Cloudlet placements,Edge computing,User task scheduling,System response time minimization,Task offloading,Mobile cloud computing

1 引言

随着移动设备硬件技术的飞速发展,移动应用变得复杂多样,由此产生的任务需求对移动设备的处理能力提出了更高的要求。适宜的尺寸虽然实现了设备的便携性,但无法完全满足移动应用对处理能力的要求。为了提高移动应用的性能表现,一种有效的解决方案是当一个应用承担多个任务时,可将其中一部分卸载到云端上处理,从而减轻移动设备本身的处理负担。遥远的云端服务器一方面因为拥有丰富的计算资源而备受移动用户的青睐;另一方面却因距离移动用户过远,在任务数据传输时产生严重的网络延时,导致用户体验变差,尤其在一些对响应时间要求较高的应用中,过长的网络延

到稿日期:2018-05-17 返修日期:2018-08-18 本文受国家自然科学基金(面上)项目(61672159),福建省工业引导性(重点)项目(2017H0011),福建省中青年教师教育科研项目(JT180098),福建省高等学校应用型学科建设(闽教高[2017]44)资助。
张建山(1995-),男,硕士生,主要研究方向为智能计算;林 兵(1986-),男,博士,讲师,主要研究方向为云计算技术;卢 宇(1974-),男,硕士,教授,主要研究方向为计算机应用技术研究,E-mail:fzluyu@163.com(通信作者);许芙蓉(1994-),女,硕士生,主要研究方向为智能计算。

迟是不允许出现的。最近有研究[1-2]提出用一些由计算机集群组成的微云替代云端来卸载移动设备任务。微云通常部署在网络中的某个数据接收点上，并且用户可以通过无线网络对其进行访问。相较于传统的云端，微云在地理位置上更靠近用户，可以在一定程度上缩短用户与处理端之间的网络延时，从而提升用户体验。

微云技术是边缘计算的重要应用。相较于传统的云计算，边缘计算能更好地支持移动计算，它具有以下明显的优点[3]：

1）极大地缓解网络带宽与数据中心的压力。思科在2015－2020 年全球云指数[4]中指出，2020 年全球的设备将产生 600ZB 的数据，其中 10％是关键数据，剩余 90％都是无需长期存储的临时数据。边缘计算在网络边缘处理大量临时数据，减轻了网络带宽与数据中心的压力。

2）增强服务的响应能力。云计算通过为移动设备提供服务来弥补其在计算能力上的缺陷，但是网络传输速度受限于通信技术的发展，复杂网络环境中更存在链接和路由不稳定等问题，这些因素造成的延时过长、波动过强、数据传输速度过慢等问题严重影响了云服务的响应能力[5]。而边缘计算在用户附近提供服务，近距离服务保证了较小的网络延时，简单的路由也减小了网络的波动，千兆无线技术的普及为网络传输速度提供了保证，这些都使得边缘服务有着比云服务更强的响应能力[6]。

3）保护隐私数据，提升数据安全性。传统云计算模式下对所有数据与应用的操作都在数据中心进行，用户很难对数据的访问与使用进行有效的追踪。而边缘计算则为关键性隐私数据的存储与使用提供了基础设施，将隐私数据的操作限制在防火墙内，从而提升了数据的安全性。

边缘计算中微云应用的研究被日益关注的同时，其在网络中的部署位置在一定程度上被忽略了。以往的研究通常将微云描述为小型的私有局域网，例如公寓或者办公室区域。在这些环境中，微云的部署位置对整个区域网络延时的影响是微乎其微的。无论微云被部署在何处，用户与微云之间的平均网络延时都是极小的，完全可以忽略不计。但是，若将部署的环境背景转变为无线城域网，部署位置的问题将变得极其重要。

虽然关于在无线城域网中使用微云的研究还很少，但是微云对无线城域网环境具有一定的适应性。首先，城市区域的人口密度高，这意味着微云可以接收到大量用户的任务需求，被闲置的概率较低，从而可以保障微云的利用率。其次，无线城域网的规模足够庞大，通过无线城域网提供微云服务的供应商可以降低部署的平均成本，使得微云服务更容易被普通大众所接受。然而，也正是因为无线城域网的规模巨大，所以某个用户与距离他最近的微云之间可能相距多个接收点。虽然在小型网络中每个接收点之间的延时可以忽略不计，但是在大规模的无线城域网中，由于较远的地理距离以及繁重的任务量，相距较远的接收点之间的数据传输可能会产生较低质量的服务以及较大的网络延时。因此，用户与为其提供服务的微云之间的距离将会严重影响移动应用的性能表现，特别是对于那些数据通信率与处理率高的应用，例如移动在线游戏。同时，我们也应当谨慎考虑哪一个用户需要调度到哪一个微云的问题。当用户被调度到最近的微云时可能有最小的网络延时，但是如果这个微云被其他用户的任务请求满载，那么此时的网络延时就会产生很大的波动。对此，目前最好的解决方案就是将用户任务调度到距离较近且工作负载较轻的微云上执行。设计微云部署方案以及用户任务调度方案，旨在最小化用户与微云之间总的任务请求延时，以最大程度地提升移动应用的性能表现，从而提升用户体验。

无线城域网中的微云部署和用户任务调度问题仍存在多方面的困难。首先，无线城域网中的用户往往不是静止的，他们常在整个城域网中移动，任何特定区域内的用户数量都可能随时间的推移而变化，决定微云的部署位置去适配用户的动态移动以及资源要求是很大的挑战。其次，用户任务调度的方案也是必须考虑的，结合寻找最佳的微云部署方案来执行最佳的用户任务调度方案，给本已困难的问题又增加了新的复杂度。

本文将对微云部署方案以及用户任务调度方案的优化问题展开研究，旨在减少被卸载任务的平均等待时间。我们专注于解决以下优化问题：根据给定的整数 $K\geqslant 1$，在无线城域网中选取 K 个接收点来部署 K 个微云，然后按一定的规则将无线城域网中的用户任务调度到微云上，以最小化用户被卸载任务的平均等待时间。

2　相关工作

将移动任务卸载到微云是一个新的研究热点。微云被定义为网络内部署在数据接收点上的计算机集群，它有着丰富的计算资源并且常被作为移动用户任务卸载的目的地[1-2,7]。因为传统的云端服务器往往距离用户较远，所以二者间的传输延时很大，对于即时性要求较高的应用场景来说是不可接受的[1]。系统响应时间对于时间敏感度较高的应用的用户体验起着决定性作用，例如增强现实应用程序和移动游戏系统。与传统的云端相比，微云在地理位置上更加贴近用户，它与用户之间的传输延时更小，从而有助于提升应用的用户体验。为了卸载任务，移动用户将任务封装在虚拟机中[8]，然后卸载到微云上执行。微云对封装在虚拟机中的任务处理完成后，执行结果将返回给用户。

现有的任务卸载框架和算法大多面向传统的云端[9-10]，但微云作为一个可选的任务卸载目的地也逐渐受到认可[11-15]。Odessa[14]就是一个具体的案例，它既可以将任务卸载到传统的云端，也可以将任务卸载到微云。也有一些研究者[16]进一步研究了同时将任务卸载到传统云和微云的问题，并基于博弈理论提出解决方案。微云也对移动云游戏的研究产生了一定影响[17-18]。例如，有研究[18]提出了一种微云辅助下的多人云游戏系统，该系统使用微云作为视频帧的高速缓存容器。

最近也有工作扩展了微云的定义，将点对点模式的计算机纳入网络中。Verbelen 等[19-20]提出了一种新的微云框架，该框架可以在网络中以点对点的形式发现附近的设备并分享资源。虽然这种体系结构确实有其优点，但是将任务卸载到点对点模式的计算机时将会产生严重的安全性与隐私问题，

因此本文将微云的定义限制为与数据接收点相连接的计算机集群。

本文将研究无线城域网环境下的微云部署问题。无线城域网是一个计算机网络,它为网络内的所有用户提供了全覆盖的无线网络服务[21]。直观看来,微云放置问题与设施放置问题类似[22],但二者在本质上是不同的。无线城域网中的用户在没有微云的情况下可以自给自足,因为他们可以将任务卸载到传统的云端,这使得传统的设施放置算法[23]很难有效地应用于微云部署问题。综上所述,本文将研究能有效应用于微云部署问题的优化算法。

3 问题定义

首先构建一个无线城域网系统模型以及这个系统模型的一种任务卸载方案;然后形式化定义无线城域网背景下的微云部署问题。

3.1 无线城域网系统模型

一个无线城域网系统可以由一个通过互联网互相连接的接收点(Access Point,AP)集合 $P=\{p_1,\cdots,p_m\}$ 和一个可以通过接收点访问网络的用户集合 $U=\{u_1,\cdots,u_n\}$ 来表示。用一个无向图 $G=(V,E)$ 来表示无线城域网中的用户与接收点之间的关联关系,其中 $V=P\cup U$。G 含有两种类型的边:一种为某个用户 u_i 与某个接收点 p_j 之间的边 (u_i,p_j),其表示 u_i 与 p_j 之间无线连接;另一种为两个接收点 p_i 与 p_j 之间的边,它表示两个接收点直接相连,即它们之间没有其余的接收点。假设图 G 是连通的,这就意味着 G 包含的任意一个接收点都通过高速的互联网访问另外任意一个接收点。此外,G 中的每一个接收点都可以通过互联网访问遥远的云端服务器。

每个移动用户产生的任务量是波动的并且无法预知,尤其是当他们在同一时间运行多个应用时。我们假定每个用户 u_i 都有一束可卸载的任务流,并且这束工作流根据泊松过程以卸载率 λ_i 随机进入系统。

为了将自己的任务需求卸载到微云上执行,用户需要通过网络 G 来传递他的任务需求。用 ω_i 来表示用户 u_i 与同他无线连接的接收点 p_j 间的无线延时。如果用户 u_i 的任务需求被调度到部署在 p_k 的微云上执行,那么任务就需要从 p_j 传到 p_k。假定被卸载的任务都由若干个单位任务组成,不同任务在同一组接收点之间传输时产生的延时由其大小决定。定义一个矩阵 $\boldsymbol{D}\in R^{m\times m}$,其中 $D_{j,k}$ 表示任务在接收点 p_j 与接收点 p_k 之间传输所产生的传输延时。

3.2 卸载系统模型

我们引入一个多用户任务卸载系统模型,这个模型模拟一个队列网络。假定 G 中部署了 K 个微云,被卸载的任务可以由 K 个微云中的一个执行或者由遥远的云端执行。每个用户根据卸载率 λ_i 卸载他的任务流到微云。如果某一时刻微云出现过载的情况,它会将接收到的工作流的一部分卸载到遥远的云端执行,以保证微云上的负载不会过大。

所有的微云被模拟成一个 $M/M/c$ 队列,其中每个微云都由 c 个拥有固定服务率 μ 的同类服务器组成。一个任务请求到达微云的等待时间由队列时间和传输时间组成。我们定义一个函数 f_Q,它的功能是根据给定的任务卸载率 λ(工作负载)返回平均队列时间。

$$f_Q(\lambda)=\frac{C(c,\frac{\lambda}{\mu})}{c\mu-\lambda} \tag{1}$$

其中:

$$C(c,\rho)=\frac{(\frac{(c\rho)^c}{c!})(\frac{1}{1-\rho})}{\sum_{k=0}^{c-1}\frac{(c\rho)^k}{k!}+(\frac{(c\rho)^c}{c!})(\frac{1}{1-\rho})} \tag{2}$$

式(2)被称为 Erlang 公式。

用 U_j 来表示被调度到部署在 p_j 处的微云的用户集合,表示为 $U_j=\{u_i\mid y_{i,j}=1\}$。如果一个微云的工作负载过重,队列时间就可能变得格外长,这就有可能使得移动用户的应用变慢。对于微云来说,当它过载时,将超出的任务量卸载到遥远的云端上处理是常用的解决方案。假定每个微云的最大工作负载被卸载率 $\lambda_{\max}$ 所限制,其他剩余的任务请求将被卸载到遥远的云端。用 ϕ_j 来表示在微云上处理的任务比例:

$$\phi_j=\begin{cases}1, & \text{if } \lambda_{\max}>\lambda(j)\\ \dfrac{\lambda_{\max}}{\lambda(j)}, & \text{otherwise}\end{cases} \tag{3}$$

其中,$\lambda(j)=\sum_{u_i\in U_j}\lambda_i$,$U_j$ 表示被调度到部署在 p_j 处的微云的用户集合。部署在 p_j 处的微云上的每个任务的等待时间为:

$$t_{clt}(j)=f_Q(\phi_j\cdot\sum_{u_i\in U_j}\lambda_i)+1/\mu \tag{4}$$

被卸载到遥远云端的任务是通过互联网传输的,我们假定这样的传输将产生固定的延时 B,并且云端拥有充足的计算资源去执行任务,云端上的队列时间忽略不计。我们将云端模拟为一个 $M/M/\infty$ 队列,并且与所有微云一样拥有固定的服务率 μ。卸载任务到云端的等待时间可以表示为:

$$t_{cld}=B+1/\mu \tag{5}$$

根据式(3)-式(5)可得,卸载用户 u_i 的任务的平均等待时间为:

$$t_i=\omega_i+D_{k,j}+\phi_j\cdot t_{clt}(j)+(1-\phi_j)\cdot t_{cld}(j) \tag{6}$$

其中,u_i 与接收点 p_k 无线连接,并被调度到部署在 p_j 处的微云。

系统中所有用户卸载任务的平均等待时间称为系统的响应时间,按式(7)计算:

$$t=\frac{1}{n}\sum_{i=1}^{n}t_i \tag{7}$$

3.3 问题的形式化定义

我们引入两个集合 X 和 Y,分别表示微云部署方案和用户任务调度方案。其中,x_j 表示在接收点 p_j 处是否部署了微云,如果部署了微云,则 $x_j=1$,否则 $x_j=0$;Y 表示用户到微云的调度方案,如果用户 u_i 被调度到了部署在 p_j 处的微云,则 $Y_{i,j}=1$,否则 $Y_{i,j}=0$。我们假定所有的微云都会被多个接收点共同连接:

$$X=\{x_j\mid 1\leqslant j\leqslant m\}$$

$$Y=\{y_{i,j}\mid 1\leqslant i\leqslant n,1\leqslant j\leqslant m\}$$

另外,还引入了一个用户与微云之间的可容忍网络延时的阈值 T_{net}。虽然无法提出一种解决方案使得每个用户的网络延时都低于 T_{net},但是在设计算法时引入阈值将具有实际意义。接下来将定义一些与问题相关的参数。用 Λ 表示用

户到达率的集合：

$$\Lambda=\{\lambda_i \mid 1\leqslant i\leqslant n\} \tag{8}$$

用 W 表示用户与其无线连接接收点之间的无线延迟的集合：

$$W=\{\omega_i \mid 1\leqslant i\leqslant n\} \tag{9}$$

无线城域网中 K 个微云的部署问题（K Cloudlet Placement Problem，KCP）可以定义为：根据给定的整数 $K\geqslant 1$ 和系统参数 $(G,\Lambda,W,D,T_{net},\lambda_{max},B,\mu,c)$，得出微云部署方案 X 和用户任务调度方案 Y，从而最小化系统响应时间：

$$\min_{X,Y} t \tag{10}$$

4　最小化系统响应时间的微云部署方案

本节针对 KCP 问题提出两种启发式算法，首先提出一种简单的负载优先部署（Heaviest-AP First，HAF）算法，而后提出一种密度优先部署（Density-Based Clustering，DBC）算法来优化前者的缺陷。

4.1　负载优先部署方案

解决 KCP 问题时，首先要找到无线城域网中的微云部署位置。解决该问题的目的是通过将微云贴近用户来缩短系统响应时间，对此一种简单有效的方案就是将微云直接部署到用户工作负载最重的接收点上。对于网络中的所有接收点，我们根据与该接收点直接无线连接的用户的累计任务卸载率降序排列，然后取前 K 个为微云部署点。确定微云部署点后，调度用户到微云。

对于每个无线连接到 p_j 的用户 u_i，找到与 p_j 之间有最小网络延时的微云 $D_{k,j}$，然后将 u_i 调度到这个微云。这样就最小化了用户与服务他的微云之间的网络延时。算法 1 给出了负载优先算法的细节。

算法 1　负载优先算法

输入：(K,G,Λ,W,D,T_{net},λ_{max},B,μ,c)

输出：(X,Y)

Step1　/ * 微云部署方案 * /

Step2　初始化：Q←Ø；/ * Q 为微云部署点的集合 * /

Step3　for k←1 to K do

Step4　　j←k，其中 k 为接收点 p_j 的索引值，$\sum_{u_i\in user(p_k)}\lambda_i=\max_{p_k'\in V-Q}\{\sum_{u_i\in user(p_k')}\lambda_i\}$，user($p_k$)为与接收点 p_k 直接无线连接的用户的集合；

Step5　　Q←Q∪{p_j}；

Step6　　X[j]←1；

Step7　　/ * 用户任务调度方案 * /

Step8　for i←1 to n do

Step9　　p_k 为用户 u_i 的连接点；

Step10　　找到一个微云 j 并调度用户的任务给他，其中 $D_{k,j}=\min_{p_j\in Q}\{D_{k,j}\}$；

Step11　Y[i,j]←1.

HAF 算法有两个主要缺点：1）有着最大工作负载的接收点不一定最接近他所服务的用户，如图 1 所示，虽然接收点 p_j 可能有最大工作负载，但是与之无线连接的用户都处在网络的边缘，这就导致如果 p_j 被选为微云部署点，那么除这些用户之外的其他用户想要访问这个边缘接收点的传输代价将是很大的，显而易见，这个模型中的接收点 p_j 并不是一个好的微云部署点；另一方面，虽然接收点 p_i 没有与任何用户无线连接，即工作负载为零，但是网络中的大部分用户都与 p_i 相距不远，因此 p_i 相较于 p_j 来说是一个更优的微云部署点。综上所述，用人口密度相对较大的区域的接收点代替直接工作负载最大的接收点作为微云的部署位置，是我们的改进措施之一。

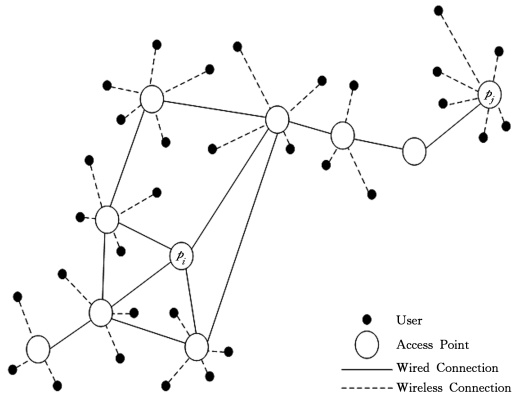


图 1　无线城域网模型图

Fig. 1　WMAN model diagram

2）一味地将用户调度到最邻近的微云，可能导致微云过载，并产生过长的队列时间，从而影响用户体验。图 2 展示了调度到微云上的工作负载与微云队列时间的关系。当工作负载增加到一定大小时，队列时间将快速增长，此时任务在微云上处理的等待时间有可能超过传输到遥远云端的等待时间。因此在用户调度时，采取怎样的调度策略能使得总的系统响应时间最小也是我们研究的重点之一。

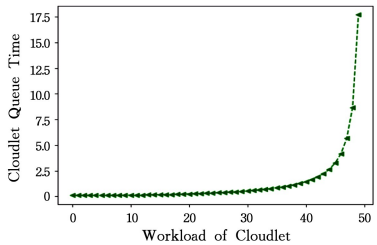


图 2　微云队列时间

Fig. 2　Cloudlet queue time

4.2　密度优先部署方案

为了弥补 HAF 算法的缺陷，本节提出 KCP 问题的主要解决方法。通过对 HAF 算法的讨论，我们找到了能更有效解决 KCP 问题的两个关键方向：1）希望在无线城域网中用户相对密集的区域部署微云，这意味着微云将更靠近大多数用户，从而减小用户与微云之间的平均网络延时；2）限制各微云的工作负载，这样可以有效减少任务的平均微云队列时间。于是，我们提出一种基于密度的微云部署方案。前面已经定义了参数 T_{net}，它表示所能容忍的网络延时的阈值；假设用 $U_{T_{net}}(j)$ 表示与 p_j 之间网络延时不超过 T_{net} 的用户集合：

$$U_{T_{net}}(j)=\{u_i \mid D_{k,j}\leqslant T_{net}\}$$

其中，p_k 为 u_i 的无线连接点。我们称 $U_{T_{net}}(j)$ 为接收点 p_j 的候选用户集合。因为 p_j 的候选用户与 p_j 之间有较小的网络延时，所以 p_j 的所有候选用户都将被调度到接收点 p_j 周围。

一个接收点的候选用户集合的大小可以用来指示该接收点所在区域的用户密度。用 $\lambda_{T_{net}}(j)$来表示 p_j 的候选工作负载：

$$\lambda_{T_{net}}(j)=\sum_{u_i\in U_{T_{net}}(j)}\lambda_i$$

首先选择有最大 $\lambda_{T_{net}}(j)$的接收点 p_j 来部署微云；然后移除网络 G 中与 p_j 直接相连的用户集合，再重新计算更新后的网络中每个接收点的候选工作负载，找到下一个微云部署点。重复以上过程 K 次，完成 K 个微云的部署。

因为互相邻近的接收点经常会共享候选用户，当一个接收点与拥有大量候选用户的接收点相邻时，这个接收点本身也可能拥有大量的候选用户。无线城域网中人口密集的区域往往有多个接收点可以被选作微云部署点。通过在 p_j 部署微云之后移除与 p_j 直接相连的用户，可以减小在人口密集区域微云部署过饱和的可能性。这意味着微云的分布大致符合移动用户的分布，使得调度用户到微云时更容易平衡微云间的工作负载。

接下来解决用户到微云的调度问题。我们找到有着最大候选用户工作负载的微云，用 λ_{avg} 表示每个微云的平均工作负载：

$$\lambda_{avg}=\frac{1}{K}\cdot\sum_{i=1}^{n}\lambda_i$$

然后调度微云的候选用户到它本身，直到该微云的工作负载超过 λ_{avg}。一旦某个微云完成了上述过程，被调度的用户集以及微云部署点将从无线城域网中移除。对下一个拥有最大候选用户集合的微云进行同样的操作，这个过程直到所有的微云都有机会去调度它的候选用户才结束。最后，剩余的未得到调度的用户将被调度到距离他最近的微云上。

当某个微云的工作负载到达一个阈值时，超出的任务将被卸载到云端执行，以保证微云队列时间不会大于任务传输到云端所产生的网络延时。算法 2 给出了负载优先算法的细节。

算法 2 密度优先部署方案

输入：(K,G,Λ,W,D,T_{net},λ_{max},B,μ,c)

输出：(X,Y)

Step1 /* 微云部署方案 */

Step2 U′←U; /* U′为未调度到微云的用户集合 */

Step3 初始化：Q←Ø; /* Q 为微云部署点的集合 */

Step4 for iteration←1 to K do

Step5 找到微云 j 使 $\lambda_{T_{net}}(j)=\max_{p_k\in V}\{\lambda_{T_{net}}(k)\}$;

Step6 Q←Q∪{p_j};

Step7 U′←U′−user(p_j)，其中 user(p_j)为与相 p_j 连接的用户；

Step8 X[j]←1;

Step9 /* 用户任务调度方案 */

Step10 for i←1 to n do

Step11 p_k 为用户 u_i 的连接点；

Step12 找到一个微云 j 并调度用户的任务给他，其中 $D_{k,j}=\min_{p_j\in Q}\{D_{k,j}\}$;

Step13 if $\lambda_j<\lambda_{max}$

Step14 Y[i,j]←1;

Step15 else

Step16 u_i 的任务被卸载到云端；

5 仿真

本节在仿真环境中评估前面所提出算法的性能表现。首先说明仿真环境的设置，然后将所提出的算法应用到一个模拟网络的案例分析中。

5.1 仿真环境

在本文的系统模型中，我们定义了 KCP 问题的一些相关参数。除非参数是实验中的自由变量，否则每个系统参数都有一个默认值。

每次实验中，用户 u_i 的任务卸载率或工作负载 λ_i 都满足均值为 2、方差为 0.5 的正态分布。λ_i 的最大值为 2.99。ω_i 由均值为 0.2、方差为 0.1 的正态分布随机产生，并且满足 $0.1\leqslant\omega_i\leqslant0.4$。微云与云端的服务率都是 10，微云由 5 个服务器组成。任务卸载到遥远云端的互联网的延时 B 为 8，微云的最大工作负载为 35。

使用迪杰斯特拉算法来构造表示网络 G 中每一对接收点之间网络延时的矩阵 $\boldsymbol{D}_{i,j}$。

表 1 系统参数

Table 1 System parameters

符号	定义	默认值
n	用户数量	30
m	接收点数量	10
λ_i	用户任务卸载率	$0\leqslant\lambda_i\leqslant2.99$
ω_i	用户与接收点之间的无线延时	$0.1\leqslant\omega_i\leqslant0.4$
μ	微云/云端服务率	10
c	每个微云的服务器数量	5
B	网络延时	8
λ_{max}	微云最大工作负载	35

5.2 案例分析

根据仿真环境构建一个无线城域网模型，这个模型中含有 10 个接收点以及 30 个用户集，连接情况如图 1 所示。根据仿真环境设定的数值范围创建网络中的研究对象，并根据所提算法进行模拟实验。

5.2.1 基准算法的引入

为了评估前面提出的 HAF 算法和 DBC 算法，引入两个基准算法。第一个基准算法为随机微云部署算法，它的工作原理是随机部署 K 个微云到网络中，然后调度用户到距离他们最近的微云。随机微云部署算法的模拟结果由 100 次随机实验的结果平均而来。另一个基准算法为最优微云部署算法，通过计算所有可能的微云部署和用户任务调度方案，得出最小的系统响应时间(System Responses Time，SRT)。因为案例学习中的无线城域网是非常抽象的，它只是一些有限的用户和接收点，所以我们可以遍历所有方案而找到时间表现最好的一种。

5.2.2 密度优先微云部署算法的性能预评估

密度优先微云部署算法的性能会受到用户与微云之间可容忍网络延时阈值 T_{net} 的影响，因此在与其他算法比较之前，我们希望先找到能产生最优结果的 T_{net} 值。基于所提出的系统，在不同阈值 T_{net} 的情况下计算不同的 K 值对应的系统响应时间，从而得到最优的 T_{net} 的预设值。

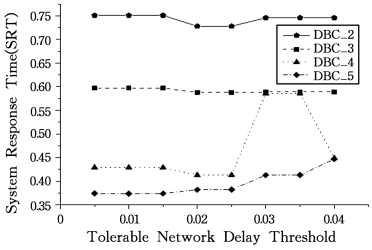


图 3　DBC 性能预评测

Fig. 3　DBC performance pre-evaluation

从图 3 中可以看出，无论 K 取值多少，能产生最小系统响应时间的可容忍网络延时阈值 T_{net} 都在同一个区间内，因此当使用密度优先微云部署算法与其他算法比较时，可以将 T_{net} 预设在这个最优区间范围内。

5.2.3　不同微云数量下算法性能的比较

分别计算从 $K=1$ 到 $K=5$ 时各算法得到的微云部署方案所产生的系统响应时间，然后绘制折线图进行比较。

图 4 展示了系统响应时间随着微云数量变化而变化的情况。密度优先微云部署算法和最优微云部署算法从 $K=1$ 增加到 $K=5$ 时，系统的响应时间始终递减。当 $K=1$ 时，由于微云的负载过大，大部分用户的任务都将卸载到云端处理，增加一个微云将缩短 62% 的系统响应时间。然而当 $K>3$ 时，网络中有了足够多的微云，以至于所有的用户任务都能被卸载到微云上，而不是遥远的云端。因为用户可以将他们的任务全部卸载到微云上，所以系统响应时间只受无线传输时间和用户与微云之间网络延时的影响，系统响应时间趋于稳定，不再有明显的减少。

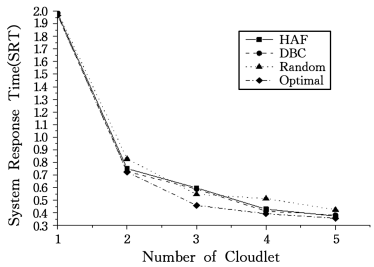
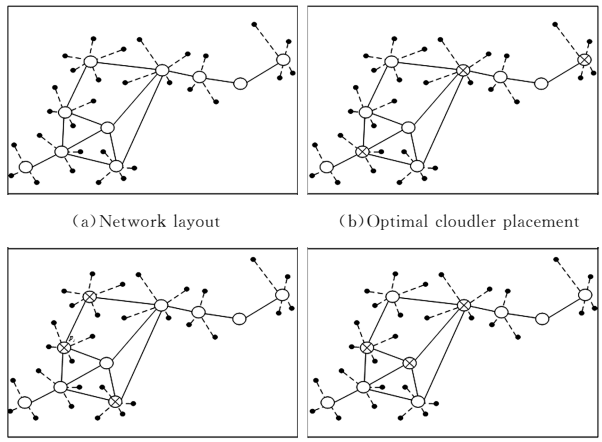


图 4　不同微云数量下的系统响应时间

Fig. 4　System response time under different cloudlet numbers

密度优先微云部署算法有着接近最优微云部署算法的性能表现，前者的系统响应时间平均只比后者多 8%。相反，负载优先微云部署算法的表现则较差，它产生的系统响应时间比最优微云部署算法平均多了 11%。随机微云部署算法的表现则更差，它产生的系统响应时间比最优微云部署算法平均多了 17%。当 K 值很小时，负载优先微云部署算法的性能表现甚至比随机微云部署算法还要差。为了更好地理解出现上述情况的原因，将在模拟网络中对比当 $K=3$ 时各种算法微云部署的具体情况，如图 5 所示。

从图 5(c)可以看出，负载优先算法将微云部署在网络中大多数用户集中的中心地带，这与图 5(b)中的最优算法明显不同，最优算法更倾向于将微云均匀分布在网络中。HAF 算法下的微云部署使得一些用户要经过 3 个接收点才能到达最近的微云部署点，而最优算法下的部署方案则完全没有出现这种情况。由 HAF 算法产生的高度集群化的微云部署也凸显了最近邻用户任务调度方案的缺陷：部署在网络外部的微云可能接收到了大部分用户的任务需求，而网络内部的微云的工作负载却很小。例如，图 5(c)中部署在 p_j 处的微云就可能只服务于与他直接相连的用户，这就可能导致微云间出现负载不均衡的情况，因此在这种情况下，HAF 算法的性能表现还不如随机部署算法。另一方面，DBC 算法通过每次放置迭代结束后移除微云位置的操作来避免微云部署过于密集的问题，有利于微云的均匀放置以及负载平衡。综上所述，我们可以得出 DBC 算法的性能最接近最优算法的结论。



(a) Network layout　(b) Optimal cloudler placement

(c) HAF coulder placement　(d) DBC cloudlet placement

图 5　$K=3$ 时各算法的微云部署情况

Fig. 5　Cloudlet deployment of each algorithm under $K=3$ condition

结束语　对于大多数想要通过减少系统响应时间来提升用户体验的移动应用来说，微云技术是至关重要的。但是，微云在网络中的部署位置以及用户任务调度方案的研究在很大程度上被忽略了。本文研究了在无线城域网部署有限个微云的问题，并提出具体的部署算法来减少系统的响应时间，从而提升移动应用的用户体验。另外，还设置了合理的仿真环境，并基于仿真环境构建仿真模型，最后在仿真模型中评测所提出的优化算法。仿真结果显示所提算法得到的部署方案非常接近最优部署方案，基本达到了前期所预想的优化目的。由于本文主要关注与微云部署位置相关的系统响应时间，因此还有其他一些重要问题需要进一步研究。例如，微云服务总代价与系统响应时间之间的关系。微云服务总代价由部署代价和维护代价两个部分组成。虽然微云技术从某种角度上说与传统的云技术是相似的，假定微云服务与云服务有相似的总代价是合理的，但是它们之间也存在着显著的差异，因此需要进行研究才能精准地估计微云服务的总代价。此外，虽然最近几年有许多研究者对微云的性能展开了研究，但是大多数研究只集中于单个微云和少量的移动用户，而事实上，用户规模是一个重要的问题。综上所述，微云服务代价问题以及大规模用户条件下的微云利用问题是我们未来研究的重点。

参 考 文 献

[1] SATYANARAYANAN M, BAHL P, DAVIES N. The Case for VM-Based Cloudlets in Mobile Computing[J]. IEEE Pervasive Computing, 2009, 8(4): 14-23.

[2] CLINCH S,HARKES J,FRIDAY A,et al. How close is close enough? Understanding the role of cloudlets in supporting display appropriation by mobile users[C]// IEEE International Conference on Pervasive Computing and Communications. IEEE,2012:122-127.

[3] ZHAO ZM,LIU F,CAI Z P,et al. Edge Computing:Platforms, Applications and Challenges[J]. Journal of Computer Research and Development,2018,55(2):327-337.

[4] Cisco Visual Networking. Cisco global cloud index:Forecast and methodology 2015-2020 [EB/OL]. [2017-08-15]. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-cll-738085.pdf.

[5] HA K,PILLAI P,LEWIS G,et al. The Impact of Mobile Multimedia Applications on Data Center Consolidation[C]// IEEE International Conference on Cloud Engineering. IEEE Computer Society,2013:166-176.

[6] HU W,GAO Y,HA K,et al. Quantifying the Impact of Edge Computing on Mobile Applications[C]// ACM Sigops Asia-Pacific Workshop on Systems. ACM,2016:5.

[7] WOLBACH A,HARKES J,CHELLAPPA S,et al. Transient customization of mobile computing infrastructure[C]// Proceedings of the First Workshop on Virtualization in Mobile Computing. ACM,2008:37-41.

[8] HA K,PILLAI P,RICHTER W,et al. Just-in-time provisioning for cyber foraging[C]// Proceeding of the International Conference on Mobile Systems,Applications,and Services. 2013:153-166.

[9] KEMP R,PALMER N,KIELMANN T,et al. Cuckoo:A Computation Offloading Framework for Smartphones[C]// International Conference on Mobile Computing,Applications,and Services. Springer Berlin Heidelberg,2010:59-79.

[10] ZHANG Y,LIU H,JIAO L,et al. To offload or not to offload:An efficient code partition algorithm for mobile cloud computing[C]// IEEE International Conference on Cloud Networking. IEEE,2013:80-86.

[11] CHUN B G,IHM S,MANIATIS P,et al. CloneCloud:elastic execution between mobile device and cloud[C]// Conference on Computer Systems. ACM,2011:301-314.

[12] CUERVO E,BALASUBRAMANIAN A,CHO D K,et al. MAUI:making smartphones last longer with code offload[C]// International Conference on Mobile Systems,Applications,and Services. DBLP,2010:49-62.

[13] KOSTA S,AUCINAS A,HUI P,et al. ThinkAir:Dynamic resource allocation and parallel execution in the cloud for mobile code offloading[C]// 2012 Proceedings IEEE INFOCOM. IEEE,2012:945-953.

[14] RA M R,SHETH A,MUMMERT L,et al. Odessa:enabling interactive perception applications on mobile devices[C]// International Conference on Mobile Systems,Applications,and Services. ACM,2011:43-56.

[15] SHIRAZ M,ABOLFAZLI S,SANAEI Z,et al. A study on virtual machine deployment for application outsourcing in mobile cloud computing[J]. Journal of Supercomputing,2013,63(3):946-964.

[16] CARDELLINI V,PERSONÉ V D N,VALERIO V D,et al. A game-theoretic approach to computation offloading in mobile cloud computing[J]. Mathematical Programming,2016,157(2):421-449.

[17] CAI W,LEUNG V C M,CHEN M. Next Generation Mobile Cloud Gaming[C]// IEEE Seventh International Symposium on Service-Oriented System Engineering. IEEE Computer Society,2013:551-560.

[18] CAI W,LEUNG V C M,HU L. A Cloudlet-Assisted Multiplayer Cloud Gaming System[J]. Mobile Networks & Applications,2014,19(2):144-152.

[19] VERBELEN T,SIMOENS P,TURCK F D,et al. Cloudlets:bringing the cloud to the mobile user[C]// ACM Workshop on Mobile Cloud Computing and Services. ACM,2012:29-36.

[20] VERBELEN T,SIMOENS P,TURCK F D,et al. Leveraging Cloudlets for Immersive Collaborative Applications[J]. IEEE Pervasive Computing,2013,12(4):30-38.

[21] JIA M,CAO J,LIANG W. Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks[J]. IEEE Transactions on Cloud Computing,2017,PP(99):1.

[22] LIU B. Theory and Practice of Uncertain Programming [J]. Studies in Fuzziness & Soft Computing,2002,102(4):295-318.

[23] SÁ G. Branch-And-Bound and Approximate Solutions to the Capacitated Plant-Location Problem [J]. Operations Research,1969,17(6):1005-1016.