基于无线城域网的微云负载均衡算法

曾金晶[1,2] 张建山[2,3] 林 兵[2,3] 张文德[1]
(福州大学经济与管理学院 福州 350116)[1]
(福建省网络计算与智能信息处理重点实验室 福州 350116)[2]
(福建师范大学物理与能源学院 福州 350117)[3]

摘 要 随着无线通信技术的发展,越来越多的商业、娱乐和社交活动建立在便携式移动设备之上。便携式移动设备的尺寸限制了它的计算能力,计算能力的不足与应用程序的高计算需求相矛盾。边缘计算使得计算任务在数据源头附近就能得到及时处理,是减小系统延迟的有效方法。微云技术是边缘计算的重要应用,部署微云是解决上述矛盾的有效方法。多个微云连接在一起形成网络,终端用户可以通过无线城域网(Wireless Metropolitan Area Networks,WMAN)来获得微云服务。如何将任务卸载并调度到合理的微云中,减少系统延迟,是目前面临的重大挑战。文中研究了如何平衡网络中多个微云之间的工作负载,以优化移动应用程序的性能表现。首先,引入一个系统模型来获取卸载任务的响应时间,并制定一个在微云之间寻找卸载任务调度的最佳方案,以最小化微云上任务的平均响应时间。其次,提出了一种快速且可扩展的启发式算法来缩短用户任务的响应时间。最后,通过仿真实验来评估所提算法的性能特征。实验结果表明,该算法在缩短用户任务响应时间方面有着积极作用。

关键词 边缘计算,微云负载平衡,任务再卸载,任务调度
中图法分类号 TP338 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.08.027

Cloudlet Workload Balancing Algorithm in Wireless Metropolitan Area Networks

ZENG Jin-jing[1,2] ZHANG Jian-shan[2,3] LIN Bing[2,3] ZHANG Wen-de[1]
(School of Economics and Management,Fuzhou University,Fuzhou 350116,China)[1]
(Fujian Key Laboratory of Network Computing and Intelligent Information Processing,Fuzhou 350116,China)[2]
(College of Physics and Energy,Fujian Normal University,Fuzhou 350117,China)[3]

Abstract With the development of wireless communication technology,more and more business,entertainments and social activities are built on portable mobile devices. The size of portable mobile devices limits their computing power,and the lack of computing power conflicts with the high computational requirement of the application. 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,and deploying cloudlet is an effective way to solve the above contradiction. Multiple cloudlet are connected to form a network,and end-user can get cloudlet services via wireless metropolitan area network(WMAN). The currently major challenges are how to offload and schedule the tasks to a reasonable cloudlet to reduce system delay. This paper investigated how to balance the workload between multiple cloudlet in a network to optimize the performance of mobile applications. It first introduced a system model to get the response times of offloaded tasks,and developed an optimal solution finding the best offloading scheme of the task between cloudlet to minimize the average responses time at cloudlets. Then,it proposed a fast,scalable heuristic algorithm for this problem to reduce the user task response time. Finally,it evaluated the performance of the proposed algorithm through experimental simulation. Experimental results show that the algorithm has a positive effect on reducing task response time and improving user experience.

Keywords Edge computing,Cloudlet workload balancing,Task re-offloading,Task scheduling

---

到稿日期:2019-03-26 返修日期:2019-06-24 本文受国家重点研发计划资助项目(2018YFB1004800),国家自然科学基金面上项目(61672159),福建省自然科学基金面上项目(2019J01286,2019J01061386),福建省高校中青年信息化课题(K8018K05A)资助。
曾金晶(1986-),女,博士生,主要研究方向为信息资源管理;**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;**张文德**(1963-),男,博士,教授,主要研究方向为信息资源管理、计算机软件及应用,E-mail:zhangwd@fzu.edu.com(通信作者)。

1 引言

近年来,随着移动计算技术的发展以及智能移动设备的普及,移动用户能够在智能终端设备上体验更加丰富的应用程序功能。便携式移动设备受到其自身尺寸的限制,计算资源十分有限[1],计算资源的不足与应用程序对计算资源需求的持续增长相矛盾。移动设备可以通过将计算密集型任务卸载到云端执行来减小其工作负载并延长电池的待机时间[2-3],这是解决移动设备计算资源匮乏问题的传统方法。然而,移动用户与远程云端之间的物理距离过长将导致较长的响应时延,引起应用程序响应的滞后,从而严重影响应用程序的用户体验。另外,过长的物理距离也限制了应用程序任务的卸载效率。为了提高任务的卸载效率,相关研究人员建议将被称为"微云"的计算机集群部署在用户网络中来接收并处理被卸载的任务,以解决移动设备卸载任务时出现的各种问题[3-9]。

微云技术是边缘计算的重要应用。相较于传统的云计算,边缘计算能够更好地支持移动计算,它具有以下几个优点[10]。

1)极大地缓解了网络带宽与数据中心的压力。思科在2015-2020年全球云指数中指出,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的临时数据,减轻了网络带宽与数据中心的压力。

2)增强了服务的响应能力。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高、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[11]。而边缘计算在用户附近提供服务,近距离服务保证了较低的网络延时,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比云服务有更强的响应能力[12]。

3)保护隐私数据,提升了数据安全性。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,提升了数据的安全性。

微云是一个受信任并且资源丰富的计算机集群,它通过无线网络与其部署位置附近的移动用户互相连接[5]。移动设备可以将应用程序产生的计算任务卸载到附近的微云,以获得对丰富计算资源的低时延访问,从而显著提高应用程序的性能[5,13],进而提升用户体验。虽然微云通常被定义为独立的"微型数据中心"[5],但将多个微云连接在一起形成网络具有明显的益处。相关研究[8-9,14]讨论了如何在公共无线城域网中部署微云,将其作为接入无线网络的附属服务。大规模市区拥有较高的人口密度,因此微云可以接收到大量用户的任务需求,这将显著提高微云的利用率,从而提高微云的成本效益。此外,由于此类网络的规模庞大,运营商在通过无线城域网提供微云服务时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。

无线城域网服务提供商当前面临的主要挑战是:如何将网络中所有的用户任务请求合理地分配到不同的微云,以使得网络中各微云的负载更加均衡,从而缩短被卸载任务的响应时间。此类问题的传统解决方案是将用户请求分配给距其最近的微云,以最小化网络访问时延。然而,这种方法被证明并不适用于无线城域网环境[8-9,14]。网络中庞大的用户数量意味着每个微云的工作负载具有波动性,如果微云过载,那么后续分配到该微云上的任务的响应时间将急剧增加,从而导致用户应用程序的响应滞后并严重影响用户体验。为了防止此类问题的发生,将用户请求分配给正确的微云至关重要,这有助于微云之间的工作负载得到更好的平衡,从而缩短被卸载任务的平均响应时间。

本文通过设计有效的调度算法来将用户请求分配给网络中合适的微云,在满足用户动态需求的条件下实现微云间的负载均衡,以缩短被卸载任务的平均响应时间。首先,引入一个系统模型来获取任务的响应时间,并以最小化给定微云集条件下卸载任务平均响应时间的最大值为目标制定一个新的优化问题;其次,提出一种快速且可扩展的微云负载均衡算法,以缩短用户任务的响应时间;最后,通过实验来评估所提算法的性能表现。实验结果表明,所提算法在缩短用户任务响应时间方面发挥着积极作用。

2 相关工作

近年来,由于移动设备计算资源匮乏而将移动应用所产生的部分计算任务卸载到云端服务器处理的相关研究已经取得了一定的成果[4,15]。移动云计算环境下的应用任务卸载系统由移动设备上的客户端和云端服务器上的服务端组成[2,16]。客户端的主要功能包括:实时监控整个网络的服务质量,预测移动应用对计算资源的需求,估计计算任务在本地执行所需的时间和在云端执行所需的时间等。根据上述工作所得到的信息,客户端可以判断出有多少任务需要被卸载执行。在最近的相关研究中,ThinkAir[17],Virtual SmartPhone[18]和CloneCloud[16]都使用了虚拟机作为任务卸载的平台。移动设备的虚拟机镜像和计算任务被传输到云端服务器,计算任务最终在云端服务器的虚拟机上得到处理。此外,为了满足用户对网络服务质量提出的高要求[19],有研究提出将微云服务器与传统云服务器相结合,共同处理从移动终端卸载的任务[3,20-22]。Hoang等[19]提出了一种面向微云的任务卸载线性规划解决方案,该方案旨在最大化微云服务提供商的商业收入。Xia等[3,20]提出了一种高效的在线算法,该算法允许微云动态地接收来自用户的任务请求。Gelenbe等[22]综合考虑任务卸载能耗和用户服务质量要求的问题,研究了应用程序在本地云服务器和传统云服务器之间的卸载问题。

将任务卸载到传统云端服务器的主要缺点是移动用户与传统云端服务器之间的网络时延较大,这将严重影响交互式应用程序(如移动在线游戏)的用户体验。微云通过提供对丰富计算资源的低时延访问来克服这一缺点,这对于提升移动应用程序的用户体验起到了积极作用。将应用程序的计算任务卸载到微云的方法一般是将计算任务封装到虚拟机中,再卸载到微云服务器[5,23];此后,移动设备的计算任务就可以在微云服务器上得到处理,处理结果将从微云传回移动设备。当网络中某个微云服务器的工作负载过大时,微云服务器的

处理速度将受到限制。最近有研究[14,21]提出了一个系统模型,其利用排队论[24]可以估计微云上任务的平均响应时间。Verbelen 等[6]提出了一种细粒度的微云卸载方法,其中移动应用程序将在运行时动态分区为独立的远程可执行组件。这种方法的优点是将组件分布在多个微云中,从而最大程度地提高并行性。此外,在网络环境不佳的情况下,卸载的组件数量可以动态缩小,从而提供更加顺畅的用户体验。

最近的几项研究进一步扩展了微云网络的定义,使其包含点对点计算机。Verbelen 等[6-7]提出了新的微云网络框架,该框架下网络中的相邻设备之间可以通过点对点模式共享资源。本文将微云定义为与接入点共存的计算机集群。

同样地,无线网络中的微云放置问题也备受关注[8-9,14]。Jia 等[14]的研究表明,对无线城域网中有限个数的微云进行有策略的部署可以大大提高移动用户设备的性能。Xu 等[8]制定了一个微云放置问题,将微云部署在无线感知网络中的一些潜在部署点上,目的是最大程度地减小移动用户和为其提供服务的微云之间的时延。

3　问题定义

本节首先构建一个无线城域网卸载系统模型,然后形式化定义微云负载均衡问题。

3.1　卸载系统模型

假定服务提供商在无线城域网中的固定位置部署了 K 个微云,这些微云部署在网络中的数据接收点上,并且每一对微云之间都通过网络彼此连接。图 1 为无线城域网的理想化模型。另外,假定用户应用程序所产生的计算任务被动态地分离为一束可卸载的任务流,它的任意一个部分都可以在网络中的任意一个微云上得到处理。

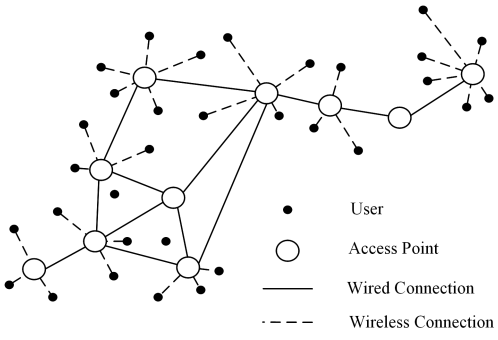


图 1　无线城域网模型

Fig. 1　WMAN model

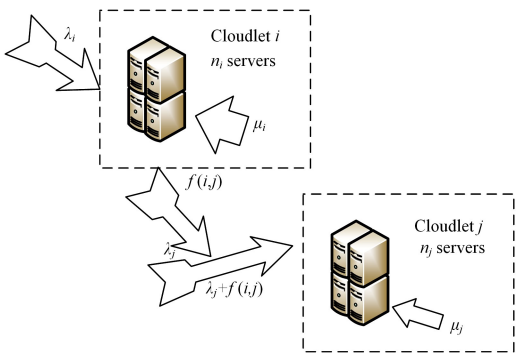


图 2　从微云 i 再卸载到微云 j 的任务流

Fig. 2　Flow of tasks from cloudlet i re-offloading to cloudlet j

如图 2 所示,每个用户的任务请求都将卸载到网络中已部署好的微云上进行处理,微云可以将接收到的任务流添加到自己的任务队列中等待处理,也可以将其部分任务流再卸载到网络中的另一个微云上。

将所有微云模拟为 $M/M/n$ 队列,其中每个微云 $i\in\{1,\cdots,K\}$ 由 n_i 个服务率为 μ_i 的服务器组成。由于用户需求具有快速变化的性质,且每个微云上传入任务的速率可能随着时间的推移频繁地波动,因此我们假设微云 i 接收到的用户任务根据泊松过程以抵达率 λ_i 随机抵达系统。微云 i 上的任务平均响应时间由微云队列时间和任务服务时间组成。假设 Q_i 表示根据给定的微云 i 的抵达率 λ_i 计算微云 i 队列时间的函数,则:

$$Q_i(\lambda_i)=\frac{C(n_i,\frac{\lambda_i}{\mu_i})}{n_i\mu_i-\lambda_i} \tag{1}$$

其中:

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

式(2)被称为 Erlang 公式。微云 i 上的任务服务时间 S_i 由组成微云的服务器的服务率决定,服务率与服务时间成反比关系,即:

$$S_i=\frac{1}{\mu_i} \tag{3}$$

根据式(1)与式(3),可得微云 i 的任务平均响应时间 T_i 为:

$$T_i(\lambda_i)=\frac{C(n_i,\frac{\lambda_i}{\mu_i})}{n_i\mu_i-\lambda_i}+\frac{1}{\mu_i} \tag{4}$$

由于不同微云的任务抵达率有较大的差异,因此网络中的部分微云可能会过载,而另一部分微云则可能未得到充分利用。我们假定所有微云之间都可以互相访问,并且每一个微云都可以将其任务流的一部分再卸载到另一个微云。当 $i\neq j$ 时,$f(i,j)$ 表示从微云 i 再卸载到微云 j 的任务流大小。$f(i,j)$ 有以下约束:

$$f(i,j)=\begin{cases}-f(j,i), & \text{if } i\neq j\\ 0, & \text{otherwise}\end{cases},\ \forall i,j\in\{1,\cdots,K\} \tag{5}$$

$$\sum_{i=1}^{K}\sum_{j=1}^{K}f(i,j)=0 \tag{6}$$

$$\sum_{j=1}^{K}\max\{f(i,j),0\}\leqslant\lambda_i,\ \forall i\in(1,\cdots,K) \tag{7}$$

式(5)确保了网络中对于任意两个微云 i 和 j,从微云 i 到微云 j 的任务流量是微云 j 到微云 i 的任务流量的负数。需要注意的是,此处的负号表示任务再卸载的方向。由于从任一个微云 i 再卸载该微云自身的任务流量皆为零,因此有 $f(i,i)=0$。式(6)用于确保所有的任务最后都能得到处理。式(7)确保微云 i 中所有再卸载的任务流的大小总和不大于其抵达率 λ_i。

假定所有被卸载的任务都由若干个相同的单位数据包组成,通过网络在一对接收点传输任务所产生的时延由任务大小以及两接收点之间的物理距离共同决定。为了模拟无线城域网中的这种网络时延,用 $\boldsymbol{d}\in\boldsymbol{R}^{K\times K}$ 表示网络时延矩阵,其中

$d_{i,j}$表示将单位任务从微云 i 再卸载到微云 j 所产生的通信时延，即微云 i 与微云 j 之间的总通信时延为$|f(i,j)|\cdot d_{i,j}$。然后，计算从其他微云再卸载到微云 i 的所有任务流所产生的网络延迟的总和 $T_{net}(i)$：

$$T_{net}(i)=\sum_{j=1}^{K}(\max\{f(j,i),0\}\cdot d_{i,j}) \tag{8}$$

使用式(4)和式(8)，我们可以计算微云 i 上任务的平均响应时间 $D(i)$：

$$D(i)=T_i(\overline{\lambda_i})+T_{net}(i) \tag{9}$$

其中，$\overline{\lambda_i}$表示最终在微云 i 上处理的任务的大小。

$$\overline{\lambda_i}=\lambda_i-\sum_{j=1}^{K}f(i,j) \tag{10}$$

3.2 问题的形式化定义

无线城域网网络 G 中的微云负载平衡问题可以有如下定义：给定一组微云$\{1,\cdots,K\}$，其中微云 i 由 n_i 个服务率为 μ_i 的服务器组成，抵达率为 λ_i，问题可以定义为在式(5)－式(7)的约束下找到一组微云内的任务流 $f=\{f(i,j)\mid i,j\in\{1,\cdots,K\}\}$，使得 K 个微云中任务响应时间的最大值最小化，即：

$$\text{minimize } \max\{D(i)\mid i\in\{1,\cdots,K\}\}$$

4 CWB 算法

本节针对无线城域网内的微云负载均衡问题提出了微云负载均衡算法(Cloudlet Workload Balancing，CWB)。该算法首先通过计算得到整个网络中所有任务的平均响应时间$\overline{D}$，之后再确定每个微云的工作负载以及每个微云有多少工作负载需要再卸载到其他微云，以使得各微云上的任务平均响应时间都接近$\overline{D}$。此外，为了后期检验 CWB 算法的性能表现，我们还引入了一种微云最大负载算法(Cloudlet Maximum Workload，CMW)作为对比算法。

4.1 任务平均响应时间

我们的目标是最小化所有微云中任务平均响应时间的最大值。由式(6)可知，任务流在系统内是守恒的，因此系统内所有的任务都需要被执行。显然，我们必须尝试将某些微云上的部分任务再卸载到其他微云，以使得每个微云具有相近的平均响应时间。本文采用的方法是首先确定每个微云的初始任务平均响应时间，之后再计算出每个过载微云的传出负载以及每个欠载微云的传入负载，最后确定从过载微云再卸载到欠载微云的特殊任务流。

设$\overline{D}$为任务的平均响应时间，为了得到$\overline{D}$和每个微云的传出/传入工作负载，我们首先估计$\overline{D}$的初始值并进行迭代计算，直到每个微云的平均任务响应时间与$\overline{D}$的差值在给定的精度阈值 ε 内。在估计$\overline{D}$的初始值之前，首先应对$\overline{D}$的取值范围进行判断。令 $T_{max}=\max\{T_i(\lambda_i)\mid 1\leqslant i\leqslant K\}$，$T_{min}=\min\{T_i(\lambda_i)\mid 1\leqslant i\leqslant K\}$，显然$\overline{D}$的取值范围为$[T_{min},T_{max}]$。根据$\overline{D}$的取值范围，可以将$\overline{D}$的初始值设定为$(T_{min}+T_{max})/2$。确定了$\overline{D}$的初始值之后，网络中所有的微云可分为两个不相交的集合，即过载微云集合 V_s：$V_s=\{i\mid T_i(\lambda_i)>\overline{D}\}$，以及欠载微云集合 V_t：$V_t=\{j\mid T_j(\lambda_j)<\overline{D}\}$。

对于每个过载微云 $i\in V_s$，我们需要计算其再卸载的任务流大小，以使得任务响应时间与$\overline{D}$的差值在精度阈值 ε 内，即我们需要找到值 ϕ_i 使得：

$$-\varepsilon\leqslant\overline{D}-T_i(\lambda_i-\phi_i)\leqslant\varepsilon \tag{11}$$

其中，ε 为精度阈值。

对于每个欠载的微云 $j\in V_t$，我们需要计算出网络中其他微云再卸载到该微云的任务流量 ϕ_j，以使得微云 j 上的任务响应时间与$\overline{D}$的差值在精度阈值 ε 内，即：

$$-\varepsilon\leqslant\overline{D}-T_j(\lambda_j-\phi_j)\leqslant\varepsilon \tag{12}$$

ϕ_i 与 ϕ_j 的计算图解如图 3 所示。

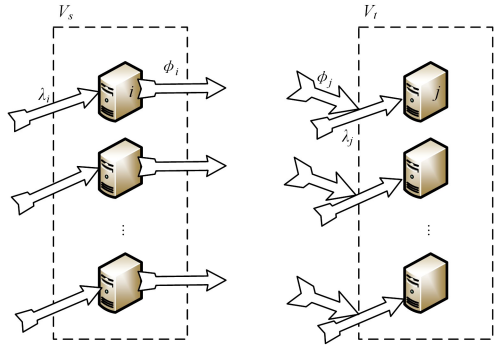


图 3 计算 ϕ_i 与 ϕ_j 的图解

Fig. 3 Diagram of computing ϕ_i and ϕ_j

要找到满足式(11)和式(12)的 ϕ_i 和 ϕ_j，需要进行迭代计算，其时间复杂度为 O(1/ε)。但是，如果预先计算能实现精度阈值 ε 的任务量，则我们可以在 O(1)的时间复杂度内找到 ϕ_i 和 ϕ_j。

想要计算出每个过载微云 $i\in V_s$ 的期望传出流量和每个欠载微云 $j\in V_t$ 的期望传入流量，我们需要确定 $f(i,j)$的值，即对于所有的 $i\in V_s$ 和 $j\in V_t$，需要从微云 i 再卸载到微云 j 的任务流大小，以使得累计的网络时延最小化。用 Δ 表示过载微云的总传出流量与欠载微云的总传入流量之间的差值，即：

$$\Delta=\sum_{i\in V_s}\phi_i-\sum_{j\in V_t}\phi_j$$

如果 $\Delta=0$，那么意味着传出任务流与传入任务流可以完美匹配。如果 $\Delta>0$，那么传出任务过剩，这意味着所选择的$\overline{D}$太小，导致过多的微云试图卸载它们的任务以缩短任务平均响应时间来满足$\overline{D}$的限制。在这种情况下，我们必须提高$\overline{D}$取值区间的下限，即 $T_{min}\leftarrow\overline{D}$。类似地，如果 $\Delta<0$，那么意味着$\overline{D}$太大，过载微云的总传出任务不足，我们必须降低$\overline{D}$取值区间的上限，即 $T_{max}\leftarrow\overline{D}$。我们在下一次迭代中将$\overline{D}$的值更新为$(T_{max}+T_{min})/2$，以此使用二分法迭代计算得到合适的$\overline{D}$。当总传出流量与总传入流量的差值 Δ 落入给定阈值区间，即$|\Delta|\leqslant\delta$时，迭代停止。

由于将任务从过载微云再卸载到欠载微云会导致目标微云出现额外的网络时延，因此我们应该进一步调整$\overline{D}$，以便在每个欠载微云 $j\in V_t$ 中，所有任务的队列时间和网络时延 $T_{net}(j)$之和尽可能接近，即 $T_j(\lambda_j)+T_{net}(j)=\overline{D}$。为此，我们必须保证来自过载微云的传出任务是不足的，即 $\Delta<0$。

我们为给定的 $\overline{D}$ 确定了一束系统内的卸载任务流 f，令 $\overline{D}'=\max\{D(j)\mid j\in V_t\}$。若$\overline{D}$与$\overline{D}'$之间的差值在给定的阈值 θ 内，则搜索完成；否则，我们需要进一步搜索$\overline{D}$。也就是说，如果$\overline{D}<\overline{D}'$，那么有过多来自过载微云的传出任务，我们需要

增大$\overline{D}$以减小每个过载微云 $i\in V_s$ 的 ϕ_i。如果$\overline{D}>\overline{D}'$,那么来自过载微云的传出任务不足,我们应该减小$\overline{D}$以使得过载微云将更多任务再卸载到欠载微云。在下一次迭代中我们将$\overline{D}$的值更新为$(\overline{D}+\overline{D}')/2$,直到$\overline{D}$与$\overline{D}'$之间的差值小于 θ。该算法的细节在算法 1 中给出。

算法 1　CWB 算法

Input:每个微云 i∈{1,…,K}的任务抵达率 λ_i,服务器数量 n_i,服务率 μ_i,网络时延矩阵 **d** 和精度阈值 θ,ε,δ

Output:微云 i,j∈{1,…,K}之间的系统内部任务流 f(i,j)

1. $T_{max}\leftarrow\max\{T_i(\lambda_i)\mid 1\leqslant i\leqslant K\}$;
2. $T_{min}\leftarrow\min\{T_i(\lambda_i)\mid 1\leqslant i\leqslant K\}$;
3. $\Delta\leftarrow\infty$;
4. /* 使用二分法迭代计算$\overline{D}$的初始值 */
5. while($|\Delta|>\delta$) do
6. 　$\overline{D}\leftarrow(T_{max}+T_{min})/2$;
7. 　$V_s\leftarrow\{i\mid T_i(\lambda_i)>\overline{D}\}$;
8. 　$V_t\leftarrow\{i\mid T_i(\lambda_i)\leqslant\overline{D}\}$;
9. 　for 每个过载微云 i∈V_s do
10. 　　找到 ϕ_i 的值,使得 $\left|\frac{\overline{D}-T_i(\lambda_i-\varphi_i)}{\overline{D}}\right|\leqslant\varepsilon$;
11. 　end for
12. 　for 每个欠载微云 j∈V_t do
13. 　　找到 ϕ_j 的值,使得 $\left|\frac{\overline{D}-T_j(\lambda_j-\phi_j)}{\overline{D}}\right|\leqslant\varepsilon$;
14. 　end for
15. 　$\Delta\leftarrow\sum_{i\in V_s}\phi_i-\sum_{j\in V_t}\phi_j$;
16. 　if $\Delta>0$ then
17. 　　$T_{min}\leftarrow\overline{D}$;
18. 　else
19. 　　$T_{max}\leftarrow\overline{D}$;
20. 　end if
21. end while
22. /* 根据微云之间的网络延迟调整$\overline{D}$的值 */
23. $V_s\leftarrow\{i\mid T_i(\lambda_i)>\overline{D}\}$;
24. $V_t\leftarrow\{j\mid T_j(\lambda_j)\leqslant\overline{D}\}$;
25. $\overline{D}'\leftarrow\infty$;
26. while($|\overline{D}-\overline{D}'|>\theta$)do
27. 　for 每个过载微云 i∈V_s do
28. 　　找到 ϕ_i 的值,使得 $|\frac{\overline{D}-T_i(\lambda_i-\phi_i)}{\overline{D}}|\leqslant\varepsilon$;
29. 　end for
30. 　for 每个欠载微云 j∈V_t do
31. 　　找到 ϕ_j 的值,使得 $\left|\frac{\overline{D}-T_j(\lambda_j-\phi_j)}{\overline{D}}\right|\leqslant\varepsilon$;
32. 　end for
33. 　调用算法 2,计算 f(i,j);
34. 　for 每个欠载微云 j∈V_t do
35. 　　根据式(9)计算 D(j);
36. 　end for
37. 　$\overline{D}'\leftarrow\max\{D(j)\mid j\in V_t\}$;
38. 　$\overline{D}\leftarrow(\overline{D}+\overline{D}')/2$;
39. end while

4.2　卸载系统模型

在确定了网络中每个微云 i 的传入/传出任务量 ϕ_i 之后,我们需要确定从过载微云 $i\in V_s$ 再卸载到欠载微云 $j\in V_t$ 的任务流 $f(i,j)$。本文将确定过载微云传出任务量的问题转化为最小成本最大流量问题,即我们将在 WMAN 中构建流网络 $G=(V,E)$,并计算得出约束条件下的最大流。

我们首先构建虚拟源节点 s 和虚拟汇聚节点 t,并根据给定的$\overline{D}$将微云划分为过载微云集合 V_s 和欠载微云集合 V_t。这样,我们将得到一组节点 $V=V_s\cup V_t\cup\{s,t\}$,以及一组边缘 $E=\{\langle s,i\rangle\mid i\in V_s\}\cup\{\langle j,t\rangle\mid j\in V_t\}\cup\{\langle i,j\rangle\mid i\in V_s,j\in V_t\}$。

用 $u(i,j)$表示 G 中边缘$\langle i,j\rangle$的容量。首先设置源节点 s 与每个过载微云 $i\in V_s$ 之间的边缘的边缘容量为 ϕ_i,其中 $i\in V_s$;再设置每个过载微云 $j\in V_t$ 与汇聚节点 t 之间的边缘的边缘容量为 ϕ_j,即对于所有边缘$\langle j,t\rangle$,$u(j,t)=\phi_j$。然后,指定源节点 s 与微云之间的边缘成本为零,即对于每个 $i\in V_s$,$c_{s,i}=0$。类似地,设置汇聚节点 t 与微云之间的边缘成本也为零,即对于每个 $j\in V_t$,$c_{j,t}=0$。对于每条过载微云 $i\in V_s$ 到欠载微云 $j\in V_t$ 的边缘,将其流容量设置为 i 处的传入容量与 j 处的传出容量的最小值,即 $u(i,j)=\min\{u(s,i),u(j,t)\}$,其成本为 $d_{i,j}$,即 $c_{i,j}=d_{i,j}$。图 4 给出了流网络 $G(V,E)$ 的结构。

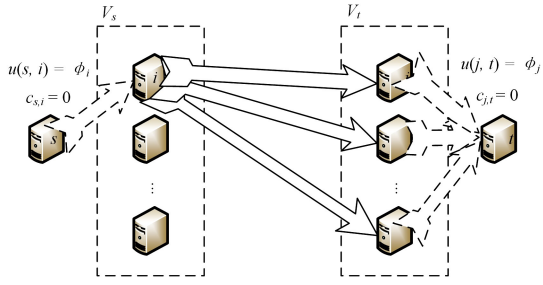


图 4　流网络的结构

Fig. 4　Structure of flow network

在构建了流网络 G 之后,可以看出,将过载微云的传出任务再卸载到欠载微云的问题被转化为了在 G 中找到从 s 到 t 的最小成本最大流量问题。也就是说,我们的优化目标是:

$$\text{minimize}\sum_{[i,j]\in E}f(i,j)\cdot c_{i,j}\tag{13}$$

受以下约束:

$$f(i,j)\leqslant u(i,j),\forall\, i,j\in V\tag{14}$$

$$f(i,j)=-f(j,i),i\neq s\text{ or }j\neq t\tag{15}$$

$$\sum_{j\in V}f(i,j)=0,i\neq s\text{ or }j\neq t\tag{16}$$

其中,$f(i,j)\cdot c_{i,j}$是任务从微云 i 再卸载到微云 j 而产生的网络延迟。

上述问题是典型的最大流问题,其目的在于找到 G 中从 s 到 t 的最大流,使得流的总成本最小化。算法 2 详述了在任务平均响应时间$\overline{D}$给定的条件下流网络 G 的构造过程。

算法 2　最小延迟流算法

Input:任务平均响应时间$\overline{D}$,服务器数量 n_i,过载微云集合 V_s,欠载微云集合 V_t 和精度阈值 ε

Output:每一对微云 i,j∈{1,…,K}之间的重定向任务流 f(i,j)

1./ * 构建具有延迟加权边的流网络 * /
2. $V \leftarrow \{1,\cdots,K\} \cup \{s,t\}$;
3. $V_s \leftarrow \{i \mid i \in \{1,\cdots,K\}, T_i(\lambda_i) > \overline{D}\}$;
4. $V_t \leftarrow \{j \mid i \in \{1,\cdots,K\}, T_j(\lambda_j) \leqslant \overline{D}\}$;
5. $E \leftarrow \emptyset$;
6. for 每个过载微云 $i \in V_s$ do
7. $E \leftarrow E \cup \{\langle s,i\rangle\}$;
8. 找到 ϕ_i 的值,使得 $\left|\frac{\overline{D}-T_i(\lambda_i-\phi_i)}{\overline{D}}\right| \leqslant \varepsilon$;
9. $u(s,i) \leftarrow \phi_i$; $c_{s,i} \leftarrow 0$;
10. end for
11. for 每个欠载微云 $j \in V_t$ do
12. $E \leftarrow E \cup \{\langle j,t\rangle\}$;
13. 找到 ϕ_j 的值,使得 $\left|\frac{\overline{D}-T_j(\lambda_j-\phi_j)}{\overline{D}}\right| \leqslant \varepsilon$;
14. $u(j,t) \leftarrow \phi_j$; $c_{j,t} \leftarrow 0$;
15. end for
16. for 每个 $i \in V_s$ do
17. for 每个 $j \in V_t$ do
18. $E \leftarrow E \cup \{\langle i,j\rangle\}$;
19. $u(i,j) \leftarrow \min\{u(s,i),u(j,t)\}$;
20. end for
21. end for
22. 调用算法 3 在边缘容量 u(u,v)的容量约束下,计算流网络 G(V,E)中的最大流。

算法 3 最大流算法

Input:流网络 G(V,E)
Output:流网络中每一条边缘 $(i,j) \in E$ 的最大流 f(i,j)
1. $f(i,j) \leftarrow \{0 \mid (i,j) \in E\}$;
2. while G 的残存网络 G_f 中存在增广路径 p do
3. 沿 p 增加流;
4. return f(i,j)

我们使用二分迭代法计算 $\overline{D}$,使得来自过载微云的传出任务需求在来自欠载微云的传入任务需求的给定精度阈值 δ 内;然后使用算法 3 计算从过载微云到欠载微云的特定任务流;最终通过迭代计算求得由任务流引起的总网络延迟,直到 $\overline{D}$ 的值在 $\overline{D}'$ 的精度阈值 θ 内,其中 $\overline{D}' = \max\{D(j) \mid j \in V_t\}$。

4.3 微云最大负载算法

为了与 CWB 算法进行对比,我们引入了 CMW 算法。CMW 算法根据组成微云的服务器数量以及服务率来确定微云的最大抵达率,超出部分将被卸载到远程云端处理,以保证卸载到微云上的任务的响应时间不大于卸载到远程云端的响应时间。我们假定任务从微云传输到云端产生的延时成本都为 d_{cloud},即当微云上的任务响应时间大于 d_{cloud} 时,应该卸载部分任务到远程云端。该算法的细节在算法 4 中给出。

算法 4 CMW 算法

Input:每个微云 $i \in \{1,\cdots,K\}$ 的任务抵达率 λ_i,服务器数量 n_i,服务率 μ_i,微云与远程云端之间的传输成本 d_{cloud}
Output:每个微云 $i \in \{1,\cdots,K\}$ 的最大抵达率阈值 ζ_i
1. for 每个微云 $i \in \{1,\cdots,K\}$ do
2. if $T_i(\lambda_i) > d_{cloud}$
3. 计算 ζ_i 的值使得 $T_i(\zeta_i) = d_{cloud}$;
4. else
5. $\zeta_i = \lambda_i$;
6. end if
7. end for

CMW 算法下各微云产生的总延时为:

$$T_{CMW\_i} = \begin{cases} T_i(\lambda_i), & \lambda_i \leqslant \zeta_i \\ T_i(\zeta_i) + d_{cloud}, & \lambda_i > \zeta_i \end{cases}$$

5 实验结果及分析

本节将在仿真环境中评估所提出算法的性能。首先解释模拟参数的设置,然后使用模拟网络评估所提算法的性能。

5.1 实验环境

首先解释微云网络的结构,我们采用类似于文献[13]中的方法来构建 WMAN。假定网络中两个 AP 之间的网络延迟与其物理距离成正比。由于实际环境下 AP 之间的距离是随机的,因此我们根据正态分布 $0.1 \leqslant N(0.15,0.05) \leqslant 0.2$ 随机分配每对直接连接的微云之间的网络延迟。这使得网络中的延迟随机化,同时保持三角形距离不等式,因为跨度为 2 的任何一对节点具有至少 0.2 的中间距离。

对于每个微云 i,根据正态分布 $N(5,2)>0$ 分配其服务速率 μ_i,并根据均值为 3 的泊松分布分配服务器数量。微云 i 的任务抵达率 λ_i 由正态分布 $0<N(15,6)<\mu_i \cdot n_i - 0.25$ 确定。值得注意的是,抵达率 λ_i 不得超过 $\mu_i \cdot n_i$,否则微云 i 的队列时间将是无限长的。

5.2 实验结果分析

从上文可以得知,当算法中的精度阈值选取不当时,可能会导致欠载微云的传入任务不足或过载微云的传出任务过多,因此我们需要在等同条件下进行多次实验模拟以找到较为合适的精度阈值。图 5(a)绘制了精度阈值 θ 的大小与微云平均任务响应时间之间的关系。当 $\theta<0.2$ 时,平均响应时间单调递减;当 $\theta>0.2$ 时,平均响应时间单调递增。因此,应当选取 0.2 为后续实验的 θ 值。

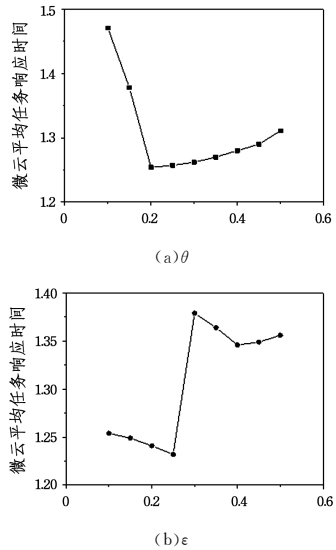


图 5 精度阈值 θ 和 ε 对任务响应时间的影响

Fig. 5 Effect of accuracy threshold θ and ε on task response time

同样地，精度阈值 ε 通过控制传出/传入任务的大小来影响微云的平均任务响应时间。

图 5(b)绘制了精度阈值 ε 的大小与微云平均任务响应时间之间的关系。从中可以看出，当设置 ε 的值为 0.25 时，微云的平均响应时间最小；当 ε 的值大于 0.25 时，平均任务响应时间虽然有所波动，但并未取到最小值。因此，后续实验将 ε 值定为 0.25。

我们对实验数据行进抽样，抽样数据如表 1 所列。

表 1　实验数据抽样

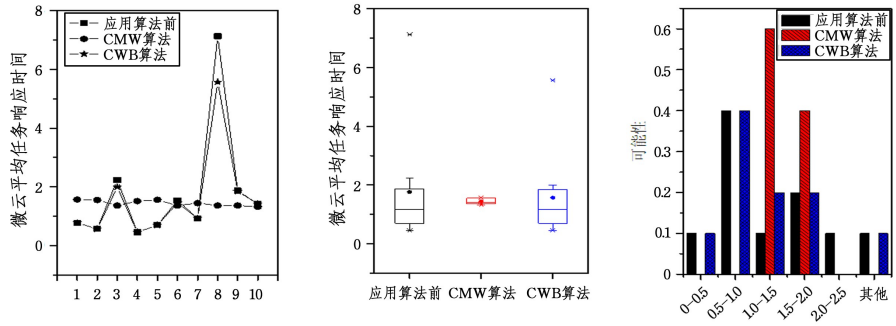
Table 1　Experimental data sampling

	抽样 1	抽样 2	抽样 3	抽样 4	抽样 5	抽样 6	抽样 7	抽样 8	抽样 9	抽样 10	均值
应用算法前	0.780	0.571	2.236	0.459	0.702	1.539	0.929	7.127	1.876	1.421	1.764
CMW 算法	0.780	0.571	1.998	0.459	0.702	1.455	0.929	5.567	1.850	1.421	1.573
CWB 算法	1.571	1.554	1.365	1.518	1.562	1.366	1.448	1.369	1.367	1.326	1.445

图 6(a)绘制了 3 种情况下抽样点的数据变化情况，从中可以看出：应用算法之前，由于微云初始负载的差异巨大，所产生的微云平均任务响应时间也迥然不同；应用 CMW 算法之后，初始负载较大的抽样点的微云平均任务响应时间得到了改善，而初始负载较小的抽样点保持原状态；应用 CWB 算法之后，虽然初始负载较小的抽样点的微云平均任务响应时间有所增加，但是优化了初始负载较大的抽样点的表现，所有抽样点的任务响应时间较为接近，符合我们设计算法的初衷。图 6(b)绘制了应用算法前以及应用两种算法后的任务响应时间的箱型图，图 6(c)则绘制了 3 种情况下微云任务平均响应时间的概率分布柱状图。

从两者中可以看出，在不使用算法的情况下，微云的任务响应时间在数值上较为分散，跨度也较大；而在应用 CMW 算法和 CWB 算法之后，任务响应时间大部分集中在一个固定的区间之内，显然，应用 CWB 算法之后的区间更小。该区间的跨度明显大于我们给定的精度阈值，原因是网络延迟会导致过载微云的传出任务需求与欠载微云的传入任务需求并不匹配。对实验数据进行计算后可得，应用 CMW 算法之后的平均任务响应时间较应用算法之前下降了 10.8%，而应用 CWB 算法后则使平均任务响应时间下降了 18.1%。

此外，从抽样数据中可以看出，应用 CMW 算法后网络中各微云的负载更加均衡，即网络中不会出现微云过载或欠载的情况。



(a)数据抽样　　(b)微云任务响应时间的箱型图　　(c)微云任务响应时间的概论分布

图 6　算法的性能评估

Fig. 6　Performance evaluation of algorithms

结束语　微云技术是边缘计算中的一项重要应用，它可以提升移动应用程序的性能，改善应用程序的用户体验。随着无线互联网的普及，公共计算资源的利用对未来移动计算技术的发展具有重要意义。文本研究了无线城域网环境下的微云负载均衡问题，提出了一种快速且可扩展的启发式算法，以缩短网络中任务的平均响应时间。实验结果表明：该算法在减少任务响应时间方面具有积极的作用。

通过对实验结果的分析可以发现，所提算法虽然对负载较大的微云的优化有积极作用，但也一定在程度上增加了原本负载较小的微云的任务平均响应时间。未来研究工作中，我们希望利用更高级的算法来优化微云负载均衡问题，如遗传算法、粒子群算法等，并对这些高级算法进行改进，以使其更加适用于微云负载均衡问题。

参 考 文 献

[1] PANG Z, SUN L, WANG Z, et al. A survey of cloudlet based mobile computing[C]// 2015 International Conference on Cloud Computing and Big Data (CCBD). 2015: 268-275.

[2] CUERVO E, BALASUBRAMANIAN A, CHO D K, et al. MAUI: making smartphones last longer with code offload[C]// Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. 2010: 49-62.

[3] XIA Q, LIANG W, XU W. Throughput maximization for online request admissions in mobile cloudlets[C]// 38th Annual IEEE Conference on Local Computer Networks. 2013: 589-596.

[4] SATYANARAYANAN M. Pervasive computing: Vision and challenges[J]. IEEE Personal communications, 2001, 8(4): 10-17.

[5] SATYANARAYANAN M, BAHL P, CACERES R, et al. The case for vm-based cloudlets in mobile computing[J]. IEEE Pervasive Computing, 2009, 8(4): 14-23.

[6] VERBELEN T, SIMOENS P, DE TURCK F, et al. Cloudlets: Bringing the cloud to the mobile user[C]// Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services. 2012: 29-36.

[7] VERBELEN T, SIMOENS P, DE TURCK F, et al. Leveraging cloudlets for immersive collaborative applications[J]. IEEE Pervasive Computing, 2013, 12(4): 30-38.

[8] XU Z, LIANG W, XU W, et al. Capacitated cloudlet placements in wireless metropolitan area networks[C]// 2015 IEEE 40th Conference on Local Computer Networks(LCN). 2015: 570-578.

[9] XU Z, LIANG W, XU W, et al. Efficient algorithms for capacitated cloudlet placements[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(10): 2866-2880.

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

[11] HA K, PILLAI P, LEWIS G, et al. The impact of mobile multimedia applications on data center consolidation[C]// 2013 IEEE international conference on cloud engineering(IC2E). 2013: 166-176.

[12] HU W, GAO Y, HA K, et al. Quantifying the impact of edge computing on mobile applications[C]// Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems. 2016: 5.

[13] 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]// 2012 IEEE International Conference on Pervasive Computing and Communications. 2012: 122-127.

[14] 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, 5(4): 725-737.

[15] BALAN R, FLINN J, SATYANARAYANAN M, et al. The case for cyber foraging[C]// Proceedings of the 10th workshop on ACM SIGOPS European workshop. 2002: 87-92.

[16] CHUN B-G, IHM S, MANIATIS P, et al. Clonecloud: elastic execution between mobile device and cloud[C]// Proceedings of the Sixth Conference on Computer Systems. 2011: 301-314.

[17] 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. 2012: 945-953.

[18] CHEN E Y, ITOH M. Virtual smartphone over IP[C]// 2010 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks(WoWMoM). 2010: 1-6.

[19] HOANG D T, NIYATO D, WANG P. Optimal admission control policy for mobile cloud computing hotspot with cloudlet[C]// 2012 IEEE Wireless Communications and Networking Conference (WCNC). 2012: 3145-3149.

[20] XIA Q, LIANG W, XU Z, et al. Online Algorithms for Location-Aware Task Offloading in Two-Tiered Mobile Cloud Environments[C]// IEEE/ACM International Conference on Utility & Cloud Computing. 2014.

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

[22] GELENBE E, LENT R, DOURATSOS M. Choosing a local or remote cloud[C]// 2012 Second Symposium on Network Cloud Computing and Applications. 2012: 25-30.

[23] HA K, PILLAI P, RICHTER W, et al. Just-in-time provisioning for cyber foraging[C]// Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services. 2013: 153-166.

[24] REDINBO G. Queueing Systems, Volume I: Theory-Leonard Kleinrock[J]. IEEE Transactions on Communications, 2003, 25(1): 178-179.