计算机科学
COMPUTER SCIENCE

用户公平保障的边缘服务缓存与任务卸载算法

吴纯, 陈龙, 孙一飞, 武继刚

引用本文

吴纯, 陈龙, 孙一飞, 武继刚. 用户公平保障的边缘服务缓存与任务卸载算法[J]. 计算机科学, 2023, 50(11A): 230200095-8.

WU Chun, CHEN Long, SUN Yifei, WU Jigang. Fairness-aware Service Caching and Task Offloading with Cooperative Mobile Edge Computing [J]. Computer Science, 2023, 50(11A): 230200095-8.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230200095

用户公平保障的边缘服务缓存与任务卸载算法

吴 纯 陈 龙 孙一飞 武继刚
广东工业大学计算机学院 广州 510006
(1366988186@qq.com)

摘 要 在边缘服务器中缓存服务可缩短请求响应时间,提升用户体验。现有研究主要从整体上优化系统性能,例如最大化系统吞吐量,而无法保障个体用户请求异构服务的公平性。针对用户异构计算任务的不公平服务问题,研究边缘协同计算中的服务缓存和任务卸载策略,基于最大最小公平原则,构建了一个最大化最小服务完成率问题,并证明了其 NP 难解性。为此,利用线性松弛将原问题从 0-1 整数规划转化为线性规划,设计了一种近似比为 $M\sqrt{S}/N(\sqrt{S}-\sqrt{2\ln S})$ 的随机舍入算法,其中 S 为边缘服务器数,N 为服务数,M 为终端设备数。同时,基于优先缓存和卸载完成率最小的服务及其任务,提出了一种快速高效的贪心算法。实验结果表明,与已有最大化系统吞吐量算法相比,提出的随机舍入算法和贪心算法将最小服务完成率分别提升至少 44.1%和 90.6%,并且其额外的系统吞吐量损失分别不超过 22.4%和 27.0%。

关键词:边缘计算;服务缓存;任务卸载;最大最小公平;随机舍入
中图法分类号 TP391

Fairness-aware Service Caching and Task Offloading with Cooperative Mobile Edge Computing

WU Chun,CHEN Long,SUN Yifei and WU Jigang
School of Computer Science and Technology,Guangdong University of Technology,Guangzhou 510006,China

Abstract Caching services in edge servers can reduce the response time of user requests and improve user experience. Most of existing works focus on optimizing the overall system performance, i.e., maximizing the system throughput, which cannot guarantee the user fairness in requesting heterogeneous services. To fill this gap, this paper investigates fairness-aware joint service caching and task offloading strategy with cooperative edge computing. A minimum service completion rate maximization problem is formulated based on max-min fairness principle, which is proved to be NP-hard. A randomized rounding algorithm with $M\sqrt{S}/N(\sqrt{S}-\sqrt{2\ln S})$-approximation ratio is proposed by transforming the original problem from 0-1 integer programming into linear programming using linear relaxation, where S, N and M are the numbers of edge servers, services and end devices, respectively. Moreover, a fast and efficient greedy algorithm is proposed by caching the service with the minimum completion rate and offloading its corresponding tasks preferentially. Extensive simulation results demonstrate that not only the minimum service completion rate can be improved by at least 44.1% and 90.6% by our two algorithms, but also the extra loss of system throughput is no more than 22.4% and 27.0%, respectively, compared with the existing algorithms for maximizing system throughput.

Keywords Edge computing, Service caching, Task offloading, Max-Min fairness, Randomized rounding

1 引言

智能设备的爆炸增长以及物联网的蓬勃发展给公众带来了许多计算密集型和时延敏感型的新兴应用,例如目标识别、虚拟现实以及在线游戏等[1]。受限于自身的计算存储资源,终端设备无法及时处理用户请求产生的计算任务[2]。作为一种新兴计算范式,边缘计算在网络边缘部署边缘服务器,可为终端设备提供低延迟的计算存储资源[3]。如果边缘服务器缓存了相应服务(指服务的可执行程序和数据),那么终端设备可将自身的任务卸载到边缘服务器进行处理[4]。例如,如果一台边缘服务器缓存了目标检测服务,那么连接该边缘服务器的终端设备可将自身的目标检测任务卸载到边缘服务器处理并迅速得到响应,这将有利于支撑自动驾驶或视频分析等时延敏感型应用[5]。此外,单台边缘服务器的计算存储资源仍是有限的,合理利用周围邻近边缘服务器的算力,边缘服务器可协同地为终端设备提供计算、存储服务。

现有研究主要是在边缘服务器计算存储资源的约束下,联合优化服务缓存和任务卸载以最大化系统吞吐量[6]、最小化请求平均响应时间[7]等。实际应用中服务是异构的[8],不同服务所需的存储空间、输入的数据量以及任务的计算量是不同的。例如用于图片识别的 VGG16 相比 AlexNet 模型复杂度更高,计算量更大[9]。在服务异构的情况下,现有研究从整体上优化系统性能的策略,会引发个体用户请求异构服务的不公平问题。例如,若以最大化系统吞吐量为优化目标,由

基金项目:国家自然科学基金(62072118,62202108);广东省自然科学基金(2023A1515011230)
This work was supported by the National Natural Science Foundation of China(62072118,62202108) and Natural Science Foundation of Guangdong Province,China(2023A1515011230).
通信作者:武继刚(asjgwucn@outlook.com)

于系统的计算存储资源有限，为充分提高系统吞吐量，计算量小的用户任务相比计算量大的用户任务，应更优先被卸载处理。在此情况下，系统中的多数小任务能被处理完成，而大任务的处理完成数则偏少，从而造成计算复杂程度较低的服务相比计算复杂程度较高的服务具备更高的服务完成率。

由此可知，在服务异构的情况下，最优化系统吞吐量的策略会导致服务完成率的差距较大，从而引发用户异构计算任务的不公平服务问题。用户是自私的，都希望自身任务能优先满足，因此用户公平性也是一个重要指标，但现有的多数研究都忽略了这点。因此，为确保用户请求异构服务的公平性，如何制定高效的服务缓存与任务卸载策略，以在提升服务完成率的同时维持不同服务完成率的均衡性，是亟待解决的关键问题。值得一提的是，在集成学习(Ensemble Learning，EL)领域[10]，一个推理任务的完成需依赖多个其他任务的结果，因此服务完成率的均衡也对集成学习模型推理效率的提升有积极意义。本文研究边缘协同计算中较少涉及但仍十分重要的用户公平性问题，主要的挑战在于服务缓存与任务卸载是紧密联系的，两者需共同决策。本文的贡献总结如下：

1)从保证用户公平性的新颖角度出发，提出了一种边缘协同计算中的服务缓存与任务卸载联合优化问题。在边缘服务器计算存储资源的约束下，最大化最小的服务完成率，并证明了所提问题的 NP 难解性。

2)通过线性松弛将 0-1 整数规划的原问题转化为线性规划问题，设计了一种近似比为 $M\sqrt{S}/N(\sqrt{S}-\sqrt{2\ln S})$ 的随机舍入算法，其中 S 为边缘服务器数，N 为服务数，M 为终端设备数。此外，基于完成率最小的服务优先被缓存，其任务优先被卸载的原则，设计了一种快速高效的贪心算法。

3)实验结果表明，与最大化系统吞吐量的算法相比，提出的随机舍入算法和贪心算法能在系统吞吐量损失较小的情况下大幅提升最小的服务完成率。

2 相关工作

在边缘设备计算存储资源的限制下，针对最优化某项或多项指标，研究人员提出了一系列关于服务缓存与任务卸载的优化技术。

在优化系统吞吐量方面，文献[6]综合考虑了边缘服务器的计算、存储以及带宽资源约束，提出了一种基于随机舍入的近似算法。为最大化系统吞吐量，文献[11]研究边缘协同计算中的模型缓存与任务卸载，考虑 DNN 模型加载时延的影响及用户对 DNN 推理任务的精度要求，提出了一种基于随机舍入的贪心算法。文献[12]面向边缘云场景，提出了一种联合服务缓存与请求调度的两阶段框架。结合启发式的请求调度方案，该文献首先提出了一种服务缓存的近似算法，然后在服务缓存方案固定的基础上，还提出了一种基于最大流的请求调度算法。

在优化任务时延方面，为最小化任务的平均完成时延，文献[7]面向边缘服务器协同计算的场景，引入广义 benders 分解解决服务缓存与任务卸载的强耦合关系，分别提出了固定服务缓存状态的任务卸载子算法和动态调整服务缓存状态的服务缓存子算法。文献[8]研究密集网络中的服务缓存与任务卸载策略，提出了一种基于李雅普诺夫的在线算法，以在系统能耗的约束下，最小化任务的计算延迟。文献[13]面向应用服务的子任务存在依赖关系的场景，提出了一种基于凸优化的服务缓存与任务卸载算法，以最小化任务的总完成时间。

在优化设备能耗方面，文献[14]研究单个边缘服务器场景中的服务缓存与任务卸载策略，以最优化用户的时延和能耗。文献[15]针对服务具体为 DNN 模型的场景，提出了一种启发式的服务缓存与请求调度算法，以在模型请求时延的约束下最小化边缘设备的能耗。文献[16]面向车辆边缘计算中服务任务存在依赖关系的场景，综合考虑车辆数据传输和本地计算的能源消耗，提出了一种基于动态规划的半分布式服务缓存与任务卸载算法，以降低车辆的能耗。

然而，实际中服务是异构的，上述研究提出的服务缓存与任务卸载策略仅是从整体上优化系统性能，无法较好地保证个体用户请求异构服务的公平性。虽然文献[17]研究了联合任务卸载和资源分配的用户公平性问题，但未考虑服务缓存的过程，并假设边缘服务器的资源能满足所有用户，这不符合实际。此外，文献[18]引入任务完成率约束，以保证用户请求异构服务的公平性，但实际中任务完成率的最小值通常是难以预知的。因此，本文面向边缘服务器协同计算的场景，考虑服务的异构性，提出了两种高效的服务缓存与任务卸载算法，以保证用户的异构计算任务能获得公平的服务。

3 系统模型

如图 1 所示，无线网络中存在多个网络基站和终端设备，终端设备随机分布在基站的覆盖范围内。基站通过高速有线链路连接，每个基站部署了边缘服务器。边缘服务器间可多跳通信，协作地处理终端设备的计算任务。边缘服务器间是连通的，设 $G(V,E)$ 表示边缘服务器构成的连通图，其中 V 代表边缘服务器形成的顶点集合，E 代表边缘服务器的边集合。令 $\mathcal{S}$ 和 $\mathcal{M}$ 分别表示边缘服务器和终端设备的集合。由于终端设备与中心云服务器的时延较大，为满足用户需求，需将服务缓存到边缘服务器中。如果边缘服务器缓存了相应的服务，那么该边缘服务器就能处理请求该服务的任务。设 $\mathcal{N}$ 表示服务的集合，对于 $\forall n\in\mathcal{N}$，采用三元组 $\langle d_n,b_n,l_n\rangle$ 来描述其属性，其中 d_n 表示服务 n 的数据输入量，b_n 表示服务 n 处理单次请求任务的计算量，l_n 表示用户请求任务对服务 n 的最大容忍时延。所有服务是异构的，不同服务所需的存储空间不同，大的服务一般具备更大的计算量。

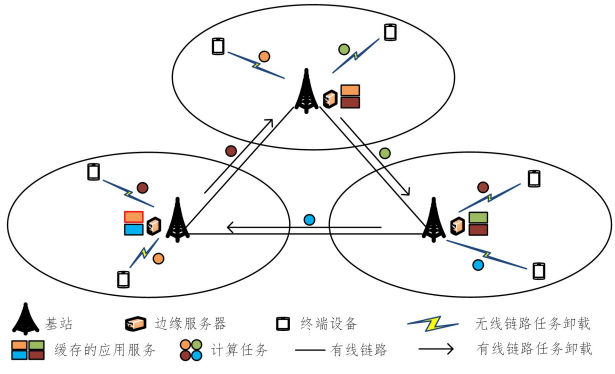


图 1 系统模型

Fig. 1 System model

3.1 服务缓存模型

设 $x_{s,n}\in\{0,1\}$ 表示服务缓存策略，服务 n 缓存在边缘服务器 s 中，记作 $x_{s,n}=1$，否则 $x_{s,n}=0$。由于存储空间的限制，

边缘服务器不可能同时缓存所有服务,因此有:

$$\sum_{n\in\mathcal{N}} x_{s,n}\cdot c_n\leqslant R_s,\ \forall\, s\in\mathcal{N} \tag{1}$$

其中,c_n 表示服务 n 的大小,R_s 表示边缘服务器 s 的存储空间大小。

3.2　任务卸载模型

假设每个终端设备每次请求一个任务,每个任务对应一个服务。该假设可很好地适用于一个终端设备同时请求多个任务的场景,只需将每个任务视作不同的终端设备处理即可。设 $z_{m,n}$ 为表示终端设备 m 是否请求服务 n 的二元变量,则有:

$$\sum_{n\in\mathcal{N}} z_{m,n}=1,\ \forall\, m\in\mathcal{M} \tag{2}$$

表 1　文中涉及的符号及其含义

Table 1　Symbols and descriptions

符号	具体含义
$\mathcal{S}$	边缘服务器集合
$\mathcal{M}$	终端设备集合
$\mathcal{N}$	服务集合
d_n	服务 n 的数据输入量
b_n	服务 n 单次任务的计算量
l_n	用户请求服务 n 的最大容忍时延
$x_{s,n}$	服务 n 是否缓存在边缘服务器 s
$y_{s,m}$	终端设备 m 的任务是否卸载到边缘服务器 s
c_n	服务 n 的大小
R_s	边缘服务器 s 的存储空间大小
$z_{m,n}$	终端设备 m 是否请求服务 n
$\mathcal{M}_s$	连接边缘服务器 s 的终端设备集合
$\gamma_{s,m}$	终端设备 m 往边缘服务器 s 的数据传输速率
s_m	终端设备 m 连接的边缘服务器
$v_{i,j}$	相连边缘服务器 i 和 j 之间的数据传输速率
$E(i,j)$	边缘服务器 i 和 j 之间最短路径的边集合
$c_{i,j}$	相连边缘服务器 i 和 j 之间的延迟
$t_{s,m}^{\text{trans}}$	设备 m 将任务卸载到服务器 s 的传输时延
$t_{s,m}^{\text{comp}}$	设备 m 的任务在边缘服务器 s 的计算时延
$t_{s,m}$	设备 m 的任务卸载到边缘服务器 s 的总时延
σ_m	设备 m 的任务是否成功
A_n	请求服务 n 的终端设备集合
a_n	请求服务 n 的终端设备成功数
g_n	服务 n 的完成率
$\mathcal{M}^n$	请求服务 n 但未被卸载的终端设备集合

将终端设备 m 的请求任务卸载到边缘服务器 s 进行处理,记作 $y_{s,m}=1$,否则 $y_{s,m}=0$。由于每个终端设备的任务都必须卸载到一个边缘服务器,因此有:

$$\sum_{s\in\mathcal{S}} y_{s,m}=1,\ \forall\, m\in\mathcal{M} \tag{3}$$

此外,终端设备的任务只能卸载到已经缓存相应服务的边缘服务器中,因此有:

$$y_{s,m}\cdot z_{m,n}\leqslant x_{s,n},\ \forall\, s\in\mathcal{S},m\in\mathcal{M},n\in\mathcal{N} \tag{4}$$

3.3　通信模型

终端设备可通过正交频分多址(Orthogonal Frequency Division Multiple Access,OFDMA)的方式与其连接的边缘服务器进行无线通信[11]。设 $\mathcal{M}_s$ 表示连接到边缘服务器 s 的终端设备集合,$\mathcal{M}_s$ 中的终端设备 m 与边缘服务器 s 之间的数据传输速率可由香农公式得到。

$$\gamma_{s,m}=W\log_2\left(1+\frac{p_m\cdot H_{s,m}}{WN_0}\right),\ \forall\, s\in\mathcal{S},m\in\mathcal{M} \tag{5}$$

其中,W 表示无线信道带宽,p_m 表示终端设备 m 的发送功率,$H_{s,m}$ 表示边缘服务器 s 与终端设备 m 之间的信道增益,N_0 表示高斯白噪声的频谱密度。信道增益的大小可由 $H_{s,m}=(D_{s,m})^{-\eta}|h_{s,m}|^2$ 给出,其中 $D_{s,m}$ 表示边缘服务器 s 与终端设备 m 之间的欧几里得距离,η 为常数 4,$h_{s,m}$ 为相应的瑞利信道衰减因子[11]。特别地,当 $m\notin\mathcal{M}_s$ 时,$\gamma_{s,m}=0$。

任务的卸载时延主要包括两个部分,即终端设备到连接边缘服务器的数据传输时延以及连接边缘服务器到目标边缘服务器的多跳传输时延。设 s_m 为终端设备 m 的连接边缘服务器,$v_{i,j}$ 为相连边缘服务器 i 和 j 之间的数据传输速率,$E(i,j)$ 为边缘服务器 i 与 j 之间最短路径的边集合,$c_{i,j}$ 为相连边缘服务器 i 和 j 之间的延迟,则终端设备 m 将任务卸载到边缘服务器 s 的时延为:

$$t_{s,m}^{\text{trans}}=\frac{\sum_{n\in\mathcal{N}} z_{m,n}d_n}{\gamma_{m,s_m}}+\sum_{e_{i,j}\in E(s_m,s)}\left(\frac{\sum_{n\in\mathcal{N}} z_{m,n}d_n}{v_{i,j}}+c_{i,j}\right),\ \forall\, s,i,j\in\mathcal{S},m\in\mathcal{M} \tag{6}$$

由于任务的计算结果数据量较小,沿用文献[17]的处理方法,本文也忽略了边缘服务器将计算结果返回给终端设备的时延。

3.4　计算模型

任务的计算时延与边缘服务器分配的计算资源大小成反比,设 $f_{s,m}$ 表示边缘服务器 s 分配给终端设备 m 的计算资源,则终端设备 m 的任务在边缘服务器 s 上的计算时延为:

$$t_{s,m}^{comp}=\frac{\sum_{n\in\mathcal{N}} z_{m,n}\cdot b_n}{f_{s,m}},\ \forall\, s\in\mathcal{S},\ m\in\mathcal{M} \tag{7}$$

此外,由于边缘服务器的计算资源是有限的,设 C_s 表示边缘服务器 s 的计算资源大小,则有:

$$\sum_{m\in\mathcal{M}} y_{s,m}\cdot f_{s,m}\leqslant C_s,\ \forall\, s\in\mathcal{S} \tag{8}$$

因此,将终端设备 m 的请求任务卸载到边缘服务器 s 的总响应时延为:

$$t_{s,m}=t_{s,m}^{\text{trans}}+t_{s,m}^{\text{comp}},\ \forall\, s\in\mathcal{S},\ m\in\mathcal{M} \tag{9}$$

3.5　问题定义

设二元变量 σ_m 表示终端设备 m 的请求任务是否可以执行成功,即任务的响应时延是否满足终端设备的需求,则有:

$$\sigma_m=\begin{cases}1, & if\ \sum_{s\in\mathcal{S}} y_{s,m}\cdot t_{s,m}\leqslant\sum_{n\in\mathcal{N}} z_{m,n}\cdot l_n\\ 0, & \text{otherwise}\end{cases} \tag{10}$$

设 A_n 表示请求服务 n 的终端设备集合,则有:

$$A_n=\{m,\ \text{if}\ z_{m,n}=1,m\in\mathcal{M}\},\ \forall\, n\in\mathcal{N} \tag{11}$$

由于计算存储资源的限制,边缘服务器不可能同时满足所有终端设备的需求。设 a_n 表示请求服务 n 的终端设备成功数,则有:

$$a_n=\sum_{m\in\mathcal{M}} z_{m,n}\sigma_m,\ \forall\, n\in\mathcal{N} \tag{12}$$

定义 1　服务 n 的完成率 g_n 为请求服务 n 的终端设备成功数与请求服务 n 的终端设备总数的比值,即:

$$g_n=\frac{a_n}{|A_n|},\ \forall\, n\in\mathcal{N} \tag{13}$$

正如前文所述,由于服务异构,在宏观优化中,请求小服务的用户比请求大服务的用户往往具有更高的完成率。为保证用户请求异构服务的公平性,基于最大最小公平性原则,本文提出如下最大化最小服务完成率问题:

$$(\mathcal{P})\ \max\min_{n\in\mathcal{N}}\{g_n\}$$

$$\text{s.t.}\ (1),(3),(4),(8)$$

$$x_{s,n},y_{s,m}\in\{0,1\},$$

$$\forall\, s\in\mathcal{S},m\in\mathcal{M},n\in\mathcal{N}$$

其中,(1)和(8)分别表示边缘服务器的存储和计算约束,(3)表示终端设备的任务必须卸载到一个边缘服务器处理的

约束,(4)表示任务必须卸载到已经缓存相应服务的边缘服务器约束。

定理 1 $\mathscr{P}$是 NP 难问题。

证明:多背包问题(Multi Knapsack Problem,MKP)是经典的 NP 完全问题[17],本文通过展示 MKP 是 $\mathscr{P}$的一个特例,证明 $\mathscr{P}$属于 NP 难问题。引入变量 δ,满足 $\delta \leqslant g_n, \forall n \in \mathscr{N}$,则 $\mathscr{P}$的目标等价于最大化 δ。假设服务是同构的,即只有一个服务,且服务已预先缓存在所有边缘服务器中,即移除约束(1)和(4)。于是 $\delta = g_n$,$\mathscr{P}$的目标变为 $\max g_n$。若将任务看作物品,每个边缘服务器看作背包,物品价值为 $1/|A_n|$,物品大小为完成任务所需的计算资源,背包容量为边缘服务器的计算能力,则 $\mathscr{P}$的这个特例可视为 MKP。由于 $\mathscr{P}$比 MKP 更复杂,且 MKP 是 NP 完全问题,则 $\mathscr{P}$是 NP 难问题。

4 问题求解

4.1 问题转化

从问题目标和式(7)可知,问题 $\mathscr{P}$是非连续且非线性的,十分难求解。为解决非线性,引入连续变量 $w_{s,m}$,表示终端设备 m 的任务卸载到边缘服务器 s 成功执行需要的最小计算资源,则:

$$w_{s,m} = \frac{\sum_{n \in \mathscr{N}} z_{m,n} \cdot b_n}{\sum_{n \in \mathscr{N}} z_{m,n} \cdot l_n - t_{s_m^{\text{trans}},s}}, \forall s \in \mathscr{S}, m \in \mathscr{M} \tag{14}$$

为服务尽可能多的终端设备,如果终端设备 m 的任务卸载到边缘服务器 s 中,则 s 分配给终端设备 m 的计算资源大小应为 $w_{s,m}$。因此有:

$$\sum_{s \in \mathscr{S}} y_{s,m} \leqslant 1, \ \forall m \in \mathscr{M} \tag{15}$$

$$\sum_{m \in \mathscr{M}} y_{s,m} \cdot w_{s,m} \leqslant C_s, \forall s \in \mathscr{S} \tag{16}$$

同时,σ_m 的计算式可以变为:

$$\sigma_m = \sum_{s \in \mathscr{S}} y_{s,m} \tag{17}$$

为解决非连续性,引入连续变量 λ,且满足:

$$\lambda \leqslant g_n, \forall n \in \mathscr{N} \tag{18}$$

同时松弛问题 $\mathscr{P}$的二元整型约束,则可将整数规划问题 $\mathscr{P}$转化为线性规划问题 P-LP。

(P-LP) $\max \lambda$

s. t. (1),(4),(15),(16),(18)

$x_{s,n}, y_{s,m} \in [0,1]$,

$\forall s \in \mathscr{S}, m \in \mathscr{M}, n \in \mathscr{N}$

线性规划 P-LP 的最优解可利用主流线性规划求解器求解。然而,P-LP 的最优解是分数解,并不是原问题 $\mathscr{P}$的解。由文献[19]可知,随机舍入算法以概率舍入分数解来适应原问题,可在多项式时间复杂度内求解 NP 难问题,且求解的质量有一定理论保证。因此,本节提出了一种随机舍入算法,并证明其近似比。随机舍入算法需求解线性规划,其时间复杂度为变量个数的立方函数[20]。针对随机舍入算法时间复杂度较高的缺陷,本节还提出了一种快速高效的贪心算法。

4.2 随机舍入算法

由于 P-LP 的最优解是分数解,并不是原问题 $\mathscr{P}$的解,需对 P-LP 的解舍入来适应原问题 $\mathscr{P}$。基于此思想,提出了一种随机舍入算法,算法的伪代码如算法 1 所示。算法 1 的思想与步骤如下:

步骤 1 使用主流线性规划求解器求解问题 P-LP 得到最优解 $x_{s,n}^{\dagger}$和 $y_{s,m}^{\dagger}$。

步骤 2 根据 $x_{s,n}^{\dagger}$和 $y_{s,m}^{\dagger}$进行舍入操作,以 $x_{s,n}^{\dagger}$的概率将服务 n 缓存在边缘服务器 s 中,计算 $y_{s,m,n}$。

$$y_{s,m,n} = \frac{y_{s,m}^{\dagger} \cdot z_{m,n}}{x_{s,n}^{\dagger}} \tag{19}$$

如果 $x_{s,n}=1$,则以 $y_{s,m,n}$的概率将终端设备 m 的请求任务卸载到边缘服务器 s。

步骤 3 针对每个边缘服务器 s,按照 $x_{s,n}^{\dagger}$的升序将服务 n 从边缘服务器 s 移除,直到满足约束(1),按照 $y_{s,m,n}$的升序将终端设备 m 的任务从边缘服务器 s 移除,直到满足约束(16)。

步骤 4 针对每个边缘服务器 s,在满足约束(1)的情况下,按照 $x_{s,n}^{\dagger}$的降序将服务 n 缓存到边缘服务器 s;在满足约束(4)和约束(16)的情况下按照 $y_{s,m,n}$的降序将终端设备 m 的任务卸载到边缘服务器 s。

步骤 5 针对每个终端设备 m,按照 $y_{s,m,n}$的升序将终端设备 m 的任务从边缘服务器 s 移除,直到满足约束(15)。

步骤 6 从完成率最小的服务中随机选择服务 n,对于每个终端设备 m,如果 m 的任务未卸载,且 $m \in A_n$,则查询每个边缘服务器 s,检查 m 的任务是否能卸载至 s,如果存在 m 的任务能卸载到 s,则将 m 的任务卸载到 s,重新执行步骤 6,否则算法终止。

算法 1 随机舍入算法(Randomized Rounding Algorithm,ROAG)

输入:边缘服务器集合 $\mathscr{S}$,终端设备集合 $\mathscr{M}$,服务集合 $\mathscr{N}$,终端设备请求 $z_{m,n}$

输出:服务缓存策略 $x_{s,n}$,任务卸载策略 $y_{s,m}$

1. 求解问题 P-LP 得到 $x_{s,n}^{\dagger}$和 $y_{s,m}^{\dagger}$;
2. 以 $x_{s,n}^{\dagger}$的概率 $x_{s,n} := 1$;
3. $x_{s,n}=1$ 时以 $y_{s,m,n}$的概率 $y_{s,m} := 1$;
4. while $\forall s \in \mathscr{S}$违背约束(1) do
5. $n^{\dagger} := \text{argmin } x_{s,n}^{\dagger}, x_{s,n}^{\dagger} := 0$
6. end while
7. while $\forall s \in S$ 违背约束(16) do
8. $m^{\dagger} := \text{arg min } y_{s,m,n}, y_{s,m^{\dagger}} := 0$;
9. end while
10. for $s \in \mathscr{S}$ do
11. 按 $x_{s,n}^{\dagger}$降序查服务,即
$n^{\dagger} := \text{arg max } x_{s,n}^{\dagger}$,满足式(1)的情况下
$x_{s,n^{\dagger}} := 1$;
12. 按 $y_{s,m,n}$降序查任务,即
$m^{\dagger} := \text{arg max } y_{s,m,n}$,满足式(4)和式(16)的情况下 $y_{s,m^{\dagger}} := 1$;
13. end for
14. while $\forall m \in \mathscr{M}$ 违背约束(15) do
15. $s^{\dagger} := \text{arg min } y_{s,m,n} \cdot y_{s^{\dagger},m} := 0$;
16. end while
17. flag := 1;
18. while flag do
19. flag := 0,更新 g_n;
20. 随机选择完成率最小的服务 n;
21. for $m \in \mathscr{M}, s \in \mathscr{S}$ do
22. if $z_{m,n}=1, y_{s,m}=0$ then
23. 满足式(4)、式(15)和式(16)的情况下
$y_{s,m} := 1$, flag := 1, break;

24.　　end if

25.　　end for

26. end while

27. return $x_{s,n}$, $y_{s,m}$

定理 2　算法 1 的时间复杂度为 $O((SM+SN)^3)$,其中 S 为边缘服务器数,N 为服务数,M 为终端设备数。

证明:首先用线性规划求解器求解,此步骤的时间复杂度为 $O((SM+SN)^3)$[20]。之后,遍历边缘服务器和终端设备,服务和终端设备进行随机舍入并检查约束(1)、约束(15)、约束(16),此步骤的时间复杂度为 $O(SM+SN)$。最后,遍历边缘服务器和终端设备以提升最小服务完成率,该过程的循环次数不会超过终端设备数 M,因此此步骤的时间复杂度为 $O(SM^2)$。综上,算法 1 的时间复杂度为 $O((SM+SN)^3)$。

推论 1　算法 1 随机舍入过程求得解的期望满足存储约束(1)和计算约束(16)。

证明:由于服务 n 缓存到边缘服务器 s 的概率为 $x^{\dagger}_{s,n}$,$x_{s,n}=1$ 时终端设备 m 的任务卸载到边缘服务器 s 的概率为 $y_{s,m,n}$,则有:

$$Pr(x_{s,n}=1)=x^{\dagger}_{s,n} \tag{20}$$

以及

$$\begin{aligned}Pr(y_{s,m}=1)&=\sum_{n\in\mathcal{N}} y_{s,m,n}\cdot Pr(x_{s,n}=1)\\&=\sum_{n\in\mathcal{N}}\frac{y^{\dagger}_{s,,m}\cdot z_{m,n}}{x^{\dagger}_{s,n}}\cdot Pr(x_{s,n}=1)\\&=\sum_{n\in\mathcal{N}}\frac{y^{\dagger}_{s,,m}\cdot z_{m,n}}{x^{\dagger}_{s,n}}\cdot x^{\dagger}_{s,n}\\&=\sum_{n\in\mathcal{N}} y^{\dagger}_{s,,m}\cdot z_{m,n}=y^{\dagger}_{s,m}\end{aligned} \tag{21}$$

因此,可得式(22)和式(23):

$$\begin{aligned}E[\sum_{n\in\mathcal{N}} x_{s,n}\cdot c_n]&=\sum_{n\in\mathcal{N}} E[x_{s,n}]\cdot c_n\\&=\sum_{n\in\mathcal{N}}\mathrm{Pr}(x_{s,n}=1)\cdot c_n\\&=\sum_{n\in\mathcal{N}} x^{\dagger}_{s,n}\cdot c_n\leqslant R_s\end{aligned} \tag{22}$$

$$\begin{aligned}E[\sum_{m\in\mathcal{M}} y_{s,m}\cdot w_{s,m}]&=\sum_{m\in\mathcal{M}} E[y_{s,m}]\cdot w_{s,m}\\&=\sum_{m\in\mathcal{M}} Pr(y_{s,m}=1)\cdot w_{s,m}\\&=\sum_{m\in\mathcal{M}} y^{\dagger}_{s,m}\cdot w_{s,m}\leqslant C_s\end{aligned} \tag{23}$$

推论得证。

推论 2　算法 1 随机舍入过程求得解的期望满足请求任务至多被卸载到一个边缘服务器,即约束(15)。

证明:由式(21)可得:

$$E[\sum_{s\in\mathcal{S}} y_{s,m}]=\sum_{s\in\mathcal{S}} Pr(y_{s,m}=1)=\sum_{s\in\mathcal{S}} y^{\dagger}_{s,m}\leqslant 1 \tag{24}$$

推论得证。

定理 3　算法 1 能以 $1-\frac{1}{S}$ 的概率获得 $\frac{M\sqrt{S}}{N(\sqrt{S}-\sqrt{2\ln S})}$ 的近似比,其中 S 为边缘服务器数,N 为服务数,M 为终端设备数。

证明:设 μ 表示请求任务卸载变量 $y_{s,m}$ 求和的期望,则有:

$$\begin{aligned}\mu&=E[\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} y_{s,m}]\\&=\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} Pr(y_{s,m}=1)\\&=\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} y^{\dagger}_{s,m}\end{aligned} \tag{25}$$

根据式(21)可知 $y_{s,m}$ 是独立的,并且 $y_{s,m}\in[0,1]$,由切诺夫界[6],对于 $\forall\varepsilon>0$,满足:

$$P(\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} y_{s,m}\leqslant(1-\varepsilon)\mu)\leqslant e^{\frac{-\varepsilon^2\mu}{2}} \tag{26}$$

根据式(18)可得:

$$\begin{aligned}\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} y_{s,m}&=\sum_{n\in\mathcal{N}}\sum_{s\in\mathcal{S}}\sum_{m\in M} y_{s,m}\cdot z_{m,n}\\&\geqslant\sum_{n\in\mathcal{N}}|A_n|\cdot\lambda=M\cdot\lambda\end{aligned} \tag{27}$$

实际中,服务的请求数近似与总终端设备数成正比,设 $g_{\max}$ 表示 g_n 的最大值,则有 $g_{\max}\leqslant M\cdot\lambda/N$。因此可得:

$$M\cdot\lambda\leqslant\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{M}} y_{s,m}\leqslant M^2\cdot\lambda/N \tag{28}$$

令 λ^* 表示问题 P-LP 最优解对应的最小服务完成率,令 $\lambda^{\dagger}$ 表示问题 $\mathcal{P}$ 的最优解,则有 $\lambda^{\dagger}\leqslant\lambda^*$。与(28)同理。

$$M\cdot\lambda^*\leqslant\mu\leqslant M^2\cdot\lambda^*/N \tag{29}$$

联合式(26)、式(28)、式(29),可以得到不等式:

$$Pr\left(\frac{M^2\cdot\lambda}{N}\leqslant(1-\varepsilon)M\cdot\lambda^{\dagger}\right)\leqslant e^{\frac{-\varepsilon^2\mu}{2}} \tag{30}$$

令不等式右边为 $1/S$,即 $\lambda\leqslant\frac{(1-\varepsilon)N\cdot\lambda^{\dagger}}{M}$ 的最大概率为 $1/S$,则有 $\varepsilon=\sqrt{\frac{2\ln S}{\mu}}$。实际中,$\mu$ 近似地与边缘服务器的数目 S 成正比,因此有 $\varepsilon=\sqrt{\frac{2\ln S}{S}}$,得到 $1-\varepsilon=1-\sqrt{\frac{2\ln S}{S}}$。此时,$\frac{\lambda^{\dagger}}{\lambda}\geqslant\frac{M\sqrt{S}}{N(\sqrt{S}-\sqrt{2\ln S})}$,定理得证。

4.3　贪心算法

随机舍入算法可以作为最大化最小服务完成率问题的一种下界算法,具有理论意义。由于该算法的时间复杂度较高,受文献[21]的启发,本节给出了一种快速高效的贪心算法,用于求解最大最小服务完成率问题,算法的基本思想是贪心地缓存完成率最小的服务,并卸载服务对应的任务,直到最小服务完成率无法提升为止。算法的伪代码如算法 2 所示。算法 2 的具体步骤如下:

步骤 1　随机选择完成率最小的服务 n,计算请求服务 n 但未卸载的终端设备集合 $\mathcal{M}^n$。

步骤 2　遍历 $\mathcal{M}^n$,对于终端设备 m,按照到 m 的数据传输时延对边缘服务器排序得到集合 $\mathcal{S}_m$。遍历 $\mathcal{S}_m$,对边缘服务器 $s\in\mathcal{S}_m$,检查 s 是否缓存服务 n 以及是否有足够的计算资源执行 m 的任务。若存在 m 和 s 满足条件,则将 m 的任务卸载到 s,转步骤 1,否则转步骤 3。

步骤 3　遍历 $\mathcal{M}^n$,对于终端设备 m,再遍历 $\mathcal{S}_m$。检查边缘服务器 $s\in\mathcal{S}_m$ 是否有足够的存储资源和计算资源缓存服务 n 并执行 m 的任务。若存在 s 满足条件,则将服务 n 缓存到 s 中,并将 m 的任务卸载到 s,转步骤 1;否则,算法终止。

算法 2　贪心算法(Greedy Algorithm,GA)

输入:边缘服务器集合 $\mathcal{S}$,终端设备集合 $\mathcal{M}$,服务集合 $\mathcal{N}$,终端设备请求 $z_{m,n}$

输出:服务缓存策略 $x_{s,n}$,任务卸载策略 $y_{s,m}$

1. flag:= 1,计算 $\mathcal{S}_m$,$\forall m\in\mathcal{M}$

2. while flag do

3.　flag:= 0,更新 g_n,随机选择完成率最小的服务 n,计算 $\mathcal{M}^n$;

4.　for $m\in\mathcal{M}^n$,$s\in\mathcal{S}_m$ do

5.　　满足式(4)和式(16)的情况下

　　$y_{s,m}$:= 1,flag:= 1,break;

6.　end for

7.　if flag=0 then

8. for $m\in\mathcal{M}$, $s\in\mathcal{S}_m$ do
9. 满足式(1)、式(4)和式(16)的情况下
$x_{s,n}:=1, y_{s,m}:=1, flag:=1,$
break;
10. end for
11. end if
12. end while
13. return $x_{s,n}, y_{s,m}$

定理 4 算法 2 的时间复杂度为 $O((S+N)M^2)$，其中 S 为边缘服务器数，N 为服务数，M 为终端设备数。

证明：循环内部首先挑选完成率最小的服务以及未被卸载的终端设备集合，此步骤可遍历服务和终端设备完成，时间复杂度为 $O(MN)$。再寻找最佳的边缘服务器缓存服务并卸载任务，此步骤可遍历终端设备和边缘服务器完成，时间复杂度为 $O(SM)$。循环次数不会超过终端设备数 M，因此算法 2 的时间复杂度为 $O((S+N)M^2)$。

5 实验结果与分析

5.1 实验配置

实验使用 Python 语言，在搭载 Windows 11 系统，内存为 16GB，处理器为 AMD Ryzen 7 6800H 3.20 GHz 的计算机上执行。如无特别说明，边缘服务器的数目设置为 9，终端设备的数目设置为 60。边缘服务器为图 2 阴影部分所示的拓扑结构[22]，相邻边缘服务器的数据传输速率均为 100 Mbps，延迟均为 25ms。边缘服务器的存储空间为[800，1 200] MB，计算能力为[50，100]GFLOPS。终端设备与边缘服务器的距离为[25，100]m。W 为 5 MHz，终端设备的传输功率为 25 dBm，边缘服务器与终端设备之间的无线信道增益为 174 dBm/Hz。

本文选择 6 种不同的 DNN 模型作为终端设备请求的服务，例如 AlexNet，VGG16 等。服务的数据输入为 224×224×3 的 RGB 图片，数据量约为 147 kB。服务所需的存储空间和计算量为对应 DNN 模型的模型大小和计算量。终端设备对 6 种服务的请求是随机产生的，如无特别说明，用户对服务的最大容忍时延要求设置为 0.6 s。

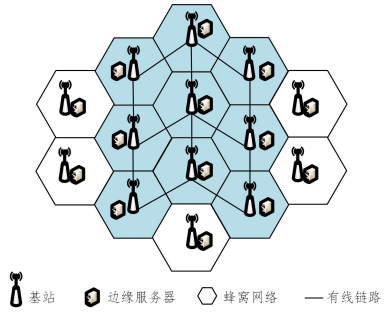


图 2 蜂窝网络中的边缘服务器拓扑图

Fig. 2 Topology of edge servers in cellular network

5.2 基准算法

实验选择如下 3 种对比算法，实验结果均是在相同实验设置下重复 100 次取平均值得到的。

随机算法(Random Algorithm，RA)：该算法每次随机选择一个终端设备 m 和一个边缘服务器 s，若 s 缓存了 m 请求的服务 n 且 s 有足够的剩余计算资源处理 m 的任务，则将 m 的任务卸载到 s；若 s 未缓存 n，但 s 有足够的剩余存储资源和计算资源缓存 n 并处理 m 的任务，则将 n 缓存到 s，并将 m 的任务卸载到 s；否则置 m 的任务为失败。重复以上过程，直到所有终端设备全部被选择完毕时算法终止。算法的时间复杂度为 $O(M)$。

SPR+Modified(SPRM)算法：文献[6]联合服务缓存和请求调度以最大化系统吞吐量，由于文献[6]未考虑边缘协同计算等因素，其 SPR 算法不适用于本文的网络模型。对 SPR 算法进行适当调整得到对比算法 SPRM。SPRM 算法允许边缘协同计算，首先求解吞吐量最大化的线性规划得到最优分数解 $x^*_{s,n}$ 和 $y^*_{s,m}$，随后以 $x^*_{s,n}$ 的概率将服务 n 缓存到边缘服务器 s，以 $y^*_{s,m}/x^*_{s,n}$ 的概率将终端设备 m 的任务卸载到边缘服务器 s，再分别按照 $x^*_{s,n}$ 和 $y^*_{s,m}/x^*_{s,n}$ 的升序从边缘服务器中移除相应服务和任务，直到所有约束满足。算法的时间复杂度为 $O((SM+SN)^3)$。

SPRM+Greedy(SPRMG)算法：目标是最大化系统吞吐量，算法在执行 SPRM 算法后，循环地从未卸载的任务中选择计算资源消耗最少的任务卸载到相应边缘服务器，直到所有边缘服务器的剩余资源无法满足其他任意任务。算法的时间复杂度为 $O((SM+SN)^3)$。

5.3 结果分析

图 3 给出了不同边缘服务器数下各算法的性能对比。由图 3(a)可知，GA 算法、ROAG 算法的最小服务完成率都大于 SPRM 算法、SPRMG 算法以及 RA 算法。特别地，当边缘服务器数为 8 时，相比 SPRMG 算法，ROAG 算法和 GA 算法分别可使最小服务完成率提升 74.7%和 129.0%。相比 SPRM 算法，ROAG 算法和 GA 算法分别可使最小服务完成率提升至少 28.4%和 63.5%。结合图 3(b)，相比 SPRM 算法和 SPRMG 算法，ROAG 算法和 GA 算法在大幅提升最小服务完成率的同时，也能使系统吞吐量损失不超过 27.8%和 32.2%。由于用户是自私且个体理性的，因此牺牲部分可接受的系统吞吐量，以保证用户异构计算任务的服务公平性是十分有必要的。

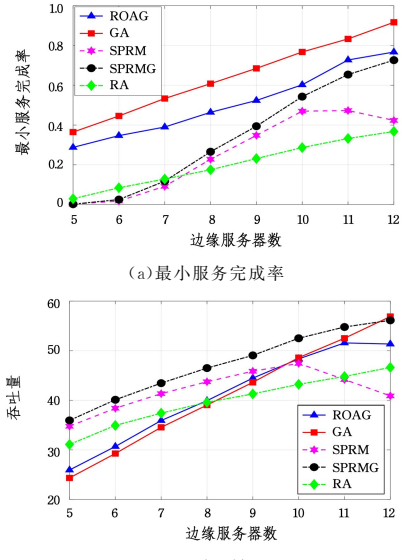


图 3 不同边缘服务器数下各算法的性能对比

Fig. 3 Performance comparisons of algorithms for different edge servers

随着边缘服务器增多，各算法的最小服务完成率和吞吐量整体逐渐增大，这是因为边缘服务器越多，系统资源更充足，可服务更多终端设备。当边缘服务器数超过 10 时，SPRM 算法的最小服务完成率和吞吐量略微下降，观察

SPRM 算法中间结果发现,当边缘服务器偏多时,系统资源较充足,任务卸载到各边缘服务器的概率十分接近,此时 SPRM 算法会将多个同一任务同时卸载到多个服务器重复执行。

表 2 列出了不同边缘服务器数下 SPRM 算法将同一任务同时卸载到多个边缘服务器的任务数和比例。从表中可知,边缘服务器越多,被重复卸载的任务占比越大。例如边缘服务器数为 12 时,被重复卸载的任务占比达到了 25.5%。边缘服务器数目偏多时,被重复卸载的任务占比偏大,多个同一任务的重复执行造成系统计算资源浪费较严重,反而使指标下降。

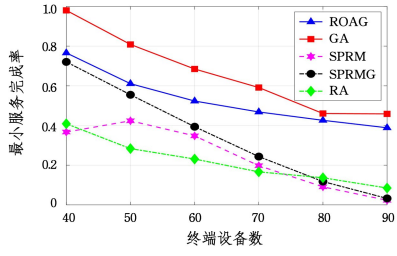
表 2　SPRM 算法重复卸载的任务数据表

Table 2　Tasks offloaded by SPRM repeatedly

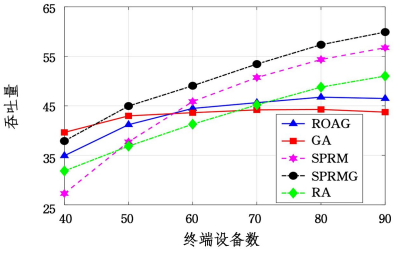
(单位:%)

S	8	9	10	11	12
任务数	0.9	1.1	3.2	9.1	15.3
任务占比	1.5	1.8	5.3	15.2	25.5

图 4 给出了不同终端设备数下各算法的性能对比。从图 4(a)可知,GA 算法、ROAG 算法的最小服务完成率都大于 SPRM 算法、SPRMG 算法以及 RA 算法。特别地,当终端设备数为 70 时,相比 SPRMG 算法,ROAG 算法和 GA 算法分别可使最小服务完成率提升 91.8%和 142.2%。相比 SPRM 算法,ROAG 算法和 GA 算法分别可使最小服务完成率提升至少 44.1%和 90.6%。结合图 4(b),相比 SPRM 算法和 SPRMG 算法,ROAG 算法和 GA 算法在大幅提升最小服务完成率的同时,也能使系统吞吐量损失不超过 22.4%和 27.0%。随着终端设备增多,各算法的最小服务完成率整体逐渐减小,这是因为终端设备增多,系统资源固定,服务的完成率降低。当终端设备数为 40 时,SPRM 算法的最小服务完成率有略微下降,与前面的分析同理,此时终端设备偏少,系统资源较为充足,SPRM 算法中任务卸载到各边缘服务器的概率十分接近。SPRM 算法的任务卸载策略会造成系统计算资源浪费,反而使指标下降。此外,随着终端设备数增多,请求小服务的任务增多,系统吞吐量逐渐增大。



(a)最小服务完成率

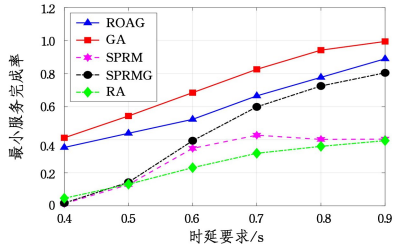


(b)吞吐量

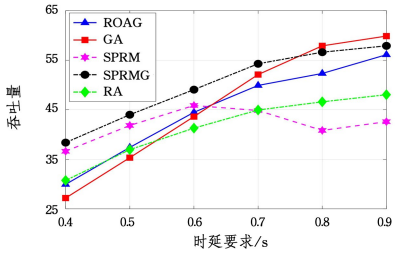
图 4　不同终端设备数下各算法的性能对比

Fig. 4　Performance comparisons of algorithms for different mobile devices

图 5 给出了不同时延要求下各算法的性能对比。从图 5(a)中可知,GA 算法、ROAG 算法的最小服务完成率都大于 SPRM 算法、SPRMG 算法以及 RA 算法。特别地,当时延要求为 0.6s 时,相比 SPRMG 算法,ROAG 算法和 GA 算法分别可使最小服务完成率提升 33.1%和 73.4%。相比 SPRM 算法,ROAG 算法和 GA 算法分别可使最小服务完成率提升至少 50.7%和 97.4%。结合图 5(b),相比 SPRM 算法和 SPRMG 算法,ROAG 算法和 GA 算法在大幅提升最小服务完成率的同时,也能使系统吞吐量损失不超过 21.9%和 29.1%。随着时延要求增大,各算法的最小服务完成率和系统吞吐量整体逐渐增大,这是因为时延要求增大,任务消耗的计算资源减少,系统可服务更多终端设备。当时延要求超过 0.7s 时,SPRM 算法的最小服务完成率和系统吞吐量有略微下降,与前面的分析同理,此时时延要求偏大,系统资源较为充足,SPRM 算法中任务卸载到各边缘服务器的概率十分接近。SPRM 算法的任务卸载策略会造成系统计算资源浪费,反而使指标下降。



(a)最小服务完成率



(b)吞吐量

图 5　不同时延要求下各算法的性能对比

Fig. 5　Performance comparison of algorithms for different latency requirements

表 3 列出了不同终端设备数下各算法的运行时间对比。

表 3　各算法的运行时间对比

Table 3　Running time comparison of different algorithms

(单位:ms)

M	RA	SPRM	SPRMG	ROAG	GA
40	0.43	178.03	178.72	201.64	3.89
50	0.42	522.63	523.43	561.10	5.98
60	0.47	1049.20	1050.01	1099.20	7.05
70	0.49	1415.13	1416.05	1491.79	7.35
80	0.52	2827.76	2829.35	2747.14	12.60
90	0.72	3346.10	3347.85	3177.67	11.71

从表中可知,各算法的运行时间随着终端设备数目的增多而逐渐增大。由于 $O(M)<O((S+N)M^2)<O((SM+SN)^3)$,RA 算法的时间复杂度最小,其运行时间也是所有算法中最短的。ROAG 算法和 SPRM 算法以及 SPRMG 算法均需求解线性规划,它们的时间复杂度相同,实际的运行时间也十分接近。GA 算法的运行时间长于 RA 算法的运行时间,短于

ROAG 算法、SPRM 算法以及 SPRMG 算法的运行时间，这与 GA 算法的时间复杂度大于 RA 算法，远小于 ROAG 算法、SPRM 算法以及 SPRMG 算法是相符的。

结束语 为保证用户请求异构服务的公平性，提出了一个新颖的 NP 难问题，在边缘服务器的计算存储以及任务响应时延约束下，联合服务缓存与任务卸载，最大化最小服务完成率。通过引入连续变量和线性松弛操作，将所提问题转化为线性规划问题，提出了一种近似比为 $M\sqrt{S}/N(\sqrt{S}-\sqrt{2\ln S})$ 的随机舍入算法 ROAG 与一种快速高效的贪心算法 GA。实验结果表明，相比最大化吞吐量算法，提出的 ROAG 算法和 GA 算法不仅能显著提升最小服务完成率，还能使系统吞吐量的额外损失维持在较低水平。未来将考虑设计基于深度强化学习的在线算法，实时调整服务缓存和任务卸载策略，以应对用户请求动态变化的问题。

参 考 文 献

[1] CHEN M, HAO Y. Task offloading for mobile edge computing in software defined ultra-dense network[J]. IEEE Journal on Selected Areas in Communications, 2018, 36(3): 587-597.

[2] SUN H T, FAN Y F, MA M X, et al. Dynamic Pricing-based Vehicle Collaborative Computation Offloading Scheme in VEC [J]. Computer Science, 2022, 49(9): 242-248.

[3] TRAN T X, HAJISAMI A, PANDEY P, et al. Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges[J]. IEEE Communications Magazine, 2017, 55(4): 54-61.

[4] TRAN T X, CHAN K, POMPILI D. Costa: cost-aware service caching and task offloading assignment in mobile edge compting [C]// IEEE International Conference on Sensing, Communication, and Networking. Boston: IEEE, 2019: 1-9.

[5] ZHANG J, LETAIEF K B. Mobile edge intelligence and computing for the internet of vehicles[J]. Proceedings of the IEEE, 2019, 108(2): 246-261.

[6] POULARAKIS K, LLORCA J, TULINO A M, et al. Service placement and request routing in MEC networks with storage, computation, and communication constraints [J]. IEEE/ACM Transactions on Networking, 2020, 28(3): 1047-1060.

[7] ZHONG S, GUO S, YU H, et al. Cooperative service caching and computation offloading in multi-access edge computing [J]. Computer Networks, 2021, 189: 107916.

[8] XU J, CHEN L, ZHOU P. Joint service caching and task offloading for mobile edge computing in dense networks[C]// IEEE International Conference on Computer Communications. Honolulu: IEEE, 2018: 207-215.

[9] SAPIJASZKO G, MIKHAEL W B. An overview of recent convolutional neural network algorithms for image recognition [C]// IEEE International Midwest Symposium on Circuits and Systems. Windsor: IEEE, 2018: 743-746.

[10] CAMPAGNER A, CIUCCI D, CABITZA F. Aggregation models in ensemble learning: a large-scale comparison[J]. Information Fusion, 2023, 90: 241-252.

[11] YAO M, CHEN L, WU Y, et al. Loading cost-aware model caching and request routing in edge-enabled wireless sensor networks[J]. The Computer Journal, 2022.

[12] FARHADI V, MEHMETI F, HE T, et al. Service placement and request scheduling for data-intensive applications in edge clouds [J]. IEEE/ACM Transactions on Networking, 2021, 29(2): 779-792.

[13] ZHAO G, XU H, ZHAO Y, et al. Offloading tasks with dependency and service caching in mobile edge computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(11): 2777-2792.

[14] BI S, HUANG L, ZHANG Y J A. Joint optimization of service caching placement and computation offloading in mobile edge computing systems[J]. IEEE Transactions on Wireless Communications, 2020, 19(7): 4947-4963.

[15] PREMSANKAR G, GHADDAR B. Energy-efficient service placement for latency-sensitive applications in edge computing [J]. IEEE Internet of Things Journal, 2022, 9(18): 17926-17937.

[16] SHEN Q, HU B J, XIA E. Dependency-aware task offloading and service caching in vehicular edge computing [J]. IEEE Transactions on Vehicular Technology, 2022, 71(12): 13182-13197.

[17] ZHOU J, ZHANG X. Fairness-aware task offloading and resource allocation in cooperative mobile-edge computing [J]. IEEE Internet of Things Journal, 2021, 9(5): 3812-3824.

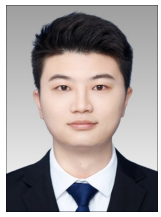
[18] BAKTIR A C, AHAT B, ARAS N, et al. Sla-aware optimal resource allocation for service-oriented networks[J]. Future Generation Computer Systems, 2019, 101: 959-974.

[19] CHEN M, WANG H, MENG Z, et al. Joint data collection and resource allocation for distributed machine learning at the edge [J]. IEEE Transactions on Mobile Computing, 2020, 21(8): 2876-2894.

[20] YANG S, LI F, TRAJANOVSKI S, et al. Delay-aware virtual network function placement and routing in edge clouds[J]. IEEE Transactions on Mobile Computing, 2019, 20(2): 445-459.

[21] WU Y, WU J, CHEN L, et al. Load balance guaranteed vehicle-to-vehicle computation offloading for min-max fairness in VANETs[J]. IEEE Transactions on Intelligent Transportation Systems, 2021.

[22] JEONG H J, LEE H J, SHIN K Y, et al. Perdnn: offloading deep neural network computations to pervasive edge servers[C]// IEEE International Conference on Distributed Computing Systems. Singapore: IEEE, 2020: 1055-1066.



WU Chun, born in 1998, postgraduate. His main research interest is edge computing.

WU Jigang, born in 1963, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include intelligent computing, and mobile computing.