计算机科学
COMPUTER SCIENCE

面向缓存的动态协作任务迁移技术研究

赵晓焱，赵斌，张俊娜，袁培燕

引用本文

赵晓焱，赵斌，张俊娜，袁培燕. 面向缓存的动态协作任务迁移技术研究[J]. 计算机科学，2024，51(2)：300-310.

ZHAO Xiaoyan, ZHAO Bin, ZHANG Junna, YUAN Peiyan. Study on Cache-oriented Dynamic Collaborative Task Migration Technology [J]. Computer Science, 2024, 51(2): 300-310.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230600128

面向缓存的动态协作任务迁移技术研究

赵晓焱[1,2]　赵　斌[1]　张俊娜[1,2]　袁培燕[1]
1 河南师范大学计算机信息与工程学院　河南 新乡 453007
2 智慧商务与物联网技术河南省工程实验室　河南 新乡 453007
(zhaoxiaoyan@htu.edu.cn)

摘　要　边缘网络中不断出现的计算密集和延迟敏感型业务推动了任务迁移技术的快速发展。然而,任务迁移过程中存在应用场景复杂多变、问题建模难度高等技术瓶颈。尤其是考虑用户移动时,如何保证用户服务的稳定性和连续性,设计合理的任务迁移策略仍是一个值得深入探讨的问题。因此,提出了一种移动感知的服务预缓存模型和任务预迁移策略,将任务迁移问题转化为最优分簇与边缘服务预缓存的组合优化问题。首先,基于用户的移动轨迹对当前执行任务状态进行预测,引入动态协作簇和迁移预测半径的概念,提出了一种面向移动和负载两种任务场景的预迁移模型,解决了何时何地迁移的问题。然后,针对需要迁移的任务,基于最大容忍时延约束分析协作簇半径和簇内目标服务器数量的极限值,提出了以用户为中心的分布式多服务器间动态协作分簇算法(Distributed Dynamic Multi-server Cooperative Clustering Algorithm,DDMC)以及面向服务缓存的深度强化学习算法(Cache Based Double Deep Q Network,C-DDQN),解决了最优分簇和服务缓存问题。最后,利用服务缓存的因果关系,设计了一种低复杂度的交替最小化服务缓存位置更新算法,求解出了最佳迁移目标服务器集合,实现了任务迁移中的服务器协作及网络负载均衡。实验结果表明,提出的迁移选择算法具有良好的鲁棒性和系统性能,相比其他迁移算法所消耗的总成本降低了至少12.06%,所消耗的总时延降低了至少31.92%。

关键词:移动边缘计算;服务缓存;动态协作簇;任务迁移;深度强化学习
中图分类号　TP393

Study on Cache-oriented Dynamic Collaborative Task Migration Technology

ZHAO Xiaoyan[1,2],ZHAO Bin[1],ZHANG Junna[1,2] and YUAN Peiyan[1]
1 College of Computer and Information Engineering,Henan Normal University,Xinxiang,Henan 453007,China
2 Engineering Lab of Intelligence Business&Internet of Things,Xinxiang,Henan 453007,China

Abstract　Task migration technology has been propelled by the continuous emergence of compute-intensive and delay-sensitive services in edge networks. However,the process of task migration is hindered by technical bottlenecks such as complex and time-varying application scenarios,as well as the high difficulty in problem modeling. Especially when considering user movement,designing a reasonable task migration strategy that ensures the stability and the continuity of user service remains a persistent challenge. Therefore,a mobile-aware service pre-caching model and task pre-migration strategy are proposed to transform the problem of task migration into an optimization problem that combines optimal clustering strategies with edge service pre-caching. First of all,the current state of the task is initially predicted based on the user's movement trajectory. To solve the problem of when and where to migrate,a pre-migration model for two task scenarios,namely mobile and load,is proposed by introducing the concept of dynamic cooperation cluster and migration prediction radius. And then,according to the tasks that need to be migrated,the maximum tolerant delay constraint is utilized to derive the limit value of cooperative cluster radius and target server quantity in a cluster. Subsequently,a user-centric distributed dynamic multi-server cooperative clustering algorithm(DDMC) and a cache-based double deep Q network algorithm(C-DDQN) for service are proposed to solve the problem of optimal clustering and service caching. Finally,a low-complexity alternate minimization service cache location update algorithm is designed using the causality of service caches to achieve the optimal set of migration target servers,which realize server collaboration and network load balancing in task migration. Experimental results demonstrate the robustness and the system performance of the proposed migration selection algorithm. Compared with other algorithms,the total cost consumed is reduced by at least 12.06%,the total latency con-

到稿日期:2023-06-15　返修日期:2023-09-26
基金项目:国家自然科学基金(62072159,61902112);河南省科技攻关项目(222102210011,232102211061)
This work was supported by the National Natural Science Foundation of China (62072159,61902112) and Science and Technology Research Project of Henan Province(222102210011,232102211061).
通信作者:张俊娜(jnzhang@htu.edu.cn)

sumed is reduced by at least 31.92%.

Keywords Mobile edge computing, Service cache, Dynamic collaborative cluster, Task migration, Deep reinforcement learning

1 引言

随着移动网络技术及智能终端设备的不断发展,各种延迟敏感的计算密集型业务不断涌现,尤其是未来云 XR、数字孪生、全息通信等潜在的沉浸式交互场景,使得各种计算任务和数据呈井喷式增长。然而,有限的终端资源和传统的云范式无法满足用户在传输时延、成本效益、可靠性等方面提出的应用需求。边缘计算作为一种分布式计算范式[1],通过对复杂计算任务进行迁移和卸载,可以加速数据处理速度,缩短服务响应时间,提高应用程序的执行效率和服务质量,解决了云计算交付时延高、传输能耗大、占用资源多的问题。

任务迁移的应用主要有两种场景,一是考虑用户移动场景时,用户的位置变化会增加终端与边缘服务器之间的连接中断概率,服务在边缘服务器之间的频繁切换也会形成乒乓效应,从而影响任务的执行效率,因此需要通过边缘服务器之间的任务迁移与协同调度,来提升用户使用体验。另一种情况则是考虑到边缘服务器的计算资源、覆盖范围和缓存空间受限,当边缘服务器由于数据量过大而负载溢出时,需要设计有效的任务迁移算法,通过边缘服务器之间的横向协作,来确保应用程序的执行效率和可靠性。

任务迁移根据迁移的类型可以分为服务迁移和数据迁移。以人脸识别应用为例,服务迁移指迁移人脸识别应用程序,数据迁移指迁移用户的脸部特征数据。现有工作中经常单一考虑数据迁移[2-4]或服务迁移[5-10]。实际上,应用程序的计算结果与输入数据具有强关联性,两者很难单独执行,例如人脸识别和交互式在线游戏。因此,在任务迁移时需要同时考虑服务迁移和数据迁移。

文献[2]根据移动用户计算数据的数据量和边缘服务器的性能特点,提出了一种基于任务预测的数据迁移算法。文献[3]研究了协同数据迁移和服务缓存模型,提出了一种有效的 Lyapunov 在线算法,用于联合执行数据迁移和动态数据缓存策略。文献[4]介绍了一种针对移动边缘计算中的数据缓存和迁移问题的智能联合策略。该策略采用了基于 Lyapunov 漂移加惩罚函数的代价自适应算法,能够在逐时隙的基础上对长期问题进行优化。

文献[5]主要研究了在移动边缘计算中基于深度强化学习的服务缓存和服务迁移策略,以实现对能耗和延迟的权衡。文献[6]尝试在考虑用户移动性的情况下优化服务迁移,设计了具有已知用户轨迹的组迁移算法。文献[7]考虑了基于 SDN 的移动边缘计算系统环境,利用 SDN 解决边缘服务器负载不均衡的情况。文献[8]研究了长期服务迁移能耗预算约束下的时延优化问题,利用 Lyapunov 优化将长期优化问题分解为一系列实时优化问题。

另一方面,现有工作在考虑任务迁移时,通常采用实时迁移方式,即仅当用户移出当前覆盖范围时才进行任务迁移。因此,边缘服务器在获取和初始化服务程序时会产生较长的延迟,大大增加了通信时延和迁移成本。事实上,目前任务迁移策略忽略了边缘服务器的预缓存功能[11]。在处理任务时,为了避免重复计算,节省通信资源,可以提前将一些常用的服务[3,12-15]或数据[1,5,7,16-18]缓存在边缘服务器中。此时,当用户请求某个应用程序或数据时,边缘服务器可以直接从缓存中获取,从而提高服务的响应速度和质量。为了区别于传统内容分发网络中的内容缓存,本文依据缓存内容称其为服务缓存和数据缓存。

文献[14]提出了一种易于处理的在线算法,该算法不考虑移动应用的到达模式,可以动态地缓存服务。文献[15]研究了服务提供商在有限预算下的边缘服务部署问题,提出了一个空间-时间边缘服务放置与缓存算法。文献[17]为了合理地缓存数据,提高缓存命中率,减少用户请求延迟,根据文件大小、文件受欢迎程度和请求时间提出了文件缓存值的概念。文献[18]将缓存问题表述为 0-1 多重背包问题,重点研究了数据缓存问题。然而,由于每个边缘服务器资源有限且计算资源、缓存大小各不相同,每个服务的大小也各不相同,如何有效地分配计算资源和缓存资源也成为任务迁移过程中需要考虑的一个重要问题,而现有服务缓存和数据缓存多考虑静态服务部署场景[3,12-15],忽略了用户移动性对缓存的影响。

综上所述,考虑资源的有限性、边缘设备的异质性以及用户的移动性,实现各种数据和服务的预缓存与迁移,即在提前部署缓存服务的基础上,决定迁移哪些任务和数据、何时迁移、迁移到哪里,这是任务迁移过程中面临的一个难点问题。为了合理分配缓存和计算资源,提高整个移动边缘系统的计算效率和可扩展性,本文在时延约束下,以迁移成本为优化目标,联合预缓存和动态协作簇技术,制定了最优任务迁移策略。本文的主要贡献如下:

1)为了保证用户的无缝服务迁移以及服务连续性,本文面向用户移动感知场景,提出了基于预缓存的任务迁移模型,通过引入迁移预测半径概念,分析不同待迁移用户类型,构造了服务迁移和数据迁移成本函数,并将基于时延约束的迁移成本最小化问题转化为对分簇变量和服务预缓存变量的优化决策。

2)利用用户最大容忍时延约束,将决策变量的组合优化问题分解为最优分簇策略和最优服务预缓存策略两个子问题。同时,利用任务低耦合性将其拆分为多个独立子任务模型,降低了问题中优化变量的维度;提出了一种分布式多服务器间的动态协作分簇算法(DDMC),解决了最优协作分簇问题。

3)利用服务器的缓存功能和服务的重用性,提出了一种面向服务缓存的深度强化学习算法(C-DDQN),解决了最优服务预缓存问题。基于用户协作簇规模和服务缓存变量,设计了目标节点选择算法,得到了最优迁移服务器集合,实现了多服务器协作及负载均衡。最后,采用合作项目背景下的真实用户脱敏数据集,验证了所提算法在时延和迁移成本等方面的优越性。

2 问题表述

2.1 系统模型

为了描述任务迁移场景，本文假定网络中有 NU 个用户、MC 个基站（小区）和 ED 个边缘服务器（ES），分别表示为 $N=\{1,2,\cdots,n,\cdots,NU\}$，$M=\{1,2,\cdots,m,\cdots,MC\}$ 和 $E=\{1,2,\cdots,e,\cdots,ED\}$。需要注意的是，考虑成本限制，并不是每个基站都会部署 ES，实际数量和放置位置可以根据场景需求进行配置，即 $ED<MC$。

为了实现高效的任务迁移，本文迁移过程主要包括两个阶段：面向用户的迁移预测和面向边缘服务器的迁移决策。用户移动轨迹是迁移预测的主要依据，本文将用户 n 在 t 时刻所在小区号表示为 $L_n(t)$，则一天内在不同时刻经过的小区向量可以描述为：

$$\boldsymbol{L}_n=\{L_n(t),L_n(t)\in M,t\in\{1,2,\cdots,T\}\}$$

其中，T 为一天内时间切片总数，并将用户在无限空间中的移动轨迹建模为二维随机游走模型[11]。考虑用户移动性及边缘服务器的负载均衡，本文允许服务器间的分布式协作，并提出了动态协作簇的概念。定义以用户 n 为中心的协作簇为 $\Re_n(t)$，$n\in N$，考虑到边缘服务器缓存、计算等资源有限，簇中服务器并不一定都缓存了用户所需的服务，因此进一步定义用户 n 的目标服务器集合 $S_n(t)$，$S_n(t)\subset\Re_n(t)$，即在协作簇中被选择用来迁移数据和任务的服务器集合。若协作簇中所有服务器均未缓存所需服务，则考虑从云端进行下载。

图 1 给出了基于缓存的任务迁移和动态协作簇在边缘计算中的实现过程。为了分析用户在何时、何处进行迁移，本文结合实际情况，考虑对基站所覆盖范围进行六边形单元格划分，$B^m_{\max}$ 表示基站 m 的最大总层数，用户在基站 m 的位置用二维坐标 (B^m_x,B^m_y) 表示，其中 B^m_x 为层数，B^m_y 为服务器在该层的编号数，有 $B^m_x\leq B^m_{\max}$。如图 1 所示，小区 A 和小区 B 的 $B^m_{\max}$ 均设置为 3，若用户 n 要从小区 A 中(2,11)位置移动到小区 B 中(3,5)位置，为了提升迁移效率，系统将以用户 n 为中心在目标区域进行动态分簇，此时服务器 B,C,D 形成了动态协作簇。如果用户 n 所需最小服务器数为 2，考虑任务的低耦合性，将任务分成两个独立的子任务，然而仅有服务器 B 缓存了用户服务，则需要在服务器 C 和 D 中再选择一个，与 B 形成目标服务器集合 $S_n(t)$。由于服务器 C 距离服务器 B 较近，因此服务器 B 和 C 形成目标服务器集合 $S_n(t)$。若服务器 C 中并未缓存用户服务，需要将服务从服务器 B 迁移到服务器 C，进行簇内服务迁移，再将两个独立子任务数据迁移到目标服务器集合 $S_n(t)$，进行多服务器协作，最终实现网络负载均衡。为了更方便地描述用户迁移预测阶段，本文给出了迁移预测半径和两种不同类型的迁移用户的定义。

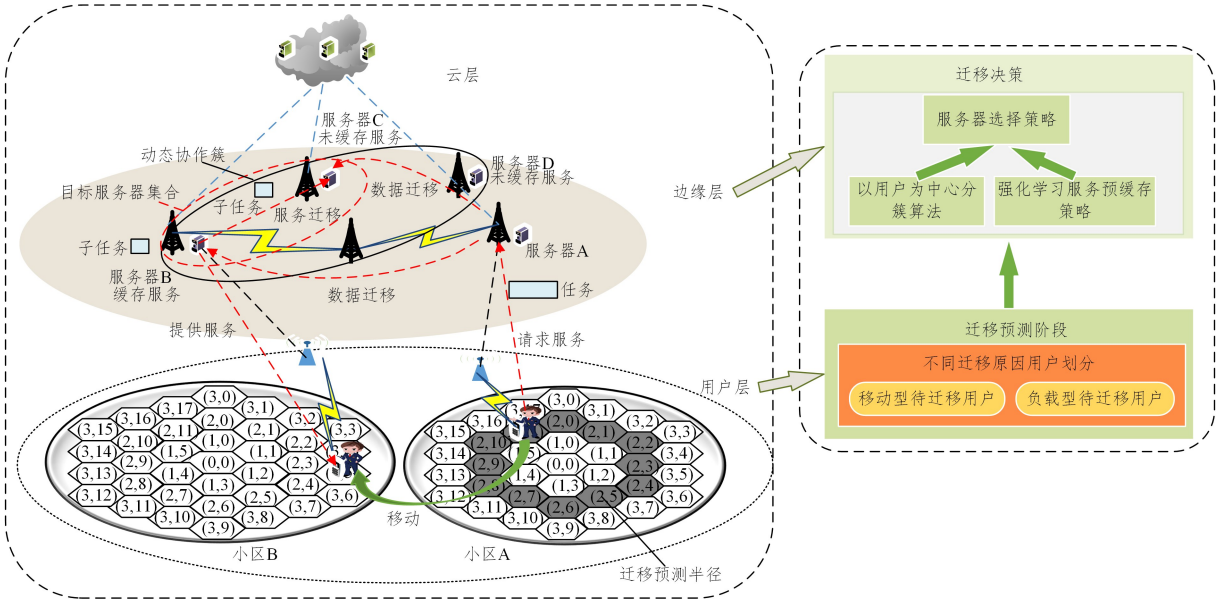

图 1 面向缓存的任务迁移和动态协作簇在边缘计算中的流程图

Fig. 1 Flowchart of cache-oriented task migration and dynamic collaborative clusters in edge computing

定义 1（迁移预测半径） 为判断 t 时刻用户 n 产生的任务，本文针对基站 m 引入距离 $r_m(t)$，并将以基站 m 为中心，$r_m(t)$ 为半径形成的圆环定义为用户是否迁移预判断位置，此时 $r_m(t)$ 称为迁移预测半径。

图 1 描述的场景中，用户在小区 A 的迁移预测半径选定为 2（即 B^m_x 为 2 的所有位置）。考虑小区内用户密度、请求数量等因素影响，每个基站 m 的迁移预测半径 $r_m(t)$ 可能会不同，其与覆盖半径 R_m 之间的关系为 $r_m(t)=R_m/(\alpha\rho_m)$，$\alpha$ 为参数变量，ρ_m 为第 m 个基站的用户密度。

定义 2（移动型待迁移用户） 移动感知情况下，当预测到用户 n 即将离开基站 m 时，用户 n 到达迁移预测半径 $r_m(t)$，对于没有执行完成的任务，若剩余任务执行完成时间 t_n^{end1} 大于预测离开小区的时间 t_n^l，即 $t_n^{\text{end1}}>t_n^l$ 时，则判断用户 n 为待迁用户并称为移动型待迁移用户。

定义 3（负载型待迁移用户） 移动感知情况下，当预测到用户 n 即将离开基站 m，且其位置在 $B^m_x\in[r_m(t),B^m_{\max}]$ 范围，若产生了新任务且新任务完成所需时间 t_n^{end2} 大于预测离开小区的时间 t_n^l，即 $t_n^{\text{end2}}>t_n^l$ 或者当前服务的服务器负载过大时，则判断用户 n 为待迁用户并称为负载型待迁移用户。

移动型待迁移用户中原服务器一定缓存了用户的服务，

然而负载型待迁移用户的原服务器并不一定缓存了用户的服务。因此,当用户的协作簇中未缓存用户服务时,对于移动型待迁移用户,仅需从原服务器将服务迁移到目标服务器集合中,而对于负载型待迁移用户,需要考虑从云端下载服务到目标服务器集合中,下文将进行详细分析。

2.2 问题建模

为了简化后面的分析,本文借鉴了文献[9]中迁移成本的表达,以常数加指数形式定义服务迁移成本函数 $S_{\text{cost}}(i,j,t)$ 和数据迁移成本函数 $D_{\text{cost}}(L_n(t),j)$。$S_{\text{cost}}(i,j,t)$ 表示时刻 t 用户 n 所需的服务,从服务器 i 迁移到服务器 j 时,所耗费的服务迁移成本。$D_{\text{cost}}(L_n(t),j)$ 表示时刻 t 用户 n 的数据,从当前小区 $L_n(t)$ 迁移到服务器 j 时,所耗费的数据迁移成本具体表述如下:

$$S_{\text{cost}}(i,j,t)=\begin{cases}0, & \text{if } H^s_{i,j}=0\\ \beta_c+\beta_l\mu^{I_{n,j}(t)H^s_{i,j}}, & \text{if } H^s_{i,j}>0\end{cases} \tag{1}$$

$$D_{\text{cost}}(L_n(t),j)=\begin{cases}0, & \text{if } H^d_{L_n(t),j}=0\\ \delta_c+\delta_l\theta^{H^d_{L_n(t),j}}, & \text{if } H^d_{L_n(t),j}>0\end{cases} \tag{2}$$

其中,$S_{\text{cost}}(i,j,t)$ 和 $D_{\text{cost}}(L_n(t),j)$ 均为单调非递减函数;β_c,β_l,δ_c,δ_l,μ 和 θ 是实数参数,取值与网络环境有关[11]。$I_{n,j}(t)$ 表示 t 时刻服务器 j 是否缓存用户 n 的服务,其中 $I_{n,j}(t)$ 为二进制变量,为 0 表示没有缓存,为 1 则表示缓存了所需服务。$H^s_{i,j}$ 表示服务迁移时服务器 i 和服务器 j 之间的跳数,$H^d_{L_n(t),j}$ 表示数据迁移时用户所在小区 $L_n(t)$ 到服务器 j 之间的跳数。

定义 $q_{\Re_n(t)}$ 表示为:

$$q_{\Re_n(t)}=\sum_{j\in\Re_n(t)}(I_{n,j}(t)\vee 0) \tag{3}$$

则 $q_{\Re_n(t)}=\{0,1\}$ 表示 t 时刻用户 n 所需的服务是否缓存在用户 n 的协作簇 $\Re_n(t)$ 中,为 0 表示否,为 1 则表示是。对于移动型待迁移用户,因为任务已经在原服务器 i 上执行,即使协作簇中没有缓存服务,也可以直接从原服务器 i 中通过边-边迁移形成目标服务器集合。但对于负载型待迁移用户,若任务为新生成任务,则可能在协作簇中没有缓存所需服务,即 $q_{\Re_n(t)}=0$,此时需要从云端将服务下载到协作簇中任一服务器 j,并通过内部迁移形成目标服务器集合,本文假设从云端下载服务所耗费的成本为 B^{cloud}_j。

为了区分两种不同类型的待迁移用户,进一步定义二进制变量 $u\in\{0,1\}$,若 $u=1$,则表示待迁移用户为移动型待迁移用户,$u=0$ 则表示待迁移用户为负载型待迁移用户。对于两种待迁移用户,目标服务器集合 $S_n(t)$ 内部的服务迁移成本为:

$$S^{\text{in}}_n(t)=\sum_{k\in S_n(t)}S_{\text{cost}}(j,k,t) \tag{4}$$

用户 n 将数据迁移到目标服务器集合 $S_n(t)$ 的数据迁移成本为:

$$D^{\text{in}}_n(t)=\sum_{j\in S_n(t)}D_{\text{cost}}(L_n(t),j) \tag{5}$$

考虑负载型待迁移用户所需服务是否从云端下载服务,用户 n 的迁移总成本可以表示为:

$$TC_n(t)=(1-q_{\Re_n(t)})(uS_{\text{cost}}(i,j,t)+(1-u)B^{\text{cloud}}_j)+\sum_{k\in S_n(t)}(S_{\text{cost}}(j,k,t)+D_{\text{cost}}(L_n(t),j)) \tag{6}$$

假设服务器 e 缓存资源大小表示为 c_e,计算资源大小为 $f^{\max}_e$,分配给用户 n 的计算资源为 f^e_n。假设网络系统有 PN 类服务,表示为 $P=\{1,2,\cdots,p,\cdots,PN\}$,服务 p 的大小为 c^{size}_p。服务器 e 根据缓存资源大小 c_e 缓存的服务类型集合表示为 $C^s_e=\{p\mid F_{e,p}=1,p\in P\}$,其中 $F_{e,p}$ 为二进制变量,表示服务器 e 是否缓存服务 p,为 1 表示缓存,为 0 表示否。为了描述需要任务迁移的用户,将 t 时刻待迁移用户集合表示为 $U(t)$。通过迁移成本分析,结合边缘服务器的缓存、计算资源及用户时延约束,本文的优化目标可以表示为:

$$\begin{aligned}
\text{P1}:\ &\min_{(X(t),I(t))}\sum_{t=0}^{T}\sum_{n\in U(t)}TC_n(t)\\
\text{s.t. c1}:\ &\|\Re_n(t)\|\leqslant E,\forall n\in N\\
\text{c2}:\ &H^d_{j,L_n(t+1)}\leqslant H_{\max},\forall i\in M,j\in E\\
\text{c3}:\ &T^n(t)\leqslant t_{\max}\\
\text{c4}:\ &q_{\Re_n(t)}\in\{0,1\},\forall n\in N\\
\text{c5}:\ &I_{n,e}(t)\in\{0,1\},\forall n\in N,e\in E\\
\text{c6}:\ &X_{n,e}(t)\in\{0,1\},\forall n\in N,e\in E\\
\text{c7}:\ &f^e_n<f^{\max}_e,\forall n\in N,e\in E\\
\text{c8}:\ &\sum_{n\in N}f^e_n\leqslant f^{\max}_e,\forall e\in E\\
\text{c9}:\ &u\in\{0,1\}\\
\text{c10}:\ &\sum_{p\in C^s_e}c^{\text{size}}_p\leqslant c_e,\forall e\in E
\end{aligned} \tag{7}$$

其中,$\|\Re_n(t)\|$ 表示集合 $\Re_n(t)$ 的长度;$X(t)=\{X_{n,e}(t)\mid n\in N,e\in E\}$,$I(t)=\{I_{n,e}(t)\mid n\in N,e\in E\}$。约束 c1 确保协作簇的规模不超过服务器总数 E;约束 c2 确保跳数不会超过网络中的最大跳数;约束 c3 表示待迁移用户 n 的总时延 $T^n(t)$ 不超过最大容忍时延 $t_{\max}$,$T^n(t)$ 的计算式如式(16)所示。约束 c5 和 c6 分别是分簇决策变量和服务预缓存决策变量约束,$X_{n,e}(t)$ 表示 t 时刻服务器 e 是否属于待迁移用户 n 的协作簇,$I_{n,e}(t)$ 表示 t 时刻服务器 e 是否缓存待迁移用户 n 的服务。其中 $X_{n,e}(t)$ 和 $I_{n,e}(t)$ 均为二进制变量,为 0 表示否,为 1 表示是。c7 表示服务器 e 分配给待迁移用户 n 的计算资源 f^e_n 应小于当前边缘服务器 e 的最大资源 $f^{\max}_e$;c8 表示服务器 e 分配的计算资源总数不超过服务器 e 的最大资源 $f^{\max}_e$;c9 确保待迁移用户属于两个类型中的一种用户;c10 确保每个服务器 e 缓存服务的资源总大小不超过服务器 e 缓存资源最大容量。

3 问题分析

本文的目标是对 $I_{n,e}(t)$ 和 $X_{n,e}(t)$ 进行求解,而 $I_{n,e}(t)$ 和 $X_{n,e}(t)$ 均为二进制变量,其实质是解决一个混合二进制整数规划问题,该问题通常在线性计算时间内没有最优解,属于 NP 完全问题。因此,无法直接求解该问题的精确解,只能寻求近似解或局部最优解。本文首先对约束问题进行理论分析。

对于约束 c3,由于不同迁移类型用户的总时延不同,尤其是负载型待迁移用户,若簇中均未缓存用户 n 所需服务,即 $q_{\Re_n(t)}=0$ 时,需要从云端下载服务到协作簇中。一般情况下,用户的总时延包括上传时延、计算时延和通信时延。然而,对于移动型待迁移用户,用户已经将任务上传至原服务器 i,因此移动型待迁移用户的总时延包括计算时延和通信时延。

假设 $g^1_n(t)$ 为 t 时刻移动型待迁移用户剩余任务的大小,

$g_n^2(t)$为 t 时刻负载型待迁移用户任务大小，r_n^{up} 表示用户 n 的上行数据速率，则负载型待迁移用户的上传时延可以表示为：

$$G_n^{up}(t)=g_n^2(t)/r_n^{up} \tag{8}$$

移动型待迁移用户计算时延 $D_n^{pro1}(t)$和负载型待迁移用户计算时延 $D_n^{pro2}(t)$可以表示为：

$$D_n^{pro1}(t)=g_n^1(t)/(\| S_n(t)\| \max_{j\in S_n(t)} f_n^j) \tag{9}$$

$$D_n^{pro2}(t)=g_n^2(t)/(\| S_n(t)\| \max_{j\in S_n(t)} f_n^j) \tag{10}$$

其中，$\| S_n(t)\|$ 表示用户 n 目标服务器集合大小，$\max_{j\in S_n(t)} f_n^j$表示 $S_n(t)$中服务器 j 分配给用户 n 最大的计算资源。

在边缘计算网络中，待迁移用户和边缘服务器之间的通信时延一般包括服务迁移时延、数据传输时延和结果回传时延。本文令 i 表示原服务器编号，j 表示目标服务器编号，表示为：

$$j=\arg\min_{k\in S_n(t)}(H^d_{L_n(t),k}) \tag{11}$$

服务迁移时延主要由 $H^s_{i,j}$(跳数)决定[19-20]。移动型待迁移用户和负载型待迁移用户的数据传输时延均由传输的数据量、传输距离和链路带宽决定。由于带宽和数据传输信息是已知的，而传输距离可以用跳数 $H^d_{L_n(t),j}$来表示，因此用户 n 的服务迁移时延和输出传输时延可以表示为跳数相关的函数。设从云端下载服务的时间为 T_c，迁移服务一跳的时延为 t_s，则移动型待迁移用户服务迁移时延 $L^n_{s_1}(t)$和负载型待迁移用户服务迁移时延 $L^n_{s_2}(t)$为：

$$L^n_{s_1}(t)=(1-q_{\Re_n(t)})H^s_{i,j}t_s \tag{12}$$

$$L^n_{s_2}(t)=(1-q_{\Re_n(t)})T_c \tag{13}$$

令传输数据一跳的时延为 t_c，数据传输时延为 $L^n_d(t)=H^d_{L_n(t),j}t_c$，结果回传时延为 $L^n_{re}(t)=H^d_{j,L_n(t+1)}t_c$。因此移动型待迁移用户的通信时延 $L^1_n(t)$和负载型待迁移用户的通信时延 $L^2_n(t)$可以分别表示为：

$$L^1_n(t)=L^n_{s_1}(t)+L^n_d(t)+L^n_{re}(t) \tag{14}$$

$$L^2_n(t)=L^n_{s_2}(t)+L^n_d(t)+L^n_{re}(t) \tag{15}$$

根据式(8)－式(15)，结合前文提到的变量 u，待迁移用户的总时延可以表示为：

$$\begin{aligned}T^n(t)=&(1-u)G^{up}_n(t)+(ug^1_n(t)+(1-u)g^2_n(t))/\\&(\| S_n(t)\| \max_{j\in S_n(t)} f^j_n)+u(1-q_{\Re_n(t)})H^s_{i,j}t_s+\\&(1-u)(1-q_{\Re_n(t)})T_ct_s+H^d_{L_n(t),j}t_c+H^d_{j,L_n(t+1)}t_c\end{aligned} \tag{16}$$

4 问题转化

从式(16)可以看出，待迁移用户 n 的总时延 $T^n(t)$与 $g^1_n(t)$，$g^2_n(t)$，$H^s_{i,j}$，$H^d_{L_n(t),j}$，$H^d_{j,L_n(t+1)}$成正比，说明待迁移用户 n 任务越大，原服务器 i 或当前用户小区 $L_n(t)$到目标服务器 j 的跳数越大，待迁移用户 n 的总时延也越大。同时，待迁移用户 n 的总时延 $T^n(t)$与 $S_n(t)$成反比，$\| S_n(t)\|$ 越大，$T^n(t)$越小。

然而，用户的任务大小是无法改变的，只能通过控制跳数和目标服务器集合数量来减小待迁移用户 n 的总时延，而跳数往往会随着目标服务器集合数量的增加而增加，目标服务器集合数量也会随着协作簇大小的增加而增加，因此本文通过控制分簇变量 $X(t)$为用户建立最优协作簇，利用协作簇半径控制 $H^d_{L_n(t),j}$和 $H^d_{j,L_n(t+1)}$的值以及目标服务器集合数量 $\| S_n(t)\|$。同时，若对于每个待迁移用户 n 总有 $q_{\Re_n(t)}=1$，则会大大降低待迁移用户 n 的总时延，通过控制服务预缓存决策变量 $I(t)$，尽可能使每个待迁移用户 n 的 $q_{\Re_n(t)}$值为 1。$q_{\Re_n(t)}$的值，不仅与服务缓存决策变量 $I(t)$有关，也与待迁移用户 n 的协作簇 $\Re_n(t)$有关，$\Re_n(t)$集合越大，$q_{\Re_n(t)}=1$ 的概率也就越大。因此，本文的优化问题可以转换为对分簇变量 $X(t)$和服务预缓存决策变量 $I(t)$的联合优化问题。

本文提出的动态协作簇是以用户为中心，假设一跳的平均距离为 α，协作簇的半径为 d，则跳数 $H^d_{j,L_n(t+1)}$之间的关系可以表示为 $d=\alpha H^d_{j,L_n(t+1)}$。协作簇半径 d 越大，意味着簇的规模越大，目标服务器集合 $\| S_n(t)\|$ 也随之增大，系统传输总时延会有效降低。然而，簇规模越大，服务部署中的搜索空间也就越大，所消耗的搜索时延和能耗也随之增高。因此，任务迁移过程中考虑系统性能的折中是必要的。

通过上文对待迁移用户总时延建模可知，P1 问题是对分簇决策变量 $X(t)$和服务预缓存决策变量 $I(t)$的组合优化问题，对于 P1 问题求解时间复杂度为 $O(2^{N^2E^2})$。因此，可以通过分别求变量 $X(t)$和 $I(t)$的局部最优解来得到 P1 的近似解，即 P1 问题可以分解为最优分簇策略 P2 和最优预缓存策略 P3 两个子问题。

最优分簇策略问题可以表示为：

$$\begin{aligned}&\text{P2}:\min_{X(t)}\sum_{n\in U(t)}A_n=\sum_{n\in U(t)}\sum_{e\in B_n}X_{n,e}(t)\\&\text{s.t. c1}-\text{c3},\text{c7}-\text{c10}\\&\quad\sum_{e\in B_n}X_{n,e}(t)\leqslant A_T\\&\quad H^d_{L_n}(t),e\leqslant d_{\max}\\&\quad 1\leqslant\| B_n\|\leqslant\| E\|\\&\quad X_{n,e}(t)\in\{0,1\},\forall n\in U(t),\forall e\in B_n\end{aligned} \tag{17}$$

服务预缓存策略问题可以表示为：

$$\begin{aligned}&\text{P3}:\max_{I(t)}\sum_{t=0}^{T}\sum_{n\in U(t)}q_{\Re_n(t)}\\&\text{s.t. c4}-\text{c6},\text{c11}\end{aligned} \tag{18}$$

对于最优分簇策略 P2，定义 B_n 表示当前用户 n 可用的备选协作边缘服务器集合，$\| B_n\|$ 表示集合 B_n 的长度，$1\leqslant\| B_n\|\leqslant\| E\|$。本文以待迁移用户为中心建立动态协作簇，考虑到协作簇半径的上限值 $d_{\max}$。由于网络内边缘服务器数量一定，对单个待迁移用户分配极多的边缘服务器，必然影响到其他用户对协作资源的使用，因此有必要对每个用户可使用的协作边缘服务器的数量设定上限。协作簇规模越大，意味着所消耗的协作资源量越多，因此本节以协作簇规模表征消耗的协作资源量，为每个迁移用户设定协作簇规模上限值 A_T，对于每个迁移用户 $n\in U(t)$的约束可以表示为：

$$\sum_{e\in B_n}X_{n,e}(t)\leqslant A_T \tag{19}$$

对于最优服务预缓存策略 P3，簇中服务器缓存的命中率越高，$q_{\Re_n(t)}$为 1 的总次数越高，服务迁移的次数越少，进而减少服务迁移时延和迁移成本。

5 问题求解

5.1 最优分簇策略问题求解

P2 属于有约束的 0-1 整数规划问题,优化变量维度为 $N\times E$,为了降低 P2 中优化变量的维度以及整数规划的复杂度,可以对待迁移用户依据预测离开小区的时间 t_n^l 排序,每次只求解一个待迁移用户的最优分簇策略,优化变量维度为 E,进而求出所有用户的最优分簇策略。完成对 $U(t)$ 内用户的排序操作后,对于当前分簇用户 $n\in U(t)$,最优分簇策略问题 P2 可以转化为问题 P4:

$$\begin{aligned}&\text{P4}:\min\sum_{e\in B_n}X_{n,e}(t)\\&\text{s.t. c1}-\text{c3},\text{c7}-\text{c10}\\&\sum_{e\in B_n}X_{n,e}(t)\leqslant A_T\\&H^d_{L_n(t),e}\leqslant d_{\max}\\&1\leqslant\|B_n\|\leqslant\|E\|\\&X_{n,e}(t)\in\{0,1\},\forall e\in B_n\end{aligned}\tag{20}$$

对于问题 P4,本文提出了分布式多服务器间动态协作分簇(DDMC)算法来进行求解。首先,将待迁移用户集合 $U(t)$ 内用户依据 t_n^l 从小到大进行排序。然后,针对每个待迁移用户,筛选簇半径 $d_{\max}$ 范围内满足用户所需资源的边缘服务器,将其加入集合 B_n 中。最后,如果 $\|B_n\|$ 不超过协作簇规模上限 A_T,则将 B_n 中服务器归入协作簇 $\hat{A}_n(t)$ 中;如果 $\|B_n\|$ 超过了协作簇规模上限 A_T,则将 B_n 中服务器依据服务器 j 到用户下一时刻小区的跳数 $H^d_{j,L_n(t+1)}$ 进行从小到大排序,将 B_n 中前 A_T 个服务器归入协作簇 $\hat{A}_n(t)$ 中。DDMC 算法如算法 1 所示。

算法 1 DDMC 算法

输入:待迁移用户集合 $U(t)$,簇中服务器最大数量 A_T

输出:每个待迁移用户的协作簇 $\Re_n(t)$

1. 使用归并排序,根据 t_n^l 对集合 U(t)中元素从小到大排序
2. for n∈U(t) do
3. 根据簇半径 d 的上限值 d_{max}
4. for e∈E do
5. if $f_n^e>f_n^{min}$ and $\alpha H^d_{L_n(t+1),e}<d_{max}$ do / * f_n^{min} 为用户 n 当前服务所需最小计算资源 * /
6. 将服务器 e 加入集合 B_n 中
7. endif
8. endfor
9. 使用归并排序,根据 $H^d_{L_n(t+1),e}(e\in B_n)$对 B_n 元素从小到大排序
10. $\alpha_n=\min(\|B_n\|,A_T)$
11. 将 B_n 中前 α_n 个元素记为 $\tilde{A}_n$
12. 集合 $\tilde{A}_n$ 中的所有服务器归入协作簇 $\Re_n(t)$,$x_{n,e}(t)=1,\forall e\in\Re_n(t)$
13. endfor

本文以迁移用户为中心建立动态协作簇,具体建簇过程如图 2 所示。假设小区 1,2,3,4 都配有服务器,且每个服务器的最大负载数为 4(同时服务的用户数),小区 1,2,3 的预迁移半径为 2 和小区 4 的预迁移半径为 1($r_1(t)=r_2(t)=r_3(t)=2,r_4(t)=1$)。当迁移用户需要迁移时,可见小区 2,4 所在中心服务器负载过大,而小区 1 和 3 所在中心服务器负载过小,有剩余资源,此时小区 1 和 3 所在中心服务器形成了迁移用户的协作簇。

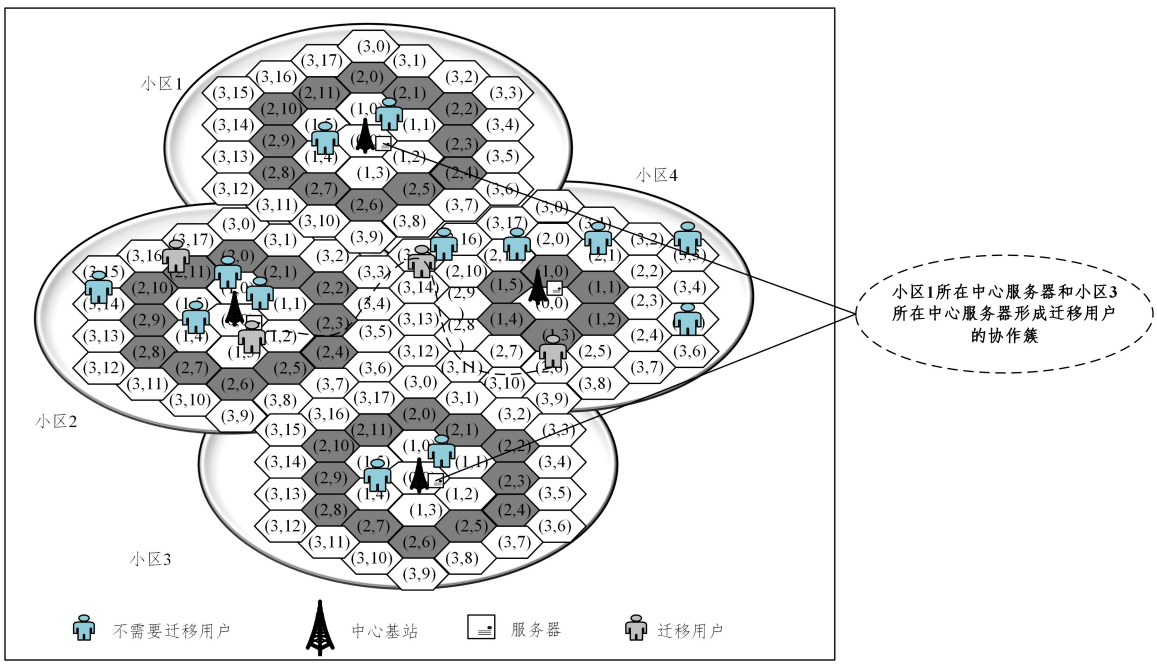

图 2 迁移用户分簇示意图

Fig. 2 Schematic diagram of migrated user clustering

5.2 服务预缓存策略问题求解

对于 P3 问题,本文将深度 Q 网络应用于服务缓存中,设计了一种面向服务缓存的深度强化学习(C-DDQN)算法。针对服务缓存场景,本文将每个边缘服务器作为智能体,并分别给出了状态空间、动作空间、奖励函数的定义。

1)状态空间。本文将任意时隙 t 处的系统状态定义为：

$$s_t \triangleq \{C_t^s, C_t^{\text{freq}}, C_t^{\text{rect}}, C_t^{\text{retime}}, C_t^{\text{size}}, C_t^{\text{remain}}\} \tag{21}$$

其中，C_t^s 表示 t 时刻缓存服务类型，C_t^{freq} 表示 t 时刻缓存服务的请求频率，C_t^{rect} 表示 t 时刻缓存服务最近请求距现在的时间间隔，C_t^{retime} 表示 t 时刻缓存服务最近请求的时间，C_t^{size} 表示 t 时刻缓存服务类型的大小，C_t^{remain} 表示 t 时刻缓存剩余空间。

2)动作空间。本文将动作空间定义为逐出缓存上哪个服务类型或不逐出的决策集合，长度为缓存大小加 1，动作集合为 $A=\{a_1, a_2, \cdots, a_r\}$。

3)奖励函数。根据环境的变化采取相应的动作，取得尽量高的奖赏。评价这个模型是否优秀，就看使用它能否取得尽量高的奖赏，因此我们需要去量化奖赏。对于缓存，本文希望命中更多的请求，以获得更大的奖赏，可以得到 R_t 的表示为：

$$R_t = \sum_{t=0}^{T} q_p' \tag{22}$$

其中，$q_p' = \{0,1\}$，$p \in P$，如果服务 p 被缓存，则 $q_p'=1$，否则为 0。将 s_{t+1} 输入到 Q 网络中得到最大输出值的动作 a^*，表达式如下：

$$a^* = \text{argmax}\ Q(s_{t+1}, a; \theta_t) \tag{23}$$

将 s_{t+1}，a^* 输入到 target 网络 $\hat{Q}$ 中，得到这个动作对应的目标网络的输出值 $\hat{Q}(s_{t+1}, a^*; \theta_t^-)$，则网络的实际值 $y_t^{\text{C-DDQN}}$ 的表达式如下：

$$y_t^{\text{C-DDQN}} = R_{t+1} + \gamma \hat{Q}(s_{t+1}, a^*; \theta_t^-) \tag{24}$$

将 $Q(s_{t+1}, a; \theta_t)$ 作为预测值，进行误差反向传播，均方差损失函数可以表示为：

$$L(\theta) = E((y_t^{\text{C-DDQN}} - Q(s_{t+1}, a^*; \theta_t))^2) \tag{25}$$

通过神经网络的梯度反向传播来更新 Q 网络的所有参数，每 N^- 步更新目标 $\hat{Q}$ 网络参数，具体伪代码如算法 2 所示。

算法 2 C-DDQN 算法

输入：迭代轮数 M，动作集 A，步长 α，衰减因子 γ，探索率 ε，经验回放池 D，当前 Q 网络参数 θ，目标 $\hat{Q}$ 网络 $\theta^- = \theta$，批量梯度下降的样本数 N_b，目标 $\hat{Q}$ 网络参数更新频率 N^-，回放池大小 N_r

```
for episode=1,2,…,M do
    初始化 s 为当前状态序列的第一个状态 s0
    for t=0,1,…,T-1 do
        使用 ε-greedy 选择动作 at
        在状态 s 执行当前动作 at，得到新状态 st+1 和奖励 Rt
        if |D|≥Nr then
            替换经验回放池 D 中最早的存储
        else
            将{st, at, Rt, st+1}存验回放集合 D
        endif
        st=st+1
        从经验回放集合 D 中采样 Nb 个样本{sj_t, aj_t, Rj_t, sj_t}，j=1,2,…,Nb 计算当前目标 Q 值
        if sj_t+1 is terminal
            yj_C-DDQN = Rj_t
        else
            yj_C-DDQN = Rj_t+1 + γ Q^(sj_t+1, argmax Q(sj_t+1, aj_t; θt); θt^-)
            使用均方差损失函数 L(θ)=E((yt_C-DDQN - Q(st+1, at; θt))^2)
        endif
        通过神经网络的梯度反向传播来更新 Q 网络的所有参数
        每 N^- 步更新目标 Q^ 网络参数
    endfor
endfor
```

5.3 最优迁移选择策略

算法 1 可以得到待迁移用户 n 的协作簇 $\mathfrak{R}_n(t)$，算法 2 可以得出服务器是否缓存用户所请求的服务，即 $I_{n,e}(t)$ 的值，其中 $e \in \mathfrak{R}_n(t)$。在任务迁移过程中，$S_n(t)$ 的选择会直接影响到待迁移用户 n 的时延和迁移成本，因此 $S_n(t)$ 的选择策略是非常重要的。为了得到迁移的目标服务器集合 $S_n(t)$，本文基于 DDMC 算法和 C-DDQN 算法的结果设计了迁移目标服务器选择算法。为了满足用户的最大时延，本文设置了为待迁移用户 n 服务的服务器最小个数 $ME_n(t)$。在 t 时间段，通过 DDMC 算法得到每个用户的动态协作簇集合 $\mathfrak{R}_n(t)$。对于每个用户的协作簇 $\mathfrak{R}_n(t)$，通过 C-DDQN 算法判断用户的请求服务是否缓存在服务器 e 中，即 $I_{n,e}(t)$ 的值，其中 $e \in \mathfrak{R}_n(t)$。如果 $I_{n,e}(t)$ 为 1 且目标服务器集合的数量 $\| S_n(t) \| < ME_n(t)$，则把服务器 e 加入目标服务器集合 $S_n(t)$ 中。如果用户的协作簇中缓存用户请求服务的服务器数小于 $ME_n(t)$ 的长度，则将从缓存用户请求服务的服务器迁移服务到簇中未缓存的 $ME_n(t) - len(S_n(t))$ 个服务器上，具体过程如算法 3 所示。

算法 3 迁移选择算法

输出：迁移目标服务器集合

```
for t∈{0,1,2,…} do
    通过 DDMC 算法得到协作簇 Rn(t)
    for n∈U(t) do
        for e∈A^n(t) do
            通过 C-DDQN 算法得到 In,e(t)
            if In,e(t)=1 and ‖Sn(t)‖<MEn(t):
                将 e 加入目标服务器集合 Sn(t)中
            else:
            将 e 加入集合 S^no_n(t)中/*S^no_n(t)未缓存服务的服务器集合*/
            endif
        endfor
        if len(Sn(t))<MEn(t)
            将 S^no_n(t)中距用户下一位置最近的 MEn(t)-len(Sn(t))个服务器加入 Sn(t)
            endif
            将 Sn(t)加入集合 S(t)中/*S(t)为所有用户的目标服务器集合*/
    endfor
endfor
```

6 实验结果与分析

为了测试算法效果，本文利用合作项目中河南某联通

公司提供的 OTT 脱敏数据进行性能验证，数据标签类别包含用户 ID、基站经纬度、时间、服务类型等。数据集的基站数目为 590 个，实验过程中选择区域内用户单天的移动行为数据，筛选后选取的测试用户数为 30 个，共 47524 条记录数据。以 2 个小时为切片进行统计，一天内的服务请求数分布如图 3 所示，由图可知 12:00－14:00 和 18:00－20:00 服务请求数最多。

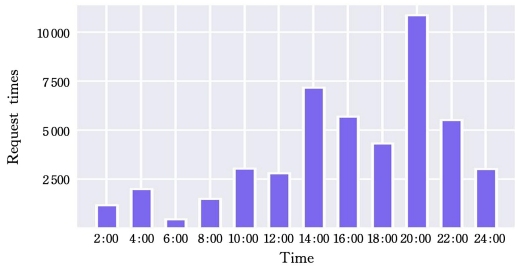

图 3 一天内的服务请求数

Fig. 3 Number of service requests in a day

为了验证服务缓存算法的性能，本文选择先进先出算法[21]（First In, First Out, FIFO）、最近最久未使用算法[22]（Least Recently Used, LRU）、最近最少使用算法[23]（Least Frequently Used, LFU）、Random 缓存、GDS[24]（Greedy Dual Size）、DQN[25] 6 种不同的边缘缓存策略作为比较算法。其中 GDS 使用目标价值函数来计算缓存中所有服务对象的价值，然后根据这个值对对象进行由高到低的排序，当缓存剩余空间不足以保存新的对象时就将缓存中价值最小的对象替换出去。

本文针对 6 种不同缓存算法以缓存命中率[7]为评价指标进行对比实验。图 4 给出了簇半径与缓存命中率之间的关系，纵坐标为一天内平均每两个小时的命中率。

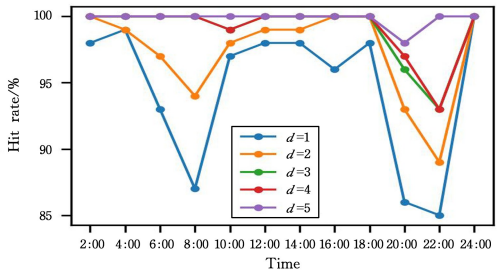

图 4 簇半径与命中率之间的关系

Fig. 4 Relationship between cluster radius and hit rate

实验过程中假设计算资源固定，设置服务器数量为 100，缓存大小为 4。从图中可以看出，协作簇半径越大，簇缓存命中率越大。由于协作簇半径越来越大，簇的规模也随之增大，目标服务器集合的数量也增大，因此簇缓存命中率也越来越大。当协作簇半径为 5 km 时，簇缓存命中率大部分时间段稳定在 100%。但是如前文分析，协作簇半径 d 不能过大，簇半径过大会增加搜索时延和能耗以及迁移成本。

图 5 给出了缓存大小与缓存命中率之间的关系，纵坐标为一天的平均命中率。实验过程中在服务器计算资源固定的情况下，设置协作簇半径为 1 km，服务器数量为 60。从图中可以看出缓存命中率与缓存大小存在正向关系，当缓存大小增大时，命中率也随之增大。由于缓存增大，服务器可以缓存的服务数量也随之增大，因此缓存命中率也会增大。本文提出的 C-DDQN 缓存算法无论缓存大小如何变化，命中率一直在 95%以上，可见该算法具有良好的鲁棒性。在缓存大小为 3 时的命中率为 95.47%，比 DQN 缓存算法提高了 10.32%。

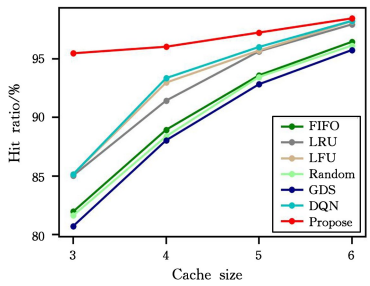

图 5 缓存大小与命中率之间的关系

Fig. 5 Relationship between cache size and hit rate

为了对迁移算法的总效果进行分析，本文选取了总是迁移（Always）、从不迁移（Never）、短视算法（Myopic）3 种不同的迁移算法进行比较。总是迁移算法也称为贪婪策略或最小跳数策略，当服务需要迁移时，服务总是迁移到最近的边缘服务器。从不迁移策略指服务一旦放置，就不再进行服务迁移。短视算法是基于服务质量观察做出迁移决策的短视策略，它同时考虑服务器间的负载均衡，选择相应的迁移动作以最小化一次性时隙开销。本文对 3 种不同迁移算法以时延、成本[26]和迁移成功率[27]为评价指标进行消融实验和对比实验。

针对本文算法进行消融实验，得到缓存大小和簇半径对实验结果的影响，如图 6 所示。

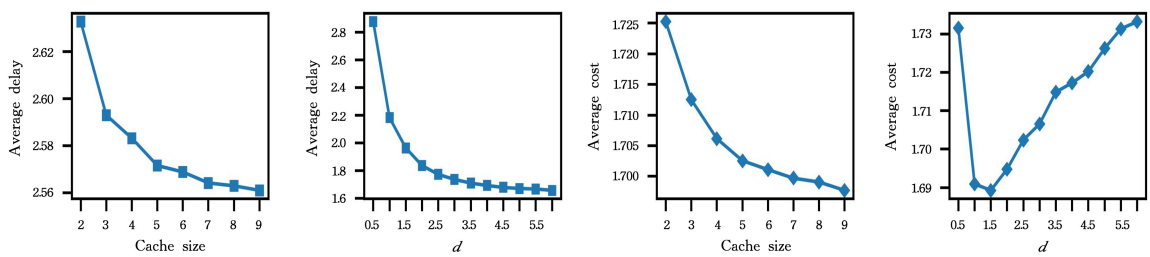

图 6 缓存大小和簇半径对实验结果的影响

Fig. 6 Effect of cache size and cluster radius on experimental results

由图 6 可知，时延与缓存大小、簇半径成反比关系，随着缓存大小和簇半径的增加，时延逐渐减小；成本与缓存大小成反比关系，随着缓存大小的增大，成本逐渐减小；成本随簇半径的增大先减小后增大，这是因为一开始随着簇半径的增大，簇缓存命中率也随之增大，进而降低了服务迁移成本。然而，随着簇半径的不断增大，簇的规模也随之增大，目标服务器

集合的数量也增大，进而增加了簇内迁移成本。图 7 分别给出了服务器数为 50，60，70，80 时一天内每两个小时平均时延的变化，纵坐标为一天内每两个小时的平均时延，实验过程中，设置缓存大小为 5，簇半径为 1.5 km。由图可知，本文算法受服务器数量变化的影响较小，Always 策略和 Myopic 策略受服务器数量变化的影响较大。因为随着服务器数量的增加，待迁移用户可选择的服务器数量也随之增加，可能会出现距离用户更近的服务器，这也与服务器放置位置存在一定的关系。本文算法无论在一天的哪个时间段，以及服务器数量如何改变，都优于其他算法，可见本文算法有良好的鲁棒性。

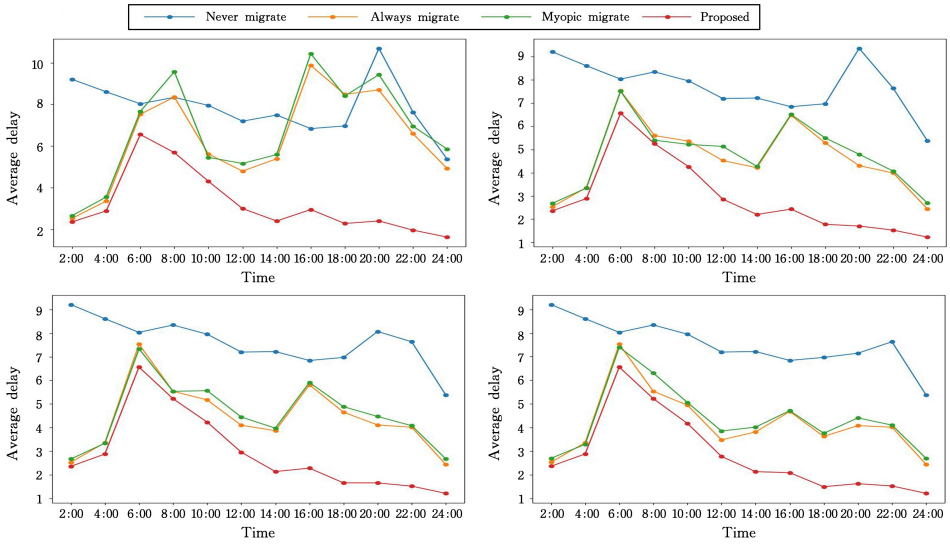

图 7 服务器数为 50，60，70，80 时不同算法的平均时延对比

Fig. 7 Comparison of average latency among different algorithms with 50，60，70，and 80 servers

图 8 给出了服务器数为 50，60，70，80 时一天内平均每两个小时成本的变化，纵坐标为一天内平均每两个小时的成本，实验中设置缓存大小为 5，簇半径为 1.5 km。由图可知，本文算法性能无论在服务器为多少时均优于其他算法，可见本文算法具有良好的鲁棒性、稳定性和优化性。

图 9 给出了一天的总时延随服务器数量的变化情况。由图可知，当服务器数为 60、缓存大小为 4、簇半径为 1 km 时，Never 策略和 Myopic 策略的总时延高于 Always 策略。Myopic 是一种短视策略，它能使某个时刻的开销最小，迁移时会考虑服务器的计算资源，是一种贪心策略，因此其总时延比 Always 策略略高一些。Always 策略总是迁移到距用户最近且有剩余计算资源的服务器上，虽然会降低数据迁移时延，但这必然会大大增加服务迁移时延。本文提出的迁移策略在总时延上比 Never 算法降低了 49.49%，比 Always 算法降低了 59.22%，比 Myopic 策略降低了 52.36%。

图 10 给出了一天的总成本随服务器数量的变化情况。由图可知，当服务器数为 60、缓存大小为 4、簇半径为 1 km 时，本文提出的迁移算法在总成本上比 Never 算法降低了 28.91%，比 Always 算法降低了 12.06%，比 Myopic 策略降低了 15.04%。本文算法在服务器数量增加时，成本依然低于其他算法，且变化幅度平缓，可见本文算法在降低总成本上依然具有良好的稳定性。

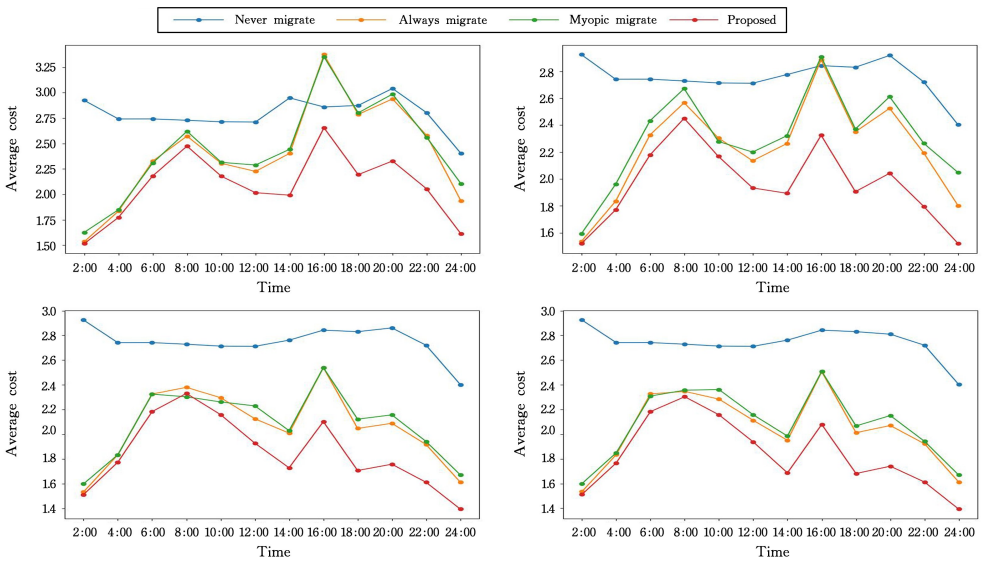

图 8 服务器数为 50，60，70，80 时不同算法的平均成本对比

Fig. 8 Comparison of average cost among different algorithms with 50，60，70 and 80 servers

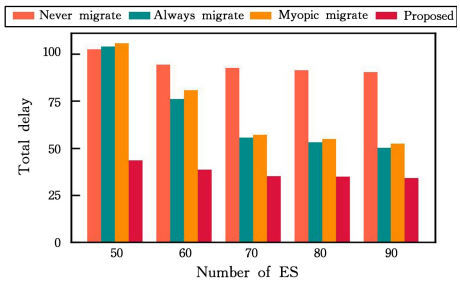

图 9 总时延与服务器数量的关系

Fig. 9 Total latency versus number of servers

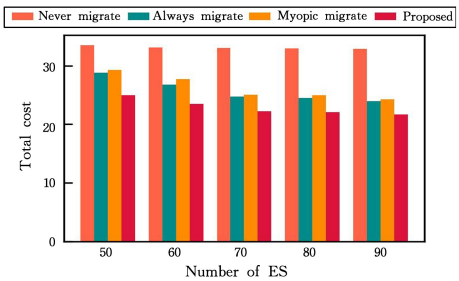

图 10 总成本与服务器数量的关系

Fig. 10 Total cost versus number of servers

图 11 给出了一天内平均每两个小时迁移成功率的变化情况。其中设置缓存大小为 5，协作簇半径为 1 km。从实验结果可以看出，18:00－20:00 服务请求数最大，在这个时间段内 Never、Always、Myopic 和本文提出的迁移策略的服务迁移成功率分别为 1.19%，24.96%，20.02%和 85.0%。因为本文算法考虑了多服务器协作，会大大降低用户时延。在这一天内，本文提出的迁移算法在任一时间段迁移成功率均高于其他迁移算法。

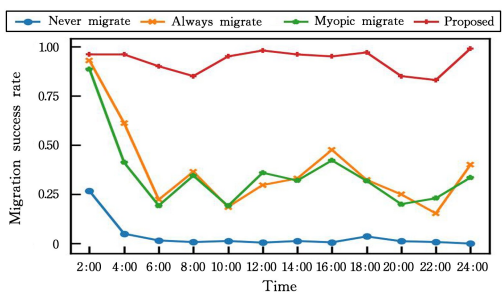

图 11 迁移成功率

Fig. 11 Migration success rate

结束语 为了保障用户的良好体验，提高任务的完成效率和质量，本文面向网络中的各种密集和延迟敏感任务，在网络性能优化、任务迁移、多服务器协作机制等方面进行了深入研究。通过构造服务迁移和数据迁移成本函数，将时延为约束的迁移成本最小化问题转化为分簇决策变量和服务预缓存决策变量组合优化问题，并分别提出了以用户为中心的动态协作簇分类 DDMC 算法以及面向服务缓存的深度强化学习 C-DDQN 算法，解决了最优分簇和服务缓存问题，有效降低了网络负载，提升了系统鲁棒性。本文研究虽然取得了部分成果，但研究场景主要针对用户移动场景，相比车联网等高速移动应用，移动速度较慢，因此未来的工作会考虑将算法部署在车联网中进行优化测试，以适应未来不同的用户业务需求。

参 考 文 献

[1] ZHANG J，HU X，NING Z，et al. Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching [J]. IEEE Internet of Things Journal，2018，6(3)：4283-4294.

[2] MIAO Y，WU G，LI M，et al. Intelligent task prediction and computation offloading based on mobile-edge cloud computing [J]. Future Generation Computer Systems，2020，102：925-931.

[3] ZHANG N，GUO S，DONG Y，et al. Joint task offloading and data caching in mobile edge computing networks [J]. Computer Networks，2020，182：107446.

[4] PENG K，NIE J，KUMAR N，et al. Joint optimization of service chain caching and task offloading in mobile edge computing [J]. Applied Soft Computing，2021，103：107142.

[5] LI C，ZHANG Y，GAO X，et al. Energy-latency tradeoffs for edge caching and dynamic service migration based on DQN in mobile edge computing [J]. Journal of Parallel and Distributed Computing，2022，166：15-31.

[6] TANG F，LIU C，LI K，et al. Task migration optimization for guaranteeing delay deadline with mobility consideration in mobile edge computing [J]. Journal of Systems Architecture，2020，112(8)：101849.

[7] LI C，ZHU L，LI W，et al. Joint edge caching and dynamic service migration in SDN based mobile edge computing [J]. Journal of Network and Computer Applications，2021，177：102966.

[8] OUYANG T，ZHOU Z，CHEN X. Follow me at the edge：Mobility-aware dynamic service placement for mobile edge computing [J]. IEEE Journal on Selected Areas in Communications，2018，36(10)：2333-2345.

[9] GE S，CHENG M，HE X，et al. A two-stage service migration algorithm in parked vehicle edge computing for internet of things [J]. Sensors，2020，20(10)：2786.

[10] YIN L，LI P，LUO J. Smart contract service migration mechanism based on container in edge computing [J]. Journal of Parallel and Distributed Computing，2021，152：157-166.

[11] WANG S，URGAONKAR R，ZAFER M，et al. Dynamic service migration in mobile edge computing based on Markov decision process [J]. IEEE/ACM Transactions on Networking，2019，27(3)：1272-1288.

[12] 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.

[13] XIE Q，WANG Q，YU N，et al. Dynamic service caching in mobile edge networks[C]// 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems(MASS). IEEE，2018：73-79.

[14] ZHAO T，HOU I H，WANG S，et al. Red/led：An asymptotically optimal and scalable online algorithm for service caching at the edge [J]. IEEE Journal on Selected Areas in Communications，2018，36(8)：1857-1870.

[15] CHEN L, XU J, REN S, et al. Spatio-temporal edge service placement: A bandit learning approach [J]. IEEE Transactions on Wireless Communications, 2018, 17(12): 8388-8401.

[16] ZHANG T, FAN H, LOO J, et al. User preference aware caching deployment for device-to-device caching networks [J]. IEEE Systems Journal, 2017, 13(1): 226-237.

[17] PENG T, WANG H, LIANG C, et al. Value-aware cache replacement in edge networks for Internet of Things [J]. Transactions on Emerging Telecommunications Technologies, 2021, 32(9): e4261.

[18] ZHANG W, WU D, YANG W, et al. Caching on the move: A user interest-driven caching strategy for D2D content sharing [J]. IEEE Transactions on Vehicular Technology, 2019, 68(3): 2958-2971.

[19] NAIR V, HINTON G E. Rectified linear units improve restricted boltzmann machines[C]// Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.

[20] TALEB T, KSENTINI A, FRANGOUDIS P A. Follow-me cloud: When cloud services follow mobile users[J]. IEEE Transactions on Cloud Computing, 2016, 7(2): 369-382.

[21] MICHAEL M M, SCOTT M L. Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors[J]. Journal of parallel and distributed computing, 1998, 51(1): 1-26.

[22] MEIZHEN W, YANLEI S, YUE T. The design and implementation of LRU-based web cache[C]// 2013 8th International Conference on Communications and Networking in China (CHINACOM). IEEE, 2013: 400-404.

[23] SOKOLINSKY L B. LFU-K: An effective buffer management replacement algorithm[C]// Database Systems for Advanced Applications: 9th International Conference (DASFAA 2004). Berlin Heidelberg: Springer, 2004: 670-681.

[24] CAO P, IRANI S. Cost-aware www proxy caching algorithms [C]// Usenix Symposium on Internet Technologies and Systems. 1997: 193-206.

[25] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning [J]. Nature, 2015, 518(7540): 529-533.

[26] LI C, SONG M, ZHANG M, et al. Effective replica management for improving reliability and availability in edge-cloud computing environment [J]. Journal of Parallel and Distributed Computing, 2020, 143: 107-128.

[27] LI C, CAI Q, LOU Y. Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud [J]. Future Generation Computer Systems, 2022, 127: 142-159.

ZHAO Xiaoyan, born in 1981, Ph.D, associate professor, is a member of CCF (No. K8282M). Her main research interests include mobile edge computing, D2D communication and Internet of things.

ZHANG Junna, born in 1979, Ph.D, associate professor, is a member of CCF (No. D2234M). Her main research interests include mobile edge computing and services computing.

(责任编辑:喻藜)