云边端异构算力网络计算任务分割与路径优化方法研究

马 博,余应洁,吴莎尘,倪 畅,陆 琴,陈 超,李传煌*
(浙江工商大学信息与电子工程学院,浙江杭州 310018)

摘 要: 在云边端算力网络中,传输、计算和存储资源的协同优化是一个关键且极具挑战性的课题. 如何有效融合高性能的云资源、低时延的边缘资源、广泛分布的节点资源以及低成本的用户资源,实现智能化的资源分发、关联、交易与调配,对于整网资源的最优化配置和高效利用意义重大. 本文针对传输与计算融合的云边端异构算力网络,构建了详细的数学模型. 从算力需求、资源分发、交易与调配等多个维度出发,将异构计算与传输资源调度中的时延和成本最小化联合优化问题,转化为混合整数非线性规划问题. 随后,本文提出了一种创新的串行子任务路径分配机制,并结合最优路径最大化分配算法(Optimal Route and Assign Maximizer algorithm,ORAM),以实现任务计算与传输路径的高效协同优化. 该机制将计算任务分割为多个子任务,感知并处理串联子任务之间的依赖关系,利用ORAM算法实时选择符合依赖关系的最优计算路径,指导计算结果以最少跳数的方式传输至目标节点,形成端到端的高效资源调度通道. 这不仅降低了传输时延和资源成本,还将传统的"先传后算"模式有效转变为"传算协同"模式. 实验结果显示,在不同的计算需求、感知范围和节点数量条件下,本文所提出的算法相较于多种基准算法,在时延、成本及路径优化等方面均表现出更优的性能.

关键词: 算力网络;任务调度;算力任务;传输路径;算力架构

基金项目: 国家自然科学基金(No.62401506, No.62301488, No.62302446);浙江省科技创新重点项目(No.2023R5211);浙江省自然科学基金(No.LZ23F010003,No.LQ23F010009);浙江省属高校基本业务费专项资金资助项目(No.QRK23009)

中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112(2025)06-1847-18

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20241050

第二十七届中国科协年会学术论文

Task Segmentation and Path Optimization in Heterogeneous Cloud-Edge-End Computing Power Network

MA Bo, YU Ying-jie, WU Sha-chen, NI Chang, LU Qin, CHEN Chao, LI Chuan-huang*
(*School of Information and Electronic Engineering*, *Zhejiang Gongshang University*, *Hangzhou* 310018, *China*)

Abstract: Collaborative optimization of transmission, computation, and storage resources in "cloud-edge-end" computing power networks is a critical and highly challenging task. Effectively integrating high-performance cloud resources, low-latency edge resources, widely distributed node resources, and low-cost user resources to achieve intelligent resource distribution, association, trading, and allocation are essential for the optimal configuration and efficient utilization of network-wide resources. This paper constructs a detailed mathematical model for the "cloud-edge-end" heterogeneous computing power networks with a focus on the integration of transmission and computation. Addressing multiple dimensions such as computing power demand, resource distribution, trading, and allocation, the joint optimization problem of minimizing delay and cost in scheduling heterogeneous computing and transmission resources is transformed into a mixed-integer nonlinear programming problem. Subsequently, an innovative serial sub-task path allocation mechanism is proposed, combined with the optimal route and assignment maximization (ORAM), to achieve efficient collaborative optimization of task computation and transmission paths. This mechanism divides computing tasks into multiple sub-tasks, perceives and manages

收稿日期:2024-11-21;修回日期:2025-05-06;责任编辑:覃怀银
*通讯作者:李传煌

the dependencies between serial sub-tasks, and utilizes the ORAM algorithm to select optimal computation paths that satisfy dependency relationships in real-time. It directs the transmission of computation results to target nodes with the fewest hops, thereby forming an end-to-end efficient resource scheduling channel. This approach not only reduces transmission delay and resource costs but also effectively transforms the traditional "transmit-then-compute" model into a "transmit-compute collaborative" model. Experimental results demonstrate that the proposed algorithm outperforms various benchmark algorithms in terms of delay, cost, and path optimization under different computational demands, sensing ranges, and node quantities.

Key words: computing power network; task scheduling; computing task; transmission path; computing power architecture

Foundation Item(s): National Science Foundation of China (No.62401506, No.62301488, No.62302446); Zhejiang Provincial Science and Technology Innovation Key Project (No. 2023R5211); Natural Science Foundation of Zhejiang Province (No.LZ23F010003, No.LQ23F010009); Fundamental Research Funds for the Provincial Universities of Zhejiang (No.QRK23009)

1 引言

随着移动5G应用的快速普及,网络环境呈现出数据海量化、组网异构化、网络拓扑动态化、边缘设备数量庞大化、应用及算力需求多样化等特征. 基于此,以算为中心、网为根基的算力网络[1]应运而生,旨在进一步融合高性能云资源、低时延边缘资源、广泛分布的节点资源以及低成本用户资源,并结合业务承载需求,提供智能化的计算、存储、网络等资源的分发、关联、交易与调配,从而实现整网资源的最优化配置和使用. 然而,算力网络面向业务智慧承载及异质资源融合管控处理也面临着新的挑战. 一方面,算力资源的种类和数量繁多,包括中央处理器(Central Processing Unit, CPU)、图形处理器(Graphics Processing Unit, GPU)、现场可编程逻辑门阵列(Field Programmable Gate Array, FPGA)等不同类型的计算资源,状态和可用性伴随时空变化,呈现明显的异质性;另一方面,算力资源分布广泛,如何将这些资源进行有效整合,形成一个统一的资源池,则需要解决多种技术和管理问题,特别是在面对不同地域、不同服务商的算力资源时. 要在这些挑战中取得突破,就需要在技术上深入研究,并制定出能够灵活应对这些复杂情况的资源调度策略. 此外,算力网络的安全性问题也日益突出,特别是在面临动态和开放环境下的安全威胁时,传统的静态防御策略显得力不从心,在以后安全性研究也相当重要[2].

已有的算力资源调度策略研究[3~6],通常是在对算力节点的资源状态和网络链路状态进行综合分析后,经过加权计算得出的. 然而,由于算力节点的设备配置在地理位置方面存在较大差异,容易因传输距离较长、网络丢包等原因导致任务传输和计算处理时延增加. 现有的异构网络分布式计算研究[4,5]通常局限于少量设备的协同计算,很少考虑到灵活调度的算力协同计算通道. 随机选择算力调度通道可能导致数据转发跳数大幅增加,从而增加传输时延[4]. 因此,研究者们[7~10]在算力成本优化方面,针对任务调度中的计算和传输时延问题展开了优化. 从用户角度来看,任务处理过程中涉及的算力交易也是一个值得关注的问题,可以将其纳入调度策略的考量范围. 对于可拆分型任务,任务的拆分情况会影响计算结果. 而现有可拆分任务研究[7,11,12]却很少考虑优化任务拆分方式. 拆分后任务常常是被无组织地分发至各计算节点,易造成通信资源占用.

针对目前研究存在的问题,本文提出了一种结合算力需求、分发、交易与调配的计算任务端到端低算力成本资源调度通道. 该通道将传统的"先传后算"模式转变为"传算协同"模式,研究了串行子任务路径分配机制以最小化计算与通信协同成本. 首先,本文对算力调度过程中的任务需求,算力供应以及算力交易进行了数学建模,并定义了一个由时延、付费组成的加权代价函数,将算力网络的计算和传输资源联合优化问题转化为混合整数非线性规划问题. 然后,将联合优化问题解耦成两个子问题,即求解最佳计算路径和求解最佳的任务分割比例问题,为了解决两个子问题,提出了一种最优路径的最大化分配算法(Optimal Route and Assign Maximizer algorithm, ORAM),该算法将当前位置到目标位置的时延与付费成本作为评估准则,在每一步选择当前最优解,通过不断做出局部最优选择来达到整体最优解. 基于此算法,以最佳路径为输入,对任务分割进行比例调整,进一步优化整体目标. 通过实验对比枚举、随机最优切割和能力最大化分配算法,从时延敏感型需求、付费敏感型需求、时间复杂度和算力感知等方面验证了ORAM算法在"云边端"算力网络架构下任务分割和时延付费优化的有效性.

2 相关工作

算力网络旨在将分布各地的计算节点互通互联、统筹调度,通过改进设计网络架构,搭建资源与服务的桥梁,实现网络中泛在计算资源的协同利用[3].

当前,算力网络的发展方兴未艾,体系架构及其诸多关键技术还未形成定论,产业化标准工作也正在不断推进中. 中国移动联合华为公司提出,算力网络是一种在云、边、端之间按需分配和灵活调度计算资源、存储资源以及网络资源的新型信息基础设施[13].《融算网络体系基础研究》[14]提出了融算网络的“三层三域”体系架构,从广义服务层、映射适配层和融合网络层的角度定义了算网融合的技术路径,并详细讨论了按需组网、多维标识和智能映射机制,这些研究为实现“云—边—端”协同提供了理论基础. 国际上,互联网工程任务组(Internet Engineering Task Force,IETF)成立了网内计算工作组(COINORG),致力于将分散的计算资源集成到网络中,推动网络与计算深度融合,实现“云—边—端”算力资源的协同调度[15]. 其利用网络中分布的各种计算资源,包括云端服务器、边缘节点、终端设备等,形成一个资源池,并通过智能任务调度和分配策略将任务分发给最合适的节点进行处理[16].

支撑异构网络与异质资源协同的网络架构及计算模式是任务调度研究的重点,Nasirian等人[17]设计了一种以服务器为中心、递归方式构建网络拓扑的数据中心模式,用以提供良好的网络容量与低延时调度,支持延迟敏感和数据密集型应用. 但其部署模式过于集中,不易拓展,难以根据算力进行任务调度. Li等人[18]提出了一种完全合作的多智能体任务调度框架,旨在通过多接入边缘计算(Multi-access Edge Computing,MEC)辅助虚拟现实(Virtual Reality,VR)应用,优化任务调度过程. 该研究强调了多节点之间的协作,而非每个节点独立决策,提出通过共享共同目标提高任务调度的成功率和延迟合规性. Song等人[5]设计了一种多服务任务计算卸载算法(Multi-Service Task Computation Offloading Algorithm,MTCOA)来寻找最优解,该方法改进了遗传算法,考虑了基因对染色体的影响因素,设计了交叉选择的概率公式,根据概率公式,收敛效果将更加明显,并且该算法在可接受的性能和时间复杂度之间进行权衡,从而得到了一个较优的任务分配,极大提高了资源的利用率. 但该卸载方法仅适用于数据回传的模式,且当任务量较大时,算法的性能和时间复杂度可能会成为瓶颈. Ren等人[19]提出了一种名为边缘矩阵的云边协同资源优化框架. 文章利用多智能体强化学习技术来适配网络系统中的任务调度组合,从而在最小化时延的基础上保证不同任务的可靠完成. 然而其提出的云边协同架构虽然在一定程度上能够满足部分用户的服务需求,但是其任务调度模式基于设备商提供的高算力设备,未考虑用户成本限制,以及算力运营商的设备部署可能面临着成本较高的问题.

随着如今用户移动终端设备性能的显著增强,许多研究学者提出将海量的终端设备也纳入到网络架构中,设计面向众包概念的终端协同计算范式[6],可以预见,通过整合高端性能的云服务器,低延时的边缘服务器以及低成本的终端设备,能够满足不断增长的计算需求,这种协同计算的模式被认为是未来算力网络的主要发展方向之一. Su等人[20]站在算力提供视角,在综合考虑了储存、服务、部署等方面成本后,设计了一种在多维约束下的联合服务放置和请求调度的在线算法,该算法将离线代价最小化问题解耦为多个凸子问题,每个时间槽都能有效进行求解,然后将凸子问题的最优分数解转化为原始代价最小化问题的整数解,旨在使服务运营成本最小化. 而对于用户而言,任务计算调度作为优化问题[11],低时延是算力网络任务调度所需考虑的关键要素之一. 面对算力网络的低时延传输需求,研究者们从不同角度给出了方案. Tang等人[21]提出服务意图感知任务调度框架(Service Intent - aware Task Scheduling,SIaTS),通过计算资源标识符(Cloud Resource IDentifier,CRID)和服务意图感知机制,实现任务意图与算力网络(Computing Power Network,CPN)资源的最优匹配. 他们还开发了基于双重维克里拍卖的调度算法,提升CPN节点收入和服务意图匹配度. 这表明服务意图感知调度能显著提升用户体验,尤其在低时延场景中. Chen等人[7]深入研究了以最小化时延为目标的任务调度问题,并提出了一个基于博弈论的分布式任务调度算法. 而任务特征和资源需求的不确定性也会对算法的性能产生影响,其可能面临计算复杂度、信息共享成本、系统动态性和博弈均衡等问题. Al-habob等人[8]提出了一种基于遗传算法的任务调度的方案,以便将一个对延迟敏感和计算密集型的任务并行和顺序卸载到多服务器上. 该方案基于遗传算法和冲突分析图模型,使得性能接近最优方案. 然而,遗传算法存在一些局限性,例如对种群进行多次迭代和适应度评估的需求,这会导致较高的计算成本. 此外,在顺序卸载过程中,遗传算法可能会导致调度时间过长,从而无法满足实时场景中对时延敏感任务的实时性要求. 任务调度问题通常可以概括为一个整数非线性规划问题,Feng等人[22]提出了一种新的任务调度方法,关注如何在计算能力网络(CPN)中实现任务调度的延迟优化与负载平衡. 该研究提出了一个约束的马尔可夫决策过程(Constrained Markov Decision Process,CMDP)模型,并通过深度强化学习算法来优化任务调度. Tun等人[9]研究了移动用户的延迟最小化问题,该问题是一个非凸、混合整数非线性规划(Mixed Integer NonLinear Programming,MINLP)问题,针对该问题文章将该其分解为三个子问题,然后采用拉格朗日松弛法和交替方向乘子法(Alternating Direction Method of Mul-

tipliers，ADMM）对分解后的问题进行求解，得到最优调度策略，但对于算力网络任务调度场景，该研究还需考虑算力成本等问题. Sun等人[23]研究了无服务器计算环境中，如何在硬性截止时间下优化任务调度与带宽分配. 他们提出了一种新的算法指针网络引导的资源调度（Pointer Network - Guided Resource Dispatch，PN-GRD），通过指针网络模型对任务优先级进行推断，并选择最适合的边缘节点进行任务分配，以最大化整体利润. You等人[10]提出了一种基于粒子群优化（Particle Swarm Optimization，PSO）的任务卸载策略，该策略将任务从资源受限的边缘设备卸载到能源效率高、延迟低的MEC服务器中，降低总成本. 但PSO算法在处理大规模任务卸载问题时，当任务数量增加时，粒子数量也会相应增加，导致算法的计算复杂度增加. 且将任务单一地卸载至某一台MEC设备，对设备的要求会很高且可能产生较长时间的设备占用. 同样，Huang等人[24]探讨了在边缘云环境中优化任务卸载的问题. 他们设计了一种多目标优化模型，综合考虑了任务执行时间、负载均衡及用户隐私保护，通过NSDE（Non-dominated Sorting Differential Evolution）算法优化了任务卸载的效率. Naouri等人[12]提出了一种由设备层、集群层和云层组成的三层任务卸载设备-集群-云（Device-Cluster-Cloud，DCC）框架，引入了一个贪婪任务图分区（Greedy Task Graph Partitioning，GTGP）卸载算法，利用贪婪优化方法，根据设备的计算能力来辅助任务调度过程，使得任务通信，计算成本最小化，但付费成本仍没有被考虑. Mtshali等人[25]提出了一种多目标模拟退火（Multi-Objective Simulated Annealing，MOSA）算法，将付费成本纳入了任务调度的考虑范围，通过在约定时间内分配任务，实现了时间和付费目标之间的最佳权衡. 然而，该算法的任务卸载方式并未考虑任务发送和接收的网络位置影响，因此在计算方式上可以考虑引入路径计算以进一步优化任务调度.

基于以上研究工作的局限性，本文旨在整合高性能云计算、低时延边缘计算、广泛分布的节点和低成本用户终端，通过感知技术实现对网络云、边、端的计算资源进行统筹管理，并灵活地调度这些资源，形成资源通道，以实现时延和成本的最小化. 同时，通过智能分配和切割任务，进一步优化资源的利用效率，确保计算资源的高效利用，并实现任务的高效处理.

3 系统模型与问题描述

3.1 系统架构模型

本文主要针对云边端协同的泛在算力网络架构，如图1所示，该网络架构主要包含云计算层、边缘计算层和接入算力网络的终端设备层. 为了适应网络设备无线化趋势，图中终端设备具备了通过无线方式连接到各个算力节点的能力，包括私网集群服务器、公网集群服务器以及云服务器，以及通过算力控制选择具体传输方式的能力. 云计算层位于最上层，离终端设备距离较远，具有强大的算力. 边缘节点通常位于核心公网内或通过私有网组成边缘计算集群，靠近用户端，承担着任务计算的主要角色. 终端设备层产生资源需求高且时延敏感任务，相较于其他层的计算设备，算力资源有限.

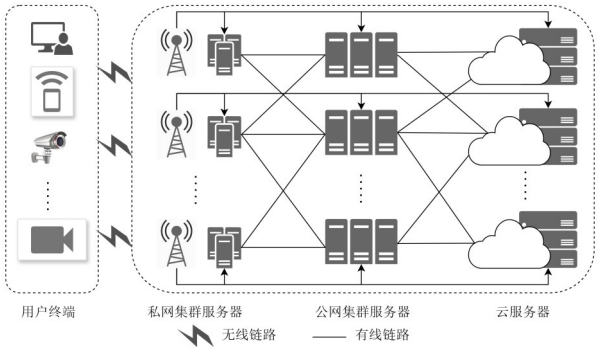


图1 多层次算力网络架构

该架构中的算力节点描述如下：

（1）云计算服务器用集合 $\mathrm{Cloud}=\{f,D,B\}$ 表示，f 为云计算资源量化值；D 为储存资源量化值；B 为云服务器带宽.

（2）私网集群内的算力服务器用集合 $N_{u,v}$ 表示，$N_{u,v}=\{f_{u,v},D_{u,v},B_{u,v}\}, u=1,2,\cdots,U, v=1,2,\cdots,V$，其中 u 为集群编号；v 为集群内服务器的编号；$f_{u,v}$ 为集群 u 内的算力服务器 v 的算力资源量化值；$D_{u,v}$ 为储存资源大小；$B_{u,v}$ 为算力路由带宽大小.

（3）核心公网中的服务器用集合 $E_e=\{f_e,D_e,B_e\}$ 表示；e 为服务器编号；f_e 为算力资源量化值；D_e 为储存资源大小；B_e 为带宽大小.（4）终端设备用集合 $M_m=\{f_m,D_m,B_m,P_m^{\mathrm{up}}\}, m=1,2,\cdots,M$ 表示，m 为终端设备的编号；f_m 为终端设备的算力资源量化值；D_m 为储存资源大小；B_m 为带宽大小；P_m^{up} 为移动设备上传数据的发射功率.

与普通边缘计算网络不同的是，算力网络能够完成跨区域跨集群的资源调度，同时利用任务传输路径上的算力节点进行计算，实现“边传边算”. 但需要同时考虑目标服务器的计算传输资源，与传输路径上的计算资源与通信状况. 本文设置在任务处理过程中，用户仅能选择一个边缘路由节点接入. 通过感知节点获取当前算力资源状态，并通过执行调度算法实现跨区域跨集群的算力资源调用. 由于感知节点通常需要收集大量的数据信息，所以通常令其承载较小的任务量.

3.2　算力感知模型

在进行任务处理的过程,需要获取算力设备的资源信息,然后进行资源调度. 感知节点通常设置在网络之中,计算节点中具有感知能力的节点即为感知节点. 令感知节点集合为 Φ,假设边缘服务器 E_e 具有感知能力,则感知节点 $\varphi_i=E_e, i=1,2,\cdots,\Phi$,$i$ 为感知节点标签. 感知节点的感知范围由网络运营商决定,当感知范围越广泛,网络内感知节点设置得越多,算力网络可调度资源也就越多,当用户利用到更多的网络资源时,对运营商的收益也会更有增益. 无法感知到的节点不能作为调度对象. 然而,随着感知节点的增加,感知节点需要通信交互的次数也就越多,带来的通信冗余、节点资源占用就会越大,因此本文考虑在较小的感知范围内进行感知节点部署.

3.3　算力任务模型

目前,大部分任务卸载模型将每个计算任务视为不可分割的独立单元,任一独立的计算任务只能选择在本地、边缘服务器或者云端进行处理,即二进制卸载[26]. 然而,对于大型计算任务,二进制卸载方式则会导致较大的处理时延,因此,近来有学者提出了非二进制卸载的任务部分卸载策略,即将任务细分为多个子任务,在终端、边缘端或者云端进行处理,降低处理时延. 本文假设在时间段 t 内,终端设备 m 有一个大型计算密集型任务 Task,该任务可拆分为多个子任务,且任务处理过程不中断. 任务 Task 可以用集合表示,其中,F_{task} 为处理任务所需的计算资源量化值;b_{task} 为任务大小,$T_{\max}$ 为用户任务允许的最大传输处理延时,$\text{Cost}_{\max}$ 为用户对完成任务所需要花费的预期最大值.

3.4　时延模型

3.4.1　通信时延

(1)无线通信

终端设备发出请求的用户业务通过无线链路传输到算力网络中的边缘无线接入点,根据香农定理可知,任务从终端 x 传输到接入网络上的任意计算节点 y 的数据传输速率如式(1)所示:

$$R_x^y=B_y\log_2\left(1+\frac{G_x^yP_x^y}{\sigma^2+\sum\limits_{\substack{j\in M\\ j\neq x}}G_j^yP_j^y}\right) \tag{1}$$

其中,$y\in\{N,E,\text{Cloud}\}$ 为除终端设备外的任意算力节点;G_x^y 表示计算路由 y 与终端 x 的信道增益;P_x^y 表示终端 x 到接入计算节点 y 的发射功率;σ^2 为高斯白噪声功率;$\sum\limits_{\substack{j\in M\\ j\neq x}}G_j^yP_j^y$ 表示接入计算节点 y 的其他终端设备对用户终端 x 产生的干扰总和. 则业务 Task 通过无线通信从终端 x 至算力节点 y 的传输时延为

$$t_x^y=\frac{b_{\text{task}}}{R_x^y} \tag{2}$$

(2)有线传输

算力节点之间采用动态的数据传输链路,假设动态链路的数据传输速率是可以感知的,则表示为

$$W=\begin{bmatrix}w_{1,1} & w_{1,2} & \cdots & w_{1,N}\\ w_{2,1} & w_{2,2} & \cdots & w_{2,N}\\ \vdots & \vdots & & \vdots\\ w_{N,1} & w_{N,2} & \cdots & w_{N,N}\end{bmatrix} \tag{3}$$

其中,$w_{i,j}$ 表示算力节点 i 与节点 j 之间的数据传输速率,$i,j\in\{\text{Cloud},N_{u,v},E_e,M_m\}$,并且满足:

$$w_{i,j}=\begin{cases}\infty, & i=j\\ w, & i\neq j,\ i\text{和}j\text{之间有链路连接}\\ 0, & i\neq j,\ i\text{和}j\text{之间没有链路连接}\end{cases} \tag{4}$$

其中,w 是一个有限实数值,则任务在算力节点 i 与节点 j 间的传输时延为

$$t_i^j=\frac{b_{i,j}}{w_{i,j}} \tag{5}$$

$b_{i,j}$ 为在节点 i 与节点 j 之间的传输任务大小. 随着任务经过每个节点,每个子任务被逐步卸载,卸载过程中,会形成一条卸载链路 $y_1,y_2,y_3,\cdots,y_l$,因此,业务 Task 到算力节点 y_l 进行处理的总传输时延如式(6)所示:

$$t=t_x^{y_1}+t_{y_1}^{y_l} \tag{6}$$

其中,y_l 表示编号为 l 的算力节点,本文设定 l 为最后一跳节点的编号;$t_x^{y_1}$ 表示任务从终端 x 到算力节点 y_1 上的传输时延;$t_{y_1}^{y_l}$ 表示任务从算力节点 y_1 到最后一跳节点 y_l 的传输时延. 定义 y_1 到 y_2 传输的数据量为 b_{y_1,y_2},y_{l-1} 到 y_l 传输的数据量为 b_{y_{l-1},y_l},$t_{y_1}^{y_l}$ 的计算公式为

$$t_{y_1}^{y_l}=\underbrace{\frac{b_{y_1,y_2}}{w_{y_1,y_2}}}_{①}+\underbrace{\frac{b_{y_2,y_3}}{w_{y_2,y_3}}}_{②}+\cdots+\underbrace{\frac{b_{y_{l-1},y_l}}{w_{y_{l-1},y_l}}}_{③} \tag{7}$$

①代表任务从第 1 个节点到第 2 个节点的传输时延;

②代表任务从第 2 个节点到第 3 个节点的传输时延;

③代表最后一部分任务在最后一段链路上的传输时延.

3.4.2　计算时延

为了更高效地处理任务,本文将其划分为更细粒度的子任务,任务 Task 的子任务集合为 $V=\{v_1,v_2,\cdots,v_k,\cdots,v_K\}$,其中 K 为子任务总数(由 ORAM 算法确定),某子任务 v_k 所占任务 Task 的百分比为 α_k,满足 $\alpha_1+\alpha_2+\cdots+\alpha_k+\cdots+\alpha_K=1$,则子任务 v_k 的计算时延为

$$t_c^k=\frac{\alpha_k F_{\text{task}}}{f_k} \tag{8}$$

其中，f_k为处理子任务v_k时节点提供的算力．本文通过测量对应节点的算力资源f来量化节点的计算能力，具体表示为每秒浮点运算次数（FLoating point Operations Per Second，FLOPS），f值越大，节点的处理能力越强，完成任务所需时间越短．

3.5 算力交易模型

在算力网络的付费模型中，通常采用按需付费、预付费、后付费等不同的模式，让用户可以更加灵活地选择和使用计算资源．

使用网络中的资源是需要付出一些金钱成本的，通常根据计算能力的不同，去衡量计算设备的租用价格．根据现有的运营商租用计算设备的策略，本文使用按用户使用设备时间付费的模型，这种模型基于用户对算力资源的实际使用时间来计费，通常以小时或秒为单位计费．用户根据他们使用的算力资源的时间长度支付相应的费用．

根据设备算力情况，本文设定了一个价值系数θ，这个系数可以根据不同服务器的性能、可靠性等因素进行设定，使得不同服务器的计费水平能够反映其相对价值．除了本地终端设备外，其他服务器的计费单价由这个价值系数调整，则其他服务器计费单价为

$$P_i=\theta f_{\text{node}_i} \tag{9}$$

其中，f_{node_i}为用户所需节点i的算力资源量化值，则当任务Task接入算力网络，并完成计算后，用户需付出的计算成本为

$$\zeta_{\text{cost}}=\sum_{k=1}^{K}p_k t_c^k \tag{10}$$

通过网络将任务传输到各个节点也需要向运营商付出网络传输成本，本文按照用户在互联网上的实际吞吐量来计费，则用户所需承担的网络成本可以定义为

$$\psi_{\text{cost}}=\sum_{k=1}^{K}b_{y_k}\psi_{\text{rate}} \tag{11}$$

其中，b_{y_k}为传到第k个节点y_k的任务大小；ψ_{rate}为网络计费的单价．因此用户使用网络的总成本为

$$\text{Cost}=\zeta_{\text{cost}}+\psi_{\text{cost}} \tag{12}$$

3.6 问题描述

为了满足短时间内的大型计算任务需求，我们提出了基于感知的算力网络模型．在这个模型中，每个节点具有不同的计算能力和带宽，节点间由于网络位置的不同其传输速度也不同．首先设置网络感知区域，如果任务接入的节点在感知范围内，该任务可以利用节点之间的信息协同计算．假设在感知范围内的计算节点总数为N，算力大脑根据任务接入以及接收位置，分析算力节点的分布情况，为用户选出可以处理任务的算力节点集合$P=\{\text{node}_1,\text{node}_2,\cdots,\text{node}_K\}$，以及划分在各个计算节点中子任务的比例$\alpha=\{\alpha_1,\alpha_2,\cdots,\alpha_K\}$，$K$为子任务总数，也是算力网络为任务提供协同计算的算力节点数量．则算力网络中任务处理的代价为处理过程产生的时延成本以及用户需为任务计算花费的付费成本加权和．

因此，本文将优化问题描述为，选择云边端异构算力网络中节点的集合P，并调整每个节点上的子任务分配比例$\alpha_k,k=1,2,\cdots,K$，最终获取的一种最佳的计算状态，使得时延T_d和付费成本Cost组成的总代价$\mathcal{G}$最小．

$$\begin{aligned}&\min_{\alpha,K,P}\{\mathcal{G}=T_d+\varepsilon\cdot\text{Cost}\}\\&\quad\text{C}_1:\text{Cost}\leqslant\text{Cost}_{\max}\\&\quad\text{C}_2:T_d\leqslant T_{\max}\\&\quad\text{C}_3:\alpha_1+\alpha_2+\cdots+\alpha_K=1\\&\quad\text{C}_4:0\leqslant\alpha_k\leqslant1\\&\quad\text{C}_5:1\leqslant K\leqslant N\\&\text{s.t.}\quad\text{C}_6:D_{y_k}-b_{y_k}\geqslant\overline{D}\\&\quad\text{C}_7:\vartheta\overline{f_{y_k}}\geqslant f_k\\&\quad\text{C}_8:\frac{\overline{f_{y_k}}}{f_{y_k}}\geqslant\tau\\&\quad\text{C}_9:\alpha_k b\in\mathbb{N},k=1,2,\cdots,K\end{aligned} \tag{13}$$

本文引入ε参数，作为时延和付费指标的平衡参数，用于归一化两不同维度的指标．不同的平衡参数也决定了任务处理的目的偏好．ε越小任务越偏向优化时延，称为时延敏感型任务，ε越大则任务越偏向优化付费，称之为付费敏感型．

（1）C_1表示任务总花费不高于用户最高付出成本$\text{Cost}_{\max}$；

（2）C_2表示任务总时延不高于最大忍受时延$T_{\max}$；

（3）C_3表示任务分割后的子任务比例和一定为1；

（4）C_4表示每个子任务比例的约束；

（5）C_5表示任务被分割后的子任务数量不能超过感知区域内所有节点的数量；

（6）C_6表示第k个节点在处理完目标子任务后仍剩余资源$\bar{D}$，当只处理单次任务时$\bar{D}=0$，D_{y_k}表示第k个节点的存储资源大小；

（7）C_7表示每个计算节点至多提供$\vartheta\overline{f_{y_k}}$的计算资源．$\overline{f_{y_k}}$为第$k$个节点的现有计算资源，$\vartheta$控制单个节点可供给子任务的资源上限，以避免计算资源被单个任务过度占用，从而确保后续任务仍能获得计算资源；

（8）C_8表示节点的剩余计算资源比例低于阈值τ时，该节点将不再参与任务分配，以防止计算资源过度集中于某些节点而导致资源枯竭．阈值τ可根据任务需求进行设定；

（9）C_9表示任务划分是以比特为单位，其不可再

分,因此是非负整数.

4　算法设计

4.1　串行子任务路径分配机制

本文首先提出串行子任务路径分配机制(如图2所示),通过在网络中搜索空闲的计算设备,建立一条连接任务发送终端和结果接收端的计算通道,使通道中串行节点快速转发数据和计算结果. 控制器将任务拆分成多个子任务,并分配控制信令到路径通道里的算力设备,组织通道形成和协同传输计算.

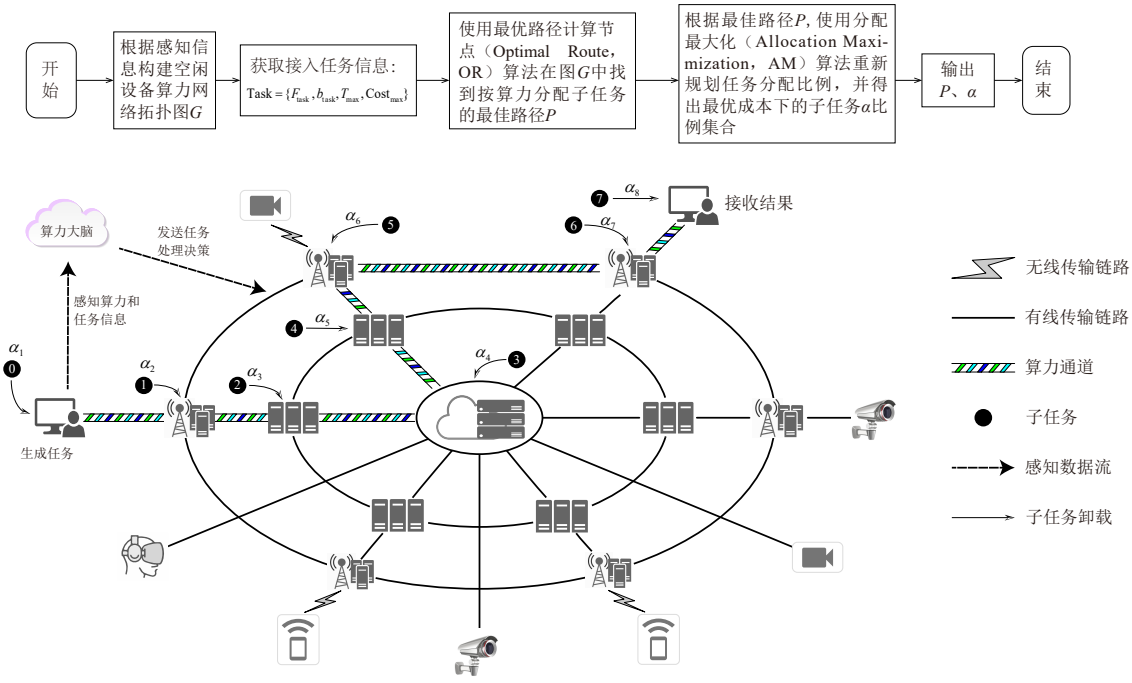


图2　ORAM算法流程图和串行子任务分配机制

当一个子任务在本地开始处理时,其他子任务则开始被分配到算力通道中的其他计算设备上进行处理. 对于计算量较大的任务,即使任务接收点处于网络中的临近位置,但在两跳卸载中仍无法满足业务需求,则任务将通过第二跳的计算节点继续卸载至第三跳,直到可以满足任务处理需求为止. 故算力通道中计算节点的选择,子任务分割的比例都将是影响任务处理结果的关键影响因素.

综上所述,当任务通过终端x接入网络,在算力通道节点集合P中传输并计算完成,根据式(7)和式(8)$b_{y_1,y_2}=(1-\alpha_1)b_{\text{task}}, b_{y_{l-1},y_l}=(1-\alpha_1-\cdots-\alpha_{l-1})b_{\text{task}}$,此时第$k$个子任务$v_k$的处理时延为

$$t_k=t_x^y+t_c^k=\begin{cases}\dfrac{a_1F_{\text{task}}}{f_x}, & k=1\\ \left(\dfrac{(1-\alpha_1)b_{\text{task}}}{w_x^{y_1}}+\dfrac{(1-\alpha_1-\alpha_2)b_{\text{task}}}{w_{y_1}^{y_2}}+\cdots\right.\\ \left.+\dfrac{(1-\alpha_1-\cdots-\alpha_{k-1})b_{\text{task}}}{w_{y_{l-1}}^{y_l}}\right)+\dfrac{a_kF_{\text{task}}}{f_{y_l}}, & 2\leqslant k\leqslant K\end{cases}\tag{14}$$

其中,x为任务接入的终端节点;l为第k个子任务v_k传输到的最后一个节点标签,$l=k-1,k=1,2,\cdots,K$. 由于任务是可以并行的,因此任务Task的总时延即为所有子任务当中最大的子任务总时延. 则大型计算任务Task总时延为

$$T_d=\text{Max}\{t_k\},k=1,2,\cdots,K\tag{15}$$

此时该任务的网络传输付费成为

$$\psi_{\text{cost}}=\begin{cases}0, & K=1\\ \left[\sum_{k=2}^{K}\left(1-\sum_{i=1}^{k-1}\alpha_i\right)\right]b_{\text{task}}\psi_{\text{rate}}, & K\geqslant 2\end{cases}\tag{16}$$

由此可见,对于串行子任务路径分配机制,任务时延及计算成本受到计算节点算力的影响,并且随着子任务所在节点的跳数的变化而变化. 因此,在进行任务拆分和分配节点时需要考虑节点的算力和传输时延等因素,以达到更高效的任务处理效果.

4.2　最优路径的最大化分配算法

路径和任务比例切割问题相互耦合,需要将问题解耦,采用简单高效的算法分别求解各子问题,提高整体求解效率. 因此本文提出了一个名为最优路径的最大化分配算法(ORAM),将问题描述中的求解最佳计算

路径状态，解耦成两个子问题：求解最佳计算路径和求解最佳的计算状态. ORAM的算法流程图如图2所示.

为了解决两个子问题，本文利用图论首先将感知后的算力网络模型抽象为一个无向图G，计算设备表示为图G的节点，设备的算力资源为节点的属性，设备间的有线无线链路表示为图G的边，链路间的传输速率为边的属性. 将任务Task发起和接收点设置为起点N_s与终点N_e，如算法1所示，本文使用最优路径计算节点(Optimal Route，OR)算法在图G中寻找最优目标的路径P和子任务总量K. 再以最佳路径P为分配最大化(Allocation Maximization，AM)算法输入，使用AM算法对路径分割进行优化，获取最小目标的比例分割α. P和α即为ORAM算法所求结果.

算法1 最优路径的最大化分配算法(ORAM)

输入：Task, N_s, N_e

输出：最佳路径P、子任务总量K和最优的比例分割α

1. $K, P \leftarrow OR(N_s, N_e, \text{Task})$

2. $\alpha \leftarrow \text{AM}(P)$

具体实现如下，本文采用OR算法(算法2)所示，赋予初始搜索最佳路径的条件，即按照节点算力能力去分配任务. 此时搜索出的路径对于整个优化问题，并不一定是全局最优解，但由于传输时延在整个算力网络处理大型计算任务时延中所占的比重并不是非常的大，因此本文选择牺牲部分传输时延，在按能力分配策略下搜索最小目标路径P为OR算法的最优解，也是整个问题的优解. 首先在步骤1~步骤4对环境进行初始化，步骤4~步骤12采用迭代的方式对路径进行搜索，直到搜索完所有路径. 在步骤14~步骤18中，对搜索到的可用路径进行筛选，筛除不符合用户需求的路径，并保留最小目标值OBJ^*的路径P. 最后根据得出的最小目标路径P，确定子任务总量K. 尽管这种方法可能会导致一些传输时延的增加，但整体上能够保证任务能够在整个算力网络中得到高效处理.

在得到了一条最小目标路径P之后，将P带入AM算法(算法3)中. 如步骤1，初始任务比例集合α为按照路径P中节点的算力的比例分配，并根据此时任务分配和路径节点情况获取此时任务总成本OBJ，以及初始化最小目标OBJ^*. 随后，通过步骤4~步骤13的迭代循环，对路径P的任务分配进行调整，可以弥补上一步策略的牺牲传输时延带来的问题. 其中，步骤8使用的Metropolis准则其核心思想是在算法的早期迭代中，以较高的概率接受性能较差的解，以促进解空间的广泛探索，避免算法过早地陷入局部最优解. 随着算法的进行，接受阈值逐渐降低，接受更差解的概率也随之降低，这有助于算法在解空间中进行更精细的搜索，并逐渐稳定在接近全局最优解的区域. 通过步骤4不断调整比例，平衡计算时延和传输时延，再经过步骤6~步骤10根据任务需求和路径节点情况进行的筛选，进一步优化整体的目标.

算法2 最优路径计算节点(OR,Optimal Route)**算法**

输入：Task, N_s, N_e

输出：最佳路径P,子任务总量K

1. 初始化：G,P为空集,OBJ^*为无穷大值
2. 获取Task需求$T_{\max}$, $\text{Cost}_{\max}$
3. 定义路径栈Stack,栈的第一个元素为路径[N_s]
4. 当路径栈Stack不为空时：
5. 获取并移除栈顶路径p
6. 获取p的最后元素e
7. 如果$e \neq N_e$：
8. 获取e在G中的相邻节点集合S
9. 对于每个在S中的元素s：
10. 如果s不在p中：
11. 创建一个新路径$p^* = p + s$
12. 将p^*压入栈Stack中
13. 否则：
14. 根据式(15)计算当前路径p时延T_d,根据式(12)计算付费Cost
15. 如果$T_d < T_{\max}$并且$\text{Cost} < \text{Cost}_{\max}$：
16. $\text{OBJ} \leftarrow T_d + \varepsilon \cdot \text{Cost}$
17. 如果$\text{OBJ} < \text{OBJ}^*$：
18. 更新$\text{OBJ}^* \leftarrow \text{OBJ}, P \leftarrow p$
19. $K \leftarrow \text{len}(P)$
20. 返回K, P

算法3 分配最大化(Allocation Maximization,AM)**算法**

输入：P

输出：最优的比例分割α

1. 初始化：$\alpha = \left[F_1, F_2, \cdots, F_K\right] / F_P$,$\alpha^*$为空集,$\text{OBJ}^*$为基于$\alpha$计算目标值
2. 获取Task需求$T_{\max}$, $\text{Cost}_{\max}$
3. 定义：迭代次数I,差值接受率r
4. for迭代次数$i = 1, 2, \cdots, I$,执行：
5. 随机选取α与对α进行归一化处理
6. 根据α和P计算根据式(15)计算当前路径时延T_d,根据式(12)计算付费Cost
7. 如果$T_d < T_{\max}$并且$\text{Cost} < \text{Cost}_{\max}$：
8. 如果根据 Metropolis 准则满足一定的差值接受率r条件：
9. $\text{OBJ} \leftarrow T_d + \varepsilon \cdot \text{Cost}$
10. 如果$\text{OBJ} < \text{OBJ}^*$：
11. 更新$\text{OBJ}^* \leftarrow \text{OBJ}, \alpha^* \leftarrow \alpha$
12. 否则：
13. 更新r
14. 返回α^*

4.3　复杂度分析

假设感知范围内算力节点的数量为N,感知后的有线传输路径数量为E,根据算法2步骤10可知,每个节点都只会被访问一次,对于每个节点而言,我们需要访问其所有的邻居节点,而由于图是无向的,因此每条边都会被访问两次,因此OR算法复杂度为$O(N\cdot E)$. 对于AM算法,根据算法3步骤4~步骤13可知,仅有外部一个I次的迭代,因此算法复杂度为$O(I)$. 综上,本文所提算法ORAM的复杂度为$O(N\cdot E+I)$.

5　实验与结果分析

5.1　实验设置

本文假设存在这样一个多层次的算力网络,参数设置如表1所示.

表1　实验参数设置

参数	取值
各层网络节点数	{1,18,5,6,5}
任务所需计算资源量化值F_{task}/TFLOPs	[1 000,6 000]
任务大小b_{task}/GB	[0.5,2]
任务最大允许时延T_{max}/s	60
用户最大付出成本$Cost_{max}$/RMB	[30,60]
云计算算力f/TFLOPS	120
云计算储存容量D/PB	4
核心边缘节点算力f_e/TFLOPS	[75,100]
核心边缘储存容量D_e/TB	[1,10]
私网集群节点算力f_{ij}/TFLOPS	[10,75]
私网集群节点储存容量D_{ij}/TB	[1,10]
有线传输速率R/Gbps	[1,10]
无线传输信噪比σ^2/dBm	−114
终端设备传输功率P_m^{up}/mW	[60,200]
信道增益G	10^{-7}
终端设备带宽B_m/MHz	[20,200]
价值系数θ/RMB/TFLOPS	0.01
网络计费的单价ψ_{rate}/RMB/GB	0.29
节点资源分配系数ϑ	[0,1]
节点最低资源可接受阈值系数τ	[0,0.05]

该网络为4层算力网络,如图3所示:第1层有1个云计算节点;第2层为核心网,有18个边缘计算节点;第3层有3个私网边缘集群,集群内分别有5、6、5个边缘计算节点;第4层由终端组成. 在图中,以不同的颜色对节点层级加以区分. 其中感知节点分布于公网与核心网中,通过信息交互,形成了新的基于部分感知的算力网络场景,见图4. 本文研究单任务计算场景,当任务接入算力网络时,通常任务接入点位于公网之内,此时接入点接收用户请求信息,假设接入点和接收点位于感知区域内,接入点接收到用户请求后,会将任务需求传达给算力大脑. 算力大脑会使用本文设计的ORAM策略为用户规划出一条连接接入点至接收点的一条最佳计算路径,并为用户设计任务分割策略. 以下为本方法的结果展示以及本方法与其他三种基础方法的对比结果.

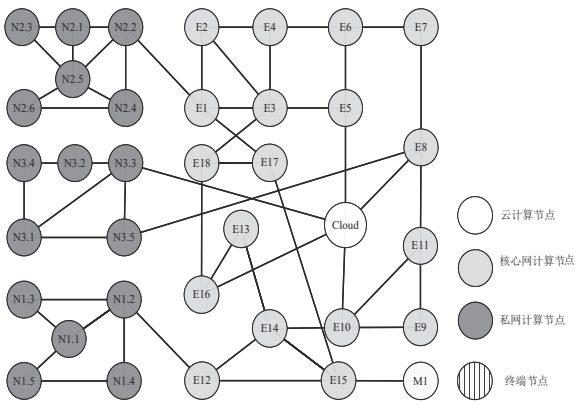


图3　原算力网络拓扑图

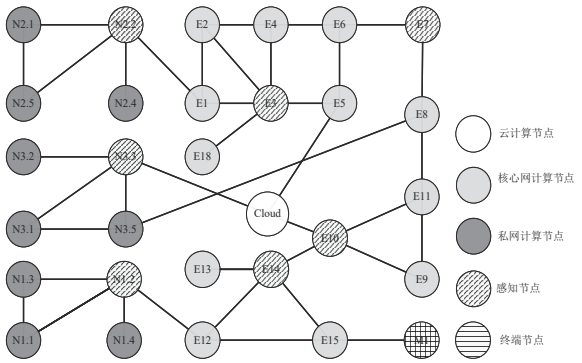


图4　感知后的算力网络拓扑图

5.2　任务分割与调度

本文提出了一种任务最佳分割的算力网络计算策略,在任务发出和计算的过程中,选择以按能力分割任务方式确定最佳路径,并在此基础上对各个节点的分配比例进行重新优化,边传输边计算. 充分利用、高效调动网内算力资源,以满足用户低时延、低预算的需求.

在相同的网络条件下,图5和图6分别展示了不同计算需求的任务计算和分配路径,例如加粗的箭头从M1终端指向E15意味着M1保留了本身需处理的部分,剩余计算任务传输至E15. 图7为不同计算需求任务在算力网络中使用的节点情况,即任务在各节点被分配的百分比. 显而易见的是,当计算需求不同时,任务的路径和分配比例都有所差异,且处理高计算需求任务比低计算需求任务使用的节点数量要多. 这是由于,在任务发出端和接收端相同的情况下,随着被处理任务计算

量的增加,更大的任务需要更多的计算资源来完成.同时,为了更好地调整分配比例以及减轻单个节点的计算压力,路径算法不仅能筛选合适的路径也能对节点数量进行控制,为后续的分配算法提供更佳的空间.

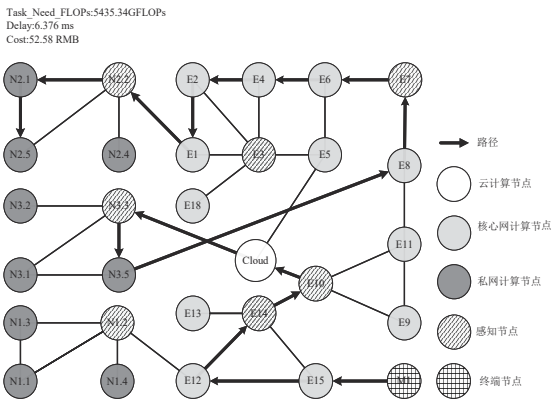


图5 低需求下路径求解结果

由于用户对任务的时延和预算需求各不相同,本文通过改变系数ε来实现不同用户需求下的路径选择和比例分割.图8和图9分别为时延和付费敏感型的任务处理结果,ε取值分别为0.008 3和0.417,当任务偏向时延敏感时,路径中节点数为19,任务处理时延较付费敏感型任务降低了3.63%.当任务偏向付费敏感时,路径长度为17节点,付费成本较时延敏感任务有所降低.可见,本文提出的方法根据感知网络内的算力资源情况,适当增加计算节点或调整任务分配比例,选择合适的节点来降低设备租用以及网络传输的成本,或以此获取更优的总时延.

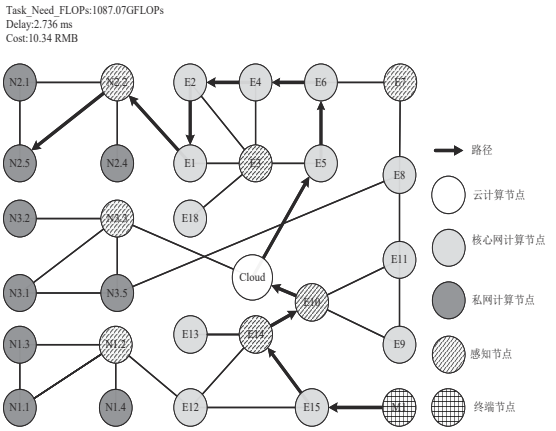


图6 高需求下路径求解结果

综上所述,在对待不同计算资源与需求敏感的任务,本文采用的算法可以灵活调用网络内的资源,通过调整路径、任务分配的比例以满足任务需求.

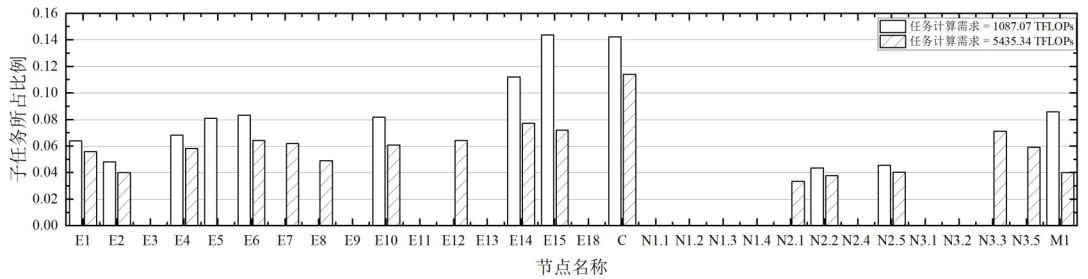
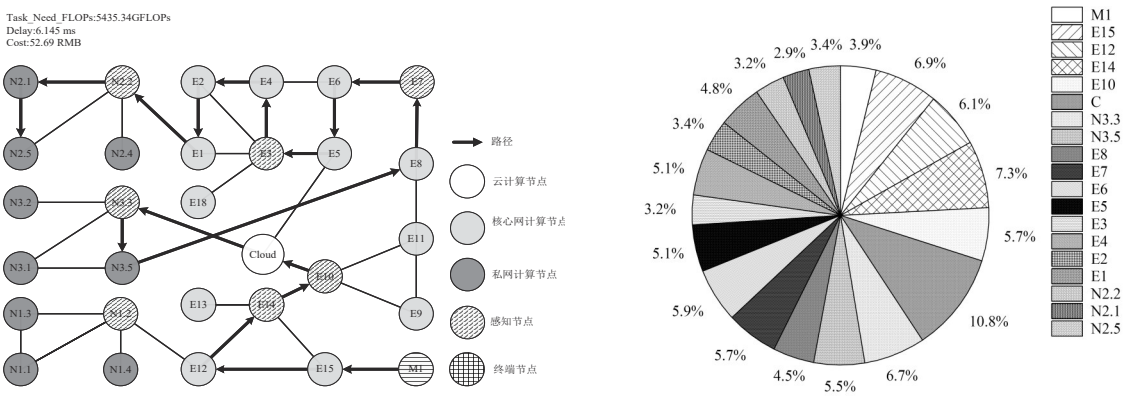
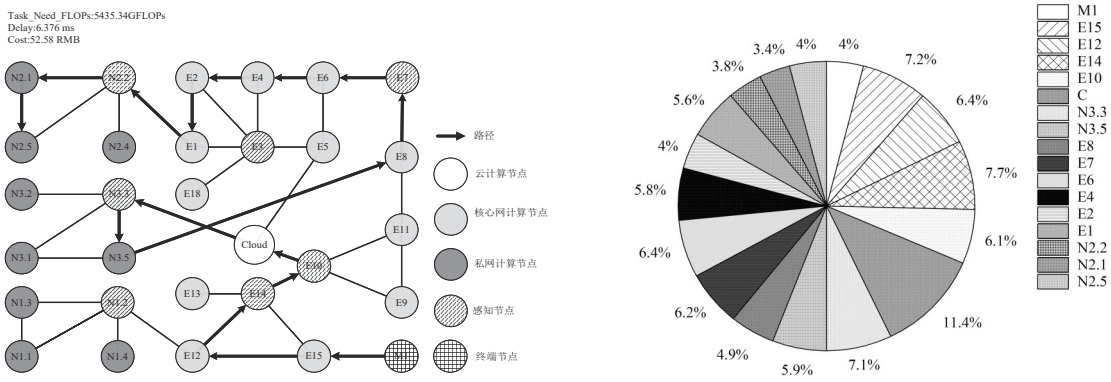


图7 两种需求下的任务分割结果



(*a*) 时延敏感型任务处理路径 (*b*) 时延敏感型任务的任务切割状态

图8 时延敏感型任务处理结果



(*a*) 付费敏感型任务处理路径　　(*b*) 付费敏感型任务的任务切割状态

图9　付费敏感型任务处理结果

5.3　不同参数下算法的性能对比

对于算力需求高的任务,任务分配比例的选择则变得更为重要. 为了获取算法最佳参数搭配,本文对比了不同参数的性能,如图10、图11所示.

如果迭代次数(iterations)较少,算法可能无法充分搜索解空间,从而无法找到较优的解. 结果可能接近初始解,可能不够稳定,也可能无法满足问题的要求. 在一定程度上,增加迭代次数有助于算法更好地搜索解空间,提高解的质量和稳定性. 然而,过多的迭代次数可能导致算法运行时间过长,不适用于实时性要求较高的问题. 初始阶段采用较高的差值接受率($\text{acceptance}_{\text{rate}}$)可以使算法更容易跳出局部最优解,并在解空间中广泛探索,从而有机会找到全局最优解. 当算法进行到后期阶段时,逐渐降低差值接受率有助于算法在局部最优解附近细致搜索,提高最终解的精度. 然而,如果差值接受率降得太快,可能导致算法陷入局部最优解,无法跳出.

由此可知,选择合适的迭代次数和差值接受率对算法的表现十分重要.因此,本文对接受率为0.99和0.95、迭代次数为1 000和10 000次的性能进行对比. 图10、图11展示了四种不同组合算法参数在不同条件下的表现.

由图10可知,当任务需求为1 087.07、5 435.34 GFLOPs时,4种优化目标值分别增长257.62%、257.64%、255.17%、256.97%,第三种组合增长最少,且第三种参数组合在任务需求变化下始终保持着最优的表现,其目标函数优化结果始终为最低. 因此,组合三优势更加明显.

在对于同计算需求的任务,当任务趋向时延敏感或趋向付费敏感时,图11显示,第三种参数组合的优化结果其平均值(6.90)相对于其他组合(组合1:6.99,组合2:6.93,组合4:6.94)为最低,因此我们将算法的参数设定为$\text{acceptance}_{\text{rate}}=0.99$和iterations = 10 000.

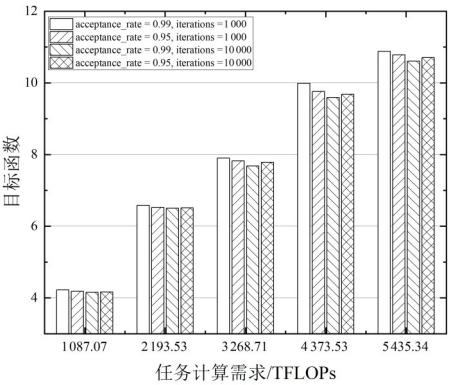


图10　不同参数组合在不同计算需求任务上的表现

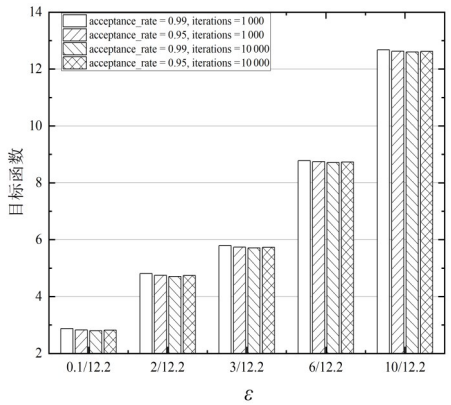


图11　不同参数组合在不同优化需求任务上的表现

5.4　不同任务计算需求下结果对比

本文通过对比枚举、随机选路和能力最大化分配的任务处理策略,对不同计算需求任务进行了研究.

(1)枚举算法(Exhaustive Optimality):搜索全图可用路径,并对每条可用路径使用优化任务分配比例分配的算法以获取最佳任务处理结果.

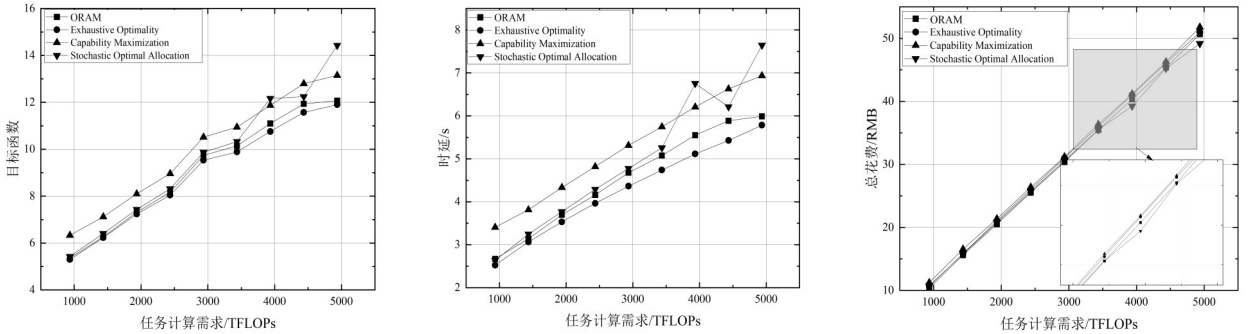
(2)随机最优切割算法(Stochastic Optimal Allocation):使用随机算法选择可用路径,并优化任务分配比

例处理任务计算.

(3)能力最大化分配算法(Capability Maximization):直接对所有可用路径以路径中所有节点算力的能力分配任务方式,获取最佳计算结果.

图12展示了ORAM算法与其他三种基准算法在不同计算需求任务下的性能对比.为了确保实验的有效性,我们将任务其他指标均设置为统一的状态.

从图12的结果可以看出,除了随机选路算法外,其他算法在三种指标下都表现出近似线性的变化趋势.能力最大化算法在所有情况下表现最差.随机选路算法在初始阶段表现接近最优值,但随着任务量的增大,性能表现变得不稳定.与其他策略相比,本文采用的ORAM算法在逼近枚举方法的目标结果方面表现最好,且其表现稳定性较高.



(a) 目标函数对比 (b) 时延对比 (c) 付费对比

图12 不同计算需求下四种处理策略指标对比

具体来说,通过图12(a)所示的总目标函数指标进行结果分析,ORAM算法的优化的最小目标函数值平均值为9.12,而枚举算法为8.94,能力最大化算法为9.98,随机最优切割算法为9.62.可见,本文提出的方法在优化总目标方面表现最接近枚举方法,且优于其他两种算法.此外,通过图12(b)和图12(c)的时延指标和付费指标对比可见,在时延方面ORAM算法的平均值为4.54 s,较能力最大化算法和随机最优切割算法分别提升了23.57%、8.28%.由于ORAM算法采用了一种牺牲部分时延指标、降低付费成本的方式,选择更合适,而非算力最大的节点,因此付费成本比枚举方法更低.

综合对比三种指标的结果,可以得出结论:本文提出的ORAM算法相较于其他方法在性能上更接近最优解(即枚举方法的结果).ORAM算法能够在计算需求变化的情况下有效地优化任务分配且使用户体验更好的服务同时付费最少.

5.5 同任务不同优化目标偏向下结果对比

本节对同条件下,不同用户优化需求下的四种算法表现进行对比.

由图13可知,对于相同的任务,由于实验结果保留了随机算法的随机性,当平衡系数ε越大时,除了随机最优切割算法其他算法目标函数均呈类似线性增长趋势.

其中时延随着平衡系数ε的减小而下降,ORAM算法与枚举算法下降得更加显著.付费随着平衡系数ε的增加而降低,本文算法与枚举算法下降更加稳定和显著.且ORAM算法目的函数值总是最接近枚举算法的结果.这是由目标函数特性决定的,ε越小,任务趋向时间敏感,则算法在处理任务时更侧重降低时延.相反,越大时,算法更偏向于通过降低付费来达到目的.

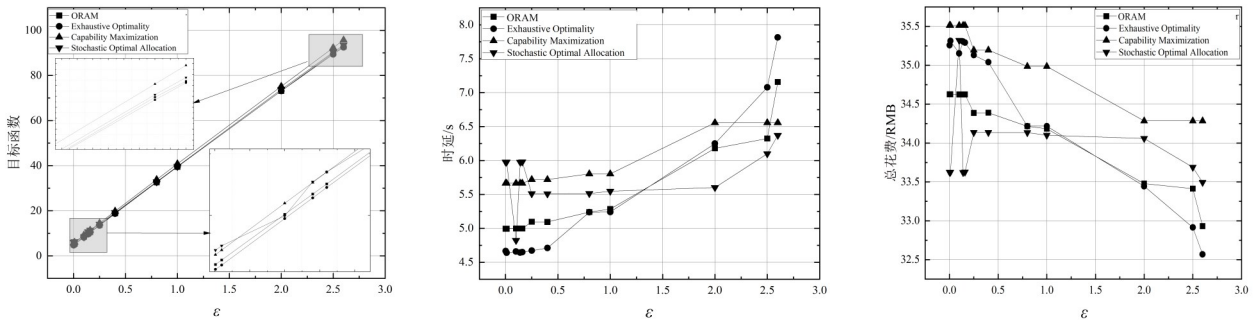
据图13(b)的结果显示,当ε为0.01时,此时任务优化目标更偏向于时延,枚举算法时延为4.62 s,ORAM算法为5.04 s,随机算法为5.96 s,能力最大化分配算法为5.73 s.同样,从图13(c)可见,当ε为2.6时,本文算法的付费为32.93 RMB,枚举算法结果为32.63 RMB,随机最优切割算法为33.75 RMB,能力最大化分配算法为34.32 RMB.可见ORAM算法在不同优化偏向下均表现优异,且接近于最优结果.

这证明了,本文采用的策略相对于其他策略接近最优,可以满足不同优化需求的变化,且我们的策略变化规律近似于枚举策略.

5.6 不同感知范围下结果对比

感知范围的大小,影响了任务利用算力网络可以使用的计算资源量.计算资源的可调动性在一定程度上影响着任务处理的选择.

随着感知域增大,此时路径的选择变多,算力网络可为处理任务调度的资源增多,可优化的路径选择更多.在任务优化偏向时延敏感时,一些可以提供较大算力的节点会更容易被选择,以达到降低时延的目的,而这些节点的收费往往也更高.如图14(b)和图14(c)可见,时延会随着感知范围的增加先降低然后趋向平衡,



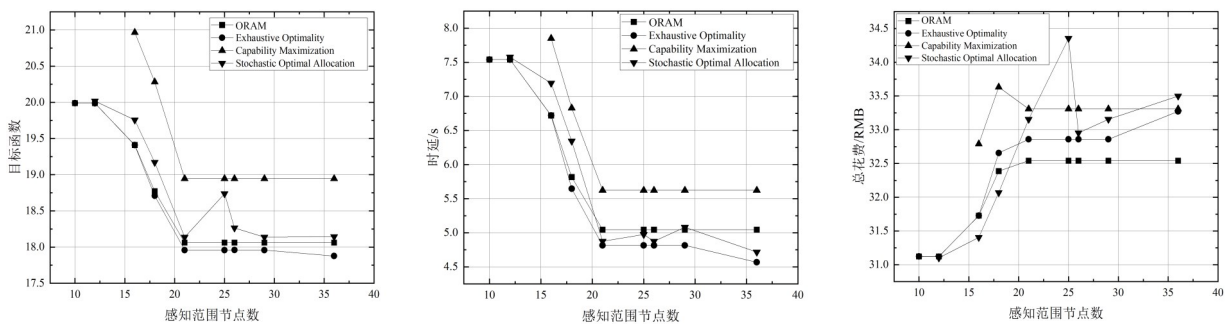
(*a*) 目标函数对比　　(*b*) 时延对比　　(*c*) 付费对比

图 13　不同目标偏向下四种处理策略指标对比

而付费指标则相反.

图 14 中,在感知区域覆盖 21 节点后,ORAM 算法和能力最大化算法结果趋于稳定,而随机算法在 25 节点和 26 节点情况下目标函数值突增,穷举算法在 36 节点时目标函数值略有减少. 因为当感知域扩充到 21 节点时,此时网络中算力已经接近饱和,最佳路径的选择也接近最优,再扩大感知范围对最优路径的选择不会产生较大的影响,而随机最优切割算法因其随机性导致性能不佳.

由此可见,本文提出的 ORAM 算法在各指标表现上都是符合预期,较为稳定,优异且接近最优的. 感知域的增大会在一定范围内影响着算法的表现. 故在实际部署感知节点时,我们可以做出推导:感知范围的设定也可以根据任务路径而适度调整,合适的部分感知范围不但可以达到很高的任务处理效率,也可以避免全感知情况下,给网络造成的通信冗余.



(*a*) 目标函数对比　　(*b*) 时延对比　　(*c*) 付费对比图

图 14　感知范围不同下四种处理策略指标对比

5.7　算法时间复杂度

图 15 展示了在感知范围内有 21 个计算节点的条件下四种算法的时间复杂度,可以看到枚举策略的时间复杂度要远高于其他三种策略,其中能力最大化算法最低,ORAM 算法略高于能力最大化算法,且低于随机最优切割算法.

结合以上几节的分析可知,枚举算法优化结果总是最优的,ORAM 算法仅次于枚举算法. 但枚举策略时间复杂度过高,在考虑实际部署时,算法处理任务时间对任务需求的影响,本文所提出的 ORAM 算法会是比枚举算法更优的选择.

能力最大化算法虽然时间复杂度最低,但根据上面的结果分析可知,其优化结果表现最差. 随机最优切割算法,虽然与 ORAM 算法时间复杂度接近,但其优化

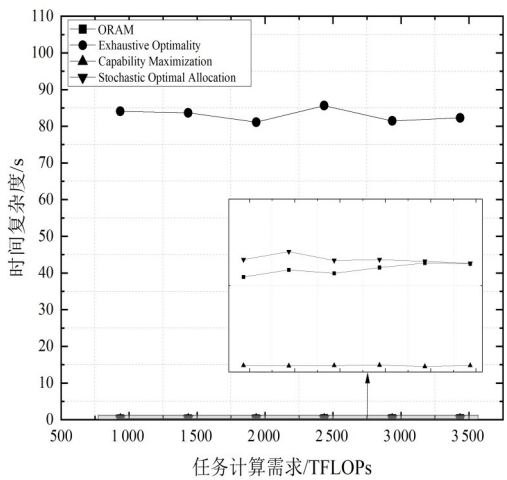


图 15　时间复杂度对比

表现十分不稳定. 因此,在任务调度算法选择上,本文策略相较能力最大化算法和随机最优切割算法也是更有优势. 综上可知,在对比了其他三种基准算法后,可以证明本文采用的ORAM是最优的.

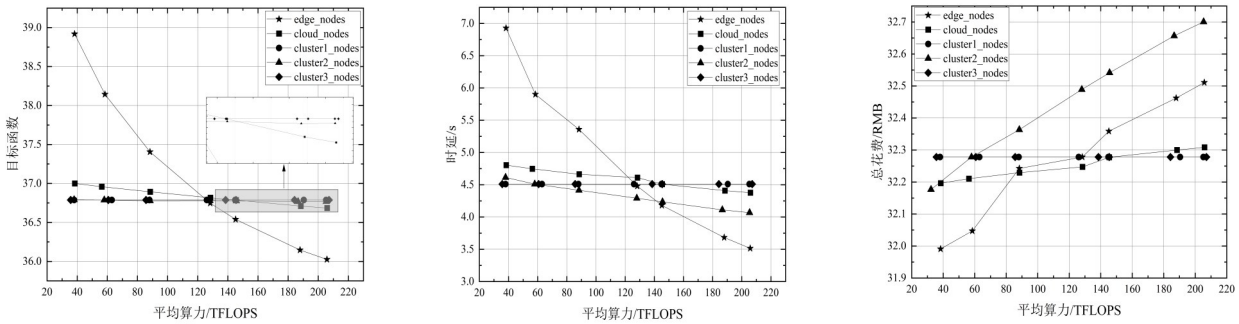
5.8 多层算力网络的各层算力改变对指标的影响

由于算力网络的异构性,分布在算力网络中的不同位置的计算节点算力对任务处理的结果也会有一定的影响. 因此,如图16所示,本文以ORAM算法为任务处理优化的算法,对比了处于网络不同位置的几种计算节点平均算力对任务处理的影响.

从图16中可以看出,随着核心网边缘节点、云节点以及私网计算集群2的平均算力的增强,任务处理都得到了提升,其中核心网边缘节点对任务处理的提升最快. 私网计算集群1、3算力变化对任务处理没有影响. 由图16(*b*)和(*c*)可知,随着核心边缘平均算力增强,任务时延优化了近49.27%,而付费指标仅上升约1.63%,任务目标函数值降低7.43%. 当云节点平均算力增强,任务时延优化8.89%,付费上升0.35%,任务目标仅降低0.85%. 私网集群2平均算力增高,任务时延降低11.83%,目标函数值仅降低0.06%,而付费上升1.63%.

这是由于边缘节点分布于核心网内,位于网络的核心位置,当任务传入网络,核心网承担着大部分的计算任务. 核心网的边缘节点计算能力增强对任务处理的效果提升具有明显的影响. 对于云节点,云节点的算力通常较其他节点要强,提升算力虽然对任务处理有着有益的影响,但在我们实验的算力网络中云节点的数量相对于核心网的边缘节点少很多,故仅提升云节点算力,在协同计算中,对任务处理的影响较核心边缘节点会小很多. 算力网络中,处于感知内的私网边缘节点也可以用做计算,如图16所示,在本文测试任务场景下,用户任务目的接收节点位于私网2内,而由于本文采用的最优选路策略结果不会经过其他感知域的私网,因此只有私网2的算力增加会影响任务处理结果. 因此,当私网计算集群需要承担部分计算任务时,增强私网算力也是可以提升任务处理的效率.

所以在实际部署中,部署者可以根据任务的计算规律,筛选不同网络位置的计算节点对其算力加以提升. 其中提高核心网的计算能力是对任务处理最有效的提升.



(*a*) 目标函数对比 (*b*) 时延对比 (*c*) 付费对比

图16 平均算力不同下四种处理策略指标对比

5.9 连续任务请求情况下结果对比

在本节中,假设单位时间内的任务请求数量遵循泊松分布(Poisson Distribution). 具体而言,任务请求的到达率由泊松分布的期望λ值来决定,此处λ表示单位时间内的平均任务请求数量,即平均每$1/\lambda$ s到达一个任务请求. 例如,当$\lambda=0.2$时,表示每5 s到达一个任务请求. 此外,每个任务请求所需的计算需求在[1 000, 6 000]范围内均匀分布,以确保任务之间的计算需求具备多样性.

图17展示了在不同λ值下,任务偏向时延敏感的情境中,四种算法在处理任务时的平均时延. 由于任务是连续到达的,如果某个任务到达时前方已有任务正在占用资源,那么该任务需要排队等待. 因此,在连续任务请求的情况下,时延不仅包括原有的计算和传输时延,还增加了排队等待时延. 从图中我们可以看出,ORAM算法在平均时延指标上的表现接近枚举算法. $\lambda=1$时,ORAM算法的平均时延相较于能力最大化算法和随机最优切割算法分别提升了13.87%和12.69%.

图18展示在不同λ值下,任务偏向付费敏感的情境中,四种算法在总花费指标上的表现. 随着λ值的增大,单位时间内达到任务数量变多,各类算法间的总花费差距也会慢慢变大. ORAM算法在总花费指标上的表现上同样接近枚举算法,在λ等于0.2到1.0的范围内平均较最大化算法和随机最优切割算法提升了21.58%和22.86%.

通过对上述两项指标的对比分析,可以得出结论:本文提出的ORAM算法不仅在单任务请求情况下表现优异,而且在常规的连续任务请求情境下,也能接近枚举算法的性能. 在连续任务请求的实际场景中,ORAM

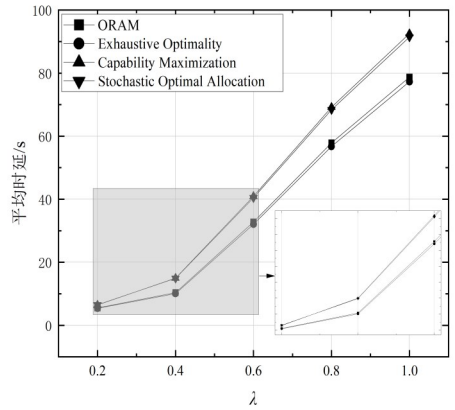


图17　不同任务到达频率下四种策略的任务平均时延

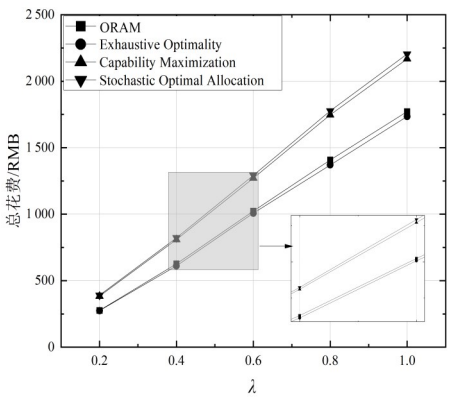


图18　不同任务达到频率下四种策略的总花费

算法将成为一个更为优选的方案.

此外,本实验假设连续任务请求之间的到达时间间隔是随机生成的,且其平均到达率由参数λ控制. 任务的到达时间间隔是随机的,但在长期观察下,任务到达的频率接近λ的倒数. 当任务到达时,如果前方已有任务尚未处理,则该任务需要排队等待,从而模拟连续任务请求调度中任务的排队过程. 图19展示了不同λ值下,任务到达后的平均排队时延. 随着λ值的增大,单位时间内到达的任务数量增多,导致网络负载加重,从而引发排队时延的增加,网络呈现出拥堵趋势. 尽管如此,ORAM算法在此情境下依然表现良好,ORAM算法能根据实际情况,选择合适节点,并分配适当比例的任务,可以一定程度上减少网络中节点排队等待情况的发生,减轻网络负担. 在λ取0.2到1范围内的5个值时,每个任务的平均排队时延达到1.525 s、6.641 s、29.139 s、54.299 s、75.129 s,接近枚举算法的1.484 s、6.402 s、28.443 s、53.219 s、73.738 s,并且相较于其他算法,比如能力最大化算法分别提升23.53%、38.69%、20.83%、16.67%、15.10%. 这表明,ORAM 算法在实际的连续任务请求场景中,能够一定程度缓解高负载下的任务调度问题,并保持较优的性能表现.

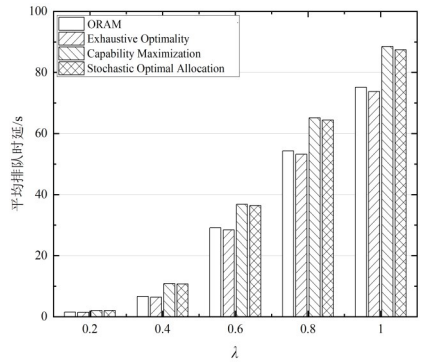


图19　不同任务到达频率下每个任务的平均排队时延对比

5.10　任务并行处理性能分析

在实际算力网络中,计算节点通常具备多核架构,能够并行处理多个任务,其并行处理能力主要取决于节点的计算资源总量以及各任务的计算需求. 此前的实验主要关注单任务串行执行的情况,即连续几个任务依次占用计算资源,而本实验进一步对比不同任务调度算法在多任务并行处理环境下的性能表现. 通过调整计算资源的分配策略,我们分析了四种不同任务调度算法在相同条件下的任务完成总时延,探讨不同算法在计算资源受限场景下的优化效果,以评估其对整体计算效率的影响.

5.10.1　计算资源对任务执行时延的影响

为了研究计算资源对任务执行时延的影响,我们修改了各类计算节点的平均计算能力,并观察任务完成时延的变化趋势. 实验结果如图20所示.

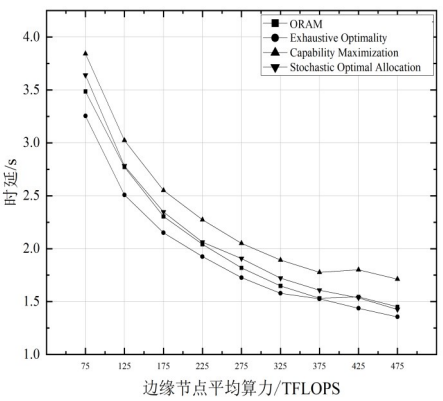


图20　不同计算资源大小下任务处理时延情况

当计算节点的平均算力较低时,任务处理时间较长,子任务在各个计算节点上的计算时延较大,导致整体任务完成时延高. 随着计算节点算力的提升,子任务能够更快地被计算节点处理,因此任务完成时延逐渐下降,呈现非线性递减趋势. 当计算节点算力达到一定阈值后,任务完成时延的下降幅度趋于平缓,表明在算

力充足的环境下,任务时延的主要影响因素开始从计算能力转换为通信时延、任务调度延迟等因素.进一步提升计算资源的边际收益下降,表明在固定计算任务下,算力优化存在饱和点,仅依靠提升计算资源难以显著降低任务总时延.因此,在高算力环境下,更高效的任务调度和资源分配策略将成为优化任务完成时间的关键.

5.10.2 资源分配系数对多任务并行处理的影响

在多任务并行计算场景下,控制器能够根据剩余计算资源动态分配节点算力.为了进一步研究资源分配大小对任务总时延的影响,本文引入资源分配系数ϑ,即计算节点在每个任务到达时,能够分配的最大计算资源占比.实验中,预先设置合适计算资源,控制ϑ从0.1逐步提升至0.7,分析不同分配策略下5个不同任务的总时延变化趋势,实验结果如图21所示.本实验中的任务总时延定义为5个任务的处理时延之和.从图中可以看出,ORAM算法相比其他算法在多任务并行处理时有比较好的表现,较枚举算法平均时延低了12.16%,同时优于能力最大化分配算法19.29%,而随机算法则整体表现较差.

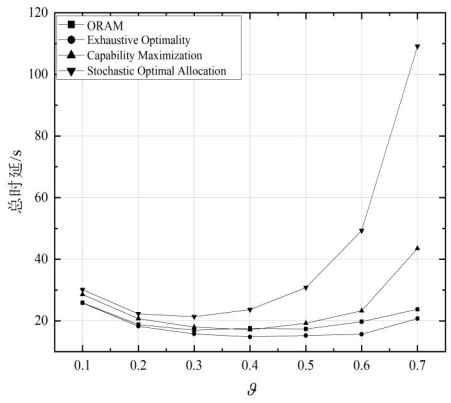


图21 不同节点资源供给量下多任务处理总时延

从整体上来看,任务总时延均呈现先下降后上升的趋势,这说明资源分配系数过小或过大都会导致任务时延增加.当ϑ较小时(0.1~0.3),由于计算节点每次分配的资源较少,任务计算能力受限,导致资源利用率低,任务处理时间相对就会长,使得整体时延较高.当ϑ逐步增大时(0.3~0.5),计算资源利用率提升,任务的计算时延降低,总时延减少,说明适当的资源分配策略能够提高计算效率.当ϑ进一步增大时(0.5~0.7),虽然前期任务分配了更多计算资源,但后续任务的可用计算资源减少,导致后续任务计算时延增加,整体任务总时延回升.

所以合适的资源分配策略能够优化计算资源利用率,提高任务吞吐量,降低多任务并行处理的总时延.若资源分配过少,计算资源利用率低;若资源分配过多,前期任务消耗过多资源,后续任务受影响,导致全局时延增加.因此,在多任务并行调度中,需要在资源利用率与任务公平性之间寻求平衡,以优化整体任务完成时间.

6 结论

本文针对算力网络中“云—边—端”资源协同优化的难题,提出了一种面向传输与计算融合的异构算力网络模型,将其建模为基于算力需求、分发、交易与调配的数学模型,并将任务调度总结为一个混合整数非线性规划问题.本文创新性地引入了算力网络串行子任务路径分配机制和ORAM任务调度算法,有效降低了传输时延和成本.为验证串行子任务路径分配机制和ORAM任务调度算法在解决“云—边—端”算力协同优化问题上的有效性和优越性.本文通过覆盖了不同计算需求、感知范围和节点数量的情况的详细实验对比,全面评估了所提出的任务调度算法相对于多种基准算法在时延、花费和路径等方面的性能表现.实验结果显示,ORAM算法在“云—边—端”架构下表现出较高的调度性能和优势.通过合理分配和利用云、边缘和终端的资源,该算法不仅能够满足不同业务特性的灵活、可调度的按需服务需求,并具有良好的服务表现.实验还表明,算力网络中不同位置的计算节点算力对任务处理的效果有着一定影响,尤其是核心网边缘节点的计算能力提升对任务处理的优化效果最为显著.未来的研究可以设计在更复杂的计算场景中,如多任务同时接入的场景.以及根据任务接入规律研究感知部署,进一步优化任务调度算法,提高算力网络的效率、性能和可用性.

参考文献

[1] TANG X Y, CAO C, WANG Y X, et al. Computing power network: The architecture of convergence of computing and networking towards 6G requirement[J]. China Communications, 2021, 18(2): 175-185.

[2] 刘颖, 夏雨, 于成晓, 等. 面向智算融合网络的自主防御范式研究[J]. 电子学报, 2024, 52(5): 1432-1441.
LIU Y, XIA Y, YU C X, et al. Research on autonomous defense paradigm for smart computing integration networks[J]. Acta Electronica Sinica, 2024, 52(5): 1432-1441. (in Chinese)

[3] 陈星延, 张雪松, 谢志龙, 等. 面向"云—边—端" 算力系统的计算和传输联合优化方法[J]. 计算机研究与发展, 2023, 60(4): 719-734.
CHEN X Y, ZHANG X S, XIE Z L, et al. A computing

and transmission integrated optimization method for cloud-edge-end computing first system[J]. Journal of Computer Research and Development, 2023, 60(4): 719-734. (in Chinese)

[4] 许小龙, 方子介, 齐连永, 等. 车联网边缘计算环境下基于深度强化学习的分布式服务卸载方法[J]. 计算机学报, 2021, 44(12): 2382-2405.
XU X L, FANG Z J, QI L Y, et al. A deep reinforcement learning-based distributed service offloading method for edge computing empowered Internet of vehicles[J]. Chinese Journal of Computers, 2021, 44(12): 2382-2405. (in Chinese)

[5] SONG S D, MA S Y, ZHAO J M, et al. Cost-efficient multi-service task offloading scheduling for mobile edge computing[J]. Applied Intelligence, 2022, 52(4): 4028-4040.

[6] ZHANG R X, YANG C P, WANG X C, et al. AggCast: Practical cost-effective scheduling for large-scale cloud-edge crowdsourced live streaming[C]//Proceedings of the 30th ACM International Conference on Multimedia. New York: ACM, 2022: 3026-3034.

[7] CHEN X. Decentralized computation offloading game for mobile cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(4): 974-983.

[8] AL-HABOB A A, DOBRE O A, ARMADA A G, et al. Task scheduling for mobile edge computing using genetic algorithm and conflict graphs[J]. IEEE Transactions on Vehicular Technology, 2020, 69(8): 8805-8819.

[9] TUN Y K, DANG T N, KIM K, et al. Collaboration in the sky: A distributed framework for task offloading and resource allocation in multi-access edge computing[J]. IEEE Internet of Things Journal, 2022, 9(23): 24221-24235.

[10] YOU Q, TANG B. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial Internet of Things[J]. Journal of Cloud Computing, 2021, 10(1): 41.

[11] XIE Y P, HUANG X Y, LI J C, et al. Computing power network: Multi-objective optimization-based routing[J]. Sensors, 2023, 23(15): 6702.

[12] NAOURI A, WU H X, NOURI N A, et al. A novel framework for mobile-edge computing by optimizing task offloading[J]. IEEE Internet of Things Journal, 2021, 8(16): 13065-13076.

[13] JIA Q M, HU Y J, ZHANG H Y, et al. Research on deterministic computing power network[J]. Journal on Communications, 2022, 43(10).

[14] 张宏科, 于成晓, 权伟, 等. 融算网络体系基础研究[J]. 电子学报, 2022, 50(12): 2928-2934.
ZHANG H K, YU C X, QUAN W, et al. Fundamental research on computing integration networking[J]. Acta Electronica Sinica, 2022, 50(12): 2928-2934. (in Chinese)

[15] LI Z T. A transcoding task offloading and routing decision-making scheme in live transmission architecture based on computing power network[J]. Journal of Networking and Network Applications, 2023, 3(1): 19-31.

[16] LEI B, LIU Z Y, WANG X L, et al. A new edge computing scheme based on cloud, network and edge fusion: Arithmetic network[J]. Telecom Science, 2019, 35(9): 44-51.

[17] NASIRIAN S, FAGHANI F. Crystal: A scalable and fault-tolerant Archimedean-based server-centric cloud data center network architecture[J]. Computer Communications, 2019, 147: 159-179.

[18] LI Z Q, ZHANG H L, LI X, et al. Distributed task scheduling for MEC-assisted virtual reality: A fully-cooperative multiagent perspective[J]. IEEE Transactions on Vehicular Technology, 2024, 73(7): 10572-10586.

[19] REN Y M, SHEN S H, JU Y L, et al. EdgeMatrix: A resources redefined edge-cloud system for prioritized services[C]//IEEE INFOCOM 2022 - IEEE Conference on Computer Communications. Piscataway: IEEE, 2022: 610-619.

[20] SU L N, WANG N, ZHOU R T, et al. Dynamic service placement and request scheduling for edge networks[J]. Computer Networks, 2022, 213: 108997.

[21] TANG Q Q, XIE R C, FENG L, et al. SIaTS: A service intent-aware task scheduling framework for computing power networks[J]. IEEE Network, 2024, 38(4): 233-240.

[22] FENG L, XIE R C, TANG Q Q, et al. Delay-prioritized task scheduling with load balancing in computing power networks[C]//2024 IEEE Wireless Communications and Networking Conference. Piscataway: IEEE, 2024: 1-6.

[23] SUN Y, ZHANG C, HUANG T. Joint task dispatching and bandwidth allocation with hard deadlines in distributed serverless edge computing systems[J]. Journal of Grid Computing, 2024, 22(2): 51.

[24] HUANG T, CHEN F M, JI G H, et al. Optimal task offloading in edge cloud environment[C]//2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress. Piscataway: IEEE, 2023: 7-13.

[25] MTSHALI M, KOBO H, DLAMINI S, et al. Multi-objec-

tive optimization approach for task scheduling in fog computing[C]//2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD). Piscataway: IEEE, 2019: 1-6.

[26] MALEKI E F, MASHAYEKHY L, NABAVINEJAD S M. Mobility-aware computation offloading in edge computing using machine learning[J]. IEEE Transactions on Mobile Computing, 2021, 22(1): 328-340.

作者简介

马 博 男,1991年9月出生于宁夏回族自治区. 现为浙江工商大学信息与电子工程学院副教授、硕士生导师. 主要研究方向为异构网优化、无人机通信、5G网络中的机器学习算法.
E-mail: mabo@mail.zjgsu.edu.cn

余应洁 男,2001年7月出生于浙江省. 现为浙江工商大学信息与电子工程学院硕士研究生. 主要研究方向为算力网络.
E-mail: 19858874414@163.com

吴莎尘 女,2000年7月出生于云南省. 现为浙江工商大学信息与电子工程学院硕士研究生. 主要研究方向为算力网络、任务调度.
E-mail: wshachen@163.com

倪 畅 男,2001年1月出生于浙江省. 现为浙江工商大学信息与电子工程学院(萨塞克斯人工智能学院)硕士研究生. 主要研究方向为无人机通信优化问题、算力网络优化问题.
E-mail: cn385@sussex.ac.uk

陆 琴 女,1999年2月出生于安徽省. 现为浙江工商大学信息与电子工程学院硕士研究生. 主要研究方向为图神经网络、深度强化学习、算力网络等.
E-mail: 21020090013@pop.zjgsu.edu.cn

陈 超 男,1986年12月出生于浙江省. 现为浙江工商大学信息与电子工程学院副教授、硕士生导师. 主要研究方向为无线通信网络技术、网络编码、机器/深度学习.
E-mail: eckio_491@zjgsu.edu.cn

李传煌 男,1980年8月出生于江西省,现为浙江工商大学信息与电子工程学院教授、硕士生导师. 主要研究方向为新一代网络技术、算力网络、人工智能应用.
E-mail: chuanhuang_li@zjgsu.edu.cn