计 算 机 研 究 与 发 展
Journal of Computer Research and Development

https://crad.ict.ac.cn
62(7): 1754−1770, 2025

本期推荐

算网融合下的多层次计算资源系统

徐家豪[1,2,3,4] 余 辰[1,2,3,4] 李 健[5,6] 金 海[1,2,3,4]

[1]（大数据技术与系统国家地方联合工程研究中心（华中科技大学） 武汉 430074）

[2]（服务计算技术与系统教育部重点实验室（华中科技大学） 武汉 430074）

[3]（集群与网格计算湖北省重点实验室（华中科技大学） 武汉 430074）

[4]（华中科技大学计算机科学与技术学院 武汉 430074）

[5]（上海市可扩展计算与系统重点实验室（上海交通大学） 上海 200240）

[6]（上海交通大学软件学院 上海 200240）

（bvalue@hust.edu.cn）

Multi-Tier Computing Resource System for Computing and Network Convergence

Xu Jiahao[1,2,3,4], Yu Chen[1,2,3,4], Li Jian[5,6], and Jin Hai[1,2,3,4]

[1]（*National Engineering Research Center for Big Data Technology and System* (*Huazhong University of Science and Technology*) , *Wuhan* 430074）

[2]（*Services Computing Technology and System Lab* (*Huazhong University of Science and Technology*), *Ministry of Education*, *Wuhan* 430074）

[3]（*Cluster and Grid Computing Lab* (*Huazhong University of Science and Technology*), *Wuhan* 430074）

[4]（*School of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074）

[5]（*Shanghai Key Laboratory of Scalable Computing and Systems* (*Shanghai Jiao Tong University*), *Shanghai* 200240）

[6]（*School of Software*, *Shanghai Jiao Tong University*, *Shanghai* 200240）

Abstract The purpose of computing first network (CFN) is to deeply integrate the ubiquitous computation with network, in order to effectively allocate multi-dimensional basic resources such as computation and storage between clouds, edges and ends through the network, allowing users to use them as transparently as water and electricity resources. Computing resources can be requested on demand and used at any time. Due to heterogeneous computing resources, dynamic network and diverse user needs, it has become one of the core challenging problems to effectively schedule and route resources in computing first network. To address this problem, we design a multi-tier computing resource system (CRS). Different from the existing resource allocation, CRS is a complete computing first network technology solution based on the application layer, considering the computing resources awareness and computational routing. The computing resource system is composed of computing and network resource awareness strategy and computing resource routing protocol. The computing and network resource awareness strategy defines the intra-domain awareness rules within the jurisdiction and the inter-domain awareness rules between different jurisdictions. Based on this, we proposed a greedy-based resource routing algorithm (GBRA), which can dynamically generate a search tree for each task. The computing resource routing protocol completes the allocation of resources through CRS request message, authorization notification message, notification confirmation message and CRS response message. Through extensive simulation experiments, compared with other algorithms, it is demonstrated that CRS can complete the resource allocation of more tasks within the maximum response latency tolerated. In addition, better load

收稿日期：2024-03-01；**修回日期**：2024-12-18

基金项目：国家自然科学基金重点项目 (62232012)；国家自然科学基金项目 (62272179)

This work was supported by the Key Program of the National Natural Science Foundation of China (62232012) and the National Natural Science Foundation of China (62272179).

通信作者：余辰（yuchen@hust.edu.cn）

balancing can be achieved among the computing nodes within the jurisdiction.

Key words computing first network; computing resource system (CRS); computing and network resource awareness strategy; computing resource routing protocol; jurisdiction

摘 要 算力网络旨在将泛在算力与网络进行深度融合，以期通过网络将计算、存储等多维基础资源在云、边、端之间进行有效调配，让用户可以像使用水电资源一样透明地使用算力资源，按需索取，随取随用. 然而对于云边端异构的计算资源、动态的网络负载和多样化的用户需求，如何有效地进行资源的调度和路由成为了算力网络面临的核心挑战之一. 为解决上述挑战，设计了一套多层次的计算资源系统（computing resource system, CRS）. 不同于现有的资源调配，CRS 是一套建立在应用层之上并且兼顾考虑算网感知和算力路由的完整的算力网络技术方案. CRS 由算网资源感知策略和算力资源路由协议组成. 算网资源感知策略定义了辖区系统内部的域内感知规则和不同辖区之间的域间感知规则，并基于此提出了一种基于贪心的资源路由算法（greedy-based resource routing algorithm, GBRA），为每个任务动态生成感知搜索树. 算力资源路由协议通过 CRS 请求报文、授权通告报文、通告确认报文和 CRS 响应报文来完成资源的申请与调配工作. 通过大量的数据仿真实验证明，与其他算法相比，CRS 可以在任务容忍的最大响应时延内，完成对更多任务的资源分配工作. 此外，对于辖区系统内部计算节点之间可以实现较优的负载均衡.

关键词 算力网络；计算资源系统；算网资源感知策略；算力资源路由协议；辖区系统

中图法分类号 TP393

DOI: 10.7544/issn1000-1239.202440124 **CSTR:** 32373.14.issn1000-1239.202440124

随着 5G 和 6G 新一代移动通信网络技术[1-2]的持续发展，网络在传输效率、覆盖范围和信息安全等多个方面得到了显著的提高，其极大地减少了用户的网络通信时延，带来了更加优质高效的网络服务，这也为建立在传统意义之上被定义为难以实现的应用提供了可能，推动着新一轮的科技革命和产业的变革. 以无人驾驶[3]、元宇宙[4]、智慧城市[5]、工业互联网[6]为代表的新型业务应用部署广泛，相较于只需要简单的数据收集和呈现的传统应用，这些新型业务应用更加侧重于对于复杂信息的提取与分析，本身会产生海量数据的同时还需要强大的智能算法的加持. 同时这类应用需要实时的数据处理、敏捷计算和分析决策，这对于当前的网络和算力提出了新的挑战.

单一的云计算[7]、雾计算[8]和边缘计算技术方案[9]已经不足以支撑当前场景下用户的请求. 为了妥善地解决这个问题，云边端协同[10-12]的复合型的技术方案成为了一种新兴的趋势. 以算为中心、网为根基的新型范式算力网络（computing first network）应运而生. 算力网络是由运营商、设备商在 ITU-SG13[13]上提出的一种基于分布式系统的计算与网络融合的技术方案，旨在通过网络将计算、存储等多维基础资源在云、边、端之间进行有效调配，实现“算力泛在、算网共生、智能编排、一体服务”目标，达成“网络无所不达、算力无所不在、智能无所不及”的愿景[14]. 国内是算力网络技术方案的率先提出方，并已得到了业界的广泛认可. 同时随着国家信息中心提出的“东数西算”工程[11]，更是为算力网络找到了切实可行的“杀手级”应用，算力网络的话题持续升温.

目前对于算力网络的研究呈现百花齐放的景象，但是由于算力网络规模的庞大性与问题的复杂性，其仍然面临着诸多挑战. 具体来说，本文主要考虑 3 个方面：

1）算网资源的感知问题. 现有的算网资源感知算法中[15-19]，往往是通过扩展现有的 BGP（Border Gateway Protocol），IGP（Interior Gateway Protocol）协议，将运行于计算节点的守护进程所捕获到的资源负载向量发送于本地的路由节点，本地路由节点再将此资源负载向量发送于远端路由节点，最终使得该计算节点的算力资源状态在整个自治系统内被通告. 这种策略忽略了计算节点的作用域与共享域特性，在大型的自治系统内路由表会变得特别庞大（IP 的路由聚合可以减少路由表的大小和复杂性，而对于算力标识与算力的聚合目前仍处在研究阶段），这将依赖于特定的硬件设备，例如具有大存储空间的 CFN 路由器；与此同时，在大型自治系统内计算节点的资源负载向量同步往往具有滞后性，这会降低当前一些调度算法的效率.

2）路由策略的实现机制. 现有解决方案[20-21]虽然针对算力网络分布式、异构化的资源特征，提出的诸多方案试图解决任务调度问题，然而这些方案仍然以集中式为主. 集中式的策略在实现上较为简单，但

是需要上层系统掌握全网节点的状态信息，对于编排层的智能计算和存储有一定要求，除此之外还会引入一部分的决策时延. 对于分布式的路由方案，其扩展性较强，与算力网络结构相适配. 同时在算力网络中对于分布式路由协议及其报文结构的研究也较少.

3)算力节点的协同问题. 部分计算卸载方面的工作考虑了云、边、端间计算资源的纵向协同，却忽略了计算节点之间的横向协作[15]. 不同区域的计算节点以及终端设备的数目和任务量也不同，若不考虑计算节点之间的协同，容易导致各个计算节点之间负载不均衡，无法充分利用节点的计算资源. 此外，还有相当一部分工作使用强化学习的方法通过将资源调度问题指定为马尔可夫决策过程来适应系统变化. 在算力网络场景中，搜索空间庞大，模型往往需要长时间的预训练，同时对于高流动、不稳定的区域模型甚至难以收敛，实际运用于协同调度的效果以及其扩展性并不好.

针对以上 3 个问题，本文设计了一套多层次计算资源系统(computing resource system, CRS). 不同于现有的资源调配，CRS 是一套建立在应用层之上并且兼顾考虑算网感知和算力路由的完整的算力网络技术方案. 本文的主要研究内容和贡献包括 4 个方面：

1)根据不同计算节点对共享性的要求不同，将算力网络在水平和垂直 2 个维度上进行划分. 水平层面划分为边缘层、多级雾层和云层. 垂直层面引入辖区系统的概念，将每一层进一步划分为多个不同的辖区系统.

2)为实现算网感知，设计并实现了算网感知策略. 感知策略定义了域内感知规则和域间感知规则，动态维护一系列最小延迟的同层节点和上层节点.

3)提出了一种基于贪心的资源路由算法，可为每个用户任务生成不同的感知搜索树；并在此基础上定义了算力资源路由协议及其报文结构，以此来完成对资源的申请与调配工作.

4)从网络协议层面设计一套多层次的计算资源系统. 该系统同时考虑了算网感知和算力路由，是一个完整的算力网络技术方案. 数据仿真结果表明，CRS 可以在任务容忍的最大响应时延内，完成对更多任务的资源分配工作. 此外，对于辖区系统内部计算节点之间可以实现较优的负载均衡.

1 相关工作

1.1 算网资源感知与算力网络架构

目前，算力网络的主要研究单位包括中国移动研究院、中国电信研究院、华为技术有限公司、中国联通研究院、中兴通讯技术以及鹏城实验室等科研院所. 各个研究单位自 2019 年以来经过多轮的探讨调研，并相继公布了系列研究成果. 中国移动研究院和华为技术有限公司在 IETF 发布了计算优先网络(compute first networking, CFN)技术方案[16-17]，结合当前计算能力状况和网络状况作为路由信息发布到网络，将计算任务报文路由到相应的计算节点，实现整体系统最优和用户体验最优. 随后中国移动研究院[18,19,22]发布了《算力感知网络技术白皮书 2019》《算力网络白皮书 2021》，分别介绍了算力感知网络(computing-aware networking, CAN)、算力网络(computing force network, CFN)的背景需求、技术架构以及关键技术. 中国电信研究院的算力网络框架(computing power network, CPN)与架构标准成为算力网络首个国际标准[13]. 文献 [23] 提出了融智算力网络架构，将 AI 赋能算力网络，实现内生智能和业务智能，同时对 3 个典型场景做了对比实验. 借鉴 DNS 解析映射机制[24-25]，将算力网络中的算力资源经过层次化抽象成树状层次化模型后按照域名组织规则进行编码、注册和管理. 文献 [24] 使用 URL 语言对多种算力资源进行统一标识，编码为：算力单元名/算力类别/算力属性 1/算力属性 2/算力属性 3/…，最终通过集中式的算力资源管理平台对算力资源进行统一的分配调度. 文献 [25] 将算力资源抽象为算力位置标识、算力身份标识和算力属性标识，通过算力资源通信地址的分级映射、算力资源信息获取和算网信息融合实现算力服务的匹配.

1.2 算网资源调度算法

为了协同泛在的计算资源以满足用户的多样化需求，相当一部分研究专注于资源分配的任务调度. 文献 [10] 总结了云—边—端计算系统优化指标、调度模型及其求解算法，同时归纳了一些典型应用案例. 文献 [12] 将资源调度问题进一步细分为资源调度、任务流调度和任务调度 3 个子问题，并结合数学模型总结了子问题的求解方法. 陈星延等人[26] 为实现云边端资源的协同优化，提出了一种多维资源融合的增广图模型，并设计了基于波利亚重球法的异构资源协同优化算法. 文献 [27] 提出了一种双层任务卸载和资源调度方法，上层使用基于贪心的流水车间调度算法处理任务卸载决策和卸载调度问题，下层采用深度强化学习技术优化服务器资源分配问题. 一些研究者也试图将泛在的计算资源和终端用户的协同调度问题抽象为传统的图论问题. 孙钰坤

等人[28]将多用户场景下的资源调度问题抽象为一个多源汇最短路问题，提出了一种基于 Floyd 算法的算力感知路由调度策略. 文献 [29] 提出了一种联合工作流任务卸载和流调度的启发式算法，利用每个任务图的知识和流程的开始时间来做任务卸载决策. 文献 [30] 通过对任务进行分解，将任务抽象为链式结构和有向无环图结构，考虑资源分配和任务调度，并依次提出一种基于势博弈的分布式工作流卸载算法和一种基于动态资源权重的启发式工作流卸载算法. 文献 [31] 将应用程序抽象划分为多个具有依赖关系的子任务，并将其抽象为有向无环图，与此同时提出了一种优先级调度算法用于任务调度、离线训练和在线部署，以最小化任务延迟的同时降低能耗. 文献 [32] 提出一个区块链赋能的资源调度算法，将计算资源调度问题建模为一个马尔可夫决策过程，通过区块链来维护资源调度信息，并使用强化学习的方法来求解. 对于高带宽的视频流任务场景，文献 [33] 提出了一种可变比特流的路由策略，将视频缓存抽象为凸优化问题，通过学习梯度来实时确定最优云边端缓存策略.

此外，算力网络进行分层调度的思想也愈发引起人们的关注[34-36]. 根据计算节点覆盖范围、计算能力和存储资源等特点，将泛在协同的计算节点进行分层，构建一个多层次的算力网络，为多样化的用户任务提供适应性的调度. 文献 [34] 提出了一个云雾混合多层次算力网络以及卸载系统，研究了以最小化时延、能耗及付费为目标的代价感知任务调度问题，将耦合型卸载决策问题重新建模为非合作博弈问题，并引入势博弈分析框架设计了一个分布式任务调度算法. 与文献 [34] 类似，巩宸宇等人[35]定义了一个由时延、能耗组成的加权代价函数以建模一个任务调度问题，并提出一个基于交叉熵的集中式不可分割任务调度算法. 张厚浩等人[36]提出了一个基于 5G 网络模型的分层网络结构，构建为一个以 ISP（Internet service provider）的利润最大化为优化目标的混合整数规划问题.

2 算网资源感知策略

本节将讨论计算资源系统中算网资源感知策略. 算网资源感知策略包含域内感知策略和域间感知策略. 首先给出了系统划分依据及辖区系统的定义. 随后，给出了计算节点需要维护的数据结构，并分别详细介绍了域内感知规则和域间感知规则的算法实现. 最后对算网感知策略的复杂度进行了分析.

2.1 系统划分

为实现计算节点的算网状态感知，多层次计算资源系统水平层面依据计算节点所拥有的计算、存储、带宽等综合能力进行划分，将算力网络划分为云层、边缘层和多级雾层. 云层位于最上层，远离终端设备，通常具有超大的网络带宽和计算资源. 边缘层位于最底层，与终端用户最近，可快速响应用户请求. 多级雾层位于边缘层与云层之间，其网络和计算能力介于边缘层与云层之间，作为云层和边缘层的缓冲调度层.

除此之外，多层次计算资源系统在垂直层面将不同层级进一步划分为多个不同的辖区系统. 在给出辖区系统的定义之前，本文对后续引入的一些名词做一个简要的解释：

1）计算节点的作用域是指特定区域内的用户集合. 以时延或者距离指标来衡量，其与计算节点所属水平层级有关. 例如边缘节点的作用域可定义为与当前节点延迟小于$\tau(\tau = 50\ \text{ms})$的用户集合，即计算节点$s$的作用域为$\{u|T_{su} \leqslant \tau\}$，$u$表示用户，$T_{su}$表示计算节点$s$与用户$u$的延迟. 计算节点所属水平层级越高，其对应的作用域也越大.

2）计算节点的共享域是指同层级下特定区域内的计算节点集合. 节点的作用域内的用户任务可挂载至该节点的共享域内的其他计算节点上. 共享域以时延指标来衡量. 计算节点s的共享域为$\{s'|T_{ss'} < T_s^{\text{high}}\}$，$s'$表示计算节点，$T_{ss'}$表示计算节点$s$与节点$s'$的延迟，$T_s^{\text{high}}$表示节点$s$与上层次计算节点的最小通信时延.

辖区系统是以某个计算节点为中心，与其共享域共同构成的计算节点的集合. 每个计算节点都有以各自为中心节点的辖区系统. 在动态的算力网络环境下，每个节点所维护的辖区系统也是动态的.

2.2 域内感知规则

域内感知规则定义了辖区系统内部的算网感知策略. 通过运行域内感知规则为每个中心节点动态维护一系列最小延迟的同层节点集合.

域内感知规则需要中心节点维护如下 2 个数据结构：

1）域内感知队列. 以负载为关键字的优先队列，队头节点负载最低；队列内为一系列与中心节点延迟最优的同层计算节点，即其内元素始终为中心节点的辖区系统内的其他计算节点. 如图 1 所示，c_i表示辖区系统内的第i个计算节点.

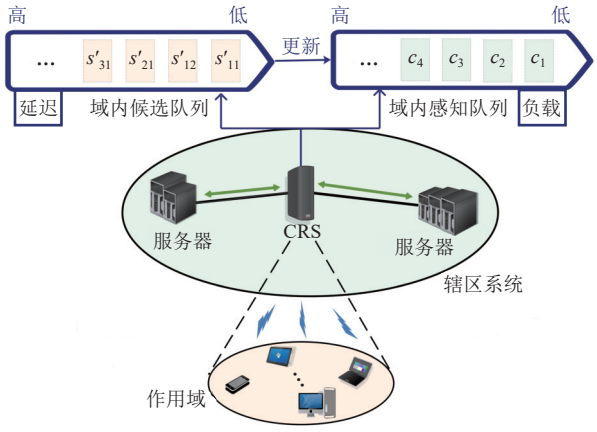


Fig. 1 Intra-domain awareness rule

图 1 域内感知规则

2)域内候选队列. 以延迟为关键字的优先队列, 队头节点与中心节点延迟最低; 其内元素是以域内感知队列内节点为中心的辖区系统内其他计算节点的集合, 如图 1 所示, s'_{ij}为以c_i为中心的辖区系统内的第j个计算节点.

算法 1. 域内感知算法.

输入: 域内候选队列 *IntraCandidateQu*(以延迟为关键字的优先队列), 域内感知队列 *IntraAwarenessQu*(以负载为关键字的优先队列).

① 初始化新的域内候选队列 *Qu*;
② while !*IntraCandidateQu.empty*()
③ 取出域内候选队列队头元素s'_{nm};
④ *IntraCandidateQu.pop*();
⑤ 主动探测节点s'_{nm}和c_n的算网信息并更新;
⑥ if c_n in *IntraAwarenessQu* 且中心节点到s'_{nm}的延迟小于中心节点到c_n的延迟
⑦ *IntraAwarenessQu.remove*(c_n); /* 移除c_n */
⑧ *IntraAwarenessQu.insert*(s'_{nm}); /* 插入s'_{nm} */
⑨ end if
⑩ end while
⑪ foreach c_n in *IntraAwarenessQu*
⑫ 主动探测节点c_n的算网信息并更新;
⑬ 将以c_n为中心的辖区系统内的其他计算节点加入 *Qu*;
⑭ end foreach
⑮ 对 *Qu* 去重;
⑯ *IntraCandidateQu* = *Qu*.

算法 1 的核心是去当前辖区系统内检查每一个计算节点c_i, 在以c_i为中心的辖区系统内是否有其他计算节点与中心节点的延迟比c_i更优. 若有, 如算法 1 中行⑥~⑨, 将该更优节点替换c_i, 以此更新域内感知队列, 动态维护一系列最小延迟的节点集合; 同时通过域内感知队列来更新域内候选队列, 如算法 1 中行⑪~⑯.

2.3 域间感知规则

域间感知规则定义了下层辖区系统与上层辖区系统之间的算网感知策略. 通过运行域间感知规则为每个计算节点动态维护一系列与上层延迟最低的节点集合.

域间感知规则需要每个中心节点维护 2 个数据结构:

1)域间感知队列. 以负载为关键字的优先队列, 队头节点负载最低; 队列内为一系列与中心节点延迟最优的上层计算节点, 如图 2 所示.

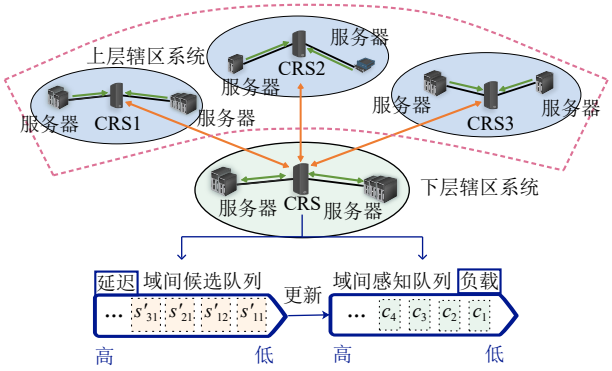


Fig. 2 Inter-domain awareness rule

图 2 域间感知规则

2)域间候选队列. 以延迟为关键字的优先队列, 其队头节点与中心节点的延迟最低; 其内元素为以域间感知队列内元素为中心的辖区系统内其他计算节点的集合, 如图 2 所示, s'_{ij}为以c_i为中心的辖区系统内的第j个计算节点.

由于不同层次之间的节点之间的通信延迟往往比同一辖区系统内部节点之间的通信延迟高得多, 同时位于同一辖区系统内的计算节点的域间感知队列往往具有相似性, 故为了加速域间感知规则, 算法使用域内感知队列来优化这个过程. 具体伪代码如算法 2 所示.

算法 2. 域间感知算法.

输入: 域间候选队列 *InterCandidateQu*(以延迟为关键字的优先队列), 域间感知队列 *InterAwarenessQu*(以负载为关键字的优先队列), 域内感知队列 *IntraAwarenessQu*.

① 初始化新的域间候选队列 *Qu*;
② while !*InterCandidateQu.empty*()
③ 取出域间候选队列队头元素s'_{nm};

④ *InterCandidateQu.pop*();
⑤ 主动探测节点 s'_{nm} 和 c_n 的算网信息并更新;
⑥ if c_n in *InterAwarenessQu* 且中心节点到 s'_{nm} 的延迟小于中心节点到 c_n 的延迟
⑦ *InterAwarenessQu.remove*(c_n); /* 移除 c_n */
⑧ *InterAwarenessQu.insert*(s'_{nm}); /* 插入 s'_{nm} */
⑨ end if
⑩ end while
⑪ foreach c_n in *IntraAwarenessQu*
⑫ foreach c'_n in c_n 的域间感知队列
⑬ if 中心节点到 c'_n 的延迟大于 CRS 到域间感知队列尾元素的延迟
⑭ c'_n 替换域间感知队列尾元素;
⑮ end if
⑯ end foreach
⑰ end foreach
⑱ foreach c_n in *InterAwarenessQu*
⑲ 主动探测节点 c_n 的算网信息并更新;
⑳ 将以 c_n 为中心的辖区系统内的其他计算节点加入 *Qu*;
㉑ end foreach
㉒ 对 *Qu* 去重;
㉓ *InterCandidateQu* = *Qu*.

域间感知规则与域内感知规则相似，同样是检查每一个域间感知队列的节点 c_i，在以 c_i 为中心的辖区系统内是否有节点与当前中心节点的延迟比 c_i 更优，若有，则将该更优节点替换 c_i，特别的域间感知规则通过对域内感知队列进行检查，如算法 2 行⑪~⑰所示，如果辖区系统内其他节点的域间感知队列内元素与中心节点的延迟小于域间感知队列队尾元素与中心节点的延迟，则进行替换. 最后，通过域间感知队列来更新域间候选队列，如算法 2 行⑱~㉓所示.

2.4 算法分析

域内感知实际上是一种穷举搜索算法. 由于其是一种动态的适应算法，实际的算法复杂度与物理拓扑密切相关. 以下提供一种粗略的分析方式以提供参考.

如图 3 所示，初始的新增节点（绿色节点），其辖区系统内为空（绿色虚线内区域）. 平衡节点为已融合后的计算节点（蓝色节点），其辖区系统内维护了一系列与自身延迟最低的节点（蓝色虚线内的区域）. 新增节点接入网络，将平衡节点加入感知队列内.

1）探测平衡节点，将 a_1 区域（蓝色与绿色重合的区域）内的节点加入自身感知队列内.

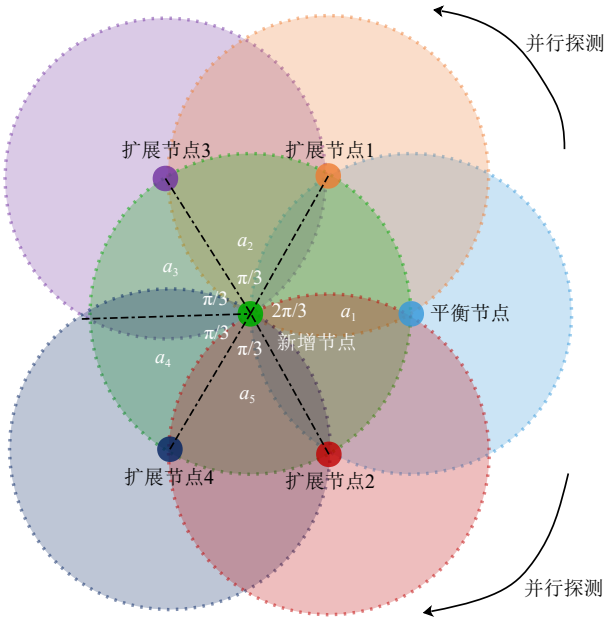


Fig. 3 Scaling process of awareness algorithm
图 3 感知算法的扩展过程

2）探测区域 a_2 内节点，如扩展节点 1，将 a_2 区域（橙色与绿色重合的区域）内的节点加入自身感知队列内.

递归以上的探测操作，直到新增节点的辖区系统被完全覆盖（a_1,a_2,a_3,a_4,a_5），即新增节点完成了自身辖区系统的构建. 经分析易知总共需要对 5 片区域内的计算节点进行探测，即时间复杂度为 $O(5n)$，其中 n 为辖区系统内计算节点的数目. 在现实场景中，对于同一区域内的计算节点可以并行地去探测，与此同时可以从区域 a_1 的两侧开始并行探测，所需要的探测时间会被大大缩短，快速达到收敛.

3 基于贪心的资源路由算法

以设定的算网感知策略作为前置工作，本节提出了一种基于贪心的资源路由算法（greedy-based resource routing algorithm, GBRA）. 该算法运行于计算资源系统之上，可为每个用户任务生成不同的感知搜索树. 首先给出了 GBRA 基于的贪心策略以及算法逻辑，其次对算法的时间复杂度进行了理论分析.

3.1 算法逻辑

为了便于表达和分析，用 U 表示边缘端用户的集合，用 S 表示计算节点的集合. 使用一个多元组 $\boldsymbol{u}_i$ 表示用户 i 的计算任务，$\boldsymbol{u}_i=[\boldsymbol{req}^i,d^i_{\max}]$，其中 $\boldsymbol{req}^i$ 是任务所需要的计算资源向量，$d^i_{\max}$ 是任务容忍的最大响应延迟. 使用 A 表示用户的卸载策略，$A=\{\boldsymbol{a}^1,\boldsymbol{a}^2,\cdots,\boldsymbol{a}^M\}$，

其中$\boldsymbol{a}^i$是一个向量，表示任务i所经过的调度节点，并最终挂载至a_k^i节点，$\boldsymbol{a}^i=\left(a_1^i,a_2^i,\cdots,a_k^i\right)(a_j^i\in S)$卸载向量中任意2个相邻节点之间的传输延迟为$t_{a_j^i a_{j+1}^i}^{\text{trans}}$.

任务提交至得到响应的总时间包含三大部分，分别是任务的传输时延、计算时延以及等待时延. 任务的计算时延往往由算力管理平台和用户所选计算资源决定，是确定的，故在此仅考虑用户任务的传输时延和等待时延，将其定义为用户i的响应时延：

$$T_i^{\text{resp}}=T_i^{\text{trans}}+T_i^{\text{wait}}=\sum_{j=1}^{k-1}t_{a_j^i a_{j+1}^i}^{\text{trans}}+t_{a_k^i}^{\text{wait}}. \tag{1}$$

GBRA基于的贪心策略为2方面：

1)最大化任务完成数(在任务容忍的最大响应延迟内完成交付)，即

$$\max_{|U,A|}\sum_{i=1}^{N}I(d_{\max}^i\geqslant T_i^{\text{resp}}), \tag{2}$$

其中$I(x)$为指示函数，当x为真，则$I(x)=1$；否则$I(x)=0$.

2)辖区系统内计算节点的负载均衡. 设辖区系统$J=\{s_1,s_2,\cdots,s_n\}$，其中s_i是辖区系统J内的一个计算节点，w_i为节点s_i的负载均衡系数. 当辖区系统内节点较多时使用方差来衡量辖区系统的负载均衡系数：

$$\hat{w}=\sum_{i=1}^{n}w_i, \tag{3}$$

$$\sigma^2(J)=\frac{1}{n}\sum_{i=1}^{n}(\hat{w}-w_i)^2. \tag{4}$$

当计算节点较少时，使用极差来描述辖区系统的负载均衡系数：

$$R(J)=\sum_{i=1}^{n}\sum_{j=1}^{n}\max_{i\neq j}|w_i-w_j|-\sum_{i=1}^{n}\sum_{j=1}^{n}\min_{i\neq j}|w_i-w_j|. \tag{5}$$

方差(极差)越大，辖区系统的负载越不均衡；方差(极差)越小，辖区系统越均衡. 固贪心策略使得所有以计算节点为中心的辖区系统的负载均衡系数尽可能小，即

$$\min_{|U,A|}=\frac{1}{M}\sum_{i=1}^{M}\sigma^2(J_i)\text{或}\min_{|U,A|}\frac{1}{M}\sum_{i=1}^{M}R(J_i). \tag{6}$$

基于以上2点贪心策略，对于每一个计算任务，在其容忍的最大延迟范围内尽可能地将其调度至更高层，为低层次计算节点保留更多的资源提供给后续产生的需要快速响应的任务；同时为了实现辖区系统内计算节点之间的负载均衡，在满足任务所需足够计算资源的前提下，优先将任务挂载负载更低的节点上. 伪代码如算法3所示.

算法3. 基于贪心策略的资源路由算法GBRA.

输入：任务所需的资源向量$\boldsymbol{req}$，任务容忍的最大时延$d_{\max}$；

输出：*CRS*.

① def *GBRA*($\boldsymbol{req}$,*CRS*)
② foreach *server* in *CRS*的域间感知队列 /* 搜索域间感知队列 */
③ if |$\boldsymbol{req}$|≤|*server*|且$d_{\max}$大于等于*CRS*到*server*的通信时延
④ 更新任务容忍的最大时延$d_{\max}$(减去*CRS*到*server*的通信开销)；
⑤ 交付至*server*；
⑥ return *GBRA*($\boldsymbol{req}$,*CRS*)；
⑦ end if
⑧ end foreach
⑨ foreach *server* in *CRS*的域内感知队列 /* 搜索域内感知队列 */
⑩ if |$\boldsymbol{req}$|≤|*server*|且$d_{\max}$大于等于*CRS*到*server*的通信时延
⑪ 更新任务容忍的最大时延$d_{\max}$(减去*CRS*到*server*的通信开销)；
⑫ 交付至*server*；
⑬ return *GBRA*($\boldsymbol{req}$,*CRS*)；
⑭ end if
⑮ end foreach
⑯ return *CRS*； /* 交付至*CRS* */
⑰ *GBRA*($\boldsymbol{req}$,*localCRS*).

算法3的运行流程如图4所示，其是一个迭代的过程，主要包含2部分：1)搜索域间感知队列，检查任务是否可以向更高层调度，即对应算法3行②~⑧；

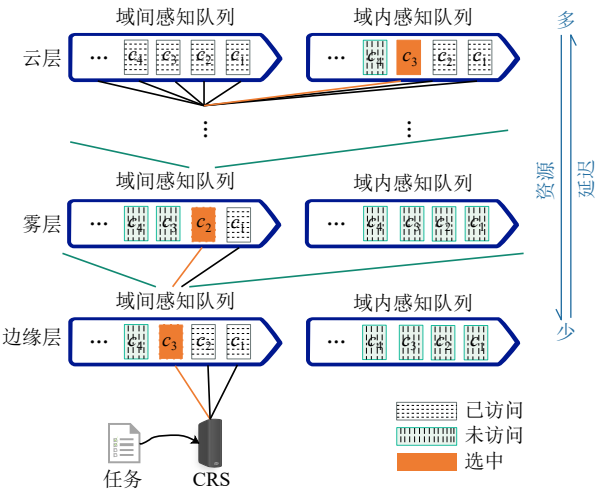


Fig. 4 Greedy-based resource routing algorithm

图4 基于贪心的资源路由算法

2)搜索域内感知队列，检查是否可以将任务挂载至负载更低的节点，即对应算法 3 行⑨~⑮.

3.2 算法分析

GBRA 算法的时间复杂度为 $O(L\times n\log n)$. 其中 n 指感知队列的大小规模，L指水平层级数的规模. 搜索域间感知队列并依次检查队头元素的时间复杂度为 $O(n\log n)$，同理，搜索域内感知队列并依次检查队头元素的时间复杂度为 $O(n\log n)$，GBRA 算法最多迭代次数与算力网络的水平层级数同一量级，故算法的整体时间复杂度为 $O(L\times n\log n)$.

计算节点根据算网感知策略生成感知搜索树，同时节点基于贪心的资源路由算法将任务分段路由交付至“合适”节点，通过 2 点贪心策略(最大化任务完成数和负载均衡)可实现节点之间的互相协同. 如图 5 所示，其中圆圈表示计算节点，虚线框表示在同一个辖区系统内(更为严谨的说法应该是，互相在以各自为中心的辖区系统内，即计算节点b_1在以b_2为中心的辖区系统内，计算节点b_2在以b_1为中心的辖区系统内).

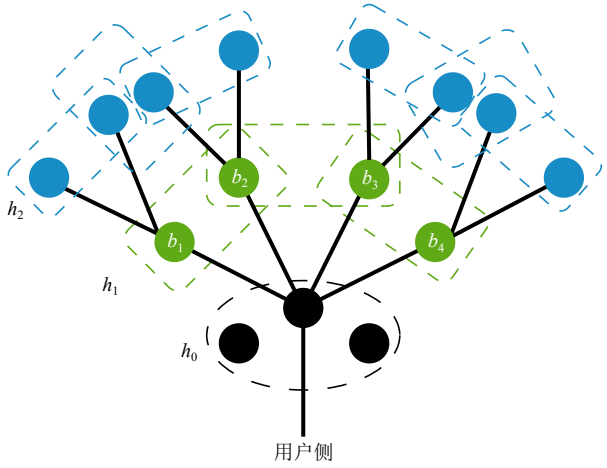


Fig. 5 Collaboration of computing nodes

图 5 计算节点的协同

1)纵向协同. 计算任务 $\boldsymbol{u}_i=(\boldsymbol{req}^i,d^i_{\max})$，任务会被挂载至$h^j_k$，即第$k$层的$j$号节点，那么节点$h^j_k$会满足

$$\begin{cases} t_{i\to h_{k-1}}\leqslant t_{i\to h^j_k}\leqslant d^i_{\max}<t_{i\to h_{k+1}}, \\ \left|\boldsymbol{req}^i\right|\leqslant\left|\boldsymbol{resource}^{h^j_k}\right|, \end{cases}$$

其中$t_{q\to r}$表示节点q与节点r之间的通信延迟.

2)横向协同. 若计算任务在纵向上被调度至h_1层，其可以根据负载状况在b_1,b_2,b_3,b_4之间进行卸载. 即位于同层的不同计算节点，即使它们不在各自的辖区系统内，其也可以通过传递的方式进行互相感知和任务的挂载($b_1\leftrightarrow b_2\leftrightarrow b_3\leftrightarrow b_4$).

4 算力资源路由协议

本节首先介绍算力资源路由协议整体流程；随后给出了对于协议考量以及设计的报文结构；最后对协议的可扩展性进行理论分析.

4.1 协议流程

算力资源路由协议整体流程如图 6 所示，主要由 CRS 请求报文、授权通告报文、通告确认报文、CRS 响应报文和资源访问请求 5 部分组成. 协议的具体工作流程为：

1) 计算任务通过无线或有线链路向本地计算资

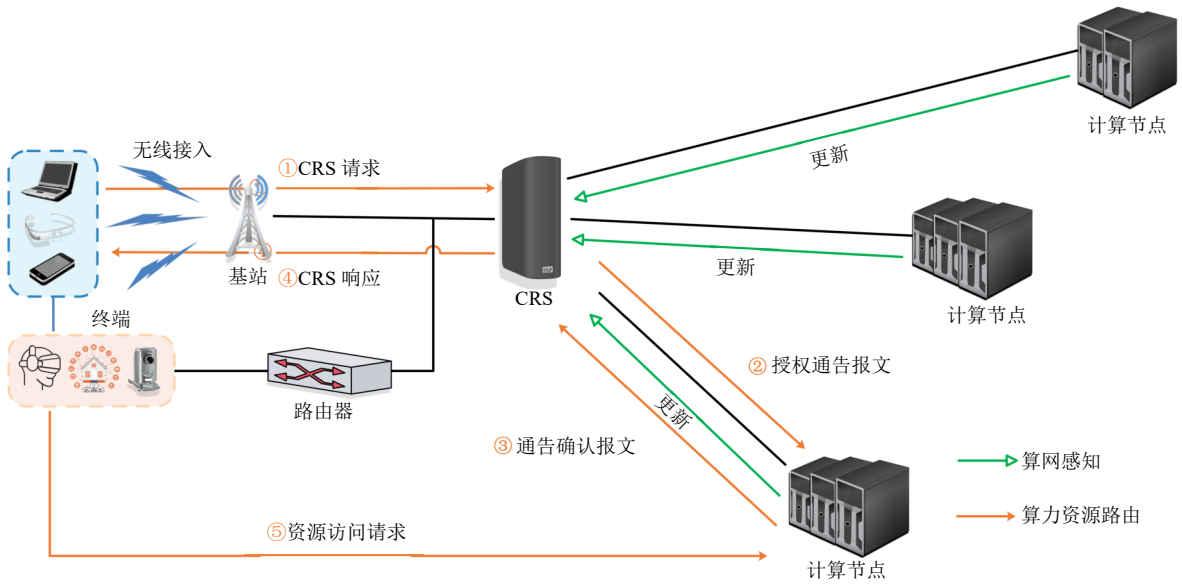


Fig. 6 Computing resource routing protocol process

图 6 算力资源路由协议流程

源系统发送 CRS 请求报文, CRS 请求报文中包含任务执行所需要的计算资源、存储资源、与运行环境相关的资源配置和响应时延等信息.

2)本地计算资源系统接收到用户发送的 CRS 请求报文后, 根据 GBRA 进行搜索, 并依次将请求向其他计算资源系统交付, 直到找到最终交付的计算节点并向计算节点发送授权通告报文.

3)计算节点接收到计算资源系统发送的授权通告报文之后, 根据报文请求体为指定的任务配置运行环境, 预留相应的计算、存储等资源; 同时向计算资源系统发送通告确认报文, 告知计算资源系统已完成预分配, 等待用户计算请求.

4)计算资源系统在收到通告确认报文之后按照预先交付路径依次返回至本地计算资源系统, 本地计算资源系统再向用户发送 CRS 响应报文.

5)用户收到 CRS 响应报文后, 查看响应码, 若请求成功, 则向报文中的计算节点发送资源访问请求, 并将任务挂载至该计算节点.

4.2 报文结构

在对算力资源路由协议的报文结构进行设计时, 我们主要考虑 3 个方面:

1)系统融合度. 算力资源路由协议和算网感知策略同属于多层次计算资源系统, 为达到更好的融合, 算力资源路由协议对算网的感知应有良好的支持. 算网状态感知可以独立地进行, 也应可以随算力资源路由一同捎带感知.

2)报文结构能体现用户需求. 多样化的用户对于算力以及响应延迟的要求不同. 除此之外, 网络中会出现一些紧急任务, 需要能够支持该种任务的优先调度与挂载.

3)节点共享性的普遍性与特异性. 运营商、企业、政府设立的计算节点对于共享性的要求不尽相同. 运营商设立节点可能是为了更多的服务用户获取更多利润, 部分企业和政府设立节点更多的可能为内部机构服务, 并不想与外部用户共享算力.

具体报文结构如图 7 所示, 主要有 3 部分组成, 分别是报文头部(Header)、申请记录(Application)和感知记录(Awareness).

1)报文头部是必须存在的, 它定义了报文的类型, 携带了报文的一些基本信息, 也决定了其他部分是否存在, 报文头部字段含义如下所示.

① Transaction ID(事务 ID): 占 32 b, 用于标识当前的资源请求.

② Version(协议版本): 占 4 b, 表示当前协议支持的版本, 例如 IPv4/IPv6.

③ Length(地址族长度): 占 8 b, 与 Version 字段相对应. 以 4 B 为单位, 用于表示报文中 IP 地址的长度. 如 IPv4 协议对应的 Length 字段值为 4.

④ Type(报文类型): 占 8 b, 用于标识 CRS 请求报文、授权通告报文、通告确认报文、CRS 响应报文、资源访问请求、探测请求、协同请求和感知报文等.

⑤ TRMAX: 占 4 b, 是任务容忍的最大响应延迟等级.

⑥ Priority(优先级): 占 8 b, 是任务处理的优先级. 当任务处于节点的等待队列中时, 根据任务优先级进行调度处理, 优先级相同则按先来先服务进行处理.

⑦ RCODE(返回码): 占 8 b, 表示响应的差错状态. 当值为 0 时, 表示无差错.

⑧ Seconds elapsed: 占 12 b, 表示用户发送请求后所经历的时间, 通过时间戳来实现.

⑨ Effective time: 占 12 b, 表示节点为用户预留资

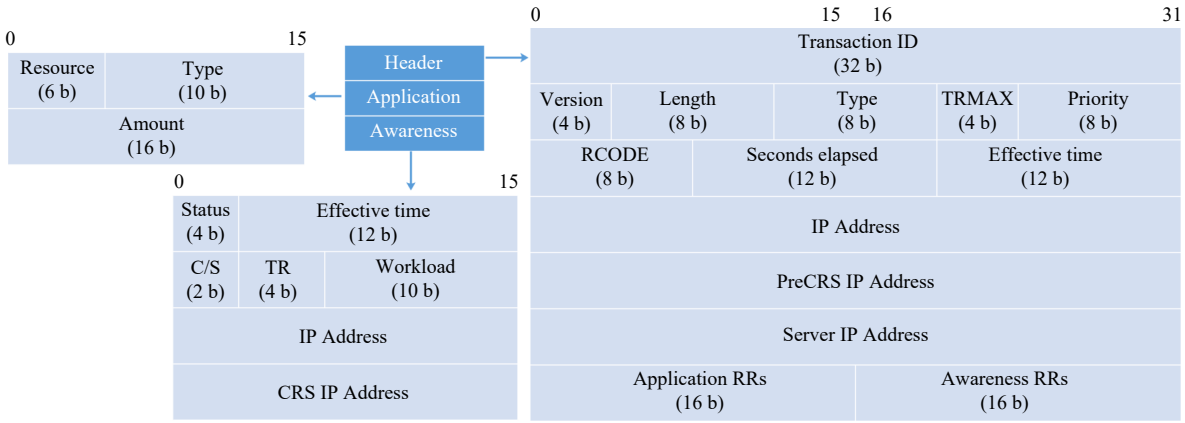


Fig. 7 Message structure of computing resource routing protocol

图 7 算力资源路由协议的报文结构

源的最大有效时间. 任务需要有效时间内将任务挂载至计算节点, 否则节点会自动释放预分配的计算资源.

⑩ IP Address: 占 32 b 或 128 b, 表示用户或节点的 IP 地址. 若为资源请求报文, 则字段表示用户的 IP 地址; 若为探测报文, 则为发出探测报文的计算节点的 IP 地址.

⑪ PreCRS IP Address: 占 32 b 或 128 b, 表示上一个交付的计算资源系统的 IP 地址.

⑫ Server IP Address: 占 32 b 或 128 b, 表示最终为任务提供计算服务的节点 IP 地址.

⑬ Application RRs: 占 16 b, 表示请求资源记录数目. 当值为 0 时, 报文中不存在申请记录部分.

⑭ Awareness RRs: 占 16 b, 表示感知资源记录数目. 当值为 0 时, 报文中不存在感知记录部分.

2)申请记录部分由多条资源申请记录组成, 每条资源申请记录由 4 B 的结构体构成, 记录的数目由头部中的 Application RRs 决定. 扩展的协议可以通过该结构体记录用于任务部署运行的相关环境配置信息. 结构体字段含义如下所示.

① Resource: 占 6 b, 表征申请的资源名称. 例如 CPU, GPU, TPU 等计算资源.

② Type: 占 10 b, 表征申请资源的类型, 与 Resource 共同表征一种具体的计算资源. 例如 Resource 为 0, Type 为 0 可用于表示 Intel® Core™ i7-10510U CPU.

③ Amount: 占 16 b, 表示申请的资源数目.

3)感知记录部分由多条感知记录组成, 每条感知记录由 4~20 B 的结构体构成, 记录的数目由头部中的 Awareness RRs 决定. 当对计算资源系统进行探测时, 该系统会同时携带该辖区系统内其他计算节点的感知信息. 结构体字段含义如下所示.

① Status: 占 4 b, 表示计算节点当前状态. 可用于表示节点的开启、繁忙和关闭等状态. 同时考虑到一些计算节点的特殊要求, 节点不具有共享性, 例如公司、研究机构或军事部分的节点, 可在此设置成相应的状态.

② Effective time: 占 12 b, 表示感知记录的有效时间. 与节点所属水平层次有关, 超过有效时间的感知记录将会被丢弃.

③ C/S: 占 2 b, 标识当前节点是以计算资源系统还是以计算节点的方式被探测. “0”表示计算节点, “1”表示计算资源系统.

④ TR(时延等级): 占 4 b, 表示当前节点与探测节点之间的时延.

⑤ Workload: 占 4 b, 表示节点负载系数.

⑥ IP Address: 占 32b 或 128b, 表示当前节点的 IP 地址.

⑦ CRS IP Address: 占 32b 或 128 b, 表示当前计算节点所属的辖区系统的计算资源系统的 IP 地址.

4.3 扩展性分析

随着移动端设备的快速发展, 催生了许多计算密集和高响应的新型网络服务, 导致早期设立的计算节点(后文称之为过载节点)算力不足以满足其作用域内的用户请求, 此时需要对该区域内的计算节点的算力进行物理上的扩展. 一种方案是在过载节点上增加物理卡的数目以增强当前节点的算力, 其仍然是一个计算节点, 并不影响网络的拓扑, 对于计算资源系统而言不需要做更多的操作. 另一种方案是在靠近过载节点的局部区域新增一个额外的计算节点, 使其融入到算力网络中, 可为其他结点分担一部分的用户请求. 计算节点需要融入到算力网络中, 需要完成以下 2 部分的工作:

1)新增节点需要被同层的计算节点发现, 以融入到其他辖区系统内.

2)新增节点需要自适应地形成以自身为中心的辖区系统, 以维护一系列延迟最小的同层节点与上层节点.

这 2 个工作是通过融合了域内感知功能的算力路由协议来实现的. 其背后的协议会做如下的处理:

1)新增节点向过载节点发送“协同请求”报文(报文结构中的 Type 作为标识).

2)过载节点收到协同请求报文, 将目标节点加入到自身的感知队列中, 同时向新增节点发送响应报文, 响应报文中 Awareness 字段内携带过载节点辖区系统内其他结点的信息.

3)新增节点收到过载节点的响应报文后, 查看响应码, 无误后即实现节点的注册接入, 同时将响应报文内的 Awareness 字段内节点加入自身感知队列中.

其中协议的第 2 点即实现了新增节点被过载节点发现, 由于其被加入到过载节点的域内感知队列内, 其他节点在对过载节点进行探测时(域内/域间感知算法), 会对过载节点辖区系统内的计算节点进行探测, 以此实现了新增节点被其他节点发现.

其中协议的第 2 点和第 3 点, 新增节点将过载节点辖区系统内的节点放入自身感知队列内, 通过对感知队列内节点进行探测, 不断更新域内队列和域间队列内节点, 构建形成以自身为中心的辖区系统. 同时对于扩展性所提算法的收敛性和复杂度, 在前文中做出了推导.

新增节点完成以上 2 个工作即融入到算力网络中，由于局部性原理，该节点作用域与过载节点作用域会有重合的部分，用户发送请求时会被网络进行分流，可达到分担负载的目的，以满足增多的计算任务. 新增节点与过载节点，对方均在以各自为中心的辖区系统内，计算任务可以互相卸载，达到分担负载的目的.

5 实验仿真

在本节中，我们通过数值仿真来评估多层次计算资源系统的性能表现. 首先介绍了在 Ubuntu20.04 下搭建的一个多层次计算资源系统的实验仿真环境；其次，介绍了实验仿真环境参数的设置方法与原则；最后通过对实验结果进行分析，验证了多层次计算资源系统的高效性及其在负载均衡上的优越性.

5.1 多层次计算资源系统实验场景

我们搭建一个 3 层次的计算资源系统，如图 8 所示. 层级由下至上分别表示边缘层、雾层和云层. 边缘层为第 1 层，云层为最高层，即第 3 层. 边缘层由 6 个计算节点组成，其中 Edge Server11, Edge Server12, Edge Server13 互相在以各自为中心的辖区系统内，定义为 J_1；它们维护的最佳上层节点为雾层的 Fog Server31, Fog Server32；Edge Server21, Edge Server22, Edge Server23 互相在以各自为中心的辖区系统内，定义为 J_2；它们维护的最佳上层节点为雾层的 Fog

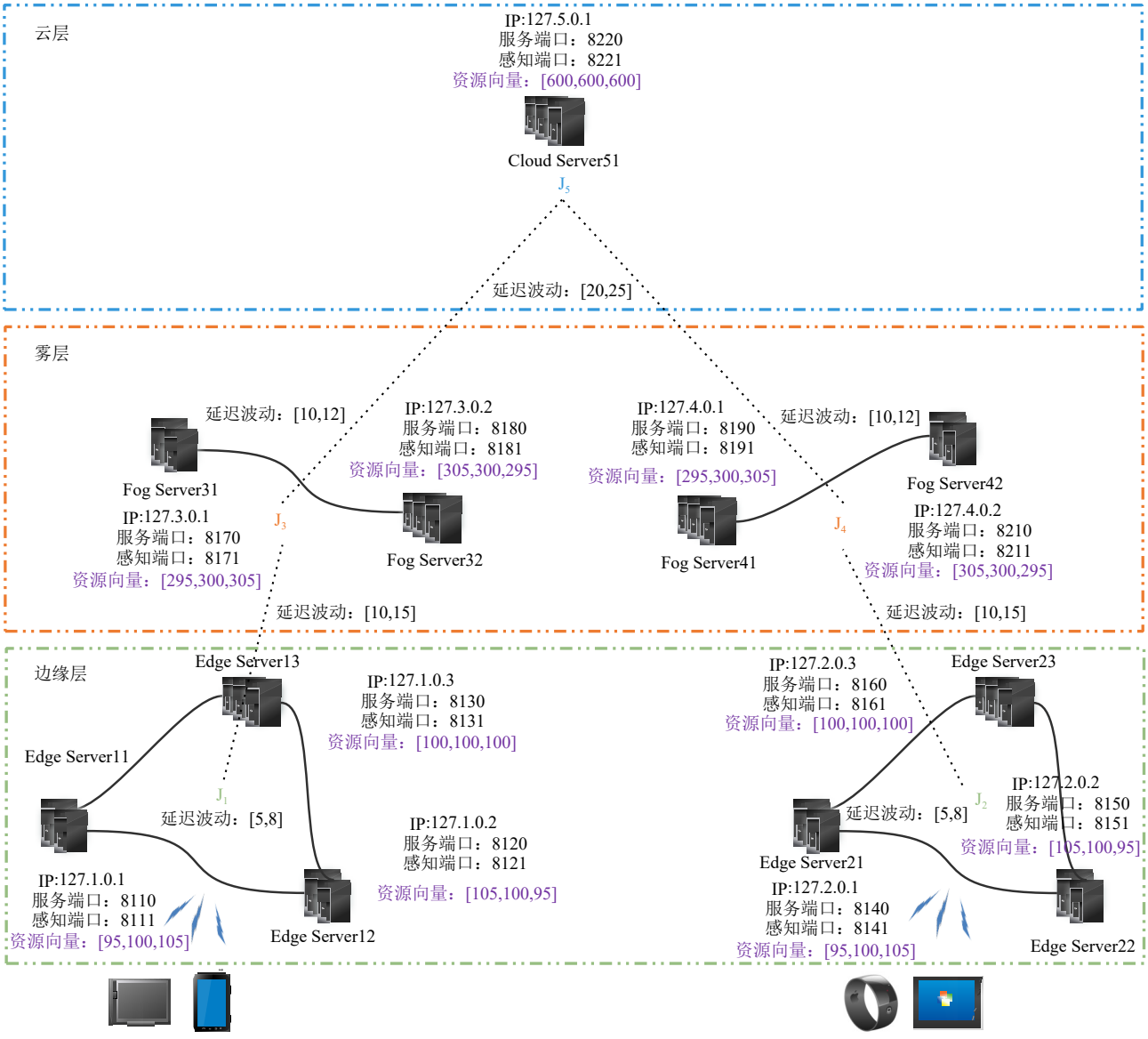


Fig. 8 Experimental scene : Three-level computing resource system

图 8 实验场景：3 层次计算资源系统

Server41, Fog Server42. 雾层由 4 个计算节点构成，其中 Fog Server31, Fog Server32 互相在以各自为中心的辖区系统内，定义为 J_3；它们维护的最佳上层节点为云层的 Cloud Server51；Fog Server41, Fog Server42 互相在以各自为中心的辖区系统内，定义为 J_4；它们维护的最佳上层节点为云层的 Cloud Server51. 云层由 1 个计算节点 Cloud Server51 组成，其独自构成一个辖区系统，定义为 J_5.

用户在边缘层的计算节点的作用域内，即用户通过有线或者无线链路首先将任务交付边缘层的计算节点进行调度处理.

为了评估算法性能，本文将所提算法与其他 2 种计算卸载算法进行对比.

1)随机卸载(Random). 用户随机地决策将任务卸载至某个计算节点进行处理. 为了优化这个过程，用户只会将任务随机挂载至相关节点上，不会将其挂载至不可达节点处，即辖区系统 J_1 侧的用户只会随机挂载至辖区系统 J_1, J_3, J_5 内的计算节点处.

2)最近卸载(Nearest). 该算法根据任务所需计算资源，将任务卸载至符合卸载条件且最靠近用户的计算节点处.

对于计算节点主要实现如下 3 个接口：

1)节点状态感知接口. 用于实现辖区系统内计算节点状态的互感知. 开启接收其他节点状态信息的监听端口，将接收到的记录用于更新的节点状态. 开启发送自身节点状态信息的同步端口，设置同步频率用于向其他节点发送自身状态信息.

2)节点的任务调度接口. 计算任务通过分段路由逐层被调度. 中间节点需要具有对任务进行调度的功能方法，分发的逻辑是基于贪心策略的资源路由算法，同时对于额外的对比算法也需做出相对应的实现逻辑.

3)任务挂载接口. 任务被调度至“合适节点”后，堵塞当前进程，并持续占有节点为任务分配的计算资源，最后任务计算时间结束时释放占有的计算资源，以此来模拟计算任务的计算场景.

在边缘侧，为每个边缘节点实现一个任务构造器. 根据任务特性在边缘端源源不断地生成计算任务，并发往边缘节点进行调度；其所需的主要参数有计算任务数、发送频率、任务资源需求和延迟响应需求. 边缘层发送的任务信息被封装于报文结构体内，同时对于每个任务会随机地产生一个 ID 作为标识. 任务产生器的 IP 作为用户的 IP 地址，完成交付后报文中的服务器 IP 才会被固定.

现实场景中会涉及到动态的辖区系统的实现，其需要对计算节点设置辖区系统的延迟阈值或计算数目做限制，每次在获取到节点状态信息时，对辖区系统内的节点做一个对比和淘汰，以此来维护基于实时变化的网络波动给辖区系统带来的影响.

5.2 参数设置

实验通过 IP 来标识不同的计算节点. 为了在一台设备上完成实验的仿真而防止端口的冲突，我们为不同的计算节点分别设置不同的服务端口号和感知端口号. 我们使用 CPU, GPU 和 TPU 数目的资源向量来表征计算节点的算力和任务需求，为不同的计算节点设置不同的资源向量来表示异构的计算资源，同时根据资源特征，为云层计算分配的资源多于雾层节点的计算资源，雾层节点的计算资源多于边缘层节点的计算资源. 如表 1 所示，给出了计算节点的参数设置，其中资源向量为一个(CPU, GPU, TPU)的资源向量.

Table 1 Computing Node Parameter Setting

表 1 计算节点参数设置

节点	IP	服务端口	感知端口	资源向量（CPU，GPU，TPU）
Server11	127.1.0.1	8110	8111	(95,100,105)
Server12	127.1.0.2	8120	8121	(105,100,95)
Server13	127.1.0.3	8130	8131	(100,100,100)
Server21	127.2.0.1	8140	8141	(95,100,105)
Server22	127.2.0.2	8150	8151	(105,100,95)
Server23	127.2.0.3	8160	8161	(100,100,100)
Server31	127.3.0.1	8170	8171	(295,300,305)
Server32	127.3.0.2	8180	8181	(305,300,295)
Server41	127.4.0.1	8190	8191	(295,300,305)
Server42	127.4.0.2	8210	8211	(305,300,295)
Server51	127.5.0.1	8220	8221	(600,600,600)

为表示动态的网络负载，将链路的延迟设置为一个随机的波动区间，同时不同层级节点之间的通信延迟应该大于下层节点之间的通信延迟，如表 2

Table 2 Delay Fluctuations Between Jurisdictional Systems

表 2 辖区系统之间的延迟波动 s

辖区系统	J_1	J_2	J_3	J_4	J_5
J_1	[5, 8]	+∞	[10, 15]	+∞	+∞
J_2	+∞	[5, 8]	+∞	[10, 15]	+∞
J_3	[10, 15]	+∞	[10, 12]	+∞	[20, 25]
J_4	+∞	[10, 15]	+∞	[10, 12]	[20, 25]
J_5	+∞	+∞	[20, 25]	[20, 25]	+∞

的 2 维矩阵表示不同辖区系统内节点之间的通信延迟, 节点与自身无通信开销.

通过 CPU、GPU、TPU、最大响应延迟和计算时间的联合向量表示多样化的用户需求, 同时任务的计算时间设置值应远大于任务最大响应延迟, CPU 数目、GPU 数目和 TPU 数目设置为 [0, 2] 内的随机值, 任务计算时间设置为 [800, 1 000] 内的随机值; 为了模拟现实场景不同地区任务类型的差异, 最大响应延迟分 4 组分别设置为 [5, 55], [5, 45], [5, 35], [5, 25] 内的随机值, 场景内高响应任务占比依次增多.

为了体现本文方法的鲁棒性, 在边缘侧营造不同的实验环境. 单次实验用户任务总量设置为 N 个, 其中 $\frac{N}{2}$ 个任务由辖区系统 J_1 侧产生, 剩余 $\frac{N}{2}$ 任务由辖区系统 J_2 侧产生. 在辖区系统 J_1 侧, 任务分别以均衡的频率发送至 Edge Server11, Edge Server12, Edge Server13 进行调度, 即 Edge Server11, Edge Server12, Edge Server13 的任务量比例为 1∶1∶1; 而在辖区系统 J_2 侧, 任务以不均衡的频率发送至 Edge Server21, Edge Server22, Edge Server23, 设置 Edge Server21, Edge Server22, Edge Server23 的任务量比例为 1∶2∶3.

与此同时, 为了实现更好的仿真, 我们分别对云、雾和边的资源比例为 1∶2∶4, 1∶3∶9, 1∶5∶25, 1∶7∶49 的实验添加了补充实验. 其中 4 组实验的边的总资源设置为 100, 为此对于边端产生的计算任务数目与计算任务响应延迟比例做了相应的调整.

5.3 实验结果与分析

如图 9 所示, 展示了不同算法的任务调度成功率与用户任务数量的关系, 任务响应的随机区间设置为 [5, 35], 每组分别进行 10 次独立重复实验. 随着用户任务数量的增多, Nearest 和 Random 算法都呈现下降的趋势, 而 GBRA 算法保持一个稳定的较高的调度成功率; 当任务数量达到 1 800 时, 出现了节点过载问题, 但 GBRA 仍可以维持一个较高的调度成功率.

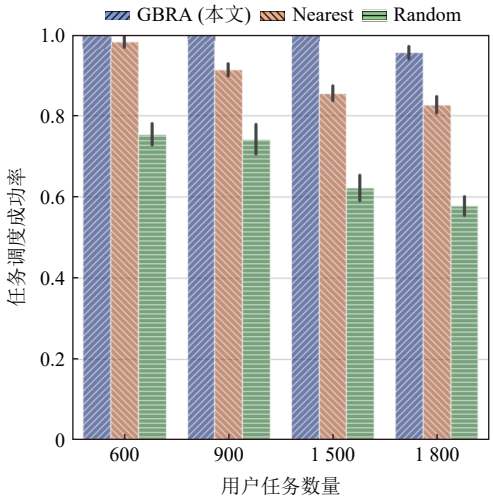


Fig. 9 Impact of the number of users' tasks on scheduling success rate of algorithms

图 9 用户任务数量对算法调度成功率的影响

如图 10 所示为不同试验场景下, GBRA, Nearest, Random 算法任务调度成功率对比图. 组 1、组 2、组 3 和组 4 的任务响应时延随机区间分别为 [5, 55], [5, 45], [5, 35], [5, 25], 每组分别进行 10 次独立重复实验. 从 4 组实验数据来看, GBRA 的调度效果远高于 Nearest 算法和 Random 算法, 并始终维持较高的调度成功率. 在组 1 实验中, 高响应任务占比相对较少, 整体对于调度的安排要求性不高, 只需要将任务调度至一个可完成响应要求的计算节点处, GBRA 和 Nearest 算法均可达到较高的成功率, Random 算法没有考虑任务的需求进行随机卸载, 故成功率不高. 随着高响应任务的占比增多, 可以发现 Nearest 算法和 Random 算法的任务调度成功率都呈现明显的下降趋势, 而 GBRA 算法会将特定的任务调度至合适的层上, 在资源足够的前提下, 其调度效率并不会受到任务占比的影响, 仍然可以保持一个较高的调度成功率. 组 4 实验中高响应任务占比进一步增大, 边缘层节点出现过载, 3 种算法的性能呈现一定程度的下降, 但 GBRA 仍然可以达到较高的调度成功率.

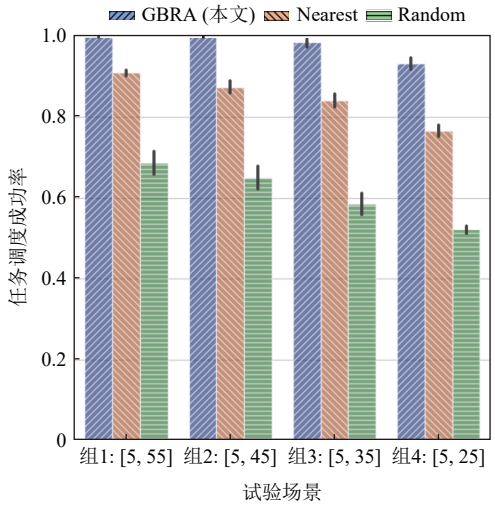


Fig. 10 Impact of task types on scheduling success rate of algorithms

图 10 任务类型占比对算法调度成功率的影响

图 11 为不同的云边端资源比例下的任务调度成功率的结果图. 从图中可以看出算法具有一定的泛化性, 在不同的云边端资源差异环境下都可以表现出较好的效果. 对于延迟要求不同的计算任务会被

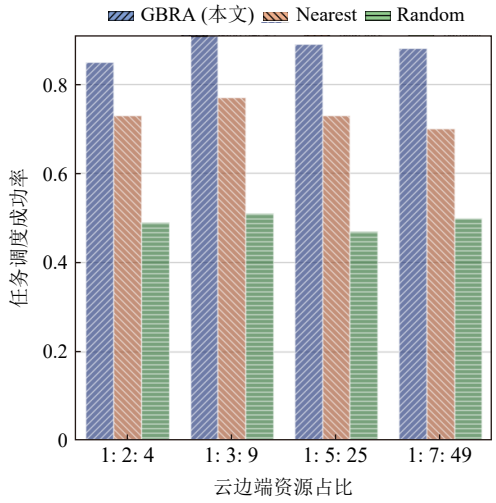


Fig. 11 Proportion of cloud-edge resources and the success rate of task scheduling

图 11 云边端资源占比与任务调度成功率

逐层分段调度至不同的层中.

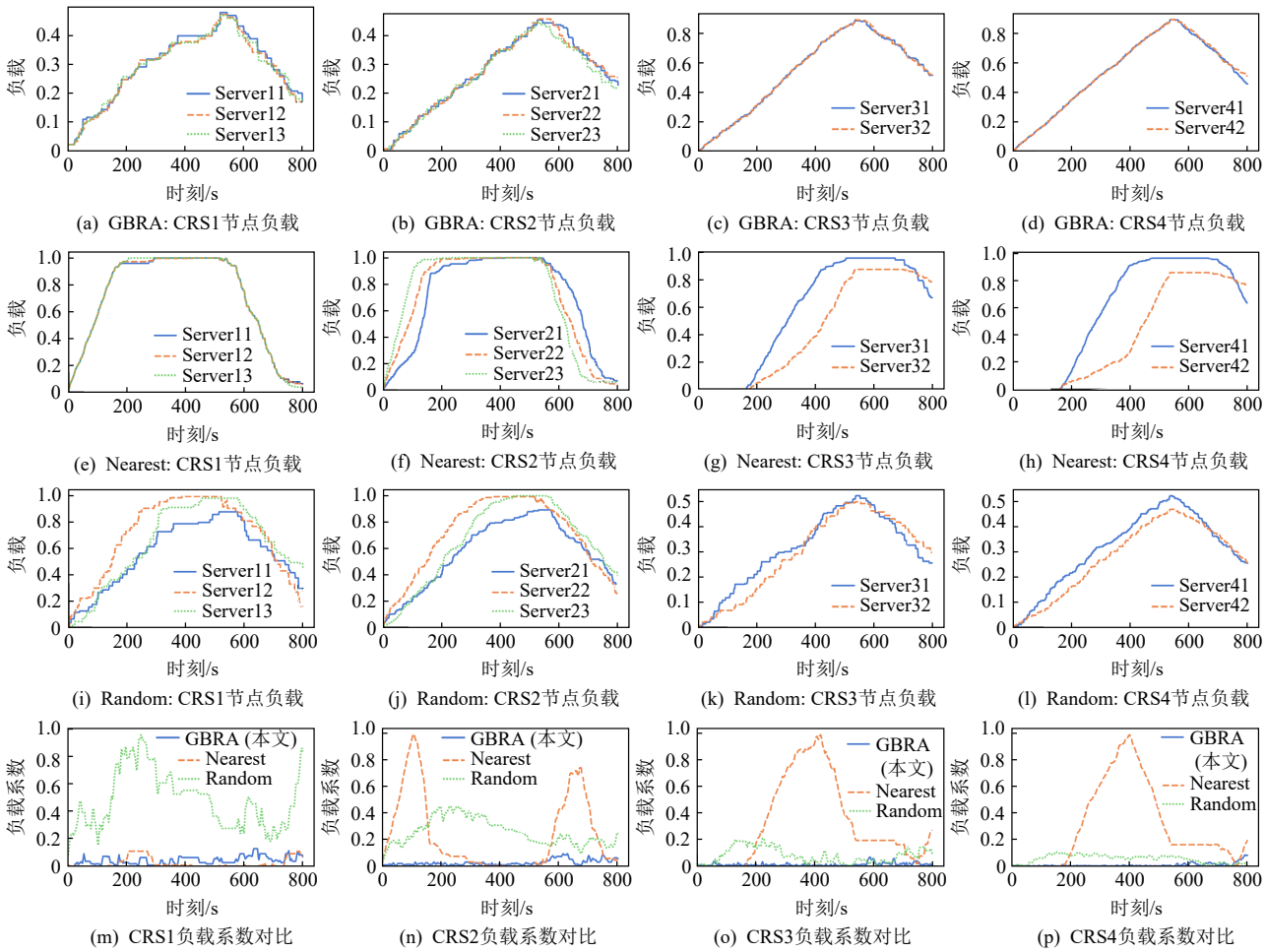
我们选取任务量为 1 800, 任务响应时延随机区间为 [5, 35] 的实验数据, 展示了不同算法下计算资源系统内节点负载均衡表现的对比图, 如图 12 所示.

我们定义计算节点i的负载系数为:

$$w_i = \frac{c_{\mathrm{CPU}} + c_{\mathrm{GPU}} + c_{\mathrm{TPU}}}{N_{\mathrm{CPU}} + N_{\mathrm{GPU}} + N_{\mathrm{TPU}}},$$

其中c_{CPU}, c_{GPU}, c_{TPU}分别代表已分配的资源数目, N_{CPU}, N_{GPU}, N_{TPU}分别代表节点初始的资源数目. 同时我们使用极差来描述辖区系统内计算节点的负载均衡系数, 如式(5)所示. 为了更加直观地观察不同算法之间的效果, 我们将极差映射到 [0, 1] 区间.

图 12(a)~(d)为 GBRA 算法下计算资源系统 CRS1, CRS2, CRS3, CRS4 各节点的负载变化; 图 12(e)~(h)为 Nearest 算法下计算资源系统 CRS1, CRS2, CRS3, CRS4 各节点的负载变化; 图 12(i)~(l)为 Random 算法下计算资源系统 CRS1, CRS2, CRS3, CRS4 各节点的负载变化; 如图 12(m)~(p)所示, GBRA 算法下计算资源系统内节点之间的极差始终维持在一个很低的水平. 同时注意到在 GBRA 算法下计算节点始终未到达满负载状态, 而 Nearest 和 Random 都一定程度上出现了节点过载现象, 如 Nearest 算法下 CRS1



(a) GBRA: CRS1节点负载 (b) GBRA: CRS2节点负载 (c) GBRA: CRS3节点负载 (d) GBRA: CRS4节点负载

(e) Nearest: CRS1节点负载 (f) Nearest: CRS2节点负载 (g) Nearest: CRS3节点负载 (h) Nearest: CRS4节点负载

(i) Random: CRS1节点负载 (j) Random: CRS2节点负载 (k) Random: CRS3节点负载 (l) Random: CRS4节点负载

(m) CRS1负载系数对比 (n) CRS2负载系数对比 (o) CRS3负载系数对比 (p) CRS4负载系数对比

Fig. 12 Node workload balancing performance within the computing resource system under different algorithms

图 12 不同算法下计算资源系统内节点负载均衡表现

和 CRS2 在 [200, 500] 区间内, 其使得计算资源系统达到一种被动的均衡现象. 如图 12(m)~(p)所示, 对于同一个算法的 CRS1(均衡的发送频率)和 CRS2(不均衡的发送频率)2 个对照区域的极差变化发现, 在 GBRA 算法下同一辖区内计算节点之间可以相互卸载, CRS1 和 CRS2 的极差变化情况相差不大; 对于 Nearest 算法, 在 CRS1 内极差波动不大且维持在一个较低的区间, 而在 CRS2 内极差出现较大波动, 发送频率较大的区域的计算节点会率先达到满负载状态, 二者的极差变换情况相差较大; 对于 Random 算法, 其卸载具有随机性, 二者的极差变化情况相差较大. 总体而言, GBRA 算法在负载均衡上具有较好的表现, 辖区系统内的计算节点可以互相卸载以动态调整卸载策略, 对不同区域任务的发送频率不敏感, 具有较好的稳定性.

6 结 论

本文设计了一套云雾混合的多层次计算资源系统, 该系统是建立在当前 TCP/IP 协议栈之上的算力网络技术方案. 首先, 根据不同计算节点对于共享性的要求不同, 创新性地将算力网络在水平和垂直 2 个维度上进行划分. 水平层面划分为边缘层、多级雾层和云层. 垂直层面引入辖区系统的概念, 将每一层进一步划分为多个不同的辖区系统. 其次, 为实现算网感知, 设计并实现了算网感知策略. 感知策略定义了域内感知规则和域间感知规则. 基于最大化任务完成数和计算的负载均衡策略, 创新性地提出了一种基于贪心的资源路由算法, 可为每个用户任务生成不同的感知搜索树; 并在此基础上定义了算力资源路由协议以及其报文结构, 以此来完成资源的申请与调配工作. 最后, 我们搭建了一个 3 层次的计算资源系统, 通过大量的仿真实验验证了系统的高效性及其在负载均衡上的优越性.

作者贡献声明: 徐家豪提出了算法思路、实验方案, 完成实验并撰写论文; 余辰、李健和金海提出指导意见并修改论文.

参 考 文 献

[1] Jin Liang, Lou Yangming, Sun Xiaoli, et al. Concept and vision of 6G wireless endogenous safety and security[J]. SCIENTIA SINICS Informationis, 2023, 53(2): 344−364 (in Chinese)
(金梁, 楼洋明, 孙小丽, 等. 6G 无线内生安全理念与构想[J]. 中国科学: 信息科学, 2023, 53(2): 344−364)

[2] Zhao Jianjin, Li Qi, Liu Shengli, et al. Towards traffic supervision in 6G: A graph neural network-based encrypted malicious traffic detection method[J]. SCIENTIA SINICS Informationis, 2022, 52(2): 270−286 (in Chinese)
(赵键锦, 李祺, 刘胜利, 等. 面向 6G 流量监控: 基于图神经网络的加密恶意流量检测方法[J]. 中国科学: 信息科学, 2022, 52(2): 270−286)

[3] Luo Quyuan, Li Changle, Luan H, et al. Collaborative data scheduling for vehicular edge computing via deep reinforcement learning[J]. IEEE Internet of Things Journal, 2020, 7(10): 9637−9650

[4] Aung N, Dhelim S, Chen Liming, et al. Edge-enabled metaverse: The convergence of metaverse and mobile edge computing[J]. Tsinghua Science and Technology, 2024, 29(3): 795−805

[5] Thirumalaisamy M, Basheer S, Selvarajan S, et al. Interaction of secure cloud network and crowd computing for smart city data obfuscation[J]. Sensors, 2022, 22(19): 7169. https://doi.org/10.3390/s22197169

[6] Zhang Hongke, Cheng Yujun, Yang Dong. State of the art of industrial network technologies: A review and outlook[J]. Chinese Journal on Internet of Things, 2017, 1(1): 13−20 (in Chinese)
(张宏科, 程煜钧, 杨冬. 工业网络技术现状与展望[J]. 物联网学报, 2017, 1(1): 13−20)

[7] Tang Xuhao, Liu Fagui, Wang Bin, et al. Survey on task scheduling in inter-cloud environment[J]. Journal of Computer Research and Development, 2023, 60(6): 1262−1275 (in Chinese)
(唐续豪, 刘发贵, 王彬, 等. 跨云环境下任务调度综述[J]. 计算机研究与发展, 2023, 60(6): 1262−1275)

[8] Hu Pengfei, Dhelim S, Ning Huansheng, et al. Survey on fog computing: Architecture, key technologies, applications and open issues[J]. Journal of Network and Computer Applications, 2017, 98: 27−42

[9] Ding Yan, Li Kenli, Liu Chubo, et al. A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(6): 1503−1519

[10] Wang Ling, Wu Chuge, Fan Wenhui. A survey of edge computing resource allocation and task scheduling optimization[J]. Journal of System Simulation, 2021, 33(3): 509−520 (in Chinese)
(王凌, 吴楚格, 范文慧. 边缘计算资源分配与任务调度优化综述[J]. 系统仿真学报, 2021, 33(3): 509−520)

[11] National Development and Reform Commission. East data and west computing[EB/OL]. 2022[2024-10-09]. https://www.ndrc.gov.cn/xwdt/ztzl/dsxs/(in Chinese)
(中华人民共和国国家发展和改革委员会. 东数西算[EB/OL]. 2022[2024-10-09]. https://www.ndrc.gov.cn/xwdt/ztzl/dsxs/)

[12] Jamil B, Ijaz H, Shojafar M, et al. Resource allocation and task scheduling in fog computing and Internet of everything environments: A taxonomy, review and future directions[J]. ACM Computing

Surveys, 2022, 54: 1−38

[13] Telecommunication Standardization Sector of ITU. Framework and Architecture of Computing Power Network[S/OL]. 2021[2024-10-09]. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2501-202109-I!!PDF-E&type=items

[14] Zhou Xu, Li Zhuo. Cloud-edge-end collaborative scheduling technology for computing power network[J]. ZTE Technology Journal, 2023, 29(4): 32−37 (in Chinese)
(周旭, 李琢. 面向算力网络的云边端协同调度技术[J]. 中兴通讯技术, 2023, 29(4): 32−37)

[15] Hao Hao, Xu Changqiao, Zhong Lujie, et al. A multi-update deep reinforcement learning algorithm for edge computing service Offloading[C]// Proc of the 28th ACM Int Conf on Multimedia. New York: ACM, 2020: 3256−3264

[16] Geng L, Willis P. Compute First Networking (CFN) Scenarios and Requirements[S/OL]. IETF RTGWG Working Group, 2019[2024-10-09]. https://datatracker.ietf.org/doc/html/draft-geng-rtgwg-cfn-req-00

[17] Liu Bing, Mao Jianwei, Xu Ling, et al. CFN-dyncast: Load balancing the edges via the network[C/OL]// Proc of IEEE Wireless Communications and Networking Conf Workshops (WCNCW). Piscataway, NJ: IEEE, 2021 [2024-10-09]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9420028

[18] China Mobile, HuaWei. Computing-aware networking whitepaper [R/OL]. 2019[2024-10-09]. https: //www. digitalelite. cn/h-nd-1105.html (in Chinese)
(中国移动, 华为. 算力感知网络技术白皮书[R/OL]. 2019 [2024-10-09]. https: //www.digitalelite.cn/h-nd-1105.html)

[19] China Mobile. Computing force network whitepaper[R/OL]. 2021 [2024-10-09]. http://www.ecconsortium.org/Uploads/file/20211108/1636352251472904.pdf (in Chinese)
(中国移动. 算力网络白皮书[R/OL]. 2021[2024-10-09]. http://www.ecconsortium.org/Uploads/file/20211108/1636352251472904.pdf)

[20] Yi Xinxin, Ma Herong, Cao Chang, et al. Analysis and discussion of routing strategy for programmable services in computing power network[J]. Frontiers of Data and Computing, 2022, 4(5): 23−32 (in Chinese)
(易昕昕, 马贺荣, 曹畅, 等. 算力网络可编程服务路由策略的分析与探讨[J]. 数据与计算发展前沿, 2022, 4(5): 23−32)

[21] Lei Bo, Liu Zengyi, Wang Xuliang, et al. Computing network: A new multi-access edge computing[J]. Telecommunications Science, 2019, 35(9): 44−51 (in Chinese)
(雷波, 刘增义, 王旭亮, 等. 基于云、网、边融合的边缘计算新方案: 算力网络[J]. 电信科学, 2019, 35(9): 44−51)

[22] Yao Huijuan, Lu Lu, Duan Xiaodong. Architecture and key technologies for computing-aware networking[J]. ZTE Technology Journal, 2021, 27(3): 7−11 (in Chinese)
(姚惠娟, 陆璐, 段晓东. 算力感知网络架构与关键技术[J]. 中兴通讯技术, 2021, 27(3): 7−11)

[23] Hu Yujiao, Jia Qingmin, Sun Qingshuang, et al. Functional architecture to intelligent computing power network[J]. Computer Science, 2022, 49(9): 249−259 (in Chinese)
(胡玉姣, 贾庆民, 孙庆爽, 等. 融智算力网络及其功能架构[J]. 计算机科学, 2022, 49(9): 249−259)

[24] Zhao Qianying, Xing Wenjuan, Lei Bo, et al. A solution of computing power network based on domain name resolution[J]. Telecommunications Science, 2021, 37(10): 86−92 (in Chinese)
(赵倩颖, 邢文娟, 雷波, 等. 一种基于域名解析机制的算力网络实现方案[J]. 电信科学, 2021, 37(10): 86−92)

[25] Zhou Gefan, Lei Bo. Computing service demand matching based on computing power identification in computing power network[J]. Frontiers of Data and Computing, 2022, 4(6): 20−28 (in Chinese)
(周舸帆, 雷波. 算力网络中基于算力标识的算力服务需求匹配[J]. 数据与计算发展前沿, 2022, 4(6): 20−28)

[26] Chen Xingyan, Zhang Xuesong, Xie Zhilong, 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)
(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)

[27] Kuang Zhufang, Chen Qinglin, Li Linfeng, et al. Multi-user edge computing task offloading scheduling and resource allocation based on deep reinforcement learning[J]. Chinese Journal of Computers, 2022, 45(4): 812−824 (in Chinese)
(邝祝芳, 陈清林, 李林峰, 等. 基于深度强化学习的多用户边缘计算任务卸载调度与资源分配算法[J]. 计算机学报, 2022, 45(4): 812−824)

[28] Sun Yukun, Zhang Xing, Lei Bo. Study on intelligent computing aware route allocation policy in edge computing-aware networks[J]. Radio Communications Technology, 2022, 48(1): 60−67 (in Chinese)
(孙钰坤, 张兴, 雷波. 边缘算力网络中智能算力感知路由分配策略研究[J]. 无线电通信技术, 2022, 48(1): 60−67)

[29] Sahni Y, Cao Jiannong, Lei Yang, et al. Multihop offloading of multiple DAG tasks in collaborative edge computing[J]. IEEE Internet of Things Journal, 2020, 8(6): 4893−4905

[30] Jiang Yulong, Dong Fang, Guo Xiaolin, et al. Potential game based workflow task offloading optimization mechanism in computing power network[J]. Journal of Computer Research and Development, 2023, 60(4): 797−809 (in Chinese)
(姜玉龙, 东方, 郭晓琳, 等. 算力网络环境下基于势博弈的工作流任务卸载优化机制[J]. 计算机研究与发展, 2023, 60(4): 797−809)

[31] Xiao Han, Xu Changqiao, Ma Yunxiao, et al. Edge intelligence: A computational task offloading scheme for dependent IoT application[J]. IEEE Transactions on Wireless Communications, 2022, 21(9): 7222−7237

[32] Zhong Lujie, Wang Mu. Blockchain-enpowered cooperative resource allocation scheme for computing first network[J]. Journal of Computer Research and Development, 2023, 60(4): 750−762 (in Chinese)
(衷璐洁, 王目. 区块链赋能的算力网络协同资源调度方法[J]. 计算机研究与发展, 2023, 60(4): 750−762)

[33] Xiao Han, Zhuang Yirong, Xu Changqiao, et al. Transcoding-enabled cloud-edge-terminal collaborative video caching in heterogeneous IoT networks: An online learning approach with time-varying information[J], IEEE Internet of Things Journal, 2024, 11(1): 296−310

[34] Liu Zening, Li Kai, Wu Liantao, et al. CATS: Cost aware task scheduling in multi-tier computing networks[J]. Journal of Computer Research and Development, 2020, 57(9): 1810−1822 (in Chinese)
(刘泽宁, 李凯, 吴连涛, 等. 多层次算力网络中代价感知任务调度算法[J]. 计算机研究与发展, 2020, 57(9): 1810−1822)

[35] Gong Chenyu, Shu Hongfeng, Zhang Xin. Centralized unsplittable task scheduling algorithm for multi-tier computing power network[J]. ZTE Technology Journal, 2021, 27(3): 35−41 (in Chinese)
(巩宸宇, 舒洪峰, 张昕. 多层次算力网络集中式不可分割任务调度算法[J]. 中兴通讯技术, 2021, 27(3): 35−41)

[36] Zhang Houhao, Li Hanlin, Gao Lin. Hierarchical resource deployment and sharing strategy in mobile edge computing[J]. Chinese Journal on Internet of Things, 2021, 5(1): 11−18 (in Chinese)
(张厚浩, 李晗琳, 高林. 移动边缘计算中的分层资源部署与共享策略[J]. 物联网学报, 2021, 5(1): 11−18)



Xu Jiahao, born in 2000. Master. His main research interests include computing first network and edge computing.
徐家豪, 2000 年生. 硕士. 主要研究生方向为算力网络、边缘计算.



Yu Chen, born in 1976. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include edge AI, edge computing, big data analysis, and mobile computing.
余 辰, 1976 年生. 博士. 教授, 博士生导师. CCF 高级会员. 主要研究方向为边缘智能、边缘计算、大数据分析、移动计算.



Li Jian, born in 1978. PhD, professor. Senior member of CCF. His main research interests include cloud computing, virtualization, and high performance networking.
李 健, 1978 年生. 博士. 教授. CCF 高级会员. 主要研究方向为云计算、虚拟化、高性能网络.



Jin Hai, born in 1966. PhD, professor, PhD supervisor. Fellow of CCF, fellow of IEEE. His main research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security.
金 海, 1966 年生. 博士, 教授, 博士生导师. CCF 会士, IEEE 会士. 主要研究方向为计算机体系结构、并行和分布式计算、大数据处理、数据存储、系统安全.