算力网络中面向计算重用的任务调度优化

马云霄[1]，吴忠辉[1]，徐祖云[1]，衷璐洁[2]，许长桥[1]
（1. 北京邮电大学网络与交换技术国家重点实验室，北京 100876； 2. 首都师范大学信息工程学院，北京 100048）

摘 要：为应对未来算力需求爆炸性增长所带来的挑战，将计算重用技术引入算力网络中，通过重用计算任务结果，来缩短服务时延并减少计算资源消耗。在此基础上，提出基于服务联盟的上下文感知在线学习算法。首先，设计重用指数来减少额外查找时延；然后，基于服务联盟机制进行在线学习，根据上下文信息及历史经验做出计算任务调度决策。仿真实验结果表明，所提算法在服务时延、计算资源消耗等方面均优于基准算法。

关键词：算力网络；计算重用；任务调度；在线学习
中图分类号：TP393
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2023206

Optimization of task scheduling for computing reuse in computing power network

MA Yunxiao[1], WU Zhonghui[1], XU Zuyun[1], ZHONG Lujie[2], XU Changqiao[1]
1. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
2. Information Engineering College, Capital Normal University, Beijing 100048, China

Abstract: To cope with the challenges posed by the future explosive growth in computing power demand, computing reuse technology was introduced into the computing power network to reduce service latency and computational resource consumption by reusing the results of computational tasks. Based on this, a service federation-based context-aware online learning algorithm was proposed. First, the reuse index was designed to reduce the extra lookup latency. Then, online learning was performed based on the service federation mechanism to make computational task scheduling decisions according to contextual information and historical experience. The experimental results show that the proposed algorithm outperforms the baseline algorithms in terms of service latency and computational resource consumption.

Keywords: computing power network, computing reuse, task scheduling, online learning

0 引言

随着 5G 的普及，元宇宙、超高清视频、自动驾驶等新型应用层出不穷，同时，在人工智能、区块链等技术的推动下，新兴业务对计算能力的需求激增。在此背景下，算力网络[1]作为一种新型资源整合方案应运而生，它结合了云计算、边缘计算以及智能控制等技术的优势，通过网络连接起泛在、分散的计算资源，推动计算与网络资源的深度融合，提高算力资源利用率。

尽管通过整合泛在的计算资源，算力网络能够缓解当前算力紧张的难题，但却难以应对未来算力资源需求的迅猛增长势头。算力资源来源于算力网络基础设施的部署，而基础设施数量不会无限制地增加。同时，大规模的计算部署带来了巨大的能源消耗，算力需求的爆炸性增长给绿色低碳的发展带

收稿日期：2023−04−19；**修回日期：**2023−09−19
通信作者：许长桥，cqxu@bupt.edu.cn
基金项目：国家自然科学基金资助项目（No.62225105）；北京邮电大学博士生创新基金资助项目（No.CX2021108）
Foundation Items: The National Natural Science Foundation of China (No.62225105), The BUPT Excellent Ph.D. Students Foundation (No.CX2021108)

来了挑战。因此，面对日益增长的算力需求，必须找到可持续发展的绿色计算道路。研究发现，算力网络中的部分计算任务是冗余的、重复执行的。例如，在智能监控系统的人脸识别服务[2]中，若多个摄像机拍摄到了相同区域，重叠区域的人脸识别即重复执行的计算任务。而在当前的算力网络架构下，若多个用户发出相同的计算任务请求，将会为不同用户分别调度算力节点来提供计算服务，这导致大量计算任务的重复执行，造成算力资源的浪费。为了降低资源消耗，有学者提出计算节约网络[3]，旨在利用最少的计算和网络资源来提供计算服务。计算节约网络的核心思想是进行计算重用，即在提供计算服务时重用已经执行的计算任务结果，这可以显著降低资源消耗，减少计算任务完成时间。因此，对于计算资源需求越来越大的计算密集型应用，计算重用将会发挥巨大优势。

然而，在算力网络中，如何利用计算重用的优势来降低计算服务时延、减少计算资源消耗还缺乏理论研究。因此，本文重点研究了在计算可重用算力网络下的计算任务调度问题。考虑到计算重用以及算力网络的特性，进行计算任务调度需要解决以下 3 个难题。1) 算力节点包含网络中泛在的计算设备，其地理位置分散，且具有较强异构性。因此，如何选择合适的节点来快速完成计算任务并传输计算结果？2) 由于引入了计算重用技术，部分算力节点中缓存了可重用的计算结果，但却不可预知。因此，如何为不同计算服务选择算力节点，以尽可能提高计算重用概率并降低计算服务时延？3) 在算力节点上执行重用计算前需要查找已缓存的可重用数据，若没有可重用数据导致不可进行计算重用，则会产生额外查找时延。因此，如何尽可能地减少额外查找时延？

为了解决以上难题，本文提出了一种基于服务联盟的上下文感知在线学习（SF-COL, service federation based context-aware online learning）算法，在实现低时延计算服务的同时降低了计算资源消耗。本文的主要贡献包括 3 个方面。

1) 本文将计算重用技术引入算力网络中，提出了计算可重用的算力网络模型。在综合考虑计算、查找以及传输时延的基础上，构建了一个以优化服务时延为目标的计算任务调度问题。

2) 本文提出了 SF-COL 算法来解决计算可重用算力网络下的计算任务调度问题。首先，维护一个历史查找表（HST）并设计重用指数，以辅助任务调度决策来减少额外查找时延。进一步地，基于服务联盟机制进行高效学习，探索不同上下文信息下算力节点的表现情况，在历史经验的基础上做出计算任务调度决策。

3) 本文开展了一系列的仿真实验来验证所提方案的性能。实验结果证明，在服务时延和计算资源消耗等方面，所提 SF-COL 算法均优于基准算法。

1 相关工作

1.1 算力网络

算力网络自提出以来，一直备受各界人士关注。在国际标准方面，互联网工程任务组（IETF，Internet Engineering Task Force）于 2019 年 2 月正式成立了网络计算研究工作组，致力于研究计算与网络的深度融合，并利用这一新兴的技术改善网络服务性能以及提高用户体验。

产业界对算力网络也尤其关注。2019 年 11 月，中国联通发布了《中国联通算力网络白皮书》[4]，认为在面向未来大数据量高智能化的数字世界背景下，亟须打造新时代的算力网络。同时，各大企业也加入算力网络相关的研究中。例如，华为和国家超级计算济南中心于 2021 年联合发布《算力互联网技术白皮书》[5]，指出算力互联网有望加速未来的计算模式转变，推动计算性能的指数级增长。

近年来，学术界也出现了很多对于算力网络的研究[1,6-10]。贾庆民等[1]针对时间敏感计算密集的业务形态，对确定性算力网络进行了研究。文献[6]针对多层次算力网络，设计了一种代价感知的分布式任务调度算法。为了在算力网络中实现高效计算服务，文献[7]提出了一种任务感知和资源感知的联邦学习模型 FedTAR。该模型通过联合优化单个计算节点的计算策略及其协作学习策略，最小化所有计算节点的总能耗。Tang 等[8]认为，算力网络可以有效满足未来 6G 网络业务对计算、网络、存储资源的灵活调度需求。针对解决 6G 网络中移动边缘计算难以满足普适计算的问题，文献[9]提出了一种无线算力网络（WCPN, wireless computing power network）模型，并在资源限制和任务需求下最小化执行时延和能耗。为了在真实环境下验证算力网络性能，文献[10]开发了算力网络原型测试平台，实现了计算建模、计算感知、计算卸载等关键技术。实验结果表明，与传统边缘计算相比，算力网络实

现了更短的服务时延和更好的负载均衡性能。针对算力网络的研究在一定程度上缓解了算力资源紧缺的局面，但面对大规模新兴应用，尤其是人工智能、大数据分析等领域，仍难以满足算力需求的迅速增长，亟须解决算力供需失衡的问题。

1.2　计算重用

随着互联网生态系统规模不断扩大，相应的计算任务变得越来越复杂，计算请求数量也呈爆炸式增长。然而计算设备的规模扩增和设备的计算性能提高速度稍显缓慢。因此，为进一步高效利用计算资源，加快计算任务执行速度，以复用已有计算结果为特征的计算重用技术[2-3,11-14]正在引起研究人员的关注。

Nour 等[3]提出了一种计算节约网络范式。它建立在边缘计算和雾计算范式之上，通过网络内计算、内容汇总和聚合等减少网络全局计算量，利用重用之前的计算结果来最小化边缘的计算负载。随后，该团队研究人员提出了一种面向边缘的计算重用架构 CoxNet[11]，将依赖性服务建模为有向无环图，进一步面向计算重用提出了融合图和重用图，并基于此实现了低时延的计算服务。进一步地，文献[12]以目标识别为应用案例，通过性能测试证明了计算节约网络的有效性。为了提高虚拟现实应用中视频处理的计算效率，文献[13]针对虚拟现实应用特性，分别研究了空间维度和时间维度的计算重用。实验结果表明，该方案减少 34%的计算量。文献[2]实现了 3 个基于计算重用的应用——矩阵乘法、人脸识别和国际象棋，并分别在各个应用中进行了一系列实验，以量化计算重用系统与不重用系统的收益。结果表明，重用系统可以使任务完成效率提升 50 倍，并大大提高计算资源的利用率。从以上文献可以看出，目前的学者大多针对计算重用系统框架或实验平台进行研究，缺乏动态网络环境下对计算重用任务卸载或资源调度方案的考虑。尽管文献[14]研究了云−边网络下的联合服务卸载和计算重用问题，设计了一种服务卸载算法 Whispering，并在节点上执行计算重用。然而，该方案在计算卸载过程中未考虑节点的数据缓存以及计算重用概率，难以充分发挥计算重用在降低计算需求、提高资源利用率方面的优势。

1.3　任务调度

任务调度策略是影响计算服务质量的关键环节。目前，常见的调度算法可以分为传统调度优化方法和基于学习的调度优化方法。

传统调度优化方法一般基于启发式、博弈论、凸优化等方法[15-18]。例如，文献[15]研究了依赖性任务卸载问题，针对所构建的 NP 难问题，提出了一种基于启发式排名的算法，并通过理论分析证明了该算法的稳定性。针对工业物联网环境下的资源分配问题，文献[16]以最大化资源利用率为目标，构建双阶段斯塔克尔伯格博弈问题，并提出相应的算法来实现纳什均衡和斯塔克尔伯格均衡。在车辆边缘计算场景下，文献[17]将复杂难解的任务卸载与资源分配联合优化问题解耦为 2 个子问题，进一步通过基于概率的方案将任务卸载子问题从离散问题转换为连续凸问题，并提出分布式优化算法得到原问题的次优解。文献[18]考虑了用户之间的资源竞争和移动性，构建了以能耗最小化为目标的计算卸载问题，并提出一种基于博弈论的分布式算法来求解该问题。

然而，随着网络环境愈发复杂，上述基于传统优化的方案已难以实现理想的性能。因此，有学者提出采用基于学习的调度方案[19-24]来适应动态变化的网络环境。文献[19]考虑边缘计算场景下异构任务卸载问题，将移动应用程序建模为有向无环图，提出了基于元强化学习的任务卸载方法，通过少量的梯度更新和样本来快速适应动态环境。文献[20]研究了多雾节点系统中异构任务的联合卸载和资源分配控制问题，并把该问题表述为一个部分可观察的随机博弈，同时利用深度循环 Q 网络方法来近似最优值函数。文献[21]考虑雾计算网络下的任务卸载问题，在稳定网络状态下提出了基于多臂赌博机理论的在线学习算法 TOD 以实现高效的计算任务调度。为了解决密集视频转码和数据传输问题，文献[22]构建了一种增广图模型，将计算和传输联合优化问题转变为路由选择问题，进一步设计了网络化多智能体强化学习方法，实现传输−计算任务的联合优化卸载。文献[23]针对依赖性物联网应用计算卸载问题提出了 CODIA 方案，该方案将优化问题解耦为 2 个子问题：调度和卸载，并设计了基于优先级的调度策略和基于深度强化学习卸载算法分别解决 2 个子问题。文献[24]考虑了观众辅助转码方案中的公平性问题，基于在线学习算法设计一个公平性保证的任务分配方案，并在众包直播服务的场景下证明了该方案的有效性。尽管上述方案在各自的场景下取得了出色的任务调度效果，但都是基于完全计算的方式，未考虑到计算的可重用性，因此无法直接应用于计算可重用的算力网络场景。

2 计算可重用的算力网络模型

2.1 计算可重用的算力网络架构

如图 1 所示，算力网络架构分为基础设施层、资源层、应用层以及负责算力网络管理的管控层。基础设施层包含算力网络中可提供计算能力的设备，例如云服务器、边缘服务器、智能网关、基站等。计算设备可通过虚拟化技术灵活分配其资源，例如 CPU、GPU、RAM 等。基础设施层的物理设备产生的虚拟资源汇聚在资源层，包括计算资源、通信资源和存储资源等。计算资源为计算任务提供算力基础，通信资源支撑计算数据的传输，存储资源则缓存了可重用的计算数据。在应用层，运行计算密集型应用，本文考虑可进行计算任务拆分的应用，如图 1 所示，可将 VR 游戏、超高清直播等新型流媒体服务拆分为多个子任务，例如视频编码、视频分析以及视频增强等。其中，部分子任务可通过算力节点上缓存的计算数据进行计算重用。管控层则负责合理调配算力网络资源来提供高质量计算服务，在该算力网络中，采用集中式管控的架构，管控层中的控制器定期同步全网资源信息、维护资源视图，并基于此做出任务调度决策。

本文考虑一个离散时间的算力网络系统，在每个等长时隙内进行计算任务调度。为了便于表达，用 $t \in \{1,2,\cdots,T\}$ 表示任意一个时隙。算力网络中共有 N 个算力节点，用 $\mathcal{N}$ 来表示算力节点集合，其中的每个节点都具有计算、通信和存储能力，将节点在时隙 t 内的可用计算资源和通信资源分别记作 $P_n(t)$和$C_n(t)$，其中 $n \in \mathcal{N}$ 表示任意一个算力节点。本文假设节点在一个时隙内的可用资源是固定不变的，而由于网络环境的动态变化，节点在不同时隙的可用资源则是动态变化的。可用计算资源 $P_n(t)$ 以期望 $\mathbb{E}[P_n(t)]=\kappa_n^P$ 在 $[P_n^{\min},P_n^{\max}]$ 内波动；类似地，可用通信资源 $C_n(t)$ 以期望 $\mathbb{E}[C_n(t)]=\kappa_n^C$ 在 $[C_n^{\min},C_n^{\max}]$ 内波动。在算力节点的存储空间中，缓存了未来可能重用的计算结果数据。本文维护一个重用表来记录所存储的计算结果数据，该表的大小会影响查找时延，因此限制该表的最大容量为 B。当未来产生类似的计算任务时，即可通过计算重用来降低服务时延并节省计算资源。

2.2 任务模型

在算力网络中，计算密集型服务可拆分成多个子任务[23]，进而执行分布式计算。本文假设任何一个子任务都可以在算力网络中的任意一个节点上执行。所有计算服务的集合为 $\mathcal{S}$，S 表示集合 $\mathcal{S}$ 中计算服务的数量。本文用 x_s 来表示计算服务 $s \in \mathcal{S}$ 的特征，该特征可以包含多个维度，例如，计算服务的应用类型、服务场景等，与该计算服务的属性相关。在每个时隙 t 内会随机产生一个计算服务请求 s，并在该时隙内完成计算。用 $\mathcal{M}_s$ 表示计算服务 s 所拆分成的子任务集合，M_s 表示集合中子任务数量。定义子任务 $i \in \mathcal{M}_s$ 的计算资源需求量为 $\varphi_i^P(t)$，子任务 i 计算结果的数据量为 $\varphi_{n,i}^C(t)$。

2.3 计算模型

在计算可重用的算力网络中，根据是否重用以及重用类型的不同，每个节点上执行的计算任务可分为以下 3 种类型。

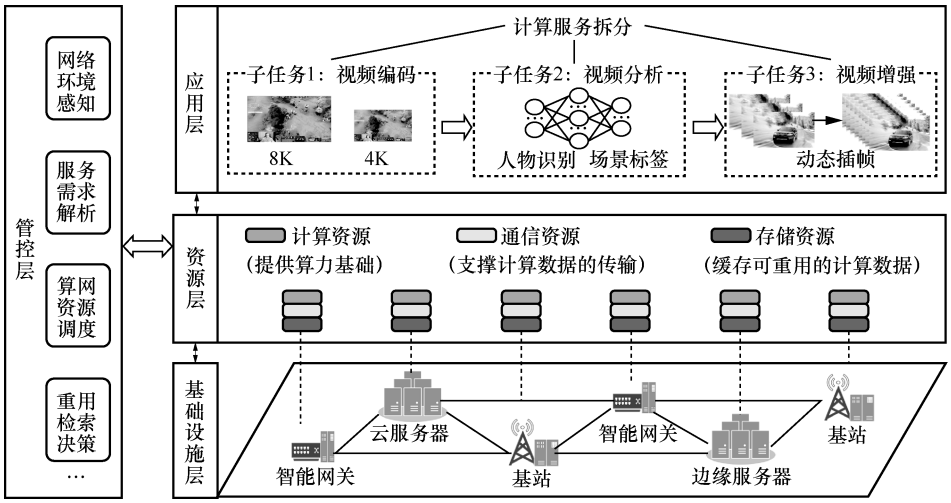

图 1 计算可重用的算力网络架构

部分重用计算：该节点已经缓存了部分该任务可重用的计算数据，计算任务可借助这些数据进行计算，从而节省了部分计算步骤。

完全重用：节点缓存了该计算任务的结果，可以直接使用缓存中的计算数据。

完全计算：节点未缓存任何可重用的数据，需要在该节点上执行完全计算。

计算任务在重用计算结果前需要在节点上进行重用缓存数据查找，该过程会花费一定的时间成本。若查找失败，即当前节点没有缓存可重用的数据，那么查找时间将会成为该机制造成的额外时延，这对时延敏感服务来说代价巨大。为了解决该问题，本文设计一个重用指数 $I_{n,i}^{\text{reuse}} \in \{0,1\}$，该指数预估了子任务 i 在节点 n 上是否可进行计算重用。若 $I_{n,i}^{\text{reuse}}=1$，说明该子任务会有较高的概率在该节点进行计算重用；若 $I_{n,i}^{\text{reuse}}=0$，则说明重用概率较小。重用指数由算力网络管控层决定，具体设计细节将在第 3 节详细介绍。如图 2 所示，在算力网络中的任一算力节点 n 上，当计算子任务 i 到达时，首先检查其重用指数。当 $I_{n,i}^{\text{reuse}}=1$ 时，说明查找成功的概率较大，则进入重用缓存查找模块；当 $I_{n,i}^{\text{reuse}}=0$ 时，则跳过查找模块，直接在该节点进行子任务的完全计算。在重用缓存查找模块中，查找缓存列表是否已经存储了子任务 i 能够利用的计算数据。若部分查找成功，即查找到该子任务所需的部分计算结果数据，则进行部分重用计算；若完全查找成功，即缓存列表中存储了该子任务的全部计算结果，则直接将该结果数据输出，传输给下一节点；若查找失败，缓存中没有任何数据可供该子任务利用，则进行完全计算。

本文将节点 n 对子任务 i 的处理时延 $D_{n,m_i}^{P}(t)$ 表示为

$$D_{n,i}^{P}(t)=\begin{cases}\dfrac{\varphi_i^{P}(t)}{P_n(t)}+I_{n,i}^{\text{reuse}}\tau, & \text{完全计算}\\ \dfrac{(1-f_{n,i})\varphi_i^{P}(t)}{P_n(t)}+\tau, & \text{部分重用计算}\\ 0+\tau, & \text{完全重用}\end{cases} \tag{1}$$

其中，$f_{n,i}$ 表示可重用的计算量占总计算量的比例，τ 表示查找时延。根据式(1)，处理时延 $D_{n,i}^{P}(t)$ 包括两部分：在算力节点上的计算时延以及查找时延。当子任务 i 在节点 n 上进行完全计算时，若 $I_{n,i}^{\text{reuse}}=0$，即跳过重用缓存查找模块，直接执行完全计算，则处理时延等于计算时延；若 $I_{n,i}^{\text{reuse}}\neq 0$，即经过重用缓存查找后执行完全计算，则处理时延等于计算时延与查找时延的和。当执行部分重用计算时，处理时延包括计算时延和查找时延，而此时的计算时延取决于剩余计算量的大小和节点的计算能力。当执行完全重用时，由于直接读取缓存中的计算结果数据，不在节点上进行实际计算，因此处理时延只包括查找时延。本文考虑计算密集型服

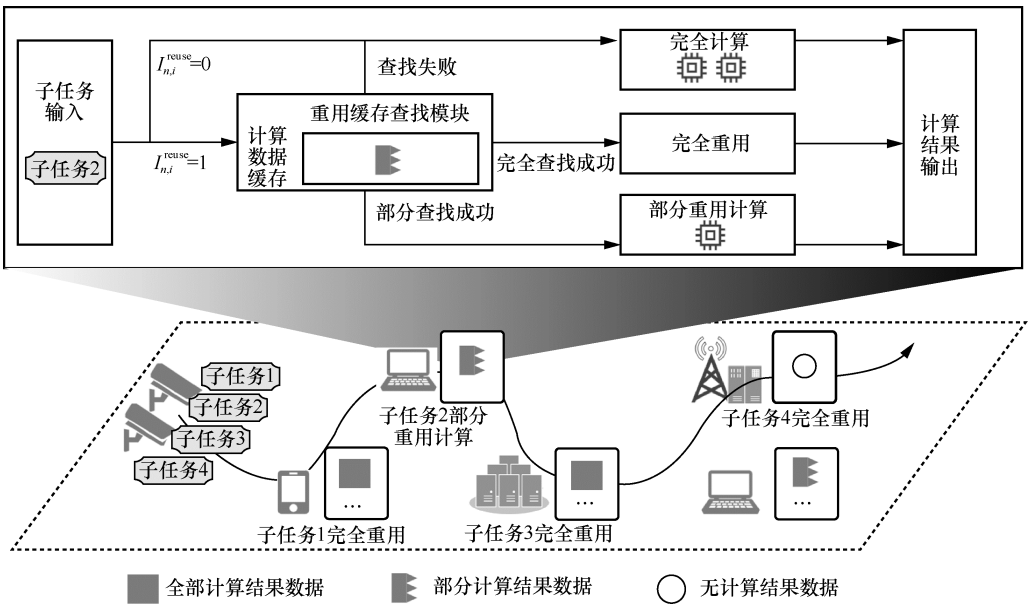

图 2　计算重用过程

务，同时设置适当的重用表大小 B，以使数据查找时延小于计算时延。因此本文希望尽可能地执行完全重用来降低任务在节点上的处理时延。

2.4 通信模型

不同算力节点之间的任务数据传递需要通信资源的支持，具有较高带宽资源的节点能够快速将计算结果传输给下一节点。根据香农定理，将节点 n 上的最大数据传输速率表示为

$$R_n(t)=C_n(t)\mathrm{lb}\left(1+\frac{p_n h_n}{\sigma^2}\right) \tag{2}$$

其中，p_n 表示节点 n 的传输功率；h_n 表示信道增益，服从指数分布，即 $h_n \sim \exp(1)$；σ^2 表示噪声功率。因此，节点 n 将子任务 i 的计算结果传输给下一算力节点的最小传输时延 $D_{n,i}^{C}(t)$ 为

$$D_{n,i}^{C}(t)=\frac{\varphi_i^{C}(t)}{R_n(t)} \tag{3}$$

2.5 服务时延最小化问题构建

服务时延表示从第一个算力节点接收到计算子任务到最后一个算力节点将计算结果返回给用户所需要的时间。因此，服务时延计算式为

$$D(t)=\sum_{i\in\mathcal{M}_s}\sum_{n\in\mathcal{N}} I_{n,i}(t)D_{n,i}(t) \tag{4}$$

其中，$I_{n,i}(t)\in\{0,1\}$ 表示任务调度结果，若子任务 i 在节点 n 上执行计算，则 $I_{n,i}(t)=1$，反之 $I_{n,i}(t)=0$；$D_{n,i}(t)=D_{n,i}^{P}(t)+D_{n,i}^{C}(t)$ 表示子任务 m_i 在节点 n 上执行计算产生的服务时延。

对于本文关注的时延敏感的计算密集型应用，本文希望为用户提供低时延的计算服务。因此，本文的目标是通过计算子任务调度决策来降低算力网络系统中计算服务的平均服务时延。将任务调度决策记作 $\boldsymbol{I}=\{I_{n,i}(t)\,|\,n\in\mathcal{N},i\in\mathcal{M}_s,t\in\mathcal{T}\}$，本文的优化目标可表示为

$$\min_{\boldsymbol{I}}\frac{1}{T}\sum_{t=1}^{T}D(t) \tag{5}$$

$$\text{s.t.}\sum_{n\in\mathcal{N}} I_{n,i}(t)=1,\forall i\in\mathcal{M}_s,\forall t\in\mathcal{T} \tag{6}$$

其中，约束式(6)保证了每个子任务都会被调度到一个算力节点上。

在该优化问题中，总服务时延包括三部分：计算时延、查找时延和通信时延。本文希望能够选择计算和通信能力较强的算力节点，并尽可能多地重用已缓存的计算数据，从而降低计算与通信时延。另外，对于查找时延，重用指数的准确性至关重要。若该指数偏大，则可能造成大量的失败查找，增加额外的查找时延；若该指数偏小，则会导致部分可重用的任务跳过重用缓存查找模块，执行完全计算。因此，需要解决两大问题：1) 设计高效的任务调度策略，以提高计算重用概率并降低计算与通信时延；2) 设计合理的重用指数。

3 基于服务联盟的上下文感知在线学习算法

为了解决上述问题，本节首先介绍基于历史查找信息的重用指数设计，并对最优任务调度问题进行分析。进一步地，在多臂赌博机（MAB, multi-armed bandit）模型[25]的基础上，提出了 SF-COL 算法。

3.1 基于历史查找信息的重用指数设计

首先，为了减少额外查找时延，本文设计了一种基于历史查找信息的重用指数。算力网络管控层维护一个历史查找表 $\boldsymbol{H}$，记录子任务是否在节点上执行重用缓存查找以及查找结果（包括部分查找成功、完全查找成功以及查找失败）。$H_{n,i}=0$ 表示子任务 m_i 未在节点 n 上执行重用缓存查找；$H_{n,i}=1$ 表示在 t 时刻查找成功，包括部分查找成功和完全查找成功 2 种情况；$H_{n,i}=-1$ 表示执行了重用缓存查找，但查找失败。定义 $H_{n,i}^{*}$ 为子任务 i 在节点 n 上的最近一次查找情况。

当新的服务请求到来时，若子任务 i 被分配到节点 n 执行计算，则检查最近一次的查找情况。若 $H_{n,i}^{*}=1$，说明在最近的一次查找中，节点 n 缓存了子任务 i 所需的计算数据，那么当前该节点仍有较大的概率存储了该数据，因此设置 $I_{n,i}^{\text{reuse}}=1$，执行重用缓存查找；否则，设置 $I_{n,i}^{\text{reuse}}=0$。虽然上述方法可以根据最近一次的查找结果来判断当前的可重用缓存情况，但却是一种静态僵化的机制，缺乏主动探索。如果出现 $H_{n,i}^{*}=-1$ 或者 $H_{n,i}^{*}=0$ 的情况，该机制认为节点 n 上没有子任务 i 所需要的数据，那么未来将不会再在节点 n 上执行查找。然而，在实际应用中重用缓存可能会动态更新，若节点 n 在未来的某个时间段缓存了子任务 i 所需要的数据，该机制将使子任务 i 失去可重用的机会。为了解决该问题，本文采用定期探索的机制来探知缓存的动态变化。定义 $Z_{n,i}$ 为 $H_{n,i}$ 连续等于 0 或−1 的次数，并设定一个长度为 δ 的探索窗口。若历史查找表中

已有连续 δ 次未执行重用缓存查找，即 $Z_{n,i}=\delta$，则下次必定执行重用缓存查找，以探知缓存中是否有新的可重用计算数据。因此，综合考虑最近一次的查找情况以及探索窗口机制，重用指数 $I_{n,i}^{\text{reuse}}$ 可以通过式(7)来生成

$$I_{n,i}^{\text{reuse}}=\begin{cases}1, & H_{n,i}^{*}=1\text{或}Z_{n,i}=\delta\\ 0, & \text{其他}\end{cases} \tag{7}$$

3.2　任务调度算法设计

3.2.1　最优任务调度

假设所有节点的重用缓存情况、计算时延、传输时延都已经预先获知，那么时隙 t 内的最优任务调度决策方案可以由式(8)给出

$$I_{n,t}^{*}(t)=\arg\min_{I_{n,t}(t)} D_{n,t}(t) \tag{8}$$

根据式(7)，每个子任务的最优卸载决策可以通过 N 次计算来找到。然而，在实际应用中，节点的计算、通信时延取决于动态变化的可用网络、计算资源，而计算重用则取决于计算服务特征以及算力节点的重用数据缓存情况。尽管算力网络管控层在尽力维护全局算网资源视图，但在实际中难以获取所有实时状态信息，且无法预知环境状态信息与总服务时延间的关系，这使最优任务调度策略在实际中并不可行。在此背景下，为实现高效任务调度，在 MAB 问题的基础上，本文提出 SF-COL 算法。

3.2.2　MAB 问题构建

为了解决上述状态信息不可提前预知的问题，本文采用在线学习算法，通过与环境的动态交互学习未知信息。在本文的任务调度问题中，系统的状态（如节点的计算资源、通信资源、计算服务类型）会影响服务时延，但是任务调度策略却不会改变系统状态，因此，它是典型的上下文感知的 MAB 问题[26]，该问题是一种经典的序列决策问题，属于强化学习的范畴。在每轮决策过程中，智能体借助上下文信息从多个臂中选择一个臂，然后得到一定的奖励。每个臂的奖励服从一定的分布，但对智能体不可知。本文将算力节点作为臂，算力节点控制器虚拟化出多个子控制器作为智能体，智能体 p_i 负责为子任务 i 做出调度决策。接下来，将任务调度问题式(5)和式(6)建模为上下文感知的 MAB 问题。

上下文。丰富的上下文信息有利于智能体快速学习环境模型，在算力网络中，管控层维护了算力节点的资源信息，例如算力节点的计算和通信能力。同时，由于计算服务的异构特性，不同服务的可重用数据不同。因此，智能体需要针对不同计算服务，调度符合其重用需求的算力节点来执行计算任务。在此背景下，该模型中的智能体可观测到的上下文信息包括计算服务信息以及算力节点信息。因此，将 t 时隙的上下文信息表示为

$$\boldsymbol{o}_{n,i}(t)=\left\{x_s,\varphi_i^P(t),\varphi_i^C(t),\kappa_n^P,\kappa_n^C\right\} \tag{9}$$

动作。智能体 p_i 在时隙 t 内根据上下文信息 $\boldsymbol{o}_{n,i}(t)$ 来选择一个臂，该动作被定义为

$$a_i(t)=\left\{n\mid I_{n,i}^{o}(t)=1\right\} \tag{10}$$

表示选择算力节点 n 来执行子任务 i。

奖励。智能体选择算力节点后，会获得相应的奖励。本文的目标是降低服务时延，因此智能体的奖励与服务时延相关，越低的服务时延对应越高的奖励。在时隙 t 内，智能体 p_i 选择算力节点 n 所获得的奖励被表示为

$$r_{n,i}(t)=-D_{n,i}(t) \tag{11}$$

在该问题中，网络状态信息是动态变化的，需要智能体在与环境不断交互的过程中学习出规律。因此，本文的目标是尽可能地增大长期平均奖励的期望，即

$$\max\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\sum_{i\in\mathcal{M}_s}\sum_{n\in\mathcal{N}}r_{n,i}(t)\right] \tag{12}$$

3.2.3　在线任务调度算法设计

接下来，介绍本文所提出的 SF-COL 算法。在 MAB 模型中，需要解决的核心问题为智能体如何处理“探索−利用”的均衡关系。该问题可以描述为在“探索更高奖励动作”与“利用历史最佳动作”之间寻求平衡。在初期阶段，智能体缺乏关于算力节点的先验知识，也无法获得不同服务对算力节点的偏好，需要先对算力节点属性及服务特征信息进行学习探索；在后期阶段，智能体根据前期探索的历史经验以及当前的服务特征状态来做出任务调度决策，以降低服务时延，从而获得更高的奖励。为解决上述“探索−利用”均衡的问题，本文在 LinUCB[26]算法的基础上设计了 SF-COL 算法，该算法可以通过服务联盟机制节省探索学习的时间成本。

1) 基于岭回归的模型系数

本文假设上下文信息与奖励呈线性关系[26]，并维护模型系数向量 $\boldsymbol{\theta}_{n,i}^{*}$ 来描述该线性关系。对于每个时隙，有

$$\mathbb{E}[r_{n,i} \mid \boldsymbol{o}_{n,i}] = \boldsymbol{\theta}_{n,i}^{*\mathrm{T}} \boldsymbol{o}_{n,i} \tag{13}$$

因此，在时隙 t 下的最优节点选择可以表示为

$$a_i^{*} = \arg\max_{n} \boldsymbol{\theta}_{n,i}^{*\mathrm{T}} \boldsymbol{o}_{n,i} \tag{14}$$

本文通过历史上下文信息及奖励，得到当前上下文与预测奖励之间的关系，即不断优化模型系数 $\boldsymbol{\theta}_{n,i}^{*}$。

本文采用岭回归方法来估计模型系数。用 $\boldsymbol{G}_{n,i} \in \mathbb{R}^{o \times t}$ 表示到时隙 t 为止智能体 p_i 收集到的关于节点 n 的历史上下文信息，其中，o 表示上下文信息的维度，$\boldsymbol{q}_{n,i} \in \mathbb{R}^{t}$ 表示智能体 p_i 选择节点 n 的历史奖励。通过岭回归方法，可以得到关于 $\boldsymbol{\theta}_{n,i}^{*}$ 的估计为

$$\hat{\boldsymbol{\theta}}_{n,i} = \left(\boldsymbol{G}_{n,i}^{\mathrm{T}} \boldsymbol{G}_{n,i} + \boldsymbol{I}_o\right)^{-1} \boldsymbol{G}_{n,i}^{\mathrm{T}} \boldsymbol{q}_{n,i} \tag{15}$$

其中，$\boldsymbol{I}_o$ 表示单位矩阵。

2) “探索”与“利用”过程

本文通过动作的置信度来指导智能体的探索过程。关于置信区间有以下引理。

引理 1 假设奖励 $r_{n,i}(t)$ 是独立随机变量且均值为 $\mathbb{E}\left[r_{n,i} \mid \boldsymbol{o}_{n,i}\right] = \boldsymbol{o}_{n,i}^{\mathrm{T}} \boldsymbol{\theta}_{n,i}^{*}$，令 $\alpha(t) = \mathcal{O}(\sqrt{\log t})$，$\boldsymbol{A}_{n,i} = \boldsymbol{G}_{n,i}^{\mathrm{T}} \boldsymbol{G}_{n,i} + \boldsymbol{I}_o$，则置信区间上界（CB, confidence bound）为

$$\left|\boldsymbol{o}_{n,i}^{\mathrm{T}} \hat{\boldsymbol{\theta}}_{n,i} - \boldsymbol{o}_{n,i}^{\mathrm{T}} \boldsymbol{\theta}_{n,i}^{*}\right| \leqslant \alpha(t) \sqrt{\boldsymbol{o}_{n,i}^{\mathrm{T}} \boldsymbol{A}_{n,i}^{-1} \boldsymbol{o}_{n,i}} \tag{16}$$

用 $\mathrm{CB}_{n,i}^{\boldsymbol{o}}(t)$ 表示在上下文信息 $\boldsymbol{o}_{n,i}$ 下为子任务 i 选择算力节点 n 的置信区间上界，即

$$\mathrm{CB}_{n,i}^{\boldsymbol{o}}(t) = \alpha(t) \sqrt{\boldsymbol{o}_{n,i}^{\mathrm{T}} \boldsymbol{A}_{n,i}^{-1} \boldsymbol{o}_{n,i}} \tag{17}$$

在每个时隙中，智能体 p_i 选择奖励估计最大的算力节点来执行计算任务，即

$$a_i^{*} = \arg\max_{c} \left[\boldsymbol{o}_{n,i}^{\mathrm{T}} \hat{\boldsymbol{\theta}}_{n} + CB_{n,i}^{\boldsymbol{o}}(t)\right] \tag{18}$$

式(18)表示了“探索−利用”的均衡。第一项 $\boldsymbol{o}_{n,i}^{\mathrm{T}} \hat{\boldsymbol{\theta}}_{n}$ 表示对历史信息的利用，第二项 $\mathrm{CB}_{n,i}^{\boldsymbol{o}}(t)$ 表示对节点的探索。

3) 基于服务联盟的协作学习机制

不同计算服务可能具有相似的特征，从而对算力节点有类似的偏好。基于此现象，本文提出基于服务联盟的协作学习机制。该机制通过使用从联盟中所有服务收集的经验来训练一个通用的联盟奖励估计模型。由于可以共享联盟内成员的学习结果，利用联盟成员的大量历史经验进行学习，大大降低了学习时间成本，在学习动态变化的算力节点状态及网络环境时表现出优秀的自适应能力。本文设定一个未知向量 $\boldsymbol{u}_{s,n,i}$，该向量决定了节点 n 在子任务 $i \in \mathcal{M}_s$ 下的平均表现。如果服务 s 与任一服务 $j \in \mathcal{S}$ 在同一联盟内，那么 $\boldsymbol{u}_{s,n,i}^{\mathrm{T}} \boldsymbol{o}_{n,i} = \boldsymbol{u}_{j,n,i}^{\mathrm{T}} \boldsymbol{o}_{n,i}$；反之，$\left|\boldsymbol{u}_{s,n,i}^{\mathrm{T}} \boldsymbol{o}_{n,i} - \boldsymbol{u}_{j,n,i}^{\mathrm{T}} \boldsymbol{o}_{n,i}\right| \geqslant \beta$，其中 $\beta > 0$ 为间隔参数。

基于以上对服务联盟的描述，本文将在上下文 $\boldsymbol{o}_{n,i}(t)$ 下选择节点 n 可获得相似奖励的服务划分在同一联盟中。具体来说，与服务请求 s 在同一联盟中的服务集合表示为

$$\begin{aligned}&\hat{\mathcal{F}}_{s,n,i}^{\boldsymbol{o}} = \\ &\left\{j \in \mathcal{S} : \left|\hat{\boldsymbol{\theta}}_{s,n,i}^{\mathrm{T}}(t-1)\boldsymbol{o}_{n,i}(t) - \hat{\boldsymbol{\theta}}_{j,n,i}^{\mathrm{T}}(t-1)\boldsymbol{o}_{n,i}(t)\right| \leqslant \right. \\ &\left. \mathrm{CB}_{s,n,i}^{\boldsymbol{o}}(t-1) + \mathrm{CB}_{j,n,i}^{\boldsymbol{o}}(t-1)\right\}\end{aligned} \tag{19}$$

为了简化表达，下文用 $\hat{\mathcal{F}}_s$ 表示 $\hat{\mathcal{F}}_{s,n,i}^{\boldsymbol{o}}$。根据形成的服务联盟，对模型参数估计以及置信区间上界进行更新

$$\hat{\boldsymbol{\theta}}_{\hat{\mathcal{F}}_s,n,i}(t-1) = \frac{1}{\left|\hat{\mathcal{F}}_s\right|} \sum_{j \in \hat{\mathcal{F}}_s} \hat{\boldsymbol{\theta}}_{j,n,i}(t-1) \tag{20}$$

$$\mathrm{CB}_{\hat{\mathcal{F}}_s,n,i}^{\boldsymbol{o}}(t-1) = \frac{1}{\left|\hat{\mathcal{F}}_s\right|} \sum_{j \in \hat{\mathcal{F}}_s} \mathrm{CB}_{j,n,i}^{\boldsymbol{o}}(t-1) \tag{21}$$

根据式(20)和式(21)，联盟的模型系数值和置信上界均取联盟内所有服务的均值。基于此，智能体 p_i 为服务 s 选择最佳算力节点，即

$$a_i^{*} = \arg\max_{n} \left[\boldsymbol{o}_{n,i}^{\mathrm{T}} \hat{\boldsymbol{\theta}}_{\hat{\mathcal{F}}_s,n,i}(t-1) + \mathrm{CB}_{\hat{\mathcal{F}}_s,n,i}^{\boldsymbol{o}}(t-1)\right] \tag{22}$$

根据式(22)选择算力节点 n^{*} 执行计算任务后，收到环境反馈的奖励 $r_{n^{*},i}(t)$。接下来，基于该奖励，更新历史信息。具体更新方法如下。

若对当前所选算力节点 n^{*} 的置信度较低，即 $\mathrm{CB}_{s,n^{*},i}^{\boldsymbol{o}}(t-1) \geqslant \dfrac{\beta}{4}$，那么只更新与当前服务 s 相关的历史信息；若当前选择的置信度较高，即 $\mathrm{CB}_{j,n^{*},i}^{\boldsymbol{o}}(t-1) < \dfrac{\beta}{4}$，则更新联盟 $\hat{\mathcal{F}}_s$ 内所有服务的历史信息。该机制是基于 $\hat{\mathcal{F}}_{s,n^{*},i}^{\boldsymbol{o}} = \mathcal{F}_{s,n^{*},i}^{\boldsymbol{o}}$ 的，也就是说，奖励 $r_{n^{*},i}(t)$ 的条件期望 $\boldsymbol{u}_{s,n^{*},i}^{\mathrm{T}}(t)\boldsymbol{o}_{n^{*},i}$ 与联盟内的任意

服务 $j\in\hat{\mathcal{F}}_s$ 的 $\boldsymbol{u}_{j,n^*,i}^{\mathrm{T}}(t)\boldsymbol{o}_{n^*,i}$ 是相等的，使 $\mathrm{CB}_{j,n^*,i}^{o}(t-1)<\dfrac{\beta}{4}$。

本文采用遗憾值来衡量算法的性能，将遗憾值定义为实际得到的奖励与最优策略所对应奖励的差值。因此，时隙 t 内的遗憾值可以表示为

$$\Xi(t)=\left(\max_{n\in\mathcal{N}}\boldsymbol{u}_{s,n,i}^{\mathrm{T}}\boldsymbol{o}_{n,i}\right)-\boldsymbol{u}_{s,n^*,i}^{\mathrm{T}}\boldsymbol{o}_{n^*,i}. \tag{23}$$

根据文献[27]的定理 1，本文有以下定理。

定理 1　假设在 β-间隔下对于所有 $s\in\mathcal{S}$，$\left(\boldsymbol{u}_{s,n,i}^{\mathrm{T}}\boldsymbol{o}_{n,i}\right)-\hat{\boldsymbol{\theta}}_{s,n,i}^{\mathrm{T}}\boldsymbol{o}_{n,i}\leqslant\mathrm{CB}_{s,n,i}^{o}(t)$ 成立，那么在给定的总时隙 T 下，算法 SF-COL 的累计遗憾 $\sum_{t=1}^{T}\Xi(t)$ 有以下上界

$$\sum_{t=1}^{T}\Xi(t)\leqslant 9\alpha(T)\left(N\,\mathrm{HD}\left(\left\{s,\boldsymbol{o}_{n,i}(t)\right\}_{t=1}^{T},\frac{16\alpha^2(T)}{\beta^2}\right)+\sqrt{d\log T\sum_{t=1}^{T}\frac{n}{\left|\hat{\mathcal{F}}_{s,n^*}^{o}\right|}}\right) \tag{24}$$

其中，HD 函数表示对于给定的序列 $\{s,\boldsymbol{o}_{n,i}(t)\}$，$\boldsymbol{A}_{n,i}$ 达到其最小特征值小于 $\dfrac{16\alpha^2(T)}{\beta^2}$ 所需要的时隙，具体定义详见文献[27]。

值得注意的是，本文的计算子任务有先后执行顺序。因此，不同智能体可以选择同一计算节点执行计算任务而不受影响，不存在典型多智能体 MAB 模型中的碰撞问题[28]。SF-COL 算法的详细过程如算法 1 所示。

算法 1　SF-COL 算法

输入　间隔参数 β

输出　任务调度决策 $\boldsymbol{I}$

初始化　对于所有 $n\in\mathcal{N},i\in\mathcal{M}_s,s\in\mathcal{S}$，设置初始化值 $\boldsymbol{G}_{n,i}=\boldsymbol{0}\in\mathbb{R}^{o\times t}$，$\boldsymbol{q}_{n,i}=\boldsymbol{0}\in\mathbb{R}^{t}$

1) for t=1:T do
2)　接收到服务请求 s，并观测到上下文信息 $\boldsymbol{o}_{n,i}(t)$；
3)　for 智能体 $p_i,i\in\mathcal{M}_s$ do
4)　/*准备阶段*/
5)　for 节点 $n\in\mathcal{N}$ do
6)　根据式(19)计算联盟成员集合 $\hat{\mathcal{F}}_s$。
7)　根据式(20)和式(21)计算联盟的模型估计参数 $\hat{\boldsymbol{\theta}}_{\hat{\mathcal{F}}_s,n,i}(t-1)$ 和置信区间上界 $\mathrm{CB}_{\hat{\mathcal{F}}_s,n,i}^{o}(t-1)$。
8)　end for
9)　/*动作执行阶段*/
10)　根据式(22)选择奖励估计最大的算力节点 n^* 来执行计算任务，并根据式(7)计算 $I_{n^*,i}^{\mathrm{reuse}}$ 决定是否进行重用缓存查找。
11)　收到奖励 $r_{n^*,i}(t)$。
12)　/*历史信息更新阶段*/
13)　if $\mathrm{CB}_{s,n^*,i}^{o}(t-1)\geqslant\dfrac{\beta}{4}$ then
14)　更新 $\boldsymbol{D}_{n^*,i}$ 与 $\boldsymbol{q}_{n^*,i}$
15)　else
16)　for all $j\in\hat{\mathcal{F}}_{s,n^*}$ do
17)　更新 $\boldsymbol{G}_{n^*,k}$ 与 $\boldsymbol{q}_{n^*,k},k\in\mathcal{M}_j$
18)　end for
19)　end if
20)　根据式(15)和式(17)更新 $\hat{\boldsymbol{\theta}}_{n,i}(t)$ 和 $\mathrm{CB}_{n,i}^{o}(t)$。
21)　end for
22)　更新 HST。
23) end for

根据算法 1，对于每个智能体 $p_i,i\in\mathcal{M}_s$，SF-COL 算法分为 3 个阶段。在准备阶段，智能体根据历史信息得到服务 s 所在的服务联盟，并计算联盟的模型估计参数和置信区间上界。在动作执行阶段，智能体根据联盟信息为服务 s 选择最佳算力节点 n^*，并基于重用指数决定是否执行重用查找，动作执行后收到环境反馈的奖励。在历史信息更新阶段，根据上述动作的置信度来进行信息更新。

4　实验与结果

为了评估所提算法的性能，本文进行了一系列的仿真实验。本节首先介绍仿真实验环境设置，然后介绍实验结果并进行分析。

4.1　实验环境设置

考虑由多个算力节点和一个算力网络控制器构成的计算可重用算力网络，主要仿真参数设置如表 1 所示。

表 1 仿真参数设置

参数	值
算力节点传输功率/mW	100
噪声功率/dBm	−100
算力节点带宽资源/MHz	[10,20]
子任务的数据量/MB	[0.3,0.5]
子任务的计算需求/GHz	[0.5,0.6]
算力节点计算资源/GHz	[1,10]
查找时延/s	0.05

在计算方面，具有较强异构性的算力节点随机分布在网络内。本文实验中用算力节点的 GPU 时钟频率来表示节点的计算资源[29]，设置算力节点的计算资源均值在[1, 10] GHz 内，即 $\kappa_n^P \in [1,10], \forall n \in \mathcal{N}$。另外，设置节点上的查找时延为固定值 $\tau = 0.05$ s，对应达到缓存容量上限时的最大查找时延。在传输方面，算力节点具有异构时变的传输能力，节点传输功率设为 100 mW，噪声功率为−100 dBm，算力节点的带宽资源范围设置为[10,20] MHz。在重用数据缓存方面，当数据超过容量限制时，采用最不常用（LFU, least frequently used）缓存淘汰算法[30]进行数据替换。

为了模拟计算子任务的调度过程，本文考虑 AR/VR 游戏场景下的视频流计算服务。每个计算服务被拆分为 4 个子任务（子任务 1～子任务 4），分别是视频编码、视频分析、视频增强以及视频渲染。对于子任务 2 视频分析，由于不同用户可能会产生相同的视频分析服务请求，例如，热度较高的建筑/人物的视频分析请求，因此可以进行计算重用。对于子任务 4 视频渲染，由于部分用户的游戏背景是相同的，因此，可以直接利用缓存中已渲染的游戏背景部分，只执行前景部分的渲染计算。总体来说，算力节点中可能缓存了子任务 2 和子任务 4 可重用的计算结果，但子任务 1 和子任务 3 需要进行完全计算。

在实验中设置不同的计算服务类型，计算服务类型将会影响子任务的具体计算内容。例如，对于雪山场景和沙漠场景的游戏服务，子任务 4 的渲染内容不同，计算重用数据也完全不同；在分析建筑的和分析人物的服务请求下，子任务 2 的计算可重用数据也不相同。另外，设置每个子任务的输出数据量为[0.3,0.5] MB，计算需求为[0.5, 0.6] GHz。同一计算类型的服务，其子任务的计算需求和输出数据量也可能不同，因此，根据计算服务类型、计算需求和输出数据量等多种因素生成了不同的计算服务。

为了展示所提算法的优越性，本文将其与 4 种基准算法进行对比，具体介绍如下。

1) TOD[21]。文献[21]研究了计算任务卸载问题，以降低任务平均卸载时延为目标，提出了基于在线学习的任务卸载算法。该算法考虑了计算与网络资源的动态性，但没有采用计算重用，即所有计算任务均执行完全计算。

2) Whispering[14]。文献[14]研究了云−边网络中的联合服务卸载和计算重用问题，通过综合考虑节点计算和通信能力，设计了一种基于排序的算法来实现服务卸载，并在节点上执行计算重用。

3) N-HST（non-HST）。在所提 SF-COL 算法的基础上，通过删掉 HST 构建了该对比算法，以验证本文所提 HST 的有效性。该算法考虑计算重用，但是没有维护 HST，仅通过在线学习算法进行任务调度，且每次执行计算任务前均进行重用查找。

4) Greedy。该算法通过初始化获得的初始值，以贪心的方式选择算力节点进行计算任务调度。

4.2 实验结果

首先，为了验证在不同算力节点数量 N 下所提 SF-COL 算法的性能，本节实验设置算力节点数量分别为 50、100、200。同时，计算服务数量设置为 40，每个时隙在 40 个计算服务中随机选择一个作为服务请求。如图 3 所示，在不同的算力节点数量下，SF-COL 均可实现算法收敛，证明了所提算法的可收敛性与可扩展性。注意到，算力节点数量越多，算法的收敛速度越慢。当网络中有 50 个算力节点时，SF-COL 算法在约 200 个时隙时即可收敛；当算力节点数量增大到 200 个时，需要约 800 个时隙才可收敛。这是因为算法会尽力探索所有节点的奖励情况，当算力网络规模变大、节点数量增多时，探索阶段所花费的时间将会变长。

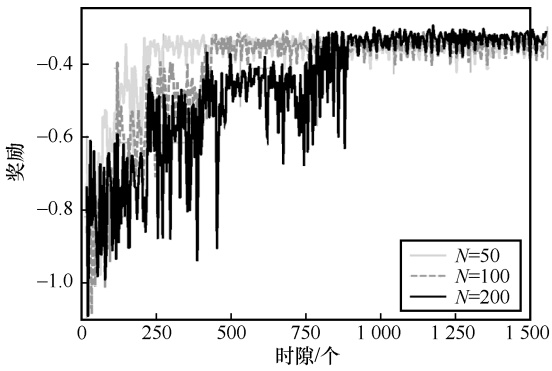

图 3 不同算力节点数量下的算法奖励

接下来，将算力节点数量固定为 100，并设置计算服务数量 S 分别为 40、80、120，实验结果如图 4 所示。与图 3 类似，所有曲线最终均实现收敛。在图 4 中，计算服务数量越少，算法收敛越快。这是因为算力节点在不同计算服务下的表现可能不同。SF-COL 算法需要探索不同服务联盟下每个算力节点产生的收益，因此，服务数量增加导致了算法的探索时间变长，收敛速度减慢。

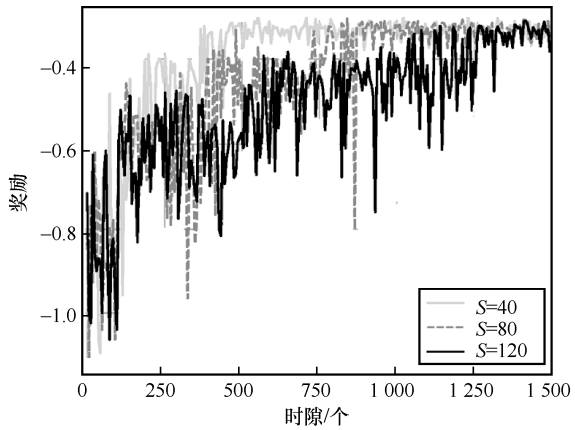

图 4　不同服务数量下的算法奖励值

图 3 和图 4 中的实验结果证明了所提算法的收敛性与可扩展性。同时，可以看出 SF-COL 算法的收敛时间与算力节点数量以及计算服务数量相关：节点数量规模越大，服务数量越多，则算法需要更多的时间去探索直至收敛。

为了对比所提算法与基准算法的性能，本节将算力节点数量和计算服务数量分别设置为固定值 100 和 40。图 5 和图 6 分别展示了 SF-COL 算法与基准算法在奖励值和累计平均奖励值方面的表现。从图 5 可以看出，所有算法均可实现收敛。其中，Greedy 根据初始化时对每个节点的了解，贪心地选择算力节点进行任务调度。然而，由于没有充分探索所有节点在不同计算服务请求下的表现，未能全面评估所有节点的潜在能力，无法选择到能力更强的节点。这一局限性使该算法在任务调度有效性方面表现较差，因此，在图 5 中 Greedy 的奖励值最低。TOD 的表现优于 Greedy，但是比 Whispering 和 N-HST 差。这是因为 TOD 采用完全计算的方式，在资源调度的过程中仅考虑节点的计算和带宽能力，并没有利用节点上的数据缓存进行计算重用，从而导致了更高的计算时延，使总体奖励较差。总体来看，本文所提 SF-COL 算法表现最佳。由于 SF-COL 中并非每次计算前都需执行重用查找，比 N-HST 节约了部分查找时延，因此性能优于 N-HST。同时，Whispering 在选择算力节点过程中没有考虑节点数据缓存对数据重用的影响，仅把节点的计算和通信能力作为任务调度依据，因此其性能比 SF-COL 差。虽然 Whispering 有一定的概率通过计算重用来降低服务时延，但由于增加了额外的查找时延，其表现和 TOD 差距不大。

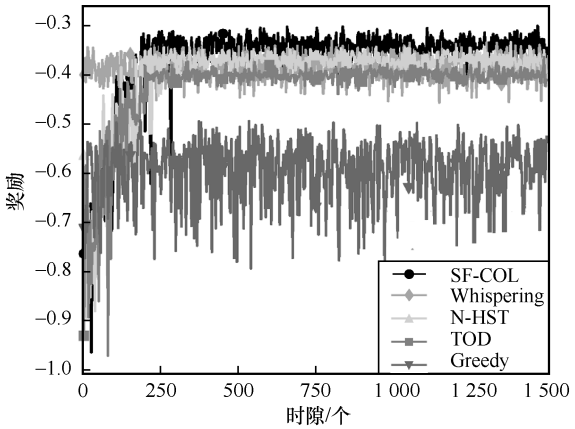

图 5　不同算法的奖励值

根据以上分析，SF-COL 与 N-HST 的性能差距取决于查找时延：查找时延越长，SF-COL 的优势越大。由于 TOD 采用完全计算的方式，SF-COL 与 TOD 的性能差距则主要依赖于重用计算部分的计算时延：若可计算重用部分的计算量越大、计算时延越长，SF-COL 可通过计算重用实现更短的总服务时延，从而达到更高的奖励值。Whispering 虽然采用计算重用的方式，但其进行任务调度时仅考虑节点的计算和通信能力，因此，SF-COL 与 Whispering 的差距则主要取决于最优计算通信能力的算力节点是否可进行计算重用：若 Whispering 所选出的最优节点不可计算重用，则 SF-COL 的优势会更大。

图 6 展示了不同算法的累计平均奖励值。与图 5 类似，SF-COL 最终实现了最优的性能，且随着时间的推移，在累计平均奖励方面的优势不断增大。在 300 个时隙之后，SF-COL 算法的曲线逐渐超越除 Whispering 外的其他算法曲线。Whispering 的累计平均奖励值前期比较高是因为提前获取了节点信息，没有探索阶段，而这在实际应用中是难以实现的，其适用性有限。SF-COL 算法通过探索的方式嗅探各节点的状态，因此前期累计平均奖励不高。但随着算法的收敛，累计平均奖励也不断增高，在大概 1 200 个时隙后，SF-COL 超过了 Whispering 并且差距逐渐变大，展现了 SF-COL 的长期优越性。

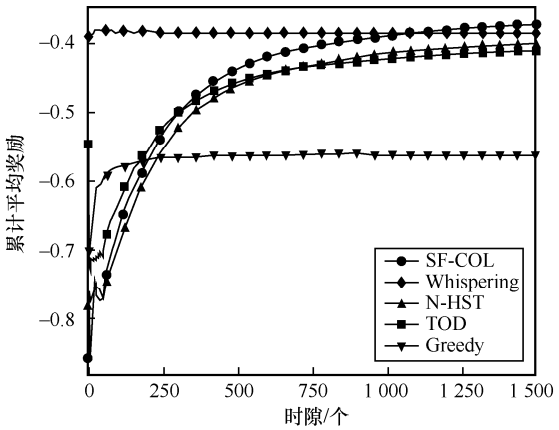

图 6　不同算法的累计平均奖励

图 7 和图 8 分别比较了在不同算力节点数量和计算服务数量下 SF-COL 算法和基准算法在时延方面的表现。图 7 和图 8 分别将计算服务数量和算力节点数量固定为 40 和 100。Greedy 的时延最长，SF-COL 算法实现了最短的时延。由于额外的重用查找时延以及无重用导致的额外计算时延，N-HST、TOD 和 Whispering 的时延均长于 SF-COL 算法。同时可以看出，算力节点数量和计算服务数量对 SF-COL 算法下的时延影响不大，证明了所提算法在大规模算力网络下的可用性。

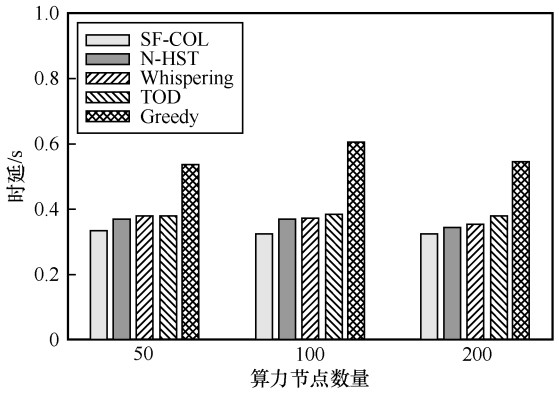

图 7　不同算力节点数量下不同算法的时延对比

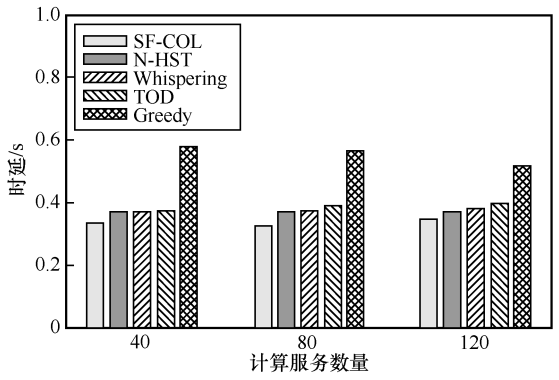

图 8 不同计算服务数量下不同算法的时延对比

定义计算资源消耗为计算任务占用的节点 GPU 时钟频率。图 9 展示了 SF-COL 算法和 TOD、Whispering 算法的计算资源消耗量对比。TOD 调度算力节点以完全计算的方式执行所有计算服务，因此，计算资源消耗量等于服务的计算资源需求量。根据本文的实验设置，每个子任务的计算资源需求为[0.5, 0.6] GHz。由于每个计算服务包含 4 个子任务，因此，TOD 的计算资源消耗量应在[2, 2.4] GHz 内波动，这与图 9 的实验结果一致。另外，从图 9 中可以看出，Whispering 的计算资源消耗量的波动范围较大。这是因为虽然 Whispering 在任务调度过程中没有考虑重用数据缓存情况，但部分服务在所选节点上可进行计算重用。根据图 9 中的实验结果，其资源消耗量最低值在[1.5, 2.4] GHz 附近波动，说明此时有 3 个子任务需要执行计算，另外一个子任务可进行完全重用；波动范围最高值与 TOD 类似，说明此时 4 个子任务均执行完全计算。与其他 2 种算法相比，SF-COL 算法可以在 450 个时隙后实现最低的计算资源消耗。SF-COL 前期一直在探索算力节点的数据缓存情况，希望能够通过计算重用来获得更高的奖励。在 0～450 个时隙，SF-COL 算法探索到了可进行计算重用的节点，因此有部分时隙的计算资源消耗有所降低。在 450 个时隙之后，由于 SF-COL 算法已获得了充分的历史经验，对所有算力节点的计算、传输以及缓存情况均有了一定的了解，因此可以精确地调度具有缓存数据的节点来提供服务。具体表现为计算资源消耗在较低的水平小幅波动，此时的计算资源主要被不可进行计算重用的子任务 1 和子任务 3 消耗，而子任务 2 和子任务 4 的每个时隙都在算力节点上执行完全重用或者部分重用计算。上述实验结果验证了所提算法在降低资源消耗方面的有效性。

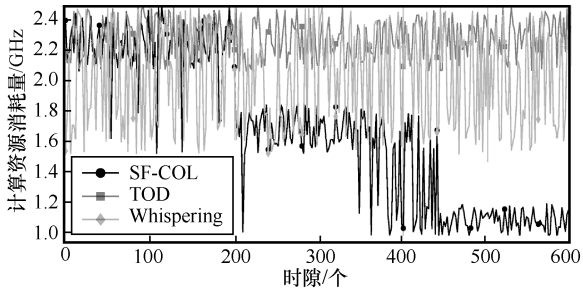

图 9　SF-COL 算法和 TOD、Whispering 算法的计算资源消耗量对比

5　结束语

本文将计算重用技术引入算力网络中，面向时

延敏感的计算密集型应用，综合考虑计算、传输与查找时延，构建了以缩短服务时延为目标的优化问题。为了解决该问题，本文提出了 SF-COL 算法，设计了重用指数来做出重用缓存查找决策。进一步地，基于服务联盟机制进行高效学习，探索不同上下文信息下算力节点的表现情况，在联盟历史经验的基础上做出计算任务调度决策。实验结果表明，与基准算法相比，SF-COL 算法在服务时延、计算资源消耗等方面均实现了更优的性能。

参考文献：

[1] 贾庆民, 胡玉姣, 张华宇, 等. 确定性算力网络研究[J]. 通信学报, 2022, 43(10): 55-64.
JIA Q M, HU Y J, ZHANG H Y, et al. Research on deterministic computing power network[J]. Journal on Communications, 2022, 43(10): 55-64.

[2] LEE J, MTIBAA A, MASTORAKIS S. A case for compute reuse in future edge systems: an empirical study[C]//Proceedings of IEEE Globecom Workshops. Piscataway: IEEE Press, 2020: 1-6.

[3] NOUR B, MASTORAKIS S, MTIBAA A. Compute-less networking: perspectives, challenges, and opportunities[J]. IEEE Network, 2020, 34(6): 259-265.

[4] 中国联通. 中国联通算力网络白皮书[R]. 2019.
China Unicom. China Unicom arithmetic network white paper[R]. 2019.

[5] 国家超级计算济南中心. 算力互联网技术白皮书[R]. 2021.
National Supercomputing Center Jinan. White paper on Internet of computing power technology[R]. 2021.

[6] 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.
LIU Z N, LI K, WU L T, et al. CATS: cost aware task scheduling in multi-tier computing networks[J]. Journal of Computer Research and Development, 2020, 57(9): 1810-1822.

[7] SUN W, LI Z J, WANG Q, et al. FedTAR: task and resource-aware federated learning for wireless computing power networks[J]. IEEE Internet of Things Journal, 2023, 10(5): 4257-4270.

[8] 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.

[9] LU Y L, AI B, ZHONG Z D, et al. Energy-efficient task transfer in wireless computing power networks[J]. IEEE Internet of Things Journal, 2023, 10(11): 9353-9365.

[10] LIU J L, SUN Y K, SU J Q, et al. Computing power network: a testbed and applications with edge intelligence[C]//Proceedings of IEEE Conference on Computer Communications Workshops. Piscataway: IEEE Press, 2022: 1-2.

[11] BELLAL Z, NOUR B, MASTORAKIS S. CoxNet: a computation reuse architecture at the edge[J]. IEEE Transactions on Green Communications and Networking, 2021, 5(2): 765-777.

[12] NOUR B, CHERKAOUI S. How far can we go in compute-less networking: computation correctness and accuracy[J]. IEEE Network, 2022, 36(4): 197-202.

[13] ZHAO S L, ZHANG H B, BHUYAN S, et al. Deja view: spatio-temporal compute reuse for energy-efficient 360° VR video streaming[C]//Proceedings of 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). Piscataway: IEEE Press, 2020: 241-253.

[14] NOUR B, MASTORAKIS S, MTIBAA A. Whispering: joint service offloading and computation reuse in cloud-edge networks[C]//Proceedings of IEEE International Conference on Communications. Piscataway: IEEE Press, 2021: 1-6.

[15] LIU J G, REN J, ZHANG Y M, et al. Efficient dependent task offloading for multiple applications in MEC-cloud system[J]. IEEE Transactions on Mobile Computing, 2023, 22(4): 2147-2162.

[16] JIE Y M, GUO C, CHOO K K R, et al. Game-theoretic resource allocation for fog-based industrial Internet of things environment[J]. IEEE Internet of Things Journal, 2020, 7(4): 3041-3052.

[17] TAN K G, FENG L, DAN G, et al. Decentralized convex optimization for joint task offloading and resource allocation of vehicular edge computing systems[J]. IEEE Transactions on Vehicular Technology, 2022, 71(12): 13226-13241.

[18] XIE Y, SUN Y Y, XU F, et al. The offloading algorithm of mobile edge computing considering mobility in the intelligent inspection scenario[J]. Transactions on Emerging Telecommunications Technologies, 2022, 33(7): e4494.

[19] WANG J, HU J, MIN G Y, et al. Fast adaptive task offloading in edge computing based on meta reinforcement learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(1): 242-253.

[20] BAEK J, KADDOUM G. Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks[J]. IEEE Internet of Things Journal, 2021, 8(2): 1041-1056.

[21] 谭友钰, 陈蕾, 周明拓, 等. 动态雾计算网络中基于在线学习的任务卸载算法[J]. 中国科学院大学学报, 2020, 37(5): 688-698.
TAN Y Y, CHEN L, ZHOU M T, et al. Online learning-based task offloading algorithms for dynamic fog networks[J]. Journal of University of Chinese Academy of Sciences, 2020, 37(5): 688-698.

[22] CHEN X Y, XU C Q, WANG M, et al. A universal transcoding and transmission method for livecast with networked multi-agent reinforcement learning[C]//Proceedings of IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2021: 1-10.

[23] XIAO H, XU C Q, MA Y X, et al. Edge intelligence: a computational task offloading scheme for dependent IoT application[J]. IEEE Transactions on Wireless Communications, 2022, 21(9): 7222-7237.

[24] MA Y X, XU C Q, CHEN X Y, et al. Fairness-guaranteed transcoding task assignment for viewer-assisted crowdsourced livecast services[C]//Proceedings of IEEE International Conference on Communications. Piscataway: IEEE Press, 2021: 1-6.

[25] AUER P, CESA-BIANCHI N, FISCHER P. Finite-time analysis of the multiarmed bandit problem[J]. Machine Learning, 2002, 47(2): 235-256.

[26] LI L H, CHU W, LANGFORD J, et al. A contextual-bandit approach to personalized news article recommendation[C]//Proceedings of the 19th International Conference on World Wide Web. New York: ACM Press, 2010: 661-670.

[27] GENTILE C, LI S, KAR P, et al. On context-dependent clustering of bandits[J]. arXiv Preprint, arXiv: 1608.03544, 2016.

[28] BOURSIER E, PERCHET V. SIC-MMAB: synchronisation involves communication in multiplayer multi-armed bandits[J]. arXiv Preprint, arXiv: 1809.08151, 2018.

[29] ZENG Q S, DU Y Q, HUANG K B, et al. Energy-efficient resource management for federated edge learning with CPU-GPU heterogene-

ous computing[J]. IEEE Transactions on Wireless Communications, 2021, 20(12): 7947-7962.

[30] JALEEL A, THEOBALD K B, STEELY S C, et al. High performance cache replacement using re-reference interval prediction (RRIP)[C]// Proceedings of the 37th annual international symposium on Computer architecture. New York: ACM Press, 2010: 60-71.

[作者简介]

马云霄（1997– ），女，山东泰安人，北京邮电大学博士生，主要研究方向为多媒体传输、边缘计算等。

吴忠辉（1997– ），男，安徽宣城人，北京邮电大学博士生，主要研究方向为区块链、人工智能和边缘计算等。

徐祖云（1999– ），男，安徽六安人，北京邮电大学硕士生，主要研究方向为移动互联网、多媒体通信等。

袁璐洁（1979– ），女，江西南昌人，博士，首都师范大学副教授，主要研究方向为通信网络、计算机系统结构、移动网络等。

许长桥（1977– ），男，江西吉安人，博士，北京邮电大学教授，主要研究方向为移动网络、多媒体通信、网络安全等。