arXiv:2305.05935v1 [cs.DC] 10 May 2023

# 基于协作学习的面向*Kubernetes*的边缘云网络调度

沈世豪，*IEEE* 学生会员，Yiwen Han，*IEEE* 学生会员，王晓飞，*IEEE* 高级会员，王世强，*IEEE* 会员，以及 Victor C.M. Leung，*IEEE* 终身会士。

**摘要—*Kubernetes* (*k8s*) 有望协调分布式边缘资源和集中式云资源，但目前缺乏适用于边缘云网络的专用调度框架。此外，异构资源的层级分布使得 k8s-导向的边缘云网络的建模与调度尤为具有挑战性。本文提出 *KaiS*，一种面向此类边缘云网络的基于学习的调度框架，旨在提高请求处理的长期吞吐率。首先，我们设计了一种协同多行动者演员-评论家算法，以适应去中心化请求分发以及边缘集群内动态变化的调度空间。其次，针对不同的系统规模和结构，我们使用图神经网络嵌入系统状态信息，并将嵌入结果与多个策略网络结合，通过逐步调度降低编排维度。最后，我们采用双时间尺度调度机制来协调请求分发与服务编排，并提出了将上述算法部署为兼容原生 组件的实现*k8s* 设计。基于真实工作负载轨迹的实验表明，*KaiS* 能够成功学习适当的调度策略，不受请求到达模式和系统规模的影响。此外，*KaiS* 能够将平均系统吞吐率提高**15。9%**，同时将调度成本降低**38。4%**，优于基线方法。**

关键词——**边缘计算、Kubernetes、强化学习、调度算法。**

稿件于2021年4月19日收到；于2022年7月22日和2023年3月23日修订；于2023年4月2日接受。王晓飞获得国家自然科学基金委员会（项目编号：62072332）、国家自然科学基金青年项目（项目编号：62002260）、中国博士后科学基金（项目编号：2020M670654）和天津信创海河实验室（项目编号：22HHX- CJC00002）的资助。Victor C.M. Leung获得广东省珠江人才计划引进项目（项目编号：2019ZT08X603）、广东省珠江人才计划（项目编号：2019JC01X235）、深圳科学与

创新科技署（项目编号 R2020A045）和加拿大自然科学与工程研究理事会（项目编号 RGPIN-2019- 06348）。本文的初步版本题为“面向 Kubernetes 的边缘云系统的定制化学习调度”，曾发表于 IEEE 国际计算机通信会议（INFOCOM），2021 [1]。本文在先前工作的基础上进行了扩展，完善了详细描述和系统设计，并增加了有关出队策略和编码方法的细粒度实验。（通讯作者：王晓飞）

沈世豪、Yiwen Han 和王晓飞就职于中国天津市天津大学智能与计算学部（邮编 300350）（电子邮箱：{shenshihao、hanyiwen、xiaofeiwang}@tju.edu.cn）。

王世强就职于美国纽约州约克敦高地的 IBM T. J. Watson研究中心（邮编 10598）（电子邮箱：shiqiang.wang@ieee.org）。

Victor C.M. Leung 任职于中国深圳市 518052 的深圳大学计算机科学与软件工程学院，同时任职于加拿大温哥华 V6T 1Z4 的不列颠哥伦比亚大学电气与计算机工程系（电子邮箱：vleung@ieee.org）。

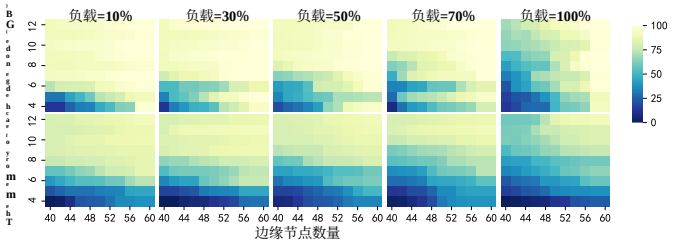



Fig. 1. The model of system throughput is highly complex and non-linear.

## I. Introduction

### A. 背景与问题陈述

为实现敏捷服务响应并减轻骨干网络的负担，边缘计算与云计算正逐渐融合，以尽可能靠近请求生成的位置托管服务，从而实现这一目标 [2]。

边缘云网络通常构建于 *Kubernetes* (*k8s*)[3]–[6] 之上，旨在无缝整合边缘和云端的分布式计算资源与分层计算资源[7]。因此，要支持高效的边缘云网络，一个基本问题是：如何在 [8]架构内进行请求调度和服务编排（放置） [9] 。*k8s* 架构。然而，原生 *k8s* 架构缺乏管理分布式边缘集群和集中式云集群中异构资源的能力。此外，现有的定制边缘云框架（例如，*KubeEdge* [4], *OpenYurt* [5] 和 *Baetyl* [6]）基于*k8s*，并未解决上述调度问题。

为满足各种请求，边缘云网络需要管理分布在边缘和云端的相应服务实体，同时还要能够确定这些请求应在哪里处理。尽管*k8s* 是管理云端部署的服务最流行的工具，但它目前尚无法同时适配边缘和云基础设施，也无法支持分布式边缘的请求调度。 在这种情况下，高效边缘云网络的关键在于（*i*）调整*k8s* 组件并扩展其现有逻辑，以连接分布式边缘与云端；以及（*ii*）设计能够适用于*k8s*的调度算法。

### B. 现有技术的局限性与研究动机

大多数用于请求调度和服务编排的调度解决方案依赖于对...

...进行精确建模或预测服务响应时间、网络波动、请求到达模式，

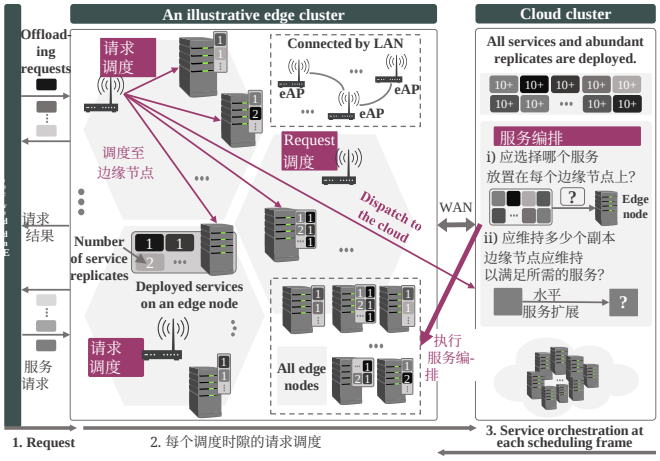



图 2。面向 *Kubernetes*的边缘云网络中的调度。

及其他因素 [10]–[12]。然而，(*i*) 异构边缘节点与云集群在不确定网络环境中相连，实际上构成了一个动态的分层计算系统。如图 1 所示，由原生 *k8s*管理的系统行为，即该系统的平均吞吐率，会随可用资源和请求负载发生显著变化（详细设置请参阅第 V 节）。更重要的是，(*ii*) 描述这一行为的底层模型高度非线性，且并非易于处理。然而，尽管有丰富的历史数据可用，仍难以准确估算这些指标 [7], [13] ，进而针对特定的请求到达、系统规模与结构或异构资源设计调度策略。此外，(*iii*) 很少有解决方案会仔细考虑所提出的调度框架或算法是否 兼容实际部署环境，即是否兼容 *k8s* 或其他平台，以便与现有云基础设施集成。因此，我们需要一种面向 *k8s* 的边缘云网络调度框架，且不依赖于关于系统动态特性的假设。

### *C.* 技术挑战与解决方案

首先，我们展示基于学习的方法 [14], [15] 如何通过自动学习有效的系统调度策略来提高整体系统效率，以应对服务请求的随机到达。我们提出了*KaiS*，一种面向 *k.8 s*且基于学习的边缘云系统调度框架。仅给定一个高级目标，例如最大化服务处理的长期吞吐量，*KaiS* 通过系统运行经验自动学习复杂的调度策略，而不依赖于对系统执行参数和运行状态的假设。

为了引导*KaiS* 学习调度策略，我们需要从以下几个方面定制学习算法：多个智能体的协同学习、系统状态的有效编码、调度动作的降维等。

对于请求调度，如图 2 所示，*KaiS* 需要扩展至数百个分布式边缘接入点（eAP）在系统中 [16]。传统学习算法，例如深度Q网络 [17] 和深度确定性策略梯度 [18], 通常使用一个集中式学习智能体，因此不适用于 *KaiS* ，因为分布式边缘接入点会导致分发动作空间爆炸 [19]。为确保及时调度，*KaiS* 要求在请求到达的边缘接入点以去中心化方式确定调度动作，而非采用集中式方法 [19]。因此，我们采用多智能体深度强化学习（MADRL） [20]并在每个边缘接入点部署一个分发智能体。

然而，这类设置（*i*）要求大量智能体在每个时刻与系统进行交互，并且（*ii*）具有取决于可用系统资源的不同调度动作空间，这使得这些智能体难以学习调度策略。因此，我们将集中式评论家与分布式行动者解耦，在评论家训练期间输入全局观测，以稳定每个智能体的学习过程，并为行动者设计策略上下文过滤机制，使其能够响应调度动作空间的动态变化。

由于不同的出队策略会通过影响请求的排队延迟而影响请求调度的性能，因此高效的出队策略对于优化系统性能至关重要。边缘云网络中的请求调度涉及两个问题。第一个问题是如何将尚未调度的请求出队以进行调度，即 eAP 从终端设备接收请求后，需要根据出队策略将请求出队并进行调度。另一个问题是如何将已调度的请求出队以进行处理，即边缘节点和云集群接收到已调度的请求后，需要根据出队策略将请求出队并进行处理。然而，请求的多样性和环境的动态变化，使得一些经典策略（例如先进先出策略或贪心算法）难以应对。因此，我们进一步考虑不同请求队列中请求的特点，并设计基于折扣经验的出队策略，以计算请求的优先级。

此外，*KaiS* 必须根据系统的全局资源编排数十种甚至更多服务，并适应不同的系统规模和结构。因此，*KaiS* 要求我们的学习技术能够（*i*）编码海量且多样的边缘云系统状态信息，并且（*ii*）表示更大、更复杂的边缘云网络编排动作空间。因此，我们采用图神经网络（GNNs） [21] 和多个策略网络 [22] 分别编码系统信息并降低编排维度，无需手动进行特征工程。与使用原始状态和固定动作空间的常见深度强化学习方案相比，我们的设计能够降低模型复杂度，从而有利于学习调度策略。

### *D.* 主要贡献

本文的部分研究成果已发表于会议版本 [1]。在前期工作的基础上，本文通过设计出队策略、完善系统设计并增加更多实验结果，进一步拓展了相关工作。

与最先进的研究相比，*KaiS* 的改进主要体现在以下几个方面。(*i*) 大多数现有研究只关注请求调度，例如

[23]，[24]，或服务编排，例如 [25]，[26]，，因此忽略了服务与请求之间的相互作用。相比之下，*KaiS* 通过双时间尺度调度框架实现了这两个方面的协同优化。（*ii*）一些基于模拟场景设计的算法更偏理论，实际应用可能有限，例如只考虑单一类型的资源限制 [25]，，或假设能够准确获知每个请求所需的计算量 [27], 等。相比之下，*KaiS* 无需对系统状态或执行参数作出假设，并能根据经验自动学习恰当的策略。（*iii*）许多研究侧重于理论和方法层面，却未考虑如何将算法部署到实际应用场景中 [27]，[28]。相比之下，*KaiS* 设计了与 k8s 交互的非侵入式组件和模式（如第 IV-B 节所示），并使用来自阿里巴巴的真实工作负载跟踪数据 [29] 进行评估，从而确保其适用于实际部署场景。 总之，我们的主要贡献如下：

- 一种用于去中心化请求调度的协同多智能体演员-评论家算法，采用策略上下文过滤机制，可处理动态调度动作空间，以应对时变系统资源。
- 一种基于折扣经验的出队策略，可通过确定请求优先级来调整请求队列中请求的排队时间，从而减少请求因超时而失败的情况。
- 一种用于服务编排的基于图神经网络的策略梯度算法，该算法利用图神经网络高效编码系统信息，并通过逐步调度使用多个策略网络来降低编排维度。
- 面向*k8s*边缘云网络的定制学习算法双时间尺度调度框架实现，即*KaiS*，以及在各种场景下使用真实工作负载轨迹（阿里巴巴集群跟踪数据）对*KaiS* 进行评估，并与基线方法进行比较。 [29]

在下文中，第 II 节介绍调度问题。第 III 节和第 IV 节详细阐述算法和实现设计。第 V 节展示实验结果。最后，第 VI 节回顾相关工作，第 VII 节总结全文。

## II. 调度问题陈述

我们重点关注边缘云网络中的请求调度与服务编排，以提高其长期吞吐率，即在系统长期运行期间，满足时延要求的已处理请求数占总请求数的比例。表 I 列出了我们将使用的主要符号。

### *A.* 边缘云网络

边缘计算和云计算不应被视为互斥的。因此，为了服务网络中分布的大量终端设备所发出的请求，端-边-云协同计算应运而生。一方面，为了满足时延敏感型服务的需求，可以将其中一些服务从云计算基础设施（云端）迁移到分布式边缘集群中部署，该集群由多个相邻的边缘节点组成。在另一方面，当边缘集群无法满足服务请求的需求时，云端可以提供强大的备用处理能力和高级别的全局管理。

如图 2 所示，相邻的 eAP 和边缘节点构成一个资源池，即一个边缘集群，并与云端相连。当请求到达 eAP 时，边缘集群会与云集群协同处理这些请求 $C$。为清晰起见，我们仅以一个边缘集群为例来说明 *KaiS*，并考虑地理分布式边缘集群之间不存在协作的情况。具体而言，每个边缘集群负责一个地理区域，不同地理区域的边缘集群之间没有协作。不过，通过为每个边缘集群维护一个服务编排器，*KaiS* 可以轻松扩展以支持地理分布式边缘集群。

- 边缘集群与边缘节点。为处理请求，边缘集群应托管相应的服务实体。一个边缘集群由一组 $\mathcal{B} = \{b_1, b_2, \ldots, b_B\}$ 个 eAP 组成，索引为 $b$，而 $\mathcal{N}_b = \{n_1, n_2, \ldots, n_{N_b}\}$ 是连接到 eAP $b$并由其管理的边缘节点集合。边缘集群中的所有边缘节点表示为 $\mathcal{N} = \sum_{b \in \mathcal{B}} \mathcal{N}_b$。所有 eAP 及其关联的边缘节点均通过局域网 (LAN) 连接。到达边缘的请求可由接纳该请求的 eAP 调度至边缘节点或云端进行处理。
- 云集群。与边缘相比，云集群拥有充足的计算和存储资源，并通过广域网（WAN）连接到 eAP，因此能够处理边缘集群无法处理的请求。此外，它负责管理所有地理分布式边缘集群，包括根据系统的可用资源对每个边缘集群中的所有服务实体进行编排。

### *B.* 提高长期系统吞吐量

我们受到 [10] 的启发，设计了一种双时间尺度机制，用于调度请求调度和服务编排。在每个时隙内 $t$，根据可用资源将服务请求调度至边缘节点或云端，同时在更长的时间帧 $\tau$内执行服务编排，这些时间帧为 $\beta$ 一个时隙的倍。 为了提升整体性能，*KaiS* 针对性地采用深度强化学习（DRL），通过运行系统获取的经验学习调度策略，并将学习到的策略编码到神经网络中。

**在 *eAP* 处调度请求。**对于每个 eAP $b \in$ B，都会维护一个请求队列$_b$ Q。此外，还会实施一种随时间变化的动态调度策略$\hat{}\pi_{b,\ t}$ 并采用出队策略。首先，时延敏感服务请求以随机方式到达队列 Q，然后根据出队策略 ，将优先级最高的请求，$_t$ 出队。此后，根据 $\hat{}\pi_{b,\ t}$，在每个时隙 $t$，每个 eAP $b$ 都会将请求$r_{b,\ t}$ 调度至已部署所需服务实体且资源充足的边缘节点，或调度至计算资源充足的云集群进行处理。处理每个请求都会消耗边缘节点或云端的计算资源和网络带宽。此外，调度

TABLE I
MAIN NOTATIONS

| Notations | Descriptions |
|---|---|
| $a_{b,t}$ | eAP $b$ 在时隙 $t$采取的请求调度动作。 |
| $b$ | eAP 的索引。 |
| $\mathcal{B}$ | 边缘接入点集合。 |
| $C$ | 云集群。 |
| $C^{r_{b,\ w}}$ | 请求 $r_{b,\ w}$ 的出队优先级。 |
| $d_{w,n}$ | 边缘节点 $n$ 中服务 $w$ 的副本数量。 |
| $\mathcal{E}_{w,t}$ | 完成类型为 $w$ 的请求所需的预测时间。 |
| $H$ | 每帧中的高价值节点数量。 |
| $n$ | 边缘节点的索引。 |
| $\mathcal{N}$ | 边缘集群中的所有边缘节点。 |
| $\mathcal{N}_b$ | 连接到 eAP $b$ 的边缘节点集合。 |
| $Q_b$ | eAP $b$ 中的请求队列。 |
| $\hat{Q}_c$ | 云集群中的请求队列。 |
| $Q'_n$ | 边缘节点中的请求队列。 |
| $r_{b,t}$ | 时隙 $t$ 时 eAP $b$ 的当前调度请求。 |
| $\mathcal{R}_{c,\tau}$ | 云端在帧 $\tau$ 中收到的请求集合。 |
| $\mathcal{R}_{n,\tau}$ | 节点 $n$ 在帧 $\tau$ 中收到的请求集合。 |
| $t$ | 时隙索引。 |
| $\mathcal{T}_{b,w,t}$ | The average time consumption corresponding to the 请求类型 $w$ 在时隙 $t$ 根据数据计算得到 由 eAP $b$ 收集。 |
| $w$ | The index of services. |
| $\mathcal{W}$ | The set of all services. |
| $\lambda$ | 超时请求的比例。 |
| $\lambda_e$ | 折扣因子。 |
| $\lambda'$ | 惩罚因子。 |
| $\xi$ | 中央处理器和内存使用率的标准差 所有边缘节点的。 |
| $\hat{\pi}_{b,t}$ | 动态请求调度策略。 |
| $\tau$ | 帧索引。 |
| $\Upsilon_\tau(\hat{Q}_c)$ | 已及时处理的请求数量 由云端处理。 |
| $\Upsilon_\tau(Q_b)$ | 到达 eAP $b$ 的请求数量。 |
| $\Upsilon_\tau(Q'_n)$ | 已及时处理的请求数量 由边缘节点 $n$。 |
| $\Phi'$ | 长期吞吐率】【。 |
| $\Psi$ | 边缘接入点的出队策略。 |
| $\Psi'$ | 边缘节点和云集群的出队策略。 |

将请求调度至云端可能会导致额外的传输延迟，因为云端距离终端设备（即请求的产生地）较远。每个边缘节点和云端都会维护一个已调度请求队列，即 $\{Q_n : n \in \mathcal{N}\}$ 表示边缘节点^，以及 队列$_c$ 表示云端。此后，服务实体会根据出队策略^，持续从请求队列$Q_n$ 和队列$_c$ 中选取请求进行处理，$\Psi'$直至没有可用的服务实体能够处理队列中的任何请求。为确保及时调度，理想情况下应由请求最先到达的 eAP 独立执行请求调度，而不是由云端或边缘以集中式方式做出调度决策，因为这可能会导致较高的调度延迟 [30]。对于未能及时处理的请求，系统会在每个时隙将其丢弃。

边缘集群中的服务编排。由于每种类型的请求都需要依赖相应类型的服务实体进行处理，因此将服务实体部署在边缘节点上会占用边缘节点的资源。因此，受单个边缘节点资源限制，无法在每个边缘节点上部署所有类型的 $\mathcal{W} = \{1, \ldots, w, \ldots, W\}$服务实体。此外，由于单个服务实体存在资源限制，当请求负载过高时，它没有足够的资源进行处理较高。因此，可以部署该类型服务实体的多个副本来分担请求，这需要对服务进行编排。在这种情况下，需要对边缘集群中的服务实体进行编排，其中包括以下问题：（$i$）应将哪种服务部署在哪个边缘节点上，以及（$ii$）边缘节点应为该服务维护多少个副本。此外，不同时刻到达的服务请求可能具有不同的模式，导致不同服务的需求强度随时间变化。因此，调度应能够捕捉并识别这些模式，并以此为依据编排服- 务，以满足随机到达的请求。与请求调度不同，由于资源受限，在边缘集群中过于频繁地进行大规模服务编排可能导致系统不稳定并产生较高的运营成本 [10]。基于这些原因，更合适的解决方案是由云端采用动态调度策略为边缘侧执行服务编排$\tilde{\pi}_\tau$，且每一帧 $\tau$执行一次。因此，云端可以采用动态调度策略 $\tilde{\pi}_\tau$，在每一帧 $\tau$ 为边缘侧执行服务编排，以确定边缘节点 $n$ 上服务 $w$ 的副本数量，记为 $d_{w,\ n} \in \mathbb{N}$。如果 $d_{w,\ n} = 0$，则表示边缘节点 $n$ 未部署服务 $w$。

调度目标是最大化长期系统吞吐量 $\Phi = \sum_{\tau=0}^{\infty} \sum_{n \in \mathrm{N}} \Upsilon_\tau(\mathrm{Q}'_n) + \Upsilon_\tau(\hat{\mathrm{Q}}_c)$，其中 $\Upsilon_\tau(\mathrm{Q}'_n)$、$\Upsilon_\tau(\hat{\mathrm{Q}}_c)$分别表示在帧$\tau$内由边缘节点 $n$ 或云端及时处理的请求数量。具体而言，我们首先将节点 $n$ 和云端在帧 $\tau$ 内接收到的请求集合分别表示为 $\mathrm{R}_{n,\tau}$ 和 $\mathrm{R}_{c,\tau}$，然后，对于每个请求 $r \in \mathrm{R}_{n,\tau}$，都存在一个时延要求 $t_r$ 和一个实际完成时间 $t'_r$。此外，我们定义，如果 $t'_r \le t_r$，则 $\Gamma(r) = 1$；否则，$\Gamma(r) = 0$。最后，我们得到 $\Upsilon_\tau(\mathrm{Q}'_n) = \sum_{r \in \mathrm{R}_{n,\tau}} \Gamma(r)$，$\Upsilon_\tau(\mathrm{Q}'_c) = \sum_{r \in \mathrm{R}_{c,\tau}} \Gamma(r)$。为了避免 $\Phi \to \infty$，我们采用一种更为现实的指标，即长期系统吞吐率 $\Phi' \in [0, 1]$，它表示在时延要求内完成的请求数量与请求总数之比。长期吞吐率 $\Phi'$ 可表示为 $\Phi' = \Phi / \sum_{\tau=0}^{\infty} \sum_{b \in \mathrm{B}} \Upsilon_\tau(\mathrm{Q}_b)$，其中$\Upsilon_\tau(\mathrm{Q}_b)$表示在帧 $\tau$内到达 eAP $b$的请求数量。在这种情况下，我们的请求调度和服务编排调度问题可以表述为

$$\max_{\{\hat{\pi}_{b,t}: b \in \mathcal{B}\}, \tilde{\pi}_\tau} \Phi' = \max_{\{\hat{\pi}_{b,t}: b \in \mathcal{B}\}, \tilde{\pi}_\tau} \Phi / \sum_{\tau=0}^{\infty} \sum_{b \in \mathcal{B}} \Upsilon_\tau(Q_b), \quad (1)$$

为便于说明，我们使用调度策略 $\{\hat{\pi}_{b,t} : b \in \mathrm{B}\}$ 和 $\tilde{\pi}_\tau$，而不是一系列调度变量来表示该问题。与 [10], 中的问题相比，我们的调度问题更加复杂，因为其中涉及整数调度变量。关于此类长期调度问题的约束条件和 NP-hard 证明，详见 [10]。在本文中，我们为*KaiS* 量身定制学习算法，以提高长期系统吞吐率。

## III. 算法设计

*KaiS* 的整体训练和调度流程如算法 1 所示。下面将介绍请求调度和服务编排的技术细节。详细的训练设置见第 IV-C 节。

**算法 1：** *KaiS* 的训练与调度过程

Initialize the system environment and neural networks.
**for** slot $t = 1, 2, \ldots$ **do**
  **if** frame $\tau$ begins **then**
    获取奖励 $\tilde{u}_{\tau-1}$ 并存储 $[\tilde{s}_{\tau-1}, \tilde{\boldsymbol{a}}_{\tau-1}, \tilde{u}_{\tau-1}]$;
    使用图神经网络将系统状态嵌入为式 (10);
    选择 $H$ 高价值边缘节点（$\tilde{a}^{\bullet}_{\tau}$），并
     计算其服务扩缩容动作（$\tilde{\boldsymbol{a}}^{\star}_{\tau}$）
     分别使用策略网络 $\theta_g$ 和 $\theta_q$;
    执行编排动作 $\tilde{\boldsymbol{a}}_{\tau} = (\tilde{a}^{\bullet}_{\tau}, \tilde{\boldsymbol{a}}^{\star}_{\tau})$;
    根据公式（12）更新图神经网络和策略网络;
  **并行执行**每个 eAP 智能体 $b \in B$
    **如果** $Q_b == \varnothing$，**则**
      **继续**
    更新请求队列 $Q_b$ 并获取奖励 $\hat{u}_{b,\,t-1}$;
    存储 $[\hat{\boldsymbol{s}}_{b,\,t-1}, \hat{a}_{b,\,t-1}, A(\hat{\boldsymbol{s}}_{b,\,t-1}, \hat{a}_{b,\,t-1}), \hat{u}_{b,\,t-1}, \boldsymbol{F}_{b,\,t-1}]$ 针对 $\theta_p$（行动者）;
    出队请求 $r_{b,\,t}$ 根据 $\Psi$;
    计算资源上下文 $\boldsymbol{F}_{b,\,t}$ 使用公式 (4);
    执行调度动作 $\hat{a}_{b,\,t}$ 针对 $r_{b,\,t}$ 使用公式 (5);
  根据 $\Psi'$ 执行调度的请求;
  存储 $[\hat{s}_{t-1}, V^*(\hat{s}_t; \theta'_v, \pi)]$ 用于 $\theta_v$（评论家）;
  更新神经网络 $\theta_p$（行动者）和 $\theta_v$（评论家）
   分别使用公式（6）和（2）进行集中式更新;
  同步 $\theta_p$ 定期同步至分布式边缘接入点。

### *A.* 面向去中心化请求分发的定制化多智能体深度强化学习

请求调度是指让每个 eAP 独立决定由哪个边缘节点或云端为到达的请求提供服务。调度的目标是通过（*i*）平衡边缘节点之间的工作负载，并在适当情况下（*ii*）进一步将部分请求卸载到云端，从而最大化长期系统吞吐率 $\Phi'$。

*1*）马尔可夫博弈建模：为了采用多智能体深度强化学习（MADRL），我们将边缘接入点（eAPs）独立执行请求调度的过程建模为一个马尔可夫博弈 G，其中智能体为 eAP 智能体 $\mathrm{B} = \{1, 2, \ldots, B\}$。形式上，博弈 $\mathrm{G} = (\mathrm{B}, \hat{\mathrm{S}}, \hat{\mathrm{A}}, \hat{\mathrm{P}}, \hat{\mathrm{U}})$ 定义如下。

- 状态。$\hat{\mathrm{S}}$ 是状态空间。在每个时隙 $t$，我们定期为每个 eAP 智能体 $b$ 构建一个局部状态 $\hat{\boldsymbol{s}}_{b,\,t}$，其由以下部分组成：(*i*) 当前调度请求 $r_{b,\,t}$ 的服务类型和延迟要求；(*ii*) eAP $b$ 处等待调度的请求队列信息 $\mathrm{Q}'_{b,\,t}$；(*iii*) 边缘节点 $\mathrm{N}_b$ 上未处理请求的队列信息，$\{\mathrm{Q}'_{n_b,\,t} : n_b \in \mathrm{N}_b\}$；(*iv*) $\mathrm{N}_b$ 的剩余中央处理器、内存和存储资源；(*v*) $\mathrm{N}_b$ 的数量，即 $|\mathrm{N}_b| = N_b$；以及 (*vi*) 测得的 eAP 与云端之间的网络延迟。同时，对于集中式评论家训练，我们 $\hat{}$ 维护全局状态 $\hat{\boldsymbol{s}}_t \in \mathrm{S}$，其中包括（*i*）所有 eAP B 和边缘节点 N 的上述信息，而不仅仅是 eAP $b$ 和 $\mathrm{N}_b$，以及（*ii*）云集群中未处理请求的队列信息 $\mathrm{Q}'_{C,t}$ $C$。
- 动作空间。G 的联合动作空间为 $\mathrm{A} = \mathrm{A}_1 \times \ldots \times \hat{} \hat{} \mathrm{A}_b \times \ldots \times \mathrm{A}_B$。对于边缘集群，我们将所有可用的边缘节点视为一个资源池，即启用的 eAP 之间的协作 $\hat{}$。在这种情况下，$\mathrm{A}_b$ 包含 $N+1$ 个离散动作，表示为 $\{i\}_0^N$，其中 $a_{b,t} = 0$ 和 $a_{b,t} \in \mathrm{N}$ 分别指定将请求调度到云端或边缘节点。在每个时隙 $t$，我们使用 $\hat{\boldsymbol{a}}_t = (a_{b,t} : b \in \mathrm{B})$ 表示所有请求所需的联合调度动作。

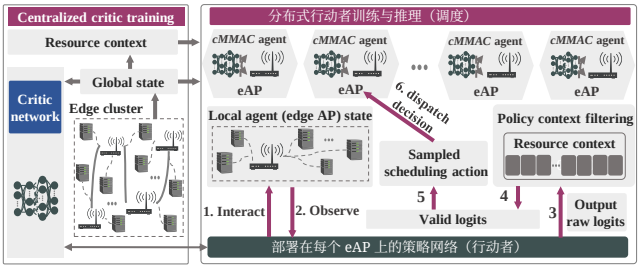



图 3. 用于请求调度的多智能体协同演员-评论家 。

在所有边缘接入点进行调度。请注意，多个请求可能会在边缘接入点中排队（$\mathrm{Q}_b, b \in \mathrm{B}$），我们只允许每个边缘接入点代理在一个时隙内调度一个请求。同时，对于 *KaiS*，我们将时隙设置为适中的值（参见第 IV 节），以确保及时服务请求到达。

- 奖励函数。同一边缘集群中的所有智能体共享奖励函数 $\hat{U} = \hat{\mathrm{S}} \times \hat{\mathrm{A}} \to \mathbb{R}$，即对于所有 $b \in \mathrm{B}$，均有 $\hat{} U_b = \hat{U}$。每个智能体都希望最大化自身的期望回报 $\mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \hat{u}_{b,\,t+i}\right]$，其中 $\hat{u}_{b,t}$ 是与动作 $a_{b,t}$ 相关联的第 $b$ 个智能体的即时奖励，且 $\gamma \in (0, 1]$ 为折扣因子。即时奖励定义为 $\hat{u}_{b,t} = \mathrm{e}^{-\lambda - \varepsilon\nu}$。具体而言，(*i*) $\lambda \in [0, 1]$ 表示在 $[t, t+1]$ 期间违反时延要求的请求比例；(*ii*) $\nu = 1/(1+\mathrm{e}^{-\xi}) \in [0$。其中，$\xi \in \mathbb{R}_{\geq 0}$ 表示所有边缘节点的中央处理器和内存使用率的标准差，(*iii*) $\varepsilon$ 则是用于控制边缘节点间负载均衡程度的权重。引入 $\nu$ 是为了稳定系统，避免过多负载被施加到某些边缘节点上。当 $\nu$ 更接近 0.5，即 $\xi \to 0$时，边缘节点的负载更加均衡，从而为调度留出更多空间。该奖励旨在提升长期吞吐量，同时确保边缘侧的负载均衡。
- 状态转移概率。$\hat{\mathrm{P}}$ 我们使用 $p(\hat{s}_{t+1} | \hat{s}_t, \hat{\boldsymbol{a}}_t)$ 表示给定联合调度动作 $\hat{\boldsymbol{a}}_t$ 时，从状态 $\hat{\boldsymbol{s}}_t$ 转移到 $\hat{\boldsymbol{s}}_{t+1}$ 的转移概率。在 G 中，动作是确定性的，即如果 $a_{b,t} = 2$，智能体 $b$ 会在时隙 $t+1$将当前请求调度到边缘节点 2。

*2)* 协调式多智能体行动者-评论家：训练这些分发智能体面临的挑战包括：(*i*) 每个智能体所处的环境都是非平稳的，因为其他智能体也在同时学习并影响环境。具体而言，每个智能体通常会学习自身随时间不断变化的策略，[20],这增加了协调的难度；(*ii*) 每个智能体的动作空间会动态变化，因为其可行的调度选项会随可用系统资源而变化，导致普通的深度强化学习算法无法处理。例如，如果边缘节点 $n$ 的内存在时隙 $t$耗尽，动作空间就不应包含将请求调度到边缘节点 $n$的选项。

因此，我们设计了协同多智能体演员-评论家（*cMMAC*），如图 3 所示：(*i*) 采用集中式评论家和分布式行动者来协调学习，即所有智能体在训练评论家时共享一个集中式状态价值函数，而在分布式行动者训练和推理期间，每个行动者

仅观察局部状态。(*ii*) 通过策略上下文过滤，我们可以调整其策略，使其能够适应动态动作空间，并在智能体之间建立明确的协调机制，以促进训练顺利进行。具体说明如下。

• 集中式状态价值函数（评论家）。eAP 智能体共享的状态价值函数可通过最小化由贝尔曼方程推导出的损失函数获得 [14]，如下所示：

$$L(\theta_v) = \left(V_{\theta_v}(\hat{\boldsymbol{s}}_{t-1}) - V^*(\hat{\boldsymbol{s}}_t; \theta'_v, \pi)\right)^2, \tag{2}$$

$$V^*(\hat{\boldsymbol{s}}_t; \theta'_v, \pi) = \sum_{a_{b,t-1} \in \hat{\boldsymbol{a}}_{t-1}} \pi\left(a_{b,t-1} \mid \hat{\boldsymbol{s}}_{b,t-1}\right)\left(\hat{u}_{b,t-1} + \gamma V_{\theta'_v}(\hat{\boldsymbol{s}}_{b,t})\right), \tag{3}$$

其中，$\theta_v$ 和 $\theta'_v$ 表示价值网络和目标价值网络的参数，以及 $\pi\left(a_{b,t-1} \mid \hat{\boldsymbol{s}}_{b,t-1}\right)$ 表示采用 $a_{b,t-1}$ 在 $\hat{\boldsymbol{s}}_{b,t-1}$的概率。总共有 $B$个 eAP 智能体，每个时隙有 $B$ 个唯一状态价值 $\{V(\hat{\boldsymbol{s}}_{b,t-1}) : b \in \mathrm{B}\}$。每个状态价值输出 $V(\hat{\boldsymbol{s}}_{b,t-1})$ 是智能体 $b$ 在时隙 $t$收到的期望回报。为稳定状态价值函数，我们固定一个由 $\theta'_v$ 参数化的目标价值网络 $V^*$，并在每个训练回合结束时更新它。

•策略上下文过滤（演员）。由于神经网络结构的限制，行动者的动作空间大小固定。然而，系统运行期间实际可用的动作空间大小会动态变化。为解决这一问题，我们为行动者设计了策略上下文过滤。具体而言，在边缘云网络运行期间，随机到达的请求会导致边缘节点的可用资源动态变化。因此，如果边缘节点的可用资源不足以处理当前请求，将请求调度到该节点就是无效动作。为尽可能避免此类无效动作，我们会在调度前为每个 eAP 智能体计算资源上下文$\boldsymbol{F}_{b,t} \in \{0, 1\}^{N+1}$。该上下文是一个用于过滤无效调度动作的二进制向量。具体而言，$\boldsymbol{F}_{b,t}$ 中元素的值定义如下：

$$\left[\boldsymbol{F}_{b,t}\right]_j = \begin{cases} 1, & \text{if edge node } j \text{ is available,} \\ 1, & \text{if } j = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

其中，(*i*) $[\boldsymbol{F}_{b,t}]_j (j = 1, \ldots\ldots, N)$ 表示将当前请求调度到第 $j$ 个边缘节点的有效性，而 (*ii*)$[\boldsymbol{F}_{b,t}]_0$表示云集群（$j = 0$）始终是请求调度的有效动作，即 $\left[\boldsymbol{F}_{b,t}\right]_0 \equiv 1$。此外，还通过基于资源上下文屏蔽可用动作空间来实现智能体之间的协调

$\boldsymbol{F}_{b,t}$。为继续讨论，我们首先使用 $\boldsymbol{p}(\hat{\boldsymbol{s}}_{b,t}) \in \mathbb{R}^{N+1}$表示在状态 $\hat{\boldsymbol{s}}_{b,t}$条件下，行动者策略网络为第 $b$ 个行动者输出的原始 logits。接下来，我们令$\bar{\boldsymbol{p}}(\hat{\boldsymbol{s}}_{b,t}) = \boldsymbol{p}(\hat{\boldsymbol{s}}_{b,t}) * \boldsymbol{F}_{b,t}$，其中运算 $*$表示逐元素乘法，用以表示考虑行动者 $b$ 的资源上下文后得到的有效 logits。此外，注意输出 logits $\boldsymbol{p}(\hat{\boldsymbol{s}}_{b,t}) \in \mathbb{R}_{>0}^{N+1}$被限制为正值，以实现

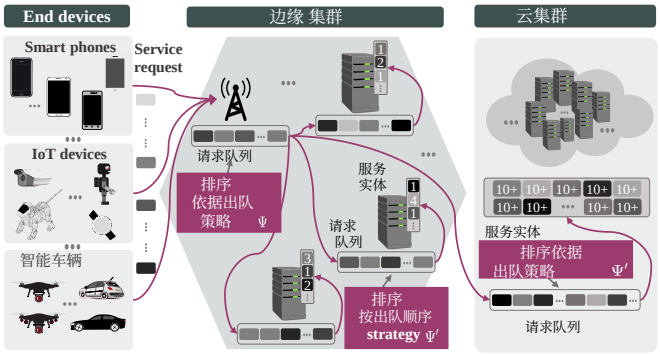



图 4. 边缘集群和云集群中的出队策略。

有效掩码。因此，代理 $b$ 执行有效调度动作的概率可表示为：

$$\pi_{\theta_p}\left(a_{b,t} = j \mid \hat{\boldsymbol{s}}_{b,t}\right) = \left[\bar{\boldsymbol{p}}\left(\hat{\boldsymbol{s}}_{b,t}\right)\right]_j = \frac{[\bar{\boldsymbol{p}}(\hat{\boldsymbol{s}}_{b,t})]_j}{\|\bar{\boldsymbol{p}}(\hat{\boldsymbol{s}}_{b,t})\|_1}, \tag{5}$$

其中 $\theta_p$ 是行动者策略网络的参数。最后，可以推导出策略梯度 $\nabla_{\theta_p} J(\theta_p)$，并且可以计算优势函数 $A(\hat{\boldsymbol{s}}_{b,t}, a_{b,t})$(与在 [31] 中使用目标网络参数有关)。

$$\nabla_{\theta_p} J\left(\theta_p\right) = \nabla_{\theta_p} \log \pi_{\theta_p}\left(a_{b,t} \mid \hat{\boldsymbol{s}}_{b,t}\right) A\left(\hat{\boldsymbol{s}}_{b,t}, a_{b,t}\right), \tag{6}$$

$$A\left(\hat{\boldsymbol{s}}_{b,t}, a_{b,t}\right) = \hat{u}_{b,t+1} + \gamma V_{\theta'_v}\left(\hat{\boldsymbol{s}}_{b,t+1}\right) - V_{\theta_v}\left(\hat{\boldsymbol{s}}_{b,t}\right). \tag{7}$$

*3*）请求优先级的出队策略：对于请求队列，需要考虑两个问题：（*i*）eAP 接收到随机到达的请求后，选择哪个请求进行调度；（*ii*）边缘节点和云集群接收到调度的请求后，如何确定这些请求的处理顺序。如图 4 所示，我们设计了如下出队策略。

• 边缘接入点中的出队策略 $\Psi$。对于每种服务类型 $w \in \mathrm{W}$，我们将完成一条类型为 $w$ 的请求所需的估计时间定义为 $\mathrm{E}_{w,t}$。为适应环境动态变化，$\mathrm{E}_{w,t}$ 可以动态更新，并设计了一个折扣因子 $\lambda_e \in (0,1)$，以提高近期经验的权重。这里的经验特指系统生成的历史数据，例如每条已完成请求的传输时间、执行时间和排队时间。为在时隙 $t$更新 $\mathrm{E}_{w,t}$ ，边缘接入点 $b$ 首先收集请求从出队到完成的实际耗时。随后，边缘接入点$b$ 按请求类型对收集的数据进行分类，以计算请求类型 $w$的实际平均耗时 $\mathrm{T}_{b,w,t}$ 。如果边缘接入点 $b$ 在时隙 $t$未收集到服务类型 $w$的任何请求耗时，则 $\mathrm{T}_{b,w,t}$ 取值为零。最后，对于实际平均耗时 $\mathrm{T}_{b,w,t} \neq 0$，算法通过以下方式更新 $\mathrm{E}_{w,t}$ ：

$$\mathcal{E}_{w,t+1} = \lambda_e \mathcal{E}_{w,t} + (1 - \lambda_e)\mathcal{T}_{b,w,t}. \tag{8}$$

之后，对于 eAP$b$ 请求队列中的每个请求$r_{b,w}$ 我们将其截止时间的剩余时间定义为 $\mathrm{F}r_{b,w}$，该值可根据请求生成时间、延迟限制和当前时间计算得出。此外，$\mathrm{F}r_{b,w}$ 请求

更有可能无法按时完成，即截止时间前的剩余时间小于预测所需时间。因此，我们在优先级 $C_{r_{b,w}}$ 的计算中加入惩罚因子 $\lambda' \in (0, 1)$，如下所示：

$$C_{r_{b,w}} = \begin{cases} 1/(\mathcal{F}_{r_{b,w}} - \mathcal{E}_{w,t}), & \text{if } \mathcal{F}_{r_{b,w}} > \mathcal{E}_{w,t}, \\ \lambda'/(\mathcal{E}_{w,t} - \mathcal{F}_{r_{b,w}}), & \text{if } \mathcal{F}_{r_{b,w}} < \mathcal{E}_{w,t}. \end{cases} \quad (9)$$

Finally, the requests in the request queue are sorted according to priority $C_{r_{b,w}}$, and the requests with the highest priority are dequeued for scheduling.

- 出队策略 $\Psi'$ 在边缘节点和云集群上。与边缘接入点的出队策略 $\Psi$ 类似，优先级 $C'_{r_{b,w}}$ 根据（8）和（9）计算，以确定请求从队列中出队的顺序。请注意，与边缘接入点的出队策略 $\Psi$ 相比，此处有两项不同的调整：(*i*) 与边缘接入点的出队策略 $\Psi$ 相比，忽略请求的传输时间；也就是说，每个边缘节点或云集群中的出队策略 $\Psi'$ 仅考虑请求从本地请求队列出队后的时间消耗；(*ii*) 与边缘接入点的出队策略 $\Psi$ 相比，每个时隙中出队的请求数量不止一个；也就是说，系统会按照优先级从高到低检查请求队列中的每个请求，判断是否有服务实体可以匹配该请求；如果存在具有空闲资源的相应服务实体，则该请求出队，并继续检查下一个较低优先级的请求。

### *B.* 基于图神经网络的学习：服务编排

本节提出一种基于图神经网络的策略梯度（*GPG*）算法，并说明如何灵活地处理系统状态信息（*i*）；以及如何将高维服务编排分解为逐步调度动作：选择高价值边缘节点，然后对其执行服务扩展（*ii*）。

对于由数量不同、资源异构的边缘节点组成的不同规模边缘集群，我们使用第 IV-C 节所述的图神经网络，即训练过程中学习到的参数化函数，反复处理边缘云网络的信息，包括网络拓扑、实时可用资源、服务请求队列等。此外，服务编排的决策间隔较长，且采用集中式决策方式，这给训练数据样本的收集带来了挑战。演员-评论家算法中的多个神经网络以及庞大的

神经网络参数的数量会进一步增加训练的难度。因此，为了更好地收敛，我们提出策略梯度算法来解决服务编排问题。

*1*）基于图神经网络的系统状态编码：如图 5 所示，*KaiS* 必须在每次观测时将系统状态转换为特征向量，然后将其传递给策略网络。常见做法是将系统状态直接堆叠成扁平向量。然而，边缘云网络实际上是由互联的边缘接入点、边缘节点和云集群构成的图。直接堆叠系统状态存在两个缺陷：(*i*)处理高维特征向量需要复杂的策略

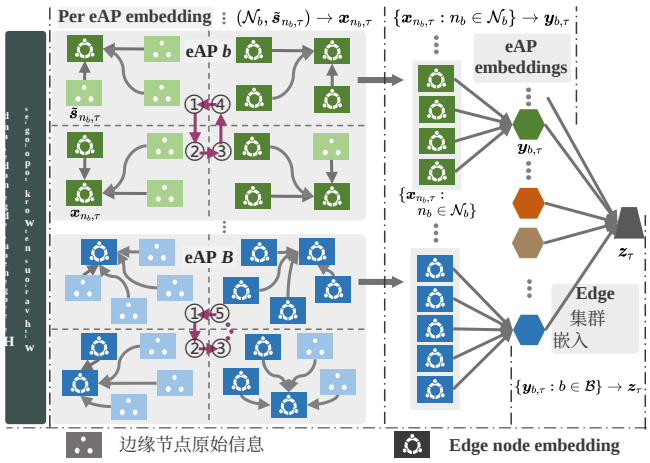



图 5. 基于 GNN 的边缘集群系统状态编码。

网络，这增加了训练难度；(*ii*) 它无法高效地对系统的图结构信息进行建模，使*KaiS* 难以泛化到各种不同规模和结构的系统。因此，我们使用图神经网络（GNNs）逐层将系统状态编码为一组嵌入向量，具体如下。

- 边缘节点的嵌入。对于与 eAP $b$相关联的边缘节点，其中每个节点 $n_b \in \mathrm{N}_b$在每一帧 $\tau$上都具有以下属性，记为向量 $\tilde{s}_{n_b,\tau}$：(*i*) 中央处理器、内存、存储等的可用资源；(*ii*) 定期测量的与 eAP $b$ 和云端之间的网络延迟；(*iii*) 当前正在处理的积压请求的队列信息，即 $\mathrm{Q}'_{n_b}$；以及 (*iv*) 已部署服务的索引和每项已部署服务的副本数。给定 $\tilde{s}_{n_b,\tau}$，KaiS 对每个边缘节点执行嵌入，表示为 $(\mathrm{N}_b, \tilde{s}_{n_b,\tau}) \rightarrow \boldsymbol{x}_{n_b,\tau}$。为执行嵌入，对于边缘节点 $n_b \in \mathrm{N}_b$，我们将其他边缘节点 $\mathrm{N}_b \setminus n_b$ 视为其邻居节点，从而构建一个虚拟图。然后，如图 5 所示，我们遍历 $\mathrm{N}_b$ 中的边缘节点，并逐一计算其嵌入结果。一个边缘节点完成嵌入后，仅为其余边缘节点后续的嵌入过程提供嵌入结果 $\boldsymbol{x}_{n_b,\tau}$。对于边缘节点 $n_b \in \mathrm{N}_b$，其嵌入结果 $\boldsymbol{x}_{n_b,\tau}$ 可以通过将其邻居节点 $\zeta(n_b) = \{\mathrm{N}_b \setminus n_b\}$ 的信息传播至自身来计算。在消息传递过程中，边缘节点 $n_b$ 汇总来自所有邻居节点的消息，并按如下方式计算其嵌入：

$$\boldsymbol{x}_{n_b,\tau} = h_1\left[\sum_{n'_b \in \zeta(n_b)} f_1(\boldsymbol{x}_{n'_b,\tau})\right] + \tilde{s}_{n_b,\tau}, \quad (10)$$

其中 $h_1(\cdot)$ 和 $f_1(\cdot)$ 均为由神经网络实现的非线性变换，组合后可表示多种聚合函数。在嵌入过程中，我们复用相同的神经网络 $h_1(\cdot)$ 和 $f_1(\cdot)$。

- ***eAP***和边缘集群的嵌入。类似地，我们利用图神经网络为每个 eAP $b$（$\{\boldsymbol{x}_{n_b,\tau} : n_b \in \mathrm{N}_b\} \rightarrow \boldsymbol{y}_{b,\tau}$）计算 eAP嵌入，并进一步为所有 eAP 计算边缘集群嵌入，$\{\boldsymbol{y}_{b,\tau} : b \in \mathrm{B}\} \rightarrow \boldsymbol{z}_\tau$。为了按照（10）计算 eAP $b$ 的嵌入，我们添加

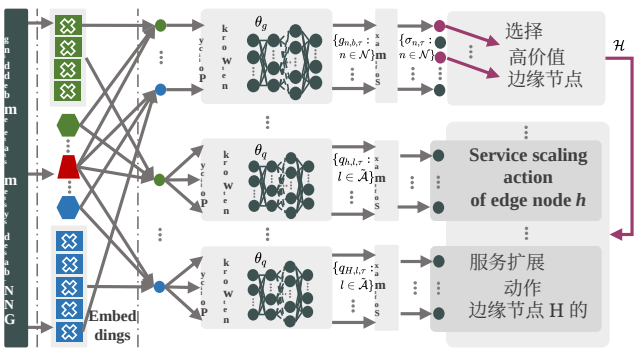



图 6. 用于服务编排的基于图神经网络的学习，即*GPG*。

一个 eAP 摘要节点到 $N_b$ ，并将 $N_b$ 中的所有边缘节点视为其邻居节点。这些 eAP 摘要节点还用于存储各自的 eAP嵌入。随后，通过聚合所有邻居节点发送的消息，可以获得每个 eAP 的 eAP嵌入，其计算方式如公式 (10) 所示。反过来，这些 eAP 摘要节点被视为边缘集群摘要节点的邻居节点，因此也可以使用公式 (10) 计算全局嵌入。尽管嵌入 $\boldsymbol{y}_{b,\tau}$ 和 $\boldsymbol{z}_\tau$ 均通过公式 (10) 计算，但非线性变换所使用的神经网络集合不同，即针对 $\boldsymbol{y}_{b,\tau}$ 使用 $(i)h_2(\cdot)$、$f_2(\cdot)$，而针对 $\boldsymbol{z}_\tau$ 使用 $(ii)$ $h_3(\cdot)$、 $f_3(\cdot)$。

*2)* 服务编排的逐步调度：编码服务编排动作的关键挑战在于，应对高维动作空间的学习和计算复杂度。一个直接的解决方案是维护一个庞大的策略网络，并基于第 III-B1 节中的嵌入结果，一次性为所有边缘节点 $\mathcal{W}$ 编排所有服务 $\mathcal{N}$。然而，采用这种方式，*KaiS* 必须从大量组合中选择动作 $(d_{w,n} \in \mathbb{N}: w \in \mathcal{W}, n \in \mathcal{N})$，从而增加样本复杂度，并减缓训练过程 [32]。此外，过于频繁的大规模服务编排会带来巨大的系统开销，并损害系统稳定性。

因此，我们考虑逐步调度：每帧首先选择 $H$ 高价值边缘节点（$H = 2$在实验中），然后在规模小得多的自定义动作空间中，为每个节点扩展服务 $2M+1$。具体而言，*KaiS* 将第 III-B1 节中的嵌入向量作为策略网络的输入，策略网络输出联合编排动作 $\tilde{\boldsymbol{a}}_\tau = (\tilde{a}^\bullet_\tau, \tilde{\boldsymbol{a}}^\star_\tau)$，包括 $(i)$ 选择高价值边缘节点的动作 $\tilde{a}^\bullet_\tau$ 以及$(ii)$相应的联合服务扩缩容动作 $\tilde{\boldsymbol{a}}^\star_\tau$。

- 高价值边缘节点的选择。在每一帧中，*KaiS*首先使用策略网络选择 $H(\leq N)$ 个高价值边缘节点，记为动作 $\tilde{a}^\bullet_\tau$。如图 6 所示，对于与 eAP $b$ 关联的边缘节点 $n$ ，系统会计算一个值 $g_{n,b,\tau} = g(\boldsymbol{x}_{n,\tau}, \boldsymbol{y}_{b,\tau}, \boldsymbol{z}_\tau)$，其中 $g(\cdot)$ 是由神经网络（NN） $\theta_g$ 实现的非线性值评估函数。引入函数 $g(\cdot)$ 是为了将嵌入向量映射为标量值。值 $g_{n,b,\tau}$ 表示 *KaiS* 在边缘节点 $n$ 执行服务扩展的优先级。使用 Softmax 运算计算概率 $\sigma_{n,\tau}$，用于

根据值 $\{g_{n,b,\tau}: n \in \mathrm{N}\}$ 选择边缘节点 $n$：

$$\sigma_{n,\tau} = \mathrm{e}^{g_{n,b,\tau}} / \sum_{b' \in \mathcal{B}} \sum_{n_{b'} \in \mathcal{N}_{b'}} \mathrm{e}^{g_{n_{b'},b',\tau}}. \quad (11)$$

根据所有边缘节点的概率 $\{\sigma_{n,\tau}: n \in \mathrm{N}\}$，*KaiS* 选择概率较高的 $H$ 个边缘节点作为高价值边缘节点 H，以执行服务扩展。

- 高价值边缘节点的服务扩展。对于选定的高价值边缘节点 $\mathrm{h} \in \mathrm{H} \subset \mathrm{N}$，KaiS 使用一个动作评估函数 $q(\cdot)$，该函数由神经网络（NN） $\theta_q$ 实现，用于计算边缘节点 h 在第 $\tau$ 帧执行服务扩缩容动作 $\tilde{a}_{\mathrm{h},\tau} = l$ 时的价值 $q_{\mathrm{h},l,\tau} = q(\boldsymbol{x}_{\mathrm{h},\tau}, \boldsymbol{y}_{b,\tau}, \boldsymbol{z}_\tau, l)$。 $l$ 的动作空间定义为 $\mathrm{A} \triangleq \{-W, \ldots, w, \ldots, W\}$，其大小为 $2W+1$，即 $l \in \mathrm{A}$。 $l$ 的含义如下：$(i)$ $l = 0$ 表示所有服务保持不变；$(ii)$ $l = -w$ 表示删除服务 $w$ 的一个副本；$(iii)$ $l = w$ 表示增加服务 $w$ 的一个副本。特别地，对于因边缘节点资源限制而无效的服务扩缩容动作，KaiS 始终将其转换为 $l = 0$。类似地，~我们对 $\{q_{\mathrm{h},l,\tau}: l \in \mathrm{A}\}$ 应用 Softmax 运算，以计算各个扩缩容动作的概率，并选择执行概率最高的动作。对于所有高价值边缘节点 H，KaiS 将在每一帧生成一个联合服务扩缩容动作 $\tilde{\boldsymbol{a}}\bigstar_\tau = (\tilde{a}_{\mathrm{h},\tau}: \mathrm{h} \in \mathrm{H})$。

虽然*KaiS* 将请求调度与服务编排解耦，但这并不影响我们提升 $\Phi'$。 事实上，通过使用定期更新的策略网络，为编排提供适当的负载均衡边缘集群，我们在优化编排的同时也隐式地优化了调度策略。

为指导 GPG，KaiS 在帧 $\tau$ 的每次服务编排后生成奖励 $\tilde{u}_\tau = \mathrm{e}^{-\sum_{n \in \mathrm{N}} |\mathrm{Q}_{n,\tau}|}$，其中 $|\mathrm{Q}_{n,\tau}|$是边缘节点 $n$ 上未处理请求的队列长度。通过这种方式，GPG 逐步学会减少未处理请求的积压，从而提高吞吐率，正如我们将在实验中展示的那样。KaiS 采用策略梯度算法训练 GPG 中使用的神经网络 $\{f_i(\cdot), \mathrm{h}_i(\cdot)\}_{i=1,2,3}$, $\theta_g$ 和$\theta_q$。为清晰起见，我们将这些神经网络的所有参数统称为 $\theta^*$，将所有 GNN 编码的系统状态统称为 $\tilde{\boldsymbol{s}}_\tau$，将联合服务编排动作记为 $\tilde{\boldsymbol{a}}_\tau$，将调度策略记为 $\pi_{\theta^*}(\tilde{\boldsymbol{s}}_\tau, \tilde{\boldsymbol{a}}_\tau)$，即在观察到状态 $\tilde{\boldsymbol{s}}_\tau$ 时采取动作 $\tilde{\boldsymbol{a}}_\tau$ 的概率。在每一帧，KaiS 收集观测值 $(\tilde{\boldsymbol{s}}_\tau, \tilde{\boldsymbol{a}}_\tau, \tilde{u}_\tau)$，并使用策略梯度更新参数 $\theta^*$：

$$\theta^* \leftarrow \theta^* + \alpha \sum_{\tau=1}^{T} \nabla_{\theta^*} \log \pi_{\theta^*}(\tilde{\boldsymbol{s}}_\tau, \tilde{\boldsymbol{a}}_\tau) \left( \sum_{\tau'=\tau}^{T} \tilde{u}_{\tau'} - \mu_\tau \right), \quad (12)$$

其中， $T$ 是*GPG* 训练回合的长度， $\alpha$ 是学习率， $\mu_\tau$ 是用于降低策略梯度方差的基线。一种计算基线的方法是：将 $\mu_\tau$ 设为从帧 $\tau$ 开始的累积奖励，并对所有训练回合 [33]取平均值。

## IV. Implementation Design

系统中的所有服务均以*Docker* 容器的形式运行。此外，*KaiS* 基于*k8s* v1.18 和*k3s*v1.0（适用于边缘的轻量级*k8s* ）实现 [34] ，运行于 Ubuntu 16.04，使用 Python 3.6。

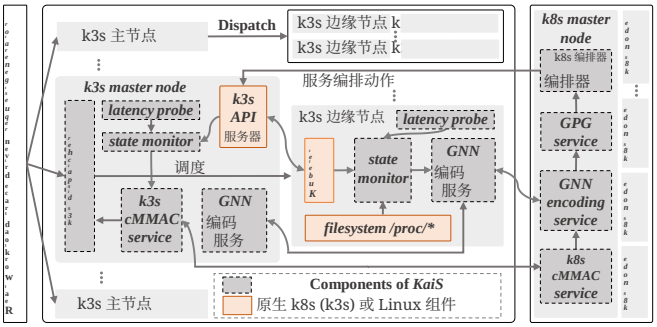



图 7. *KaiS*的实现与原型设计。

### *A.* 系统设置

• 请求。我们对来自阿里巴巴的真实工作负载轨迹 [29] 进行修改，并用其生成服务请求。我们将该轨迹中的工作负载请求划分为 30 项服务。我们没有使用真实的终端设备，而是实现了一个请求生成器 来生成服务请求，然后将其转发至 *k3s* 主节点（边缘接入点）。

•边缘集群和节点。默认情况下，我们在 Google Cloud Platform (GCP) 的不同地理区域中搭建 5 个*k3s*集群，以模拟地理分布；每个集群由一个*k3s* 主节点 和 8 个*k3s* 边缘节点组成。*k3s* 主节点和*k3s* 边缘节点 分别采用 GCP 虚拟机 (VM) 配置“2 个虚拟中央处理器、4 GB 内存和 0.3 TB 磁盘”以及“1–2 个虚拟中央处理器、2–4 GB 内存和 0.3 TB 磁盘”。此外，我们使用性能更强的*k3s* 主节点来加速离线训练。

• 云集群。我们构建了一个由 15 台同构虚拟机组成的集群作为云集群，每台虚拟机配置为“4 个虚拟中央处理器、16 GB 内存和 1 TB 磁盘”。我们在其中一台虚拟机上部署了 *k8s* 主节点 来管理其他虚拟机。我们手工创建了 30 个中央处理器和内存消耗各不相同的服务，并将其 Docker 镜像存储在云端。

• 网络控制。我们使用 *LinuxTC*，有意控制边缘云系统的网络延迟，以模拟实际场景。参见 [35]和 [36]，参数设置如下：(*i*) 对于终端设备和边缘集群，延迟约为 20 ms，带宽约为 50 Mbps。(*ii*) 由于边缘集群内部通过局域网连接，网络状况较好，因此延迟低于 10 ms，带宽约为 1 Gbps。(*iii*) 由于云集群和边缘集群通常相距甚远，因此使用广域网进行连接。因此，延迟约为 100 ms，带宽约为 100 Mbps。

### *B. KaiS* 的主要组件

如图 7 所示，我们将 *KaiS* 解耦为两个主要部分。

•去中心化请求分发器。*KaiS* 在每个*k3s* 主节点上部署一个*k3s* 调度器，通过状态监视器按以下方式定期观测并收集当前系统状态。每个*k3s* 边缘节点(*i*)运行一个*Kubelet*进程，并且 (*ii*)读取*Linux*中的虚拟文件系统 */proc/**，以收集*Docker* 服务和物理节点的状态。对于网络状态，每个*k3s* 边缘节点和*k3s* 主节点都部署一个延迟探针，用于测量网络延迟。*k3s* 边缘节点上的状态监视器会定期将上述收集到的系统状态推送至*k3s* 主节点上的状态监视器进行融合。为实现*cMMAC*，我们在每个*k3s* 主节点上部署一个*cMMAC* 代理，作为*k3s cMMAC* 服务；同时在*k8s* 主节点上运行*k8s cMMAC* 服务以支持训练。每个调度时隙为 0.25 秒，该时长根据图 10 所示的实验结果经验确定。每个*k3s* 主节点上的*k3s cMMAC* 服务通过观测状态监视器提供的本地状态来计算调度动作，然后通知*k3s* 调度器为当前请求执行该调度。

• 集中式服务编排器。为实现*GPG*，*KaiS*在*k3s* 边缘节点、*k3s* 主节点和*k8s* 主节点上持有一组使用不同图神经网络的图神经网络编码服务 （第 III-B1 节）。这些图神经网络编码服务 彼此通信，分别用于计算边缘节点、边缘接入点（即*k3s* 主节点）和边缘集群的嵌入。一旦*KaiS* 完成基于图神经网络的编码，图神经网络编码服务 便会在*k8s* 主节点 上合并所有嵌入结果。*GPG*的其余部分，即策略网络，则实现为*GPG* 服务 并部署在*k8s* 主节点上。帧长度设为 100× 个时隙，以确保系统稳定性。在每一帧中，*GPG* 服务 从图神经网络编码服务 拉取所有嵌入并计算编排行动。随后，*GPG* 服务 调用*k8s* 编排器 ，与特定的*k3s API* 服务器通信，通过*python-k8sclient*完成服务扩展。与其他扩展操作不同，只有当服务处于空闲状态时，*k3s API* 服务器才能将其删除。否则，*KaiS* 会推迟扩展，直到满足该条件。

### *C.* 训练设置

我们使用 TensorFlow 1.14 实现算法 1。详细设置如下。***cMMAC***：*cMMAC* 包含一个评论家网络和一个行动者策略网络。两个网络均使用 Adam 优化器进行训练，固定学习率为 $5 \times 10^{-4}$。评论家 $\theta_v$ 采用四层 ReLU 神经网络进行参数化，各层的节点数分别为 256、128、64 和 32。行动者 $\theta_p$ 采用三层 ReLU 神经网络实现，每层分别包含 256、128 和 32 个隐藏单元。请注意，行动者的输出层使用 ReLU+1 作为激活函数，以确保原始 logits 中的元素均为正值。***GPG***：*GPG* 使用 (*i*) 六个图神经网络，即 $\{ f_i(\cdot), h_i(\cdot)\}_{i=1,\ 2,\ 3}$ 以及 (*ii*) 两个策略网络，包括 $\theta_g$ 和 $\theta_q$。其中，$\{ f_i(\cdot), h_i(\cdot)\}_{i=1,\ 2,\ 3}$ 采用双隐藏层神经网络实现，每层分别包含 64 和 32 个隐藏单元。此外，$\theta_g$ 和 $\theta_q$ 均为三隐藏层神经网络，从第一层到最后一层的节点数分别为 128、64 和 32。所有与*GPG*相关的神经网络均使用 Adam 优化器，并以 $10^{-3}$ 的学习率进行更新。

对于请求调度和服务编排，传统的深度强化学习算法都无法应对持续到达的服务请求。到达请求的随机性使学习算法难以区分不同服务请求模式的影响，以及

算法决策对系统性能变化的影响。

此外，基于学习的调度策略在训练初期难免会做出错误决策。因此，在服务请求持续到达的情况下，尚未充分训练的调度策略必然会降低系统吞吐量，导致大量服务请求积压且无法及时处理。在这种情况下，继续花费大量时间探索更好的动作并不能提高调度策略的准确性。

为了解决上述问题，我们采用了渐进式训练方法。受 [37], 提出的课程学习启发，我们首先使用简单且较短的服务请求序列进行训练，然后逐步适度引入更复杂的请求序列，从而逐步改进调度策略。

## V. 性能评估

接下来，我们从以下几个方面评估 *KaiS* 的设计和原型实现：（*i*）贪心（用于调度），将每个请求调度到资源利用率最低的边缘节点；（*ii*）原生（用于编排），即默认的水平 *Pod* 自动伸缩器 [38], 位于*k8s*中。它会定期观测系统中的特定资源指标，并将其与预定义的阈值进行比较，最终根据两者之间的差距做出决策。（*iii*）***GSP-SS*** [10]（两者兼用），假设每种服务的请求到达率是预先已知的。这是一种联合服务部署与请求调度的双时间尺度框架，并基于线性规划（LP）计算出的影子请求调度设计了一种贪心服务部署算法。（*iv*）***Firmament*** [39]（用于调度），旨在以最优方式将工作（请求）分配给云集群资源。它是一种集中式调度方法，首先将系统状态观测概括为图结构数据，然后通过基于最小费用最大流（MCMF）的算法解决请求调度问题。

我们考虑三个主要性能指标：（*i*）帧吞吐率$\Phi_f = \left\lfloor \sum_{n\in\mathcal{N}} \Upsilon_\tau (\mathcal{Q}'_n) + \Upsilon_\tau (\hat{\mathcal{Q}}_c) \right\rfloor / \sum_{b\in\mathcal{B}} \Upsilon_\tau (\mathcal{Q}_b)$，其反映了$\Phi'$的短期特性；（*ii*）调度延迟 $\Phi_d$，即执行调度操作所需的时间；（*iii*）调度成本 $\Phi_c$，主要体现为网络带宽消耗，包括额外的数据包

由于请求调度而产生的转发，以及服务编排期间边缘从云端拉取服务 *Docker* 镜像所消耗的带宽。为清晰起见，我们对部分指标进行了必要的归一化，并根据多次实验的结果给出了它们的统计特征。

对于请求特征，它们是根据阿里巴巴的真实工作负载轨迹生成的 [29]: (*i*) 使用数据集中的“开始时间”生成请求到达时间；(*ii*) 将数据集中每个请求的“结束时间”减去“开始时间”，以生成其时延要求；(*iii*) 使用数据集中的“任务类型”生成每个请求的类型；(*iv*) 使用数据集中的“计划CPU”生成每个请求的 CPU需求；(*v*)-使用数据集中的“计划内存”生成每个请求的内存需求。

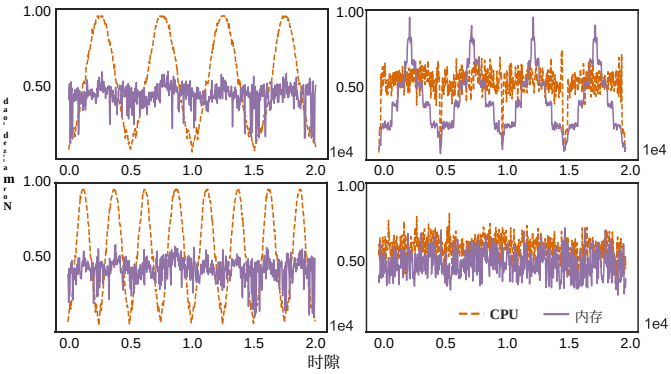



图 8. 按服务类型采样的 4 种请求到达模式示例：（a）$\mathbb{P}_1$（左上），（b）$\mathbb{P}_2$（右上），（c）$\mathbb{P}_3$（左下），（d）$\mathbb{P}_4$（右下）。

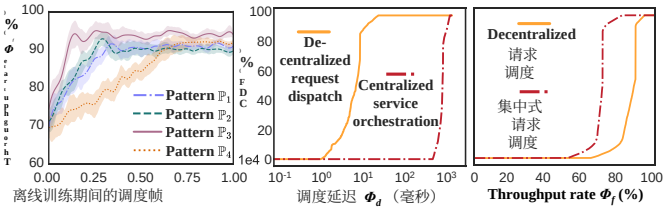



图 9.（a）KaiS 针对具有各种模式的请求到达的学习能力（左），以及（b）KaiS 执行去中心化请求调度和集中式服务编排时的调度延迟（中），以及（c）KaiS 使用去中心化和集中式调度时的性能（右）。

### A. *KaiS* 的学习能力与实用性

*KaiS* 应能够学习如何应对具有潜在统计模式的请求到达，甚至随机请求到达。

在第 IV-A 节中，我们对工作负载数据集 Ω 进行采样或截取，以获得具有四种模式的请求到达序列，如图 8 所示，即（*i*）模式 $\mathbb{P}_1$：周期性波动的中央处理器总负载；（*ii*）模式 $\mathbb{P}_2$：周期性波动的内存总负载；（*iii*）模式 $\mathbb{P}_3$：$\mathbb{P}_1$，其 2× 频率发生波动；（*iv*）模式 $\mathbb{P}_4$：从 Ω截取的原始随机请求到达。此外，图 9（b）和（c）中使用累积分布函数（CDF）。为介绍该函数，我们首先将某一变量 $X = x$ 的概率定义为 $P(X = x)$（$x$ 为常数），则相应的累积分布函数为 $F_X(x) = P(X \le x)$。

• 学习能力。图 9(a) 展示了*KaiS* 在不同请求模式下训练期间的性能演变。吞吐率 $\Phi_f$ 在所有情况下都随时间推移而提高，这表明 *KaiS* 能够逐渐学会应对不同的请求模式。特别是，*KaiS* 在应对随机请求到达（1。2 ）时，需要至少经历比其他情况（$\mathbb{P}_4$）多 $\mathbb{P}_{1-3}$*KaiS* 才能实现稳定调度。不过，一旦其收敛，针对不同模式请求的调度性能差距在 *4*。5% 以内。

• 去中心化还是集中式调度？图 9(b) 显示，集中式服务编排的调度延迟几乎 9× 大于 去中心化请求调度的延迟，而后者可在约 10 毫秒内完成。此外，我们为云端的每个 *cMMAC* eAP 配备一个智能体，以

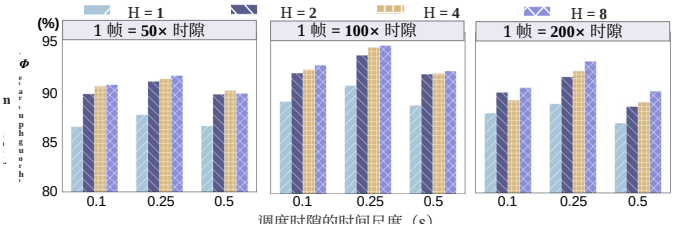



图 10. *KaiS*在不同设置下的调度性能：(a) 1 帧= 50 × 时隙（左），(b) 1 帧 = 100 × 时隙（中），(c) 1 帧 = 200 × 时隙（右）。

采用集中式方式调度请求进行比较。从图 9(c) 可以看出，去中心化调度能够带来更高的吞吐率，因为集中式调度需要额外时间上传本地观测值（$\hat{s}_{b,t}$）并等待调度决策。然而，对于某些时延敏感服务请求而言，这些额外延迟不容忽视。

• 双时间尺度调度与逐步编排。频繁调度未必能带来更好的性能。如图 10 所示，当时隙为 0。1秒时，*cMMAC* 智能体在相邻时隙中常会遇到相似的系统状态，从而削弱其学习能力。当时隙过长（0。5秒）时，调度不及时也会降低性能。此外，服务编排过于频繁会增加调度成本，使 *cMMAC* 智能体难以收敛。虽然在每帧中选择更多高价值边缘节点进行服务编排有利于提升吞吐量，但当$H \geq 2$时，性能提升非常有限，而更大的$H$则会带来更多调度成本。上述因素会影响 *KaiS* 的能力。我们将展示，与基线方法相比，默认配置“0。25秒（时隙），25秒（帧），$H = 2$”已能取得不错的性能。值得注意的是，参数取值会受到许多因素的影响，例如应用的性质、请求负载、边缘集群规模、网络状态等。因此，这些取值并不通用；若应用于其他场景，应事先开展类似实验并进行调整。

### *B. Impact of Load Balancing*

在图 11 中，我们展示了在 $\hat{u}_{b,t} = e^{-\lambda-\varepsilon\nu}$ 中，使用 $\varepsilon$ 的不同设置训练的 *KaiS* 的调度性能；$\varepsilon$ 表示边缘负载均衡的程度。*KaiS* 在 $\varepsilon = 1$ 时实现了最佳吞吐率，而在 $\varepsilon = 4$ 时，其性能则急剧下降。这种性能差距是因为，当 $\varepsilon = 4$ 时，*KaiS* 过于注重负载均衡，在许多情况下放弃了能够更高效地处理请求的调度选项。此外，当 $\varepsilon = 0$，即不考虑负载均衡时，吞吐率 $\Phi_f$ 和负载均衡仍优于 $\varepsilon = 4$ 的情况。这一事实表明，即使我们没有刻意关注负载均衡，这里只精品
设计 *cMMAC* 时，*KaiS* 仍然可以学习有利于提升吞吐量的负载均衡策略。然而，为奖励函数设置适中的 $\varepsilon$ 可以让 *KaiS* 更有效地学习。

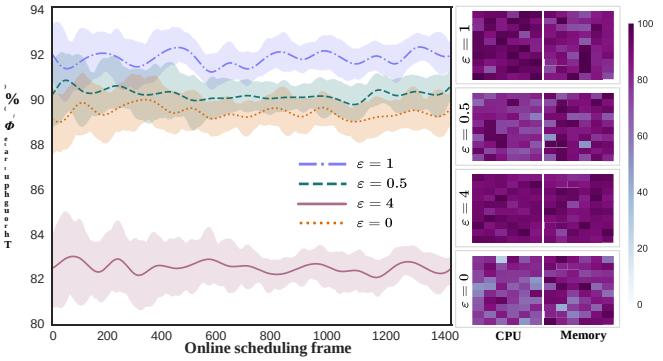



图 11. 负载均衡对*KaiS*的调度性能的影响：(a) 吞吐率（左）和 (b) 边缘节点的平均负载（右）。

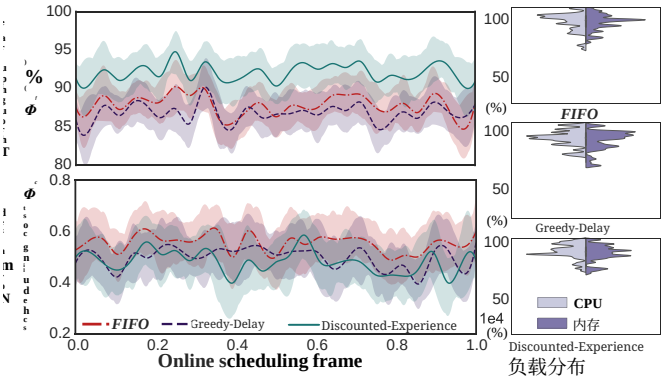



图 12. 不同出队策略在以下方面的性能：（a）吞吐率（左上）和（b）成本（左下），以及（c）不同资源占用率的节点百分比（右）。

### *C.* 出队策略的有效性

之后，我们设计了三种出队策略进行比较。（*i*）先进先出（***FIFO***），即所有请求均按照进入请求队列的时间顺序出队。（*ii*）延迟贪心，即由于请求对延迟较为敏感，因此按照时延要求对请求队列中的请求进行排序，超时时间越近的请求，其排队优先级越高。（*iii*）***Discounted-Experience***，即基于折扣经验计算不同类型请求的预测完成时间，该策略优先处理预测完成时间接近截止时间的请求的出队操作。

如图 12 所示，我们比较了上述三种出队策略的有效性。可以发现，三种策略在成本方面的差异相对较小，大致处于同一水平。然而，在吞吐量方面，先进先出 与贪心-延迟 的性能相近，而 *Discounted-Experience* 则具有明显优势。产生上述实验结果的原因如下。对于 先进先出 策略，请求的到达时间无法准确反映其处理优先级，因为不同请求的时延要求各不相同。因此，先进先出 无法有效处理时延要求极高的请求，从而造成资源浪费并降低吞吐量。此外，贪心-延迟

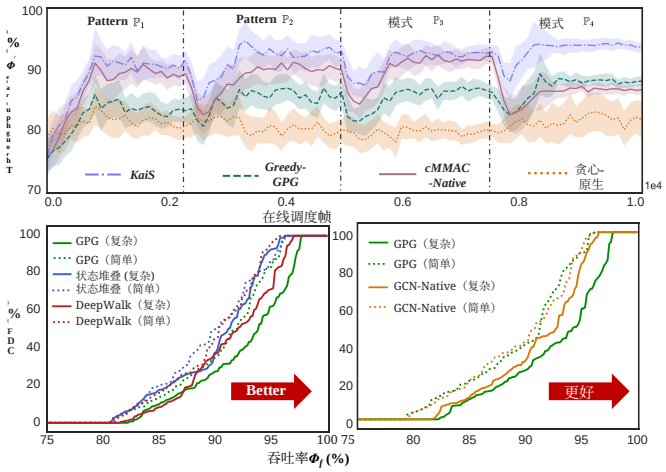



图13(a). (a) *GPG* 应对模式波动的请求到达的能力（上）。(b, c) 在不同系统规模下，与其他状态编码方法的比较（左），以及有无自定义修改的比较（右）。

也存在一些不足。对于即将超过延迟要求的请求，即使立即将其出队，由于剩余时间很短，也可能无法按时完成。然而，在贪心-延迟出队策略下，这些请求具有较高的优先级，处理它们只会造成资源浪费，并对吞吐量产生负面影响。相比之下，我们设计的*Discounted-Experience*出队策略具有明显优势。该策略能够避免另外两种出队策略的缺点，并通过合理安排每个请求的排队时间来减少资源浪费。

### *D.* 基于*GNN*的服务编排的作用

接下来，我们首先将四种模式的请求到达序列组合成一个长序列，以评估*GPG*应对模式波动的请求到达的能力。请注意，这些较长的请求到达序列旨在反映高变异性场景。

•应对随机请求到达。在图13(a)中，展示了*KaiS*、贪心-*GPG*、*cMMAC*-原生和贪心-原生在高变异性场景下的调度性能。结果表明以下几点：(*i*) *KaiS* 的平均吞吐率高于最接近的竞争基线方法；尤其是在请求到达模式发生变化时，*KaiS* 仍能快速学习适应新模式的策略；(*ii*) 对于模式$\mathbb{P}_1$、2、3，*cMMAC*-原生能够实现接近*KaiS*的调度性能，这是因为高效的请求调度算法（例如*cMMAC*）已经能够应对具有明显模式的请求到达；(*iii*) 对于复杂模式 $\mathbb{P}_4$，即请求像原始轨迹 Ω所示那样随机到达，由于缺乏能够自适应地释放和获取全局系统资源的服务编排，*cMMAC*-原生和贪心-原生的性能会下降。

• 基于图神经网络的编码与其他方法的比较。我们在图 13(b) 中展示了基于图神经网络的系统状态编码（第 III-B1 节）。为了进行评估，我们构建了两个具有不同系统规模的边缘集群：（*i*）第 IV-A 节介绍的默认设置；（*ii*）一个复杂设置，包含 10 个*k3s* 主节点，每个主节点管理 3–15 个异构的*k3s* 边缘节点（总计 100 个）。此外，我们选择了两种状态编码方法，即状态堆叠 和*DeepWalk*，进行比较。具体而言，状态堆叠 是将系统状态直接堆叠为扁平向量。*DeepWalk* 不同于*GPG*基于神经网络的非线性映射，它采用随机游走策略生成图的低维表示 [40], ，该方法已应用于边缘云网络相关的研究工作中 [41]。从图 13(b) 中，我们可以观察到以下几点：（*i*）在不同的系统规模下，*GPG* 均优于其他方法，与*DeepWalk* 的差距较小，但与状态堆叠的差距较大；（*ii*）随着系统规模的扩大，不同状态编码方法之间的差距也随之增大；（*iii*）状态堆叠在更大的系统规模下并未表现出显著改进，这是因为规模扩大将带来更加复杂的状态，而该方法无法准确提取状态信息，导致模型训练效果不佳，从而显著降低推理准确性和整体系统效率。总之，基于 GNN 的编码能够显著降低 *KaiS* 对神经网络模型复杂度的依赖，这是实现高效、快速学习的关键。此外，该方法还嵌入了网络延迟和系统结构信息，有助于*KaiS* 扩展至大规模边缘云网络。

• 自定义修改的重要性。 *GPG* 在第三节B部分中介绍，其基于通用图卷积网络（GCN），针对边缘云网络进行了自定义修改。我们设计了*GCN*-原生 模型作为对比，以检验所采用的自定义修改能否提升性能。具体而言，*GCN*-原生 直接基于通用GCN对由全局原生系统状态组成的图数据进行编码，这与*GPG*的消息聚合和多层级嵌入方式不同。如图13(c)所示，在小规模边缘云系统中，*GPG* 与 *GCN*-原生 之间没有显著差异，但随着系统规模的扩大，*GPG* 表现出更优的性能。上述实验结果表明，自定义修改有助于*KaiS* 高效地从环境中提取信息，从而增强 *KaiS* 对大规模边缘云网络的适应能力。

### *E.* 与基线方法的性能比较

为了评估*KaiS*，我们需要同时考虑调度性能和成本。我们对工作负载数据集 Ω 进行截取，获得 50 个长度相同的请求到达序列，并用它们评估*KaiS*。从图 14(b-c)中可以观察到，在几乎所有情况下，无论请求的负载和时延要求如何波动（图 14(a)），*KaiS*都能实现高出 14.3% 的吞吐率 $\Phi_f$，且调度成本 $\Phi_c$ 比最接近的竞争基线方法低 34.7%。

尤其是在某些帧中请求负载和时延要求较低时，*GSP-SS* 的调度性能 $\Phi_f$ 可以非常接近*KaiS*。然而，与 *KaiS*相比，*GSP-SS* 的调度性能会下降

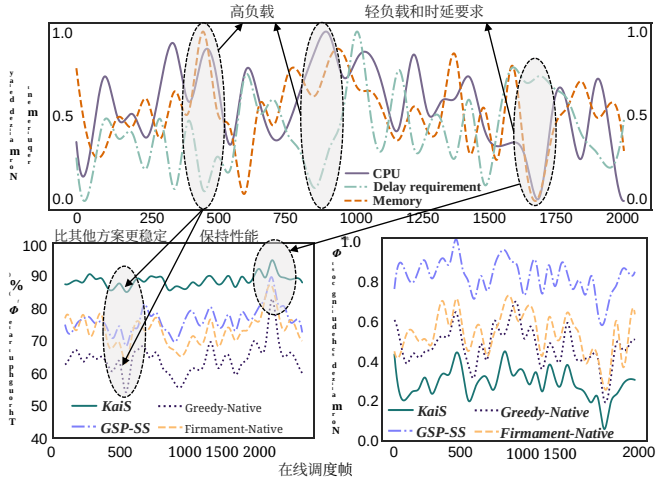



图 14. (a) 随机请求到达（上图）时，(b, c) *KaiS* 与基线方法在吞吐率（左）和成本（右）方面的性能。

在高负载帧期间：由于它无法了解系统处理请求的能力，因此当请求负载水平较高时，无法对边缘集群进行负载均衡以分摊这些负载，从而缩小可用的调度空间。此外，(*i*) *KaiS* 采用双时间尺度调度，与 *GSP-SS* 在每一帧都执行大规模编排不同，(*ii*)它仅选择固定数量 $H$ 的高价值˜边缘节点来执行受 A 限制的服务编排。因此，*KaiS* 在每一帧中的调度成本都有上限，从而降低总体成本，如图 14(b) 所示。

## VI. 相关工作

尽管现有优化研究探索了调度性能的上限，但由于存在各种模型假设，这些研究并不适用于实际部署环境（例如，*k8s*）。此外，目前尚无能够适应去中心化请求分发的系统设计研究。

### *A.* 理论分析

当前大多数理论研究探讨两个问题之一，即请求调度和服务编排。对于请求调度， [28], [42], [43] 针对边缘计算场景中的不同优化目标提出了调度算法，但其仿真模型中的一些关键系统参数，例如任务大小、设备的计算能力和网络通信能力，均采用静态常数或取值范围。对于服务编排， [44]–[46]结合不同的算法设计视角来提供解决方案，但二者都无法充分利用集中式云集群，也无法充分考虑硬件资源约束。

许多研究，例如 [30], [47], ，为卸载随机计算或服务请求提供了调度解决方案，可作为本研究的补充。 [10]–[12]为联合优化请求调度和服务编排奠定了理论基础。然而， [11], [12] 中提出的一次性调度优化无法处理持续到达的服务请求，即未考虑调度的长期影响。在 [10],中，作者提出采用两个不同的时间尺度进行优化，以最大化每次调度中完成的请求数量。然而， [10] 中的长期优化依赖于对未来服务请求的准确预测，这在实践中很难实现。最后但同样重要的是，这些研究 [10]–[12]不适用于实际场景，原因如下：（*i*）它们假设能够准确建模或预测计算资源、网络需求或特定请求的处理时间；（*ii*）调度采用集中式方式进行，而这必须额外耗时等待整个系统上下文信息的汇总。

### *B.* 系统设计

已有许多高效的调度器被开发用于*k8s*-基于云集群。相关研究要么通过最小费用最大流优化为通用工作负载调度所有任务， [39]要么利用特定领域知识（例如深度学习）来提升特定工作负载下的整体集群利用率 [48]。然而，由于其调度器以集中式方式部署在云端，因此无法适应边缘侧的去中心化请求分发。文献 [49] 提出了一种服务编排方法，通过定期测量边缘节点之间的延迟，估计服务请求的预期处理延迟是否能够满足要求。与我们最相关的研究是 [50], ，该研究使用基于模型的强化学习处理服务编排，并兼容地理分布式边缘集群。尽管如此， [49]和 [50]都没有考虑在边缘集群中调度请求。据我们所知，当前关于 *k8s* 系统设计的研究仅关注服务编排问题，而忽略了请求调度问题。因此，当前缺乏与请求和服务协同优化相关的系统设计。

## VII. 结论

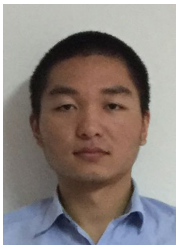
利用*k8s* 无缝融合分布式边缘与云端，是边缘云网络的未来。本文介绍了*KaiS*，一种集成定制学习算法的调度框架，面向*k8s*-基边缘云网络，能够动态学习请求调度和服务编排策略，以提升长期系统吞吐率。为此，我们针对*KaiS*定制了学习算法，包括一种用于去中心化请求调度的协同多智能体演员-评论家算法，以及一种基于折扣经验的出队策略；还包括一种用于集中式服务编排的基于GNN的策略梯度算法。为了验证*KaiS*的有效性，我们在边缘云网络中开展了系统性实验。结果展示了*KaiS* 在不同场景下的表现，并表明*KaiS* 至少可将平均系统吞吐率提升15。9%，同时将调度成本降低38。4%。此外，通过修改调度动作空间和奖励函数，*KaiS* 还适用于其他调度优化目标，例如最小化长期系统开销。

## 参考文献

[1] Y. Han、S. Shen、X. Wang、S. Wang 和 V. Leung，《面向 Kubernetes 的边缘云系统的定制学习调度》，发表于 IEEE INFOCOM，2021年。[2] W. Shi、J. Cao 等，《边缘计算：愿景与挑战》，IEEE Internet Things J.，第 3 卷第 5 期，第 637–646 页，2016年10月。[3] B. Burns、B. Grant、D. Oppenheimer、E. Brewer 和 J. Wilkes，《Borg、Omega 和 Kubernetes》，Commun. ACM，2016年4月。[4] 《KubeEdge：Kubernetes 原生边缘计算框架（CNCF 项目）》。 [在线]。网址：https://github.com/kubeedge/kubeedge [5] 《OpenYurt：将原生 kubernetes 扩展至边缘》。 [在线]。网址：https://github.com/alibaba/openyurt[6] 《Baetyl：将云计算、数据和服务无缝扩展至边缘设备》。 [在线]。网址：https://github.com/baetyl/baetyl [7] X. Wang、Y. Han、V. C. Leung、D. Niyato、X. Yan 和 X. Chen，《边缘计算与深度学习的融合：综合综述》，IEEE Commun. Surv. Tutor.，第 22 卷第 2 期，第 869–904 页，2020年。[8] H. Tan、Z. Han、X.-y. Li 和 F. C. Lau，《边缘云中的在线作业调度与调度》，发表于 IEEE INFOCOM，2017年。[9] S. Pasteris、S. Wang、M. Herbster 和 T. He，《异构边缘计算系统中具有可证明保证的服务部署》，发表于 IEEE INFOCOM，2019年。[10] V. Farhadi、F. Mehmeti、T. He、T. L. Porta、H. Khamfroush、S. Wang 和 K. S. Chan，《边缘云中面向数据密集型应用的服务部署与请求调度》，发表于 IEEE INFOCOM，2019年。[11] K. Poularakis、J. Llorca、A. M. Tulino、I. Taylor 和 L. Tassiulas，《多小区移动边缘计算网络中的联合服务部署与请求路由》，发表于 IEEE INFOCOM，2019年。[12] X. Ma、S. Wang 等，《移动边缘计算中的协作式服务缓存与工作负载调度》，发表于 IEEE INFOCOM，2020年。[13] J. A. Ayala-Romero、A. Garcia-Saavedra、M. Gramaglia、X. Costa-Perez、A. Banchs 和 J. J. Alcaraz，《vrAIn：一种在虚拟化无线接入网中定制计算与无线电资源的深度学习方法》，发表于 ACM MobiCom，2019年。[14] R. S. Sutton 等，《强化学习：导论》。[15] H. Mao、M. Schwarzkopf、S. B. Venkatakrishnan、Z. Meng 和 M. Al- izadeh，《学习数据处理集群的调度算法》，发表于 ACM SIGCOMM，2019年，第 270–288 页。[16] J. Ren、D. Zhang、S. He、Y. Zhang 和 T. Li，《端-边-云编排网络计算范式综述》，ACM Comput. Surv.，第 52 卷第 6 期，第 1–36 页，2019年10月。[17] V. Mnih 等，《通过深度强化学习实现人类水平控制》，《自然》，第 518 卷第 7540 期，第 529–533 页，2015年2月。[18] T. P. Lillicrap 等，《利用深度强化学习进行连续控制》，arXiv 预印本 arXiv:1509.02971，2015年。[19] F. Wang、F. Wang、J. Liu、R. Shea 和 L. Sun，《网络边缘的智能视频缓存：一种多智能体深度强化学习方法》，发表于 IEEE INFOCOM，2020年。[20] L. Busoniu、R. Babuska 和 B. De Schutter，《多智能体强化学习综合综述》，IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.，第 38 卷第 2 期，第 156–172 页，2008年。[21] Z. Zhang、P. Cui 和 W. Zhu，《图上的深度学习：综述》，IEEE Trans. Knowl. Data Eng.（在线优先），2020年。[22] D. Silver、G. Lever、N. Heess、T. Degris、D. Wierstra 和 M. Riedmiller，《确定性策略梯度算法》发表于 ICML，2014年。[23] J. Chen、Y. Yang、C. Wang、H. Zhang、C. Qiu 和 X. Wang，《基于有向无环图的边缘计算多任务卸载策略优化》，IEEE Internet of Things Journal，第 9 卷第 12 期，第 9367–9378 页，2021年。[24] Z. Liu、J. Song、C. Qiu、X. Wang、X. Chen、Q. He 和 H. Sheng，《通过移动边缘计算中的多出口深度神经网络加速推理流卸载》，IEEE《移动计算汇刊》，2022年。[25] W. Lv、Q. Wang、P. Yang、Y. Ding、B. Yi、Z. Wang 和 C. Lin，《基于深度Q学习的边缘计算微服务部署》，IEEE Trans Parallel Distrib Syst，2022年。[26] Z. Ding、S. Wang 和 C. Jiang，《具有动态资源分配的 Kubernetes 面向微服务部署》，IEEE Trans. on Cloud Comput.，第 01 期，第 1–1 页，2022年。[27] X. Wang、Z. Ning 和 S. Guo，《普适边缘计算的多智能体模仿学习：一种去中心化计算卸载算法》，IEEE Trans Parallel Distrib Syst，第 32 卷，425，2021年 2，411no. pp. – 。[28] S. Jo˘silo 和 G. D´an，《移动边缘计算中周期性任务的计算卸载调度》，IEEE/ACM Trans. Netw.，第 28 卷第 2 期，第 667–680 页，2020年。[29] “Aliababa-clusterdata。” [在线]。地址：https://github.com/alibaba/ clusterdata[30] X. Chen、L. Jiao、W. Li 和 X. Fu，“面向移动边缘云计算的高效多用户计算卸载”，IEEE/ACM 网络汇刊，第 24 卷，第 5 期，第 2795–2808 页，2016年。[31] V. Mnih、K. Kavukcuoglu、D. Silver、A. A. Rusu、J. Veness、M. G. Bellemare、A. Graves、M. Riedmiller、A. K. Fidjeland、G. Ostrovski 等，“通过深度强化学习实现人类水平控制”，《自然》，第 518 卷，第 7540 期，第 529–533 页，2015年。[32] D. Silver 等，“借助深度神经网络和树搜索掌握围棋”，《自然》，第 529 卷，第 7587 期，第 484–489 页，2016年。[33] E. Greensmith 等，“强化学习中梯度估计的方差缩减技术”，《机器学习研究杂志》，2004年。[34] “轻量级 kubernetes。” [在线]。地址：https://github.com/ rancher/k3s[35] R. Hong 和 A. Chandra，“DLion：微云中的去中心化分布式深度学习”，载于 USENIX HotCloud，2019年。[36] A. Barbalace、M. L. Karaoui、W. Wang、T. Xing、P. Olivier 和 B. Ravindran，“边缘计算：异构 ISA 容器迁移的理由”，载于 ACM VEE，2020年，第 73–87 页。[37] Y. Bengio、J. Louradour、R. Collobert 和 J. Weston，“课程学习”，载于 ACM ICML，2009年，第 41–48 页。[38] “K8s 文档：水平 Pod 自动伸缩器。”[在线]。地址：https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ [39] I. Gog、M. Schwarzkop 等，“Firmament：大规模快速集中式集群调度”，载于 USENIX OSDI，2016年。[40] B. Perozzi、R. Al-Rfou 和 S. Skiena，“DeepWalk：社会表示的在线学习”，载于 ACM SIGKDD，2014年，第 701–710 页。[41] X. Kong、S. Tong、H. Gao、G. Shen、K. Wang、M. Collotta、I. You 和 S. Das，“可穿戴物联网的移动边缘协作优化：一种基于网络表示的框架”，IEEE 工业信息学汇刊，2020年。[42] X. Huang、R. Yu、S. Xie 和 Y. Zhang，“车载边缘计算与网络中用于计算卸载的任务－容器匹配博弈”，IEEE 智能交通系统汇刊，2020年。[43] L. Dong、M. N. Satpute、J. Shan、B. Liu、Y. Yu 和 T. Yan，“面向多用户的移动边缘计算卸载”，载于 IEEE ICDCS。IEEE，2019年，第 841–850 页。[44] Y. Liang、J. Ge、S. Zhang、J. Wu、L. Pan、T. Zhang 和 B. Luo，“边缘计算中面向交互的服务实体放置”，IEEE 移动计算汇刊（提前出版），2019年。[45] P. Kayal 和 J. Liebeherr，“雾计算中的分布式服务放置：一种迭代组合拍卖方法”，载于 IEEE ICDCS。IEEE，2019年，第 2145–2156 页。[46] H. Badri、T. Bahreini、D. Grosu 和 K. Yang，“移动边缘计算中的能耗感知应用放置：一种随机优化方法”，IEEE 并行与分布式系统汇刊，第 31 卷，第 4 期，第 909–922 页，2019年。[47] J. Zou、T. Hao、C. Yu 和 H. Jin，“A3C-DO：一种基于深度强化学习的边缘场景区域资源调度框架”，IEEE 计算机汇刊，第 70 卷，第 2 期，第 228–239 页，2021年2月。[48] W. Xiao、Z. Han 等，“Gandiva：面向深度学习的内省式集群调度”，载于 USENIX OSDI，2018年。[49] D. Haja、M. Szalay、B. Sonkoly、G. Pongracz 和 L. Toka，“让 Kubernetes 更适用于边缘”，载于 ACM SIGCOMM 海报与演示，2019年。[50] F. Rossi、V. Cardellini、F. Lo Presti 和 M. Nardelli，“使用 Kubernetes 高效部署地理分布式容器”，《计算机通信》，第 159 卷，第 161–174 页，2020年6月。



**沈世豪**于2019年获得中国天津大学理学学士学位。目前，他正在天津大学攻读博士学位。他的研究兴趣包括边缘计算、集群调度、强化学习和随机优化。

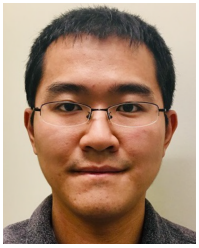


**Yiwen Han** 于2015年获得中国南昌大学通信工程学士学位（优秀毕业生），并分别于2018年和2022年获得中国天津大学理学硕士学位和博士学位（优秀毕业生）。他于2016年和2021年两次获得国家奖学金。目前，他从事公共安全信息技术研究，研究兴趣包括边缘计算、强化学习和深度学习。



**王晓飞** 获得了中国华中科技大学的理学学士学位，并在韩国首尔的首尔大学获得了硕士和博士学位。2014年至2016年，他在加拿大温哥华的不列颠哥伦比亚大学担任博士后研究员。目前，他是中国天津天津大学智能与计算学部的教授。他专注于边缘计算、边缘智能和边缘系统研究，已发表160多篇技术论文，刊载于

IEEE JSAC、TCC、ToN、TWC、IoTJ、COMST、TMM、INFOCOM、ICDCS等。他曾获得IEEE ICC和ICPADS最佳论文奖，并于2017年荣获“IEEE ComSoc Fred W. Ellersick奖”，于2022年荣获“IEEE ComSoc亚太杰出论文奖”。



**王世强** 是美国纽约州 IBM T. J. Watson研究中心的资深研究科学家。他于2015年获得英国伦敦帝国理工学院博士学位。目前，他的研究聚焦于分布式计算、机器学习、网络和优化的交叉领域，应用范围广泛，包括数据分析、基于边缘的人工智能（边缘人工智能）、物联网和未来无线系统。他在边缘计算和联邦学习领域作出了奠基性贡献，带来了学术方面的成果。

以及产业影响力。王博士担任 IEEE《移动计算汇刊》和 IEEE《并行与分布式系统汇刊》的副编辑。他于2021年获得 IEEE通信学会（ComSoc）Leonard G. Abraham奖、2021年获得 IEEE通信学会工业界最佳青年专业人士奖、2019年、2021年和2022年获得 IBM杰出技术成就奖（OTAA），自2016年以来多次获得 IBM发明成就奖，并于2019年入围 IEEE国际图像处理会议（ICIP）最佳论文奖，2015年获得网络与信息科学国际技术联盟（NIS-ITA）最佳学生论文奖。



**Victor C.M. Leung** 是中国深圳大学计算机科学与软件工程杰出教授。他同时是不列颠哥伦比亚大学（UBC）电气与计算机工程荣休教授，以及无线网络与移动系统实验室主任。其研究涵盖无线网络和移动系统等广泛领域，并在这些领域发表了大量论文。他的著作累计获得超过 50,000 次引用。他目前被列入了

科睿唯安“高被引研究者”名单。Leung博士担任IEEE绿色通信与网络汇刊、IEEE云计算汇刊、IEEE计算社会系统汇刊、IEEE Access及其他多种期刊的编委。他获得了1977年APEBC金奖、1977至1981年NSERC研究生奖学金、IEEE温哥华分会百年纪念奖、2011年UBC基拉姆研究奖、2017年加拿大电信研究奖、2018年IEEE TCGCC杰出技术成就表彰奖以及2018年ACM MSWiM雷金纳德·费森登奖。他参与撰写的论文曾获得2017年IEEE ComSoc弗雷德·W·埃勒西克奖、2017年IEEE Systems Journal最佳论文奖、2018年IEEECSIM最佳期刊论文奖以及2019年IEEE TCGCC最佳期刊论文奖。他是IEEE终身会士，也是加拿大皇家学会（科学院）、加拿大工程院和加拿大工程学会会士。