

本 科 毕 业 设 计（论文）



题目: 基于QOS保障的分布式任务分配方法的研究和仿真

姓 名 林子川

学 院 人工智能学院

专 业 信息工程

班 级 2020219106

学 号 2020212327

指导教师 李珂

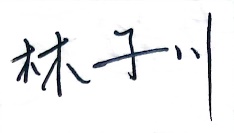
2024 年 5 月

北 京 邮 电 大 学

本科毕业设计（论文）诚信声明

本人声明所呈交的毕业设计（论文），题目《 基于QOS保障的分布式任务分配方法的研究和仿真 》是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。



本人签名： 日期： 2024年5月7日

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存。汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）



本人签名： 日期： 2024年5月7日

导师签名： 日期： 2024年5月7日

**基于QOS保障的分布式任务分配方法的研究和仿真**

**摘 要**

智能万物互联时代带来了数据的爆炸式增长，为缓解海量数据处理对计算资源的需求，云边端的分布式服务架构应运而生。因此，任务处理由集中式处理向分布式协同处理方向发展。然而，由于车联网或工业互联网等对服务质量提出较高要求，在分布式架构下，如何进行有效的任务协同分配以保障系统的服务质量，是一个亟需解决的关键问题。因此，本文以保障服务质量为目标，关注任务的截止时刻，提出两种任务分配方法，具体如下：

1. 基于MaxUtil改进的延迟确认策略

MaxUtil考虑了一个最优化函数，按照最优化函数对任务优先级进行排序，并按排序对任务分配计算资源，相比于随机任务分配取得了更好的效果，降低了完成多个任务的能耗，但是MaxUtil并没有考虑到先后到达任务的紧迫性不同，这导致在任务到达密集的情况下，超时的任务数不足够少。针对此问题，本文对服务器计算资源的选择提出了延迟确认策略。当任务被分配到计算资源，但是并不是立即开始执行时，便将任务放入延迟列表中，和后续到达任务一同安排计算资源。实验结果表明，使用延迟确认策略时，考虑单位时间内到达任务数不同，大部分情况下，相比于单独使用MaxUtil策略，超时任务数有不同程度的减少。

1. 基于强化学习的任务分配方法

相比于传统方法，基于强化学习的任务分配方法能够更好捕捉到任务的特征。本文提出一种使用近端策略优化强化学习训练的任务分配方法。并且任务分配计算资源的步骤在时间上是连贯的，这和近端策略优化强化学习中使用马尔可夫过程计算奖励相对应，因此强化学习能够较好地在任务分配计算资源场景上进行应用。在训练过程中，本文结合了行为者-评价者的训练方式，考虑了基于一个边缘服务器集群有多台服务器提供服务的场景的观察空间和动作空间的构建，并针对行为者输出变量的离散和连续情况，提出了共享输入层和隐藏层的神经网络。在动作空间范围设置无法达到场景要求的情况下，针对不同的出错情况设置不同的奖励函数。最终训练得到了一个用于任务分配计算资源的模型，其输入是降成一维并补足输入长度的观察空间向量，输出分为三个部分，分别是两个离散变量和一个连续变量。实验结果表明，针对服务质量、考虑任务超时的任务场景，基于强化学习的任务分配方法对数据集和训练结果的要求较高。

**关键词** 服务质量 分布式计算 任务分配 近端策略优化 强化学习

Research and Simulation of Distributed Task Allocation Method Based on QoS Guarantee

ABSTRACT

The era of intelligent Internet of Everything has led to an explosive growth of data. To alleviate the demand for computing resources required to process massive amounts of data, a distributed service architecture of cloud-edge collaboration has emerged. Consequently, task processing shifted from centralized processing to distributed collaborative processing. However, due to the high demands on service quality posed by applications like the Internet of Vehicles or Industrial Internet, how to effectively allocate tasks in a distributed architecture to ensure system service quality is a critical issue that urgently needs to be addressed. Therefore, this paper aims to ensure service quality by focusing on task deadlines and proposes two task allocation methods, which are as follows:

1. A delay confirmation strategy based on a traditional task allocation method

While MaxUtil considers an optimization function to prioritize tasks and allocate computing resources accordingly, it fails to account for the differing urgency of task arrivals. To address this issue, a delay confirmation strategy for server computing resource selection is proposed. When a task is allocated to computing resources but not immediately executed, it is placed in a delay list and scheduled along with subsequent tasks. Experimental results indicate that when considering different numbers of tasks arriving per unit time, in most cases, compared to using the MaxUtil strategy alone, the number of timeouts varies when employing the delay confirmation strategy.

1. A task allocation method based on reinforcement learning

Compared to traditional methods, the reinforcement learning-based task allocation method can better capture task characteristics. This paper proposes a task allocation method that utilizes proximal policy optimization for reinforcement learning training. Additionally, the steps for allocating computing resources for tasks are coherent in time, corresponding to the use of Markov processes to compute rewards in proximal policy optimization reinforcement learning. Therefore, reinforcement learning can be better applied to scenarios involving task allocation of computing resources. During training, this paper combines the actor-critic training method, considering the construction of observation and action spaces based on scenarios where multiple servers in an edge server cluster provide services, and proposes a neural network with shared input and hidden layers to address discrete and continuous output variables of the actor. In cases where the action space range cannot meet scenario requirements, different reward functions are set for different error scenarios. Finally, a model for task allocation of computing resources is trained, with its input being a one-dimensional vector reduced from the observation space and supplemented with input length, and its output divided into three parts, namely, two discrete variables and one continuous variable. Experimental results show that for task scenarios considering task timeouts and focusing on quality of service, the task allocation method based on reinforcement learning has high requirements for the dataset and training results.

**KEY WORDS**  Quality of Service Distributed Computing Resource Scheduling

Proximal Policy Optimization Reinforcement Learning

目 录

[第一章 引言 1](#_Toc168586563)

[1.1 研究背景与意义 1](#_Toc168586564)

[1.1.1 研究背景 1](#_Toc168586565)

[1.1.2 研究意义 1](#_Toc168586566)

[1.2 研究目标 1](#_Toc168586567)

[1.3 国内外研究现状 2](#_Toc168586568)

[1.3.1 分布式任务分配方法的分类 2](#_Toc168586569)

[1.3.2 分布式任务分配方法的应用 6](#_Toc168586570)

[1.3.3 任务分配主要指标 7](#_Toc168586571)

[1.4 论文组织结构 8](#_Toc168586572)

[第二章 理论背景和相关技术 11](#_Toc168586573)

[2.1 匹配理论介绍 11](#_Toc168586574)

[2.1.1 Pareto最优 11](#_Toc168586575)

[2.1.2 匈牙利算法 11](#_Toc168586576)

[2.1.3 延迟接受算法 11](#_Toc168586577)

[2.2 强化学习技术介绍 12](#_Toc168586578)

[2.2.1 基于值函数的方法 12](#_Toc168586579)

[2.2.2 策略梯度方法 12](#_Toc168586580)

[2.2.3 行为者-评价者方法 12](#_Toc168586581)

[2.2.4 深度强化学习 12](#_Toc168586582)

[第三章 基于MaxUtil改进的延迟确认策略 15](#_Toc168586583)

[3.1 任务分配方法MaxUtil概述 15](#_Toc168586584)

[3.2 问题建模 15](#_Toc168586585)

[3.2.1 符号表示 15](#_Toc168586586)

[3.2.2 任务模型 16](#_Toc168586587)

[3.2.3 分布式任务分配网络模型 18](#_Toc168586588)

[3.2.4 评价指标 19](#_Toc168586589)

[3.2.5 仿真流程 19](#_Toc168586590)

[3.3 基于传统算法的改进 20](#_Toc168586591)

[3.3.1 计算资源间隙优先级排序 21](#_Toc168586592)

[3.3.2 任务优先级排序 21](#_Toc168586593)

[3.4 性能对比 23](#_Toc168586594)

[3.4.1 实验参数 23](#_Toc168586595)

[3.4.2 实验结果 23](#_Toc168586596)

[3.4.3 性能分析与讨论 25](#_Toc168586597)

[3.5 本章小结 29](#_Toc168586598)

[第四章 基于强化学习的任务分配方法 31](#_Toc168586599)

[4.1 机器学习任务分配方法现状 31](#_Toc168586600)

[4.1.1 传统机器学习任务分配方法 31](#_Toc168586601)

[4.1.2 深度学习任务分配方法 31](#_Toc168586602)

[4.1.3 强化学习任务分配方法 31](#_Toc168586603)

[4.2 近端策略优化方法建模 32](#_Toc168586604)

[4.2.1 观察空间构建 32](#_Toc168586605)

[4.2.2 动作空间构建 34](#_Toc168586606)

[4.2.1 训练流程 35](#_Toc168586607)

[4.2.2 行为者网络 35](#_Toc168586608)

[4.2.3 奖励和环境更新 35](#_Toc168586609)

[4.2.4 评价者网络 37](#_Toc168586610)

[4.2.5 模型运用 37](#_Toc168586611)

[4.3 性能对比 37](#_Toc168586612)

[4.3.1 实验参数 37](#_Toc168586613)

[4.3.2 实验结果 40](#_Toc168586614)

[4.3.1 本章小结 41](#_Toc168586615)

[第五章 总结与展望 43](#_Toc168586616)

[5.1 论文工作总结 43](#_Toc168586617)

[5.2 任务分配方法展望 43](#_Toc168586618)

[5.2.1 传统策略方面 43](#_Toc168586619)

[5.2.2 强化学习方面 43](#_Toc168586620)

[5.2.3 场景扩展 43](#_Toc168586621)

[参考文献 45](#_Toc168586622)

[致 谢 51](#_Toc168586623)

[附 录 53](#_Toc168586624)

# **引言**

## 研究背景与意义

### 研究背景

近年来，信息与通信技术高速发展和第五代移动通信技术的出现与普及[1]使得人们对互联网的利用愈加便捷，业务种类也进一步增加，诸如虚拟现实、远程医疗、物联网技术等应用得到了普及和进一步发展。随着计算数据量的增加，云计算架构逐渐从仅有单一的云服务器向云边端分布式架构进行转变。根据中国信通院的预测，2025年前云计算市场规模增长率未来将保持在30%以上[2]。

相比于云计算，边缘计算将服务部署在更加靠近用户的边缘云服务器，降低数据传播时延。边缘云服务器和中心云服务器进行协同计算，根据任务类型的不同对任务进行调度，降低任务处理的时间[3–5] 。因此为了满足应用的低时延要求，边缘计算架构应运而生，但是边缘计算架构还面临有以下两个问题：

服务量急剧增加对服务在数据中心处的服务质量带来了新的挑战。在服务质量方面，随着用户对于互联网服务的需求不断增加，服务提供商需要确保其提供的服务在性能、可靠性和安全性等方面能够满足用户的要求。然而，由于服务量的增加可能导致服务质量下降，例如延迟增加、响应时间变慢以及服务中断的风险增加。这些问题可能会影响用户体验，降低用户对服务的满意度，甚至导致用户流失，对服务提供商造成损失。

服务量的急剧增加也带来了对能源的巨大压力。随着互联网服务规模的扩大，数据中心的数量和规模也在不断增加，而这些数据中心需要消耗大量的能源来运行和维护，并且随着互联网服务规模的增长，其能源消耗量也在不断增加。

综上所述，如何使用在边缘计算架构中提高服务质量、降低服务能耗成为了互联网服务提供商面临的一个紧迫问题。

### 研究意义

和以往相比，当前的应用对低时延有更高的追求。类似于计算机网络中的数据传输过程的时延组成，在服务过程中，用户从提出请求到获取数据所消耗的时间，包括传播时延、传输时延，另外也包括服务器对任务处理所需要的时间和用户设备对收到数据内容进行可视化处理的时间。在时延的多种组成部分[6]中，本文面向服务器负载较高的情境，选择对服务器处理任务时从输入服务器到输出服务器所需要的时间长度进行优化，使得在高负载的情况下，服务器处理任务和数据有更高的服务质量。

## 研究目标

为了进一步地提高服务质量，尽可能使得任务能够在所需要的时间内完成，本文研究和探索，在调度范围内的多台边缘服务器组成的集群中，在单位时间内到达任务密集程度不同的情况下，进行计算资源分配和安排的方法。具体而言，本文改进传统策略、另外提出使用深度强化学习方法，对任务和不同虚拟机上的计算资源进行匹配。

在本研究中，计算资源可以被看作是以多维属性分布在不同的虚拟机上的。计算资源包含的属性可以有：资源空闲率、开始时刻、结束时刻、所在的虚拟机编号。而对于一个计算任务，其包含的属性可以有：资源占用率、到达时刻、处理所需的时长、截止时刻。针对时间和占用率，本研究以服务质量保障为目标，让计算资源可以和计算任务在资源占用率/空闲率和占用时间上进行合适地匹配，使得尽可能多的任务在截止时刻之前尽早完成，同时关注不同的任务分配方式对能耗带来的影响。

## 国内外研究现状

### 分布式任务分配方法的分类

分布式任务分配主要指的是在分布式系统中任务被分配到不同的位置进行处理，可以是在一个边缘服务器的不同单元，也可以是在云边端网络上的不同的服务器中。其中，部分研究会考虑到多种不同的算法对分布式任务分配方法进行求解。

#### 最优化方法

最优化方法旨针对设置的最优化函数找到一个问题的局部或者全局最优解，其中可以加入一些条件的限制。在边缘计算计算资源分配的应用场景中，为了使得资源分配在各指标上最优，不同的研究提出了不同的最优化方法。另外，考虑到某些情况下最优化方法的时间复杂度较高，部分研究还基于最优的策略提出了时间复杂度降低、效果相近的近似方案。例如[7] 研究了物联网网络中雾资源配置的可靠性最大化和系统成本最小化的权衡，提出了一个整数线性规划问题。文献[8]提出了一个服务质量感知的雾服务提供框架FOGPLAN，并提出了一个可能的优化公式和两种有效的贪心算法。文献[9]研究了基于时分多址和正交频分多址的多用户移动边缘计算卸载系统的资源分配，将基于时分多址的最优资源分配表述为计算时延约束下最小化加权和移动能耗的凸优化问题，将基于正交频分多址的最优资源分配表述为混合整数问题，分别提出了次优算法。文献[10]提出了一种基于链的物联网应用服务请求模型，并将其描述为整数线性规划问题，提出了一种两阶段优化方案。文献[11]提出了随机混合整数非线性规划问题，引入Lyapunov优化理论将原始问题分解为4个独立的子问题，通过凸分解方法和匹配博弈求解。文献[12]从数学上阐释了包括任务卸载、执行顺序和计算资源分配这三个子问题在内的动态任务卸载和调度问题，并基于基于逻辑的Benders分解技术设计了一种新颖的分解，最终使得主问题和子问题能够收敛以产生最优解。文献[13]针对服务器队列积压提出了双队列工作负载调度模型，通过聚合3个子问题求解策略提出基于Lyapunov分解策略的边缘业务调度在线匹配算法LOMES，在保证移动设备和异构边缘服务器负载均衡的同时，获得最优的系统效用。

总之，最优化算法主要是将资源分配问题建模成凸优化问题、非凸优化问题或者非确定性多项式时间难题。在建模问题后，部分研究会将问题通过Lyapunov分解策略将一个问题分解成多个子问题进行解决。对于将资源分配问题建模为凸优化问题的方法，其优点在于该方法较为成熟并较容易获得次优结果，但是计算方法算法复杂度较高，在实际系统中实施具有一定的难度。而近似方法虽然解决了算法复杂度的问题，但是容易陷入局部最优。

#### 启发式算法

启发式算法主要通过启发式信息来引导搜索过程，以在大规模搜索空间中寻找到较优解或近似解，而启发式信息主要来自于生活或者自然。启发式算法包括简单启发式算法和元启发式算法。简单启发式算法是一种基本的启发式搜索方法，通常使用一些简单的规则或者启发式函数来引导搜索过程。而元启发式算法则是一种更高级的启发式搜索方法，其特点是能够自动地学习、调整和生成启发式信息，以改进搜索过程并提高算法性能。在边缘计算任务分配计算资源领域中，目前大多数运用启发式算法的工作使用贪婪算法，而一些工作使用局部搜索算法[14]。

一些研究使用简单启发式算法。文献[15]使用贪婪的策略，以能耗作为目标提出了ECTC和MaxUtil两种资源整合方法。文献[16]建立了一个建立了一个雾计算的多副本数据放置模型 iFogStorM，来解决雾节点上需要放置多少个数据副本以及如何优化数据放置的问题，另外提出了一种基于贪婪算法的数据副本放置策略 MultiCopyStorage以减少总延迟，其使用剪枝方法对劣等解决方案进行过滤。文献[17]构建了两个模型，分别使用随机排序和提出一种基于上限的技术进行解决。文献[18] 研究了基于用户服务功能链请求的虚拟网络功能放置问题，考虑了虚拟网络的实施方法和链接，并将问题转化为一个整数线性规划模型，目标资源总消耗最小；其利用最优化求解器，得到了 SFCrs 数目较小时的最优解，并在问题规模较大时，提出了一种多项式时间启发式算法。

还有一些研究使用了元启发式算法。元启发式算法包括遗传算法、蚁群算法、粒子群优化、模拟退火和禁忌搜索等[14]。文献[19]考虑雾节点当前负载和传感器与雾节点的通信延迟，提出了一个针对雾节点上映射数据流问题的优化模型，设计了一种基于遗传算法的可扩展启发式算法，用于解决服务放置问题。文献[20]提出了计算卸载模型，并将该问题表述为NP-hard的混合整数非线性规划问题，利用遗传算法和粒子群优化的优势，设计了一种次优算法。文献[21]考虑负载均衡，提出了基于用户访问的服务放置算法和基于模拟退火算法的请求调度算法。文献[22]提出了一种名为COM的计算卸载方法，考虑移动设备的执行时间和能耗，确定数据/控制约束的计算任务的动态调度，并采用非支配排序遗传算法III解决了云边缘计算中任务卸载的多目标优化问题。文献[23]设计了一种基于禁忌搜索的任务调度启发式算法，研究了多用户干扰环境中的多用户协作移动边缘计算卸载系统，共同优化了卸载决策和计算资源分配，提出了一种称为BiJOR的双层优化方法，通过蚁群系统求解上层优化问题[14]。文献[24]提出了一种多目标灰狼优化器算法，以降低服务质量目标延迟和能耗。

启发式算法的优势在于，简单的算法即可起到较好的作用。但是同最优化方法中的近似算法一样，容易得到局部最优而非全局最优。元启发式算法的参数较多，不利于调整，且难以复用计算结果；另外部分元启发式算法，例如遗传算法的计算复杂度较高，不利于实际应用[3,14]。

#### 机器学习方法

在机器学习的发展过程中，机器学习越来越能够适用于不同的领域，同时机器学习解决问题的能力越来越强。在边缘计算任务分配和资源调度研究领域中，近年来机器学习被广泛应用于优化资源利用、提高性能和降低能耗。传统方法通常是对复杂优化问题的静态解，无法根据不同环境进行动态变化，而机器学习方法可以比较好地解决这一问题。

通常任务和边缘环境的交互可以被建模成马尔可夫决策过程，可以通过强化学习来解决交互过程中的分配和调度问题。近期强化学习流行的方法包括近端策略优化方法、深度Q网络方法及深度确定性策略梯度算法。文献[25]基于双深度Q网络和深度确定性策略梯度算法提出一种双强化学习计算卸载算法，该算法联合决定计算卸载的卸载决策、CPU频率和发射功率。文献[26]基于在线到达模式和实际应用中的各种延迟约束，提出了一个任务迁移和合并的最后期限违约率 最小化计算卸载方案，开发了一个支持迁移的多优先级任务排序算法，提出了一种基于深度确定性策略梯度的学习算法来寻找最优卸载策略。文献[27]在车载边缘计算的背景下，在常规近端梯度优化算法的基础上设计了一种名为多作用和环境自适应近端策略优化算法（Multi-action and Environment-adaptive Proximal Policy Optimization algorithm, MEPPO）的新型算法，并提出了一种基于所设计的MEPPO算法的联合任务调度和资源分配方法。文献[28]设计了一个用于任务加速的自适应调度框架，在充分考虑数据依赖性、资源条件和网络条件的情况下，提出Meta-AC调度算法，将策略梯度与元学习相结合，将平均任务时延和超时任务比例降至最低。

由于分布式的特性，联邦学习也被广泛地运用于分布式任务分配过程中。联邦学习是一种机器学习技术，利用这种技术可以在多个不交换本地数据样本的分布式边缘设备或服务器上训练资源调度算法[29]。文献[30] 为基于移动边缘计算的联邦学习制定了一种能源感知资源管理，其中模型训练损失和总时间消耗被最小化，同时考虑了移动设备的能量限制。考虑了一个两层次的资源配置和激励机制设计问题。文献[31]应用进化博弈理论来模拟集群选择过程的动力学，提出了一个基于深度学习的拍卖机制来推导每个簇头服务的价值。文献[32] 首次研究了异构计算和无线功率传输在联邦学习中的应用，提出了一个基于异构移动体系结构的异构计算和资源分配框架，以有效实现联邦学习。文献[33] 为了处理边缘动力学问题，提出了一种自适应异步联邦学习机制。根据网络情况在不同时代智能地改变局部更新模型的数量，以便进行全局模型聚合；并提出基于深度强化学习的经验驱动算法，以自适应地确定两种情况下的学习任务（单个学习任务和多个学习任务）在每代的最优值，从而在资源约束下实现更少的训练完成时间。

强化学习方法在解决复杂问题方面具有优势，在一定场景下可以做到比传统方法更好的效果，但是在训练过程中需要的数据量较大，训练消耗的计算资源较多，可解释性较低，训练时间较长，需要持续改善网络结构和调整参数。联邦学习允许在本地设备上执行模型训练，然后将模型参数更新汇总到中央服务器进行集成，从而实现全局模型的更新，使得整体模型能够获得较好的性能提升；并且其无需将原始数据上传到服务器，保障了用户隐私。但是在模型更新的过程中频繁传递参数需要较大的通信开销，同时通信过程中传递的模型参数可能仍然含有一定的用户隐私，可能被窃取。

#### 博弈论方法

博弈论推理已经渗透到经济理论中，并在其他社会科学和行为科学中得到广泛应用[34]。它能够比较好地考虑到多方利益，促进多方协商。在边缘计算任务分配和资源调度领域，博弈论可以考虑多方需求，因此被广泛应用于分析和设计多方参与的资源分配和协作策略。在边缘计算中，部分研究通过建立博弈模型来分析边缘节点之间的合作与竞争关系，设计合理的资源分配策略。文献[35]提出了一种机会主义合作卸载 （Opportunistic Cooperative Offloading, OCO） 策略，研究了系统信息可用性3种情况下的帮助程序调度，即全局、部分和距离信息案例，并针对每种情况制定了卸载-中断最优调度方案。文献[36]提出了一种在多路边单元 车载边缘计算场景下边缘-边缘协同的车载边缘计算联合任务卸载和资源分配，将卸载到高负载边缘服务器的任务进一步卸载到其他低负载边缘服务器，采用精确势博弈对路侧单元间任务卸载的竞争过程进行建模，采用两阶段迭代算法将优化问题分解为两个阶段并迭代求解。文献[37]在车载边缘计算场景下，将协同资源优化问题分解为任务卸载和资源分配；将任务卸载子问题建模为精确势博弈，并提出了一种多智能体分布式深度确定性策略梯度（Multi-Agent Distributed Distributional Deep Deterministic Policy Gradient, MAD4PG）来实现纳什均衡；将资源分配子问题划分为两个独立的凸优化问题，利用梯度迭代方法和KKT条件提出最优解。文献[38]针对任务分配和调度规划，提出了一个多服务器多任务分配调度（Multi-server Multi-task Allocation and Scheduling, MMAS）问题，以实现移动边缘计算系统利润最大化，并就问题提出一种分布式方案和一种集中式方案；在分布式方案中，将多服务器多任务分配调度问题转化为非合作博弈，提出了一种收敛到纳什均衡的低复杂度响应更新算法；集中式方案基于贪婪的想法，并以集中方式在移动边缘计算控制器上运行。

博弈论能够解决较为复杂的决策问题，并且具有可解释性，但是对于大规模的边缘计算资源分配问题，博弈论方法可能需要考虑大量的参与者和多个决策变量，导致问题的求解难度和计算复杂度增加，难以建立准确的模型。另外在博弈论方法中，可能存在多个均衡解或者局部最优解，难以确定哪一个是全局最优解，且这些解可能会随着参与者策略的变化而发生变化，解的稳定性较差。

#### 匹配理论

匹配理论是近十几年来兴起的经济学理论，是对匹配市场的一种研究[39]。在使用匹配理论的时候，如果只有两部分类型的内容需要进行匹配，可以将匹配理论的应用场景抽象成一个二分图，可以将需要进行匹配的两方分别看成是图的一部，将二分图的两部中分别选择一个节点进行连接即代表内容的匹配。匹配的内容类型也可以不只有两种类型，也可以有多种类型，此时可以根据需要将问题建模为多分的节点和边，但多分的情况较为少见。对于边缘计算，可以将任务和计算资源分别看成两类节点。文献[40] 建立了一个匹配理论框架，将三个相互冲突的目标视为任务、服务功能和无人机的偏好，然后提出一种基于分布式匹配理论的重新分配算法。文献[41] 针对雾网络提出了一种分布式计算卸载框架（Distributed Computation Offloading framework, DISCO）以使用匹配理论卸载可拆分任务。文献[42]以通过共同优化单个计算决策、用户的传输能力和服务器上的计算资源来最大限度地减少系统范围的计算开销为目标，采用了一种次优方法，将原始问题分为计算卸载决策和联合资源分配，研究了两种匹配算法。文献[43]提出了两种Fog域中的虚拟函数放置方法，所考虑的解决方案制定了一个具有外部性的匹配博弈，旨在最大限度地减少最差的应用程序完成时间和中断的应用程序数量。

匹配理论能够有效地处理多对多的资源分配问题，适用于边缘计算中多个设备提供服务的场景，具有较强的适应性和稳定性。但是在解决边缘计算资源分配问题时，对于信息的准确度要求较高，另外计算复杂程度较高，在有大量信息的情况下计算得到解决方案需要较多的时间。此外，匹配理论具有抗策略性问题。如果一方使用虚假的信息获取对自身更有利的资源，则会导致整体的资源分配效果下降。

### 分布式任务分配方法的应用

分配式任务分配方法已经被应用于多个领域，包括但不限于无人机联网、自动驾驶车联网、视频服务、智慧医疗、智能城市、智能健康、智能制造和智能家居等应用[14]。以下以无人机联网、自动驾驶车联网和视频服务为例介绍目前研究对于分布式任务分配方法的应用。

#### 物联网

物联网是万物的联网。物联网可以根据连接物体的属性而被定义为某种特定的网络，例如无人机联网、自动驾驶车联网等。其中每一种物体的属性又会影响物联网内数据传输的类型，以及影响任务分配方法的策略。

在无人机联网方面，文献[44] 采用深度 Q 学习算法针对一个无人航空载具提供移动计算服务的情境，研究了任务优先级和移动计算服务对移动边缘计算网络的联合影响。文献[45] 研究了多无人机移动边缘计算网络中，通过联合优化用户关联、功率控制、计算容量分配和位置规划来实现总功率最小化的非凸问题，提出了一种低复杂度的算法，迭代求解三个子问题。文献[46] 通过联合优化无人机轨迹、用户发射功率和计算负载分配来最大限度地提高无人机的能量效率，将得到的最佳化问题对应于非凸分数规划，采用丁克尔巴赫算法和逐次凸近似（Successive Convex Approximation, SCA）技术求解，并进一步将问题分解为多个子问题，用于分布式并行问题求解。

在车联网方面，文献[47] 针对传统的不对称网络资源分配的上行链路和下行链路连接会大大降低性能的车辆连接的应用程序的问题，提出了一种基于边缘计算的端到端汽车边缘网络系统 FAIR，其核心是部署在边缘服务器上的对称网络资源分配算法和智能车辆上的服务自适应算法。文献[48]为了解决大多数关于 车辆边缘计算网络 中任务卸载的现有工作并没有考虑到边缘服务器上计算资源的负载平衡的问题，和尽量减少车辆计算任务的处理延迟，在 FiWi 增强型 车辆边缘计算网络中引入 软件定义网络，为集中式网络和车辆信息管理提供支持。文献[49] 提出了一个基于自学习的分布式计算卸载算法来达到一个纳什均衡点，即在车辆之间得到一个相互满意的解决方案，而且没有车辆愿意改变其决策。

#### 视频服务

在视频服务中，分布式任务分配方法通过智能地分配任务给不同的服务器或节点，以优化视频服务的性能和质量。这些算法可以根据服务器的负载、带宽、延迟等因素，动态地将用户请求分配给最合适的服务器，以提供更快速、稳定和高质量的视频播放体验。与基于云的解决方案相比，分析邻近边缘节点上的视频帧可以有效地降低响应延迟。同时，视频服务的进一步发展也推动了增强现实、虚拟现实、自动驾驶等各种应用的发展。文献[50]设计了一个基于边缘协调结构的联合视频查询调度和资源分配问题，以适应终端摄像机、边缘节点或云上的实时视频查询。在文献[51]中，VideoEdge被提出以优化计算机视觉组件的放置，其中解决了两个挑战，包括由多个资源提供者和合并冲突引起的指数级大搜索空间。文献[52] 研究了多边缘节点协同完成 增强现实 任务的问题，其中增强现实任务的组成部分被建模为具有依赖性的有向无环图，设计了一个基于整数粒子群优化的算法和一种启发式算法。

### 任务分配主要指标

#### 服务时延

大部分情况下，服务质量指标指的是服务时延。边缘计算或边缘云计算通过以分布式方式巧妙地部署各种服务，在降低服务延迟方面具有巨大潜力。以服务时延作为任务分配评价指标的有以下案例：文献 [53] 积极地将雾计算和 医疗网络物理系统集成在一起，共同研究了基站关联、任务分配和虚拟机布局，首先把问题表述成一个混合整数非线性规划，线性化后并为了解决计算复杂度问题，进一步提出了一种基于线性规划的两阶段启发式算法。文献[54]设计并开发了一个轻量级的深度学习模型来支持移动边缘计算在交通网络物理系统中的应用。文献[55] 提出了一种基于雾云辅助虚拟物理系统的医疗保健服务，为患有蚊传疾病的受感染用户提供远程监测和早期检测；在该系统中，通过计算相似因子来区分多目标决策树，然后使用 J48决策树分类器对每个用户的感染类别进行分类。

#### 能耗

当前，大量终端设备是供电有限的。在此情况下，终端设备在参与进边缘计算时的能耗成为了一件需要考虑的事情。因此，部分研究以能耗为评价指标：文献[56] 着重研究了基于雾计算的网络物理物联网系统中通信资源和计算资源的联合优化问题，以最大限度地提高系统的整体能量效率，并提出了一种概率分布方法将该问题重构为非概率形式，并采用基于 Dinkelbach 算法和拉格兰奇对偶方法的资源分配算法来有效地解决该问题。文献[57]共同考虑包括延迟、能源消耗和云租赁费用的系统成本，将问题表述为一个随机最佳化问题，用李亚普诺夫技术将长期最佳化问题分解为一系列仅使用当前系统信息的单时隙优化混合整数线性问题。文献[58] 基于近端梯度优化算法设计了一种新的算法——多动作环境自适应近似策略优化算法，并在此基础上提出了一种联合任务调度和资源分配方法。

#### 安全和隐私

在进行任务分配的时候，尤其是任务分配算法在多台设备上进行的时候，可能会涉及到用户的安全隐私问题。文献[59]提出了一种基于动作约束深度强化学习的方法来安全地分配计算资源。文献[60] 使用 Wyner 的窃听编码方案来获得可达到的保密率，并保证机密信息不会被多个移动窃听者破译，以在确保无线卸载过程安全的同时，尽量减少系统处理延迟，并将最佳化问题表述为一个多智能体协同最优决策问题，使用双深度 Q 学习算法解决。文献[61] 考虑了一个具有窃听器的海上边缘计算系统，其中无人水面飞行器和无人无人航空载具协同卸载和计算任务，在考虑物理层安全的情况下最大限度地提高系统的整体性能。

#### 系统可靠性

在日常生活中，硬件设备会以一定的频率随机发生故障。为了保证整个边缘计算系统的可靠性和稳定性，不得不考虑组成系统的单个设备故障的情况。对于某些应用场景，对硬件设备故障的发现、排除，以及在单个设备故障的时候整个系统依然能够稳定运行是一项需要仔细考虑的情况。文献[62]研究了基于层共享的微服务部署优化问题，目的是在满足所有任务延迟要求的前提下最大化系统的可靠性；其考虑到动态任务的产生以及不同触发器下各种决策变量的异步化，利用改进的李雅普诺夫技术，结合随机舍入、拉格朗日方法和凸优化，设计了一种在线优化算法，在不同时间尺度上迭代求解问题。文献[63] 提出了一个基于多队列模型中平均等待时间的概率分布的更通用的可靠性度量；为了激励多用户单元分担更多的任务，提高边缘服务器提供商的经济效用，该研究还采用了 Stackelberg 博弈模型来模拟 边缘服务器提供商 和多用户单元之间的动态交互，利用交替方向乘子算法推导出 边缘服务器提供商和移动用户的最优策略。文献[64]提出了一种基于可靠性的在线调度方案，通过利用在线反馈和离线数据来了解 边缘服务器 的不确定可用性该研究将可靠性感知的深度神经网络推理任务调度问题描述为一个新的约束组合多臂老虎机（Combinatorial MultiArmed Bandit, CMAB）问题，结合 Lyapunov最佳化技术、老虎机学习、近似子模块最大化和历史数据，设计了一个带有老虎机学习算法的可靠性感知任务调度方案（Reliability-aware Task scheduling scheme with a Bandit Learning, RTBL）来解决这个问题，以最大限度地提高 深度神经网络推理任务的推理精度和服务可靠性。

本文重点关注服务时延和能耗这两个指标，并且在服务时延的基础上，考虑任务的截止时刻，以任务分配策略使用生成的数据集在测试过程中超时的任务数来评价任务分配计算资源策略的好坏。以此为目标对算法进行改进，同时提出一种新的基于强化学习训练的模型的任务分配计算资源的方法，使得超时的任务数尽可能减少，同时新的策略又不会使得完成多个任务的能耗有较大的增加。

## 论文组织结构

本论文一共分为五章。图 1‑1展示了论文的组织结构。

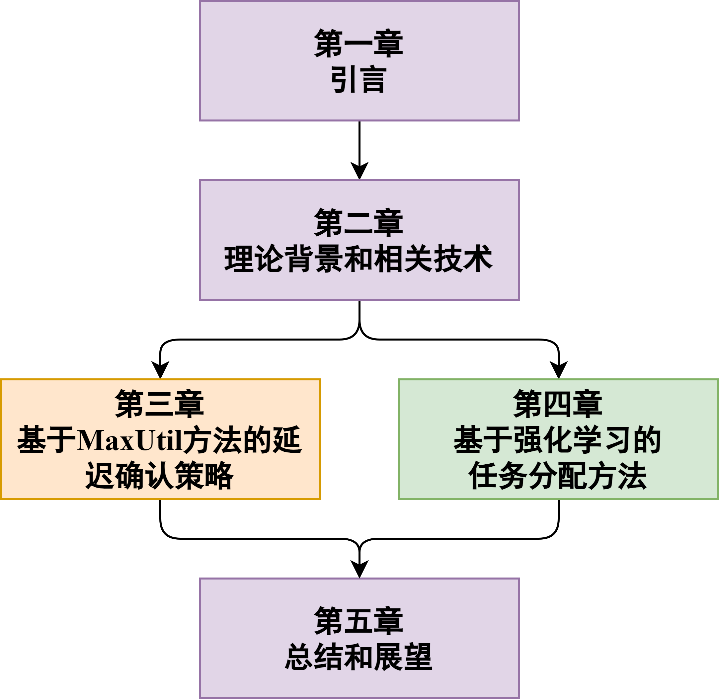


图 1‑1 论文组织结构

本章为引言，主要介绍了研究背景与意义、研究目标与问题陈述、当前国内外研究现状。其中国内外研究现状主要考虑了当下分布式任务分配方法的分类、对任务分配方法的应用和各种任务分配方法的主要评价指标。

第二章主要介绍在以服务质量作为评价指标，对任务分配方法进行改进和仿真的过程中进行的学习和调研，主要包括匹配理论、强化学习、图机器学习和分布式计算部署平台技术进展。

第三章提到了一种基于MaxUtil[15]进行改进的任务分配方法，改进可以使得其在服务质量这一指标上更优而能耗没有太大增长。主要介绍了MaxUtil算法、问题的建模和随机生成的测试数据集的构建方法、实验结果。

第四章主要介绍使用近端策略优化强化学习方法对任务分配模型进行训练并对模型进行运用。其中包括了机器学习任务分配计算资源方法的现状、训练和测试模型过程的建模及使用模型的效果和传统策略的对比。

第五章主要介绍了传统和使用强化学习策略在未来可能的发展以及论文采用的任务分配方法在未来的工作中可能可以进行的改进。

# 理论背景和相关技术

## 匹配理论介绍

匹配理论对边缘计算任务分配的影响显著。通过匹配算法，可以根据任务的性质、资源的可用性以及通信延迟等因素，有效地将任务分配给边缘节点，实现任务的优化分配和执行。这种匹配机制能够提高边缘计算系统的效率和性能，降低通信延迟，同时最大程度地利用边缘节点的计算资源，从而更好地满足用户需求。对于匹配目标，通常目标分为最大匹配和稳定匹配。

### Pareto最优

有多个目标时，由于目标之间的冲突和无法比较的现象，一个解在某个目标上是最好的，在其他的目标上可能是最差的，这些在改进任何目标函数的同时，必然会削弱至少一个其他目标函数的解称为非支配解或Pareto解。

帕累托最优描述的是一种资源最优化配置的状态。在帕累托最优的条件下，是没有办法在不让某一参与资源分配的一方利益受损的情况下，令另一方获得更大利益。

Pareto最优解可以有多个。

### 匈牙利算法

匈牙利算法[65]的基本思想是通过不断寻找增广路径来寻找最大匹配。它的核心步骤包括：

1. 初始化：将所有顶点都标记为未匹配状态，将所有边的权重都设置为0。

2. 不断寻找增广路径：从未匹配的二分图的一部顶点开始，尝试找到一条增广路径，即一条交替经过未匹配边和已匹配边的路径。如果找到增广路径，则将匹配状态更新，并更新路径上的边的权重。

3. 重复步骤2，直到无法找到增广路径为止。

匈牙利算法的时间复杂度为 ，其中是顶点的数量。

基于匈牙利算法改造的KM算法是一种解决加权二分图带权最大匹配问题的有效算法。具体而言，在一个二分图中，每条边都有一个权重，目标是找到一种匹配方式，使得在满足最大匹配的条件下，所有匹配的边的权重之和最大化。

### 延迟接受算法

匹配的稳定是这样定义的：一个匹配是稳定的，如果它没有被任何个人限制且没有被任何配对限制。

没有被个人限制指的是组成的匹配为双方且接受，没有被任何配对限制指的是不存在更优的匹配方案使得双方收益，那么这个匹配方案是稳定的。

延迟接受算法[66]的步骤如下：

1. 初始化：所有未被匹配的申请人（如学生）向他们最偏好的一方（如学校）提交申请。

2. 女方考虑申请：每所学校考虑所有收到的申请，并暂时接受其最偏好的申请者。

3. 拒绝与重新申请：被学校暂时拒绝的申请者将向他们的次优选择提交申请。

4. 重复过程2、3，直到所有申请人都被匹配，或者没有新的申请可以提交（即每个申请人都已向所有偏好的一方提交过申请）。

匈牙利算法和延迟接受算法最大的区别就是，匈牙利算法追求最大匹配，而延迟接受算法追求稳定匹配。

## 强化学习技术介绍

### 基于值函数的方法

#### Q学习

Q学习是一种经典的基于值函数的强化学习算法。其核心思想是通过更新动作值函数（Q值函数）来优化策略，从而使得长期累积奖励最大化。Q学习通过贝尔曼方程进行迭代更新，将当前状态下的动作价值估计与后继状态的最大动作价值之和作为目标。这一算法在实际应用中具有简单、直观、易于理解的特点，但对于大规模状态空间问题的处理存在挑战。

#### 时序差分学习

时序差分学习是另一种基于值函数的强化学习方法，旨在通过自举的方式逐步更新值函数的估计。与Q学习不同，时序差分学习在更新过程中不需要考虑后继状态的最大动作价值，而是直接使用当前状态的即时奖励和后继状态的估计值函数。这一方法的代表性算法包括TD(0)算法和TD()算法，它们在处理连续状态空间问题时具有较好的效果。

### 策略梯度方法

策略梯度方法直接优化策略函数，而不是值函数。这些方法通常通过梯度上升更新策略参数，以最大化长期累积奖励。其中REINFORCE算法通过对策略函数的梯度进行估计和更新，来使得期望累积奖励最大化。然而，这些方法往往面临着高方差和收敛性能差的问题，限制了它们在实际任务中的应用。

### 行为者-评价者方法

行为者-评价者方法将策略梯度方法与值函数方法相结合，通过评价者（值函数）的帮助来优化策略。其核心思想是同时学习策略和值函数，以提高训练的稳定性和效率。其中，异步优势行为者-评价者算法通过异步训练多个代理来加速学习过程，并结合了优势函数来估计动作的价值，从而实现了更加稳定和高效的训练。

### 深度强化学习

#### 深度Q网络

深度Q网络（Deep Q-Learning, DQN）是一种结合了深度学习和Q学习的方法，用于解决高维状态空间和离散动作空间的问题。它利用深度神经网络来近似和优化动作值函数，通过经验回放和目标网络等技术来提高训练的稳定性和效率。

#### 深度确定性策略梯度

深度确定性策略梯度（Deep Deterministic Policy Gradient, DDPG）是一种用于解决连续动作空间问题的深度强化学习算法。它结合了策略梯度和值函数方法，通过引入确定性策略和目标网络来提高训练的稳定性和收敛性能。

#### 近端策略优化

近端策略优化（Proximal Policy Optimization, PPO）是一种用于优化策略的近端方法，通过引入近端优化机制来限制策略更新的幅度，以提高训练的稳定性和效率。近端策略优化算法通过最大化更新前后策略的相对性能来优化策略参数。

近端策略优化算法在实际应用中常常结合了行为者-评价者方法。行为者-评价者结构中的行为者网络负责生成动作，而评价者网络则负责估计动作的价值。评价者网络可以是一个深度神经网络，用于估计状态-动作对的值函数。通过结合行为者-评价者方法，PPO算法可以利用评价者网络的价值估计来指导策略优化，从而提高训练的效率和稳定性。

#### 信赖域策略优化

信赖域策略优化（Trust Region Policy Optimization, TRPO）是一种改进的策略优化方法，通过引入信赖域来限制策略更新的幅度，以确保策略更新的安全性和稳定性。这一方法在训练过程中保持了较高的样本利用率，并具有较好的收敛性能和稳定性。

# 基于MaxUtil改进的延迟确认策略

## 任务分配方法MaxUtil概述

MaxUtil是一个基于贪心算法的边缘计算任务分配策略。其在每一步对每一个任务选择最优的服务器。其定义的优化目标如下：

其中为任务执行时间，表示第个任务资源占用率，表示第个任务在第个资源（可以是服务器）上执行时的优化目标值。表 3‑1是MaxUtil算法的伪代码表示[15]。

表 3‑1 MaxUtil流程伪代码

|  |  |
| --- | --- |
| **算法1** MaxUtil算法 | |
| 输入：任务和一个拥有个云资源的集合 | |
| 输出：一组任务和资源的匹配 | |
|  | 令 |
|  | **for**  **do** |
|  | 计算任务在计算资源上执行的代价函数 |
|  | **if**  **then** |
|  | 令 |
|  | 令 |
|  | end if |
|  | end for |
|  | 将任务分配到上执行 |

## 问题建模

### 符号表示

假设指的是全部任务集合， ，指的是任务集之中的任务；针对任务假设寻找的时间段起点为，寻找的时间段终点为；当前时刻为；任务的到达时刻为，任务的最长等待时长（不包含处理时间）为，任务的处理时长为，任务接收时长为，那么

假设最后选择出的间隙为集合，集合包含若干个时间区间； 指的是寻找计算资源间隙的任务的计算资源占用率；指的是处于调度范围的服务器集，指的是第个服务器；指的是寻找的属于的一个计算资源间隙的开始时刻，指的是寻找的属于的这个间隙的结束时刻；那么考虑所有调度范围内的服务器，集合满足以下条件：

其中还需要满足

指的是对于一个服务器在区间内每一个时刻的计算资源占用率中的最大值，指的是时刻的计算资源占用率。

其中可以表示为以下形式：

指的是上的某一个任务，指的是这个任务的开始处理时刻，指的这个任务的结束处理时刻，则需要是时刻正在执行任务的其中一个任务，指的是的计算资源占用率。

考虑集合中所有元素（时间段、间隙）对应的计算资源占用率中可以取的最大值可以如下定义

以此和进行区分。

另外对于集合中的某一个间隙，考虑间隙时长有

另外对于一个任务的截止时刻，可以如下表示：

### 任务模型

本文对于任务的到达和离开采用M/M/s/排队模型，其中可以代表处于调度范围内的服务器数量。

#### 任务到达

模拟单位时间内到达的任务数满足为参数的泊松分布，即单位时间内到达任务数为的概率是

即任务到达的时间间隔满足参数为的指数分布，即在上一个任务到达后时间内，有新任务到达，即相邻任务到达时刻的间隔的概率满足指数分布，概率密度函数为

对应的分布函数为

#### 任务等待

对于使用的数据集，我们设置对于每一个任务，有

满足截断的正态分布

其中是标准正态分布的分布函数。

#### 任务处理和离开

排队论中的“离开”指的是服务系统中的任务在完成服务后离开系统的过程。在本文使用的模型中，使用台服务器进行任务处理时，每台服务器单位时间内可以处理的任务数满足以为参数的泊松分布，那么对于单位时间内单台服务器处理任务数为的概率是

单个任务的处理时长满足的分布为

单位时间内所有服务器处理任务数为的概率是

另外，由于任务数据具有一定的长度，我们设置任务数据接收时长满足参数为的指数分布

#### 任务占用率

对于测试数据集，任务对服务器的占用率满足0至1间的均匀分布：

### 分布式任务分配网络模型

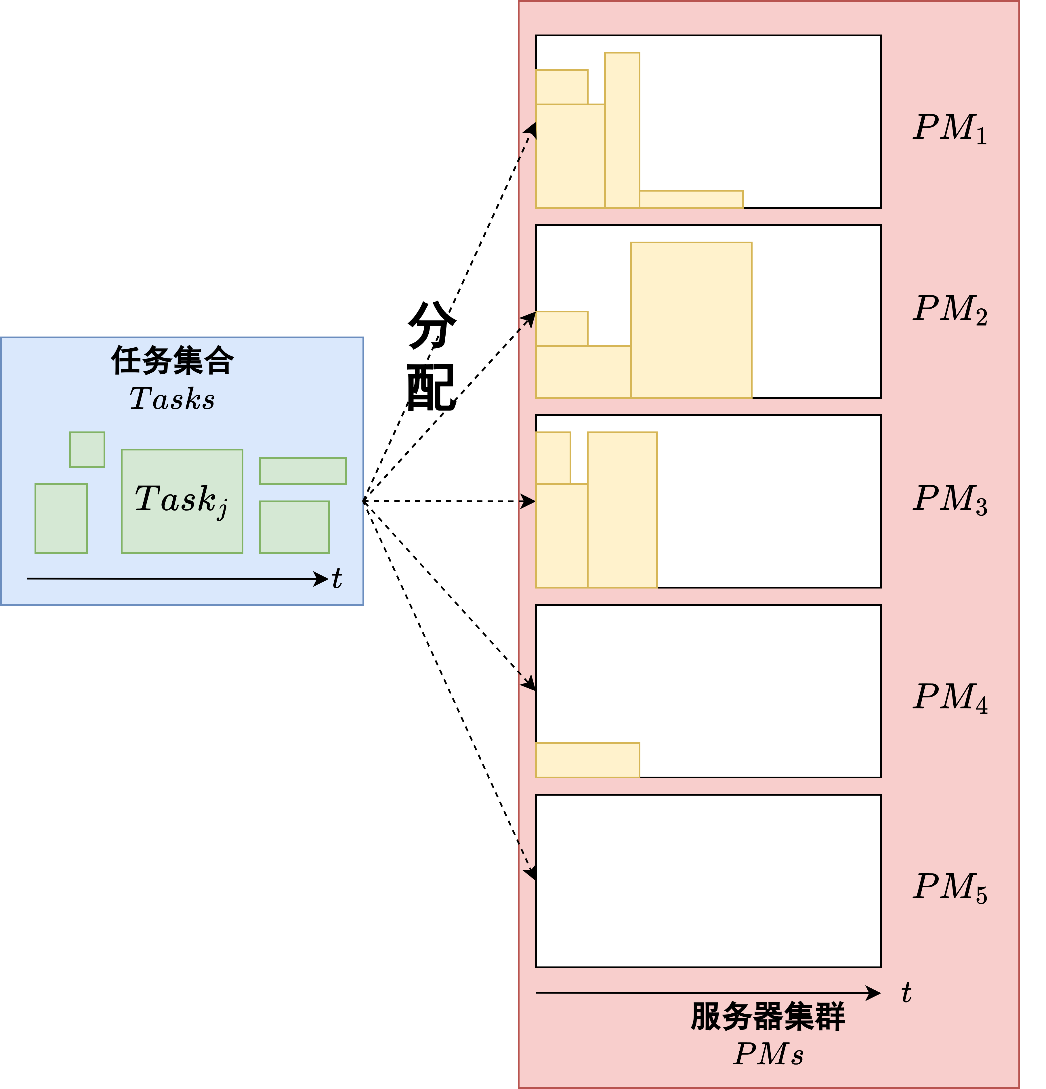


图3‑1 分布式任务分配网络模型

图3‑1为分布式任务分配网络模型。在该网络模型中，任务集合中的具有不同属性的任务被任务分配算法分配到服务器集群中的不同服务器上，并确认任务开始执行的时刻。被分配的任务在不同的服务器上占用不同时长、不同占用率大小的资源。

在任务集合中，每一个任务对每一台服务器的占用率不同，占用时长也不同。在任务集合到达服务器集群时，分配算法将任务分配给不同的服务器。

### 评价指标

在本研究中，在同一下，使用相同测试数据集对不同的策略进行测试。针对不同的采用的数据集总任务个数相同。

#### 服务质量

服务质量可以使用多种方式衡量，从数据发送到数据接收、数据在服务器中停留的时长等均可以作为服务质量进行评价。本研究用使用不同策略时使用相同数据集超时的任务数来衡量不同下不同策略的优劣。超时的任务数越多，则任务分配方法的服务质量越差；超时的任务数越少，则任务分配方法的服务质量越高。

#### 能耗

假设控制范围内、集群中每一台服务器是相同配置的，功率采用和计算资源占用率的线性模型：

其中指的是一台服务器最大功率，指的是一台服务器在工作状态下的最小功率，指的是服务器在时刻的计算资源占用率。当服务器占用率为0时，其处于休眠状态，功率则为工作状态下最小功率的倍。

### 仿真流程

#### 事件驱动仿真

三阶段法是离散事件仿真中的一种经典方法，用于模拟和分析复杂系统的行为。三阶段法将仿真过程分为三个主要阶段：初始化、事件调度和事件执行。在初始化阶段，设置仿真系统的初始状态，创建初始事件列表，并将仿真时钟设定为零或仿真开始的时间点。在事件调度阶段，维护一个按时间顺序排列的事件列表，并根据事件发生的时间推进仿真时钟。事件执行阶段则是当仿真时钟达到某个事件的预定时间时，执行该事件的处理程序，更新系统状态，并生成可能的新事件。这种方法通过事件驱动的方式推进仿真时间，避免了不必要的计算，提高了仿真的效率和精度。三阶段法流程如图 3‑2所示。

本文实验仿真策略使用三阶段法，仿真通过事件驱动的方式来模拟调度的过程。

B事件列表如表 3‑2：

表 3‑2 B事件列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | 类型 | 状态变化 | 规定的下一事件 |
| B1 | 到达 | 任务到达并进入等待队列 | B1 |
| B2 | 完成 | 任务完成，虚拟机减少负载 |  |

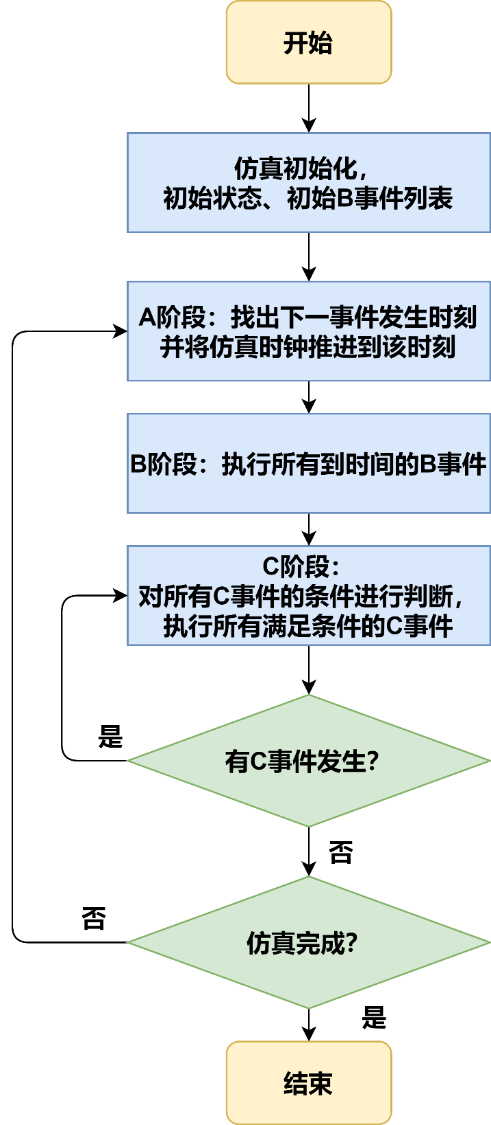


图 3‑2 三阶段法流程[[1]](#footnote-1)

C事件列表如表3‑3：

表3‑3 C事件列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | 类型 | 条件 | 状态变化 | 规定的下一事件 |
| C1 | 开始动作 | 有任务到达 | 检测是否存在可用间隙，  如果存在间隙满足要求则将任务从队列中取出，分配给服务器按照计划进行工作；  如果服务器不符条件则保持在等待队列中。 | B2 |

## 基于传统算法的改进

在对任务和计算资源进行优先级排序的时候，分别考虑2种计算资源间隙优先级排序和3种任务优先级排序。一共考虑到6种策略情况。

相比于MaxUtil考虑按照计算资源间隙选择优先级进行排序而缺乏对任务调度顺序的考虑，本章从任务角度考虑了两种任务优先调度策略，使用不同的方法对任务队列中的待处理任务进行优先级排序，以期让紧急的任务先进行处理。在第四章中本文还考虑了使用强化学习方法训练的模型将任务和计算资源进行匹配。如图 3‑3。

对于MaxUtil中使用的按照任务到达的时刻安排至间隙至间隙的策略，本文称之为先到先服务。相比于先到先服务，本章分别提出了先结束先服务和延迟确认这两种任务优先调度策略。在先到先服务和先结束先服务的策略中，任务到达即被安排进服务器中，在没有计算资源可供被安排的任务使用时，则将任务放入延迟列表，先到先服务的策略让延迟列表中到达时刻早的任务被优先调度，先结束先服务的策略让延迟列表中截止时刻早的任务被优先调度。而对于延迟确认策略，则还会考虑任务被安排间隙之后是否会立即开始。如果被安排开始的时刻和当前时刻不一致，则将任务放进延迟列表，进入下一轮。

### 计算资源间隙优先级排序

相比于随机的计算资源间隙选择优先级排序，MaxUtil提出了优化函数，对应一种新的计算资源间隙优先级选择排序（见表 3‑4）。

表 3‑4 MaxUtil计算资源间隙选择优先级排序

|  |  |  |
| --- | --- | --- |
| 优先级 | 字段 | 排序方式 |
| 1 | 间隙开始时刻 | 升序 |
| 2 | 间隙中最大占用率 | 降序 |

### 任务优先级排序

#### 先到先服务

在服务器全忙的情况下，有任务到达无法即时安排间隙（根据搜索的间隙范围，任务必定超时），即可将其录入进延迟列表。在每一次有任务结束运行时再次考虑安排间隙。此时按照任务到达的先后顺序来进行间隙上的安排。即每次取任务列表中最小的任务进行间隙安排。

#### 先结束先服务

在搜索的间隙范围服务器全忙的情况下，任务必定超时，此时可以将任务加入延迟列表。在每一次有任务结束运行或有新任务到达时再次考虑安排间隙。每次取任务列表中的任务，即截止事件最早的任务进行间隙安排。

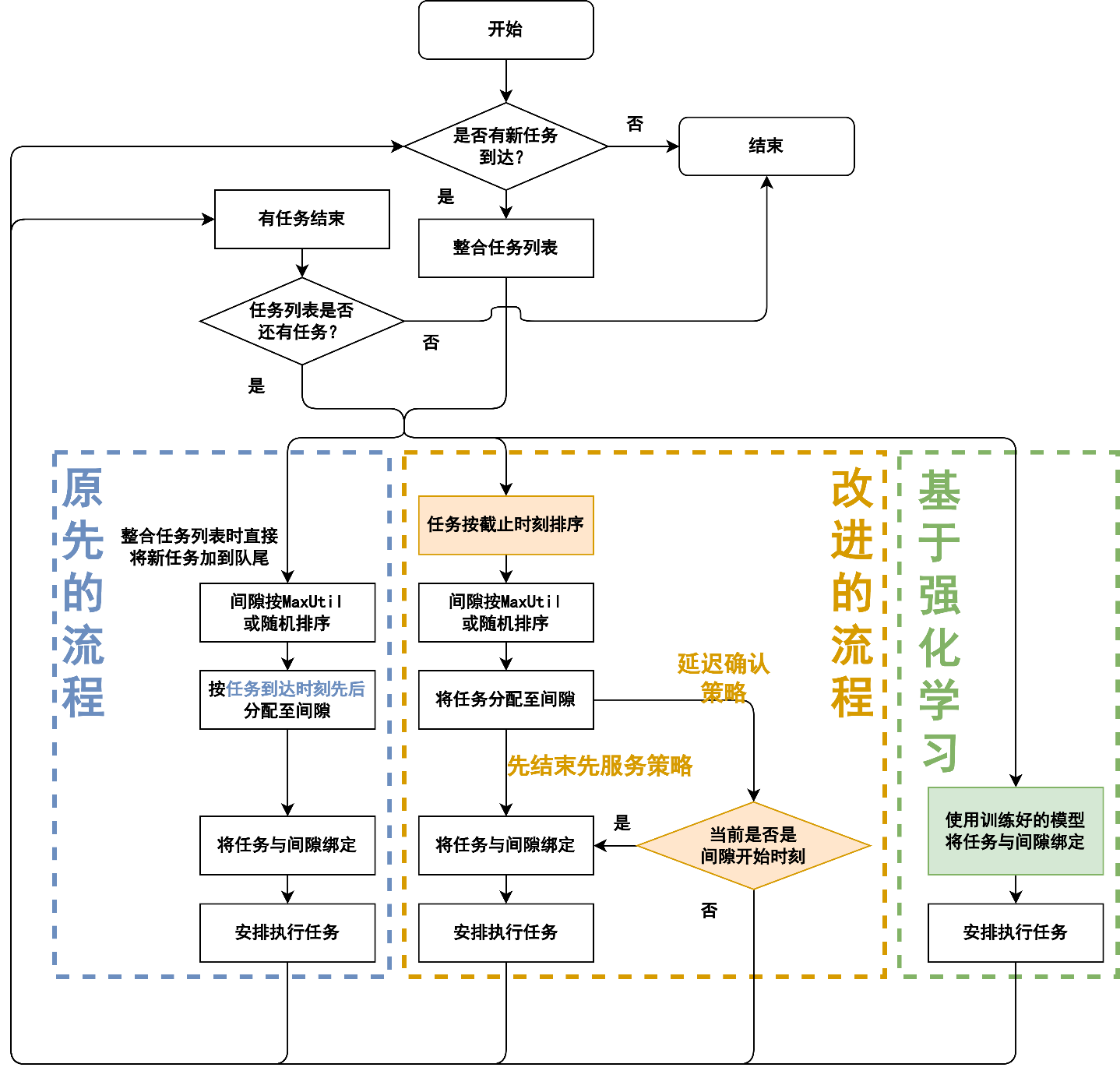


图 3‑3 在传统算法上的改进和基于强化学习的方法的流程

需要注意的是，先到先服务和先结束先服务指的是延迟任务和当前寻找间隙的任务共同组成的列表任务优先调度策略。如果任务不进入延迟列表而在任务到达时已经安排了间隙，其就已经确认占用间隙。而后来的任务即使可能早结束，如果先到达的任务没有进入延迟列表已经确认了间隙占用，后到达的任务无法改变间隙安排，需要寻找新的间隙。

#### 延迟确认

延迟确认的做法，即是在每次对一个任务（不只是即时到达的任务）选出合适的间隙的时候，如果选出的间隙开始时刻和当前时刻不等，那么将任务打入延迟列表，等待有任务结束运行或有新任务到达时综合新任务和延迟列表中的任务进行安排。也就是说，没有开始执行的任务均会被放在列表中。在每一次进行安排的时候，优先按照对任务进行升序排序并按顺序进行间隙安排。

此处不考虑升序排序是因为只会使得任务没有被及时处理，服务质量变差（超时任务可能变多）而能耗也并不会变好（因为均是按照先到先服务进行的，延迟考虑只会使得部分服务没有及时预约空隙导致其它服务重复预约空隙而最终没有得到空隙最终可能超时的情况。

## 性能对比

### 实验参数

本研究点所涉及的实验参数如表 3‑5所示。

表 3‑5 仿真实验使用参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数含义 | 参数 | 值 | 单位 |
| 单位时间内到达任务量满足泊松分布参数值 |  | 20,25,30,35,40,45,50 | 个 |
| 单位时间内单个服务器处理任务数  满足泊松分布参数值 |  | 12 | 个 |
| 接收时长满足指数分布参数值 |  | 20 | 个 |
| 任务等待时长满足的截断的正态分布参数值 |  | 2.0 | ms |
|  | 0.1 | ms2 |
| 实验中服务器集群中服务器数量 |  | 4 | 台 |
| 服务器集群中一台服务器最大功率 |  | 300 | W |
| 服务器集群中一台服务器在工作状态下的最小功率 |  | 200 | W |
| 服务器集群中一台服务器处于休眠状态下的功率和处于工作状态下最小功率的比 |  | 0.1 |  |
| 总任务样本量 |  | 10000 | 个 |

### 实验结果

#### 服务质量

表 3‑6和图 3‑4为仿真结果在服务质量上的表现，指的是使用泊松分布模型单位时间到达任务数。总任务样本量为10000个。其中蓝色背景为基线，加粗部分为改进部分。

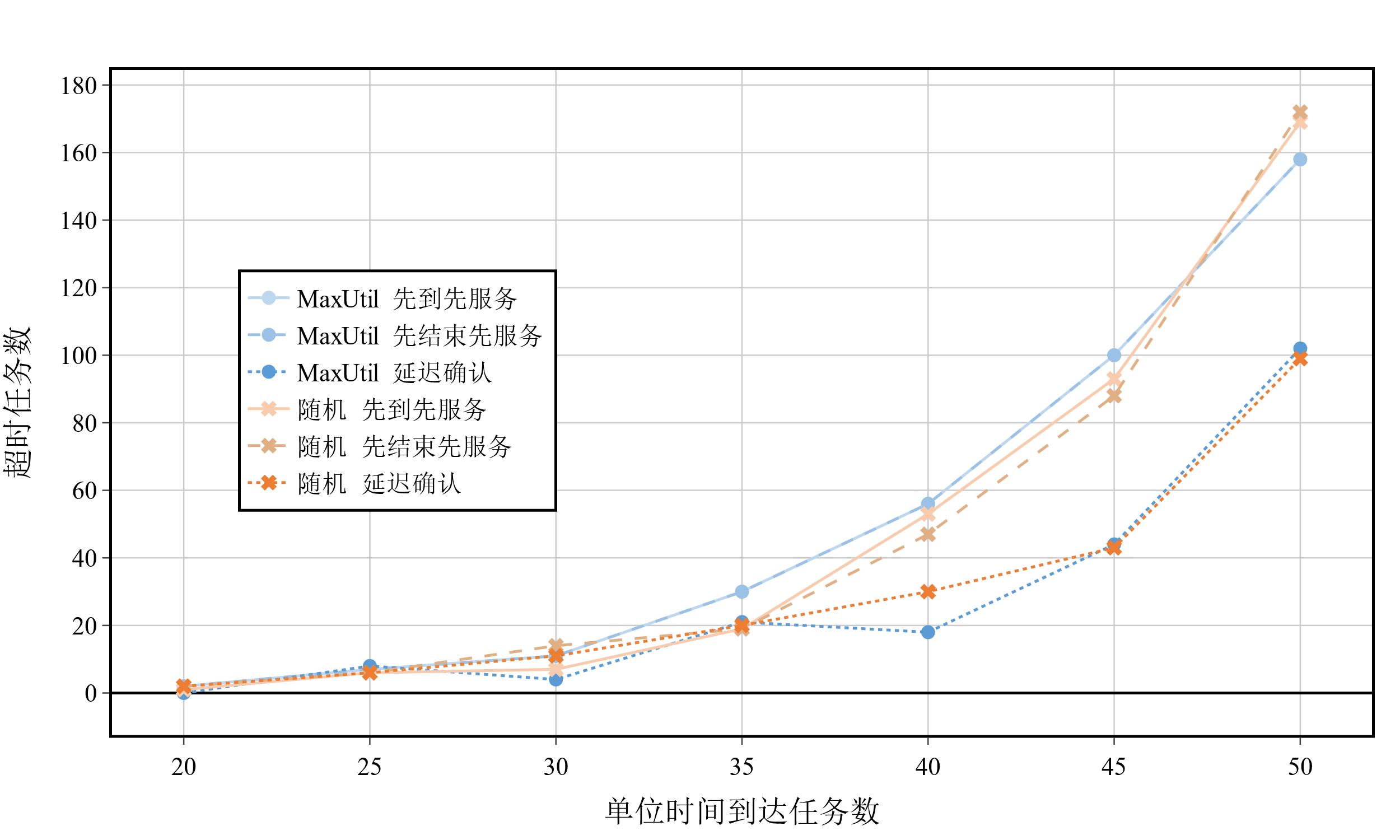
随着单位时间到达任务数的增加，在相同的任务优先调度策略中，除了延迟确认策略在单位时间到达的任务数较低的情况，超时任务数随着的增加而增加。

MaxUtil和随机任务优先调度策略分别在相同的情况下，采用先到先服务和先结束先服务的间隙优先调度策略时，超时任务数结果相近。

无论对于MaxUtil还是随机的任务优先调度策略，延迟确认的间隙优先调度策略在很低和较高的情况下，能够使得超时任务数相比于另外两种间隙优先选择调度策略少。这说明针对计算资源间隙的延迟确认调度策略是有必要的。

表 3‑6使用不同间隙选择策略和任务优先调度策略分别的超时任务数（单位：个）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 任务优先调度策略 |  | 间隙优先选择策略 | | |
| 先到先服务 | **先结束先服务** | **延迟确认** |
| MaxUtil | 20 | 2 | **2** | **0** |
| 25 | 7 | **7** | **8** |
| 30 | 11 | **11** | **4** |
| 35 | 30 | **30** | **21** |
| 40 | 56 | **56** | **18** |
| 45 | 100 | **100** | **44** |
| 50 | 158 | **158** | **102** |
| 随机 | 20 | 1 | **2** | **2** |
| 25 | 6 | **6** | **6** |
| 30 | 7 | **14** | **11** |
| 35 | 19 | **19** | **20** |
| 40 | 53 | **47** | **30** |
| 45 | 93 | **88** | **43** |
| 50 | 169 | **172** | **99** |

图 3‑4 不同单位时间到达任务数情况下超时任务数

#### 能耗

表 3‑7为仿真结果在能耗上的表现，指的是使用泊松分布模型单位时间到达任务数。总任务样本量为10000个。蓝色是基线，加粗为改进部分。

对于相同的，无论采用哪种延迟任务优先调度策略，在相同的间隙选择策略下，能耗大致相同，并且在能耗上MaxUtil优于随机任务选择算法。

随着的增加，在相同间隙选择策略和相同延迟任务优先调度策略的情况下，能量消耗减少，是因为本次仿真仿真10000个任务。随着的增大，任务到达更加密集的同时，统计的时间段缩短。在统计的时间段内，服务器有空闲的情况减少，因此能量消耗减少。

表 3‑7使用不同间隙选择策略和任务优先调度策略分别消耗能量（单位：J）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 任务优先调度策略 |  | 间隙优先选择策略 | | |
| 先到先服务 | **先结束先服务** | **延迟确认** |
| MaxUtil | 20 | 201.778 | **201.778** | **201.798** |
| 25 | 190.044 | **190.044** | **190.041** |
| 30 | 183.222 | **183.223** | **183.151** |
| 35 | 173.954 | **173.954** | **174.020** |
| 40 | 172.571 | **172.569** | **172.571** |
| 45 | 167.311 | **167.312** | **167.309** |
| 50 | 165.251 | **165.244** | **165.619** |
| 随机 | 20 | 218.646 | **218.612** | **218.161** |
| 25 | 205.641 | **205.932** | **204.374** |
| 30 | 198.011 | **197.496** | **197.473** |
| 35 | 188.149 | **188.225** | **186.992** |
| 40 | 184.108 | **183.564** | **182.939** |
| 45 | 177.416 | **177.276** | **176.807** |
| 50 | 174.330 | **173.975** | **172.939** |

### 性能分析与讨论

详见附录1。附录1以不同情况下到达的两个任务为例，主要分析了先到先服务和先结束先服务在MaxUtil算法中结果相近的可能的原因以及高峰时期延迟确认策略对MaxUtil算法服务质量的影响。以下是对这两个方面的简要描述。

对于以下分析所使用的图，图例如图3‑5所示。

#### 先到先服务和先结束先服务在MaxUtil算法中结果相近的可能的原因

为了便于分析，假设服务器在时刻只要有任务占用，其占用率均为100%。

如图3‑6，构造了两个任务和、服务器集，其中比先到达，但是的截止时刻早于，即

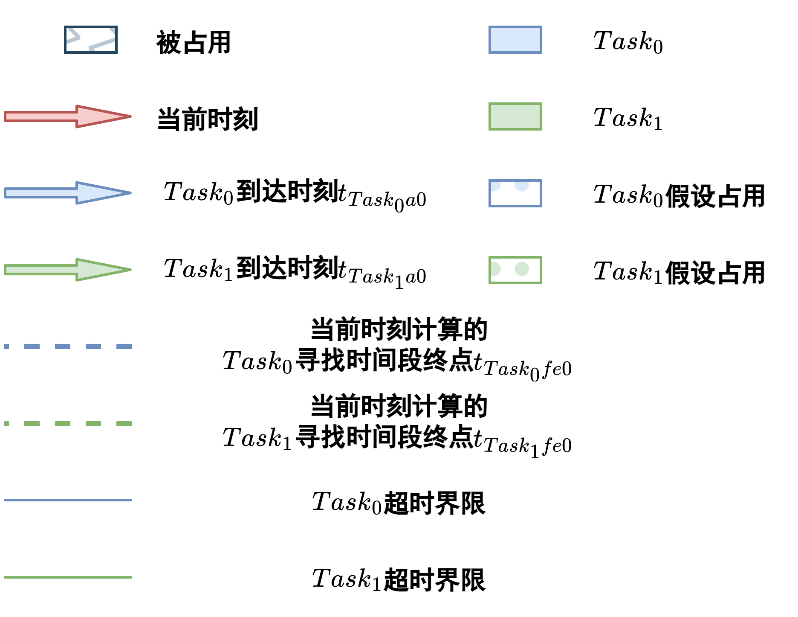


图3‑5 任务分配调度示意图图例

在选择先到先服务和先结束先服务的延迟任务优先调度策略的情况下，不论是超时任务数还是能耗，在MaxUtil算法中结果相近。

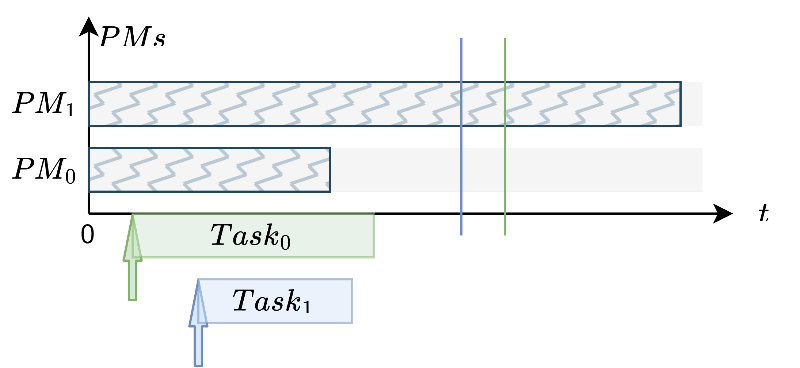


图3‑6 构造的两个任务和和不同的截止时间

任务到达较密集时，如表 3‑8，可以分为5种情况。

由图3‑8可以看出，5种简单情况下，不论是采用先到先服务的策略还是先结束先服务的策略，超时任务数相同，在MaxUtil算法中结果相近。这是由于在使用先到先服务和先结束先服务的延迟任务调度策略时，如果按照间隙选择找到合适的间隙，即会确定任务占用该间隙，这与延迟确认的做法不同。也正是因为找到间隙即确定间隙，在任务到达稀疏的情况下，服务器大部分空闲而几乎不会有任务进入延迟列表（并在下一次B事件发生时）和其它任务进行排序比较。

但是由于实际资源分配过程中可能多个任务按不同策略分配会相互影响，因此在某些较为特殊的情况下先到先服务和先结束先服务结果不一致。

#### 任务到达密集时延迟确认策略对MaxUtil算法服务质量的影响

任务到达密集时，如 可以分为5种情况。

对于高峰时期延迟确认策略对MaxUtil算法服务质量的影响，通过实验数据可以得出，延迟确认策略可以避免因为间隙已被安排导致任务出现超时的现象，服务质量较先到先服务的情况好。

表 3‑8 任务到达密集时针对先到先服务和先结束先服务情况讨论的情况列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 情况 | 结果是否超时 | | | | 情况图  编号 | 结果图  编号 |
| 先到先服务 | | 先结束先服务 | |
|  |  |  |  |
| 和均没有机会不超时 | 是 | 是 | 是 | 是 | 图 3‑7(a) | 图3‑8(a) |
| 没有机会不超时  而有机会不超时 | 是 | 否 | 是 | 否 | 图 3‑7(b) | 图3‑8(b) |
| 有机会不超时  而没有机会不超时 | 否 | 是 | 否 | 是 | 图 3‑7(c) | 图3‑8(c) |
| 和均有机会不超时  但不能全部不超时 | 否 | 是 | 否 | 是 | 图 3‑7(d) | 图3‑8(d) |
| 和均有机会不超时且可以全部不超时， | 否 | 是 | 否 | 是 | 图 3‑7(e) | 图3‑8(e) |
| 和均有机会不超时且可以全部不超时， | 否 | 否 | 否 | 否 | 图 3‑7(f) | 图3‑8(f) |

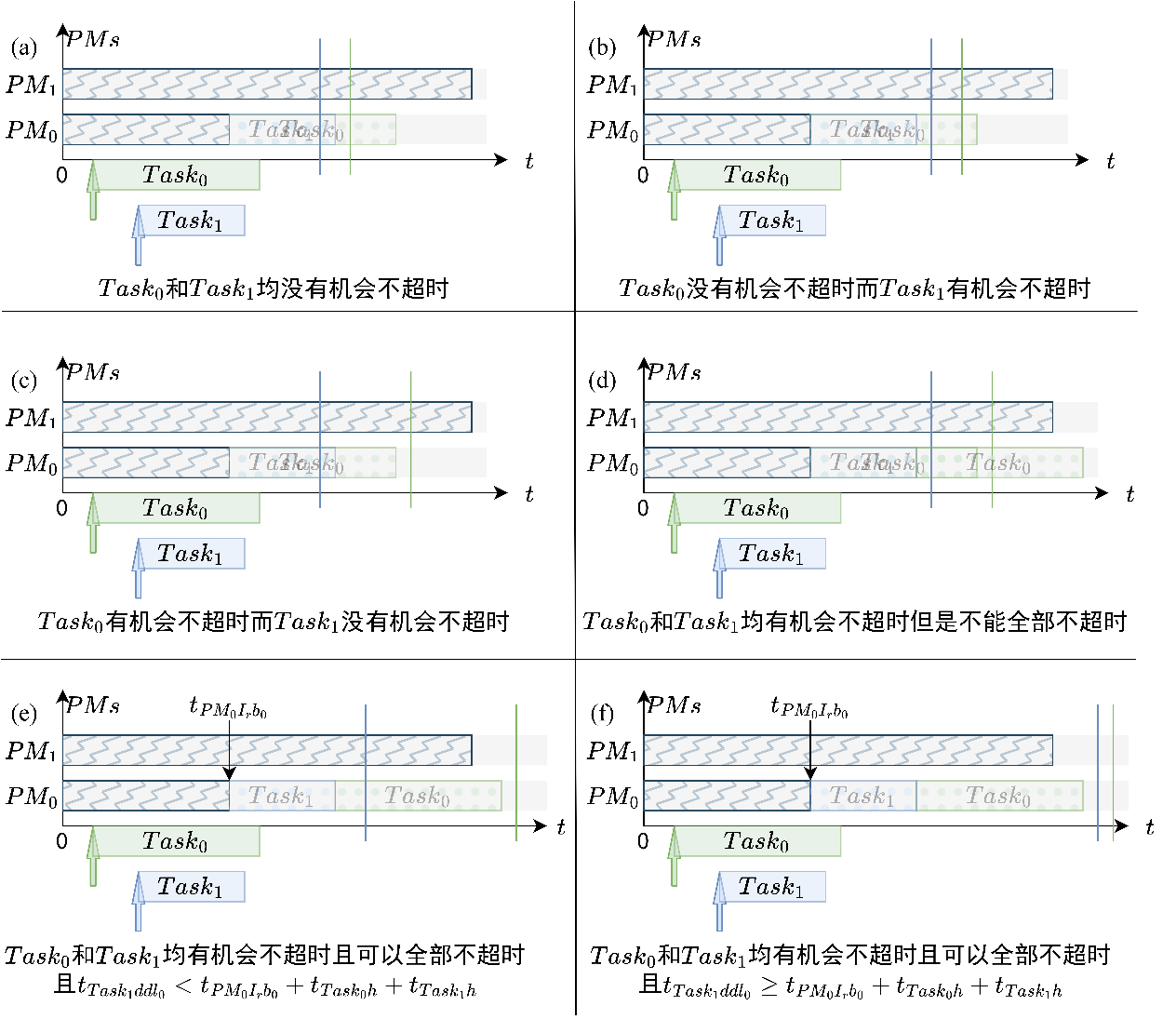


图 3‑7 任务到达密集时讨论的情况

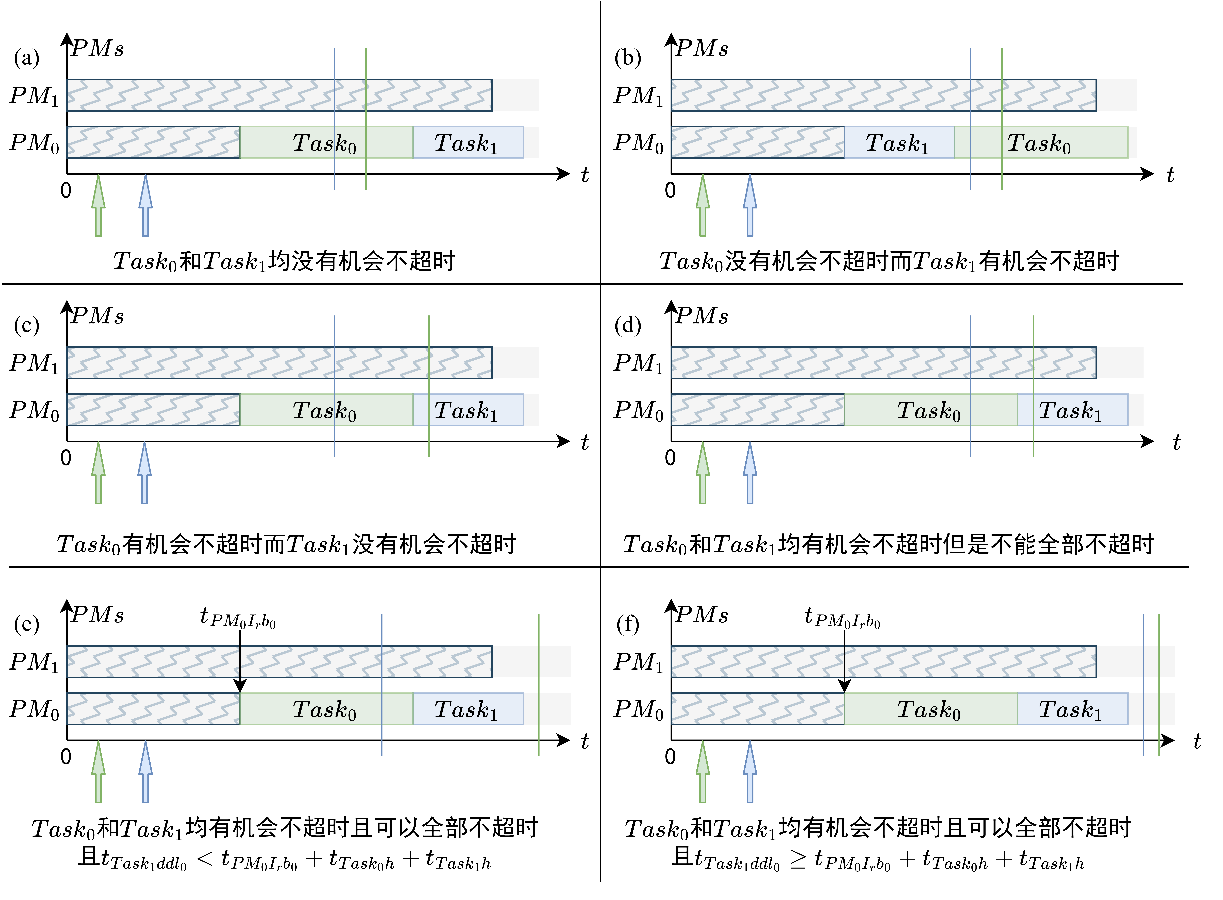


图3‑8 任务到达密集时讨论情况在使用先到先服务和先结束先服务策略时的结果

表 3‑9 任务到达密集时针对延迟确认策略的影响讨论的情况列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 情况 | 结果是否超时 | | | | 情况图  编号 | 结果图  编号 |
| 先到先服务 | | 延迟确认 | |
|  |  |  |  |
| 和均没有机会不超时 | 是 | 是 | 是 | 是 | 图 3‑7(a) | 图3‑9 (a) |
| 没有机会不超时  而有机会不超时 | 是 | 否 | 是 | 否 | 图 3‑7(b) | 图3‑9 (b) |
| 有机会不超时  而没有机会不超时 | 否 | 是 | 是 | 是 | 图 3‑7(c) | 图3‑9 (c) |
| 和均有机会不超时  但不能全部不超时 | 否 | 是 | 否 | 是 | 图 3‑7(d) | 图3‑9 (d) |
| 和均有机会不超时且可以全部不超时， | 否 | 是 | 否 | 否 | 图 3‑7(e) | 图3‑9 (e) |
| 和均有机会不超时且可以全部不超时， | 否 | 否 | 否 | 否 | 图 3‑7(f) | 图3‑9 (f) |

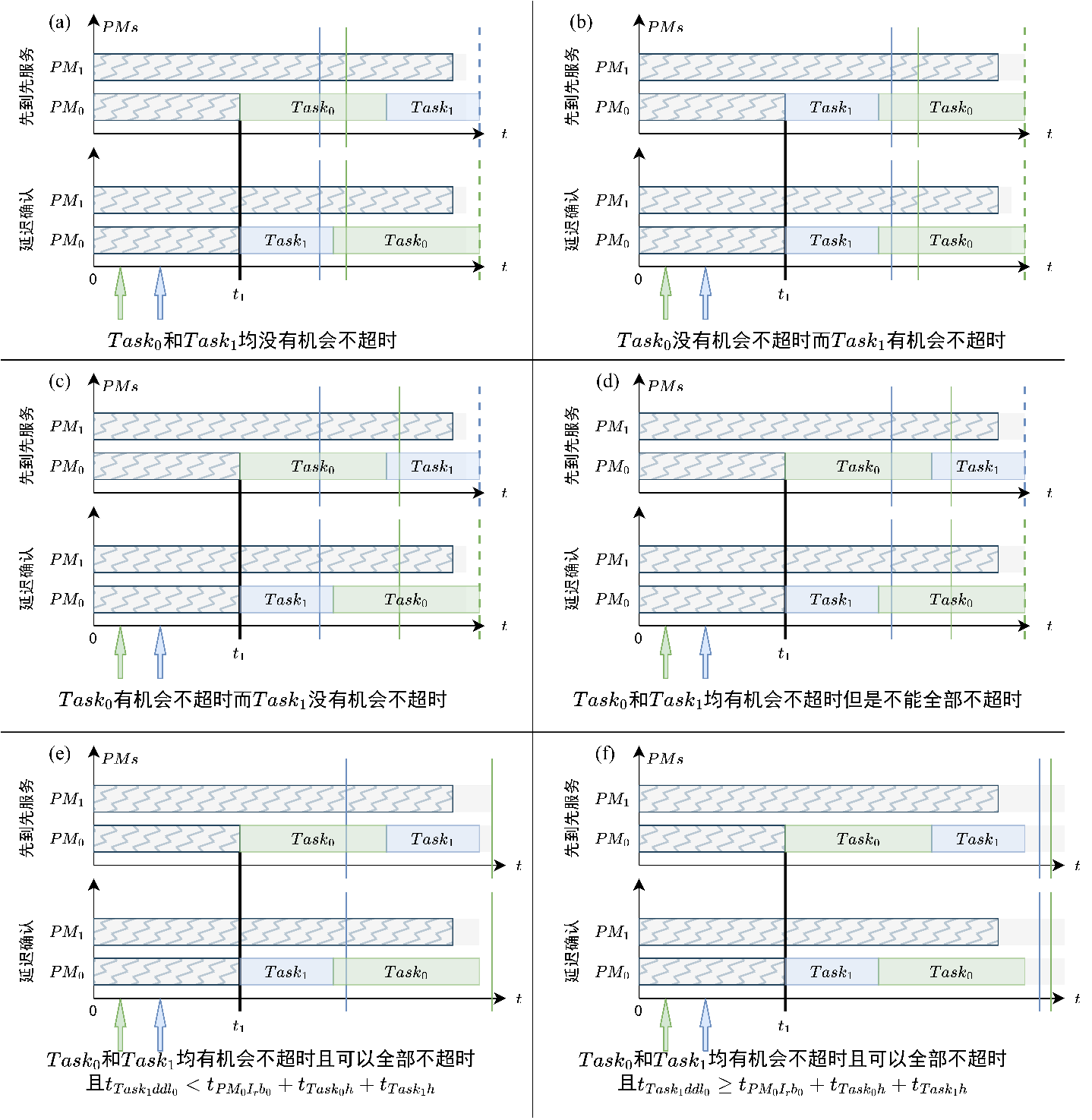


图3‑9 任务到达密集时在使用先到先服务和延迟确认策略时不同的结果

## 本章小结

本章主要探讨了基于传统任务分配方法的策略改进。首先介绍了MaxUtil算法的基本概念和优化目标，随后详细阐述了问题建模的过程，包括符号表示、任务模型、分布式任务分配网络模型、评价指标和仿真流程。在此基础上，提出了对传统算法的改进策略，特别关注了计算资源间隙优先级排序和任务优先级排序的多种策略情况。 通过实验参数的设置和实验结果的对比分析，本章展示了改进策略在服务质量上的提升，尤其是在使用延迟确认策略时，超时任务数有显著减少。然而，能耗方面的表现与随机任务选择算法相近，表明在能耗优化方面还有改进空间。 最后，本章还对性能对比结果进行了分析和讨论。

# 基于强化学习的任务分配方法

## 机器学习任务分配方法现状

目前，许多研究和应用已经将人工智能和边缘计算这两个热点领域结合起来，其动机大致可以分为两个方面：

•边缘计算的优化和部署需要人工智能算法的辅助；

•边缘计算为需要部署在终端设备附近的人工智能应用程序提供了必要的计算功能，以实现低延迟和高网络稳定性[67]。

近年来，人工智能算法也被广泛运用在任务分配计算资源方面，并且随着人工智能技术的发展，用于任务分配计算资源的算法也从传统的机器学习逐渐转变为深度学习、强化学习等更新的算法。

### 传统机器学习任务分配方法

传统的机器学习方法可以分为监督学习和无监督学习两大类。在监督学习中，模型根据有标签的训练数据进行训练，包括线性回归、逻辑回归、决策树、支持向量机等。这些方法通过学习输入特征与标签之间的关系，来预测未知数据的标签或值。而在无监督学习中，模型在没有标签的情况下从数据中学习其结构和模式，包括聚类和降维方法，如K均值聚类、层次聚类、主成分分析等。

相较于深度学习，传统机器学习方法在处理高维复杂数据时可能表现不佳，但在数据量较小、特征工程较为关键的场景下仍然具有重要作用，其相比于深度学习、强化学习等更加容易部署。由于边缘计算的分布式特性，可以根据每个边缘和终端设备的资源情况和任务需求，合理选择合适的算法，因此传统的机器学习算法也可以依靠这些优势在边缘计算中找到自己的位置[67]。例如[68]运用K均值和混合整数二次规划在工作负载均衡和通信时延方面取得了较好的效果。

### 深度学习任务分配方法

深度学习算法是由脑网络启发而来的。其可以包括很多层，包括卷积、池化、全连接层等，也可以在层间使用不同的激活函数，例如半波整流、反正切函数等。

深度学习的模型主要包括深度神经网络、卷积神经网络和循环神经网络等。其中循环神经网络常用来处理输入具有前后时序的应用场景。基于循环神经网络，[69]使用门的思想构建长短期记忆网络，使得网络效果在各应用中的效果进一步提升。

### 强化学习任务分配方法

和监督学习和无监督学习不一样，强化学习是通过和环境的交互来训练模型。其思想是智能体接收到环境状态，并根据历史经验采取行动，使得奖励最大化。在边缘计算的任务分配计算资源的过程中，因为输入的数据时序相关，这与根据历史经验采取行动、 使用马尔可夫过程的算法特点相匹配，因此在近年来强化学习被越来越多用于任务分配计算资源的任务。

近年来，强化学习中的典型算法有深度Q学习、近端策略优化方法等。[70] 提出了一个基于网络状态和任务特征（如边缘服务器和设备的计算能力、通信信道质量、资源利用率和服务的延迟需求等）的多任务深强化学习算法，用于作出资源分配决策。另外，[71–74]也使用了Q学习或其变种来进行针对各自的评价指标对边缘计算资源分配进行优化。

## 近端策略优化方法建模

本研究尝试使用近端策略优化算法方法训练任务分配方法，以服务质量为目标，使得尽可能多的任务在截止时刻前完成。

使用基于强化学习的任务分配方法时，符号表示、任务模型、任务分配网络模型、评价指标和仿真流程沿用3.2所述。

### 观察空间构建

考虑到计算资源和任务可以分别抽象成两类不同类型的节点，计算资源和任务之间的多多匹配可以看成是二分图之间的连接。

考虑到在工程实现方面，图类型空间当前在gymnasium库中无法设置每一个抽象成节点的任务或计算资源的特征属性，因此最终使用字典的形式来构建观察空间和动作空间。

#### 观察空间抽象

考虑将计算资源和任务的多多匹配抽象成二分图。

沿用3.2所述，计算资源间隙的集合为，任务的集合为，那么可以将抽象成节点簇的计算资源间隙节点集合表示为，任务节点集合为。其中，任意一个计算资源间隙节点，任意一个任务节点。计算资源和任务之间的连接可以看成是多多匹配，如图 4‑1所示。

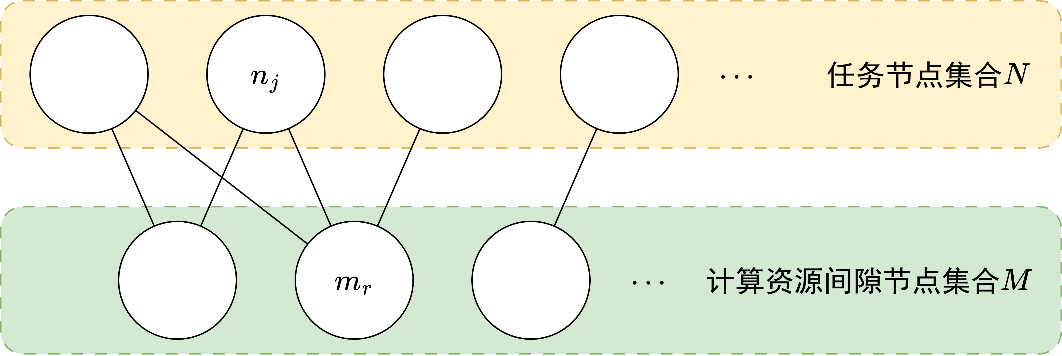


图 4‑1 计算资源间隙和任务多多匹配示意

考虑到一个计算资源间隙节点在匹配多个任务节点的时候，不同任务节点结束的时刻不同，而导致一个计算资源间隙节点，在间隙时间范围内，所有任务的占用率和在不同时刻不同。为了工程上的方便，可以考虑在对任务分配计算资源节点的时候将计算资源间隙节点进行拆分。

例如，如图 4‑2，在任务被安排在还有间隙、的服务器上时，需要将间隙进行占用率的扣除以及间隙的拆分，将间隙、拆分成、、、，那么对应的节点、也需要被拆分出、、、。考虑间隙拆分之后，每一个间隙不存在间隙的占用率随着时间变化而变化的情况。

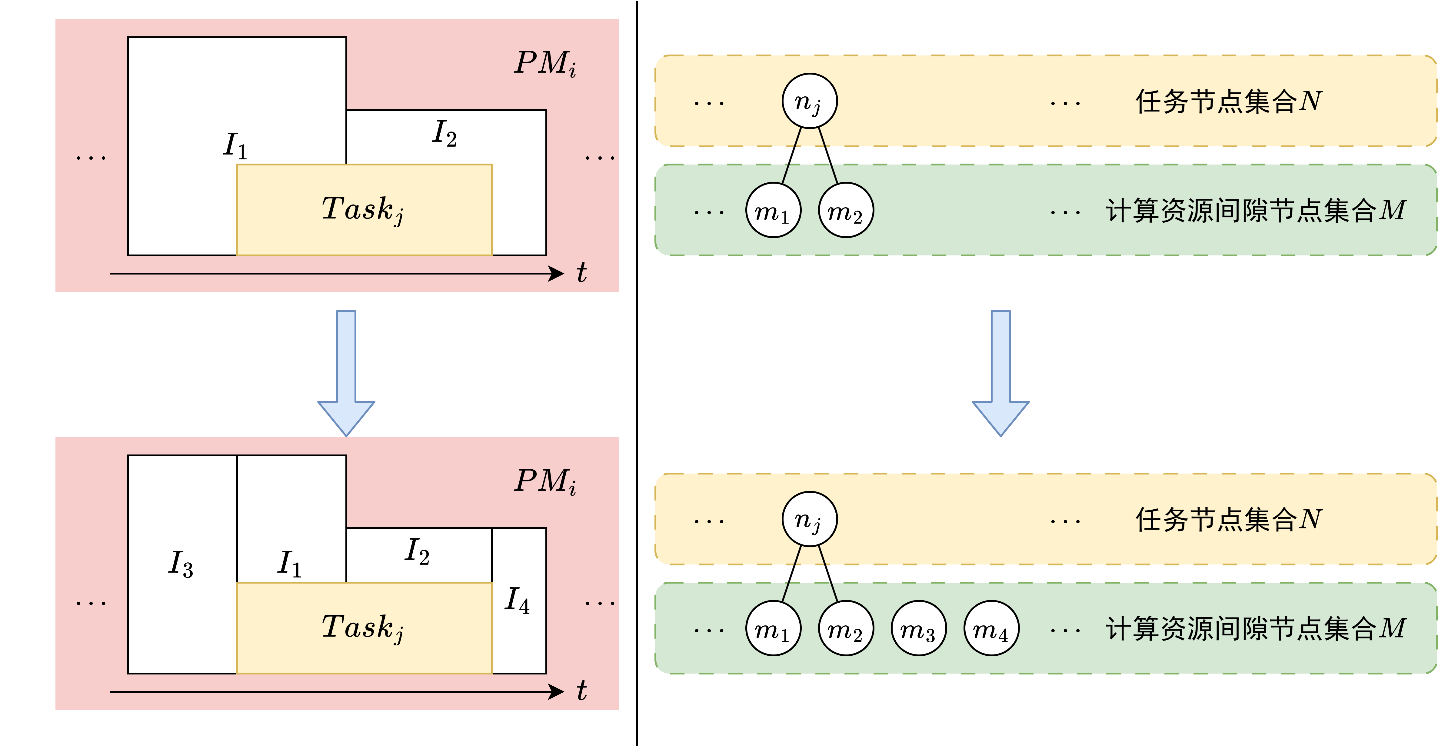


图 4‑2 计算资源间隙拆分和对应节点拆分示意

#### 仿真实现

gymnasium库可以实现观察空间中二分图的构建。但是由于gymnasium.spaces.Graph不支持内嵌容器类型的spaces且无法平展，因此使用字典嵌套gymnasium.spaces.Box的方式构建，具体如表 4‑1：

表 4‑1 观察空间构建[75]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | | | 空间类型 | 空间维度 | | 范围 | | 数据类型 |
| Node | intervals | index | Box | (间隙节点个数,) | | [0,节点个数] | np.int32 | |
| usage | Box | (间隙节点个数,) | | [0.0,1.0] | np.float32 | |
| start | Box | (间隙节点个数,) | | [0,) | np.float32 | |
| end | Box | (间隙节点个数,) | | [0,) | np.float32 | |
| PM | Box | (间隙节点个数,) | | [0,PM个数] | np.int32 | |
| tasks | index | Box | (任务节点个数,) | | [0,节点个数] | np.int32 | |
| usage | Box | (任务节点个数,) | | [0.0,1.0] | np.float32 | |
| arrival | Box | (任务节点个数,) | | [0,) | np.float32 | |
| duration | Box | (任务节点个数,) | | [0,) | np.float32 | |
| deadline | Box | (任务节点个数,) | | [0,) | np.float32 | |
| PM\_choice | Box | (任务节点个数,) | | [0,PM个数] | np.int32 | |
| Link |  |  | Sequence  嵌入  Multi-  Discrete | | 不定 | (任务节点个数,间隙节点个数) | np.int32 | |

需要注意使用gymnasium无法根据另外一个变量进行观察空间构建。例如无法限制duration小于deadline-arrival。限制需要通过在训练随机数据集的时候加入。

### 动作空间构建

和观察空间的构建方法相同，动作空间也使用字典这一容器类空间进行构建，如表 4‑2 。

表 4‑2 动作空间构建

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 空间类型 | 空间维度 | 范围 | 数据类型 |
| task\_index | Discrete | 任务节点个数 | [0,任务节点个数] | 默认整数 |
| PM\_choice | Discrete | 服务器节点个数 | [0,服务器节点个数] | 默认整数 |
| begin\_point | Box | 1 | [0,) | np.float32 |

对于gymnasium库，如果使用Discrete的空间类型，则空间维度默认是Discrete空间中可能出现的数值。

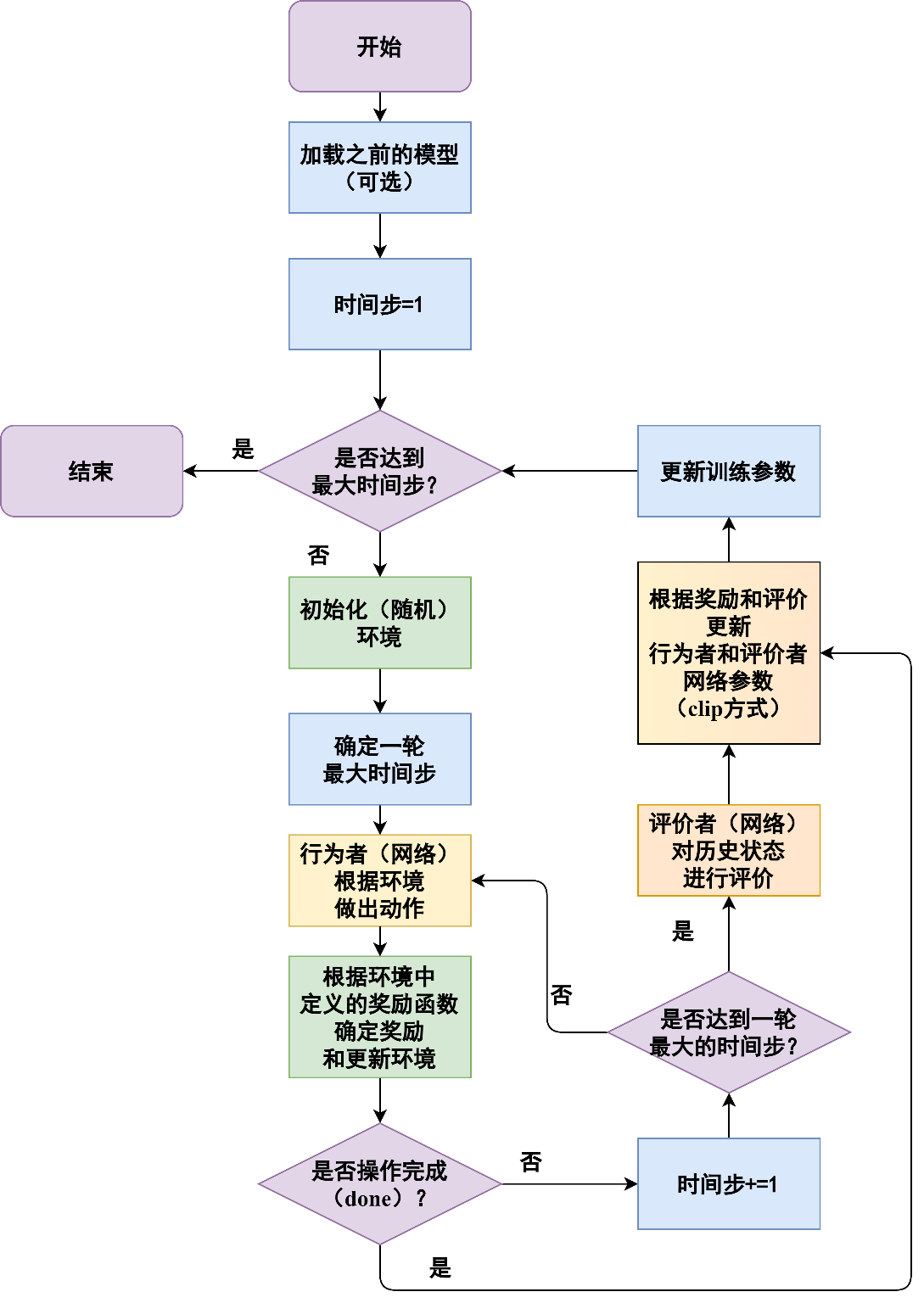


图 4‑3 近端策略优化用于任务分配训练流程

### 训练流程

如图 4‑3为训练流程。本研究采用近端梯度优化强化学习的训练方式，结合行为者-评价者的训练方法，对行为者网络进行训练，最终使用行为者网络确定任务被安排的计算资源。

在训练过程中，对随机环境时平均计算资源间隙数目、平均任务数、训练的梯度每隔一段频率就会发生变化。在更新参数时对梯度进行剪裁。

### 行为者网络

由于环境是不断变化的，输入层为行为者对环境的观察情况。其中，观察空间中计算资源间隙和任务个数在每一轮训练和使用中并不一致，且每一次输入的序列具有相关联的时间特征。因此可以考虑使用变长输入，如长短期记忆网络，也可以抽象到自然语言处理领域和图领域，考虑使用词向量嵌入、节点向量嵌入和注意力机制[76,77]。本研究采用补全输入、使用统一的输入维度的方式进行输入。输出定长，因此不需要另外考虑。

流程和采用的网络如图 4‑4所示。根据[78]推荐，层间的激活函数使用反正切函数[79]。根据输出变量为离散或连续，共用一步分相同的网络，同时分支使用不同的网络。

### 奖励和环境更新

针对模型在训练过程中可能出现的错误，选择使用奖励（惩罚）来进行限制。

奖励评价流程如图 4‑5。对应图 4‑3中所提到的“根据环境中定义的奖励函数确定奖励”部分。其中考虑到了任务节点不存在、服务器编号不存在、已经处理过相同任务节点、根据网络正向传播得到的开始时刻比到达时刻早、安排计算资源时某一时刻任务占用率超出计算资源空闲程度的出错情况，并根据不同的错误情况设置了不同的惩罚。

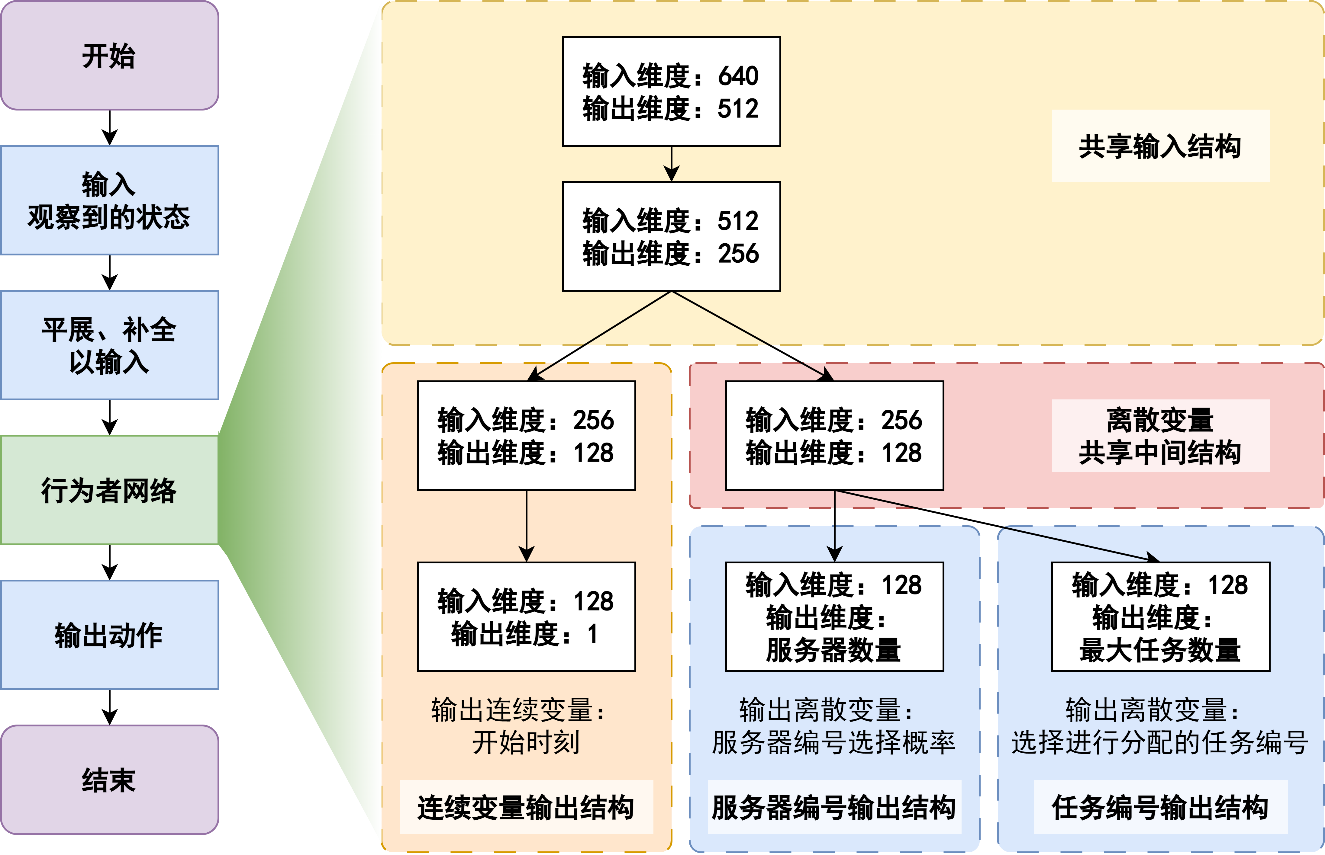
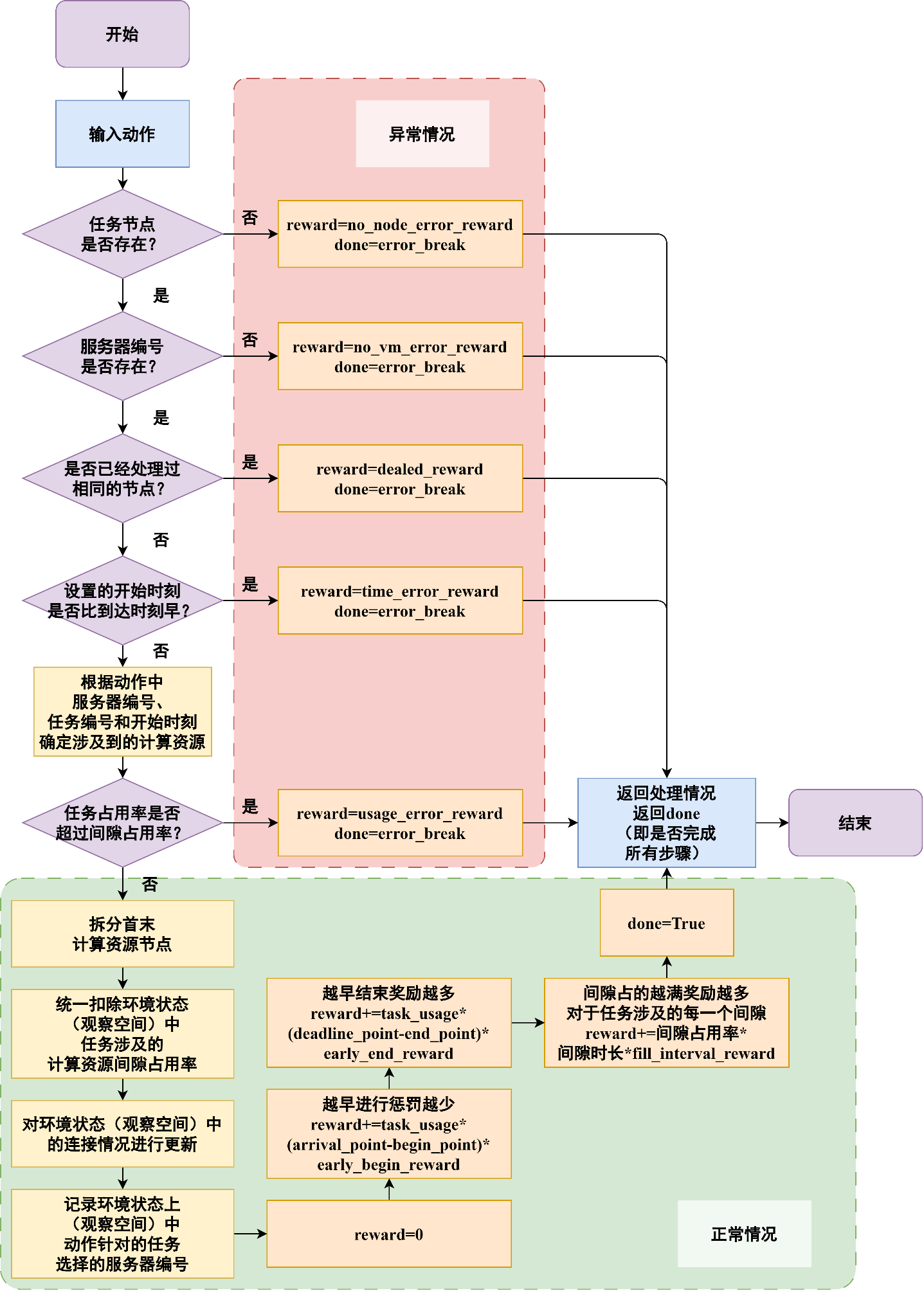


图 4‑4 行为者根据环境做出动作的步骤和行为者网络

图 4‑5 奖励流程

### 评价者网络

在考虑进行反向传播的时候，使用缓存的状态让评价者进行评价。流程和评价如图 4‑6图 4‑6所示。层间激活函数使用反正切函数。



图 4‑6 评价者评论流程和网络结构

### 模型运用

如图 3‑3为模型在任务分配时的运用方式。按照事件驱动的三阶段法对模型进行应用，其中包括数据类型的转换、环境的更新和日志的记录等。

## 性能对比

### 实验参数

#### 模型训练流程

奖励相关使用的参数表格如表 4‑3。

其中，error\_break用于决定训练过程中发生异常是否结束该轮训练。为了使得模型可以学习到更多，将error\_break设置为False。

另外，在训练过程中，由于部分间隙的末端时刻是正无穷，故使用大数10000来代替np.inf用于环境建模。

对于训练模型使用的网络结构，考虑可能输入的任务数量和间隙数量，将输入统一平展到640维。

表 4‑3 奖励相关参数

|  |  |  |  |
| --- | --- | --- | --- |
| 动作情况 | 解释 | 参数名称 | 参数值 |
| 异常 | 任务节点不存在 | no\_node\_error\_reward | -20.0 |
| 服务器编号不存在 | no\_vm\_error\_reward | -20.0 |
| 已经处理过相同的任务节点 | dealed\_reward | -30.0 |
| 根据网络正向传播得到的开始时刻  比到达时刻早 | time\_error\_reward | -30.0 |
| 安排计算资源时某一时刻任务占用率  超出计算资源空闲程度 | usage\_error\_reward | -20.0 |
| 是否因出错而结束这一轮 | error\_break | False |
| 正常 | 对任务设置较早开始的时刻 | early\_begin\_reward | 50.0 |
| 任务较早完成 | early\_end\_reward | 50.0 |
| 间隙占用率尽可能达到100% | fill\_interval\_reward | 30.0 |

对于训练时随机生成的初始观察环境，使用参数如表 4‑4所示。

表 4‑4 观察环境生成使用参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数含义 | 参数 | 参数生成  使用的概率分布 | 单位 |
| 单位时间内到达的任务数满足泊松分布、  任务到达时刻间隔满足指数分布参数 |  |  | 个 |
| 单位时间内一个服务器完成的任务数满足泊松分布、  任务处理时长满足指数分布参数 |  |  | 个 |
| 接收时长满足指数分布参数 |  |  | 个 |
| 任务等待时长满足的截断的正态分布参数 |  |  | ms |
|  |  | ms2 |

对于训练参数更新，如表 4‑5所示。

其中，在间隙和任务训练数据的生成中，每轮生成的计算资源间隙数量为随机生成的正整数，生成范围为。每轮生成的任务数量满足截断的泊松分布，泊松分布在最大任务数量处截断。

表 4‑5 训练参数

|  |  |  |  |
| --- | --- | --- | --- |
| 功能划分 | 功能 | 数值 | 单位 |
| 日志记录 | 频率 | 6000 | 时间步 |
| 打印 | 频率 | 600 | 时间步 |
| 优化策略更新和反向传播 | 单轮训练  计算和更新次数 | 10 | 次/轮 |
| clip策略参数 | 0.2 |  |
| 马尔可夫链奖励计算参数 | 0.99 |  |
| 输出动作的连续变量标准差 | 初始值 | 0.1 | ms |
| 更新频率 |  | 时间步 |
| 更新倍率 | 0.8 |  |
| 最小值 | 0.005 | ms |
| 保存检查点（模型） | 频率 |  | 时间步 |
| 学习率 | 初始值（行为者） |  |  |
| 初始值（频率家） |  |  |
| 更新频率 |  | 时间步 |
| 更新倍率（行为者） | 0.8 |  |
| 更新倍率（评价者） | 0.8 |  |
| 训练数据生成  （计算资源间隙） | 最大计算资源数量 | 29 | 个 |
| 更新频率 | 1.2 |  |
| 更新倍率（频率、最大数量） |  | 时间步 |
| 训练数据生成（任务） | 泊松分布参数初始值 | 2 | 个 |
| 泊松分布参数最大值 | 5 | 个 |
| 最大任务数量 | 20 | 个 |
| 更新频率 |  | 时间步 |
| 更新倍率（频率、泊松分布参数） | 1.05 |  |

#### 仿真使用参数

同3.4.1所述，如表 4‑6所示。

表 4‑6 仿真实验使用参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数含义 | 参数 | 值 | 单位 |
| 单位时间内到达任务量满足泊松分布参数值 |  | 20,25,30,35,40,45,50 | 个 |
| 单位时间内单个服务器处理任务数  满足泊松分布参数值 |  | 12 | 个 |
| 接收时长满足指数分布参数值 |  | 20 | 个 |
| 任务等待时长满足的截断的正态分布参数值 |  | 2.0 | ms |
|  | 0.1 | ms2 |
| 实验中服务器集群中服务器数量 |  | 4 | 台 |
| 服务器集群中一台服务器最大功率 |  | 300 | W |
| 服务器集群中一台服务器在工作状态下的最小功率 |  | 200 | W |
| 服务器集群中一台服务器处于休眠状态下的功率  和处于工作状态下最小功率的比 |  | 0.1 |  |
| 总任务样本量 |  | 10000 | 个 |

### 实验结果

#### 服务质量

因为模型训练有待提高，在服务质量方面，训练出的模型并没有取得比较好的效果。如图 4‑7。其中蓝色背景为基线，加粗部分为使用近端策略优化强化学习模型得到的结果。

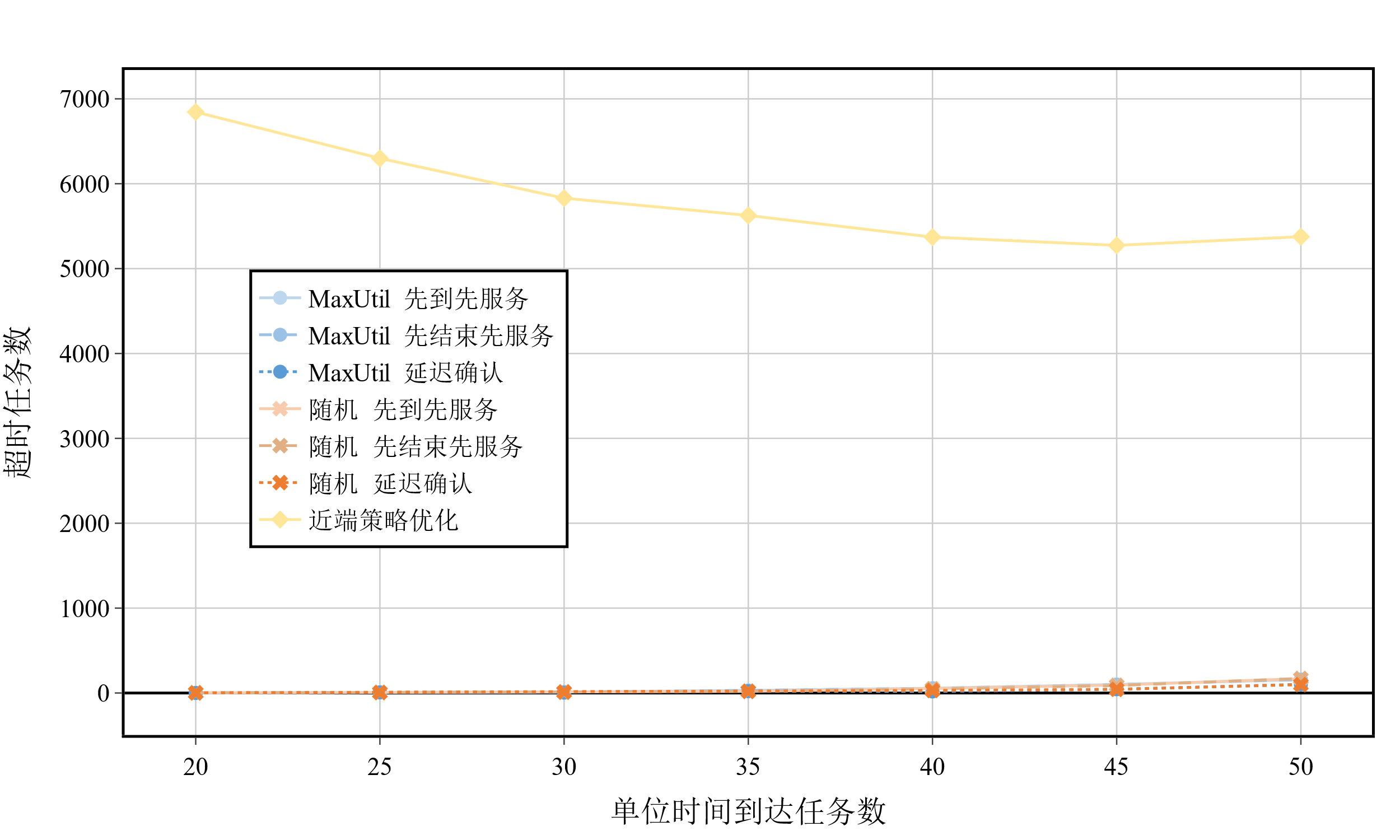


图 4‑7 使用近端策略优化算法训练的模型

在不同的单位时间到达任务数（）情况下超时任务数量的情况

可以看到使用该模型调度的时候超时任务数远超其它方法。这说明模型需要进一步改进。另外随着单位时间到达任务数增加，超时任务数下降，这或许说明模型更适合在单位时间到达任务数较多的情况下进行使用。

在测试过程中，可以看到模型在服务器上安排的任务开始时刻与任务到达时刻的差值经常比任务执行所需要的时长大一个数量级。说明对于模型对任务开始时刻的选择并没有进行有效的训练。

#### 能耗

如表4‑8所示。表中蓝色背景为基线，加粗部分为使用近端策略优化强化学习模型得到的结果。在相同的下，无论是选择任意一种间隙优先选择策略，使用近端策略优化强化学习模型并不会比使用MaxUtil提升较多的能耗。

表4‑7使用强化学习任务分配方法和MaxUtil分别的超时任务数（单位：个）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 任务优先调度策略 |  | 间隙优先选择策略 | | |
| 先到先服务 | 先结束先服务 | 延迟确认 |
| MaxUtil | 20 | 2 | 2 | 0 |
| 25 | 7 | 7 | 8 |
| 30 | 11 | 11 | 4 |
| 35 | 30 | 30 | 21 |
| 40 | 56 | 56 | 18 |
| 45 | 100 | 100 | 44 |
| 50 | 158 | 158 | 102 |
| 近端策略优化  强化学习模型 | 20 | **6845** | | |
| 25 | **6299** | | |
| 30 | **5830** | | |
| 35 | **5626** | | |
| 40 | **5371** | | |
| 45 | **5274** | | |
| 50 | **5376** | | |

### 本章小结

本章聚焦于基于强化学习的任务分配方法。本章首先回顾了机器学习任务分配方法的现状，包括传统机器学习、深度学习和强化学习方法在任务分配中的应用。随后，详细介绍了近端策略优化（Proximal Policy Optimization, PPO）方法在任务分配模型中的应用，包括观察空间和动作空间的构建、训练流程、行为者网络和评价者网络的设计，以及模型的实际运用。

在性能对比部分，本章展示了基于强化学习的任务分配方法与MaxUtil方法在服务质量和能耗方面的对比结果。尽管强化学习方法在某些情况下显示出了潜力，但在服务质量方面的表现并不稳定，尤其是在处理高负载任务时，超时任务数较多，需要进一步调整和优化奖励函数和模型结构。

表4‑8使用MaxUtil方法和使用近端梯度策略优化强化学习模型分别消耗能量（单位：）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 任务优先调度策略 |  | 间隙优先选择策略 | | |
| 先到先服务 | 先结束先服务 | 延迟确认 |
| MaxUtil | 20 | 201.778 | 201.778 | 201.798 |
| 25 | 190.044 | 190.044 | 190.041 |
| 30 | 183.222 | 183.223 | 183.151 |
| 35 | 173.954 | 173.954 | 174.020 |
| 40 | 172.571 | 172.569 | 172.571 |
| 45 | 167.311 | 167.312 | 167.309 |
| 50 | 165.251 | 165.244 | 165.619 |
| 近端策略优化  强化学习模型 | 20 | **215.318** | | |
| 25 | **201.977** | | |
| 30 | **194.680** | | |
| 35 | **183.540** | | |
| 40 | **180.503** | | |
| 45 | **174.205** | | |
| 50 | **170.443** | | |

# 总结与展望

## 论文工作总结

本文对任务在一个服务器集群中进行分配的应用场景进行了建模，考虑了基于传统任务分配方法的策略改进和一种基于强化学习的任务分配方法，并分别进行了实验。其中基于传统任务分配方法的策略改进在服务质量上相比于MaxUtil有所提高，但是基于强化学习的任务分配方法在服务质量上效果不佳，需要持续改进奖励函数和模型结构。

## 任务分配方法展望

对于本文中提出的两个策略，可以分别进行改进。

### 传统策略方面

可以考虑使用拍卖理论和匹配理论来改进任务分配方法，可以参考背包问题对问题进行解决。可以使用静态情境下的帕累托最优的解的分数来衡量任务分配方法。可以考虑其它方式的多个任务分组进行匹配[80]。

### 强化学习方面

既然可以将任务分配计算资源的过程建模为二分图两部节点之间的连接，那么就可以使用二部图相关的图机器学习算法，对任务分配的效果进行改进。

对于本研究中提出的基于强化学习的任务分配方法，考虑到输入数据在时间上成序列，可以考虑变长输入的神经网络层，例如长短期记忆网络和循环神经网络，也可以考虑使用注意力机制来进行维度统一。

考虑到Transformer在自然语言处理任务中有很强的效果，可以考虑将动作者的网络通过一定的适配，使用Transformer结构。

考虑到强化学习不只有近端梯度优化算法，还可以考虑深度Q学习、多臂老虎机等其它强化学习相关算法。

可以考虑将来实现调度和学习同时进行，使得算法能够在线运行。

### 场景扩展

本文只考虑了服务器集群中多台服务器的任务分配问题。在同时使用多个距离较远的服务器或服务器集群的时候，可以考虑在此基础上进行建模，并可以考虑可以运行在多处的分布式任务分配方法。

# 参考文献

[1] 中国5G发展和经济社会影响白皮书（2023年）[R/OL]. 中国信通院, 2023. http://www.caict.ac.cn/kxyj/qwfb/bps/202401/P020240326603524901232.pdf.

[2] 云计算白皮书（2023年）: 202303[R/OL]. 中国信息通信研究院, 2023. http://www.caict.ac.cn/kxyj/qwfb/bps/202307/P020240326634505750782.pdf.

[3] 周绪, 苗辉, 杨静, 等. 边缘计算资源调度:历史、架构、建模与方法分析[J/OL]. 计算机集成制造系统: 1-43. DOI:10.13196/j.cims.2023.0132.

[4] WANG S, ZHANG X, ZHANG Y, 等. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications[J/OL]. IEEEl Access, 2017, 5: 6757-6779.DOI:10.1109/ ACCESS.2017.2685434.

[5] GU H, ZHAO L, HAN Z, 等. AI-Enhanced Cloud-Edge-Terminal Collaborative Network: Survey, Applications, and Future Directions[J/OL]. IEEE Communications Surveys & Tutorials, 2023: 1-1. DOI:10/gtcmpc.

[6] (美)詹姆斯·F. 库罗斯(JAMES F. KUROSE),(美)基思·W. 罗斯(KEITH W. ROSS)著;陈鸣译., 库罗斯 (KUROSE, JAMES F.). 计算机网络: 自顶向下方法 = Computer networking[M]. Di 1 ban. Beijing: 机械工业出版社, 2018.

[7] YAO J, ANSARI N. Fog Resource Provisioning in Reliability-Aware IoT Networks[J/OL]. IEEE Internet of Things Journal, 2019, 6(5): 8262-8269. DOI:10.1109/JIOT.2019.2922585.

[8] YOUSEFPOUR A, PATIL A, ISHIGAKI G, 等. FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework[J/OL]. IEEE Internet of Things Journal, 2019, 6(3): 5080-5096. DOI:10.1109/JIOT.2019.2896311.

[9] YOU C, HUANG K, CHAE H, 等. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading[J/OL]. IEEE Transactions on Wireless Communications, 2017, 16(3): 1397-1411. DOI:10.1109/TWC.2016.2633522.

[10] GU S, LUO X, GUO D, 等. Joint Chain-Based Service Provisioning and Request Scheduling for Blockchain-Powered Edge Computing[J/OL]. IEEE Internet of Things Journal, 2021, 8(4): 2135-2149. DOI:10.1109/JIOT.2020.3026183.

[11] ZHANG Q, GUI L, HOU F, 等. Dynamic Task Offloading and Resource Allocation for Mobile-Edge Computing in Dense Cloud RAN[J/OL]. IEEE Internet of Things Journal, 2020, 7(4): 3282-3299. DOI:10.1109/JIOT.2020.2967502.

[12] ALAMEDDINE H A, SHARAFEDDINE S, SEBBAH S, 等. Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing[J/OL]. IEEE Journal on Selected Areas in Communications, 2019, 37(3): 668-682. DOI:10.1109/JSAC.2019.2894306.

[13] DONG X, DI Z, WANG L, 等. Load Balancing of Double Queues and Utility-Workload Tradeoff in Heterogeneous Mobile Edge Computing[J/OL]. IEEE Transactions on Wireless Communications, 2023, 22(7): 4313-4326. DOI:10.1109/TWC.2022.3224301.

[14] LUO Q, HU S, LI C, 等. Resource Scheduling in Edge Computing: A Survey[J/OL]. IEEE Communications Surveys & Tutorials, 2021, 23(4): 2131-2165. DOI:10/gpb3n2.

[15] LEE Y C, ZOMAYA A. Energy efficient utilization of resources in Cloud computing systems[J/OL]. The Journal of Supercomputing, 2010, 60: 268-280. DOI:10.1007/s11227-010-0421-3.

[16] HUANG T, LIN W, LI Y, 等. A Latency-Aware Multiple Data Replicas Placement Strategy for Fog Computing[J/OL]. Journal of Signal Processing Systems, 2019, 91(10): 1191-1204. DOI:10.1007/s11265-019-1444-5.

[17] KIANI A, ANSARI N, KHREISHAH A. Hierarchical Capacity Provisioning for Fog Computing[J/OL]. IEEE/ACM Transactions on Networking, 2019, 27(3): 962-971. DOI:10.1109/TNET.2019.2906638.

[18] Virtual network function placement and resource optimization in NFV and edge com-puting enabled networks[J/OL]. Computer Networks, 2019, 152: 12-24. DOI:10.1016/j.comnet. 2019.01.036.

[19] CANALI C, LANCELLOTTI R. GASP: Genetic Algorithms for Service Placement in Fog Computing Systems[J/OL]. Algorithms, 2019, 12(10): 201. DOI:10.3390/a12100201.

[20] GUO F, ZHANG H, JI H, 等. An Efficient Computation Offloading Management Scheme in the Densely Deployed Small Cell Networks With Mobile Edge Computing[J/OL]. IEEE/ACM Transactions on Networking, 2018, 26(6): 2651-2664. DOI:10.1109/TNET.2018.2873002.

[21] LIU H, LI Y, WANG S. Request Scheduling Combined With Load Balancing in Mobile-Edge Computing[J/OL]. IEEE Internet of Things Journal, 2022, 9(21): 20841-20852.DOI:10.1109/ JIOT.2022.3176631.

[22] XU X, LIU Q, LUO Y, 等. A computation offloading method over big data for IoT-enabled cloud-edge computing[J/OL]. Future Generation Computer Systems, 2019, 95: 522-533.DOI:10.1016/ j.future.2018.12.055.

[23] HUANG P Q, WANG Y, WANG K, 等. A Bilevel Optimization Approach for Joint Offloading Decision and Resource Allocation in Cooperative Mobile Edge Computing[J/OL]. IEEE Transactions on Cybernetics, 2020, 50(10): 4228-4241. DOI:10.1109/TCYB.2019.2916728.

[24] SAIF F A, LATIP R, HANAPI Z M, 等. Multi-Objective Grey Wolf Optimizer Algorithm for Task Scheduling in Cloud-Fog Computing[J/OL]. IEEE Access, 2023, 11: 20635-20646.DOI:10.1109/ ACCESS.2023.3241240.

[25] LIAO L, LAI Y, YANG F, 等. Online computation offloading with double reinforcement learning algorithm in mobile edge computing[J/OL]. Journal of Parallel and Distributed Computing, 2023, 171: 28-39. DOI:10.1016/j.jpdc.2022.09.006.

[26] LIU S, YU Y, LIAN X, 等. Dependent Task Scheduling and Offloading for Minimizing Deadline Violation Ratio in Mobile Edge Computing Networks[J/OL]. IEEE Journal on Selected Areas in Communications, 2023, 41(2): 538-554. DOI:10.1109/JSAC.2022.3233532.

[27] LI P, XIAO Z, WANG X, 等. EPtask: Deep Reinforcement Learning Based Energy-Efficient and Priority-Aware Task Scheduling for Dynamic Vehicular Edge Computing[J/OL]. IEEE Transactions on Intelligent Vehicles, 2024, 9(1): 1830-1846. DOI:10.1109/TIV.2023.3321679.

[28] FENG Y, HU S, CHEN L, 等. An intelligent scheduling framework for DNN task acceleration in heterogeneous edge networks[J/OL]. Computer Communications, 2023, 201: 91-101.DOI:10.1016/ j.comcom.2023.01.019.

[29] KONEČNÝ J, MCMAHAN B, RAMAGE D. Federated Optimization: Distributed Optimization Beyond the Datacenter[M/OL]. arXiv, 2015[2024-05-04]. http://arxiv.org/abs/1511.03575.DOI:10.48550/ arXiv.1511.03575.

[30] ZAW C W, PANDEY S R, KIM K, 等. Energy-Aware Resource Management for Federated Learning in Multi-Access Edge Computing Systems[J/OL]. IEEE Access, 2021, 9: 34938-34950. DOI:10.1109/ACCESS.2021.3055523.

[31] LIM W Y B, NG J S, XIONG Z, 等. Decentralized Edge Intelligence: A Dynamic Resource Allocation Framework for Hierarchical Federated Learning[J/OL]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(3): 536-550. DOI:10.1109/TPDS.2021.3096076.

[32] FENG J, ZHANG W, PEI Q, 等. Heterogeneous Computation and Resource Allocation for Wireless Powered Federated Edge Learning Systems[J/OL]. IEEE Transactions on Communications, 2022, 70(5): 3220-3233. DOI:10.1109/TCOMM.2022.3163439.

[33] LIU J, XU H, WANG L, 等. Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing[J/OL]. IEEE Transactions on Mobile Computing, 2023, 22(2): 674-690.DOI:10.1109/ TMC.2021.3096846.

[34] OSBORNE M J, OTHERS. An introduction to game theory: 卷 3[M]. Oxford university press New York, 2004.

[35] YANG L, JIANG H, SHI J, 等. Achieving Cooperative Mobile-Edge Computing Using Helper Scheduling[J/OL]. IEEE Transactions on Communications, 2023, 71(6): 3419-3436.DOI:10.1109/ TCOMM.2023.3262833.

[36] FAN W, HUA M, ZHANG Y, 等. Game-Based Task Offloading and Resource Allocation for Vehicular Edge Computing With Edge-Edge Cooperation[J/OL]. IEEE Transactions on Vehicular Technology, 2023, 72(6): 7857-7870. DOI:10.1109/TVT.2023.3241286.

[37] XU X, LIU K, DAI P, 等. Joint task offloading and resource optimization in NOMA-based vehicular edge computing: A game-theoretic DRL approach[J/OL]. Journal of Systems Architecture, 2023, 134: 102780. DOI:10.1016/j.sysarc.2022.102780.

[38] TENG H, LI Z, CAO K, 等. Game Theoretical Task Offloading for Profit Maximization in Mobile Edge Computing[J/OL]. IEEE Transactions on Mobile Computing, 2022: 1-1. DOI:10.1109/TMC.2022.3175218.

[39] Stable matching theory[Z/OL]//Wikipedia. (2024-03-16)[2024-05-03]. https://en.wikipedia. org/w/index.php?title=Stable\_matching\_theory&oldid=1214084998.

[40] WANG Y, WEI X, WANG H, 等. Distributed matching theory-based task re-allocating for heterogeneous multi-UAV edge computing[J/OL]. China Communications, 2024, 21(1): 260-278.DOI:10.23919/ JCC.fa.2022-0247.202401.

[41] TRAN-DANG H, KIM D S. DISCO: Distributed computation offloading framework for fog computing networks[J/OL]. Journal of Communications and Networks, 2023, 25(1): 121-131.DOI:10.23919/ JCN.2022.000058.

[42] PHAM Q V, LEANH T, TRAN N H, 等. Decentralized Computation Offloading and Resource Allocation for Mobile-Edge Computing: A Matching Game Approach[J/OL]. IEEE Access, 2018, 6: 75868-75885. DOI:10.1109/ACCESS.2018.2882800.

[43] CHITI F, FANTACCI R, PAGANELLI F, 等. Virtual Functions Placement With Time Constraints in Fog Computing: A Matching Theory Perspective[J/OL]. IEEE Transactions on Network and Service Management, 2019, 16(3): 980-989. DOI:10.1109/TNSM.2019.2918637.

[44] ZHOU W, FAN L, ZHOU F, 等. Priority-Aware Resource Scheduling for UAV-Mounted Mobile Edge Computing Networks[J/OL]. IEEE Transactions on Vehicular Technology, 2023, 72(7): 9682-9687. DOI:10.1109/TVT.2023.3247431.

[45] YANG Z, PAN C, WANG K, 等. Energy Efficient Resource Allocation in UAV-Enabled Mobile Edge Computing Networks[J/OL]. IEEE Transactions on Wireless Communications, 2019, 18(9): 4576-4589. DOI:10.1109/TWC.2019.2927313.

[46] LI M, CHENG N, GAO J, 等. Energy-Efficient UAV-Assisted Mobile Edge Computing: Resource Allocation and Trajectory Optimization[J/OL]. IEEE Transactions on Vehicular Technology, 2020, 69(3): 3424-3438. DOI:10.1109/TVT.2020.2968343.

[47] WANG H, XIE J, MUSLAM M M A. FAIR: Towards Impartial Resource Allocation for Intelligent Vehicles With Automotive Edge Computing[J/OL]. IEEE Transactions on Intelligent Vehicles, 2023, 8(2): 1971-1982. DOI:10.1109/TIV.2023.3234888.

[48] ZHANG J, GUO H, LIU J, 等. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution[J/OL]. IEEE Transactions on Vehicular Technology, 2020, 69(2): 2092-2104.DOI:10.1109/ TVT.2019.2959410.

[49] LUO Q, LI C, LUAN T H, 等. Self-Learning Based Computation Offloading for Internet of Vehicles: Model and Algorithm[J/OL]. IEEE Transactions on Wireless Communications, 2021, 20(9): 5913-5925. DOI:10.1109/TWC.2021.3071248.

[50] LIN J, YANG P, WU W, 等. Edge Learning for Low-Latency Video Analytics: Query Scheduling and Resource Allocation[C/OL]//2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS). 2021: 252-259[2024-05-05]. https:// ieeexplore.ieee.org/abstract/document/9637817. DOI:10.1109/MASS52906.2021.00041.

[51] HUNG C C, ANANTHANARAYANAN G, BODIK P, 等. VideoEdge: Processing Camera Streams using Hierarchical Clusters[C/OL]//2018 IEEE/ACM Symposium on Edge Computing (SEC). 2018: 115-131[2024-05-05]. https://ieeexplore.ieee.org/document/8567661. DOI:10.1109/SEC.2018.00016.

[52] LIU J, ZHANG Q. Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality[J/OL]. IEEE Access, 2019, 7: 11222-11236. DOI:10.1109/ACCESS. 2019.2891113.

[53] GU L, ZENG D, GUO S, 等. Cost Efficient Resource Management in Fog Computing Supported Medical Cyber-Physical System[J/OL]. IEEE Transactions on Emerging Topics in Computing, 2017, 5(1): 108-119. DOI:10.1109/TETC.2015.2508382.

[54] ZHOU J, DAI H N, WANG H. Lightweight Convolution Neural Networks for Mobile Edge Computing in Transportation Cyber Physical Systems[J/OL]. ACM Transactions on Intelligent Systems and Technology, 2019, 10(6): 67:1-67:20. DOI:10.1145/3339308.

[55] SOOD S K, MAHAJAN I. Fog-cloud based cyber-physical system for distinguishing, detecting and preventing mosquito borne diseases[J/OL]. Future Generation Computer Systems, 2018, 88: 764-775. DOI:10.1016/j.future.2018.01.008.

[56] CHEN X, ZHOU Y, HE B, 等. Energy-efficiency Fog Computing Resource Allocation in Cyber Physical Internet of Things Systems[J/OL]. IET Communications, 2019, 13. DOI:10.1049/iet-com.2019.0121.

[57] MA H, HUANG P, ZHOU Z, 等. GreenEdge: Joint Green Energy Scheduling and Dynamic Task Offloading in Multi-Tier Edge Computing Systems[J/OL]. IEEE Transactions on Vehicular Technology, 2022, 71(4): 4322-4335. DOI:10.1109/TVT.2022.3147027.

[58] LI P, XIAO Z, WANG X, 等. EPtask: Deep Reinforcement Learning Based Energy-Efficient and Priority-Aware Task Scheduling for Dynamic Vehicular Edge Computing[J/OL]. IEEE Transactions on Intelligent Vehicles, 2024, 9(1): 1830-1846. DOI:10.1109/TIV.2023.3321679.

[59] ZHANG H, WANG J, ZHANG H, 等. Security computing resource allocation based on deep reinforcement learning in serverless multi-cloud edge computing[J/OL]. Future Generation Computer Systems, 2024, 151: 152-161. DOI:10.1016/j.future.2023.09.016.

[60] JU Y, CHEN Y, CAO Z, 等. Joint Secure Offloading and Resource Allocation for Vehicular Edge Computing Network: A Multi-Agent Deep Reinforcement Learning Approach[J/OL]. IEEE Transactions on Intelligent Transportation Systems, 2023, 24(5): 5555-5569. DOI:10.1109/TITS.2023.3242997.

[61] WANG Y, ZHENG Y, LIU J. Secure Task Offloading and Resource Scheduling in Maritime Edge Computing Systems[C/OL]//2023 IEEE/CIC International Conference on Communications in China (ICCC). 2023: 1-6[2024-05-05]. https://ieeexplore.ieee.org/ abstract/document/10233531.DOI:10.1109/ ICCC57788.2023.10233531.

[62] SHI Y, YANG Y, YI C, 等. Towards Online Reliability-Enhanced Microservice Deployment With Layer Sharing in Edge Computing[J/OL]. IEEE Internet of Things Journal, 2024: 1-1. DOI:10.1109/JIOT.2024.3385816.

[63] PENG K, YANG Y, WANG S, 等. Reliability-Aware Proactive Offloading in Mobile Edge Computing Using Stackelberg Game Approach[J/OL]. IEEE Internet of Things Journal, 2024, 11(9): 16660-16671. DOI:10.1109/JIOT.2024.3354700.

[64] MA H, LI R, ZHANG X, 等. Reliability-Aware Online Scheduling for DNN Inference Tasks in Mobile-Edge Computing[J/OL]. IEEE Internet of Things Journal, 2023, 10(13): 11453-11464. DOI:10.1109/JIOT.2023.3243266.

[65] KUHN H W. The Hungarian method for the assignment problem[J/OL]. Naval Research Logistics Quarterly, 1955, 2(1-2): 83-97. DOI:10.1002/nav.3800020109.

[66] ROTH A E. Deferred Acceptance Algorithms: History, Theory, Practice, and Open Questions[J].

[67] HUA H, LI Y, WANG T, 等. Edge Computing with Artificial Intelligence: A Machine Learning Perspective[J/OL]. ACM Computing Surveys, 2023, 55(9): 184:1-184:35. DOI:10.1145/3555802.

[68] GUO Y, WANG S, ZHOU A, 等. User allocation-aware edge cloud placement in mobile edge computing[J/OL]. Software: Practice and Experience, 2020, 50(5): 489-502. DOI:10.1002/spe.2685.

[69] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory[J/OL]. Neural Computation, 1997, 9(8): 1735-1780. DOI:10.1162/neco.1997.9.8.1735.

[70] CHEN J, CHEN S, WANG Q, 等. iRAF: A Deep Reinforcement Learning Approach for Collaborative Mobile Edge Computing IoT Networks[J/OL]. IEEE Internet of Things Journal, 2019, 6(4): 7011-7024. DOI:10.1109/JIOT.2019.2913162.

[71] LI J, GAO H, LV T, 等. Deep reinforcement learning based computation offloading and resource allocation for MEC[C/OL]//2018 IEEE Wireless Communications and Networking Conference (WCNC). Barcelona: IEEE, 2018: 1-6[2024-05-10]. https://ieeexplore.ieee.org/document/8377343/.DOI:10.1109/ WCNC.2018.8377343.

[72] MIN M, XIAO L, CHEN Y, 等. Learning-Based Computation Offloading for IoT Devices With Energy Harvesting[J/OL]. IEEE Transactions on Vehicular Technology, 2019, 68(2): 1930-1941. DOI:10.1109/TVT.2018.2890685.

[73] CHEN X, ZHANG H, WU C, 等. Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning[J/OL]. IEEE Internet of Things Journal, 2019, 6(3): 4005-4018. DOI:10.1109/JIOT.2018.2876279.

[74] YANG T, HU Y, GURSOY M C, 等. Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks[C/OL]//2018 15th International Symposium on Wireless Communication Systems (ISWCS). Lisbon: IEEE, 2018: 1-5[2024-05-10]. https://ieeexplore.ieee.org/document/8491089/. DOI:10.1109/ISWCS.2018.8491089.

[75] Gymnasium Documentation[EB/OL]. [2024-05-08]. https://gymnasium.farama.org/api/spaces/ fundamental.html.

[76] GAO M, CHEN L, HE X, 等. BiNE: Bipartite Network Embedding[C/OL]//The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. New York, NY, USA: Association for Computing Machinery, 2018: 715-724[2024-03-28]. https://dl.acm.org/doi/10.1145/3209978.3209987. DOI:10.1145/3209978.3209987.

[77] HUANG W, LI Y, FANG Y, 等. BiANE: Bipartite Attributed Network Embedding[C/OL]// Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: Association for Computing Machinery, 2020: 149-158[2024-03-31]. https://dl.acm.org/doi/10.1145/3397271.3401068. DOI:10.1145/ 3397271.3401068.

[78] ENGSTROM L, ILYAS A, SANTURKAR S, 等. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO[M/OL]. arXiv, 2020[2024-05-09]. http://arxiv.org/abs/2005.12729. DOI:10.48550/arXiv.2005.12729.

[79] 影响PPO算法性能的10个关键技巧（附PPO算法简洁Pytorch实现）[EB/OL]//知乎专栏. [2024-05-09]. https://zhuanlan.zhihu.com/p/512327050.

[80] WANG Y, TONG Y, LONG C, 等. Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach[C/OL]//2019 IEEE 35th International Conference on Data Engineering (ICDE). 2019: 1478-1489[2024-04-03]. https://ieeexplore.ieee.org/document/ 8731455. DOI:10.1109/ICDE.2019.00133.

# 致 谢

感谢王妮娜老师、邹国学学长在论文编写方面提供的建议和帮助！

感谢邹文浩学长、邹国学学长在强化学习训练方面提供的指导！

感谢吴思凡学姐，为此文传统算法改进提供了参照的案例！

虽然说实验结果并不是很好，但是依然感谢我的舍友——谭志宇、黄自锴、韩泽杰、莫繁滨、王鲁顺、王湘天穆、周子萱，在我在床上桌捧着电脑看着挂着的显示器炼丹精神接近崩溃时，提供心理上的支持！

感谢一起跑步的郑少飞学长、一起参加定向的向致远学长，让我在本科期间能够保持一个比较好的身体素质。希望郑少飞学长能够顺利毕业，找到合适的工作！

我还要感谢我的父母，在我长达6个月、漫长而艰辛的推免求学阶段陪伴着我，帮我渡过了最难受的第三学期，并提供了丰富的经验和建议！

感谢李珂老师作为我的班主任，在科研工作中还要忙碌行政事务，辛苦了！

感谢徐晓慧、刘江、李珂、望育梅、张彬等各位老师在我往夏令营投递时的推荐！

感谢严林学长（大青蛙）、林志同学让我养成了编码的工程习惯，让我学会了如何更加优雅地编程！

感谢为人类生产力发展贡献一份力量的无数开发者，让我的生活更加便利，让我的生产效率得到提高。

另外，我还想要感谢我的高中同学。彭康桂为我提供了感情上的建议，周子涵在我做物理竞赛经常没有闲暇吃饭的时候，帮我带了饭。

我还想要感谢我在大学四年里认识的每一位老师和同学，在两千多公里往返学校的路上、在日常的生活中遇见的每一个人。是大家让我认识到了世界的不同，我的价值观和世界观在环境中不断地变化。大学四年让我初尝了人生冷暖。学得越多，越觉得自己只是知道冰山一角；在普速列车上和无座的人们交谈越深，越觉得自己生活的幸福和人生的参差。我深刻感受到我或许追不上人类社会的脚步，但是各位告诉我，我一定要好好生活好，生活不只有痛苦，也有快乐；适时可以停下来歇一歇看看路边的风景而不是只盯着追求耀眼的明珠，或许有新的收获；或许不情愿地走上了自己不想走的路，但可能殊途同归。

或许，有人在杀死知更鸟，或许，关山难越，谁悲失路之人，萍水相逢，尽是他乡之客，但更多的人在赠我芳香。

感谢大家。

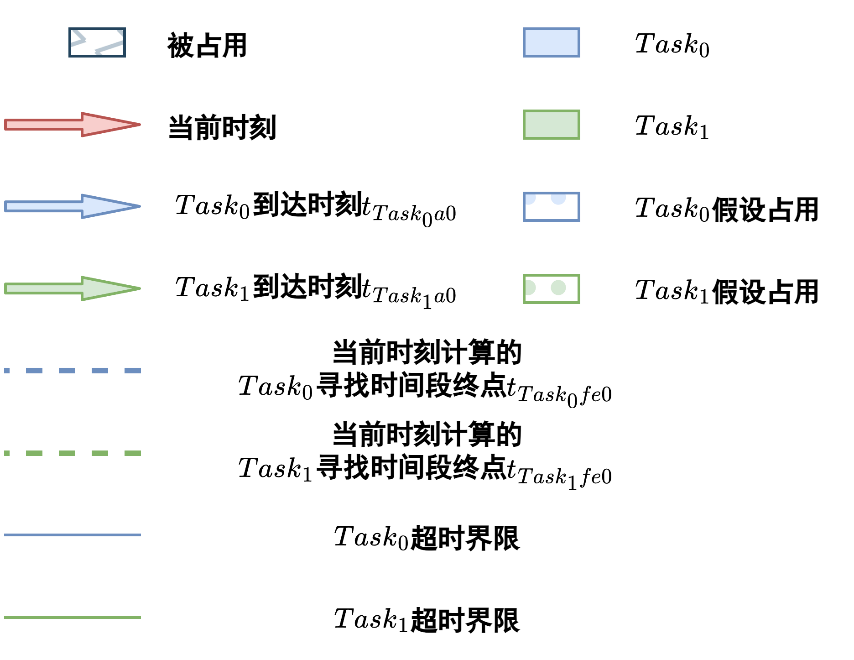
# 附 录

基于MaxUtil改进的延迟确认策略时延结果分析

**先到先服务和先结束先服务在MaxUtil算法中结果相近可能的原**因

在选择先到先服务和先结束先服务的延迟任务优先调度策略的情况下，不论是超时任务数还是能耗，在MaxUtil算法中结果相近。

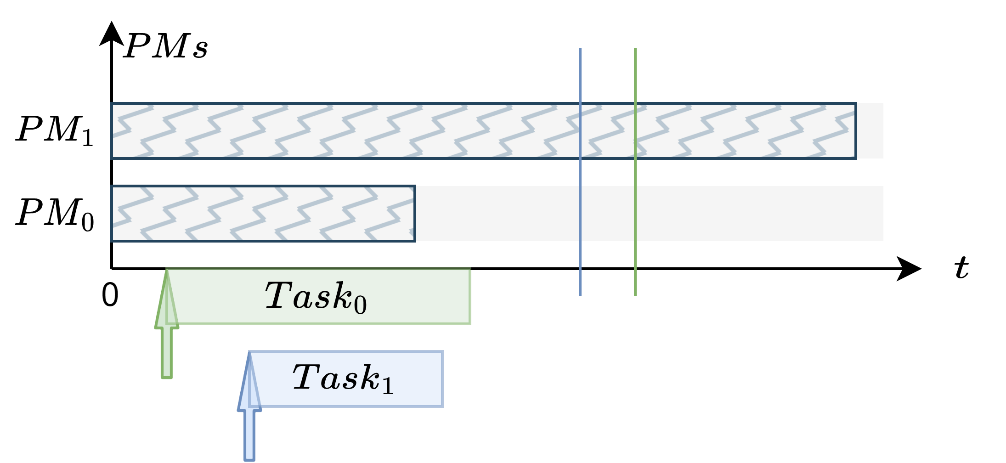
这是由于在使用先到先服务和先结束先服务的延迟任务调度策略时，如果按照间隙选择找到合适的间隙，即会确定任务占用该间隙，这与延迟确认的做法不同。也正是因为找到间隙即确定间隙，在任务到达稀疏的情况下，服务器大部分空闲而几乎不会有任务进入延迟列表（并在下一次B事件发生时）和其它任务进行排序比较；在任务到达较密集的情况如下：（各任务调度示意图图例如附录图 1‑1）



附录图 1‑1 各任务分配调度示意图图例

为了便于分析，假设服务器在时刻只要有任务占用，其占用率均为100%。

如附录图 1‑2，构造了两个任务和、服务器集，其中比先到达，但是的截止时刻早于，即

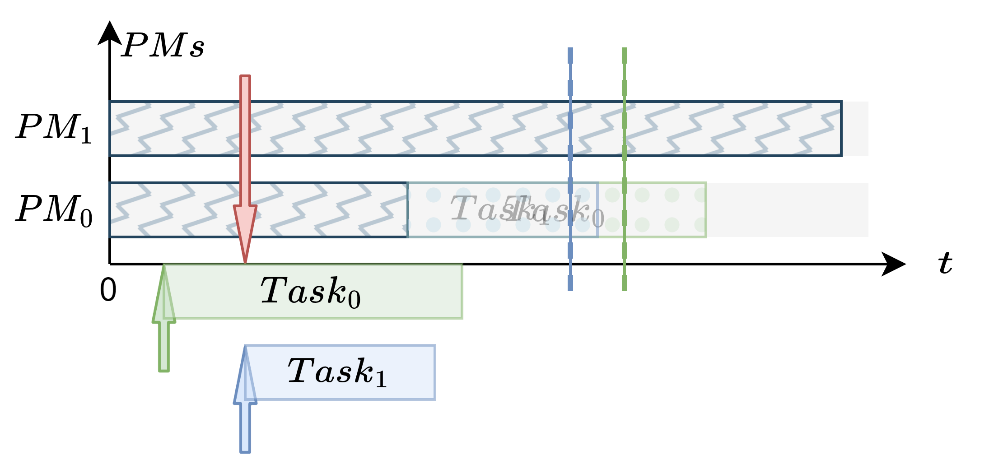


附录图 1‑2 构造的两个任务和

此时使用MaxUtil，可以分为五种情况。

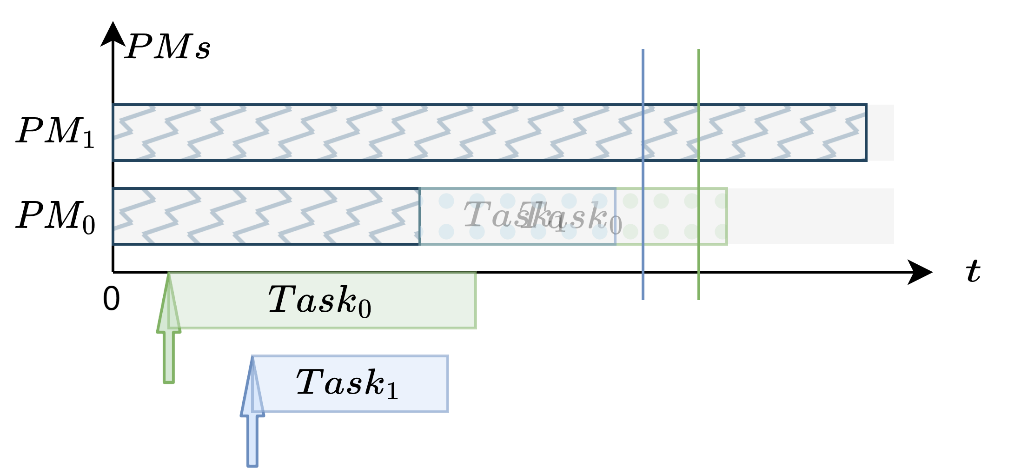
若和均没有机会不超时

如附录图 1‑3。不论如何使用先到达先服务策略还是先结束先服务两者超时任务数相同。



附录图 1‑3 和均没有机会不超时

若没有机会不超时而有机会不超时



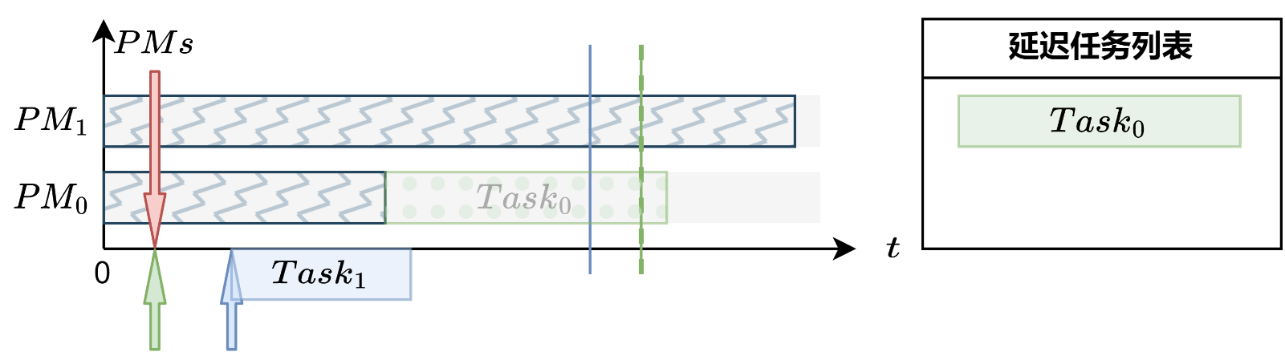
附录图 1‑4 有机会不超时而没有机会不超时

如附录图 1‑4所示，没有机会不超时而有机会不超时。

从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。

第一步：到达

如附录图 1‑5，到达。此时可以确定间隙寻找范围终点。寻找到的间隙无法容纳下的处理时长，因此将列入延迟任务列表



附录图 1‑5 到达

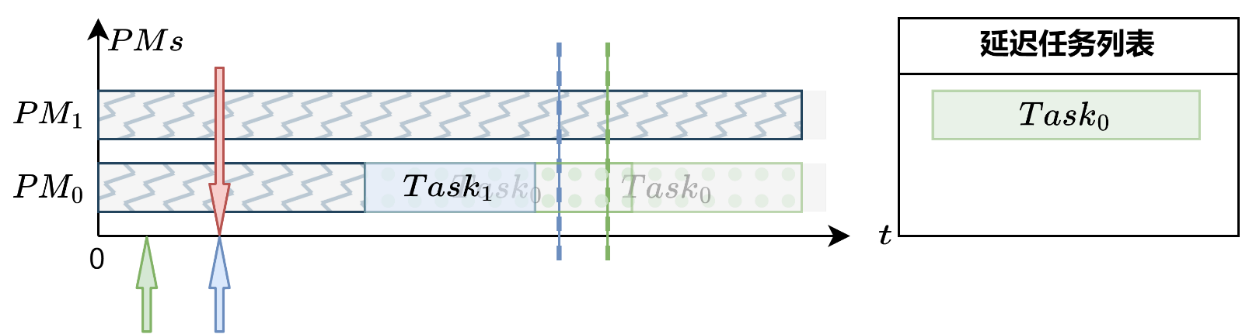
第二步：到达

当到达时，需要将延迟任务列表中的任务和进行排序，再根据任务优先调度策略按照优先指标排序任务按序寻找和分配。

当采用先到先服务的任务优先调度策略时，任务排序为。因为对于无法再寻找间隙的范围内（从红色箭头至绿色虚线）中找到合适的间隙进行安排，所以跳过，开始安排任务。对于任务可以找到唯一合适的间隙，并将该任务安排进间隙，如附录图 1‑6所示。

当采用先结束先服务的任务调度策略时，由于，所以任务排序为。对于是可以找到唯一合适间隙的，因此直接将该任务安排进间隙，亦如附录图 1‑6所示。

因此无论采用先到先服务还是先结束先服务的策略，在有机会不超时而没有机会不超时的情况下，调度的结果一致，因此超时的任务数相同，调度结果也相同。

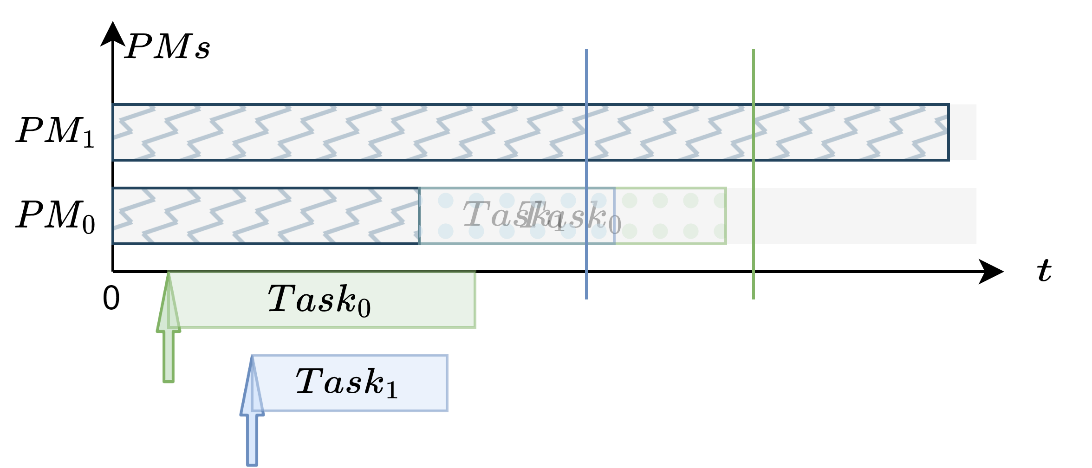


附录图 1‑6 到达

若有机会不超时而没有机会不超时

如附录图 1‑7所示，有机会不超时而没有机会不超时。

从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。



附录图 1‑7 有机会不超时而没有机会不超时

第一步：到达

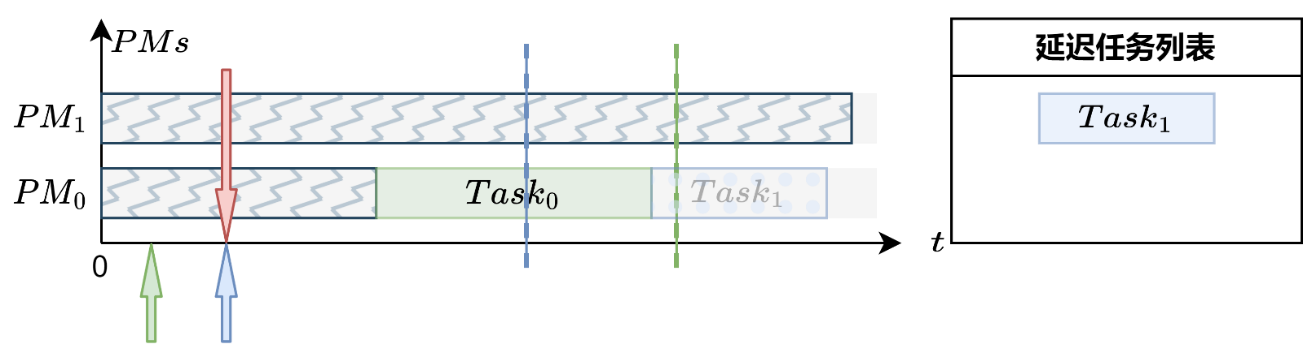
如附录图 1‑8所示，当到达时，由于任务列表中只有（还未到达因此不在任务列表中）此时对于只有一个间隙可以供选择。无论使用先到先服务还是先结束先服务的任务优先调度策略，由于只有一个任务待调度，因此结果相同。



附录图 1‑8 到达

第二步：到达

当到达时，由于此时任务列表中只有一个任务，并且由于找不到间隙，最终一定超时，如附录图 1‑9。不论是使用先到先服务还是先结束先服务的策略，超时任务数相同。

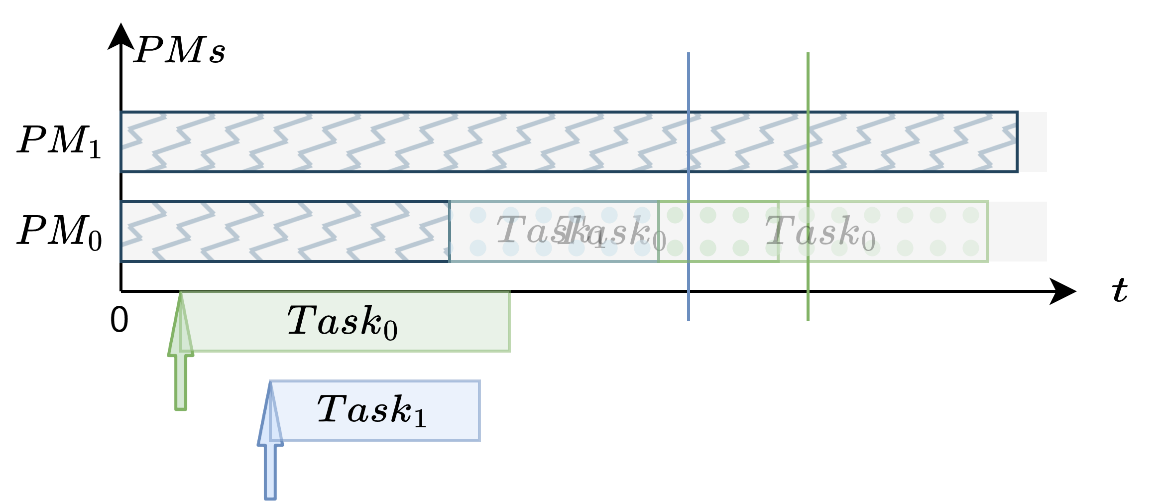


附录图 1‑9 到达

若和均有机会不超时但是不能全部不超时

如附录图 1‑10所示， 和均有机会不超时但是不能全部不超时。

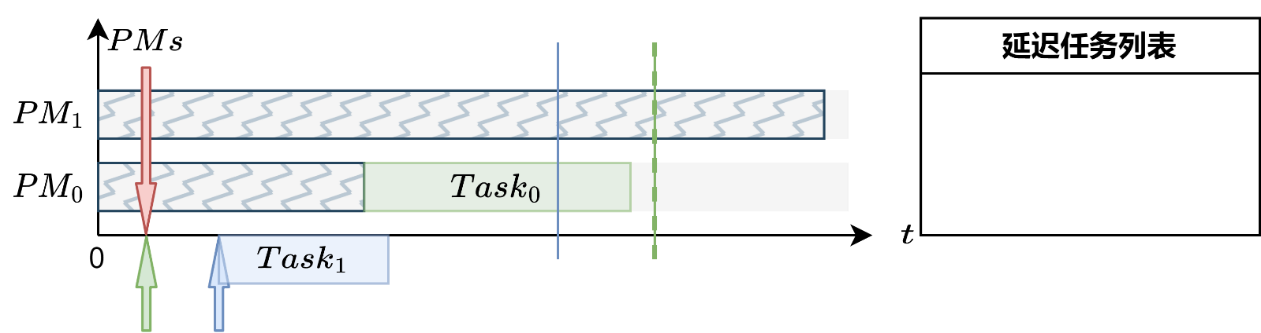
从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。



附录图 1‑10 和均有机会不超时但是不能全部不超时

第一步：到达

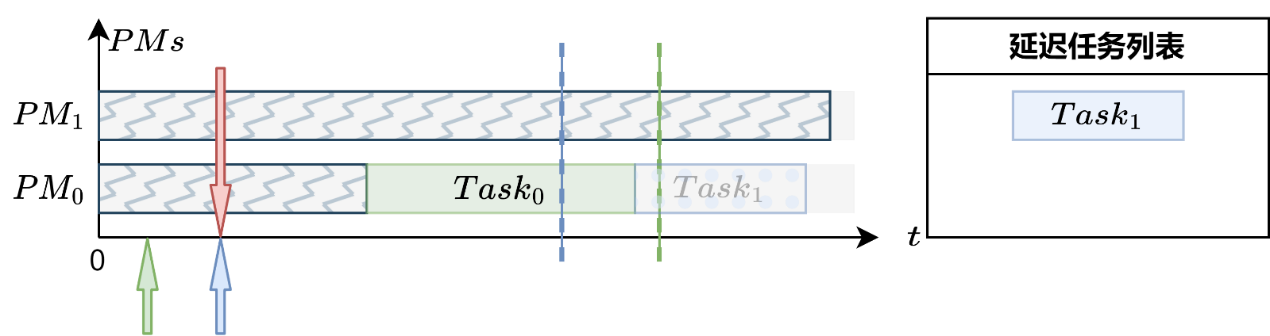
如附录图 1‑11 到达。此时任务列表只有一个任务。对只有一个间隙可供选择。选择间隙如图所示。



附录图 1‑11 到达

第二步：到达

如附录图 1‑12 到达。此时任务列表只有一个。此时对于无间隙可供选择。因此将加入延迟列表。因此无论是使用先到先服务还是先结束先服务的任务选择策略超时任务数均一致。



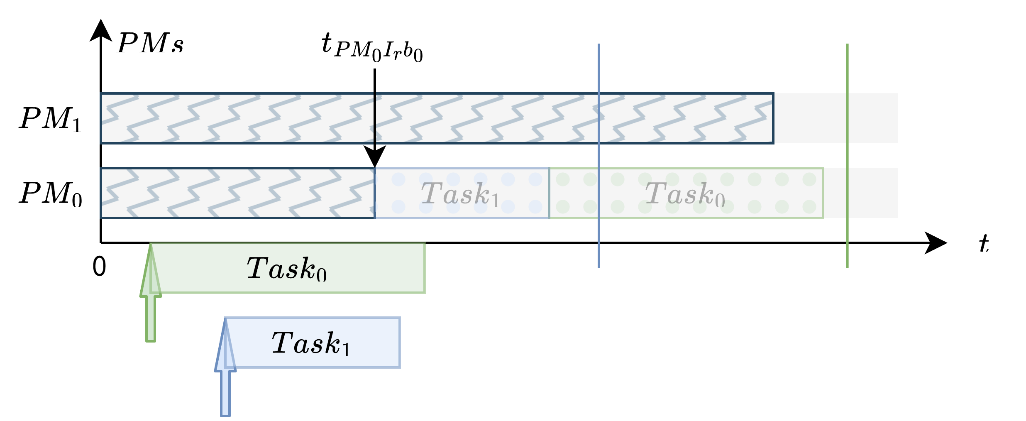
附录图 1‑12 到达

若和均有机会不超时且可以全部不超时

此时无论是使用先到先服务还是先结束先服务的策略，超时任务数均一致。

此时可以分为两种情况讨论。

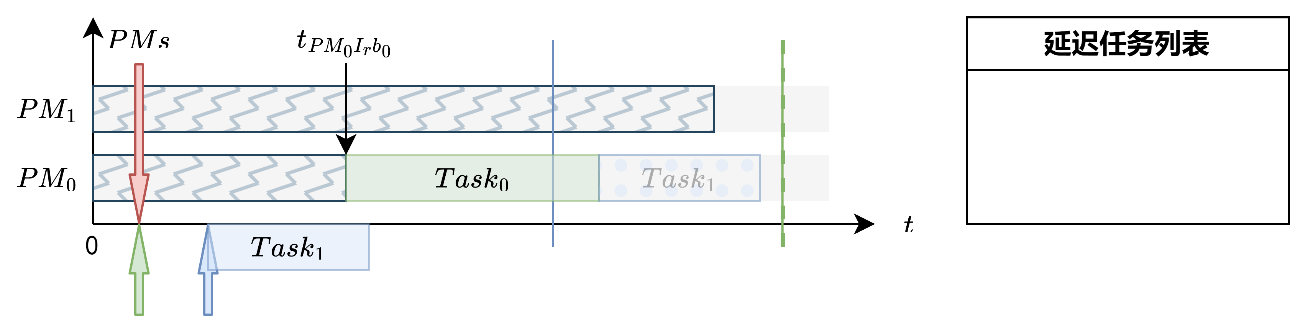
若，如附录图 1‑13。



附录图 1‑13 和均有机会不超时且可以全部不超时

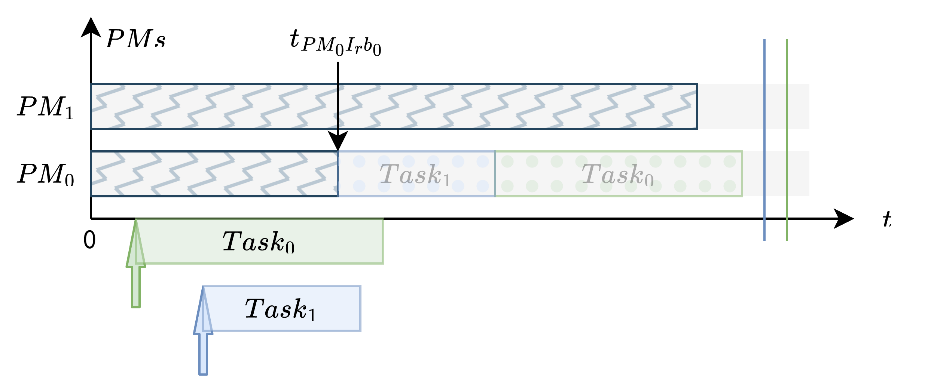
且

无论是使用先到先服务还是先结束先服务的策略，均在优先将安排进间隙。此时必定超时，如附录图 1‑14所示。不超时。



附录图 1‑14 时，到达时的安排及之后对的安排

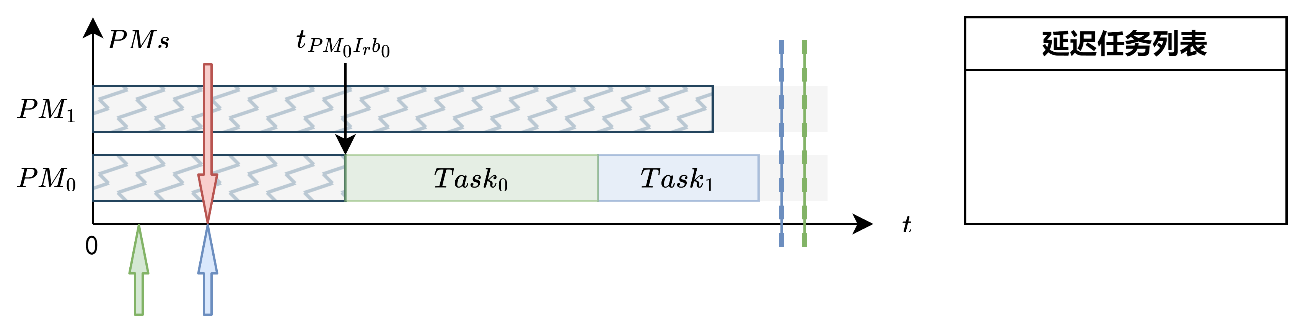
若，如附录图 1‑15所示。



附录图 1‑15 和均有机会不超时且可以全部不超时

且

无论是先到先服务还是先结束先服务的策略，和均不会超时。如附录图 1‑16所示。



附录图 1‑16 时，到达时的安排情况

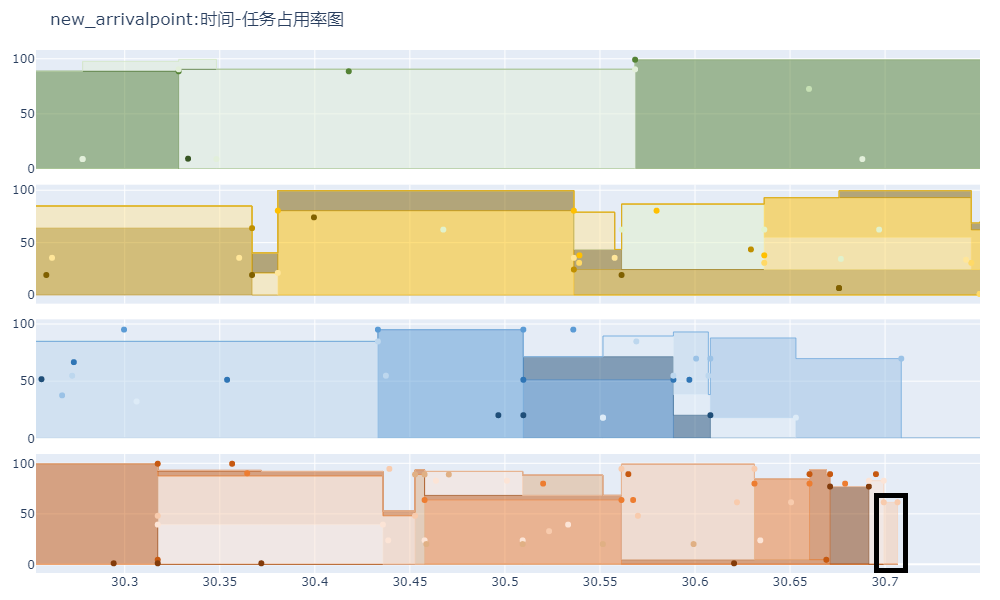
改变任务的计算资源占用率对结果的影响

上述考虑的情况均为任务对计算资源占用率为100%的情况，如果改变计算资源占用率，亦是先考虑对进行调度，再考虑对进行调度。若其中有一项未找到间隙，即加入延迟任务列表。大部分情况下不论是采用先到先服务的策略还是先结束先服务的策略，结果相同。

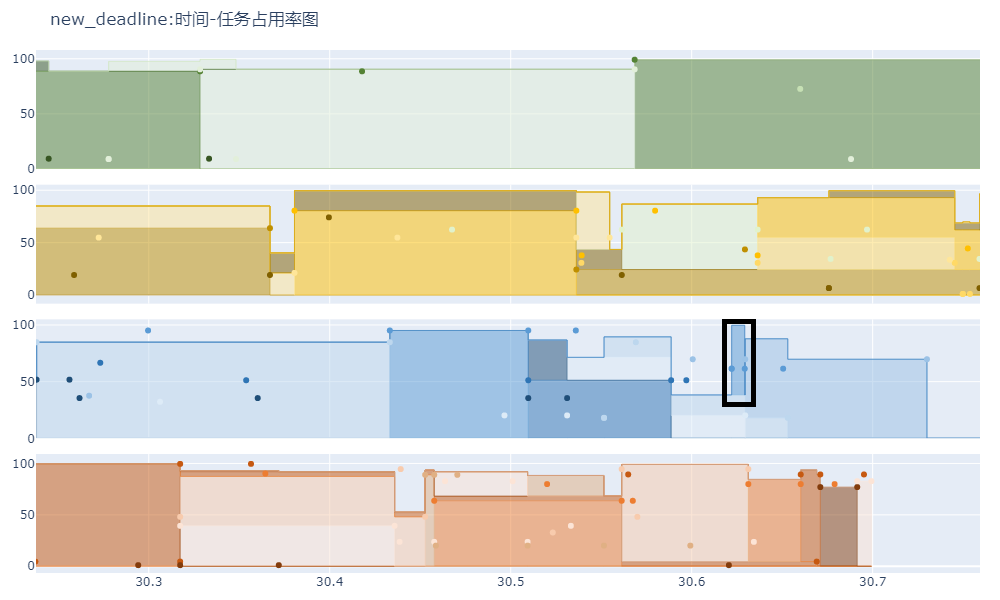
另外，在某些特殊情况下先到先服务和先结束先服务会出现结果不一致。

由于采用先到先服务和先结束先服务的调度策略不同导致不同时刻各服务器的计算资源占用率不同，也就导致了不同时间段可用间隙的计算资源占用率大小不同，可能会导致某些任务的延误情况不同。

如附录图 1‑17和附录图 1‑18是一个使用其它算法进行仿真的一个例子，但是依然适用于MaxUtil的情况。



附录图 1‑17 采用先到先服务的任务优先选择策略下的情况



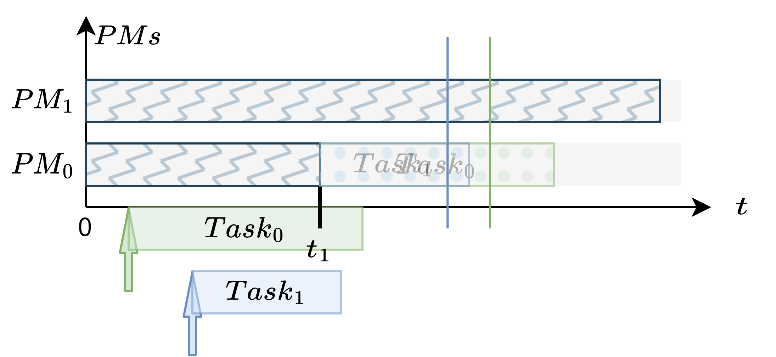
附录图 1‑18 采用先结束先服务的任务优先选择策略下的情况

高峰时期延迟确认策略对MaxUtil算法服务质量可能的影响

在高峰时期，有一定可能会出现任务延期。考虑以下几种情况以比较先到先服务/先结束先服务与延迟确认策略的不同。由于先到先服务/先结束先服务在上述的探讨中发现结果相近，并且延迟确认策略对延迟任务列表优先级排序也是先结束先服务，因此以先到先服务为例与延迟确认策略在以下情况的调度超时任务数量进行比较：

若和均没有机会不超时

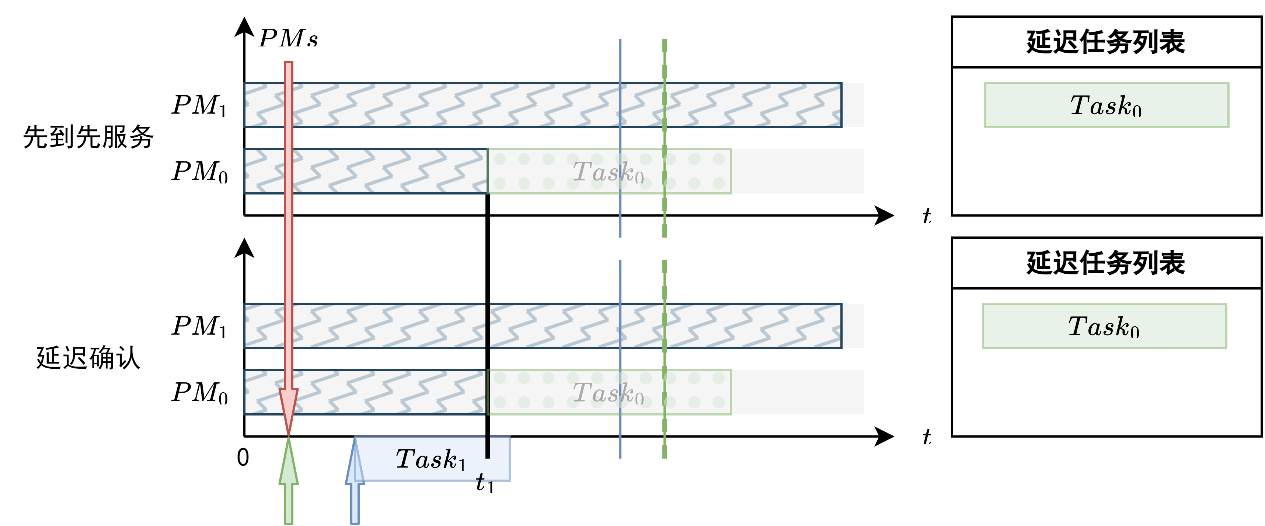
如附录图 1‑19。不论如何使用先到达先服务策略还是延迟确认策略两者超时任务数相同但是具体调度安排的顺序不同



附录图 1‑19和均没有机会不超时

第一步：到达

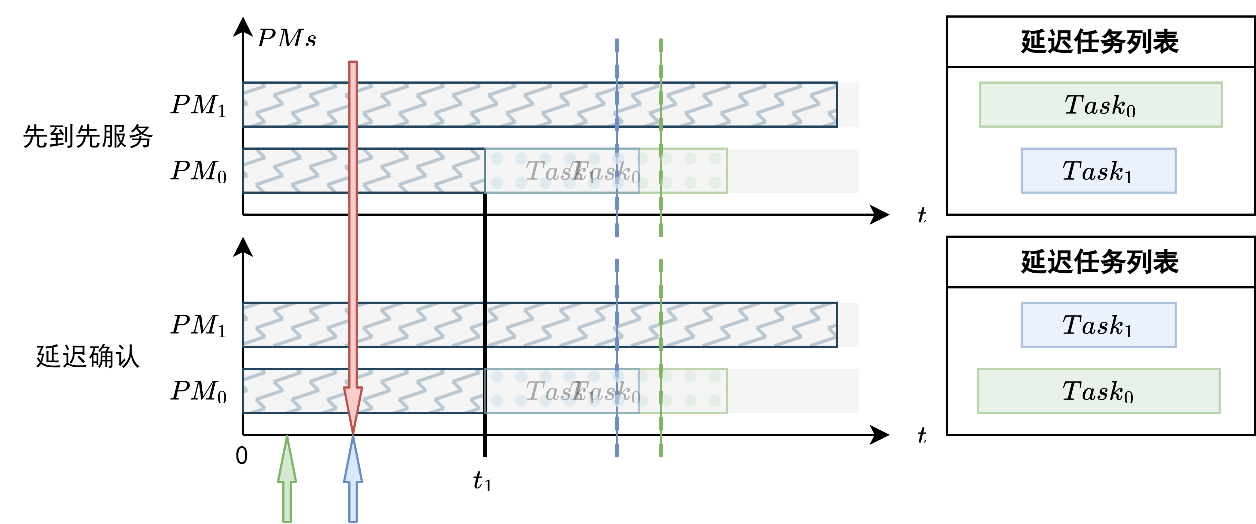
如附录图 1‑20，到达。此时由于均找不到间隙，因此两种任务优先调度策略均将加入延迟任务列表。



附录图 1‑20 到达

第二步：到达

如附录图 1‑21 ，到达，此时由于均找不到间隙，因此两种任务优先调度策略均将加入延迟任务列表。



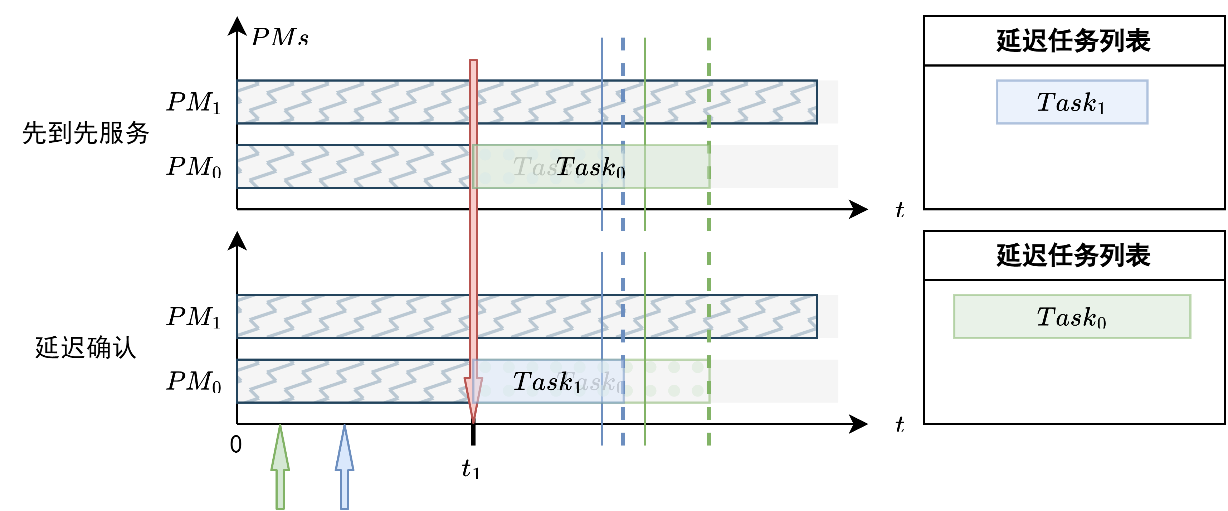
附录图 1‑21 到达

第三步：到达时刻

如附录图 1‑22，到达时刻时，无论是对于先到先服务还是延迟确认的策略，根据

寻找的间隙结束时刻均由于任务不得不超时而发生变化。

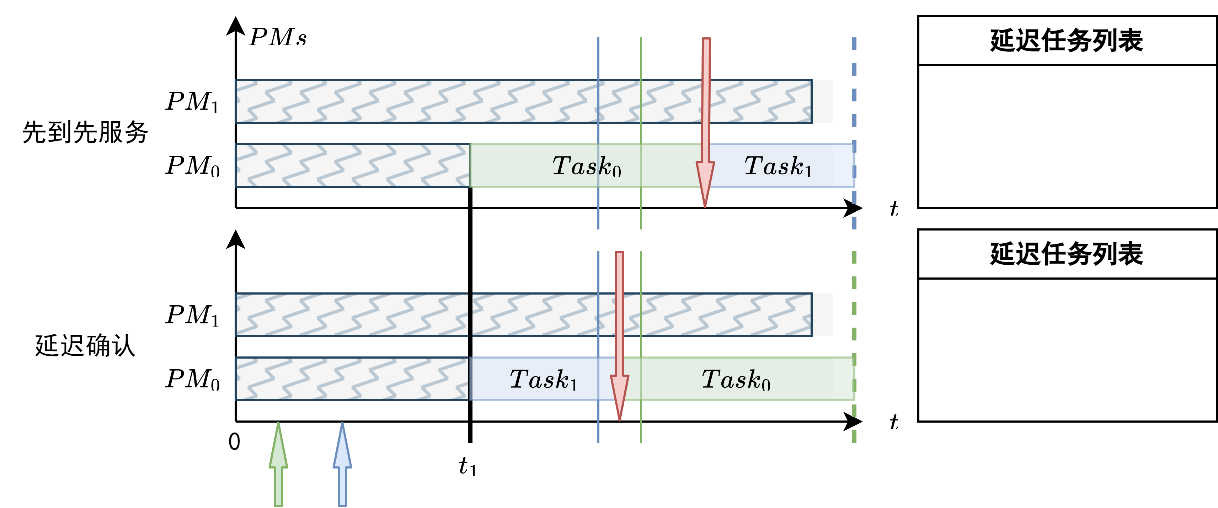
此时对于先到先服务的任务优先调度策略，由于先到达因此优先排序；对于延迟确认的任务优先调度策略，此时因为截止时间（细实线）早于，因此优先调度



附录图 1‑22 到达时刻

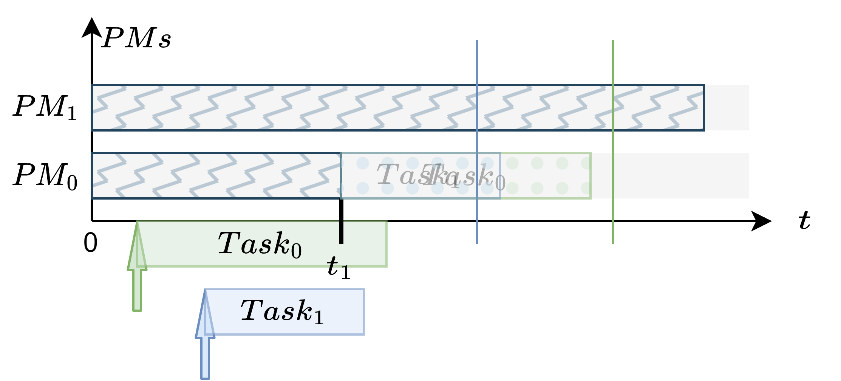
第四步：后续

最终，如附录图 1‑23，完成和两个任务，对于使用先到先服务和延迟确认策略消耗的时间相同；由于两者均超时，其服务质量也相同。



附录图 1‑23 后续

若有机会不超时而没有机会不超时



附录图 1‑24 有机会不超时而没有机会不超时

如附录图 1‑24所示，有机会不超时而没有机会不超时。

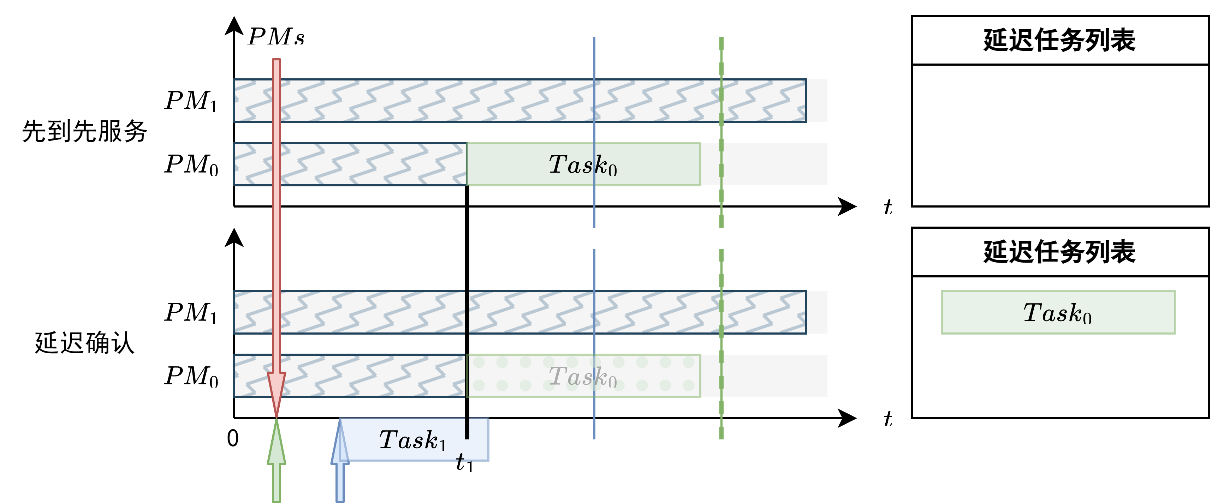
从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。

第一步：到达

如附录图 1‑25所示，到达。

在使用MaxUtil的间隙选择策略条件下，对于先到先服务的任务选择策略，由于在查找间隙过程中发现了唯一的适合于的间隙，因此直接将安排进间隙。

对于延迟确认的任务选择策略，虽然说找到了适合的间隙，但是由于间隙开始时刻并非当前时刻，因此将加入延迟任务列表中，待之后再进行调度



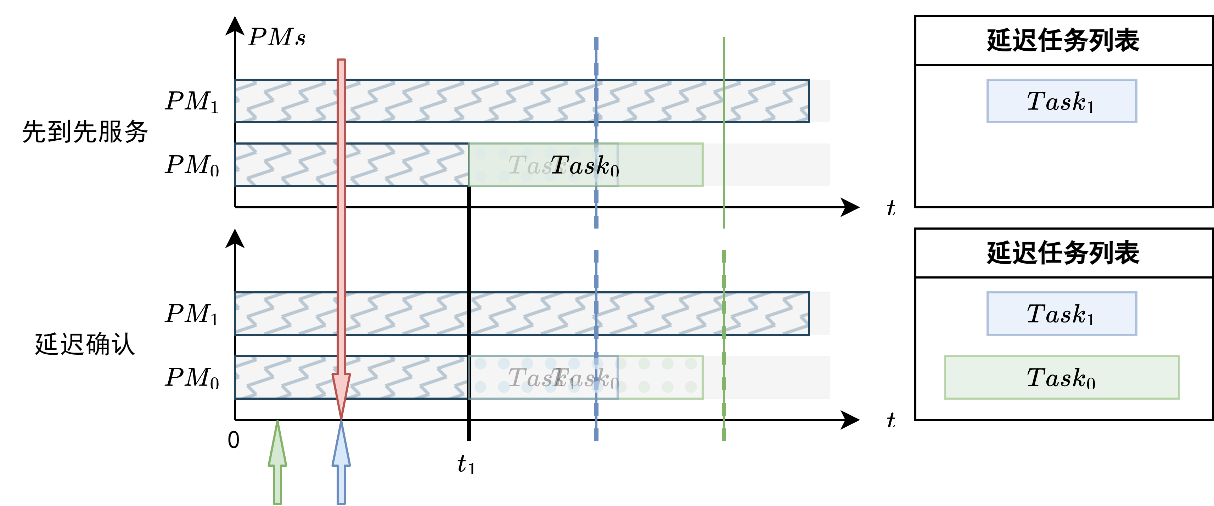
附录图 1‑25 到达

第二步：到达

如附录图 1‑26所示，到达。

对于先到先服务的任务选择策略，由于在查找间隙过程中没有发现适合的间隙，因此将其放进延迟任务列表。

对于延迟确认的任务选择策略，因为相比于早结束，此时对所有任务的间隙检索顺序为。先对进行间隙安排，没有找到合适的间隙，将保持在延迟任务列表中。接着对进行间隙安排，发现了截止时间前的一个合适的间隙，但是由于该间隙开始时刻并非当前时刻，因此仍然将保持在延迟任务列表中。



附录图 1‑26 到达

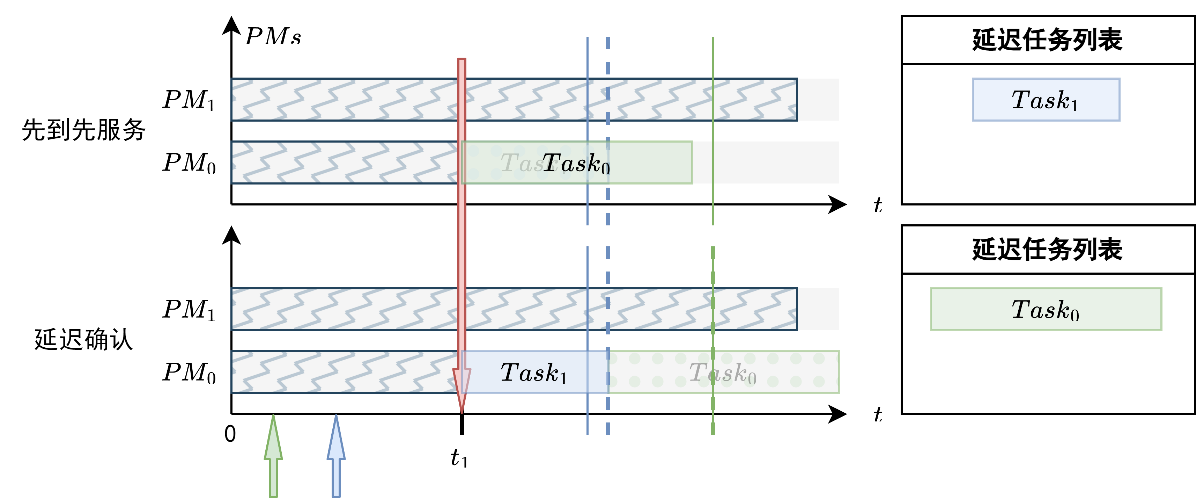
第三步：到达时刻

如附录图 1‑27所示。到达时刻时，对于先到先服务和延迟确认两种任务优先调度策略有不一样的调度结果。

对于先到先服务的任务优先调度策略，其先对进行调度。此时由于没有找到合适的间隙，只能够将保持在延迟任务列表中。

对于延迟确认的任务优先调度策略，根据

，所以任务调度优先顺序为。首先，对于可以找到一个超时的间隙。对于则不能找到合适的间隙。因此将保持在延迟任务列表中。



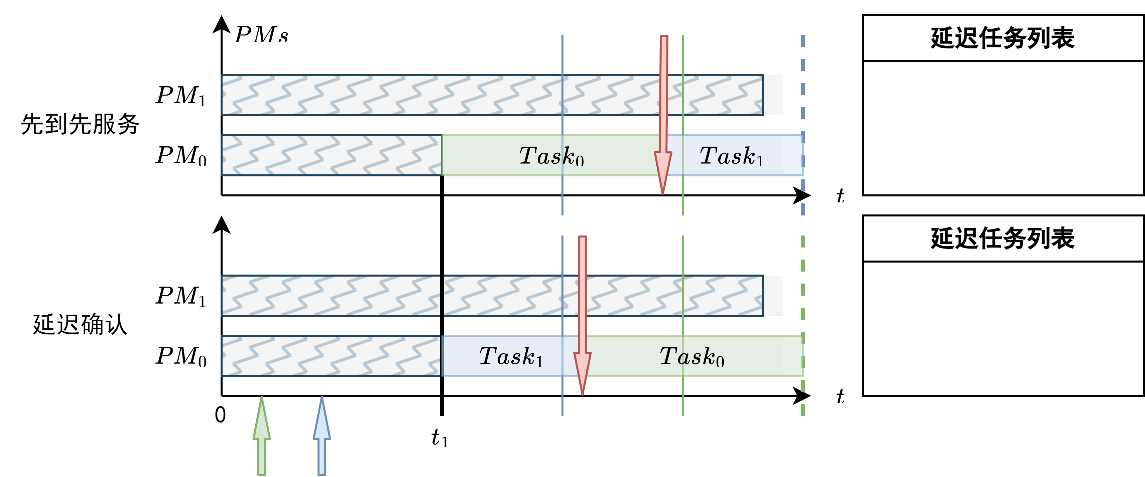
附录图 1‑27 到达时刻

第四步：后续

如附录图 1‑28对于先到先服务的任务优先调度策略，当完成时，对进行间隙安排。最终，没有超时而超时。

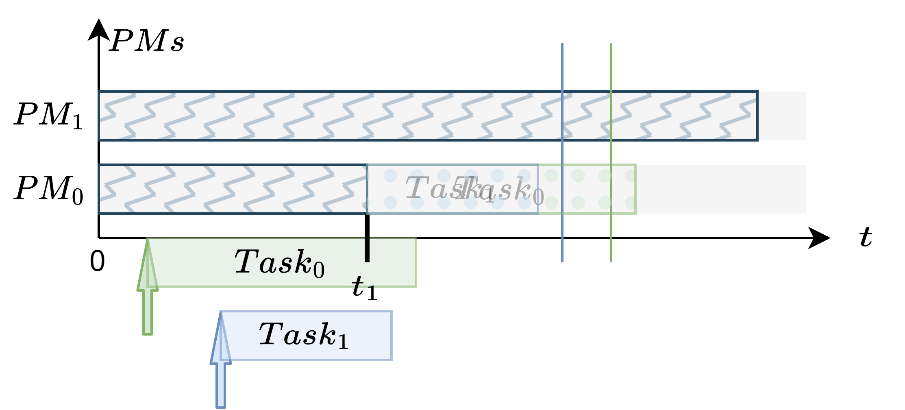
对于延迟确认的任务优先调度策略，当完成时，对进行间隙安排。最终，和均超时。

在这种情况下，延迟确认的任务优先调度策略任务超时数量高于先到先服务的情况，服务质量较差。



附录图 1‑28 后续

若有机会不超时而没有机会不超时



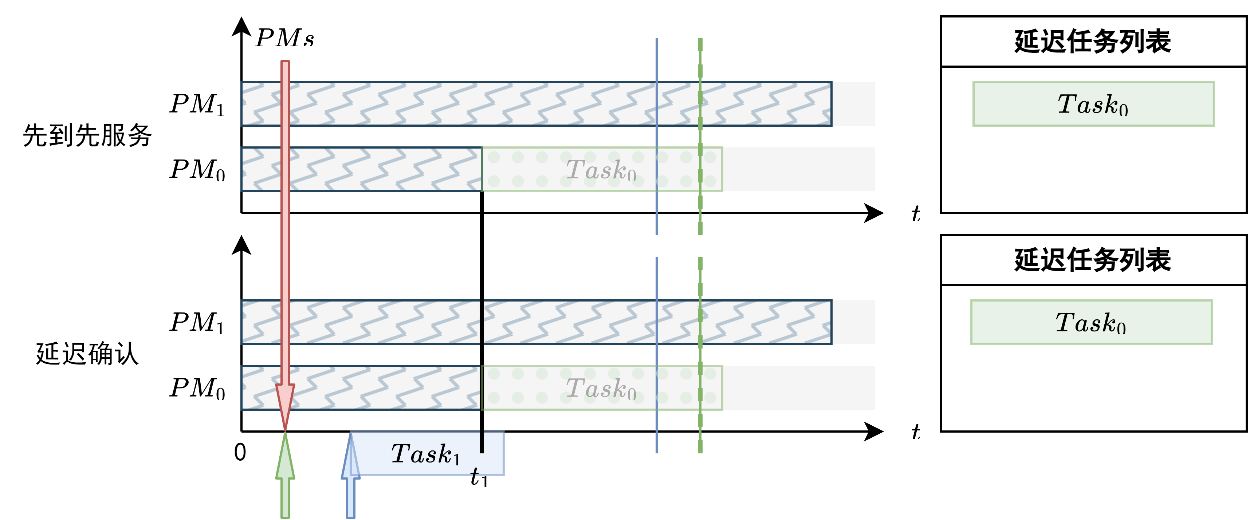
附录图 1‑29 有机会不超时而没有机会不超时

如附录图 1‑29所示，有机会不超时而没有机会不超时。

从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。

第一步：到达

如附录图 1‑30，到达。此时可以确定间隙寻找范围终点。无论是使用先到先服务还是延迟确认的策略，寻找到的间隙无法容纳下的处理时长，因此将列入延迟任务列表 。



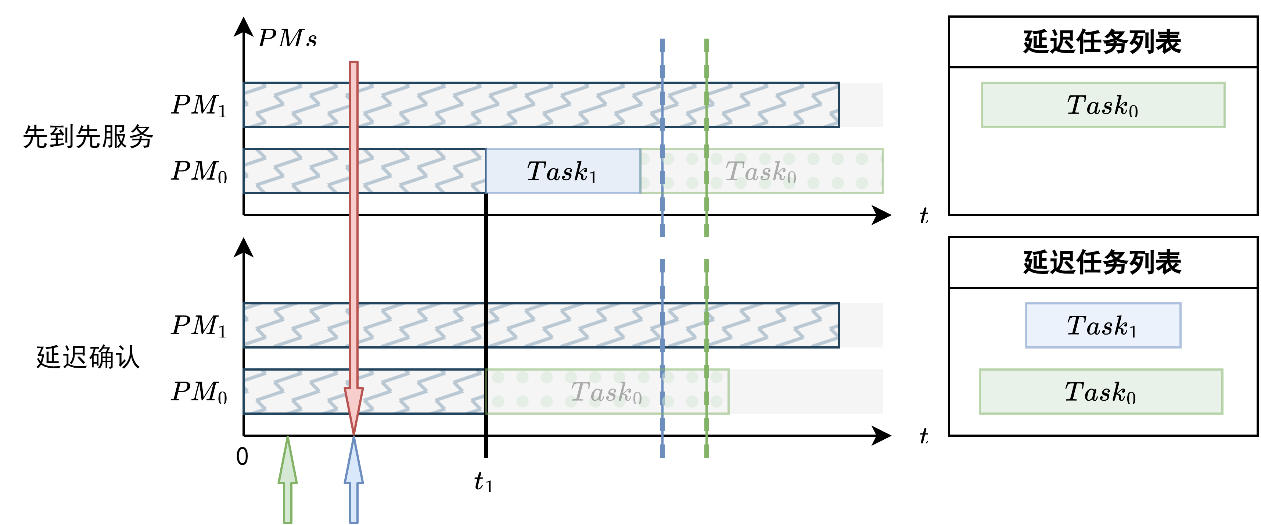
附录图 1‑30 到达

第二步：到达

当到达时，需要将延迟任务列表中的任务和进行排序，再根据任务优先调度策略按照优先指标排序任务按序寻找和分配。

当采用先到先服务的任务优先调度策略时，任务排序为。因为对于无法再寻找间隙的范围内（从红色箭头至绿色虚线）中找到合适的间隙进行安排，所以跳过，开始安排任务。对于任务可以找到唯一合适的间隙，并将该任务安排进间隙，如附录图 1‑31所示。对于，此时没有找到可用的间隙因此保持在延迟列表。

对于延迟确认策略，因为的截止时间早于，所以任务排序为。因为对于唯一可用的间隙和当前时刻不同，因此将任务放进延迟任务列表。对于没有找到可用的间隙因此将其保持在延迟列表中，如附录图 1‑31所示。

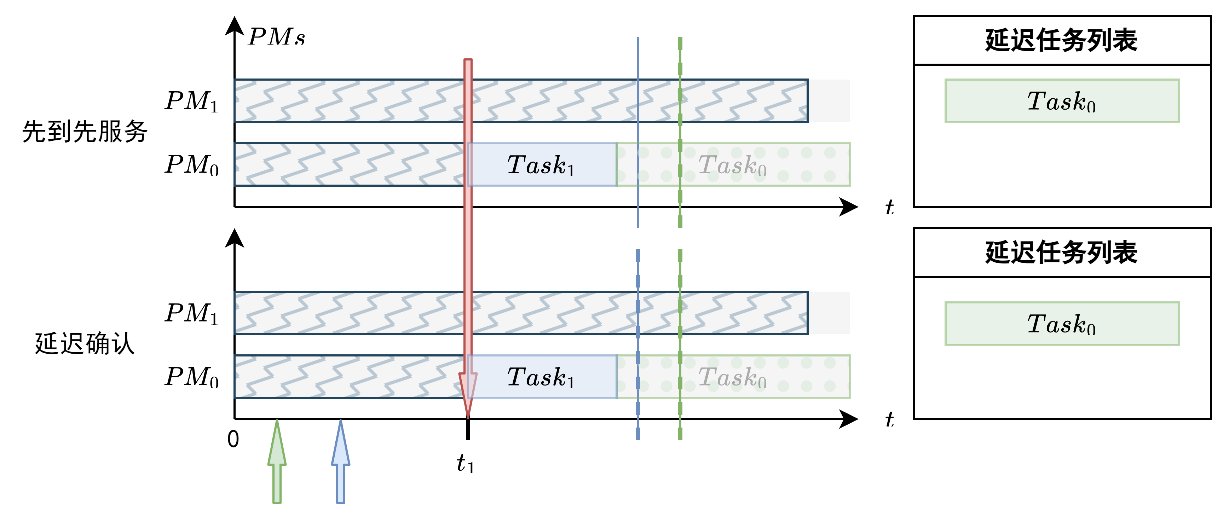


附录图 1‑31 到达

第三步：到达时刻

如附录图 1‑32，对于先到先服务的任务优先调度策略，需要安排间隙的任务只有。此时没有找到间隙，因此将保持在延迟任务列表中。

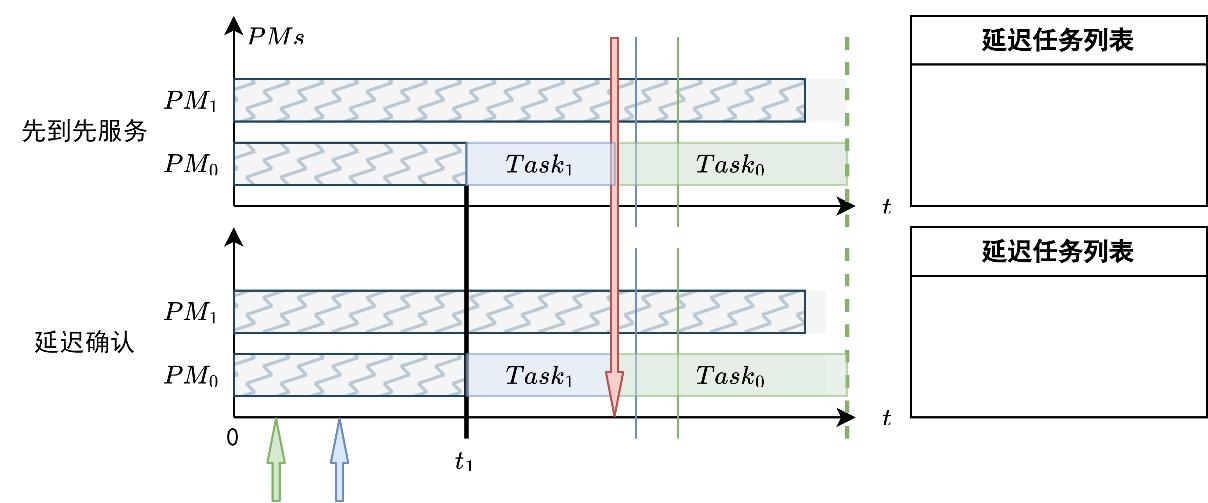
对于延迟确认的策略，因为的截止时间早于，所以任务排序为。对于有找到合适的间隙，并且当前时刻与间隙开始时刻相同，因此将排入并立即开始执行。对于没有找到合适的间隙，因此保留在延迟任务列表中。



附录图 1‑32 到达时刻

第四步：后续

无论是先到先服务还是延迟确认的任务优先调度策略，对于均不得不超时。如附录图 1‑33所示。

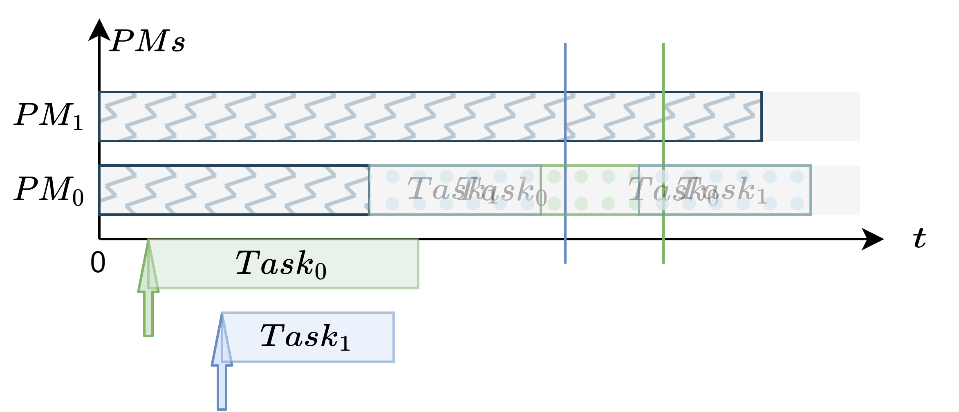


附录图 1‑33 后续

若和均有机会不超时但是不能全部不超时

如附录图 1‑34所示， 和均有机会不超时但是不能全部不超时。

从时刻开始，按照事件驱动，确定在B事件发生时延迟任务列表的变化。



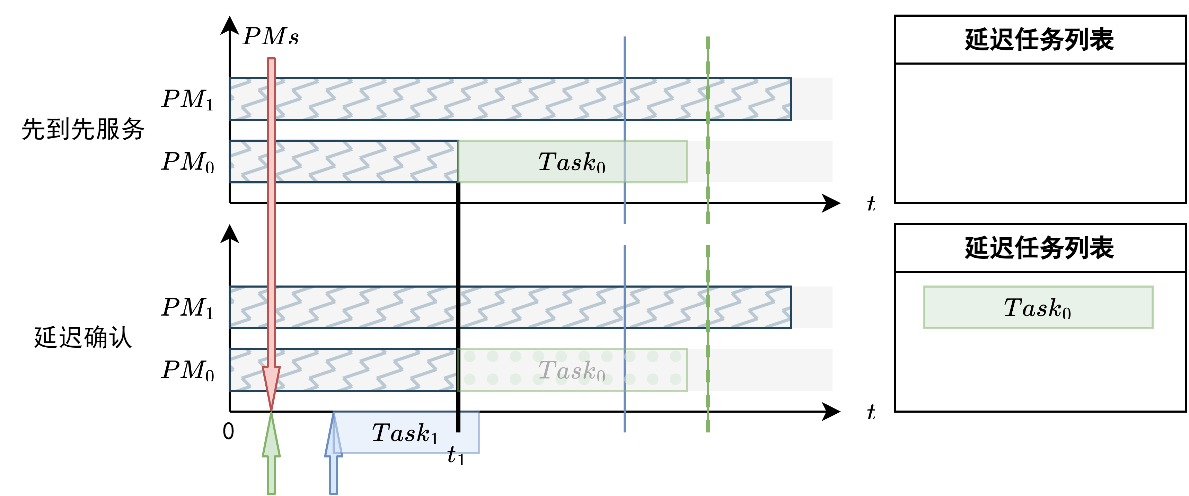
附录图 1‑34 和均有机会不超时但是不能全部不超时

第一步：到达

如附录图 1‑35所示，到达。

对于先到先服务的任务优先调度策略，对于有一个合适的间隙，被直接放进间隙，最终可以不超时地完成。

对于延迟确认的任务优先调度策略，可以找到合适的间隙，但是由于间隙起始时间和当前时刻不同，因此将放进延迟任务列表中。



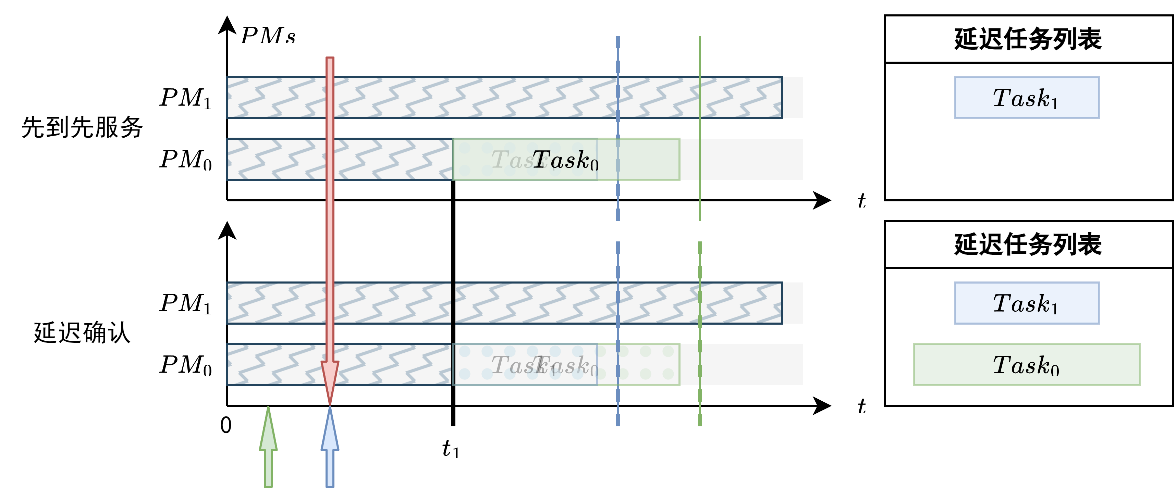
附录图 1‑35 到达

第二步：到达

如附录图 1‑36，到达。

对于先到先服务的任务优先调度策略，不能找到合适的间隙，因此被放进延迟任务列表

对于延迟确认的任务优先调度策略，对于所有任务的进行排序，排序结果为。可以找到合适的间隙，但是因为间隙的开始时刻和当前时刻不同，因此被放进延迟任务列表。同样是有合适的间隙，也是因为间隙的开始时刻和当前时刻不同，因此被放进延迟任务列表中。



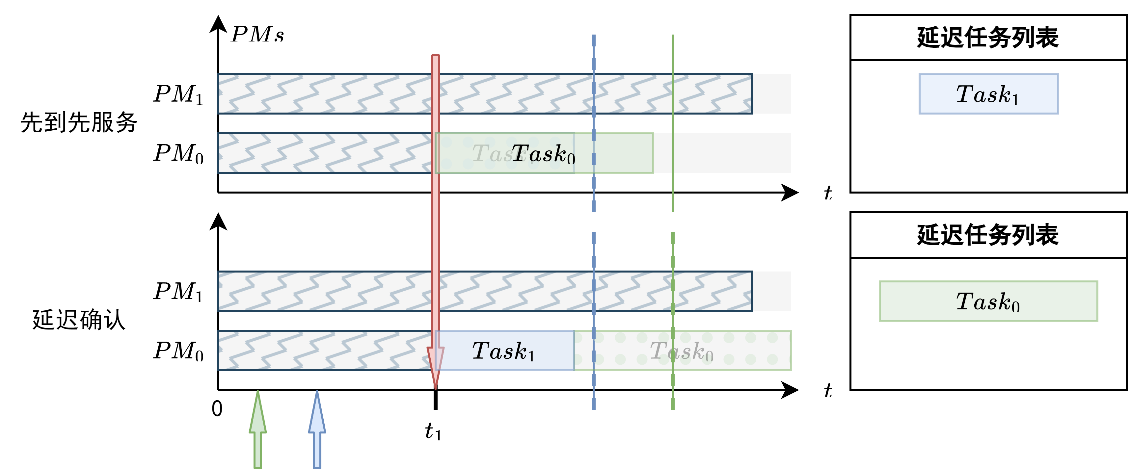
附录图 1‑36 到达

第三步：到达时刻

如附录图 1‑37，到达时刻。

对于先到先服务的任务优先调度策略，此时只考虑的的间隙选择。因为没有选择到合适的间隙，所以将保留在延迟任务列表中。

对于延迟确认的任务优先调度策略，此时根据每一个任务的对所有任务进行排序决定优先调度顺序，结果为。对于可以找到合适的间隙，并且可以不超时地完成。之后对进行调度，没有找到合适的间隙，因此将保留在延迟任务列表中。



附录图 1‑37 到达时刻

第四步：后续

如附录图 1‑38，对于先到先服务的任务优先调度策略，当结束时，考虑的间隙。此时只能超时完成。

对于延迟确认的任务优先调度策略，当结束时，考虑的间隙，此时只能超时完成。

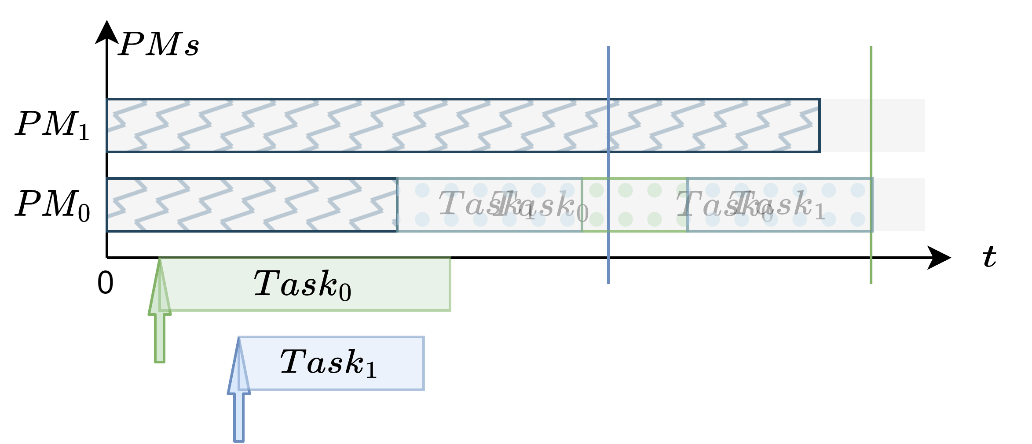


附录图 1‑38 后续

若和均有机会不超时且有可能全部不超时

如附录图 1‑39进行任务的安排，若，那么不论是延迟确认策略还是先到先服务策略两个任务无论如何调度均不超时。

若，延迟确认策略和先到先服务策略，服务质量结果不同。以下针对这种情况进行分析。



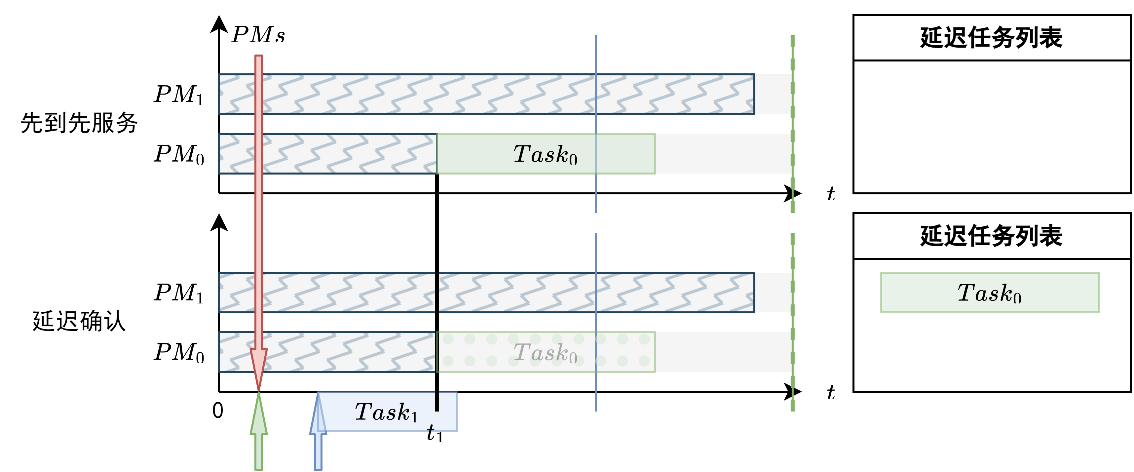
附录图 1‑39 和均有机会不超时且有可能全部不超时，且

第一步：到达

如附录图 1‑40，到达。

对于先到先服务的任务优先调度策略，对于有找到唯一一个合适的间隙。都被直接安排进这个间隙。

对于延迟确认的任务优先调度策略，可以找到合适的间隙，但是由于间隙起始时间和当前时刻不同，因此将放进延迟任务列表中。



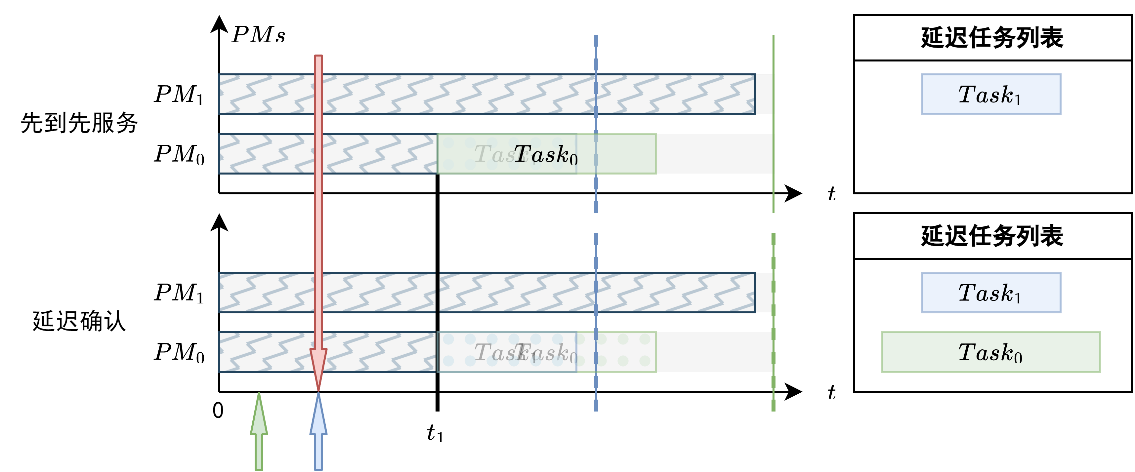
附录图 1‑40 到达

第二步：到达

如附录图 1‑41，到达。

对于先到先服务的任务优先调度策略，没有找到合适的间隙，因此被放到延迟任务列表中。

对于延迟确认的任务优先调度策略，对于所有任务的进行排序，排序结果为。可以找到合适的间隙，但是因为间隙的开始时刻和当前时刻不同，因此被放进延迟任务列表。同样是有合适的间隙，也是因为间隙的开始时刻和当前时刻不同，因此被放进延迟任务列表中。



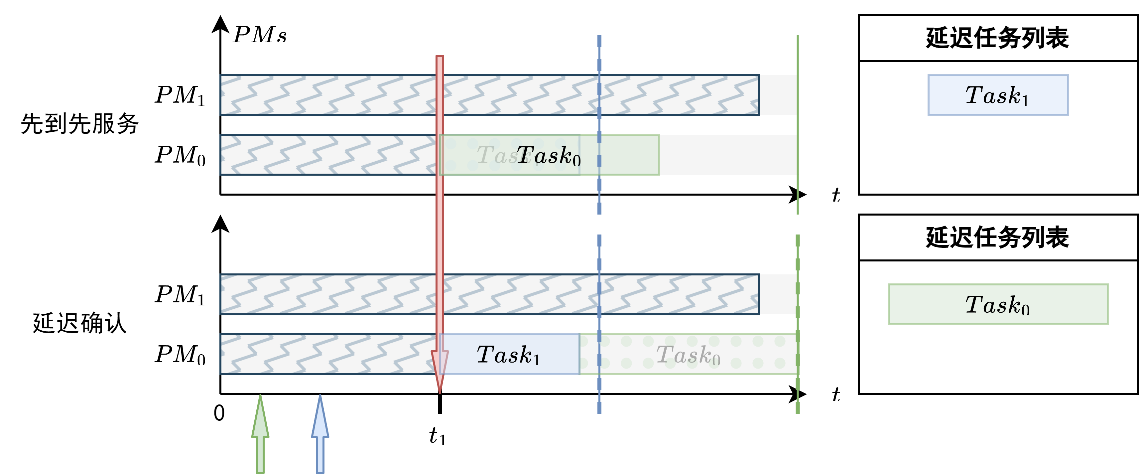
附录图 1‑41 到达

第三步：到达时刻

如附录图 1‑42，到达时刻。

对于先到先服务的任务优先调度策略，此时只考虑的的间隙选择。因为没有选择到合适的间隙，所以将保留在延迟任务列表中。

对于延迟确认的任务优先调度策略，此时根据每一个任务的对所有任务进行排序决定优先调度顺序，结果为。对于可以找到合适的间隙，并且可以不超时地完成。之后对进行调度，虽然可以找到合适的间隙，但是间隙开始时刻和当前时刻不同，因此将保留在延迟任务列表中。



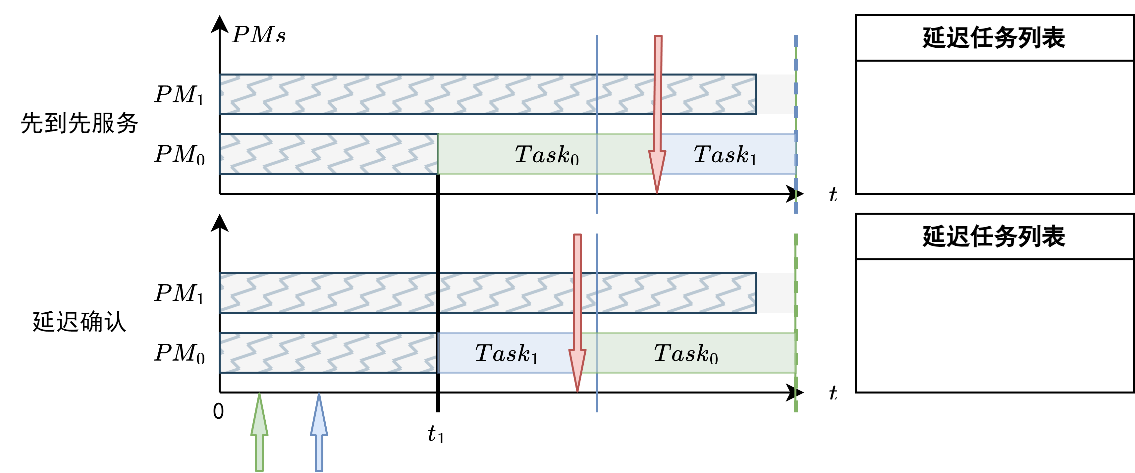
附录图 1‑42 到达时刻

第四步：后续

如附录图 1‑43对于先到先服务的任务优先调度策略，当结束时，考虑的间隙。此时只能超时完成。

对于延迟确认的任务优先调度策略，当结束时，考虑的间隙，此时能够不超时地完成。

因此，对于和均有机会不超时且有可能全部不超时，且的情况下，延迟确认策略可以避免因为间隙已被安排导致任务出现超时的现象，服务质量较先到先服务的情况好。



附录图 1‑43 后续

综上所述，对于1.2.2情况，延迟确认的策略服务质量劣于先到先服务策略的服务质量。对于1.2.5情况，延迟确认的策略服务质量优于先到先服务策略的服务质量。

代码仓库链接

训练、实验代码将上传到代码仓库中，具体如何使用和训练模型请阅读README文件。

代码仓库链接：<https://github.com/lll123github/Graduate>

外文译文

译文1 关于端边云编排网络计算范式的综述

摘要

在过去几十年中，将数据发送到云端进行分析是一个突出趋势，推动云计算成为主导的计算范式。然而，在物联网时代，设备数量和数据流量的急剧增加，给容量有限的互联网带来了巨大的负担和不可控的服务延迟。仅使用云计算很难满足物联网应用的延迟敏感和上下文感知服务需求。面对这些挑战，计算范式正在从集中式云计算转向分布式边缘计算。包括透明计算、移动边缘计算、雾计算和Cloudlet在内的几种新的计算范式已经出现，以利用网络边缘的分布式资源来提供及时和上下文感知的服务。通过集成终端设备、边缘服务器和云，它们形成了分层的物联网架构，即端边云编排架构，以提高物联网系统的性能。本文从端边云编排的角度对这些新兴的计算范式进行了全面调查。具体来说，我们首先介绍并比较了不同计算范式的架构和特点。然后，进行了全面的调查，讨论了计算卸载、缓存、安全和隐私方面的最新研究。最后，为促进持续的研究工作设想了一些潜在的研究方向。

CCS概念：

* 以人为本的计算 → 泛在计算
* 计算机系统组织 → 云计算

其它关键词：端边云编排、网络计算、透明计算、移动边缘计算、雾计算、Cloudlet

介绍

过去几十年，互联网和无线通信技术的发展为人们的日常生活提供了非常便捷的信息交流渠道。到 2019 年，全球移动终端数量将成倍增长，达到约 28 亿台。特别是随着人工智能和智能科学的发展，智能轻量级设备的数量呈指数级增长，万物互联已成为无线通信网络和互联网发展的主要趋势[125]。这也意味着部署在网络边缘的大量传感器、执行器和移动设备的物联网（IoT）时代的到来。思科公司的一份报告显示，到 2021 年，全球每月移动数据流量将达到 49 艾字节，从 2016 年到 2021 年的复合年增长率为 47%[7]。这些设备产生的相当一部分计算任务，如虚拟现实、增强现实和工业控制等，都需要及时的上下文感知处理。因此，处理海量数据流量是未来互联网和无线通信系统的一个关键特征。此外，高数据传输速率和低传输延迟成为未来互联网和无线通信网络的两个关键性能指标。这意味着，强大的计算设备需要处理海量数据流量，而互联网和无线通信网络也分别需要高数据速率的传输链路来传输数据流量。

从无线通信系统的角度来看，超密集网络、大规模多输入多输出（MIMO）和高频通信被认为是满足未来无线通信不断增长的需求的有前途的方法，例如5G无线系统。与 4G 相比，预计通过利用这些技术，5G 的容量将增加 1,000 倍。除了容量的大幅提升外，还有望在数据传输速率、网络可靠性、频谱和能源效率等方面实现显着提高[133]。这也意味着5G系统中设计的未来无线通信技术将提供强大的能力来传送各种通信设备产生的数据流量。

然而，在过去的几十年里，各种计算架构和范式被设计来提供从互联网角度处理数据流量的强大能力。自从第一台计算机ENIAC问世以来，计算机、互联网和信息技术的发展使人们进入了信息爆炸的时代。为进一步满足信息社会的各种要求，需要计算机网络、计算模式、存储模式、应用模式发生革命性变革。计算模式的发展经历了单机计算、集群计算、网络计算、云计算等阶段。集群计算的出现就是为了解决单机计算无法处理海量计算数据服务的缺点。发展更加灵活的网络计算，提高集群计算在异构性、动态性、分布性、可扩展性等方面的业务处理能力。尽管网络计算可以提供相当大的计算能力来处理数据，但它仍然无法满足指数级增长的移动设备和数据流量所带来的不断增长的需求。

集中计算和存储资源的云计算被认为是第二代网络计算，被认为是21世纪最有前途的技术之一，为应对计算挑战提供了强大的计算和存储能力。特别是，云计算可以通过广域网（WAN）为最终用户提供弹性服务和数据密集型分析。因此，用户无需构建新的计算基础设施即可获得看似无限的资源。Gartner预测云计算带来的全球收入将从2092亿美元增长至2468亿美元。凭借云计算的显著经济效益，它可能会牢牢地留在计算领域[129]。然而，尽管计算资源在云端的集中化便于资源管理和维护，但云计算在满足物联网时代时延敏感应用新趋势的服务需求方面存在困难。第一个问题是不可接受的广域网延迟，在可预见的未来不太可能得到改善，因为广域网的设计目标主要集中在提高带宽和链路的效率上[130]。第二个问题是，物联网设备生成的数据量急剧增加，广域网的流量将受到重大挑战。例如，在机场监控应用中，出于安全目的部署了数千个摄像机，每个摄像机以12 Mbps的速度产生数据[10]。要在中央云服务器上仅分析视频数据，需要数百Gbps的带宽来收集视频数据，这远远超过了当前WAN的流量容量。

为了解决这些问题，出现了几种新的网络计算范式，以在最终用户附近提供计算资源。通过这种方式，可以在没有WAN参与的情况下提供延迟敏感和上下文感知服务[90]。透明计算（TC）、雾计算（Fog）、移动边缘计算（MEC）和Cloudlet等新兴网络计算范式引起了业界和学术界的广泛关注。这些范式采用计算资源有限的小型边缘服务器，及时为网络边缘的最终用户提供服务。边缘服务器可以是临时设备，如智能手机、笔记本电脑、高级路由器和微型服务器等，也可以是附近的一些基础设施。Fog、MEC 和 cloudlet 可以被视为将云服务扩展到网络边缘，因为它们利用了类似的计算卸载和存储管理方案。然而，在TC的愿景中，计算和存储被分为终端设备和远程服务器[176]。具体来说，TC鼓励终端设备及其附近的设备承担计算任务，并从远程服务器获取软件和数据。因此，现代设备的计算能力可以得到充分利用。值得注意的是，上述所有计算范式都强调为边缘的最终用户提供服务，并为物联网环境中的延迟敏感应用程序提供服务。

在新兴计算范式进步的推动下，我们可能会看到一种可以彻底改变当前云计算架构的分层计算架构[50]。它由大规模中央服务器、部署在网络边缘的众多边缘服务器以及大量分布式终端设备组成。大多数应用程序不将它们视为独立的部分，而是要求对所有部分进行精心编排，以便在不同的时间和空间尺度上提供可靠的服务。例如，在机场监控应用中，边缘服务器可以在将整批视频上传到中央服务器之前对视频流进行分析和过滤，这样可以显著减少广域网上的流量，在不损失性能的情况下减轻中央服务器的负担。此外，考虑到最终用户从提供缓存服务的边缘服务器请求某些感兴趣的内容的情况，中央服务器可以作为存储容量有限的边缘服务器的补充。

一些研究从不同的角度调查了云计算或边缘计算。在参考文献[68]中，作者讨论了移动云计算的架构和性能优化方法，它采用集中式云服务器通过广域网为移动用户提供计算卸载和存储。参考文献[174]调查了大数据分析的计算技术，包括云计算、TC和雾计算。人们认为，由于广域网的带宽有限和上下文感知要求，在网络边缘而不是中央云服务器分析数据可能是物联网环境中更好的解决方案。在参考文献[113]中，作者讨论了边缘计算范式中计算任务卸载的现有工作，主要关注能耗和延迟之间的权衡。参考文献[150]调查了边缘计算增强的无线接入网（RAN）背景下的缓存策略的工作。

尽管上述调查很有启发性，但它们都没有特别关注分层计算架构中的研究问题，这为终端设备、边缘和云的编排提供了巨大的好处。本调查的范围涵盖了不同网络计算范式的比较和不同的研究问题，包括分层计算架构下的计算卸载、缓存、安全和隐私。本次调查的其余部分组织如下。第 2 节介绍并比较了新兴的计算范式。第3节回顾了新兴计算范式中的计算卸载研究。第 4 节和第 5 节分别总结了缓存策略以及安全和隐私保护机制。第6节概述了这一新兴研究领域的一些潜在的未来方向，随后在第7节中给出了结论。

新兴的计算范式和评估标准

在过去的几年中，为了解决海量数据计算问题，人们提出了各种计算范式来提供及时和资源节约型的服务。在本节中，我们将介绍近年来出现的计算范式，包括透明计算（以下简称TC）、雾计算（Fog）、移动边缘计算（MEC）和Cloudlet。尽管它们在处理异构设备产生的大量计算和存储任务时面临一些共同的问题，例如计算和存储资源的管理以及网络，但由于各种原始驱动力，它们也表现出不同的特征，例如TC的说服力计算和雾计算的物联网应用。例如，所有这些计算范式都需要掩盖各种设备的异构性，以简化资源管理。为此，TC采用了操作系统（OS）级别的解决方案，例如Meta OS[178]，而其他三种范式则侧重于虚拟化和容器化解决方案。

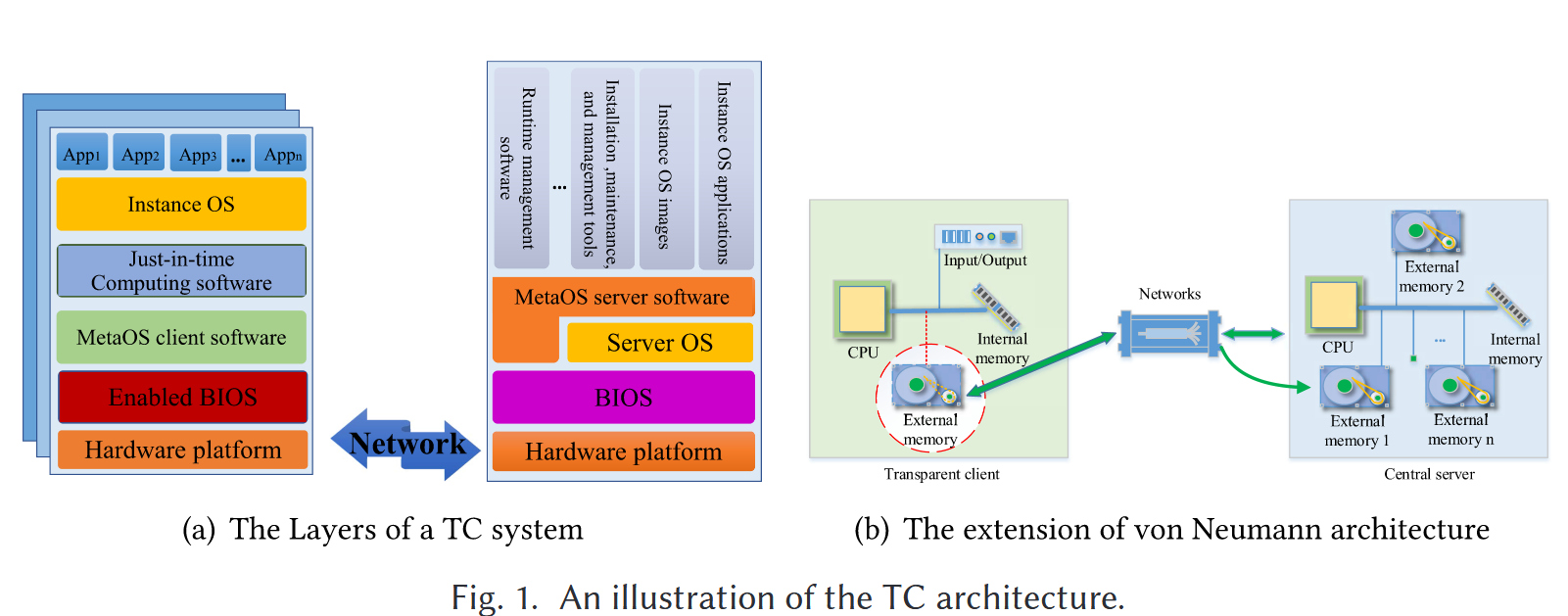
透明计算

透明计算被提出将软件（包括操作系统（OS））与物联网设备的异构硬件解耦[172]。它掩盖了服务供应的细节，并以完全“透明”的方式为用户服务[123]。为此，TC使设备能够通过网络按需选择业务，而无需考虑业务发放的细节，例如软件的升级和管理。一般来说，TC可以具有以下特性：

（1）TC在客户端-服务器模式下工作，将分布在网络上的设备逻辑集成为一个系统。系统根据设备的功能和网络的状况智能地提供服务。服务器端负责对联网客户端进行集中资源管理，提供弹性服务。

（2）为了将软件与异构客户端设备的硬件进行逻辑分离，TC开发了按需服务加载和执行架构，使客户端设备能够通过高速网络从远程服务器动态执行跨平台服务。

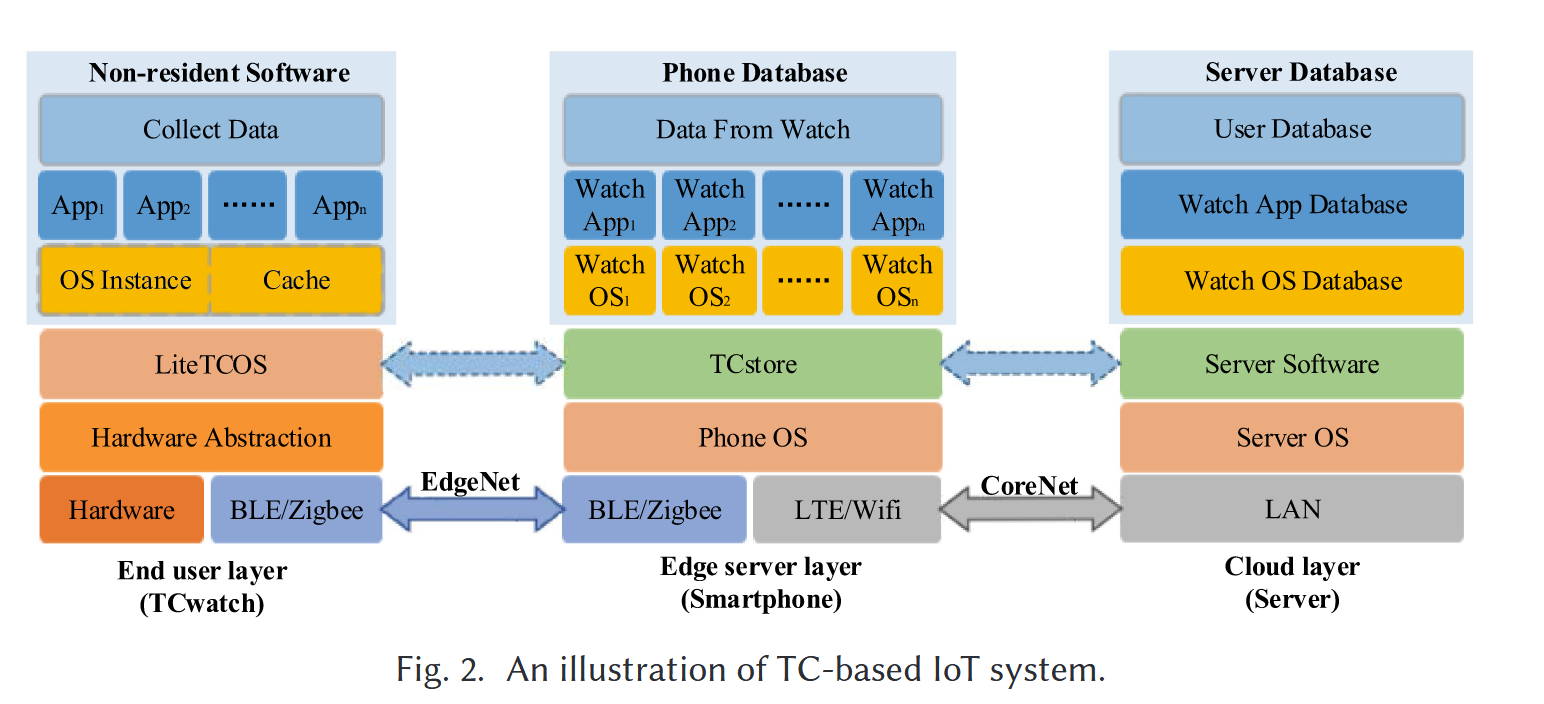
（3）为了充分利用客户端设备中的计算资源，TC使客户端设备能够按需从服务器端获取远程服务，并以块流方式在本地执行。块流执行意味着当用户请求特定服务时，只会在客户端设备上加载和执行与服务相关的必要代码部分，而不是整个软件。因此，可以显著提高能源效率和服务提供的延迟。



外文译文图 1‑44 透明计算框架插图

我们在图 1 中描述了 TC 的典型实现，它在空间和时间领域扩展了经典的冯·诺依曼架构。在 TC 的体系结构中，将单台计算机扩展到联网的计算机/设备。如图 1 所示，服务器存储的系统/服务软件可以动态加载到客户端设备，以便通过 Meta OS 平台执行。Meta OS 建立在底层操作系统之上，旨在屏蔽硬件的异构性，并将接口统一到上层。这样，Meta OS平台就可以初始化和管理各种商品操作系统和应用程序。即时计算层旨在使客户端设备能够通过块流从服务器加载所需程序的指令和用户数据。远程加载后，客户端设备可以及时利用客户端设备的本地资源进行计算。在这种架构下，客户端设备的存储被扩展到其他联网设备（或服务器），I/O中断从本地设备的系统总线重定向到网络[173]。

最近，TC的优势在物联网时代得到了进一步的认可，这迫切需要一个精心设计的解决方案来管理高度异构的硬件基础设施的大量软件，从高端服务器、笔记本电脑和智能手机到低端传感器。Ren等[123]提出了一种可扩展的基于TC的物联网架构，如图2所示，可在网络边缘提供灵活、及时的服务。在他们提出的架构中，附近的边缘服务器充当TC服务器，存储一些常用的App和数据，并及时响应物联网设备的服务请求，而云则充当存储和管理中心，为整个系统提供集中控制。LiteTCOS是物联网设备的MetaOS，负责商用操作系统加载和块流服务执行。此外，He等[67]提出了一种基于TC的新服务模型，称为Block-stream as a Service（BaaS），以实现物联网设备的环境服务计算。它们呈现出清晰的 BaaS 架构，如图 3 所示，其中任何类型的软件（包括操作系统、库、中间件和应用程序）都被划分为一组代码块。当服务器收到服务请求时，部分代码块将动态配置到物联网设备，以便及时高效响应。

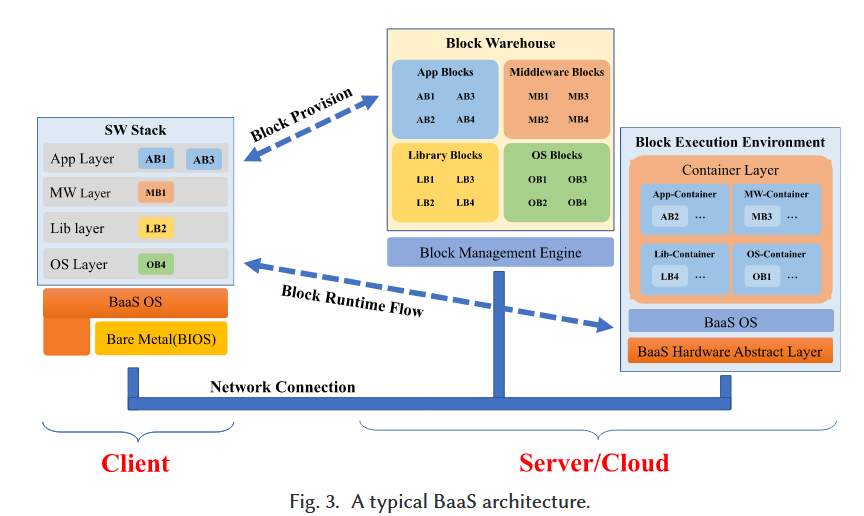


外文译文图 1‑45 基于透明计算的物联网系统插图

移动边缘计算

MEC由欧洲电信标准协会发起，旨在使移动用户附近的云计算服务成为可能[9,27,96]。通过在宏观或微观基站部署MEC服务器，MEC可以通过在网络边缘处理用户请求来改善用户体验，减少延迟和位置感知，并减轻核心网络的负载，如图4[12]所示。MEC与网络功能可视化和软件定义网络一起，被认为是迈向5G时代的关键使能技术[70]。

为了促进供应商、服务提供商和第三方参与者对 MEC 的无缝参与，ETSI 成立了一个行业标准化小组，为标准化和开放的 MEC 环境制定规范。ETSI MEC ISG的成员包括华为、英特尔、诺基亚、沃达丰、NTT DOCOMO等。2014 年发布了第一份介绍性技术白皮书，明确了 MEC 的概念和 MEC 平台的参考架构 [1]。此外，它还讨论了 MEC 中的关键使能技术和挑战。在2015年至2017年初期间，该小组记录了MEC的几个规范，从MEC的术语、服务场景到技术要求[2,4,5,9]。尽管该小组声称MEC在早期主要专注于将云计算技术集成到蜂窝网络中，但现在MEC的名称已更改为多接入边缘计算，这可以反映出非蜂窝运营商日益增长的兴趣[3]。

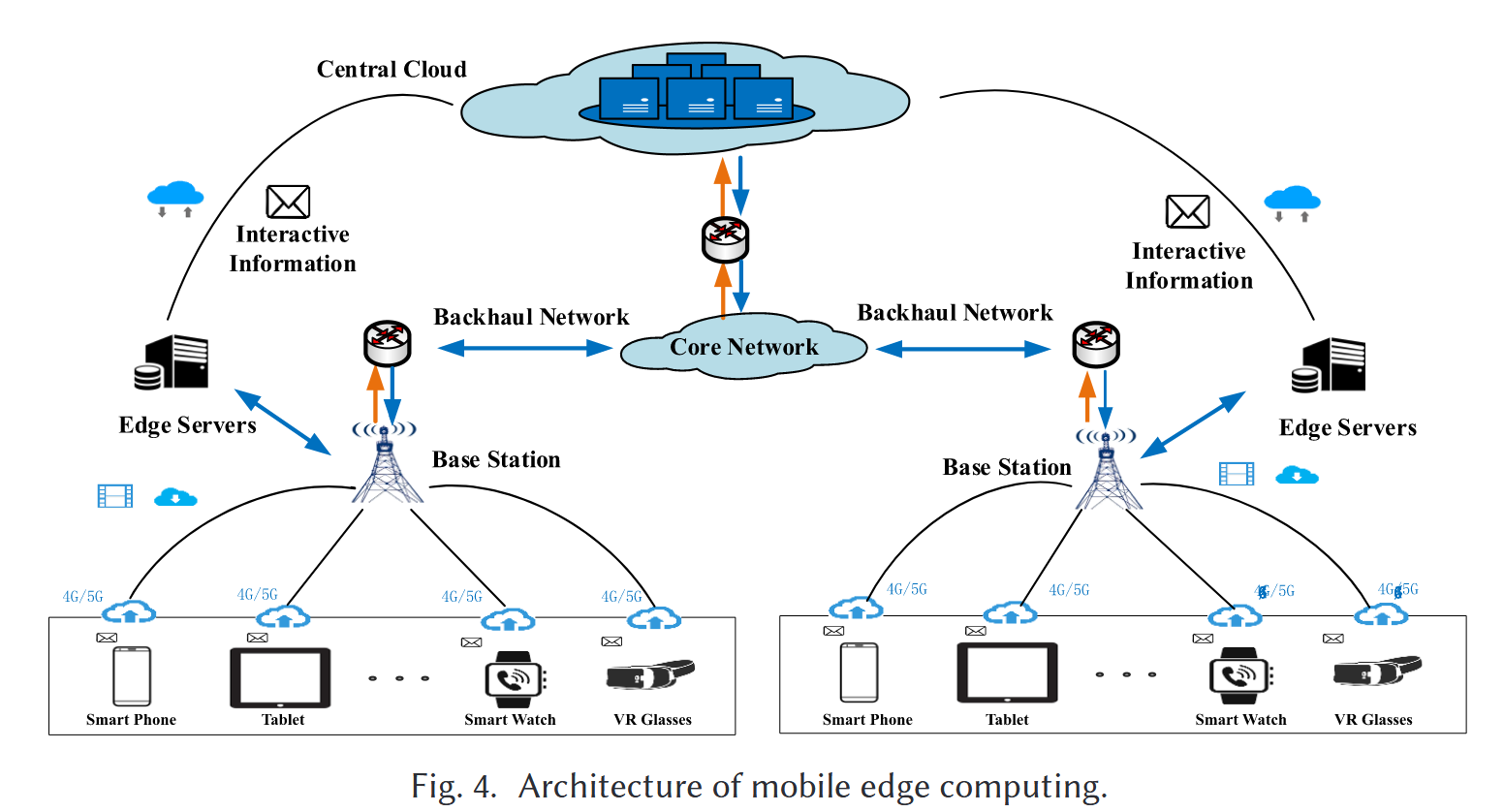


外文译文图 1‑46 一种典型的块流即服务架构

近年来，学术界和工业界都提出了许多解决方案来增强MEC的性能，如建模、多用户资源分配、系统实现等。一些调查从不同角度关注MEC的进展[12]。参考文献[12,96,113,139,150]中讨论了一些实时MEC应用场景。他们还从不同的角度讨论了 MEC 的分类，例如特征、参与者、接入技术、应用程序、关键推动因素等。参考文献[139]中介绍了MEC的基本关键使能技术。它通过考虑系统性能和 MEC 平台来讨论 MEC 编排，并阐明了不同的编排部署。此外，作者还介绍了MEC的架构和典型部署场景。参考文献[96]的作者调查了最先进的MEC研究，重点关注通信和计算资源的联合管理。参考文献[113]中的调查讨论了MEC中的关键用例，即计算卸载。参考文献[150]的作者调查了MEC的关键推动因素，包括云计算、SDN/NFV和智能设备。此外，他们还讨论了移动边缘网络的一些关键技术，包括云技术、SDN/NFV以及智能设备。

雾计算

雾计算范式最早是由思科于 2012 年创造的 [22]。从某种意义上说，雾计算与MEC的概念相似。然而，它也是一种新颖的网络计算架构，提供了网络边缘的计算能力[27]。雾计算最初是为物联网环境而提出的，除了无线接入和移动支持外，还需要位置感知和及时响应。此外，雾计算采用n层架构，提供更灵活的服务，凸显了数据路由路径上的所有网络设备都可以为终端设备提供数据计算和存储服务。如图 5 所示，雾层在云和 IoT 设备之间物理放置，以实现计算、存储和网络资源池。雾层由大量异构纳米服务器组成，包括边缘路由器、设置盒等专用设备，以及智能手机、高端传感器和车辆等临时设备[21]。

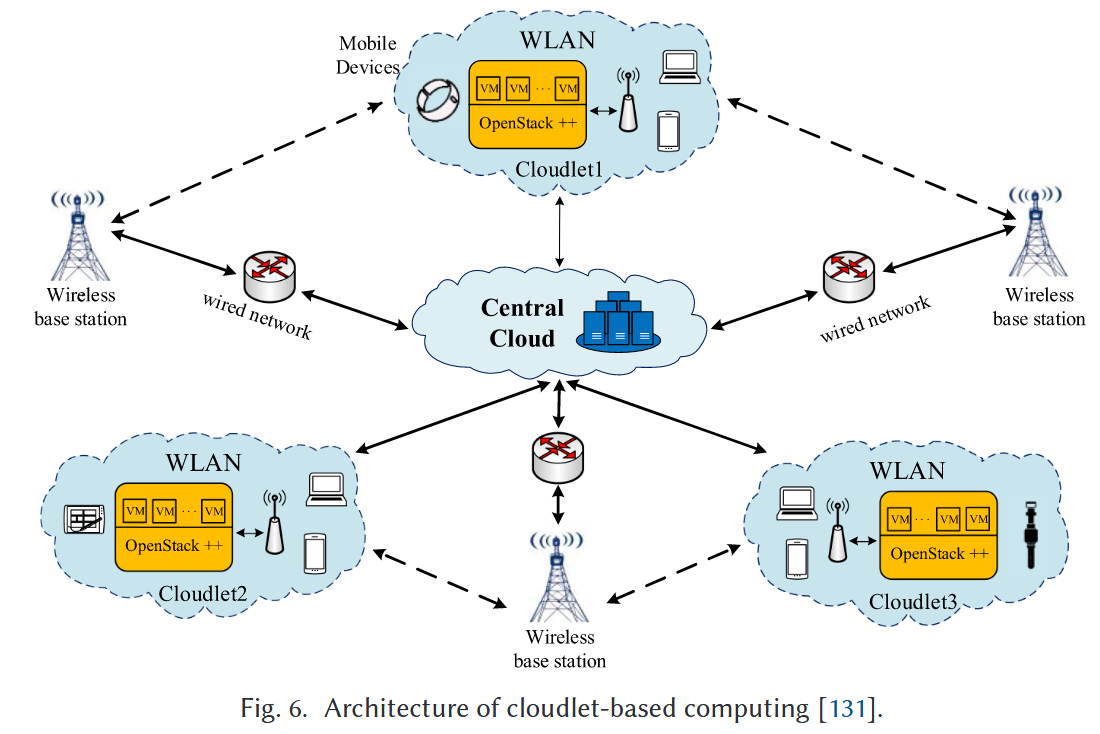


外文译文图 1‑47 移动边缘计算架构

许多文献以不同的重点调查了雾计算的研究问题[27,144]。Vaquero和Rodero-Merino[144]概述了雾计算的使能技术和未来发展。参考文献[161]中的调查概述了雾计算的定义和一些典型的应用场景，并清楚地提出了雾计算系统设计和实现的挑战。参考文献[160]的作者声称，雾计算有望成为许多有前途和具有挑战性的物联网场景的天然平台。参考文献[47]的作者描述了雾计算在物联网中的优势，并介绍了适用于雾计算的各种应用场景。参考文献[104]的作者提出了10个问题，并给出了相应的答案，以证明雾计算与其他现有计算范式相比的优势。最近，参考文献[27]的作者提出了一套简明的雾计算评估标准。

Cloudlet

2009年，卡内基梅隆大学（Carnegie Mellon University）的一个研究小组首次提出了云的概念[130]。术语“cloudlet”是指放置在移动用户附近的微型数据中心，例如在咖啡店或教室中。Cloudlet 背后的关键动机是提高移动应用程序的交互性能，尤其是那些具有严格的对端到端延迟和抖动的要求。此类应用要求响应延迟在毫秒级，这不太可能通过 因特网。为了填补这一空白，cloudlets的邻近性使服务器能够为移动用户提供高度响应的云服务，从而补充了三层云层次结构，即移动用户-cloudlet-云，如图 6 所示，以提供及时的服务， 这种层次结构通过在将私有数据发布到云之前对其进行变性来加强其所有者的隐私[131]。已经确定了 cloudlets 的几个典型用例，例如辅助驾驶支持、运动训练助手等。对端到端延迟和抖动的要求。



外文译文图 1‑48 基于cloudlet的计算架构

在Cloudlet发明后，CMU的研究小组发表了一系列工作，主要集中在两个主题上，即确定Cloudlet的有价值的应用[129]和设计虚拟机以支持用户移动性[66]。在参考文献[129]中，作者指出，实时认知辅助，例如面部和语音识别，可以成为Cloudlet的“杀手级应用程序”，它需要及时响应以不显眼地引导用户的注意力。参考文献[66]基于一个名为OpenStack的云计算开源生态系统，提出了VM覆盖层的设计，以实现移动环境中的Cloudlet发现和切换，同时最大限度地减少Cloudlet之间的数据交换。由于Cloudlet带来的潜在商机，CMU与英特尔、诺基亚、Crown Castle、沃达丰、德国电信等多家领先的工业公司组成了开放边缘计算计划[108]，旨在开发围绕Cloudlet的关键技术并进行用户验收测试。

一些现有的工作不是在用户附近部署专用的计算设施，而是利用移动设备的空闲计算资源来执行计算服务，这被称为“Ad hoc cloudlet”[36]。在自组织云中，移动设备通过短距离无线电通信技术相互连接，为计算能力有限的物联网设备（例如 RFID 传感器）执行数据分析。尽管利用移动设备的空闲资源需要最少的设施部署和维护工作，但移动设备的移动性和动态本地负载对在自组织云中实现高效计算和服务供应提出了重大挑战。为了解决这个问题，一些工作设计随机优化方法来提高特设云对系统动力学的适应性[119]。

外文译文表1-1 多种计算计算范式的异同对比



讨论异同

上述新兴计算范式的共同特征是降低硬件实现的复杂性和成本，减少延迟，以及提高体验质量和网络效率。但是，有一些细微的差异需要澄清，如表1所示。上述新兴计算范式的共同特征是降低硬件实现的复杂性和成本，减少延迟，以及提高体验质量和网络效率。但是，有一些细微的差异需要澄清，如表1所示。

首先，TC范式的动机是解决物联网网络中各种终端用户的异构性，并支持不同的操作系统和跨平台应用软件。因此，降低了硬件实现的复杂性和成本，并增强了应用程序的兼容性。相反，引入 fog、MEC 和 cloudlet 范式的动机是在终端设备附近处理数据，以减少响应延迟并改善用户体验和网络效率。请注意，Cloudlet 主要使用虚拟机进行虚拟化，而 MEC 和 fog 则考虑容器。

其次，对于强调提供面向服务的计算解决方案的TC范式，通过从边缘/云服务器获取操作系统和应用软件，在终端用户/边缘节点执行计算。此外，通过块流式服务执行方法，轻量级终端只能执行必要的代码，而不是应用程序的整个程序，以提高服务计算的能效。对于 Fog、MEC 和 cloudlet 范例，通常，终端设备和边缘服务器都会安装操作系统和指定的应用程序软件。边缘服务器可以为终端设备的服务请求提供及时的数据处理，而终端设备可以通过将一些计算密集型任务卸载到边缘服务器或云服务器来提高其能效。

第三，从安全的角度来看，这些计算范式有不同的研究重点。TC旨在利用终端设备和边缘服务器的计算能力，为物联网设备提供安全服务。安全机制一般部署在MetaOS层（终端设备端和边缘服务器端）以检测恶意软件和OS级别的攻击，因为MetaOS在商用操作系统下工作，并且具有更高的优先级，可以通过偶尔挂断上层操作系统来直接检查硬件中运行的程序代码。但对于 fog、MEC 和 cloudlet 范式，研究人员主要关注保护边缘服务器的计算环境，并将安全负担（如加密和身份验证）从终端设备转移到边缘服务器。因此，寻找利用终端设备、边缘服务器和云的编排来设计复杂安全解决方案的方法成为所有网络计算范式的热门研究趋势[124]。

评定标准

上述新兴计算范式的全部优势被利用在一些共同的性能标准上，用于研究这些网络的架构和优化算法[27]。

第一个标准（C1）是需要支持资源的异质性。在新兴的计算范式中，网络边缘的不同接入节点和终端用户在计算和存储能力方面表现出很强的异构性。因此，在设计计算架构和优化算法时，需要考虑各种接入节点的异构性。例如，透明计算的边缘服务器需要提供各种操作系统和应用软件来支持各种终端用户，这些终端用户通常具有不同的硬件配置。为了适应异构最终用户的异构服务，英特尔提供了基于 HTML5 的解决方案来实现 TC。与之前的版本相比，HTML 5 拥有了一些新的特性，例如丰富的语义信息和多线程支持。尽管 HTML5 具有跨平台支持和低开发成本的优势，但由于浏览器和 Web 安全之间的差异，挑战仍然存在。TC范式支持在Meta OS层实现浏览器引擎，以屏蔽浏览器之间的差异[173]。

第二个标准（C2）是新兴计算范式中严格的QoS要求。在未来的通信网络中，低时延、高数据传输速率、高频谱、能效等QoS要求是评估新型通信计算技术性能的关键性能指标。引入新兴计算范式的主要动机是减少数据传输延迟，同时提高数据传输速率[49]。例如，车对车通信或智能车载通信网络的传输时延需要小于10ms，虚拟现实的传输速率需要大于千兆位。因此，在网络架构的设计和新兴计算范式的优化算法中，需要考虑这些要求。

第三个标准 （C3） 是对弹性可伸缩性的需求。新兴计算范式的主要目标是在物联网场景中为各种终端用户实现低延迟、高数据速率和高能效。计算范式有望为数百万甚至更多的最终用户和应用程序提供服务。因此，他们需要具备为各种终端提供弹性按需服务的能力。这也意味着新兴计算范式的架构和相应的算法需要具备适应网络规模变化的能力。

第四个标准（C4）是计算范式是否可以支持移动性。例如，在车辆网络中，车辆的运动会显着影响通信系统的性能。特别是，对于新兴的计算应用，需要仔细考虑通道的快速变化。

第五个标准（C5）是联合和互操作性的需要。在新兴的计算范式中，边缘服务器具有广泛的地理分布部署，并由不同的所有者提供。不同提供商之间的合作可以促进各种服务的使用并改善最终用户的体验。为此，它需要精心设计的数据和控制接口，以便在不同级别的提供商和架构上实现互操作性。例如，在TC范式中，Meta OS负责管理硬件和软件资源，包括操作系统和各种应用软件（App），并为用户提供安全可靠的计算服务[172]。

1. 来自王思野老师的《信息网络建模与仿真》课程课件 [↑](#footnote-ref-1)