临沂大学

LINYI UNIVERSITY

硕士专业学位论文

云雾协作环境下任务动态分配方法研究

Research on Task Dynamic Assignment Method in Cloud-Fog Collaborative Environment

学　　号：2020120119
姓　　名：霍东岳
专业学位类别：电子信息
专业学位领域：计算机技术
学　　院：信息科学与工程学院
指 导 教 师：康与云 教授
行 业 导 师：刘增明 高级工程师

论文提交时间：2023 年 05 月

临沂大学
LINYI UNIVERSITY

硕士专业学位论文

云雾协作环境下任务动态分配方法研究

学　　号：2020120119
姓　　名：霍东岳
专业学位类别：电子信息
专业学位领域：计算机技术
学　　院：信息科学与工程学院
指 导 教 师：康与云 教授
合 作 导 师：刘增明 高级工程师

论文提交时间：2023 年 05 月

Research on Task Dynamic Assignment Method in Cloud-Fog Collaborative Environment

A Thesis Submitted to

Linyi University

For the Professional Degree of Master of Engineering

By

Huo Dongyue

Supervised by

Prof. Kang Yuyun

And

Senior Engineer Liu Zengming

School of Information Science and Engineering

Linyi University

May, 2023

摘 要

随着物联网技术的高速发展，终端设备产生的数据呈现爆发式的增长，这为云计算带来了巨大的压力。雾计算的出现减轻了云计算的压力，同时也弥补了终端设备在存储资源、计算能力等方面的不足。如何将所有的计算资源利用起来以提高用户的体验质量是雾计算中的重要挑战之一。同时，终端设备最本质的特征就是移动性，而雾节点的覆盖范围具有一定的局限性，设备的每一次移动都可能会导致所连接到的雾节点群的变化，因此，如何在该情况下将终端设备的任务合理高效地分配到相应的计算节点以保证用户的体验质量是一个值得被研究的问题。针对以上两个问题，本文主要工作如下：

（1）研究了单雾节点、多终端设备场景下以时延和能耗为优化目标的任务分配方法。首先将以时延和能耗联合优化的任务分配问题建立为 0-1 整数规划模型。其次提出二进制天牛须搜索算法（Binary Beetle Antennae Search Algorithm，BSBAS），在天牛须搜索算法的基础上加入 Sigmoid 函数，并以 0.5 为界，将天牛须更新后的位置映射到 0 或者 1。最后，以粒子群优化算法、轮询算法、随机算法、全部卸载方法以及全部在终端设备执行方法为基线算法进行仿真实验。仿真结果表明，无论是终端设备的数量发生变化还是雾节点及终端设备的计算能力发生变化，所提算法在任务动态分配问题上都具有最良好的表现。同时，实验还验证了权重对时延和能耗的影响，结果表明，若权重偏向时延，则相较于能耗来说，算法对时延的优化力度更大，反之亦然。

（2）研究了多雾节点、多终端设备场景下以任务的平均时延为优化目标的任务分配方法。首先将软件定义网络（Software Defined Networking，SDN）的优势应用到云-雾-端架构中，通过提供对终端设备、雾节点及云服务器的资源等信息的集中控制的解决方案来使任务动态分配更加合理高效。其次提出了基于 K-means 算法的任务优先级划分方法先对终端设备的任务进行优先级划分，然后提出了基于改进的量子鸽群优化算法（Improved Quantum Pigeon-inspired Optimization，DQPIO）的任务分配方法对划分好优先级的任务集进行任务分配。最后，在不同的场景下对所提的方法进行了仿真实验，结果表明，与轮询算法、天牛须搜索算法、鸽群优化算法以及随机生成算法相比，所提的方法在终端设备的数量、任务的数据量、雾节点的数量等方面均取得了最好的表现。

（3）研究了考虑终端设备移动性的场景下以最小化时延为优化目标的任务分配方法。首先将终端设备产生的任务构建为一个有向无环图，并介绍了以最小化时延为目标的工作流任务分配优化模型。其次为了在终端设备移动过程中保持任务的连续性，当终端设备移

动到敏感区域时，根据目前所有计算节点的计算资源和任务的迁移成本等指标设计了任务迁移决策机制，同时还介绍了基于深度 Q 网络（Deep Q Network，DQN）算法的任务分配方法。最后通过仿真实验，以 Q-learning 算法为基线算法在任务的数量、任务的数据量等方面验证了所提方法的有效性。

关键词：云雾协作；任务分配；软件定义网络；终端移动性

Abstract

With the rapid development of Internet of Things technology, the data generated by terminal devices has shown explosive growth, which has brought enormous pressure to cloud computing. The emergence of fog computing has relieved the pressure of cloud computing, and at the same time made up for the shortage of terminal equipment in terms of storage resources and computing power. How to utilize all computing resources to improve user experience quality is one of the important challenges in fog computing. At the same time, the most essential feature of terminal equipment is mobility, and the coverage of fog nodes has certain limitations. Every movement of a device may cause changes in the group of fog nodes it is connected to. Therefore, how to It is a problem worthy of research to allocate the tasks of the terminal equipment to the corresponding computing nodes reasonably and efficiently to ensure the user's quality of experience. In view of the above two problems, the main work of this paper is as follows:

(1) The task allocation method with the optimization goal of delay and energy consumption in the scenario of single fog node and multi-terminal equipment is studied. Firstly, the task allocation problem based on joint optimization of delay and energy consumption is established as a 0-1 integer programming model. Secondly, a binary beetle antennae search algorithm (Binary Beetle Antennae Search Algorithm, BSBAS) is proposed, and the Sigmoid function is added on the basis of the beetle antennae search algorithm, and the updated position of the beetle antennae is mapped to 0 or 1 with 0.5 as the boundary. Finally, simulation experiments are carried out with polling algorithm, random algorithm, all offloading methods and all execution methods on terminal devices as the baseline algorithms. The simulation results show that whether the number of terminal devices changes or the computing power of fog nodes and terminal devices changes, the proposed algorithm has the best performance in the task dynamic allocation problem. At the same time, the experiment also verified the impact of the weight on the delay and energy consumption. If the weight is biased towards the delay, the algorithm will optimize the delay more than the energy consumption, and vice versa.

(2) The task allocation method with the average delay of the task as the optimization goal in the scene of multi-fog nodes and multi-terminal devices is studied. Firstly, the advantages of software-defined network (Software Defined Networking, SDN) are applied to the cloud-fog-device architecture, and the dynamic allocation of tasks is made more reasonable and efficient by providing a solution for centralized control of information such as terminal

devices, fog nodes, and cloud server resources. Secondly, a task prioritization method based on the K-means algorithm is proposed. First, the tasks of the terminal equipment are prioritized, and then a task allocation method based on the improved quantum pigeon-inspired optimization algorithm (Improved Quantum Pigeon-inspired Optimization, DQPIO) is proposed. Assign tasks to prioritized task sets. Finally, the proposed method is simulated in different scenarios, and the results show that, compared with the polling algorithm, beetle whisker search algorithm, pigeon group optimization algorithm and random generation algorithm, the proposed method is more effective than the terminal equipment. The number, data volume of tasks, and the number of fog nodes have all achieved the best performance.

(3) The task allocation problem with the optimization goal of minimizing the delay is studied in the scenario of considering the mobility of the terminal equipment. Firstly, the tasks generated by the terminal equipment are constructed as a directed acyclic graph, and an optimization model of workflow task allocation with the goal of minimizing time delay is introduced. Secondly, in order to maintain the continuity of the task during the mobile process of the terminal equipment, when the terminal equipment moves to the sensitive area, the task migration decision-making mechanism is designed according to the current computing resources of all computing nodes and the migration cost of the task. A task allocation method for the Deep Q Network (Deep Q Network, DQN) algorithm. Finally, through simulation experiments, the effectiveness of the proposed method is verified in terms of the number of tasks and the amount of data of the tasks using the Q-learning algorithm as the baseline algorithm.

Key words: cloud-fog collaboration; task allocation; software-defined network; terminal mobility

目 录

1 绪论.....1
1.1 研究背景及意义.....1
1.2 国内外研究现状.....2
1.3 论文主要研究内容及组织结构.....4
2 相关概念及技术概述.....6
2.1 云雾协作环境概述.....6
2.2 任务分配技术概述.....6
2.2.1 任务分配概述.....6
2.2.2 任务分配常用算法.....9
3 单雾节点多终端场景下的任务动态分配方法.....12
3.1 引言.....12
3.2 系统模型.....12
3.2.2 云雾协作计算模型.....13
3.2.3 优化目标.....14
3.3 基于改进的天牛须搜索算法的任务分配方法.....14
3.3.1 天牛须搜索算法.....14
3.3.2 改进的天牛须搜索算法.....15
3.4 仿真实验结果与分析.....17
3.4.1 实验参数设置.....17
3.4.2 实验结果分析.....18
3.5 本章小结.....20
4 多雾节点多终端场景下的任务动态分配方法.....21
4.1 引言.....21
4.2 系统模型.....21
4.2.1 终端计算模型.....23
4.2.2 雾节点计算模型.....24
4.2.3 云服务器计算模型.....24
4.2.4 优化目标.....25
4.3 任务优先级划分与任务动态分配方法.....26
4.3.1 基于 K-means 算法的任务优先级划分方法.....26
4.3.2 基于 DQPIO 算法的任务动态分配方法.....27
4.4 仿真实验结果与分析.....32
4.4.1 实验参数设置.....33
4.4.2 实验结果分析.....33

4.5 本章小结.....38
5 考虑终端移动性的任务动态分配方法.....39
5.1 引言.....39
5.2 系统模型.....39
5.2.1 工作流任务模型.....39
5.2.2 雾节点信号覆盖范围模型.....40
5.2.3 终端移动路径模型.....41
5.2.4 考虑终端移动性的工作流任务分配模型.....41
5.3 任务迁移决策机制与任务动态分配方法.....42
5.3.1 任务迁移决策机制.....42
5.3.2 基于 DQN 算法的任务动态分配方法.....43
5.4 仿真实验结果与分析.....44
5.4.1 实验参数设置.....44
5.4.2 实验结果分析.....44
5.5 本章小结.....47
6 总结与展望.....48
6.1 工作总结.....48
6.2 工作展望.....48
参考文献.....50

1 绪论

1.1 研究背景及意义

在过去的几年里，随着物联网技术的不断提升，终端设备的数量持续增加。在使用终端设备的过程中会产生大量的数据，为了保证这些终端设备的正常使用，这些数据必须被运输、储存然后尽快处理。但终端设备的计算及存储能力十分有限，无法做到高效处理这些数据。在这种背景下，出现了云计算的概念，云服务器具有强大的数据处理和存储能力，把数据传输到云端可以提高数据处理的速度。因此云计算为用户提供了更便捷、更可靠的服务。

近几年，随着对可穿戴设备、智能家居、智慧城市、医疗设备等物联网应用的关注度的持续上升，终端设备产生的数据呈现爆发式的增长，这给集中式云计算数据中心带来了巨大的压力。如果继续把这些分布式的数据集中到一个中心位置进行处理和储存无论是在经济上还是在技术上都是行不通的[1]。另外，在某些应用中，系统需要快速响应和移动性支持，例如智慧医疗、智慧交通以及智能电网中的应急响应[2-4]等等。但是，仅仅依靠云服务器有时很难满足用户的需求。为了克服云计算所带来的这些高延迟、网络带宽拥堵以及安全问题等挑战，思科[5]提出了一种分布式计算技术“雾计算”，它作为云服务器和终端设备之间的中间层以满足如智能医疗、智能家居等延迟敏感的应用的计算需求[6-8]。因此，雾计算作为一种新兴的分布式计算模式引起了诸多学者的注意[9, 10]。

终端设备的电池电量比较低，计算和存储容量也十分有限，尤其是在运行在线游戏、虚拟现实等一些实时性应用时难以满足用户体验质量。为了实现这些应用的正常工作，必须将终端设备的任务有效地卸载到电量充足、计算能力以及存储容量更加充裕的设备上。雾计算作为云计算的延伸使得云服务器更加接近终端设备从而减少延迟、网络流量、能耗以及运营成本等。此外，相较于云计算来说，雾计算具有支持用户移动性、实时交互、低延迟等优点[11, 12]。但是，与计算能力强大的云服务器相比，雾节点的计算能力以及存储容量是十分有限的[13]。因此，雾计算更适合处理实时的小数据的任务以及对时延要求敏感的任务，而云计算更适合处理计算密集的任务。在复杂多变的现实环境下，无论是只使用云计算模式还是只使用雾计算模式都很难满足所有任务的时延要求。此时，在云雾端协同的架构下对终端设备的任务进行合理地分配是最高效的。

云雾协作环境下的任务分配实际上就是为每个终端设备产生的任务寻找合适的计算节点的过程。任务的响应时延、设备的能耗以及成本费用等等都是非常重要的指标。对于终端设备的使用者来说，当其使用一些应用时肯定希望任务的响应时延越短越好。任务的响

应时延越短，使用者的体验质量就越好。对于能耗方面来说，如果某个终端设备的任务数据量太小，将其卸载至在雾节点或者云服务器上执行反而会由于通信而增加能耗。能耗的增加有悖于国家所倡导的节能减排的目标。此外，由于经济因素，任务执行的成本费用也是常被优化的目标之一，如何在保证用户的体验质量的前提下将任务执行的成本费用降到最低是作为企业来讲十分看中的。无论以哪一个作为优化目标都需要一个合理的分配策略。以时延为例，对于一个终端设备的任务来说，在设备本身不具备计算以及储存的情况下则需要将其分配到周边的雾节点或者云服务器上执行。若将其分配到周边的雾节点上，则可以获得较低的传输时延以使得任务可以尽快地被雾节点所执行。若将任务分配到云服务器上执行，由于云服务器具有强大的计算能力与储存容量，那么任务的执行时延则会被降到最低。对于这个任务来说，其总时延是由传输时延和执行时延之和所决定的。如果此时有若干个任务需要被执行，那么应该如何权衡这些任务的传输时延与执行时延才可以使得任务的总时延最低是一个需要被解决的重要问题。因此，如何依据不同的优化目标对多个终端设备所产生的多个任务做出合理的分配策略是一个值得被研究的问题。

综上所述，5G 的逐渐普及使得世界进入万物互联时代。为了顺应时代的发展，同时提高用户对各种物联网应用的体验质量，将终端设备、雾计算与云计算相结合，融合三者的优点可以更好地依照优化目标对设备的任务进行合理地分配，这体现出了云雾端协作的架构的重要应用价值。同时，如何在云雾端协作的环境下对物联网设备的任务做出合理的分配策略也是一项具有重要意义和发展前景的研究。

1.2 国内外研究现状

目前，研究人员在雾计算环境下或者云雾协作环境下在任务分配方面做了大量的工作，其所采用的优化目标多为总时延、能耗以及两者的联合优化。

在总时延优化方面。Amit 等[14]提出了一种智能蚁群优化任务分配的算法，并且与轮询算法、节流、MPSO 和 BLA 算法相比，在任务分配时间上分别缩短了 12.88%、6.98%、5.91%和 3.53%。Azizi 等[15]首先用数学方式描述了任务分配模型，其次提出了优先级感知贪婪算法（PSG）和具有多启动过程的 PSG（PSG-M）算法将任务有效地分配到合适的雾节点上。实验结果表明，与其他算法相比，所提出的算法满足最后期限要求的任务百分比提高了 1.35 倍，并将总期限的违规时间降低了 97.6%。谭等[16]基于多臂老虎机理论提出了一种适用于任务在线分配的算法，可以针对终端设备产生的任务以及每个计算节点的资源利用率以及内存占用情况等参数的实际情况做出实时的任务分配策略，经过大量的仿真实验后，证明了所提算法在优化任务分配时延方面得到了十分理想的结果。由于单纯的雾环

境仍未将所有的资源利用起来共同解决终端设备的任务的分配问题。因此，云雾协作的架构被很多研究者用来解决终端设备的任务分配问题。Syed 等[17]采用了一种混合火烈鸟搜索算法与遗传算法的新模型来实现更好的任务分配。并且利用了 7 个基础的测试函数测试了该算法的性能。同时，该算法在任务完成期限百分比、完成时间和成本方面与蚁群算法、粒子群算法、遗传算法等相比具有更好的效果。钟等[18]将工业物联网下的任务分配问题看作为 0-1 整数规划问题，同时选择采用混合整数线性规划算法来解决任务分配问题。通过仿真实验表明，与局部卸载等方法相比，所提出的针对任务卸载的方案在时延上降低了 4%。与模拟退火算法相比，所提出的方法在任务总时延上降低了 10%。

在能耗优化方面。Wang 等[19]提出了一种迭代系统优化算法用于解决联合任务分配和功率分配的问题。在经过大量的实验验证后，与基准方案相比，所提出的方案可以显著降低雾节点的总能耗。研究者们也研究了以能耗为优化目标在云雾协作环境下终端设备的任务的合理分配与卸载问题。葛等[20]提出了一种任务迁移候选目的节点集生成算法，该算法用于解决所有终端设备的任务的总能耗问题。同时，为了保证所有的雾节点的能量消耗是均衡的，作者又提出了目的节点公平选择算法。基于这两个算法，作者进行了仿真实验，实验结果表明使用这两个算法可以在最优化能耗的前提下使得每个雾节点之间的能耗消耗均衡，同时使得平均雾节点的存活率上升了 10.9%。

在二者联合优化方面，Ghanavati 等[21]提出了一种蚂蚁交配优化算法，其目标是降低雾计算平台的总系统完成时间和能耗。Movahedi 等[22]首先提出了一个在雾环境下的处理任务分配的架构，其次采用混沌鲸鱼优化算法以优化时延和能耗两个用户体验质量参数为优化目标来解决雾环境下的任务分配问题。通过与人工蜂群算法、粒子群优化算法和遗传算法对比后，有效地证明了所提出算法的效率。张等[23]将任务分配问题建立为 0-1 规划模型，然后选择使用分支定界算法来解决任务卸载在时延和能耗的相互制约的问题。经过仿真实验表明，使用分支定界算法解决该问题可以有效地降低终端设备产生的任务的时延和能耗。在云雾协作环境下，联合优化时延和能耗这两个用户体验质量参数的研究逐渐成为热点。Fairouz 等[24]针对云雾协作环境下以成本、时间和能源消耗为共同目标的任务分配问题，提出了一种基于非优势排序的多目标粒子群优化算法。所提出的算法在最后可以为用户输出一组帕累托最优解以供用户进行选择。Parvinder 等[25]以最小化能耗和时延为目标对任务进行分配。为解决这一问题采用飞蛾火焰算法进行求解最优分配策略。该算法与作业调度算法、多层雾计算算法和计算装载游戏算法相比能耗分别降低 22%、25%和 29%。吴等[26]首先将均衡时延和能耗这两个用户体验质量参数的联合优化问题分解为三个不同的子问题，

然后采用凸优化条件、低复杂度的上行功率分配方法、分布式博弈论方法这三种方法分别用来解决分解后的三个子问题。基于这一整套方案，作者进行了大量的仿真实验，仿真结果表明，所提出的方案与其他的方案相比在时延和能耗上均有更加优秀的表现。

1.3 论文主要研究内容及组织结构

当前大部分研究集中在解决单雾环境下或者单云环境下的任务分配问题，这使得在计算资源利用率方面具有一定的局限性。针对这一问题，云雾协作的架构被用于构建任务分配这一问题。针对如何寻找最佳任务分配策略这一问题，目前常用的算法主要集中在元启发式算法和深度强化学习算法中。对分配方法进行不断优化以给出系统性能最大化的分配策略具有重大意义。本文主要工作如下：

（1）在单雾节点、多终端设备的任务分配场景下，研究了以时延和能耗联合优化的任务分配问题。将以时延和能耗的联合优化的系统模型转化为 0-1 整数规划模型，并采用改进后的离散的二进制的天牛须算法对任务进行合理分配。

（2）在多雾节点、多终端设备的任务分配场景下，将 SDN 与云-雾-端架构相结合，以平均时延为优化目标，首先使用 K-means 聚类算法对终端设备产生的任务进行优先级划分，然后使用改进过的混合量子鸽群优化算法对任务进行合理分配。

（3）在考虑终端设备移动性的场景下，研究了以最小化时延为优化目标的问题。首先，将终端设备产生的任务流描述为一个有向无环图，随后构建了以最小化时延为目标的工作流任务模型。其次，当终端设备不断移动时将任务的迁移决策机制与 DQN 相结合以保证任务连续性的同时使任务的总时延最低。

本文具体的章节结构安排如下：

第一章为绪论，描述了云雾协作环境下任务动态分配这一课题的背景，总结了进行课题研究的意义，同时探讨了在优化目标以及调度算法这两个方面的国内外研究现状。

第二章介绍了云雾协作环境的发展，包括云雾协作环境的优势及其架构。同时，按照不同的分类分别介绍了任务卸载的概念以及任务卸载常用的算法。

第三章在多个终端设备、单雾节点的任务分配场景下，研究了以时延和能耗联合优化的任务分配问题。首先介绍了以时延和能耗联合优化的任务分配的系统模型并将其转化为 0-1 整数规划模型，其次介绍了改进的连续型天牛须搜索算法使其适用于离散的二进制问题。最后，可视化了仿真实验的结果，证明了当前使用的算法的有效性和适用性。

第四章在多个终端设备、多雾节点的任务分配场景下，研究了以平均时延为优化目标的任务分配问题。首先介绍了 SDN 技术与云-雾-端架构融合的步骤。其次阐述了基于 K-

means 算法的任务优先级划分的流程，将所有终端设备产生的所有任务进行优先级划分，随后介绍了改进的混合量子鸽群优化算法以平均时延为优化目标对每个优先级的任务进行合理的分配。最后，在不同的实验环境下对所提方法进行仿真实验，与其他四种基线算法相比，所提方法在优化最低时延方面具有更良好的表现。

第五章研究了考虑终端移动性的任务动态分配问题。首先，构建了以最小化延迟为目标的工作流任务分配优化模型。其次，采用 DQN 算法合理分配任务，当终端设备移动时，启动任务迁移决策机制来进行任务迁移。最后，通过仿真实验验证了该方法的有效性，结果表明该方法有效地降低了时延。

第六章针对该课题已做的工作进行了总结，并且对未来的工作进行了展望。

2 相关概念及技术概述

2.1 云雾协作环境概述

随着互联网的发展，大量的数据和应用程序开始转移至互联网，传统的计算模式已经无法满足这些数据和应用程序的存储和处理需求，因此云计算应运而生。云计算是一种提供计算服务的新模式，其通过提供可扩展性、灵活性、高可用行和弹性计算能力等优势解决了在传统计算模式下的资源利用率低、业务扩展难、成本高昂等问题。

近些年来，随着物联网设备的不断增加，海量数据时代已经到来，为了解决大数据环境下的数据管理、处理和传输等带来的延迟和安全等问题，思科[5]在 2011 年提出了一种分布式计算技术“雾计算”以解决传统云计算模式下物联网应用的问题。雾计算的引入使得几乎所有的节点和设备都可以相互通信和交互，这支撑了物联网的概念，同时也在很大程度上优化了关键的服务质量参数如带宽、延迟、存储、响应时间等等。雾计算减轻了传统云计算数据中心的负担[27, 28]，并在分散计算概念的基础上工作，提高了计算速度。雾计算的发展为雾环境下的存储和处理数据提供了多种选择[29]。在工业中，一些应用程序需要非常快地处理数据。因此，雾计算在终端设备之间建立了一个低延迟的网络接口，以减少响应时间，使得任务数据的通信和处理更快[30]。雾计算框架支持终端设备在物联网节点附近监视、分析、处理、测量、分发和控制计算、提供存储、通信和决策[31]。目前，一些企业在交通运输、智能电网、智慧城市、智慧农业和医疗保健等领域都使用了雾计算。

雾计算具有支持用户移动性、实时交互、低延迟等优点。但是，与计算能力强大的云服务器相比，雾节点的计算能力以及存储容量是十分有限的。因此，雾计算更适合处理实时的小数据的任务以及对时延要求敏感的任务，而云计算更适合处理计算密集的任务。因此，在云雾端协同的架构下对终端设备的任务进行合理地卸载是最高效的。

云雾协作的架构如图 2.1 所示。其主要由三层组成，分别是物终端设备、雾节点和云数据中心。终端设备与雾节点之间通过局域网连接通信，终端设备与云数据中心、雾节点与云数据中心之间通过广域网连接[32, 33]。

2.2 任务分配技术概述

2.2.1 任务分配概述

任务分配是指终端设备按照一定的优化目标将其产生的资源密集型或者时延敏感性的任务合理地分配到附近的雾节点或者云数据中心的过程。换句话说，任务分配解决的是是否要将任务进行分配、将任务分配到哪个地方的问题。在云雾协作环境中，

下层由终端设备组成，任务由终端设备产生，终端设备将根据分配决策将其产生的任务传输到相应的雾节点或者云数据中心，其过程如图 2.2 所示。有效地分配任务可以大大的提高系统的整体性能，因此，使用何种方法可以将不同类型的任务合理地分配到附近的雾节点或者云数据中心以实现优化目标是目前最重要的研究领域之一。

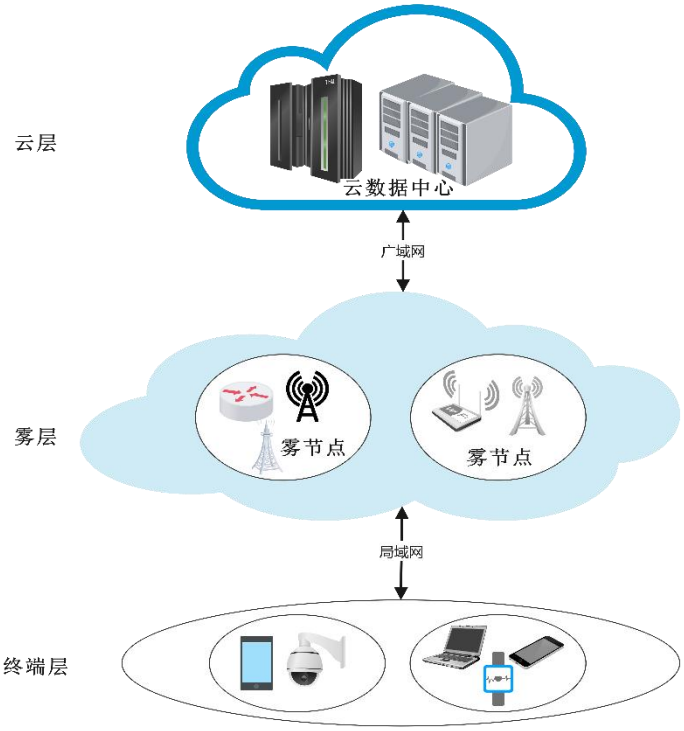


图 2.1 雾计算架构

Fig.2.1 Fog computing architecture

根据任务可传输到的雾节点群的数量，任务分配的方式可以分为单雾节点类型和多雾节点类型两类。单雾节点类型是指终端设备可以连接到的雾节点群里只有一个雾节点。Liu 等[34]利用排队理论分别应用在移动设备、雾和云中心，并考虑了无线链路的数据速率和功耗。通过找到每个移动设备的最佳卸载概率和发射功率，制定一个多目标优化问题以最小化能耗、执行延迟和支付成本。通过大量的仿真研究证明了所提方案的有效性，与其他几种方案相比也具有更优越的性能。多雾节点类型是指终端设备可以连接到的雾节点群里包含多个雾节点。换句话说，终端设备产生的任务可以传输到多个雾节点进行并行处理，以保证用户体验质量的要求。Meng 等[35]研究了混合计算卸载问题，考虑了云计算服务器和雾计算服务器两种卸载地点的计算能力和通信能力以在给定的延迟约束内完成计算任务的同时最大限度地减少通信和计算的总能耗。为了解决这一问题，作者将其划分为四个子问题，针对每个子问题给出了一个封闭式计

算卸载解。经过大量的仿真结果表明，在时延约束下，所提出的混合计算卸载方案比传统的单类型计算卸载方案的能耗更低。

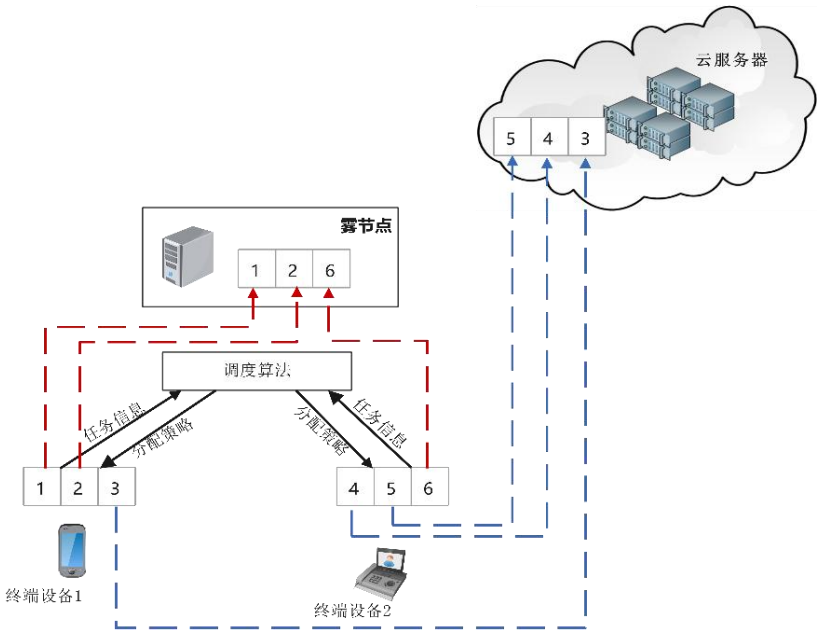


图 2.2 任务分配过程

Fig.2.2 Task assignment process

根据终端设备产生的任务是否可以进行拆分这一属性，任务分配可以分为整体卸载和部分卸载两种方式。整体卸载指的是设备所产生的任务是一个整体，不可以进行拆分。一个任务只能要么全部分配到一个计算节点上，要么均不分配到某一个计算节点上。Lyu 等[36]设计了一种半分布式的启发式分配决策算法用于解决任务是否应该被分配到除本地以外的其他节点以及要被分配到哪里这一问题。这一算法降低了查找局部最优值的复杂性。经过仿真结果表明，所提算法的平均性能在最佳性能的 5%以内，与其他解决方案相比，随着用户数量的增加，其性能越显著。部分卸载指的是一个完整的任务是可以进行拆分的，可以选择将拆分的每个子任务分配到不同的计算节点上进行处理，无需强制性地集中分配到一个计算节点上进行执行与处理。部分卸载要比整体卸载更加复杂。因为不仅要考虑是否应该被分配到除本地以外的其他节点还要考虑分配多少。Liu 等[37]提出了一种基于价格的分布式方法来管理用户的计算任务的卸载。每一个计算节点都有固定的价格，每个终端设备在本地做出决策以最小化自己的成本。其中每个任务可以被任意地按位划分以进行部分卸载。仿真结果验证了所提方案的有效性。

2.2.2 任务分配常用算法

任务分配已被认证为 NP-hard 问题。一个好的任务分配调度器必须能够根据不断变化的场景和任务类型调整其给出的分配策略。任务分配问题被认为是具有复杂目标的非线性问题，该类问题的解决方法主要有传统方法、元启发式算法和深度强化学习算法。传统方法往往只注重局部搜索，不能保证全局最优，而且也无法保证可以有效地解决高度非线性、多峰的问题。同时，传统方法本质大多数是具有不确定性的，没有随机性。在元启发式算法逐渐进入研究者的视线后，传统方法便不被研究者们广泛使用了。

元启发式算法是启发式算法的改进，是随机算法与局部搜索算法相结合的产物。元启发式算法主要可以分为基于物理学、生物学、化学和进化这四种类型的算法。

基于物理学的算法主要有引力搜索算法[38]、智能水滴算法[39]、模拟退火算法[40]等等。Choudhary 等[41]在平衡负载的基础上提出了一种改进的引力搜索算法，通过引入多样性和进度参数来防止陷入局部最优。经过仿真实验的结果表明，与其他技术相比，所提出的方法最小化了总成本。Kalra 等[42]修改了智能水滴算法的概率函数用于解决最小化最大完工时间的问题。经过实验验证，该算法在大型工作流的情况下性能更好。

基于生物学的算法构成了自然启发算法的最大部分。基于生物学的算法分为群智能算法和非群智能算法。群智能算法主要有粒子群算法、蚁群算法等。非群智能算法主要有混合蛙跳算法、共生生物搜索算法等。Zhang 等[43]采用粒子群算法在期限约束下对混合云中的 BoT 进行调度，以实现利润最大化。他们使用一个映射算子粒子映射解决方案。除此之外，一个快速的任务调度方法与加速策略是用来计算集成解决方案的目标。Movahedi 等[22]以能耗和时延为优化目标，将任务分配问题表述为整数线性规划优化模型，提出了一种基于对立的混沌鲸鱼优化算法以提高原始鲸鱼优化算法的性能。Hashemi 等[44]提出了一种多目标灰狼优化算法用来检查资源状态和管理任务。所提方法与 700 个节点、1000 个节点和 5000 个节点的 15 个场景进行了对比，证明了所提方法的有效性。Yu 等[45]为了减少雾计算-生物信息物理系统中的制造跨度与总响应时间，提出了遗传-蚁群优化算法，经过仿真实验验证了所提算法的有效性。Wang 等[46]针对雾医疗系统中的任务调度问题，提出了一种混合遗传算法和粒子群优化策略。使用该策略实现了降低制造时间和整体响应时间。仿真结果表明，所提算法与遗传算法和粒子群优化算法相比具有更好的性能。Huang 等[47]提出了一种基于李雅普诺夫框架的启发式粒子群优化算法解决了物联网节点的计算能耗、传输能耗和雾节点计算能耗之间的平衡，从而以最小的能耗完成任务。与原有的粒子群优化算法和贪婪算法相比，所提算法的性能有了显著的提升。Subramoney 等[48]提出了一种多群

粒子群优化算法，以改善云雾环境下工作流程的调度。在多群粒子群优化算法中，粒子被分成几个群，每个群都有自己的认知和社会学习系数。经过与粒子群优化算法、遗传算法、差分进化算法相比，所提算法的性能指标都优于其他算法。Ning 等[49]在云环境下提出了一种改进的混合蛙跳算法，将反向学习方法用于初始化，将交叉因子用于局部最优解，并将进化策略用于全局最优解。Abdullahi 等[50]提出了共生生物搜索算法的离散版本，用于在云中调度独立任务。在最大完工时间、响应时间和不平衡度等方面，将所提方法与粒子群算法的不同变体进行了比较。然而，没有考虑能量效率和成本。此外，将该方法与粒子群算法的变体进行了比较并取得了良好的表现。

基于化学的算法主要有化学反应优化算法[51]和化学反应算法[52]。Yan 等[53]在云中使用了化学反应优化算法，以成本为优化目标进行科学合理地工作流调度。同时，采用正交实验设计来调整与调度问题相关的参数。通过在两个真实的工作流上进行实验验证，该方法可以有效地降低成本。

进化算法是优化算法的重要组成部分。其中，常用的算法主要有遗传算法、克隆选择算法等。Szabo 等[54]采用了一种渐进式方法来最大限度地减少运行时间并优化实例数据的局部性，分别使用分配染色体和排序染色体对解进行编码，其中，分配染色体用于分配任务，排序染色体用于定义任务的执行顺序。该方法在执行时间和数据传输时间上分别提高了 10%和 80%。Liu 等[55]为了解决绿色调度的问题，引入了基于克隆的调度算法。仿真实验表明，该方法在减少执行时间和能耗方面具有明显的效果，但不适用于动态任务调度环境。

深度强化学习是机器学习的一个新兴分支，它为智能体提供智能决策以有效应对环境的动态变化。基于深度强化学习的算法采用学习过程来学习动作和不断变化的环境，以丰富的经验在长期操作中得出最佳决策[56]。在交互的每个时间步长 t 中，智能体对环境的状态进行观测，然后决定采取行动。同时，智能体可以从环境中得到回报，表示当前的状态是好还是坏。

目前，深度强化学习已经越来越多地被研究和应用，以有效地解决许多不确定计算环境中的资源分配相关问题。Dai 等[57]提出了多用户边缘云协作网络，所有的设备和基站都具有计算能力。终端设备可以将计算密集型或者延迟敏感型的任务分配到雾节点上进行处理，在文中作者将任务分配和资源分配问题表示为马尔科夫决策过程，然后使用 DRL 算法设计了一个优化决策来降低系统的能量消耗。

深度强化学习算法主要有 Q-Learning、DQN、Actor-Critic 算法等。Q-Learning 是强化学习中的一种 Value-Based 算法，通过计算出一张 Q 表，用来记录每种状态下采取每个动作能够获得的最大奖励，然后通过这张表就能找到每个状态下的最优动作，从而保证最后的收益最大化。为了应对不同环境下的不确定性，Kiran 等[58]认为强化学习除了考虑及时奖励外还会将长期目标考虑进去，这对于动态环境来说是十分重要的，因此提出了一种 RL 优化框架，使用 Q-Learning 算法和合作 Q-Learning 算法解决无线移动边缘计算中的任务卸载和资源分配问题。仿真结果表明，与 Q-Learning 相比，该算法的总时延降低了 62.10%。Hossain 等[59]在边缘计算场景中，以时延和能耗的联合成本函数作为目标，使用 Q-Learning 算法对数据任务进行决策，决定是在边缘设备处理还是在本地处理。Q-Learning 本质上就是使用 Q 表建立了“状态和动作”和“在该状态下执行该动作所产生的奖励值”之间的映射关系。也就是说，Q 表的大小取决于状态数量与动作数量的乘积。当环境比较复杂时，状态数量和动作数量也会变得很多，那么此时 Q 表就会变得很大，查找和储存的效率就会变得很低。为了解决 Q-Learning 的这一问题，DQN 算法被提出。DQN 算法采用神经网络来代替 Q 表，以状态和动作作为输入，输出对应的 Q 值，或者以状态作为输入，输入各种动作下对应的 Q 值。Huang 等[60]提出了一种基于 DQN 的多任务分配和资源分配算法。作者将该问题转化为混合整数非线性规划问题。在多用户多任务的移动边缘计算环境下设计联合任务卸载和带宽分配优化，以最大限度地降低能源成本、计算成本时延成本方面的整体卸载成本。经过大量的仿真实验后，证明了所提出的方法可以实现接近最优的性能。Sarkar 等[61]提出了一种基于深度强化学习的雾网络服务供应策略，以最大限度地降低网络的能耗。作者首先将求解的问题建模为马尔科夫决策过程并采用 DQN 的概念进一步解决。此外，还提出了一个任务迁移策略，以确保计算设备的高可用性。最后，通过实验结果表明，与基线策略相比，所提出的算法在平均成本方面取得了显著的改善。

3 单雾节点多终端场景下的任务动态分配方法

3.1 引言

随着 5G 时代的来临，物联网行业快速发展，越来越多的大规模的终端设备连接到云数据中心，给集中式云计算数据中心带来了巨大的挑战。雾计算是为了减轻云服务器的压力而产生的，雾节点的计算能力虽然远小于云服务器的计算能力，但是相较于终端设备来说却大得多，因此将终端设备的小任务需求上传至距离用户最近的雾节点进行处理可以在一定程度上降低计算任务的总时延，同时提高用户的体验质量。但是，一个雾节点上若被分配了过载的任务会因为等待时延的增长而导致总时延的增加，甚至可能会导致任务执行失败的情况。与此同时，如果终端设备的任务的数据量过小，在雾节点上执行反而会由于通信而增加时延和能耗。对于一些任务量较大或者某一个雾节点的覆盖范围内的终端设备产生的任务的数量过大时，仅仅依靠雾节点和终端设备本身是很难满足其需求的。为此，在本节中选取云-雾-端协作的架构，考虑以时延和能耗联合优化的任务动态分配。

本节首先介绍了以时延和能耗联合优化的系统模型并将其转化为 0-1 整数规划模型，然后将问题进行公式化。随后，详细描述了二进制天牛须搜索算法，该算法在天牛须搜索算法的基础上引入了 Sigmoid 函数，将天牛须的位置映射到 0,1 之间，并以 0.5 为界将其设置为 0 或者 1。最后，以粒子群优化算法、轮询算法、随机算法、全部卸载方法以及全部在终端设备执行方法为基线方法进行仿真实验，仿真结果表明，无论是终端设备的数量发生变化还是雾节点及终端设备的计算能力发生变化，二进制天牛须搜索算法在任务动态分配问题上都具有最良好的表现。同时，实验还验证了权重对时延和能耗的影响，若权重偏向时延，则相较于能耗来说，算法对时延的优化力度更大，反之亦然。

3.2 系统模型

该模型由 N 个终端设备、一个雾节点以及一个云服务器组成。每个终端设备可以产生若干个任务，这些任务可以在终端执行，也可以卸载到雾节点或者云服务器执行。每个任务都具有原子性，是不可分割的。终端设备的集合定义为$TD = \{1,2,\ldots,N\}$。

3.2.1 终端计算模型

当任务在终端执行时，由于任务未进行传输，因此任务时延等于任务执行时间，其计算过程如公式 3-1 所示。

$$T_{lo} = \frac{D_i \times C}{C_{local}} \tag{3-1}$$

其中，D_i表示任务i的数据量，C表示 CPU 处理每 bit 任务所需要的 CPU 周期数。

当任务在本地执行时的能耗如公式 3-2 所示。

$$E_{lo} = T_{lo} \times P_{local} \tag{3-2}$$

其中，P_{local}表示终端设备的计算功率。

3.2.2 云雾协作计算模型

用户将任务发送至雾节点的数据传输速率为R，其计算过程如公式 3-3 所示。

$$R = B \times log_2 (1 + \frac{P_{lau} \times d^{-r}}{\sigma^2}) \tag{3-3}$$

其中，B为终端设备与雾节点之间的带宽，d^{-r}为终端设备与雾节点之间的信道系数，σ^2为信道的噪声功率。

当任务在雾节点执行时，任务执行时延由数据发送时延和任务执行时延组成，其计算过程如公式 3-4 所示。

$$T_{fog} = \frac{D_i}{R} + \frac{D_i \times C}{f\_fog} \tag{3-4}$$

其中，f_fog表示雾节点的计算能力。

当任务上传至雾节点执行时，由于雾节点一直处于通电状态，因此本文只考虑移动终端的能耗，故能耗如公式 3-5 所示。

$$E_{fog} = P_{lau} \times \frac{D_i}{R} \tag{3-5}$$

其中，P_{lau}表示终端设备将数据发送给雾节点的传输功率。

若任务i被分配至雾节点，但此时雾节点的内存和 CPU 均不满足任务i的执行需求时，任务将会被上传至云服务器进行处理。任务i由云服务器执行的总时延包括任务传输至雾节点的传输时延、任务由雾节点传输至云服务器的传输时延以及任务在云服务器的执行时延组成。其任务i执行的总时延如公式 3-6 所示。

$$T_{cloud} = \frac{D_i}{R_{td}} + \frac{D_i}{R_{fog}} + \frac{D_i \times C}{f\_cloud} \tag{3-6}$$

其中，R_{td}表示终端设备将任务发送至雾节点的传输速率，R_{fog}表示雾节点将任务发送至云服务器的传输速率。f_cloud表示云服务器的计算能力。

当任务上传至云服务器执行时，本文只考虑终端设备向雾节点传输任务的能耗以及雾节点向云服务器传输任务的能耗，因此能耗的计算过程如公式 3-7 所示。

$$E_{cloud} = P_{lau} \times \frac{D_i}{R_{td}} + P_{lau\_fog} \times \frac{D_i}{R_{fog}} \tag{3-7}$$

其中，P_{lau}表示终端设备将数传输给雾节点的传输功率。P_{lau_fog}表示雾节点将数据传输给云服务器的传输功率。

3.2.3 优化目标

本节的优化目标是联合优化任务的总任务延迟以及能耗。λ_i，$\lambda_i \in \{0,1\}$被用来描述任务i是否被卸载雾节点上。当$\lambda_i = 0$时表示任务i由终端设备执行，当$\lambda_i = 1$时表示任务由雾节点执行。γ_i被定义为任务i是否上传至云服务器执行，$\gamma_i \in \{0,1\}$。当$\gamma_i = 0$时表示任务i不上传至云服务器，当$\gamma_i = 1$时表示任务i被卸载至云服务器执行。只有当$\lambda_i = 1$时，即只有当任务被分配至雾节点执行时才会有$\gamma_i = 1$。综上所述，所有任务的总时延和能耗分别为如公式 3-8 和 3-9 所示。

$$T_{total} = \sum_{i=1}^{m} (1-\lambda_i) \times T_{lo} + \lambda_i(1-\gamma_i) \times T_{fog} + \lambda_i\gamma_i \times T_{cloud} \tag{3-8}$$

$$E_{total} = \sum_{i=1}^{m} (1-\lambda_i) \times E_{lo} + \lambda_i(1-\gamma_i) \times E_{fog} + \lambda_i\gamma_i \times E_{cloud} \tag{3-9}$$

因此，目标优化问题的表示如公式 3-10 所示。

$$W = (1-\omega) \times T_{total} + \omega \times E_{total} \tag{3-10}$$

其中，ω表示权重，$\omega \in (0,1)$。由于终端设备和雾节点有内存和 CPU 资源数的限制，因此，在终端设备上以及雾节点上执行的任务所需要的内存和 CPU 资源数都应该小于执行的计算节点所拥有的。因此，约束条件如公式 3-11 至公式 3-15 所示。

$$\sum_{i=1}^{len(que_{fog})} store_i \le store_{fog} \tag{3-11}$$

$$\sum_{i=1}^{len(que_{fog})} CPU_i \le CPU_{fog} \tag{3-12}$$

$$\sum_{i=1}^{len(que_n)} store_i \le store_n \tag{3-13}$$

$$\sum_{i=1}^{len(que_n)} CPU_i \le CPU_n \tag{3-14}$$

$$\sum_{i=1}^{len(task)} T_{total}^i \le deadline_i \tag{3-15}$$

其中，公式 3-11、3-12 分别表示在雾节点上执行的任务的所需要的内存和 CPU 资源数都应该小于雾节点所拥有的。公式 3-13、3-14 分别表示在终端设备上执行的任务的所需要的内存和 CPU 资源数都应该小于雾节点所拥有的。公式 3-15 表示每一个任务的时延都应该小于其最低要求时延。

3.3 基于改进的天牛须搜索算法的任务分配方法

3.3.1 天牛须搜索算法

天牛须搜索优化算法于 2017 年提出，是一种在线优化算法[62]，天牛须搜索算法部分变量定义如图 3.1 所示。

（1）在 D 维空间中，天牛须的位置被定义为$X = \{x_1, x_2, ..., x_D\}$。天牛须的左须和右须的位置在第 *t* 次迭代时被定义为公式 3-16 和公式 3-17。

$$X_l = X^t + d^t \times \vec{b} \tag{3-16}$$

$$X_r = X^t + d^t \times \vec{b} \tag{3-17}$$

其中X^t表示天牛的质心位置，d^t表示两须的距离。$\vec{b}$被定义为天牛的朝向向量，其具体计算如公式 3-18 所示。

$$\vec{b} = \frac{rand(D,1)}{\|rand(D,1)\|} \tag{3-18}$$

其中$rand(*)$表示随机函数。

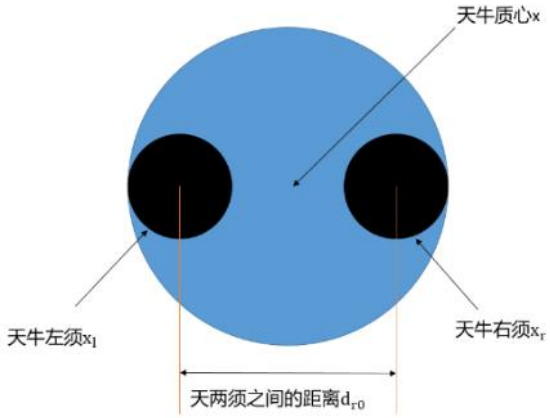


图 3.1 天牛须搜索算法部分变量定义

Fig. 3.1 Partial variable definition of BAS

（2）根据天牛须的左须和右须的对气味浓度的对比，判断天牛的下一步的位置，具体的位置更新公式如公式 3-19 所示。

$$X^{t+1} = X^t + \delta^t \times \vec{b} \times sign(f(x_r) - f(x_l)) \tag{3-19}$$

其中δ^t为第 t 次迭代时天牛的步长，$f(*)$为适应度值函数，$sign(*)$为符号函数，定义如公式 3-20 所示。

$$sign(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, otherwise \end{cases} \tag{3-20}$$

（3）更新步长，步长的更新方式如公式 3-21 所示。

$$\delta^{t+1} = \delta^t \times \delta_e \tag{3-21}$$

其中δ_e表示递减因子。

天牛须搜索优化算法的流程图如图 3.2 所示。

3.3.2 改进的天牛须搜索算法

根据目标函数可知，任务的分配方案由λ_i和γ_i共同决定，因此，对于任务i的分配方案来说应该设置λ_i和γ_i这两个变量。又因为$\lambda_i \in \{0,1\}$并且$\gamma_i \in \{0,1\}$，所以该模型可以视为 0-1 整数规划模型。综上所述，该问题的维度为 $2 \times N$。设现共有 3 个任务，其分配方案分别

为：任务 1 在雾节点执行，任务 2 在终端执行，任务 3 在云服务器执行，则其所对应的方案的编码为：X = [1 0 0 0 1 1]。第 1、2 位表示任务 1 的分配方案，10 分别表示$\lambda_i = 1$、$\gamma_i = 0$。第 3、4 位表示任务 2 的分配方案，00 分别表示$\lambda_i = 0$、$\gamma_i = 0$。第 5、6 位表示任务 3 的分配方案，11 分别表示$\lambda_i = 1$、$\gamma_i = 1$。由于当$\gamma_i = 1$时$\lambda_i = 1$，因此，对于任务i，如果出现方案编码为 01 时，应按照适应度值较小的方案将其调整为 00 或者 11。由于天牛须搜索算法用于解决连续型问题，而目前待解决的为 0-1 整数规划问题，因此，需对原始的天牛须搜索算法进行改进使其适用于该问题。因此，提出二进制天牛须搜索算法，将任务分配方案映射为算法中的天牛的位置信息，故需保证$X^{t+1} \in \{0, 1\}$。因此，在计算完公式 3-19 进行完位置更新后，引入 Sigmoid 函数，Sigmoid 的函数定义如公式 3-22 所示。

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3-22}$$

对于$x \in (-\infty, +\infty)$有$Sigmoid(x) \in (0,1)$。将天牛的位置信息作为参数输入至 Sigmoid 函数，如果公式 3-23 所示。为了保证$X^{t+1} \in \{0, 1\}$，则继续对天牛的位置信息进行公式 3-24 的计算，将其转化为二进制形式。

$$X^{t+1} = Sigmoid(X^{t+1}) \tag{3-23}$$

$$X^{t+1} = \begin{cases} 1, X^{t+1} > 0.5 \\ 0, X^{t+1} \leq 0.5 \end{cases} \tag{3-24}$$

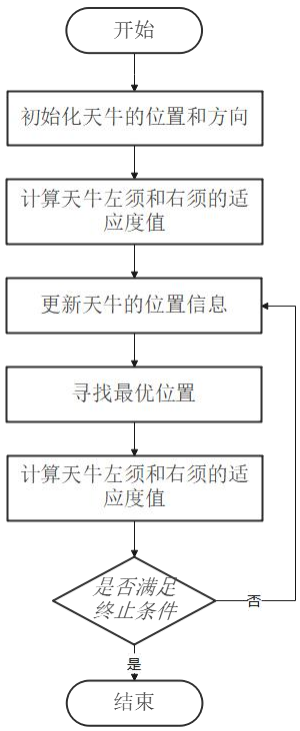


图 3.2 天牛须搜索算法的流程图

Fig. 3.2 Flow chart of BAS

二进制天牛须搜索的伪代码如算法 3.1 所示。

算法 3.1 BSBAS 算法

Algorithm. 3.1 BSBAS algorithm

输入: 建立目标函数$f(X^t)$，其中$X^t = [x_1, ..., x_i]^T$，初始化参数x_0, d_0, δ_0.

输出: x_{bst}, f_{bst}.

function BSBAS

while $(t < T_{max})\ or\ (stop\ criterion)$ **do**

按照公式 3-18 计算天牛的朝向向量$\vec{b}$;

根据公式 3-16 和 3-17 对天牛须的左须和右须进行位置更新;

根据公式 3-19 对位置X^{t+1}进行更新;

根据公式 3-23 和 3-24 对位置信息X^{t+1}进行二进制转换;

if $X^{t+1} < 0.5$ **then**

$X^{t+1} = 0$;

else

$X^{t+1} = 1$;

end if

if $f(X^t) < f_{bst}$ **then**

$f_{bst} = f(X^t),\ x_{bst} = X^t$;

end if

根据公式 3-10 更新步长和搜索距离;

end while

end function

3.4 仿真实验结果与分析

本节首先介绍了仿真实验所使用的平台，以及设置了一些在实验中不可缺少的参数。其次，使用粒子群优化算法（PSO）、轮询算法算法（RR）、随机方法（Random）、全部卸载（Full Offload）以及全部在终端设备执行（Full Local）的方法作为基线算法在同种环境下与 BSBAS 算法相对比，并对实验结果进行可视化。最后，在不同的实验环境下对仿真实验的结果进行分析。

3.4.1 实验参数设置

本节的仿真实验是基于 Windows 10 的操作系统完成的，所以使用的 Python 版本为 3.8。在仿真实验中，将终端设备的计算功率设置为 600mw、终端设备与雾节点之间的带宽 B 为 20MHZ、终端设备与雾节点之间的距离为 160 米、信道的噪声功率为 100mw、终端设备将数据发送给雾节点的传输功率为 100mw、雾节点将数据发送给云服务器的传输功率为 200mw、数据计算量的大小设置为 2000-2500KB。

3.4.2 实验结果分析

为了更好地评价算法的性能，因此本节选用了 PSO、RR、Random、Full Offload 以及 Full Local 五种基线算法在不同的实验环境下进行实验。实验环境包括任务量的大小、终端设备的数量、雾节点的计算能力以及不同的权重系数。

（1） 终端设备的数量与总代价的关系

图 3.3 展示了包括所提方法在内的六种算法在不同的终端设备的数量(4 台、6 台、8 台、10 台、12 台、14 台、16 台)下的总代价的实验结果。由图可知，六种任务分配算法所得的总代价均随着终端设备数量的增加而增加。这是因为随着终端设备数量的不断增加，任务数据量也随之增加，导致任务总时延和终端能耗不断增加。终端设备数量越多，本地或雾节点上处理任务的时间越长，处理任务所产生的移动终端能耗也在增加。同时，无论终端设备的数量是多还是少，本节所提的 BSBAS 算法在所有的分配方案中均得到了最小的系统代价。随着终端设备的数量不断增加，所提算法与其他算法的总代价差也不断增大。因此，所提算法的优势随着终端设备数量的增多而更加显著。

（2） 雾节点的计算能力与总代价的关系

图 3.4 展现了不同雾节点的计算能力（6GHz、8GHz、10GHz、12GHz、14GHz）的情况下，使用五种任务分配算法所得总代价的结果。由图可以看出，随着雾节点的计算能力的增加，BSBAS、PSO、RR、Random、Full Offload 均得到了逐渐降低的总代价。由于雾节点的计算能力与全部在终端设备执行任务并没有关系，因此，随着雾节点的计算能力的增加，Full Local 方法在总代价上并没有变化。无论雾节点的计算能力为多少，所提算法与其他基线算法相比都取得了最好的表现。

（3） 终端设备的计算能力与总代价的关系

将雾节点的计算能力设为定值，改变终端设备的计算能力，使其取值范围在（1,3）。如图 3.5 所示，六种任务分配算法所得到的总代价随着终端设备的数量的增加而增加，而 BSBAS 算法得到的总代价值明显小于其他五种算法，同时，所使用算法的显著性随着终端设备的增加而更加明显。

（4） 权重对总时延与能耗的影响

当雾节点数为 10 时，本文提出的分配方法在不同权重下产生的任务延迟和能耗如图 3.6 所示。从图 3.6(a)-(b)可以看出，当权重增加时，BSBAS 算法更加关注任务的总延迟，此时系统产生的延迟更小。但此时系统优化能耗的能力会减弱，从而增加能耗。因此，在

相同的实验环境下，权值越大，时延越小，终端能耗越低。此外，还可以看出，随着数据大小的增加，权值对延迟和终端能耗的影响也在增加。

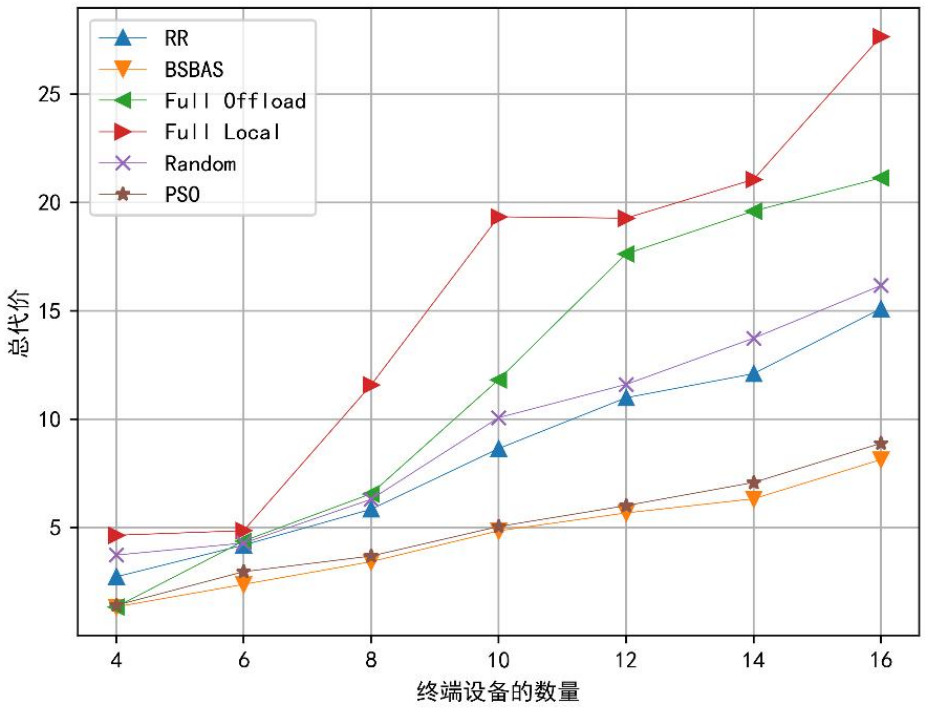


图 3.3 不同终端数量对总代价的影响

Fig. 3.3 The influence of different terminal number on total cost

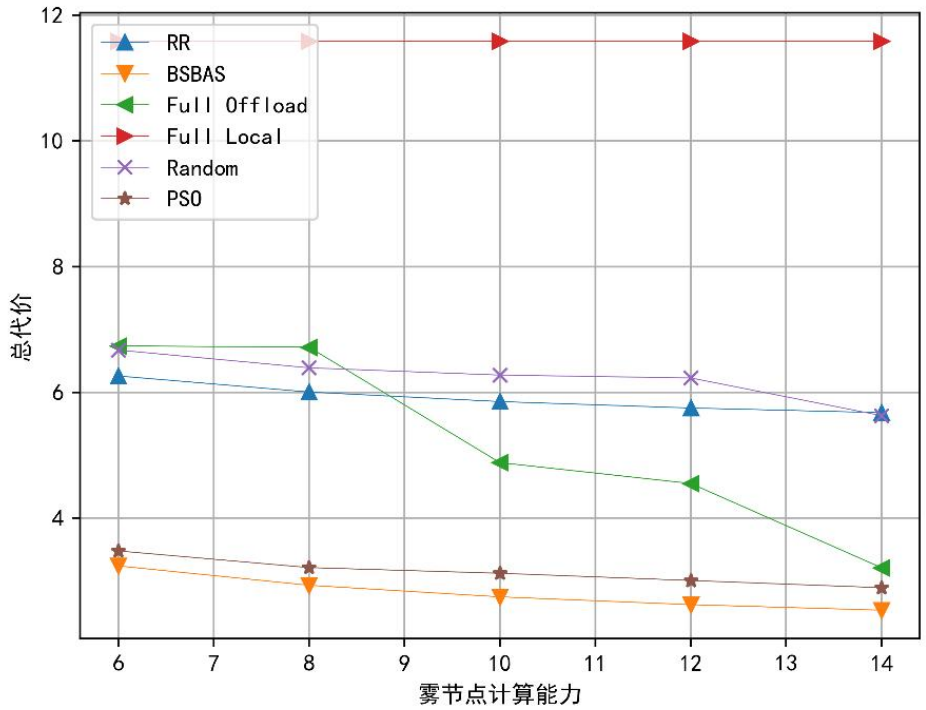


图 3.4 不同雾节点计算能力对总代价的影响

Fig. 3.4 The influence of computing power of different fog nodes on total cost

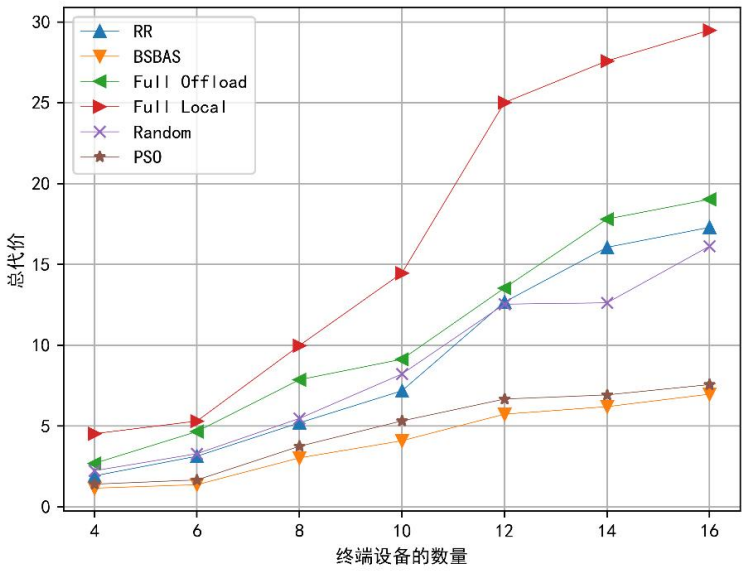
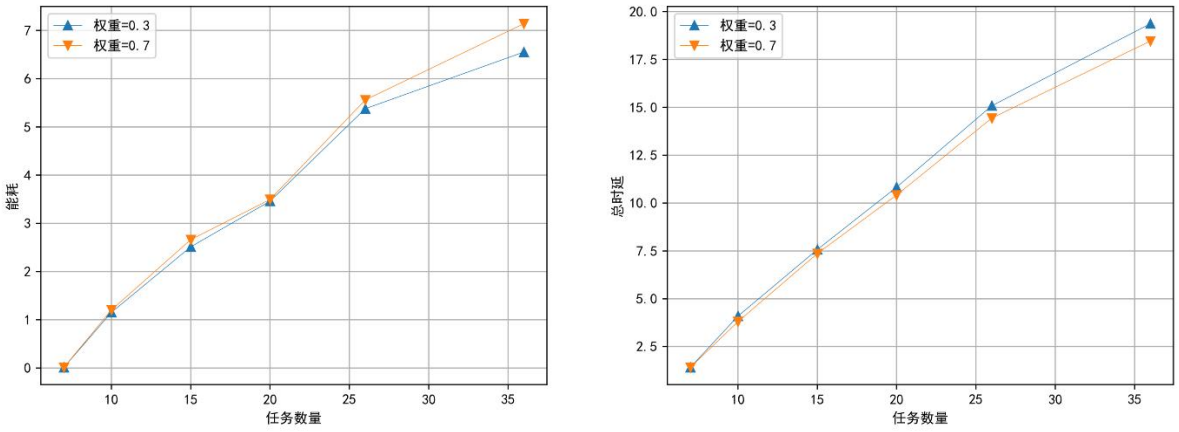


图 3.5 不同终端设备计算能力对总代价的影响

Fig. 3.5 The effect of computing power of different terminal devices on total cost



（a）权重对能耗的影响　　（b）权重对总时延的影响

图 3.6 权重随任务数量对总时延和能耗的影响

Fig. 3.6 The effect of computing power of different terminal devices on total cost

3.5 本章小结

本章首先介绍了以时延和能耗联合优化的任务分配的系统模型，随后将联合优化的问题转化为 0-1 整数规划模型并进行公式化描述，采用改进的二进制天牛须算法对建立的模型进行最优解寻找。最后，对所提算法进行了仿真实验，并以粒子群优化算法、轮询算法、随机算法、全部卸载方法以及全部在终端设备执行这五种算法为基线算法进行对比，在不同的终端设备的数量、不同的雾节点的计算能力、不同的终端设备的计算能力这三种环境下所提的算法均取得了良好的表现。同时还验证了权重 ω 对优化时延和能耗的影响。

4 多雾节点多终端场景下的任务动态分配方法

4.1 引言

SDN 指的是数据平面与控制平面相分离，将所有的控制和管理任务都部署在一个集中的控制器上，其产生的目的是降低实现网络设备的硬件和软件所涉及的设计复杂性。在云雾协作环境下，由于每个时间段的终端设备、每个雾节点以及云服务器的资源状况和状态信息将直接导致任务的执行成功或失败，故终端设备、每个雾节点以及云服务器的所有信息对于用户体验质量来说极其重要，所以，在对某一段时间内收集的终端设备的所有任务进行分配的时候若能实时掌握所有计算节点的状态信息，则可以更高效地将任务分配到适合的计算设备上。因此，将 SDN 的优势应用到云-雾-端架构中，通过提供对终端设备、雾节点及云服务器的资源等信息的集中控制的解决方案来使任务动态分配更加合理高效。

本章首先介绍了基于 SDN 的云-雾-端架构，并建立了系统模型。该模型由多个终端设备、多个雾节点和一个云服务器以及一个 SDN 控制器组成。SDN 控制器用来收集每个终端设备、雾节点和云服务器的资源等所有信息，同时收集所有终端设备产生的任务信息。一个终端设备有多个任务需要处理。这些任务可以被选择在终端设备上进行处理，也可以被选择卸载到雾节点或者云服务器上执行。每个任务是不可分割的，每个任务之间没有依赖性。它们可以同时将不同的任务卸载到相应的雾节点或者云服务器进行处理。云服务器和每个雾节点都是一个多线程服务器，可以同时处理多个终端设备任务。随后，分别介绍了基于 K-means 算法的任务优先级划分方法和基于改进的量子鸽群优化算法的任务动态分配方法。在任务优先级划分方法中，以任务的计算量以及最低时延要求作为度量相似相的属性，采用欧氏距离来评估样本点与聚类中心的相似性。在任务动态分配方法中，将鸽子的位置信息映射为任务的分配方案。在地图和地标算子阶段，使用蒙特卡罗方法对鸽子的位置进行更新，同时为了避免鸽群出现假性欺骗的情况，将量子进化算法（Quantum Evolutionary Algorithm，QEA）中的量子旋转门概念与鸽群优化算法相结合。与此同时，为了解决改进的量子鸽群优化算法不适用于离散型问题的障碍，又对该算法进行了离散化处理。最后，通过仿真实验验证了所提方法的有效性。

4.2 系统模型

基于 SDN 的云-雾-端架构主要包含两部分，一是对终端设备、雾节点以及云服务器进行监控，二是对终端设备的任务进行合理分配与调度并给出最优的分配策略。监控模块主要是负责收集终端设备、雾节点以及云服务器的状态以及资源信息。任务分配模块主要是

依据监控模块收集到的信息使用任务优先级划分方法以及任务分配方法对任务进行合理分配。基于 SDN 的云-雾-端架构如图 4.1 所示。整个任务分配流程主要由四个步骤组成。

第一步：终端设备将任务信息上传至 SDN 控制器。同时 SDN 控制器收集终端设备、雾节点和云服务器的状态和资源信息。

第二步：SDN 控制器根据终端设备、雾节点及云服务器的状态和资源信息，利用任务优先级划分方法和任务分配方法合理地分配任务。最后，SDN 控制将分配策略发送到每个终端设备。

第三步：每个终端设备根据收到的任务分配策略将任务发送到相应的雾节点或者云服务器。

第四步：雾节点和云服务器将执行的任务结果返回到终端设备。

由于步骤 1 和步骤 2 是每次任务分配必须经历的一步，因此，步骤 1 所产生的时延被忽略掉。由于在很多应用中，任务在处理完之后只会返回一个很简单的结果，其结果的数据量远远小于终端设备产生的任务数据量，因此，进行结果回传的时延也就是步骤四所产生的时延也被忽略掉。

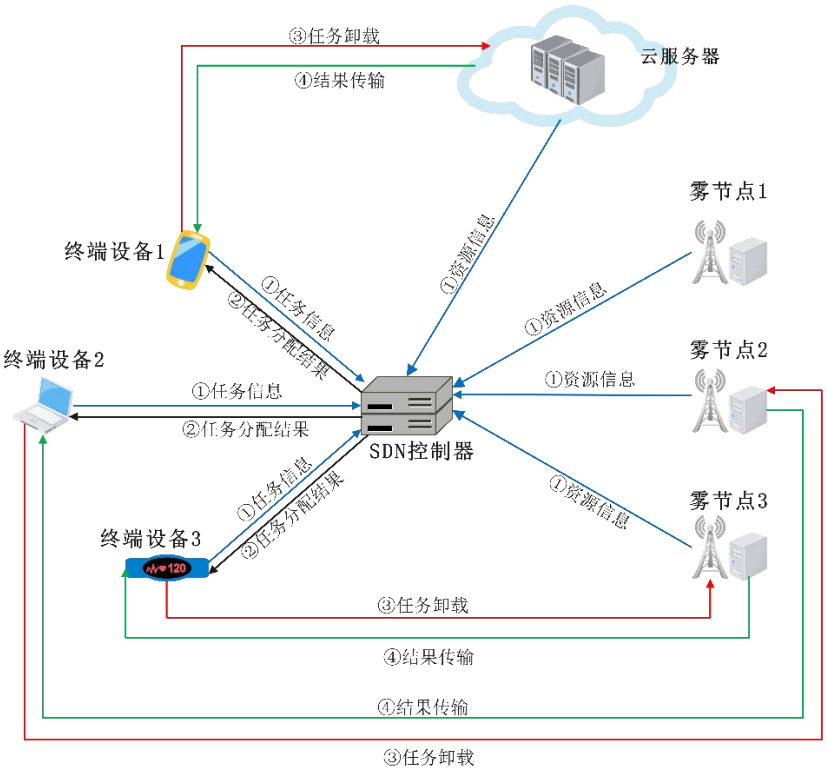


图 4.1 基于 SDN 的云-雾-端架构图

Fig. 4.1 Cloud-fog-terminal architecture based on SDN

为了更好地构建系统模型并将其公式化，定义了一些用来描述问题的变量，如表 4.1 所示。

表 4.1 文中使用的变量定义

Tab. 4.1 The variable definition used in this article

参数	描述
F	雾节点数据集
f_m	m雾节点的计算能力
f_{cloud}	云服务器的计算能力
P_m^{lau}	m雾节点的传输功率
crc_m	m雾节点所拥有的计算资源数
crc_now_m	m雾节点目前所拥有的计算资源数
C_{local}	终端设备的计算能力
que_sub	终端设备的计算队列
que_fog_m	雾节点m的计算队列
$\mathrm{nop\_fog}_m$	雾节点m的并行个数
que_cloud	云服务器的计算队列
Sto_lo_n	终端设备n的内存大小
$Sto_{fog}\_now_m$	雾节点m的目前的内存大小
Sto_fog_m	雾节点m的内存大小
Sto_cloud	云服务器的内存大小
Task	任务集
T_i	任务i
D_i	任务i的数据大小
$storage_i$	任务i所需要的内存
C	CPU 处理每个位任务所需的周期数
ln_k	m终端设备所拥有的计算资源数
rn_i	处理i任务所需的雾节点资源数

4.2.1 终端计算模型

当任务i在终端设备n上执行时，任务i的时延等于任务i的执行时间与等待时延的总和。计算公式如公式 4-1 所示。任务i的执行时延如公式 4-2 所示，等于任务的数据量的大小除以终端设备的计算能力。任务i的等待时延如公式 4-3 所示，等于任务i之前的所有任务的执行时延。

$$T_i^{local} = T_i^{compute} + T_i^{waite} \tag{4-1}$$

$$T_i^{compute} = \frac{D_i * C}{C_{local}} \tag{4-2}$$

$$T_i^{waite} = \begin{cases} 0, & que\_sub = null \\ \sum_{j=1}^{len(que\_sub_i)-1} T_j^{compute}, & else \end{cases} \tag{4-3}$$

4.2.2 雾节点计算模型

当任务i在雾节点m上执行时，任务i的时延由数据传输时延、任务执行时延和等待时延组成，如公式 4-4 所示。

$$T_i^{fog}=\frac{D_i}{r_m^{UF}}+\frac{D_i\times C}{f_m}+T_i^{fog\_waite} \tag{4-4}$$

r_j^{UF}被定义为描述终端设备到 j 雾节点的数据传输速率，如公式 4-5 所示。

$$r_m^{UF}=B\times\log_2\left(1+\frac{P_{lau}\times d_m^{-r}}{\sigma^2}\right) \tag{4-5}$$

B表示的是终端设备与雾节点m之间的带宽。d_m^{-r}为终端设备与m雾节点之间的信道系数。σ^2为信道的噪声功率。$T_i^{fog_waite}$表示任务i在雾节点m上执行时的等待时延。具体的计算方式如公式 4-6 所示。为了更好地解释当到达的任务数大于雾节点的最大并行数量时的处理过程，以图 4.2 所示，假设雾节点的最大并行数量是 3。目前，已经有三个任务正在被处理，其计算完成的时间分别为 *t1*、*t2* 和 *t3*。此时，任务 4 到达雾节点。在节点的可用计算资源大于任务所需的计算资源的情况下，选择三个正在执行的任务中最先被执行完的任务$task_2$，在$task_2$执行完毕后$task_4$被执行。因此，$task_4$的等待时延等于$task_2$的执行时延。

$$T_i^{fog\_waite}=\begin{cases}0, & len(que\_\mathrm{fog}_m)<\mathrm{nop\_fog}_m\\ \min\limits_{i=(1,\mathrm{nop\_fog}_m)}\left(T_i^{fog}\right), & else\end{cases} \tag{4-6}$$

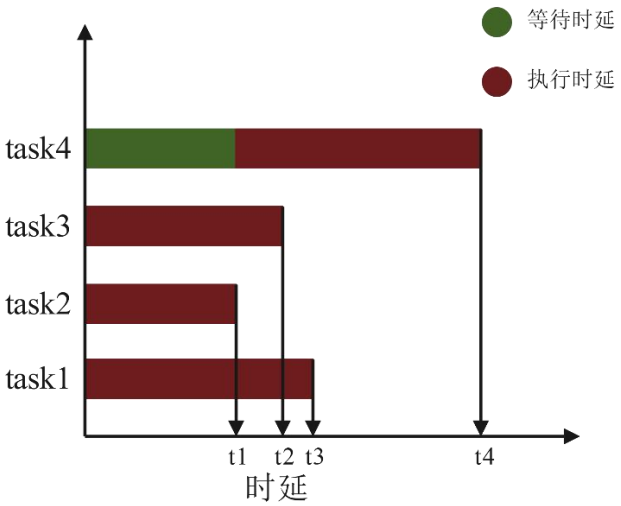


图 4.2 雾节点并行计算的过程

Fig. 4.2 The process of parallel computation of fog node

4.2.3 云服务器计算模型

当任务i在云服务器上执行时，任务i的时延等于任务i执行时间与任务传输时延的总和。计算公式如公式 4-7 所示。任务i的执行时延由两部分组成，首先将任务通过无线网络传送

到距离最近的雾节点k，然后再通过有线带宽由雾节点传送到云服务器。计算公式如公式4-8所示。任务i的计算时延如公式4-9所示，等于任务i的数据量的大小除以终端设备的计算能力。

$$T_i^{cloud} = T_i^{tra\_cloud} + T_i^{com\_cloud} \tag{4-7}$$

$$T_i^{tra\_cloud} = T_i^{ue\_to\_fog} + T_i^{fog\_to\_c} \tag{4-8}$$

$$T_i^{com\_cloud} = \frac{D_i \times C}{f_{cloud}} \tag{4-9}$$

4.2.4 优化目标

在这项工作中，我们的目标是最小化任务的平均延迟。将任务集向量设为$\vec{T} = \{t_1, t_2 \ldots, t_N\}$。将$M$个雾节点设为$M = \{1,2 \ldots, M\}$。$\lambda_{n,m}$和$\gamma_{m,c}$被用来描述任务$n$是否被分配到雾节点$m$或者云服务器上。其中$\lambda_{n,m} \in \{0,1\}$，$\gamma_{m,c} \in \{0,1\}$。如果$\lambda_{n,m} = 1$，那么任务$n$被分配到了雾节点$m$上。如果$\lambda_{n,m} = 0$，则表示任务$n$在本地执行。如果$\gamma_{m,c} = 1$，那么任务$n$被雾节点$m$传输到了云服务器上。如果$\gamma_{m,c} = 0$，则表示任务$n$在雾节点$m$上执行。故对于$\gamma_{m,c} = 1$，则一定有$\lambda_{n,m}$ =1。

因此，目标优化问题的表示如公式4-10所示。

$$W = (\sum_{k=1}^{N} (1 - \lambda_{n,m}) \times T_k^{local} + \lambda_{n,m} \times (1 - \gamma_{m,c}) \times T_k^{fog} + \gamma_{m,c} \times T_k^{tra\_cloud})/N \tag{4-10}$$

对于每个分配到终端设备、雾节点以及云服务器的所有任务的总和不能超过其拥有的资源总数。执行任务的设备所拥有的内存总和应该小于所有任务所需要的内存总和。因此，约束条件如公式4-11至4-14所示。

$$\sum_{i=1}^{len\left(que_{\mathrm{fog}_m}\right)} rn_i \le crc_m \tag{4-11}$$

$$\max_{i=\left(1, len(que_{sub})\right)} (rn_i) \le ln_k \tag{4-12}$$

$$\sum_{i=1}^{len\left(que_{\mathrm{fog}_m}\right)} storage_i \le Sto_{fog_m} \tag{4-13}$$

$$\max_{i=\left(1, len(que_{sub})\right)} (storage_i) \le Sto_{lo_n} \tag{4-14}$$

公式4-11、4-13分别表示在雾节点上执行的任务的所需要的内存和CPU资源数都应该小于雾节点所拥有的。公式4-12、4-14分别表示在终端设备上执行的任务的所需要的内存和CPU资源数都应该小于雾节点所拥有的。

4.3 任务优先级划分与任务动态分配方法

4.3.1 基于 K-means 算法的任务优先级划分方法

K-means 算法被用来划分任务优先级。K-means 算法是一种迭代求解的聚类分析算法。K-means 算法的主要目的是将一个样本数据集中的具有较高相似度的样本划分为多个类别。K-means 算法的步骤如下：

第一步：确定聚类数目 k。在样本中随机选取 k 个样本作为初始的聚类中心。根据相似性度量函数分别计算其他样本距离聚类中心的距离。根据计算结果分配各个数据对象到距离最近的类中。

第二步：计算每一个类中所有样本的平均值。并将平均值作为新的聚类中心。然后，根据相似性度量函数分别计算其他样本距离新的聚类中心的距离。

第三步：判断聚类中心是否改变或者误差是否小于一定的范围。如果满足该条件，那么迭代结束，得到按照类别被划分好的数据集。

K-means 算法的流程图如图 4.3 所示。

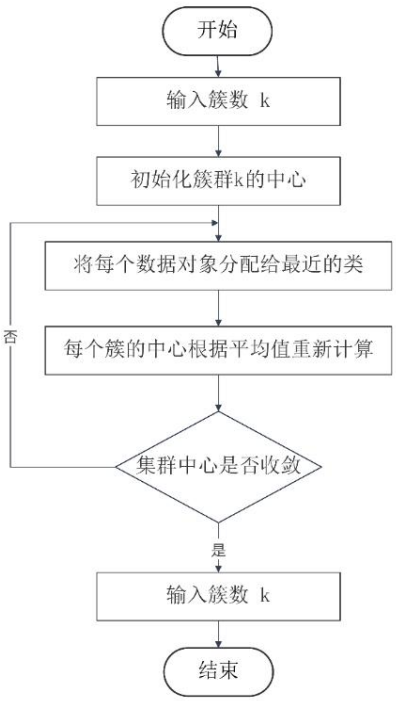


图 4.3 K-means 算法的流程图

Fig. 4.3 Flowchart of the K-means algorithm

在本项工作中采用欧氏距离来评估样本点与聚类中心的相似性。在划分任务优先级中，将任务的最低时延要求与计算量共同作为度量相似性的两个属性。设现有 N 个任务，这 N 个任务的计算量与最低时延要求分为定义为$\{data_1, data_2, ..., data_N\}$和$\{latency_1, latency_2, ...,$

$latency_N\}$。由于任务的计算量与最低时延要求存在单位差异，因此，在计算任意两个任务的相似性之前需要对两个属性进行归一化操作。对计算量与最低时延要求进行归一化的方法如公式 4-15、4-16 所示。则任意两个任务$task_i$和$task_j$之间的相似性$dis_{i,j}$的计算公式如公式 4-17 所示。

$$data_i^{'} = \frac{data_i}{\sum_{j=1}^{N} data_j} \tag{4-15}$$

$$latency_i^{'} = \frac{latency_i}{\sum_{j=1}^{N} latency_j} \tag{4-16}$$

$$dis_{i,j} = \sqrt[2]{(data_i^{'} - data_j^{'})^2 + (latency_i^{'} - latency_j^{'})^2} \tag{4-17}$$

4.3.2 基于 DQPIO 算法的任务动态分配方法

鸽群优化算法是 Duan 等在 2014 年提出的[63]。灵感来自鸽群在归巢过程中的特殊导航行为。它模拟了鸽子在寻找目标的不同阶段使用不同导航工具的机制。该算法主要由地图罗盘算子和地标算子两部分组成。在地图和指南针算子阶段，对于 *D* 维目标搜索空间中的优化问题，在 *N* 只鸽子中，每只鸽子 *i* 代表一个可行解，每只鸽子 i 的速度信息和位置信息分别表示为$V_i = (v_{i1}, v_{i2}, ..., v_{iD}), i = 1,2..., N$和$X_i = (x_{i1}, x_{i2}, ..., x_{iD}), i = 1,2..., N$。鸽子 *i* 在第 *t* 次迭代中的速度和位置更新方法分别如公式 4-18 和 4-19 所示。其中，$V_{i,s}$表示搜索方向，$V_{i,c}$表示收敛方向，R 为地图和罗盘因子，取值范围设为 0~1，rand 为取值范围为 0~1 的随机数。X_{gb}是经过 *t-1* 次迭代后的全局最优位置。在地标算子阶段，首先对鸽子的适应度值进行排序，每次迭代后种群大小减半。剩余鸽子中的中心位置X_{Center}被视为地标。鸽子个体的更新方法如公式 4-20 至 4-22 所示。

$$V_i(t+1) = V_{i,s}(t) + rand \times V_{i,c}(t) = e^{-Rt} \times V_i(t) + rand\left[X_{gb} - X_i(t)\right] \tag{4-18}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{4-19}$$

$$N(t) = N(t-1)/2 \tag{4-20}$$

$$X_{Center} = \frac{\sum X_i(t) \times F(X_i(t))}{N(t) \sum F(X_i(t))} \tag{4-21}$$

$$X_i(t+1) = X_i(t) + rand[X_{Center} - X_i(t)] \tag{4-22}$$

为了解决鸽群优化算法在面对复杂的环境下所出现的容易陷入局部最优、过早收敛的问题，在地图与地标算子阶段使用蒙特卡罗方法对鸽子的位置进行更新以保证鸽群的多样性，具体的位置更新如公式 4-23 所示。

$$X_i(t+1) = \begin{cases} (X_{best\_i}(t) \times X_{Gbest}(t) + V_i(t+1)) + \frac{L(t)}{2} \times \ln\frac{1}{u(t)}, u(t) \geq 0.5 \\ (X_{best_i}(t) \times X_{Gbest}(t) + V_i(t+1)) - \frac{L(t)}{2} \times \ln\frac{1}{u(t)}, u(t) < 0.5 \end{cases} \tag{4-23}$$

其中，$X_{best_i}(t)$表示鸽子 i 在第 t 次迭代的最优位置。$X_{Gbest}(t)$表示在第 t 次迭代的全局最优位置。$u(t)\sim u(0,1)$。其中，$L(t)$的计算公式如公式 4-24 所示。

$$L(t)=2\times\left(\omega_{max}-(\omega_{max}-\omega_{min})\times\frac{t}{MaxIter}\right)\times\left|\frac{\sum_{i=1}^{N_p}X_{best\_i}(t)}{N_p}-X_i(t)\right| \tag{4-24}$$

其中，ω_{max}和ω_{min}分别表示权重的最大值和最小值。

同时为了避免鸽群出现假性欺骗的情况，将 QEA 算法中的量子旋转门概念与鸽群算法相结合。在 QEA 算法中，一个量子可以用“0”态和“1”态表示，即正常态和伪态。量子位的状态如公式 4-25 所示，式中α和β分别表示两种状态的线性概率，并满足公式 4-26。其中$|\alpha|^2$是量子位处于“0”状态的线性概率。$|\beta|^2$是量子位处于“1”状态的线性概率。

$$|\psi\rangle=\alpha|0\rangle+\beta|0\rangle \tag{4-25}$$

$$|\alpha|^2+|\beta|^2=1 \tag{4-26}$$

一个量子位的概率幅度可以定义为$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$。

根据量子旋转门的策略，概率幅度的更新方式如公式 4-27 所示。

$$\begin{bmatrix}\alpha_i(t+1)\\ \beta_i(t+1)\end{bmatrix}=\begin{bmatrix}\cos\Delta\theta & -\sin\Delta\theta\\ \sin\Delta\theta & \cos\Delta\theta\end{bmatrix}\begin{bmatrix}\alpha_i(t)\\ \beta_i(t)\end{bmatrix} \tag{4-27}$$

其中$\Delta\theta$为旋转角度，相当于定义收敛到当前最优解速度的步长。旋转角度的更新策略如公式 4-28，其中$\Delta\theta_g(t)$的计算方法如公式 4-29 所示。

$$\Delta\theta_i(t+1)=\Delta\theta_{gGbest}(t)\times e^{-Rt}+rand[\Delta\theta_g(t)] \tag{4-28}$$

$$\Delta\theta_g(t)=\begin{cases}2\pi+\theta_{gGbest}-\theta_{i,j},\theta_{gGbest}-\theta_{i,j}<-\pi\\ \theta_{gGbest}-\theta_{i,j},-\pi\le\theta_{gGbest}-\theta_{i,j}\le\pi\\ \theta_{gGbest}-\theta_{i,j}-2\pi,\theta_{gGbest}-\theta_{i,j}>\pi\end{cases} \tag{4-29}$$

其中$\Delta\theta_i(t+1)$表示第 i 只鸽子在第 t+1 次迭代之后的旋转角度，$\Delta\theta_{gGbest}$表示全局最优的旋转角度，$i=1,2,\ldots,N_p$，$j=1,2,\ldots,D$。

由于$|\alpha|^2+|\beta|^2=1$，公式 4-27 可以化简为公式 4-30。因此，$\alpha_i(t+1)$的更新公式如公式 4-31 所示。量子旋转门通过改变旋转角度，改变了α_i的概率振幅，从而增加了个体向全局最优解的收敛速度。为了避免在用量子位生成二进制解时出现固定解，因此对$\alpha_i(t+1)$进行公式 4-32 的操作。

$$\begin{bmatrix}\alpha_i(t+1)\\ \beta_i(t+1)\end{bmatrix}=\begin{bmatrix}\cos(\Delta\theta) & -\sin\Delta\theta\\ \sin\Delta\theta & \cos\Delta\theta\end{bmatrix}\begin{bmatrix}\alpha_i(t)\\ \sqrt{1-[\alpha_i(t)]^2}\end{bmatrix} \tag{4-30}$$

$$\alpha_i(t+1)=[\cos(\Delta\theta)-\sin\Delta\theta]\begin{bmatrix}\alpha_i(t)\\ \sqrt{1-[\alpha_i(t)]^2}\end{bmatrix} \tag{4-31}$$

$$\alpha_i(t+1)=\begin{cases}\sqrt{\epsilon}, & if\ \alpha_i(t+1)<\sqrt{\epsilon}\\ \alpha_i(t+1), & if\ \sqrt{\epsilon}<\alpha_i(t+1)<\sqrt{1-\epsilon}\\ \sqrt{1-\epsilon}, & if\ \alpha_i(t+1)>\sqrt{1-\epsilon}\end{cases} \tag{4-32}$$

每只鸽子的收敛方向被重新定义为公式 4-33，速度的更新公式被定义为公式 4-34。

$$V_{i,c}=\widehat{X}_{Gbest}-X_i \tag{4-33}$$

$$V_i(t+1)=\ e^{-Rt}\times V_i(t)+V_{i,c}(t) \tag{4-34}$$

其中$\widehat{X}_{Gbest}$是全局最优解的观察值，其定义为公式 4-35.

$$\widehat{X}_{Gbest}=rand\times|\omega(X_i)|^2\times\left(X_{i,max}-X_{i,min}\right) \tag{4-35}$$

其中$\omega(x_i)$是一个复变函数。$|\omega(x_i)|^2$称为概率密度，表示量子态出现在对应位置和时间的概率，定义为公式 4-36。其中，μ_i和σ_i分别代表期望值和标准差。期望值选用当前最优候选解表示。标准差的计算方式如公式 4-37 所示。$|\psi_i\rangle$的计算过程如公式 4-38 所示。

$$|\omega(x_i)|^2=\frac{1}{\sqrt{2\pi}}\times\exp\left(-\frac{(x_i-\mu_i)^2}{2\sigma_i}\right),\ i=1,2,\ldots,n \tag{4-36}$$

$$\sigma_i^2(|\psi_i\rangle)=\begin{cases}1-|\alpha_i|^2, & if\ |\psi_i\rangle=|0\rangle\\ |\alpha_i|^2, & if\ |\psi_i\rangle=|1\rangle\end{cases} \tag{4-37}$$

$$|\psi_i\rangle=\begin{cases}|0\rangle, & if\ rand\le\alpha_i^2\\ |1\rangle, & if\ rand>\alpha_i^2\end{cases} \tag{4-38}$$

综上所述，混合量子鸽群算法的主要流程为：

第一步：初始化控制参数和鸽群的信息。

第二步：评估每只鸽子的适应度值并记录全局最优解以及全局最优值，记录每只鸽子的历史最优解以及历史最优值。

第三步：在地图和指南针算子阶段根据公式 4-34 进行速度更新，如果全局最优解没有发生变化则根据公式 4-35 至 4-38 以及公式 4-27 至 4-32 进行速度更新，如果全局最优解发生变化则将α_i和β_i被初始化为$\sqrt{2}/2$。然后根据公式 4-23 至 4-29 进行位置以及旋转角度更新。在地标算子阶段，按照公式 4-20 至 4-22 进行位置以及适应度值的更新。

第四步：如果迭代次数小于最大迭代次数，则重复第三步操作，直到满足迭代次数要求，循环结束。

由于改进的量子鸽群优化算法是用来解决连续性问题的，而云雾协作环境下的任务分配问题为离散型问题，为了使改进的量子鸽群优化算法应用于该问题，建立了如下映射关系：鸽子的位置代表任务的分配决策，在混合量子鸽子群优化算法中，将寻找分配决策的过程转化为信鸽归巢的过程，同时对改进的量子鸽群优化算法进行离散化处理。向量中元素的数值即表示该设备需要将任务卸载的位置。

假设 W 个终端设备共产生了 N 个任务，雾节点数量为 M。使用位置信息中的 0 表示为本地计算，第 $1,2,\ldots,M$位表示卸载到对应的雾节点上端执行，$M+1$ 表示将任务分配到云服务器。每只鸽子的位置代表一个 1 行 N 列的卸载决策向量。速度和位置的初始化公式如公式 4-39 和 4-40 所示。$randi(M+1)$表示在[0,$M+1$]上随机生成一个整数。

$$V_i = randi(M+1) \tag{4-39}$$

$$X_i = randi(M+1) \tag{4-40}$$

针对地图和地标算子阶段和地标算子阶段的两处位置更新操作会使卸载决策出现非整数情况，离散化处理方案如公式 4-41 所示。

$$\mathrm{X_i(t)} = \begin{cases} [(X_i(t)]\,\%\,(M+2) & if\ [(X_i(t)] \geq 0 \\ ([(X_i(t)] + M + 2)\,\%\,(M+2) & if\ [(X_i(t)] < 0 \end{cases} \tag{4-41}$$

综上所述，离散化的 DQPIO 算法流程图如图 4.4 所示。

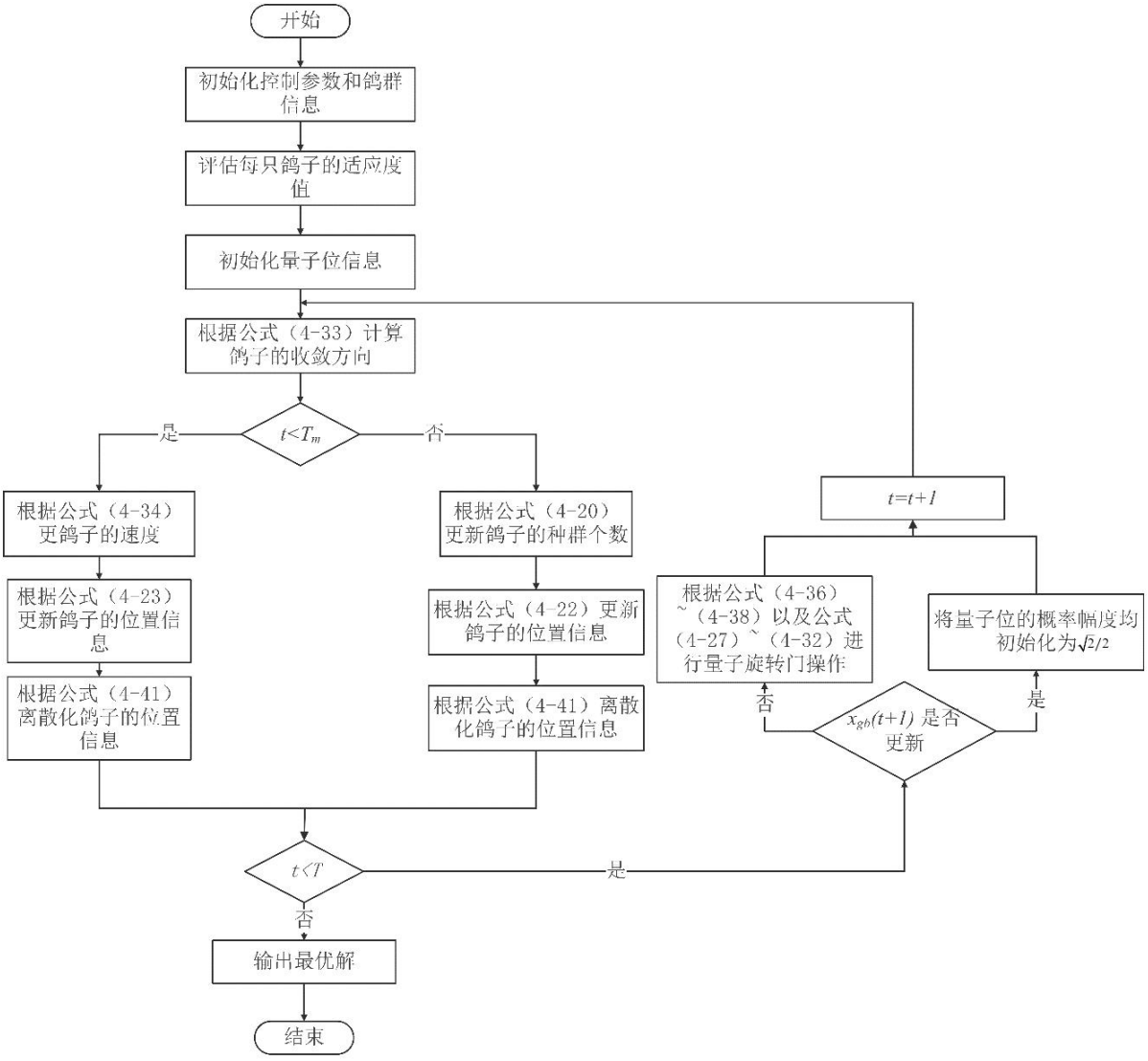


图 4.4 DQPIO 流程图

Fig. 4.4 Flow chart of DQPIO

为了进一步说明将 DQPIO 算法进行离散化的过程，现引入一个具体的例子。现有$[-0.9140734, -2.19864497, -1.67259765]$。其中，0 表示在本地计算，1-5 表示分别在雾节点 1、雾节点 2、雾节点 3、雾节点 4 和雾节点 5 执行。6 表示任务被分配到云端执行。根据公式 4-34 计算出更新后的速度。然后将更新后的速度进行取整操作，得到结果$[-1, 0, 2]$。按照公式 4-23 对位置进行更新，假设得到鸽子的位置为$[-3.121, 6.3587, 8.251]$，将其取整所得的结果为$[-3, 6, 8]$，经过公式 4-41 离散化的结果为$[4, 6, 1]$。该过程的流程图如图 4.5 所示。

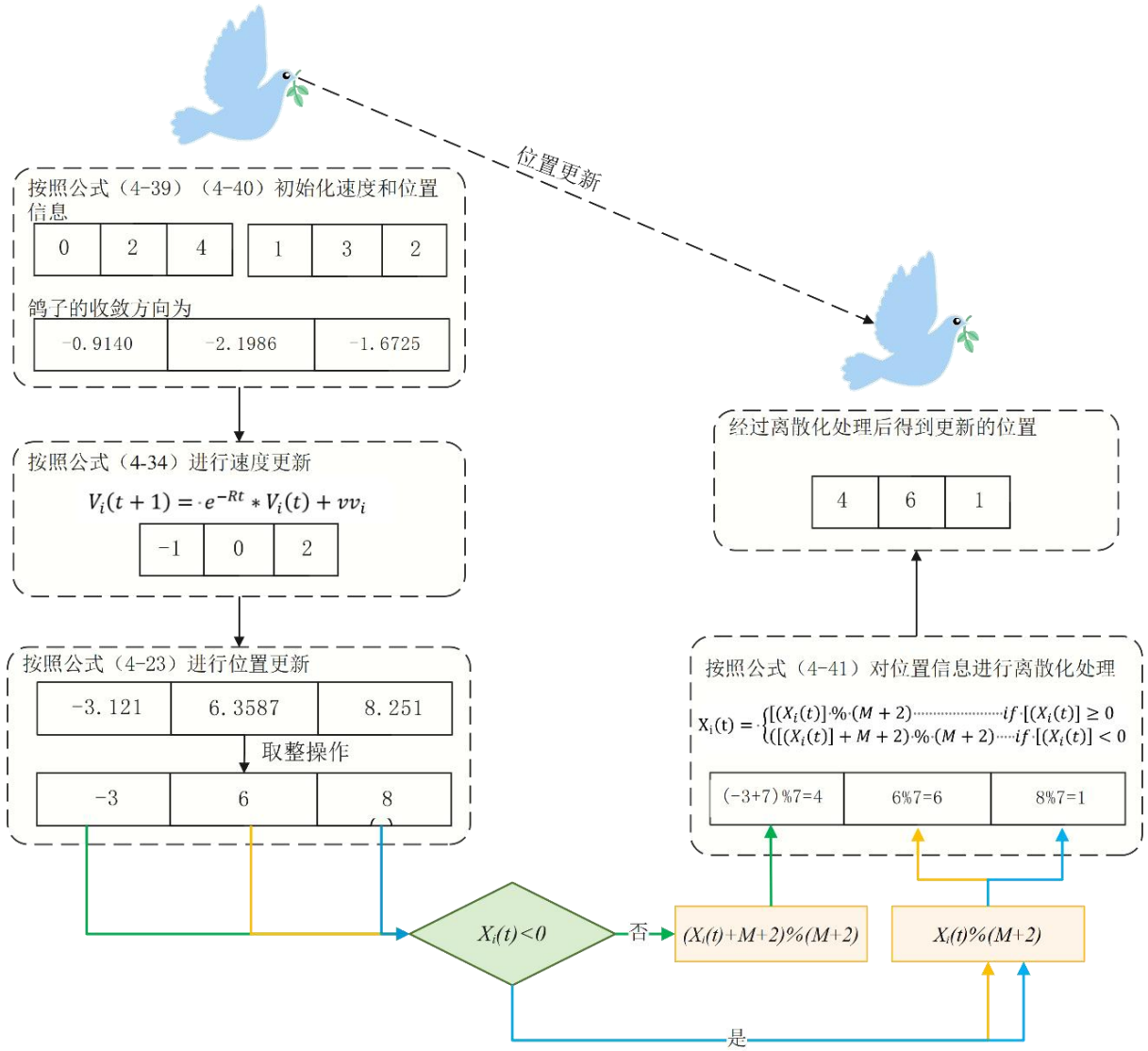


图 4.5 案例流程图

Fig. 4.5 Case flow chart

首先，根据公式 4-39 和 4-40 初始化速度和位置信息。在地图和指南针算子阶段，根据式 4-34 更新速度，如果全局最优解发生变化则将α_i和β_i被初始化为$\sqrt{2}/2$。随后根据公式

4-23 更新位置信息并利用公式 4-41 对位置信息进行离散化处理。在地标算子阶段，按照公式 4-20 至 4-22 进行位置更新，然后使用公式 4-41 对位置信息进行离散化处理并进行适应度值的更新。一直重复这一步骤直到迭代结束。

DQPIO 算法的伪代码描述如算法 4.2 所示。

算法 4.2 DQPIO 算法

Algorithm. 4.2 DQPIO algorithm

输入: 终端设备信息和任务数据信息

输出: 总任务时延和终端能耗

function DQPIO

初始化雾节点环境;

初始化参数设置;

使用泊松分布模拟终端设备的任务生成;

for t = 0 to T **do**

if $T < T_m$ **then**

根据公式 4-33 计算鸽子的收敛方向;

根据公式 4-34 更新速度;

根据公式 4-23 更新位置信息;

根据公式 4-41 将鸽子的位置信息进行离散化;

else

根据公式 4-20 更新鸽子的数量;

根据公式 4-22 更新位置信息;

根据公式 4-41 将鸽子的位置信息进行离散化;

end

if $x_{gb}(t+1)$更新 **then**

$\alpha_i = \sqrt{2}/2$;

$\beta_i = \sqrt{2}/2$;

else

根据公式 4-36 至 4-38 和公式 4-27 至 4-32 进行量子旋转门操作;

end

t=t+1;

输出任务总时延和终端能耗;

end function

4.4 仿真实验结果与分析

在本部分首先介绍了实验平台，并且设置了一些在实验中不可缺少的参数。其次，介绍了用来与 DQPIO 作对比的算法，并将最后的结果进行可视化。最后，在不同的实验环境下对仿真实验的结果进行分析。

4.4.1 实验参数设置

本文使用的操作系统是 Windows 10，所以使用的编译环境是 python 3.8。在仿真实验中，f_m设置为 5~10 之间的整数，f_{cloud}设置为 50GHz，crc_m和C_{local}均设置为 3~10 之间的整数，C_{local}被设置为 1GHz，Sto_lo_i被设置为 5MB。

为了验证所提算法的性能，使用天牛须搜索算法（BAS）、鸽群优化算法（PIO）、轮询算法（RR）和随机生成算法（Random）与本文提出的算法进行比较。为了避免偶然性，最后的结果使用了 6 次实验的平均结果。

4.4.2 实验结果分析

为了更好地评价算法的性能，在不同的实验环境下进行了实验，包括任务数据大小、终端设备数量、雾节点数量以及不同的权重系数。

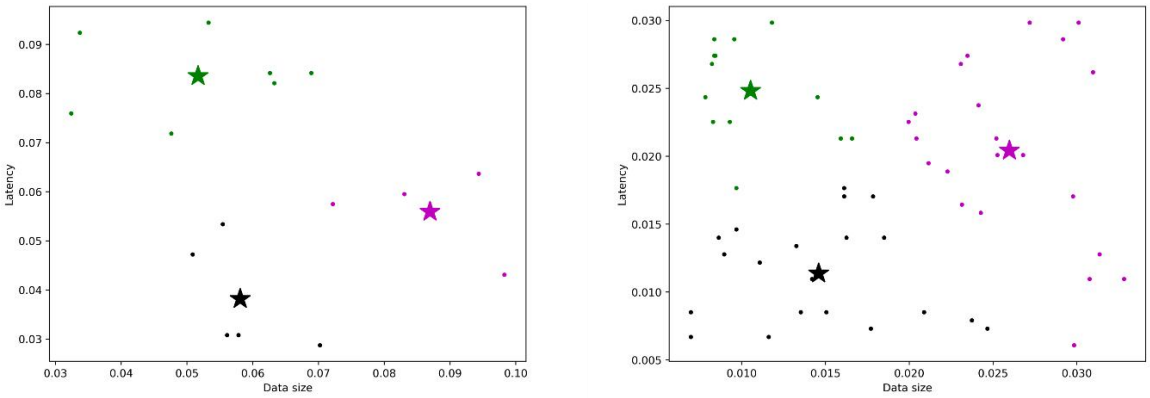
（1）任务优先级划分的结果与分析

图 4.6 展示了当终端设备的数量分别为 10 和 30 时产生的任务的优先级的划分结果。由图可知，所有的任务根据归一化之后的数据大小和时延被划分成了 3 簇。3 簇任务的类别分别由黑色、紫色和绿色所表示。每簇任务的中心点分别由相应颜色的星星符号所表示。根据 5.3 节中的定义，将已经分好的 3 簇任务的中心点按照纵轴的时延大小分别构建由低到高的优先级。图 4.6（a）是当终端数量为 10 的任务优先级划分结果。任务的优先级划分结果为 *priority1_TD10=['color of cluster':black, 'number of task': 5]*、*priority2_TD10=['color of cluster':purple, 'number of task': 4]*和 *priority3_TD10=['color of cluster':green, 'number of task': 7]*。图 4.6（b）展示的是当终端数量为 30 的任务优先级划分结果。任务的优先级划分结果为 *priority1_TD30=['color of cluster':black, 'number of task': 20]*、*priority2_TD30=['color of cluster':purple, 'number of task': 22]*和 *priority3_TD30=['color of cluster':green, 'number of task': 12]*。

（2）不同任务数据大小的结果与分析

终端设备产生的任务请求的数据量是不确定的。为了研究任务数据大小对目标优化效果的影响，考虑在不同任务数据大小下进行实验，包括 10-100KB、100-500KB、500-1000KB、1000-1500KB、1500-2000KB 和 2000-2500KB。这六种任务数据大小的总任务平均时延的实验结果如图 4.7 所示。图 4.7（a）-（f）代表了任务数据量由小到大的实验结果。图中的横轴代表五种不同的算法。任务执行的平均时延随着任务数据量的增加而增加。任务数据量越大，传输时延以及在本地节点或雾节点上的任务处理时间都会增大，所以平均时延也会有所增加。在六种不同任务数据量的环境下，DQPIO 算法和 PIO 算法的结果相对

较好，且 DQPIO 算法的结果优于 PIO 算法，具有更低的时延。RR 算法和 Random 算法的平均时延是最高的，BAS 次之。与群智能算法相比，BAS 只需要一个个体。因此，该算法所需的参数、计算量和复杂度都比群智能算法小得多。失去了种群的多样性，单一性使得 BAS 算法非常容易陷入局部最优。



（a） 终端设备=10　　（b） 终端设备=30

图 4.6 任务优先级划分结果；(a)终端设备=10；(b)终端设备=30

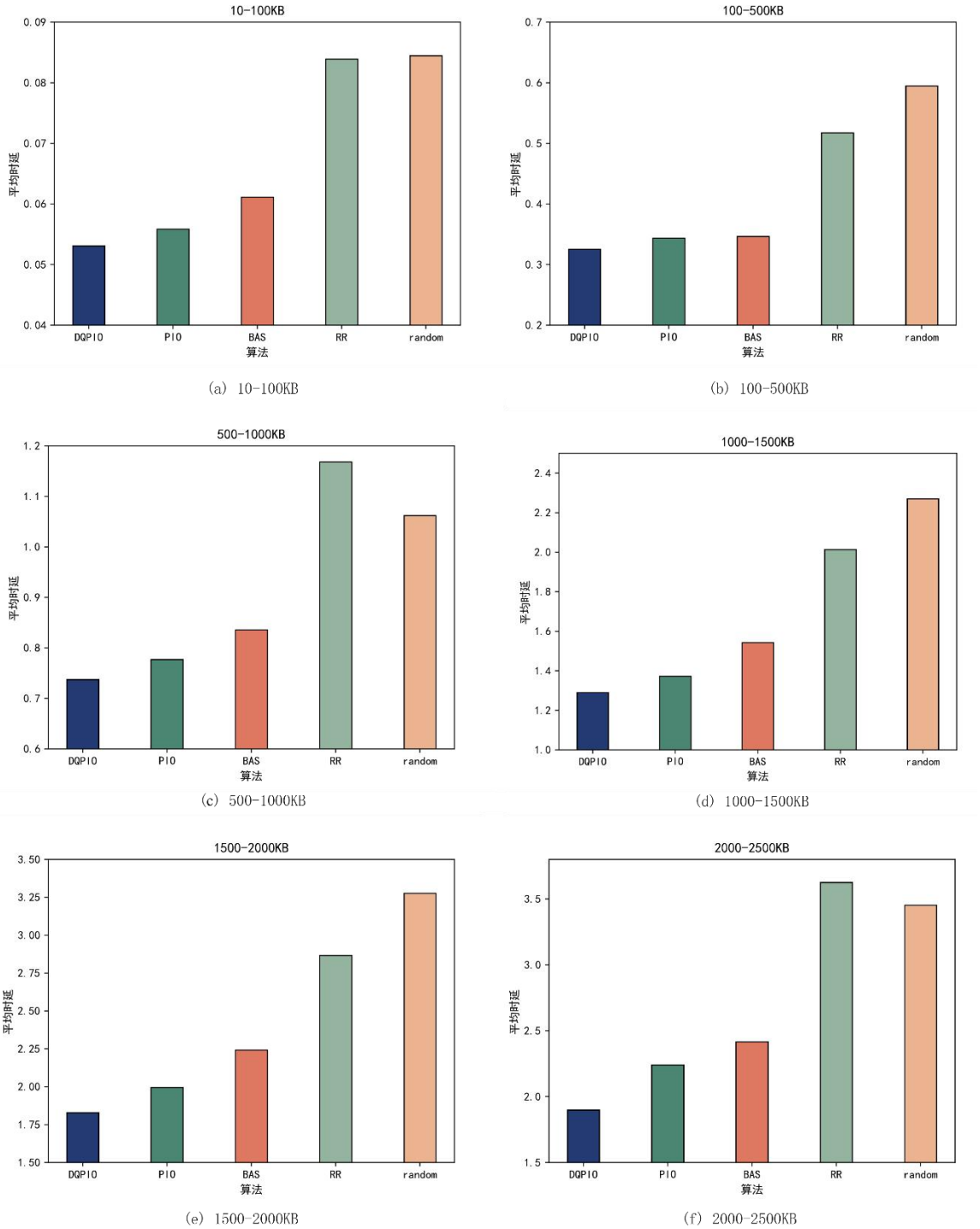
Fig. 4.6 Results of task priority division; (a) Terminal equipment =10; (b) Terminal equipment =30

（3）不同终端设备数量的结果和分析

图 4.8 展示了五种算法在不同终端设备数量(10 个、20 个、50 个、80 个、100 个、130 个和 160 个终端设备)下的任务平均时延的实验结果。图 4.8（a）-（f）表示终端设备数量由小到大的实验结果。图中的横轴代表五种不同的算法。任务的平均时延随着终端设备数量的增加而增加。这是因为随着终端设备数量的不断增加，任务数据量也随之增加，从而导致任务总时延的不断增加，最终导致了任务平均时延的增加。在六种不同终端设备数量的环境下，与其他四种算法相比，DQPIO 在任务平均时延方面有更好的实验结果。

（4）不同数量雾节点的结果与分析

雾节点数在一定程度上代表了雾层的计算能力。雾节点数量越多，说明雾层的计算能力越强。当雾节点的数量增多时，计算节点的环境复杂性随之提高，此时任务分配器的压力则会增大。因为环境复杂性的增加在一定程度上加大了寻找最优分配策略的难度。在本节中展示了雾层节点数分别为 4、8、12、16、24、32 时的实验结果。实验结果如图 4.9 所示。图 4.9（a）-（f）表示雾节点由小到大的实验结果。图中的横轴代表五种不同的算法。因为在设置的环境中，计算资源相对于任务数据总量是非常丰富的，所以随着雾节点数量的增加，任务总延迟和终端能耗并没有十分明显的变化。DQPIO 算法的实验结果优于其他三种算法。随机算法的实验结果在总任务延迟和终端能耗方面最差。



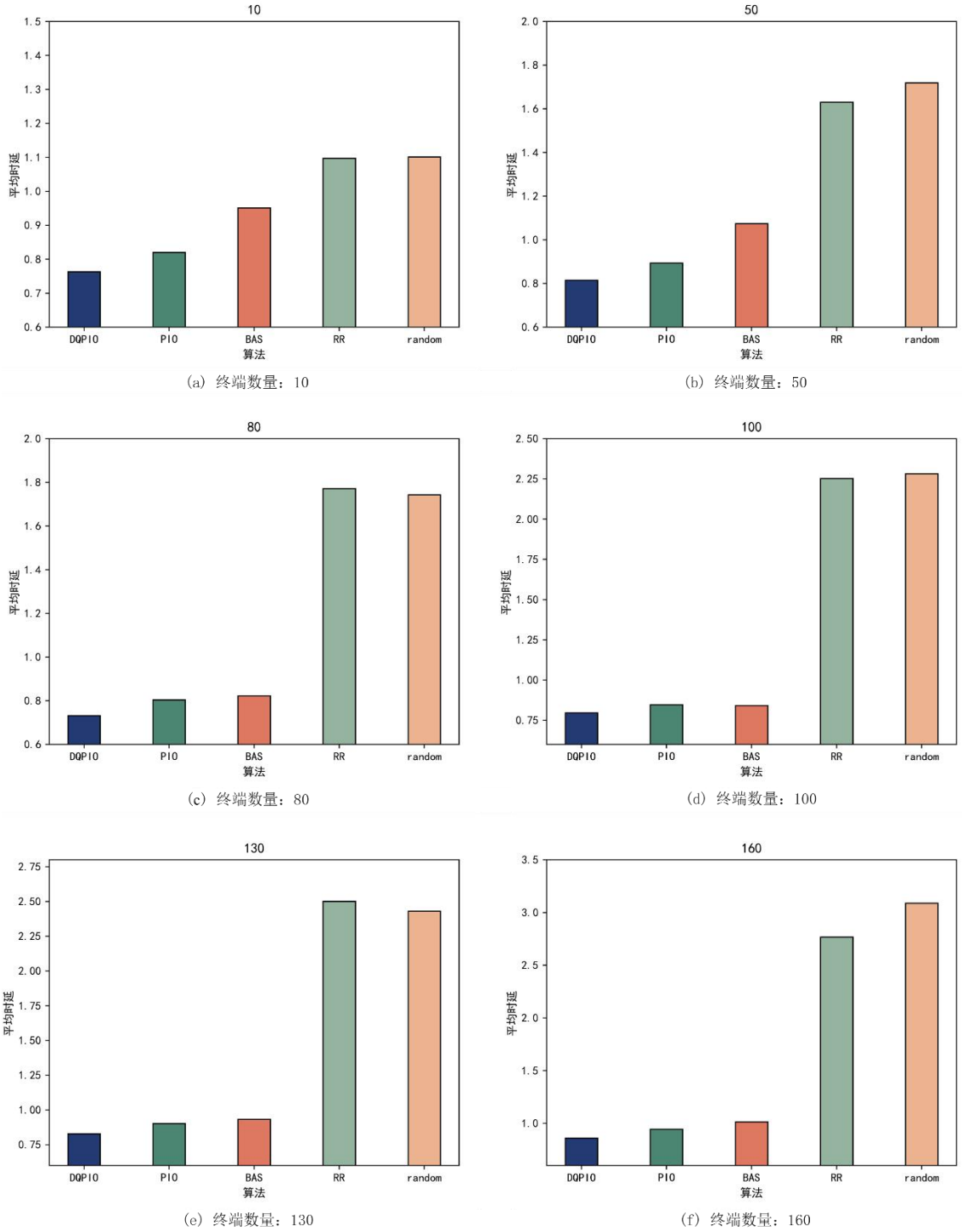
(a) 10-100KB (b) 100-500KB

(c) 500-1000KB (d) 1000-1500KB

(e) 1500-2000KB (f) 2000-2500KB

图 4.7 不同任务数据大小的平均延迟；(a) 10-100KB；(b)100-500KB；(c)500-1000KB；(d)1000-1500KB；(e)1500-2000KB；(f)2000-2500KB

Fig. 4.7 Average latency for different task data sizes; (a) 10-100KB; (b)100-500KB; (c)500-1000KB; (d)1000-1500KB; (e)1500-2000KB; (f)2000-2500KB



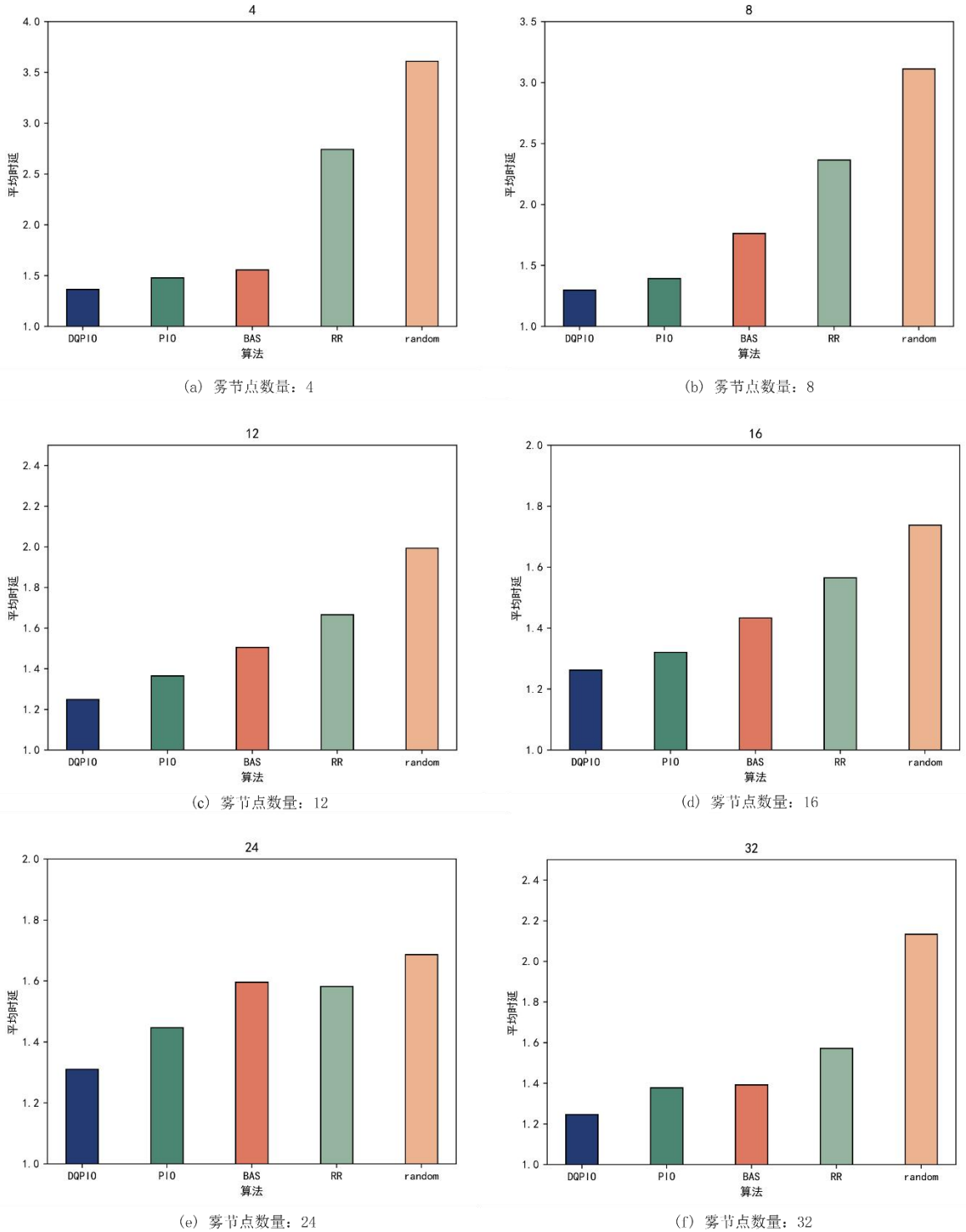
(a) 终端数量：10　(b) 终端数量：50

(c) 终端数量：80　(d) 终端数量：100

(e) 终端数量：130　(f) 终端数量：160

图 4.8 不同数量终端设备的平均延迟；(a) 终端数量=10；(b) 终端数量=50；(c) 终端数量=80；(d) 终端数量=100；(e) 终端数量=130；(f) 终端数量=160

Fig. 4.8 Average latency of different number of terminal devices; (a) Number of terminals =10; (b) Number of terminals =50; (c) Number of terminals =80; (d) Number of terminals =100; (e) Number of terminals =130; (f) Number of terminals =160



(a) 雾节点数量：4　(b) 雾节点数量：8

(c) 雾节点数量：12　(d) 雾节点数量：16

(e) 雾节点数量：24　(f) 雾节点数量：32

图 4.9 不同数量雾节点的平均延迟；(a) 雾节点数量=4；(b) 雾节点数量=8；(c) 雾节点数量=12；(d) 雾节点数量=16；(e) 雾节点数量=24；(f) 雾节点数量=32

Fig. 4.9 Average latency of different number of fog nodes; (a) Number of fog nodes =4; (b) Number of fog nodes =8; (c) Number of fog nodes =12; (d) Number of fog nodes =16; (e) Number of fog nodes =21; (f) Number of fog nodes =32

4.5 本章小结

本章首先以任务的平均时延作为优化目标，建立基于 SDN 的任务调度模型。其次，利用 K-means 聚类算法将一段时间内到达的任务按照最低时延要求和任务的计算量这两个属性进行聚类，从而生成多个任务簇，并依据最低时延要求分别为其设置优先级。然后，设置一种改进的量子鸽群优化算法按照优先级顺序分别对任务选择合适的计算节点进行分配。在该分配方法中，将每只鸽子的位置信息映射为对所有任务的分配策略。同时，对算法的编码方式进行离散化处理。最后，对提出的方法进行了仿真实验验证和评价。结果表明，与其他的方法相比，该方法在降低总时延方面相较于其他方法来说具有更良好的表现。

5 考虑终端移动性的任务动态分配方法

5.1 引言

终端设备最本质的特征就是移动，而雾节点所覆盖的范围是有限的，所以每次移动后移动终端的任务可以请求的雾节点的资源就受到了限制。若移动终端已经将其中的一个任务上传至某一雾节点进行执行，而此时终端设备离开了该雾节点的覆盖范围，但在该雾节点上还有未完成的任务，若不进行任何操作，该任务就会被中断，严重时会导致重要数据的丢失。因此，在对终端设备产生的任务进行分配时充分考虑终端设备的移动性是十分必要的。

当终端设备在雾环境下移动时，终端设备与雾节点之间的任务合理分配是一个难题，未完成的任务进行迁移以保持任务的连续性是另一个难题。为了解决这两个问题，首先构建了以最小化时延为目标的工作流任务分配优化模型。其次，为了在终端设备移动过程中保持任务的连续性，当终端设备移动到敏感区域时，根据目前所有计算节点的计算资源和任务的迁移成本等指标设计了任务迁移决策机制，并将任务迁移决策机制与 DQN 算法相结合用于寻找最优的任务分配策略。最后通过仿真实验，以 Q-learning 算法为基线算法在任务的数量、任务的数据量等方面验证了所提方法的有效性。

5.2 系统模型

5.2.1 工作流任务模型

一个工作流由一个有向无环图表示。工作流之间的约束关系使用 V 进行描述，$V = \{(w_{pred}, w_{succ}) | w_{pred}, w_{succ} \in W\}$，即任务$w_{succ}$的开始执行时间必须晚于$w_{pred}$的完成时间。如图 5.1 所示工作流所得出的任务集合 W 与约束关系集合 V 分别为 *W={w1,w2,w3,w4, w5, w6,w7}*，*V={(w1,w2),(w1,w3),(w1,w5),(w2,w4),(w3,w6),(w5,w7),(w4,w6),(w5,w7),(w4,w6),(w6,w7)}*。

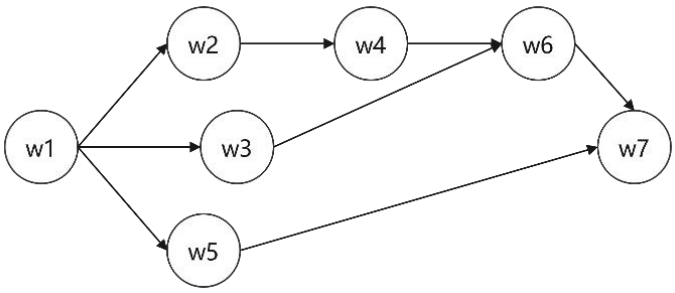


图 5.1 抽象工作流 DAG 图

Fig. 5.1 Abstract Workflow DAG

5.2.2 雾节点信号覆盖范围模型

对于雾节点的分布，一般分为正方形拓扑结构、六边形拓扑结构、随机拓扑结构。在网络较为稳定的特定场景下，六边形拓扑结构最为流行，因此本文雾节点分布采用正六边形的拓扑结构。雾节点分布如图 5.2 所示。相邻雾节点间均有一些重叠区域，在本文对各个区域的划分如图 5.3 所示，共分为三个区域，包括危险区、安全区、公共区。其中，景危险区以及公共区定义为敏感区域。

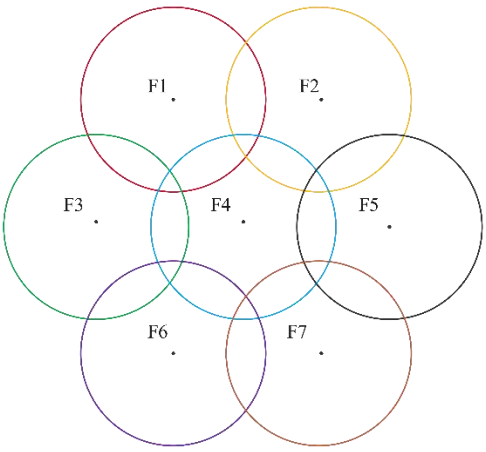


图 5.2 雾节点分布图

Fig. 5.2 The distribution of fog nodes

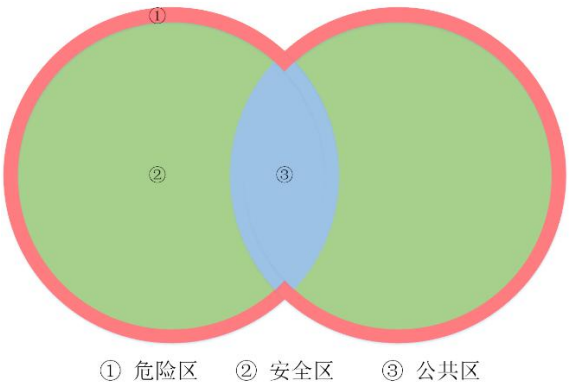


图 5.3 区域划分图

Fig. 5.3 The division map of the region

（1）危险区。危险区是某个雾节点覆盖范围的最外层，若移动终端进入至危险区，则认为传感器节点与雾节点之间的连接大概率会中断，此时依据第 5.3.1 节中的切换算法启动切换机制。

（2）安全区。若移动终端的位置停留在安全区，则认为传感器节点与雾节点之间的连接在大多数情况下不会中断，因此此时任务不发生中断，无需运行任务切换算法。

（3）公共区。公共区域是两个或多个雾节点之间的重叠区域，若移动终端进入到公共区，则根据第 5.3.1 节中的切换算法启动切换机制。

5.2.3 终端移动路径模型

当终端设备在某场景中移动时，首先将该场景转化为一个处于第一象限的平面图，随后将终端的移动路径中的每一个拐点都对应为一个坐标。故在终端设备移动的 n 段路径中由 $n+1$ 个具有偏序关系的坐标组成，即路径 P 被定义为$P = \{p_i | p_i = (x_i, y_i), i \in n+1\}$。其中，$p_i$表示路径 P 中的第i个坐标，x_i和y_i分别表示p_i的横坐标和纵坐标。

5.2.4 考虑终端移动性的工作流任务分配模型

在某一特定场景下，给定起点与终点，会有多条路径可由起点到达终点。在进行决策时，应结合终端设备的移动速度和位置、任务迁移代价以及任务完成时延分别给出不同的卸载决策下的时延模型。所构建的任务卸载时延模型由本地执行时延模型、雾节点执行时延模型、云服务器执行时延模型、任务迁移时延模型组成。相关变量的定义如表 5.1 所示。

表 5.1 文中使用的变量的定义

Tab. 5.1 Experimental parameter setting

符号	描述
W	工作流集合
w_i	工作流中的第 i 个任务
l_i	任务 i 的输入量
C_i	计算任务 i 所需要的周期数
f_{local}	移动终端的计算能力
f_{fog_j}	雾节点 j 的计算能力
f_{cloud}	云服务器的计算能力
nre_i	任务 i 剩余的数据量
Cre_i	任务 i 剩余的数据量所需要的周期数
n_{local}	在移动终端上执行的任务的数量
n_{fog}	在雾节点上执行的任务的数量
n_{cloud}	在云服务器上执行的任务的数量

（1）本地执行时延

当任务w_i在终端执行时，由于任务未进行传输，因此任务时延等于任务执行时间，其计算方式如公式 5-1 所示。

$$T_{local} = \sum_{i=1}^{n_{local}} \frac{l_i \times C_i}{f_{local}} \tag{5-1}$$

（2）雾节点执行时延

当任务w_i在雾节点 j 执行时，任务执行时延由数据发送时延、执行时延组成，其计算方式如公式 5-2 所示。

$$T_{fog} = \sum_{i=1}^{n_{fog}} T_{fog_j}^{lau} + \sum_{i=1}^{n_{fog}} \frac{l_j \times C_i}{f_{fog_j}} \tag{5-2}$$

（3）任务迁移执行时延

当执行任务迁移算法启动任务迁移机制时，任务会有上传至相邻雾节点 k、上传至云端两种决策，这两种决策的时延分别为如公式 5-3 和公式 5-4 所示。

$$\Delta T_{migr}^{fog} = \sum_{i=1}^{n_{re}} \frac{nre_i}{R_{fog_k}} + \sum_{i=1}^{n_{re}} \frac{nre_i \times Cre_i}{f_{fog_k}} \tag{5-3}$$

$$\Delta T_{migr}^{cloud} = \sum_{i=1}^{n_{re}} \frac{nre_i}{R_{cloud}} + \sum_{i=1}^{n_{re}} \frac{nre_i \times Cre_i}{f_{cloud}} \tag{5-4}$$

综上所述，目标优化问题的表示如公式 5-5 所示。

$$T_{total} = T_{local} + T_{fog} + \Delta T_{migr}^{fog} + \Delta T_{migr}^{cloud} \tag{5-5}$$

5.3 任务迁移决策机制与任务动态分配方法

5.3.1 任务迁移决策机制

当任务卸载至雾节点 j 执行时，此时移动会出现以下两种情况：

（1）若移动终端的位置进入至雾节点 j 与雾节点 j+1 的公共区域$A_{(j,j+1)}$时，启用任务迁移决策算法，如图 5.4 所示。设$y_1 = k_1 \times x + b_1$，$y_2 = k_2 \times x + b_2$，移动终端的位置 A 坐标为(x_{nowa}, y_{nowa})，移动后的位置坐标为(x_{movea}, y_{movea})，若$(y_1(x_{movea}) - y_{movea}) \times (y_1(x_{nowa}) - y_{nowa}) \leq 0$，则将任务打包发至雾节点 j+1。同样的，如果$(y_2(x_{movea}) - y_{movea}) \times (y_2(x_{nowa}) - y_{nowa}) \leq 0$，也将任务打包发至雾节点$j$+2。

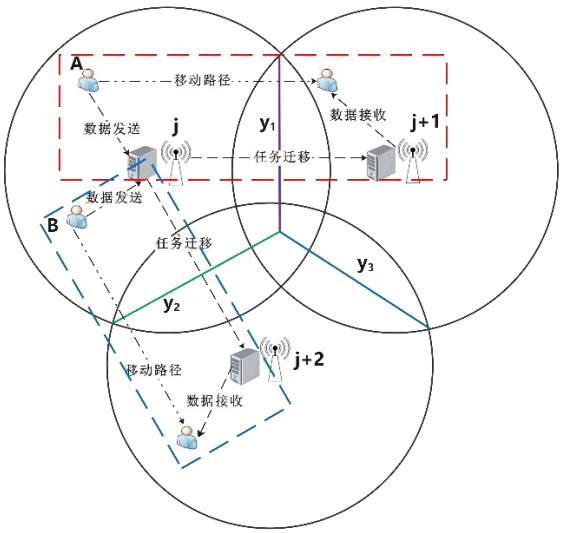


图 5.4 情况一的任务迁移决策图

Fig. 5.4 Task migration decision graph for Case 1

（2）若移动终端的位置进入至雾节点 j 的危险区域A_{edge}时，启用任务迁移决策算法，如图 5.5 所示。现设移动终端的位置 A 坐标为(x_{nowa}, y_{nowa})，雾节点 j 的位置坐标为$\left(x_{fog_j}, y_{fog_j}\right)$，雾节点 j 的覆盖范围为 10m，危险区域的内侧边界线距离雾节点 j 的覆盖范

围最外侧边界线直线距离 1m，计算$r_{act}=\sqrt{\left(x_{nowa}-x_{fog_j}\right)^2+\left(y_{nowa}-y_{fog_j}\right)^2}$，若 $9<r_{act}<10$，则将任务打包上传至云端。

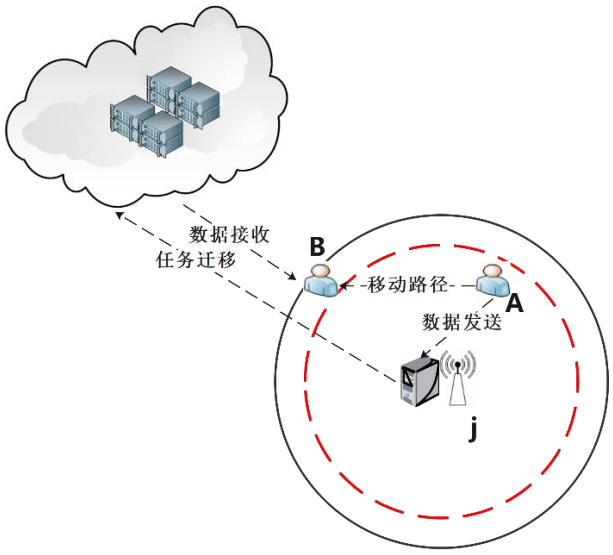


图 5.5 情况二的任务迁移决策图

Fig. 5.5 Task migration decision graph for Case 2

5.3.2 基于 DQN 算法的任务动态分配方法

DQN 算法由状态、动作和反馈奖励所构成。现设任务流中共有 N 个任务，M 个雾节点。其状态、动作和反馈奖励被定义为：

（1）状态：在每次调度完成后，分别将任务总时延和实际调度方案所需的各个雾节点的计算能力作为每次执行动作后的状态 $S(s_0,s_1,s_2,...,s_M)$，根据s_0的大小可判断每次调度方案时候最小化系统的总时延，根据$(s_1,s_2,...,s_M)$的大小来判断是否超越了雾节点的计算能力，以及是否充分利用了雾节点的计算能力。

（2）动作：定义由N维列向量构成的任务决策矩阵$\vec{\lambda}=\{\lambda_1,\lambda_2,...,\lambda_i,...,\lambda_N\}$，$\vec{\lambda_i}\in\{0,1\}$。当$\vec{\lambda_i}=1$ 时表示任务 i 被分配到雾节点上执行。当$\vec{\lambda_i}=0$ 时表示任务 i 在本地执行。任务决策矩阵$\vec{\lambda}$作为 DQN 算法中的动作 a。

（3）奖励：奖励是用来评价所采取的动作的好坏程度的，如果采取的动作得到的总时延低于上一次所得到的总时延，那么将奖励值置为 1 否则将奖励值设为 0。如果分配策略中存在超过雾节点或者终端设备的计算能力的情况，则将奖励值置为-1。

基于 DQN 算法与任务迁移决策机制的任务分配方法（WTMDQN）的流程图如图 5.6 所示。首先初始化一个神经网络，将状态 S 作为神经网络的输入，将所有动作对应的 Q 值作为神经网络的输出。然后利用当前的状态和神经网络使用ε-greedy 策略选择一个动作，

在执行动作的过程中判断该动作是否会导致任务中断，如果是则启用任务迁移决策机制，更新动作与状态信息。随后观察奖励并且进入下一个状态。将当前的状态、动作、奖励以及下一个状态存入到经验回放记忆池中。接下来进行网络训练，首先在经验回放池中随机选取一些样本，计算目标 Q 值。其计算方法如公式 5-6 所示。

$$Q_{target} = r + \gamma \times \max\left(Q(s^{'}, a^{'}, \theta_{target})\right) \tag{5-6}$$

其中，r表示执行完动作 a 之后的所获得的奖励，γ表示衰减系数，$Q(s^{'}, a^{'}, \theta_{target})$表示目标网络估计的下一个状态下各个动作的 Q 值。计算当前状态下采取各个动作的损失，损失函数采如公式 5-7 所示。随后使用梯度下降算法来更新深度神经网络的权重，以最小化损失函数。一直重复上述过程，直至达到预定的训练次数，或者达到预定的性能要求。

$$L = (\theta_{target} - Q(s, a, \theta))^2 \tag{5-7}$$

5.4 仿真实验结果与分析

5.4.1 实验参数设置

本节使用 Python 以及 Tensorflow 框架来实现所提出的方法，并通过不同环境下的仿真实验对所提方法的性能进行了评价。在实验中，任务数据大小服从均匀分布 $l_i \sim U(100,500)$，并计算所需的周期数据服从均匀分布$C_i \sim U(100,500)$，复杂的高斯白噪声信道的带宽是 10MHz，雾中节点的计算能力是 1GHz,云服务器的计算能力被定义为 1.6GHz。随机生成点作为移动终端的移动路径。

5.4.2 实验结果分析

为了验证所提方法的性能，选用 Q-learning 算法作为基线算法与之进行对比。同时，在任务的数量、任务的数据量以及雾节点的数量这三种不同的情况下分别对所提方法进行了验证以保证所提方法的泛化能力。

（1）任务数量与时延的关系

图 5.7 展示了任务的数量与任务的总时延之间的关系。实验选取的任务数量分别为 10 个、15 个、20 个、25 个、30 个，图中横轴表示任务的数量，纵轴表示系统为完成这些任务所产生的时延。随着任务数量的不断增大，完成任务所产生的时延也在不断上升。当任务数量为 10 时，Q-learning 算法与 WTDQN 算法具有相似的时延，由于任务的数量过少，因此 WTDQN 还未展现出其优势。随着任务数量的上升，与 Q-learning 算法相比，WTDQN 算法在寻找最低时延上具有更加优秀的表现。

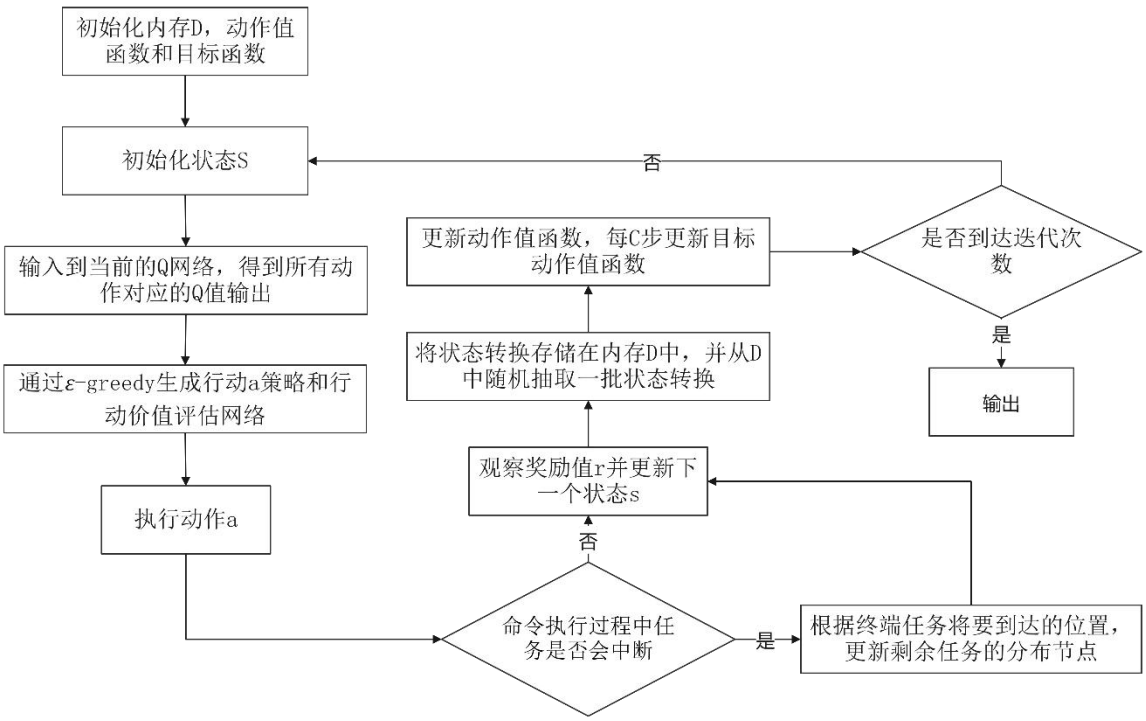


图 5.6 WTMDQN 的流程图

Fig. 5.6 Flow chart of WTMDQN

（2）任务数据量与时延的关系

对于终端设备来说，其产生的任务的数据量并不总是保持在一个小范围内，因此，选取了 10-100KB、100-500KB、500-1000KB、1000-1500KB 这四种不同范围的数据量来对 WTDQN 算法进行评估。如图 5.8 所示，随着任务数据量的不断增大，其传输任务的时延、执行任务的时延以及迁移任务的时延都会在一定程度上有所增加，因此，无论是 Q-learning 算法还是 WTDQN 算法在图中均为上升趋势。当任务的数据量过小时，WTDQN 所得的时延略小于 Q-learning 算法，但是随着任务数据量的增大，两算法之间的差距也逐渐显现出来，简言之，随着任务数据量的增大，WTDQN 算法的优势越显著。

（3）雾节点数量与时延的关系

图 5.9 展现了固定任务数量为 12，任务的数据量在 100-500KB 时，不同雾节点的数量（3 个、5 个、7 个）的情况下，使用两种任务分配算法所得时延的结果。由图可以看出，随着雾节点数量的增加，Q-learning 算法和 WTDQN 算法均得到了逐渐降低的时延。同时无论雾节点的数量为多少，所提算法与其他基线算法相比都取得了最好的表现。

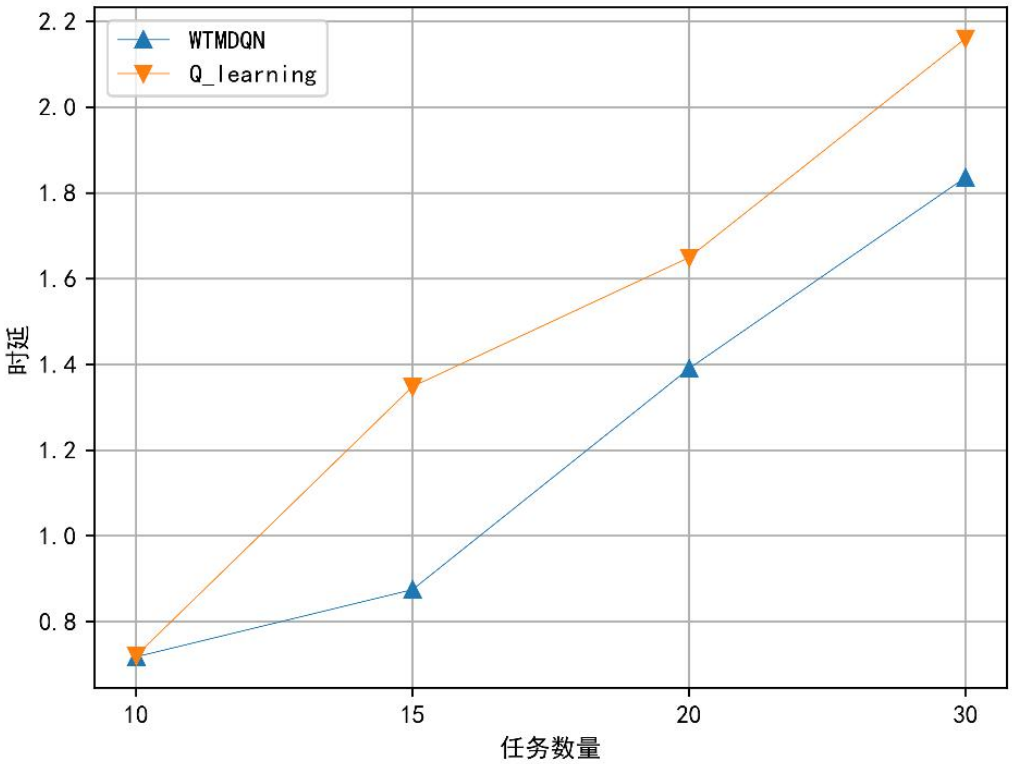


图 5.7 任务数量与时延的关系

Fig. 5.7 Relationship between the number of tasks and the latency

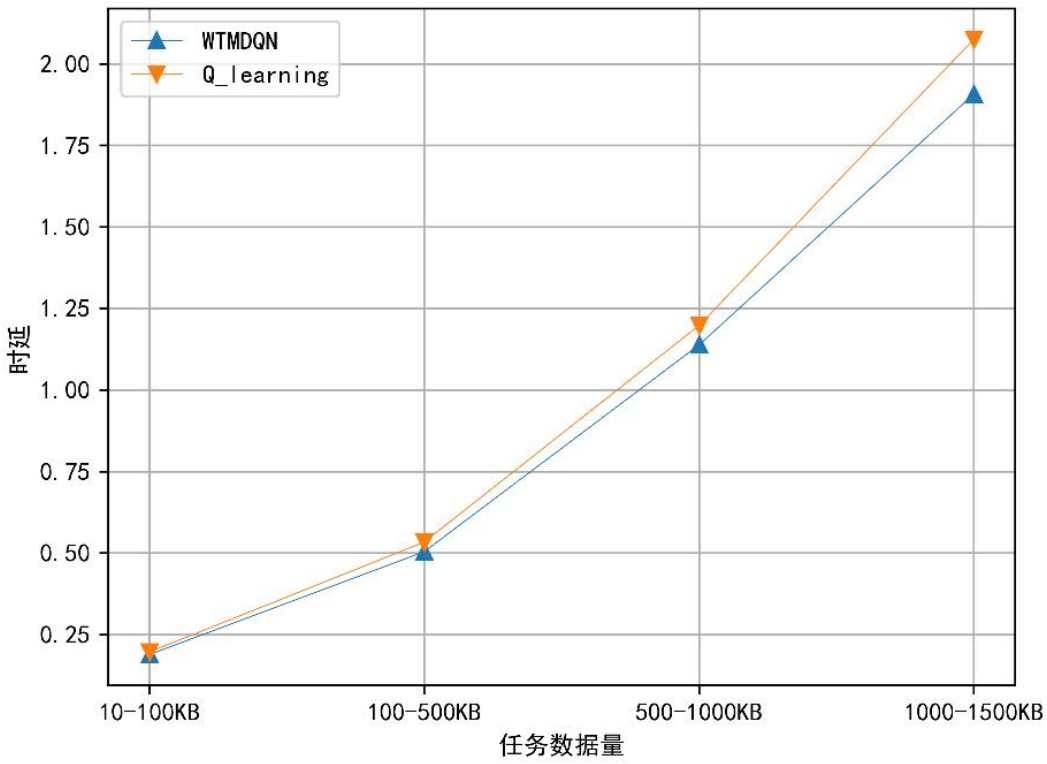


图 5.8 任务数据量与时延的关系

Fig. 5.8 The relationship between task data and latency

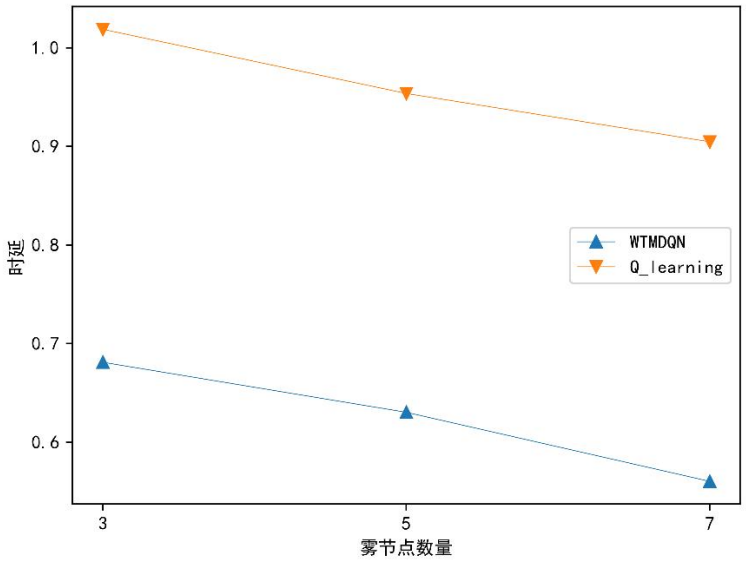


图 5.9 雾节点数量与时延的关系

Fig. 5.9 The relationship between the number of fog nodes and the latency

5.5 本章小结

本章首先介绍了以最小化延时为目标的工作流任务分配优化模型。其次，介绍了考虑计算节点的计算能力、任务是否需要迁移以及迁移成本的任务迁移决策机制。当终端设备移动至敏感区域时，激活任务迁移决策机制执行任务迁移。然后介绍了基于 DQN 算法的任务分配方法。最后，通过仿真实验验证了该方法的有效性，结果表明该方法合理有效地降低了时延。

6 总结与展望

6.1 工作总结

随着智能设备的不断增多，数据流量爆发式增长，这为云计算带来了巨大的压力，而雾计算作为云计算的补充与延伸在一定程度上减轻了云服务器的压力。而将云雾协作的架构应用到构建任务分配这一问题可以突破所有计算节点的计算资源利用率不充足的局限性。因此本文讨论了云雾协作环境下的任务动态分配问题。主要的研究工作如下：

（1）以时延和能耗为联合优化目标，在多个终端设备、单雾节点的环境下对任务进行合理分配。首先将该任务分配问题建立为 0-1 整数规划模型，然后提出一种二进制天牛须搜索算法对该问题进行求解，该算法在原始的天牛须算法的基础上引入了 Sigmoid 函数，以 0.5 为界，将天牛须更新后的位置映射到 0 或者 1。最后，以粒子群优化算法、轮询算法、随机算法、全部卸载方法以及全部在终端设备执行方法为基线方法进行仿真实验，实验结果表明，所提算法无论在何种环境下均取得了最优的表现，同时也验证了权重对策略结果的影响，即则相较于能耗来说，算法对时延的优化力度更大，反之亦然。

（2）以平均时延为优化目标，在多个终端设备、多个雾节点的环境下对任务进行合理分配。将 SDN 的集中控制的优势应用到云-雾-端架构中，同时提出了一种任务优先级划分方法应用在任务分配操作之前。该方法采用 K-means 聚类算法，将任务划分为不同的等级。随后，提出了基于改进的量子鸽群算法的任务分配方法，对划分好优先级的任务集进行任务分配。最后进行仿真实验，实验结果表明，在不同的场景下所提方法与轮询算法、天牛须搜索算法、鸽群算法以及随机生成算法相比均取得了最好的表现。

（3）以时延为优化目标，研究了考虑终端设备的移动性的任务分配问题。首先将终端设备的任务构建为一个有向无环图，并介绍了以最小化延时为目标的工作流任务分配优化模型。随后介绍了任务迁移决策机制以及基于 DQN 算法的任务分配方法。最后对所提的方法进行了仿真实验验证，实验结果表明，以 Q-learning 算法为基线算法在任务的数量、任务的数据量等方面验证了所提方法的有效性。

6.2 工作展望

本文对云雾协作环境下的任务分配问题进行了研究，但是由于本文所建立模型均为粗粒度，所以仍然存在许多不完善的地方。未来的工作方向总结如下：

（1）将研究的目光由粗粒度转向细粒度。由全部卸载转为部分卸载，将终端设备的任务按照最适合的比例划分，然后分配到不同的雾节点或者云服务器进行处理以得到更低

的时延。将对任务的传输时延造成影响的参数考虑到模型中，例如信道分配、信道中的噪声干扰等，使得构建的模型更加实际化，以得到更加符合实际的解决方案。因此，构建细粒度模型是一个值得被研究的问题。

（2）在第五章考虑终端移动性的任务分配研究中，对移动终端的移动路径是提前设定好的，而在现实生活中，终端设备的移动路径往往是随机的，是不可提前预知的。随机移动相对于固定移动路径来说难度增大了许多，如何在终端设备随机移动的情况下进行任务合理分配是一个十分有意义的研究。

参考文献

[1] Adnan K, Muhammad S. Service architecture models for fog computing: A remedy for latency issues in data access from clouds [J]. KSII Transactions on Internet and Information Systems, 2017, 11(5): 2310-2345.

[2] Hajam S S, Sofi S A. IoT-Fog architectures in smart city applications: A survey [J]. China Communications, 2021, 18(11): 117-140.

[3] Abbasi M, Yaghoobikia M, Rafiee M, et al. Energy-efficient workload allocation in fog-cloud based services of intelligent transportation systems using a learning classifier system [J]. Iet Intelligent Transport Systems, 2020, 14(11): 1484-1490.

[4] Zhu L, LI M, Zhang Z, et al. Privacy-Preserving Authentication and Data Aggregation for Fog-Based Smart Grid [J]. IEEE Communications Magazine, 2019, 57(6): 80-85.

[5] Bonomi F, Milito R A, Zhu J, et al. Fog computing and its role in the internet of things [C], in Proceedings of the MCC '12, 2012, 13-16.

[6] Amin S U, Hossain M S. Edge Intelligence and Internet of Things in Healthcare: A Survey [J]. IEEE Access, 2021, 9: 45-59.

[7] Devarajan M, Ravi L. Intelligent cyber-physical system for an efficient detection of Parkinson disease using fog computing [J]. Multimedia Tools and Applications, 2019, 78(23): 32695-32719.

[8] Rocha Filho G P, Brandao A H, Nobre R A, et al. HOsT: Towards a Low-Cost Fog Solution via Smart Objects to Deal with the Heterogeneity of Data in a Residential Environment [J]. Sensors, 2022, 22(16): 6257.

[9] Miah M S, Schukat M, Barrett E. An enhanced sum rate in the cluster based cognitive radio relay network using the sequential approach for the future Internet of Things [J]. Human-Centric Computing and Information Sciences, 2018, 8.

[10] Deng Y, Chen Z, Zhang D, et al. Workload scheduling toward worst-case delay and optimal utility for single-hop Fog-IoT architecture [J]. Iet Communications, 2018, 12(17): 2164-2173.

[11] 李智勇,王琦,陈一凡等. 车辆边缘计算环境下任务卸载研究综述[J]. 计算机学报, 2021, 44 (05): 963-982.

[12] Raouf H, Abdallah R, Soliman H Y M, et al. Mobility-Aware Task Offloading Enhancement in Fog Computing Networks [C] , The 8th International Conference on Advanced Machine Learning and Technologies and Applications (AMLTA 2022), 2022, 569-580.

[13] Rabie A H, Ali S H, Ali H A, et al. A fog based load forecasting strategy for smart grids using big electrical data [J]. Cluster Computing-the Journal of Networks Software Tools and Applications, 2019, 22(1): 241-270.

[14] Kishor A, Chakarbarty C. Task Offloading in Fog Computing for Using Smart Ant Colony Optimization [J]. Wireless Personal Communications, 2021, 127(2): 1683-1704.

[15] Azizi S, Shojafar M, Abawajy J, et al. Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach [J]. Journal of Network and Computer Applications, 2022, 201: 103333.

[16] 谭友钰, 陈蕾, 周明拓, et al. 动态雾计算网络中基于在线学习的任务卸载算法 [J]. 中国科学院大学学报, 2020, 37(05): 688-698.

[17] Hussain S M, Begh G R. Hybrid heuristic algorithm for cost-efficient QoS aware task scheduling in fog–cloud environment [J]. Journal of Computational Science, 2022, 64: 101828.

[18] 钟云峰, 宋伟宁. 基于云边协同多任务计算卸载策略 [J]. 计算机技术与发展, 2022, 32(04): 69-73.

[19] Wang K, Zhou Y, Li J, et al. Energy-Efficient Task Offloading in Massive MIMO-Aided Multi-Pair Fog-Computing Networks [J]. IEEE Transactions on Communications, 2021, 69(4): 2123-2137.

[20] 葛欣炜, 段聪颖, 陈思光. 基于雾计算的能耗最小化公平计算迁移研究 [J]. 计算机技术与发展, 2022, 32(03): 107-113.

[21] Ghanavati S, Abawajy J, Izadi D. An Energy Aware Task Scheduling Model Using Ant-Mating Optimization in Fog Computing Environment [J]. IEEE Transactions on Services Computing, 2022, 15(4): 2007-2017.

[22] Movahedi Z, Defude B, Hosseininia A M. An efficient population-based multi-objective task scheduling approach in fog computing systems [J]. Journal of Cloud Computing, 2021, 10(1): 1-31.

[23] 张先超, 任天时, 赵耀, et al. 移动边缘计算时延与能耗联合优化方法 [J]. 电子科技大学学报, 2022, 51(05): 737-742.

[24] Fakhfakh F, Cheikhrouhou S, Dammak B, et al. Multi-objective approach for scheduling time-aware business processes in cloud-fog environment [J]. The Journal of Supercomputing, 2022, 79: 8153-8177.

[25] Singh P, Singh R. Energy-Efficient Delay-Aware Task Offloading in Fog-Cloud Computing System for IoT Sensor Applications [J]. Journal of Network and Systems Management, 2021, 30: 14.

[26] 吴学文, 廖婧贤. 云边协同系统中基于博弈论的资源分配与任务卸载方案 [J]. 系统仿真学报, 2022, 34(07): 1468-1481.

[27] Mukherjee M, Shu L, Wang D. Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges [J]. IEEE Communications Surveys & Tutorials, 2018, 20(3): 1826-1857.

[28] Kunal S, Saha A, Amin R. An overview of cloud‐fog computing: Architectures, applications with security challenges [J]. Security and Privacy, 2019, 2(4): e72.

[29] Hosseinioun P, Kheirabadi M, Kamel Tabbakh S R, et al. A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm [J]. Journal of Parallel and Distributed Computing, 2020, 143: 88-96.

[30] Sood S K, Kaur A, Sood V. Energy efficient IoT-Fog based architectural paradigm for prevention of Dengue fever infection [J]. Journal of Parallel and Distributed Computing, 2021, 150: 46-59.

[31] Ganek A G, Corbi T A. The dawning of the autonomic computing era [J]. IBM Systems Journal, 2003, 42(1): 5-18.

[32] Naha R K, Garg S, Georgakopoulos D, et al. Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions [J]. IEEE Access, 2018, 6: 47980-48009.

[33] Roman R, Lopez J, Mambo M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges [J]. Future Generation Computer Systems, 2018, 78: 680-698.

[34] Liu L, Chang Z, Guo X, et al. Multiobjective Optimization for Computation Offloading in Fog Computing [J]. IEEE Internet of Things Journal, 2018, 5(1): 283-294.

[35] Meng X, Wang W, Zhang Z. Delay-Constrained Hybrid Computation Offloading With Cloud and Fog Computing [J]. IEEE Access, 2017, 5: 21355-21367.

[36] Lyu X, Tian H, Sengul C, et al. Multiuser Joint Task Offloading and Resource Optimization in Proximate Clouds [J]. IEEE Transactions on Vehicular Technology, 2017, 66(4): 3435-3447.

[37] Liu M, Liu Y. Price-Based Distributed Offloading for Mobile-Edge Computing With Computation Capacity Constraints [J]. IEEE Wireless Communications Letters, 2018, 7(3): 420-423.

[38] Rashedi E, Nezamabadi-Pour H, Saryazdi S. GSA: a gravitational search algorithm[J]. Information Sciences, 2009, 179(13): 2232-2248.

[39] Shah-Hosseini H. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm[J]. International Journal of Bio-inspired Computation, 2009, 1(1-2): 71-79.

[40] Kirkpatrick S, Gelatt Jr C D, Vecchi M P. Optimization by simulated annealing[J]. science, 1983, 220(4598): 671-680.

[41] Chaudhary D, Kumar B. Cloudy GSA for load scheduling in cloud computing[J]. Applied Soft Computing, 2018, 71: 861-871.

[42] Kalra M, Singh S. Application of intelligent water drops algorithm to workflow scheduling in cloud environment[C], the 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2017: 1-7.

[43] Zhang Y, Sun J. Novel efficient particle swarm optimization algorithms for solving QoS‐demanded bag of tasks scheduling problems with profit maximization on hybrid clouds[J]. Concurrency and Computation: Practice and Experience, 2017, 29(21): e4249.

[44] Hashemi S M, Sahafi A, Rahmani A M, et al. GWO-SA: Gray Wolf Optimization Algorithm for Service Activation Management in Fog Computing[J]. IEEE Access, 2022, 10: 107846-107863.

[45] Yu J, Wang M, JH Y, et al. A new approach for task managing in the fog-based medical cyber-physical systems using a hybrid algorithm[J]. Circuit World, 2021: 1-11.

[46] Wang B, Wu P, Arefzaeh M. A new method for task scheduling in fog‐based medical healthcare systems using a hybrid nature‐inspired algorithm[J]. Concurrency and Computation: Practice and Experience, 2022, 34(22): e7155.

[47] Huang C, Wang H, Zeng L, et al. Resource scheduling and energy consumption optimization based on Lyapunov optimization in fog computing[J]. Sensors, 2022, 22(9): 3527.

[48] Subramoney D, Nyirenda C N. Multi-Swarm PSO Algorithm for Static Workflow Scheduling in Cloud-Fog Environments[J]. IEEE Access, 2022, 10: 117199-117214.

[49] Ning B, Gu Q, Wang Y. Research based on effective resource allocation of improved SFLA in cloud computing[J]. International Journal of Grid and Distributed Computing, 2016, 9(3): 191-198.

[50] Abdullahi M, Ngadi M A. Symbiotic organism search optimization based task scheduling in cloud computing environment[J]. Future Generation Computer Systems, 2016, 56: 640-650.

[51] Lam A Y S, Li V O K. Chemical-reaction-inspired metaheuristic for optimization[J]. IEEE transactions on evolutionary computation, 2009, 14(3): 381-399.

[52] Melin P, Astudillo L, Castillo O, et al. Optimal design of type-2 and type-1 fuzzy tracking controllers for autonomous mobile robots under perturbed torques using a new chemical optimization paradigm[J]. Expert Systems with Applications, 2013, 40(8): 3185-3195.

[53] Yan C, Luo H, Hu Z. Scheduling deadline-constrained scientific workflow using chemical reaction optimisation algorithm in clouds[J]. International Journal of Embedded Systems, 2018, 10(5): 378-393.

[54] Szabo C, Sheng Q Z, Kroeger T, et al. Science in the cloud: Allocation and execution of data-intensive scientific workflows[J]. Journal of Grid Computing, 2014, 12: 245-264.

[55] Liu Y, Shu W, Zhang C. A Parallel Task Scheduling Optimization Algorithm Based on Clonal Operator in Green Cloud Computing[J]. J. Commun., 2016, 11(2): 185-191.

[56] Gedeon J, Brandherm F, Egert R, et al. What the fog? edge computing revisited: Promises, applications and future challenges[J]. IEEE Access, 2019, 7: 152847-152878.

[57] Dai Y, Zhang K, Maharjan S, et al. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond[J]. IEEE Transactions on Vehicular Technology, 2020, 69(10): 12175-12186.

[58] Kiran N, Pan C, Wang S, et al. Joint resource allocation and computation offloading in mobile edge computing for SDN based wireless networks[J]. Journal of Communications and Networks, 2019, 22(1): 1-11.

[59] Hossain M S, Nwakanma C I, Lee J M, et al. Edge computational task offloading scheme using reinforcement learning for IIoT scenario[J]. ICT Express, 2020, 6(4): 291-299.

[60] Huang L, Feng X, Zhang C, et al. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing[J]. Digital Communications and Networks, 2019, 5(1): 10-17.

[61] Sarkar I, Adhikari M, Kumar S, et al. Deep reinforcement learning for intelligent service provisioning in software-defined industrial fog networks[J]. IEEE Internet of Things Journal, 2022, 9(18): 16953-16961.

[62] Jiang X, Li S. BAS: beetle antennae search algorithm for optimization problems. 2017[J]. arXiv preprint arXiv:1710.10724.

[63] Duan H, Qiao P. Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning[J]. International journal of intelligent computing and cybernetics, 2014, 7(1): 24-37.

硕士专业学位论文

中图分类号：TP391　　密　　级：公开

U D C：004.8　　单位代码：10452