哈爾濱工業大學

本科毕业论文（ 设计 ） 中期报告

题　目：面向公平性的云边微服务系统部署方法

专　　业　软件工程

学　　生　付书煜

学　　号　2021111824

指导教师　贺祥

日　　期　2025 年 3 月 5 日

哈尔滨工业大学教务处制

目 录

1 论文（设计）工作是否按开题报告预定的内容及进度安排进行 1

2 已完成的研究工作及成果 1

2.1 云-边协同微服务系统的设计与建模 1

2.1.1 场景描述 1

2.1.2 系统建模 2

2.1.3 约束条件 3

2.1.4 优化目标 4

2.2 问题求解 5

2.2.1 问题分析 5

2.2.2 算法实现思路 5

2.2.3 算法具体流程 5

2.2.4 算法复杂度分析 7

2.2.4 对比算法 7

2.3 实验平台开发 7

2.3.1 功能描述 7

2.3.2 需求分析 8

2.3.3 设计与实现 8

2.3.4 模块功能流程 8

2.3.5 模块展示 9

2.3.6 平台功能模块与本研究的结合 10

2.4 实验 10

2.4.1 实验设计 10

2.4.2 实验过程 11

2.4.3 实验结果与分析 11

3 后期拟完成的研究工作及进度安排 16

4 存在的问题与困难 16

5 论文按时完成的可能性 16

6 主要参考文献 16

1 论文（设计）工作是否按开题报告预定的内容及进度安排进行

开题报告中的工作进度安排如表 1-1 所示：

表 1-1 工作进度安排表

工作安排	周数	起止时间
系统建模与设计阶段	3	2024.11.20-2024.12.15
算法设计与实现	5	2024.12.16-2025.01.19
实验与数据收集	4	2025.01.20-2025.02.16
敏感性分析与模型调优	3	2025.02.17-2025.03.09
成本效益分析与优化验证	4	2025.03.10-2025.04.06
结题，论文编写	2	2025.04.07-2025.04.20

目前为止，基本按照开题报告中的规划进行，已完成了系统建模、算法设计以及部分实验。

2 已完成的研究工作及成果

在视频流等实时服务的实际应用中，用户群体广泛分布在不同的地理位置，涵盖城市中心用户以及偏远地区的用户，他们通过各类终端设备接入系统。为了保证高效的服务质量，本研究采用云-边协同计算架构，在多个地理位置部署边缘服务器，并结合计算资源丰富的云服务器提供支持。然而，在这一架构下，如何确保不同用户在资源受限的环境下获得公平的服务体验成为了核心挑战。

用户公平性问题具体体现在：其一，由于地理位置不同，用户的网络条件存在较大差异。靠近边缘服务器的用户传输延迟低，而远离的用户则可能面临较长传输延迟，致使服务体验呈现显著的不均衡。其二，在实际应用里，常有不同类型用户并存的情况。例如，请求同一服务的用户中，既有普通用户，也有付费用户。服务系统依据用户付费水平将其划分为不同级别，本研究中将其定义为用户优先级。用户优先级也是影响公平性的关键因素，像常见的视频服务中，高优先级用户（如 VIP 用户或付费用户）一般应享有更多计算资源，以获取更短响应时间，而普通用户的请求在资源紧张时可能会被延迟甚至受限。在资源受限环境中，保障高优先级用户的权益，同样是公平性问题的重要方面。

本研究旨在边缘服务器资源有限的前提下，通过优化资源分配和服务实例的部署，让不同优先级的用户都能够在合理的响应时间内获得服务，最大程度提升系统的公平性。同时，还需要确保云-边协同架构在部署实例时能够平衡成本和资源消耗，避免不必要的资源浪费或过度负载，从而实现系统的高效运行和经济可行性。

为达成上述目标，本研究工作从以下几个方面展开：云-边协同微服务系统的设计与建模、问题求解、实验相关平台开发及实验。接下来，将详细介绍已完成的研究工作及所取得的成果。

2.1 云-边协同微服务系统的设计与建模

2.1.1 场景描述

本研究的云-边协同微服务系统由一组服务实例组成，这些服务实例部署在不同的边缘服务器以及云服务器上，为用户提供计算服务。用户分布在不同的地理位置，并根据付费水平等因素被赋予不同的优先级，从而享有相应的计算资源。每个用户的请求数据需在满足计算资源约束的前提下，由合适的服务器进行处理。系统通过优化连接关系，合理分配云-边资源，以确保不同用户的服务公平性，同时控制部署成本，防止边缘节点超载或云端延迟过高，从而实现高效的云-边协同计算。

2.1.2 系统建模

在云-边协同的计算场景下，为有效解决上述资源受限环境下公平性挑战等问题，本研究构建了一套全面的系统数学模型，该模精确定义了系统中用户、服务器等实体，并详细阐述了服务资源分配、传输延迟等关键环节的联系。具体建模内容如下：

以视频流场景为例，系统中共有 n 个用户，定义用户集为 $U = \{u_1, u_2, \cdots, u_n\}$，每个用户 u_i 被表示为一个多维变量：

$$u_i = < loc_i,\ Q_i,\ W_i,\ D_i > \tag{2-1}$$

其中，$loc_i = (x_i, y_i)$ 表示用户所处的地理位置；$Q_i = \{1, 2, \cdots\}$ 是系统根据用户的重要程度（如付费用户、普通用户等）来为用户赋予的变量，代表用户的优先级。Q_i 值越大，则该用户的重要程度越高。W_i 为用户权重，由系统依据 Q_i 来分配，并且通常与 Q_i 成正比关系，即用户优先级越高，权重越大 。W_i 在模型中有两个关键作用：其一，影响服务器对用户的资源分配过程，权重高的用户（即高优先级用户）会被服务器优先分配到更丰富资源；其二，在计算公平性指标时，W_i 会确保高优先级用户在公平性计算中占据更大的比重，以此来体现对重要用户的侧重考量，因此 W_i 的取值通常大于 1，具体的计应用方式将在优化目标部分详细阐述；D_i 表示用户 u_i 发往服务器（云或边缘）的数据量（比特），该值取决于用户实际的计算需求，直接影响服务器计算资源分配、传输延迟及整体响应时间。

系统中的服务器包括边缘服务器和云服务器，用集合 S 表示所有服务器：$S = S_{edge} \cup S_{cloud}$。边缘服务器描述为：

$$s_j^{edge} = < loc_j, R_j^{server}, P_j,\ c_{fixed_j}^{e} > \tag{2-2}$$

其中，$loc_j = (x_j, y_j)$ 表示边缘服务器的地理位置，其影响用户连接的选择；$R_j^{server} = \{r_j^{cpu\_total}, r_j^{mem\_total}, r_j^{b\_total}\}$ 表示边缘服务器的 CPU、内存以及带宽资源总量；P_j 代表边缘服务器的总计算能力；$c_{fixed_j}^{e}$ 为租赁边缘服务器的固定费用。云服务器描述为：

$$s_j^{cloud} = < loc_j, R_j^{server}, P_j,\ p_{net}^{c} > \tag{2-3}$$

式中 loc_j、R_j^{server}、P_j 与边缘服务器定义相同，p_{net}^{c} 为云服务器单位流量（bit）成本。

系统中的用户需要被分配到某个服务器来获取服务。用户与服务器之间的连接关系用二进制变量 x_{ij} 来表示：

$$x_{ij} = \begin{cases} 1，\text{用户 } u_i \text{ 连接到服务器 } s_j \\ 0，\text{用户与服务器没有连接} \end{cases} \tag{2-4}$$

在本实验场景中，所有用户均请求相同的服务实例，这些服务实例部署在服务器上。

每个服务器具有部署一个或多个此类服务实例的能力；若某个服务器未连接任何用户，那么该服务器也可以选择不部署服务实例。

由于用户的优先级之间存在差异，用户与服务器之间的资源分配进行了如下处理。当用户 u_i 连接到服务器 s_j 时，s_j 分配给该用户的资源集合为：

$$R_i^{user} = \{r_i^{cpu},\ r_i^{mem},\ r_i^b\} \tag{2-5}$$

其中，$r_i^{cpu} = f_{cpu}(D_i) \cdot W_i$，$r_i^{mem} = f_{mem}(D_i) \cdot W_i$，$r_i^b = f_b(D_i) \cdot W_i$，分别表示分配给用户的 CPU、内存以及带宽资源量；$f_x(D_i)$ 为用户数据请求量 D_i 与 CPU、内存以及带宽等资源分配量的转换函数，其根据实际转换情况进行设置；通过用 W_i 进行加权，使得高优先级用户能获取更多资源，体现了模型对不同优先级用户资源分配的差异化处理。

服务器分配给用户的处理能力与资源分配密切相关，定义服务器 s_j 分配给用户 u_i 的处理能力为：

$$P_{ij} = \left(\frac{R_i^{user}}{\sum_{u_k \epsilon U} R_k^{user}}\right) \cdot P_j,\ x_{kj} = 1 \tag{2-6}$$

其中，$\frac{R_i^{user}}{\sum_{u_k \epsilon U} R_k^{user}}$ 表示用户 u_i 在服务器 s_j 上所获得的资源量占连接到该服务器的所有用户分配的资源总和的比例，通过分配比例，每个用户获得的计算能力与其分配到的资源量成正比。

用户 u_i 从服务器 s_j 获得服务的响应时间用 t_{ij} 来表示。其由发送时延 $t_{send_{ij}}$、传播时延时 $t_{trans_{ij}}$ 以及处理时延 $t_{proc_{ij}}$ 三部分组成。即：

$$t_{ij} = t_{send_{ij}} + t_{trans_{ij}} + t_{proc_{ij}} = \frac{d_{ij}}{v} + \frac{D_i}{b_{ij}} + \frac{D_i}{P_{ij}},\ s_j \in S \tag{2-7}$$

其中，d_{ij} 为用户 u_i 与服务器 s_j 之间的通信链路长度；v 为链路中信号的传播速度；b_{ij} 为用户 u_i 的终端设备与服务器 s_j 之间的带宽，即（2-5）中的 r_i^b；D_i 为用户请求的数据量；P_{ij} 为服务器的数据处理能力。

系统服务部署的总成本表示为各个边缘服务器及云服务器的部署成本之和。其中，边缘服务器的部署成本由其租赁的固定成本 $c_{fixed_j}^e$ 与资源消耗成本 $c_{usage_j}^e$ 两部分组成，即

$$c_j^e = c_{fixed_j}^e + \sum_{i=1}^n x_{ij} \cdot (r_i^{cpu} \cdot p_{cpu}^e + r_i^{mem} \cdot p_{mem}^e + r_i^b \cdot p_b^e) \tag{2-8}$$

p_{cpu}^e、p_{mem}^e 和 p_b^e 为边缘节点的资源单价。

而云服务器的部署成本则由其网络流量成本 $c_{net_j}^c$ 与资源消耗成本 $c_{usage_j}^c$ 组成，即：

$$c_j^c = \sum_{i=1}^n x_{ij} \cdot D_i \cdot p_{net}^c + \sum_{i=1}^n x_{ij} \cdot (r_i^{cpu} \cdot p_{cpu}^c + r_i^{mem} \cdot p_{mem}^c + r_i^b \cdot p_b^c) \tag{2-9}$$

其中，网络流量成本 $c_{net_j}^c$ 是根据用户从不同区域访问云节点所产生的额外网络传输费用来计算的；资源消耗成本的计算方式与边缘服务器相同。

系统的总部署成本可表示为：

$$C_{total} = \sum_{s_j \in S_{edge}} c_j^e + \sum_{s_j \in S_{cloud}} c_j^c \tag{2-10}$$

2.1.3 约束条件

为确保优化结果的可行性，还必须考虑资源消耗、部署成本即最大平均响应时间等多

方面的约束条件。具体约束包含以下四个方面：

（1）**平均响应时间约束**　为确保各个优先级类别用户的平均响应时间不会超过服务质量要求，为每个优先级设定了响应时间上限，如公式（2-11）所示：

$$\frac{1}{|U_{Q_i}|} \leq T_{Q_i}^{max}, \ \forall Q_i \tag{2-11}$$

其中，$T_{Q_i}^{max}$ 表示优先级为 Q_i 的用户的平均响应时间上限。

（2）**边缘节点资源限制**　确保每个边缘节点 s_j 的资源消耗不得超过其最大可用资源，约束公式如下：

$$\sum_{i=1}^{n} x_{ij} \cdot r_i^{cpu} \leq R_j^{cpu\_total}, \forall s_j \in S_{edge} \tag{2-12}$$

$$\sum_{i=1}^{n} x_{ij} \cdot r_i^{mem} \leq R_j^{mem\_total}, \forall s_j \in S_{edge} \tag{2-13}$$

$$\sum_{i=1}^{n} x_{ij} \cdot r_i^{b} \leq R_j^{b\_total}, \forall s_j \in S_{edge} \tag{2-14}$$

（3）**部署成本限制**　确保边缘节点与云节点的部署成本总和不得超过整体预算。约束公式如下：

$$C_{total} = \sum_{s_j \in S_{edge}} c_j^e + \sum_{s_j \in S_{cloud}} c_j^c \leq C_{max} \tag{2-15}$$

其中，C_{max} 为设定的系统的部署成本上限。

（4）**用户与服务器连接限制**　每个用户 u_i 必须连接到唯一一个服务器，保证请求能够正确地路由到合适的服务实例，约束公式如下：

$$\sum_{j=1}^{m} x_{ij} = 1, \forall u_i \in U \tag{2-16}$$

2.1.4 优化目标

本研究优化问题的目标就是在满足各项约束条件的基础上，最大化系统的公平性。以用户响应时间的公平性作为系统公平性的衡量指标，采用用户响应时间的加权 Jain 指数进行公平性评价 。公平性目标函数的具体定义如下：

$$f = max \ F_{Jain} = max \ \frac{\left(\sum_{i=1}^{n} t_{ij}^{weight}\right)^2}{n \cdot \sum_{i=1}^{n} \left(t_{ij}^{weight}\right)^2} \tag{2-17}$$

公式中，分子为所有用户加权响应时间总和的平方，反映了加权时间的集中程度，若各用户的加权响应时间较为相近，总和会较大，平方后的值也更大，这表明用户间加权响应时间差异较小；分母是所有用户加权响应时间的平方和，体现了系统中所有用户响应时间的散布程度，若各用户加权响应时间相差较大，这个和将会较大，意味着公平性较差。加权响应时间能起到调节公平性的作用，若高权重用户的响应时间更短，其加权响应时间会更接近低权重用户的加权响应时间，使分子总和更集中，差异减小；同时会减少分母中高权重用户平方项的贡献，让整体分母变小。

整个目标函数通过最大化 F_{Jain} 来优化公平性，当 F_{Jain} 趋近于 1 时，代表所有用户的加权响应时间几乎完全相同，系统达到公平状态；当 F_{Jain} 趋近于 0 时，说明加权响应时间差异非常大，系统不公平。该目标函数不仅可用于公平性评估，还能通过引入加权响应时间，根据用户优先级调节不同用户之间的响应时间差异，保证不同优先级用户的响应时间公平。

故优化问题的完整形式可以形式化表达为：

$$
\begin{aligned}
& objective: \quad max\ F_{Jain} \\
s.t. \quad & \frac{1}{|U_{Q_i}|} \leq T_{Q_i}^{max},\ \forall Q_i \\
& \sum_{i=1}^{n} x_{ij} \cdot r_i^{cpu} \leq R_j^{cpu\_total}, \forall s_j \in S_{edge} \\
& \sum_{i=1}^{n} x_{ij} \cdot r_i^{mem} \leq R_j^{mem\_total}, \forall s_j \in S_{edge} \\
& \sum_{i=1}^{n} x_{ij} \cdot r_i^{b} \leq R_j^{b\_total}, \forall s_j \in S_{edge} \\
& \sum_{s_j \in S_{edge}} c_j^e + \sum_{s_j \in S_{cloud}} c_j^c \leq C_{max} \\
& \sum_{j=1}^{m} x_{ij} = 1, \forall u_i \in U
\end{aligned}
$$

2.2 问题求解

2.2.1 问题分析

由上述建模描述可知，本研究的优化问题涉及云-边协同计算环境下的资源分配与公平性优化。由于用户优先级、服务器资源限制以及服务部署成本等因素，问题复杂度较高，属于约束优化问题（Constrained Optimization Problem），并且因用户-服务器匹配的多种可能性，整体计算复杂度为 NP-hard。精确优化方法（如整数线性规划 MILP）虽然能得到最优解，但计算成本高，求解时间随问题规模的增长呈指数级上升，难以适用于大规模场景。相比之下，基于贪心[1]的启发式方法计算复杂度较低，能够在多项式时间内找到较优解，同时具备较强的适应性和可扩展性，能有效应对动态变化的资源环境。通过优先级排序和逐步优化，该方法能够快速调整资源分配方案，以确保公平性并提升计算效率。

2.2.2 算法实现思路

基于前述分析，本研究采用基于贪心的优化方法来优化云-边协同计算中的公平性服务部署问题。该方法的核心思想是按照用户的优先级，逐步为每个用户选择最优的服务器资源分配方案，以最大化系统的公平性指数。在具体实现过程中，算法首先对所有用户按照优先级进行排序，并依次为每个用户分配计算资源，每次选择能够最大化公平性指数的服务器作为当前的局部最优解。与此同时，算法动态调整服务器资源占用情况，确保系统整体资源不超载，并满足各类约束。

2.2.3 算法具体流程

（1）**初始化**　初始化用户到服务器的分配矩阵为零矩阵，并设置每个服务器的资源使用情况。

（2）**用户优先级排序**　根据用户的优先级（例如，高付费用户优先）将所有用户进行排序。优先级较高的用户将在分配资源时得到优先考虑。

（3）**分配服务器资源**　根据每个用户的需求，从可用的服务器中选择一个最适合的服务器进行资源分配。该选择过程旨在最大化加权 Jain 公平性指数，同时满足服务器的资源限制。

（4）**更新资源使用情况**　每分配一次资源，更新相应服务器的资源使用情况，确保不会超过服务器的最大资源限制。

（5）**检查约束条件**　在每一步分配之后，检查当前资源分配是否满足所有约束条件，包括服务器负载、部署成本以及响应时间上限等。如果所有条件满足，则继续分配下一用户的资源。

（6）**退出条件**　如果成功为所有用户分配服务器并满足所有约束，返回最终的分配矩阵；若在多次尝试后仍未找到满足约束条件的结果，则退出并给出警告。

基于贪心的优化方法的伪代码如下所示：

Greedy User-Server Allocation

Input: U ← user set, S ← server set, resourceLimits, maxAttempts
Output: X ← connection matrix

```
X ← initializeZeroMatrix(U, S)
valid ← false
attempt ← 0
while not valid and attempt < maxAttempts do
    X ← resetMatrixAndResources(U, S)
    for each user u in U do
        bestServer ← -1
        for each server s in S do
            tempAssign(u, s)
            if checkResourceConstraints(s, u, resourceLimits) then
                jainIndex ← calculateJainIndex(X, S)
                if jainIndex > bestJain then bestJain ← jainIndex, bestServer ← s
            end if
            undoTempAssign(u, s)
        end for
        if bestServer ≠ -1 then assignUserToServer(X, u, bestServer)
    end for
    if checkAllConstraints(X, S, resourceLimits) then valid ← true
    attempt ← attempt + 1
```

```
20: end while
21: if attempt ≥ maxAttempts then
22：    Warning
23：    return None
24: return X
```

2.2.4 算法复杂度分析

基于贪心的优化的时间复杂度主要由用户遍历和服务器遍历两个嵌套循环决定。假设用户大小为 U，服务器集合大小为 S，那么对于每个用户，需要遍历所有服务器以寻找最优资源分配方案，因此单次资源分配的复杂度为 O(S)。此外，Jain 指数的计算涉及所有用户和服务器，最坏情况下需要 O(U + S) 的计算时间。因此，单轮分配的复杂度可表示为 O(U * S * (U + S))。同时，外层 while 循环最多执行 maxAttempts 次，设其上限为 A，最终算法的整体复杂度为 O(A * U * S * (U + S))。在实际应用中，maxAttempts 通常是一个较小的常数，可以视为 O(1)，因此最终可简化为 O(U * S * (U + S))。这一复杂度表明，算法能够在多项式时间内找到较优解，计算成本较低，能够适用于大规模云-边协同计算的场景。尽管贪心方法无法保证绝对最优解，但在计算效率与公平性优化之间取得了良好的平衡。

2.2.5 对比算法

为了验证基于贪心的优化方法的有效性，本研究将其与几种其他优化算法进行了对比：

（1）**优化求解器（Gurobi）** Gurobi [2] 使用数学优化工具对资源分配进行全局优化，目标是最大化系统的公平性并满足约束条件。优化求解器能够找到理论上的最优解，但计算开销较大，尤其在大规模问题中。

（2）**遗传算法（GA）** 遗传算法[3,4]是一种基于自然选择和遗传学原理的启发式优化算法。通过模拟自然选择过程，GA 能够在大规模问题中找到近似最优解，但相比于贪心算法，计算时间较长。

（3）**无公平性算法** 此算法以最小化平均响应时间为优化目标，不考虑用户之间的公平性。它可以在某些情况下降低用户响应时间，但由于忽视了优先级和资源分配的差异，将无法确保系统公平性，在本实验中主要是用来对照。

2.3 实验平台开发

在本云-边协同服务部署场景中，各服务器的资源是根据分配情况实时发生变化的。因此本研究依托 MicroForge 平台进行实验，重点开发并优化了数据源管理模块，以支持云-边协同计算环境下的资源监控和优化算法的实验测试。在实验过程中，可以通过该模块实时获取各项资源参数，保证实验数据的完整性和可用性。

2.3.1 功能描述

数据源管理模块是 MicroForge 平台的重要组成部分，负责管理系统中的所有数据源，并提供对资源使用情况的实时监测。该模块的核心功能包括：

（1）**数据源管理** 提供统一的界面，展示所有数据源的基本信息，包括数据类型、

连接状态及资源使用情况。

（2）**资源监控**　实时监测 CPU、内存、带宽等资源的使用情况，并提供可视化展示，以便研究人员直观分析系统负载。

（3）**实验数据采集与分析**　记录实验过程中各类关键指标，如服务器负载、用户响应时间等。

（4）**可视化展示**　提供直观的界面，动态展示资源使用情况，便于研究人员进行实验分析。

2.3.2 需求分析

为了确保数据源管理模块能够满足实验需求，其设计需满足以下功能性要求：

（1）**实时数据更新**　通过 WebSocket 监听资源状态变化，确保系统能够动态响应负载变化，提供精准的资源监控。

（2）**可扩展性**　支持不同规模的用户和服务器环境，适用于大规模云-边协同计算实验。

（3）**灵活的数据管理**　允许用户增删改查数据源，并调整实验参数，以适应不同实验场景。

（4）**可视化分析**　直观展示实验结果，便于研究人员分析系统的资源分配情况，并导出数据用于进一步研究。

2.3.3 设计与实现

数据源管理模块采用模块化架构，包括以下核心部分：

（1）**前端交互层**　基于 React 开发，提供数据展示与交互操作，包括资源监控和实验数据可视化功能。

（2）**后端计算层**　采用 SpringBoot 进行数据管理，支持资源状态的实时更新。

（3）**数据采集与监控层**　结合 WebSocket 技术，实现实验数据的实时更新，支持多用户访问。

（4）**网络环境模拟模块**　利用流量控制工具（如 Linux TC 命令），模拟带宽波动、延迟和丢包情况。

实验环境采用 Docker + Kubernetes 进行容器化部署，以保证平台的可扩展性和稳定性。

2.3.4 模块功能流程

图 2-1 展示了数据源管理模块的运行流程。

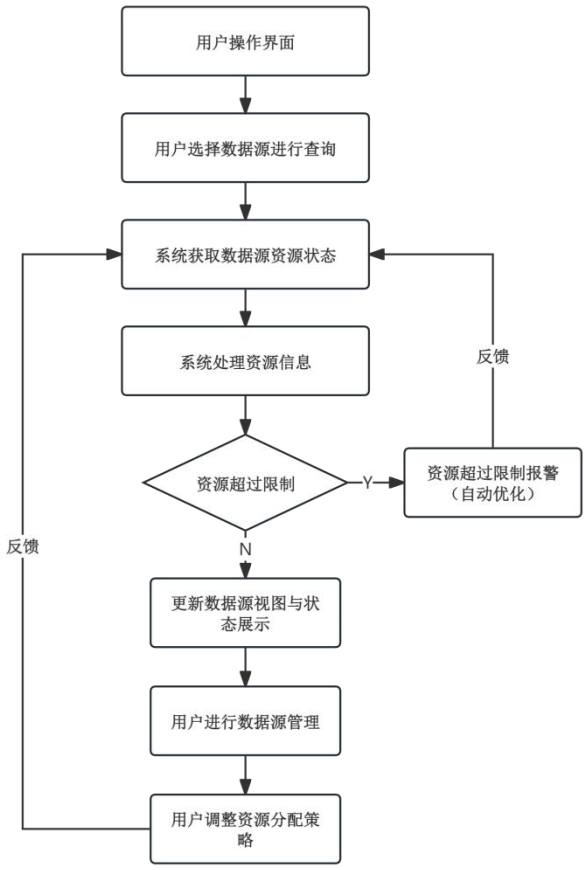


图 2-1 数据源管理模块功能流程

2.3.5 模块展示

图 2-2 (a)、2-2 (b)、2-2 (c) 分别展示了平台上数据源管理模块的不同功能页面。

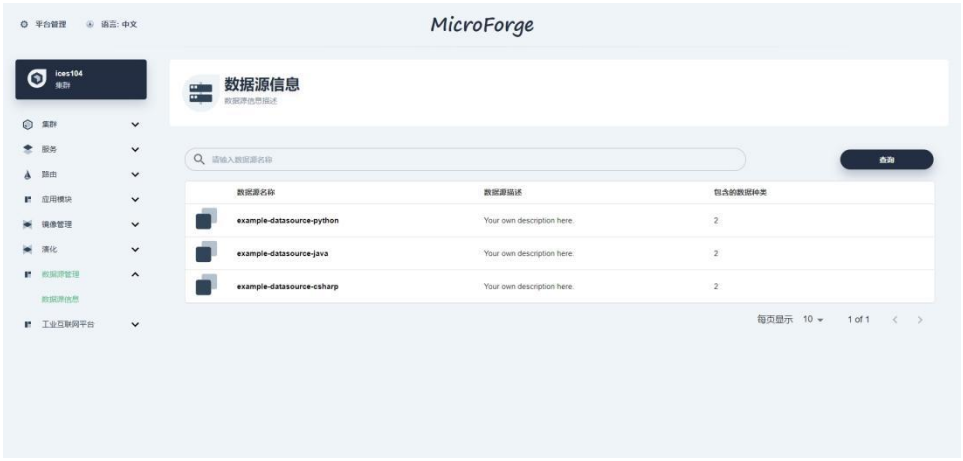


图 2-2 (a) 数据源信息展示页面

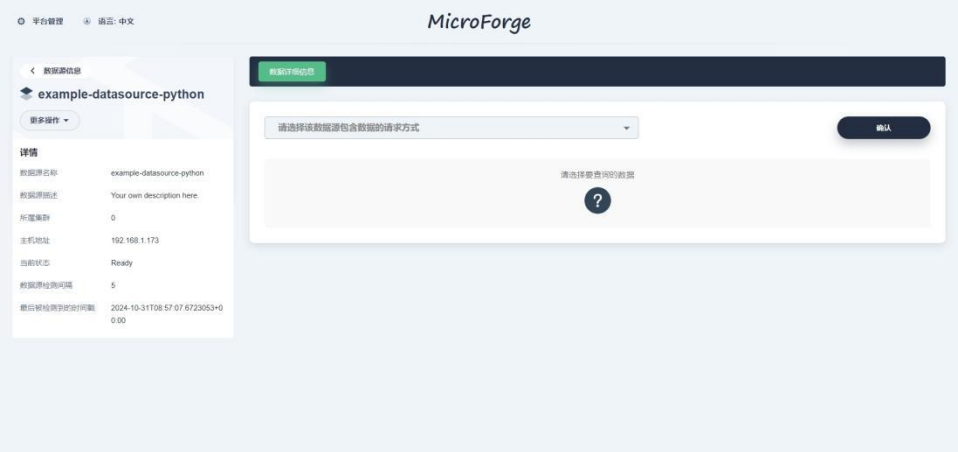


图 2-2 (b) 资源信息获取页面

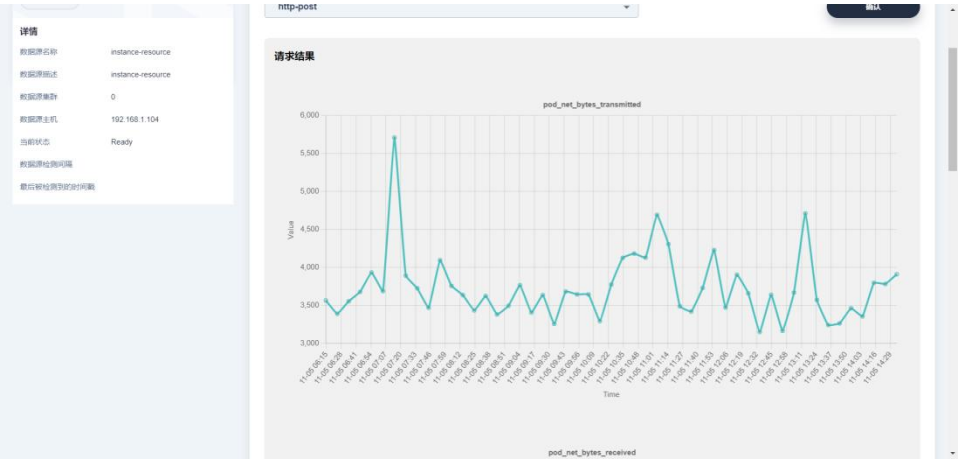


图 2-2 (c) 资源信息展示页面

2.3.6 平台功能模块与本研究的结合

数据源管理模块在 MicroForge 平台中的核心作用是提供详细的资源监控信息，为本研究的公平性优化和资源分配策略提供强有力的数据支撑。研究人员可以通过该模块实时获取服务器资源状态，优化资源分配策略，并确保高优先级用户能够获得更精确的资源分配。通过该模块，MicroForge 平台为本研究提供了一个灵活且强大的实验环境，使得不同网络条件和资源使用情况下的算法优化与测试变得更加高效和直观。

2.4 实验

2.4.1 实验设计

为了验证所提出的云-边协同系统在不同网络环境下的性能表现以及资源分配算法的有效性，本研究设计了一系列实验。实验的主要目的是评估不同算法在处理不同用户数量和优先级时的平均响应时间和公平性表现。

实验设置如下：

（1）**用户数量** 选择 100、150 和 200 个用户，来模拟不同负载下的用户场景。

（2）**用户优先级设置** 为每个用户分配一个优先级，本实验中包括优先级 1、2、3 三个类别，以测试在优先级差异下的公平性。

（3）**用户权重设置**　为不同优先级的用户设置权重，权重值分别为 1、2、3，与优先级对应，以评估资源分配算法在不同用户权重下的公平性表现（实际应用环境中，系统会定义一定评估指标来确定权重值；在本研究实验环境下先进行简单自定义）。

（4）**算法选择**　实验中使用四种算法进行对比，包括基于贪心的优化方法、Gurobi 优化求解器、遗传算法（GA）以及无公平性的算法。

（5）**实验平台**　本实验应用到了 MicroForge 平台，平台的数据源管理模块为实验提供了必要的实验环境。

2.4.2 实验过程

实验过程分为四个阶段，涵盖系统初始化、算法执行、数据采集和实验验证。

（1）**初始化阶段**　为每个实验配置用户数量、优先级以及相应的服务请求。并初始化每个算法的输入参数。

（2）**算法执行**　根据所选算法，执行资源调度与服务实例部署过程。每个算法根据用户的优先级、请求大小和网络延迟等条件进行资源分配。

（3）**数据采集与分析**　收集每个用户的响应时间、系统资源利用率等数据，并进行统计分析。重点关注不同优先级用户的平均响应时间差异，以及不同算法在公平性优化方面的表现。

（4）**实验验证**　通过实验验证模型和算法的正确性，并对不同算法的性能进行对比分析，特别是在平均响应时间和公平性方面的差异。

2.4.3 实验结果与分析

2.4.3.1 实验结果展示

在本部分中，我将以 200 个用户为例，展示各算法的运行结果。

（1）**用户与服务器初始分布**　图 2-3 展示了用户与边缘服务器和云服务器的初始分布情况。

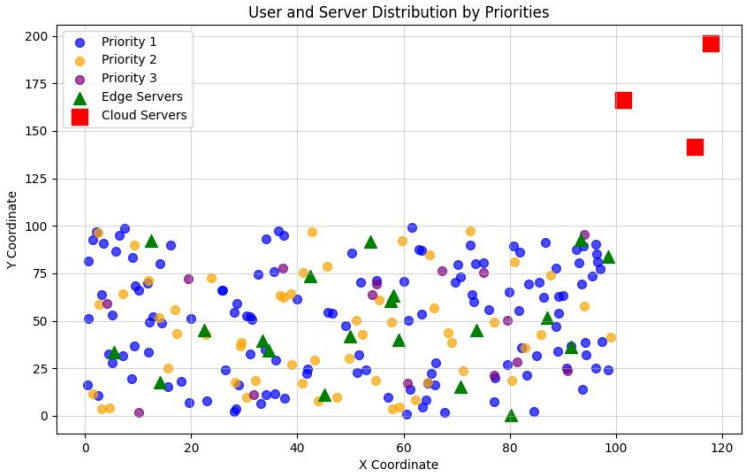
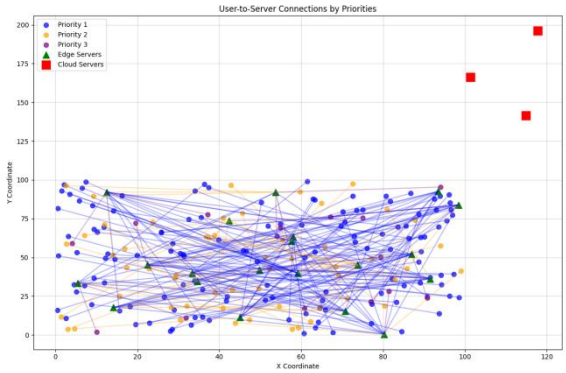
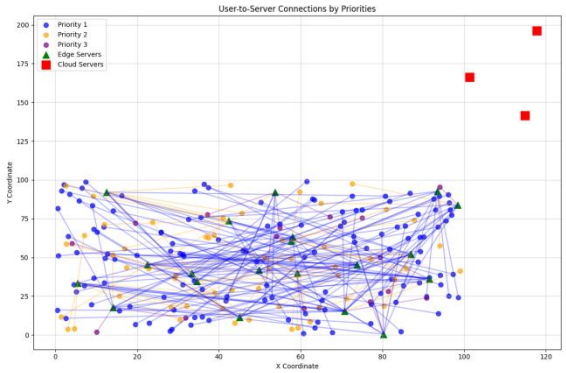


图 2-3 用户与服务器的初始分布

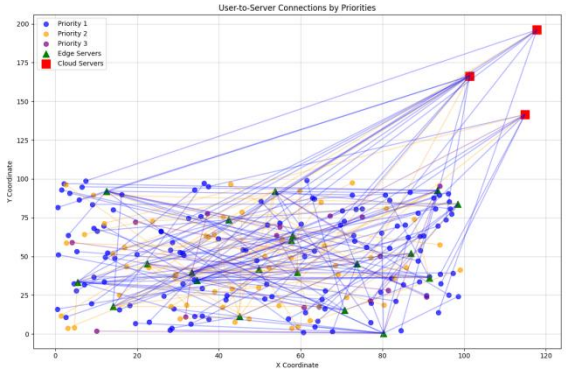
（2）**用户与服务器连接情况**　图 2-4 (a)、2-4 (b)、2-4 (c) 、2-4 (d) 分别展示了基于贪心的优化方法、Gurobi 优化求解器、遗传算法（GA）以及无公平性的算法运行后的用户与服务器连接情况。



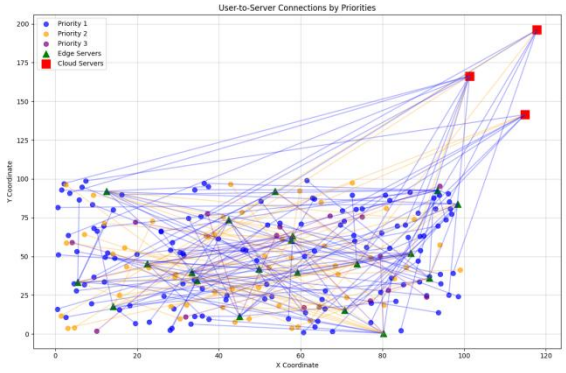
(a) 基于贪心的优化方法



(b) Gurobi 优化求解器



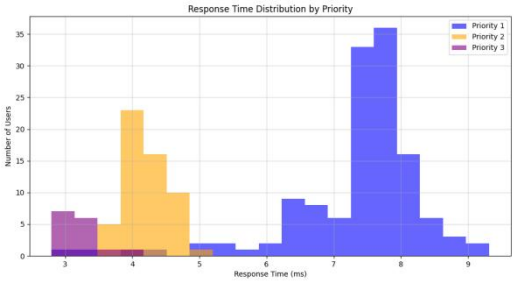
(c) GA 算法



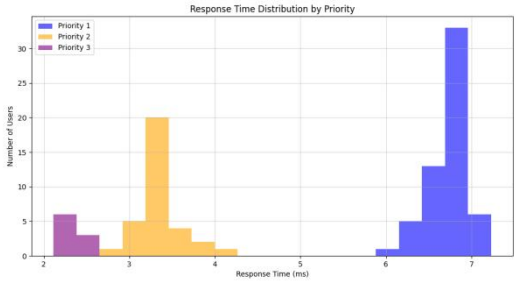
(d) 无公平性算法

图 2-4 用户与服务器连接情况

（3）**各优先级用户的响应时间分布**　图 2-5 (a)、2-5 (b)、2-5 (c) 、2-5 (d) 分别展示了基于贪心的优化方法、Gurobi 优化求解器、遗传算法（GA）以及无公平性的算法运行后的各个优先级用户的响应时间分布情况。

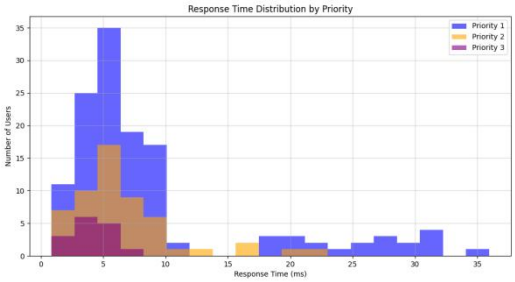


(a) 基于贪心的优化方法

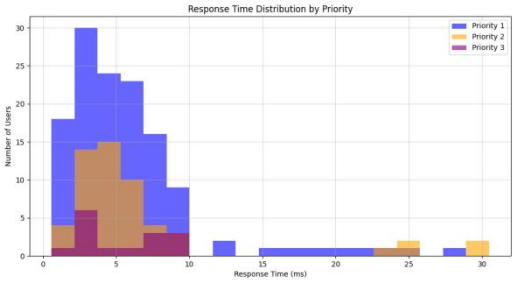


(b) Gurobi 优化求解器

图 2-5 各优先级用户的响应时间分布情况



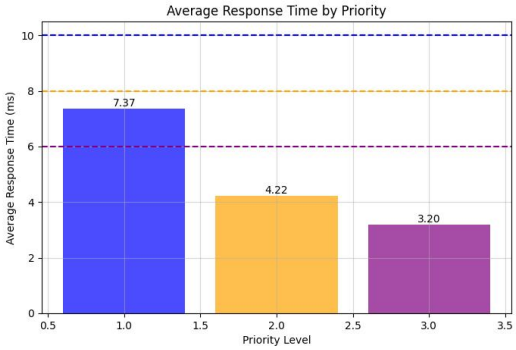
(c) GA 算法



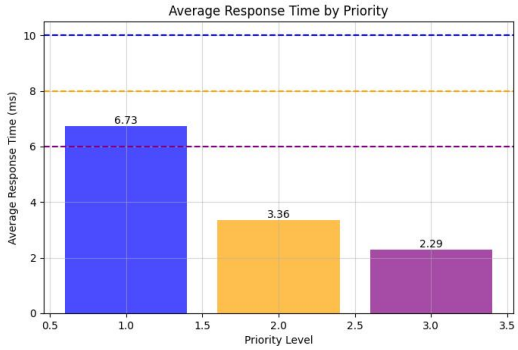
(d) 无公平性算法

图 2-5 各优先级用户的响应时间分布情况（续）

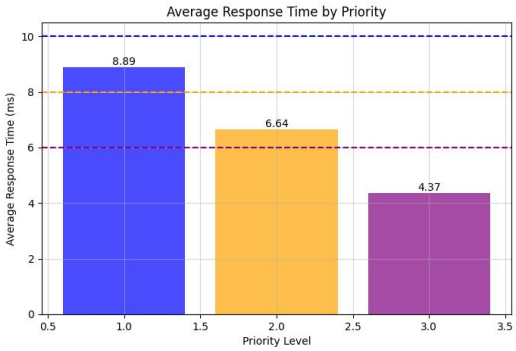
（4）**各个优先级用户的平均响应时间** 图 2-6 (a)、2-6 (b)、2-6 (c) 、2-6(d) 分别展示了基于贪心的优化方法、Gurobi 优化求解器、遗传算法（GA）以及无公平性的算法运行后的各个优先级用户的平均响应时间。



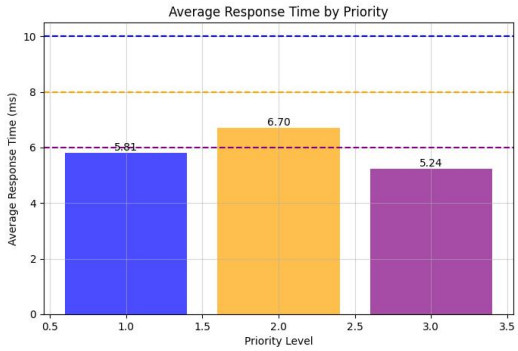
(a) 基于贪心的优化方法



(b) Gurobi 优化求解器



(c) GA 算法



(d) 无公平性算法

图 2-6 各优先级用户的平均响应时间

2.4.3.2 实验结果分析

在本部分中，我将展示不同算法在不同用户数量和优先级设置下的性能表现，具体从不同优先级用户平均响应时间和系统公平性表现等方面进行分析。

（1）**不同用户数量下各算法的平均响应时间** 图 2-7 (a)、2-7 (b)、2-7 (c) 分别展示了用户数量为 100、150 和 200 时，各算法不同优先级用户的平均响应时间。

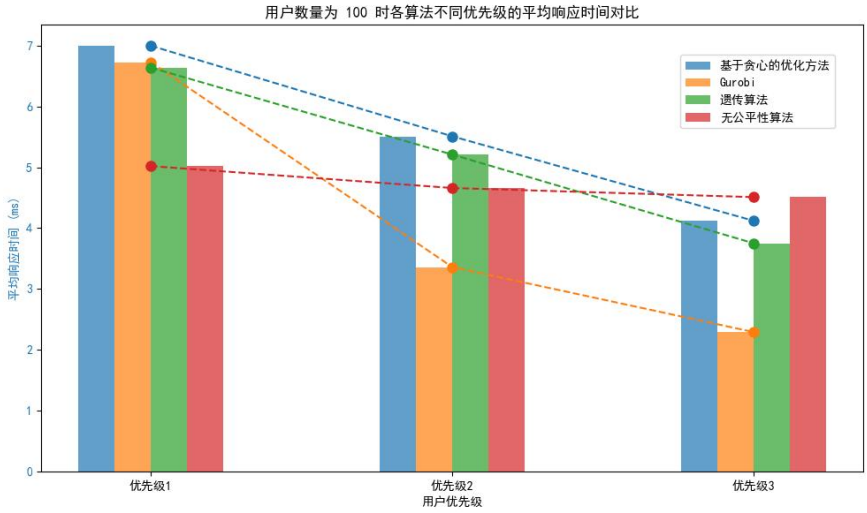


图 2-7 (a) 用户数量为 100 时各算法平均响应时间对比

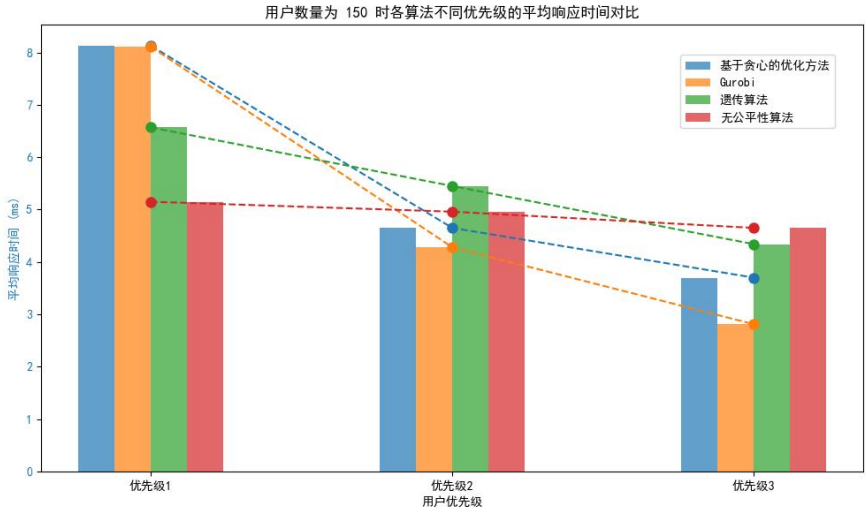


图 2-7 (b) 用户数量为 150 时各算法平均响应时间对比

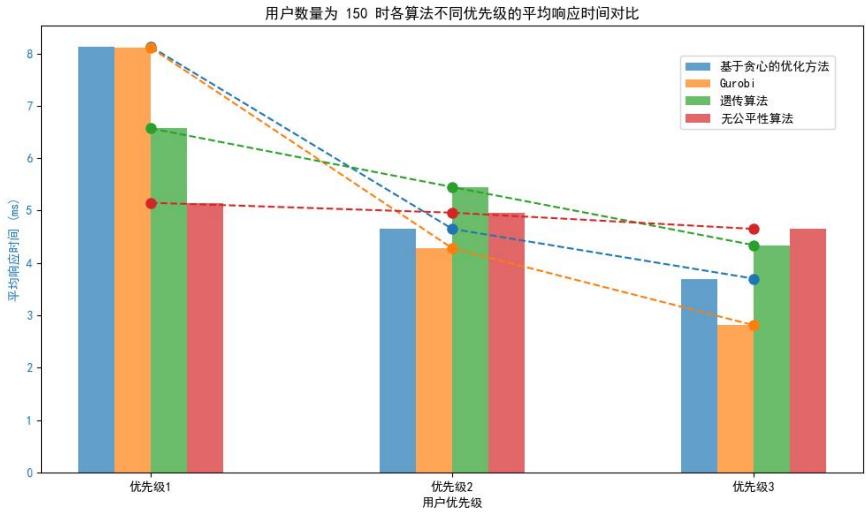


图 2-7 (c) 用户数量为 200 时各算法平均响应时间对比

我从以下两个方面对实验结果进行分析：

首先，是在同一算法下不同优先级用户的平均响应时间情况。理论上，在资源有限的情况下优先级越高的用户应优先获得资源，因此他们的响应时间应该更短。实验结果表明，基于贪心的优化方法和 Gurobi 优化算法能够有效根据优先级分配资源，确保高优先级用户的响应时间明显较短，而低优先级用户的响应时间则较长。相比之下，无公平性算法没有考虑优先级差异，导致所有用户的响应时间趋于一致，无法有效区分高低优先级用户，影响高优先级用户的体验。

其次，四种算法的对比结果显示，基于贪心的优化方法在资源分配方面表现较为均衡，尤其在大规模用户场景下，能够稳定保持低响应时间。Gurobi 优化求解器虽然提供最优解，但计算复杂度较高。遗传算法在小规模场景表现良好，但在用户数量增多时响应时间波动较大，且计算时间较长。无公平性算法忽视了优先级差异，尽管减少了平均响应时间，但无法保证公平性。总的来说，基于贪心的优化方法在平衡公平性和响应时间方面表现最佳，尤其适合大规模用户场景。

（2）**不同用户数量下各算法的加权 Jain 指数** 图 2-8 展示了不同用户数量下，各算法的加权 Jain 指数，该指数用于衡量系统的公平性，值越大表示系统的公平性越好。

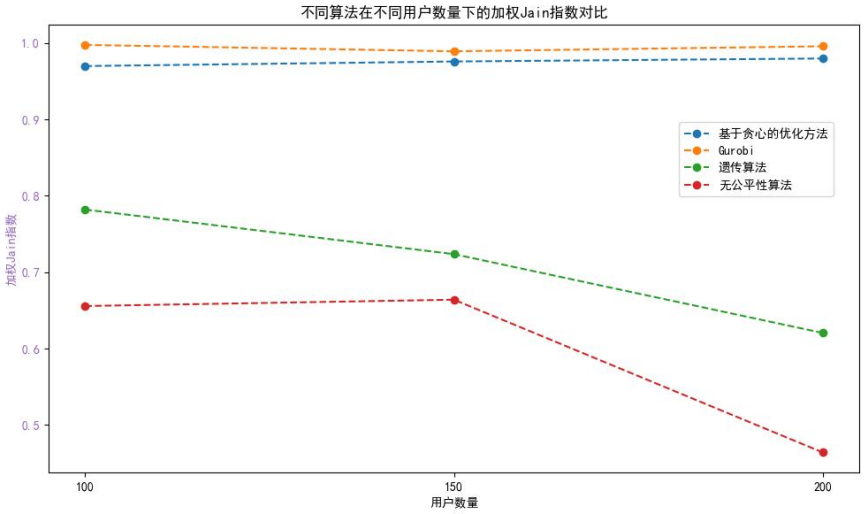


图 2-8 不同用户数量下各算法的加权 Jain 指数

从实验结果可以看出，随着用户数量的增加，所有算法的加权 Jain 指数均有所下降，表明系统的公平性在用户数量增多时有所降低。然而，基于贪心的优化方法和 Gurobi 优化求解器在所有用户数量下的加权 Jain 指数始终较高，显示出较好的公平性，尤其在大规模用户场景中，二者的公平性明显优于其他算法，说明它们能提供更优化的资源分配方案。尽管如此，由于基于贪心的优化方法的实现简单且计算开销较低，它在实际应用中表现更为优越，特别是在资源有限的情况下。相比之下，无公平性算法的加权 Jain 指数最低，表明该算法未能考虑用户优先级和公平性，导致不同用户之间的响应时间差异较大。总体来看，基于贪心的优化方法在平衡实现简单性和系统公平性方面表现突出，特别适合应用于大规模用户场景。

3 后期拟完成的研究工作及进度安排

后期工作将按以下进度安排完成：

（1）**模型调优与验证**（2025.03.10-2025.03.31） 进一步对现有模型进行验证，评估模型在不同环境下的表现，确保其具备实际应用的可行性。

（2）**算法优化**（2025.04.01-2025.04.30） 优化算法性能，提升算法的计算效率和稳定性，确保大规模问题处理的可行性。

（3）**论文撰写与总结**（2025.04.01-2025.04.30） 总结研究成果，并完成论文的编写与完善。

4 存在的问题与困难

（1）**实验数据的收集与分析困难** 实验需要在实际的云边协同环境中运行，数据收集面临一定的挑战。尤其是在多种网络环境和设备条件下进行实验时，数据的准确性和代表性可能会受到不同网络延迟、带宽波动以及设备性能差异的影响。此外，实验场景中的资源动态变化，使得数据采集必须具有时效性和一致性，进一步增加了数据整合的难度。尽管通过仿真实验模拟了一部分场景，但与实际环境相比，仍然存在差异，如何保证实验数据的真实性和代表性仍然是一个关键问题。

（2）**公平性评估的主观性** 在公平性优化过程中，不同用户的需求差异较大，如何权衡高优先级用户的资源分配与低优先级用户的服务保障仍然是一个挑战。此外，公平性评估指标的选取具有一定的主观性，例如不同权重的 Jain 指数计算方式可能会导致不同的优化结果，影响实验的公平性评估标准。进一步地，在面对动态变化的资源环境时，公平性标准可能需要动态调整，使得评估过程更为复杂。因此，如何合理地定义公平性衡量标准，并在不同应用场景下保持评估方法的通用性和可靠性，是一个需要进一步研究的问题。

5 论文按时完成的可能性

根据目前的进度安排，研究工作和论文编写的各个环节都在按计划推进。系统建模、算法设计以及部分实验已经完成，且模型调优和成本效益分析也已规划到位。在接下来的两个月内，重点将放在完成算法优化和模型验证上，以确保系统的实际应用可行性和稳定性。所有的实验和算法优化工作预计将在 3 月底完成，论文撰写阶段将从 4 月开始并顺利进入总结和完善阶段。

尽管存在一些实验数据收集和公平性评估的挑战，但通过已有的研究成果和数据支持，项目按时完成的可能性较高。在接下来的时间内，确保顺利完成剩余的工作，应该能够按计划完成毕业论文的撰写和提交。

6 主要参考文献

[1] Jungnickel D, Jungnickel D. The greedy algorithm[J]. Graphs, networks and algorithms,1999: 129-153.

[2] Pedroso J P. Optimization with gurobi and python[J]. INESC Porto and Universidade do Porto, Portugal, 2011, 1.

[3] Reeves C, Rowe J E. Genetic algorithms: principles and perspectives: a guide to GA

theory[M]. Springer Science & Business Media, 2002.
[4] 葛继科，邱玉辉，吴春明，等. 遗传算法研究综述[J]. 计算机应用研究, 2008, 25(010): 2911-2916.