学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

密级\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**武汉大学本科毕业论文**

**边缘环境下基于指针网络的**

**服务请求调度策略**

院（系）名 称：计算机学院

专 业 名 称 ：软件工程

学 生 姓 名 ：姜德纶

指 导 教 师 ：李兵 教授

二○二一年四月

**郑 重 声 明**

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名： 日期：

摘要

近年来，随着互联网的高速发展，云计算技术在各行各业都得到了广泛应用。但在某些场景下，传统的云计算方式也会暴露出它的缺点。比如当下快速发展的物联网(IoT)中,由于传统的云计算方式依靠任务卸载将数据传输到云端中进行处理并将结果返回给用户终端，导致许多对通讯时延敏感性较高的服务，无法得到及时的响应。而在边缘计算环境下，通过将服务部署到网络边缘，极大地缩短了任务时延，能有效解决时延敏感性问题。但边缘服务器本身的资源，相对于云端来说还是十分有限，如何充分利用网络边缘环境下的服务资源，尽可能多地满足服务请求，是边缘计算面临的主要问题之一。由此可见，为边缘服务器下的服务请求设计一套高效合理的调度策略，用有限的资源响应更多的请求，保持良好的用户体验与服务质量，是十分必要的。

在这篇论文中，我们模拟真实场景：针对边缘计算环境下的多服务器多用户请求场景。一个边缘服务器能接受多个服务请求，同时一个用户能向多个边缘服务器发送服务请求，在该场景下进行服务请求的并行调度。我们将所有边缘服务器的运行时间平均值，任务优先级和总超时率作为调度策略的优化目标，基于指针网络，使用强化学习的方式设计了一套训练模型。将传统服务请求特征中的优先级、运行所需时间、超时时间，与服务器资源中的CPU、I/O、宽带、内存占用率结合在一起，并且容纳了边缘计算环境中边缘服务器与用户之间的覆盖关系一特征，形成数据集进行实验，并将结果与贪心算法、随机算法以及操作系统中的多级反馈队列调度算法进行对比。实验结果显示，我们的基于指针网络和强化学习的服务请求调度策略，在平均运行时间，任务优先级，总超时率方面的表现均优于贪心算法、随机算法以及多级反馈队列调度算法。

**关键词：**边缘计算；指针网络；强化学习；调度算法

**ABSTRACT**

With the rapid development of the Internet in recent years, cloud computing technology has been widely used in all walks of life. But in some scenarios, conventional cloud computing will also expose its shortcomings. For example, with the rapidly developing Internet of Things (IoT), due to conventional cloud computing rely on task offloading to transfer data to the cloud for processing and return the results to the user terminal, many services sensitive to communication delays cannot get timely responses. But in the edge computing environment, by deploying services to the edge of the network, the delay of tasks is greatly reduced, which can effectively solve the problem of delay sensitivity. However, the resources of the edge server are still very limited compared to the cloud. How to make full use of service resources in the edge of network and satisfy as many service requests as possible is one of the main problems in edge computing. It can be seen that it is very necessary to design an efficient and reasonable scheduling strategy for tasks under the edge server, respond to more requests with limited resources, and maintain a good user experience.

In this paper, we simulated a real scenes, for the multi-server and multi-user requests scenarios in the edge computing environment. An edge server can accept multiple service requests, and a user can send service requests to multiple edge servers. parallel scheduling of service requests is performed inside this scene. We made the average running time of all edge servers, task priority and total timeout rate as the optimization goals of the scheduling strategy, then designed a training models based on pointer network using reinforcement learning. We combined the task priority, running time and timeout time in traditional task features with CPU, I/O, bandwidth, and memory occupancy in server resources，and accommodated the feature of the coverage relationship between edge servers and users in the edge computing environment to form a data set for experimentation, then we compare the results with greedy algorithm, random algorithms and Multilevel Feedback Queue Scheduling algorithm in operating system. The results of experiments showed that our rquest for service scheduling strategy based on pointer network and reinforcement learning was better than reedy algorithm, random algorithms and Multilevel Feedback Queue Scheduling algorithm in terms of all average running time, task priority, and total timeout rate.

**Key words:** Edge computing; Pointer network; Reinforcement learning; Scheduling algorithm

**目录**

[**1绪论**](#_Toc69476617)

[1.1 研究背景与意义 1](#_Toc69476618)

[1.2 国内外研究现状 2](#_Toc69476619)

[1.3 论文主要工作 4](#_Toc69476620)

[**2场景与问题描述**](#_Toc69476621)

[2.1 场景介绍 5](#_Toc69476622)

[2．1．1 场景示例 5](#_Toc69476623)

[2．1．2 数据格式 6](#_Toc69476624)

[2.2 问题描述 6](#_Toc69476625)

[**3基于指针网络的服务请求调度**](#_Toc69476626)

[3.1 问题建模 8](#_Toc69476627)

[3．1．1 数据定义 8](#_Toc69476628)

[3．1．2 输入输出 9](#_Toc69476629)

[3．1．3 优化目标 10](#_Toc69476630)

[3.2 指针网络 11](#_Toc69476631)

[3．2．1 指针网络结构 12](#_Toc69476632)

[3．2．2 指针网络的注意力机制 13](#_Toc69476633)

[3.3 强化学习 13](#_Toc69476634)

[3．3．1策略梯度 14](#_Toc69476635)

[3．3．2 行动者-评论家模型 14](#_Toc69476636)

[3.4 训练效果 15](#_Toc69476637)

[**4实验设计与结果分析**](#_Toc69476638)

[4.1 数据集生成 17](#_Toc69476639)

[4.2 对比算法 17](#_Toc69476640)

[4．2．1 随机算法 18](#_Toc69476641)

[4．2．2 贪心算法 18](#_Toc69476642)

[4．2．3 多级反馈队列调度算法 19](#_Toc69476643)

[4.3 实验结果 19](#_Toc69476644)

[4．3．1 实验参数设置 19](#_Toc69476645)

[4．3．2 实验结果分析 21](#_Toc69476646)

[**结论** 25](#_Toc69476647)

[**参考文献** 27](#_Toc69476648)

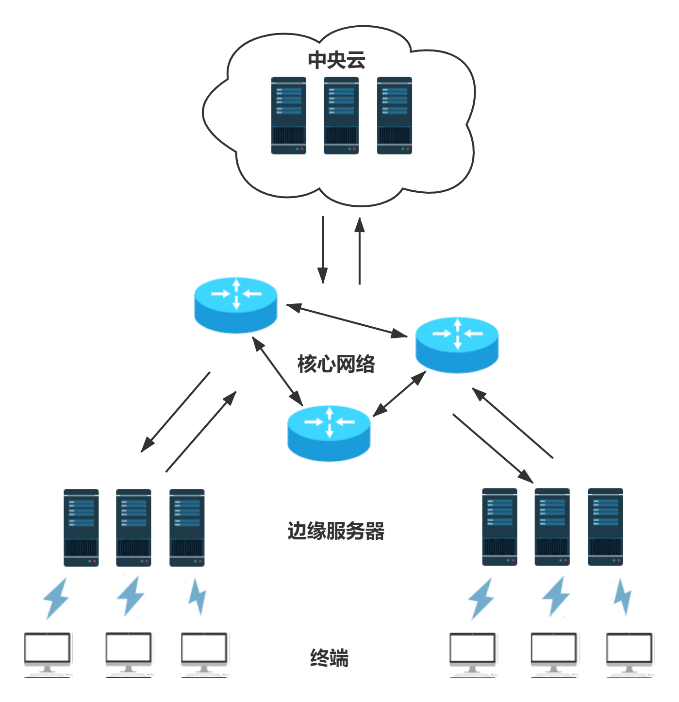
[**致谢** 31](#_Toc69476649)

**1绪论**

**1.1研究背景与意义**

近年来，随着互联网技术的高速发展，网络设备的数量急速升高，根据预测，到2025年，连接到互联网上的设备将超过500亿台[1]。不同设备所需要请求的服务不尽相同，但用户终端设备的计算能力与数据承载能力十分有限，当需要执行计算密集型任务的时候，仅仅依靠用户终端设备的计算能力，往往无法满足其需求。并且，目前互联网提供的许多服务，如人工智能领域下的人脸识别等，都需要庞大的数据量进行支撑。为了解决终端设备计算能力与数据承载能力有限的问题，云计算技术应运而生。云是一个包含大量可用虚拟资源(例如硬件、开发平台以及I/O服务)的资源池[2]，云计算通过将需要复杂计算以及大量数据进行支撑的服务部署在云端，从而达到解放终端设备本身的计算能力与数据存储空间的效果。用户仅需要通过网络请求云端的服务，就能得到相应结果。但在某些场景下, 传统的云计算方式也会暴露出它的缺点。在物联网(IoT)领域中,如智慧城市，智慧交通，在这些场景下的设备请求服务的时候，往往要求服务能够快速响应。然而由于传统的云计算依靠任务卸载方式将数据传输到云端中进行处理并将结果返回给用户终端，通信距离远，时延往往较高，导致许多对通讯时延敏感性较高的服务，无法得到及时的响应[3]。与此同时，传统的云计算方式还面临着庞大数据的存储问题。随着物联网的发展，我们将进入后云时代，在后云时代，沉浸在我们日常生活中的事物将产生大量高质量的数据[4]。而将应用部署到网络边缘环境中，从而消耗海量数据，承担计算任务，缩短传输时延，是解决上述问题的重要途径之一。这种技术称为边缘计算，边缘计算系统架构如图1.1所示。

根据Evans在2011年的预测显示，到2019年，45%的物联网数据将在网络边缘环境进行存储、分析和处理[5]。但边缘服务器本身的资源，相对于云端来说还是十分有限，如何充分利用网络边缘环境下的服务资源，尽可能多地满足服务请求，是边缘计算面临的主要问题之一。为了解决这个问题，国内外针对边缘环境下的服务部署，资源分配等方向做出了许多研究，以减少边缘计算成本，提高效率。而本文则将重点放在边缘环境下的服务请求调度问题。为边缘服务器下的服务请求设计一套高效合理的调度策略，从而可以用有限的资源响应更多的请求，提高用户体验的同时节约成本。同时，本文的研究成果可以为后续丰富边缘计算环境下的调度问题研究提供一定参考。



图**1.1** 边缘计算系统架构

**1.2国内外研究现状**

边缘环境下的服务请求调度问题本质上是将任务调度问题应用于边缘环境当中。当一个边缘服务器同一时段接收到多个用户请求的时候，这些请求会在服务器中进行排队，而对于调度问题的研究实际上是研究如何安排请求的执行顺序，最终使得服务质量提高。

目前，对于边缘计算环境下的其他主要问题，如用户分配，服务部署，负载均衡等问题，国内外已有许多研究。如文献[6]针对边缘环境下的用户分配问题，将真实环境中的用户分配问题抽象成可变尺寸的向量装箱问题(VSVBP)，利用序列目标规划(Lexicographic Goal Programming)技术实现最优边缘用户分配。文献[7]针对边缘环境下微服务应用的部署问题，提出一种分布式冗余部署框架SAA-RP和GA-based Server Selection(GASS)算法，考虑到服务复合属性的同时提高了部署方案的高可用性。

对于边缘计算环境下的调度问题，相关研究在近年来逐渐被重视起来。文献[8] 提出了一种移动应用卸载算法来调度MECC环境中的数据密集型移动应用。该算法生成应用执行计划，该计划考虑应用大小、输入数据大小、可用网络带宽和移动设备能量。文献[9]研究了具有社会群体的复杂网络在边缘计算环境下的最大完工时间的最小化工作流调度问题，并将该问题表述为一个整数规划。他们提出了一系列任务分配策略来处理工作流中不同类型的任务。这些策略进一步与贪婪策略相结合，以构造改进的贪婪搜索(IGS)算法，该算法能够生成满足所有约束的可行解。此外，他们还提出了一种改进的复合启发式(ICH)算法，该算法使用IGS来初始化可行解，并使用两层改进方案来进一步提高初始解的质量。文献[10]研究了边缘计算中在线截止期感知的任务调度问题。使用网络和计算资源的联合优化来研究具有带宽约束的截止期感知任务调度和调度。考虑网络和计算资源的管理，以满足最大数量的截止日期。提出了一种在线算法，称为D-dedas，它贪婪地调度新到达的任务，并考虑是否替换一些现有的任务，以满足新的截止日期。文献[11]研究了边缘云系统中的在线作业调度和调度问题，其中作业由移动设备以任意顺序和时间发布，并卸载到具有上传和下载延迟的无关服务器，并为这个问题提出了一个通用模型，目标是使所有工作的总WRT最小化，称之为OnDisc。文献[12]基于重复的Stackelberg博弈提出了一种在线任务调度的边缘计算。文献[13]将P-BPM任务处理模式应用到边缘计算场景下，基于蚁群算法提出了一种边缘计算环境下多目标优化离线任务调度算法，称之为P-MACO(multi-object ant colony optimization schedule algorithm)算法。但蚁群算法往往需要很长的运行时间进行迭代才能得出较优解，无法满足边缘环境下服务请求的低时延要求。文献[14]提出了一种基于动态反馈的Kubernetes调度算法，提升调度算法对节点硬件资源的感知能力，以提高边缘计算环境下的应用调度效率和硬件资源的利用效率。但该算法仅关注应用对资源的需求情况，最终的优化目标是服务器之间的负载均衡效果，对于服务请求的调度策略研究，能够提供的参考十分有限。文献[15]提出了一种基于改进GA的云计算任务调度算法，但云计算环境下的任务调度策略，其有效性在边缘计算环境下并不能得到保证。有些文献选择从边缘云架构、负载均衡等角度设计调度策略。文献[16]设计了一套用于移动计算的分层边缘云架构，以在高峰时段平衡边缘服务器的负载，从而提高任务处理速度。文献[17]将工作负载调度问题转化为马尔可夫决策过程，并使用Lyapunov优化策略解决该问题，从而满足任务调度的实时性。有些文献将工作重点放在缩短任务的响应时间和提高资源利用效率，如文献[18]基于Min-Min算法提出了LBIMM算法，优化了任务完成时间和资源利用率，但忽略了任务超时问题。

综上所述，可以看出目前国内外对于边缘计算环境下的服务请求调度问题的研究尚不充分，往往只从任务运行中的资源占用、运行时间等特征中选取一个重点，而忽视了其他优化目标。本文将服务请求中的资源占用率、优先级、运行时间、超时率特征结合起来，充分考虑各项优化指标，研究服务请求调度策略，提高边缘计算环境下的服务质量。

**1.3论文主要工作**

边缘计算环境下需要同时考虑到有限的服务资源以及低响应时间的需求。基于此，本文模拟实际应用场景，提取任务运行时的主要特征，使用指针网络和强化学习技术建立模型，设计了一套边缘环境下的服务请求调度策略。本文具体的工作内容如下：

(1) 根据边缘环境下的服务请求场景，提取出服务请求中的主要特征：资源占用率（包括CPU、I/O、带宽、内存）；优先级；超时时间与运行所需时间。结合服务请求与边缘服务器的覆盖情况。抽象成服务请求向量作为本文的数据格式。

(2) 基于指针网络和强化学习，设计训练模型，按照任务向量的格式生成数据集作为模型的训练样本，将任务的执行顺序作为输出，将多个服务器完成所有任务所需总时间的平均值、任务优先级、总超时率作为优化目标。使用强化学习的方式对该指针网络进行训练，得出训练结果。

(3) 参考真实场景数据集生成数据进行实验，将训练好的模型效果与贪心算法、随机算法以及操作系统中的多级反馈队列调度算法进行比较。从而证明了该模型下的服务请求调度策略，在平均运行时间、优先级、总超时率方面的优化效果，要优于贪心算法、随机算法和多级反馈队列调度算法。

本文主体部分的剩余内容安排如下：

第二章是场景与问题描述，介绍了本文涉及到的边缘计算环境下服务请求场景，并提取任务特征作为特征向量，介绍特征向量各个维度含义与约束。对本文待解决的问题作出简单描述。

第三章是解决方案设计，对第二章所描述的问题进行数学定义并建立模型，对于本文用到的指针网络结构，强化学习训练过程等进行了阐释。

第四章是实验设计与结果分析，生成数据集进行实验，将本文中的模型与其他算法进行比较，得出结论。

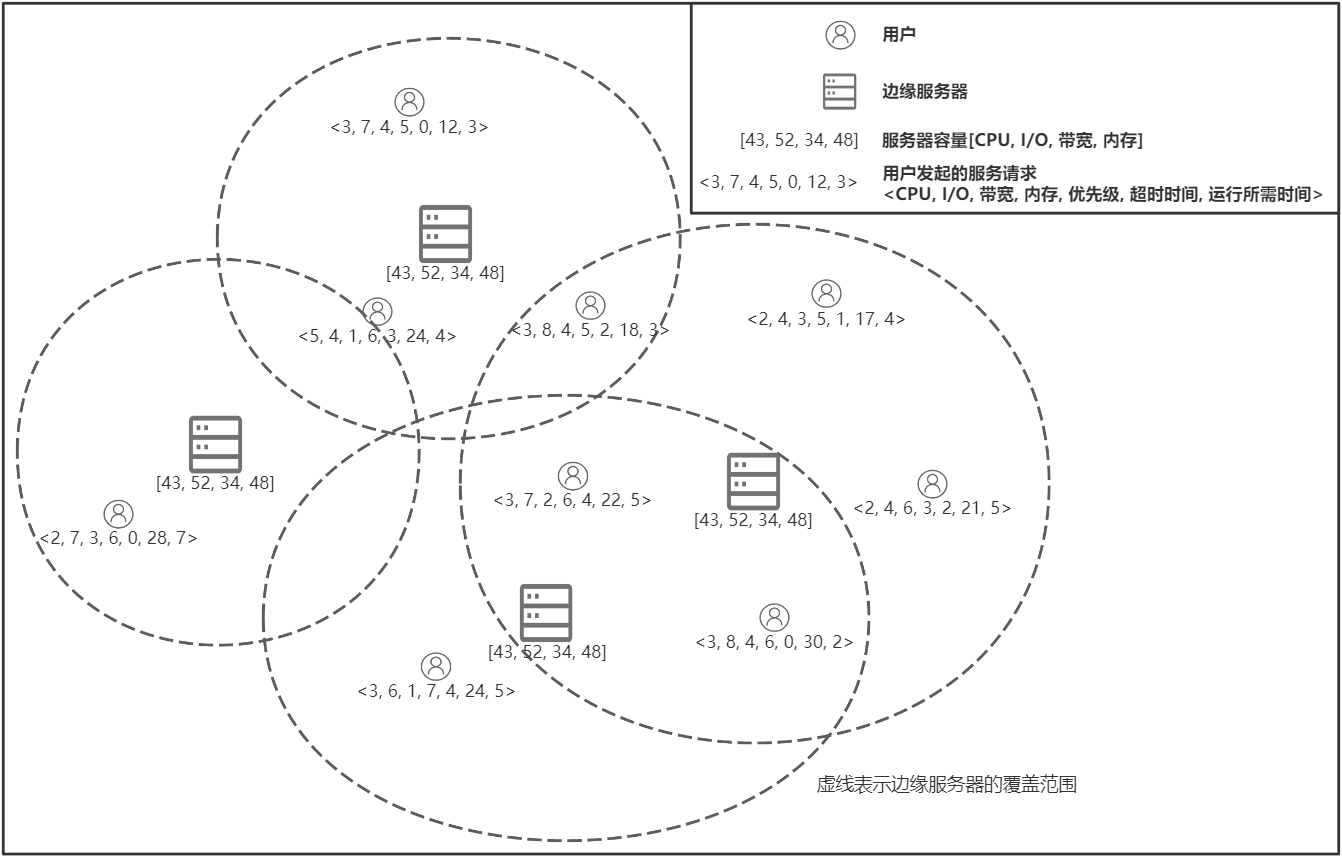
**2场景与问题描述**

**2.1场景介绍**

本节主要介绍了论文涉及到的边缘计算环境下服务请求场景，以及该场景下的数据格式定义，包括数据中不同维度的含义与约束。

**2．1．1** **场景示例**

手机游戏是边缘计算应用场景下的一个典型示例，尤其是大型多人在线游戏(MMOG) [20]。对于这种类型的游戏，它的运行往往需要大量资源，由于这些消耗大量资源的游戏实例是在远程的云服务器上运行的，因此即便是使用手机这种性能十分有限的客户端的玩家，也可以轻松游玩大型多人在线游戏。然而，在游戏运行的过程中会把复杂的运算任务如图形渲染通过任务卸载的方式将其放在云中运行，而远程云服务器与终端之间的高数据传输时延会显著增加终端玩家的响应延迟，从而影响到玩家的游戏体验。因此，将这些高资源消耗的服务部署在距离玩家更近的边缘环境，利用边缘计算的能力解决上述问题是十分有效且常见的解决方案[21]。图2.1将该边缘计算场景进行了抽象展示。



图**2.1** 边缘计算场景示例

**2．1．2** **数据格式**

如图2.1所示，对于边缘服务器而言，我们将它的负载能力，抽象成为一个四维向量。其中代表服务器CPU容量；代表服务器I/O容量；代表服务器总带宽；代表服务器内存容量。只要资源充足，边缘服务器在同一时刻就能运行尽可能多的服务请求。每一台边缘服务器都有不同的覆盖范围，在图2.1中由虚线表示，在覆盖范围内的用户，就可以向该边缘服务器请求服务。而对于用户，我们认为每个用户就对应着一个不同的服务请求，我们将传统服务请求特征中的优先级、运行所需时间、超时时间，与服务器资源中的CPU、I/O、宽带、内存占用结合在一起抽象成为一个七维向量。其中前四维对应着本次服务请求所需要占用的边缘服务器资源，分别是CPU、I/O、带宽、内存，和服务器容量一一对应。后三维中，代表该请求的优先级，通常是人为定义的，在该场景下，我们定义值越低，代表服务请求具有越高的优先级；代表该请求的超时时间，当请求到达服务器后经历时刻仍未得到处理，则认为该请求超时；代表边缘服务器处理该请求所需要的时间。由于一个边缘服务器能接受多个服务请求，同时一个用户能向多个边缘服务器发送服务请求。因此该场景下还隐含着边缘服务器与服务请求之间的覆盖关系。在数据处理中，我们根据服务请求的坐标、边缘服务器的坐标、边缘服务器的覆盖范围这三类数据统计出能接收某一服务请求的所有边缘服务器的集合，将其作为该服务请求的第八个特征，具体过程将在第三章的问题建模部分进行展开说明。

**2.2问题描述**

这一节针对本论文所涉及到的边缘计算环境中多服务器服务请求调度问题做出了详细描述。

对于一台边缘服务器，同一时刻通常有多个用户去请求它的服务，当查询率(QPS)高到一定程度的时候，边缘服务器有限的资源使得它往往不能同时满足大量的服务请求。此时就需要这些服务请求到达边缘服务器后进行排队，如表2.1所示。

对于多个边缘服务器，每台服务器都能接受不同的服务请求，本文中只考虑服务器内部的调度策略，不考虑服务器之间的协同。对边缘服务器而言，越早执行完等待中的任务，意味着服务器资源利用率越高，提供的服务质量也就越高；对于用户而言，优先满足重要性高的服务请求，同时保证尽可能的不超时，能显著提升用

表**2.1** 服务请求排队

|  |  |
| --- | --- |
| 序 号 | 服务请求 <C, O, B, M, λ, T, t> |
| 0 | <3, 8, 4, 5, 2, 18, 3> |
| 1 | <2, 4, 3, 5, 1, 17, 4> |
| 2 | <3, 7, 2, 6, 4, 22, 5> |
| 3 | <2, 4, 6, 3, 2, 21, 5> |
| 4 | <3, 8, 4, 6, 0, 30, 2> |
| … | …… |

户体验。基于此，本论文所讨论的问题是：如何安排等待中的服务请求执行顺序，最终使得场景下所有边缘服务器执行完全部服务请求所需时间尽可能的短，同时兼顾高优先级任务提前执行，以及低的超时率。

综上所述，本论文描述的调度策略即是边缘计算环境下服务器的服务请求执行调度策略，其本质是一个多目标优化问题。共有三个优化目标，分别为平均运行时间、优先级和总超时率。下一章我们提出通过指针网络结合强化学习的方式对该调度策略进行构建。

**3基于指针网络的服务请求调度**

**3.1问题建模**

本节的主要内容是针对第二章所描述的问题进行建模，对于各项数据及优化指标做出数学定义。

**3．1．1** **数据定义**

(1) 边缘服务器

我们将边缘服务器集合定义为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.1)** |

其中单个服务器使用一个四维向量表示为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.2)** |

向量的各个维度分别表示边缘服务器的CPU、I/O、带宽、内存容量。

(2) 服务请求

假设在某一时刻，某台边缘服务器中有个服务请求正在等待执行。我们将服务请求集合定义为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.3)** |

其中单个请求使用一个八维向量表示为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.4)** |

其中前四维向量分别表示执行请求所需的CPU、I/O、带宽、内存；表示请求的优先级，我们定义值越低，优先级越高；表示请求的超时时间；表示请求的运行所需时间；根据边缘服务器对于服务请求的覆盖关系统计得出，它表示能够接收该服务请求的所有边缘服务器集合。用表示该服务请求的坐标；表示边缘服务器的坐标，表示边缘服务器的覆盖半径。表示为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.5)** |

(3) 请求超时

对于请求集合中的每个请求，都有自己的超时时间和运行所需时间，假设在一次调度中，请求在时刻得到响应。我们将超时的请求集合定义为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.6)** |

(4) 超时惩罚

当服务请求已经超时，默认边缘服务器是无法对它进行执行的，因此，该策略下会产生一个潜在问题：当调度结果显示的超时率较高，导致许多服务请求超时而未得到执行，使得总运行时间缩短。而该条件下的运行时间缩短并不是我们想要的结果。因此，在服务请求超时的情况下，我们定义时间惩罚集合：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.7)** |

集合中的每个元素代表各个边缘服务器的惩罚时间。

具体策略如下：定义服务器负载系数，其值为一个正整数，它的含义是：平均情况下每台边缘服务器同时能容纳的服务请求个数，如当时，在平均情况下每台边缘服务器能同时运行5个服务请求。在每次调度之初，定义值为0，表示第台边缘服务器的惩罚时间。当服务请求发生超时的时候，修改的值，修改规则为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.8)** |

**3．1．2** **输入输出**

(1) 输入

在该问题模型下，首先根据服务请求中的特征，随机选取边缘服务器作为服务请求发送请求的目标服务器。将不同服务器接收到的服务请求集合作为问题模型输入，在多服务器环境下，每台边缘服务器都有属于自己的输入集合。

(2) 输出

通过模型调度策略得出的结果，以服务请求的下标序列作为输出展示。输出结果即代表边缘服务器集合对于服务请求集合的执行顺序，需要注意的是，当边缘服务器剩余资源充足的时候，能同时运行多个任务，即服务请求在服务器中是并行执行的。输出定义为：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.9)** |

在多服务器环境下，模型调度策略在边缘服务器内部独立执行，互不干涉。每台服务器都有属于自己的输出。表示与输入的请求集合相关联的解的序列，其长度与的长度相同，中的每一个元素都代表一个服务请求的下标，与旅行商(TSP)问题中将城市下标序列作为输出结果相类似。

**3．1．3** **优化目标**

(1) 平均运行时间

平均运行时间指所有边缘服务器执行完服务请求所需要时间的平均值，在不考虑请求超时的情况下，已知边缘服务器集合。根据一个输入输出对(,)计算运行时间的算法伪代码如算法1所示。

|  |  |
| --- | --- |
| Algorithm 1 运行时间计算 | |
| 1: | /\*m: 服务器数，n: 服务请求数，C: 模型输出，Q: 模型输入\*/  map ← []\*m /\*记录正在服务器中运行的任务\*/  T\_map ← [0]\*m /\*记录各个服务器所耗费的时间\*/  **for** i = 0 **to** n-1  idx ←  q ← Q[idx]  /\*根据的特征随机选取服务器\*/  **if** [0] < q[0] **or** [1] < q[1] **or** [2] < q[2] **or** [3] < q[3]  **while** MIN(t) > 0 /\*t[j]为map[j]中所有任务的执行所需时间集合\*/  T\_map[j] ← T\_map[j] + 1  t[j] ← t[j] – 1  **end while**  **while** min(t) <= 0 /\*将所有执行完毕的任务从map中移除\*/  k ←ARGMIN(t)  ←+ map[k] /\*归还占用的资源\*/  map[j].REMOVE(k)  **end while**  **else**  ←– [q[0], q[1], q[2], q[3]]  map[j].ADD(q) /\*将q加入map\*/  **end for**  T\_map[j] ← T\_map[j] + MAX(t[j])  **return** T\_map |
| 2: |
| 3: |
| 4: |
| 5: |
| 6: |
| 7: |
| 8: |
| 9: |
| 10: |
| 11: |
| 12: |
| 13: |
| 14: |
| 15: |
| 16: |
| 17: |
| 18: |
| 19: |
| 20: |
| 21: |
| 22: |
| 23: |

返回结果即为该(,)对应的边缘服务器运行时间集合。运行时间优化指标用表示如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.10)** |

代表第台边缘服务器的运行时间；代表第台边缘服务器的惩罚时间(公式(3.8))。该优化目标是使的值尽可能的小。

(2) 优先级

该策略下优先级的优化目标为：让优先级高的服务请求尽可能先运行，优先级低的服务请求后运行。令表示中第个元素对应服务请求的优先级。在该模型下，的值越小，代表该服务请求具有越高的优先级。我们针对每个边缘服务器的输入输出对(,)计算它的优先级，最后对所有边缘服务器优先级指标进行求和。将最终的优先级优化指标用表示如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.11)** |

公式(3.11)中表示所有服务请求中优先级的最大值；表示服务请求的总个数。公式(3.11)的作用是，为越早执行的服务请求的优先级赋予越大的权重，然后进行求和，使得公式最终的效果是：当优先级越高的服务请求越早执行，的值也就越小。因此该优化目标是使的值尽可能的小。

(3) 总超时率

在算法1运行的过程中，同时要考虑到服务请求的超时问题，在判断边缘服务器剩余容量(line 8)之前，要根据当前已运行时间(line 3)以及当前服务请求q(line 6)中的超时时间与运行所需时间，来判断服务请求q是否已经超时。将超时率优化指标用表示如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.12)** |

其中表示所有服务请求中超时的服务请求个数，表示服务请求总个数。该优化目标是使的值尽可能的小。

(4) 归一化

为了解决该多目标优化问题，我们在本文中选取的策略是采用加权线性组合(WLC)的方式将多目标优化问题转化为单目标优化问题。在此策略下，要想获得更好的效果，重点是选取合适的权重赋予各个优化目标。针对本文提出的问题，我们认为以上三种优化目标具有相同的重要程度，基于此，我们选择对这三种优化目标进行归一化处理，从而使得每种优化目标的改变对于模型的影响程度大致相同。得出最终的优化目标用表示如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.13)** |

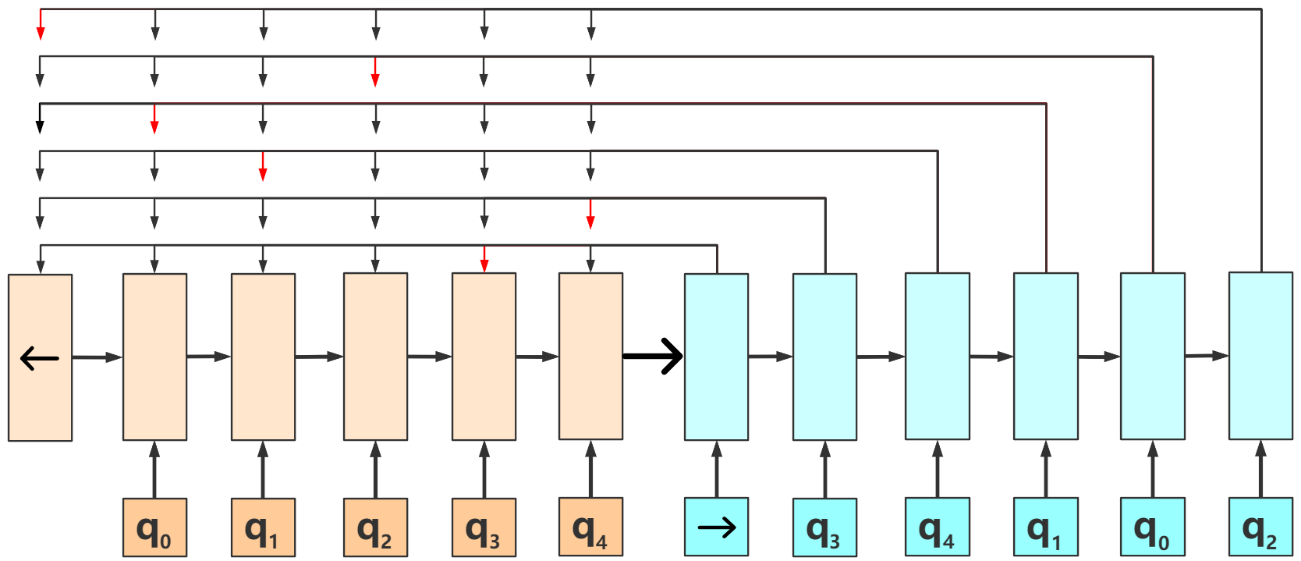
**3.2指针网络**

由于在该场景下，输入和输出均可以进行序列化的操作，采用Seq2Seq(sequence to sequence)模型可以有效解决该类问题。如文献[22]利用两个长短期记忆网络(LSTM)分别作为序列的编码(encode)阶段和解码(decode)阶段解决语言翻译问题。Seq2Seq模型有很多种选择，传统Seq2Seq模型的缺点是其输出序列长度取决于字典长度，而字典长度是事先定义好的，如解决旅行商(TSP)问题时，将城市序号作为字典，当输入的城市数量大于时该方法将失效，因此传统Seq2Seq模型在解决输出序列长度依赖输入序列长度的问题时可拓展性较差。

本文选择采用文献[23]中提到的指针网络模型，该模型改进了传统Seq2Seq模型的注意力机制[24]，使指针网络的输出能有效指向输入序列的特定位置。它与其他Seq2Seq模型的区别是，传统Seq2Seq输出的是针对字典的概率分布；而指针网络输出的是针对输入序列的概率分布。从而使输出序列不再依赖于字典而是与输入序列相关。

**3．2．1** **指针网络结构**

指针网络由两个循环神经网络(RNN)组成，分别称之为编码器(encoder)和解码器(decoder)，每个循环神经网络均由多个长短期记忆网络(LSTM)单元组成。其简要结构如图3.1所示。



图**3.1** 指针网络结构

其中左边为编码器，右边为解码器。对输入进行序列化后由编码器读取，每步读取一个服务请求，图3.1中表示第个服务请求，编码器作为一个循环神经网络，将输入序列转化为一种中间向量的表达形式，作为解码器的输入。解码器在每一步中利用注意力机制选取输入序列中权重最大的服务请求，当一个服务请求被选定后，它将作为解码器下一步的输入。

**3．2．2** **指针网络的注意力机制**

图3.1中使用表示编码器在第步的隐藏层输出；使用表示解码器在第步的隐藏层输出。

对于传统Seq2Seq模型使用的注意力机制，其原理是将编码器(encoder)的隐藏层输出按照一定权重加和之后拼接(或者直接加和)到(decoder)的隐藏层输出上，以此作为额外信息，起到所谓“软对齐”的作用，从而提高了整个模型的预测准确度。传统注意力机制公式如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.14)** |
|  |  | **(3.15)** |
|  |  | **(3.16)** |

公式(3.14)中的、、可以训练的参数。公式代表解码器在第步时分配给输入序列的权重。公式(3.16)使用权重与编码器在第步的隐藏层输出加权求和，把得到的加到解码器在第步的隐藏层输出上，最后解码器根据相加后的新隐藏层状态进行预测得出模型输出。

通过公式(3.15)可以看出本身就是针对于输入序列的权重。因此指针网络简化了传统的注意力机制，将看作指向输入序列的指针，每一步解码器的输出都选择权重最大的元素。因此指针网络的注意力机制公式如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.17)** |
|  |  | **(3.18)** |

公式(3.18)使用softmax函数将向量归一化为输出序列的概率向量，其中输出的向量维度和输入保持一致，公式(3.17)中的、、是指针网络待训练的参数。

**3.3强化学习**

上一节对指针网络的构造及原理做出了阐释，在这一节我们考虑如何对指针网络进行训练。

在3.1节问题建模中我们将边缘计算环境下的服务请求调度问题定义为一个多目标优化问题，各个优化目标之间互不相关。因此，本文所讨论的是一个NP难问题。对于NP难问题，使用传统的带标签的数据集进行训练是不可行的，因为模型的训练效果和标签质量密切相关，但对于NP难问题获取高质量的标签代价十分高昂。本文我们选择使用强化学习(RL)进行训练，强化学习不需要标注标签的特性能满足求解NP难问题的基本需求。

**3．3．1策略梯度**

本文采用文献[25]提出的基于策略梯度的强化学习方法来优化指针网络的参数，将指针网络中的参数定义为，用表示已知服务请求集合的情况下，采取策略时的函数值。它的期望定义如下：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.19)** |

公式(3.19)中的就代表指针网络的优化目标：最小化的期望。策略梯度的执行过程采用文献[26]提出的REINFORCE算法对指针网络参数进行优化：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.20)** |

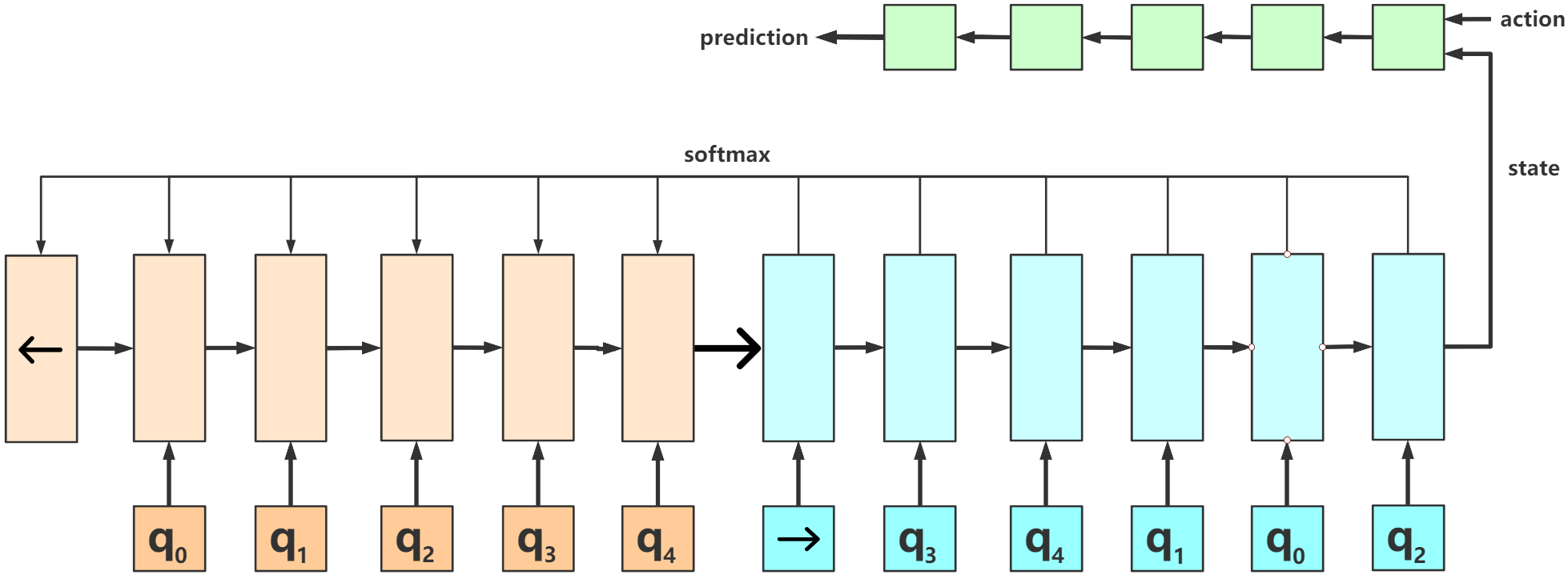
公式(3.20)中的是一个与采取的策略无关的基线函数，它的作用是通过估计的值从而减小梯度的方差。通常选取指针网络训练过程中得到的奖励的移动指数平均值。

考虑到指针网络实际训练时是将多个输入序列组合成一个batch进行并行训练，将训练数据的batch大小定义为，公式(3.20)可以近似等于：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.21)** |

**3．3．2 行动者-评论家模型**

我们选择行动者-评论家(Actor-Critic)模型对指针网络进行训练，该模型分为两个主要部分，行动者(Actor)网络和评论家(Critic)网络。针对本文的场景，行动者网络即为图3.1所展示的基本指针网络。而评论家网络基于行动者网络所产生的行为，对行为得分做出评判，从而指导行动者网络进行网络的参数更新[27]。在本文场景下，我们将评论家网络设计为由长短期记忆网络(LSTM)单元组成的循环神经网络(RNN)，与指针网络结构中的编码器或解码器相类似。其结构如图3.2所示。



图**3.2** 行动者-评论家结构

其中下半部分为行动者网络，上半部分为评论家网络。将评论家网络的参数定义为。该网络在已知输入的服务请求集合的情况下，根据行动者网络的最终状态，对采取的策略的值做出预测。评论家网络将预测的值与策略实际值的均方误差作为优化目标，采用随机梯度下降的方式进行训练：

|  |  |  |
| --- | --- | --- |
|  |  | **(3.22)** |

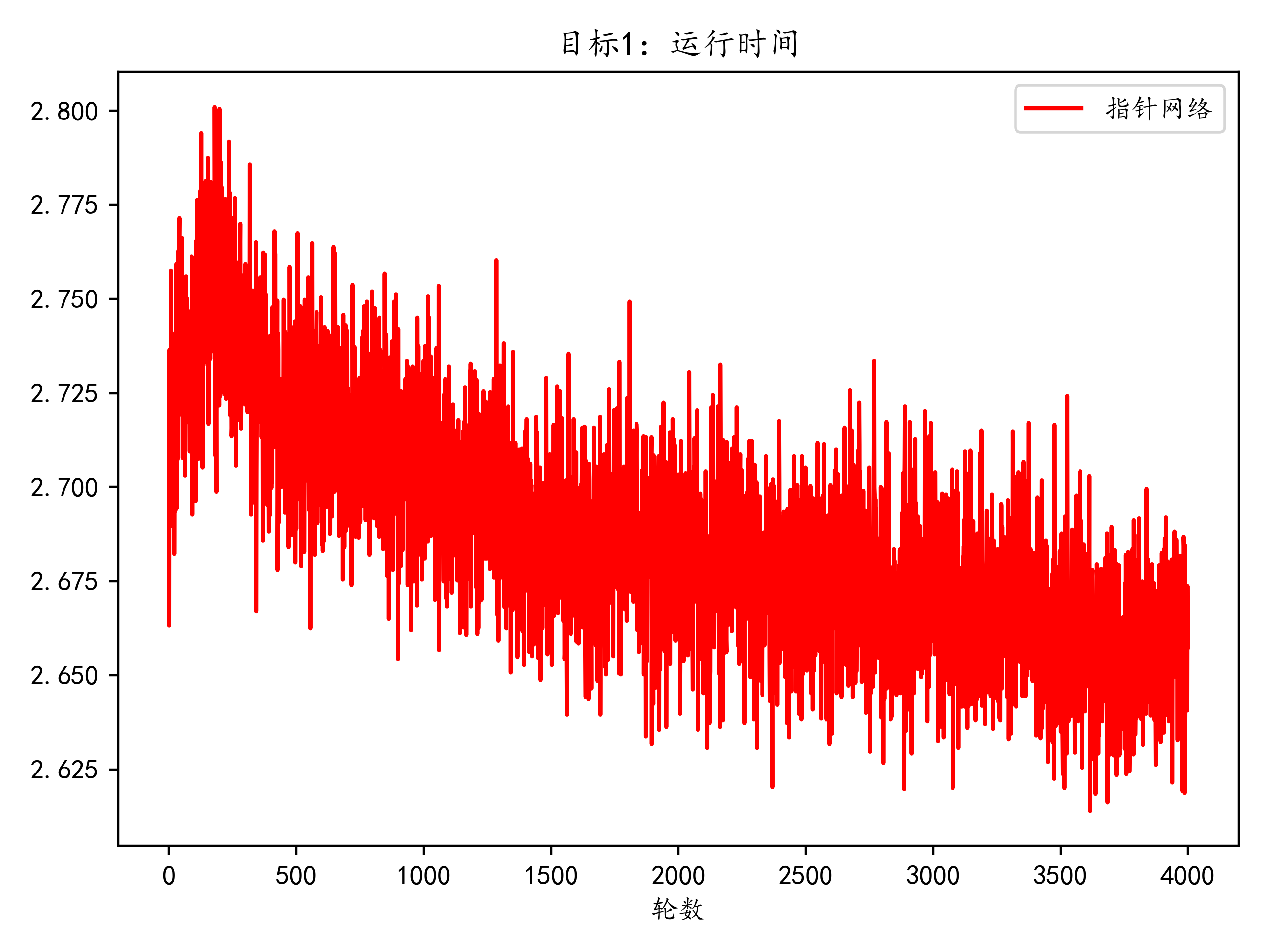
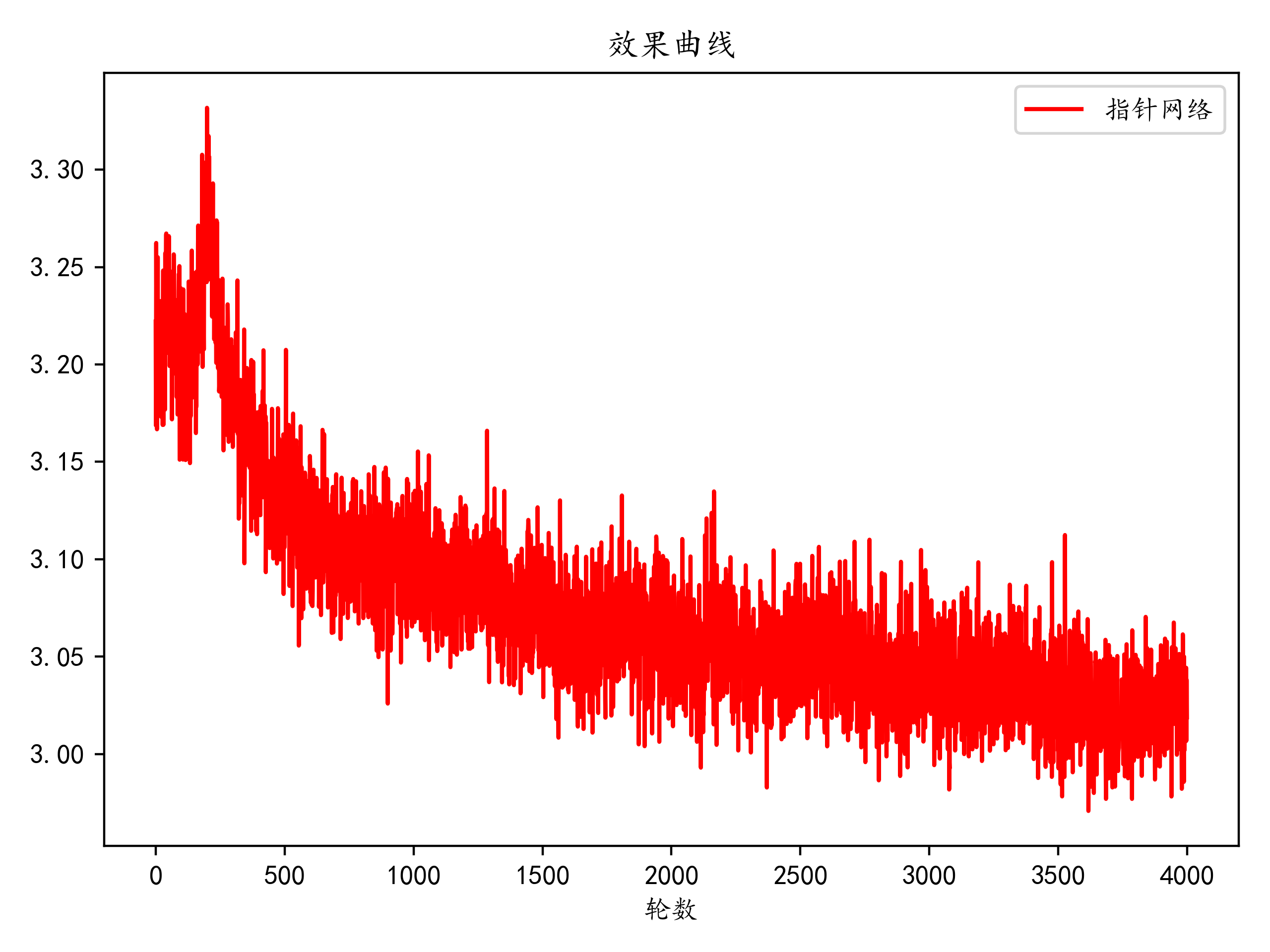
公式(3.22)中为评论家网络预测的值；为实际值。

训练过程伪代码如算法2所示。

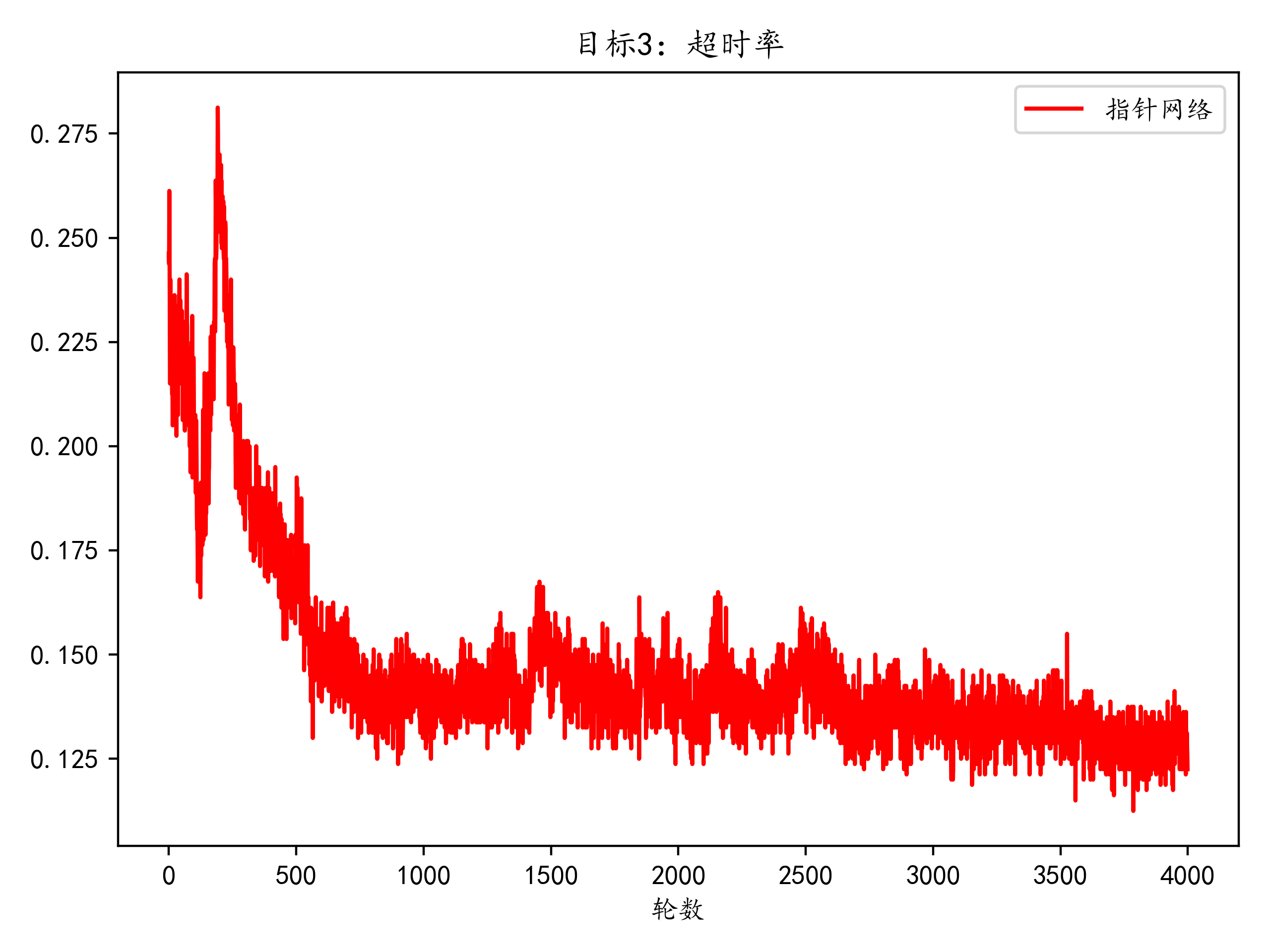
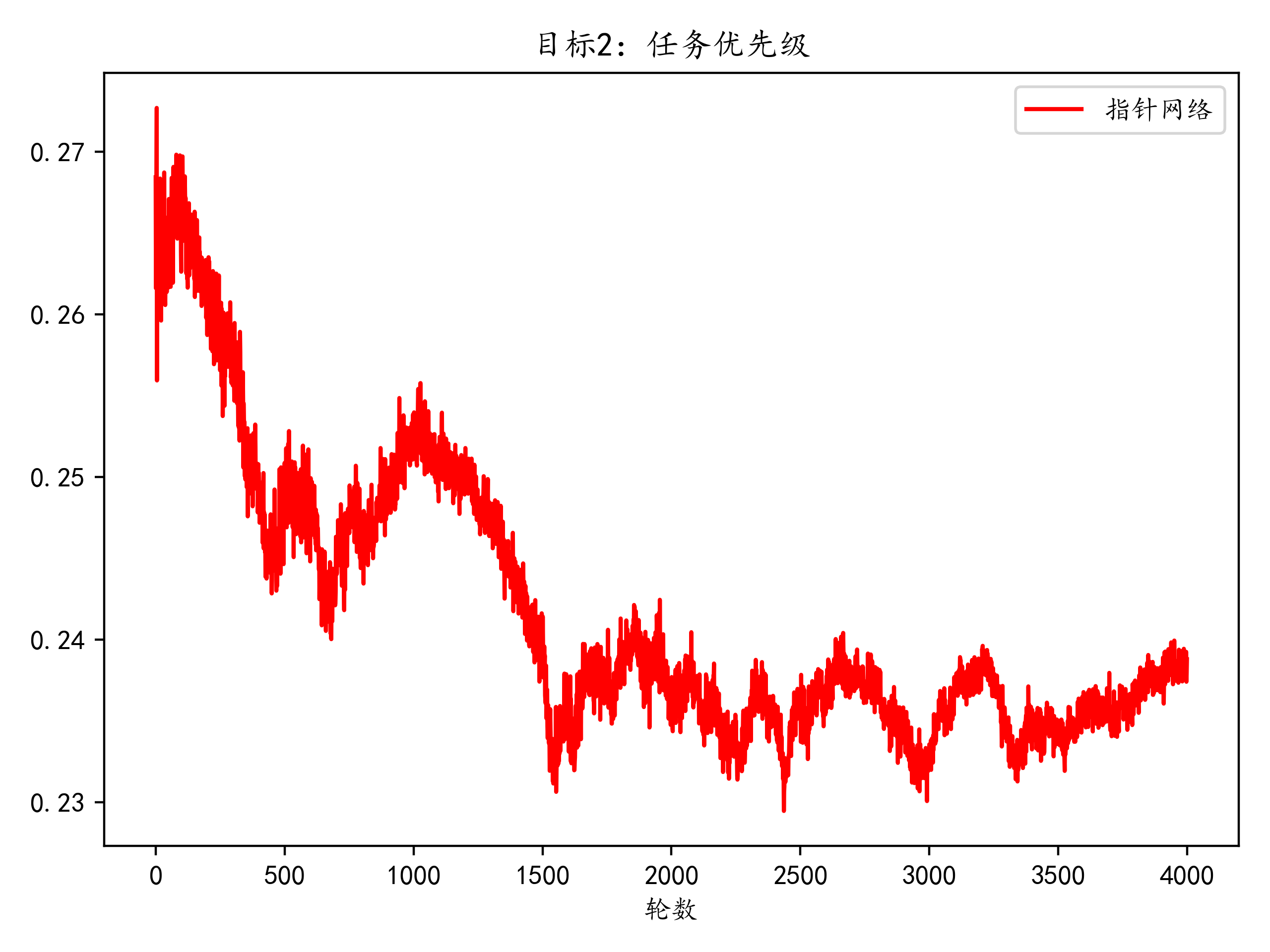
|  |  |
| --- | --- |
| Algorithm 2 训练过程 | |
| 1: | /\*定义为每轮的训练样本；为训练轮数；为每轮输入的batch大小\*/  /\*  初始化行动者网络参数&评论家网络参数  /\*  **for** step = 1 **to** N  for  for |
| 2: |
| 3: |
| 4: |
| 5: |
| 6: |
| 7: |
| 8: | for |
| 9: |  |
| 10: |  |
| 11: |  |
| 12: |  |
| 13: | **end for** |

**3.4训练效果**

对于指针网络中的编码器和解码器，以及评论家网络。我们定义它们的循环神经网络(RNN)均包含128个LSTM神经元。关于模型训练的相关参数，我们规定学习率(Learning rate)为0.0001；学习率衰减周期为500轮；衰减率为0.96；一次训练的样本数(Batch size)为16。训练4000轮后得到的效果曲线如图3.3所示，可以看到三项优化目标经过多轮训练后均显著降低。由此证明了该模型的可行性。



|  |  |
| --- | --- |
| (a) | (b) |



|  |  |
| --- | --- |
| (c) | (d) |

图**3.3** 训练效果

**4实验设计与结果分析**

**4.1数据集生成**

我们参考真实的数据集EUA①生成实验数据。该数据集包含墨尔本中央商务区中125个基站和那些基站周围的816个移动用户的地理位置[28]。我们将基站作为本文场景下的边缘服务器，将移动用户看作服务请求。在每次实验中，我们选取台边缘服务器和在这些边缘服务器覆盖范围内的个服务请求。对于公式(3.2)定义的边缘服务器的各个特征，我们按照一定规则进行随机生成；对于公式(3.4)定义的服务请求的各个特征，除了按照选择的边缘服务器和服务请求间的覆盖关系进行统计(公式(3.5))，其余特征我们按照一定规则进行随机生成。具体规则如下。

对于服务请求关于资源占用的前四维，我们将它们赋值为1~10之间的随机数；对于优先级，我们将它赋值为0~4之间的随机数；对于运行所需时间，我们将它赋值为1~20之间的随机数。

我们根据3.1.1节定义的服务器负载系数，规定边缘服务器的各个特征数据生成规则如下，以为例：

|  |  |  |
| --- | --- | --- |
|  |  | **(4.1)** |

其中是求平均值函数，是生成随机数函数，边缘服务器其余三种特征的生成规则与的生成规则相类似。

最后，我们规定服务请求中的超时时间生成规则为：

|  |  |  |
| --- | --- | --- |
|  |  | **(4.2)** |

**4.2对比算法**

我们将训练好的指针网络模型效果与随机算法、贪心算法以及操作系统中的多级反馈队列调度算法进行比较。在多服务器场景下，算法在各个边缘服务器内部独立运行，互不干涉。这一节将介绍以上三种算法的原理以及如何将以上三种算法应用到本文的场景当中。

|  |  |
| --- | --- |
|  |  |
| ① https://github.com/swinedge/eua-dataset | |

**4．2．1** **随机算法**

在该算法下，每次的服务请求执行顺序完全随机，将服务请求集合作为输入，将用随机算法打乱后的执行顺序作为输出。根据公式(3.8)、公式(3.9)、公式(3.10)、公式(3.11)分别计算出对应的、值。

**4．2．2** **贪心算法**

贪心算法在对[问题求解](https://baike.baidu.com/item/%E9%97%AE%E9%A2%98%E6%B1%82%E8%A7%A3/6693186)时，总是做出在当前看来是最好的选择，不从整体最优上加以考虑[29]。在该算法下，对于服务请求集合中的元素，我们首先对它的各项特征除和之外进行归一化后求和，我们简单认为对于前四维来说，其值越小，意味着边缘服务器能同时容纳越多的服务请求，并发量越大，所需的运行时间也就越短；对于优先级，其值越小，意味着该服务请求优先级越大；对于超时时间，其值越小，代表着越有可能发生超时，因此需要让其排在靠前位置运行从而减小超时率。综上所述，我们规定贪心算法的选择策略是：每一步选择各项特征归一化求和后的最小值对应的服务请求作为执行顺序。贪心算法的具体执行过程的伪代码如算法3所示。

|  |  |
| --- | --- |
| Algorithm 3 贪心算法 | |
| 1: | /\*定义为某服务器下输入的服务请求集合，为服务请求个数\*/  /\* : {, …, } \*/  /\* : () \*/  timeout\_avg AVG(, …, ) \* 2 /\*取超时时间的平均值再×2\*/  greed []  **for** i = 0 **to** n-1  load\_sum SUM(, , , )  greed.ADD(load\_sum / 40 + / 4 + / timeout\_avg) /\*对各个特征进行归一化并求和\*/  **end for**  []  **for** i = 0 **to** n-1  min\_idx ARGMIN(greed) /\*贪心选择策略，每次选择最小值\*/  .ADD(min\_idx)  greed[min\_idx] 10000  **end for**  **return** |
| 2: |
| 3: |
| 4: |
| 5: |
| 6: |
| 7: |
| 8: |
| 9: |
| 10: |
| 11: |
| 12: |
| 13: |
| 14: |
| 15: |
| 16: |

贪心算法的返回结果(line 16)就是经过贪心算法选择得出的服务请求执行顺序。再根据公式(3.10)、公式(3.11)、公式(3.12)、公式(3.13)分别计算出对应的、、、值。

**4．2．3** **多级反馈队列调度算法**

多级反馈队列调度算法是一种典型的CPU[处理机调度](https://baike.baidu.com/item/%E5%A4%84%E7%90%86%E6%9C%BA%E8%B0%83%E5%BA%A6)算法，[UNIX操作系统](https://baike.baidu.com/item/UNIX%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)采取的便是这种调度算法。它的优点是既能使高[优先级](https://baike.baidu.com/item/%E4%BC%98%E5%85%88%E7%BA%A7/5643121)的作业得到响应又能使短作业（进程）迅速完成[30]。由于本文所讨论的问题本质上也是一个任务调度问题，所以我们选择跟多级反馈队列调度算法也进行一次比较。我们改进了传统的多级反馈队列调度算法，使其能够运用在本文所讨论的场景当中。

在多级反馈队列调度算法中，定义了多个队列、、……。每个队列的优先级是不同的，其中优先级最高，以此类推。我们根据这一特征，将服务请求的优先级，映射到不同优先级队列中。在优先级队列内部，遵循的是时间片轮转算法，不同优先级队列分配不同大小的时间片。一般来说，优先级高的队列所拥有的时间片也相对较长。若队列中的某个服务请求经过一个时间片后尚未执行完毕，则将它移入到队列的队尾等待执行。

与传统多级反馈队列调度算法不同的是，我们的边缘服务器内部对于服务请求的执行是并行执行，而传统的时间片轮转算法，同一时刻时间片只能分给一个任务进行执行。也就是说，划分的粒度不同，本文我们只考虑服务请求在边缘服务器上的宏观并行，而传统多级反馈队列调度算法考虑的是CPU级别的微观串行。为了能让传统多级反馈队列调度算法适应本文的场景，我们将其简化为：时间片同一时刻能分给所有拥有服务器资源的服务请求进行执行。将粒度从微观串行拉回到宏观并行。改进后的多级反馈队列调度算法具体执行过程的伪代码如算法4所示。

其中LAST函数(line 18)代表当执行到最后一个优先级队列的时候算法的具体过程，当执行到最后一个优先级队列的时候，将参考算法1按照队列顺序将剩余的服务请求全部执行完，而不再采用时间片轮转的方式。算法4的返回值分别代表公式(3.10)、公式(3.11)、公式(3.12)的三个优化目标、，再根据公式(3.13)得出的值。

**4.3实验结果**

**4．3．1** **实验参数设置**

通过4.1节对于数据集生成过程的描述，我们可以看出除了随机生成的数据之

|  |  |
| --- | --- |
| Algorithm 4 多级反馈队列调度算法 | |
| 1: | /\*定义为某服务器下输入的服务请求集合，为服务请求个数，为该边缘服务器的容量\*/  /\* : {, …, } \*/  /\* : () \*/  /\* : \*/  multi\_queue [[], [], [], [], []]  T 0  priority 0  0  **for** i = 0 **to** n-1  multi\_queue[].ADD()  **end for**  slice [10, 8, 6, 4] /\*定义时间切片\*/  **for** i = 0 **to** 4  **if** i = 4  /\*当执行到最后一个队列的时候，  按照队列顺序将剩余的服务请求全部执行完，  不再采用时间片轮转的方式\*/  T, priority, = LAST(multi\_queue[i])  **else**  **for** j = 0 **to** LEN(multi\_queue[i])  multi\_queue[i][j]  **if** T + > /\*\*/  + 1  /\*根据公式(3.9)统计priority\*/  **continue**  **end if**  **if** S[0] < [0] or S[1] < [1] or S[2] < [2] or S[3] < [3] /\*资源不够\*/  T T + slice[i] /\*更新时间\*/  [] /\*归还资源\*/  **end if**  – [[0], [1], [2], [3]]  – slice[i] /\*更新时间\*/  **if** 0 /\*执行完毕\*/  /\*根据公式(3.9)统计priority\*/  **else**  multi\_queue[i+1].ADD() /\*加入下一优先级队列\*/  **end for** |
| 2: |
| 3: |
| 4: |
| 5: |
| 6: |
| 7: |
| 8: |
| 9: |
| 10: |
| 11: |
| 12: |
| 13: |
| 14: |
| 15: |
| 16: |
| 17: |
| 18: |
| 19: |
| 20: |
| 21: |
| 22: |
| 23: |
| 24: |
| 25: |
| 26: |
| 27: |
| 28: |
| 29: |
| 30: |
| 31: |
| 32: |
| 33: |
| 34: |
| 35: |
| 36: |
| 37: |
| 38: | **return** T, priority, |

外，该实验主要有三个变量：每次选取的服务请求数量；边缘服务器数量；服务器负载系数。基于此，我们采取控制变量法，每次固定两个变量，改变第三个变量，形成三个实验组进行实验。对于服务请求数量，我们选取的值为{300, 400, 500}；边缘服务器数量，我们选取的值为{5, 10, 15}；对于服务器负载系数，我们选取的值为{5, 6, 7, 8, 9}。具体参数设计如表4.1所示。

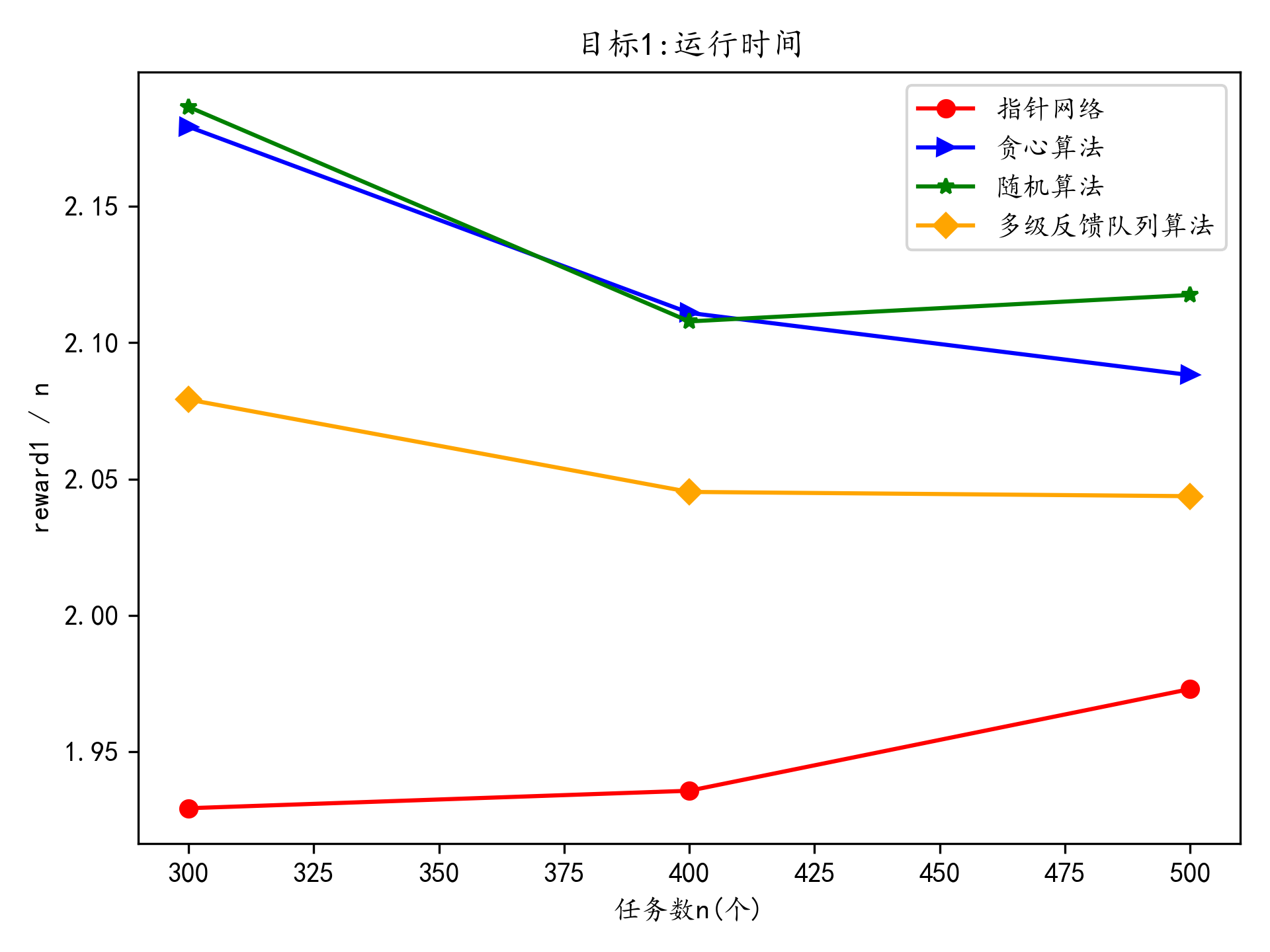
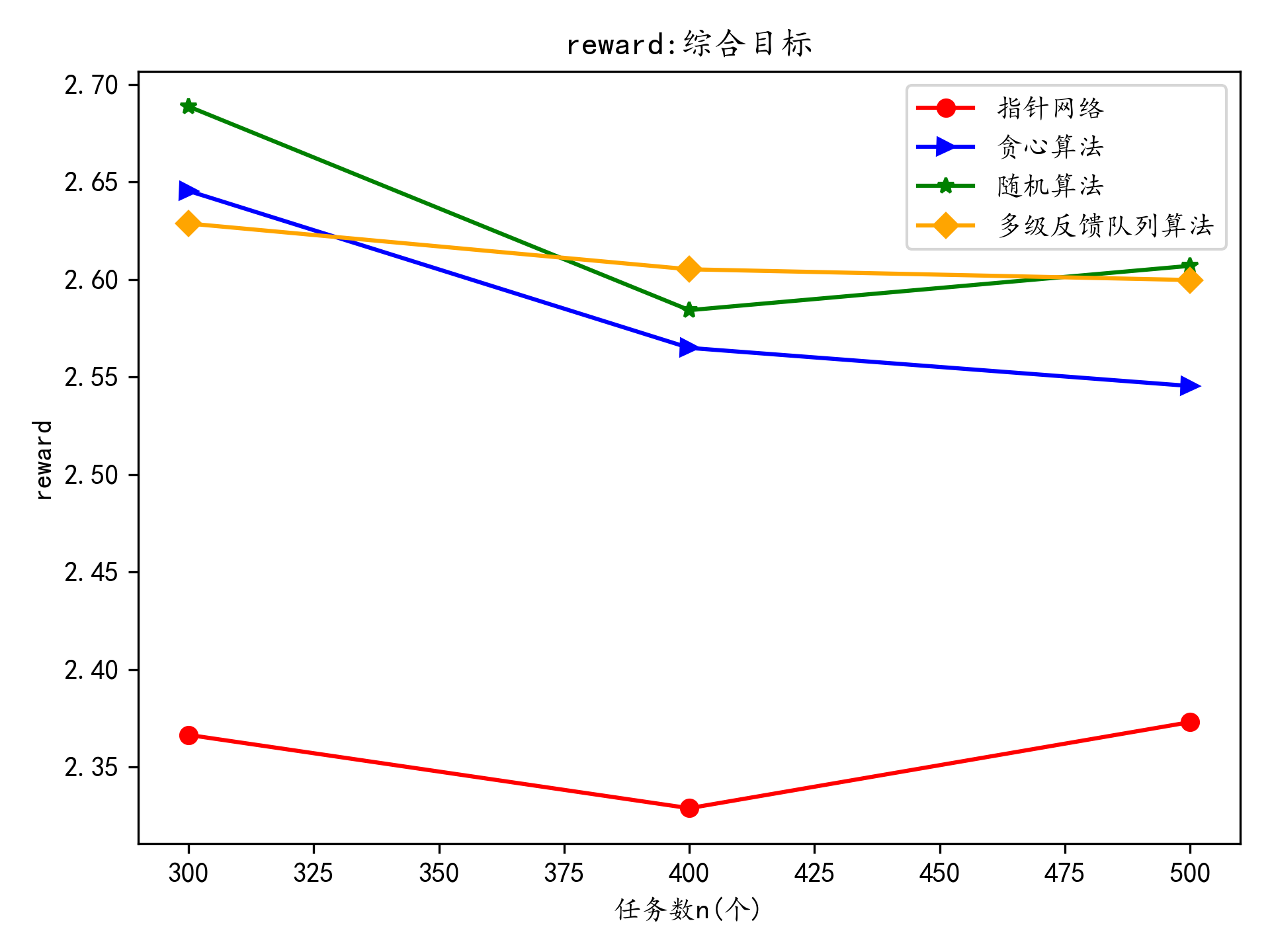
表**4.1** 实验参数设计

|  |  |  |  |
| --- | --- | --- | --- |
| 实验组 | 服务请求数量 | 边缘服务器数量 | 服务器负载系数 |
| #1 | {300, 400, 500} | 5 | 5 |
| #2 | 500 | {5, 10, 15} | 5 |
| #3 | 500 | 5 | {5, 6, 7, 8, 9} |

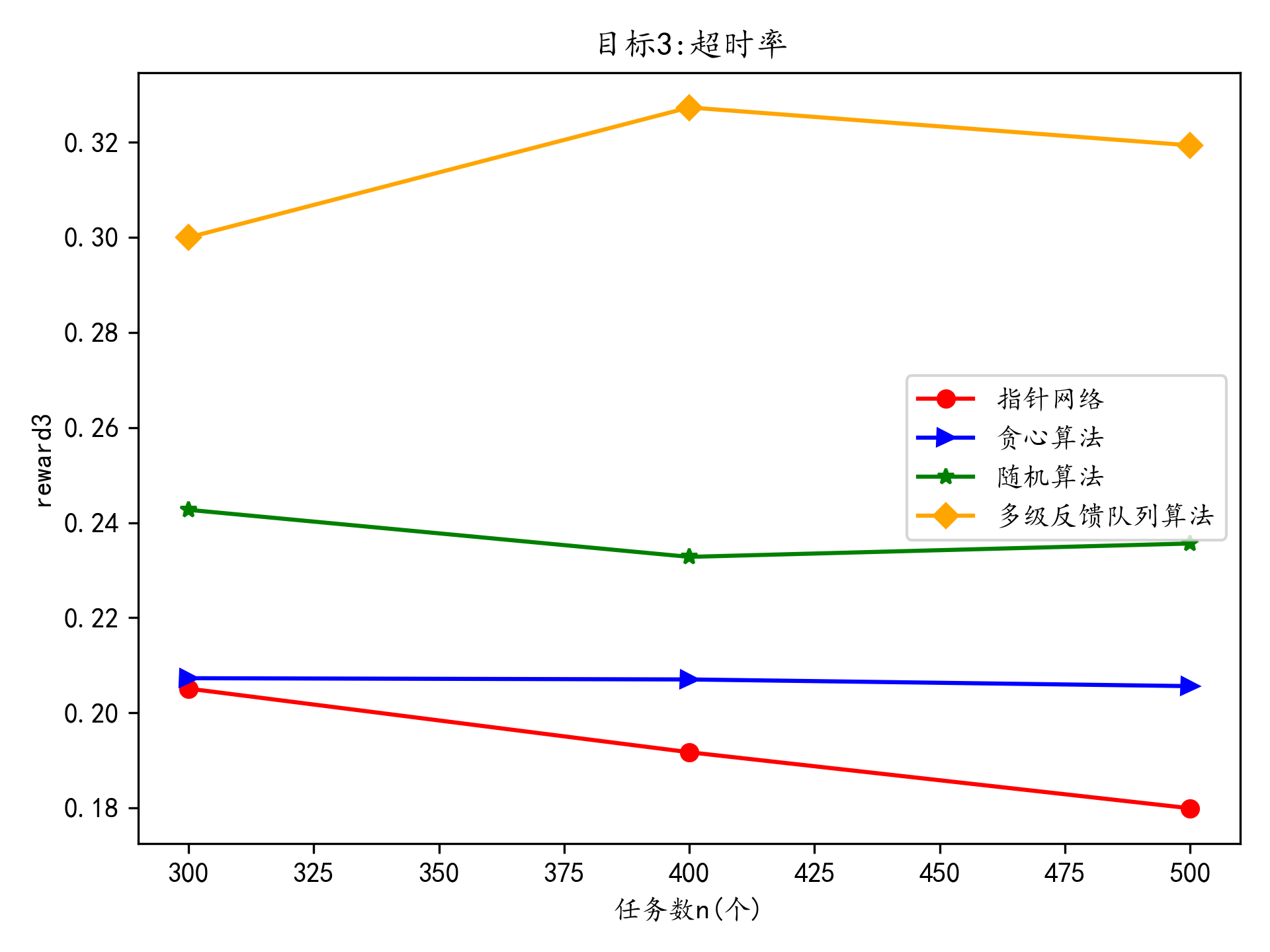
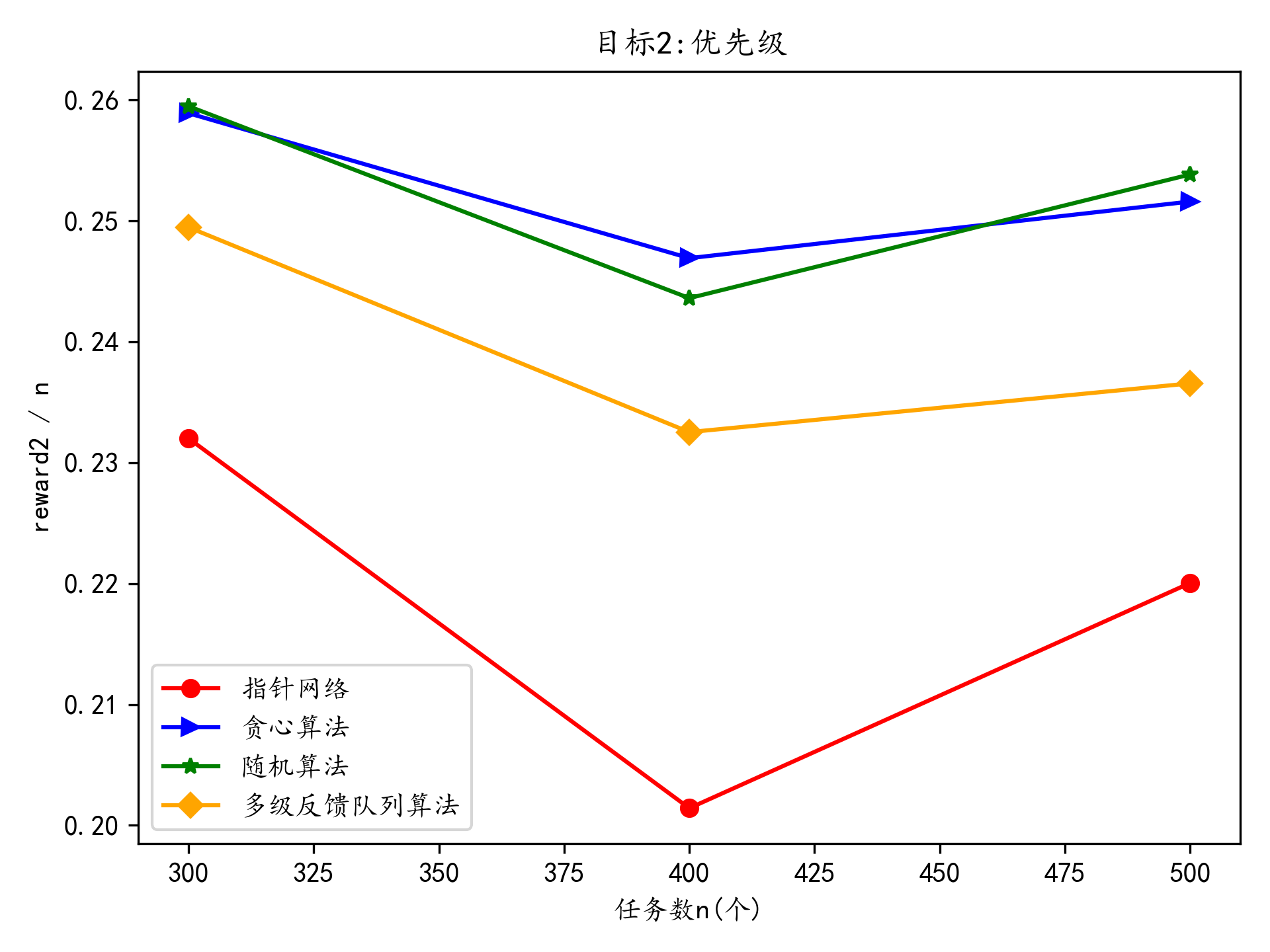
**4．3．2** **实验结果分析**

(1) 实验组#1

对于实验组#1，我们固定边缘服务器数量为5；服务器负载系数为5，改变服务请求数量的值分别为300、400、500。按不同的优化目标进行分组，将服务请求数量作为横坐标，不同优化目标的值作为纵坐标，把我们训练好的模型效果与4.2节介绍的其他算法进行对比，得出对比结果如图4.1所示。其中4.1(a)代表综合指标；4.1(b)代表平均运行时间指标；4.1(c)代表优先级指标；4.1(d)代表超时率指标。所有指标均是由公式(3.13)进行归一化后的结果。



|  |  |
| --- | --- |
| (a) | (b) |



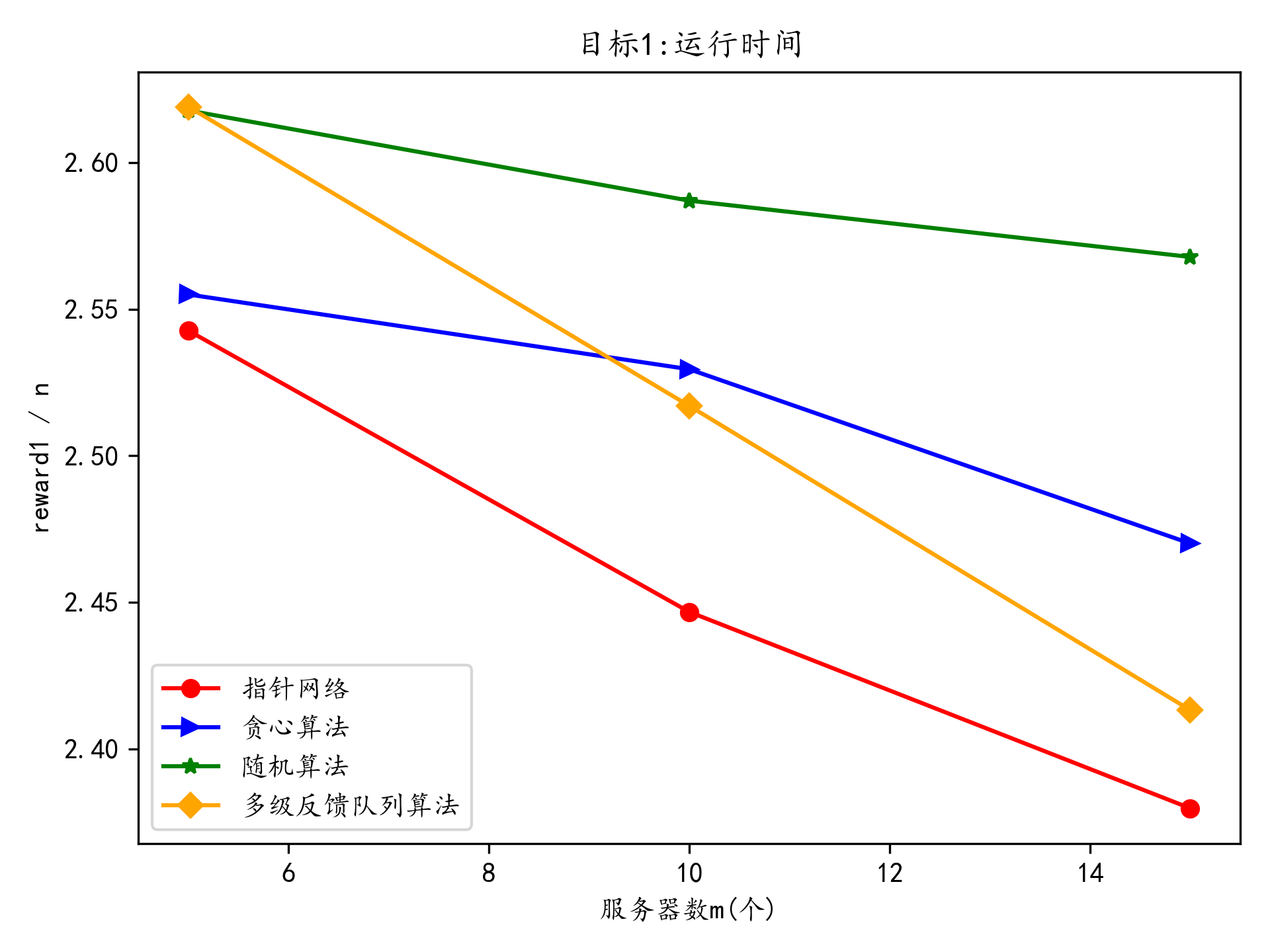
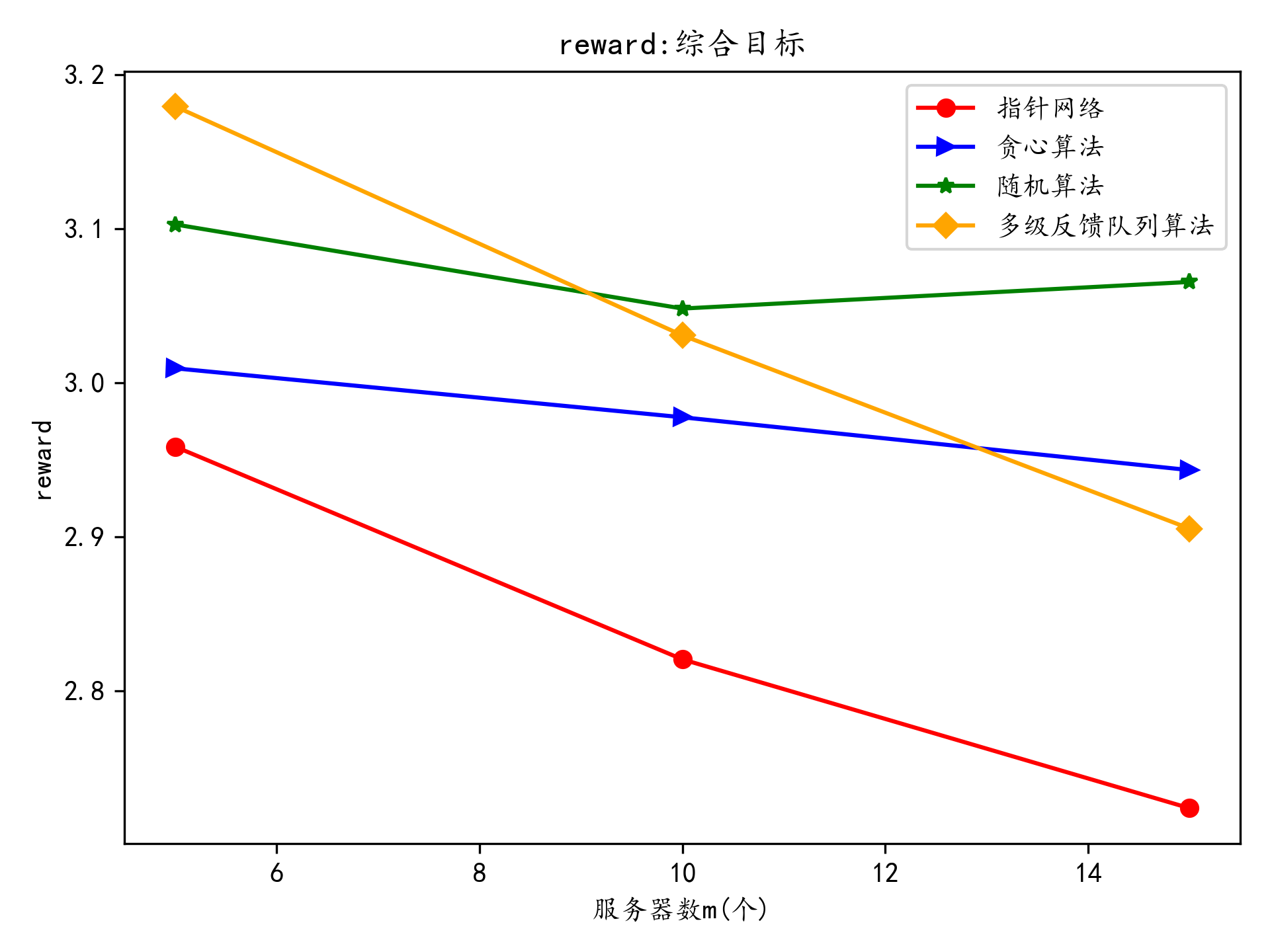
|  |  |
| --- | --- |
| (c) | (d) |

图**4.1** 实验组#1结果

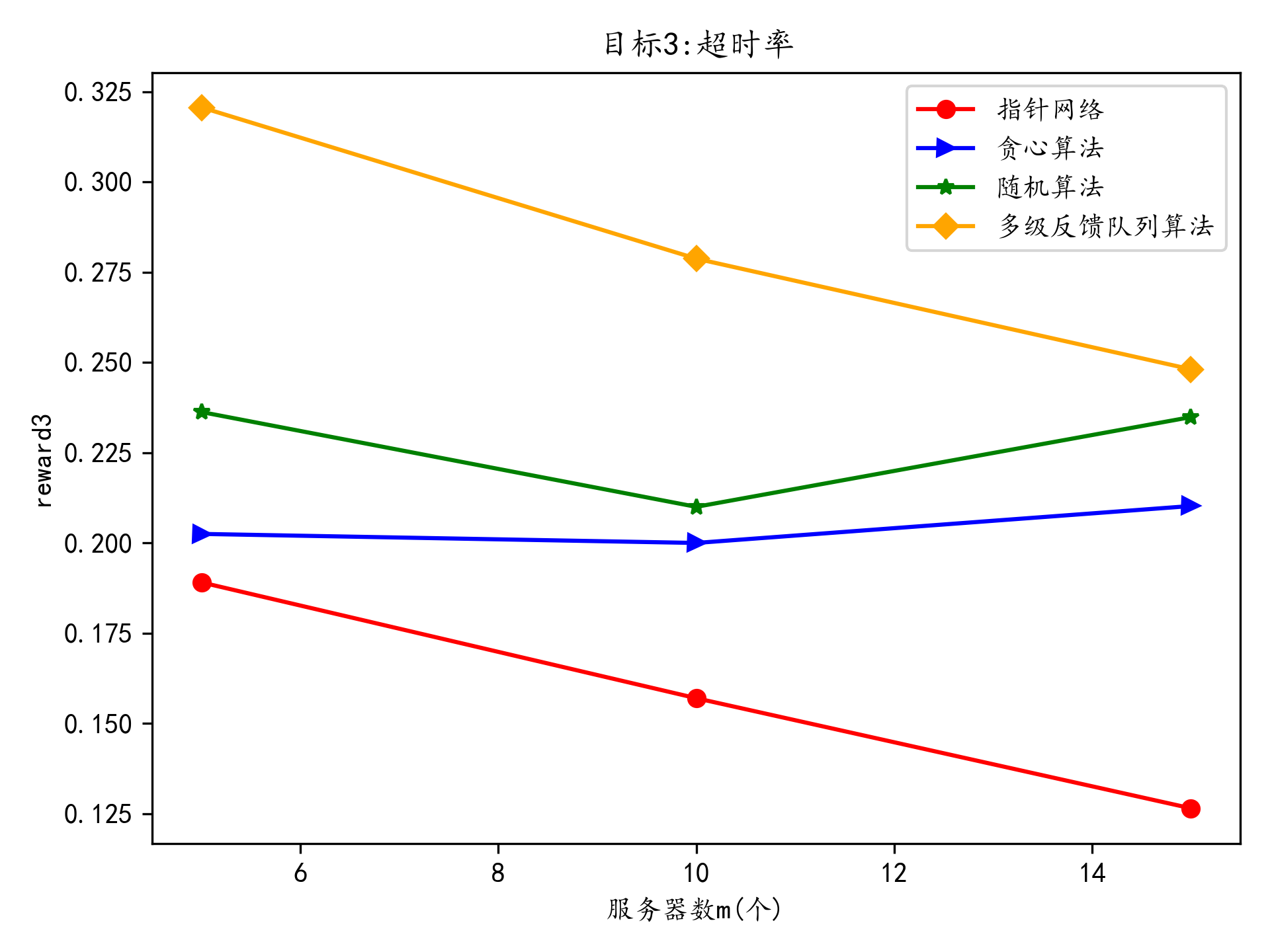
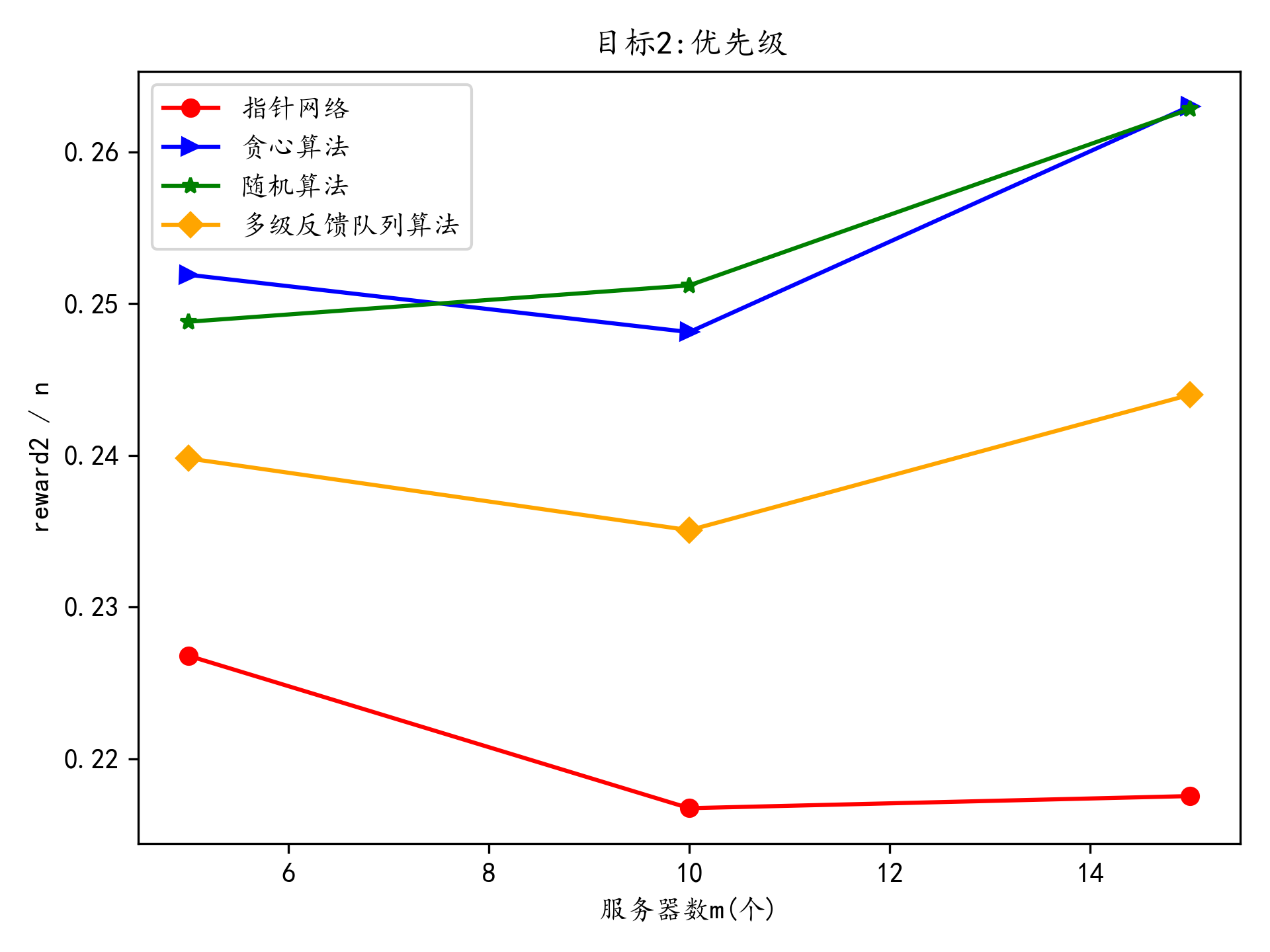
根据实验组#1的实验结果，我们可以看出在边缘服务器数量与服务器负载系数不变的情况下，改变服务请求数量的值得出的结果，我们经由强化学习训练后的指针网络模型在各个优化目标的表现均要优于其他算法。

(2) 实验组#2

对于实验组#2，我们固定服务请求数量为500；服务器负载系数为5，改变边缘服务器数量的值分别为5、10、15。按不同的优化目标进行分组，将边缘服务器数量作为横坐标，不同优化目标的值作为纵坐标，把我们训练好的模型效果与4.2节介绍的其他算法进行对比，得出对比结果如图4.2所示。其中4.2(a)代表综合指标；4.2(b)代表平均运行时间指标；4.2(c)代表优先级指标；4.2(d)代表超时率指标。所有指标均是由公式(3.13)进行归一化后的结果。



|  |  |
| --- | --- |
| (a) | (b) |



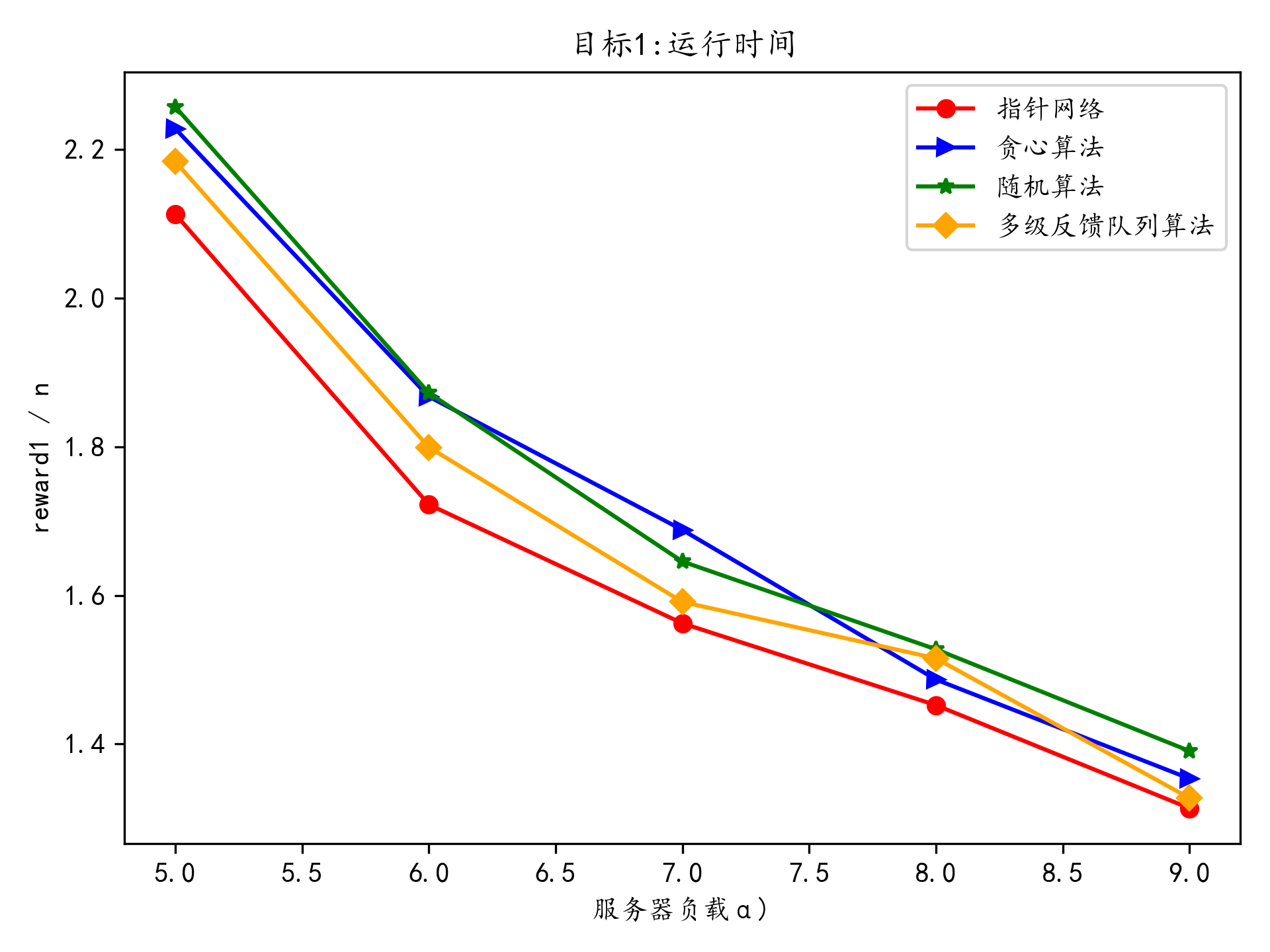
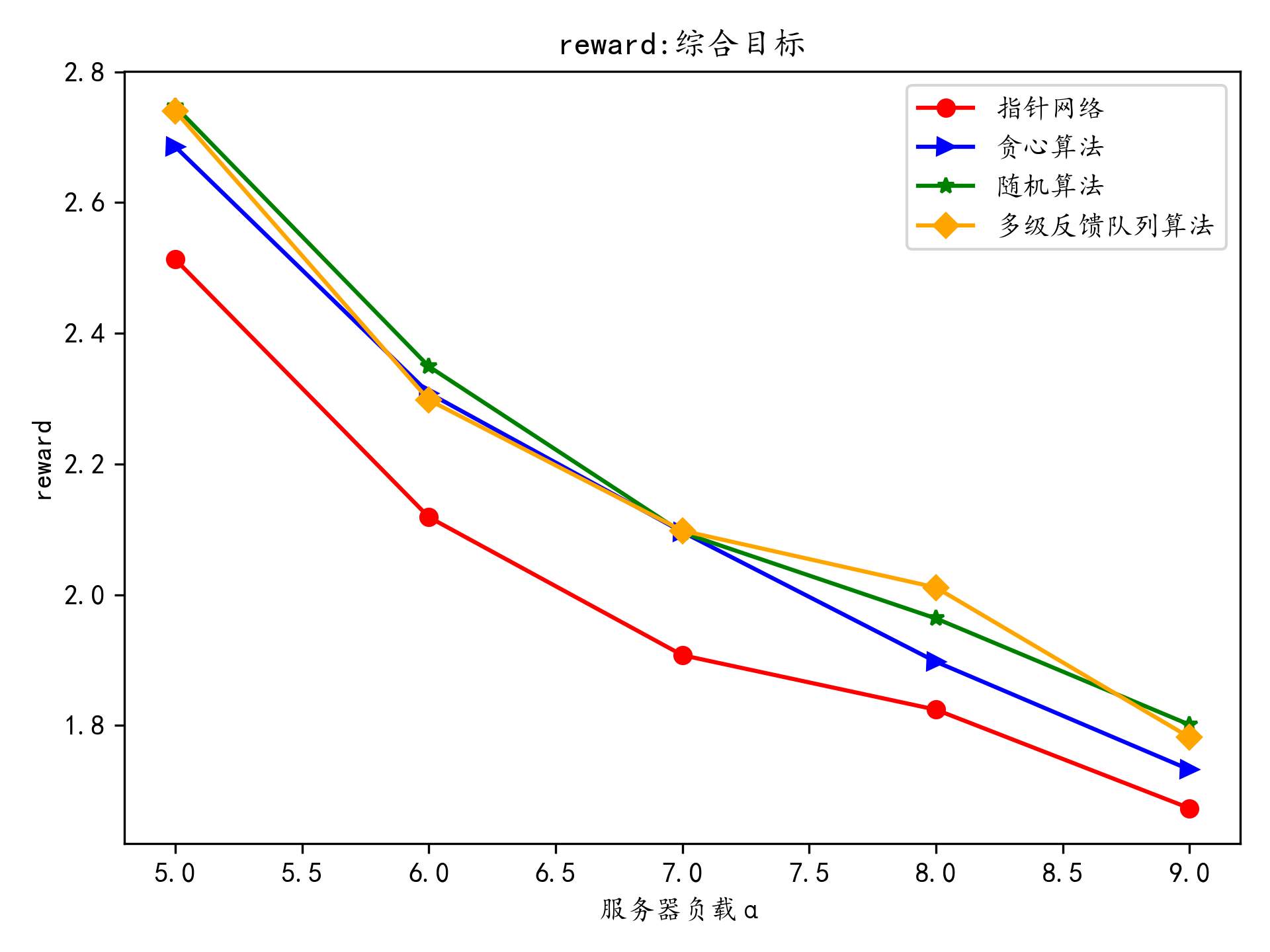
|  |  |
| --- | --- |
| (c) | (d) |

图**4.2** 实验组#2结果

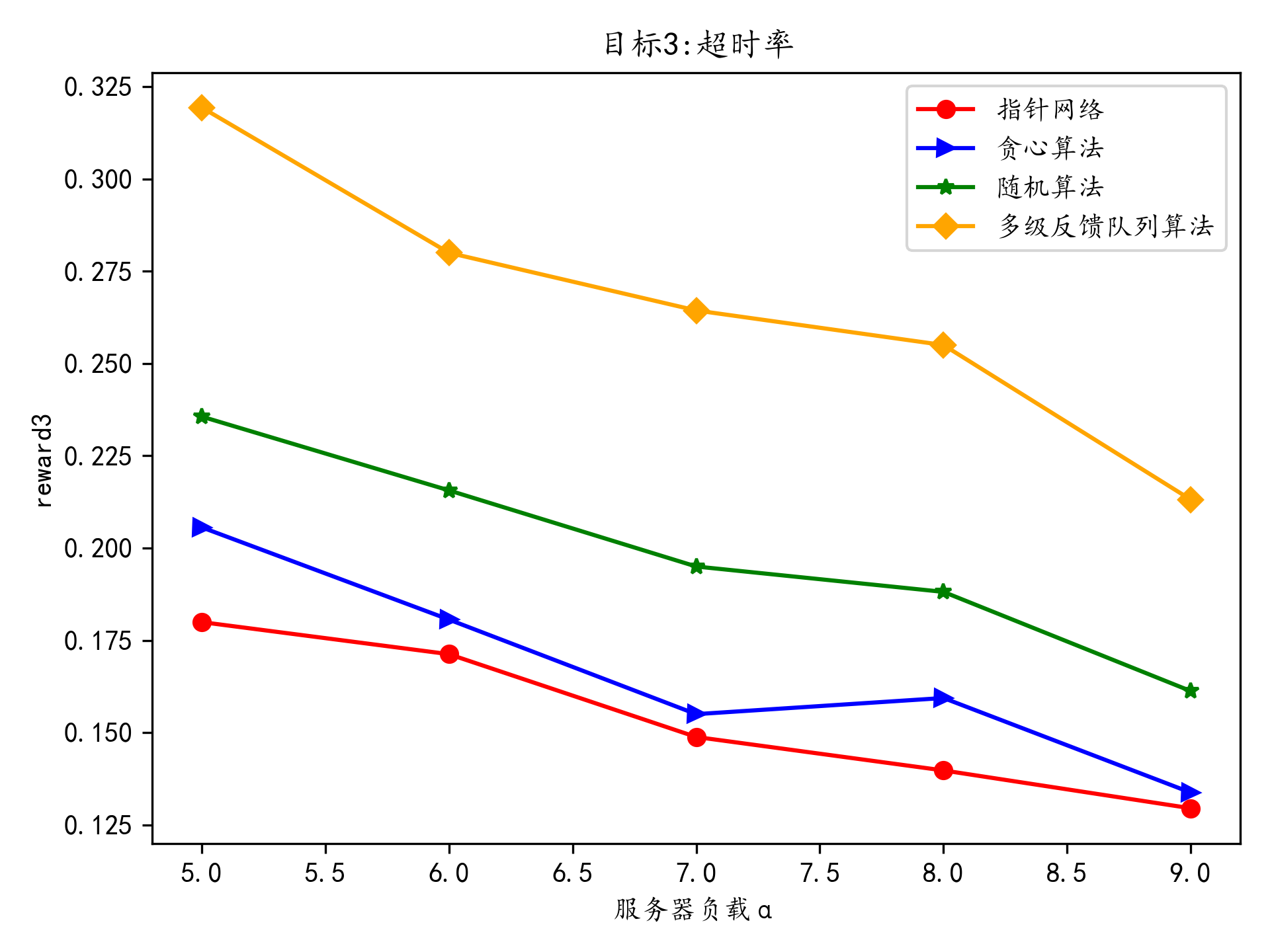
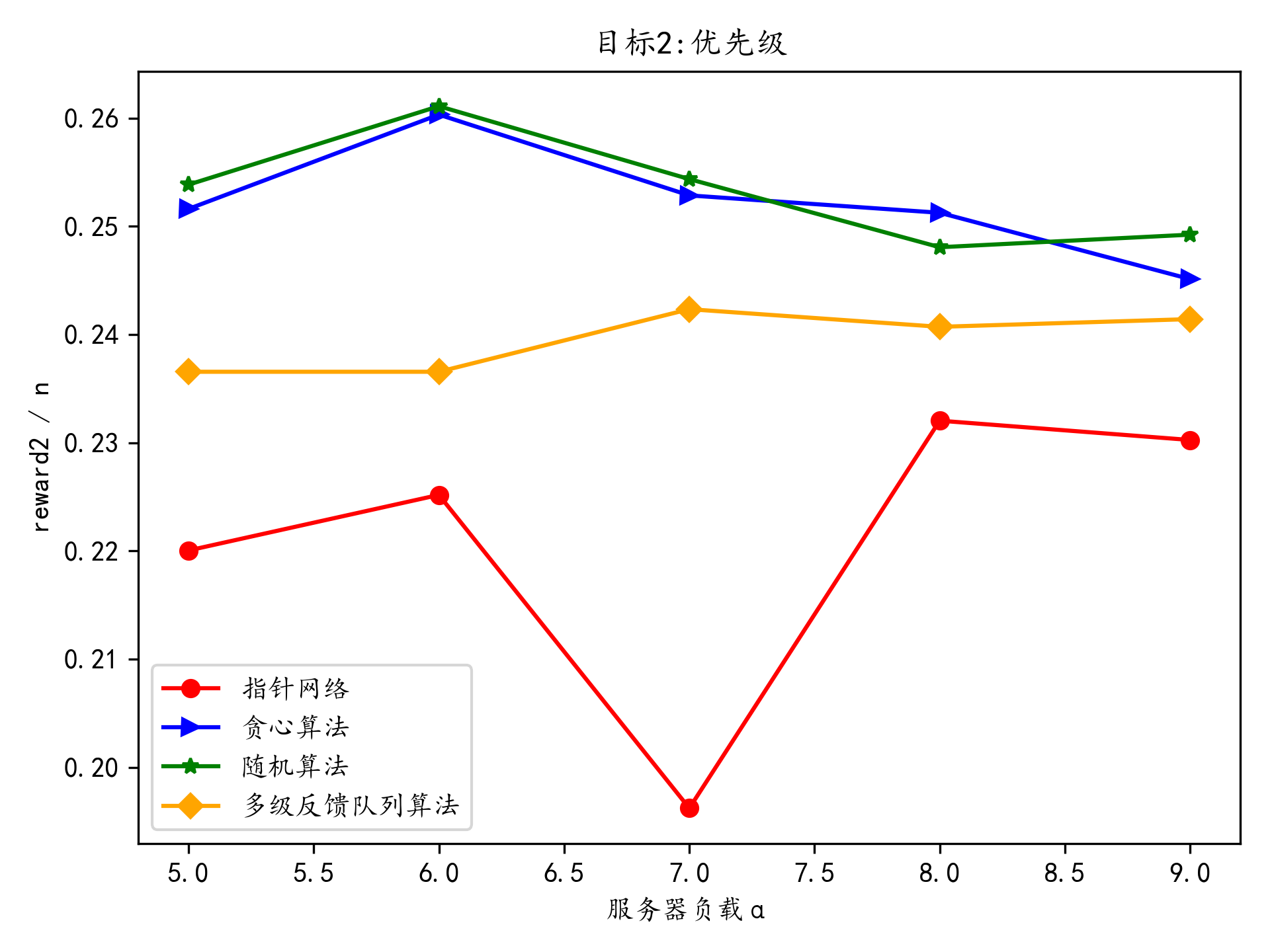
根据实验组#2的实验结果，我们可以看出在服务请求数量与服务器负载系数不变的情况下，改变边缘服务器数量的值得出的结果，我们经由强化学习训练后的指针网络模型在各个优化目标的表现均要优于其他算法。

(3) 实验组#3

对于实验组#3，我们固定服务请求数量为500；边缘服务器数量为5，改变服务器负载系数的值分别为5、6、7、8、9。按不同的优化目标进行分组，将服务器负载系数作为横坐标，不同优化目标的值作为纵坐标，把我们训练好的模型效果与4.2节介绍的其他算法进行对比，得出对比结果如图4.3所示。其中4.3(a)代表综合指标；4.3(b)代表平均运行时间指标；4.3(c)代表优先级指标；4.3(d)代表超时率指标。所有指标均是由公式(3.13)进行归一化后的结果。



|  |  |
| --- | --- |
| (a) | (b) |



|  |  |
| --- | --- |
| (c) | (d) |

图**4.3** 实验组#3结果

根据实验组#3的实验结果，我们可以看出实验组#3的结果主要与运行时间以及超时率相关，当边缘服务器负载增大的时候，运行时间与超时率随之减小。其中在服务器负载的时候，由于数据生成的随机性以及训练的不确定性，使得指针网络在优先级的优化目标下产生了一定震荡。总体而言，在服务请求数量与边缘服务器数量不变的情况下，改变服务器负载系数的值得出的结果，我们经由强化学习训练后的指针网络模型在各个优化目标的表现均要优于其他算法。

综上所述，我们可以得出结论：使用真实数据集产出的数据进行实验，本文所设计的边缘计算环境下基于指针网络和强化学习的服务请求调度策略，在边缘服务器运行时间、服务请求优先级、超时率方面，均要优于随机算法、贪心算法以及多级反馈队列调度算法。

**结论**

随着互联网技术的高速发展，网络终端设备数量急速上升。云计算技术通过将计算复杂度高的任务上传到远程云中运行，解决了终端设备资源与运算能力有限的问题。然而，在物联网等环境下，由于远程云距离较远，通信时延较高，无法满足时延敏感型任务的运行。边缘计算将服务部署在网络边缘环境，降低通信距离，从而大幅度缩短时延，能有效解决该问题。在边缘计算领域下，有大量问题正在被发现并研究。本文将研究重点放在了边缘计算环境下，多服务器与多用户的服务请求调度问题。由于边缘服务器相对于远程云来说，资源仍是有限的，因此，为边缘计算环境设计一套合理的服务请求调度策略，能有效降低成本，提高边缘服务器资源利用效率，提高服务质量。

本文根据边缘计算环境下的真实场景，将传统服务请求特征中的优先级、运行所需时间、超时时间，与服务器资源中的CPU、I/O、宽带、内存占用结合在一起。从服务器与用户角度综合考虑了一个合理的调度策略需要优化的各项指标。当处于网络高峰期时，对边缘服务器的QPS较高，此时服务请求到达服务器后无法立即得到执行，需要在服务器内部进行排队。而我们的调度策略目标就是：如何安排这些排队中的服务请求的执行顺序，从而尽可能优化服务器的运行时间、优先级与超时率这三项指标。

基于此，我们使用指针网络构建出模型，将调度问题抽象成一个Seq2Seq问题，并通过强化学习的方式进行训练。最终使我们的三项优化目标尽可能达到最优。我们使用真是数据集EUA生成数据进行实验，通过控制变量更改不同参数设计实验组，将模型训练结果与随机算法，贪心算法以及多级反馈队列调度算法进行全方面对比。得出的结论是，我们的模型在服务器平均运行时间、任务优先级以及总超时率方面的表现均要优于其他算法。

本文提出了边缘计算环境下的多服务器多用户服务请求调度策略，并验证了该策略的可行性。但仍然存在一些问题需要进一步改进：

(1) 本文提出的调度策略只针对边缘服务器内部进行调度，并未考虑服务器之间的协调及负载均衡。当边缘服务器交叉覆盖时，同时处于多个边缘服务器范围内的用户，其发出的服务请求对于服务器的选择策略暂时为随机选择。在后续的研究中，我们将会同时考虑到用户对于边缘服务器的选择策略，也就是考虑到服务器之间的负载均衡。

(2) 本文提出的调度策略目前是一种静态的调度，边缘服务器在一次调度结束后再取正在排队的服务请求进行下一次调度，缺乏动态性。在后续的研究中，我们将在保留静态调度策略的同时考虑动态情况下的调度策略。

**参考文献**

[1] Ericsson Inc.CEO to Shareholders: 50 Billion Connections 2020[EB/OL]. http://www.ericsson.com/thecompany/press/releases/2010/04/1403231.2010-04-14.

[2] Vaquero L M, Rodero-Merino L, Caceres J, et al.A break in the clouds: towards a cloud definition[J].Acm Sigcomm Computer Communication Review, 2008, 39(1):50-55.

[3] Pan J, Mcelhannon J.Future Edge Cloud and Edge Computing for Internet of Things Applications[J].IEEE Internet of Things Journal，2017：1-1.

[4] Shi W, Jie C, Quan Z, et al.Edge Computing: Vision and Challenges[J]. Internet of Things Journal, IEEE, 2016, 3(5):637-646.

[5] Evans D.The Internet of Things: How the Next Evolution of the Internet is Changing Everything.CISCO White Paper, 2011, 1-11.

[6] Lai P, He Q, Abdelrazek M, et al.Optimal Edge User Allocation in Edge Computing with Variable Sized Vector Bin Packing[J]. 2019.

[7] H Zhao, Deng S, Liu Z, et al.Distributed Redundant Placement for Microservice-based Applications at the Edge. 2019.

[8] Alkhalaileh M, Calheiros R N, Nguyen Q V, et al.Data-intensive application scheduling on Mobile Edge Cloud Computing[J].Journal of Network and Computer Applications, 2020, 167:102735.

[9] Sun J, Yin L, Zou M, et al.Makespan-Minimization Workflow Scheduling for Complex Networks with Social Groups in Edge Computing[J].Journal of Systems Architecture, 2020, 108:101799.

[10] Meng J, Tan H, Li X Y, et al.Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing[J].IEEE Transactions on Parallel and Distributed Systems, 2020, 31(6):1270-1286.

[11] Han Z, Tan H, Li X Y, et al.OnDisc: Online Latency-Sensitive Job Dispatching and Scheduling in Heterogeneous Edge-Clouds[J].IEEE/ACM Transactions on Networking, 2019, PP(99):1-14.

[12] Jie Y, Tang X, Su S, et al.Online task scheduling for edge computing based on repeated Stackelberg game[J].Journal of Parallel and Distributed Computing, 2018, 122:159-172.

[13] 赵星光.边缘计算环境下的并行任务调度算法研究[D].安徽大学,2019.

[14] 林博，张惠民.边缘计算环境下基于动态反馈的Kubernetes调度算法[J].信息技术与信息化，2019，No.235(10)：107-109+114.

[15] 朱宗斌，杜中军.基于改进GA的云计算任务调度算法[J].计算机工程与应用，2013，49(5)：77-80.

[16] Liang T, Yong L, Wei G.A hierarchical edge cloud architecture for mobile computing[C].IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications.IEEE, 2016.

[17] Urgaonkar R, Wang S, He T, et al.Dynamic service migration and workload scheduling in edge-clouds[J].Performance Evaluation, 2015, 91(C):205-228.

[18] Chen H, Wang F, Helian N, et al.User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing[C].National conference on Parallel Computing Technologies (PARCOMPTECH-2013).IEEE, 2013.

[19] Bello I, Pham H, Le Q V, et al.Neural Combinatorial Optimization with Reinforcement Learning.2016.

[20] Lin Y, Shen H.Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Experience[J].IEEE Transactions on Parallel and Distributed Systems, 2017, 28(2):431-445.

[21] Lai P, He Q, Abdelrazek M, et al.Optimal Edge User Allocation in Edge Computing with Variable Sized Vector Bin Packing[J].2019.

[22] Sutskever I, Vinyals O, Le Q V.Sequence to Sequence Learning with Neural Networks[J].Advances in neural information processing systems, 2014.

[23] Vinyals O, Fortunato M, Jaitly N.Pointer Networks[J].Computer Science, 2015, 28.

[24] Bahdanau D, Cho K, Bengio Y.Neural Machine Translation by Jointly Learning to Align and Translate[J].Computer Science, 2014.

[25] Bello I, Pham H, Le Q V, et al.Neural Combinatorial Optimization with Reinforcement Learning[J].2016.

[26] Williams R J.Simple statistical gradient-following algorithms for connectionist reinforcement learning[J].Machine Learning, 1992, 8(3-4):229-256.

[27] 康康.基于强化指针网络的TSP问题的求解与优化[D].华中科技大学, 2019.

[28] X Xia, Chen F, He Q,et al.Online Collaborative Data Caching in Edge Computing[J].IEEE Transactions on Parallel and Distributed Systems, 2020, PP(99):1-1.

[29] 康桂花.计算概论[M].中国铁道出版社,2016.

[30] 彭民德, 彭浩.计算机操作系统[M].清华大学出版社,2014.

[31] 谢涛, 陈火旺, 康立山.多目标优化的演化算法[J].计算机学报, 2003, 26(008):997-1003.

[32] 唐瑞.基于Kubernetes的容器云平台资源调度策略研究[D].电子科技大学, 2017.

[33] 钟守楠, 钟良, 蔡晓芬.基于神经网络的多目标演化优化方法[J].数学杂志(4):453-458.

[34] 陈全, 邓倩妮.云计算及其关键技术[J].计算机应用, 2009(09):254-259.

[35] See A, Liu P J, Manning C D.Get To The Point: Summarization with Pointer-Generator Networks[J].2017:1073-1083.

[36] Gu J, Lu Z, Li H, et al.Incorporating Copying Mechanism in Sequence-to-Sequence Learning[J].2016:1631-1640.

[37] Sun F, Jiang P, Sun H, et al.Multi-Source Pointer Network for Product Title Summarization[J].2018.

[38] Sutskever I, Vinyals O, Le Q V.Sequence to Sequence Learning with Neural Networks[J].2014, 4:3104-3112.

[39] Graves A, Wayne G, Danihelka I.Neural Turing Machines[J].Computer Science, 2014.

[40] Hochreiter S, Schmidhuber J.Long Short-Term Memory[J].Neural Computation, 1997, 9(8):1735-1780.

[41] Bi S, Huang L, Zhang Y.Joint Optimization of Service Caching Placement and Computation Offloading in Mobile Edge Computing Systems[J]. IEEE Transactions on Wireless Communications, 2020, PP(99):1-1.

**致谢**

时光荏苒，转眼间四年本科生活即将画上句号。在武汉大学的这四年，无论是在学习还是科研方面，我都得到了许多优秀的老师、学长、学姐的指导与帮助。在即将毕业之际，我想借此机会向所有在本科包括毕业论文撰写期间给予我支持、帮助与鼓励的人致以最诚挚的谢意。

首先，感谢李兵老师给予我继续攻读他的研究生的机会，李兵老师的实验团队氛围融洽，积极热情，充满朝气。老师对于毕业论文方向的把控，使得我的论文最终得以充分完善，老师严谨治学的态度值得我终身学习。

其次，感谢在我撰写毕业论文期间给予我充分指导的王健老师，王健老师学识渊博，为人亲和，善于引导。能及时发现我的论文中不足之处并提出合理的改进意见，使我的论文更加合理，完整。

再次，感谢从开题一直到定稿阶段持续给予我帮助、鼓励与关怀的夏唯学姐和赵玉琦学长。从开题阶段的场景引入，实验时的方法介绍，到论文撰写时的建议与帮助。让我得以从对边缘计算场景的一无所知到大胆创新，加以思考，最终完成我的毕业论文。

毕业在即，虽然还要在武汉大学继续我的两年研究生生涯，但在我心中已是结束了人生中的一个重要阶段。感谢所有关心我的亲人，老师，朋友，我会铭记你们的教诲，带着你们的祝福，在未来的人生路上继往开来，不懈努力！