基于动态模式学习与资源感知的时变工作负载调度研究

重庆大学硕士学位论文

（学术学位）

学生姓名：

指导教师：

学科门类：工　学

学科名称：软件工程

研究方向：资源管理与优化

答辩委员会主席：

授位时间：

Research on Time-Varying Workload Scheduling Based on Dynamic Pattern Learning and Resource Awareness

A thesis submitted to Chongqing University

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Software Engineering

by

Supervisor:

June, 2025

摘 要

随着云计算技术的快速发展和大规模服务集群的广泛应用，云平台上运行的工作负载规模不断扩大，涵盖了来自不同业务场景的大量任务。在云计算环境下，时变工作负载调度的核心目标是在不断变化的任务需求与异构集群资源之间实现高效分配，确保资源利用与服务质量的最佳平衡。高效的调度不仅能够优化资源利用、降低系统开销，还能提升云平台的服务质量和经济效益，这对于云服务提供商和用户而言都具有重要的现实意义。当前，云计算环境下时变工作负载调度面临的关键挑战主要包括：

①挑战一：在时变工作负载调度场景中，精准预测未来的资源需求是提升调度效果的关键。而工作负载序列模式呈现出显著的时空动态特征，任务执行时长和资源需求跨度大，如何构建能够灵活应对不同类型负载的预测方法成为一大挑战。同时，预测模型必须在计算效率与预测精度之间平衡，避免预测消耗过多算力影响实时调度。

②挑战二：在云计算环境中，现有调度方法存在双向资源感知机制的显著局限性，未能有效结合集群资源特性与工作负载时序特征，缺乏跨时间步长的资源预测能力，导致调度优化不足。另外，在动态在线调度场景中，调度决策要求在毫秒级时间窗口内生成，过度优化调度可能导致适应性下降，因此调度策略必须具备动态调节机制以提升泛化能力和响应能力。

针对上述挑战，本文设计了一个整合多种深度神经网络、优先级队列及深度强化学习的协同调度框架，通过时变工作负载序列预测驱动调度，旨在有效解决云计算环境下时变工作负载的调度难题。具体工作如下：

①研究内容一：基于动态模式学习与专家协同的时变工作负载预测。为了捕捉不同类型工作负载的时变分布特征，本研究利用流式模型的可逆神经网络，动态学习工作负载序列的时空演化模式，并基于学习到的序列模式构建距离度量空间，从而生成专家选择权重；为减少预测的时间开销，本研究采用混合专家模型，根据生成的专家选择权重动态激活最优专家子集，降低模型推理时的计算复杂度；为了提升对时变工作负载的建模与预测能力，本研究构建了融合时频域分析的专家单元，每个专家单元由快速傅里叶变换滤波器与长短期记忆网络组成，并对应单个类型的工作负载，各专家输出通过加权聚合形成序列预测。

②研究内容二：基于优先级调控与资源感知的时变工作负载调度。为了提高集群资源分配的匹配度，本研究设计了一个优先级队列，它利用跨各种工作负载类的时间资源利用模式和服务器集群的当前状态来对需调度的工作负载进行排序；为了加强对集群内资源分布的理解，并实现对集群状态未来变化的更精确的预测，本研究利用资

源感知图注意力层来提取服务器的面向维度和面向时间的关系，动态地捕获集群的维度特征和时间动态；为了增强调度器适应集群内不同工作负载到达模式的能力，同时协调跨服务器的工作负载分配，本研究提出了一种基于策略梯度的算法。通过与动态集群环境的持续交互，调度器探索各种决策，并利用奖励信号优化策略。

关键词：云计算；资源管理；工作负载预测；工作负载调度；强化学习

Abstract

With the rapid advancement of cloud computing technology and the widespread adoption of large-scale service clusters, the scale of workloads running on cloud platforms continues to expand, encompassing a vast number of tasks from diverse business scenarios. In a cloud computing environment, the core objective of time-varying workload scheduling is to efficiently allocate resources in response to dynamic task demands while managing a heterogeneous cluster. The goal is to strike an optimal balance between resource utilization and service quality. Efficient scheduling not only enhances resource efficiency and reduces system overhead but also improves the overall service quality and economic benefits of cloud platforms—an aspect of critical importance to both cloud service providers and users. However, time-varying workload scheduling in cloud environments presents several key challenges, including:

①In time-varying workload scheduling, accurately predicting future resource demands is key to improving scheduling efficiency. However, workload sequence patterns exhibit significant spatiotemporal dynamics, with substantial variations in task execution time and resource requirements. Developing a prediction method that can adapt flexibly to different types of workloads remains a major challenge. Additionally, the prediction model must strike a balance between computational efficiency and prediction accuracy, ensuring that the forecasting process does not consume excessive computing power and compromise real-time scheduling performance.

②In a cloud computing environment, existing scheduling methods exhibit significant limitations in bidirectional resource awareness mechanisms, failing to effectively integrate cluster resource characteristics with workload temporal features. They also lack the ability to predict resource demands across multiple time steps, resulting in suboptimal scheduling performance. Moreover, in dynamic online scheduling scenarios, scheduling decisions must be made within millisecond-level time windows. Excessive optimization may reduce adaptability, making it crucial for scheduling strategies to incorporate dynamic adjustment mechanisms to enhance generalization and responsiveness.

To address these challenges, this thesis proposes a collaborative scheduling framework that integrates multiple deep neural networks, priority queues, and deep reinforcement learning. By leveraging time-varying workload sequence prediction to drive scheduling decisions, the

framework aims to effectively tackle the challenges of time-varying workload scheduling in cloud computing environments. The main contributions of this thesis are summarized as follows:

①**Time-varying workload prediction based on dynamic pattern learning and expert collaboration.** To capture the time-varying distribution characteristics of different types of workloads, this thesis utilizes reversible neural networks within a flow-based model to dynamically learn the spatiotemporal evolution patterns of workload sequences. Based on the learned patterns, a distance metric space is constructed to generate expert selection weights. To reduce the time overhead of prediction, a mixture of experts model is employed to dynamically activate an optimal subset of experts according to the generated weights, thereby reducing computational complexity during inference. To enhance the modeling and prediction capabilities for time-varying workloads, expert units incorporating time-frequency domain analysis are designed. Each expert unit, composed of a fast Fourier transform filter and a long short-term memory network, specializes in a specific type of workload. The final sequence prediction is produced by the weighted aggregation of the expert outputs.

② **Time-varying workload scheduling based on priority control and resource awareness.** To improve the matching degree of cluster resource allocation, this thesis designs a priority queue that ranks workloads awaiting scheduling by leveraging temporal resource utilization patterns across different workload classes and the current state of the server cluster. To enhance the understanding of resource distribution within the cluster and achieve more accurate predictions of future cluster state changes, a resource awareness graph attention layer is employed to extract both dimension-oriented and time-oriented relationships among servers, dynamically capturing the cluster's dimensional characteristics and temporal dynamics. Furthermore, to strengthen the scheduler's adaptability to varying workload arrival patterns within the cluster and coordinate workload distribution across servers, this thesis proposes a policy gradient-based algorithm. By continuously interacting with the dynamic cluster environment, the scheduler explores various decision-making strategies and optimizes its policy through reward signals.

Keywords: Cloud computing; Resource management; Workload prediction; Workload scheduling; Reinforcement learning

目 录

1 绪论

1.1 研究背景与意义

云计算[1]作为数字时代的核心计算范式，已经成为全球数字化转型的重要基础设施服务。它依托虚拟化技术和分布式计算架构，能够提供弹性、可扩展且按需分配的计算、存储和网络资源。这种相较传统计算模式的优势，体现在资源的动态调度和灵活计费机制上，极大提升了服务的效率和灵活性[2]。随着越来越多企业将核心业务迁移至云平台，云计算的应用领域不断扩展，涵盖了互联网搜索、电子商务、社交媒体、智能分析、人工智能（Artificial Intelligence，AI）训练等多个行业。主流云服务平台如 Amazon EC2 和 Google Compute Engine，已经承载着海量的应用实例，处理着极其庞大的任务负载。为了满足日益增长的服务需求，云服务供应商（Cloud Service Providers, CSPs）不断优化其数据中心集群，实施多租户资源共享策略，进一步实现规模效益。

在云计算环境下，时变工作负载调度[3]成为一个关键的研究方向。时变工作负载指的是由于用户需求、业务高峰或突发事件等因素，导致负载在时间维度上发生波动。这些波动通常表现为对中央处理器（Central Processing Unit，CPU）、内存、存储、网络等资源需求的变化，在电商大促、社交平台热点事件等场景下尤为明显。如图 1.1 所示，工作负载 A 和 B 对 CPU 的资源需求随时间步波动，并在特定时刻出现峰值和下降，体现了其时变特性和资源需求的差异。随着微服务架构和容器化技术的广泛应用，云平台面临着秒级波动的业务请求，这种负载的动态变化使得资源调度变得更加复杂。

时变工作负载调度不仅具备重要的理论研究价值，也具有广泛的实践意义。从理论角度来看，如何准确预测和调度波动的多维资源需求、如何应对负载变化，考验着调度算法的智能性与适应性。从实践角度看，高效的调度方法能够有效提升资源利用率，降低空闲率，优化整体运行效率，进而提升用户体验和服务质量，为 CSPs 创造可观收益。因此，云平台亟需一种高效、动态、精准的调度方法，以保证资源合理分配，最大化资源使用效率，同时提升 CSPs 的竞争力。

然而，时变工作负载调度面临着多重挑战。首先，时变负载往往表现为多维资源需求的强耦合性，例如 AI 训练任务中图形处理器（Graphics Processing Unit，GPU）显存和计算单元之间的协同需求，使得调度变得极为复杂。其次，负载的到达速率和波动性通常难以预测[4]，尤其在社交平台热点事件或电商促销期间，负载激增会让传统静态调度策略无从应对。再者，云服务器的异构性质以及可用资源的分散分配，使得资源调度和管理变得更加复杂，可能导致资源使用过度或不足的现象，影响云平台的稳定性和服务质量。最后，在满足服务级别协议的前提下，如何平衡多个用户的资源需求，确保公平性和高效性，也是一项极具挑战的任务。

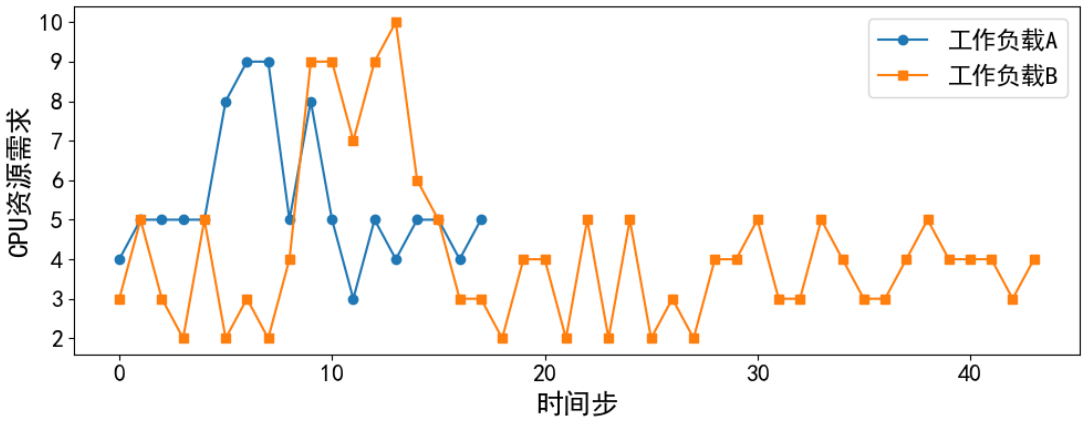

图 1.1 工作负载对 CPU 的资源需求变化

Fig. 1.1 The variation in CPU resource demand of workloads

为了应对这些挑战，本文提出了一种基于动态模式学习与资源感知的时变工作负载调度方法。该方法首先对时变工作负载的时空演化模式进行建模，并通过专家协同预测，提供负载变化趋势等关键信息，以辅助在线调度。随后，在调度过程中，方法能够实时感知集群资源状态，动态调整工作负载的调度优先级，并结合在线强化学习，自适应地做出调度决策。这种智能调度模型不仅能够有效应对时变工作负载的复杂性，且能在跨时间维度上优化资源利用，同时保障服务质量，为云平台提供了更平衡、灵活的时变工作负载调度解决方案。

1.2 国内外研究现状

1.2.1 时变工作负载预测

在时变工作负载预测领域，当前研究主要围绕工作负载的时间序列建模展开，通过深度解析其动态演化过程构建具备时序依赖建模能力的预测框架。相关工作按方法可以分为基于统计模型的预测和基于深度学习的预测。

（1）基于统计模型的预测

基于统计模型的方法在时变工作负载预测中，主要通过对历史数据进行深入分析，挖掘数据中的趋势、季节性和随机波动，从而实现对未来工作负载的预测。这些方法通常假设时间序列数据具有一定的规律性，通过数学模型来捕捉数据的结构和波动。

其中，Arima 模型[5]是一种经典的时间序列预测方法，通过结合自回归、差分和移动平均来描述时间序列的趋势与季节性。该方法能够有效地去除非平稳性并提取数据中的内在规律，适用于单一变量的预测。另一个常见的方法是 ES[6]，它通过对近期数据赋予更高的权重，来捕捉数据的趋势和季节性变化。ES 的种类有单指数、双指数和平滑霍尔特-温特斯方法，适用于不同的数据模式，特别适合于短期的工作负载预测。

为了结合这两种方法的优点，Arima-ES 混合模型[7]应运而生，结合 Arima 模型的长期趋势预测和 ES 的短期波动调整，使得预测更加准确，特别适用于具有复杂模式的数据。此外，GARCH 模型[8]则更适用于波动性较大的数据，能够捕捉并预测数据中的波动性变化。该模型通过建模时间序列中的条件方差特性，尤其适用于金融、网络流量等波动性较强的场景。而 VAR 模型[9]则专门用于多变量时间序列的预测。它能够考虑多个变量之间的相互影响，适用于同时预测多个相关工作负载指标，尤其是在多因素共同作用下，可以提高预测的精度和可靠性。

这些基于统计模型的方法在实际应用中存在几个问题。首先，它们通常假设数据是平稳的或遵循特定的趋势和季节性模式，但实际工作负载可能存在复杂的非线性波动，导致模型难以适应。其次，这些方法对历史数据的依赖较强，数据不完整或异常值会显著降低预测精度。最后，过拟合是常见问题，尤其在数据噪声较大时，模型可能过度拟合历史数据，失去泛化能力。

（2）基于深度学习的预测

为了解决统计模型中存在的问题，基于深度学习的预测方法应运而生。它通过神经网络模型能够自动提取数据中的复杂特征，不依赖于传统的假设，如数据平稳性，能够适应非线性和高度波动的工作负载变化。深度学习方法还能够通过大规模数据集进行训练，有效处理缺失数据和异常值，减少传统统计模型中的过拟合问题。

Wang 等人[10]提出了一个多尺度表示增强的时序流融合模型，采用自监督多尺度表征学习捕获长期历史周期特征，通过时序流融合模块建模近端动态趋势，利用注意力机制融合多尺度特征并引入标准化流处理非高斯分布，实现长周期负载精准预测。Karimunnisa 等人[11]提出了一种基于深度信念网络与自适应狮群优化的预测方法，通过动态调整网络隐层结构捕获复杂时序特征，结合改进资源供给策略解决过载、欠载问题，在保障服务质量与协议约束下实现云资源的精准预测。Rossi 等人[12]提出了一种基于贝叶斯深度学习与迁移学习的预测方法，通过量化预测不确定性及多资源联合建模捕捉复杂时序关联，结合跨场景迁移策略适应新数据中心的异构负载特征。Kim 等人[13]提出一种基于多变量时间序列与集成学习的预测方法，通过异常检测机制清洗训练数据提升模型鲁棒性，融合 Transformer 与线性时序模型的互补优势构建动态加权集成框架，结合真实制造系统虚拟机器数据验证，在复杂负载波动场景下实现高精度长周期预测。Li 等人[14]提出了 EvoGWP 模型，通过深度图进化学习预测云计算负载的长期演化特征，设计双层级形状基元提取机制显式识别细粒度资源使用模式，构建动态演化图网络建模多负载间时空维度干扰效应，结合图卷积编码-解码框架融合跨负载模式关联。Maiyza 等人[15]提出 VTGAN 模型，通过混合生成对抗网络联合预测云计算负载数值与变化趋势，融合技术指标与傅里叶和小波变换增强时序特征表征能力，设计多步滑动窗口预测机制优化输入特征维度。Gupta 等人[16]提出基于多控制托佛利门驱动的自

适应量子神经网络模型，通过量子位编码动态负载数据特征，在量子神经网络隐层及输出层引入多控制目标门调控量子态演化路径，结合均匀自适应量子学习算法优化网络参数，利用量子叠加态并行计算优势捕捉复杂负载波动模式。Kaim 等人[17]提出了一种集成卷积神经网络与注意力增强双向长短期记忆网络的深度学习方法，通过多尺度特征融合捕捉云负载时序关联性。Dogani 等人[18]提出了一种基于卷积神经网络与注意力门控循环单元的混合预测方法，通过多变量时间序列建模提取负载的时空关联特征，有效解决传统变量单步预测的局限性。Yuan 等人[19]提出了基于变分模态分解与双向门控循环单元的混合预测框架，通过分解非平稳时序噪声并建模资源的前后依赖关系实现预测。Zhao 等人[20]提出了一种融合时频分析与通道独立策略的深度学习预测方法，通过短时傅里叶变换提取频域周期特征，结合多头注意力机制捕捉负载时序关联。Bi 等人[21]设计了“分解-动态滤波-多维度融合”三级架构，结合自适应滤波优化高频噪声抑制，并引入双向长短期记忆网络（Long Short-Term Memory, LSTM）、网格 LSTM 与多头注意力机制，显式建模多资源的时空耦合关联。Zheng 等人[22]提出 WorkloadDiff 模型，首次将去噪扩散概率模型引入云计算负载预测，通过双路径网络融合原始、含噪信号的多尺度时序特征，结合重采样策略提升条件一致性，在数据集上实现概率化预测。Zhao 等人[23]提出了 MSCNet 多尺度卷积网络，通过多尺度建模模块分阶段提取云计算负载的多周期特征：首先利用多尺度分块策略将原始时序拆解为不同时间粒度的子序列，通过 Transformer 编码器捕捉跨尺度的全局依赖关系；继而采用多尺度卷积模块融合局部时序模式。Zuo 等人[24]提出了一种融合混合数据增强与对比迁移学习的小样本负载预测方法，通过混淆生成分布相似样本构建正负样本对，利用对比学习对齐源域与目标域的表征关系，显著缓解小样本训练不足导致的过拟合问题。

尽管现有方法在预测领域取得了一定突破，但仍存在一些不足：一方面，现有模型对不同类的工作负载特征缺乏针对性建模能力；另一方面，复杂的网络架构导致较高的计算开销，难以满足预测场景下的实时性需求。

1.2.2 时变工作负载调度

在时变工作负载调度领域，当前研究主要聚焦于不同调度场景下的集群资源管理，并根据云计算集群中的可用资源是由单片服务器管理还是由分布式服务器管理，将这些研究分为集中式资源调度和分布式资源调度。

（1）集中式资源调度

集中式资源调度是一种经典的云计算资源管理范式，其核心思想是将整个集群抽象为一台虚拟的集成服务器，忽略服务器间的通信开销和资源异构性，并将集群资源视为统一的整体进行分配。在这种模型下，调度策略主要从全局工作负载队列中选择任务并将其分配到集成服务器中执行。由于其部署简单且易于实现，集中式资源调度

成为云计算环境中最早出现且研究最为广泛的调度模式，相关算法研究已积累了丰富的理论成果和实践经验[25]。

集中式资源调度中，经典的启发式算法如 min-min 算法[27]的基本思想是先将小的工作负载分配到服务器上执行，实现简单，但是容易出现饥饿现象与负载不均衡问题。max-min 算法[28]改进 min-min 算法，保证长期工作负载的执行，但该算法面临部分资源过度利用和部分未充分利用，无法改善参数等问题。先来先服务算法[29]按照用户的工作负载请求到达的先后顺序执行，导致短期工作负载等待时间长。因此，Devi 等人[30]基于增强加权循环进行资源分配，改善了服务质量（Quality of Service, QoS），但频繁的工作负载切换导致额外的开销。元启发式算法通过融合启发式规则与随机化策略，为解决复杂动态的工作负载调度问题提供了全局优化框架。在工作负载调度场景中，蚁群优化算法[31]（Ant Colony Optimization, ACO）通过模拟蚂蚁群体觅食行为，构建工作负载到资源的动态分配路径。针对云计算中突发性工作负载引发的瞬时峰值调度问题，Duan 等人[32]提出基于ACO的Pre-Ant Policy算法，其集成负载预测模型优化工作负载分配序列，显著提升了调度成功率和资源利用率等核心指标。

随着云计算环境中工作负载的动态性与异构性持续加剧，启发式与元启发式算法因其依赖固定规则与随机搜索的局限性，在复杂调度场景下面临解质量与实时性难以兼顾的挑战。深度学习凭借其多层神经网络结构能够逐层学习和提取复杂模式，在工作负载调度任务中展现出了显著优势。Mao 等人[33]首次将强化学习应用于工作负载调度，提出的方法 DeepRM 将服务器集群和工作负载状态表示为图像，奖励设计为平均作业放缓。Fan 等人[34]提出DRAS 采用滑动窗口和两级网络方法，由滑动窗口划分优先级，一级网络进行调度操作，二级网络进行小型工作负载的回填。Zhang 等人[35]提出了 SchedInspector，对启发式方法提出的调度决策进行检查，决定是否拒绝这个决策。Xie 等人[36]提出了一种基于转发和计算双优先级机制的延迟优先可靠调度策略，并引入李雅普诺夫优化，将约束转化为瞬时优化，实现计算和网络资源的长期平衡负载。Zhu 等人[37]开发了新的基于图神经网络的工作负载嵌入表示，并设计了一个基于近端策略优化的在线学习调度程序。Mangalampalli等人[38]使用了Cloudsim 工具包中的HPC2N 和 NASA 数据集来模拟工作负载，并采用深度Q网络来为工作负载搜索最佳调度序列。

在集中式资源调度研究中，传统的调度模型通常基于简化的假设，例如同构服务器和静态资源环境，然而这些假设已无法适应现代分布式系统的复杂性和动态性。随着云计算的快速发展，CSPs往往需要在大范围内调度异构服务器资源，这些服务器可能随时动态加入或退出系统。集群中的服务器本质上是资源异构且独立的，当多个服务器协同处理单个工作负载时，必须综合考虑资源分配、传输成本和延迟等关键因素。因此，集中式调度模型因其过度简化的假设和局限性，已难以满足现代分布式系统的需求，亟需更普适和高效的调度方法。

（2）分布式资源调度

分布式资源调度是一种灵活的云计算资源管理范式，其核心思想是将集群中的每个服务器抽象为独立的调度决策主体，突破集中式架构的资源聚合视角，采用离散化资源管理模式进行动态分配。在这种模型下，调度策略通过实时分析节点资源状态，从全局工作负载队列中提取任务，并基于多维评估机制将其定向部署至最优服务器执行。由于具备细粒度控制能力和横向扩展优势，分布式资源调度逐渐发展为云计算环境中处理时变工作负载的主流调度模式，相关算法研究在资源利用率优化和跨节点协同调度领域已形成系统化的方法论与实践体系[39]。

在调度器可知工作负载完整序列的情况下，Berkey 等人[41]提出了最佳拟合算法，使得完成工作负载调度同时减少资源浪费。Mondal 等人[42]提出了最短作业优先（Shortest Job First, SJF）调度算法，但节点负载不均匀，引发饥饿现象。Cheng 等人[43]以降低虚拟机执行成本和保证用户预期响应时间的最大实现为目标，引入了一个成本感知优化模型，选择具有最短剩余响应时间的工作负载替换掉当前正在服务器上运行的工作负载。Xiu 等人[44]指出现有的强化学习调度方法难以在不同的环境之间迁移，因而采用了元强化学习，其由两个学习的“循环”组成，“外循环”利用许多环境的经验来逐步调整元策略的参数。“内循环”通过少量的梯度更新来快速适应特定的任务。Wei 等人[45]定义多维资源平衡指数（Multi-dimensional Resource Balance Index, MRBI）来量化多维资源平衡利用的等级，并根据 MRBI 设计相应的强化学习算法进行调度。Fan 等人[46]设计了一种分布式的模型，将调度的过程分为两步：用户设备产生工作负载传输给调度器的策略网络，调度器将中间结果回传给用户设备上的优势值网络从而产生调度决策。Tuli 等人[47]结合了异常检测与工作负载调度，提出了一个基于生成对抗网络的架构。该架构首先检测工作负载调度后是否会发生异常并定位异常的资源维度，随后由生成器模块生成改进的调度策略，最后由判别器模块判定采取原调度策略还是改进调度策略。Zeng 等人[48]提出了一种负载均衡多群落粒子群优化工作负载调度方法，它通过改进的适应度函数对工作负载的最大完工时间和各服务器完工时间方差进行组合优化以提升集群的负载均衡性。Li 等人[49]提出了 MAIFS，它采用多智能体注意力机制学习和共享从不同智能体观察到的相互关联的资源状态特征，然后通过动态协调图机制协调智能体在综合调度过程中的交互式工作负载放置决策。Lu 等人[50]提出了基于演员评论家算法的实时工作负载调度技术，可根据工作负载要求进行各种超参数调整，还通过引入负载均衡因子来评估模型的负载均衡能力。

上述方法在调度器的设计中，工作负载的完整信息（包括资源需求和预计运行时间等）是关键的输入参数，并通常从配置文件中获取。但在实际场景中，权限受限或配置文件缺失可能导致只能获取部分信息，从而影响调度决策的有效性。因此，亟需

研究在无法获取工作负载完整信息的情况下，如何构建一个高效的调度器，使其能够动态适应具有不同特性的工作负载数据输入，并持续生成最优的调度决策。

针对调度器无法获取工作负载完整序列的情况，Yang等人[51]提出了一种将贪婪优化与机器学习相结合的新模型，首先采用多层感知机利用历史数据预测工作负载执行时间，然后使用边际增量效用最大化贪婪算法执行调度。Shi等人[52]提出了一种模型辅助的自适应深度强化学习算法，实现工作负载卸载决策、通信资源和计算资源的联合配置。Ma等人[53]提出了一个名为 JobFusion 的框架，它通过将分层聚类方法与连通性约束相结合来构建高效的资源分区方案，随后利用图卷积神经网络捕获工作负载的关键信息。Liang 等人[54]提出了一种基于LSTM和遗传算法的智能资源调度策略，首先利用LSTM模型预测未来的资源需求，然后利用遗传算法优化资源配额。Staffolani等人[55]构建了基于强化学习的分布式队列，同时考虑了不同类工作负载和服务器集群的异构性。来自不同生产者的工作负载会先被分类，服务器会将自己的资源划分为固定大小的工作者节点。Mondal等人[56]采用等价类映射和状态图表示对时变工作负载进行调度，首先会通过一个分类器将工作负载映射到等价类，然后采用图像表示整体的服务器集群状态，并最终产生调度决策。Islam等人[57]将异构的云虚拟机根据资源容量进行分类，调度器产生的调度决策会被提交给集群管理器，集群管理器会在相应的云虚拟机中为当前工作负载创建一个执行器。Zhao等人[58]分析了深度学习模型训练不同阶段的资源需求，通过交错在不同资源类型上存在瓶颈的多个模型训练任务，以有效地利用资源，减少任务排队时间。Li等人[59]将调度过程分解为任务卸载和资源分配决策，采用指针网络来解码集群特征及其相互关联的语义从而进行任务卸载，资源分配时掩码无效的操作，然后从条件概率的链式规则中推导出分配策略。Chen等人[60]提出了一种基于深度强化学习的资源分配方法，它通过工作负载时间窗口在资源分配过程中同时考虑当前和未来的工作负载，并设计了一种反馈控制机制，通过迭代执行管理操作，构建当前系统状态下的客观资源分配计划。Jin等人[61]提出了一种通过动态画像和工作负载-资源最优匹配的自适应云配额方案，从文本、统计和分形三个维度聚合信息，以建立动态画像。在此基础上，设计了双向混合专家模型来匹配最适合工作负载的资源组合。

尽管在无法获取完整工作负载信息的情况下，上述研究已取得一定进展，但随着分布式计算环境的复杂度提升，工作负载规模与异构性的增长对服务器资源的时空动态性建模提出了更高要求，现有研究在深入挖掘服务器资源的多维度特征（如CPU、内存利用的时间特性）以及工作负载的调度优先级方面仍存在不足。

1.3 本文的研究内容

论文的总体研究框架如图 1.2 所示。通过对国内外研究现状进行分析归纳，云计算场景下时变工作负载的调度仍存在以下挑战：

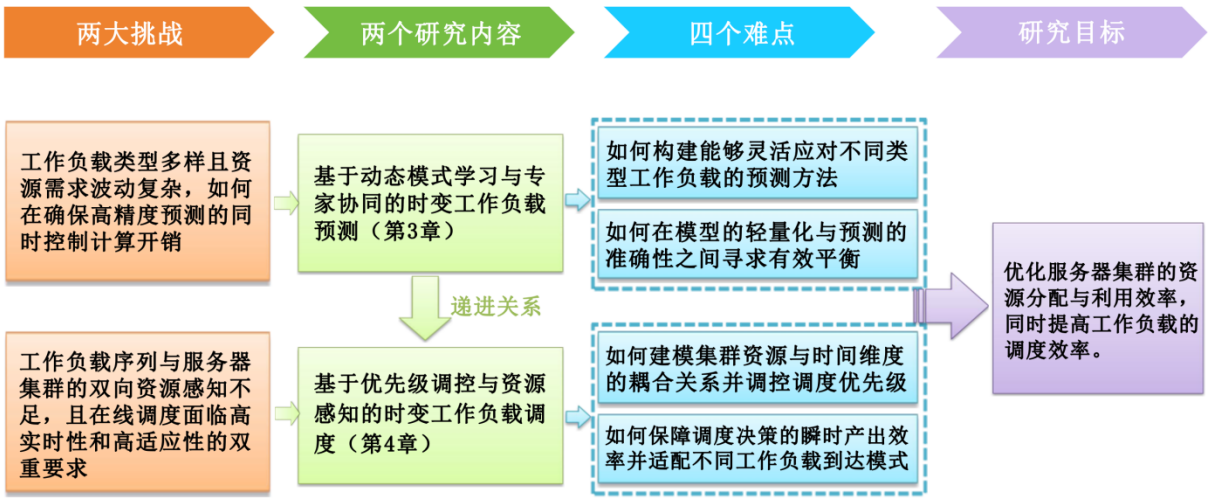

图 1.2 论文的研究背景与研究内容间的关系

Fig. 1.2 Relationship between research background and research contents of the thesis

（1）挑战一：在时变工作负载调度场景中，精准预测未来的资源需求是提升调度效果的关键。调度的质量依赖于对未来负载的准确判断，预测越精确，调度时可利用的信息就越充分，从而实现更优的资源分配。然而，工作负载的类型并非显式给定，而是需要通过学习从历史数据中提取特征，以掌握其时空维度上的变化模式及不同资源需求之间的动态关联。由于负载在计算、内存等多维资源上的需求存在波动和非线性耦合，且任务执行时长跨度极大（从秒级实时任务到月级批量作业），如何构建能够灵活应对不同类型负载的预测方法成为一大挑战。同时，在实际调度过程中，预测的计算开销不容忽视。如果仅追求高精度的时序拟合而忽略计算效率，可能导致预测过程本身消耗过多算力，削弱调度的实时性。因此，必须在模型的轻量化与预测的准确性之间寻求有效平衡，既能快速响应工作负载的变化，又能提供足够精确的预测支持高效调度。

（2）挑战二：在云计算环境中，针对时变工作负载序列与异构服务器集群的协同调度问题，现有方法未能有效建模集群资源维度与时间维度的耦合关系，并且未充分利用工作负载中隐含的时序关联特征进行调度。这种双重视角缺失导致调度器难以构建跨时间步长的集群资源模型，无法通过序列感知机制动态调控工作负载调度优先级并生成更优的调度决策。与此同时，若调度算法过度追求最优解而忽视适应性，可能导致过拟合的决策机制难以应对突发性负载到达模式变动，进而削弱系统整体的响应能力。因此，必须设计具备动态调节机制的调度策略，在计算效率与模型泛化能力之间建立协同机制，既要保障调度决策的瞬时产出效率，又要维持对不同到达模式工作负载的感知与动态适配能力。

针对上述挑战，本文设计了一个整合多种深度神经网络、优先级队列及深度强化学习（Deep Reinforcement Learning, DRL）的协同调度框架，通过时变工作负载序列预测驱动调度，旨在有效解决云计算环境下时变工作负载的调度难题。具体工作如下：

研究内容一：基于动态模式学习与专家协同的时变工作负载预测。①为了捕捉不同类型工作负载的时变分布特征，本研究利用流式模型的可逆神经网络，动态学习工作负载序列的时空演化模式，并基于学习到的序列模式构建距离度量空间，从而生成专家选择权重。②为减少预测的时间开销，本研究采用混合专家模型（Mixture of Experts, MOE），根据生成的专家选择权重动态激活最优专家子集，降低模型推理时的计算复杂度。③为了提升对时变工作负载的建模与预测能力，本研究构建了融合时频域分析的专家单元，每个专家单元由快速傅里叶变换（Fast Fourier Transform, FFT）滤波器与长短期记忆网络组成，并对应单个类型的工作负载，各专家输出通过加权聚合形成序列预测。

研究内容二：基于优先级调控与资源感知的时变工作负载调度。①为了提高集群资源分配的匹配度，本研究设计了一个优先级队列，它利用跨各种工作负载类的时间资源利用模式和服务器集群的当前状态来对需调度的工作负载进行排序。②为了加强对集群内资源分布的理解，并实现对集群状态未来变化的更精确的预测，本研究利用资源感知图注意力层（Graph Attention Network, GAT）来提取服务器的面向维度和面向时间的关系，动态地捕获集群的维度特征和时间动态。③为了增强调度器适应集群内不同工作负载到达模式的能力，同时协调跨服务器的工作负载分配，本研究提出了一种基于策略梯度的算法，通过与动态集群环境的持续交互，调度器探索各种决策，并利用奖励信号优化策略。

1.4 本文的组织结构

本文的组织结构如下：

第一章：绪论。首先介绍所选课题的研究背景及意义，阐述云计算的特点以及时变工作负载调度的应用；其次介绍时变工作负载预测和调度的研究现状，并总结目前研究的不足；最后介绍本文的研究内容以及全文的组织架构。

第二章：相关理论和技术基础。介绍本文涉及的相关理论知识和技术基础，包括时变工作负载调度概述、信号处理方法和神经网络相关理论知识，为后续构建预测、调度模型奠定了坚实的理论基石与算法基础。

第三章：基于动态模式学习与专家协同的时变工作负载预测方法。本章所提出的方法聚焦于时变工作负载预测。该预测方法由流式模型、混合专家模型、FFT-LSTM三部分组成。首先通过流式模型捕捉工作负载序列的时空演化模式，利用距离度量生成专家选择权重；其次基于混合专家模型，根据实时权重激活最优的专家子集，在维

持预测精度的同时实现计算效率提升；此外，设计融合 FFT 和 LSTM 的专家单元，分别提取序列的周期分量与时域依赖模式，通过动态加权融合多专家输出完成预测。最后，在公开数据集 Google traces 上进行实验，验证本章提出方法的有效性。

第四章：基于优先级调控与资源感知的时变工作负载调度方法。本章所提出的方法聚焦于时变工作负载调度，且第三章的预测是本章调度的基础。调度环境下工作负载以在线的方式动态地到达和离开具有异构服务器的云集群。该调度方法将感知工作负载的优先级队列和感知服务器集群的并行 GAT 层相结合，从而有效地在集群中的服务器之间调度负载。此外，该方法还通过一个基于 DRL 的调度策略来生成在线决策。最后，在公开数据集 Google traces 上进行实验，验证本章提出方法的有效性。

第五章：总结与展望。总结本文工作和研究成果，并提出未来研究的方向。

1.5 本章小结

本章首先介绍了本文的研究背景和意义，随后介绍了云计算环境下时变工作负载预测和调度问题的研究现状，接着通过现有研究的不足及存在的挑战引出本文的研究内容，最后列出了本文的组织结构。

2 相关理论和技术基础

为了对时变工作负载预测与调度的理论和技术进行研究，本文阅读并学习了大量相关文献，包括时变工作负载调度的基本概念，信号处理方法以及神经网络理论。为支撑后文所提模型和实验，本章将对这些技术和理论进行简单介绍。

2.1 时变工作负载调度概述

2.1.1 时变工作负载数据特征

时变工作负载序列与常规静态工作负载存在显著差异，其动态演化特性对云计算系统资源调度提出了更高要求。具体而言，时变工作负载序列具有以下关键特征：

①任务执行时长跨度悬殊：时变工作负载的执行时间呈现数量级差异，既包含毫秒级响应的即时任务，又涉及持续数小时的批处理作业。这种极端的时间跨度对资源调度策略的适应性形成严峻挑战，要求系统具备动态调整时间片分配和优先级调度的能力。例如，实时数据分析任务需抢占计算资源，而离线训练任务则需维持长周期资源稳定性，二者的共存加剧了调度算法的复杂度。

②多维资源需求存在波动性：时变工作负载的 CPU 核数、内存容量、磁盘容量等资源需求通常呈现非平稳变化特征，其中某些特殊负载可能在同一时间窗口内发生显著波动。这种不确定性可能导致传统的静态资源预留机制失效，难以有效满足任务需求。此外，不同工作负载对同一资源的需求在特定时刻可能同时增加，从而引发性能下降。因此，在工作负载调度中，需要实时监控和预测多维资源需求，并动态调整资源分配，以优化整体系统性能。

③非线性耦合特性：时变工作负载序列中的多维资源需求之间可能存在非线性的耦合关系，即某一资源的变化可能对其他资源产生非线性的影响。例如 CPU 利用率的增加可能导致内存需求的非线性增长。这种耦合效应使得基于独立维度分析的资源优化方法失效，需采用多变量协同建模技术揭示隐式关联规律，构建跨维度的资源均衡调度策略。

2.1.2 时变工作负载调度示例

图 2.1 是一个关于时变工作负载调度的简单示例。考虑一个场景在某个特定时间步时，云计算集群中有三个等待调度的工作负载。集群由两台服务器组成，每台服务器都配备了两个维度的资源，CPU和内存。每个工作负载都与一个跨越多个时间步的资源需求序列相关联。服务器中的黑色矩形描述了各服务器中正在运行的工作负载所占用的资源。为了最大限度地利用集群资源，并尽快满足每个工作负载的资源需求，首选的调度时间表如下：

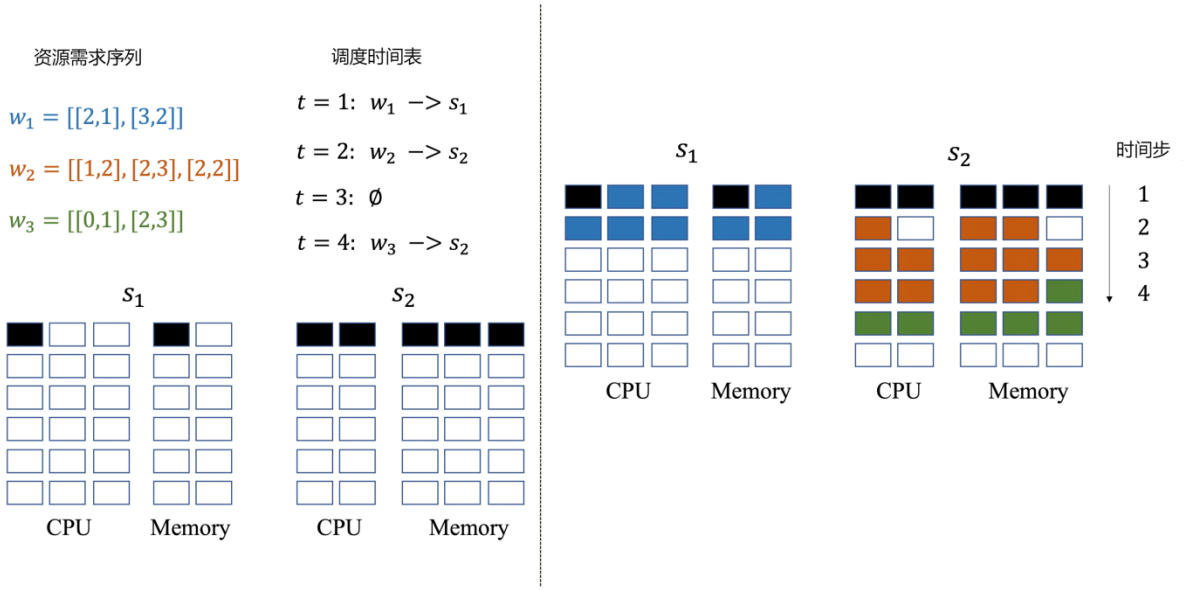

图 2.1 时变工作负载调度示例

Fig. 2.1 An example of the time-varying workload scheduling

①在时间步$t = 1$时，将工作负载w_1调度到服务器s_1中。

②在时间步$t = 2$时，将工作负载w_2调度到服务器s_2中。

③在时间步$t = 3$时，不执行任何调度。虽然此时服务器s_1空闲，但将工作负载w_3调度到服务器s_1不仅会使得当前时间步的服务器空闲资源被浪费，在后续时间步中服务器s_1也无法满足工作负载w_3的资源需求。

④在时间步$t = 4$时，将工作负载w_3调度到服务器s_2中。

虽然这个例子相对简单，但云计算环境下的真实调度场景要复杂得多，通常涉及大量的异构服务器，更多维的资源，以及动态到达集群的具有不同资源需求的工作负载。这些因素大大增加了优化调度过程的复杂性。

2.2 信号处理方法

信号处理方法常用于分析和处理时序数据，旨在提取特征、增强信号，并识别其中的模式或趋势。常用的信号处理方法包括小波变换、希尔伯特变换和快速傅里叶变换等。本文采用的快速傅里叶变换[62]是一种高效的计算离散傅里叶变换（Discrete Fourier Transform, DFT）的算法。傅里叶变换将一个信号从时域转换到频域，从而可以进行周期性分析和模式识别，是信号处理、图像处理、语音分析等多个领域的重要工具。傅里叶变换最早由法国数学家 Jean-Baptiste Joseph Fourier 提出，他认为任何周期性函数都可以表示为一系列正弦波的叠加。对于一个长度为N的时序数据$\{x(0),x(1),\ldots,x(N-1)\}$，DFT将其从时域转换到频域：

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-i2\pi\frac{kn}{N}},\quad k=0,1,2,\ldots,N-1 \tag{2.1}$$

其中，$x(n)$是时域信号，$X(k)$是频域信号，N是信号的长度，$e^{-i2\pi\frac{kn}{N}}$是复数的指数形式，代表旋转因子。DFT 的计算复杂度是$O(N^2)$，因为需要进行N次求和，每次求和涉及N次乘法和加法。

FFT 通过分治的方式将 DFT 分解为多个较小的 DFT，从而降低了计算复杂度。最著名的 FFT 算法是 Cooley-Tukey 算法[63]。它的基本思想是将信号的长度N进行分解，利用递归的方式计算，最终将大规模的计算转化为较小的子问题。具体步骤如下：

（1）**分裂数据**：将输入数据$x(n)$（长度为N）分成两部分：

①**偶数部分**：$x(0),x(2),x(4),\ldots$

②**奇数部分**：$x(1),x(3),x(5),\ldots$

（2）**递归计算**：对偶数部分和奇数部分分别进行 DFT 计算，得到频域结果。

（3）**合并结果**：将递归计算出的子问题结果合并，得到最终的频域结果：

$$X(k)=X_{even}(k)+e^{-i2\pi\frac{k}{N}}X_{odd}(k), \tag{2.2}$$

$$X(k+N/2)=X_{even}(k)-e^{-i2\pi\frac{k}{N}}X_{odd}(k), \tag{2.3}$$

其中，$X(k)$是最终的频域信号，$X_{even}(k)$和$X_{odd}(k)$分别是偶数部分和奇数部分的 DFT 结果，$e^{-i2\pi\frac{k}{N}}$是旋转因子，称为“蝶形因子”。这个合并过程被称为“蝶形计算”，它是 FFT 中的核心操作。每个“蝶形”计算涉及两次复数乘法和两次复数加法，因此相较于直接计算 DFT，计算量大大减少，将时域到频域的转换复杂度从$O(N^2)$降低到$O(N\log N)$。

2.3 神经网络相关理论

2.3.1 流式模型

流式模型是一种生成模型，于 2015 年由 Rezende 等人[64]提出，它的核心思想是通过一系列可逆变换，将输入数据从复杂的分布映射到潜在空间，并通过精确的对数似然来学习数据的分布特征，然后通过逆变换进行数据生成。如图 2.2 所示为流式模型的基本结构示意图。假设输入流式模型的训练样本为$\mathcal{X}=\{x_1,x_2,\ldots,x_T\}$，流式模型将数据集中的每个点$x_t$通过一系列可逆变换$\{f_{\theta_1},f_{\theta_2},\ldots,f_{\theta_N}\}$映射到潜在空间$z_t$:

$$z_t=f_{\theta_N}\circ f_{\theta_{N-1}}\circ\ldots\circ f_{\theta_1}(x_t), \tag{2.4}$$

其中，θ_n是每层变换函数的参数，经过这些变换后，潜在变量z_t服从一个先验分布（例如，标准高斯分布）。通过这种方式，流式模型能够捕捉并重建输入数据的复杂分布特征和模式。变换函数f_θ学习到的是数据分布的潜在规律，而最终的潜在空间z_t是通过这些变换捕捉到的输入数据的分布特征和模式，并能够用于数据生成、密度估计等任务。在流式模型中，变换是可逆的，数据的概率密度通过以下公式进行计算：

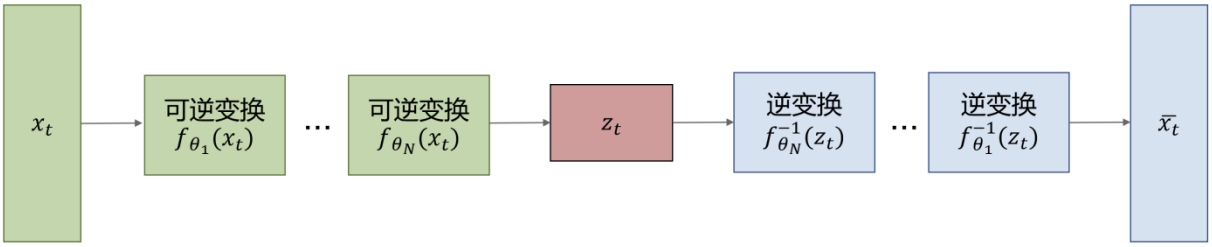

图 2.2 流式模型框架

Fig. 2.2 The framework of flow-based model

$$p_X(x_t) = p_Z(z_t)|\det\frac{\partial f_\theta(x_t)}{\partial x_t}|^{-1}, \tag{2.5}$$

其中，$p_X(x_t)$是数据x_t的概率密度，$p_Z(z_t)$是潜在变量z_t的概率密度，通常假设z_t服从标准高斯分布。此公式的意义是，数据x_t的概率密度由其在潜在空间的概率密度$p_Z(z_t)$和变换的雅可比行列式$|\det\frac{\partial f_\theta(x_t)}{\partial x_t}|$共同决定。雅可比行列式反映了变换过程中数据的拉伸或压缩，它是计算数据分布变化的关键。

流式模型的训练目标是最大化数据的对数似然，或者最小化负对数似然。数据的对数似然$\log p_X(x_t)$计算为：

$$\log p_X(x_t) = \log p_Z(f_\theta(x_t)) - \log |\det\frac{\partial f_\theta(x_t)}{\partial x_t}|, \tag{2.6}$$

$p_Z(f_\theta(x_t))$是潜在空间z_t的概率密度，$\log |\det\frac{\partial f_\theta(x_t)}{\partial x_t}|$是变换的对数雅可比行列式，表示输入空间的概率如何通过变换变化。为了训练流式模型，通常最小化负对数似然：

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T}\log p_X(x_t), \tag{2.7}$$

通过最小化负对数似然，可以使得变换后的潜在分布尽可能与真实数据分布相似。梯度下降算法被用来优化变换函数的参数θ，并且可以通过反向传播方法计算每一层变换的梯度。

训练完成后，流式模型从潜在空间的先验分布采样，并通过这些变换的逆函数$\{{f_{\theta_1}}^{-1}, {f_{\theta_2}}^{-1}, \ldots, {f_{\theta_N}}^{-1}\}$将潜在变量$z_t$映射回数据空间，生成新的数据样本$\bar{x_t}$：

$$\bar{x_t} = {f_{\theta_1}}^{-1} \circ {f_{\theta_2}}^{-1} \circ \ldots \circ {f_{\theta_N}}^{-1}(z_t). \tag{2.8}$$

2.3.2 混合专家模型

混合专家模型是一种判别模型，于 1991 年由 Jacobs 等人提出[65]，旨在通过使用多个专家（子模型）并根据输入数据的不同来选择不同的专家，从而提高模型的表达能力和效率。每个专家都是一个特定的子模型，可以专注于问题的某个部分，而整个模型根据输入的不同选择使用某些专家。混合专家模型的核心思想是：通过加权组合多个专家模型的输出，以便能够灵活地处理不同类型的输入。在训练过程中，每个专家

模型的参数会根据不同的任务或者数据分布进行优化。输入数据通过一个门控网络来决定哪一个或哪些专家会被激活，门控网络通常输出一个概率分布，表示每个专家的选择权重。

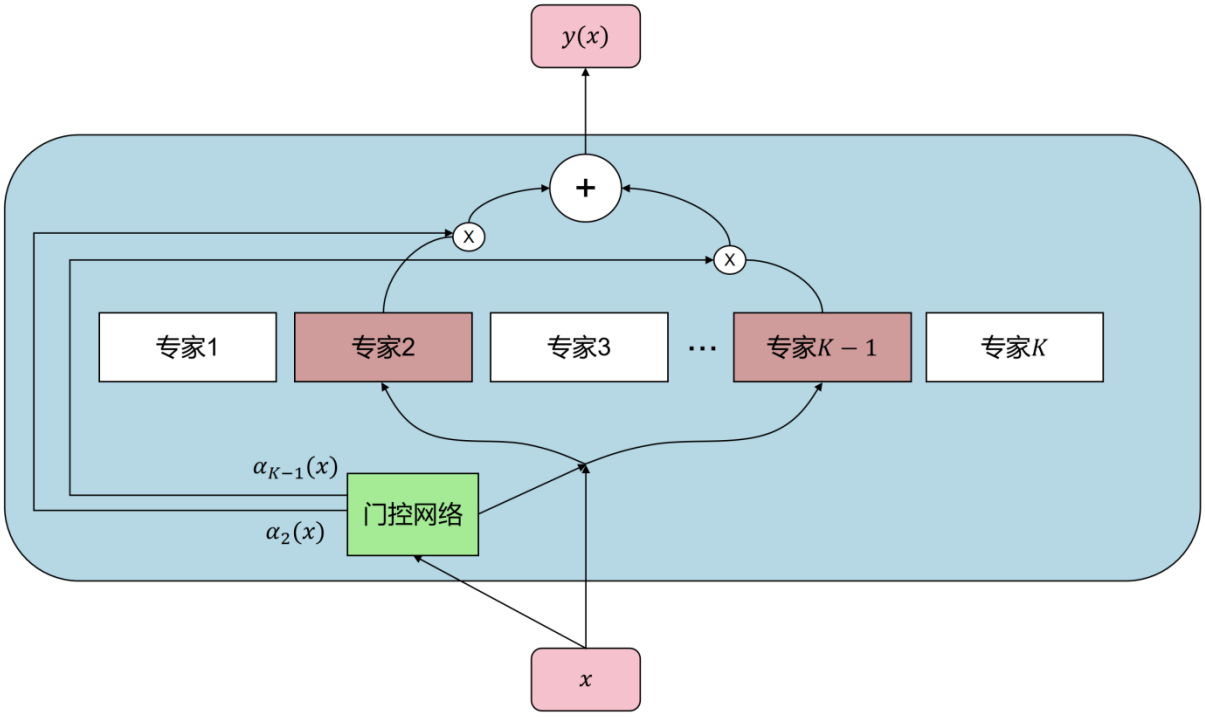

图 2.3 混合专家模型结构

Fig. 2.3 The framework of mixture of experts

如图 2.3 所示为混合专家模型的基本结构示意图。假设有K个专家模型，每个专家为f_k，它们根据输入x生成预测y_k：

$$y_k = f_k(x). \tag{2.9}$$

然后，使用一个门控网络$g(x)$来为每个专家分配一个权重$\alpha_k(x)$，这个权重反映了专家k在当前输入下的激活程度。门控网络的输出$\alpha_k(x)$通常是一个概率分布，满足：

$$\sum_{k=1}^{K} \alpha_k(x) = 1. \tag{2.10}$$

最后，模型的最终预测是各个专家输出的加权和：

$$y(x) = \sum_{k=1}^{K} \alpha_k(x)\, f_k(x). \tag{2.11}$$

门控网络$g(x)$通常使用一个 softmax 函数来为每个专家分配一个权重，对于一个给定的输入x，门控网络生成一组权重$\alpha_k(x)$，这些权重通过下式计算：

$$\alpha_k(x) = \frac{e^{g_k(x)}}{\sum_{i=1}^{K} e^{g_i(x)}}, \tag{2.12}$$

其中，$g_k(x)$是门控网络为专家k输出的标量值。训练混合专家模型时，目标是通过最小化某个损失函数来优化专家模型和门控网络的参数。常见的损失函数包括均方误差或交叉熵损失，具体取决于任务的类型。在训练过程中，专家模型和门控网络的参数都会被更新，以便更好地预测目标变量。

混合专家模型的优点在于其高效性、灵活性和扩展性。通过只激活少数几个专家，它能够显著减少计算量，特别是在面对复杂输入数据时。同时，每个专家可以专注于数据的某个特定方面，使得模型在处理异构数据时更具优势。此外，混合专家模型的结构具有很好的扩展性，可以通过添加更多的专家来提升模型的能力和性能，从而适应更广泛的应用场景。

2.3.3 长短期记忆网络

LSTM 是一种特殊的循环神经网络架构，用于处理和预测时序数据。它由 Hochreiter 等人[66]于 1997 年提出，旨在解决传统循环神经网络在处理长期依赖时出现的梯度消失问题。LSTM通过引入门控机制来有效地控制信息的流动，从而克服了传统循环神经网络的缺陷，能够更好地捕捉和记忆长期信息。

如图 2.4 所示，LSTM 的核心在于其由多个门机制和一个细胞状态组成的结构。通过这些门，LSTM能够选择性地记住或忘记信息，并控制信息流动，确保模型能够更好地捕捉长时间跨度的依赖关系。LSTM 在每个时间步t的处理可以分为以下几个步骤：

①遗忘门：决定了前一时刻的细胞状态中有多少信息应该被丢弃。

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{2.13}$$

其中，f_t是遗忘门的输出，W_f是权重矩阵，b_f是偏置，h_{t-1}是上一时刻的隐藏状态，x_t是当前输入，σ是 Sigmoid 激活函数，输出值在 0 和 1 之间，控制着细胞状态的遗忘比例。

②输入门：决定当前时刻的输入信息有多少应当被存储到细胞状态中。

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{2.14}$$

$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{2.15}$$

其中，i_t是输入门的输出，$\widetilde{C_t}$是当前输入信息的候选值，tanh是双曲正切激活函数，控制当前时刻输入信息的范围。

③更新细胞状态：通过遗忘门和输入门共同更新细胞状态。

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C_t}, \tag{2.16}$$

其中，C_t是当前时刻的细胞状态，C_{t-1}是上一时刻的细胞状态，控制了哪些信息被保留，哪些被丢弃。

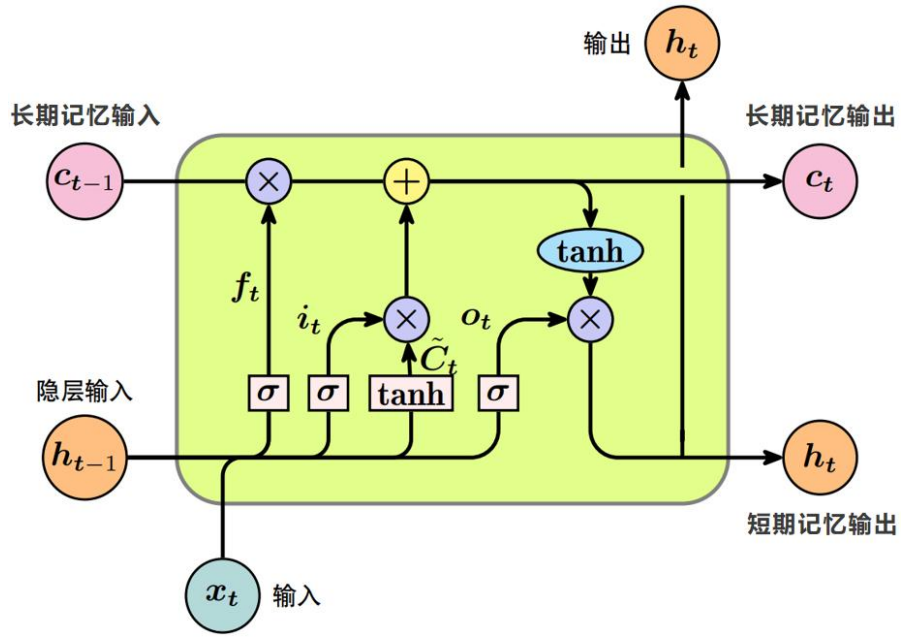

图 2.4 长短期记忆网络结构

Fig. 2.4 The framework of LSTM

④输出门：决定了当前时刻的隐藏状态（即输出）应当包含哪些信息。

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{2.17}$$

$$h_t = o_t \times \tanh(C_t), \tag{2.18}$$

其中，o_t是输出门的输出，h_t是当前时刻的隐藏状态（即 LSTM 的输出）。

LSTM 通过引入遗忘门、输入门、输出门和细胞状态，允许神经网络在处理长时间依赖的任务时，不会遇到梯度消失问题，从而能更好地捕捉和利用时序数据中的信息。

2.3.4 图注意力网络

在传统的图卷积网络[67]中，节点的特征通过与其邻居节点的特征进行加权平均来更新。但这种加权方式是固定的，通常依赖于图的结构信息。而在 GAT 中，引入了注意力机制，即对于每个节点的邻居，计算一个注意力系数，反映该邻居对当前节点的重要性。这样，不同的邻居节点对当前节点的影响程度可以不同。GAT 由 Velickovic 等人[68]在 2017 年提出，是一种用于处理图结构数据的深度学习模型，它通过引入注意力机制来动态调整图神经网络中的邻居节点权重，从而提升模型在不同图结构上的适应性和表达能力。如图 2.5 所示，对于图中的每个节点i，GAT 会根据节点特征和其邻居节点特征计算注意力权重，得到节点i的新特征。

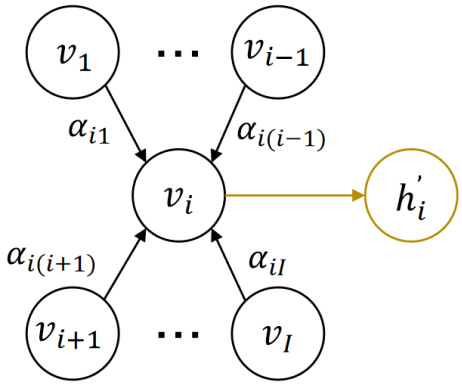

图 2.5 图注意力网络的节点特征计算

Fig. 2.5 Node feature calculation in GAT

GAT 使用一个注意力函数来计算节点之间的关系（即重要性）。通常，这个注意力函数包括一个前馈神经网络层，将节点特征进行映射。假设节点i和邻居节点j的特征分别为h_i和h_j，注意力权重α_{ij}计算如下：

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^T[Wh_i||Wh_j]))}{\sum_{k\in\mathcal{N}(i)}\exp(\text{LeakyReLU}(a^T[Wh_i||Wh_k]))}, \tag{2.19}$$

其中，a是一个注意力权重向量，W是节点特征的线性变换矩阵，$[Wh_i||Wh_j]$是节点特征的拼接，$\mathcal{N}(i)$是节点i的邻居节点集合。

计算完每个邻居节点的注意力系数后，节点i的新特征通过加权平均其邻居节点的特征来获得：

$$h_i' = \sigma(\sum_{j\in\mathcal{N}(i)} \alpha_{ij} W h_j). \tag{2.20}$$

如果需要提升模型的稳定性、增强特征表达能力或捕捉更多不同的邻域信息，则可以在 GAT 中使用多头注意力机制。多头注意力允许模型在多个独立的注意力空间中学习不同的邻域关系，从而减少单个注意力头可能带来的偏差，提高训练的鲁棒性。此外，在任务较为复杂或数据规模较大时，多头注意力可以帮助模型更好地提取全局和局部信息，提升最终的性能。多头注意力机制并行地计算注意力系数和加权特征，最后将它们的输出结果进行合并（通常是拼接或平均）：

$$h_i' = ||_{k=1}^{K}\sigma(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_j), \tag{2.21}$$

其中，K是注意力头的数量，$||$表示拼接操作。

2.3.5 强化学习

强化学习（Reinforcement Learning, RL）起源于控制理论和心理学中的行为主义学习理论。而现代强化学习的理论框架主要由 Sutton 等人在 1980 年代发展，并在 1998 年发表[69]，奠定了强化学习的基础。由于强化学习具备自主探索、长期决策优化和适应

性学习的能力，它在多个领域都有广泛的应用，如机器人控制、游戏 AI、智能推荐系统等。强化学习是一种以试错方式学习最优策略的机器学习方法，其中智能体与环境进行交互，通过执行动作获得奖励，并基于奖励优化策略以最大化长期回报。一般将智能体与环境交互的马尔可夫决策过程认为是强化学习的决策过程。

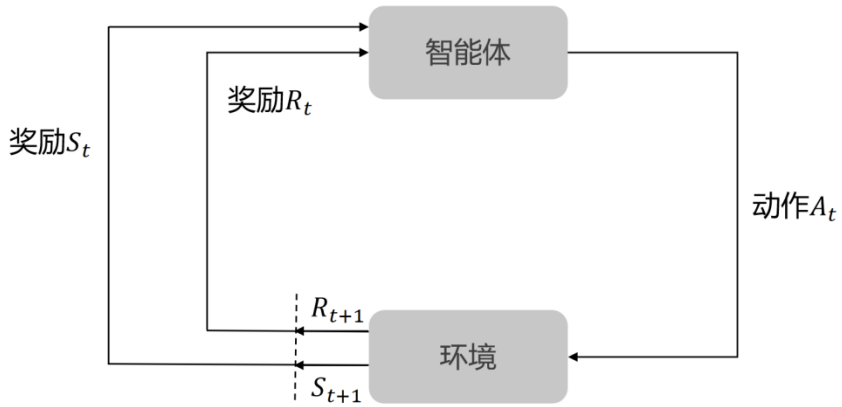

图 2.6 强化学习过程

Fig. 2.6 The progress of reinforcement learning

如图 2.6 所示是一个标准的 RL 模型，在t时刻，智能体从环境中得到状态S_t，并根据学习好的策略π将当前状态S_t映射到动作A_t。智能体在得到动作A_t后，在环境中采取该动作，而后环境会根据概率$p_{S_t S_{t+1}}^{A_t}$（表示智能体在状态S_t时采取动作A_t后转换为下一状态S_{t+1}的概率）反馈给智能体下一时刻的状态S_{t+1}，以及一个即时奖励R_{t+1}（反映的是执行A_t后，环境在$t+1$时刻返回的反馈）。

在强化学习中，问题目标被建模为最大化长期累计回报。所以强化学习就需要在智能体与环境不断的交互过程中更新映射策略π，以求能在接下来的时刻中选出最佳的动作A，使得累计回报最大化。智能体在状态S_t时采取动作A_t之后若转换为状态S_{t+1}，则获得的即时奖励表示为$R_{t+1} = r_t(S_t, A_t, S_{t+1})$，$r_t$表示奖励函数。智能体根据奖励值不断调整策略，并进入到下一个决策过程，重复进行这个交互过程，智能体就可以得到一个轨迹$(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \cdots)$。在观测到后续状态生成的即时奖励后，累计回报$G_t$可以表示为：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0} \gamma^k R_{t+1+k}\,, \tag{2.22}$$

其中$\gamma \in [0,1]$为折扣因子，表示后续状态所处时刻与当前时刻t越远，其产生奖励的未知性就越高，因而需要削弱其对当前时刻的影响。特别地，当γ越趋近于 0 时，表示智能体越关注短期奖励，当γ越趋近于 1 时，表示越注重长期累积回报。

2.4 本章小结

本章首先对时变工作负载数据的特征和时变工作负载调度进行了介绍。接着介绍了信号处理方法中的快速傅里叶变换。最后介绍了相关神经网络理论，包括流式模型、混合专家模型、长短期记忆网络、图注意力网络和强化学习。

3 基于动态模式学习与专家协同的时变工作负载预测方法

在云计算集群中，工作负载的序列预测直接影响着调度效率。通过准确预测工作负载的变化趋势，调度器可以提前优化资源分配，避免资源浪费或性能瓶颈，从而提升整体调度效率和服务质量。随着边缘计算、物联网和微服务架构的普及，现代云计算环境中的工作负载呈现出高度动态化和异构化的特征。这种时变特性不仅体现在任务请求量的波动上，更涉及资源需求类型和时空关联性等多维度的复杂耦合，其演化规律往往具有非平稳性和长程相关性。

面对云计算环境下时变工作负载的复杂时序特性，本文提出了一种基于动态模式学习与专家协同的时变工作负载预测方法 MOEP（Mixture of Experts Prediction）。其中，第 3.1 节简要定义了时变工作负载序列及其预测的含义；第 3.2 节阐述了本文设计该预测方法的动机；第 3.3 节构建了包含动态模式学习、时频域聚合与专家协同的预测架构；第 3.4 节通过真实数据集验证了模型的有效性；第 3.5 节总结了本章内容。本章主要阐述了所提出模型的定义及其数学表达，并通过整体框架图对各个模块的功能与设计目标进行了深入剖析。

3.1 问题定义

时变工作负载预测是云计算资源管理中的关键问题。在企业将服务部署至云端后，用户访问行为会动态生成工作负载，这些工作负载对计算、存储等资源的需求构成了一个多元时间序列。然而，由于系统监控的局限性和实时性要求，该序列往往只能被部分观测到：一方面，调度决策需要在完整序列尚未完全可知的情况下及时做出；另一方面，某些工作负载的资源需求特征只有在服务实际运行时才能被准确获取，无法通过被动等待获得完整信息。因此，时变工作负载预测的核心目标是通过预测模型，基于可观测的部分序列，构建完整的多维资源需求时间序列，为云工作负载调度提供可靠的数据支持。这一预测过程需要考虑工作负载的动态性、资源需求的关联性以及观测数据的稀疏性等特征，以实现对资源需求的准确预估。

定义 1：（时变工作负载序列）时变工作负载序列数据是一种包含多个变量（即不同资源维度需求）的在连续时间步上收集的数据形式。每个变量的取值随时间的推移而变化形成一个时间序列$W_i \in R^{n_i \times D}$，表示第i个工作负载的时变资源需求序列：

$$W_i = [w_i^1, \ldots, w_i^l, \ldots, w_i^{n_i}]^{\mathrm{T}}, \tag{3.1}$$

其中，w_i^l是在时间步l的一个D维资源向量矩阵，n_i是工作负载观测到的时间步长。

定义 2：（时变工作负载预测）为了建模预测时间步中的资源需求和已知序列之间的依赖关系，考虑一个长度为L的滑动窗口：

$$g_{n_i} = [w_i^{n_i-L+1}, \ldots, w_i^{n_i}]. \tag{3.2}$$

本章的目标是基于观测到的序列W_i，预测接下来T_i个时间步中工作负载的多维资源需求序列P_i，即：

$$P_i = [\hat{w}_i^{n_i+1}, \ldots, \hat{w}_i^{n_i+t}, \ldots, \hat{w}_i^{n_i+T_i}]^{\mathrm{T}}, \tag{3.3}$$

其中$\hat{w}_i^{n_i+t} \in R^{1\times D}$。

3.2 方法动机

本节首先对时变工作负载的模式开展量化分析，随后通过一个简单的初步实验验证了基于混合专家模型的预测框架在该场景下的有效性，这为构建自适应工作负载预测方法提供了关键支撑。

3.2.1 时变工作负载模式分析

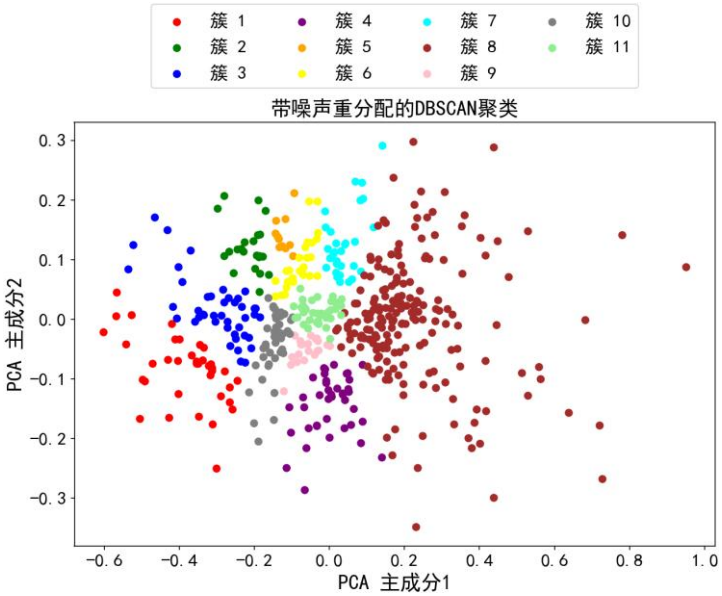

图 3.1 时变工作负载聚类结果

Fig. 3.1 Clustering results of time-varying workloads

本文采用数据驱动方法揭示时变工作负载的潜在模式特征。具体而言，从构建的时变工作负载轨迹中随机抽取 500 个具有代表性的工作负载序列作为分析对象，每个序列包含多个资源维度上的需求特征。为有效捕捉其内在相似性，首先通过主成分分析（Principal Components Analysis, PCA）进行特征空间重构：对原始工作负载矩阵进行标准化处理后，将多维时序数据投影至正交特征空间，选取累计方差贡献率最高的前两个主成分实现降维，在保留主要动态特征的同时，将每个工作负载序列映射为二维平面上的坐标点，如图 3.1 所示。在此基础上，采用带噪声的基于密度的空间聚类算法[70]（Density-Based Spatial Clustering of Applications with Noise，

DBSCAN）对降维后的数据进行无监督分组。算法成功识别出 11 个簇结构。针对算法标记的噪声点，进一步计算其与各簇质心的欧氏距离，采用最近邻原则进行二次归类，确保所有工作负载均获得可解释的类别归属。

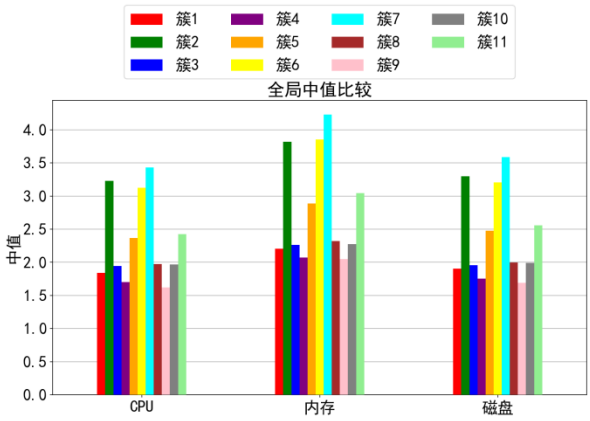

图 3.2 中值分析

Fig. 3.2 Median analysis

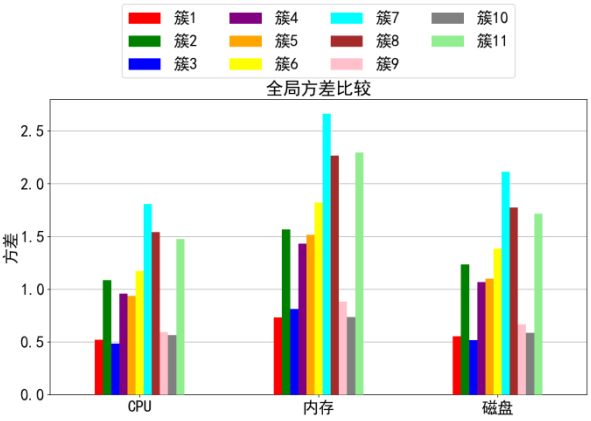

图 3.3 方差分析

Fig. 3.3 Variance analysis

随后，对每个簇的工作负载数据进行计算，求得每个特征的中值和方差。每个簇的第i个特征的中值M_i计算如下：

$$M_i = \text{median}(x_{i,1}, \ldots, x_{i,j}, \ldots, x_{i,J}), \tag{3.4}$$

其中，$x_{i,j}$是第j个工作负载在第i个特征上的取值，J是样本总数。

每个簇的第i个特征的方差V_i计算如下：

$$V_i = \frac{1}{J}\sum_{j=1}^{J}(x_{i,j} - \mu_i)^2, \tag{3.5}$$

其中，μ_i是第i个特征的全局均值：

$$\mu_i = \frac{1}{J}\sum_{j=1}^{J} x_{i,j}, \tag{3.6}$$

计算得到的每个簇（即不同的时变工作负载类）在每个特征上的中值及方差如图 3.2 和 3.3 所示。从中可以看出，不同的簇在特征上的分布呈现出一定的规律性。例如，簇 2、6 和 7 在各个特征上的中值普遍高于其他簇，表明这些簇的工作负载在各个特征上具有较高的平均水平，可能反映了这些簇的工作负载比其他簇具有更高的资源需求。此外，簇 7、8 和 11 在各个特征上的方差普遍高于其他簇，表明这些簇的工作负载在各个特征上的变化性较大，可能包含了较强波动性和不稳定性的样本。

这些中值和方差的差异反映了工作负载模式的多样性，因此在时变工作负载预测中，采用针对性模型进行独立训练是一个可行的方案。这也是本文选择使用流式模型

与 MOE 架构的核心动机。通过对不同类别的时变工作负载分别训练专门的专家模型，并在实际预测中使用流式模型捕捉工作负载模式产生专家激活权重，能够更精准地建模工作负载的特征和动态变化。这种方法不仅提高了模型对复杂工作负载模式的适应能力，还显著增强了预测的准确性和可靠性。

3.2.2 混合专家模型初步实验

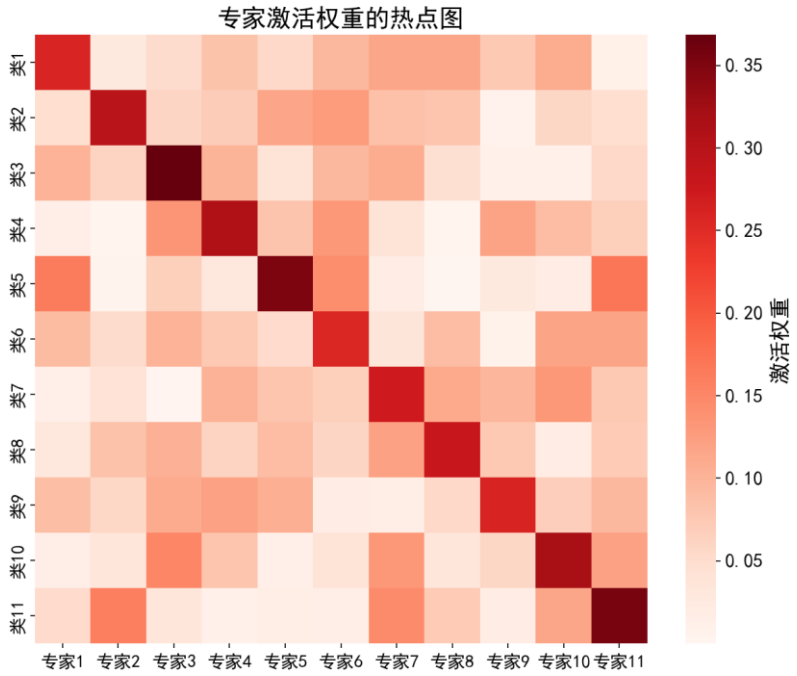

图 3.4 专家激活权重的热点图

Fig. 3.4 Heatmap of expert activation weights

为了进一步验证 MOE 架构的必要性，本文进行了一个简单的初步实验。实验中，所采用的时变工作负载预测模型是一个简单的 MOE 架构，设定专家数量为 11，并且每个专家均为标准的全连接网络。3.2.1 节中不同簇的时变工作负载被用于分别对各个专家进行训练。训练完成后，本文随机从时变工作负载轨迹中抽取了 200 个新样本，输入到 MOE 架构中进行测试。通过测试过程，得到了专家激活频率的热点图，如图 3.4 所示，可见不同类的时变工作负载在 MOE 架构中激活的专家和相应的权重均不同。

接着，本文选取在 11 个类中激活权重总和最高的 6 个专家组成一个新的 MOE 预测模型，名为 TOP-6。同时，随机选取 6 个专家组成另一个新的 MOE 预测模型，名为 Random-6。最后，再次随机从时变工作负载轨迹中抽取 200 个新样本，输入到这三个 MOE 预测模型中进行测试，比较它们的损失和用时。如图 3.5 所示，TOP-6 预测产生的损失稍稍落后于使用所有专家的模型，但却远优于 Random-6。如图 3.6 所示，TOP-6 用时最低且相比于使用所有专家的模型有明显的提升。由此可见，在预测任务中，

并不需要激活所有专家，就可以在大量降低预测用时的同时保持一个相对较高的预测精度。因此，本文通过采用 MOE 架构动态激活最优专家子集，降低模型推理时的计算复杂度，也就减少了预测时的时间开销，提升了模型的实时性。

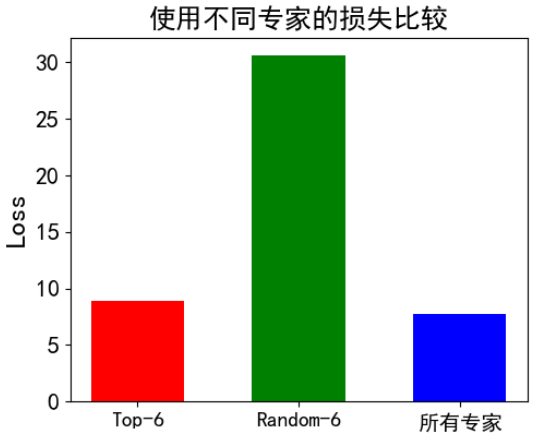

图 3.5 使用不同专家的损失比较

Fig. 3.5 Comparing loss from different experts

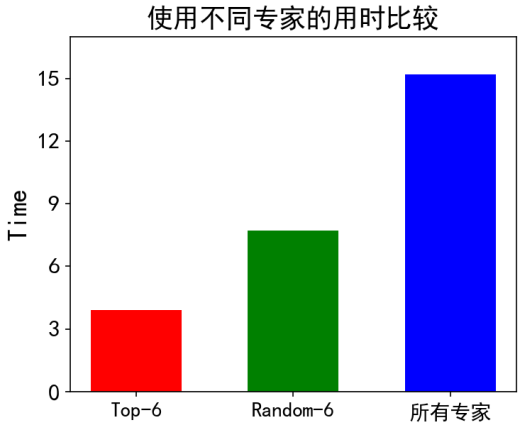

图 3.6 使用不同专家的用时比较

Fig. 3.6 Comparing time from different experts

3.3 预测模型总体框架

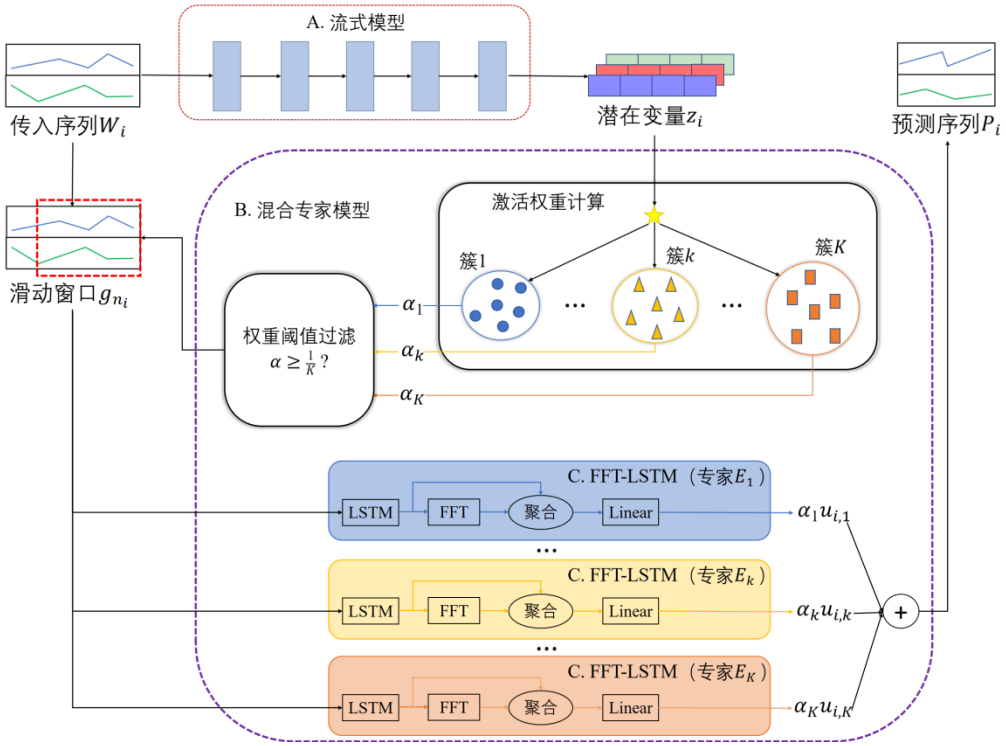

图 3.7 MOEP 的工作流程

Fig. 3.7 Workflow of MOEP

为了提升预测的有效性和实时性，本文提出了一种新的基于动态模式学习与专家协同的时变工作负载预测框架 MOEP，如图 3.7 所示。首先，将传入的工作负载输入流式模型，通过一系列可逆变换捕捉其序列模式，将其映射到潜在空间。其次，通过潜在空间中工作负载与已知类的距离度量，得到 MOE 中不同专家的激活权重。然后，采用一个权重阈值过滤模块动态构建最优专家子集，将工作负载的滑动窗口输入最优专家子集中的每个专家。接着，最优专家子集中的每个专家，采用 FFT-LSTM 对滑动窗口进行时频域聚合分析，产生各自的预测输出。最后，MOE 根据不同专家的激活权重加权计算得到最终的序列预测结果。接下来将详细说明框架内的主要部件。

3.3.1 流式模型

时变工作负载具有强烈的时序依赖性和高度非线性，这使得传统模型难以有效捕捉其复杂模式。流式模型通过一系列可逆变换将数据映射到潜在空间，能够自动学习这些非线性和时变的依赖关系，避免了手工特征工程的需求。其可逆性和高效的密度估计能力，使得流式模型能够捕捉到工作负载的复杂序列模式，成为处理此类数据的理想选择。

在应用流式模型时，首先需要对时变工作负载数据进行适当的预处理。假设时变工作负载数据集为$\{W_1, W_2, \ldots, W_N\}$，其中每个$W_i \in R^{n_i \times D}$。为了让数据更适合流式模型的训练，本文对每个维度的数据进行标准化处理，使其具有零均值和单位方差：

$$\widetilde{W_i} = \frac{W_i - \mu}{\sigma}, \tag{3.7}$$

其中，μ和σ分别是数据的均值和标准差。标准化后的数据$\widetilde{W_i}$更容易适应流式模型的训练，并减少训练过程中的数值不稳定性。

流式模型的关键在于通过一系列可逆变换将输入数据从原始空间映射到潜在空间。假设流式模型有M层变换，这些变换以可逆的仿射变换层作为基础，本文将每一层的变换函数f_m设计为：

$$f_m(x) = \begin{pmatrix} y_a \\ y_b \end{pmatrix} = \begin{pmatrix} x_a \\ x_b \end{pmatrix} \odot \exp(s(x_a)) + t(x_a), \tag{3.8}$$

其中，$y = \begin{pmatrix} y_a \\ y_b \end{pmatrix}$是输出数据，只保留$y_b$部分。$x = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$是输入数据的分割，其中$x_a$是用于计算变换参数的部分，而$x_b$通过变换得到输出部分。输入的时变工作负载按照维度$D$进行分割，组合其中$D-1$维作为$x_a$，剩下的 1 维作为$x_b$。最终产生的$D$组$y_b$通过拼接得到最终输出。$\odot$表示逐元素相乘，$s(x_a)$和$t(x_a)$分别是可学习的变换参数，即由神经网络生成的函数。 变换函数通过$s(x_a)$和$t(x_a)$来调整x_b，使得变换具有足够的灵活性来捕捉输入数据的分布特征。

数据在每一层变换中都会经历类似的过程，直到经过M层的变换，最终得到潜在变量z_i：

$$z_i = f_M \circ f_{M-1} \circ \ldots \circ f_1(\widetilde{W_i}). \tag{3.9}$$

这个过程中，变换函数f_m学习到的是数据分布的潜在规律，而最终的潜在变量z_i是通过这些变换捕捉到的工作负载的分布特征和模式。根据流式模型的对数似然公式，数据的对数概率密度可以表示为：

$$\log p(\widetilde{W_i}) = \log p(z_i) - \log \left|\det \frac{\partial f_M}{\partial \widetilde{W_i}}\right|, \tag{3.10}$$

其中，$\log p(z_i)$是潜在空间中潜在变量的概率密度，可表示为标准高斯分布的对数：

$$\log p(z_i) = -\frac{1}{2}||z_i||^2 - \frac{D}{2}\log(2\pi). \tag{3.11}$$

而雅可比行列式$\left|\det \frac{\partial f_M}{\partial \widetilde{W_i}}\right|$描述了每一层变换中数据空间的拉伸或压缩，反映了变换的线性特性。对于每一层的变换，都需要计算其雅可比行列式并将其累积，以便正确计算变换后的数据的概率密度。

为了预训练流式模型，目标是最小化负对数似然。具体地，训练目标为：

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log p(\widetilde{W_i}). \tag{3.12}$$

在预训练过程中，会通过反向传播计算每一层变换的梯度。具体来说，对于每一层变换f_m，变换函数的梯度计算为：

$$\nabla_{\theta_m}\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \nabla_{\theta_m} \log p(\widetilde{W_i}), \tag{3.13}$$

然后使用梯度下降算法更新变换函数的参数θ_m，直至对数似然收敛为止。

3.3.2 混合专家模型

MOE 通过多专家协同提升时变工作负载的预测精度，其并行架构设计同时保障计算效率，支持实时动态响应。传统的 MOE 模型中，门控网络负责根据输入选择最合适的专家进行预测。在针对时变工作负载的预测任务中，本文提出一种基于距离度量的激活权重计算方法，以增强模型的适应性和计算效率。

本文从集群的历史轨迹中收集了一批工作负载$\{W_1, W_2, \ldots, W_N\}$，这些数据通过流式模型映射到潜在空间$\{z_1, z_2, \ldots, z_N\}$。然后，本文利用动态时间归整（Dynamic Time Warping, DTW）算法[71]来计算潜在变量间的距离度量。潜在变量z_i和z_g之间的距离度量的计算方法为：

$$\mathcal{D}(z_i, z_g) = \mathcal{D}_{\min}(z_i^{n_i}, z_g^{n_g}), \tag{3.14}$$

其中，n_i和n_g分别是潜在变量z_i和z_g的长度。该计算的递归公式为：

$$\begin{aligned}\mathcal{D}_{\min}(z_i^{n_i}, z_g^{n_g}) = \min\{&\mathcal{D}_{\min}(z_i^{l_i}, z_g^{l_g-1}), \mathcal{D}_{\min}(z_i^{l_i-1}, z_g^{l_g}),\\ &\mathcal{D}_{\min}(z_i^{l_i-1}, z_g^{l_g-1})\} + \mathcal{M}(z_i^{l_i}, z_g^{l_g}),\end{aligned} \tag{3.15}$$

其中，$\mathcal{M}(z_i^{l_i}, z_g^{l_g}) = \sum_{d=1}^{D} ||z_{i,d}^{l_i} - z_{g,d}^{l_g}||^2$，$D$表示潜在变量的维度。

最后，本文应用 DBSCAN 聚类算法来对这批潜在变量进行聚类，生成C个簇。该算法需要两个参数：ϵ 和 minPts。对于每个潜在变量z_i，如果它的 ϵ-邻域内至少含有 minPts 个潜在变量，那z_i被视为一个核心点。ϵ-邻域指的是一个给定点周围 ϵ 半径内的所有点。如果一个点的 ϵ-邻域满足这个条件，则认为该点是一个核心点。核心点在聚类中至关重要，因为它们代表了密度更高的区域，这有助于定义聚类的结构。首先，如果z_i是一个核心点，且$\mathcal{D}(z_i, z_g) <= \epsilon$，则认为从$z_i$到$z_g$的路径是直接密度可达的。密度可达性是 DBSCAN 中的一个关键概念，用于定义聚类的边界。其次，给定核心点z_i，z_g和z_e，如果从z_i到z_g的路径和从z_g到z_e的路径都是直接密度可达的，那么从z_i到z_e的路径是密度可达的。然后，如果一个核心点使得从z_g到z_i和z_g到z_e的路径都是密度可达的，那么z_i和z_e被定义为密度相连。这种连通性反映了数据点之间的密度关系。即使两个点之间没有直接的密度可达路径，只要它们可以通过中间的核心点连接起来，它们就被认为是同一簇的一部分。最后，所有密度相连的点都形成了一个簇。每个簇C_k对应一个专家网络E_k，如图3.7所示，每个专家独立地使用该簇数据预训练以捕捉该簇的数据特性。

对于待预测工作负载W_i，首先通过流式模型捕捉其序列模式并将其映射到潜在空间z_i。接下来计算z_i与每个簇的质心C_k之间的距离$\mathcal{D}(z_i, C_k)$。基于这些距离度量，可以计算专家激活权重为：

$$\alpha_k = \frac{e^{-\mathcal{D}(z_i, C_k)}}{\sum_{k=1}^{K} e^{-\mathcal{D}(z_i, C_k)}}. \tag{3.16}$$

距离越小，激活权重越高，意味着工作负载与该簇的相似度越大，应该由该专家更多的参与预测。同时，为了动态激活最优专家子集，降低模型推理时的计算复杂度，本文引入了一个权重阈值过滤模块。具体来说，如果专家E_k的激活权重$\alpha_k \geq \frac{1}{K}$，则将该专家纳入最优专家子集$\phi$，使其参与后续的序列预测任务，如图 3.7 所示。此外，为了确定待预测工作负载W_i的预测序列P_i的时间步长T_i，本文同样通过加权处理。先计算得到最优专家子集簇中的，所有时变工作负载完整序列的平均时间步长$\mathbb{T}_k$，再通过加权计算得到待预测时变工作负载的长度T_i：

$$T_i = \sum_{k \in \phi} \alpha_k \mathbb{T}_k - n_i. \tag{3.17}$$

基于距离度量的激活权重计算方法替代了传统 MOE 中的门控网络，带来了计算复杂度的减少、适应性增强和模型泛化能力的提升。该方法避免训练门控网络，只需简单的距离计算即可实现专家选择，且距离度量能精准反映工作负载与不同工作负载类的相似性，使得模型能够自适应不同工作负载特性，从而提升预测准确性并更好应对动态变化的工作负载环境。

3.3.3 FFT-LSTM

在得到 MOE 中最优专家子集的激活权重及预测序列的时间步长后，就需要通过加权计算其中各个专家的输出得到最终的序列预测结果。如图 3.8 所示，本文将专家模块设计为 FFT-LSTM，该模型的核心思想是结合时域特征与频域特征，通过 LSTM 捕捉序列的时序依赖，利用 FFT 提取周期性或趋势性频域信息，最终融合两类特征提升预测精度。首先，将滑动窗口$g_{n_i} = [w_i^{n_i-L+1},\ldots,w_i^{n_i}]$输入 LSTM 层，逐时间步更新隐藏状态，输出时域隐藏状态序列$H_i = [h_i^{n_i-L+1},\ldots,h_i^{n_i}]$。接着，将$H_i$输入 FFT 层，提取频域信息$F_i = [x_i^{n_i-L+1},\ldots,x_i^{n_i}]$。然后，在时频域聚合模块中，将 LSTM 的时域输出H_i和 FFT 转换后的频域输出F_i进行拼接聚合：

$$Q_i = H_i \oplus F_i, \tag{3.18}$$

其中，Q_i是时频域聚合后的表示。通过这种方式，时域和频域的信息得以结合，形成更丰富的特征表示。最后，将时频域聚合后的结果Q_i通过一个全连接的线性层进行处理，得到每个专家E_k最终的预测输出$u_{i,k} = [\hat{w}_{i,k}^{n_i+1},\ldots,\hat{w}_{i,k}^{n_i+T_i}]$。

此时，最终的序列预测结果P_i计算为：

$$P_i = \sum_{k\in\phi} \alpha_k u_{i,k}. \tag{3.19}$$

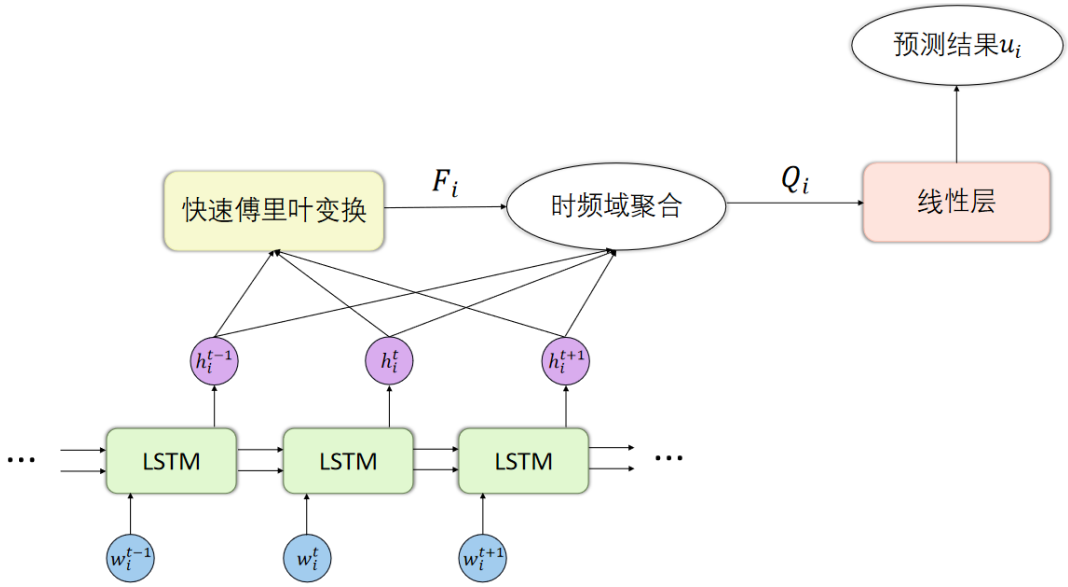

图 3.8 FFT-LSTM 详解

Fig. 3.8 The details of FFT-LSTM

为了在在线预测中训练整个预测框架中的所有神经网络，本文设计了独特的损失函数表示：

$$\mathcal{L}_{All} = \mathcal{L}_{rc} + \mathcal{L}_{cl} + \mathcal{L}_{pr}, \tag{3.20}$$

$$\mathcal{L}_{rc} = \sum_{t=1}^{n_i} (r_i^t - w_i^t) * (r_i^t - w_i^t)^{\mathrm{T}}, \tag{3.21}$$

其中，r_i^t表示z_i^t经流式模型逆变换重构后得到的数据，$\mathcal{L}_{rc}$也就是原始序列与重构序列间的损失表示，该损失函数确保映射后得到的潜在变量z_i能保留最完整的信息。

$$\mathcal{L}_{cl} = \mathcal{D}(z_i, C_i), \tag{3.22}$$

$\mathcal{D}(z_i, C_i)$表示通过 DTW 算法计算潜在变量z_i与其最近邻簇质心C_i间的距离，$\mathcal{L}_{cl}$即对应的损失，它尝试将工作负载序列映射到已知的模式中。

$$\mathcal{L}_{pr} = \sum_{t=1}^{T_i} (\widehat{w}_i^{n_i+t} - y_i^{n_i+t}) * (\widehat{w}_i^{n_i+t} - y_i^{n_i+t})^{\mathrm{T}}, \tag{3.23}$$

$y_i^{n_i+t}$表示预测时间步中的真实值，$\mathcal{L}_{pr}$也就是预测序列与真实序列间的损失表示。最终通过计算$\mathcal{L}_{All}$，可同时更新流式模型以及 MOE 中被激活专家的网络参数。

3.3.4 算法步骤与复杂度分析

在前面的几节中，已经详细介绍了每个组件的具体实现和应用。在这里，本文在算法 3.1 中总结了 MOEP 的方法流程。它封装了上述三个组件的主要操作和交互，为 MOEP 提供了一个统一的框架。

算法 3.1 中每次预测的复杂度为$O(DM + Kn_i + \phi L \log L)$。其中，$O\ (DM)$考虑了与流式模型映射相关的复杂度。$O\ (Kn_i)$表示计算$W_i$与$K$个簇的距离度量的复杂度，因为在 DTW 计算中会有递归调用。$O\ (\phi L \log L)$表示ϕ个专家产生预测输出的复杂度。

3.4 实验评估

在本节中，本文使用真实数据集进行实验，对本文提出的方法进行了评估，并与其它几种方法进行了比较，最后根据实验结果进行分析。

3.4.1 数据集介绍

本文采用的Google traces[72] 数据集是谷歌公开的大规模分布式集群运行数据，其覆盖约12,000台物理服务器长达29天的运行记录，包含超过6.7万个应用和4,000万个任务事件。数据集通过结构化日志（如CSV格式）记录了多维信息，包括机器静态配置（CPU、内存容量）、动态任务事件（提交/终止时间、资源需求）以及细粒度资源使用指标（CPU/内存/磁盘的秒级监控数据）。此外，其高真实性与完整性使其成为分布式系统资源调度、能耗优化及故障预测等领域研究的基准数据集，为学术界和工业界提供了大规模生产环境的实证分析基础。本文从Google traces提取与时变工作负载预测相关的信息如表3.1所示。

首先，本文从Google traces中随机抽取10个异构服务器。通过“Machine_events”和“Task_resource_usage”中的“服务器ID”，找出已经在这些服务器上运行过的工作负载的“Task ID”。随后在“Task_events”中，根据“时间戳”与“Task ID”提取它们跨$D = 3$个资源维度（CPU、内存和磁盘）的，可观测的时变资源需求，用作本文

的待预测工作负载W。为了得到完整的时变工作负载序列用于训练和测试，再从“Task_resource_usage”中，根据时间戳与服务器资源使用情况，构建出完整的时变工作负载序列。

算法 3.1：在线预测和训练

输入：待预测的工作负载 W_i；滑动窗口g_{n_i}

输出：预测序列P_i；

按 3.3.1 节所述利用式（3.12）与式（3.13）预训练流式模型；
按 3.3.2 节所述利用 DTW 与 DBSCAN 算法预训练 MOE；
$z_i \leftarrow$ 流式模型(W_i)；
for $k = 1$ **to** K **do:**
    利用式（3.14）计算距离度量$\mathcal{D}(z_i, C_k)$；
end for
for $k = 1$ **to** K **do:**
    利用式（3.16）计算激活权重α_k；
    if $\alpha_k \geq \frac{1}{K}$ **do:**
        最优专家子集ϕ中加入专家E_k；
    end if
end for
利用式（3.17）计算预测序列长度T_i；
for k **in** ϕ **do:**
    $H_{i,k} \leftarrow$ 专家E_k的 LSTM(g_{n_i})；
    $F_{i,k} \leftarrow$ FFT$(H_{i,k})$；
    $Q_{i,k} \leftarrow H_{i,k} \oplus F_{i,k}$；
    $u_{i,k} \leftarrow$ 专家E_k的线性层$(Q_{i,k})$；
end for
预测序列$P_i \leftarrow$ MOE 加权计算$(\alpha_k, u_{i,k})$；
利用式（3.21）计算重构损失$\mathcal{L}_{rc}$；
利用式（3.22）计算映射损失$\mathcal{L}_{cl}$；
利用式（3.23）计算预测损失$\mathcal{L}_{pr}$；
利用式（3.20）计算总损失$\mathcal{L}_{All}$并更新网络参数；
return 预测序列P_i；

表3.1谷歌数据集信息

Table 3.1 The statistics of Google traces

Machine_events	Task_events	Task_resource_usage
时间戳	时间戳	时间戳
服务器ID	Task ID	服务器ID Task ID
资源容量	资源请求	服务器资源使用情况
	事件类型	

表 3.2 实验环境

Table 3.2 Experimental environment

项目	描述
CPU	18 vCPU AMD EPYC 9754 128-Core Processor
GPU	GeForce GTX 4090D
内存	60G
显存	24G
操作系统	Linux
编程语言	Python3.1.0
IDE	PyCharm Professional Edition 2021.3.1
支持库	Pytorch 2.1.0

3.4.2 实验环境与参数设置

在本文的实验过程中采用 Linux 操作系统作为实验平台，并采用 PyCharm 作为开发工具。为了实现本章模型 MOEP，使用深度学习框架 Pytorch。具体的实验环境如表 3.2 所示。表 3.3 给出了算法实验过程中神经网络模型的主要参数设置，在 MOEP 训练过程中使用 Adam 优化器来更新网络中的权重参数。

3.4.3 基线算法

为了说明所提出的时变工作负载预测方法 MOEP 的有效性，本文使用 8 种预测算法作为基线算法，包括了 3 个基于统计模型的算法和 5 个基于深度学习的算法。

（1）Arima[5]：该方法通过自回归、差分和移动平均的组合建模时间序列的线性趋势和周期性，捕捉历史数据的统计依赖关系进行预测。

（2）ES[6]：该方法基于指数加权平均对历史数据进行衰减加权，赋予近期数据更高权重，直接拟合时间序列的局部趋势。

表 3.3 神经网络模型参数设置

Table 3.3 The neural network model parameter settings

参数名	参数值
滑动窗口L	20
流式模型变换层数M	4
专家数量K	12
LSTM 层数	1
隐藏单元数	32
潜在空间维度	3
预测步长	4
学习率	0.001

（3）Hybrid model of Arima and ES[7]：该方法结合 Arima 的线性建模能力与 ES 的非线性动态捕捉，通过串联或残差修正提升预测鲁棒性。

（4）DLinear[73]：该方法利用神经网络中的线性层堆叠，通过多层线性变换提取时间序列的高阶线性特征实现端到端预测。

（5）NLinear[73]：该方法采用非线性激活函数构建神经网络，学习时间序列中复杂的非线性模式和多变量交互关系。

（6）Transformer[74]：该方法通过自注意力机制动态分配不同时间步的关联权重，捕捉长期依赖和全局时序模式以建模复杂波动。

（7）TimesNet[75]：该方法将时间序列转换到频域提取多周期分量，并通过卷积网络建模周期内细粒度变化以优化预测。

（8）VAE-MOE[76]：相比于 MOEP，该方法将流式模型替换为变分自编码器（Variational Auto Encoder, VAE），一种基于变分推断的生成模型。

3.4.4 评价指标

本节设计了三种评估指标来量化时变工作负载序列预测模型的性能。基于时序预测的核心目标是缩小预测值与真实值的偏差，因此采用均方误差（Mean Squared Error, MSE）作为基础评价指标，其平方特性可敏感捕捉极端偏差对系统资源调度造成的风险。同时，为了衡量预测误差在动态工作负载量级下的相对影响，采用平均绝对百分比误差（Mean Absolute Percentage Error, MAPE）来评估不同负载波动区间的百分比偏离程度。此外，平均绝对误差（Mean Absolute Error, MAE）也被纳入指标体系，其线性计算特性能够直观反映预测序列与真实序列的整体偏离幅度。MSE、MAPE 和 MAE 计算如下：

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y_i})^2, \tag{3.24}$$

$$\mathrm{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \overline{y_i}}{y_i}\right|, \tag{3.25}$$

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \overline{y_i}|, \tag{3.26}$$

其中，y_i是真实值，$\overline{y_i}$是预测值，N是序列长度。

3.4.5 实验结果与分析

表 3.4 预测的整体性能

Table 3.4 The overall performance on prediction

方法	MSE	MAPE	MAE
Arima	$2.654e^{-7}$	8.192%	$3.295e^{-4}$
ES	$2.872e^{-7}$	8.627%	$3.462e^{-4}$
Arima and ES	$2.477e^{-7}$	7.885%	$2.828e^{-4}$
DLinear	$5.943e^{-6}$	46.462%	$1.465e^{-3}$
NLinear	$3.290e^{-6}$	19.566%	$9.790e^{-4}$
Transformer	$2.364e^{-7}$	7.586%	$2.683e^{-4}$
TimesNet	$2.475e^{-7}$	7.784%	$2.767e^{-4}$
VAE-MOE	$2.305e^{-7}$	7.403%	$2.581e^{-4}$
MOEP	**$2.265e^{-7}$**	**7.160%**	**$2.396e^{-4}$**

如表 3.4 是各个算法分别在 MSE、MAPE 和 MAE 三个指标上的性能评估。MOEP 在所有评估指标上都表现得优于其他方法，展现了其在工作负载序列预测中的优越性能。

基于统计学的 Arima、ES 及其混合模型表现相对较弱。这源于其对线性假设的依赖和固定参数化设计：Arima 难以建模非线性趋势与突变模式，而 ES 对季节性变化的敏感性导致长期预测误差累积。虽然混合模型通过结合两者提升了效果，但仍受限于统计方法对复杂时序特征的表征能力。

DLinear 和 NLinear 的预测误差显著偏高，尤其 DLinear 的 MAPE 高达 46.46%。这类线性网络将时间序列分解为趋势和周期项，但线性投影层难以捕捉序列中的动态交互关系，且分解过程中的信息损失导致对噪声和非平稳数据的适应性较差。

尽管 Transformer 和 TimesNet 通过注意力机制和多周期建模提升了效果，但其性能仍落后于 MOEP。Transformer 的自注意力机制在长序列建模时存在局部模式关注偏差，

而 TimesNet 的频域特征与时域信息缺乏深入交互，容易引入与任务无关的伪周期信号。同时，两者均因计算复杂度过高导致训练数据利用效率不足。

VAE-MOE 虽通过 VAE 和混合专家结构达到次优效果，但其 VAE 组件的隐变量采样过程会损失序列细粒度特征，且证据下界优化目标与预测任务的匹配度不足。相比之下，本文采用的流式模型通过学习机制直接捕捉序列演化模式，避免了生成式模型的分布假设偏差问题。

3.4.6 复杂度比较

表 3.5 复杂度比较

Table 3.5 Complexity comparison

方法	#Params (K)	平均预测时间 (ms)
Arima	/	128723
ES	/	115774
Arima and ES	/	277527
DLinear	0.60	< 0.01
NLinear	0.80	< 0.01
Transformer	562.69	131.53
TimesNet	222.08	1377.72
VAE-MOE	15.85	17.28
MOEP	**15.52**	**16.87**

表 3.5 统计了各个算法的平均预测时间，以及基于深度学习的算法的模型参数规模。DLinear 和 NLinear 的预测时间极低，但二者在预测精度上存在显著缺陷。其轻量化设计以牺牲模型表达能力为代价，导致预测误差远超其他方法。

MOEP 与 VAE-MOE 均采用 MOE 架构，通过距离度量机制实现计算资源的按需分配。相较于传统大模型（如 Transformer），MOE 结构在参数规模缩减 97%的同时，将预测时间压缩至 1/8，同时通过专家网络的竞争性激活精准建模序列特征，兼顾效率与预测性能。

Transformer 和 TimesNet 虽然在预测时能够提供较为可靠的结果，但它们的模型复杂度较高，且预测时的时间开销也大。这使得它们在需要快速响应的实时预测任务中，可能无法提供足够的预测速度，限制了其实际应用的广泛性。

基于统计学的 Arima、ES 及其混合模型因依赖极大似然估计和网格搜索进行参数优化，计算复杂度呈指数级增长。例如混合模型需对 Arima 和 ES 的超参数空间进行联合搜索，导致单次预测耗时高达 277 秒，远低于神经网络模型的推理效率。

3.4.7 消融实验

为了验证 MOEP 中所提出的组件的有效性，本文设计了 MOEP 的三个变体进行消融实验，分别是：①删除流式模型，采用门控网络加权；②删除 FFT-LSTM，专家替换为 LSTM；③删除 MOE 架构，仅使用 FFT-LSTM 预测。表 3.6 显示了 MOEP 及其三种变体之间的比较。可以看到，MOEP 取得了比其他变体更好的结果，分析得到以下结论：

①移除流式模型后，各个指标均有下降。这表明流式模型能通过概率密度聚类动态捕捉工作负载的时空演化模式，而替代的门控网络因静态权重分配机制无法自适应序列突变，因滞后响应导致专家激活权重与真实模式失配。

②当 FFT-LSTM 被替换为普通 LSTM 时，各个指标均有下降。这是由于 LSTM 仅依赖时域特征，而 FFT-LSTM 通过快速傅里叶变换提取频域周期性信息（如工作负载需求中的固定间隔峰值），二者联合建模可增强对复杂负载模式的表征。

③删除 MOE 结构后，模型性能显著下降。MOE 利用距离度量机制将输入分配给不同的专家模块，例如针对高吞吐负载的专家和针对周期性负载的专家，从而更精准地适应不同模式的工作负载。相比之下，单一的 FFT-LSTM 由于参数共享，对不同模式的建模存在冲突。例如，在负载模式波动较大的时段，MOE 能够动态激活更适合当前变化的专家，而单一模型由于参数固定，难以及时调整以适应快速变化的负载特征。

表 3.6 消融实验

Table 3.6 The ablation experiments

方法	MSE	MAPE	MAE
w/o 流式模型	$2.331e^{-7}$	7.532%	$2.614e^{-4}$
w/o FFT-LSTM	$2.562e^{-7}$	7.896%	$2.901e^{-4}$
w/o MOE	$2.615e^{-7}$	8.019%	$3.217e^{-4}$
MOEP	$\mathbf{2.265e^{-7}}$	**7.160%**	$\mathbf{2.396e^{-4}}$

3.5 本章小结

在本章中，本文提出了一种新的基于动态模式学习与专家协同的方法 MOEP 来预测时变工作负载。具体来说，本文提出了一个流式模型和聚类密度的组合，捕捉时变工作负载模式并生成专家激活权重。然后，本文基于不同工作负载模式训练相应的专家，共同构建 MOE 以降低推理开销并提高预测精度。此外，FFT-LSTM 作为预测网络，聚合了来自工作负载的时频域信息，以产生更优的预测结果。在 Google traces 上进行的模拟实验表明，MOEP 优于现有的基线。

4 基于优先级调控与资源感知的时变工作负载调度方法

第三章针对时变工作负载的预测为本章的动态调度策略奠定了基础。云计算环境下，用户请求、数据处理任务等工作负载往往呈现显著的时变特性，具有不同的需求峰值与波动趋势，云端服务器集群也展现出异构性，处理能力及运载能力各异。面对如此复杂且动态变化的情形，本文提出了一种基于优先级调控与资源感知的时变工作负载调度方法 PRAS（Priority and Resource Awareness Scheduling），通过灵活的工作负载调度策略，实现资源的最优分配与动态调度。其中，第 4.1 节定义了云计算环境下的时变工作负载调度；第 4.2 节构建了包含优先级调控与资源感知的调度架构；第 4.3 节通过真实数据集验证了模型的有效性；第 4.4 节总结了本章内容。

4.1 问题定义

本文考虑一个跨云计算集群中所有服务器的时变工作负载调度场景，其中工作负载以在线的方式到达和离开，如图 4.1 所示。调度器接收传入到集群的工作负载，并考虑工作负载序列的资源需求和服务器的现状，然后根据其调度策略将工作负载分配给指定的服务器进行处理。

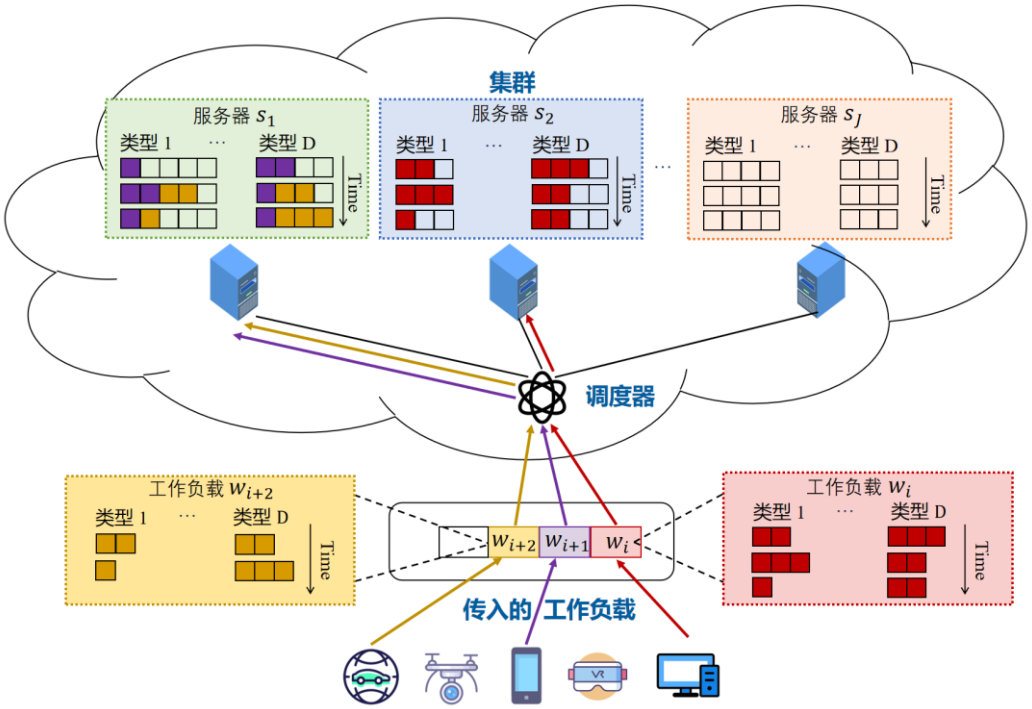

图 4.1 时变工作负载调度场景的一个图解

Fig. 4.1 An illustration of the time-varying workload scheduling scenario

本文考虑一个由J个服务器组成的集群的情况，表示为$\{s_1,\ldots,s_j,\ldots,s_J\}$。每个服务器都拥有 D 个资源空间维度，如 CPU、内存等。对于每个服务器s_j，在资源维度d上的资源容量都是异构的，记为$c_{j,d}$。在时间步t时，资源维度d上的可用资源容量表示为$a_{j,d}^t$。

对于每个工作负载w_i，记录它到达集群的时间t_i，并根据第三章的序列预测结果得知其预计运行时间n_i。相应的时变资源需求同样由序列预测产出，并提交给集群的调度器。调度器得到的资源需求序列可以表示为 $r_i=[r_i^1,\ldots,r_i^l,\ldots,r_i^{n_i}]^T$，其中$r_i^l=[r_{i,1}^l,\ldots,r_{i,d}^l,\ldots,r_{i,D}^l]$，表示工作负载$w_i$在其第$l$ 个运行时间步所需的跨D类资源的容量。

为了有效地调度时变工作负载，调度器不应只关注单个调度决策对集群的直接影响。相反，它应该通过在整个服务提供期间做出一系列有效的调度决策，努力在以下指标上优化集群的整体性能[77]：

①最小化所使用的服务器数。如果工作负载能够在较少数量的服务器上有效运行，那么将大大减少集群总体开支，因为待机状态下的服务器可以被关闭或转换为低功耗休眠模式。在整个观测期间集群使用的平均服务器数$\mathcal{AN}$定义为：

$$\mathcal{AN}=\frac{\sum_{t=1}^{T}B(t)}{T}, \tag{4.1}$$

其中，$B(t)$表示在时间步t时集群中正在被使用的服务器数，T是观测周期的长度。

②最大化资源利用率。通过有效的工作负载调度来优化集群内资源的利用，对减少 CSPs 的成本和运营支出起着关键作用。用于衡量这种优化的指标$\mathcal{AU}$表示为：

$$\mathcal{AU}=\sum_{d=1}^{D}\frac{\sum_{t=1}^{T}\sum_{j\in B(t)}(c_{j,d}-a_{j,d}^t)}{T\times max_t B(t)\times c_{j,d}}. \tag{4.2}$$

③最小化资源碎片化率。当未使用的资源分散在服务器之间，并且没有被分配给新的工作负载时，就会导致集群效率低下和资源浪费。平均资源碎片化率$\mathcal{AF}$度量了集群中集中的未使用资源的比例：

$$\mathcal{AF}=\sum_{d=1}^{D}(1-\sum_{t=1}^{T}\frac{max_{j\in B(t)}a_{j,d}^t}{T\times\sum_{j\in B(t)}a_{j,d}^t}). \tag{4.3}$$

④**最小化工作负载等待时间。**在强调集群性能的重要性的同时，绝对不能忽视与工作负载相关的 QoS 度量，例如等待时间，特指工作负载的调度延迟。总等待时间的度量$\mathcal{AW}$可以作为集群管理高并发传入工作负载的能力的指标：

$$\mathcal{AW}=\sum_{t=1}^{T}\sum_{w_i\in E(t)}(t-t_i), \tag{4.4}$$

其中，$E(t)$表示在时间步t时集群中等待调度的工作负载的集合。

综上所述，时变工作负载调度的总体优化目标可以简明地表述为：最大化$\mathcal{AU}$的同时最小化$\mathcal{AN}$、$\mathcal{AF}$和$\mathcal{AW}$。

4.2 时变工作负载调度算法设计

为了优化调度的总体目标，本文提出了一种新的基于优先级调控与资源感知的时变工作负载调度框架 PRAS。首先，通过第三章的混合专家预测构建完整的时变工作负载序列。随后，将得到的工作负载映射到不同的等价类，并使用标签表示其特征。其次，通过为每个工作负载分配分数，本文构建了一个优先级队列，该队列利用了不同工作负载类之间资源使用的时变特征。然后，本文构建了资源感知 GAT 层，以捕获服务器内部固有的维度特征和时间动态。最后，本文设计了一种基于 DRL 的调度策略来适应动态环境条件。PRAS 的完整工作流程如图 4.2 所示。接下来将详细说明框架内的主要部件。

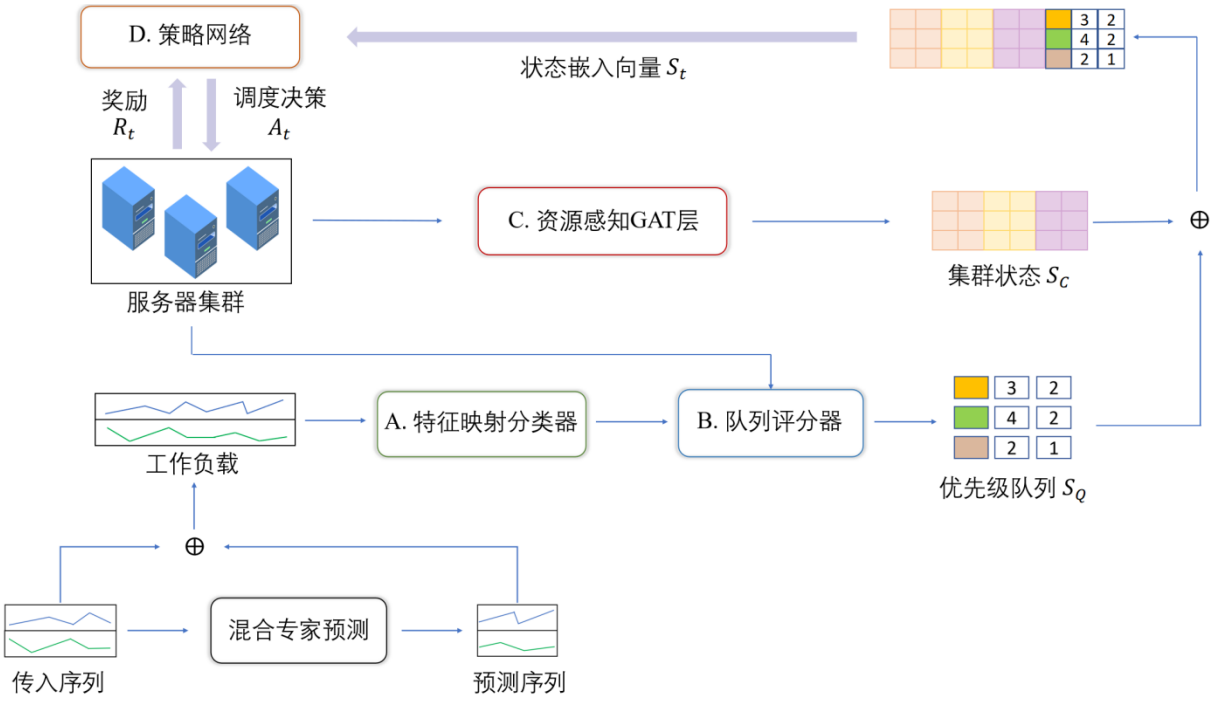

图 4.2 PRAS 的工作流程

Fig. 4.2 Workflow of PRAS

4.2.1 特征映射分类器

由于时变工作负载的多样性和动态性，对每个工作负载的特征进行建模是不切实际的。因此，本文设计了一个特征映射分类器，将传入的工作负载映射到不同的等价类，帮助调度器根据生成的类标签来区分和学习不同工作负载之间的资源使用特征。

为了离线训练特征映射分类器，本文从集群的历史轨迹中收集了一批工作负载。然后，本文利用 DTW 算法来计算工作负载间的距离度量。工作负载w_i和w_g之间的距离度量的计算方法为：

$$\mathcal{D}(w_i, w_g) = \mathcal{D}_{\min}(r_i^{n_i}, r_g^{n_g}). \tag{4.5}$$

该计算的递归公式为：

$$\mathcal{D}_{\min}\left(r_i^{l_i}, r_g^{l_g}\right) = \min\{\mathcal{D}_{\min}(r_i^{l_i}, r_g^{l_g-1}), \mathcal{D}_{\min}(r_i^{l_i-1}, r_g^{l_g}), \mathcal{D}_{\min}(r_i^{l_i-1}, r_g^{l_g-1})\} + \mathcal{M}(r_i^{l_i}, r_g^{l_g}), \tag{4.6}$$

其中，$\mathcal{M}(r_i^{l_i}, r_g^{l_g}) = \sum_{d=1}^{D} ||r_{i,d}^{l_i} - r_{g,d}^{l_g}||^2$。

最后，本文应用 DBSCAN 聚类算法来对这批工作负载进行聚类，生成C个等价类。在建立等价类之后，所有的工作负载及其各自的类标签都被用作分类器f_c的训练数据。

在在线强化学习阶段，特征映射分类器f_c接收到一个新的工作负载，并分配一个标签c_i来生成工作负载的精细表示，如下：

$$\overline{r_i} = [\vec{r}_i^1, \ldots, \vec{r}_i^l, \ldots, \vec{r}_i^{n_i}]^T, \tag{4.7}$$

其中，$\vec{r}_i^l = [\vec{r}_{i,1}^l, \ldots, \vec{r}_{i,d}^l, \ldots, \vec{r}_{i,D}^l, c_i]$。

4.2.2 队列评分器

传统的工作负载调度方法通常依赖于先到先得机制或预设的优先级，这忽略了服务器集群内资源利用的当前状态。为了解决这一不足，本文设计了一种新的优先级队列机制。

在每个时间步t中，队列评分器评估未调度的工作负载，并根据四个关键参数对队列中的每个工作负载进行排序：集群内所有服务器上可用的主体资源、工作负载所需的主体资源、工作负载被分配的类和已等待时间。工作负载w_i的分数q_i计算为：

$$q_i = \alpha_1 \mathbb{I}(\max_d \sum_{j \in B(t)} a_{j,d}^t \cdot \max_d r_{i,d}^1) - \alpha_2 N_{c_i} + \alpha_3 (t - t_i). \tag{4.8}$$

在等式(4.8)中的第一项表示集群内空闲资源数量最高的资源维度d是否匹配工作负载所需的主体资源。如果匹配上，则指示函数$\mathbb{I}()$返回值 1，否则返回 0。变量N_{c_i}表示当前在集群中执行的属于等价类c_i的工作负载的数量。最终选择得分最高的前K个工作负载来形成优先级队列状态矩阵$S_Q \in Z^{K \times (1+D)}$，其中，第一列是所分配的标签，随后的$D$列表示得分最高的工作负载的所需资源。

如图 4.3 所示，在一个特定的时间步$t = 7$，假设集群内有三个工作负载w_1、w_2和w_3在等待调度，并且已知它们各自的跨$D = 3$ 个资源维度的时变资源需求。首先，将这三个工作负载输入到一个训练过的特征映射分类器中，以生成它们各自的类标签，并得到它们的精细表示。随后，将标签信息与工作负载的主体资源（即特定工作负载中比其他资源需求最多的资源类型）集成到队列评分器中。然后，队列评分器从集群中获取信息，包括集群的主体资源、当前时间步以及与三个输入工作负载属于同一类的正在集群中运行的工作负载的数量。

队列评分器执行计算，并根据每个工作负载的分数构建优先级队列。对于在时间步$\mathrm{t}_1 = 2$到达集群的工作负载w_1，它的主体资源与集群的主体资源不匹配，因此它从

第一部分得到的分数为 0。由于w_1属于等价类 3，并且集群中当前运行着 4 个等价类 3 的工作负载，因此它在第二部分的评分为-1.6。它的等待时间是$7-2=5$，导致第三部分的得分为 1.5 分。w_1的最终分数是$0-1.6+1.5=-0.1$。其它两个工作负载的分数同样由此计算，并得到最终的优先级队列排序为$w_2 > w_1 > w_3$。队列评分器使用每个工作负载的类标签，并按其分数的降序排列（注意，标签 0 表示没有标签，这意味着优先级队列尚未被工作负载填满）。最后，将最高优先级工作负载w_2的资源请求序列与类标签连接起来，形成优先级队列状态矩阵S_Q。

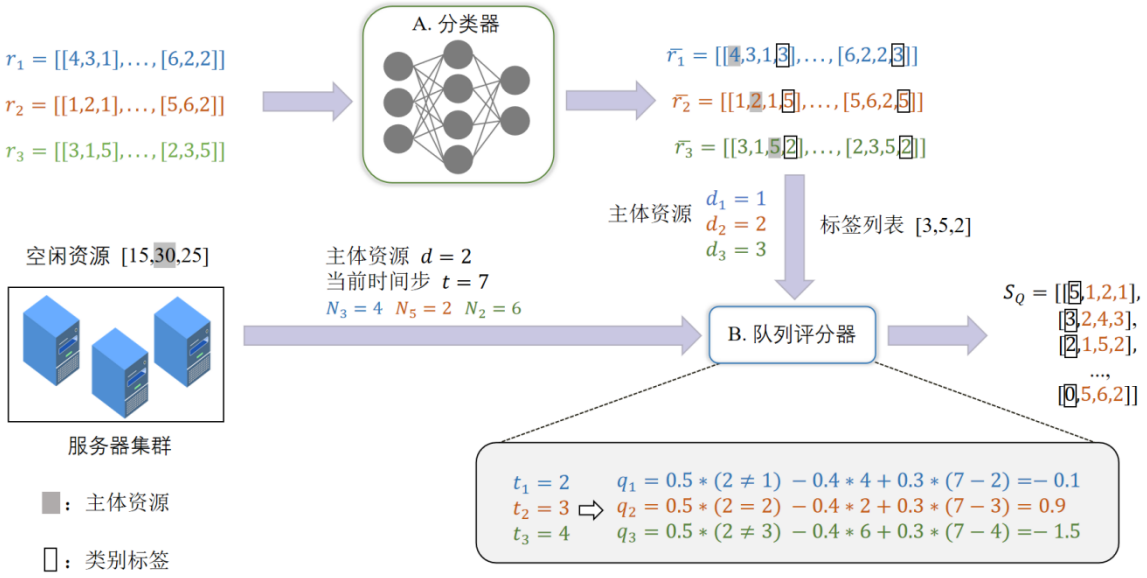

图 4.3 优先级队列构建过程的一个示例

Fig. 4.3 An example of the priority queue construction process

4.2.3 资源感知 GAT 层

为了有效地建模服务器之间的维度和时间特征，本文设计了资源感知 GAT 层，并将每个服务器的特征作为图中的一个节点来处理。如图 4.4 所示，资源感知 GAT 层主要由两个并行的 GAT 层组成，其中面向维度的 GAT 层旨在不需要先验知识的情况下捕获服务器之间资源的维度特征，而面向时间的 GAT 层侧重于揭示时间序列中的时间依赖性，以更好地理解服务器状态的演化。该组件的工作流程如下。

首先，利用一维卷积从每个服务器s_j的资源使用时间序列中提取高级特征v_j：

$$v_j = conv(o_j), \tag{4.9}$$

其中，$o_j = M \times \sum_{d=1}^{D} c_{j,d}$，$M$是服务器可以读取的历史时间步长，水平像素（$\sum_{d=1}^{D} c_{j,d}$）表示服务器$s_j$内的资源使用状态。图 4.4 表明标记有颜色的像素被分配给有等价类的工作负载使用，而没有被标记则表示资源仍然空闲。

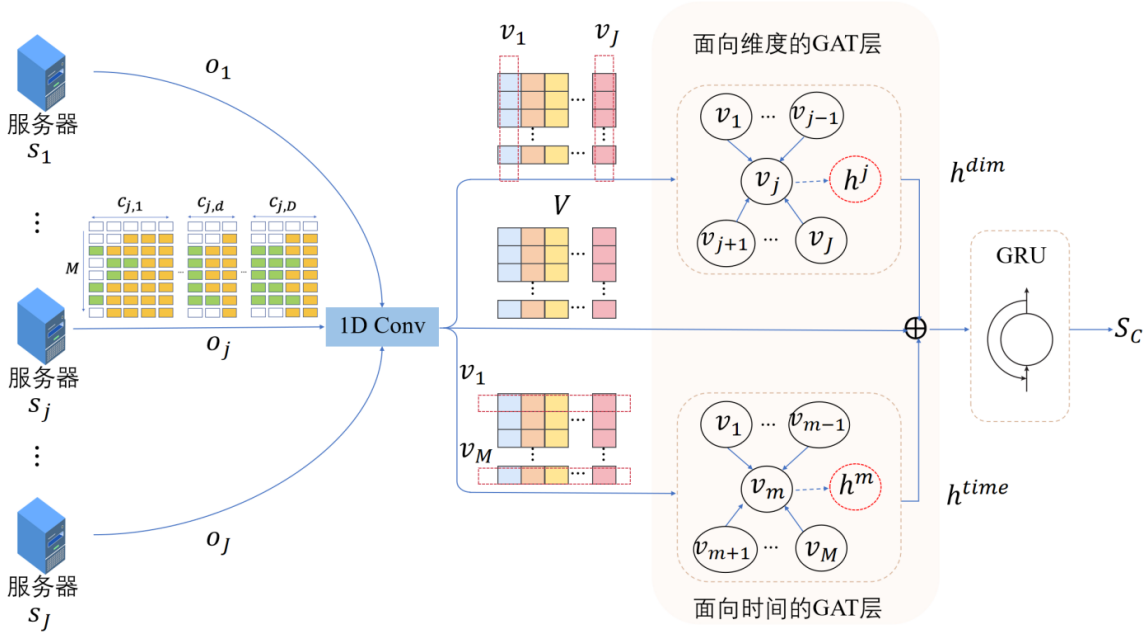

图 4.4 资源感知 GAT 层的构建

Fig. 4.4 Construction of resource awareness GAT layer

然后V由两个 GAT 层处理，面向维度的 GAT 层将其视为一个完整的图，其中每个节点代表一个单独的服务器特征v_j，每条边表示对应服务器之间的关系。这样，通过图注意力操作，可以仔细地捕获相邻节点之间的关系。具体来说，每个节点用一个顺序向量$v_j = [v_{j,1},\ldots,v_{j,m},\ldots,v_{j,M}]$表示，共$J$个节点。由此得到的维度图注意力状态为：

$$h^{dim} = [h^1,\ldots,h^j,\ldots,h^J], \tag{4.10}$$

$$h^j = \sigma(\sum_{g\in\mathcal{N}_j} \alpha_{jg} v_g), \tag{4.11}$$

其中，σ表示 Sigmoid 激活函数；α_{jg}是度量节点g对节点j贡献的注意力分数，$\mathcal{N}_j$表示节点j的相邻节点。注意力评分α_{jg}可计算为：

$$e_{jg} = \text{LeakeyReLU}(w^{\mathrm{T}} \cdot (v_j \oplus v_g)), \tag{4.12}$$

$$\alpha_{jg} = \frac{\exp(e_{jg})}{\sum_{l\in\mathcal{N}_j}\exp(e_{jl})}. \tag{4.13}$$

同时，利用面向时间的 GAT 层来捕获时间序列中的时间依赖关系。具体来说，一个节点$v_m = [v_{m,1},\ldots,v_{m,j},\ldots,v_{m,J}]$表示时间步$m$处的一个特征向量，其相邻节点包含当前滑动窗口内的所有其他时间步。所有的节点都是通过一个完全连接的注意力操作来建模的：

$$h^{time} = [h^1,\ldots,h^m,\ldots,h^M], \tag{4.14}$$

$$h^m = \sigma(\sum_{p\in\mathcal{N}_m} \alpha_{mp} v_p), \tag{4.15}$$

其中，α_{mp}的计算类似于面向维度的 GAT 层。

最后，将来自两个并行的 GAT 层的输出连接起来，得到一个$M \times 3J$的形状，其中每一行代表每个时间步的一个$3j$维特征向量。然后，该融合向量被输入一个门控循环单元（Gated Recurrent Unit, GRU）层，该层被设计用来捕获主要的序列模式。GRU 层的输出表示服务器集群的隐藏状态S_C：

$$V = [v_1, \ldots, v_j, \ldots, v_J], \tag{4.16}$$

$$S_C = GRU(V \oplus h^{dim} \oplus h^{time}). \tag{4.17}$$

4.2.4 策略网络

PRAS 将调度策略集成到一个策略网络中，作为一个代理通过观察和反馈机制与环境交互。具体来说，代理在时间步t时观察到的状态表示为S_t，随后输入到策略网络中。然后，策略网络选择相应的动作A_t。在动作执行完成后，环境过渡到随后的状态$S_{\bar{t}}$，观察到奖励R_t并作为反馈提供给代理。代理利用这种奖励来训练和优化策略网络，从而不断地细化其调度策略。此基于 DRL 的设计中所涉及的元素如下。

（1）状态空间。将集群状态S_C和优先级队列的状态S_Q连接起来，形成在时间步t时的最终状态S_t，如下所示：

$$S_t = S_C \oplus S_Q. \tag{4.18}$$

（2）动作空间。在每个时间步t时，调度器将优先级队列中的第一个工作负载分配给相应的服务器，动作空间表示为$\{\emptyset, 1, \ldots, J\}$，其中$A_t = j$ 表示将工作负载分配给第j个服务器；$A_t = \emptyset$ 表示“无效”操作，代表调度器拒绝在当前时间步调度工作负载。

（3）奖励设计。本文使用以下惩罚作为负奖励来训练代理：

①资源争用惩罚。为了避免代理训练的策略使分配在同一服务器上的，各工作负载在资源利用上同时激增，本文采用了一个资源争用惩罚P_C：

$$P_C = \sum_{d=1}^{D} \sum_{j=1}^{J} K_c * C_r(s_j, d). \tag{4.19}$$

具体来说，$C_r(s_j, d)$表示跨资源维度d的与服务器s_j相关的争用分数，其计算方法为：

$$C_r(s_j, d) = \sum_{w_i \in s_{j,W}} \sum_{w_g \in s_{j,W}, g>i} < r_{i,d}, r_{g,d} >, \tag{4.20}$$

其中，$s_{j,W}$表示服务器s_j上运行的工作负载集，而$r_{i,d}$是表示服务器s_j上的工作负载w_i在维度d上资源使用的时间序列。K_c是一个决定资源争用惩罚的权重的常数。资源争用惩罚放大了工作负载之间资源使用同时出现峰值的影响，从而减轻了在同一服务器上运行的工作负载之间的资源竞争，同时有效地减少了服务器资源碎片化。

②利用不足惩罚。由于本文的目标是通过促进调度器在最小数量的服务器上优化工作负载分布来提高集群的整体资源利用率，因此设计了一个利用不足惩罚，称为P_U。

此惩罚与集群内正在运行的服务器中未使用资源的数量成正比，如果服务器容纳了至少一个工作负载，则被视为正在运行的。在形式上，利用不足惩罚P_U表示如下：

$$P_U = \sum_{d} \sum_{j \in B(t)} |a_{j,d}^{t}|^{K_u}, \tag{4.21}$$

其中，常数项K_u作为尺度系数，在这里表示为指数。通过利用不足惩罚，调度器可以更有效地利用服务器资源，并减少集群中所需的服务器数量。

③等待时间惩罚。为了阻止调度器长时间保留调度请求来产生更优越的调度决策，本文引入一个等待时间惩罚P_W，它由集群内等待调度的工作负载总数$|E\ (t)|$，和一个常数K_w的乘积得到，计算如下：

$$P_W = K_w * |E\ (t)|. \tag{4.22}$$

最终，奖励R_t可以表示为：

$$R_t = -P_C - P_U - P_W. \tag{4.23}$$

4.2.5 在线调度和训练算法

在前面的几节中，已经详细介绍了每个组件的具体实现和应用。在这里，本文在算法 4.1 中总结了整个方法流程。它封装了上述四个组件的主要操作和交互，为本文提出的方法 PRAS 提供了一个统一的框架。

算法 4.1 中每一个 episode 的复杂度为$O(T \times (W + WlogW + J^2 + W^2 + J))$。其中，$O\ (W)$考虑了与等价类划分和传入工作负载的队列评分相关的复杂度。$O\ (WlogW)$和$O\ (J^2)$分别表示构建优先级队列和资源感知 GAT 层的复杂度，因为在资源感知 GAT 层中的每个节点都与图中的所有其他节点相连。$O\ (W^2 + J)$表示计算集群产生的奖励的复杂度，对应于资源争用和利用不足惩罚，而等待时间惩罚的复杂度$O\ (W)$可以由$O\ (W^2)$包含。

在算法 4.1 中也描述了策略网络的训练过程。在每次迭代中，第 40 行表示记录一个轨迹，包括每一个 episode e中所有时间步的状态、动作和奖励。第 42 行表示利用这些轨迹在每个时间步t处使用折扣因子γ 计算折扣累积奖励。在完成所有 episode 后，使用带有基线的 REINFORCE 算法来训练策略网络。在计算基线的过程中，如第 45 行和第 46 行所述，每一个 episode e都显示出独特的运行时间步数T_e。这些时间步数表示所有工作负载都已完全执行的时间点，并用T_{max}来表示最大的T_e，并以 0 填充所有比T_{max}短的序列$y^e = [y_1^e, \ldots, y_{T_e}^e]$。随后，在第 47 行和第 48 行的梯度计算过程中，从y_t^e的原始长度中减去相应步长的基线z_t。值得注意的是，资源感知 GAT 层和策略网络是以端到端的方式配置的。这种配置允许通过反向传播产生的梯度更新这两个组件的网络权重，如第 51 行所示。

算法 4.1：在线调度和训练

输入：集群$\{s_1,\ldots,s_j,\ldots,s_J\}$；
传入的工作负载集 W；

输出：调度决策；

初始化：策略网络参数θ；
初始化环境；
/**决策 Episode**/ （episode $e=1$）
for $t=1$ **to** T **do:**
　调度标志 ← **True**；
　for $w_i \in$ 时间步t传入的工作负载集 **do:**
　　$c_i \leftarrow f_c(w_i.r)$；
　　$w_i.\bar{r} \leftarrow w_i.r \oplus c_i$；
　　$E\ (t)$.append(w_i)；
　end for
　while 调度标志 **do:**
　　for $w_i \in E\ (t)$ **do:**
　　　$q_i \leftarrow$ 队列评分器$(a_{j,d}^t,\ w_i.\bar{r}, w_i.t_i)$；
　　end for
　　按分数的降序对$E\ (t)$进行排序；
　　$S_Q \leftarrow$ 优先级队列状态矩阵；
　　$\{o_1,\ldots,o_J\} \leftarrow$ 获取集群特征；
　　$S_C \leftarrow$ 资源感知 GAT 层$(o_1,\ldots,o_J)$；
　　$S_t \leftarrow S_C \oplus S_Q$；
　　$A_t \leftarrow$ 策略网络由状态S_t产生动作；
　　$j \leftarrow A_t$；
　　if j != $\emptyset$ **do:**
　　　$w_{curr} \leftarrow E\ (t)$.pop(0);
　　　调度决策中加入二元组(w_{curr}, s_j)；
　　　将工作负载w_{curr}调度到服务器s_j中；
　　　$S_{\bar{t}} \leftarrow$ 获取下一个状态；
　　　$R_t \leftarrow$ 利用式（4.23）由状态$S_{\bar{t}}$计算得到奖励；
　　　轨迹 1 中加入三元组$(S_t,\ A_t,\ R_t)$；
　　end if
　　else do:

算法 4.1：在线调度和训练（续）

调度标志 ← **False**；
end else
end while
end for
/**模拟 Episodes**/
for 每一轮 **do:**
初始化环境；
$\Delta\theta \leftarrow 0$；
运行 episode $e = 2, \ldots, N$:
轨迹$e \leftarrow \{S_1^e, A_1^e, R_1^e, \ldots, S_{T_e}^e, A_{T_e}^e, R_{T_e}^e\}$；
for episode $e = 1, \ldots, N$ **do:**
计算累积奖励：$y_t^e = \sum_{x=t}^{T_e} \gamma^{x-t} R_x^e$；
end for
/**网络训练**/
for $t = 1$ **to** T_{max} **do:**
计算基线：$z_t = \frac{1}{N} \sum_{e=1}^{N} y_t^e$；
for $e = 1$ **to** N **do:**
$\Delta\theta \leftarrow \Delta\theta + \tau \nabla\theta \log_{\pi_\theta}(S_t^e, A_t^e)(y_t^e - z_t)$；
end for
end for
$\theta \leftarrow \theta + \Delta\theta$；
end for
return 调度决策；

4.3 实验评估

在本节中，本文使用真实数据集进行实验，对PRAS进行了评估，并与其它几种方法进行了比较，最后根据实验结果进行分析。

4.3.1 实验数据集

本文使用Google traces进行实验，它包含了跨越29天的工作负载调度记录。该数据集包括异构云服务器的资源容量和工作负载的时变资源需求，以及工作负载的到达和持续时间。本文从Google traces中随机抽取一组$J = 10$的异构服务器来形成集群。通过分析记录中已经在这些服务器上运行过的工作负载，本文提取了它们跨$D = 3$个资源维度（CPU、内存和磁盘）的，到达集群时可观测的时变资源需求，并根据第三章的

预测产生后续的时变资源需求以及这些需求持续存在的时间。预测后得到的信息用于构建工作负载轨迹。这些工作负载来自于选定服务器的运行历史记录，其中包括各种模式，如长时间运行的工作负载和吞吐量较大的工作负载。本文使用“事件类型”字段过滤了在Google traces的调度历史中被驱逐、杀死或失败的工作负载，以获得最终的轨迹。然后将这些轨迹输入到集群中进行调度。

4.3.2 实验设置

（1）参数设置。PRAS 中每个神经网络模块的参数配置如下：

①特征映射分类器。该模型由三个全连接层组成，隐藏层有 64 个神经元，输出层含有 ReLU 激活函数。它使用 Adam 优化器和分类交叉熵损失进行训练。

②资源感知 GAT 层。该模块的参数配置如表 4.1 所示。

表 4.1 资源感知 GAT 层参数

Table 4.1 Resource awareness GAT layer parameters

参数名	参数值
kernel_size	7
window_size	20
gru_n_layers	1
gru_hid_dim	150
dropout	0.2

其中，kernel_size 表示一维卷积中使用的卷积核的大小，window_size 表示面向维度和面向时间的 GAT 层的输入序列长度。gru_n_layers 和 gru_hid_dim 分别指 GRU 层中使用的层数和尺寸。dropout 同时应用于 GAT 层和 GRU 层，以防止过拟合。

③策略网络。本文采用了一个神经网络架构，包括一个含有 30 个神经元的单隐藏层，和一个神经元数与动作总数相同的输出层。输出层使用一个 softmax 激活函数来确保适当的动作选择。本文的优化策略采用了 Adam 优化器，训练利用 REINFORCE 算法且涉及了 10 个工作负载轨迹，并持续了 1000 次迭代。与训练相关的参数汇总见表 4.2。

（2）基线算法。本文通过与六种基线算法比较来评估所提出的方法 PRAS，这些算法包括三种启发式算法和三种基于 DRL 的算法。

① Best-fit[41]是一种启发式算法，它将工作负载的主体资源需求与具有该主体资源且可用主体资源数最少的服务器相匹配。在这种情况下主体资源是指，对于给定的工作负载其需求最高的资源维度，而对于服务器，它表示剩余资源最多的资源维度。

② Tetris [77]是一种启发式算法，它将工作负载调度到调度完成后剩余资源最少的服务器上。

表 4.2 策略网络参数

Table 4.2 Policy network parameters

变量	值	描述
K	20	优先级队列中的工作负载数
M	20	服务器的历史时间步长
K_c	0.1	资源争用惩罚权重
K_u	3	利用不足惩罚权重
K_w	50	等待时间惩罚权重
T	100	训练前的时间步长
N	10	episodes 的数目
γ	0.9	折扣因子
τ	0.001	学习率

③ SJF [42]是一种启发式算法，它选择需要最少执行时间的工作负载，将其调度到随机服务器上。

④ DeepRM + Best-fit[33]。DeepRM 是一种基于 DRL 的算法，它将服务器集群和工作负载状态表示为图像输入网络，该网络生成一个动作，用于在调度槽中选择工作负载。然而，由于 DeepRM 不支持直接对时变工作负载进行调度，因此本文按照 Mondal 等人[56]的描述对其进行了修改。因为由 DeepRM 生成的动作只指定了调度哪个工作负载，本文使用 Best-fit 算法来识别合适的服务器，然后调度工作负载。

⑤ RLQ [55]是一种基于 DRL 的调度算法，它通过先到先得的机制选择工作负载。它将服务器资源分配给不同类的多个工作者，其中同一类的工作者在所有维度中持有相同数量的资源。每个工作者都负责执行单个工作负载。

⑥ TVW-RL[56]是一种基于 DRL 的调度算法，它通过先到先得的机制选择工作负载。它通过将状态表示为像素级图像来学习时间资源使用模式。

4.3.3 实验结果与分析

所构建的轨迹中的工作负载动态地到达和离开集群，并遵循泊松过程。本文通过配置不同到达率来建立三种不同的集群负载场景： 30%、50%和 80%。对于每个轨迹，训练集和测试集分别由 1200 个和 360 个不同的工作负载组成，每个工作负载均通过第三章提出的 MOEP 方法预测得到完整序列，随后由本章提出的 PRAS 方法以及各基线算法进行调度。PRAS 的策略网络是在一台配备 18 个 CPU 核心、60GB 内存、1024GB SSD 和 RTX 4090D 显卡的计算机上训练的，运行在 windows11 上。该实验使用 PyTorch 版本 2.1.0 进行，以 Python 3.10 作为编程语言，而 PyCharm 作为集成的开发环境。本文使用以下指标[55]来评估算法的性能。

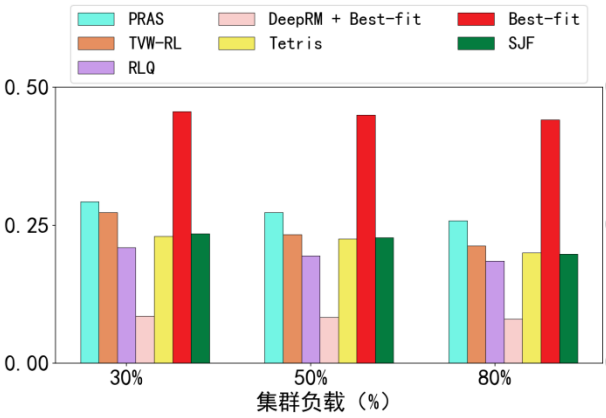

图 4.5 平均 CPU 利用率

Fig. 4.5 Average CPU utilization

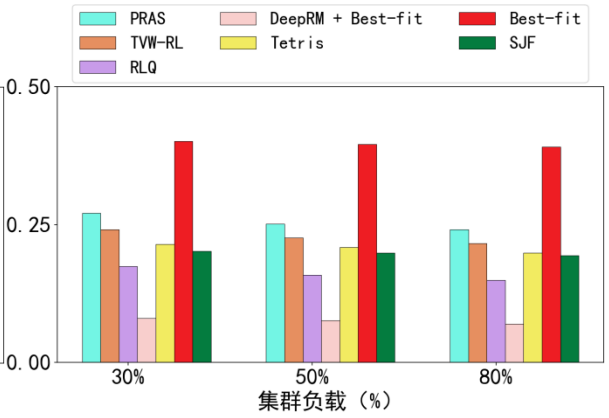

图 4.6 平均内存利用率

Fig. 4.6 Average memory utilization

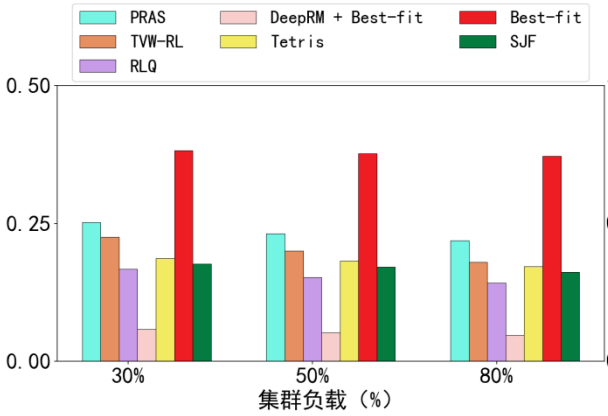

图 4.7 平均磁盘利用率

Fig. 4.7 Average disk utilization

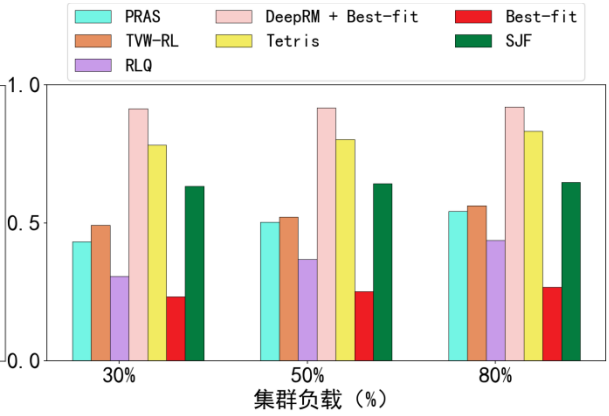

图 4.8 平均 CPU 碎片化率

Fig. 4.8 Average CPU fragmentation

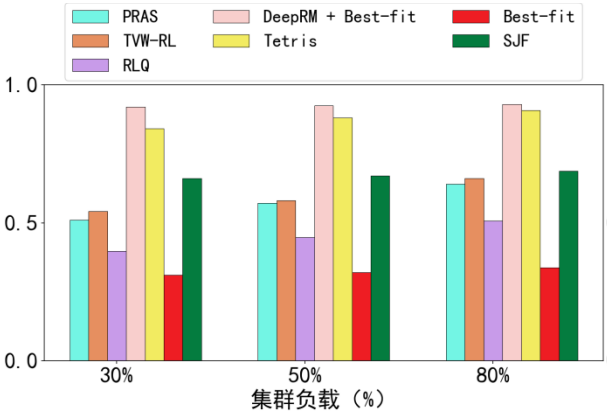

图 4.9 平均内存碎片化率

Fig. 4.9 Average memory fragmentation

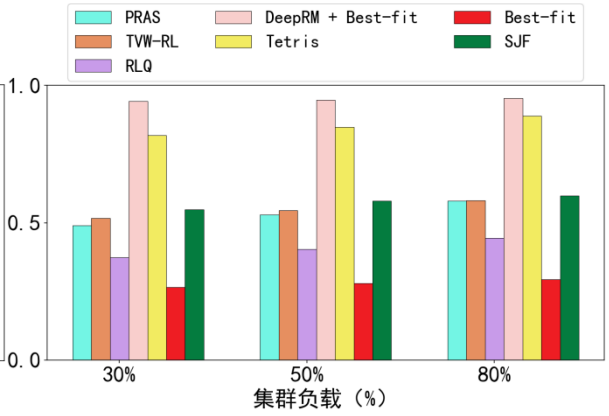

图 4.10 平均磁盘碎片化率

Fig. 4.10 Average disk fragmentation

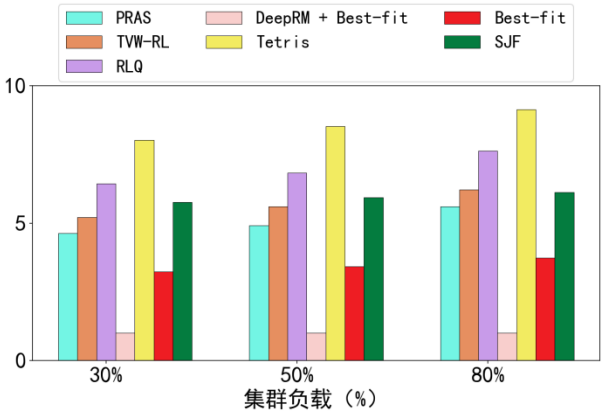

图 4.11 使用的服务器数

Fig. 4.11 Servers used

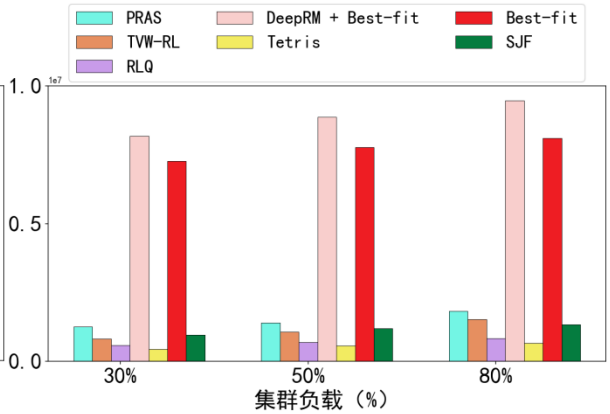

图 4.12 总等待时间

Fig. 4.12 Total waiting time

（1）资源利用率。图 4.5、图 4.6 和图 4.7 展示了 PRAS 在平均资源利用率方面与基线之间的比较。与 TVW-RL 相比，PRAS 平均 CPU 利用率提高了 8%到 21%，平均内存利用率提高了 7%到 12%，平均磁盘利用率提高了 12%到 18%。特别是在高集群负载下，CPU 利用率的性能提升非常显著。与 RLQ 相比，PRAS 在 CPU、内存和磁盘利用率方面分别提高了 32%到 37%、46%到 50%和 44%到 47%，在高负载条件下获得了提升更大。与 DeepRM + Best-fit 相比，PRAS 在 CPU、内存和磁盘利用率方面都有了显著的提高，分别从 213%到 224%、181%到 187%和 237%到 244%，这些提升在低负载条件下尤其明显。在不同的集群负载条件下，与 Tetris 和 SJF 相比，PRAS 的资源利用率分别提高了 20-28%和 21-34%。值得注意的是，虽然与 PRAS 相比，Best-fit 实现了更高的资源利用率，但它是以牺牲工作负载的等待时间过长为代价的。

（2）资源碎片化率。图 4.8、4.9 和 4.10 描述了算法的平均资源碎片化率。DeepRM + Bestfit 表现出过度的资源碎片化，导致集群总体资源利用率明显较低。相比之下，PRAS 比 TVW-RL 有了实质性的改进，平均 CPU 碎片化率减少了 7%到 15%，平均内存碎片化率减少了 3%到 6%，平均磁盘碎片化率减少了 2%到 7%。这些改进在低集群负载条件下尤其明显。此外，PRAS 在不同的集群负载场景中始终优于 Tetris 和 SJF，它分别降低了 33%到 45%和 6%到 37%的平均资源碎片化率。值得注意的是，RLQ 将服务器资源分配给工作者，然后由工作者执行工作负载。这种方法可能导致工作者级别的资源未得到充分利用，这就解释了其资源利用率较低的原因。但是，从总体集群资源的角度来看，使用更少的资源可能会留下更大的连续可用资源块。因此，RLQ 在资源碎片化率方面优于 PRAS 的性能。类似地，PRAS 在资源碎片化率方面落后于 Best-fit。

（3）使用的服务器数。图 4.11 描述了不同算法在不同集群负载条件下使用的服务器数量。由于 DeepRM+Best-fit 采用了单片服务器架构来处理工作负载，因此服务器数

量始终保持在1。此外，由于本场景不存在资源超调，Best-fit方法可能涉及等待特定的服务器资源可用于调度，从而导致使用的服务器数量相对较少。在剩下的5种方法中，PRAS 利用了最少的服务器数量，在资源利用率和减少资源碎片化率方面表现出了优越的性能。

（4）总等待时间。工作负载等待时间的统计数据如图 4.12 所示。很明显，与其他五种方法相比，DeepRM +Best-fit 和 Best-fit 表现出明显更大的等待时间。这是因为 Best-fit 消耗了过多的等待时间，用以等待特定的服务器资源可用于运行新的工作负载。这一特性导致该方法使用了最少数量的服务器，从而在资源利用率和碎片化率方面产生了异常有利的结果。但是，这种长时间的调度请求搁置无法满足用户的 QoS 目标，从而不利于服务器集群的长期运作。与其余四种方法相比，PRAS 显示了相对较高的等待时间。然而，随着集群负载的增加，这种差异逐渐减小。相比之下，Tetris 由于使用了更多的服务器，一直保持着非常低的等待时间。

4.3.4 消融实验

本文在 50%的集群负载下进行消融实验，且本文的所有后续实验都是在此配置下进行的。

（1）特征映射分类器的影响。从表 4.3 中可以看出，与删除分类器或队列评分器的情况相比，PRAS 在所有指标上都取得了最好的性能。具体来说，与没有分类器的场景相比，使用分类器使平均资源利用率提高了 26-50%，平均资源碎片化率减少了 15-24%，所需的服务器数量减少了 12%，总等待时间减少了 14%。这主要是因为在没有分类器的情况下，调度器只接收来自集群的状态信息和调度队列中的第一个工作负载的资源请求序列。如果无法访问等待调度的所有工作负载的特征，调度器就无法全面评估当前调度决策对集群的潜在影响。因此，由此产生的调度决策可能并不合适。

（2）队列评分器的影响。如表 4.3 所示，与没有队列评分器的场景相比，使用队列评分器能让多个性能指标取得显著改进。其中，平均资源利用率增加7-34%，平均资源碎片化率减少 12-20%，所需服务器数量减少 7%，总等待时间缩短 8%。这是因为先到先得的调度策略没有考虑到工作负载优先级、资源可用性或动态工作负载条件，所以经常导致次优资源利用率且增加资源碎片化。此时，调度器无法针对最关键的工作负载生成优化的调度决策，这会导致集群效率不佳。

（3）资源感知 GAT 层的影响。如表 4.3 所示，本文比较了在资源感知 GAT 层中使用并行 GAT 层，只使用一个 GAT 层，以及完全不使用 GAT 层对于调度的影响。与只使用一个 GAT 层相比，PRAS 提高了 29-54%的平均资源利用率，减少了 9-33%的平均资源碎片化率，减少了 3-11%的服务器使用数量，并减少了 9-13%的总等待时间。很明显，当只使用一个 GAT 层时，调度器不能提取集群的所有特征。由于缺乏对集群当前状态的理解，导致产生了次优调度决策。面向维度的GAT层和面向时间的GAT层在

表 4.3 消融实验

Table 4.3 Ablation study

算法	利用率(%)↑			碎片化率(%)↓			使用服务器数↓	总等待时间↓
	CPU	内存	磁盘	CPU	内存	磁盘		
w/o 分类器	18.16	16.73	18.41	66.24	66.95	63.30	5.58	1638805
w/o 队列评分器	21.76	18.72	21.66	62.75	65.22	60.03	5.27	1520875
w/o GAT	17.84	15.49	16.01	68.93	69.97	66.95	6.09	1750805
w/o 维度 GAT	18.63	16.31	17.19	65.19	62.74	62.46	5.06	1614360
w/o 时间 GAT	18.82	16.81	18.03	66.94	64.72	66.49	5.52	1537146
PRAS	**27.27**	**25.09**	**23.24**	**50.23**	**57.11**	**53.02**	**4.91**	**1404557**

这四个指标上都有自己的优势。面向维度的 GAT 层捕获了集群资源的维度特征，使其更倾向于有效地利用服务器资源。因此，它会导致更高的服务器使用数、资源利用率和资源碎片化率，但总等待时间相对较低。另一方面，面向时间的 GAT 层侧重于揭示时间序列中的时间依赖性，以更好地理解集群状态的演化。因此，它更加强调了服务器资源的时间序列对调度的影响。它更仔细地分配资源，减少碎片化以保留更大的碎片资源，确保能够满足新工作负载的需求。这将导致更多的总等待时间，但会降低服务器使用数、资源利用率和资源碎片化。与不使用 GAT 相比，PRAS 提高了 45-62%的平均资源利用率，减少了 18-27%的平均资源碎片化率，减少了 19%的服务器使用数量，同时还减少了 20%的总等待时间。很明显，如果不使用 GAT，调度器很难解释从集群中获得的状态信息。由此产生的调度决策很差，这导致了集群效率的降低。

4.3.5 参数灵敏度实验

（1）等价类数量。如表 4.4 所示，本文对特征映射分类器和资源感知 GAT 层进行了参数灵敏度实验，PRAS 在原始设置下在所有指标上都取得了最好的性能。在针对特征映射分类器的实验中，本文比较了 DBSCAN 算法的两个参数 ϵ 和 minPts 的不同值对调度决策的影响。在最初的设置中，使用$\epsilon = 0.5$和$\text{minPts} = 5$，最终的聚类结果是 24 个等价类。相比之下，当$\epsilon = 0.7$和$\text{minPts} = 10$时，集群产生了 5 个等价类。与只有 5 个等价类的情形相比，24 个等价类的情况有一定提升：平均资源利用率增加了 5-17%，平均资源碎片率减少了 8-15%，服务器数量减少了 5%，总等待时间减少了 5%。显然，DBSCAN 的参数设置对调度有显著的影响。当等价类数量较高时，可以更精细地提取不同工作负载的特征，使调度器获得更准确的信息，从而生成更优的调度决策。

表 4.4 参数灵敏度实验

Table 4.4 Sensitivity study

参数	值	利用率(%)↑			碎片化率(%)↓			使用服务器数↓	总等待时间↓
		CPU	内存	磁盘	CPU	内存	磁盘		
等价类数量	5	23.21	21.45	22.14	58.96	62.02	60.85	5.15	1480030
	24	**27.27**	**25.09**	**23.24**	**50.23**	**57.11**	**53.02**	**4.91**	**1404557**
GAT 层数	0	17.84	15.49	16.01	68.93	69.97	66.95	6.09	1750805
	1	18.82	16.81	18.03	65.19	62.74	62.46	5.06	1537146
	2	**27.27**	**25.09**	**23.24**	**50.23**	**57.11**	**53.02**	**4.91**	**1404557**
服务器数量	4	37.50	31.65	34.52	41.58	45.34	47.95	3.25	2625564
	6	28.64	24.85	27.15	57.66	61.31	60.71	3.95	2018310
	8	27.67	25.43	26.31	53.32	56.67	55.36	4.18	1719733
	10	**27.27**	**25.09**	**23.24**	**50.23**	**57.11**	**53.02**	**4.91**	**1404557**

（2）GAT 层数。如表 4.4 所示，当 GAT 层数为 0 时，调度器无法完全了解集群的状态，导致资源利用率降低，更高的碎片化率，使用更多的服务器，以及更长的总等待时间。与使用 GAT层相比，它的性能明显更差。当使用单一的 GAT层时，调度器可以捕获集群的一些特征，但它对集群状态的理解仍然有限。这些指标仍然落后于使用两个 GAT 层来调度的指标。在这个阶段，GAT 层在一定程度上有助于改进调度决策，但它仍然不足以达到最佳性能。当使用两个 GAT 层时，调度器可以提取更多的集群特征，捕获更深的资源和时间依赖性，这显著提高了调度效率，并进一步优化了资源利用率、碎片化率、服务器使用数和总等待时间。特别是在复杂的集群条件下，两层 GAT 能够更好地理解和调整资源分配，实现最佳的调度性能。

（3）服务器数量。表 4.4 显示了调度器如何在不同数量的服务器上执行调度。可以看出，随着服务器数量的减少，平均资源利用率和总等待时间也相应增加。平均资源碎片化率最初会上升，但当服务器数量减少到 4 个时会急剧下降。与此同时，所使用的服务器数量也在稳步下降。这可以归因于服务器的可用性不足，等待调度的某些工作负载具有的资源需求是只有特定的服务器才能满足的。这导致一些服务器在持续的高负载下运行，因此当其资源未被释放时，调度过程受阻。这反映在总等待时间的显著增加上，从而导致了高资源利用率和低资源碎片化率。这种行为与 Best-fit 算法的行为一致，它们的调度过程变得严重依赖于特定的服务器。

4.3.6 序列预测对于调度的影响

表 4.5 序列预测对于调度的影响

Table 4.5 The impact of prediction on scheduling

对比实验	利用率(%)↑			碎片化率(%)↓			使用服务器数↓	总等待时间↓
	CPU	内存	磁盘	CPU	内存	磁盘		
不进行序列预测	22.98	21.83	18.47	57.26	66.25	59.38	5.84	1737605
已知序列信息	29.32	27.27	25.54	46.51	52.39	49.55	4.38	1242971
进行序列预测	**27.27**	**25.09**	**23.24**	**50.23**	**57.11**	**53.02**	**4.91**	**1404557**

如表 4.5 所示，为系统评估第三章提出的时变工作负载序列预测机制对调度的影响，本文设置了两个对比实验组：①基于可观测数据的静态调度策略（不引入预测机制）；②理想化的全信息调度基准（假设未来工作负载序列完全已知）。

实验结果表明：①全信息调度基准展现出最优性能，验证了完备时序信息对动态调度决策的重要支撑作用；②本文提出的预测驱动调度方案在未知未来序列的条件下，通过时变特征建模与序列预测，也在调度时取得了不错的效果，证实了预测机制对信息缺失的有效补偿；③静态调度策略由于对时变工作负载的信息认知不足，其指标下降比较明显，凸显了时序预测对调度的必要性。实验数据表明，在无法获取完整工作负载信息的现实场景下，本文方法通过序列预测有效缩小了与理想调度方案的性能差距。

4.3.7 复杂度比较

表 4.6 复杂度比较

Table 4.6 Complexity comparison

算法	#Params (M)	平均预测时间 (ms)
TVW-RL	0.15	1.48
RLQ	0.06	1.27
DeepRM + Best-fit	0.24	1.89
Tetris	/	2.44
Best-fit	/	2.23
SJF	/	1.86
w/o GAT	0.63	1.33
w/o 分类器	1.16	3.82
w/o 队列评分器	1.17	3.89
PRAS	**1.17**	**3.97**

如表 4.6 所示，PRAS 和其他算法都可以在毫秒级的决策时间内运行，这在实际的调度环境中是可以接受的。尽管与其他算法相比，PRAS 的决策时间略长，但它在资源利用率和碎片化率等关键性能指标方面优于其他算法。表 4.6 显示，GAT 是决策时间较长的主要原因。在未来的工作中，作者将继续探索更轻量级的特征提取方法来解决这一不足。

如表 4.6 所示，在真实的调度环境中，如果硬件基础设施能够支持 1.17M 的参数规模，并在 3.97 ms 范围内生成单个调度决策，那么 PRAS 就可以部署在相应的基础设施上。如果需要减少调度器的平均决策时间，可以选择更强大的 GPU，并在扩大集群大小时考虑硬件升级。

4.4 本章小结

在本章中，本文提出了一种新的基于优先级调控与资源感知的方法 PRAS 来调度异构服务器集群中的时变工作负载。具体来说，本文提出了一个预先训练的特征映射分类器和队列评分器的组合，为传入的工作负载生成优先级队列。然后，本文设计了资源感知 GAT 层来捕获服务器集群的维度和时间特征。此外，策略网络作为调度器，集成了来自优先级队列和集群的状态，以生成自适应的调度决策。在 Google traces 上进行的模拟实验表明，PRAS 优于现有的基线。

5 总结与展望

5.1 本文工作总结

在云计算环境下，实现对时变工作负载的高效调度已成为当前领域中至关重要且极具挑战性的问题。然而，由于云计算环境中任务需求存在较大波动、资源异构性强、负载模式复杂多变，传统的调度方法可能难以精准捕捉工作负载在时间与空间维度上的动态变化特征，导致资源利用率和服务质量难以兼顾。同时，工作负载的执行时间从毫秒级的实时计算到长周期的批量任务跨度极大，且资源需求在计算、存储等多个维度上存在波动，不同资源类型之间还存在复杂的非线性关联，给负载预测与调度带来极大挑战；另一方面，云计算平台的资源分配与调度机制本身复杂度较高，既要满足低延迟、高吞吐的实时需求，又要兼顾全局优化目标，使得现有方法在调度效率与系统性能方面的有效性受到限制。为了更好的解决时变工作负载调度问题，本文提出了基于动态模式学习与专家协同的时变工作负载预测方法 MOEP，以及基于优先级调控与资源感知的时变工作负载调度方法 PRAS。论文主要工作总结如下：

①基于动态模式学习与专家协同的时变工作负载预测方法：研究探索了使用基于动态模式学习与专家协同的方法 MOEP 来解决时变工作负载预测问题。具体如下：设计了一种融合流式模型、MOE 与时频域分析的深度学习框架，以精准建模工作负载的时变分布特征和周期性信息。模型首先通过流式模型利用可逆神经网络捕捉工作负载的序列模式，并构建距离度量空间以动态生成专家选择权重；随后，采用 MOE 依据生成的专家选择权重动态激活最优专家子集，以降低推理过程中的计算复杂度；最后，专家单元融合了时频域分析，每个专家模块由 FFT-LSTM 构成，通过时频域联合建模提取工作负载序列特征，并通过加权聚合各专家输出以生成最终的工作负载预测结果。

②基于优先级调控与资源感知的时变工作负载调度方法：研究探索了使用基于优先级调控与资源感知的方法 PRAS 来解决时变工作负载调度问题。具体如下：设计了一个优先级队列，该队列结合了各类工作负载的时间资源利用模式以及服务器集群的当前状态，对需调度的工作负载进行排序，以实现资源的高效分配；为了更准确地理解集群资源分布并预测集群状态的未来变化，采用了资源感知 GAT 层来提取服务器在维度和时间上的关系，动态捕捉集群的维度特征和时间变化；同时，提出了一种基于策略梯度的算法，旨在增强调度器在应对集群内不同工作负载到达模式时的适应性，并优化跨服务器的工作负载分配。通过与动态集群环境的持续交互，调度器探索各种决策并通过奖励信号优化其策略，以提升调度效率和系统性能。

5.2 未来工作展望

本文提出了一种针对云计算环境下时变工作负载调度问题的新方法，并通过实验验证了其有效性。然而，随着 CSPs 业务规模的不断扩展以及对系统可靠性要求的提升，仍然存在一些亟待解决的挑战。未来研究可从以下两个方向展开：

① 鲁棒性评估：当前提出的方法在面对故障时的适应性分析不足，尤其是在服务器停机、网络延迟、带宽波动等异常情况下的表现尚未得到充分评估。在实际云计算环境中，硬件故障、网络不稳定等是不可避免的因素，若调度策略缺乏鲁棒性，可能会导致任务执行失败、资源利用率降低，甚至影响服务质量。因此未来研究可引入容错机制，如基于副本的调度策略、动态负载迁移或预测性维护等 [78]，以增强调度策略的可靠性并提高系统的稳定性。

② 多云与多集群场景：本研究主要聚焦于单集群环境下的调度优化，而现实世界的云计算可能涉及多个云服务提供商或地理位置分布广泛的数据中心。在多云、多集群环境中，调度问题更为复杂，需要综合考虑集群间的负载均衡、不同云平台的定价策略以及数据传输成本等因素。未来研究可探索跨集群的协同调度方法，例如引入多集群分布式调度架构、基于边缘计算的分层调度机制，或结合联邦学习等技术 [80]，以实现更高效的全局资源管理。

参考文献

[1] Odun-Ayo I, Ananya M, Agono F, et al. Cloud computing architecture: A critical analysis[C]//2018 18th International Conference on Computational Science and Applications (ICCSA). IEEE, 2018: 1-7.

[2] Feng B, Ding Z, Jiang C. Heterogeneity-aware proactive elastic resource allocation for serverless applications[J]. IEEE Transactions on Services Computing, 2024, 17(5): 2473-2487.

[3] Stavrinides G L, Karatza H D. Scheduling a time-varying workload of multiple types of jobs on distributed resources[C]//2020 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS). IEEE, 2020: 1-6.

[4] Li J, Xiao D, Yao J, et al. Learning scheduling policies for co-located workloads in cloud datacenters[J]. IEEE Transactions on Cloud Computing, 2023, 11(4): 3725-3736.

[5] Kontopoulou V I, Panagopoulos A D, Kakkos I, et al. A review of ARIMA vs. machine learning approaches for time series forecasting in data driven networks[J]. Future Internet, 2023, 15(8): 255.

[6] Ajiono A, Hariguna T. Comparison of three time series forecasting methods on linear regression, exponential smoothing and weighted moving average[J]. International Journal of Informatics and Information Systems, 2023, 6(2): 89-102.

[7] Xie Y, Jin M, Zou Z, et al. Real-time prediction of docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing[J]. IEEE Transactions on Cloud Computing, 2020, 10(2): 1386-1401.

[8] Hong J, Yan Y, Kuruoglu E E, et al. Multivariate time series forecasting with GARCH models on graphs[J]. IEEE Transactions on Signal and Information Processing over Networks, 2023, 9: 557-568.

[9] Qiu Z, Lazar E, Nakata K. VaR and ES forecasting via recurrent neural network-based stateful models[J]. International Review of Financial Analysis, 2024, 92: 103102.

[10] Wang S, Chu Z, Sun Y, et al. Multiscale representation enhanced temporal flow fusion model for long-term workload forecasting[C]//Proceedings of the 33rd ACM International Conference on Information and Knowledge Management. 2024: 4948-4956.

[11] Karimunnisa S, Gopu A, Rao T P, et al. A novel workload forecasting model for cloud computing using ALAA-DBN algorithm[J]. Multimedia Tools and Applications, 2024: 1-25.

[12] Rossi A, Visentin A, Carraro D, et al. Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning[J]. Cluster Computing, 2025, 28(4): 258-277.

[13] Kim Y M, Song S, Koo B M, et al. Enhancing long-term cloud workload forecasting framework: Anomaly handling and ensemble learning in multivariate time series[J]. IEEE Transactions on Cloud Computing, 2024, 12(2): 789-799.

[14] Li J, Yao J, Xiao D, et al. Evogwp: Predicting long-term changes in cloud workloads using deep graph-evolution learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(3): 499-516.

[15] Maiyza A I, Korany N O, Banawan K, et al. VTGAN: hybrid generative adversarial networks for cloud workload prediction[J]. Journal of Cloud Computing, 2023, 12(1): 97-127.

[16] Gupta I, Saxena D, Singh A K, et al. A multiple controlled toffoli driven adaptive quantum neural network model for dynamic workload prediction in cloud environments[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024, 46(12): 7574-7588.

[17] Kaim A, Singh S, Patel Y S. Ensemble cnn attention-based bilstm deep learning architecture for multivariate cloud workload prediction[C]//Proceedings of the 24th International Conference on Distributed Computing and Networking. 2023: 342-348.

[18] Dogani J, Khunjush F, Mahmoudi M R, et al. Multivariate workload and resource prediction in cloud computing using CNN and GRU by attention mechanism[J]. The Journal of Supercomputing, 2023, 79(3): 3437-3470.

[19] Yuan H, Bi J, Li S, et al. An improved lstm-based prediction approach for resources and workload in large-scale data centers[J]. IEEE Internet of Things Journal, 2024, 11(12): 22816-22829.

[20] Zhao F, Lin W, Lin S, et al. TFEGRU: Time-Frequency Enhanced Gated Recurrent Unit With Attention for Cloud Workload Prediction[J]. IEEE Transactions on Services Computing, 2024, 18(1): 467-478.

[21] Bi J, Ma H, Yuan H, et al. Accurate prediction of workloads and resources with multi-head attention and hybrid LSTM for cloud data centers[J]. IEEE Transactions on Sustainable Computing, 2023, 8(3): 375-384.

[22] Zheng W, Chen Z, Zheng K, et al. WorkloadDiff: Conditional Denoising Diffusion Probabilistic Models for Cloud Workload Prediction[J]. IEEE Transactions on Cloud Computing, 2024, 12(4): 1291-1304.

[23] Zhao F, Lin W, Lin S, et al. MSCNet: Multi-Scale Network with Convolutions for Long-term Cloud Workload Prediction[J]. IEEE Transactions on Services Computing, 2025: 1-14.

[24] Zuo Z, Huang Y, Li Z, et al. Mixed contrastive transfer learning for few-shot workload prediction in the cloud[J]. Computing, 2025, 107(1): 5-27.

[25] Khallouli W, Huang J. Cluster resource scheduling in cloud computing: literature review and research challenges[J]. The Journal of Supercomputing, 2022, 78(5): 6898-6943.

[26] Wang K, Zhou Q, Guo S, et al. Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey[J]. IEEE Communications Surveys & Tutorials, 2018, 20(4): 3560-3580.

[27] Liu J, Liu P. The research of load imbalance based on Min-Min in grid[C]//2010 International Conference on Computer Design and Applications. IEEE, 2010, 4: V4-1-V4-4.

[28] Elzeki O M, Reshad M Z, Elsoud M A. Improved max-min algorithm in cloud computing[J]. International Journal of Computer Applications, 2012, 50(12): 22-27.

[29] Schwiegelshohn U, Yahyapour R. Analysis of first-come-first-serve parallel job scheduling[C]//SODA. 1998, 98: 629-638.

[30] Devi D C, Uthariaraj V R. Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks[J]. The Scientific World Journal, 2016, 2016(1): 3896065.

[31] Elcock J, Edward N. An efficient ACO-based algorithm for task scheduling in heterogeneous multiprocessing environments[J]. Array, 2023, 17: 100280.

[32] Duan H, Chen C, Min G, et al. Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems[J]. Future Generation Computer Systems, 2017, 74: 142-150.

[33] Mao H, Alizadeh M, Menache I, et al. Resource management with deep reinforcement learning[C]//Proceedings of the 15th ACM Workshop on Hot Topics in Networks. 2016: 50-56.

[34] Fan Y, Li B, Favorite D, et al. Dras: Deep reinforcement learning for cluster scheduling in high performance computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12): 4903-4917.

[35] Zhang D, Dai D, Xie B. Schedinspector: A batch job scheduling inspector using reinforcement learning[C]//Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing. 2022: 97-109.

[36] Xie R, Feng L, Tang Q, et al. Delay-prioritized and Reliable Task Scheduling with Long-term Load Balancing in Computing Power Networks[J]. IEEE Transactions on Services Computing, 2024, 17(6): 3359-3372.

[37] Zhu K, Zhang Z, Zeadally S, et al. Learning to optimize workflow scheduling for an edge‐cloud computing environment[J]. IEEE Transactions on Cloud Computing, 2024, 12(3): 897-912.

[38] Mangalampalli S, Karri G R, Kumar M, et al. DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing[J]. Multimedia Tools and Applications, 2024, 83(3): 8359-8387.

[39] Ali N N, Zeebaree S R M. Distributed resource management in cloud computing: a review of allocation, scheduling, and provisioning techniques[J]. The Indonesian Journal of Computer Science, 2024, 13(2): 1970-1994.

[40] Zangana H M, khalid Mohammed A, Zeebaree S R M. Systematic review of decentralized and collaborative computing models in cloud architectures for distributed edge computing[J]. Sistemasi: Jurnal Sistem Informasi, 2024, 13(4): 1501-1509.

[41] Berkey J O, Wang P Y. Two-dimensional finite bin-packing algorithms[J]. Journal of the Operational Research Society, 1987, 38(5): 423-429.

[42] Mondal R K, Nandi E, Sarddar D. Load balancing scheduling with shortest load first[J]. International Journal of Grid and Distributed Computing, 2015, 8(4): 171-178.

[43] Cheng L, Wang Y, Cheng F, et al. A deep reinforcement learning-based preemptive approach for cost-aware cloud job scheduling[J]. IEEE Transactions on Sustainable Computing, 2023, 9(3): 422-432.

[44] Xiu X, Li J, Long Y, et al. MRLCC: an adaptive cloud task scheduling method based on meta reinforcement learning[J]. Journal of Cloud Computing, 2023, 12(1): 75-86.

[45] Wei W, Gu H, Wang K, et al. Multi-dimensional resource allocation in distributed data centers using deep reinforcement learning[J]. IEEE Transactions on Network and Service Management, 2022, 20(2): 1817-1829.

[46] Fan Y, Ge J, Zhang S, et al. Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning[J]. IEEE Transactions on Mobile Computing, 2023, 23(4): 2765-2779.

[47] Tuli S, Casale G, Jennings N R. PreGAN+: semi-supervised fault prediction and preemptive migration in dynamic mobile edge environments[J]. IEEE Transactions on Mobile Computing, 2023, 23(6): 6881-6895.

[48] 曾磊, 白金明, 刘琦. 多群落粒子群优化供应链数据中心任务调度[J]. 应用科学学报, 2023, 41(3): 419-430.

[49] Li J, Xiao D, Yang D, et al. Integrated and Fungible Scheduling of Deep Learning Workloads Using Multi-Agent Reinforcement Learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 36(3): 391-406.

[50] Lu J, Yang J, Li S, et al. A2C-DRL: Dynamic scheduling for stochastic edge-cloud environments using A2C and deep reinforcement learning[J]. IEEE Internet of Things Journal, 2024, 11(9): 16915-16927.

[51] Yang Y, Shen H, Tian H. Scheduling workflow tasks with unknown task execution time by combining machine-learning and greedy-optimization[J]. IEEE Transactions on Services Computing, 2024, 17(3): 1181-1195.

[52] 施建锋, 陈忻阳, 李宝龙. 面向物联网的云边端协同计算中任务卸载与资源分配算法研究[J]. 电子与信息学报, 2024, 47: 1-12.

[53] 马玲, 樊漆亮, 许婷, 等. 基于强化学习的在线离线混部云环境下的调度框架[J]. 通信学报, 2023, 44(6): 90-102.

[54] 梁拓, 王利众, 许震. 基于 LSTM 和遗传算法的容器资源智能调度策略研究[J]. 计算机科学与应用, 2024, 14: 132-141.

[55] Staffolani A, Darvariu V A, Bellavista P, et al. RLQ: Workload allocation with reinforcement learning in distributed queues[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(3): 856-868.

[56] Mondal S S, Sheoran N, Mitra S. Scheduling of time-varying workloads using reinforcement learning[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2021, 35(10): 9000-9008.

[57] Islam M T, Karunasekera S, Buyya R. Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(7): 1695-1710.

[58] Zhao Y, Liu Y, Peng Y, et al. Multi-resource interleaving for deep learning training[C]//Proceedings of the ACM SIGCOMM 2022 Conference. 2022: 428-440.

[59] Li Y, Zeng T, Zhang X, et al. Tapfinger: Task placement and fine-grained resource allocation for edge machine learning[C]//IEEE INFOCOM 2023-IEEE Conference on Computer Communications. IEEE, 2023: 1-10.

[60] Chen X, Yang L, Chen Z, et al. Resource allocation with workload-time windows for cloud-based software services: a deep reinforcement learning approach[J]. IEEE Transactions on Cloud Computing, 2022, 11(2): 1871-1885.

[61] Jin Z, Tao D, Qi P, et al. An adaptive cloud resource quota scheme based on dynamic portraits and task-resource matching[J]. IEEE Transactions on Cloud Computing, 2024, 12(4): 996-1010.

[62] Galán-Sales F J, Reina-Jiménez P, Carranza-García M, et al. An approach to enhance time series forecasting by fast Fourier transform[C]//International Conference on Soft Computing Models in Industrial and Environmental Applications. 2023: 259-268.

[63] Wan Y, Hu Z, Zhang C, et al. LCBCTA: The Lowest Computation Burden Cooley-Tukey FFT Algorithm[C]//Proceedings of the 2023 7th International Conference on Electronic Information Technology and Computer Engineering. 2023: 867-872.

[64] Rezende D, Mohamed S. Variational inference with normalizing flows[C]//International Conference on Machine Learning. PMLR, 2015: 1530-1538.

[65] Jacobs R A, Jordan M I, Nowlan S J, et al. Adaptive mixtures of local experts[J]. Neural Computation, 1991, 3(1): 79-87.

[66] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.

[67] Bruna J, Zaremba W, Szlam A, et al. Spectral networks and deep locally connected networks on graphs[C]//International Conference on Learning Representations, ICLR, 2014.

[68] Veličković P, Cucurull G, Casanova A, et al. Graph Attention Networks[C]//International Conference on Learning Representations, ICLR, 2018.

[69] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Cambridge: MIT press, 1998.

[70] Singh H V, Girdhar A, Dahiya S. A Literature survey based on DBSCAN algorithms[C]//2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2022: 751-758.

[71] Bringmann K, Fischer N, van der Hoog I, et al. Dynamic dynamic time warping[C]//Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Society for Industrial and Applied Mathematics, 2024: 208-242.

[72] Jomah S, Aji S. Google cloud trace: characterization of terminated jobs[C]//2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT). IEEE, 2024: 517-522.

[73] Zeng A, Chen M, Zhang L, et al. Are transformers effective for time series forecasting?[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2023, 37(9): 11121-11128.

[74] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017, 30.

[75] Wu H, Hu T, Liu Y, et al. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis[C]//International Conference on Learning Representations, ICLR, 2023.

[76] Kingma D P, Welling M. Auto-Encoding Variational Bayes[J]. Stat, 2014, 1050: 1-14.

[77] Grandl R, Ananthanarayanan G, Kandula S, et al. Multi-resource packing for cluster schedulers[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 455-466.

[78] Park J, Kim D, Kim J, et al. Carbon-Aware and Fault-Tolerant Migration of Deep Learning Workloads in the Geo-Distributed Cloud[C]//2024 IEEE 17th International Conference on Cloud Computing (CLOUD). IEEE, 2024: 494-501.

[79] Zhu M, He F, Oki E. Optimization model for primary and backup resource allocation with workload-dependent failure probability[J]. IEEE Transactions on Network and Service Management, 2021, 19(1): 452-471.

[80] Yu L, Sun X, Albelaihi R, et al. Dynamic client clustering, bandwidth allocation, and workload optimization for semi-synchronous federated learning[J]. Electronics, 2024, 13(23): 4585.

[81] Narula M, Meena J, Vishwakarma D K. Federated Workload-Aware Quantized Framework for Secure Learning in Data-Sensitive Applications[J]. Future Generation Computer Systems, 2025, 168: 107772.

附 录

A. 作者在攻读硕士学位期间发表的论文目录

[1] **本人（第一作者），导师，…….** Time-Varying Workload Prediction via Dynamic Pattern Learning and Expert Collaboration[J]. IEEE Transactions on Cloud Computing.（在投，JCR-Q1，第三章内容）

[2] **本人（第一作者），导师，…….** DRL-Based Time-Varying Workload Scheduling with Priority and Resource Awareness[J]. IEEE Transactions on Network and Service Management.（已接收，JCR-Q1，第四章内容）

B. 作者在攻读硕士学位期间申请的专利目录

[1] **导师，本人（第二发明人），…….**，基于优先级和资源感知的时变工作负载调度方法。中国发明专利，专利申请号：202411010621.8.

[2] **导师，…….，本人（第七发明人），**一种基于多智能体强化学习的边缘缓存替换方法。中国发明专利，专利授权号：ZL202210513240.6.

C. 作者在攻读硕士学位期间参加的科研项目

[1] 2022.8-2025.3，重庆市自然科学基金面上项目“基于网络感知的服务功能链资源优化分配”，课题编号 CSTB2022NSCQ-MSX1104。（主要参与人）

[2] 2022.10-2024.9，重庆市技术创新与应用发展专项面上项目“面向智能制造供应链数据安全隐私保护机制研究与应用”，课题编号 CSTB2022TIAD-GPX0017。（主要参与人）

D. 学位论文数据集

<table>
<tr><td colspan="2">关键词</td><td colspan="2">密级</td><td>中图分类号</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>学位授予单位名称</td><td>学位授予单位代码</td><td colspan="2">学位类别</td><td>学位级别</td></tr>
<tr><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2">论文题名</td><td colspan="2">并列题名</td><td>论文语种</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>作者姓名</td><td colspan="2"></td><td>学号</td><td></td></tr>
<tr><td colspan="3">培养单位名称</td><td colspan="2">培养单位代码</td></tr>
<tr><td colspan="3"></td><td colspan="2"></td></tr>
<tr><td>学科专业</td><td>研究方向</td><td>学制</td><td colspan="2">学位授予年</td></tr>
<tr><td></td><td></td><td></td><td colspan="2"></td></tr>
<tr><td>论文提交日期</td><td></td><td>论文总页数</td><td colspan="2"></td></tr>
<tr><td>导师姓名</td><td></td><td>职称</td><td colspan="2"></td></tr>
<tr><td colspan="3">答辩委员会主席</td><td colspan="2"></td></tr>
<tr><td colspan="5">电子版论文提交格式
文本（） 图像（） 视频（） 音频（） 多媒体（） 其他（）</td></tr>
</table>

致　谢