杭州电子科技大学

硕 士 学 位 论 文

题　　目: 基于负载预测和强化学习的容器弹性伸缩策略研究

研 究 生 胡陈慧

专　　业 计算机技术

指导教师 林菲　教授

完成日期 2023 年 5 月

杭州电子科技大学硕士学位论文

基于负载预测和强化学习的容器弹性伸缩策略研究

研 究 生： 胡 陈 慧

指导教师： 林 菲 教授

2023 年 5 月

Dissertation Submitted to Hangzhou Dianzi University

for the Degree of Master

Research on Container Elastic Scaling Strategy Based on Load Forecasting and Reinforcement Learning

Candidate: Hu Chenhui

Supervisor: Prof. Lin fei

May, 2023

摘要

弹性伸缩是容器云平台的重要特性，旨在提高应用程序应对负载动态变化的能力。由于传统的响应式伸缩策略存在着弹性滞后以及配置复杂问题，更智能的主动式弹性伸缩吸引了众多学者的关注。主动式弹性伸缩策略通过主动预测未来的资源需求，对可能出现的情况做出预先布置，能够有效避免违反服务水平协议（Service Level Agreement，SLA），提高资源利用率。然而，精确预测未来负载并做出合理的弹性伸缩动作，是主动式弹性伸缩任务中的难点问题。针对这一问题，本文首先开展了负载预测模型研究，在此基础上，提出了一种基于负载预测和强化学习的容器混合弹性伸缩策略。主要研究内容及创新点如下：

（1）针对云负载序列时序变化模式多样导致的难以准确预测的问题，提出了基于多尺度空洞卷积和 LSTM 的负载预测模型。该模型利用平行和堆叠的空洞卷积残差网络构成多尺度特征提取块，从连续和非连续的输入时间步中提取时序特征，从而同时学习负载短期波动、持续性变化和周期性信息，再由 LSTM 从提取的特征中进一步学习长期时序依赖。在多个云负载数据集上进行了负载预测实验，得到的结果表明，本文提出的模型的预测精度优于对比模型。

（2）针对云环境下弹性伸缩的复杂性导致的难以做出合理弹性伸缩动作的问题，提出了基于负载预测和强化学习的混合弹性伸缩策略。该策略的主动式伸缩模块充分利用应用程序的当前负载状态和预测信息，由强化学习智能体做出对未来更合理的主动伸缩决策。同时，为了应对极端流量峰值场景，设计了结合 SLA 的响应式伸缩模块，能够根据实时服务质量进行及时响应和调整。最后，为了协调以上两种伸缩策略并结合两者的优点，设计了混合伸缩控制器，以兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

（3）为了验证所提出的弹性伸缩策略的有效性，在 Kubernetes 平台上实现了弹性伸缩系统。该系统的监控模块采用基于 Prometheus 的方案实现，能够全面采集伸缩指标数据并持久化存储；强化学习模块基于 OpenAI Gym 工具实现，能够便利地抽象 Kubernetes 环境并搭建强化学习模型；伸缩控制模块基于自定义控制器实现，能够有效扩展伸缩策略控制逻辑。为了模拟不同的负载场景，选取了两种具有不同特点的请求负载进行弹性伸缩对比实验。实验结果表明，相比于现有的方法，本文提出的伸缩策略能更有效地减少 SLA 违规，提高资源利用率。

关键词：云计算，弹性伸缩，负载预测，深度强化学习，Kubernetes

ABSTRACT

Elastic scaling is an important feature of container cloud platforms that aims to improve the ability of applications to cope with dynamic changes in load. As traditional reactive scaling strategies suffer from elastic lag as well as configuration complexity, smarter proactive elastic scaling has attracted the attention of many scholars. By proactively predicting future resource demands and making preemptive arrangements for possible scenarios, proactive elastic scaling strategies can effectively avoid Service Level Agreement (SLA) violations and improve resource utilization. However, accurately predicting future loads and making reasonable elastic scaling actions are difficult problems in the task of proactive elastic scaling. To address this problem, this thesis first conducts a load prediction model study, based on which, a hybrid elastic scaling strategy based on load prediction and reinforcement learning is proposed. The main research contents and innovation points are as follows:

(1) For the problem that it is difficult to predict accurately due to the diverse patterns of temporal changes in cloud load sequences, a load prediction model based on multi-scale dilated convolution and LSTM is proposed. The model uses parallel and stacked dilated convolution residual networks to form multi-scale feature extraction blocks to extract timing features from continuous and discontinuous input time steps, thus learning short-term load fluctuations, persistent changes and periodic information simultaneously, and then the LSTM further learns long-term timing dependencies from the extracted features. Load prediction experiments are conducted on several cloud load datasets, and the obtained results show that the prediction accuracy of the model proposed in this thesis is better than comparative models.

(2) For the problem that it is difficult to make reasonable elastic scaling actions due to the complexity of elastic scaling in cloud environments, a hybrid elastic scaling strategy based on load prediction and reinforcement learning is proposed. The proactive scaling module of this strategy makes full use of the current load state and prediction information of the application, and the reinforcement learning agent make more reasonable proactive scaling decisions for the future. Meanwhile, in order to cope with extreme traffic peak scenarios, a reactive scaling module combined with SLA is designed to be able to respond and adjust in time according to real-time quality of

service. Finally, in order to coordinate the above two scaling strategies and combine the advantages of both, a hybrid scaling controller is designed to balance proactive scaling decisions and effective scaling under extreme peak traffic.

(3) In order to verify the effectiveness of the proposed elastic scaling strategy, an elastic scaling system is implemented on the Kubernetes platform. The monitoring module of the system is based on the Prometheus solution, which can comprehensively collect the scaling metrics data and store them persistently; the reinforcement learning module is based on the OpenAI Gym tool, which can conveniently abstract the Kubernetes environment and build the reinforcement learning model; the scaling control module is based on a custom controller, which can effectively extend the scaling strategies control logic. In order to simulate different load scenarios, two kinds of request loads with different characteristics are selected for elastic scaling comparison experiments. The experimental results show that the scaling strategies proposed in this thesis can reduce SLA violations and improve resource utilization more effectively than the existing methods.

Keywords: Cloud Computing, Elastic Scaling, Load Prediction, Deep Reinforcement Learning, Kubernetes

目　录

第 1 章 绪论

1.1 研究背景及意义

云计算已成为当今互联网应用的主要支撑方式，以其灵活的资源管理、高度的可扩展性和经济性受到广泛关注[1,2]。其中，容器化技术以其轻量级、高效性、易于部署的特点被越来越多的应用所采用。而 Kubernetes 作为一种容器编排和管理平台，能够自动化地管理容器，实现部署、负载均衡和故障恢复等功能，为应用的开发和运维提供了高度的便利性。其中，弹性伸缩功能是 Kubernetes 的一项重要特性，能够根据实际的负载情况动态地调整应用的资源配额，保障应用的高可用性和稳定性[3]。

Kubernetes 中的弹性伸缩策略采用的是响应式伸缩的方法，其原理是通过内部伸缩控制器周期性地轮询监控应用的负载状态，并根据 CPU、内存等资源指标的阈值规则进行动态调整资源分配[4]。然而，这种基于阈值规则的响应式伸缩策略存在弹性滞后和指标设定困难的问题。弹性滞后问题是由于响应式伸缩策略的伸缩动作根据阈值规则触发，因此在应用程序出现资源瓶颈后才会进行扩容操作，而增加资源并部署新的应用程序需要一定的时间，导致应用的弹性滞后于负载状态的变化，进而影响服务质量。指标设定困难是指在不同的应用负载模式和场景下，设定合适的指标和阈值标准依赖于用户对应用的负载特性和性能需求的深刻理解。如果阈值设定过高，会导致应用出现资源瓶颈，无法满足用户需求；而阈值设定过低，则会导致资源浪费，从而降低整体资源利用率。

相比于响应式伸缩方式，主动式伸缩更加具有主动性和智能性，可以有效地避免违反服务水平协议（Service Level Agreement，SLA），并提高资源利用率，因此备受学术界的关注。这种伸缩方式利用历史负载数据建立预测模型，从而预测未来所需的资源，并基于预测值提前调整资源供给[5]。然而，云负载数据时序变化模式多样，呈现波动性、持续性和多周期性的特点，给精准预测未来负载带来了巨大挑战。同时，确定合理的伸缩动作需要考虑多种因素，其中包括负载波动情况、资源使用情况、成本和性能之间的平衡以及可靠性等因素，而不合理性的伸缩动作可能会导致系统性能下降或资源浪费。因此，精确预测未来负载并做出合理的弹性伸缩动作，是弹性伸缩任务中的难点问题。针对这一问题，研究优化 Kubernetes 的弹性伸缩策略，是保证容器云平台高效稳定运行、保障平台服务质量的关键，具有重要的现实意义。

1.2 国内外研究现状

本小节分别从时间序列负载预测和弹性伸缩技术两方面对国内外研究现状进行阐述，并总结了相关研究中存在的问题。

1.2.1 时间序列负载预测的研究现状

负载预测是容器云平台实现主动式弹性伸缩的基础。容器云平台利用精确的负载预测技术对未来的资源需求变化提前预知，能够快速、准确地调整资源供应，以提高资源利用率并降低成本。目前，云资源负载预测模型的研究可分为传统统计方法、机器学习方法、深度学习方法和混合模型方法四大类。

（1）**传统统计方法：**包括差分自回归移动平均法（Auto Regressive Integrated Moving Average，ARIMA）、线性回归（Linear Regression，LR）、指数平滑法等。Calheiros 等人[6]提出了一种基于 ARIMA 的云工作负载预测模型，可以动态地为预测的请求配置虚拟机资源，该模型对于季节性 Web 请求负载的预测精度达到了 91%，但它不适用于非季节性工作负载预测。Tang 等人[7]集成了 LR 和小波神经网络，用于短期负载预测，但该方法在长期预测中具有较高的误差。Xie 等人[8]提出了一种基于 ARIMA 和三指数平滑的混合预测模型，可以预测容器负载序列中的线性和非线性关系，并根据预测值自动进行 CPU 和内存资源使用的调度优化。虽然传统统计方法计算简单且可解释性强，但依赖于数据满足平稳性假设，因此难以满足对非平稳的云资源负载进行精准预测的需求。

（2）**机器学习方法：**包括支持向量回归（Support Vector Regression，SVR）、贝叶斯岭回归（Bayesian Ridge Regression，BRR）、人工神经网络（Artificial Neural Network，ANN）等。Liu 等人[9]提出了一种自适应负载预测方法，该方法首先根据负载特征将负载分成不同的类别，再自动分配给 LR 模型或 SVR 模型进行预测。Gao 等人[10]比较了 ARIMA、SVR 和 BRR 在预测谷歌集群主机 CPU 使用情况时的性能，得出 SVR 对大型数据集表现不佳，ARIMA 对突发数据表现不佳，BRR 对高波动的大型数据集表现最好。Borkowski 等人[11]提出的预测模型采用了 ANN 来预测任务持续时间和资源利用率，该模型与简单线性预测模型相比，产生的预测误差小于 20%。虽然基于机器学习的负载预测方法相对于传统统计方法具有更强大的学习能力，能够自主学习复杂的时间序列关系，但这种方法对于特征工程的要求相对较高，需要进行深入的特征选择和处理，以提高其预测准确性。

（3）**深度学习方法：**包括循环神经网络（Recurrent Neural Network，RNN）、长短期记忆网络（Long Short-Term Memory，LSTM）、一维卷积神经网络（One-

Dimensional Convolutional Neural Network，1D-CNN）和时间卷积网络（Temporal Convolutional Network，TCN）等。RNN 是专门为涉及顺序数据的任务而设计的，能够自适应地学习时间序列中的依赖关系，对于时序预测任务具有良好的适应性。Duggan 等人[12]使用 RNN 在 CloudSim 模拟的云环境中预测主机上虚拟机的 CPU 利用率，模型使用基于时间的反向传播算法来优化网络权重，相比于反向传播神经网络具有更好的学习效果和更快的收敛速度。

然而，由于 RNN 的训练过程中存在梯度消失和梯度爆炸的问题，导致难以学习长期的时序依赖。为了解决这个问题，学者们提出了 LSTM 作为 RNN 的改进模型，通过加入门控机制能够更好地学习长期时序依赖。Song 等人[13]将 LSTM 应用于负载预测任务中，使用单个 LSTM 层生成多个时间步长度的预测序列。随着深度学习的发展，还出现了一系列 LSTM 的变体模型，如门控循环单元（Gated Recurrent Unit，GRU）、双向 LSTM（Bidirectional LSTM，Bi-LSTM）、编码器-解码器结构的 LSTM（LSTM Encoder-Decoder，LSTM-ED）等，用于解决时序预测中的不同问题。文献[14,15]通过堆叠 GRU 层，用于预测谷歌集群主机的 CPU 使用率，由于与 LSTM 相比门控机制的数量更少，GRU 需要更少的参数来训练，并且具有良好的收敛特性[16]。Tang 等人[17]通过 Bi-LSTM 方法提出了一个容器负载预测模型，该模型使用容器过去的 CPU 利用率负载来预测未来的负载，Bi-LSTM 在计算每个时间步的隐藏状态时，同时考虑了过去和未来的信息，因此可以更好地捕捉序列中的长期依赖关系。Nguyen 等人[18]使用 LSTM-ED 对云主机负载进行预测，利用编码器 LSTM 将输入序列编码成一个固定维度的向量，解码器 LSTM 通过接收编码器 LSTM 的输出向量和前一个时间步的预测结果，一步一步生成多时间步预测结果。但该方法在生成多步预测时，每个预测步中的误差会逐步累积。

此外，1D-CNN 由于其在特征提取方面具有优势，也被广泛应用于时序预测任务中。Borovykh 等人[19]提出了一种时间序列的预测模型，该模型使用多层空洞卷积来从广泛的输入上下文中学习。Wang 等人[20]使用多个 1D-CNN 层来学习多元时间序列中的周期模式。TCN 是一种基于 CNN 的时序预测模型，其具有并行处理时序数据和长期依赖建模能力，有着更灵活的感受野可以从很长的输入历史中学习，在时序预测中取得了优异性能。文献[21,22]将 TCN 应用于云工作负载预测，利用因果空洞卷积扩大感受野，并通过堆叠不同尺度的卷积层在多个时间尺度上捕捉序列的不同特征。

随着 Transformer 在自然语言处理[23,24]和计算机视觉任务[25]中的创新，基于 Transformer 的模型也被应用于时间序列预测[26,27][26]。Transformer 模型利用了注意力机制的特性，因此它可以更好地捕捉序列中的全局依赖关系，从而在长时间

序列预测问题中表现更好。但是，由于 Transformer 局部信息获取能力不强以及计算空间复杂度大的缺点，目前在短期负载预测任务中应用较少。

（4）**混合模型方法：**该类方法将不同的预测模型组合在一起，以充分利用它们之间的优势。混合模型方法可以组合传统的统计模型、机器学习模型和深度学习模型中的两种或多种，通过综合不同模型的优势来提高预测精度。在混合模型中，1D-CNN 和基于 RNN 的模型结合的混合预测方法已被广泛应用于多个领域的预测任务中，如功率消耗[28,29]、空气污染[30]、飞机轨迹[31]、网络故障预测[32]等领域。这些混合模型都使用 CNN 作为特征提取层，再通过基于 RNN 的模型进一步学习时序特征[33]。Xu 等人[34]结合 1D-CNN 和 GRU 来预测 CPU 使用率，以有效地自动扩展云资源。该模型通过 1D-CNN 层从多元输入中提取特征，对局部时间序列数据与后续趋势之间的短期相关性进行建模，再由 GRU 对时间序列数据进行建模，以捕捉长期的时序依赖。Karim 等人[35]使用堆叠的 1D-CNN、LSTM 和 GRU 层来进行虚拟机 CPU 使用情况的预测。Al-Asaly 等人[36]提出了结合扩散卷积 RNN 和 GRU 的方法，用于对云工作负载中的时间依赖关系进行建模，以实现自适应资源配置。然而，这些方法仅具有单一卷积层，只能学习短期空间模式。Patel 等人[37]提出了一种并列 1D-CNN 层和 LSTM 的预测模型 pCNN-LSTM，该模型的 1D-CNN 层由三个平行的一维空洞 CNN 层组成，具有不同的膨胀率，使模型能够学习不同尺度的 CPU 负载变化，再由一个 LSTM 层学习原始负载值和 1D-CNN 层提取的模式中的时间依赖关系。然而，该模型由于受到最后一层有效感受野的限制，忽略输入序列中的全局信息可能导致无法捕获所有的模式。

综上所述，基于深度学习的预测方法在负载预测任务中非常有效，但需要根据任务的特点选择合适的模型，并且单一模型仍然有改进空间。针对云环境下负载数据时序变化模式多样，表现出波动性、持续性和多周期性的特点，1D-CNN 与基于 RNN 的模型相结合的混合模型方法能充分利用不同模型之间的优势，具有较好的应用前景。然而，现有的研究仅关注于某种负载变化特征，不能对云负载的复杂特点进行全面建模，在预测精度上仍有一定的欠缺。

1.2.2 弹性伸缩技术的研究现状

弹性伸缩是云计算平台的重要特性，它可以根据当前资源和负载情况自动调整应用程序的资源供应，在高负载时自动分配资源，保证服务质量；在低负载时自动释放资源，从而降低成本并提高资源利用率。弹性伸缩的优点在于它可以使系统具有更好的高可用性、灵活性和可扩展性。如图 1.1 所示，本文从伸缩方向、伸缩模式和技术方案三个方面对弹性伸缩技术进行分类，分别介绍相关研究现状。

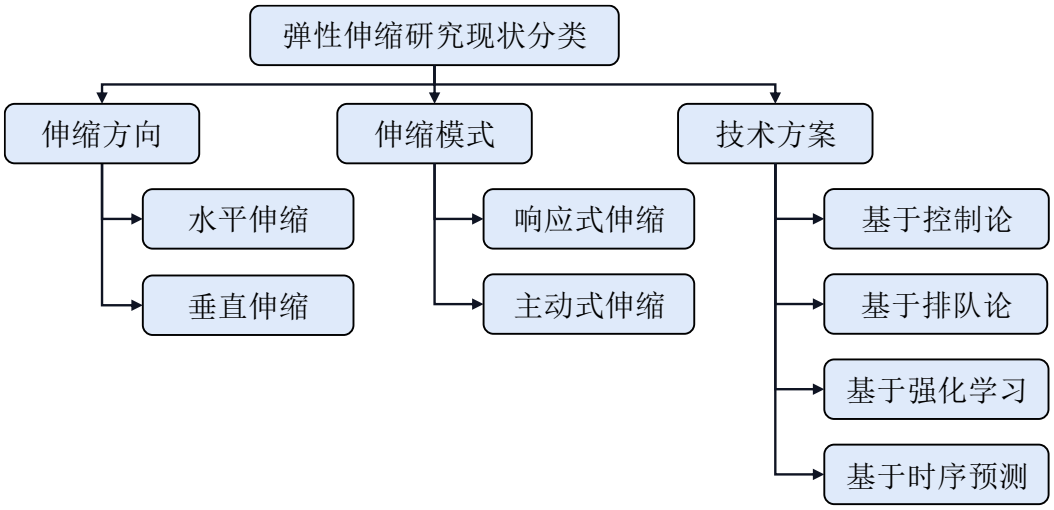


图 1.1 弹性伸缩研究现状分类图

按照伸缩方向，弹性伸缩技术可以分为水平伸缩和垂直伸缩[3]两种类型。水平伸缩技术是指增加或减少应用程序实例的数量，以应对负载变化的需求。在水平伸缩中，不需要对单个实例进行修改或调整，而是通过增加或减少实例的数量来提高或降低系统的处理能力。垂直伸缩技术是指增加或减少单个应用实例的计算资源，以应对负载变化的需求。垂直伸缩通常涉及增加或减少 CPU、内存、存储等硬件资源的数量和容量，以提高或降低应用程序的性能和可用性。垂直伸缩在修改实例的计算资源后，需要重启实例才能继续提供服务，会出现短暂的停机时间，从而影响应用程序的可用性和服务质量。因此，相比于水平伸缩，垂直伸缩目前尚未被广泛应用。本文选择目前应用广泛的水平弹性伸缩策略进行研究。

按照伸缩模式，弹性伸缩技术可以分为响应式伸缩和主动式伸缩两种类型。响应式伸缩是指通过轮询监控应用的负载状态，根据预先设置的阈值规则来触发弹性伸缩动作。响应式伸缩的相关研究主要集中在基于阈值的方法，该类方法通过选择合适的资源度量指标来反应系统状态，并针对资源度量指标设置不同的阈值范围来实现合理伸缩。例如，针对常用的上限和下限两个阈值不能反应指标变化趋势的问题，Hasan 等人[38]提出的方法额外增加了略低于上限阈值和略高于下限阈值这两个阈值，从而可以根据趋势采取伸缩动作。文献[39,40]提出了一种自适应阈值调整的方法，能够根据应用的负载情况自动调整阈值大小。Botrán 等人[41]提出的方法将阈值的初始值设置为静态，然后根据违反 SLA 的情况动态调节阈值。然而，基于阈值的响应式伸缩策略依赖于用户对应用的负载特征和性能需求的深刻理解，每个应用的阈值并不适用于其他应用，不能很好地满足实际需求。另外，由于响应式伸缩策略是基于阈值规则触发伸缩动作的，因此在扩容阶段时，应用程序往往已经出现资源瓶颈。而新部署的应用程序需要一定的时间来启动和达到稳定状态，才能提供服务。这导致应用程序的弹性反应滞后于负载状态的变

化，从而影响服务质量。相比于响应式伸缩方式，主动式伸缩更加具有主动性和智能性，可以有效地避免违反服务水平协议（SLA），并提高资源利用率，因此备受学术界的关注。这种伸缩方式利用历史负载数据建立预测模型，从而预测未来所需的资源，并基于预测值提前配置资源以应对未来负载变化。

应用于主动式或响应式伸缩策略的技术方案主要有以下四种：

（1）**基于控制论**：控制论的核心思想是通过建立反馈回路模型来描述系统的行为，设计和优化控制器以调节系统的输出保持在期望值范围内，实现目标控制[42]。Barna 等人[43]以比例积分微分（PID）控制算法为基础，建立了 CPU 利用率与响应时间之间的关系模型，并使用 PID 控制器对系统进行了动态调整，从而实现了对云 Web 应用的自动伸缩。Zhu 等人[44]针对云环境中的自适应应用程序，提出了一种带时间资源预算约束的资源配置方法，该方法基于反馈控制理论动态调节 CPU 和内存资源供应。基于控制论的方法能够以算法的执行效果为依据，对控制参数进行动态调整，具有良好的应用伸缩效果。但是，控制理论方案设计过于复杂繁琐，难以在系统的输入和输出之间建立可靠的映射。

（2）**基于排队论**：排队论是一种数学理论，用于研究服务系统为服务对象提供服务的过程，并分析系统的性能指标。排队论基于概率统计和运筹学的基础，通过对请求到达时间以及服务时间的概率分布进行建模，求解排队系统中的各项性能指标，如请求平均等待时间、系统利用率等，以此评估排队系统的性能。通过排队论的分析和优化，可以提高系统的效率、降低成本、提高服务水平。Gias 等人[45]提出了一个名为 ATOM 的自动伸缩系统，该系统使用分层排队网络对微服务应用程序建模，并使用遗传算法优化资源使用。Tong 等人[46]提出了一种基于平衡 Jackson 排队网络的整体自动伸缩策略，优化了多服务应用的 SLA 违规率和资源成本。Ding 等人[47]提出了一种名为 COPA 的自伸缩方法，该方法将微服务的伸缩问题根据工作量和响应时间建模为排队网络的 M/M/c 模型。排队理论通过先验假设来评估不同工作负载和实例数量条件下的应用程序性能，但是，当到达时间或服务时间显著偏离假设分布时，性能估计的准确性会降低。

（3）**基于强化学习**：强化学习（Reinforcement Learning，RL）[48]是一种机器学习方法，它通过智能体与环境的交互来学习最优的行为策略。在该方法中，智能体会根据当前的环境状态做出动作，并根据环境反馈的奖励或惩罚逐步学习最优策略，以实现最大化累积奖励的目标。RL 不依赖于先验知识，能够自主学习并适应不断变化的环境，因此在处理如弹性伸缩等时序决策问题方面具有独特优势[49]。已有的基于 RL 的弹性伸缩方法大多基于 Q-Learning 或 SARSA 算法。Horovitz 等人[50]提出了一种基于 Q-Learning 的动态改变水平伸缩阈值的方法，与传统 RL 方法相比，通过利用 Q 函数的单调性大大减少了状态空间的维度。帅

斌等人[51]提出了一种虚拟机弹性伸缩算法，通过引入决策后状态使算法只学习动态未知的状态信息，并采用值函数多步更新方式优化了 Q-Learning 算法的收敛速度。Zhang 等人[52]提出了基于 SARSA 算法的 FScaler 策略，以解决 Kubernetes 雾集群中的动态资源伸缩问题。上述方法都采用表格来存储 Q 值，然而微服务的资源指标状态拥有巨大且连续的状态空间，这将导致表格的维度急剧增加，学习过程缓慢。

深度强化学习通过神经网络来估计 Q 值函数，能够处理高维状态空间的复杂问题。Tesauro 等人[53]提出了一种基于 RL 和排队模型的混合方法用于资源自动分配，该方法由排队模型控制系统，RL 根据收集的数据进行离线训练，并用非线性函数逼近的方式替代表格查找。但是，该方法使用从零到最大可用实例数量的范围来定义状态空间，包含了所有可能的伸缩数量，这会增加计算复杂度，导致较长的训练时间。Xiao 等人[54]提出了一种基于深度强化学习的伸缩策略 Dscaler，该策略以 CPU 利用率、请求率和当前分配的 Pod 数量作为状态空间，以伸缩一至两个的 Pod 实例数作为动作空间，采用深度 Q 网络学习伸缩策略。然而，该方法中动作空间的伸缩幅度过小，可能无法满足实际的伸缩需求，导致系统在面对负载剧烈变化时，需要更多的时间进行伸缩操作。另外，大多数研究方法通常只考虑当前状态信息而无法提前感知状态变化，可能会导致智能体的行为策略不够灵活，从而影响了算法的收敛性和学习效率。总体而言，由于 RL 算法不依赖于人为干预或深层领域知识，可以通过与环境交互自动调整策略，能够适应云环境中的动态变化，适合于解决云环境下的弹性伸缩问题。

（4）**基于时间序列预测**：基于时间序列预测的伸缩策略包括两个阶段，第一个阶段是工作负载预测，第二个阶段是根据预测的工作负载做出伸缩动作。本文在 1.2.1 章节中介绍了云资源负载预测模型的研究主要可分为传统统计方法、机器学习方法、深度学习方法和混合模型方法四类。通过预测模型对未来负载和资源使用情况进行预测后，利用预测值根据预设的阈值规则确定伸缩动作。闫承鑫等人[55]提出了一种突发负载场景下的容器资源供给策略，分别通过基于三次指数平滑法的趋势预测算法和灰度预测算法计算容器资源的供给点和供给数量。Toka 等人[56]提出使用基于人工智能的预测方法进行主动扩展策略，该预测方法包括自回归、层级时间记忆模型和 LSTM 三种模型，每个模型都用于学习和预测传入 Web 请求的速率，每个周期会根据评估指标计算预测模型的准确性，并选择最佳模型计算需要的实例数量。Dang-Quang 等人[57]提出了一种基于 Bi-LSTM 的主动式伸缩策略，该策略使用 Bi-LSTM 模型对未来请求量负载进行预测，在负载上升时根据预测值和预设的单个实例可处理最大请求量的比值确定目标实例数，在负载降低时移除部分资源，以便处理可能发生的突发工作负载。基于时

序预测的方案可以提前配置资源以适应未来的负载变化，有效避免违反 SLA，是一种良好的选择方案。

综上所述，以上四种技术方案中，基于时序预测和基于强化学习的方法相对于其他方法在弹性伸缩问题上有着更好的泛化能力和有效性，是目前学术界的研究热点，但仍然存在一些不足。基于时序负载预测的方法依赖于预测模型的准确性，并且如何进一步根据预测值确定合理的伸缩动作仍存在困难。基于强化学习的方法需要针对弹性伸缩问题合理设计状态动作空间、奖励函数以及学习算法，并且通常只考虑当前状态信息而无法提前感知状态变化。针对这些问题，可以考虑一种新的主动式伸缩策略，以克服两种方法各自存在的缺点，并充分利用两种方法的优点，从而准确做出合理的弹性伸缩动作。

此外，通过对响应式伸缩策略和主动式伸缩策略的分析，可以得出，响应式伸缩策略存在着弹性滞后以及配置复杂问题，但其确定伸缩动作依据的是实时资源指标，能准确反映服务当前状态。而主动式伸缩策略能够提前进行弹性资源规划，从而有效避免 SLA 违规并提高资源利用率，但在极端突发流量场景下，预测或建模结果不够准确容易导致策略失去有效性。针对这一问题，需要探索一种更加全面和完善的伸缩策略体系，以兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

1.3 研究内容

精确预测未来负载并做出合理的弹性伸缩动作，是弹性伸缩任务中的关键。然而，无论是时序负载预测还是弹性伸缩技术，都仍然存在一些问题：1）在负载预测任务中，现有方法仅关注于某种负载变化特征，缺乏对复杂特点进行全面建模，在预测精度上仍有一定的欠缺；2）在应用于弹性伸缩策略的技术方案中，现有方法存在伸缩决策不准确、动态性差等的问题，由此产生的不合理伸缩动作会导致系统性能下降或资源浪费；3）现有的弹性伸缩策略研究缺乏对极端突发流量场景的考虑，单一地依靠主动式伸缩策略存在一定的弹性失效风险，从而影响服务的可用性。针对以上问题，本文设计了一种基于负载预测和强化学习的容器弹性伸缩策略，目标是在满足 SLA 的同时最大化资源利用率。本文研究内容及其关系如图 1.2 所示，主要研究内容总结如下：

（1）为了有效捕获短期波动以及多个尺度上持续性和周期性的负载变化特征，以提高负载预测的准确性，本文提出了基于多尺度空洞卷积和 LSTM 的负载预测模型。该模型主要由多尺度特征提取模块和双通道时序预测模块组成。多尺度特征提取块利用平行和堆叠的空洞卷积残差网络，从连续和非连续的输入时间步中提取时序特征，从而同时学习负载短期波动、持续性变化和时间步级别周期

性信息。然后，将多尺度特征提取块的输出结果利用 LSTM 进一步对长期时间依赖进行建模。双通道时序预测模块中的周期特征学习通道将输入数据进行周期性时间编码，从而更好地学习每日和每周级别的周期性模式。实验结果表明，本文提出的模型的预测精度优于其他对比模型。

（2）为了准确确定合理的伸缩动作，以在保证服务质量的同时提高资源利用率，本文提出了一种基于负载预测和强化学习的主动式伸缩策略。该策略充分利用应用程序的当前资源负载状态和预测信息，由强化学习智能体做出合理的弹性伸缩决策，能够更加准确和智能地调整应用程序实例个数。此外，为了应对极端流量峰值场景，本文进一步地提出了一种主动式和响应式的混合弹性伸缩策略。该策略的响应式伸缩模块考虑了请求响应时间和资源利用率，作为主动式伸缩策略失效时的备选方案。并通过混合伸缩控制器协调主动式和响应式伸缩策略，采用“快扩容慢缩容”和平衡冲突思想，做出最终伸缩决策，兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

（3）为了验证所提出的弹性伸缩策略的有效性，本文在 Kubernetes 平台上实现了弹性伸缩系统，系统包括监控模块、预测模块、强化学习模块和伸缩控制模块，并设计了平稳波动和异常突发两种负载场景下的弹性伸缩对比实验。实验结果表明，本文提出的弹性伸缩策略相比于对比方法，具有更低的 SLA 违规率和更高的资源利用率。

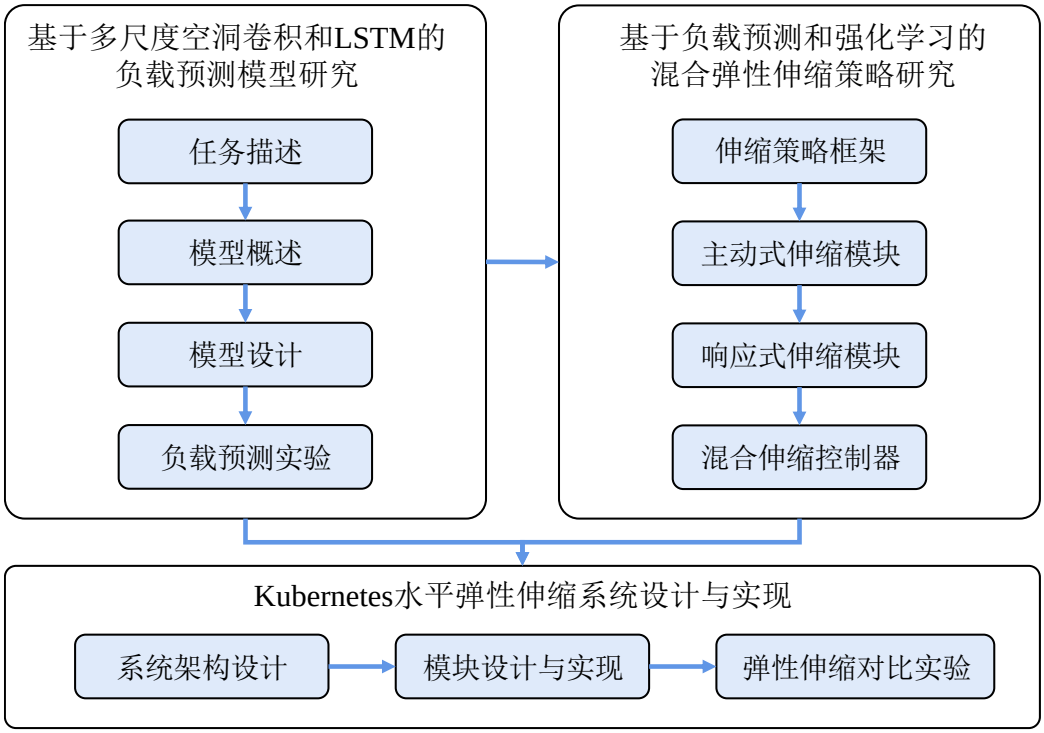


图 1.2 研究内容及其关系图

1.4 组织结构

以下是每个章节的主要内容：

第 1 章，绪论。首先介绍了本文工作的研究背景及意义，其次调查了近几年国内外在时间序列负载预测和弹性伸缩技术的研究现状，总结了其中存在的问题，最后阐述了本文的研究内容和组织结构。

第 2 章，相关理论。首先介绍了 Kubernetes 默认弹性伸缩原理，包括 Kubernetes 核心概念与架构、水平弹性伸缩策略的使用和原理。接着介绍了时序预测模型相关理论，包括一维卷积网络、长短期记忆网络。最后介绍了强化学习相关理论，包括基本概念、马尔科夫决策过程和深度强化学习算法。

第 3 章，基于多尺度空洞卷积和 LSTM 的负载预测方法研究。首先介绍了负载预测任务中面临的挑战和存在的问题，接着对负载预测任务进行了数学描述，然后针对负载预测任务中存在的问题，提出了一种基于多尺度空洞卷积和 LSTM 的负载预测模型，并详细介绍了该模型的多尺度特征提取模块和双通道时序预测模块的设计，最后对所提出模型进行了实验验证。

第 4 章，基于负载预测和强化学习的混合弹性伸缩策略研究。首先介绍了弹性伸缩任务中存在的问题，然后描述所提出的水平混合弹性伸缩策略的整体框架，并分别详细介绍了基于深度 Q 网络的主动式伸缩策略设计、基于原生响应式伸缩策略的改进优化以及主动式和响应式混合伸缩控制器设计。

第 5 章，Kubernetes 水平弹性伸缩系统设计与实现。首先介绍了本文提出的伸缩策略在 Kubernetes 环境中进行实现的系统整体架构设计，然后分别详细介绍了监控模块、预测模块、强化学习模块、伸缩控制模块的设计与实现，最后设计了弹性伸缩对比实验，评估并分析了实验结果。

第 6 章，总结与展望。总结了本文的研究工作，并且对未来可以开展的工作进行了展望。

第 2 章 相关理论

2.1 Kubernetes 弹性伸缩策略相关理论

2.1.1 Kubernetes 核心概念与架构

Kubernetes 是由谷歌开发的开源容器编排系统，主要致力于协助开发者、管理员和系统工程师等专业人员在快速、高效的环境下，实现云原生应用程序的部署、扩展和管理。该系统提供了一套完整的容器编排机制和工具，包括容器的调度、自动伸缩、故障恢复、负载均衡、网络管理等，能够使得应用程序在集群中的部署、管理和维护更加简单、可靠和可扩展。经过多个版本迭代和社区贡献，Kubernetes 已经成为了一种容器编排标准，被广泛应用于云原生应用程序和微服务架构的开发、部署和管理，是现代云原生计算架构中不可或缺的一环。

（1）Kubernetes 核心概念

Kubernetes 资源对象是一组核心抽象概念，用于描述和管理集群中的各种资源及容器应用，与弹性伸缩相关的概念如下：

1）Pod

Pod 是 Kubernetes 中最基本的部署和管理单元。Pod 由一个或多个容器组成，这些容器可以共享网络和存储资源，并被视作一个逻辑主机。作为 Kubernetes 的调度和部署单元，Pod 可分布式部署于一个或多个节点。此外，Pod 的生命周期主要由 Deployment 进行管理和控制，以便更好地监测和维护容器的状态。

2）Label

Label 是 Kubernetes 中用于对资源进行分类和组织的机制。Label 是一组键值对，可以为 Pod、ReplicaSet、Deployment、Service 等对象打上标签，并可以根据标签来选择和操作这些对象。通过标签，用户可以更灵活地组织和管理 Kubernetes 资源。

3）Deployment

Deployment 是 Kubernetes 中用于控制应用程序更新的机制。Deployment 定义了应用程序的期望状态和副本数量，以及如何将应用程序从当前状态更新到期望状态。Deployment 可以滚动更新应用程序，以避免出现应用程序不可用的情况。在具体实现中，应用程序的副本数量由 ReplicaSet 资源对象负责控制。

4）Service

Service 是 Kubernetes 中用于公开应用程序服务的机制。Service 定义了一组

Pod 的访问方式和负载均衡规则，为应用程序提供了一个稳定的网络入口。通过 Service，用户可以将多个 Pod 组合成一个逻辑单元，并为其分配一个稳定的 IP 地址或域名。Service 可以根据负载均衡算法将请求分发到多个 Pod 中，以实现高可用和可伸缩性。

（2）Kubernetes 系统架构

Kubernetes 架构图如图 2.1 所示，包括 Master 节点和 Node 节点两个层次。

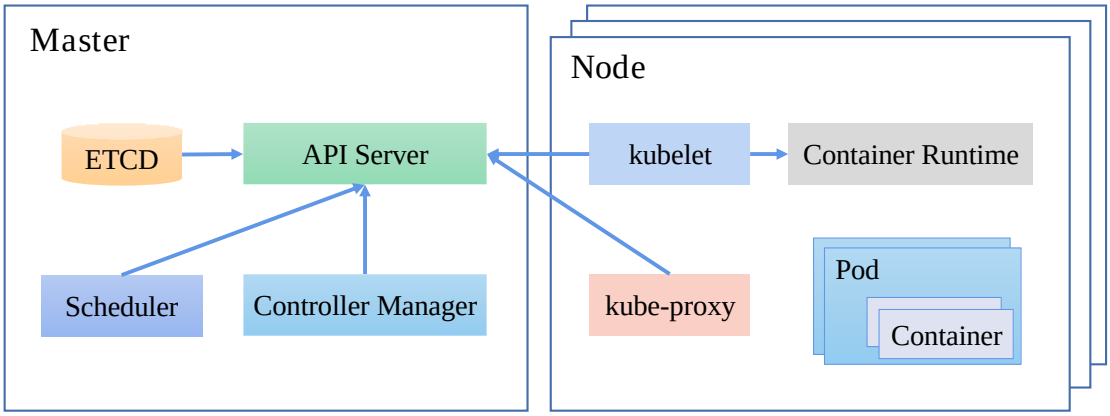


图 2.1 Kubernetes 架构图

Master 节点是集群中的管理节点，包括 Etcd、API Server、Scheduler 和 Controller Manager 等组件。其中，Etcd 用于存储集群状态信息；API Server 用于提供 REST API 接口，负责接收和处理来自用户、管理员和其他系统组件的请求；Scheduler 用于对新的 Pod 进行调度，并将其分配到合适的 Node 上；Controller Manager 用于控制集群状态，保证所有资源都达到预期状态。

Node 节点是集群中的工作节点，包括 Kubelet、Kube-Proxy 等组件。Kubelet 是 Node 节点上的代理程序，用于管理 Pod 的生命周期，负责在 Node 节点上创建、启动、停止和删除 Pod，同时与 Master 节点的 API Server 交互以获取 Pod 的调度信息。Kube-Proxy 是集群的网络代理，用于维护集群内部的网络连接，为 Pod 提供网络代理服务，将 Pod 暴露给其他节点或外部网络。

2.1.2 Kubernetes 水平弹性伸缩策略原理

Pod 是 Kubernetes 中基本的调度和部署单元，因此弹性伸缩即为对部署应用的 Pod 进行伸缩。在 Kubernetes 中，水平弹性伸缩称为 HPA（Horizon Pod Autoscaling）。HPA 是一种通过动态增加或减少应用的 Pod 实例数来调整资源供给的策略。这些 Pod 实例是基于相同的配置文件创建的，它们被组织成一个副本集，并通过 Service 负载均衡地向用户提供服务。因此，HPA 监控的对象是整个 Pod 副本集，HPA 会根据整个副本集中 Pod 的当前平均资源指标值和指定的资源指标目标值，来确定具体的伸缩动作。

为了使用 HPA 实现自动伸缩应用程序，需要创建一个 HPA 资源对象，并在其上设置适当的参数，以确保自动伸缩有效进行。表 2.1 列出了 HPA 资源对象的主要参数。

表 2.1 HPA 资源对象的主要参数

参数名称	参数含义
scaleTargetRef	HPA 作用的伸缩对象
MinReplicas	允许扩容的最大副本数
maxReplicas	允许缩容的最小副本数
Metrics	伸缩指标以及目标值

在 Kubernetes 中使用 HPA 实现自动伸缩应用程序的步骤如下：

（1）创建 Deployment 资源对象，指定应用的容器镜像、每个 Pod 所需的资源和所需的 Pod 数量，由 Deployment 对象管理应用的 Pod 副本数和升级。

（2）创建 Service 资源对象，将其与 Deployment 关联，并指定要创建服务的名称、所需的端口信息，从而将 Pod 副本集组织成逻辑单元，为应用程序提供稳定的访问入口。

（3）创建 HPA 资源对象，通过 scaleTargetRef 参数将其与 Deployment 关联，并利用 metrics 参数设置伸缩指标类型和目标值，以及设置 Pod 副本数的最大和最小值等参数。

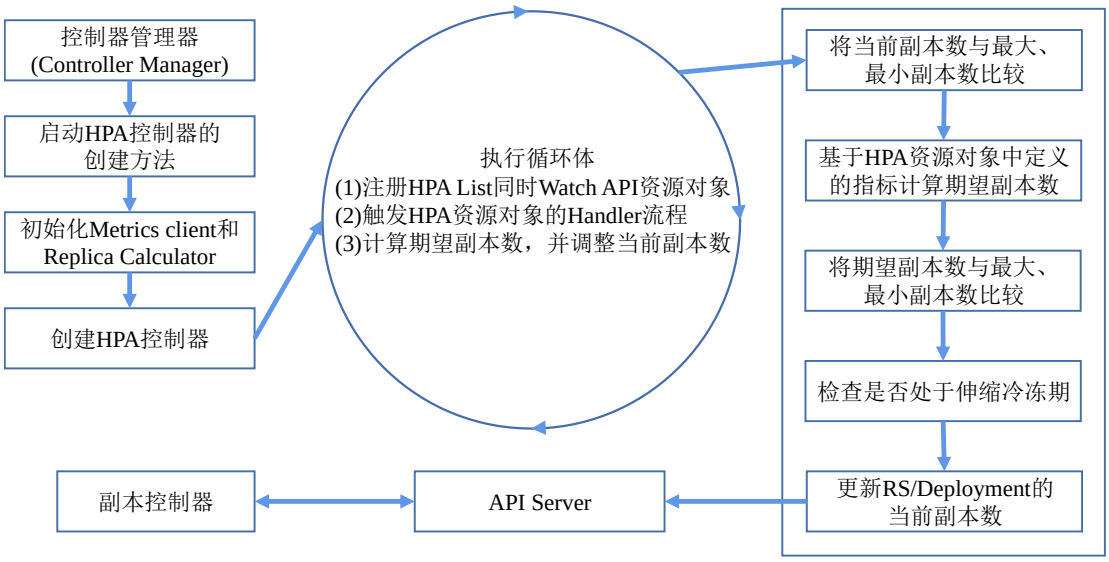


图 2.2 HPA 控制器的控制流程图

在完成资源对象的创建之后，HPA 控制器会自动执行弹性伸缩控制，根据 HPA 资源对象的配置信息进行 Pod 数量调整，具体的控制流程如图 2.2 所示[58]。首先由 Controller Manager 发起 HPA 控制器的创建过程，该过程中涉及对 Metrics Client 和 Replica Calculator 组件进行实例化，以分别提供获取伸缩指标数据的接

口和计算 Pod 副本数的方法。HPA 控制器执行控制循环，以周期性监控 HPA 对象的状态，并根据需要执行自动伸缩，具体包括以下步骤：

（1）根据 HPA 对象中关联的 Deployment 以及伸缩指标类型，通过 Metrics Client 向 API Server 查询每个 Pod 的伸缩指标当前值。

（2）利用伸缩指标中的资源当前值与 HPA 中定义的资源目标值，根据 Replica Calculator 中定义的计算方法，从而得到期望的 Pod 副本数 DesiredReplicas。具体而言，计算公式可以表示为式（2.1）：

$$DesiredReplicas = ceil\left[\frac{V_c}{V_t}\right] * CurrentReplicas \tag{2.1}$$

其中，V_c表示 Pod 副本集的当前平均资源值，V_t表示定义的资源目标值，CurrentReplicas 表示当前副本数。

（3）将期望副本数分别与最大副本数、最小副本数比较，如果期望副本数大于最大副本数或小于最小副本数，则修改期望副本数以确保在指定的范围内。

（4）检查当前时间是否处于缩容冷却期内。如处于缩容冷却期内，则不进行缩容操作以避免因负载波动导致应用频繁伸缩。

（5）根据上述计算得到的期望副本数，通过调用相关接口将 Deployment 中的副本数修改为期望副本数。

通过以上对 HPA 弹性伸缩策略的流程进行梳理，可以发现尽管其具有简单易懂的特点，但其在实施过程中仍存在一些缺陷：

（1）HPA 仅根据当前伸缩指标值与目标阈值之间的关系，以计算期望副本数，因此是一种响应式伸缩策略。在不同的应用负载模式和场景下，设定合适的指标和阈值标准依赖于用户对应用的负载特性和性能需求的深刻理解。如果阈值设定过高，会导致应用出现资源瓶颈，无法满足用户需求；而阈值设定过低，则会导致资源浪费和过剩，从而降低整体资源利用率。

（2）在扩容阶段，由于新创建的 Pod 副本需要一定时间来启动和达到稳定状态，才能处理请求，因此应用的弹性会滞后于负载状态的变化，引起请求响应时间增加和应用性能下降，从而影响服务质量。

（3）在缩容阶段，云负载呈现出复杂的时序模式，并且具有较大的波动性。具体而言，负载可能会先下降，然后再次上升。这种现象可能导致频繁的伸缩操作，这同样会影响应用的服务质量。

为了解决上述问题，本文提出了一种负载预测模型，该模型通过结合多尺度空洞卷积和 LSTM 实现对应用负载的准确预测。同时，本文基于此模型，提出了一种混合弹性伸缩策略，该策略结合了负载预测和强化学习方法，旨在对 HPA 中默认的响应式伸缩策略进行改进。

2.2 时序预测模型相关理论

2.2.1 一维卷积网络

一维卷积网络（1D-CNN）是一种常见的神经网络架构，广泛应用于时间序列数据和文本数据的处理任务中。1D-CNN 主要基于一维卷积操作，通过学习卷积核参数来提取输入数据中的特征信息。一般而言，1D-CNN 可以分为普通一维卷积、因果卷积和空洞卷积三种不同类型。

（1）普通一维卷积

普通一维卷积的实现方法与二维卷积类似，它需要定义一个卷积核，通过在输入信号上滑动卷积核并执行卷积操作来生成输出信号。在时序预测任务中，输入数据即为时间序列数据，卷积核仅在时间这一维度上移动。在一维卷积中，卷积核通常是一个长度小于输入信号长度的一维向量。对于每个卷积核的位置，它将卷积核的值与对应的输入序列值相乘，然后将结果相加，得到输出序列的一个值，从而从输入中的子区域提取局部特性[59]，可以表示为式（2.2）：

$$(k \star x)[i] = \sum_{j=0}^{m-1} x[i-j]k[j] \tag{2.2}$$

其中，x表示输入向量，k表示卷积核，m表示卷积核大小，$\star$表示卷积运算。

步长是指卷积核在输入序列上移动的时间步长，一般为正整数，用于控制卷积核每次移动的步幅。在传统的卷积操作中，卷积核通常每次移动一个时间步长，这种方式能够有效地学习到输入数据的细节特征。当卷积核每次移动的步长大于 1 时，称之为步长卷积。步长卷积会导致输出向量的尺寸变小，从而具有下采样的效果，可以减少计算量。但是，步长卷积也会导致部分输入信息的丢失，因此在选择步长时需要进行权衡，以平衡模型的计算效率和精度之间的关系。

（2）因果卷积

因果卷积是一种专门用于处理时序数据的卷积操作，其设计基于因果关系的概念，其网络结构如图 2.3 所示。在时序数据中，每个时间步的值都是由之前的时间步所决定的，因此需要一种能够捕捉这种因果关系的卷积操作。与传统的一维卷积操作相比，因果卷积具有明显的优势。在传统的卷积操作中，卷积核可以访问整个输入序列的值，从而可能会在预测未来时使用未来的信息，这会导致模型出现错误。而在因果卷积中，卷积核在计算当前时间步的值时只能使用之前的时间步的信息，而不能使用之后的时间步的信息。因此，因果卷积可以有效地防止信息泄漏，避免模型在预测未来时出现错误。

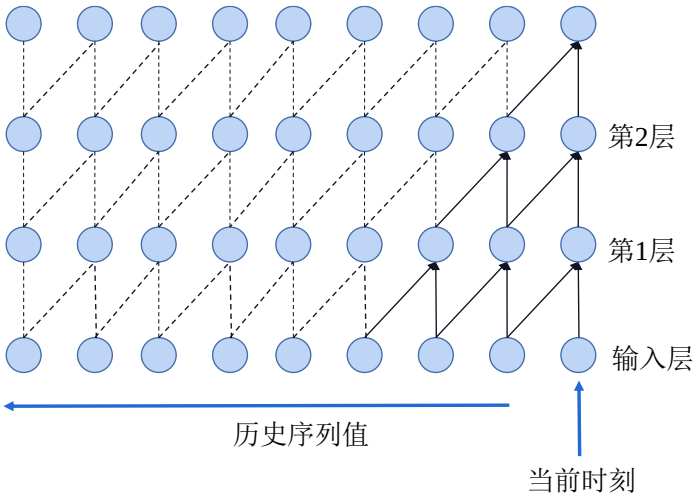


图 2.3 因果卷积

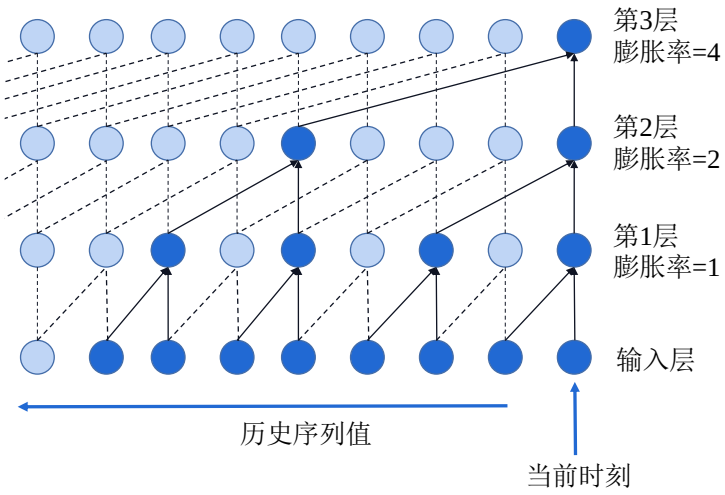


图 2.4 空洞卷积

（3）空洞卷积

空洞卷积的核心概念是通过在卷积核中插入一定数量的零值，从而扩大卷积核的感受野（Receptive Field），增强了网络对输入数据全局信息的理解能力[60]，其网络结构如图 2.4 所示。空洞卷积通过膨胀率（Dilation Rate）参数来控制卷积核中间的零值数量，可以通过增加膨胀率的方式，灵活地调整卷积操作的感受野大小，从而适应不同尺度的输入数据。假设膨胀率为d，空洞卷积运算表示为$\star_d$，在序列x中i位置上的运算值表示为式（2.3）：

$$(k \star_d x)[i] = \sum_{j=0}^{m-1} x[i - d \cdot j]k[j] \tag{2.3}$$

当膨胀率为 1 时表示卷积核中不存在零值的标准卷积，当膨胀率大于 1 时表示空洞卷积。堆叠的因果空洞卷积层如图 2.4 所示，图中深色的点代表每一层的感受野。膨胀率大小为d的卷积核对输入的每d个元素进行卷积，即跳过中间$d-1$个元素，从而扩大了感受野。通过每一层指数级增加膨胀率，同时保持卷积核

大小固定，来构建一个大的感受野。当膨胀率为 2 时，卷积核函数计算每两个元素的点积；当膨胀率为 4 时，卷积核函数计算每四个元素的点积。因此，第 1 层、第 2 层和第 3 层的感受野分别为 2、4 和 8，而卷积核大小仍为 2。

1D-CNN 由于卷积操作中简单的点乘积运算而具有低计算复杂度。此外，使用空洞卷积实现对输入的特征学习和感受野的灵活设计，使 1D-CNN 能够具备平移不变性。因此，在捕获嘈杂的云负载变化的特征方面，1D-CNN 是一个合适的选择。

2.2.2 长短期记忆网络

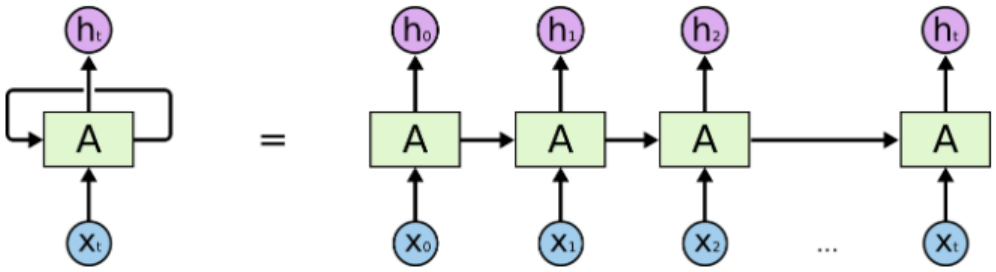


图 2.5 循环神经网络结构图

循环神经网络（RNN）是一种深度学习模型，能够处理序列数据并保留其时间相关性，被广泛用于处理序列数据。如图 2.5[1]所示，RNN 的基本原理是通过循环连接，将前一时刻的输出作为当前时刻的输入，并通过学习序列内部的时间状态来映射这些依赖关系到目标变量上。然而，RNN 在处理长序列时存在一个严重问题，即由于参数更新的逐渐减小，梯度很快就会消失，导致模型忘记历史上较远的上下文信息，进而影响模型的性能[61]。这一局限性严重限制了 RNN 在实际应用中的效果。因此，普通 RNN 对于信息的长期依赖问题没有很好的处理办法。为了克服这个问题，Hochreiter 等人提出了长短期记忆（LSTM）网络，可以更好地学习长期依赖信息[62]。

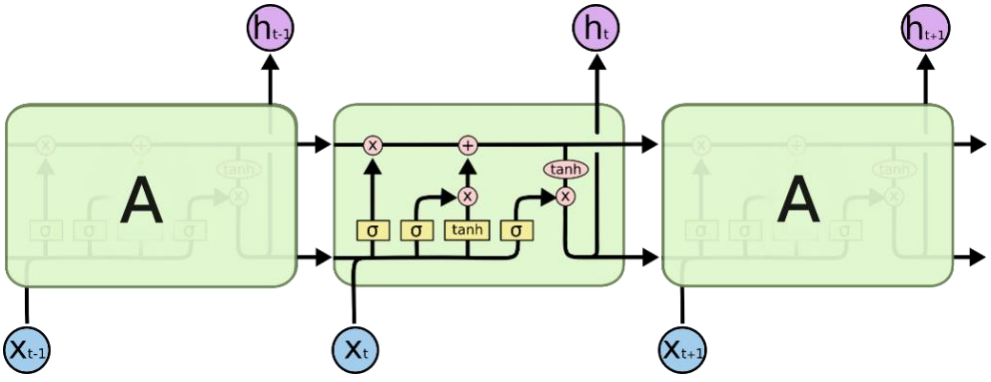


图 2.6 LSTM 网络模型

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

如图 2.6[1] 所示，LSTM 网络相较于传统的 RNN，它增加了输入门、遗忘门和输出门三个门控机制，可以选择性地遗忘或记住历史信息，并在网络中传递重要信息[63]。LSTM 的这种特殊结构设计使其能够记住很早时刻的信息，而不需要额外的计算代价，从而解决长期依赖问题。

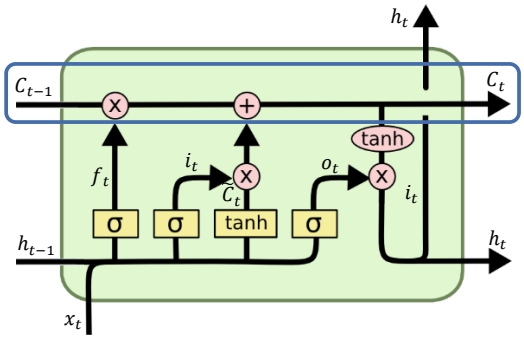


图 2.7 LSTM 单元状态传播

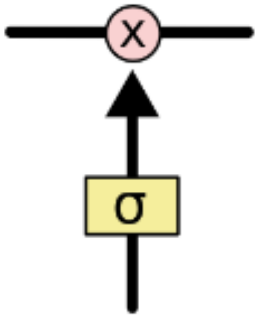


图 2.8 LSTM 门控机制

LSTM 的核心思想是单元状态传播和门控机制。如图 2.7[1] 所示，单元状态C_t用于存储当前状态信息，并在下一个时刻传递给下一个 LSTM。具体而言，当前 LSTM 接收来自上一时刻的单元状态C_{t-1}，并结合当前输入x_t进行作用，从而生成当前 LSTM 的C_t。LSTM 门控机制如图 2.8[1] 所示，门控机制用于控制信息选择性地传递，通过 sigmoid 函数和按位乘法操作引入或删除单元状态的信息。

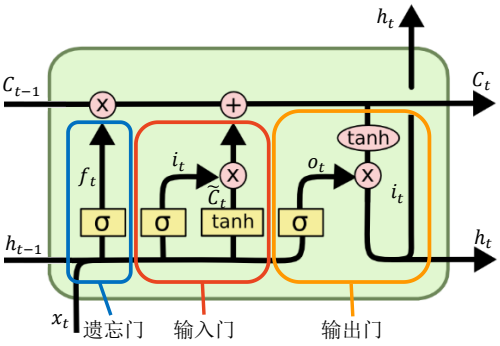


图 2.9 LSTM 门结构

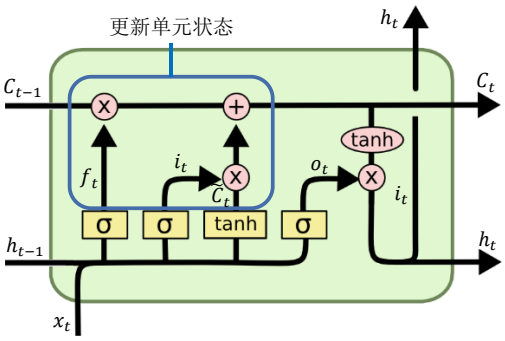


图 2.10 LSTM 更新单元状态

如图 2.9[1] 和图 2.10[1] 所示，展现了 LSTM 的三个门结构和单元状态更新过程，图中，×和+是逐元素的乘法和加法运算，σ是 sigmoid 函数，下面分别介绍原理：

（1）遗忘门：遗忘门用于决定单元状态C_{t-1}中将被遗忘的信息，其结构如图 2.9 所示。遗忘门将上一时刻的输出h_{t-1}和当前时刻的输入x_t作为输入，经过 sigmoid 函数后，输出一个向量f_t。并在单元状态更新时，与C_{t-1}相乘来决定C_{t-1}中的哪些信息将被舍弃或保留。其计算方式为式（2.4）：

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.4}$$

其中，W和b表示网络参数。

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

（2）输入门：输入门用于控制信息的输入，决定新输入的信息x_t和h_{t-1}中哪些信息将被保留，其结构如图 2.9 所示。输入门包含两个部分，sigmoid 神经网络层输出i_t值，tanh 神经网络层输出状态候选向量$\tilde{C}_t$。输入门的输出是i_t与$\tilde{C}_t$的乘积，表示将被加入单元状态C_t的信息。其计算方式为：

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.5}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.6}$$

（3）单元状态更新：如图 2.10 所示，将遗忘门与输入门的输出合并，从而更新单元的状态。具体来说，首先将上一时刻的单元状态C_{t-1}与遗忘门的输出f_t相乘来确定遗忘和保留的信息；然后将这部分信息与输入门的输出$i_t * \tilde{C}_t$相加，得到新的单元状态C_t。其计算方式为：

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.7}$$

（4）输出门：输出门用于控制信息的输出，其结构如图 2.9 所示。首先，根据当前时刻的输入x_t和前一时刻的隐状态h_{t-1}，计算出一个介于 0 和 1 的值o_t。然后，C_t经过 tanh 运算并与o_t相乘得到当前时刻的输出h_t。其计算方式为：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.8}$$

$$h_t = o_t * \tanh(C_t) \tag{2.9}$$

LSTM 网络通过隐状态h表示的循环连接，使得网络能够学习短期时间依赖关系；同时，通过单元状态C来记住来自长期历史的重要信息，从而能够更好地学习序列数据的长期依赖性。因此，LSTM 在处理序列数据时具有广泛的应用价值。本文提出的模型就利用 LSTM 的这种能力，来学习云负载中的长期依赖性。

2.3 强化学习相关理论

2.3.1 基本概念

强化学习[65]（Reinforcement Learning）是一种机器学习方法，旨在通过自主与环境互动，从中学习并改进行为策略，以最大化长期累积回报。强化学习的基本框架如图 2.11 所示，由智能体（Agent）、环境（Environment）、状态（State）、动作（Action）、奖励（Reward）和策略（Policy）组成。在强化学习中，智能体与环境互动，从环境中获取状态信息，根据当前状态和策略，选择动作，并根据环境返回的奖励来更新策略。通过反复迭代，智能体逐渐调整其策略，以最大化长期累积回报。

强化学习在各种实际场景中得到了广泛应用，包括建筑控制[66]、决策判断[67]、游戏 AI[68]等。它的优点在于可以在没有先验知识的情况下自主学习，适用于复杂环境和多变任务，并能够实现长期回报最大化的智能决策。

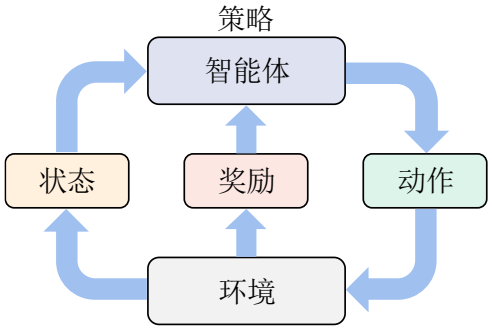


图 2.11 强化学习基本框架图

2.3.2 马尔科夫决策过程

马尔科夫决策过程（Markov Decision Process，MDP）[69]是强化学习的理论基础之一。MDP 是一种数学框架，用于描述强化学习中的环境和智能体之间的交互。MDP 将环境划分为状态集合和动作集合，并在状态转移和奖励上施加一些概率分布。这些概率分布满足马尔科夫性质，即未来状态和奖励只与当前状态和动作相关，而与过去的状态和动作无关。一个 MDP 可以由五元组构成，即$MDP = (S, A, P, \gamma, R)$，其中各元素具有以下含义：

S表示状态空间，由多个状态s_i组成的一个集合，即$S = (s_1, s_2, \cdots, s_n)$，其中$s_i$表示时间步$i$的状态。

A表示动作空间，由多个动作a_i组成的一个集合，即$A = (a_1, a_2, \cdots, a_n)$，其中$a_i$表示时间步$i$的动作。

P表示状态转移概率，由状态空间S中任一个状态s_i执行动作空间A中任一个动作a_i后，状态转移到另一个状态s_j的概率分布所组成的一个矩阵。

γ表示折扣因子，且$\gamma \in [0,1]$。折扣因子用于表示长期奖励的价值，通常随着时间步的增长，折扣因子的作用会越来越小，即离当前状态越近的行为奖励贡献越大，越远的行为奖励对当前的贡献越少。

R表示奖励函数，在发生状态转移时，通过奖励函数计算奖励r，即$r = R(s, a)$。

通常情况下，强化学习问题的建模可以借助 MDP 框架完成，并且使得智能体能够学习和解决问题。这是因为在 MDP 中，不仅考虑了状态s，而且还考虑了动作a对下一个状态的影响，因此强化学习模型所转换到的下一个状态取决于当前状态s和所采取的动作a。图 2.12 直观地展示智能体在 MDP 框架下的决策过程，智能体在状态s_0下采取动作a_0并执行，然后到达状态s_1，在状态s_1下采取动作a_1并执行，并到达状态s_2。该过程将不断循环往复，直到任务完成或终止，到达最终状态s_n。

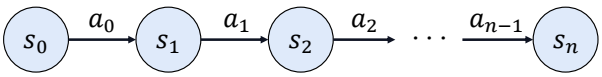


图 2.12 基于 MDP 表示的强化学习

2.3.3 深度强化学习

传统的强化学习算法使用表格存储的方法，该方法将强化学习问题建模为一个表格，其中每个单元格代表某种状态和动作的组合，同时存储了相应的值函数。基于表格存储的传统强化学习方法简单易懂，但面临着许多问题。首先，由于表格的大小受到状态空间的限制，当状态空间非常大时，需要大量的时间和计算资源来存储和更新表格。其次，表格存储方法对于连续动作空间和连续状态空间的问题难以适应，因为需要离散化这些空间，这可能会导致信息丢失和不准确的近似。最后，表格存储方法可能会陷入局部最优解，难以跳出这些最优解并找到更优的解决方案。

为了克服这些问题，深度强化学习[70]应运而生。深度强化学习利用神经网络来逼近值函数或策略，可以灵活地处理连续动作空间和连续状态空间，并且可以处理非线性关系。与传统的表格存储方法相比，深度强化学习具有更高的表示能力和更强的泛化能力，因此可以在更广泛的问题上获得更好的性能。此外，深度强化学习还可以通过使用经验回放和目标网络等技术来克服局部最优解问题，从而获得更好的学习效果。

2.4 本章小结

本章重点介绍了后续章节所使用的相关技术。首先，对 Kubernetes 默认的弹性伸缩原理进行了详细介绍，包括 Kubernetes 的核心概念与架构，以及水平弹性伸缩策略的使用和原理。随后，介绍了时序预测模型相关的理论，包括一维卷积网络和长短期记忆网络。最后，介绍了强化学习相关的理论，包括强化学习的基本概念、马尔科夫决策过程和深度强化学习算法。这些技术的介绍为后续章节的内容提供了理论基础。

第 3 章 基于多尺度空洞卷积和 LSTM 的负载预测模型研究

负载预测是主动式弹性伸缩的基础，容器云平台利用精准的负载预测技术，能够提前预知资源需求变化，从而快速、准确地调整资源供应，以提高资源利用率并降低成本。然而，云负载数据时序变化模式多样，表现出波动性、持续性和多周期性的特点，给精准预测未来负载带来了巨大挑战。其中，波动性是指由用户行为模式、热点事件等一系列不确定的因素引起的短期波动性负载；持续性是指负载在长时间内高度相关，存在较强的自相关性；多周期性是指负载数据中存在多个周期，呈现出随天、周、月等时间长度的变化周期。这些特点的复杂性增加了负载预测的难度。因此，针对这些特点，提高负载预测的准确性，以满足云计算环境下弹性伸缩的需求，成为了研究的重要问题。尽管已经有很多研究致力于解决负载预测问题，但仍然存在一些缺陷。例如，基于 LSTM 的方法[13,18]难以捕捉云负载中的短期波动；以 TCN 为代表的 1D-CNN 方法[19-22]对长期依赖问题的处理能力有限；在 CNN-LSTM 混合模型方法中，具有单个卷积层的方法[34-36]只能学习短期时间模式，而堆叠卷积的方法[37]由于受到最后一层有效感受野的限制，忽略了输入序列中的全局信息。这些研究仅关注于某种负载变化特征，不能对这些复杂特点进行全面建模，在预测精度上有一定的欠缺。

针对上述问题，本章提出了基于多尺度空洞卷积和 LSTM 的负载预测模型，能够捕获短期波动以及多个尺度上持续性和周期性的负载变化，从而提高负载预测精度。首先，利用由平行和堆叠的空洞卷积残差网络构成的多尺度特征提取块，从连续和非连续的输入时间步中提取时序特征，从而同时学习负载短期波动、持续性变化和时间步级别周期性信息。然后，通过自相关分析，观察和了解负载序列中的多周期性，以此指导多尺度特征提取块中的参数确定和周期特征学习通道的每日每周级别周期时间编码。最后，在双通道预测模块中，将多尺度特征提取块的输出结果利用 LSTM 进一步对长期时间依赖进行建模，并将基本时序特征学习通道和周期时序特征学习通道提取的特征融合，经全连接神经网络输出预测序列，实现负载预测。

3.1 任务的形式化描述

负载预测是指根据历史时间序列的负载预测未来预期时间序列的负载。资源使用指标和应用业务指标是云负载的两种主要类型，前者通常包括 CPU 利用率和内存利用率等指标，能够反映资源利用情况；后者包括请求率和请求响应时间

等指标，能够反映应用的服务质量。用户可以根据自身需求自主选择不同的指标进行训练。预测模型的输入表示为：

$$X_{1:T} = \{x_1, x_2, \dots, x_T\} = \{x_t \in R \mid t \in 1 \dots T\} \tag{3.1}$$

其中，x_t表示时间序列中t时刻的负载状态，T表示模型输入的时间序列长度，对应的目标值为服务的未来负载并表示为：

$$Y_{T+1:T+s} = \{y_{T+1}, y_{T+2}, \dots, y_{T+s}\} = \{y_t \in R \mid t \in T+1 \dots T+s\} \tag{3.2}$$

其中，y_t表示应用在时间序列中时间t的负载状态，时间序列模型可以表示为：

$$(y_{T+1}, y_{T+2}, \dots, y_{T+s}) = f(x_1, x_2, \dots, x_T) \tag{3.3}$$

其中，s表示模型输出的预测时间序列长度，$f(\cdot)$表示预测模型的实现。

在后文中介绍的模型输入模块会将输入序列的时间戳经过正弦余弦变换转化为周期性特征编码，表示为：

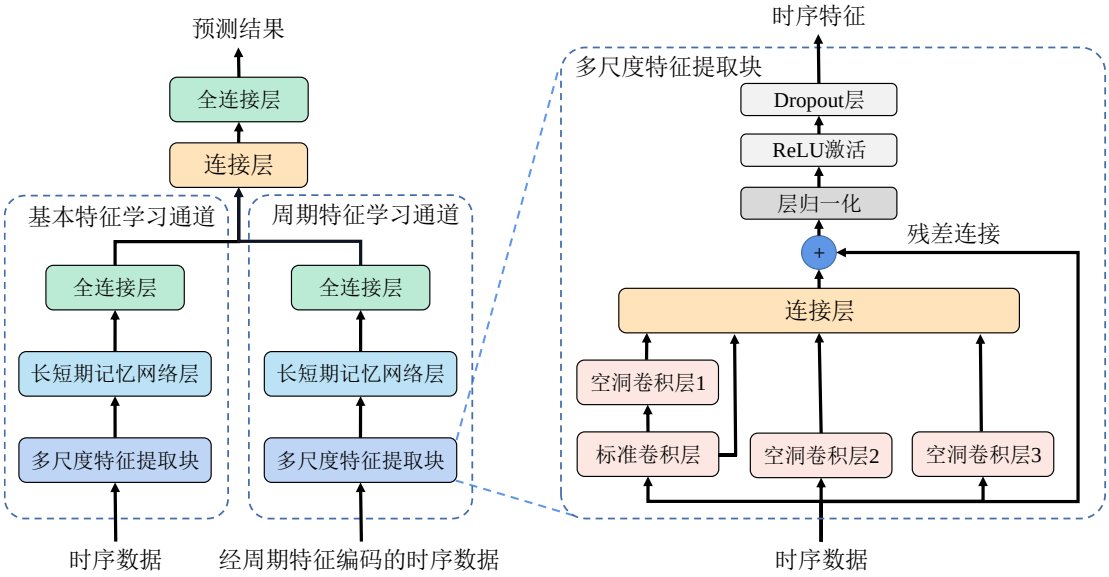
$$P_{1:T} = \{t_1^s, t_2^s, \dots, t_T^s; t_1^c, t_2^c, \dots, t_T^c\} \tag{3.4}$$

其中，$\{t_i^s, t_i^c\}$表示对第i个时间步时间值的周期性特征编码。

3.2 基于多尺度空洞卷积和 LSTM 的负载预测模型

3.2.1 模型概述

本文提出的基于多尺度空洞卷积和 LSTM 的负载预测模型结构如图 3.1(a)所示，主要分为输入模块、多尺度特征提取模块和双通道时序预测模块。



(a) 模型结构　　(b) 多尺度特征提取块结构

图 3.1 模型结构图

各个模块介绍如下：

（1）**输入模块**。本文提出的模型有两个不同的输入，一个输入是历史负载单变量序列$\{X_{1:T}\}$，将输入到多尺度特征提取模块以提取基本时序特征。另一个

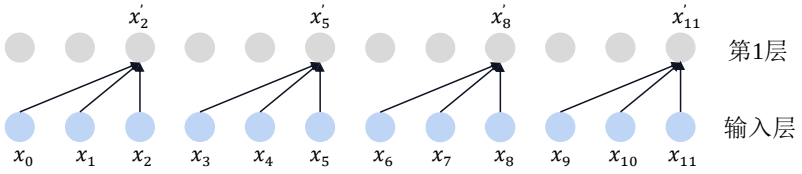
输入是历史负载序列经过周期性时间编码后的多变量输入$\{X_{1:T}, P_{1:T}\}$，将输入到多尺度特征提取块以提取周期时序特征。

（2）**多尺度特征提取模块**。该模块结构如图 3.1(b)所示，由平行和堆叠的一维空洞卷积网络和残差连接网络组成，用于从输入的多个子序列中提取多尺度时序特征。空洞卷积网络中的标准卷积层和空洞卷积层 1 用于学习连续输入序列的短期波动特征和长期特征，空洞卷积层 2 和空洞卷积层 3 用于学习非连续时间步之间的依赖关系。

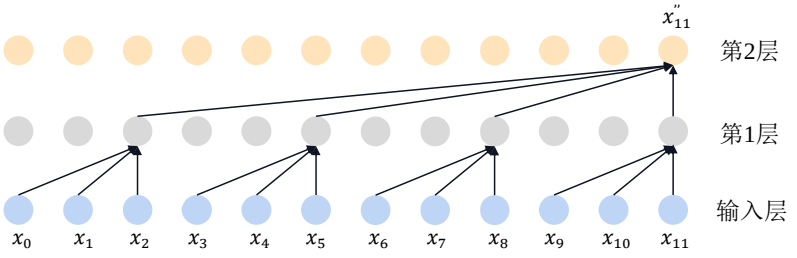
（3）**双通道时序预测模块**。该模块通过两个通道来分别处理不同的输入。每个通道中，对多尺度特征提取块的输出结果利用 LSTM 网络进一步学习时序特征。最后将两个通道提取的基本时序特征和周期时序特征融合，经前馈神经网络输出预测序列，完成负载预测任务。

3.2.2 多尺度特征提取块

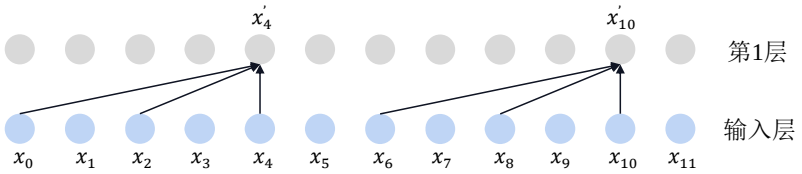
1D-CNN 可以在时间序列的每一个时间步上进行卷积操作，提取特征序列，以帮助 LSTM 学习到更具有代表性的特征。受 pCNN-LSTM 模型[37]启发，本节提出了基于 1D-CNN 的多尺度特征提取块。如图 3.1(b)所示，多尺度特征提取块主要由四个 1D-CNN 层和残差连接块组成，1D-CNN 层分别是标准卷积层、空洞卷积层 1、空洞卷积层 2 和空洞卷积层 3，卷积层之间呈平行或堆叠关系。



(a) 标准卷积层



(b) 空洞卷积层 1



(c) 空洞卷积层 2 和空洞卷积层 3

图 3.2 不同卷积核大小和膨胀率的感受野示意图

（1）卷积层

如图 3.2 所示，为了从历史负载序列输入中学习不同的变化模式，本文使用不同的卷积核大小和膨胀率，为每个卷积层构建不同长度的感受野。图中，假设用具有 12 个历史时间步长的序列作为输入，模型的实际输入使用更多的历史值。

图 3.2(a)展示了标准卷积层的感受野，当卷积核大小为 3 个时间步长，膨胀率设置为 1 时，$x_2', x_5', x_8', x_{11}'$的感受野是 3 个时间步长。图 3.2(b)展示了空洞卷积层 1 的感受野。第 1 层是具有 3 个时间步长感受野的标准卷积层，在第 2 层中，卷积核大小为 4 个时间步长，膨胀率设置为 3，这使x_{11}''的感受野从整个输入中学习特征，这是通过堆叠两个卷积层获得的。图 3.2(c)展示了空洞卷积层 2 和空洞卷积层 3 的感受野。卷积核大小同样为 3 个时间步长，但膨胀率设置为 2，此时x_4'和x_{10}'的感受野是 5 个时间步长，但与从连续输入值中提取特征的标准卷积层不同，空洞卷积层 2 和空洞卷积层 3 学习非连续输入值之间的相关性。

综上所述，多尺度特征提取块中的四个 1D-CNN 层，当使用卷积核大小为k和膨胀率为d的空洞卷积运算时，感受野（RF）可以用式（3.5）表示：

$$RF(k,d) = \left(x_{j-(k-1)d}, x_{j-(k-2)d}, \dots, x_{j-d}, x_j\right), 0 \le j < p \tag{3.5}$$

其中，p是输入序列的长度。在本文中，会对每个输入样本分成r个子序列，p就是指子序列的长度。

如图 3.2(a)所示，标准卷积层学习输入序列中的短期特征。由式（3.5）可得，卷积核大小为$k_{RegularConv}$，膨胀率为 1 的标准卷积层的感受野为式（3.6）：

$$RF_{RegularConv} = RF\left(k = k_{RegularConv}, d = 1\right), 1 \le k_{RegularConv} \ll p \tag{3.6}$$

如图 3.2(b)所示，空洞卷积层 1 叠加在标准卷积层上，并使用与前一层卷积核大小相等的膨胀率来构建横跨整个历史窗口的感受野。这使模型能够通过在连续的层上构建特征层次结构，来学习更大时间跨度内的负载使用模式。空洞卷积层 1 的感受野表示为式（3.7）：

$$\begin{gathered} RF_{DilatedConv1} = RF\left(k = k_{DilatedConv1}, d = k_{RegularConv}\right), \\ k_{DilatedConv1} = p \; div \; d \end{gathered} \tag{3.7}$$

如图 3.2(c)所示，空洞卷积层 2 和空洞卷积层 3 对模型输入序列使用了空洞卷积，学习非连续时间步之间的依赖关系。对输入序列使用空洞卷积的目的是为了学习较低级别的周期性负载特征。空洞卷积层 2 的感受野表示为式（3.8）：

$$\begin{gathered} RF_{DilatedConv2} = RF(k = k_{DilatedConv2}, d = k_{DilatedConv2}), \\ 1 < d_{DilatedConv2} < p, k_{DilatedConv2} = p \; div \; d_{DilatedConv2} \end{gathered} \tag{3.8}$$

空洞卷积层 3 的感受野表示为式（3.9）：

$$\begin{gathered} RF_{DilatedConv3} = RF(k = k_{DilatedConv3}, d = d_{DilatedConv3}), \\ 1 < d_{DilatedConv3} < d \end{gathered} \tag{3.9}$$

以上k和d的取值是为了使感受野在历史负载序列中传递有意义的依赖关系。通过从连续的时间步长学习，标准卷积层对短期时序特征建模，而空洞卷积层 1 从整个输入中学习长期时序特征。通过使用大于 1 的膨胀率，空洞卷积层 2 和空洞卷积层 3 提取了非连续时间步之间的相关性，使模型能够学习每隔几个时间步出现的较低级别周期模式。

（2）残差连接层

提取到感兴趣的特征后，四个卷积层的特征输出被连接起来，并与输入层数据进行残差连接。由于输入层向量维度与特征连接后的维度不同，先对输入层进行线性投影，再与特征连接后的向量相加。残差连接确保通过网络的后续层获得梯度信息，从而避免了深度神经网络的退化问题[64]。相加后的特征经过层归一化层[71]转化，将一个批次的每个训练样本的特征转换为符合均值为 0，方差为 1 的分布，从而加速模型训练并提高模型的泛化能力。再通过 ReLU 激活函数和 Dropout 层，最后输出为提取到的多尺度时序特征。

3.2.3 周期性模式提取

时间序列的周期性是指数据中存在重复出现的模式，这些模式可以是固定周期的（如每天、每周、每月或每年）或者非固定周期的（如季节性或事件性）。周期性是时序数据中不可忽视的重要特征，对于时序预测具有重要的影响。周期性的存在可以帮助人们更好地理解时序数据的内部变化规律，从而准确地预测未来的变化趋势。通常情况下，周期性是指单一的周期，例如一个时间序列可能会呈现出每年一次的季节性波动或者每周一次的周期性波动。然而，时间序列可能会同时表现出多个周期性的变化，例如同一时间序列可能会同时呈现出每年一次、每月一次和每周一次的周期性波动。并且，不仅同一个周期内部的时间点存在依赖关系（例如今天 1 点和 2 点），相邻周期内的时间点也存在依赖关系（例如今天 1 点和明天 1 点）。多周期性的存在使得时间序列分析变得更加复杂，因为多个周期性的影响可能会相互叠加，导致难以进行有效的预测和分析。

自相关分析是一种常用的时间序列分析方法，可以用来检验时间序列中的周期性。自相关分析是指计算时间序列在不同时间点的自相关系数（Autocorrelation Function，ACF），即观察时间序列在不同时间点之间的相关性。如果时间序列存在周期性，那么在特定的滞后期间内，自相关系数通常会显示出较强的相关性。使用自相关分析方法分析时间序列周期性的步骤如下：

（1）计算自相关系数。可以通过计算时间序列在不同时间点的自相关系数来检验其周期性。自相关系数的取值范围为-1 到 1 之间，当取值为 0 时，表示两个值之间没有相关性；当取值为 1 或-1 时，分别表示两个值之间完全正相关或负

相关。偏自相关系数则是在自相关系数的基础上去除了中间时间点的干扰，仅衡量每个时间点与当前时间点之间的相关程度。具体地，自相关系数的计算公式为：

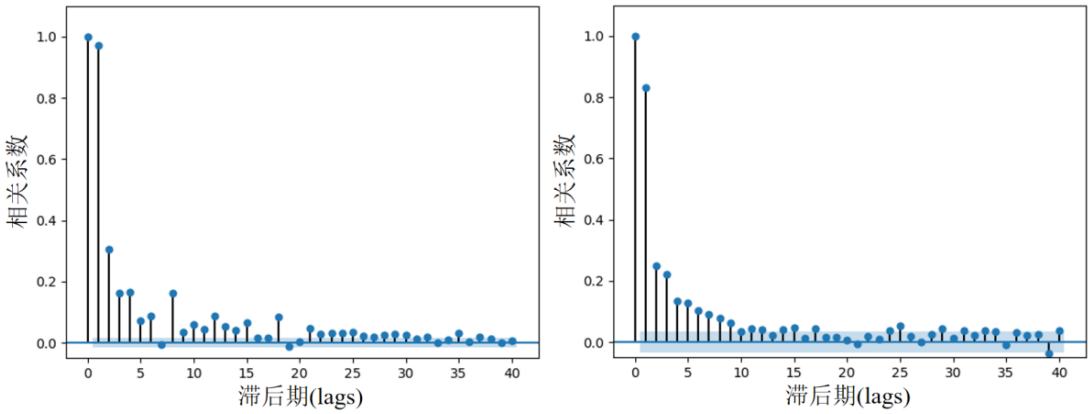
$$ACF(k) = \frac{\sum_{t=1}^{n-k}(x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2} \tag{3.10}$$

其中，k表示滞后期间，x_t表示时间序列在第t个时间点的取值，$\bar{x}$表示时间序列的平均值。

（2）绘制自相关系数图。通过绘制自相关系数图，可以观察自相关系数在不同滞后期间的变化情况。并计算自相关图的置信区间，用于判断数据样本自相关系数是否与总体自相关系数有显著不同。

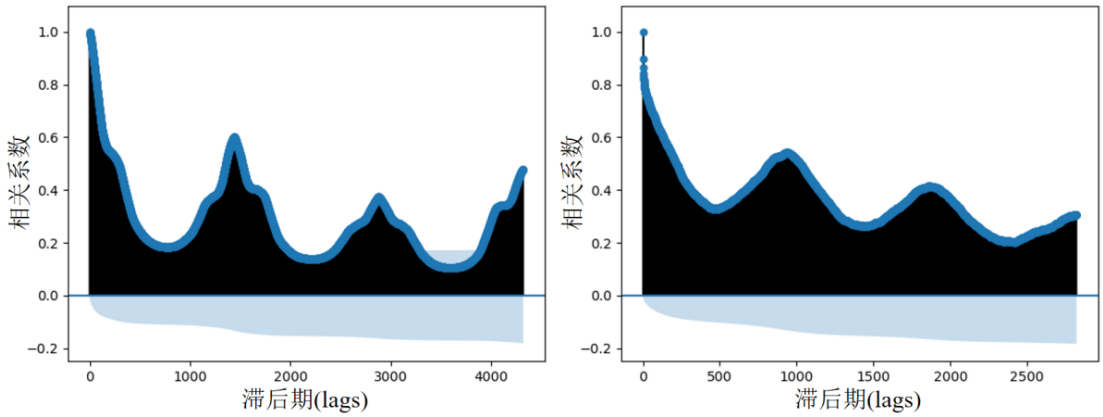
（3）判断是否具有显著周期性。通过比较自相关系数与置信区间的关系来判断时间序列是否具有显著周期性。如果自相关系数超过其置信区间，则表示该时间序列越倾向于在一定时间段内出现重复模式，即时间序列具有显著的周期性。然而，如果自相关系数在置信区间内，则表示时间序列没有显著的周期性。

（4）确定周期。可以根据自相关系数图中的高峰和低谷来确定时间序列的周期，通过计算滞后期间的平均值或最大值来确定时间序列的周期。



(a) WorldCup 数据集 Web 请求率　　(b) Alibaba Cluster 数据集 CPU 利用率

图 3.3 偏自相关图



(a) WorldCup 数据集 Web 请求率　　(b) Alibaba Cluster 数据集 CPU 利用率

图 3.4 自相关图

以世界杯 Web 请求数据集（WorldCup）和阿里巴巴集群数据集（Alibaba Cluster）为例，分别对其中的 Web 请求率和 CPU 利用率进行相关性分析。图 3.3 是两个数据集上滞后期间为 40 个时间步的偏自相关图，由图可知 WorldCup 数据集中的 Web 请求率之间的显著相关性相隔 6 个时间步长，而 Alibaba Cluster 数据集中的 CPU 利用率之间的显著相关性高达 12 个时间步长。类似地，图 3.4 是两个数据集上滞后期间为 3 天的自相关图，均显示了每日负载的变化规律。

自相关图像有助于深入了解云负载的行为模式。本文提出的负载预测模型，通过多尺度特征提取块中的空洞卷积层 2 和空洞卷积层 3 学习相隔几个时间步的较低级别的周期性负载模式，通过对每日和每周进行周期性时间编码学习更高级别的周期性负载模式，从而对负载序列的多周期性进行建模。经过以上对数据集中历史负载序列的自相关分析，观察序列中的多周期性，以此指导空洞卷积层 2 和空洞卷积层 3 的卷积核大小和膨胀率的设置以及每日和每周周期性时间编码的计算。

（1）时间步间隔级别的周期性模式

在多尺度特征提取块中，空洞卷积层 2 和空洞卷积层 3 对输入序列使用了膨胀率大于 1 的空洞卷积，学习非连续时间步之间的依赖关系，使模型能够学习相隔几个时间步的较低级别的周期性特征。本文实验使用的负载数据集中，单个时间步长时间间隔有两种，分别是 1 分钟和 5 分钟。通过对自相关图的分析，设置式（3.8）中空洞卷积层 2 的膨胀率为 6，则表示对于时间步长为 1 分钟的数据集学习每 6 分钟时间间隔的负载周期模式，对于时间步长为 5 分钟的数据集学习每 30 分钟时间间隔的负载周期模式。设置式（3.9）中空洞卷积层 3 的膨胀率为 12，则表示对于时间步长为 1 分钟的数据集学习每 12 分钟时间间隔的负载周期模式，对于时间步长为 5 分钟的数据集学习每 60 分钟时间间隔的负载周期模式。

（2）每日和每周级别的周期性模式

在时间序列中，月、日、时、分这些周期性的变量如果单纯用数字作为特征是不合理的。例如，23 时和凌晨 1 时两者相差两小时，单纯用 23 和 1 这样的数字作为特征，会存在跳跃而不连续，若模型不进行周期性编码，则会将时间差误编码为 1-23=-22，影响学习的结果。因此，有必要对周期性变量进行周期性时间编码。如前所述，本文将历史负载序列经过周期性时间编码后的多变量数据$\{X_{1:T}, P_{1:T}\}$作为模型输入之一，输入到多尺度特征提取块以提取每日和每周级别的周期性模式。

本文采用的方法是对每个数据样本的时间戳t经过正弦和余弦变换，将一维时间数值变量转化为二维的正弦值和余弦值编码，如式（3.11）所示：

$$t_{\sin} = \sin\left(\frac{2\pi t}{period}\right), t_{\cos} = \cos\left(\frac{2\pi t}{period}\right) \tag{3.11}$$

其中，$period$表示期望编码的周期长度。

正弦和余弦函数遵循 0 到2π的循环，值在-1 到 1 之间。如图 3.5 所示，一天 24 小时可以转化为具有周期信息的二维编码，正弦和余弦函数具有的周期性保证了 23 时和次日 0 时的时间间隔是 1 小时，而不是 23 小时。

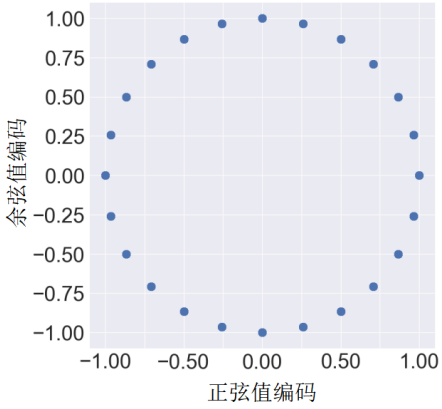


图 3.5 每日周期编码

3.2.4 双通道时序预测

如图 3.1(a)所示，为了提取输入中的短期波动、持续负载变化和周期性模式，本文设计了基本特征学习通道和周期特征学习通道来处理不同的输入。基本特征学习通道输入原始历史负载序列，学习基本时序特征；周期特征学习通道输入经过周期时间编码的历史负载序列，学习更高级别的每日每周周期性时序特征。首先，每个通道的输入经过多尺度特征提取块，以学习短期特征、长期特征和较低级别的时间步间隔周期性特征。提取多尺度特征后，再由各自的 LSTM 层进一步学习时序模式。相较于直接使用 LSTM 从长时间序列中学习时间模式的方法，这种连接方式只需要学习由多尺度特征提取块提取的较小特征序列之间的依赖关系，从而减少了训练时间，同时也改善了 LSTM 遗忘机制使信息指数衰减，导致难以捕获长时间尺度信息的问题。最后，通过连接层对两个通道的输出进行融合，并经全连接层非线性变化输出预测序列。

3.3 实验设计与结果分析

3.3.1 实验数据集和参数设置

为了评估本文提出的负载预测模型的预测效果，本文使用了在负载预测任务中主流的数据集，包括美国航天局 Web 请求数据集（NASA）[2]、世界杯 Web 请

[2] https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

求数据集（WorldCup）[3]、谷歌集群数据集（Google Cluster）[4]和阿里巴巴集群数据集（Alibaba Cluster）[5]。其中，WorldCup 和 NASA 包含的是 Web 请求数据，属于应用业务指标，能够反映应用程序的服务质量。而 Google Cluster 和 Alibaba Cluster 数据集包含的是 CPU、内存、网络 IO 等数据，属于资源使用指标。考虑到 CPU 利用率指标相对更能反映 Web 服务的系统资源负载，且在工业界使用较多，因此对其中的 CPU 利用率进行预测。各数据集具体介绍如下：

（1）**NASA**：NASA 请求日志数据集是由美国航天局收集的 Web 服务器请求数据，包含了从 1995 年 7 月至 8 月共 3,461,612 项的请求数据。本文通过将同一分钟内发生的所有日志聚合到一个累积记录中来预处理原始数据集，得到每分钟的总请求数。

（2）**WorldCup**：WorldCup 数据集记录了 1998 年 4 月至 7 月期间访问世界杯网站的请求日志。在此期间，该网站收到了 1,352,804,107 项请求。该数据集的分布比 NASA 数据集更复杂，它没有像 NASA 数据集那样有显著的时间序列模式，并且有较多的异常值和难以预测的峰值。与 NASA 数据集类似，本文通过聚合日志的方式对 WorldCup 数据集进行预处理，得到每分钟总请求数。

（3）**Google Cluster**：Google Cluster 数据集是谷歌发布的真实集群跟踪数据集，包含了约 12500 台服务器在 29 天内的运行数据，包括资源消耗情况、任务调度信息等[72]。数据集的采样时间间隔为 5 分钟，本文从中随机选择了一台服务器的 CPU 负载数据来进行实验。

（4）**Alibaba Cluster**：Alibaba Cluster 数据集是阿里云发布的真实生产环境集群跟踪数据集，包含了 4000 多台服务器和服务器中的容器在 8 天内的资源使用情况[73]。与 Google Cluster 数据集仅包含服务器指标不同，Alibaba Cluster 数据集包含了容器的负载。本文根据每个容器对应的微服务 ID，将属于同一微服务的容器 CPU 利用率合并为对应微服务的 CPU 利用率。数据集原始采样的时间间隔为 10 秒，本文将 6 个连续的时间点合并，即一分钟为一个时间间隔。

在进行训练之前，首先需要对数据集进行预处理。本文采用滑动窗口的方式构造数据集，并按时间顺序分成训练集、验证集和测试集，划分比例为 7:1:2。然后对数据进行归一化处理以提高模型的学习效果。为了提高模型对周期特征的学习能力，需要对周期特征学习通道的输入数据进行周期性时间编码。本文根据式（3.11）将周期分别设置为 24*60*60 和 7*24*60*60，对时间戳进行计算，得到每日和每周的编码，并将其作为周期特征加入到输入数据中。每个输入样本的历史序列长度为 144 个时间步，分割成长度为 24 个时间步的 6 个子序列后输入网

[3] https://ita.ee.lbl.gov/html/contrib/WorldCup.html
[4] https://github.com/google/cluster-data
[5] https://github.com/alibaba/clusterdata

络。为了验证模型在不同预测长度下的性能，实验将预测长度的时间步设置为6,12,18,24。

模型的超参数设置如下。特征提取块中，标准卷积层的卷积核大小设置为6，三个空洞卷积层的卷积核大小分别设置为4、4、3，膨胀率分别设置为6、6、12，Dropout 设置为 0.01。LSTM 层数设置为 1，每个 LSTM 层的隐藏节点数量设置为 128，Dropout 设为 0.01。LSTM 层后的全连接层的维度设置为 128。输出全连接层的输出维度设置为与预测长度相同。在训练过程中，采用 Adam 优化器进行模型优化，学习率设置为 0.01，批大小设置为 64。为了防止模型过拟合，采用早期停止技术，当模型在验证集上的性能连续 5 轮没有上升时，训练就会停止。采用均方误差和 L2 权重正则化作为损失函数，以进一步减小过拟合的风险。实验重复三次，结果取平均值。本文实验使用的测试机器为一台具有 RTX2060 显卡台式工作站，采用深度学习框架 Keras 搭建负载预测模型。对于对比模型均采用对应论文中所指出的模型参数。

3.3.2 对比模型

为了评估提出的负载预测模型的性能，本文选择了 LSTM、GRU、LSTM-ED、TCN、Transformer 和 pCNN-LSTM 六种预测模型作为对比模型。并设置了单通道版本的本文模型作为消融对照组，以验证本文提出的各模块在模型中的功能：

（1）**LSTM**[13]：2018 年由 Song 等人将 LSTM 应用于负载预测任务中。该方法利用 LSTM 的门控机制学习时序依赖，使用单个 LSTM 层生成多个时间步长度的预测序列。

（2）**GRU**[15]：2019 年由 Cheng Y 等人将 GRU 应用于负载预测任务中。该方法与 LSTM 相比门控机制的数量更少，需要更少的参数来训练。

（3）**LSTM-ED**[18]：2019 年由 Nguyen 等人将编码器-解码器架构的 LSTM 应用于负载预测任务中，利用编码器将输入序列编码成一个固定维度的向量，解码器通过接收该输出向量和前一个时间步的预测结果，生成多时间步预测结果。

（4）**TCN**[22]：2021 年由 Golshani 等人将 TCN 应用于负载预测任务中。该方法利用因果空洞卷积扩大感受野，并通过堆叠不同尺度的卷积层在多个时间尺度上捕捉序列的不同特征。

（5）**Transformer**[26]：2019 年由 Li S 等人将 Transformer 应用于时序预测任务中。该方法针对时序数据预测任务的特点，在 Transformer 的基础上改进了注意力机制的计算方式以适应时序数据。

（6）**pCNN-LSTM**[37]：2022 年由 Patel 等人提出并应用于负载预测任务中。该方法使用三个平行的具有不同卷积核大小和膨胀率的因果空洞卷积层，在多尺

度上学习输入特征，再由 LSTM 层提取时间依赖性。该方法的卷积层使用与预测序列长度相对应的长度的感受野。

（7）**单通道版本的本文模型**：仅保留了基本特征学习通道，去除了周期特征学习通道。作为消融对照组，与 pCNN-LSTM 对比验证多尺度特征提取块的有效性，与双通道版本的本文模型对比验证周期特征学习通道的有效性。

3.3.3 评价指标

本文选取了均方误差（Mean Square Error，MSE）、均方根误差（Root Mean Square Error，RMSE）和平均绝对误差（Mean Absolute Error，MAE）三个指标来衡量模型的预测精度：

（1）**MSE** 是预测值与真实值之间误差的平方和的平均值，衡量的是误差的方差，表示为式（3.12）：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.12}$$

（2）**RMSE** 是 MSE 的平方根，衡量的是误差的标准偏差。RMSE 的值与原始数据单位相同，更容易被理解和解释，表示为式（3.13）：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.13}$$

（3）**MAE** 是预测值与真实值之间误差的绝对值的平均值，衡量的是误差的平均值，表示为式（3.14）：

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.14}$$

以上指标公式中的n为测试样本数，y_i为真实值，$\hat{y}_i$为预测值。以上指标都在计算误差，结果值越小，说明模型预测准确率越高。

3.3.4 实验结果与分析

本文在四个数据集上比较了不同预测时间步长的未来负载预测结果，并与对比模型进行比较，其中在 NASA 和 WorldCup 数据集上预测请求率负载，在 Google Cluster 和 Alibaba Cluster 数据集上预测 CPU 利用率负载。NASA 和 WorldCup 数据集上的实验结果如表 3.1 所示，Google Cluster 和 Alibaba Cluster 数据集上的实验结果如表 3.2 所示。从表中可以得出，在所有预测长度上，本

文提出的模型在 MSE、RMSE、MAE 三个指标上均优于其他对比模型，具有更高的预测精度。本文模型的总体 MSE 指标在 NASA 数据集上相对于最佳对比模型实现了 8.81%的改进，在 WorldCup 数据集上实现了 9.09%的改进，在 Google Cluster 数据集上实现了 8.08%的改进，在 Alibaba Cluster 数据集上实现了 7.52%的改进，具体实验分析如下。

表 3.1 在 NASA 和 WorldCup 数据集上模型结果对比（单位：10^{-2}）

模型	预测步长	NASA 数据集			WorldCup 数据集		
		MSE	RMSE	MAE	MSE	RMSE	MAE
LSTM	6	0.241	0.491	2.987	0.394	0.628	4.820
	12	0.291	0.539	3.299	0.439	0.663	4.804
	18	0.326	0.571	3.575	0.513	0.716	5.371
	24	0.355	0.596	3.710	0.560	0.748	5.618
GRU	6	0.244	0.494	3.144	0.386	0.621	4.790
	12	0.285	0.534	3.338	0.432	0.657	4.908
	18	0.324	0.569	3.575	0.501	0.707	5.367
	24	0.352	0.593	3.831	0.551	0.742	5.635
LSTM-ED	6	0.231	0.481	2.978	0.362	0.601	4.115
	12	0.282	0.531	3.303	0.408	0.639	4.477
	18	0.318	0.564	3.506	0.472	0.687	4.914
	24	0.340	0.583	3.701	0.515	0.718	5.222
TCN	6	0.216	0.464	2.832	0.327	0.571	3.899
	12	0.266	0.516	3.112	0.408	0.638	4.503
	18	0.295	0.543	3.313	0.441	0.664	4.602
	24	0.309	0.556	3.406	0.488	0.698	4.942
Transformer	6	0.208	0.456	2.726	0.318	0.564	4.078
	12	0.258	0.508	3.018	0.401	0.633	4.651
	18	0.271	0.520	3.045	0.430	0.656	4.781
	24	0.278	0.527	3.067	0.441	0.664	4.846
pCNN-LSTM	6	0.204	0.452	2.532	0.309	0.556	3.823
	12	0.257	0.506	2.920	0.393	0.627	4.405
	18	0.283	0.532	3.098	0.444	0.666	4.760
	24	0.289	0.538	3.151	0.475	0.690	4.916
本文模型（单通道）	6	0.193	0.440	2.402	0.299	0.547	3.826
	12	0.237	0.486	2.739	0.369	0.607	4.312
	18	0.270	0.519	2.956	0.422	0.650	4.749
	24	0.281	0.530	3.075	0.451	0.672	4.973
本文模型（双通道）	6	**0.187**	**0.432**	**2.396**	**0.291**	**0.540**	**3.634**
	12	**0.226**	**0.475**	**2.682**	**0.358**	**0.598**	**4.076**
	18	**0.255**	**0.505**	**2.950**	**0.393**	**0.627**	**4.187**
	24	**0.274**	**0.524**	**2.995**	**0.432**	**0.657**	**4.442**

表 3.2 在 Google Cluster 和 Alibaba Cluster 数据集上模型结果对比（单位：10^{-2}）

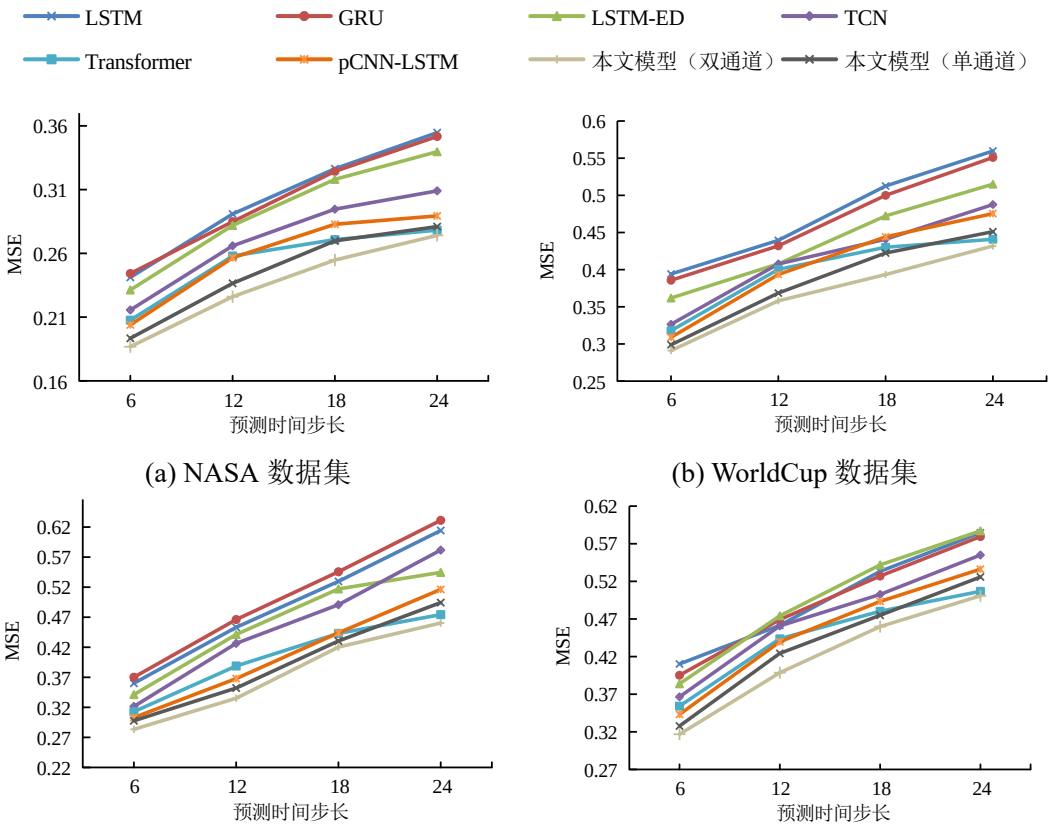
模型	预测步长	Google Cluster 数据集			Alibaba Cluster 数据集		
		MSE	RMSE	MAE	MSE	RMSE	MAE
LSTM	6	0.360	0.600	4.459	0.410	0.640	4.666
	12	0.453	0.673	5.139	0.461	0.679	5.066
	18	0.530	0.728	5.803	0.533	0.730	5.548
	24	0.614	0.784	6.425	0.585	0.765	5.927
GRU	6	0.370	0.608	4.616	0.395	0.629	4.688
	12	0.466	0.683	5.306	0.469	0.685	5.198
	18	0.546	0.739	5.976	0.527	0.726	5.409
	24	0.631	0.794	6.572	0.580	0.761	5.813
LSTM-ED	6	0.341	0.584	4.393	0.384	0.620	4.694
	12	0.441	0.664	5.170	0.474	0.689	5.184
	18	0.517	0.719	5.699	0.542	0.736	5.679
	24	0.545	0.738	5.933	0.587	0.766	5.893
TCN	6	0.321	0.567	4.222	0.367	0.605	4.376
	12	0.426	0.653	4.990	0.461	0.679	5.090
	18	0.491	0.701	5.518	0.503	0.709	5.246
	24	0.582	0.763	6.412	0.555	0.745	5.625
Transformer	6	0.313	0.559	4.012	0.354	0.595	4.451
	12	0.389	0.624	4.711	0.444	0.666	5.092
	18	0.443	0.666	5.130	0.480	0.693	5.007
	24	0.474	0.689	5.321	0.507	0.712	**5.386**
pCNN-LSTM	6	0.303	0.550	3.763	0.343	0.586	4.246
	12	0.368	0.606	4.187	0.440	0.663	4.922
	18	0.444	0.666	4.864	0.493	0.702	5.287
	24	0.516	0.718	5.620	0.536	0.732	5.544
本文模型（单通道）	6	0.298	0.546	3.719	0.328	0.572	4.091
	12	0.352	0.593	4.167	0.424	0.651	4.855
	18	0.430	0.656	4.884	0.475	0.689	5.025
	24	0.494	0.703	5.546	0.526	0.725	5.418
本文模型（双通道）	6	**0.283**	**0.532**	**3.507**	**0.317**	**0.563**	**4.064**
	12	**0.335**	**0.579**	**3.751**	**0.399**	**0.631**	**4.630**
	18	**0.420**	**0.648**	**4.504**	**0.460**	**0.678**	**4.830**
	24	**0.460**	**0.679**	**4.760**	**0.501**	**0.708**	5.408

对比实验中，LSTM 和 GRU 可以在一定程度上学习长期的非线性时间依赖关系，具有基本的时序预测能力。LSTM-ED 采用编码器解码器结构，避免了直接建模长序列的复杂性，因此相比于 LSTM 和 GRU 的结果有所提升，但在生成多步预测时，每个预测步中的误差会逐步累积。然而，LSTM、GRU 和 LSTM-ED 都是通过状态传递来进行特征提取，对输入数据的局部特征提取能力相对较弱，无法有效地捕捉输入数据中的关键特征，因此在对比方法中取得了较差的预

测结果。而 TCN 通过因果空洞卷积扩大感受野，使其能够从非常长的历史中学习时序特征，并且可以通过堆叠不同尺度的卷积层在多个时间尺度上捕捉序列的不同特征，因此在实验中取得了比 LSTM、GRU 和 LSTM-ED 更好的预测性能，这表明利用 1D-CNN 挖掘时序特征在负载预测任务中的有效性。Transformer 利用了注意力机制，可以更好地捕捉序列中的全局依赖关系，从而在长时间序列预测问题中表现更好。因此模型随着预测步长的增加，预测性能相对于其他模型逐渐提升，但在预测步长较短时的预测能力有限。pCNN-LSTM 和本文提出的模型都融合了 1D-CNN 块提取时序特征，使得 LSTM 能够更有效地学习输入特征，因此模型的预测性能相比于其他对比模型有了更大的提升，这说明 1D-CNN 和 LSTM 的有效组合在负载预测任务中具有更好的学习能力。

pCNN-LSTM 和本文提出的模型都是基于 1D-CNN 和 LSTM 的混合预测模型，先使用基于空洞卷积的 CNN 层提取多尺度特征，再利用 LSTM 进一步学习提取的时序特征中的时间依赖性，充分利用两者的优势。pCNN-LSTM 的 1D-CNN 层由三个平行的空洞 CNN 层组成，具有不同卷积核大小和膨胀率，从而使模型能学习不同尺度的负载特征。本文提出的模型的多尺度特征提取块利用平行和堆叠的空洞卷积残差网络，从连续和非连续的输入时间步中提取时序特征，从而同时学习负载短期波动、持续性变化和时间步级别周期性信息。本文提出的模型的多尺度特征提取块相较于 pCNN-LSTM 的 1D-CNN 层而言，pCNN-LSTM 的 1D-CNN 层使用与预测长度相对应的长度的感受野，忽略了输入序列中的全局信息。而本文提出的模型的多尺度特征提取块从整个输入子序列中提取特征，而不是仅限于特定预测长度的特征。因此，本文提出的单通道版本模型的预测性能优于 pCNN-LSTM，证明了多尺度特征提取块的有效性。此外，pCNN-LSTM 并没有考虑历史负载中的周期性。而本文提出的模型通过自相关分析方法分析时间序列中的周期性，利用多尺度特征提取块中的空洞卷积层提取时间步间隔级别的周期性模式，并将历史负载序列经过周期性时间编码后作为模型输入之一，提取每日和每周级别的周期性模式。因此，本文提出的双通道版本模型在所有对比模型中达到了最好的预测性能，这表明周期时间模式提取能够有效提升预测准确性。

针对不同数据集上的所有预测长度，本研究对比了每种预测方法的 MSE 指标。如图 3.6 所示，与许多多步预测问题一样，对于较短的序列，各预测方法可以获得较低的 MSE，但随着预测长度的增加，MSE 也逐渐增大。这是因为，序列值之间的相关性随时间推移而逐渐降低，使得预测模型很难捕捉到较长序列中的弱依赖关系。此外，从图中还可以得出，使用 LSTM 和 CNN 相结合来学习时序特征的预测模型比纯粹依赖于 LSTM 对时间序列建模的预测模型具有更好的预测性能。



(a) NASA 数据集　(b) WorldCup 数据集

(c) Google Cluster 数据集　(d) Alibaba Cluster 数据集

图 3.6 MSE 结果图

3.4 本章小结

云负载数据时序变化模式多样，表现出波动性、持续性和多周期性的特点，给精准预测未来负载带来了巨大挑战。为了有效捕获这些负载变化特征，本章提出了基于多尺度空洞卷积和 LSTM 的负载预测模型。模型采用平行和堆叠的空洞卷积残差网络构成多尺度特征提取块，以学习连续和非连续输入时间步的时序特征，再通过 LSTM 进一步对长期时间依赖进行建模。多尺度特征提取块通过使用卷积核大小和膨胀率的创新组合，使模型可以同时利用负载序列中的自相关和偏自相关信息。模型设计了两个通道来处理不同的输入，基本特征学习通道输入原始历史负载序列，学习短期波动和持续性等基本负载特征；周期特征学习通道输入经过周期时间编码的历史负载序列，学习多周期性负载特征。在四个云负载数据集进行了不同预测时间步长的负载预测实验。实验结果表明，本章提出的模型在 MSE、RMSE、MAE 三个指标上均优于对比模型，具有更高的预测精度。

第 4 章 基于负载预测和强化学习的混合弹性伸缩策略研究

弹性伸缩是容器云平台的重要特性，它可以根据当前资源和负载情况自动调整 Pod 服务实例的数量，以满足业务需求。在弹性伸缩技术中，准确确定合理的伸缩动作是至关重要的。确定合理的伸缩动作需要考虑多种因素，包括负载波动情况、系统资源使用情况、成本和性能之间的平衡以及可靠性等因素，而不合理的伸缩动作可能会导致系统性能下降或资源浪费，因此，确定合理的伸缩动作成为了研究的重要问题。

目前，现有研究主要关注于基于负载预测和基于强化学习的弹性伸缩策略。基于负载预测的弹性伸缩策略通过预测未来的负载情况来决定是否进行伸缩操作。虽然该方法可以提前配置资源以适应未来的负载变化，但通常作为阈值控制方法的补充，仍然采用规则来确定实例个数，难以适应复杂场景。相对而言，基于强化学习的伸缩策略则可以更加智能地确定伸缩动作，因为它可以自适应地学习当前的环境状态并相应地调整策略。但是，基于强化学习的伸缩策略需要设计合理的状态空间、动作空间和奖励函数，并且通常只考虑当前状态信息而无法提前感知状态变化，因此其学习效率较低，在实际场景中应用时存在困难。

为了克服基于负载预测和基于强化学习的弹性伸缩策略各自存在的缺点，本章提出了一种新的主动式伸缩策略。该策略对两种方法的优点进行融合，根据应用的当前资源负载状态和预测信息，由强化学习智能体做出合理的弹性伸缩决策，能够更加准确和智能地调整 Pod 服务实例个数。

另外，在应对极端场景下的异常突发流量时，主动式伸缩策略可能存在一定的不足。相对于响应式伸缩策略，主动式伸缩策略不能及时反映当前的状态，从而导致其可能失去有效性。因此，使用单一的主动式伸缩策略存在一定的风险，易对服务的可用性造成影响。针对这一问题，需要探索更加全面和完善的伸缩策略体系，以应对不同场景下的需求变化。

为解决以上问题，本章进一步地提出了一种主动式和响应式的混合弹性伸缩策略。该策略的主动式伸缩模块进行对未来的主动资源规划；响应式伸缩模块考虑了请求响应时间和资源利用率，作为主动式伸缩策略失效时的备选方案，以应对极端流量峰值时进行及时响应和调整；最后，混合伸缩控制器协调以上两种伸缩策略，结合主动式和响应式伸缩策略的优点，实现兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

4.1 混合弹性伸缩策略框架

为了准确做出合理的弹性伸缩动作，并兼顾主动伸缩决策和在极端峰值流量下的有效伸缩，本文提出了基于负载预测和强化学习的水平混合弹性伸缩策略。该策略的优化目标是最小化服务水平协议（SLA）违规率和最大化资源利用率，即在满足服务 SLA 的基础上降低运行成本。图 4.1 展示了该策略的整体框架，主要由负载预测模块、主动式伸缩模块、响应式伸缩模块和混合伸缩控制器组成。

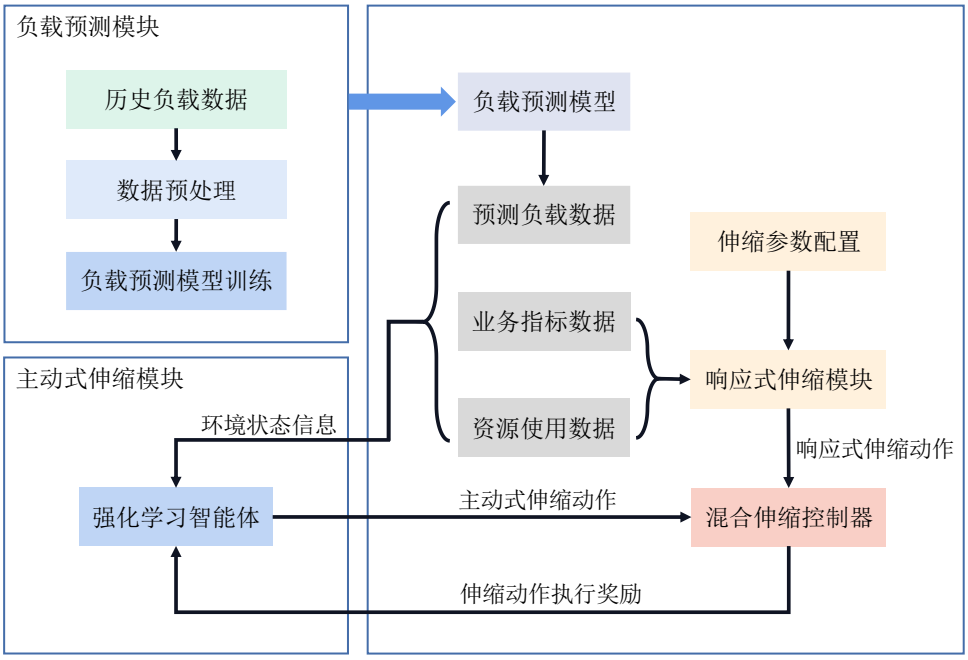


图 4.1 基于负载预测和强化学习的水平混合弹性伸缩策略框架图

各部分介绍如下：

（1）负载预测模块。负载预测模块采用本文第 3 章提出的预测模型在历史负载序列数据上进行离线训练，得到训练完成的模型。并在每次伸缩周期，用训练完成的模型对未来的资源使用指标或业务指标进行预测。

（2）主动式伸缩模块。主动式伸缩模块根据应用的当前资源使用指标数据（资源利用率）、当前业务指标数据（请求响应时间、请求率）以及负载预测模块提供的预测负载数据，由强化学习智能体做出合理的弹性伸缩决策。

（3）响应式伸缩模块。响应式伸缩模块以当前资源使用指标数据和应用业务指标数据为输入，根据 SLA 响应时间阈值、资源利用率阈值、资源利用率目标值等参数配置，计算得到响应式伸缩动作。响应式伸缩策略作为主动式伸缩策略失效时的备选方案，以应对极端流量峰值时进行及时响应和调整。

（4）混合伸缩控制器。混合伸缩控制器用于协调主动式和响应式伸缩策略并同时发挥两者不同的优点，采用“快扩容慢缩容”和平衡冲突思想，做出最终伸缩决策，兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

4.2 基于深度 Q 网络的主动式伸缩策略设计

如图 4.2 展示了应用于云环境弹性伸缩任务的强化学习框架图，主要是由智能体、容器云环境、状态、伸缩动作和奖励组成。首先，智能体从容器云环境中获取资源使用指标、应用业务指标信息和负载预测值，构成状态信息，通过策略选择弹性伸缩动作。伸缩动作作用于容器云环境，使得服务 Pod 实例个数发生变化，进而各项状态指标发生改变，到达一个新的状态。然后，奖励函数基于资源利用率和 SLA 违规率计算奖励值，并反馈给智能体。最后，智能体根据容器云环境状态变化、选取的伸缩动作和奖励值进行不断学习。智能体会根据优化策略选择新的动作并执行，循环迭代这个过程，不断与环境进行交互学习，最终找到最优的伸缩策略。在下文中，将有针对地设计状态空间、动作空间、奖励函数以及算法流程，使强化学习有效应用于弹性伸缩任务。

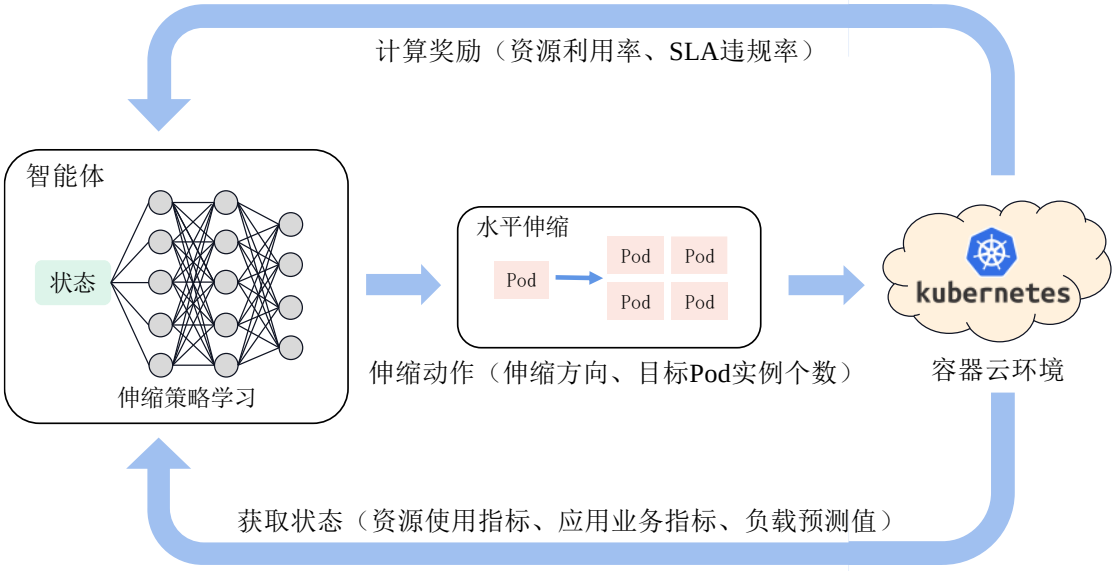


图 4.2 云环境弹性伸缩任务的强化学习框架图

4.2.1 状态空间设计

为了最小化 SLA 违规率和最大化资源利用率，本文将强化学习应用于弹性伸缩任务中。在强化学习中，状态空间是描述智能体所处环境的所有可能状态的集合。状态空间设计的质量直接影响着强化学习算法的性能，合理的状态空间设计必须考虑环境的特征和智能体的任务需求。本节对状态空间进行设计，以有效地表示云应用 Pod 实例的状态，并提供足够的信息以指导策略的决策。

本文考虑了一组可以影响弹性伸缩决策的指标，包括资源使用指标、应用业务指标和 Pod 实例信息。资源使用指标反映了已部署 Pod 的系统资源使用情况，包括 CPU 平均利用率、内存平均利用率。应用业务指标反映了应用的服务质量

情况，包括所提供服务的平均请求响应时间、请求率和 SLA 违规率。Pod 实例信息包括当前处于正常运行状态的 Pod 实例个数。表 4.1 中列出了用于构成状态空间的指标完整列表。

表 4.1 用于构成状态空间的指标

指标名称	指标描述	单位	指标类型
CPU 平均利用率	正常运行状态 Pod 实例的 CPU 使用率的平均百分比	百分比	资源使用指标
内存平均利用率	正常运行状态 Pod 实例的内存使用率的平均百分比	百分比	资源使用指标
平均请求响应时间	正常运行状态 Pod 实例的处理请求的平均请求响应时间	毫秒	应用业务指标
请求率	每秒访问服务 Pod 实例的总请求数	个数/秒	应用业务指标
SLA 违规率	所有请求中发生 SLA 违规的百分比	百分比	应用业务指标
当前 Pod 实例数量	正常运行状态的 Pod 实例个数	个数	Pod 实例信息

以上指标有效表示了容器云环境的当前环境状态，与现有相关研究中相似，只考虑当前状态信息的状态空间设计方法没有考虑未来可能发生的变化和不确定性，因此可能导致智能体无法做出最优决策。为了克服这个问题，本文通过加入预测信息来扩展状态空间，以使智能体能够更好地理解环境的动态特征和未来发展趋势，提前感知状态变化，从而做出更优秀的决策。因此，最终设计的状态空间可以表示为$S = \{C, M, R, Q, V, N, Q'\}$，其中$C$表示当前所有 Pod 的平均 CPU 利用率，$M$表示当前所有 Pod 的平均内存利用率，$R$表示平均请求响应时间，$Q$表示请求率，$V$表示 SLA 违规率，$N$表示当前 Pod 实例数量，$Q'$表示由负载预测模型预测的未来请求率。这些指标都被归一化，用作深度神经网络的输入。

4.2.2 动作空间设计

在基于强化学习的弹性伸缩任务中，动作通常定义为 Pod 实例伸缩的数量，而动作空间的大小对性能有较大影响。较大的动作空间能够增加策略表达能力，使得模型能够学习到更复杂的策略，从而获得更精确的伸缩动作以达到更好的优化目标。例如，使用从零到最大可用实例数量的范围来定义状态空间[53]，包含了所有可能的伸缩数量，但是会增加计算复杂度，导致较长的训练时间。相反，较小的动作空间计算复杂度低，收敛速度快。例如，动作空间定义为每次只允许伸缩一至两个实例数量[54]，可以更快速地学习策略，但是学习到的策略将会较为简单，过小的伸缩幅度可能无法满足实际伸缩需求，难以获得较好的优化目标。因此，需要平衡伸缩动作控制粒度和收敛速度，设计一种合理的动作空间表示方式。

本文设计了伸缩调整率a_t，用当前实例个数N_t和a_t计算伸缩动作的目标实例数量为$N_{t+1} = N_t \times (1 + a_t)$。将动作空间定义为一组不同伸缩幅度的调整率，$a_t$的下界和上界分别设置为-0.6 和 2，用间隔为 0.2 进行离散化，完整动作空间表示为$a_t = \{-60\%, -40\%, -20\%, 0\%, 20\%, \ldots, 180\%, 200\%\}$，其中，0 表示不进行伸缩，小于 0 的伸缩调整率表示缩容动作，最小可以调整为当前实例数量的 40%，大于 0 的伸缩调整率表示扩容动作，最大可以调整为当前实例数量的 3 倍。a_t的下界和上界可以根据实际业务伸缩幅度进行调整，从而在保证伸缩动作幅度较大的同时，减少动作数量维度。

4.2.3 奖励函数设计

奖励函数表示在当前资源分配和负载下，伸缩动作执行后根据资源利用率和服务质量的变化计算奖励。奖励函数的设计要能够反映算法的优化目标，为了保证服务的 SLA，提高整体资源利用率，奖励值r_t表示为整体资源利用率$Util$和服务质量$Perf$的乘积，如式（4.1）所示：

$$r_t = Util * Perf \tag{4.1}$$

其中，资源利用率$Util$表示为资源使用数量与资源分配数量的比值：

$$Util = \frac{Resource_{consumed}}{Resource_{deployed}} \tag{4.2}$$

服务质量$Perf$表示为：

$$Perf = \frac{RT_{sla}}{RT_{obs}} - penalty_{RT} \tag{4.3}$$

其中，第一项是服务 SLA 规定的最大请求响应时间延迟RT_{sla}与服务所有实例在当前周期内的平均响应时间延迟RT_{obs}的比值，平均响应时间延迟越短，服务质量奖励越大。第二项的负项表示当平均响应时间延迟高于最大请求响应时间延迟时，违反服务 SLA 的惩罚，可以用式（4.4）表示：

$$penalty_{RT} \begin{cases} w_{RT} * \dfrac{RT_{obs}}{RT_{sla}}, RT_{obs} > RT_{sla} \\ 0, \qquad\qquad otherwise \end{cases} \tag{4.4}$$

其中，w_{RT}表示可调整的惩罚因子。

4.2.4 算法描述

由于本章状态空间设计为连续空间且动作空间设计为离散空间，因此采用深度 Q 网络模型（Deep Q Network，DQN），其主要思想是利用深度神经网络近似值函数，采用经验回放机制和固定目标网络，来提高算法的稳定性和收敛速度。该模型由评估网络和目标网络两部分组成，如图 4.3 所示。

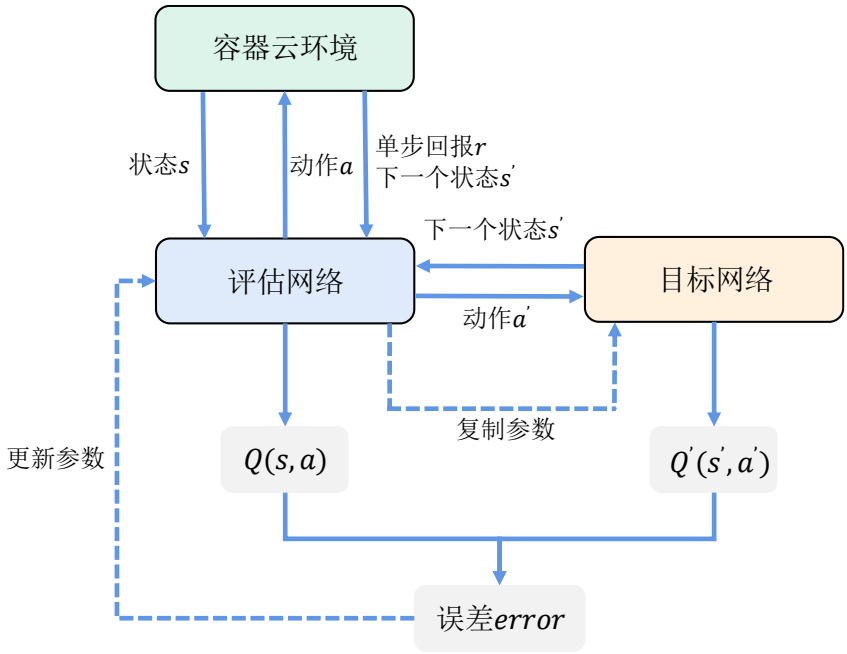


图 4.3 双网络结构的深度 Q 网络模型

在开始训练之前，需要初始化评估网络的参数θ和目标网络的参数θ^-，以及创建经验回放池。经验回放池用于存储在环境中观察到的状态、动作、奖励和下一个状态的四元组(s,a,r,s')，用于后续的训练。在每个时间步，将当前环境状态s输入评估网络，计算出每个可能动作的 Q 估计值，选择具有最大 Q 估计值的动作a作为输出。然后，采用$\varepsilon-greedy$策略，以概率ε随机选择一个动作，以便探索新的状态空间，而以概率$1-\varepsilon$选择当前状态下评估网络输出的最佳动作，以便利用已有的知识。执行选择的动作a，从环境中获取动作带来的奖励 r 和下一个状态s'，并将四元组(s,a,r,s')作为训练经验样本存储到经验回放池中。接着，在经验回放池中随机采样一批经验样本，以减小训练数据的相关性，并用于网络训练。使用式（4.5）作为损失函数，通过反向传播算法，更新评估网络参数θ以最小化损失函数。

$$error=\left(Q(s,a)-\left(r+\gamma*Q'(s',a')\right)\right)^2 \tag{4.5}$$

其中，γ是折扣因子，$Q'(s',a')$是目标网络给定状态s'和a'后输出的目标 Q 值，a'是评估网络根据输入状态s'选出的具有最大 Q 值的动作，表示为式（4.6）：

$$a'=\underset{a_i\in A}{argmax}Q(s',a_i) \tag{4.6}$$

为了解决 DQN 算法中的强相关性问题，每隔一个目标网络参数更新周期，将评估网络的参数复制到目标网络中，以减少训练过程中目标 Q 值的波动，提高算法的稳定性。

在训练过程中，采用经验回放机制来提升学习效果。经验回放技术是一种将之前的经验存储在经验池中，以便后续使用的技术。通过将智能体在环境中的交

互过程存储在经验池中，然后从中随机取出一小批经验，称为 mini-batch，来训练深度神经网络，以更新智能体的行为策略。这种方法与传统的在线学习方法不同，后者只在当前状态下学习，因此可能会忽略重要的状态转换信息。而使用经验回放技术，智能体可以在多个状态下学习，并从之前的经验中学习，从而更好地适应复杂的环境和任务。此外，经验回放技术还可以有效地减少数据的相关性，因为深度神经网络的训练过程依赖于样本之间的独立性假设。如果样本之间存在相关性，那么训练过程可能会导致网络不稳定，甚至不收敛。通过从经验池中随机采样 mini-batch，可以减少数据的相关性，使得训练过程更加稳定。因此，经验回放技术不仅可以提高训练效率和稳定性，还可以有效地减少数据的相关性，使得 DQN 算法更加健壮。

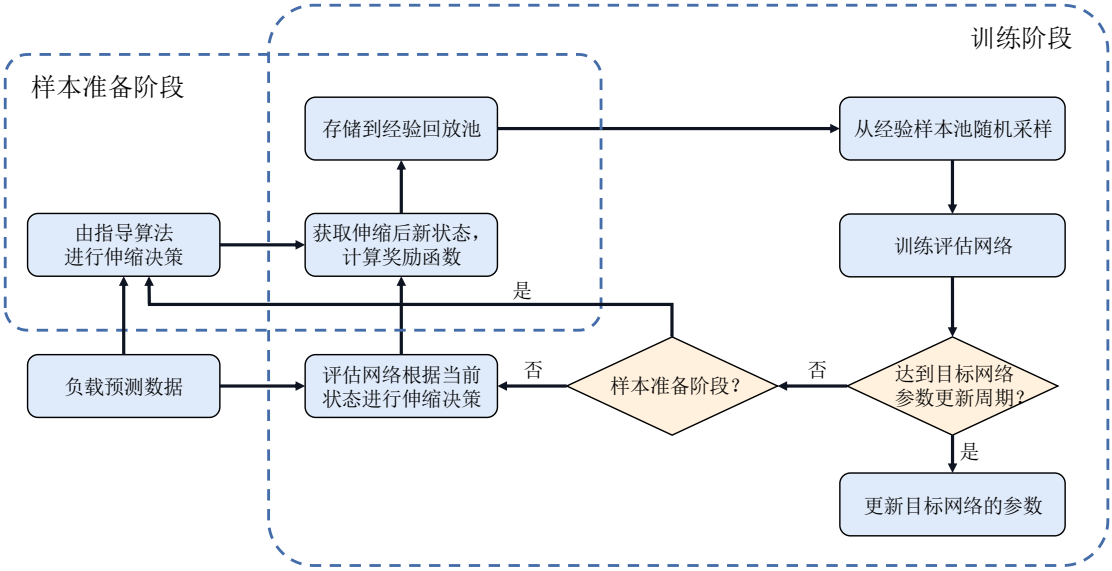


图 4.4 基于 DQN 的强化学习算法流程图

如图 4.4 展示了本章基于 DQN 的强化学习算法流程，算法流程分为样本准备阶段和训练阶段。由于在训练初期，DQN 通过随机化产生的参数尚未开始收敛，Q 值估计较为不准确，因此可能会选择到一些效果较差的动作，导致得到的奖励较低。同时，训练初期经验回放池中的样本数据过少并且样本经验质量较差，会导致神经网络的训练效果较差，进而影响算法的性能表现和训练效率。因此，有必要在早期阶段准备优秀经验的样本，从而引导算法更快地收敛。本文采用由 Dang-Quang 等人提出的基于 Bi-LSTM 的预测式伸缩策略方法[57]作为外部指导算法，从而在算法训练初期实现性能提升。在样本准备阶段，伸缩动作由外部指导算法确定并存储经验样本，同时 DQN 不断地接收由外部指导算法产生的经验样本进行学习，直到达成一个初步可接受的策略。然后，停止使用指导算法，改由 DQN 和环境交互，控制伸缩动作。伸缩动作执行后，获取最新状态并计算奖励函数，新的经验样本进入经验回放池。DQN 使用来自经验回放池的数据进行

训练，经过不断迭代，经验回放池的初始样本数据被来自更优动作的样本数据所取代，不断提升算法的性能。算法 4.1 描述了整个流程。

算法 4.1 基于 DQN 的强化学习算法

算法步骤：

1. 创建经验回放池
2. 初始化评估网络的参数θ
3. 初始化目标网络的参数θ^-
4. **while**
5. 获取当前时刻应用的资源指标和业务指标
6. 用负载预测模型预测未来负载
7. 将资源指标、业务指标和预测负载构建成状态数据s
8. **if** 数据准备阶段
9. 由外部指导算法确定伸缩动作
10. **else**
11. 以随机概率ε选择随机伸缩动作，
否则由 DQN 的评估网络根据当前状态选择伸缩动作
12. 将伸缩动作发送给伸缩控制器，执行伸缩动作a
13. 等待伸缩执行后，获取最新状态s'，根据式 4.1 计算奖励r
14. **if** 经验池满
15. 移除最早添加的经验样本
16. 存储样本(s, a, r, s')到经验回放池
17. 从经验回放池随机采样一批样本数据
18. 根据式 4.5 计算损失，使用反向传播算法更新评估网络的参数
19. **if** 达到目标网络的参数更新周期
20. 把评估网络的参数复制给目标网络

4.3 基于原生响应式伸缩策略的改进优化

Kubernetes 的原生弹性伸缩策略基于应用程序当前的资源利用率与用户设定的目标资源利用率，计算所需的应用 Pod 实例数并执行水平伸缩动作。当应用的当前资源利用率大于目标资源利用率时，就执行扩容动作；当资源利用率低于目标资源利用率时，就执行缩容动作。该策略只考虑了资源的利用率，没有考虑应用的 SLA。而请求响应时间是应用 SLA 的最基本形式，同时也是用户或其他应用最关心的指标。因此，本节结合 SLA，对 Kubernetes 原生响应式伸缩策略改进优化。

在该策略中，有以下关键参数需要设定。SLAThreshold 是用户设置的应用必

须满足的最大响应延迟，是应用提供给用户的 SLA 保证。upperUtilizationThreshold 是系统资源利用率上限，用于触发应用的扩容行为。lowerUtilizationThreshold 是系统资源利用率下限，用于触发应用的缩容行为。targetUtilization 是系统资源利用率的目标值。resourcePerUnitWorkload 是每个应用实例处理单元请求负载所需的资源数量，这是在应用的初始运行期间或在上线之前的测试期间确定的。如果有多个应用程序，可以为每个应用程序设置不同的 SLA 最大响应延迟，使每个应用程序都能满足其 SLA 标准。

为了及时处理突发流量，将响应式弹性伸缩策略的执行周期设置为较短的时间间隔，具体可以概括为以下过程。首先从监控模块获取当前应用的请求响应时间。当响应时间超过用户设置的最大响应延迟时，将根据当前应用的请求负载计算伸缩后的应用 Pod 实例数。策略在考虑服务响应时间的同时，还获取当前资源利用率。当当前资源利用率高于设定的资源利用率上限时，执行应用的扩容操作，根据当前值和设置的目标资源利用率计算伸缩后的 Pod 实例数。当当前资源利用率低于设定的资源利用率下限时，基于相同的逻辑执行缩容操作。整个响应式弹性伸缩策略如算法 4.2 所示。

算法 4.2 响应式伸缩策略算法

输入： SLAThreshold：最大响应延迟
upperUtilizationThreshold：系统资源利用率上限
lowerUtilizationThreshold：系统资源利用率下限
targetUtilization：目标资源利用率
resourcePerUnitWorkload：单个应用实例处理单元请求负载所需的资源数量

输出： N：目标实例数

算法步骤：

1. 获取平均请求响应时间 responseTime
2. **if** responseTime > SLAThreshold
3. 获取请求负载 workload
4. 需要的资源 $=$ $\text{workload} \times \frac{\text{resourcePerUnitWorkload}}{\text{targetUtilization}}$
5. 目标实例数$N_1 = \frac{\text{需要的资源}}{\text{单个实例分配的资源}}$
6. 获取平均资源利用率 realUtilization
7. **if** realUtilization > upperUtilizationThreshold
   or realUtilization < lowerUtilizationThreshold
8. 目标实例数$N_2 = \frac{\text{realUtilization}}{\text{targetUtilization}} \times$ 当前实例数
9. 目标实例数$N = Max(N_1, N_2)$

4.4 主动式和响应式的混合伸缩控制器设计

本文在 4.2 和 4.3 章节分别提出了基于强化学习的主动式伸缩策略和改进的响应式伸缩策略。其中，主动式伸缩策略利用负载预测模块提供的预测负载数据，由强化学习智能体做出针对未来的主动弹性伸缩决策。但在应对一些极端场景下的异常突发流量时，主动式伸缩策略可能由于预测结果不够准确或未及时反映当前的状态，做出不合理的伸缩动作。因此，在单一地依靠主动式伸缩策略时，可能存在一定的风险，从而影响服务的可用性。而响应式伸缩策略虽然依赖于人为设计阈值参数，但确定伸缩动作依据的是实时资源负载指标，能准确反映服务当前状态，可以作为主动式伸缩策略失效时的备选方案，在应对极端流量峰值时进行及时响应和调整。因此，需要结合两种伸缩策略的优点，兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。但这两种策略获得弹性伸缩动作的决策逻辑是不一致的，这可能会产生不同的弹性伸缩动作，导致决策冲突。为了协调两种伸缩策略并同时发挥两者不同的优点，本文提出了混合伸缩控制器的概念。混合伸缩控制器作为最终决策模块，共同控制主动式和响应式伸缩策略的执行。该模块应该具有以下三个特点：

（1）响应式伸缩策略和主动式伸缩策略得出的伸缩动作决策都应考虑。响应式伸缩策略仅基于对当前指标的观察，得出的伸缩动作通常被认为是可信的，而主动式伸缩策略得出的伸缩动作则是经过预先考虑的。

（2）混合伸缩决策应具有周期性。为了防止伸缩动作过于频繁可能影响服务的正常使用，造成系统混乱，混合伸缩控制应该有一个冷却时间。

（3）用户部署好模块的所有算法信息后，模块可以自动加载集群中每个服务的伸缩功能，无需手动配置。

针对上述特点分析，应该周期性地调度两种伸缩策略执行。响应式伸缩根据当前的系统监控数据和一系列预设的组合规则做出伸缩动作决策，而主动式伸缩则预测未来一段时间内应用 Pod 实例的运行状态，并做出主动伸缩动作决策。参考 Kubernetes 原生 HPA 对同一 Pod 副本集的两种伸缩时间要求，混合伸缩控制器每次扩容动作分配实例后的默认冷却时间为 3 分钟，每次缩容动作释放实例后的冷却时间为 5 分钟。混合伸缩控制器每 30 秒做出主动式和响应式策略的伸缩动作决策，伸缩动作的输出信息表示为式（4.7），包括伸缩策略的类型（响应式或主动式）、扩容或缩容的应用 Pod 实例数等。当一种类型的伸缩策略输出了伸缩动作决策而另一种类型的伸缩策略尚未输出时，第一种类型的伸缩策略进入等待，直到另一种伸缩策略输出伸缩动作。

$$E = \{E_{\mathrm{P}}, E_R\} = \{(M_P, N_{\mathrm{P}}), (M_R, N_R)\} \tag{4.7}$$

其中，P表示主动式伸缩策略（Proactive）的伸缩动作，R表示响应式伸缩（Reactive）策略的伸缩动作。E、M和N分别表示伸缩方法、扩容或缩容的动作类型、扩容或缩容的 Pod 实例数。M为正表示伸缩动作为扩容分配实例，M为负表示执行动作为缩容释放实例。N_P和N_R分别表示主动式策略和响应式策略得出的扩容或缩容 Pod 实例数。

为了实现无需人工配置即可做出伸缩决策的要求，本文为伸缩控制器制定了混合伸缩控制规则。伸缩控制器根据混合伸缩控制规则做出最终伸缩动作，规则制定如表 4.2 所示。

表 4.2 混合伸缩控制规则

主动式伸缩方向 P / 响应式伸缩方向 R	扩容	缩容	不伸缩
扩容	扩容 $Max(N_P, N_R)$个	$\begin{cases}\text{扩容}, N_R - N_{\mathrm{P}} > 0\\ \text{缩容}, N_R - N_{\mathrm{P}} < 0\end{cases}$ 个数：$\lvert N_R - N_{\mathrm{P}}\rvert$	扩容N_{R}个
缩容	$\begin{cases}\text{扩容}, N_{\mathrm{P}} - N_R > 0\\ \text{缩容}, N_{\mathrm{P}} - N_R < 0\end{cases}$ 个数：$\lvert N_{\mathrm{P}} - N_R\rvert$	缩容 $Min(N_P, N_R)$个	不伸缩
不伸缩	扩容N_{P}个	不伸缩	不伸缩

当响应式伸缩与主动式伸缩的伸缩方向一致时，或者有任意一个伸缩策略不进行伸缩时，采用“快扩容慢缩容”的谨慎伸缩策略。具体地，当两种伸缩策略的伸缩方向都为扩容时，最终扩容动作的分配 Pod 数采用两种扩容动作的较大值$Max(N_P, N_R)$，以更多地分配 Pod 数，充分保证服务 SLA。当两种伸缩策略的伸缩方向都为缩容时，最终缩容动作的释放 Pod 数采用两种缩容动作的较小值$Min(N_P, N_R)$，以最小化 Pod 调整数量，避免伸缩抖动。当一种伸缩策略执行扩容动作并且另一种策略不进行伸缩时，最终伸缩动作即为扩容相应的 Pod 数。当一种伸缩策略执行缩容动作并且另一种策略不进行伸缩时，则最终不进行伸缩。当两种伸缩策略都不进行伸缩时，则最终同样不进行伸缩。

当响应式伸缩和主动式伸缩的伸缩方向相反时，采用平衡思想解决冲突，即由扩容动作的分配 Pod 数减去缩容动作的释放 Pod 数，得到的结果为正则执行扩容动作，为负则执行缩容动作，用两种伸缩策略的伸缩 Pod 实例数之差作为最终扩容或缩容实例数。例如，主动式伸缩策略采取缩容动作，响应式伸缩策略采取扩容动作，且扩容动作分配实例数比缩容动作释放实例数更多，则最终伸缩动作为扩容$N_R - N_{\mathrm{P}}$个 Pod 数，在充分保证服务 SLA 的条件下，提高资源利用率。

4.5 本章小结

在弹性伸缩技术中，准确确定合理的伸缩动作是至关重要的。为了克服现有研究中基于负载预测和基于强化学习的弹性伸缩策略各自存在的缺点，本章提出了基于强化学习的主动式伸缩策略。该策略有针对地设计了动作空间、状态空间、奖励函数和算法流程，使强化学习有效应用于弹性伸缩任务。具体地，状态空间设计考虑了资源使用指标、应用业务指标，以有效表示云应用的资源利用和服务质量状态，并且通过加入负载预测信息来扩展状态空间，使智能体能够更好地理解环境的动态特征和未来发展趋势，从而做出更优秀的决策；动作空间设计采用了伸缩调整率来表示伸缩动作，在保证伸缩动作幅度的同时，减少动作数量维度；奖励函数设计考虑了服务质量和资源利用率，以反映算法的优化目标；强化学习算法采用双网络结构的深度 Q 网络模型和经验回放机制，来提高算法的稳定性和收敛速度，并通过在样本准备阶段由外部指导算法确定伸缩动作，来解决训练初期优质经验样本不足导致的训练效率低的问题。该策略充分利用了基于负载预测和基于强化学习的方法的优点，根据应用的当前资源负载状态和预测信息，由强化学习智能体做出合理的弹性伸缩决策，能够更加准确和智能地调整 Pod 服务实例个数。

针对在极端流量峰值场景下主动式伸缩策略可能失去有效性的问题，本章进一步地提出了一种主动式和响应式的混合弹性伸缩策略。其中，响应式伸缩策略结合请求响应时间指标，对仅考虑资源利用率的 Kubernetes 原生 HPA 策略改进优化，并将其作为主动式伸缩策略失效时的备选方案；混合伸缩控制器协调主动式和响应式伸缩策略，采用“快扩容慢缩容”和平衡冲突思想，做出最终伸缩决策，兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

第 5 章 Kubernetes 水平弹性伸缩系统设计与实现

为了验证第 4 章所提出的弹性伸缩策略的有效性，本章在 Kubernetes 平台上实现了相应的弹性伸缩系统，并设计了不同负载场景下的弹性伸缩对比实验。

5.1 系统架构设计

水平弹性伸缩系统架构如图 5.1 所示，主要包括监控模块、预测模块、强化学习模块和伸缩控制模块，各模块间交互流程如下：

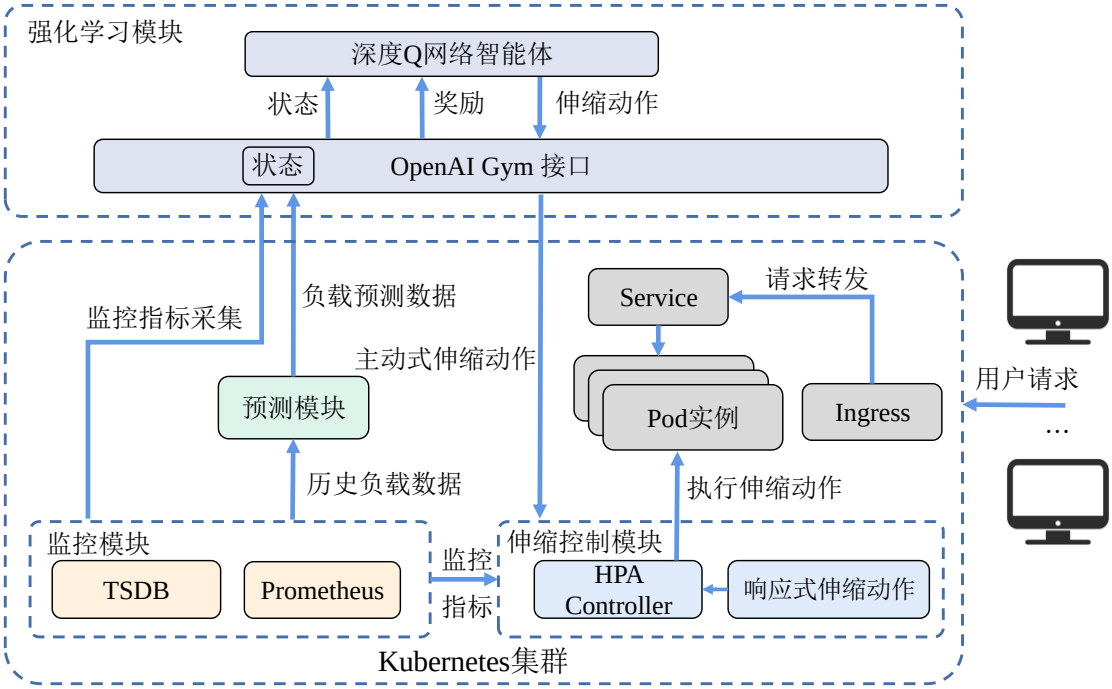


图 5.1 水平弹性伸缩系统架构图

（1）监控模块采集应用程序 Pod 实例的资源使用和服务质量信息，并存储到时序数据库中。

（2）预测模块根据监控模块采集的历史负载数据，使用训练好的负载预测模型预测未来负载。

（3）强化学习模块将监控模块提供的资源使用和服务质量信息，以及预测模块提供的负载预测数据，作为环境状态信息，由智能体根据优化策略做出主动式伸缩动作决策，输出给伸缩控制模块。

（4）伸缩控制模块中，利用监控模块提供的资源使用和服务质量信息根据响应式伸缩策略做出响应式伸缩动作，再由混合伸缩控制器从响应式伸缩动作和强化学习模块输出的主动式伸缩动作之中，决策出最终伸缩动作并执行。

5.2 模块设计与实现

5.2.1 监控模块

资源使用指标和应用业务指标的监控采集是负载预测和弹性伸缩的前提。Kubernetes 的默认监控方案由两个主要组件构成，分别是内置于节点的 Kubelet 组件中的 cAdvisor 监控工具和 Metrics Server 指标服务器。cAdvisor 用于收集和监控各个容器的系统资源指标数据，而 Metrics Server 则负责将 cAdvisor 收集到的数据进行汇总，并通过 Kubernete 内置的 API Server 实现了一种名为 Metrics API 的接口，向弹性伸缩控制器提供所汇总的数据。然而，该方案仍然存在一些问题。首先，cAdvisor 只能监控资源使用指标，如容器的 CPU 和内存利用率等，而不能监控应用的业务指标，如请求率和请求响应时间。其次，该监控方案没有提供将监控数据在系统中长期保存的机制，预测模块无法获取足够的历史负载数据进行预测。为解决以上两个问题，本文结合 Prometheus 设计了监控模块，其架构如图 5.2 所示。

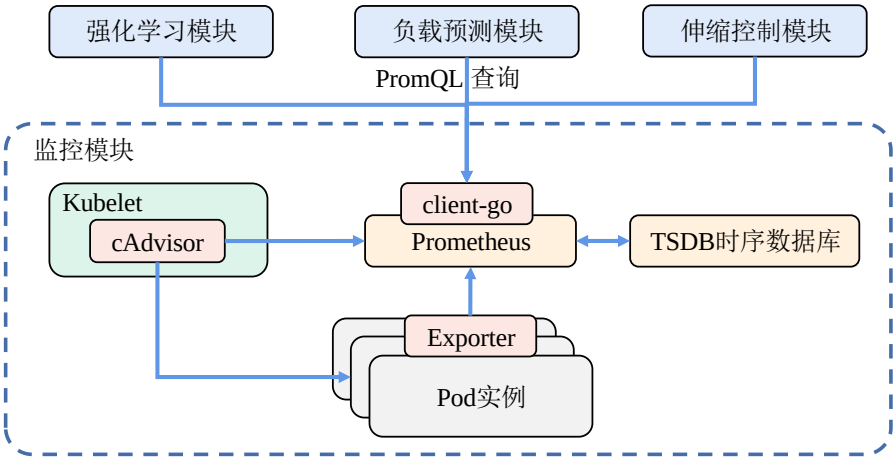


图 5.2 监控模块

Prometheus 是一款开源的系统监控工具，具有主动拉取多个监控目标（包括 cAdvisor）的周期性监控指标数据的能力[74]，并提供了专用于存储时序数据的数据库和灵活的查询语言。本文主要考虑对 Pod 的资源使用指标和应用业务指标进行监控，分别由 cAdvisor 和 Exporter 采集。Exporter 是向 Prometheus 提供监控数据的程序，应用可以使用 Prometheus 社区提供的 Exporter 实现向 Prometheus 暴露应用业务指标数据。Prometheus 通过轮询的方式定期从监控目标中采集监控数据，采集到的指标数据存储到 TSDB 时序数据库进行持久化。强化学习模块、负载预测模块和伸缩控制模块通过向 Prometheus 的 client-go 客户端发送 PromQL 查询语句，直接从 TSDB 读取指标。

5.2.2 预测模块

预测模块包括负载预测模型训练程序和模型调用程序两部分。负载预测模型训练程序由 Python 语言编写，实现本文第 3 章提出的负载预测模型，并在历史负载序列数据上离线训练，保存训练完成的模型。模型调用程序由 Go 语言编写，通过向监控模块的 client-go 客户端发送 PromQL 查询语句，获取采集的历史负载数据，使用 Go 的 os/exec 包来执行 Python 脚本，实现对训练完成的预测模型的调用，得到未来负载预测值。

5.2.3 强化学习模块

OpenAI Gym[75]是一个开源的 Python 工具包，用于测试和开发强化学习算法。本文基于 OpenAI Gym 实现了强化学习模块的运行环境构建。主要包含 KubernetesEnv 类、DeepQNetworkAgent 类和 Experiment 类，类图如图 5.3 所示。

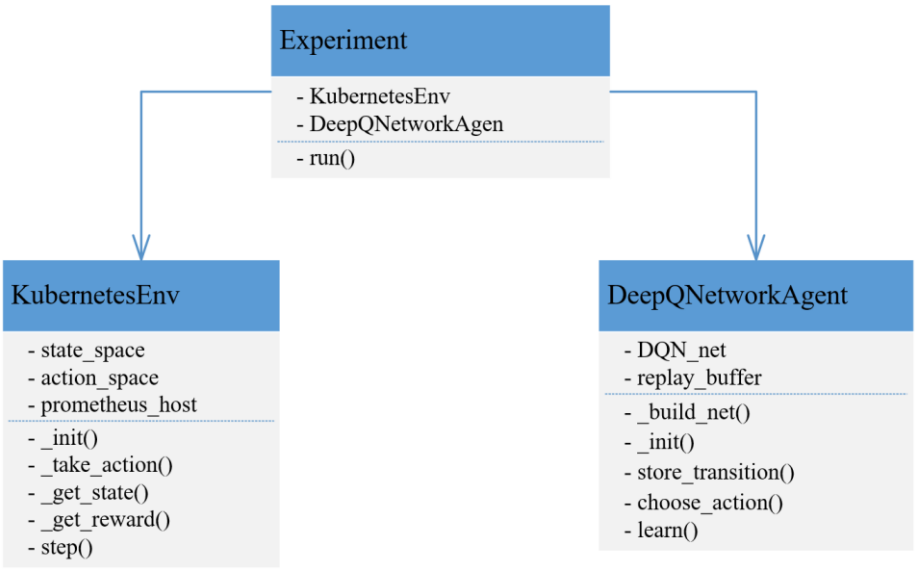


图 5.3 强化学习相关类图

KubernetesEnv 类继承自 gym.Env 类，用于抽象 Kubernetes 环境，主要定义 init()、_take_action()、_get_state()、_get_reward()和 step()等方法。init()方法包括状态空间、动作空间、Prometheus API 查询地址等参数的定义。_take_action()方法将主动伸缩动作目标 Pod 数通过 Kubernetes client 修改到 HPA 对象的相应属性中。_get_state()方法通过向 Prometheus 的 API 查询请求获取当前 CPU 利用率、请求率、请求响应时间信息，通过 Kubernetes client 获取当前 Pod 实例数和负载预测值，返回状态列表。_get_reward()方法根据传入的状态列表，根据奖励函数计算奖励值。step()方法表示一次决策动作，返回新的状态值和奖励值。

DeepQNetworkAgent 中实现了深度 Q 神经网络和强化学习算法，主要定义了_build_net()、init()、store_transition()、choose_action()和 learn()等方法。_build_net()方法用于构建深度 Q 网络，包含网络结构相同的评估网络和目标网络。init()方法用于初始化神经网络的学习率、状态空间维度、动作空间维度、折扣因子γ、$\varepsilon-greedy$概率、经验池大小、批量选取经验样本个数等参数。store_transition()方法用于存储经验样本，当经验池满时旧经验样本会被新经验样本替换。choose_action()方法用于选择伸缩动作，以随机概率选择伸缩动作，否则由 DQN 的评估网络根据当前状态生成所有动作的Q值，选择Q值最大的伸缩动作。learn()方法用于学习更新深度 Q 网络的参数。首先从经验池中随机抽取一批样本，然后分别经过评估网络和目标网络得到评估 Q 值和目标 Q 值，最后计算损失并反向传播更新评估网络。并每隔一个目标网络参数更新周期，将评估网络的参数复制到目标网络中。

Experiment 类用于启动实验，根据算法 4.1，使深度 Q 网络智能体不断与 Kubernetes 环境进行交互学习和试错，最终找到最优的伸缩策略。

5.2.4 伸缩控制模块

为了实现本文提出的伸缩策略，需要对 Kubernetes 进行扩展。通常，使用 Kubernetes 中的自定义资源定义（Custom Resource Definitions，CRD）和自定义控制器来扩展 Kubernetes 平台。CRD 提供了一种扩展 Kubernetes API 的方法，它允许用户创建自定义资源类型，并使用 Kubernetes 已有的工具进行管理和控制。自定义控制器则是一种能够监视自定义资源状态变化的自动化机制，可用于处理和响应资源的状态变化。

首先，创建自定义资源类型，主要包括定义 spec 字段和 status 字段。spec 字段用于描述资源的期望状态，可以由用户指定修改。status 字段用于描述资源的实际状态，通常是一个只读的字段，由控制器自动计算和更新，用户不能手动修改它的值。本文自定义的 HPA 资源类型的主要相关参数和说明如表 5.1 所示。

表 5.1 自定义的 HPA 资源类型的主要相关参数和说明

字段类型	参数名称	说明
spec	scaleTargetRef	指定目标 Deployment
	minReplicas	最大扩容副本数
	maxReplicas	最小缩容副本数
	metrics	配置弹性资源指标，如 CPU、Memory 等
	scaleStrategy	伸缩策略类型，可选 proactive 主动式、reactive 响应式、hybrid 混合式

续表

字段类型	参数名称	说明
spec	upperUtilizationThreshold	期望平均资源利用率上限
	lowerUtilizationThreshold	期望平均资源利用率下限
	targeUtilization	期望资源利用率
	SLAThreshold	SLA 保证，服务必须满足的最大响应延迟
	subresources.status. proactiveDesiredReplicas	子资源 status 属性，由强化学习模块修改主动式伸缩策略期望副本数
status	currentReplicas	当前副本数
	desiredReplicas	期望副本数

创建自定义的 HPA 资源类型后，需要编写相应的自定义控制器来处理自定义资源类型的状态变化。Kubernetes 资源编排的核心思想是控制循环，通过对比资源的实际状态和期望状态的差异，自动调整资源的状态以实现期望状态，其逻辑见算法 5.1。

算法 5.1 控制循环伪代码

```
1.  for
2.      实际状态 = 获取集群中资源对象 X 的实际状态
3.      期望状态 = 获取集群中资源对象 X 的期望状态
4.      if 实际状态 != 期望状态
5.          将实际状态调整为实际状态
6.  end
```

本文的自定义 HPA 控制器的主要逻辑如下：

（1）从当前自定义 HPA 资源对象中，获取 spec 中的信息。

（2）从监控模块的 Prometheus client 获取所管理 Pod 的资源使用指标和应用业务指标。

（3）如果伸缩策略模式为 reactive 响应式，则根据当前指标状态、平均资源利用率上限、平均资源利用率下限、目标资源利用率和 SLA 阈值，计算响应式伸缩策略期望副本数，跳转步骤 6。

（4）如果伸缩策略模式为 proactive 主动式，则从 subresources.status.proactive DesiredReplicas 属性中获取由强化学习模块修改的主动式伸缩策略期望副本数，跳转步骤 6。

（5）如果伸缩策略模式为 hybrid 混合式，则将步骤 3 和 4 的响应式、主动式期望副本数根据混合伸缩规则确定最终期望副本数。

（6）将期望副本数和最大最小副本数进行比较调整，以确保期望副本数在

指定的范围内，并对当前时间是否处于伸缩冷却期进行检查。

（7）调用相关接口将 Deployment 中的副本数修改为期望副本数。然后，Deployment 会将目标 Pod 的当前副本数调整至期望副本数，实现弹性伸缩。

5.3 弹性伸缩对比实验

5.3.1 实验设计

本实验设置在一台具有 Intel 酷睿 i7-9700（8 核 3.6 GHz）和 32GB 内存的物理机上。Kubernetes 集群由一个主节点和两个工作节点组成，每个节点都是一个 4 核 8GB 内存的 CentOS7 虚拟机，Kubernetes 的软件版本基于 1.22。

为了验证本文提出的弹性伸缩策略在不同负载场景下的伸缩性能，实验从 World Cup 负载数据集的测试集中选取了两种具有不同特点的请求负载，一种是平稳波动的负载，一种是异常突发的负载。为了便于实验，将原始请求负载数据进行了等比例缩放，具体请求负载如图 5.4 和图 5.5 所示。

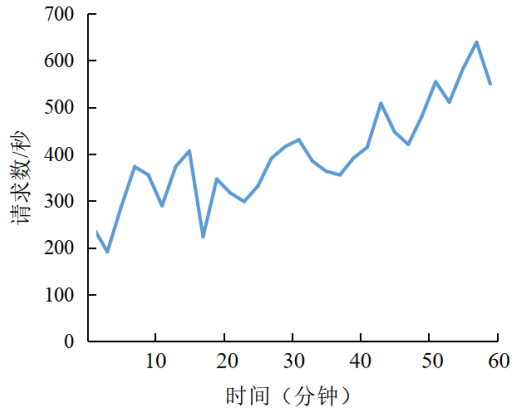


图 5.4 平稳波动负载

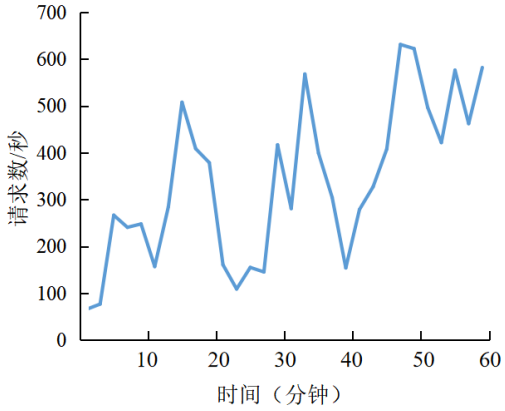


图 5.5 异常突发负载

实验具体步骤如下：

（1）构建容器化应用。实验所采用的测试应用程序基于 SpringBoot 开发，能够模拟真实的 Web 应用场景，且使用了 Prometheus Exporter 来暴露应用程序的请求响应时间和请求率等业务指标，并将应用程序构建为容器镜像。

（2）创建 Deployment 和 Service 资源对象。Deployment 中定义每个应用 Pod 实例的资源分配为 0.5 核 CPU 和 512MB 内存。Service 负责将服务暴露给用户访问，并将请求负载均衡地分发到每个 Pod 上。

（3）创建各对比方法的 HPA 资源对象。其中，原生 HPA 伸缩策略的伸缩指标设置为 CPU 利用率，阈值设为 60%。

（4）使用 Locust 工具生成请求负载序列并对 Web 测试应用进行访问，观察请求响应时间和 CPU 利用率的变化情况，对比不同的伸缩策略的效果。

5.3.2 对比方法

为了验证本文第 4 章提出的弹性伸缩策略的有效性，实验选择了三种容器弹性伸缩策略方法分别与本文的主动式伸缩策略、混合式伸缩策略对比：

（1）**原生 HPA**：Kubernetes 原生水平弹性伸缩策略，属于基于阈值的响应式伸缩策略。

（2）**Bi-LSTM-HPA**[57]：2021 年由 Dang-Quang 等人提出的基于负载预测的伸缩策略，该策略通过 Bi-LSTM 对请求负载进行预测，再根据负载预测值由阈值规则计算目标 Pod 数量。

（3）**DScaler**[54]：2022 年由 Xiao 等人提出的基于深度强化学习的伸缩策略，该策略以 CPU 利用率、请求率和当前分配的 Pod 数量作为状态空间，以伸缩一至两个的 Pod 实例数作为动作空间。

5.3.3 评价指标

实验评价指标包括 SLA 违规率和平均 CPU 利用率，具体如下：

（1）**SLA 违规率**：SLA 规定了云服务提供商向客户保证的服务质量。在本文中，SLA 被定义为应用程序响应时间的上限阈值RT_{sla}，实验中设置为 400ms。SLA 违规率被定义为违规时间与总时间的比率，其计算公式表示为式（5.1）：

$$SLA_{violate} = \frac{T_{violate}}{T} \tag{5.1}$$

其中，$T_{violate}$表示应用程序的响应时间超过RT_{sla}的时间段。

（2）**平均 CPU 利用率**：平均 CPU 利用率是所有实例的 CPU 利用率的平均值，其计算公式表示为式（5.2）：

$$Avg_{resourse} = \frac{1}{T}\frac{1}{N}\sum_{t=0}^{T}\sum_{i=1}^{N} Util_{ti} \tag{5.2}$$

其中，N表示实例数，$Util_{ti}$表示第i个实例在时间t的 CPU 利用率。

5.3.4 实验结果与分析

本文使用平稳波动和异常突发两种类型的请求负载分别进行弹性伸缩实验，比较各对比方法下的 SLA 违规率和平均 CPU 利用率指标，实验结果如图 5.6 和图 5.7 所示。

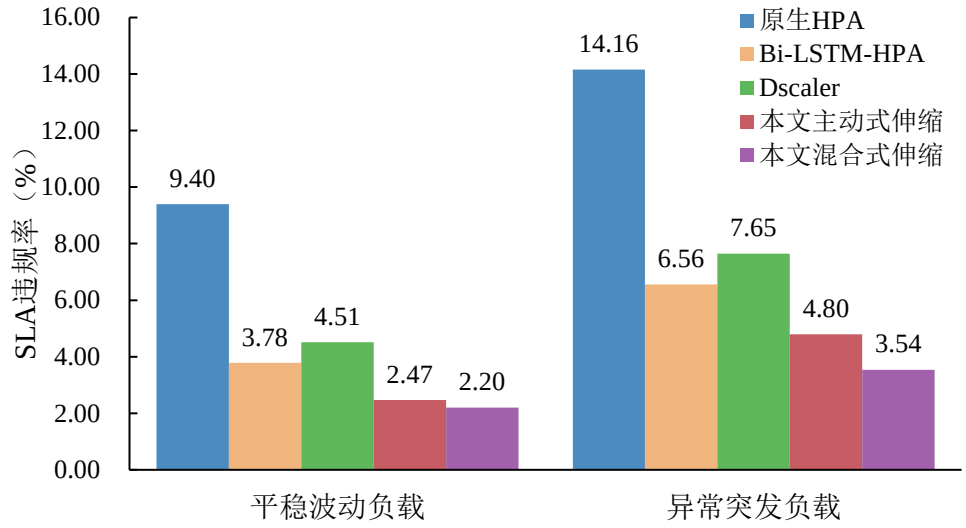


图 5.6 SLA 违规率结果

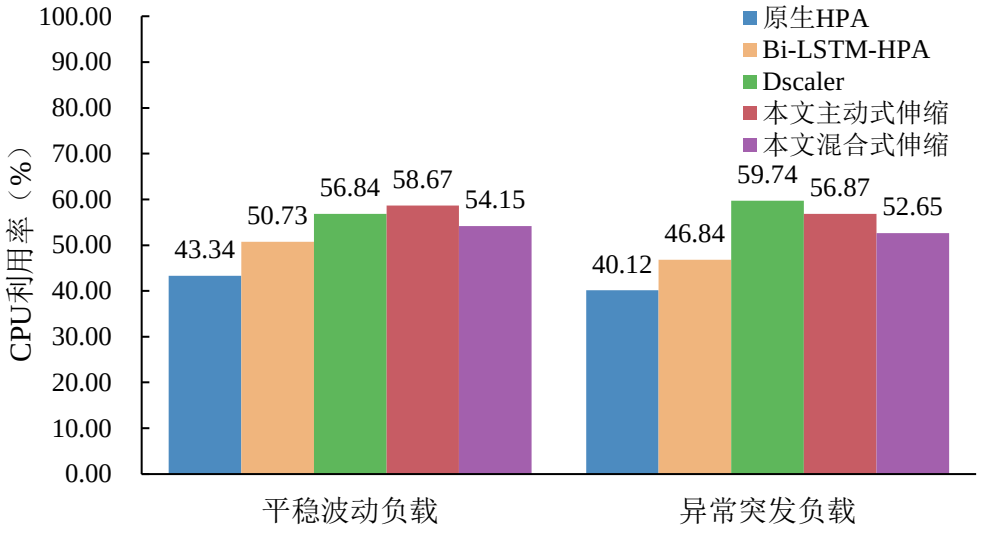


图 5.7 CPU 利用率结果

从图中可以看出，在平稳波动负载下，本文提出的混合式伸缩策略达到了 2.2%的 SLA 违规率和 54.15%的 CPU 利用率；在异常突发负载下，本文提出的混合式伸缩策略达到了 3.54%的 SLA 违规率和 52.65%的 CPU 利用率。本文提出的混合式伸缩策略与对比方法相比，具有较低的 SLA 违规率和较高的 CPU 利用率，整体上优于其他对比方法。具体分析如下：

（1）本文主动式伸缩策略对比原生 HPA：在平稳波动负载下，本文主动式伸缩策略与原生 HPA 相比，SLA 违规率降低了 6.93%，CPU 利用率提高了 15.33%；在异常突发负载下，SLA 违规率降低了 9.36%，CPU 利用率提高了 16.75%。原生 HPA 是一种基于阈值的响应式伸缩策略，由于其仅根据当前资源指标计算副本数，在扩容阶段存在滞后性，影响了 SLA，并且在面对负载下降后再次上升的波动情况时，出现响应时间抖动，因此 SLA 违规率在所有对比方法中最高。同时，原生 HPA 即使滞后分配了 Pod，也无法消除已经产生的 SLA 违规，造成了

资源浪费，因此 CPU 利用率较低。本文主动式伸缩策略通过负载预测，提前预知了负载上升和短暂下降情况，能够在面对负载上升时提前扩容，面对负载短暂下降时维持 Pod 数量，因此评价指标都明显优于原生 HPA，这表明主动式伸缩策略比原生的响应式伸缩策略更有效。在面对异常突发负载时，两种方法都难以快速响应有效面对，相比于平稳波动负载的评价指标都更差。

（2）本文主动式伸缩策略对比 Bi-LSTM-HPA：这两种方法都基于负载预测，能对未来负载提前做出反应，与原生 HPA 策略相比，SLA 违规率、资源利用率都有了较大优化。Bi-LSTM-HPA 根据负载预测值由规则计算目标 Pod 副本数，且未考虑资源利用率，而本文方法利用负载预测值和当前状态由智能体确定目标 Pod 副本数，并以响应时间和资源利用率作为优化目标，能够做出更合理的伸缩动作。因此，在面对平稳波动负载时，本文主动式伸缩策略与 Bi-LSTM-HPA 相比，SLA 违规率降低了 1.31%，CPU 利用率提高了 7.94%。在面对异常突发负载时，两种方法都难以准确预测突发负载，导致不能快速响应有效应对，评价指标相较于平稳波动负载都更差。其中，Bi-LSTM-HPA 由于目标副本数直接由请求负载预测值计算得到，影响程度更大，相较于平稳波动负载的 SLA 违规率提高了 2.78%，CPU 利用率降低了 3.89%。而本文主动式伸缩方法除了利用负载预测值之外，还充分利用了当前负载值、响应时间、资源利用率等信息由智能体确定目标副本数，缓解了预测值不准确带来的影响，相较于平稳波动负载的 SLA 违规率提高了 2.33%，CPU 利用率降低了 1.8%，影响幅度较小。

（3）本文主动式伸缩策略与 Dscaler 对比：这两种方法都基于强化学习，通过不断与环境进行交互学习和试错，最终找到最优的伸缩策略。其中，Dscaler 根据当前状态做出推荐伸缩动作，在一定程度上仍属于响应式伸缩。而本文方法在状态空间中使用了负载预测值，使智能体能更好地感知环境，做出更合理的伸缩动作。因此，在面对平稳波动负载时，本文主动式伸缩策略与 Dscaler 相比，SLA 违规率降低了 2.04%，CPU 利用率提高了 1.83%。在面对异常突发负载时，Dscaler 动作空间的伸缩幅度较小，导致分配的 Pod 实例少，CPU 利用率更高，但也产生了 7.65%的 SLA 违规率。而本文主动式伸缩策略使用伸缩调整率来定义动作空间，具有更大的伸缩动作控制粒度，更能满足实际伸缩需求，因此本文主动式伸缩策略相比于 Dscaler，SLA 违规率降低了 2.85%。

（4）本文混合式伸缩策略与其他方法对比：在平稳波动负载下，本文主动式伸缩策略与原生 HPA 相比，SLA 违规率降低了 7.2%，CPU 利用率提高了 10.81%；在异常突发负载下，SLA 违规率降低了 10.62%，CPU 利用率提高了 12.53%。在面对平稳波动负载时，混合伸缩策略与本文主动式伸缩策略相比，评价指标优化不明显。在面对异常突发负载时，本文的响应式伸缩策略根据当前峰值负载确定

所需 Pod 实例个数，混合伸缩策略会在主动式伸缩策略和响应式伸缩策略中选择扩容更大的伸缩动作，弥补了负载预测不准确以及基于强化学习的伸缩方法动作空间伸缩幅度较小的缺陷，因此与 Bi-LSTM-HPA、Dscaler 和本文主动式伸缩策略相比，SLA 违规率分别降低了 3.02%、4.11%和 1.26%。同时，由于混合伸缩规则采用“快扩容慢缩容”的思想，因此会分配较多 Pod 实例，CPU 利用率有所降低。总体来说，在平稳波动和异常突发两种负载下，本文混合式伸缩策略与对比方法相比，具有较低的 SLA 违规率和较高的 CPU 利用率。

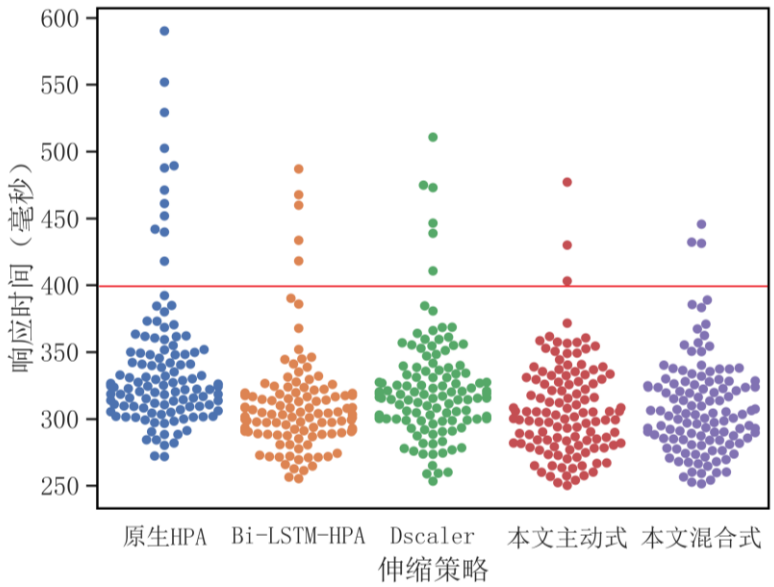


图 5.8 平稳波动负载下响应时间对比

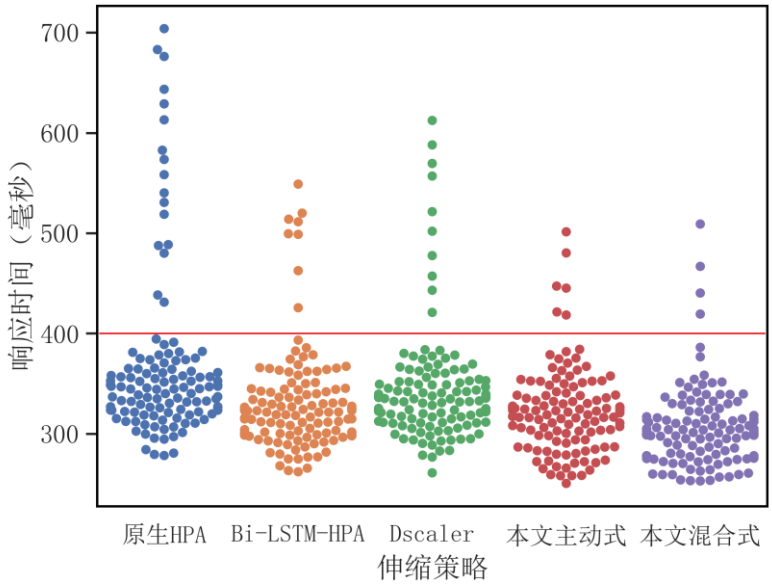


图 5.9 异常突发负载下响应时间对比

实验还对比了在平稳波动和异常突发负载下的不同伸缩策略的请求响应时间，如图 5.8 和图 5.9 所示，图中红色的水平线表示 SLA 规定的响应时间阈值上限，每个点表示单位时间段内的平均请求响应时间。从图中可以看出，在原生 HPA 下，有较多时间段超过了响应时间阈值上限，产生了较多的 SLA 违规；在 Bi-

LSTM-HPA 和 Dscaler 下，SLA 违规时间段相比于原生 HPA 有一定程度的减少；在本文提出的伸缩策略下，SLA 违规时间段相比于其他对比方法都更少，并且响应时间的分布更下沉，具有更低的总体平均响应时间。这表明本文提出的伸缩策略在减少 SLA 违规率的同时，还提升了应用程序处理请求的性能。

优化 Kubernetes 的弹性伸缩策略是一项艰巨任务，它受到众多因素的限制与影响。本文采用的实验硬件条件有限，应用程序规模较小，整体负载水平不高。从实验结果可以得出，相较于对比方法，本文提出的伸缩策略实现了一定程度的优化，但优势不够明显。因此，未来需要在更为充足的硬件设施基础上，扩大应用程序规模，并在更多的应用场景中进行实验，以更全面、更准确地验证策略的有效性和优势性。

5.4 本章小结

本章对第 4 章提出的伸缩策略在 Kubernetes 环境中进行实现，详细描述了弹性伸缩系统的架构设计和各模块的设计与实现，并设计弹性伸缩对比实验以验证伸缩策略的有效性。实验使用平稳波动和异常突发两种类型的请求负载，选择原生 HPA、基于负载预测的 Bi-LSTM-HPA 和基于强化学习的 Dscaler 作为对比方法。实验结果表明，本文提出的主动式伸缩策略相比于对比方法，具有更低的 SLA 违规率和更高的资源利用率；本文提出的混合式伸缩策略在应对异常突发请求负载时，弥补了负载预测不准确以及基于强化学习的伸缩方法动作空间伸缩幅度较小的缺陷，有效减少 SLA 违规。

第 6 章 总结与展望

6.1 本文工作总结

弹性伸缩作为容器云平台的重要特性，能够根据实际的负载情况动态地调整应用的资源配额，保障应用的高可用性和稳定性。其中，主动式弹性伸缩策略通过主动预测未来的资源需求，对可能出现的情况做出预先布置，得到众多学者的广泛研究。本文以在满足服务 SLA 的同时最大化资源利用率为目标，从精准预测未来负载和准确做出合理弹性动作两方面对弹性伸缩任务开展研究。本文主要工作总结如下：

（1）基于多尺度空洞卷积和 LSTM 的负载预测模型研究

云负载数据时序变化模式多样，表现出波动性、持续性和多周期性的特点，给精准预测未来负载带来了巨大挑战。为了有效捕获这些负载变化特征，本文提出了基于多尺度空洞卷积和 LSTM 的负载预测模型。模型采用平行和堆叠的空洞卷积残差网络构成多尺度特征提取块，以学习连续和非连续输入时间步的时序特征，再通过 LSTM 进一步对长期时间依赖进行建模。多尺度特征提取块通过使用卷积核大小和膨胀率的创新组合，使模型可以同时利用负载序列中的自相关和偏自相关信息。模型设计了两个通道来处理不同的输入，基本特征学习通道输入原始历史负载序列，学习短期波动和持续性等基本负载特征；周期特征学习通道输入经过周期时间编码的历史负载序列，学习多周期性负载特征。在四个云负载数据集进行了不同预测时间步长的负载预测实验。实验结果表明，本文提出的模型在 MSE、RMSE、MAE 三个指标上均优于其他对比模型，具有更高的预测精度。本文模型的总体 MSE 指标在 NASA 数据集上相对于最佳对比模型实现了 8.81%的改进，在 WorldCup 数据集上实现了 9.09%的改进，在 Google Cluster 数据集上实现了 8.08%的改进，在 Alibaba Cluster 数据集上实现了 7.52%的改进。

（2）基于负载预测和强化学习的水平混合弹性伸缩策略研究

在弹性伸缩技术中，准确确定合理的伸缩动作需要考虑多种因素，其中包括负载波动情况、系统资源使用情况、成本和性能之间的平衡以及可靠性等因素。本文对现有研究中基于负载预测和基于强化学习的伸缩策略的优缺点进行了分析，并提出了一种新的主动式伸缩策略，以充分融合两种方法的优点。该策略有针对地设计了动作空间、状态空间、奖励函数和算法流程，使强化学习有效应用于弹性伸缩任务。策略根据应用的当前资源负载状态和预测信息，由强化学习智能体做出合理的弹性伸缩决策，能够更加准确和智能地调整 Pod 服务实例个数。

此外，针对在极端流量峰值场景下主动式伸缩策略易失去有效性的问题，本文进一步地提出了一种主动式和响应式的混合弹性伸缩策略。该策略的响应式伸缩策略结合请求响应时间指标，对仅考虑资源利用率的 Kubernetes 原生 HPA 策略改进优化，将其作为主动式伸缩策略失效时的备选方案。并通过混合伸缩控制器协调主动式和响应式伸缩策略，采用“快扩容慢缩容”和平衡冲突思想，做出最终伸缩决策，兼顾主动伸缩决策和在极端峰值流量下的有效伸缩。

（3）Kubernetes 水平弹性伸缩系统设计与实现

对本文提出的伸缩策略在 Kubernetes 环境中进行系统实现，系统包括监控模块、预测模块、强化学习模块和伸缩控制模块。监控模块采用基于 Prometheus 的方案实现，用于收集资源使用指标和业务指标监控数据；预测模块通过负载预测模型预测未来负载；强化学习模块基于 OpenAI Gym 工具实现强化学习环境搭建与训练；伸缩控制模块基于自定义控制器实现伸缩策略控制逻辑。为了验证本文提出的弹性伸缩策略的有效性，选取了平稳波动和异常突发两种类型的请求负载进行测试，并选择原生 HPA、基于负载预测的 Bi-LSTM-HPA 和基于强化学习的 Dscaler 作为对比方法。实验结果表明，本文提出的主动式伸缩策略相比于对比方法，具有更低的 SLA 违规率和更高的资源利用率；本文提出的混合式伸缩策略在应对异常突发请求负载时，弥补了负载预测不准确以及基于强化学习的策略动作空间伸缩幅度较小的缺陷，有效减少 SLA 违规。

6.2 未来工作展望

本文从负载预测和弹性伸缩两方面开展了相关研究，并取得了一些研究成果，但是仍然有所欠缺，未来将在以下几个方面展开工作：

（1）结合异常检测技术完善对负载状态的观测。本文实现的云负载预测模型面对极端异常突发负载时，难以准确预测负载状态。在未来工作中，考虑结合异常检测技术，以识别并分析云负载中的异常事件，完善云负载预测模型，进一步减少 SLA 违规，保障应用程序正常运行。

（2）结合垂直伸缩技术优化弹性伸缩策略。本文仅从改变 Pod 实例数量的角度设计了水平弹性伸缩策略，在未来工作中，考虑结合垂直伸缩技术，以动态调整单个 Pod 实例的计算资源，进一步提高资源利用率。

（3）将弹性伸缩策略应用于 Serverless 云场景中。本文主要是面向微服务应用在 Kubernetes 平台上实现了弹性伸缩策略，而近年来无服务器计算（Serverless Computing）作为一种全新的云计算技术架构在云原生应用中占有重要地位，在未来工作中，考虑将弹性伸缩策略应用于 Serverless 云场景中，探索更加灵活和可扩展的弹性伸缩策略，以适应不同场景和需求的变化。

参考文献

[1] 陈全, 邓倩妮. 云计算及其关键技术[J]. 计算机应用, 2009, 29(9): 2562-2567.

[2] 罗军舟, 金嘉晖, 宋爱波, 等. 云计算:体系架构与关键技术[J]. 通信学报, 2011, 32(07): 3-21.

[3] Al-Dhuraibi Y, Paraiso F, Djarallah N, et al. Elasticity in cloud computing: state of the art and research challenges[J]. IEEE Transactions on Services Computing, 2017, 11(2): 430-447.

[4] Kumbhare A G, Simmhan Y, Frincu M, et al. Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure[J]. IEEE Transactions on Cloud Computing, 2015, 3(2): 105-118.

[5] Singh P, Gupta P, Jyoti K, et al. Research on auto-scaling of web applications in cloud: survey, trends and future directions[J]. Scalable Computing: Practice and Experience, 2019, 20(2): 399-432.

[6] Calheiros R N, Masoumi E, Ranjan R, et al. Workload prediction using ARIMA model and its impact on cloud applications' QoS[J]. IEEE Transactions on Cloud Computing, 2014, 3(4): 449-458.

[7] Tang X, Liao X, Zheng J, et al. Energy efficient job scheduling with workload prediction on cloud data center[J]. Cluster Computing, 2018, 21: 1581-1593.

[8] 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.

[9] Liu C, Liu C, Shang Y, et al. An adaptive prediction approach based on workload pattern discrimination in the cloud[J]. Journal of Network and Computer Applications, 2017, 80: 35-44.

[10] Gao J, Wang H, Shen H. Machine learning based workload prediction in cloud computing[C]. 2020 29th International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020: 1-9.

[11] Borkowski M, Schulte S, Hochreiner C. Predicting cloud resource utilization[C]. Proceedings of the 9th International Conference on Utility and Cloud Computing. 2016: 37-42.

[12] Duggan M, Mason K, Duggan J, et al. Predicting host CPU utilization in cloud computing using recurrent neural networks[C]. 2017 12th International Conference

for Internet Technology and Secured Transactions (ICITST). IEEE, 2017: 67-72.

[13] Song B, Yu Y, Zhou Y, et al. Host load prediction with long short-term memory in cloud computing[J]. The Journal of Supercomputing, 2018, 74: 6554-6568.

[14] 贺小伟, 徐靖杰, 王宾等. 基于 GRU-LSTM 组合模型的云计算资源负载预测研究[J]. 计算机工程, 2022, 48(05): 11-17.

[15] Cheng Y, Wang C, Yu H, et al. Gru-es: Resource usage prediction of cloud workloads using a novel hybrid method[C]. 2019 IEEE 21st International Conference on High Performance Computing and Communications(HPCC). IEEE, 2019: 1249-1256.

[16] Chung J, Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv preprint arXiv:1412.3555, 2014.

[17] Tang X, Liu Q, Dong Y, et al. Fisher: An efficient container load prediction model with deep neural network in clouds[C]. 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications (ISPA). IEEE, 2018: 199-206.

[18] Nguyen H M, Kalra G, Kim D. Host load prediction in cloud computing using long short-term memory encoder–decoder[J]. The Journal of Supercomputing, 2019, 75: 7592-7605.

[19] Borovykh A, Bohte S, Oosterlee C W. Dilated convolutional neural networks for time series forecasting[J]. Journal of Computational Finance, 2018, 22: 73-101.

[20] Wang K, Li K, Zhou L, et al. Multiple convolutional neural networks for multivariate time series prediction[J]. Neurocomputing, 2019, 360: 107-119.

[21] 廖恩红, 舒娜, 李加伟, 等. 基于时序卷积网络的云服务器性能预测模型[J]. 华南师范大学学报(自然科学版), 2020, 52(04): 107-113.

[22] Golshani E, Ashtiani M. Proactive auto-scaling for cloud environments using temporal convolutional neural networks[J]. Journal of Parallel and Distributed Computing, 2021, 154: 119-141.

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

[24] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[25] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv preprint arXiv:2010.11929, 2020.

[26] Li S, Jin X, Xuan Y, et al. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting[J]. Advances in Neural Information Processing Systems, 2019, 32.

[27] Zhou H, Zhang S, Peng J, et al. Informer: Beyond efficient transformer for long sequence time-series forecasting[C]. Proceedings of the AAAI Conference on Artificial Intelligence. 2021, 35(12): 11106-11115.

[28] Kim T Y, Cho S B. Predicting residential energy consumption using CNN-LSTM neural networks[J]. Energy, 2019, 182: 72-81.

[29] Le T, Vo M T, Vo B, et al. Improving electric energy consumption prediction using CNN and Bi-LSTM[J]. Applied Sciences, 2019, 9(20): 4237.

[30] Qin D, Yu J, Zou G, et al. A novel combined prediction scheme based on CNN and LSTM for urban PM 2.5 concentration[J]. IEEE Access, 2019, 7: 20050-20059.

[31] Ma L, Tian S. A hybrid CNN-LSTM model for aircraft 4D trajectory prediction[J]. IEEE Access, 2020, 8: 134668-134680.

[32] Wang H, Yang Z, Yu Q, et al. Online reliability time series prediction via convolutional neural network and long short term memory for service-oriented systems[J]. Knowledge-Based Systems, 2018, 159: 132-147.

[33] 王悦悦, 谢晓兰, 郭杨, 等. 基于自适应神经网络的云资源预测模型[J]. 科学技术与工程, 2021, 21(25): 10814-10819.

[34] Xu M, Song C, Wu H, et al. esDNN: deep neural network based multivariate workload prediction in cloud computing environments[J]. ACM Transactions on Internet Technology (TOIT), 2022, 22(3): 1-24.

[35] Karim M E, Maswood M M S, Das S, et al. BHyPreC: a novel Bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine[J]. IEEE Access, 2021, 9: 131476-131495.

[36] Al-Asaly M S, Bencherif M A, Alsanad A, et al. A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment[J]. Neural Computing and Applications, 2021: 1-18.

[37] Patel E, Kushwaha D S. A hybrid CNN-LSTM model for predicting server load in cloud computing[J]. The Journal of Supercomputing, 2022, 78(8): 1-30.

[38] Hasan M Z, Magana E, Clemm A, et al. Integrated and autonomic cloud resource scaling[C]. 2012 IEEE Network Operations and Management Symposium. IEEE, 2012: 1327-1334.

[39] Vaquero L M, Morán D, Galán F, et al. Towards runtime reconfiguration of

application control policies in the cloud[J]. Journal of Network and Systems Management, 2012, 20: 489-512.

[40] Beloglazov A, Buyya R. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers[C]. Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science. ACM, 2010: 1-6.

[41] Botrán T L, Alonso J M, Lozano J A. Comparison of auto-scaling techniques for cloud environments[C]. Actas de las XXIV Jornadas de Paralelismo. Limencop, 2013: 187-192.

[42] Ullah A, Li J, Shen Y, et al. A control theoretical view of cloud elasticity: taxonomy, survey and challenges[J]. Cluster Computing, 2018, 21: 1735-1764.

[43] Barna C, Fokaefs M, Litoiu M, et al. Cloud adaptation with control theory in industrial clouds[C]. 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW). IEEE, 2016: 231-238.

[44] Zhu Q, Agrawal G. Resource provisioning with budget constraints for adaptive applications in cloud environments[C]. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. 2010: 304-307.

[45] Gias A U, Casale G, Woodside M. ATOM: Model-driven autoscaling for microservices[C]. 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019: 1994-2004.

[46] Tong J, Wei M, Pan M, et al. A holistic auto-scaling algorithm for multi-service applications based on balanced queuing network[C]. 2021 IEEE International Conference on Web Services (ICWS). IEEE, 2021: 531-540.

[47] Ding Z, Huang Q. COPA: A combined autoscaling method for Kubernetes[C]. 2021 IEEE International Conference on Web Services (ICWS). IEEE, 2021: 416-425.

[48] Sutton R S, Barto A G. Introduction to reinforcement learning[M]. Cambridge: MIT press, 1998.

[49] Garí Y, Monge D A, Pacini E, et al. Reinforcement learning-based application autoscaling in the cloud: A survey[J]. Engineering Applications of Artificial Intelligence, 2021, 102: 104288.

[50] Horovitz S, Arian Y. Efficient cloud auto-scaling with SLA objective using Q-learning[C]. 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, 2018: 85-92.

[51] 帅斌, 龙士工. 一种基于强化学习的云应用弹性伸缩算法[J]. 计算机应用与

软件, 2022, 39(09): 285-290.

[52] Zhang S, Wu T, Pan M, et al. A-SARSA: A predictive container auto-scaling algorithm based on reinforcement learning[C]. 2020 IEEE International Conference on Web Services (ICWS). IEEE, 2020: 489-497.

[53] Tesauro G, Jong N K, Das R, et al. A hybrid reinforcement learning approach to autonomic resource allocation[C]. 2006 IEEE International Conference on Autonomic Computing. IEEE, 2006: 65-73.

[54] Xiao Z, Hu S. DScaler: A Horizontal Autoscaler of Microservice Based on Deep Reinforcement Learning[C]. 2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2022: 1-6.

[55] 闫承鑫, 陈宁江, 刘文斌, 等. 面向突变负载的容器资源弹性供给服务策略[J]. 小型微型计算机系统, 2019, 40(04): 787-792.

[56] Toka L, Dobreff G, Fodor B, et al. Adaptive AI-based auto-scaling for Kubernetes[C]. 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 2020: 599-608.

[57] Dang-Quang N M, Yoo M. Deep learning-based autoscaling using bidirectional long short-term memory for Kubernetes[J]. Applied Sciences, 2021, 11(9): 3835.

[58] 单朋荣, 杨美红, 赵志刚等. 基于 Kubernetes 云平台的弹性伸缩方案设计与实现[J]. 计算机工程, 2021, 47(01): 312-320.

[59] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.

[60] Yu F, Koltun V. Multi-scale context aggregation by dilated convolutions[J]. arXiv preprint arXiv:1511.07122, 2015.

[61] Mao J, Xu W, Yang Y, et al. Deep captioning with multimodal recurrent neural networks (m-rnn)[J]. arXiv preprint arXiv:1412.6632, 2014.

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

[63] Graves A. Generating sequences with recurrent neural networks[J]. arXiv preprint arXiv:1308.0850, 2013.

[64] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[65] Botvinick M, Ritter S, Wang J X, et al. Reinforcement learning, fast and slow[J]. Trends in Cognitive Sciences, 2019, 23(5): 408-422.

[66] Wang Z, Hong T. Reinforcement learning for building controls: The opportunities

and challenges[J]. Applied Energy, 2020, 269: 115036.

[67] Ni Z, Paul S. A multistage game in smart grid security: A reinforcement learning solution[J]. IEEE Transactions on Neural Networks and Learning Systems, 2019, 30(9): 2684-2695.

[68] Torrado R R, Bontrager P, Togelius J, et al. Deep reinforcement learning for general video game ai[C]. 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2018: 1-8.

[69] Shani G, Heckerman D, Brafman R I, et al. An MDP-based recommender system[J]. Journal of Machine Learning Research, 2005, 6(9): 1265-1295.

[70] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.

[71] Ba J L, Kiros J R, Hinton G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450, 2016.

[72] Reiss C, Wilkes J, Hellerstein J L. Google cluster-usage traces: format+ schema[J]. Google Inc., White thesis, 2011, 1: 1-14.

[73] Guo J, Chang Z, Wang S, et al. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces[C]. Proceedings of the International Symposium on Quality of Service. 2019: 1-10.

[74] 陈晓宇, 杨川胡, 陈啸. 深入浅出 Prometheus：原理、应用、源码与拓展详解[M]. 北京: 电子工业出版社, 2019. 13-17.

[75] Brockman G, Cheung V, Pettersson L, et al. Openai gym[J]. arXiv preprint arXiv:1606.01540, 2016.