计算机工程
Computer Engineering
ISSN 1000-3428,CN 31-1289/TP

《计算机工程》网络首发论文

题目：基于 Kubernetes 的集群节能策略研究
作者：李俊俊，董建刚，李坤
网络首发日期：2024-01-22
引用格式：李俊俊，董建刚，李坤．基于 Kubernetes 的集群节能策略研究[J/OL]．计算机工程. https://link.cnki.net/urlid/31.1289.TP.20240119.1457.011

网络首发时间：2024-01-22 19:24:27
网络首发地址：https://link.cnki.net/urlid/31.1289.TP.20240119.1457.011

计 算 机 工 程
Computer Engineering

基于 Kubernetes 的集群节能策略研究

李俊俊，董建刚，李坤
(新疆大学，软件学院，新疆 乌鲁木齐 830091)

摘　要：近年来，随着集群容器化与虚拟化技术的不断发展，Kubernetes 已经成为容器编排领域的事实标准。在 Kubernetes 中，HPA 具备自动扩展 Pod 的能力，它可以根据流量的波动情况，在高峰时增加 Pod 数量以应对需求，而在低谷时减少数量以节省资源。然而，由于 HPA 是根据当前 Pod 的性能指标来进行扩展的，当流量激增时，可能会对应用服务的可用性产生不利影响。并且当压力较小时，算力资源的空载导致电子资源的浪费。为了改善上述问题，本文研究并验证了一种基于时序预测的集群资源自动缩放与智能休眠唤醒策略，该策略使用了 GC-TimesNet 模型对集群资源的使用情况进行预测。当资源利用率较低时，该策略计算出需要关闭的算力节点数量，将这些节点设置为不可调度状态，并驱逐节点现有的 Pod，然后将这些机器置于睡眠状态。相反，当资源需求增加时，会唤醒足够数量的机器，并通过 HPA 控制器增加所需数量的 Pod 副本。实验结果表明，该方法能够较为准确地预测集群负载的变化趋势，结合实施智能的休眠与唤醒策略，提升优化集群的运维管理能力，最大程度地提高了计算资源的利用率，为降低集群能源开销提供了数据支撑，实现节能减排。

关键词：Kubernetes；容器编排；集群节能；时间序列预测；GC-TimesNet 模型；卷积神经网络；注意力机制；节能减排

Research on Energy-Saving Strategies for Kubernetes-based Clusters

LI Junjun, DONG Jiangang, LI Kun

(School of Software, Xinjiang University, Urumqi 830091, China)

【Abstract】In recent years, with the continuous development of cluster containerization and virtualization technologies, Kubernetes has emerged as the de facto standard in the field of container orchestration. Within Kubernetes, HPA possesses the automatic Pod scaling capability. It can adjust the number of Pods based on fluctuations in traffic, increasing the Pod count during peak periods to meet demand and reducing it during off-peak times to conserve resources. However, because HPA scales based on the current performance metrics of Pods, sudden traffic surges can potentially have a detrimental impact on the availability of application services. Additionally, during periods of low demand, idle compute resources lead to resource wastage. To address these challenges, this paper investigates and validates a cluster resource autoscaling and intelligent sleep-wake strategy based on time series forecasting. This strategy utilizes the GC-TimesNet model to predict cluster resource usage. When resource utilization is low, the strategy calculates the number of compute nodes that need to be shut down, marks these nodes as unschedulable, evicts existing Pods from them, and then places these machines in a sleep state. Conversely, when resource demand increases, a sufficient number of machines are awakened, and the HPA controller is used to increase the required number of Pod replicas. Experimental results demonstrate that this approach can reasonably accurately predict trends in cluster load changes. When combined with the implementation of intelligent sleep and wake strategies, it enhances operational management capabilities for optimizing clusters, maximizes the utilization of computing resources, provides data support for reducing cluster energy expenses, and achieves energy savings and emissions reduction.

【Key words】Kubernetes; container orchestration; cluster energy efficiency; time series forecasting; GC-TimesNet model; convolutional neural network; attention mechanism; energy saving and emission reduction.

0　概述

随着社会与计算机技术的发展，互联网应用已经渗透到人们生活的各个领域，这些应用不断产生和请求大量的数据，导致数据和网络流量在短时间内迅速激增[1]，服务集群的运维成本也急剧攀升。

作者简介：李俊俊（1999—），男，硕士研究生，主研方向为时间序列预测、云计算；董建刚（1974—），男，硕士，高级工程师、硕士生导师，主研方向为软件工程；李坤（1995—），男，硕士研究生，主研方向为时序预测、特征选择。
E-mail：107552204765@stu.xju.edu.cn

在这种背景下，云计算技术一经提出，便以其独特的优势得到了广泛的关注和应用。传统的云计算服务提供商通常以虚拟机为最小资源调度单位，这种方式对底层系统有着较高的依赖性，对云平台服务层的可用性与可维护性有一定的负面影响[2]。为了应对这些问题，容器化技术如 Docker 等开始兴起。容器技术实现了对虚拟技术的“轻量化”，容器能够独立运行于宿主机之上，与其他进程互不干扰[3]。Docker 提供一致的运行环境并与操作系统解耦，从而实现“构建一次，随处运行”的目标[4]。然而，当容器数量较多时，手动管理它们是不切实际的。因此，使用自动化工具对容器进行编排管理是必要的[5]。在多个容器编排工具中，Kubernetes（以下简称为 k8s）已经成为了流行的事实标准。k8s 是谷歌公司的第三代容器编排管理工具[6]，支持任何实现 Kubernetes CRI（容器运行环境）的容器工具。它利用容器化技术来处理应用程序扩展和故障转移，并提供部署、服务定义、服务发现以及基本的负载均衡功能[7]。k8s 中的最小资源调度单位称为 Pod，它是一个或多个容器的集合。HPA（Horizontal Pod Autoscaler）是 k8s 实现集群 Pod 水平伸缩的控制器，当现有的 Pod 负载较高时，HPA 将会计算出期望的 Pod 副本数并创建，反之则会缩减 Pod 的数量[8]。尽管 k8s 已有相对完善的自动扩展机制，但其资源报警是基于阈值实现的响应式机制[9]。当流量激增时，k8s 的资源伸缩反应存在一定的滞后，这可能降低应用的服务质量，影响用户的使用体验。此外，集群负载处于较低水平时，k8s 虽然会缩减 Pod 副本数量，但空闲节点仍处于唤醒状态，依然会浪费一定的资源。此时，可以通过建立资源调度机制，关停部分空闲资源，实现能耗有效管理。

为改进和提高资源监控和弹性管理，本文提出了一种基于预测值的资源管理策略。依据集群与 Pod 的资源历史使用情况来预测未来一段时间内的资源占用趋势，根据趋势阈值，计算出需调整 Node 与 Pod 的数量，更精准地实现休眠与唤醒策略，为资源调度机制提供数据支持。

1 相关研究

有效的管理集群资源可以提升其可用性并降低资源消耗以减少企业开支，同时也有助于碳减排。因此，使用预测算法提升集群资源的利用率成为了企业与学术界关心的重要内容。石硕等人针对单个应用负载过高或过低的情况，提出了一种基于综合负载的预测式弹性伸缩策略，利用双层 LSTM（长短时记忆神经网络）模型预测未来一段时间内的综合负载情况，将其作为 k8s 弹性伸缩的依据[10]。为了解决 k8s 默认的弹性伸缩对于动态的 Web 请求不灵活和不主动的问题，陈烨提出了一种面向 Web 应用的 k8s 容器弹性伸缩的方法，使用改进的 LSTM 预测出理想的 Pod 数量并进行主动伸缩[11]。Balla D 等人为了克服 k8s 默认的伸缩策略无法适应集群环境的变化而提出了一种 Libra 扩展算法，通过收集 Pod 的性能指标，决定是否对 Pod 进行水平或垂直伸缩[12]。Farahnakian F 等人提出基于线性回归方法的 CPU 使用率预测方法，当负载不足时，将虚拟机迁移出部分主机并使这部分主机进入休眠状态[13]。Calheiros R N 等人提出基于 ARIMA 差分自回归移动平均模型的云工作量预测[14]。D. Janardhanan 等人使用了 LSTM 对数据中心机器 CPU 使用率进行预测[15]。为了解析复杂的时间变化，Wu H 等人引入了一种称为 TimesNet 的方法，将一维的时间序列扩展到 2 维空间进行分析[16]。此外，基于注意力机制的 Transformer 架构在时间序列预测任务中表现出良好的性能，能较好的捕捉时间之间的长期依赖性[17]。

然而，真实世界的时间序列的变化通常过于复杂，难以被预定义的模式捕捉，限制了这些传统方法的适用性。而基于 RNN（循环神经网络）及其变体通常难以捕捉到序列的长期依赖关系，重要的信息可能会被忽略[18]。尽管 TimesNet 表现出良好的性能，但其更多的是用于捕捉时间的长期相关性和多周期性，而难以识别部分隐藏在复杂时间序列中的局部特征，并且容易受到噪声与异常值的干扰。注意力机制则是难以从离散的时间点中找到可靠的依赖关系，因为在长时预测中，时间之间的依赖关系可能深深地隐藏在复杂的时间模式之中[19]。

为了解决上述问题，本文建立一种时间序列预测模型 GC-TimesNet，使用高斯白噪对训练数据集混入随机噪声，提升模型对噪声与异常值的适应能力。利用 TimesNet 捕捉序列的长期依赖关系和周期

性，并设计一种 CBAM-Inception 卷积块以用以改善其提取复杂时间序列中的局部特征能力。

2 Kubernetes 默认自动缩放策略

2.1 Kubernetes 简介

k8s 集群由一个或多个 Master 主节点与一个或多个 Node 从节点组成[20]。Master 节点控制资源调度与访问控制等工作，是集群的控制中心，默认不在其中创建 Pod。Node 节点上运行着多个 Pod，每个 Pod 都包含一个或多个容器。这些容器共享 Node 节点的计算和存储资源，运行应用程序和服务，其总体架构如图 1 所示。

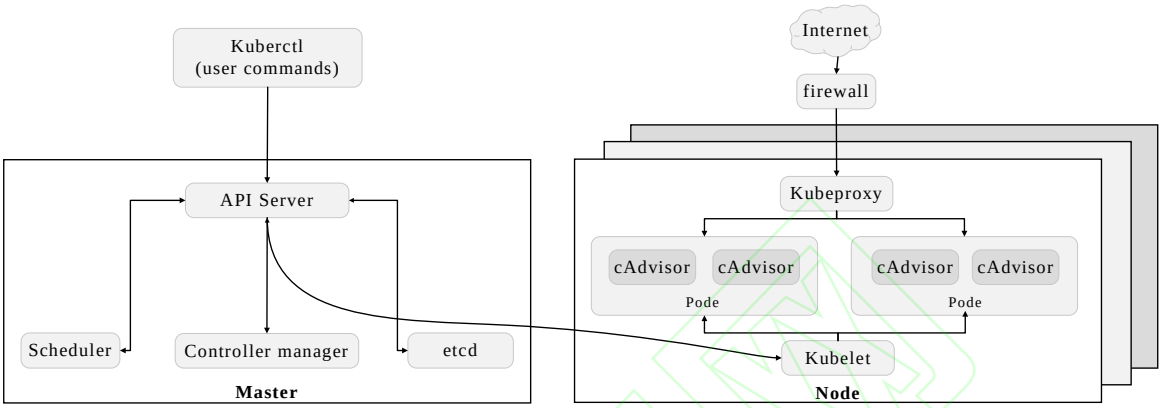

图 1　Kubernetes 总体架构图

Fig.1　Kubernetes Overall Architecture Diagram

2.2 自动缩放机制

在 k8s 集群中，由 HPA 控制器控制 Pod 的缩放，根据观察到的 Pod 当前资源的占用率与预定阈值对比，做出是否需要增减 Pod 副本数量的决策。该机制通过 informer 获取到当前 HPA 对象，然后 Merics Server 将采集到的 Pod 性能指标数据通过聚合 API 提供给 HPA。最后，HPA 根据期望副本数公式计算出目标期望的 Pod 副本数后与当前存在的 Pod 数量进行对比，决定是否扩缩容并完成 Pod 的调度与副本数量控制。HPA 使用式（1） 计算出期望的 Pod 副本数。

$$ER = \lceil CR \bullet (CM / EM) \rceil \tag{1}$$

其中 *ER* 为最终计算出的期望副本数，*CR* 为当前已有的 Pod 副本数，*CM* 表示当前指标值，*EM* 表示为期望的指标值。即期望副本数等于当前副本数乘以(当前指标值 / 期望期望指标值)，最后向上取整。HPA 在得到期望的副本数后，做出缩扩容的决策。但做决策之前还需要一个周期，例如当发现一个 Pod 持续一段时间内的 cpu 占用率处以一个较高的状态，那么就增加 Pod 的副本数量以减轻压力。而当一段时间压力都较低时，则减少 Pod 副本的数量以节省集群资源。

2.3 Kubernetes 集群存在的问题

由上述简介可以知道，k8s 已经在一定程度上实现了对资源调度的自动化。但通过对 k8s 默认的缩放策略研究发现，由于 k8s 中的资源报警策略是基于预定的阈值的响应式机制，通过当前的负载数据与预定的阈值进行对比，从而对 Pod 进行弹性伸缩。也是就是说，k8s 必须等到流量高峰来临，Pod 的负载增大后才能被 HPA 感知，这时再增加副本数量已经相对滞后。并且当 HPA 做出扩容决策后，Pod 的初始化也需要一定的时间，这种机制难以应对流量突发激增，而人为的对应用负载趋势进行判断并不现实，也更容易出现错误的决策。此外，k8s 集群在低负载期间，虽然会缩减 Pod 的数量，节省集群的资源，但一部分空闲的节点仍处于唤醒状态，仍然会浪费一定的资源。

3 预测模型 GC-TimesNet 介绍

TimesNet 在复杂的数据集或相对较小的数据集中容易过度拟合训练数据，导致在没见过的数据中可能过拟合。而现实世界的实际数据通常会充斥着不确定的噪声和异常值，TimesNet 可能难以适应这种情况。并且，其捕捉隐藏在复杂时间序列中的局部特征能力仍存在不足。为了解决上述缺点，本文基于 TimesNet 提出了一种改进的模型

GC-TimesNet。该模型首先使用高斯白噪（White Gaussian Noise）对训练数据混入一定的噪声，模拟现实世界中的噪声源、提升模型的鲁棒性，使其更能够适应真实世界中的不确定性。提出一种带有注意力机制的多尺度卷积块 CBAM-Inception，在训练过程中使用不同大小的卷积核多尺度提取特征，并在中间尺度引入 CBAM 注意力机制。通过通道注意力和空间注意力来提高模型的局部感知能力，自适应地学习通道和空间注意力权重，以提高卷积神经网络的特征表达能力，在不同维度上捕获时序特征之间的相关性，尽可能的提高模型预测的准确性。

3.1 White Gaussian Noise

在神经网络训练过程中添加高斯白噪声能够改善模型的泛化性，这种带有噪声的训练，也相当于一种形式的正则化[21]。“高斯”是指噪声的概率分布遵循正态分布，在高斯白噪声中，每个时间点上的噪声值都是从正态分布中独立地随机抽取。白噪声是一种特殊类型的信号或噪声，其一阶矩（均值）为常数且不随时间变化，同时其二阶矩（方差）是有限的。这意味着在白噪声中，噪声的平均值不随时间变化，而且噪声的幅度是有限的。此外，白噪声的不同时间点上的样本之间是不相关的，也就是说，噪声在时间上没有相关性，它们之间是独立的。本文按照信噪比 SNR 大小给训练数据添加白噪声，具体公式如（2）所示。其中 P_{singal} 表示信号的有效功率，P_{noise} 表示噪声的有效功率。

$$SNR = 10\log_{10}\frac{P_{singal}}{P_{noise}} \tag{2}$$

3.2 CBAM 注意力机制

CBAM（Convolutional Block Attention Module）是一种用于前馈卷积神经网络的注意力模块[22]。给定一个特征图，CBAM 会沿着通道与空间两个维度依次推断注意力图，然后将注意力图乘以输入的特征图进行自适应特征修饰。并且 CBAM 是一种轻量级的注意力模块，可以将其集成到任何卷积神经网络架构中。CBAM 整体结构如图 2 所示。

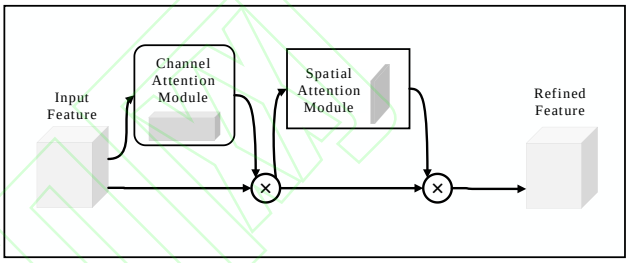

图 2　CBAM 整体结构图

Fig.2　CBAM Overall Architecture Diagram

3.2.1 通道注意力

CBAM 的通道注意力利用特征之间的通道关系产生一个通道注意力图，模块结构如图 3 所示。

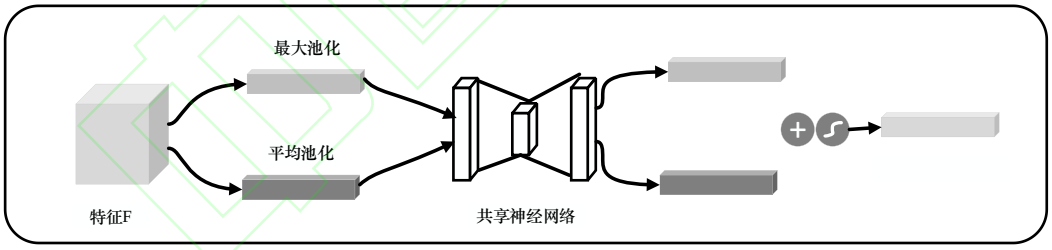

图 3　通道注意力模块示意图

Fig.3　Schematic diagram of channel attention module

经过通道注意力模块时，会将尺寸为高×宽×通道数的特征图 F 经过平均池化（AvgPool）与全局最大池化（MaxPool）后，获得两个 1×1×通道数的通道特征，接着将这两个通道特征输入进两层的共享神经网络(MLP)，共享神经网络默认使用 ReLu 激活函数，本方法将其更换为 GELU (Gaussian Error Linear Units)，提高模型的非线性表达能力[23]。其近似计算公式如式（3）所示，其中 x 为输入值。共享神经网络输出两个特征的元素，将其中的元素进行乘积与加和，最后再通过 Sigmoid 激活函数（σ）获得通道注意力图 M_c。综上，通道注意力计算过程总结为如式（4）所示。其中 W_0 与 W_1 分别表示隐层权重和输出层权重，F^c_{avg} 和 F^c_{max} 分别表示为通道维度的平均池化与最大池化特征图。

$$0.5x(1+\tanh(\sqrt{\frac{2}{x}}\bullet(x+0.044715x^3))) \tag{3}$$

$$M_c = \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F))) = \sigma(W_1(W_0(F^c_{avg})) + W_1(W_0(F^c_{\max}))) \quad (4)$$

3.2.2 空间注意力

特征图经过通道注意力模块时，主要关注图像中有用的特征是什么。而将输出的特征经过空间注意力时，主要关注于特征信息在哪，对通道注意力进行补充，形成含有通道与空间两个维度的特征图。空间注意力结构如图 4 所示。空间注意力模块将经过通道注意力模块处理过的特征图 F'的各个通道通过使用最大（MaxPool）与平均池化(AvgPool)将维度进行压缩以聚合通道信息，然后使用一个 3×3 的卷积层对这些通道特征进行融合，最后使用 Sigmoid 激活函数（σ）将权重系数与输入的特征 F'相乘得到空间注意力图 M_s，形式化描述如式（5）所示。其中，$f^{3\times3}$ 表示经过卷积核大小为 3×3 的卷积层。

$$M_s = \sigma(f^{3\times3}([AvgPool(F');MaxPool(F')])) \quad (5)$$

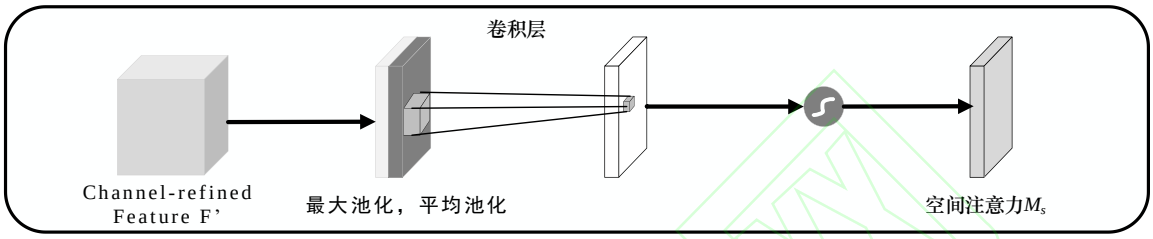

图 4 空间注意力模块示意图

Fig.4 Schematic diagram of Spatial attention module

3.3 CBAM-Incpetion

谷歌团队于 2014 年在 GoogLeNet 中提出了 Inception（初始块）的概念，用以构造基础神经元，搭建一个稀疏的高性能计算的网络结构[24]。在 GoogleNet 问世之前，视觉骨干网络的主要发展方向是通过增加网络的深度和神经元数量来提高性能。然而，纯粹地增加网络结构的规模会导致模型参数的急剧增加，容易引发过拟合。随着网络深度的增加，梯度的传播变得更加困难，同时计算复杂性也急剧增加，这使得模型应用变得难以应用。

GoogleNet 的问世改变了这一格局。与传统的深度网络不同，GoogleNet 采用了一种全新的网络结构，即 Inception 结构。这个结构在网络中引入了并行的卷积操作，允许网络同时学习不同尺度和层次的特征。这种并行结构有助于减少参数的数量，同时提高了网络的性能。本文在基于 Inception 初始块的思想上，设计了一种带有 CBAM 注意力机制的 CBAM-Incpetion 卷积块，在多个尺度中进行特征提取，并在 5×5 的卷积尺度中应用 CBAM，以改进网络在通道和空间位置的特征学习。在经过 CBAM 后添加一个 3×3 的卷积，深度提取出更具抽象和表征能力的特征，提高性能与泛化能力。该模块用以优化 TimesNet 中的 TimesBlock 以提高模型性能，其结构如图 5 所示。

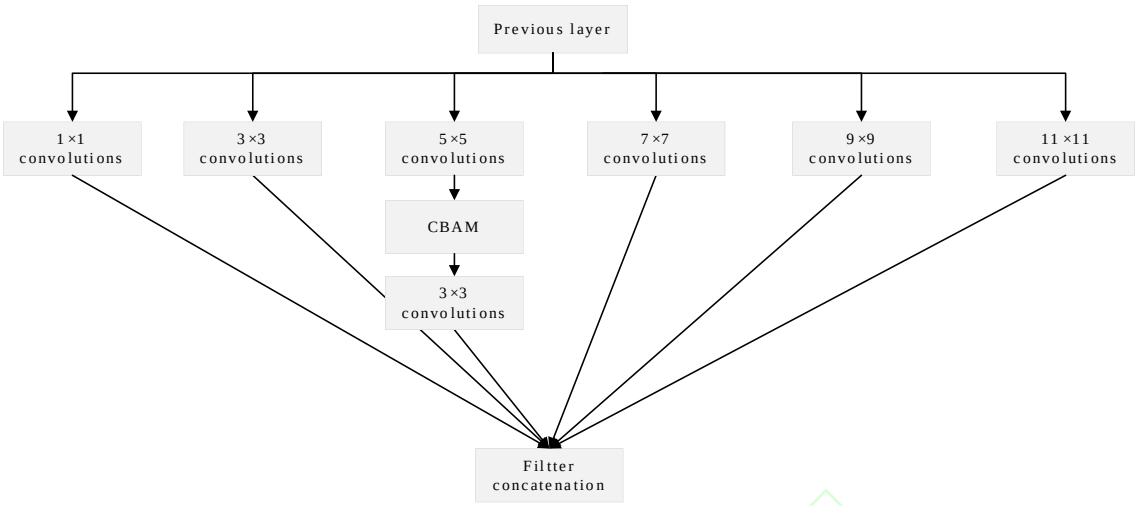

图 5　CBAM-Incpetion 结构图

Fig.5　CBAM-Inception Architecture Diagram

3.4 TimesNet

在实际情况下，时间序列数据通常呈现出多个周期性，并且这些周期之间相互作用并产生影响，引起非常复杂的波动。这种复杂的波动对预测中的时间序列数据建模带来了极大的挑战。传统的时序分析方法通常侧重于时间序列的连续性与递归性，但忽略了其多周期性。与之不同，TimesNet 使用模块化的设计思路，即一个模块捕捉由某一特定周期主导的时序变化，将复杂的时间变化分解，有利于后续建模。但一维的时间序列结构难以表现出周期内与周期间两种不同的时序变化。为此，TimesNet 通过模块化结构将复杂的时序变化分解至不同周期，并通过将原始的一维时间序列转化至二维空间实现周期内与周期之间时序变化的统一建模。

TimesNet 由多个堆叠的 TimesBlock 组成，TimesBlock 集成了上述过程，并使用 Inception 卷积块有效的对周期内与周期间的变化进行有效的建模，其结构如图 6 所示。首先输入的一维时间序列 $X_{1D} \in R^T \times C$，其周期性使用快速傅里叶变换（*FFT*）计算得出，见式（6）。

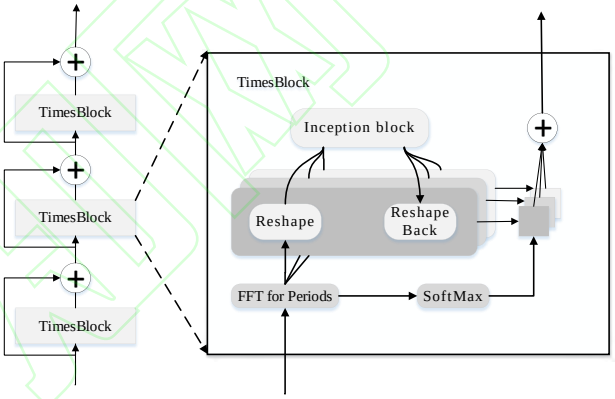

图 6　TimesNet 结构图

Fig.6　TimesNet Structure Diagram

$$A = Avg\left(Amp(FFT(X_{1D}))\right)$$
$$f_1,...,f_k = \arg TopK(A),\ f_* \in \{1,...,[\frac{T}{2}]\} \tag{6}$$
$$p_1,...,p_k = [\frac{T}{f_1}],...,[\frac{T}{f_k}]$$

其中，$A \in R^T$ 代表了 X_{1D} 中每个频率分量的强度，强度最大的 k 个频率$\{f_1,...,f_k\}$对应着最显著的 k 个周期长度$\{p_1,...,p_k\}$。接下来基于选定的周期将原始的一维时间序列 X_{1D} 进行折叠，该过程可形式化为式（7）所示。

$$X^i{}_{2D} = reshape_{p_i,f_i}(padding(X_{1D}))$$
$$i \in \{1,...,k\} \tag{7}$$

其中，padding 表示在序列末尾补 0 以使得序列长度可以被 p_i 整除。通过上述操作即可得到一组二维张量$\{X^1{}_{2D},X^2{}_{2D},...,X^i{}_{2D}\}$，$X^i{}_{2D}$ 对应着由周期 p_i 主导

的二维时序变化。该张量的每列与每行分别对应着相邻的时刻与相邻的周期，而邻近的时刻与周期往往含有着类似的时序变化。因此，上述二维张量表现出二维局部性，可以通过 2D 卷积捕捉信息。具体地，TimesBlock 包括的过程如图 7 所示。

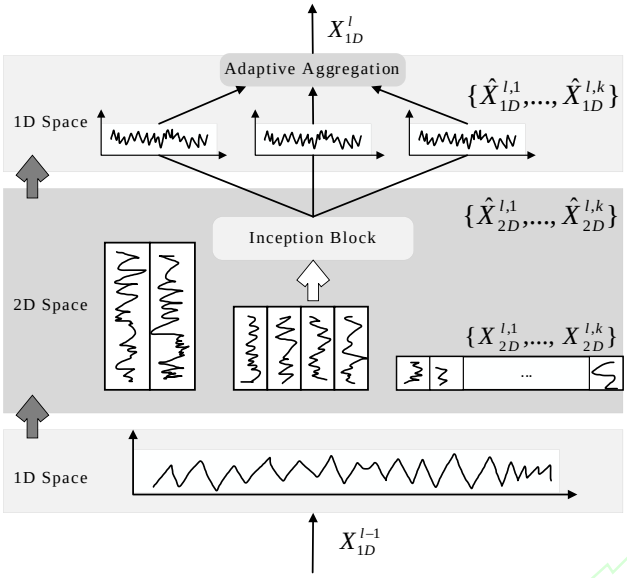

图 7　TimesBlock 过程图

Fig.7　TimesBlock Process Diagram

TimesBlock 其过程主要分为以下四步：

（1）首先对输入的一维时序特征提取其周期，并将其转换为二维张量以表示二维的时序变化，即上文列出的式（6）与式（7）。

（2）对于上面得到的二维张量，由于其具有二维局部性，TimesNet 使用了 Inception 进行 2D 卷积提取二维时序变化特征。

（3）将上一步提取后的二维时序变化特征转回一维进行信息聚合。

（4）自适应融合：将得到的一维特征对应频率的强度进行加权求和，得到最终输出。

3.5 GC-TimesNet 算法训练过程

针对 TimesNet 在复杂环境与局部特征捕捉存在的不足，本文建立一种基于 TimesNet 的改进预测算法模型 GC-TimesNet，以对集群资源负载进行预测。首先对训练数据混入高斯白噪声模拟现实世界的复杂情况，增强模型的鲁棒性，并在 TimesNet 网络的基础上，将一维时序转化至二维后，使用 CBAM-Inception 提取时序特征再转回一维。最后，将得到的一维特征对应频率的强度进行加权求和，全连接后得到输出。其训练过程如图 8 所示。

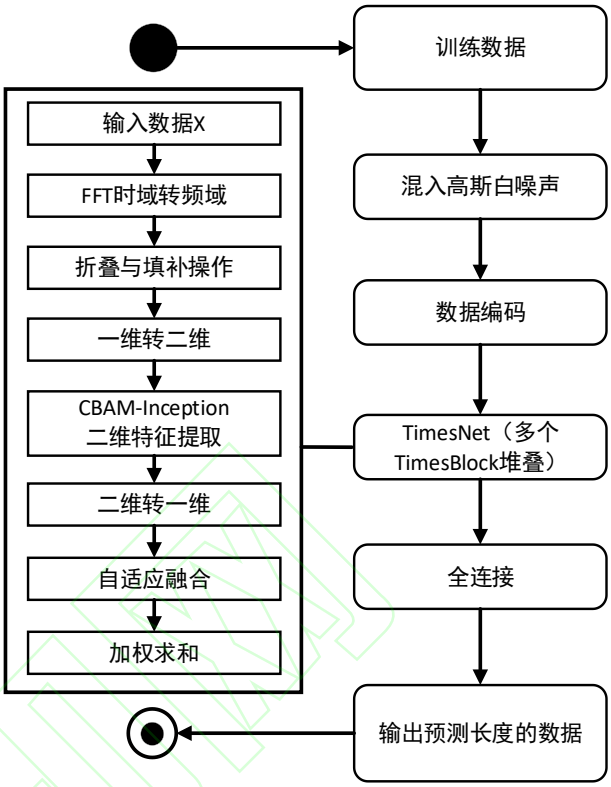

图 8　模型训练过程图

Fig.8　Model Training Process Diagram

4 预测算法实验

4.1 数据集选取

表 1　数据集基本特征

Table 1　Basic Characteristics of the Dataset

数据集	数据量	时间间隔（分钟）
ETTh1	17420	60
ETTh2	17420	60
ETTm2	69680	15
AzureTraces	8640	5

为了验证本文提出的模型的性能及其通用性，本文分别在公开的电力数据集 ETTh1、ETTh2、ETTm2 与微软的集群公开数据集 Microsoft Azure traces（以下简称 AzureTraces）等 4 种数据集进行测试，详细信息如表 1 所示。其中，AzureTraces 为相同时间集群内的机器 CPU 资源使用率进行加和，以模拟真实集群负载情况。所有的数据集划分为训练集（70%）、验证集(10%)、测试集(20%)。

4.2 实验细节设置

对比模型：模型对比实验除基准模型 TimesNet

外，还以深度学习中主流的时间序列模型 LightTS、Autoformer、Pyraformer、informer 作为对比模型。

实验环境：GC-TimesNet 代码由 python3.11.4 实现，使用 Pytorch 2.0.1 深度学习框架。在 window10 上进行实验，GPU 为单张 RTX4060。

在深度学习领域，参数设置是确保模型性能的关键因素之一。本文基于先前的时序预测研究和实践经验，配置了一系列超参数，以达到训练出最佳时序预测模型的目标。为了提高模型性能，采用了一些关键策略。首先，固定了随机数种子，以确保训练的可重复性，使结果更具可比性。其次，引入提前停止策略，这可以有效地避免模型过拟合，并减少训练时间，具体参数如表 2 所示。

表 2　模型主要参数配置

Table 2　Key Configuration Parameters of the Model

参数名称	参数值
激活函数	GELU
优化器	Adam
初始学习率	0.0001
损失函数	MSE
Dropout 系数	0.3
Inception 卷积核数量	6
批处理大小	32
CBAM 压缩比	8

4.3 模型评价标准

本文采用两种常用的评估指标，分别为均方差（Mean Squared Error，MSE）、平均绝对误差（Mean Absolute Error，MAE）.其计算公式见式（8）、（9）。其中 n 为数据预测值的个数，y_i 为数据的实际值，$\hat{y}_i$ 为数据的预测值。这两种指标的值越小，则代表误差越低，模型性能越好。

$$MSE=\frac{1}{n}\sum_{j=1}^{n}(y_i-\hat{y}_i)^2 \tag{8}$$

$$MAE=\frac{1}{n}\sum_{j=1}^{n}\left|y_i-\hat{y}_i\right| \tag{9}$$

4.4 对比实验

表 3 为 GC-TimesNet 与其他对比模型在测试集上的性能评估结果，其中，AzureTraces 的预测长度设置为[12，48，288]，即表示预测该数据集未来一小时、四小时、二十四小时内的预测结果。对于其他数据集，预测长度设置为[96，192，336，720]，表示预测 ETTm1 数据集的未来 24 小时、48 小时、84 小时、180 小时内的预测结果、ETTh1 和 ETTh2 未来 96 小时、192 小时、336 小时、720 小时内的预测结果。其中 GC-TimesNet 在上述的实验环境中进行，其余数据集的对比模型除 AzureTrances 数据集外的实验数据均来自于文献[16]。

表 3　不同数据集上 6 种模型的性能评估结果

Table 3　Performance Evaluation Results of 6 Models on Different Datasets

模型		GC-TimesNet		TimesNet		LightTS		Autoformer		Pyraformer		informer	
指标		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.176**	**0.246**	0.187	0.267	0.209	0.308	0.255	0.339	0.435	0.507	0.365	0.453
	192	**0.240**	**0.284**	0.249	0.309	0.311	0.382	0.281	0.340	0.730	0.673	0.533	0.563
	336	**0.302**	**0.321**	0.321	0.351	0.442	0.466	0.339	0.372	1.201	0.845	1.363	0.887
	720	**0.399**	**0.378**	0.408	0.403	0.675	0.587	0.433	0.432	3.625	1.451	3.379	1.338
	平均	**0.279**	**0.307**	0.291	0.333	0.409	0.436	0.327	0.371	1.498	0.869	1.410	0.810
ETTh1	96	**0.378**	**0.394**	0.384	0.402	0.424	0.432	0.449	0.459	0.664	0.612	0.865	0.713
	192	**0.431**	**0.424**	0.436	0.429	0.475	0.462	0.500	0.482	0.790	0.681	1.008	0.792
	336	**0.488**	**0.452**	0.491	0.429	0.518	0.488	0.521	0.496	0.891	0.738	1.107	0.809
	720	**0.503**	**0.476**	0.521	0.500	0.547	0.533	0.514	0.512	0.963	0.782	1.181	0.865
	平均	**0.450**	**0.437**	0.458	0.450	0.491	0.479	0.496	0.487	0.827	0.703	1.040	0.795
ETTh2	96	**0.306**	**0.339**	0.340	0.374	0.397	0.437	0.346	0.388	0.645	0.597	3.755	1.525
	192	**0.382**	**0.385**	0.402	0.414	0.520	0.504	0.456	0.452	0.788	0.683	5.602	1.931
	336	**0.420**	**0.414**	0.452	0.452	0.626	0.559	0.482	0.486	0.907	0.747	4.721	1.835
	720	**0.452**	**0.443**	0.462	0.468	0.863	0.672	0.515	0.511	0.963	0.783	3.647	1.625
	平均	**0.390**	**0.395**	0.414	0.427	0.602	0.543	0.450	0.459	0.826	0.703	4.431	1.729

Azure Trances	12	**0.183**	**0.325**	0.205	0.346	0.219	0.349	0.409	0.490	0.422	0.506	0.465	0.537
	48	**0.345**	**0.457**	0.357	0.463	0.430	0.512	0.532	0.550	0.617	0.635	0.541	0.586
	288	**0.525**	**0.568**	0.555	0.583	0.732	0.685	0.767	0.697	0.966	0.817	0.774	0.726
	平均	**0.351**	**0.450**	0.372	0.464	0.460	0.515	0.570	0.579	0.668	0.653	0.593	0.616

从表 3 可以看出，本文提出的 GC-TimesNet 模型在这四种数据集中的的 MSE、MAE 指标都相对较低。相对于基准模型 TimesNet，MSE 平均降低约 1.78%~6.15%，MAE 平均降低约 2.97%~8.47%。总体而言，本文提出的方法在一般长时预测上具有一定的优势。

4.5 消融实验

本次实验选择在 AzureTraces 数据集中进行，预测长度为 12，其指标评价结果如表 4 所示。从表 4 中可以看到，无论是 MSE 还是 MAE，GC-TimesNet 的性能都更加优异。这主要是因为在训练集混入了噪声，增强了模型对噪声与异常值的抵抗能力。其次，通过采用改进的 CBAM-Inception 卷积块，模型能够进行更深层次的特征提取，得到更具抽象和表征能力的特征表示，从而提升预测性能。

表 4　消融实验评价指标表

Table 4　Metrics Table for Ablation Experiment Evaluation

模型	指标	
	MSE	MAE
TimesNet	0.205	0.346
CBAM-Inception-TimesNet	0.195	0.335
Gaussian-TimesNet	0.189	0.330
GC-TimesNet	0.183	0.325

5 集群节能策略

5.1 策略实现过程介绍

k8s 默认的自动扩容必须要在 HPA 感知到资源使用率上升到达阈值后，才开始调整 Pod 数量。而这时再做出扩容动作已经相对滞后，可能会对应用提供的服务质量造成一定的影响。而默认的资源缩容，只是会单纯的减少集群中 Pod 的数量，当集群中的部分节点处于空闲状态时，仍然会浪费一定的资源。为此，本文设计了一种基于预测值的集群资源自动缩放与智能休眠唤醒策略，尝试改善这两个问题，具体由下列三个模块构成：

1）监控模块

Prometheus 是一个开源的集群资源监控组件，可以做到资源监控与报警，并且集成了时间序列数据库，其基本原理是通过 http 协议在预定的时间间隔内周期性抓取集群内的资源状态，本方法设置 Prometheus 每隔五分钟抓取一次资源指标。

2）预测模块

利用 Prometheus 提供的集群资源与 Pod 的使用情况的历史值后，调用 GC-TimesNet 预测算法，对未来一个小时的集群或 Pod 资源使用情况进行预测，得到 12 个长度的预测值。

3）智能休眠唤醒模块

将预测模块提供的集群资源使用情况与监控模块提供的当前集群资源使用情况进行对比。当预测到未来一个小时内的资源情况均低于当前的集群资源使用情况时，选取其中最高的预测值并使用式（10）计算出需要休眠的节点数量，然后随机选取这些数量的节点标记为污点，从而将这些节点的 Pod 漂移到其他的节点上。最后，给这些标记为污点的节点宿主机发送休眠指令，使其进入休眠状态。

$$SN=\lfloor (cru-pru)/120 \rfloor \tag{10}$$

式（10）中，SN 表示需要休眠的节点数量，cru 为当前集群资源的使用值，pru 为预测的资源使用值。假设集群中的节点有着相同的性能，记每个节点的可用 CPU 资源值为 100，则需要休眠的节点数量为当前资源的使用值与预测值的差值除以 1.2 个节点的资源值即 120 并向下取整，这样的目的是让集群有着一定的冗余资源，避免资源使用的波动情况，影响集群的服务质量。当预测值高于集群当前的资源使用值时，若符合式（11）则单纯扩充 Pod 副本数。否则提前通过式（12）计算出需要唤醒的节点数量 RN。当这些节点启动以后，将其设置为可调度状态。然后，再通过预测模块预测出期望 Pod 的数量并创建副本，就能做到 Pod 在用户访问高峰之前提前扩容，提升服务质量。

$$(\lceil cru \rceil - cru) > (pru - cru) \tag{11}$$

$$RN=\lceil (pru-cru)/80 \rceil \tag{12}$$

5.2 策略验证

对于本文提出的策略，在虚拟机中搭建两个 k8s

集群进行验证，一个使用 k8s 原生的资源自动缩放策略，一个使用本文提出的集群节能策略，均采用 Prometheus 采集监控集群资源使用情况。每个集群由一个 Master 节点和两个 Node 节点组成，每个节点的配置均为 2 核心 CPU 与 4GB 内存。两个集群都部署相同的 SpringBoot 项目的 Pod，每个 Pod 的 CPU 配额为 128m，伸缩阈值为百分之 50。

实验使用 JMeter 压测工具模拟用户请求并发，给予预测模型一定的历史值。首先启动一个含 30 个线程的线程组请求此 Pod，持续 10 分钟。在第一个线程组启动 2 分钟后再启动一组含有 30 个线程的线程组，持续 8 分钟，在第一个线程组启动 5 分钟后，再次启动一个含有 30 个线程组的线程组，持续 5 分钟。每次访问请求发送后均间隔 0.1 秒后再发送下一次请求，每隔 10 秒记录一次 Pod 副本数变化。副本变化如图 9 所示。

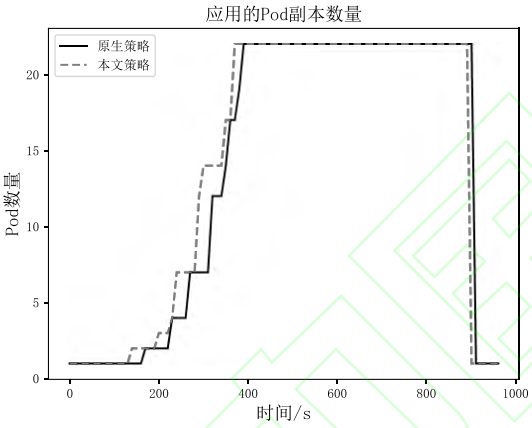

图 9　Pod 副本数量变化图

Fig.9　Pod Replica Count Variation Chart

可以看到，预测算法正确预测到了 CPU 使用率的上升变化并提前做出了扩容动作。脚本在 5 分钟左右正确发送了唤醒节点指令并扩容，在流量高峰结束五分钟后，将 node2 的 Pod 漂移至其他节点并在迁移完成后发送了一个休眠指令。集群 Node 的数量变化如图 10 所示。

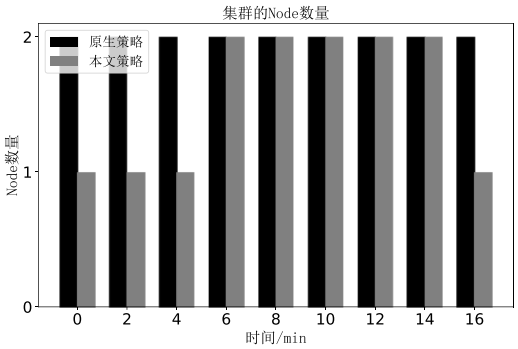

图 10 集群 Node 数量变化图

Fig.10　Cluster Node Count Variation Chart

6 结语

针对 k8s 集群原生的资源缩放策略与能耗管理机制，本文提出了一种基于预测值的集群资源自动缩放与智能休眠唤醒策略。使用 GC-TimesNet 模型预测未来一段时间的集群与 Pod 负载情况，在高峰时唤醒足够多的休眠节点并对 Pod 进行扩容，保证了应用的服务质量。在流量低谷时，漂移 Pod 并使闲置节点进入休眠状态。实验结果表明，本文提出的预测模型较好地预测了未来一定时间内资源的占用趋势，较为准确地预测到集群与 Pod 的负载变化，为资源调度机制提供有效的数据支持，提高应用系统可靠性的同时，降低能耗。

本文提出的模型以 CPU 使用率为例进行预测。事实上，该模型也可以通过多个性能指标如内存、网络使用情况等进行预测并输出多个预测值。该策略建立在同构计算集群中，未来将考虑将此策略扩充至异构计算集群中；结合调度机制，通过资源自动化的弹性伸缩和启停，提高运维质量，降低能耗。

参考文献

[1] Linthicum, David S. Cloud Computing Changes Data Integration Forever: What's Needed Right Now[J]. IEEE Cloud Computing, 2017, 4(3):50-53.

[2] 杨清波,陈振宇,刘东等. 基于容器的调控云 PaaS 平台的设计与实现[J]. 电网技术, 2020, 44(06):2030-2037
YANG Q B, CHEN Z Y, LIU D, et al. Design and Implementation of Dispatching and Control Cloud PaaS

Platform Based on Container[J]. Power System Technology, 2020, 44(06):2030-2037.(in Chinese)

[3] 刘洋,赵瑞锋,李波等.基于 Docker 技术的静态安全分析云计算应用[J]. 电力科学与技术学报, 2021, 36(04):181-187.
LIU Y, ZHAO R F, LI B, et al. Application research of static security analysis cloud computing based on Docker technology[J]. Journal of Electric Power Science and Technology, 2021, 36(04):181-187.(in Chinese)

[4] 吴逸文,张洋,王涛等. 从 Docker 容器看容器技术的发展:一种系统文献综述的视角[J]. 软件学报, 2023, 11(2560/TP): 1-25.
WU Y W, ZHANG Y, WANG T, et al. Development Exploration of Container Technology Through Docker Containers: A Systematic Literature Review Perspective[J]. Journal of Software, 2023, 11(2560/TP): 1-25.(in Chinese)

[5] 杜金涛,董建刚,承华青.Kubernetes 水平伸缩策略的改进研究[J].现代电子技术, 2023, 46(10):129-136.
DU J T,DONG J G, CHENG H Q. Research on improvement of Kubernetes horizontal scaling strategy[J]. Modern Electronics Technique,2023, 46(10):129-136. (in Chinese)

[6] Brendan B, Brain G, David O, et al. Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade[J]. Association for Computing Machinery, 2016, 14(1):70-93.

[7] Gilly, Katja, Carlos Juiz, Ramon Puigjaner. An up-to-date survey in web load balancing[J]. World Wide Web, 2011, 14(2): 105-131.

[8] Nguyen, Thanh-Tung, Yu-Jin Yeom, Taehong Kim. Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration[J]. Sensors, 2020, 20(16): 4621.

[9] 苏逸凡. 基于 Kubernetes 的动态资源调度策略研究与实现[D]. 浙江: 浙江理工大学, 2022.
Research and Implementation of Dynamic Resource Scheduling Strategy Based on Kubernetes[D]. zhejiang: Zhejiang Sci-Tech University, 2022.(in Chinese)

[10] 石硕,魏振辉,刘晓菲等.一种基于 LSTM 的 Kubernetes 容器云弹性伸缩策略研究[J].制造业自动化, 2023, 45(09):189-196.
SHI S,WEI Z H, LIU X F,et al.Research on an LSTM-based kubernetes container cloud elastic scaling strategy[J]. Manufacturing Automation, 2023, 45(09):189-196. (in Chinese)

[11] 陈烨. 面向 Web 应用的 Kubernetes 容器弹性伸缩方法的研究[D]. 上海：华东师范大学,2023.
CHEN Y. Research on Flexible Scaling Method of Kubernetes Container for Web Application[D]. Shanghai:East China Normal University, 2023. (in Chinese)

[12] Balla D, Simon C, Maliosz M. Adaptive scaling of Kubernetes pods[C]//NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2020: 1-5.

[13] Farahnakian F, Liljeberg P, Plosila J. LiRCUP: Linear Regression Based CPU Usage Prediction Algorithm for Live Migration of Virtual Machines in Data Centers[C]//Euromicro Conference on Software Engineering & Advanced Applications. IEEE Computer Society, 2013: 23.

[14] 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.

[15] D. Janardhanan and E. Barrett. CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models [C]//2017 12th International Conference for Internet Technology and Secured Transactions (ICITST). Cambridge, UK: 2017, 55-60.

[16] Wu H, Hu T, Liu Y, et al. Timesnet: Temporal 2d-variation modeling for general time series analysis[J]. arXiv, 2023, arXiv preprint arXiv:2210.02186.

[17] Lara-Ben íez P, Gallego-Ledesma L, Carranza-Garc á M, et al. Evaluation of the Transformer Architecture for Univariate Time Series Forecasting[C]//Advances in Artificial Intelligence, M álaga: Springer International Publishing, 2021: 106-115.

[18] 刘杭,殷歆,陈杰等.基于混合网络模型的多维时间序列预测[J].计算机工程,2023,49(01):121-129.
LIU H, YIN X, CHEN J,et al. Multi-Dimensional Time-Series Prediction Based on Hybrid Network Models[J].Computer Engineering,2023,49(01):121-129.

(in Chinese)

[19] Wu H, Xu J, Wang J, et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting[J]. Advances in Neural Information Processing Systems, 2021, 34: 22419-22430.

[20] Kubernetes Introduction. Kubernetes official website [EB / OL].[2023-07-15]. https://kubernetes.io/zh-cn/docs/concepts.

[21] Bishop C M. Training with noise is equivalent to Tikhonov regularization[J]. Neural computation, 1995, 7(1): 108-116.

[22] Woo S, Park J, Lee J Y, et al. Cbam: Convolutional block attention module[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 3-19.

[23] HENDRYCKS D, GIMPEL K. Gaussian error linear units (GELUs) [EB/OL]. [2023-7 -22]. https://arxiv.org/abs/1606.08415.

[24] SZEGEDY C, LIU W, JIA Y Q ,et al. Going deeper with convolutions[C]//Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2015: 1-9.