Software Engineering and Applications 软件工程与应用, 2023, 12(6), 793-809
Published Online December 2023 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2023.126077

Hans汉斯

融合三支决策的预测式容器伸缩优化策略

吝兴银，杨　朝，钟崇楷

哈尔滨师范大学计算机科学与信息工程学院，黑龙江　哈尔滨

收稿日期：2023年10月26日；录用日期：2023年12月1日；发布日期：2023年12月13日

摘　要

当前Kubernetes中基于固定阈值的响应式伸缩策略存在无法根据集群负载动态调整扩缩容力度以及存在的时间滞后性等突出性问题。针对该问题，利用负载预测模型在检测实时工作负载的同时对未来的工作负载进行预测，同时，根据历史负载变化情况并结合三支决策对负载波动进行三分，即低负载波动期、正常负载波动期、高负载波动期，针对每一种负载波动期，对伸缩力度进行细粒化，最后基于负载波动期和预测的CPU利用率执行相应的容器伸缩决策。通过对比Kubernetes原生算法和同类算法，所提出的伸缩优化策略能够有效地应对负载波动，降低SLA违约率，在保证QoS的同时降低了资源的浪费。

关键词

负载预测，容器，三支决策，伸缩策略

Predictive Container Scaling Optimization Strategy Integrating Three-Way Decisions

Xingyin Lin, Zhao Yang, Chongkai Zhong

College of Computer Science and Information Engineering, Harbin Normal University, Harbin Heilongjiang

Received: Oct. 26th, 2023; accepted: Dec. 1st, 2023; published: Dec. 13th, 2023

Abstract

The current responsive scaling strategy based on fixed thresholds in Kubernetes has outstanding problems such as the inability to dynamically adjust the expansion and contraction intensity according to the cluster load and the existence of time lag. To address this problem, the load prediction model is used to predict future workloads while detecting real-time workloads. At the same time, load fluctuations are divided into three categories based on historical load changes and three decisions, namely, low load fluctuation period and normal load. During the fluctuation period and high load fluctuation period, the scaling intensity is fine-grained for each load fluctuation

文章引用：吝兴银，杨朝，钟崇楷．融合三支决策的预测式容器伸缩优化策略[J]．软件工程与应用，2023，12(6)：793-809. DOI: 10.12677/sea.2023.126077

period, and finally the corresponding container scaling decision is executed based on the load fluctuation period and the predicted CPU utilization. By comparing the Kubernetes native algorithm and similar algorithms, the proposed scaling optimization strategy can effectively cope with load fluctuations, reduce SLA default rates, and reduce resource waste while ensuring QoS.

Keywords

Load Forecasting, Container, Three-Way Decisions, Scaling Strategy

1. 引言

云计算中最关键、最核心的技术是虚拟化技术[1]。起初，云计算主要使用虚拟机技术来隔离不同资源以达到应用独立的目的。然而，虚拟机技术过多的抽象给用户带来了不便[2]。2013 年，随着 Docker 容器技术的快速发展，基于容器的虚拟化技术迅速成为各大云计算厂商和云计算开发者的首选。对比虚拟机，容器技术具有镜像小、资源消耗少、应用部署灵活和启动速度快等优点[3]，但在实际的云环境中需要管理的容器数目难以想象，并且容器与容器之间的关系也可能及其复杂。所以在云环境实际应用中，一般使用容器的编排工具对容器实施相应的编排部署。目前，最流行的容器的编排部署工具是 Kubernetes。

Kubernetes 是谷歌内部使用的集群管理系统 Borg 的开源版本，是一个基于容器技术的分布式架构和集群管理系统[4]。而容器的弹性伸缩是 Kubernetes 中的一种功能特性。在 Kubernetes 容器云平台上，所采用的默认的弹性伸缩策略是基于固定阈值的响应式伸缩策略。这种伸缩策略通常会提前设定好若干组固定阈值，每组阈值对应一种性能指标的上下限，常见的性能指标有 CPU、RAM、带宽等。在实际运行过程中，Kubernetes 的监控系统会实时监测当前工作负载下的性能指标数据，基于系统所设定的固定阈值和实际的性能指标数据，系统的伸缩控制器会做出对应的弹性伸缩行为，最终使得应用负载的性能指标处于上下限阈值区间内。然而，这种基于固定阈值的响应式伸缩策略的缺点也非常明显，首先，这种伸缩策略是先监测实时工作负载，然后基于实时工作负载和固定阈值的关系来进行伸缩，系统的扩缩容调整往往会晚于实际的工作负载变化，导致扩缩容不及时，难以及时响应服务资源请求，进而造成服务质量变差，用户体验糟糕。其次，这种伸缩策略在伸缩时的扩缩容力度固定不变，在高负载或低负载阶段就会反复进行扩缩容，降低了扩缩容的效率，甚至可能导致集群崩溃等严重问题。

目前，在学术研究方面，针对容器云环境的伸缩策略也有很多的改进改良研究。耿伟等人提出了一种基于排队理论的容器云弹性伸缩策略 ESBQT 模型，该模型将容器云中心看成一个 M/M/s 等待制排队模型，基于排队理论有效解决了大流量冲击导致的稳定性问题[5]，但该伸缩策略无法对未来时刻负载进行预测，仍然存在扩缩容不及时带来的服务质量问题。孙婧等人提出了基于 Docker 容器自动伸缩技术，解决了应用程序在有更多访问流量时，快速增加服务器数量，而在流量下降时，降低服务器数量减少成本[6]，该伸缩策略仍然无法对未来时刻负载进行预测。徐胜超等人提出了基于深度学习的容器云弹性伸缩方法，通过全面衡量负载特性并对应用类型进行分析，利用深度学习方法中 LSTM 神经网络预测模型完善预测功能，使应用负载预测更准确，同时可根据负载预测值和实际值双标准控制伸缩动作[7]，但该伸缩策略没有考虑在进行伸缩决策时的扩缩容力度问题，仍然存在容器伸缩时扩缩容不足或者扩缩容过大等问题，进而影响了容器的扩缩容效率。马小淋提出了一种基于负载特征预测的容器云弹性伸缩策略，

该策略通过综合多个指标衡量复合型应用的负载，再通过预测值和负载值共同进行伸缩决策[8]，该伸缩策略也没有考虑在进行伸缩决策时的扩缩容力度问题，容易在高负载或低负载阶段出现反复伸缩行为，影响伸缩效率。

针对文献[5] [6]所提出的伸缩策略不能对未来时刻负载进行有效预测和文献[7] [8]中所提出的伸缩策略没有考虑在进行伸缩决策时的扩缩容力度问题，本文提出了一种融合三支决策的预测式容器伸缩优化策略，该策略不仅依托于三支决策将扩缩容力度细粒化，避免高负载或低负载阶段时反复进行伸缩，提高容器的伸缩效率，而且结合负载预测模型可以更好的对未来时刻的工作负载进行预测，基于预测负载值和设定好的固定阈值应用响应式伸缩策略快速做出伸缩动作，从而降低服务请求的响应时间，提高服务的质量，促进用户体验。

2. 三支决策

姚一豫教授提出了三支决策的概念，其主要思想是以三为本、基三思维、用三而治，是基于“三”的思维方式、基于“三”的问题求解方法和基于“三”的信息处理模式[9] [10] [11] [12] [13]。三支决策是结合决策情景提出的一种“三分而治”和“化繁为简”的决策理论，用于处理不确定性决策问题[14]。三支决策的主要思想是将一个论域划分为三个不相交的区域，并设计不同的策略对每一部分进行有效处理[15]。

三支决策的判别方式与以往的二支决策有所不同，对于二支决策而言，通常只有是或者不是两种选择方式，三支决策则是通过一个映射将一个整体 OB 分成三个两两互不相交的部分，即正域 POS、边界域 BND 和负域 ENG，记为 $\pi=\{P_1,P_2,P_3\}$。

$$f:U\to\pi$$

其中：f可以看作是一个映射或函数，即一个对象通过 f 映射到三个区域，且 P1，P2，P3$\subseteq$OB，OB = P1$\cup$P2$\cup$P3，P1$\cap$P2 = $\varnothing$，P2$\cap$P3 = $\varnothing$，P1$\cap$P3 = $\varnothing$。

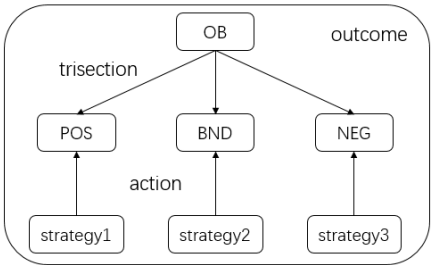

Figure 1. TAO model of three-way decision-making
图 1. 三支决策的 TAO 模型

三支决策是以“三分而治”为主要思想的一种决策方式，即将一个整体合理地分为三个部分，并对每一个部分施加对应的策略，从而获得整体所需要的效果[16]。进一步，姚一豫教授考虑了“效”的问题，提出了三支决策的 TAO(trisecting-acting-outcome)模型[11]，其模型如图 1 所示。TAO 模型在基于三支决策的“分”和“治”的思想上加上了“效”[17]，其基本思想为[18]：(1) 将整体细粒度划分；(2) 设计处理对应部分的策略；(3) 获取理想的效果。

目前，云计算环境中三支决策的应用和研究都已趋于成熟，然而，云计算中的三支决策主要以云任务和虚拟机调度策略为研究侧重点，这是因为云任务和虚拟机都会受到很多复杂变量的影响，并且也具有很多可以细粒化的研究属性[19] [20] [21] [22]。从各种学术研究来看，三支决策在云任务和虚拟机调度

策略的研究领域表现出了很大的优势。其实，云容器的伸缩问题也需要细粒度的划分，三支决策也能够应用于容器伸缩容的研究领域，且具备良好的优化效果。

3. 伸缩策略优化

3.1. 伸缩技术

目前，容器的伸缩技术主要包括响应式伸缩策略[23]和预测式伸缩策略[24]两种，关于两种容器伸缩技术的解释和伸缩过程如下。

3.1.1. 响应式伸缩技术

响应式伸缩策略以所设定固定阈值为基础，在策略被应用于相应算法并处于运行状态时，系统会实时监测相关的云资源(CPU、RAM 等)使用情况，当监控到某些云资源指标超出了所设定的阈值范围，系统的监控模块就会发出相应的伸缩命令。其伸缩过程具体来说，如图 2 所示，在 t1 时刻，系统会对相关的云资源进行实时监测，当在 t2 时刻，系统监测到相关云资源超出所设定的阈值范围时，系统开始执行伸缩，系统在 t4 时刻完成了容器伸缩。由于基于固定阈值的响应式伸缩策略无法对未来时刻的工作负载进行准确预测，只能通过对系统中的云资源的使用状况进行实时监测，系统每次都是先监测云资源，后执行容器伸缩，故而系统的容器伸缩行为往往会晚于云环境的工作负载变化，进而造成扩缩容不及时，难以及时响应工作负载的变化，最终导致应用的服务质量变差，用户体验糟糕，但是由于基于固定阈值的响应式伸缩策略对云环境集群的突发式工作负载具有良好的伸缩效果，并且其算法实现较为简单，所以该策略被广泛应用于各种云平台伸缩决策当中[25]。

3.1.2. 预测式伸缩策略

预测式伸缩策略通对历史时间段内系统中云资源的利用情况进行监测，然后结合相关的算法和数学模型对云环境未来时刻的负载情况进行预测[25]。该策略与基于固定阈值的响应式伸缩策略不同，不会出现云环境中容器扩缩容不及时、难以及时响应应用的服务请求等问题。其伸缩策略的具体执行过程，如图 3 所示，当在 t1 时刻时，系统通过对过去云资源的利用情况进行实时监测并根据指定指标的历史数据的特点结合相关的预测模型预测到在 t5 时刻时，系统中的云资源的使用率会超过所设置的阈值范围，因而系统会在 t2 时刻请求容器伸缩，此时系统有足够的时间完成容器的伸缩操作，进而避免了容器扩缩容不及时带来的应用服务质量变差、用户体验不佳等突出性问题[26]。

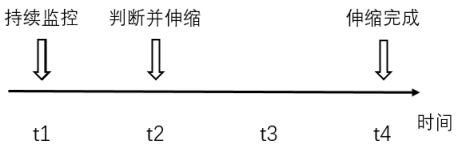

Figure 2. Responsive scaling strategy
图 2. 响应式伸缩策略

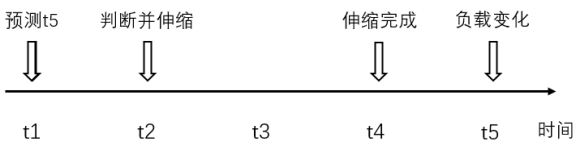

Figure 3. Predictive scaling strategy
图 3. 预测式伸缩策略

通过对基于固定阈值的响应式伸缩策略和预测式伸缩策略各自执行过程和优劣势的分析，本文实验

将两种伸缩策略巧妙结合，在对未来时刻工作负载预测的同时，也利用到响应式伸缩策略的响应效果，因而避免了单一响应式伸缩策略带来的容器扩缩容不及时等突出问题，提高了系统的响应速度，用户也因此具有更好的体验。

3.2. 负载预测模型

目前，机器学习方法和时间序列分析方法是两种最常用的用于预测未来时刻工作负载的方法，但由于机器学习方法需要长时间的模型训练，对于短期的预测并不适用。考虑到各方面因素，本文实验采用时间序列分析方法中的指数平滑法。

指数平滑法作为一种常用的时间序列分析方法，它是由移动平均法发展而来的，指数平滑法在继承移动平均法优势的同时，又考虑到远近期的时间序列数据对未来时刻工作负载的不同影响，因而指数平滑法也是一种特殊的加权平均法。指数平滑可继续拆分为一次平滑，二次平滑和三次平滑(即 Holt-Winters 法)，一次平滑法为历史数据的加权预测，二次平滑法适用于具有一定线性趋势的数据，三次平滑法在二次平滑法基础上再平滑一次，其适用于具有一定曲线趋势关系时使用[27]，因此对于本文实验的短期预测，选用三次指数平滑法对未来时刻的工作负载值进行预测。

指数平滑预测对不同时期的观察值确定不同的权重，最近的数据选择较大的权重，以前的数据选择较小的权重，加大最近观察数值对实际需求反映的变化，从而得到下一期预测值[28]。指数平滑预测法由一次到二次再到三次，三次指数平滑法就是在二次指数平滑值的基础上进行第三次指数平滑，从而计算预测系数，建立预测模型。三次指数平滑法的建模过程[29]如下所示：

设初始序列为 y_t， y_{t-1}， $y_{t-2}\cdots$，则 t 时刻的一次指数平滑值为：

$$S_t^{(1)} = \alpha y_t + \alpha(1-\alpha) y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \tag{1}$$

其中，$S_t^{(1)}$ 表示第 t 期的一次指数平滑值，α 为平滑系数(也称为权重系数)，且 $0<\alpha<1$。由式(1)得原始值 y_t， y_{t-1}， y_{t-2} 的权重分别为 α， $\alpha(1-\alpha)$， $\alpha(1-\alpha)^2$。

变换式(1)得：

$$S_t^{(1)} = \alpha y_t + (1-\alpha) S_{t-1}^{(1)} = S_{t-1}^{(1)} + \alpha\left(y_t - S_{t-1}^{(1)}\right) \tag{2}$$

则预测公式为：

$$S_{t+1}^{(1)} = \alpha y_{t-1} + (1-\alpha) S_t^{(1)} \tag{3}$$

二次指数平滑值为：

$$S_t^{(2)} = \alpha S_t^{(1)} + (1-\alpha) S_{t-1}^{(2)} \tag{4}$$

预测模型为：

$$F_{t+T} = a_t + b_t T \tag{5}$$

其中，$S_t^{(2)}$ 为二次指数平滑值，F_{t+T} 为 T 期的预测值，T 为预测超前期数，$T=1$，2，3……，n。a_t， b_t 为第 T 期的预测系数。

$$a_t = 2S_t^{(1)} - S_t^{(2)} \tag{6}$$

$$b_t = \alpha \Big/ (1-\alpha)\left(S_t^{(1)} - S_t^{(2)}\right) \tag{7}$$

进一步采用三次指数平滑法进行预测，三次指数平滑值为：

$$S_t^{(3)} = \alpha S_t^{(2)} + (1-\alpha) S_{t-1}^{(3)} \tag{8}$$

预测模型为：

$$F_{t+T} = a_t + b_t T + c_t T^2 \tag{9}$$

预测系数为：

$$a_t = 3S_t^{(1)} - 3S_t^{(2)} + S_t^{(3)} \tag{10}$$

$$b_t = \alpha \Big/ 2(1-\alpha)^2 \left((6-5\alpha) S_t^{(1)} - 2(5-4\alpha) S_t^{(2)} + (4-3\alpha) S_t^{(3)} \right) \tag{11}$$

$$c_t = \alpha \Big/ 2(1-\alpha)^2 \left(S_t^{(1)} - 2S_t^{(2)} + S_t^{(3)} \right) \tag{12}$$

其中，$S_t^{(3)}$ 为三次指数平滑值。

利用预测模型进行预测之前，需要对指数平滑系数进行确定。α 值越大，最近的时间序列数据值所占的比重就越大，预测结果更偏向于近期的数据值变化。当然，如果过大，可能会造成预测结果对近期数据值过于“依赖”，反而忽略了远期的历史数据值变化，从而导致预测结果不是时间序列值的根本性变化。α 值越小，则远期的历史时间序列数据值所占的比重就越大，预测结果就缺乏近期数据的影响，可能造成对未来时刻的预测不准确。关于指数平滑系数 α 的选择，具有一定的主观性，所以需要根据自己的需求对 α 进行选择。在进行短期预测时，如果希望尽快反映观测值的变化，可以选取较高的 α 值，可取 α = 0.6~0.8，如果是希望消除季节波动对时间序列的影响，反应时间序列的长期趋势规律，可以选择较小的值，可取 α = 0.1~0.3，如果在历史数据很少的情况下进行预测，推荐选择较高的值进行短期预测[30]。

3.3. 伸缩算法设计

3.3.1. 伸缩决策过程

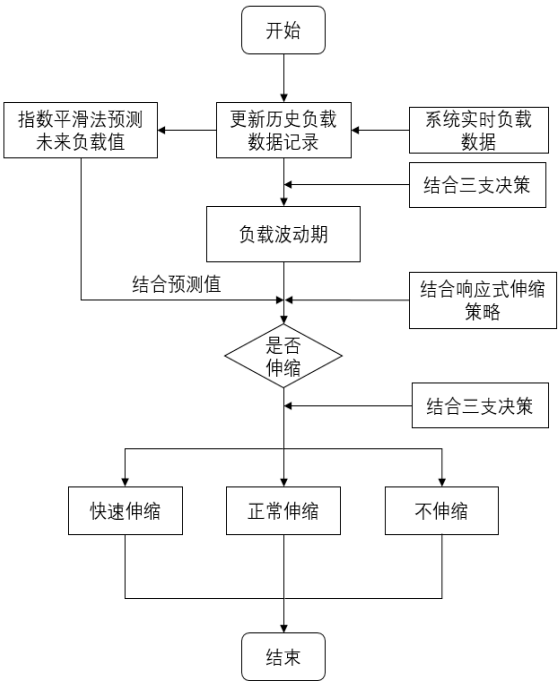

Figure 4. Container scaling decision-making process
图 4. 容器伸缩决策过程

在算法程序运行期间，系统会根据实时负载数据值对历史负载数据记录进行更新，然后依据历史负载数据结合指数平滑法对未来时刻工作负载值进行预测，依托系统预测值和阈值基于响应式伸缩策略来判定是否进行伸缩，最后依据三支决策划分执行对应扩缩容力度的容器伸缩。本文提出的融合三支决策的预测式容器伸缩优化策略的决策过程，如图 4 所示。

3.3.2. 伸缩决策过程

自动伸缩阶段包括扩容阶段和缩容阶段，由于容器的缩容阶段不存在响应延迟的问题，并且经过实验的测试，预测式的容器缩容策略可能会频繁地触发伸缩策略，进而造成不必要的颠簸，因此本文的容器缩容阶段采用响应式的伸缩策略，扩容阶段采用融合三支决策的预测式伸缩策略。系统实时收集 Pod 历史 CPU 利用率，并且每隔一段时间对未来时刻的工作负载进行预测，根据最近的历史数据判定当前的负载波动期(低负载波动期，正常负载波动期，高负载波动期)，然后结合预测的工作负载，对每一种伸缩情况设置不同的伸缩力度。

考虑到负载预测模型存在一定的误差，因此本文结合响应式伸缩策略的同时通过参数 δ 设置合理阈值缓冲区来弥补预测式伸缩策略在扩容时的不足。

基于上述阐述，现在给出具体的算法实现，具体算法流程如算法 1 和算法 2 所示。

算法 1 扩容算法

Input：云任务负载 Tn = {t1, t2, …, tn}

CPU 利用率下限阈值 α 和利用率上限阈值 β

负载波动下限阈值 M 和负载波动上限阈值 N

伸缩因子 Fi(i = 1, 2, 3…)和阈值缓冲参数 δ

Output：需要伸缩的资源数量 scalingAmount

1. Begin
2. for(every moment)//实时监控
3. collect(getCpuPercentUtilization());//收集 CPU 历史利用率数据
4. collect(getLoadOfHistory());//收集历史负载数据
5. if(time ∈ scaling)//当前需要进行伸缩判定
6. R ← getAvgRateOfGrowth();//计算平均增长率
7. loadPrediction(the next moment);//对下一时刻负载进行预测
8. P ← calculatePredictUtilization();//计算预测的 CPU 利用率
9. if(R ∈ Low load fluctuation period)//R < M 时，当前处于低负载波动期
10. if(N < α && N + δ < α)
11. no scaling;//防抖动
12. if(N + δ ≥ α && N + δ ≤ β) //allocatedResource 为已分配的资源数量
13. scalingAmount ← F1 * allocatedResource; //正常扩容
14. if(N > β || N + δ > β)
15. scalingAmount ← F2 * allocatedResource;//快速扩容
16. if(M ∈ Normal load fluctuation period)//M ≤ R ≤ N 时，当前处于正常负载波动期
17. if(N < α && N + δ < α)
18. no scaling;//防抖动
19. if(N + δ ≥ α && N + δ ≤ β)

20. scalingAmount ← F3 * allocatedResource;//正常扩容
21. if(N > β || N + δ > β)
22. scalingAmount ← F4 * allocatedResource;//快速扩容
23. if(M ∈ High load fluctuation period)//R > N 时，当前出去高负载波动期
24. if(N < α && N + δ < α)
25. no scaling;//防抖动
26. if(N + δ ≥ α && N + δ ≤ β)
27. scalingAmount ← F5 * allocatedResource;//正常扩容
28. if(N > β || N + δ > β)
29. scalingAmount ← F6 * allocatedResource;//快速扩容
30. end if
31. End

算法 2 缩容算法

Input：云任务负载 Tn={t1,t2,…,tn}
CPU 利用率下限阈值 α
伸缩因子 F

Output：需要伸缩的资源数量 scalingAmount

1. Begin
2. for (every moment)//实时监控
3. P ← get Cpu Percent Utilization ();//获取当前 CPU 利用率
4. if (P < α) //allocatedResource 为已分配的资源数量
5. scalingAmount ← F * allocatedResource;
6. end if
7. end for
8. End

为了帮助读者更好地理解本文算法，下面结合相关数据给出一个具体的算法实例。

在本算法实例当中，通过在指定范围内随机生成云任务负载的方式模拟三种负载波动期(低负载波动期、正常负载波动期、高负载波动期)。为了方便实时的监控集群的负载变化，每 1 秒收集一次 Pod 负载数据和 CPU 利用率数据并对其进行持久化。vCPUs 初始值为 4，每秒的 CPU 利用率计算公式如下：

$$CPUusage = \frac{CloudletNum * CloudletPE}{vCPUs} \tag{13}$$

其中 *CloudletNum* 是指此刻正在执行的云任务数量， *CloudletPE* 是指云任务的 pe 数。

伸缩后的 CPU 核数计算公式如下：

$$\mathrm{vCPUs} = \mathrm{CPUs} \pm \mathrm{ceil}\left(\mathrm{scalingAmount}\right) \tag{14}$$

其中 vCPUs 指伸缩后的 CPU 核数， CPUs 指当前 CPU 核数， scalingAmount 指伸缩资源的数量。

伸缩资源数量的计算公式如下：

$$\mathrm{scalingAmount} = \mathrm{F} * \mathit{a}\mathrm{llocatedResource} * \mathrm{n} \tag{15}$$

其中 F 指伸缩因子， *a*llocatedResource 指已分配的 CPU 核数。

算法实例中使用到的参数及含义描述如表 1 所示。

Table 1. Each parameter and its meaning
表 1. 各参数及其含义

序号	参数	描述
1	M	负载波动下限阈值
2	N	负载波动上限阈值
3	δ	用于设置阈值缓冲区的参数
4	α	CPU 利用率下限阈值
5	β	CPU 利用率上限阈值
6	E	全部历史负载数据集合
7	F	最近历史负载数据集合
8	U	预测的 CPU 利用率
9	R	负载平均增长率
10	LP	预测的负载值
11	scalingAmount	需要伸缩的资源数量
12	scalingFactor	伸缩因子

在本实例中，20 个云任务负载会在第 1 秒同时到达(初始化集群)，之后每秒的真实负载数据采用的是通过在指定范围内随机生成云任务负载的方式随机模拟三种负载波动期(低负载波动期、正常负载波动期、高负载波动期)，并且持续有负载数据。为了方便观察实验的结果并收集相关数据信息，设置每 1 秒收集一次 Pod 的负载数据，预测 CPU 负载需要 0.5 秒时间，对 Pod 进行伸缩判定并修改 Pod 的资源限额然后重启该 Pod 需要 0.5 秒时间。同时，为尽可能地避免频繁伸缩，系统将根据近期的负载波动状况动态地调整 Pod 的伸缩周期，设置低负载波动期时 Pod 的伸缩周期为 3 秒，正常负载波动期时 Pod 的伸缩周期为 2 秒，高负载波动期时 Pod 的伸缩周期为 1 秒。扩容时三种负载波动期的 CPU 利用率下限阈值 α 均设置为 0.55，扩容时三种负载波动期的 CPU 利用率上限阈值 β 均设置为 0.7，负载波动下限阈值 M 设置为 3，负载波动上限阈值 N 设置为 5，伸缩因子 scalingFactor 取 0.1，δ 取 0.03。(其中 α 和 β、M 和 N 以及 scalingFactor 均可根据实际情况进行调整)。

(1) 实验开始时，系统程序开始收集每秒的历史负载数据，从第 5 秒开始，系统会首先对当前时刻的负载数据进行收集，得到 1~5 秒的历史负载数据集合 E = [20, 34, 39, 50, 56]。利用历史负载记录结合预测算法对第 6 秒的负载进行预测得到负载预测值 LP = 62，然后根据公式(13)计算出相应预测的 CPU 利用率 U = 88.57%。通过查询历史负载数据集合 E 得到含有最近五次历史负载数据的集合 F = [20, 34, 39, 50, 56] (因此时集合 E 只有 5 秒的历史数据，所以两个集合中的负载数据是一样的)，结合 F 中的数据计算得出 R = 9，然后开始进行第 1 次伸缩判定，此时 R 满足 R > N，系统判定当前为高负载波动期，此时修改系统的伸缩周期为 1 秒。而 U 满足 U > β，系统判定此时需要进行快速扩容，根据公式(15)计算得出 scalingAmount = 17，然后系统根据伸缩的资源数量执行相应的伸缩行为。

(2) 在第 6 秒时，开始进行第 2 次伸缩的相关计算和判定，同步骤(1)的方法相同，此时 E = [20, 34, 39, 50, 56, 72]，通过预测得到负载预测值 LP = 91，然后计算得出 U = 104.6%，查询集合 E 得到 F = [34, 39, 50, 56, 72]，计算得出 R = 9.5，此时 R 满足 R > N，系统判定当前为高负载波动期，此时系统的伸缩周期保持 1 秒不变。而 U 满足 U > β，系统判定此时需要进行快速扩容，通过计算得出 scalingAmount = 22，然后系统根据伸缩的资源数量执行相应的伸缩行为。

(3) 在第 10 秒时，开始进行第 6 次伸缩的相关计算和判定，此时 E = [20, 34, …, 56, 72, 83, 100, 112,

119]，通过预测得到负载预测值 LP = 123，计算得出 U = 62.12%，查询历史利用率集合得到 L = [72, 83, 100, 112, 119]，计算得出 R = 11.75，此时 R 满足 R > N，系统判定当前为高负载波动期，此时系统的伸缩周期保持 1 秒不变。而 U 满足 $\alpha \le U + \delta \le \beta$，系统判定此时需要进行正常扩容，通过计算得出 scalingAmount = 24，然后系统根据伸缩的资源数量执行相应的伸缩行为。

(4) 在第 15 秒时，继续进行伸缩的相关计算和判定，此时 E = [20, 34, 39, …, 119, 122, 139, 145, 167, 172]，通过预测得到负载预测值 LP = 175，计算得出 U = 51.78%，查询 E 得到 F = [122, 139, 145, 167, 172]，计算得出 R = 12.5，此时 R 满足 R > N，并且 U 满足 $U < \alpha$，系统判定当前为高负载波动期但为了防止抖动，无需扩容，同时系统的伸缩周期仍然保持 1 秒不变。

(5) 在第 25 秒时，继续进行伸缩的相关计算和判定，此时 E = [20, 34, 39, …, 233, 237, 241, 245, 250]，通过预测得到负载预测值 LP = 255，计算得出 U = 50.3%，查询集合 E 得到 F = [233, 237, 241, 245, 250]，计算得出 R = 4.25，此时 R 满足 $M \le R \le N$，并且 U 满足 $U < \alpha$，系统判定当前为正常负载波动期但为了防止抖动，无需扩容，此时修改系统的伸缩周期为 2 秒。

(6) 在第 27 秒时，进行伸缩的相关计算和判定，此时 E = [20, 34, …, 237, 241, 245, 250, 254, 258]，通过预测得到负载预测值 LP = 262，计算得出 U = 51.68%，查询集合 E 得到 F = [241, 245, 250, 254, 258]，计算得出 R = 4.25，此时 R 满足 $M \le R \le N$，系统判定当前为正常负载波动期但为了防止抖动，无需扩容，同时系统的伸缩周期仍然保持 2 秒不变。

(7) 在第 72 秒时，继续进行伸缩的相关计算和判定，此时 E = [20, 34, …, 391, 393, 395, 397, 399, 401]，通过预测得到负载预测值 LP = 403，计算得出 U = 52.75%，查询集合 E 得到 F = [393, 395, 397, 399, 401]，计算得出 R = 2，此时 R 满足 R < M，系统判定当前为低负载波动期，同时修改系统的伸缩周期为 3 秒。而 U 满足 $\alpha \le U + \delta \le \beta$，系统判定此时需要进行正常扩容，通过计算得出 scalingAmount = 41，然后系统根据伸缩的资源数量执行相应的伸缩行为。

(8) 在第 75 秒时，继续进行伸缩的相关计算和判定，此时 E = [20, 34, 39, …, 397, 399, 401, 402, 404, 406]，通过预测得到负载预测值 LP = 408，计算得出 U = 50.68%，查询集合 E 得到 F = [399, 401, 402, 404, 406]，计算得出 R = 1.75，此时 R 满足 R < M，并且 U 满足 $U < \alpha$，系统判定当前为低负载波动期但为了防止抖动，无需扩容。同时系统的伸缩周期仍然保持 3 秒不变。

4. 实验验证及结果分析

本文通过 CloudSimPlus 云平台进行一系列的模拟仿真实验，将本文中提出的算法与 Kubernetes 原生算法和同类算法进行对比。

4.1. 实验环境

为了验证所提出的一种融合三支决策的预测式容器伸缩优化策略的有效性，又由于 CloudSimPlus 模拟器具有很好的可扩展性和使用便捷性，所以本文实验利用 IDEA 基于 CloudSimPlus 模拟云平台，在云平台上创建云数据中心，构造物理主机、虚拟机和云任务等云平台的各种资源。实验环境采用 IDEA2022.2.4，JDK17，CloudSimPlus8.1.0，并利用 Matlab2022 进行实验数据分析。

4.2. 实验数据

为了充分验证本文提出的一种融合三支决策的预测式容器伸缩优化策略，本次实验的负载数据采用随机生成的方式，即每秒的云任务数量在指定范围内随机产生，选取大约 1500 个左右的云任务快速访问集群。

下面详细阐述实验资源参数设置及执行过程。物理主机(Host)和虚拟机(VM)的配置参数如表 2 所示。

Table 2. Physical host (Host) and virtual machine (VM) configuration
表 2. 物理主机(Host)和虚拟机(VM)配置

配置	物理主机(Host)	虚拟机(VM)
内存(ram)	20000 MB	1200 MB
带宽(bw)	100000 MB	1000 MB
存储(storage)	10000000 MB	10000 MB
CPU 主频(mips)	1000 MIPS	1000 MIPS
内核数(pes)	40	4

在本次模拟实验中，设置一个云任务的初始长度为 1000，云任务的总长度也是以在指定范围内随机生成的方式产生。为了初始化集群，设置前 60 个云任务的延迟时间为 0，即前 60 个云任务同时被执行。后面每秒到达的云任务负载均在指定范围内随机生成。每个云任务的内核数均设置为 1。为了实时的监控集群的负载变化以及对伸缩做出快速反应，在实验中将默认轮询时间改为 1 秒。

为了充分验证本文所提出的算法的优越性，将本文算法与 Kubernetes 原生算法 VPA 进行对比，在此基础上还与同类算法 TVPA 进行比较。

4.3. 性能评价指标

4.3.1. SLA 违约率(Service Level Agreement, SLA)

SLA 规定和衡量了云服务提供商对客户保证的服务质量水平。由于在实际的 Kubernetes 集群环境中，负载波动存在高度的动态性，即使当前可能处于低负载波动期，但云计算平台的资源需求可能会在短时间内快速增长，从而导致容易出现预测值小于实际值的情况，最终影响服务的质量。在本文中，将 SLA 违约率定义为当前时刻 CPU 利用率大于 CPU 利用率上限阈值 UpperThreshold 的样本数量占总整体 CPU 利用率样本总数的比例，其计算公式如下。

$$SLA = count\left(U_i > U_{UpperThreshold}\right)/n \tag{16}$$

其中，U_i 表示第 i 个样本点的 CPU 实际利用率，$U_{UpperThreshold}$ 表示 CPU 利用率的上限阈值，n 表示样本总数。

4.3.2. 平均 CPU 利用率(Avg CPU Util)

在本文中，将平均 CPU 利用率定义为所有实例的 CPU 利用率的平均值，其计算公式如下。

$$AvgCpuUtil = \frac{1}{T}\frac{1}{N}\sum_{t=0}^{T}\sum_{n=1}^{N} Util_{tn} \tag{17}$$

其中，N 表示实例数，$Util_{tn}$ 表示第 n 个实例在时间 t 时的 CPU 利用率。

4.3.3. 均方根误差(Root Mean Squared Error, RMSE)

均方根误差(Root Mean Squared Error, RMSE)是一种用于衡量预测模型在连续性数据上的预测精度的指标。它衡量了预测值与真实值之间的均方根差异，表示预测值与真实值之间的平均偏差程度，其计算公式如下。

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2} \tag{18}$$

其中，y_i 和 $f(x_i)$分别表示第 i 个样本的真实值和预测值，n 为样本个数。

4.4. 实验结果及分析

4.4.1. 工作负载预测效果对比

机器学习方法和时间序列分析方法是两种最常用的用于预测未来时刻工作负载的方法，但由于机器学习方法依赖于在目标指标上经过充分训练的模型，且训练周期较长，对于短期的预测并不适用。所以本文实验采用时间序列分析方法中的指数平滑法对未来时刻的工作负载进行预测。

使用二次指数平滑和三次指数平滑分别进行负载预测，负载数据序列采用在指定范围内随机生成的方式。图 5 展示的是基于二次指数平滑的负载预测模型预测的负载值和真实值的对比情况。

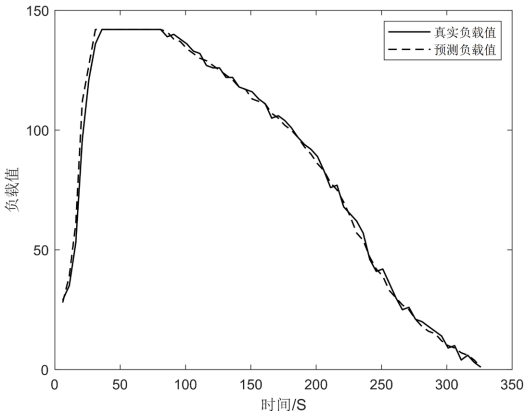

Figure 5. Comparison of load forecasting results based on cubic exponential smoothing
图 5. 基于三次指数平滑的负载预测结果对比

通过多次实验测试，发现二者的 RMSE 基本维持在 2.7~2.9 之间，经过对比分析，可以看出二者均保持了较好的预测精度，但是三次指数平滑适用于具有一定曲线趋势的关系，而这种关系更符合实际的工作负载情况，所以本文依然采用三次指数平滑的方法对未来时刻负载进行预测，使预测模型的预测方法能够很好地预测未来时刻的工作负载。

4.4.2. 伸缩过程和资源扩容量对比

为了验证本文提出的融合三支决策的预测式伸缩策略的有效性和优越性，对比 Kubernetes 原生算法和同类其他算法(无预测)的伸缩过程及其资源扩容情况。

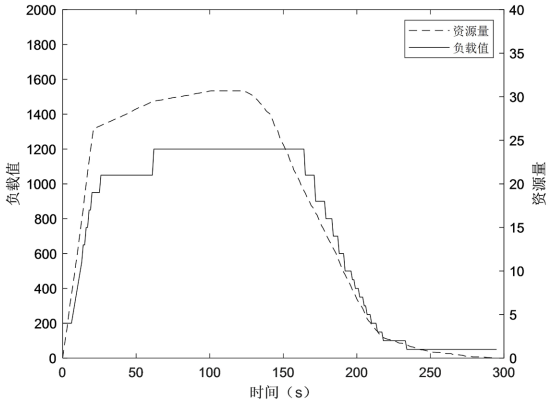

Figure 6. The scaling process of native algorithms
图 6. 原生算法的伸缩过程

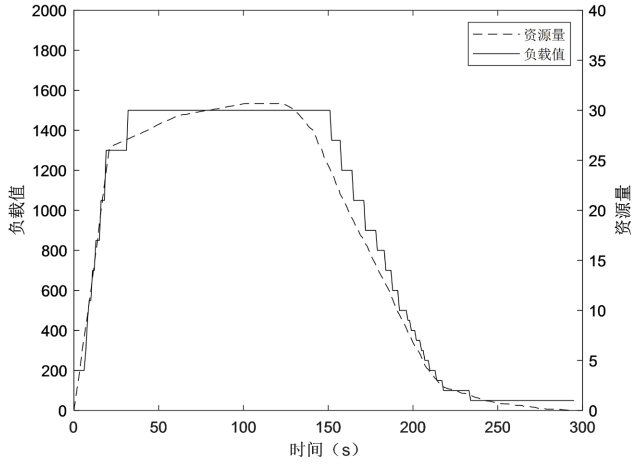

Figure 7. The scaling process of similar algorithms
图 7. 同类算法的伸缩过程

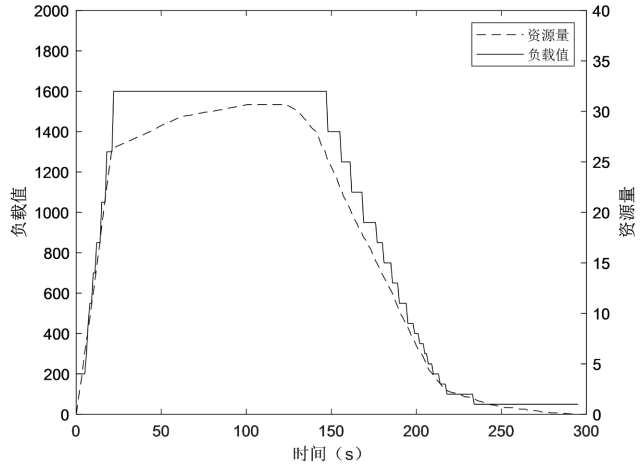

Figure 8. The scaling process of this algorithm
图 8. 本文算法的伸缩过程

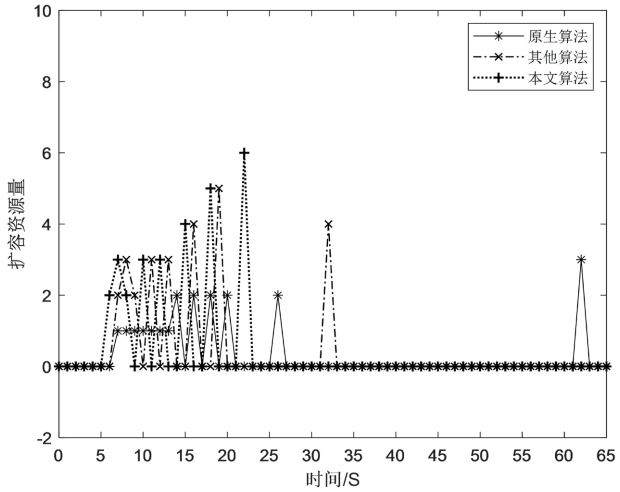

Figure 9. Comparison of resource expansion situations
图 9. 资源扩容情况对比

对比图 6，图 7，图 8 以及结合图 9 的分析可以发现，在 0~20 秒，负载处于快速上升阶段(高负载波动期)，原生方法并不能提前做出扩容反应，当 CPU 利用率超过所设置的上限阈值时，原生算法的伸缩策略才开始触发扩容策略。同类方法虽然能在不同伸缩力度的基础上有着不错的伸缩效果，但由于这种方法仍然属于反应式的伸缩方式，对于持续的高负载，可能会由于扩容不及时而出现 CPU 利用率过高的问题，造成系统压力过大，严重时会导致集群崩溃，影响用户体验。本文方法在预测的方式下能在负载上升之前触发扩容策略，并且由于伸缩力度的细粒化，在处于高负载波动期时，系统会增大伸缩的力度，执行大幅度的扩容操作，进而避免了因固定的伸缩力度造成 QoS 降低，影响用户体验，最终使得高负载波动下，系统始终保持有可用的资源。在 20~60 秒，负载处于正常上升阶段(正常负载波动期)，本文方法在上一阶段的基础上降低了伸缩的力度，系统执行小幅度的扩容操作，避免了大幅度扩容造成不必要的资源浪费。在 60~100 秒，负载处于慢速上升阶段(低负载波动期)。三种伸缩策略因伸缩力度和扩容时间的不同导致最终扩容的资源数量不同，综合来看，融合三支决策的预测式容器伸缩优化策略比 Kubernetes 原生算法和同类其他算法的伸缩策略在负载变化的云环境更适合，每次都能在负载上升之前做好了细粒度的扩容的准备。

4.4.3. 平均 CPU 利用率和 SLA 违约率对比

图 10 是整个伸缩过程的 CPU 利用率对比情况。三种伸缩策略所设置的上限阈值均为 0.7，本文算法的下限阈值为 0.55，原生算法的下限阈值为 0.4，并将本文方法扩容阈值处的 δ 设置为 3%，伸缩因子取 0.1。

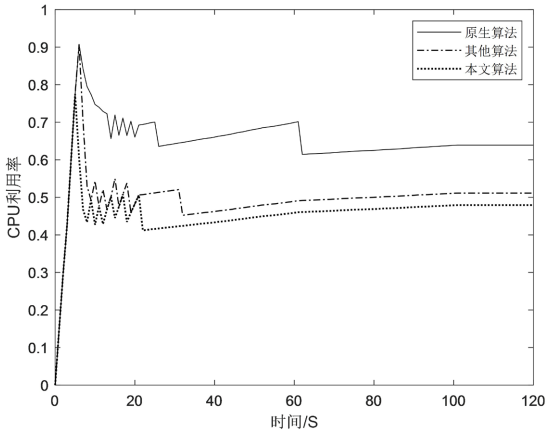

Figure 10. Comparison of resource utilization
图 10. 资源利用率情况对比

由于本文算法在扩容阶段采用的是预测式伸缩，而缩容阶段采用的是反应式伸缩，所以此处只统计负载上升阶段的 CPU 利用率数据，数据对比情况如表 3 和图 11 所示。表中原生算法是指 Kubernetes 原生伸缩策略，其他算法是指未使用预测式伸缩方式的本文同类方法。

Table 3. Comparison of average CPU utilization and SLA breach rate
表 3. 平均 CPU 利用率和 SLA 违约率对比

Algorithm	AvgCpuUtil	SLA
原生算法	65.22%	13.73%
同类算法	49.76%	2.94%
本文算法	45.66%	0.98%

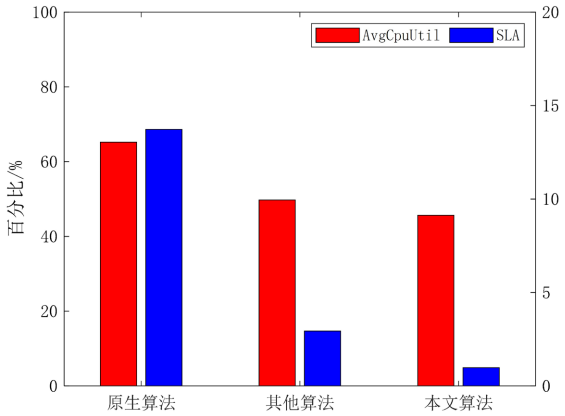

Figure 11. Comparison of average CPU utilization and SLA breach rate
图 11. 平均 CPU 利用率和 SLA 违约率对比

通过分析图 10 以及结合图 11 和前面的相关数据可知，在第 0~20 秒左右，Pod 的工作负载是处于不断升高的趋势，Kubernetes 原生方法虽然也有对其进行相应的扩容操作，但是由于该伸缩策略存在时间滞后性的特点，并且扩容力度始终保持不变，导致可用资源始终得不到满足，致使 CPU 利用率过高。其他同类方法虽能在动态伸缩力度的基础上使得原生方法所出现的问题有所改善，但由于反应式伸缩策略的限制，这种方式仍然存在个别时刻的 CPU 利用率过高和 SLA 违约的问题。融合三支决策的预测式容器伸缩优化策略由于提前预测了未来时刻的工作负载，并且结合负载波动状况和预测负载值对伸缩的力度也进行了细粒度划分，从而提前做出了相应伸缩力度的扩容操作，有效缓解了当前系统的资源使用压力，使得 CPU 利用率始终保持在 50%~60%左右，提高了系统的稳定性。在 20~100 秒左右，两种伸缩策略的 CPU 利用率处于平稳期，但是 Kubernetes 原生方法的伸缩策略始终处于接近 70%左右的 CPU 利用率，CPU 利用率偏高。本文方法的伸缩策略提供了稳定 CPU 利用率，始终处于 50%左右，保证了 QoS，提高了用户体验，降低了系统压力。在 100 秒以后，随着 CPU 利用率的降低，两种伸缩算法均采用响应式的缩容策略，进而提高了系统的稳定性，保证了服务的质量。

4.5. 实验结果及分析

本文提出的融合三支决策的预测式容器伸缩优化策略，一方面，在预测未来时刻负载进行伸缩时保留了响应式伸缩策略，即具有良好响应效果的同时，还能对未来时刻的负载情况进行预测，提高了容器对集群资源请求的响应能力。另一方面，通过融合三支决策思想，对不同负载波动期的伸缩力度进行细粒度划分，改变了以往基于固定伸缩力度的伸缩策略，使得资源得到高效利用，从而降低了系统压力，提升了用户体验。

5. 结束语

针对目前容器调度界主流标准 Kubernetes 基于阈值的响应式伸缩策略所存在的时间滞后性，难以及时响应集群服务资源请求等问题，提出了一种融合三支决策的预测式容器伸缩优化策略。该策略可以对未来时刻工作负载进行准确预测并依托三支决策快速高效地做出伸缩响应。通过设计工作负载场景对改进后容器集群伸缩方法进行了测试对比，验证了该策略的可行性和有效性。但该策略也存在不足之处，即对短周期性工作负载具有较好的预测效果，而对其他类型工作负载的预测效果可能并不理想。所以后续的研究工作开展可以围绕机器学习的方法来建立预测模型，以提高预测模型在各种工作负载场景下的预测准确性。

参考文献

[1] 冷海涛, 马思思. 虚拟化技术在云计算中的应用及平台架构探索[J]. 中国管理信息化, 2022, 25(21): 176-178.

[2] Felter, W., Ferreira, A., Rajamony, R., *et al*. (2015) An Updated Performance Comparison of Virtual Machines and Linux Containers. 2015 *IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS*), Philadelphia, 29-31 March 2015, 171-172. https://doi.org/10.1109/ISPASS.2015.7095802

[3] 武志学. 云计算虚拟化技术的发展与趋势[J]. 计算机应用, 2017, 37(4): 915-923.

[4] 郭雷, 毛玲燕. 基于 Kubernetes 的企业级容器云平台设计与实践[J]. 信息技术与标准化, 2022(9): 73-77.

[5] 耿伟, 周起如, 眭小红, 谷国栋, 赵瑜. 基于排队理论的容器云弹性伸缩策略研究[J]. 电脑编程技巧与维护, 2023(1): 88-90. https://doi.org/10.16184/j.cnki.comprg.2023.01.011

[6] 孙婧, 曹兆元, 郭锦杰. 基于 Docker 容器自动伸缩技术的调度方法[J]. 信息化研究, 2020, 46(4): 25-31+36.

[7] 徐胜超, 熊茂华. 基于深度学习的容器云弹性伸缩方法[J]. 云南师范大学学报(自然科学版), 2021, 41(6): 21-24.

[8] 马小淋. 一种基于负载特征预测的容器云弹性伸缩策略[J]. 信息安全研究, 2019, 5(3): 236-241.

[9] Yao, Y.Y. (2019) Tri-Level Thinking: Models of Three-Way Decision. *International Journal of Machine Learning and Cybernetics*, **11**, 947-959. https://doi.org/10.1007/s13042-019-01040-2

[10] Yao, Y.Y. (2021) The Geometry of Three-Way Decision. *Applied Intelligence*, **51**, 6298-6325. https://doi.org/10.1007/s10489-020-02142-z

[11] Yao, Y.Y. (2016) Three-Way Decisions and Cognitive Computing. *Cognitive Computation*, **8**, 543-554. https://doi.org/10.1007/s12559-016-9397-5

[12] Yao, Y.Y. (2018) Three-Way Decision and Granular Computing. *International Journal of Approximate Reasoning*, **103**, 107-123. https://doi.org/10.1016/j.ijar.2018.09.005

[13] Yao, Y.Y. (2020) Set-Theoretic Models of Three-Way Decision. *Granular Computing*, **6**, 133-148. https://doi.org/10.1007/s41066-020-00211-9

[14] 刘盾, 李天瑞, 杨新, 梁德翠. 三支决策-基于粗糙集与粒计算研究视角[J]. 智能系统学报, 2019, 14(6): 1111-1120.

[15] 马新宇, 黄春梅, 姜春茂. 基于三支决策的 KNN 渐进式文本分类方法[J]. 计算机应用研究, 2023, 40(4): 1065-1069. https://doi.org/10.19734/j.issn.1001-3695.2022.08.0457

[16] 姚一豫, 祁建军, 魏玲. 基于三支决策的形式概念分析、粗糙集与粒计算[J]. 西北大学学报(自然科学版), 2018, 48(4): 477-487. https://doi.org/10.16152/j.cnki.xdxbzr

[17] 胡声丹, 苗夺谦, 姚一豫. 基于三支标签传播的半监督属性约简[J]. 计算机学报, 2021, 44(11): 2332-2343.

[18] 索郎王青, 杨海龙, 姚一豫. 三元思维: 三支决策理论与实践[J]. 陕西师范大学学报(自然科学版), 2022, 50(3): 7-16. https://doi.org/10.15983/j.cnki.jsnu.2022102

[19] 姜春茂, 王凯旋. 基于三支队列的实时云任务节能调度算法[J]. 郑州大学学报(理学版), 2019, 51(2): 66-71. https://doi.org/10.13705/j.issn.1671-6841.2018224

[20] 徐晓霞, 姜春茂, 黄春梅. 一种基于三支决策的移动云任务节能卸载方法[J]. 南京理工大学学报, 2019, 43(4): 447-454. https://doi.org/10.14177/j.cnki.32-1397n.2019.43.04.010

[21] 吴俊伟, 姜春茂. 负载敏感的云任务三支聚类评分调度研究[J]. 智能系统学报, 2019, 14(2): 316-322.

[22] 刘帅帅, 姜春茂. 能耗感知下云资源三支粒度调度策略研究[J]. 计算机应用研究, 2023, 40(3): 810-815. https://doi.org/10.19734/j.issn.1001-3695.2022.07.0365

[23] 苗立尧. 基于 Docker 容器的混合式集群伸缩方法研究[D]: [硕士学位论文]. 西安: 西安邮电大学, 2016.

[24] 杨忠. 面向 Docker 容器的动态负载集群伸缩研究[J]. 舰船电子工程, 2018, 38(8): 109-115.

[25] 刘钱超, 吴利, 郑礼辉. 一种基于二次移动平均法的容器云伸缩策略[J]. 计算机技术与发展, 2019, 29(10): 15-20.

[26] 杨茂, 陈莉君. 基于 Kubernetes 的容器自动伸缩技术的研究[J]. 计算机与数字工程, 2019, 47(9): 2217-2220+2232.

[27] 黄芳, 闫锐, 牛金洲. 基于时序指数平滑法的冷库能耗预测模型构建[J]. 上海节能, 2023(3): 336-341. https://doi.org/10.13770/j.cnki.issn2095-705x.2023.03.016

[28] 徐彦农, 王一帆, 王浩淳, 田辰蔚, 张兆欣, 李昕潞. 基于一次/二次指数平滑法的风功率预测方法[J]. 南方农机, 2021, 52(21): 26-28.

[29] 黄炜达, 朱维骏, 蓝映彬. 基于二次指数平滑法的能源分析预测方法[J]. 节能与环保, 2023(2): 63-65.

[30] 李钢, 陈自然, 田伟, 等. 应用二次指数平滑法的光栅信号细分方法研究[J]. 重庆理工大学学报: 自然科学, 2018, 32(2): 86-92, 236.