

云环境下自适应容器弹性扩缩容优化方法

丁宇¹, 袁麒景², 才振功², 左海余², 李晨昊¹, 杨波威³

(1. 浙江大学 计算机科学与技术学院, 杭州 310027; 2. 浙江大学 软件学院, 宁波 315000; 3. 浙江大学 航空航天学院, 杭州 310027)

摘要: 针对云服务中容器弹性扩缩容过程中, 在满足服务质量要求的前提下实现整体资源利用率提升与成本降低的问题, 提出了一种双向长短期记忆网络、极值理论与动态利用率算法相结合的自适应容器弹性扩缩容优化方法 DEBI-HPA, 该方法能根据负载变化动态地调整 Pod 数量, 有效地估计负载超高位, 并自适应地调整目标利用率, 以提高服务质量和资源利用率, 降低资源成本。在负载不高时采用预测式扩缩容来应对未来可能持续增长的负载, 而在负载已经处于超高位的情况下, 采用响应式的扩缩容, 能在低负载情况下保证足够 Pod 数以供缓冲, 在高负载情况下大幅度提高资源利用率。本文与原生的 Kubernetes 横向扩缩容策略进行了对比, 实验结果表明: 在不同负载情况下此算法可使资源利用率最高提升约 30%, 节约资源成本最高约 47%, 与已有通过预测模型进行扩缩容的研究方法 (ARIMA-HPA、LSTM-HPA 策略) 相比, 该算法能将资源利用率提升约 18%, 资源使用成本降低约 37%。本文方法仅使用常见的预测模型也能达到比较好的扩缩容效果, 降低了对预测模型效果的依赖性, 为云环境下的容器扩缩容提供了新的解决方案。

关键词: 弹性扩缩容; 极值理论; 动态目标利用率; 自适应算法; 双向长短期记忆网络

中图分类号: TP399 **文献标志码:** A **文章编号:** 1671-5497(2026)07-1984-12

DOI: 10.13229/j.cnki.jdxbgxb.20241348

Adaptive container auto-scaling optimization method in cloud environments

DING Yu¹, YUAN Qi-jing², CAI Zhen-gong², ZUO Hai-yu², LI Chen-hao¹, YANG Bo-wei³

(1. College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China; 2. College of Software Technology, Zhejiang University, Ningbo 315000, China; 3. College of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, China)

Abstract: This paper introduces DEBI-HPA, an adaptive container auto-scaling method for cloud services that integrates Bi-directional Long Short-Term Memory (BiLSTM), extreme value theory, and dynamic utilization ratios. DEBI-HPA dynamically adjusts the number of Pods based on workload changes, accurately estimates extreme load levels, and adaptively sets target utilization ratios, thereby enhancing service quality, resource utilization, and cost efficiency. It employs predictive scaling during low-load periods to anticipate growth and reactive scaling under high loads to ensure efficient resource allocation.

收稿日期: 2024-12-23.

基金项目: 国家重点研发计划项目 (2024YFB4505900).

作者简介: 丁宇 (1982-), 男, 博士研究生. 研究方向: 云计算. E-mail: 12021191@zju.edu.cn

通信作者: 才振功 (1983-), 男, 副研究员, 博士. 研究方向: 云原生, 边缘智能. E-mail: cstcaizg@zju.edu.cn

Experimental results demonstrate that DEBI-HPA outperforms Kubernetes' native Horizontal Pod Autoscaler (HPA) by increasing resource utilization by up to 30% and reducing costs by up to 47%. Compared to other predictive scaling methods like ARIMA-HPA and LSTM-HPA, DEBI-HPA improves resource utilization by approximately 18% and lowers costs by about 37%. Additionally, DEBI-HPA achieves effective scaling with common prediction models, reducing dependency on model accuracy. This method offers a novel and robust solution for container auto-scaling in cloud environments.

Key words: auto-scaling; extreme value theory; dynamic target utilization; adaptive algorithm; bidirectional LSTM

0 引言

随着云计算与容器技术的快速发展,Kubernetes已广泛应用于云环境的弹性扩缩容管理。然而,原生扩缩容策略(HPA^[1]、VPA^[2]、MPA^[3])多为响应式的基于阈值方法^[4-6],仅能根据当前负载进行扩缩容,缺乏前瞻性且需人工设定目标利用率,易导致滞后、震荡与资源浪费。为提升扩缩容效率和灵活性,学界与工业界尝试通过动态阈值^[7,8]、闭环控制^[9]、及负载预测(如ARIMA^[10]、STL^[11]、灰色预测^[12]、VMD-TCN^[13]、ARIMA+NN^[14]、LSTM^[15])等方法提前预测未来负载变化,从而降低服务违约(Service level agreement, SLA)及资源浪费。但复杂深度预测模型的训练周期长、资源消耗大,且在动态多变的生产环境中难以保证模型自适应性,需频繁针对不同场景重新训练^[16-18],不利于普通企业和用户的实际应用。

针对上述问题,本文提出了一种融合双向长期记忆网络(Bidirectional long short-term memory, BiLSTM)、极值理论(Extreme value theory, EVT)与动态目标利用率策略的自适应扩缩容优化算法。该算法(EVT-BiLSTM-HPA)在负载低时利用BiLSTM进行负载预测,提前扩容应对未来增长趋势,提高服务可靠性;在负载已处于极高水平时则采用响应式扩缩容策略,通过极值理论降低继续大幅增长的概率,避免不必要的资源浪费。

在此基础上,本文进一步提出DEBI-HPA(Dynamic target utilization-EVT-BiLSTM-HPA)算法,通过在负载低谷期下调目标利用率,使系统在负载上升前就已拥有足够资源,从而减少SLA违约率;在负载高峰时提升目标利用率,提高整体资源利用率和成本效益。

本研究在Kubernetes平台上进行了实验验证。与原生HPA及常见优化方法相比,所提出的

方法在保证服务质量与可用性的同时,有效提高了资源利用率并降低了成本,为云环境中弹性扩缩容策略的设计与实现提供了新的解决思路。

1 相关理论和方法

1.1 Kubernetes弹性伸缩原理及现有缺陷

在Kubernetes中,Metrics Server持续采集所有Pod副本的指标数据,并通过聚合API(如metrics.k8s.io、custom.metrics.k8s.io)为HPA控制器提供查询服务^[19]。HPA控制器根据用户定义的扩缩容规则,使用式(1)计算目标Pod副本数量:

$$\text{desiredReplicas} = \frac{\text{desiredReplicas} \times \text{currentMetricValue}}{\text{desiredMetricValue}} \quad (1)$$

式中:desiredReplicas为期望的副本数;currentReplicas为当前所有Pod副本的副本数;currentMetricValue为当前Pod副本的指标数据值;desiredMetricValue为期望的指标数据值; $\lceil \cdot \rceil$ 表示向上取整操作。

如果计算出的期望副本数与当前副本数不一致,HPA控制器将向Pod控制器(如Deployment或ReplicaSet)发出Scale操作,调整Pod的数量,实现弹性扩缩容。图1展示了HPA的工作原理。

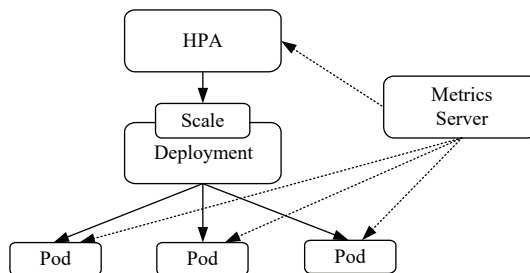


图1 HPA工作原理

Fig. 1 Working principle of HPA

此外,在扩缩容计算过程中,若 Pod 正在被删除、当前指标值无法获取,或 Pod 状态为启动中且未处于 Ready 状态,该 Pod 将不参与计算。此外,研究表明,一个好的伸缩方法应具备以下几种性质^[20]:

(1) 系统稳定性:伸缩应保证系统整体的稳定性,避免因频繁伸缩引起的窗口抖动,从而影响系统性能。

(2) 自定义控制循环周期:能根据不同场景自定义控制循环时间周期(Control loop time period, CLTP)。

(3) 平缓的 Pod 减少:在 Pod 数量减少时应采取平缓的方式,防止因应用负载突然增长而导致服务中断。

Kubernetes 的 HPA 通过滑动时间窗口和容忍度机制较好地实现了上述要求。容忍度可以通过 kube-controller-manager 的启动参数 horizontal-pod-autoscaler-tolerance 进行配置。假设设置的容忍度值为 α ,用户设定的目标平均资源利用率为 β ,则当当前平均资源利用率处于 $[\beta \times (1 - \alpha), \beta \times (1 + \alpha)]$ 范围内时,HPA 不会触发扩缩容操作。这一机制在一定程度上防止了 Pod 副本数的频繁抖动。

此外,HPA 在执行扩缩容操作前,会将计算得到的 Pod 副本数建议记录在一个默认为 5 min 的滑动时间窗口中。在实际进行扩缩容时,HPA 总是选择该窗口中的最大 Pod 建议数作为目标 Pod 数,从而有效减少 Pod 副本数的波动,保持 Pod 数量平稳下降的趋势。用户还可以根据实际需求调整扩缩容计算的时间间隔和滑动窗口长度,确保 CLTP 的灵活性。

Kubernetes 的原生 HPA 属于响应式伸缩方案,其核心在于根据当前 Pod 副本的负载状况计算目标 Pod 数,并做出伸缩决策。当负载增加时,HPA 根据计算结果决定扩容,但新创建的 Pod 提供服务仍需经历一定的延迟,如图 2 所示的流程,以及对应的扩容时序图(图 3)。

在 HPA 扩缩容过程中,Pod 的初始化经历 4 个阶段(T1 至 T4):T1:HPA 根据当前 Pod 副本的负载情况计算期望副本数;T2:通过 Deployment 操作,将期望副本数发送给调度器;T3:调度器将新增的 Pod 放入调度队列,并为其找到合适的 Node;T4:在相应 Node 上,Kubelet 从镜像

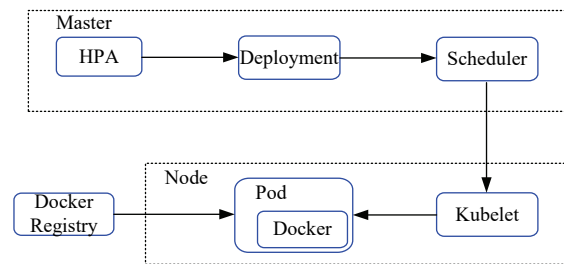


图 2 HPA Pod 扩容流程图

Fig. 2 HPA pod expansion flowchart

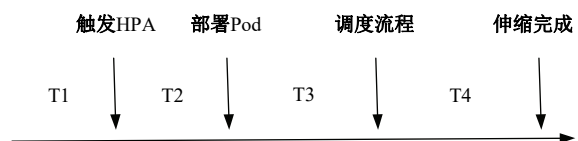


图 3 HPA 扩容时序图

Fig. 3 HPA expansion timing diagram

仓库拉取镜像并启动容器。Pod 完成初始化后, kube-proxy 修改路由表,将 Pod 加入新的转发表,Pod 才能开始提供服务。因此,Pod 的初始化时间 t_{init} 为 4 个阶段时间的总和:

$$t_{init} = \sum_{i=1}^4 t_i \quad (2)$$

其中,第四阶段的时间通常最长。当 Pod 能提供服务时,资源申请量可能与实际消耗不匹配。因此,采用预测式的扩缩容方法,提前预测未来负载变化,并结合智能的弹性扩缩容策略,可以降低资源不必要的消耗,提高资源使用率,减少服务质量违约的可能性。

1.2 负载预测方法

自回归积分滑动平均模型(Autoregressive integrated moving average, ARIMA)是一种常用的时间序列预测模型,是自回归移动平均模型(Autoregressive moving average model, ARMA)的推广,适用于处理非平稳时间序列。ARIMA 模型由 3 个部分组成:自回归、移动平均和差分。其模型表示为 $ARIMA(p, d, q)$, 见公式(3)。

$$x_t = \varphi_1 x_{t-1} + \cdots + \varphi_p x_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \quad (3)$$

式中: p 为自回归阶数; d 为差分次数; q 为移动平均阶数; φ_i 和 θ_i 分别为自回归和移动平均系数; ε_t 为白噪声过程^[21]。

ARIMA 模型的优点是简单易用,适合多种时间序列;缺点是需要手动确定参数,且无法处理非线性和多变量序列。

长短期记忆网络(Long short-term memory,

LSTM)是一种循环神经网络(Recurrent neural network, RNN),能处理长期依赖问题,即当前输出不仅依赖于当前输入,还依赖于过去的输入^[22]。LSTM由多个LSTM单元组成,每个单元包含一个细胞状态和3个门:输入门、遗忘门和输出门。细胞状态在单元内部传递信息,通过门控机制控制信息的流入、流出和保留。LSTM在时刻 t 的状态计算如式(4)~(9),网络结构见图4。

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (4)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (5)$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (6)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \quad (7)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (8)$$

$$h_t = o_t * \tanh(c_t) \quad (9)$$

式中: f_t 、 i_t 、 o_t 分别为遗忘门、输入门和输出门; c_t 为细胞状态; $\tilde{c}_t$ 为细胞状态候选值; h_t 为隐藏层状态; W 和 b 为权重和偏置。

LSTM通过上述机制有效解决了梯度消失问题,能够长期记忆某一时刻的状态值,解决了长期依赖问题。

双向长短期记忆网络(Bidirectional long short-term memory, BiLSTM)是一种双向循环神经网络(Bidirectional recurrent neural network, BRNN),能同时考虑过去和未来的信息,从而提高预测准确性。BiLSTM由两个方向相反的LSTM网络组成,一个从前向后处理输入序列,另一个从后向前处理输入序列^[23]。其网络结构如图5所示。BiLSTM的输出是两个LSTM网络输出的结合,能利用双向信息,提高预测性能。

1.3 极值理论

EVT是一种用于分析极端事件发生概率及

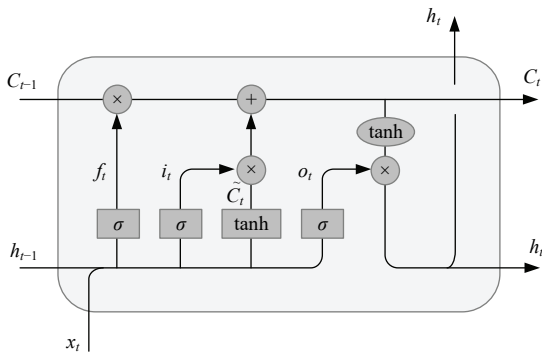


图4 LSTM模型基本结构^[24]

Fig. 4 Basic structure of the LSTM model^[24]

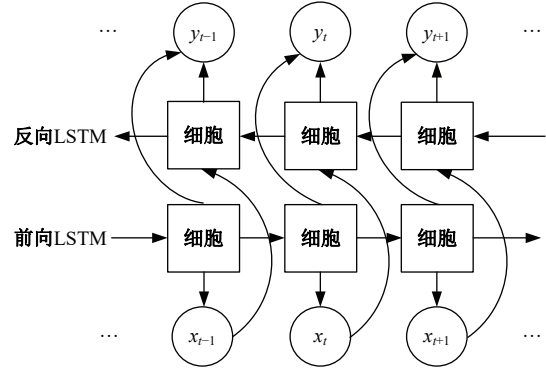


图5 BiLSTM模型结构

Fig. 5 Structure of the BiLSTM model

其影响程度的数学理论。基于极限定理,EVT能够从大量数据中提取极端值并建立其分布模型。

极值理论最早可以追溯到Fréchet发表的最大值渐进分布论文^[25],论文中表明,来自于不同分布,但具有共性的最大值具有相同的渐进分布,并且提出了最大值稳定原理^[26],这为极值理论的研究奠定了基础。后来,Fisher和Tipper的研究表明小概率事件服从另一种概率分布。同高斯分布一样,如果事件越极端,其发生的概率就越低。如果要计算具体的发生概率,只要使用以往该极端事件的发生情况拟合曲线即可,再随着在次序统计量的分布性质和关于底分布的超阈值性质研究方向都取得了长足的发展,逐渐建立起了完整的极值理论研究体系^[27]。在极值理论中,对于序列 $X = \{x_1, x_2, x_3, \dots, x_n\}$,其中 $x_i \in \mathbb{R}$,其极端值满足以下分布定义(式10):

$$G_\gamma(x) = e^{-\frac{(1-\gamma x)^{\frac{1}{\gamma}}}{\gamma}} > 0 \quad (10)$$

任何极值的标准分布均服从 G_γ ,其中参数 γ 依赖于序列 X 的具体分布。例如,当序列 X 满足独立同分布时,最大值 $x_m = \max(X)$ 服从 G_γ 分布。

设 X 为随机变量,其累计分布函数为 $F(x) = P(X \leq x)$,则其尾部分布函数为 $\bar{F}(x) = 1 - F(x) = P(X > x)$ 。引入概率常量 q ,那么 $P(X \leq y) \geq 1 - q$ 则等价于 $P(X > y) \leq q$ 。

如果已知 X 的分布,可以通过累计分布函数 $F(x)$,对固定值 q 求出对应的 y ,对固定值 y 求出对应的 q 。对于尾部分布 $\bar{F}(x)$,随着 X 的增大,函数值趋近于0。EVT认为尾部分布的函数形状相对固定,主要有3种类型,可用 G_γ 进行近似拟合,如表1所示。

极值理论的优点在于能有效估计极端事件的

表 1 G_y 分布尾部形态^[28]
Table 1 G_y distribution tail shape^[28]

尾部形状	范围	示例
厚重尾部 $P(X > x) \approx x^{-\frac{1}{y}}$	$y > 0$	Frechet
指数尾部 $P(X > x) \approx e^{-x}$	$y = 0$	Gamma
有界尾部 $P(X > x) \approx 0, x \geq \tau$	$y < 0$	Uniform

发生概率及其影响。然而,其缺点是需要合理选择区域大小或高位阈值,否则会影响结果的准确性。

2 本文方法

2.1 基于 DEBI-HPA 的自适应容器弹性扩缩容优化方法主要过程

在应对持续增长的负载时,延迟扩容易导致资源消耗与实际需求不匹配。为此,本文提出一种基于 DEBI-HPA 的自适应容器弹性扩缩容优化方法。与传统 HPA 略显被动的响应式扩缩容策略不同,本方法在负载较低时利用 BiLSTM 预测模型进行预测式扩缩容,应对可能的未来负载增长;当负载已接近超高位时,通过极值理论判断负载进一步大幅增长的概率极低,此时转为响应式扩缩容,以减少不必要的资源分配。

在此基础上,DEBI-HPA 引入动态目标利用率调整算法,解决无法根据 Pod 实际运行情况和历史数据自动调整目标利用率的问题。在负载低谷阶段降低目标利用率,留出缓冲空间以减少负载上升时引发的 SLA 违约;在负载高峰且增长空间有限的情况下,提高目标利用率,提升整体资源利用率并降低成本。

DEBI-HPA 的整体流程如图 6 所示。系统首先从 Metrics Server 中采集当前 Pod 集合的平均 CPU 利用率负载 W_1 ;接着通过 BiLSTM 预测模型获取当前时刻的预测负载 W_2 ;随后利用极值理论算法计算超高位负载 W_3 。综合 W_1 、 W_2 与 W_3 得到最终负载 W ,并基于此由动态目标利用率算法对 Pod 副本数进行自适应调整。

预测算法通过分析和总结相同场景下的历史数据,在场景不变的前提下预测未来情况。对于负载数据,其与时间具有一定的非线性关系,属于时间序列数据,并且大多数在线业务的资源负载具有较强的周期性^[29]。因此,时间序列算法适用于负载预测。时序预测方法本质上是一种回归预测算法,基于对过去数据的统计分析,假设事物发展是可持续的,从而推测其趋势。然而,事物的发展往往受

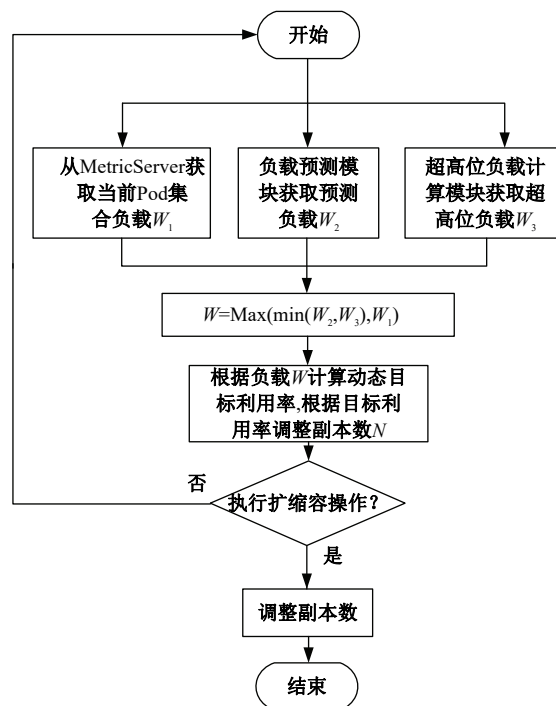


图 6 自适应弹性扩缩容策略整体流程图

Fig. 6 Overall flowchart of the adaptive elastic scaling strategy

到外界因素的影响,使用时序预测算法时需注意避免噪声数据的干扰。在 Kubernetes 运行过程中,本文通过历史负载数据预测未来时刻的负载,并根据预测结果进行相应操作。由于预测过程会占用额外的资源,需选择资源消耗和预测耗时较低的算法,以防止预测模块对集群性能产生影响或导致预测不及时。

预测算法模型采用常见的 BiLSTM 模型。本文设计的 BiLSTM 模型由一个双向 LSTM 层和一个线性层组成。双向 LSTM 层包含一个正向 LSTM 和一个反向 LSTM,分别从左到右和从右到左处理输入序列。每个方向有 100 个隐藏单元,输入和输出的维度均为 1。两个 LSTM 的隐藏状态被拼接后输入到线性层生成输出。负载预测模块通过学习历史 Pod 集合的平均 CPU 利用率数据,利用该 BiLSTM 预测算法预测当前时刻的负载 W_2 。

2.2 极值理论确定负载超高位算法

本文将负载超高位问题定义为:设 $X = \{x_1, x_2, x_3, \dots, x_n\}$, 其中 $x_i \in R$, 为负载数据的时序序列,存在阈值 z , 满足序列中任意数据 $x_i > z$ 的概率小于设定概率 q 。其中, z 为超高位负载的临界值,即前文提到的超高位负载 W_3 。

根据极值理论的性质,无需了解原始数据的具体分布,即可计算极值发生的概率。问题转化为求解极值分布中的参数 γ 。一旦确定了 γ ,便可通过数据尾部进行拟合,并在给定概率 q 下求出满足 $P(X > z) < q$ 的 z 值。超阈值 (Peaks-Over-Threshold, POT) 方法是估计 γ 的有效方法,其理论基础来源于 Pickands-Balkema-de Haan 定理。该定理严格证明了当阈值充分大时,超出量的条件概率分布可收敛于广义帕累托分布 (Generalized pareto distribution, GPD)。

Pickands-Balkema-de Haan 定理指出,当累积分布函数 F 属于 D_γ^1 类时,且存在函数 $\sigma(t)$ 对任意 $x_t \in R$, 有 $1 + \gamma x > 0$, 则:

$$F(t + \sigma(t)x) \approx \tau(1 + \gamma x)^{-\frac{1}{\gamma}} \quad (11)$$

简化后得到:

$$\bar{F}_t(x) = P(X - t > x | X > t) \sim \left(1 + \frac{\gamma x}{\sigma(t)}\right)^{-\frac{1}{\gamma}} \quad (12)$$

式 (12) 表明,超过阈值 t 的部分 $X - t$ 近似服从参数为 γ 和 σ 的广义帕累托分布。由于是将超过阈值 t 的部分采用 GPD 拟合,因此本文忽略 GPD 的第三个参数,通过拟合 γ 和 σ 得到估计值 $\hat{\gamma}$ 和 $\hat{\sigma}$,进而计算满足 $P(X > z) < q$ 的 z 值:

$$z_q \approx t + \frac{\hat{\sigma}}{\hat{\gamma}} \left(\left(\frac{qn}{N_t} \right)^{-\frac{1}{\gamma}} - 1 \right) \quad (13)$$

式中: t 为设定的阈值 (通常为较高的分位数); q 为设定的概率; n 为样本数; N_t 为样本中满足 $X_i > t$ 的个数。

在基于极值理论参数求解中,极大似然估计 (Maximum likelihood estimation, MLE) 是常用的方法之一,可以采用 Grimshaw 方法进行简化求解。尽管通过限制搜索范围和引入启发式策略可以在一定程度上加速求解过程,但该方法在实际应用中仍存在较高的计算复杂度和较长的求解时间,特别是在需要快速响应的场景下表现不佳。受研究启发,本文采用矩估计法 (Peaks-over-threshold based on MOMents, POT_MOM), 与 MLE 相比, POT_MOM 在计算复杂度和速度方面效率更高,更适用于本文所描述的场景需求^[30]。

矩估计的推导过程详见文献[31]。在满足 $\gamma\sigma > -1$ 的条件下,样本方差和均值与参数 γ, σ 满足关系式 (14),从而可以通过式 (15) 给出 γ' 和

σ' 的估计值,整体算法框架见算法 1。

$$X = \sigma(1 + \gamma), s^2 = \frac{\sigma^2(1 + 2k)}{(1 + k)^2} \quad (14)$$

$$\gamma' = \frac{1}{2}(X^2 s^2 - 1), \sigma' = \frac{1}{2}X(X^2 s^2 + 1) \quad (15)$$

算法 1 POT_MOM 确定负载超高位算法

输入: 负载序列 $X = \{x_1, x_2, x_3, \dots, x_n\}$, 概率阈值 q

输出: 超高位负载阈值 z , 分位数 t

1. 初始化负载序列 $X = \{x_1, x_2, x_3, \dots, x_n\}$, 得到其 q 分位数 t
2. 计算负载序列 X 中超过阈值 t 的样本 $Y_t = \{x_i - t | x_i > t\}$
3. $z = \text{MOM}(Y_t, n)$
4. return z, t
5. func $\text{MOM}(Y_t, n)$
6. $\bar{Y} = Y_t$ 的均值
7. $s^2 = Y_t$ 的方差
8. $\gamma' = \frac{1}{2}(\frac{\bar{Y}^2}{s^2} - 1)$
9. $\sigma' = \frac{1}{2}\bar{Y}(\frac{\bar{Y}^2}{s^2} + 1)$
10. $z = t + \frac{\hat{\sigma}}{\hat{\gamma}} \left(\left(\frac{qn}{N_t} \right)^{-\frac{1}{\gamma}} - 1 \right)$
11. return z
12. end func

2.3 动态目标利用率扩缩容算法

传统的固定目标利用率方法难以适应不同业务负载变化,可能导致服务质量下降或资源浪费。例如,当负载较低且稳定时,固定目标利用率可能导致 Pod 实例过多,资源利用率低下,成本增加;而在负载较高且不稳定时,固定目标利用率可能无法及时调整 Pod 数量,导致服务性能下降,引起 SLA 违反的可能性。为此,本文提出动态目标利用率方法:

(1) 低负载时: Pod 数量较少,承受负载变化能力有限。通过降低目标利用率,使实际利用率维持在较低水平,增加负载变化的缓冲空间,减少 SLA 违反概率。

(2) 高负载时: Pod 数量较多,承受相同负载变化的能力更强,且负载增长空间有限。通过提高目标利用率,提升整体资源利用率,降低资源需求,节省成本。

自适应弹性扩缩容策略的整体流程包括:从 Metrics Server 采集集群当前 Pod 集合的平均 CPU 利用率负载 W_1 ;负载预测模块提供当前时刻预测的平均 CPU 利用率负载 W_2 ;超高位负载计算

模块利用极值理论算法计算超高位负载 W_3 ; 综合三个负载值, 通过式(16)计算最终负载 W :

$$W = \max(\min(W_2, W_3), W_1) \quad (16)$$

用户需设定基本参数: Pod 的资源请求量、Pod 的最大副本数 $N_{\max}$ 与最小副本数 $N_{\min}$ 、目标利用率的上限 $U_{\text{upperbound}}$ 和下限 $U_{\text{lowerbound}}$ 。这些参数基于流量、应用性能和成本考虑, 最大副本数和最小副本数分别对应最高和最低资源消耗; 资源请求量决定单个 Pod 的合理资源分配; 目标利用率的上限和下限确保在不同负载下 Pod 的处理能力, 防止性能瓶颈导致 SLA 违反。动态目标利用率的设定如下: 当 Pod 副本数 $N_{\text{now}} = N_{\min}$ 时, 设定目标利用率为下限 $U_{\text{target}} = U_{\text{lowerbound}}$ 。

固定负载缓冲空间:

$$U_{\text{loadbuffer}} = (U_{\text{upperbound}} - U_{\text{lowerbound}}) \cdot N_{\min} \quad (17)$$

计算目标利用率:

$$U_{\text{target}} = U_{\text{upperbound}} - \frac{U_{\text{loadbuffer}}}{N_{\text{now}}} \quad (18)$$

整体算法框架见算法 2。

算法 2 动态目标利用率 Pod 副本数推荐值

计算

输入: $W, \Delta, N_{\text{now}}, N_{\max}, N_{\min}$

输出: 下一个时刻的扩缩容副本数推荐值

1. U_t = 计算副本数 N_{now} 对应的目标利用率
2. if $U_t \cdot (1 - \Delta) < W < U_t \cdot (1 + \Delta)$ then
3. return N_{now}
4. end if
5. U_{Tmin} = 计算副本数 $N_{\min}$ 对应的目标利用率
6. if $U_t < U_{\text{Tmin}} \cdot (1 + \Delta)$ then
7. return $N_{\min}$
8. end if
9. $U_{\text{total}} = W \cdot N_{\text{now}}$
10. $\text{diff} = \text{INF}$
11. $N_{\text{res}} = -1$
12. for n in range($N_{\min} + 1, N_{\max}$) do
13. U_{Tn} = 计算副本数 n 对应的目标利用率
14. $\text{diff}_n = \text{abs}(U_{\text{Tn}} \cdot n - U_{\text{total}})$
15. if $\text{diff}_n < \text{diff}$ then
16. $N_{\text{res}} = n$
17. $\text{diff} = \text{diff}_n$
18. end if
19. end for
20. if $N_{\text{res}} \neq -1$ then
21. return N_{res}
22. end if
23. return $N_{\max}$

为防止 Pod 频繁变化导致服务质量下降, 采

用滑动窗口方法。根据前 5 min 的真实负载平均利用率, 计算动态目标利用率算法推荐的副本数, 并与当前推荐副本数比较, 选择较大者作为最终副本数。

3 实验与结果分析

本节通过实验验证本文提出的自适应容器弹性扩缩容优化方法的可行性和有效性。

实验数据选自阿里巴巴微服务 2022 数据集^[32]的部分数据。该数据集包含近两万个微服务的详细运行时指标, 数据采集自 2022 年 13 天内阿里巴巴生产集群中超过万个裸机节点。此数据集在云平台弹性扩缩容研究中广泛用于评估扩缩容策略的性能。原始负载数据经过提取、处理和模拟, 通过在集群上压测 Nginx 应用生成了 3 类 CPU 负载数据集: 负载周期性较强、负载波动平缓、负载波动频繁且无明显周期性。3 类负载随时间变化如图 7、图 8、图 9 所示, 横坐标为时间 (min), 纵坐标为同一服务多个 Pod 的平均 CPU 利用率 (%)。

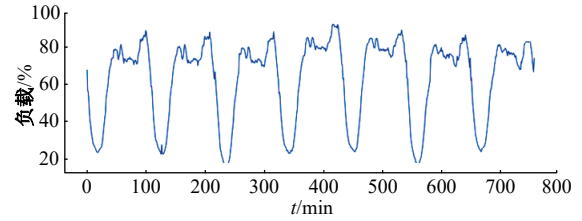


图 7 负载周期性较强的数据集

Fig. 7 Data set with strong load periodicity

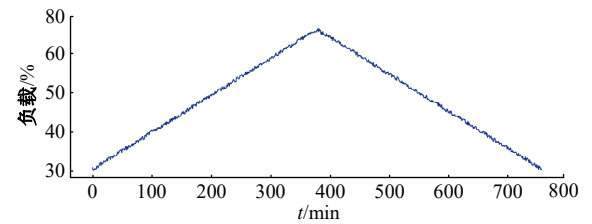


图 8 负载波动平缓的数据集

Fig. 8 Data set with smooth load fluctuations

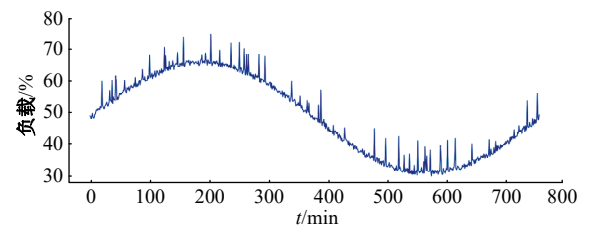


图 9 负载波动频繁且无明显周期的数据集

Fig. 9 Data set with frequent load fluctuations and no clear periodicity

实验采用真实负载数据集驱动的仿真方式。通过修改 Kubernetes HPA 源代码,将真实负载数据输入 HPA 模块,模拟 Metrics Server 采集的集群真实负载数据。数据经过负载预测模块、负载超高位估计模块和动态目标利用率模块处理后,得到最终的 Pod 推荐副本数。实验均在配置如下的实验平台上进行:CPU 为 Intel i9-14900HX(24 核 32 线程,主频 5.8 GHz),内存 16 GB,网络带宽 1 000 Mbit/s,硬盘 250 GB,操作系统 CentOS 7.4, Kubernetes 版本 1.23.20, Docker 版本 20.10.03,同时配备 NVIDIA GeForce RTX 3060 GPU。

实验中使用 ARIMA、LSTM 和 BiLSTM 三种预测模型对负载进行预测。模型训练和预测前,对原始数据进行缩放和一阶差分处理以增强数据平稳性,预测结果通过逆差分 and 逆缩放还原。预测模块使用 760 min 的负载数据,前 90% 作为训练集,后 10% 作为验证集进行模型训练和验证。LSTM 和 BiLSTM 的参数设置为:迭代次数 10 000 次,损失函数小于 $1e-4$ 时停止迭代,隐藏层单元数为 100,采用 Adam 优化器进行梯度下降。

3.1 极值理论确定负载超高位预测式扩缩容实验

本文在实验中针对概率 $q=0.1$ 的设定(即负载超过超高位负载阈值的概率为 10%)对所提出的扩缩容策略进行了评估。为了验证策略的有效性,除与原生 HPA 方案对比外,还选取了使用 ARIMA、LSTM 等预测方法进行优化的扩缩容策略^[33]进行比较。在周期性负载、平缓负载以及无明显周期的频繁波动负载三类典型数据集上,分别执行原生 HPA、ARIMA 与 LSTM 优化的策略、基于 BiLSTM 的 BiLSTM-HPA 方案,以及本文提出的融合 BiLSTM 与极值理论的 EVT-BiLSTM-HPA 策略。由图 10~图 12(彩图参见电子版,下同)可见,当负载达到高峰时,EVT-BiLSTM-HPA 策略通过在此时刻从预测式扩缩容转向响应式扩缩容,不仅有效降低了资源使用成本,还使 Pod 副本数更接近实际负载所需水平。

实验采用资源占用面积指标衡量资源使用成本,资源占用面积可以使用式(19)计算:

$$S = R * \sum_{t=0}^T N_t \quad (19)$$

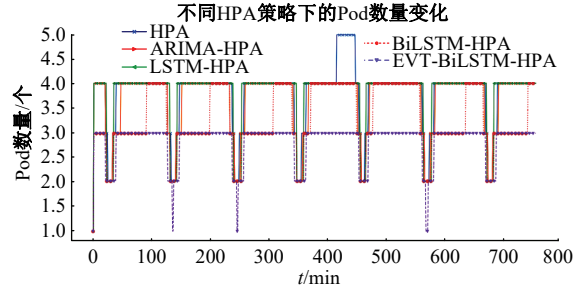


图 10 周期性负载情况下各策略 Pod 数变化

Fig. 10 Changes in the number of Pods for each strategy under periodic load conditions

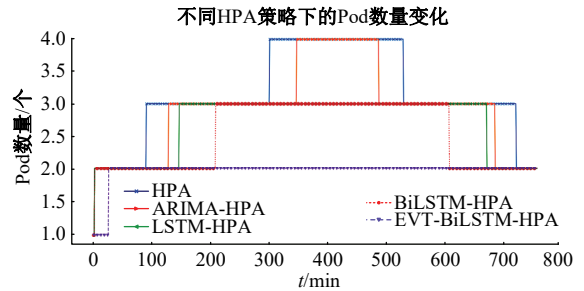


图 11 平缓负载情况下各策略 Pod 数变化

Fig. 11 Changes in the number of Pods for each strategy under smooth load conditions

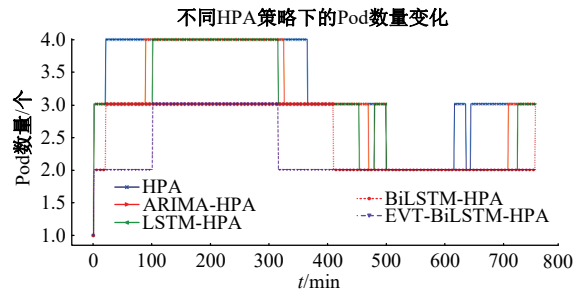


图 12 波动频繁且无明显周期负载情况各策略 Pod 数变化

Fig. 12 Changes in the number of Pods for each strategy under conditions of frequent and non-periodic load fluctuations

式中: R 为资源请求值; t 为时间; T 为时间序列长度; N_t 为 t 时刻的 Pod 副本数; S 为在整个时间序列内,由 Pod 总资源请求量曲线与时间轴围成的面积。 S 值越小,表示在此时间段内所占用的资源总量越低,对应的资源成本也更低。

表 2、表 3、表 4 展示了相应的实验结果。对比表中 HPA 模型、BiLSTM-HPA 模型和 EVT-BiLSTM-HPA 模型的 S 值可以看出,由于预测式扩缩容需要提前申请资源,EVT-BiLSTM-HPA 算法相比于 BiLSTM-HPA 算法的资源占用情况在周期性较明显负载情况下平均能够减少约 14% 的资源成本,在平缓负载情况下平均能减少

表 2 周期性负载情况下不同扩缩容策略总体资源占用面积

Table 2 Total resource usage area for different scaling strategies under periodic load conditions

模型名称	S
HPA	42 930
BiLSTM-HPA	37 200
EVT-BiLSTM-HPA	31 860

表 3 平缓负载情况下不同扩缩容策略总体资源占用面积

Table 3 Total resource usage area for different scaling strategies under smooth load conditions

模型名称	S
HPA	35 520
BiLSTM-HPA	28 620
EVT-BiLSTM-HPA	22 290

表 4 波动频繁且无明显周期负载情况下不同扩缩容策略总体资源占用面积

Table 4 Total resource usage area for different strategies under conditions of non-periodic load fluctuations

模型名称	S
HPA	37 260
BiLSTM-HPA	28 500
EVT-BiLSTM-HPA	25 860

约 22% 的资源成本,而在波动频繁且无明显周期负载的情况下平均也能节对比表中 HPA 模型、BiLSTM-HPA 模型和 EVT-BiLSTM-HPA 模型的 S 值可以看出,由于预测式扩缩容需要提前申请资源,但 EVT-BiLSTM-HPA 算法相比于 BiLSTM-HPA 算法的资源占用情况在周期性较明显负载情况下平均能减少约 14% 的资源成本,在平缓负载情况下平均能减少约 22% 的资源成本,而在波动频繁且无明显周期负载的情况下平均也能节省 9% 的资源成本。实验结果表明,采用了极值理论的预测式扩缩容算法 EVT-BiLSTM-HPA 能够有效减少弹性扩缩容过程中的资源使用成本。

3.2 动态目标利用率扩缩容实验

在基于 EVT-BiLSTM-HPA 算法的基础上引入动态目标利用率后,形成了 DEBI-HPA 算法。实验中将动态目标利用率上下限分别设定为 30%~70%,容忍系数为 0.1,CPU 请求值为 5,最小与最大副本数分别为 1 和 50。图 13~图 15 展示了采用不同扩缩容策略时 Pod 数量的变化情况。

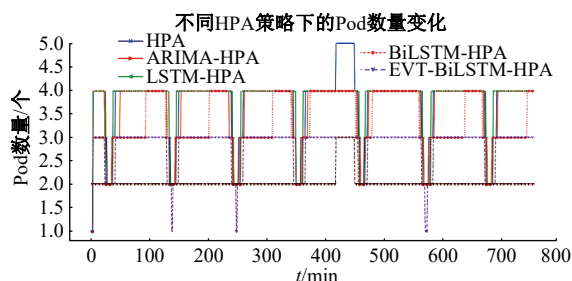


图 13 周期性负载情况下各算法 Pod 数量变化

Fig. 13 Changes in the number of Pods for each strategy under periodic load conditions

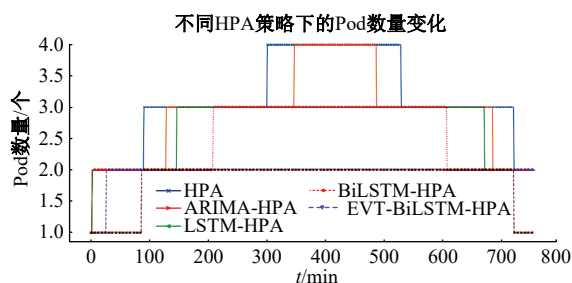


图 14 平缓负载情况下各算法 Pod 数量变化

Fig. 14 Changes in the number of Pods for each algorithm under periodic load conditions

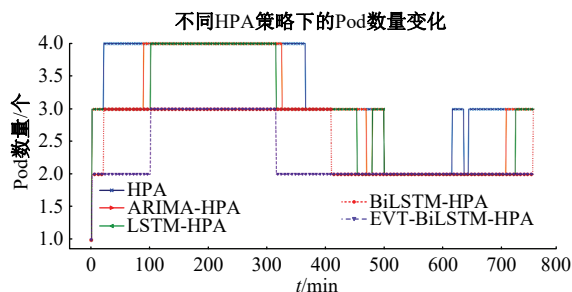


图 15 波动频繁且无明显周期负载情况 Pod 数量变化

Fig. 15 Changes in the number of Pods for each algorithm under conditions of frequent and non-periodic load fluctuations

结果表明,动态目标利用率有效改善了低负载向高负载过渡阶段的伸缩问题。在此阶段,如果目标利用率过高且无动态调整,容易出现扩容不及时、Pod 副本数过少,从而导致缓冲空间不足、服务质量下降。而引入动态目标利用率后,可在低负载时主动降低目标利用率,提前预留足够缓冲空间;当负载升高时,再提高目标利用率,使系统在更高负载下维持较好的资源利用率。

此外,与其他策略相比,采用动态目标利用率的 HPA 算法在 Pod 数变化方面更为平稳,减少了频繁扩缩容带来的服务不稳定和容器启动、删除的额外成本。即使在频繁波动且无明显周期性的负载场景下,DEBI-HPA 仍能展现出良好的平

稳定性和适应性。

由图 16 可以看出提出的 DEBI-HPA 算法相比于原生 HPA 来说,不仅能提前预测未来的负载变化,防止由于滞后性和震荡性,导致服务延迟或资源浪费的现象,而且能根据 Pod 的实际运行情况和历史数据来自动调整目标利用率,提高资源利用率,降低成本。由表 5 可知,该算法相比原生 HPA 策略在周期性负载情况下资源利用率提升约 30.79%,资源使用成本降低约 47.00%。由表 6 可知,在平缓负载情况下,该算法相比原生 HPA 策略资源利用率提升约 21.67%,资源使用成本降低约 41.30%;由表 7 可知,在波动频繁且无明显周期性负载情况下,该算法相比于原生 HPA 策略资源利用率提升约 18.6%,资源使用成本降低约 36.15%;而相比于其他常见的扩缩容优化算法,该算法也能将资源利用率提升约 18%,资源使用成本降低约 37%。

综合上述实验可知,基于动态目标利用率的弹性扩缩容算法在服务处于低负载时可以提供一定缓冲空间减少出现服务质量问题的可能性,而在服务处于高负载时能够一定程度上减少成本和

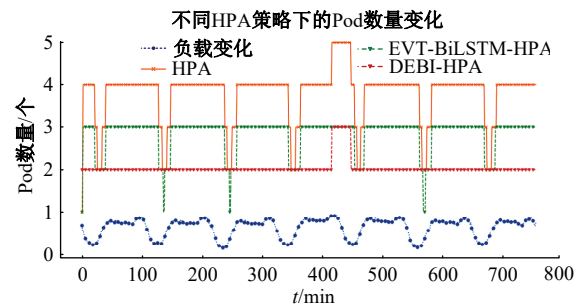


图 16 不同策略下 Pod 数变化及对应负载变化
Fig. 16 Changes in the number of Pods and corresponding load variations under different strategies

表 5 周期性负载情况不同扩缩容模型平均资源利用率和资源占用面积

模型名称	平均资源利用率/%	S
HPA	33.54	42 930
ARIMA-HPA	34.50	41 730
LSTM-HPA	35.64	39 302
BiLSTM-HPA	38.70	37 200
EVT-BiLSTM-HPA	45.19	31 860
DEBI-HPA	64.33	22 380

表 6 平缓负载情况不同扩缩容模型平均资源利用率和资源占用面积表

模型名称	平均资源利用率/%	S
HPA	30.81	35 520
ARIMA-HPA	33.07	33 090
LSTM-HPA	35.87	30 510
BiLSTM-HPA	38.23	28 620
EVT-BiLSTM-HPA	49.09	22 290
DEBI-HPA	52.48	20 850

表 7 波动频繁且无明显周期负载情况不同扩缩容模型平均资源利用率和资源占用面积表

模型名称	平均资源利用率/%	S
HPA	29.78	37 260
ARIMA-HPA	32.42	34 230
LSTM-HPA	33.21	33 420
BiLSTM-HPA	38.94	28 500
EVT-BiLSTM-HPA	42.91	25 860
DEBI-HPA	48.38	23 790

提高资源利用率,实验验证了此方法的可行性和有效性。

4 结束语

本文针对目前 HPA 响应式的扩缩容,提出了一种自适应性的预测式弹性扩缩容方法,即在负载不高时采用预测式扩缩容来应对未来可能持续增长的负载,而在负载已经处于极高的情况下,采用响应式的扩缩容,因为此时负载持续增加的可能性已经极低,并且在极高负载的情况下也会有数目较多 Pod 来均摊增长不大的负载增量,这样既维护了应用服务的质量,也最大限度地节省了资源。在自适应预测式弹性扩缩容方法基础上创新性设计并实现了基于动态目标利用率横向容器自动扩缩容方法。Pod 在副本较少时降低目标利用率以维持足够的缓冲空间,从而减少出现服务质量问题的可能性;在副本较多时提高目标利用率,从而减少了整体资源的占用并提高了资源利用率。本文提出的 DEBI-HPA 算法相比于原生 HPA 来说,不仅能提前预测未来的负载变化,防止由于滞后性和震荡性,导致服务延迟或资源浪

费,而且能根据 Pod 的实际运行情况和历史数据来自动调整目标利用率,同时在不同类型负载情况下资源利用率最高提升约 30%,资源使用成本最高降低约 47%,而相比于其他常见的扩缩容优化算法,该算法也能将资源利用率提升约 18%,资源使用成本降低约 37%。而针对目前研究中的预测模型训练时间长、训练成本高,且难以自适应复杂生产环境的问题,本文算法仅使用常见的预测模型 BiLSTM 也能达到比较好的扩缩容效果,降低了对预测模型的依赖性,为弹性扩缩容研究提供了一种新思路。目前,本文研究重点关注提高资源利用率和降低资源使用成本。在未来,可以对服务质量保证方法做进一步深入优化,并且动态目标利用率上下限仍需要人工设置,未来可在自动化设定目标利用率上下限上做进一步研究并优化动态目标利用率算法。

参考文献:

- [1] Kubernetes Community. kubernetes horizontal pod autoscaler[EB/OL]. [2024-05-06]. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale>.
- [2] Google Kubernetes Engine. Google kubernetes engine vertical pod autoscaler[EB/OL]. [2024-05-06]. <https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler>.
- [3] Google Kubernetes Engine. Google kubernetes engine multidimensional pod autoscaler[EB/OL]. [2024-05-06]. <https://cloud.google.com/kubernetes-engine/docs/how-to/multidimensional-pod-autoscaling>.
- [4] AL-HAIDARI F, Sqalli M, Salah K. Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources[C]//IEEE 5th International Conference on Cloud Computing Technology and Science. IEEE, 2013, 2: 256-261.
- [5] Iqbal W, Dailey M, Carrera D. SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud[C]//IEEE International Conference on Cloud Computing. Springer, Berlin, Heidelberg, 2009: 243-253.
- [6] Lim H C, Babu S, Chase J S, et al. Automated control in cloud computing: challenges and opportunities [C]//Proceedings of the 1st workshop on Automated control for datacenters and clouds, 2009: 13-18.
- [7] Lim H C, Babu S, Chase J S. Automated control for elastic storage[C]//Proceedings of the 7th international conference on Autonomic computing, New York: ACM, 2010: 1-10.
- [8] Hasan M Z, Magana E, Clemm A, et al. Integrated and autonomic cloud resource scaling[C]//IEEE network operations and management symposium. Piscataway: IEEE, , 2012: 1327-1334.
- [9] Baresi L, Guinea S, Leva A, et al. A discrete-time feedback controller for containerized cloud applications [C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY: ACM Press, 2016: 217-228.
- [10] 倪海峰. 基于 Kubernetes 的云平台 HPA 算法的优化与实现[D]. 上海: 上海交通大学电子信息与电气工程学院, 2018.
Ni Hai-feng. Design and implementation of container autoscaling algorithm platform based on Kubernetes [D]. Shanghai: School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 2018.
- [11] 田野. 面向容器的云计算资源自动伸缩问题研究 [D]. 上海: 上海交通大学电子信息与电气工程学院, 2018.
Tian Ye. Study on auto scaling framework for container in cloud computing[D]. Shanghai: School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 2018.
- [12] 张松霖. 基于 Kubernetes 的容器资源弹性伸缩策略研究[D]. 太原: 太原理工大学电气与动力工程学院, 2020.
Zhang Song-ling. Research on flexible scaling strategy of container resources based on Kubernetes[D]. Taiyuan: College of Electrical and Power Engineering, Taiyuan University of Technology, 2020.
- [13] 罗良杰. 基于负载预测的容器弹性伸缩技术研究 [D]. 武汉: 武汉纺织大学计算机与人工智能学院, 2021.
Luo Liang-jie. Research on container elastic scaling technology based on load prediction [D]. Wuhan: School of Computer Science and Artificial Intelligence, Wuhan Textile University, 2021.
- [14] 张树波, 吴天阳, 潘茂林, 等. 基于强化学习的预测式容器自动伸缩算法 A-SARSA[C]//第 17 届 IEEE 国际万维网服务会议, 北京, 2020: 423-430.
- [15] Imdoukh M, Ahmad I, Alfaiakawi M G. Machine learning-based auto-scaling for containerized applica-

- tions[J]. *Neural Computing and Applications*, 2020, 32(13): 9745-9760.
- [16] Barrett E, Howley E, Duggan J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud[J]. *Concurrency and Computation: Practice and Experience*, 2013, 25(12): 1656-1674.
- [17] Tesauro G, Jong N K, Das R, et al. A hybrid reinforcement learning approach to autonomic resource allocation[C]//*IEEE International Conference on Autonomic Computing*. Piscataway: IEEE, 2006: 65-73.
- [18] Rao J, Bu X P, Xu C Z, et al. Vconf: a reinforcement learning approach to virtual machines auto-configuration[C]//*Proceedings of the 6th international conference on Autonomic computing*. New York: ACM, 2009: 137-146.
- [19] 张磊. 深入剖析 Kubernetes[M]. 北京:人民邮电出版社, 2021.
- [20] Taherizadeh S, Grobelnik M. Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications[J]. *Advances in Engineering Software*, 2020, 140: 102734.
- [21] 徐虎博, 史东辉. 基于 ARIMA 和 LSTM 模型的建筑安全事故预测[J]. *软件工程*, 2023, 26(3): 9-14.
Xu Hu-Bo, Shi Dong-hui. Prediction of construction safety accidents based on ARIMA and LSTM models[J]. *Software Engineering*, 2023, 26(3): 9-14.
- [22] 吴忠强, 卢雪琴. 基于深度迁移学习和 LSTM 网络的微电网故障诊断[J]. *计量学报*, 2023, 44(4): 582-590.
Wu Zhong-qiang, Lu Xue-qin. Microgrid fault diagnosis based on deep transfer learning and LSTM network[J]. *Journal of Measurement*, 2023, 44(4): 582-590.
- [23] 李志兵, 肖健梅, 王锡淮. 基于多粒度 NRS 和改进 Bi-LSTM 的电力系统暂态稳定评估[J]. *电气工程学报*, 2023, 18(3): 232-241.
Li Zhi-bing, Xiao Jian-mei, Wang Xi-huai. Transient stability assessment of power system based on multi-granularity neighborhood rough set and improved bi-directional long-short-term memory network [J]. *Journal of Electrical Engineering*, 2023, 18(3): 232-241.
- [24] Hochreiter S, Schmidhuber J. LSTM can solve hard long time lag problems[J]. *Advances in neural information processing systems*, 1996, 9(8), 1735-1780.
- [25] Fréchet M. Sur la loi de probabilité de l'écart maximum[J]. *Annales Societatis Mathematicae Polonae*, 1927, 6(1): 93-116.
- [26] 史道济. 实用极值统计方法[M]. 天津: 天津科学技术出版社, 2006.
- [27] 高宇. 大数据系统异常检测与资源预估算法研究[D]. 哈尔滨: 哈尔滨工业大学计算机科学与技术学院, 2020.
Gao Yu. Research on methods of anomaly detection and resource estimation for big data systems [D]. Harbin: School of Computer Science and Technology, Harbin Institute of Technology, 2020.
- [28] Coles S, Bawa J, Trenner L, et al. An Introduction to Statistical Modeling of Extreme Values[M]. Springer, 2001, 208.
- [29] Guo J, Chang Z H, 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.
- [30] Pickands III J. Statistical inference using extreme order statistics[J]. *the Annals of Statistics*, 1975, 1(1): 119-131.
- [31] Grimshaw S D. Computing maximum likelihood estimates for the generalized Pareto distribution[J]. *Technometrics*, 1993, 35(2): 185-191.
- [32] Alibaba Cloud. Alibaba cluster trace program-Microservices trace 2022[EB/OL]. [2024-12-23]. <https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022>
- [33] Jananee M, Nimala K. Allocation of cloud resources based on prediction and performing auto-scaling of workload[C]//*International Conference on Artificial Intelligence and Knowledge Discovery in Concurrent Engineering*. Chennai, India: IEEE, 2023: 1-7.