基于强化学习的云资源混合式弹性伸缩算法

吴晓军[1]，张成[1]，原盛[1]，任晓春[2]，王玮[2]
（1.西安交通大学软件学院，710049，西安；
2.轨道交通工程信息化国家重点实验室（中铁一院），710043，西安）

摘要：为解决强化学习（RL）应用于云资源弹性伸缩问题时算法空间随应用规模变化而呈指数级变化、可伸缩性受限、算法训练时间长、收敛慢及设置水平伸缩动作空间时难以兼顾系统性能与稳定性等问题，提出了一种基于强化学习的分组云资源混合式伸缩算法（BGRL）。将应用实例进行逻辑分组，使算法空间固定，解决了算法空间爆炸及可伸缩性受限问题；采用并行学习，加快了学习速度，解决了算法收敛慢的问题；通过汇集多组的学习结果决定水平伸缩动作，解决了现有算法难以同时保证应用稳定性和资源调整及时性的问题；采用水平和垂直两个方向上的混合式伸缩，在保证应用能力范围的同时，解决了局部性能问题。通过重放实际应用数据集而产生的工作负载模式进行云应用仿真，结果表明：BGRL的应用资源量最贴合负载变化，资源利用率最高，稳定在80%左右；在消耗的资源量和违反服务质量请求的百分比方面，比其他算法分别减少了15%～20%和0.1%～3.26%。

关键词：云资源；混合式弹性伸缩；强化学习；逻辑分组；局部性能问题
中图分类号：TP181 **文献标志码**：A
DOI：10.7652/xjtuxb202201016 **文章编号**：0253-987X(2022)01-0142-09

OSID码

Blended Elastic Scaling Method for Cloud Resources Following Reinforcement Learning

WU Xiaojun[1], ZHANG Cheng[1], YUAN Sheng[1], REN Xiaochun[2], WANG Wei[2]
(1. School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049, China;
2. State Key Laboratory of Rail Transit Engineering Informatization (FSDI), Xi'an 710043, China)

Abstract: During applying reinforcement learning (RL) to the elastic scaling of cloud resources, we face these problems: the model space changing with application scale exponentially, limited scalability, long training time, slow convergence and difficulty in balancing the performance and stability of system as setting horizontal scaling action. A blended grouped scaling method of cloud resources based on reinforcement learning (BGRL) is proposed. The application examples are logically grouped to make the size of the model space fixed, which solves the problem of model space explosion versus limited algorithm scalability. The parallel learning method is adopted to speed up the model learning rate and solve the problem of slow algorithm convergence. By pooling the learning results of multiple groups to determine the horizontal scaling action, it solves the problem in the existing methods that are difficult to ensure the stability of application and the timeliness of resource adjustment at the same time. Blended expansion and contraction in both horizontal and vertical directions solve local performance problems while ensuring the scope of

收稿日期：2021-07-02。 作者简介：吴晓军（1968—），男，副教授；任晓春（通信作者），男，教授级高级工程师。
网络出版时间：2021-08-10 网络出版地址：http://kns.cnki.net/kcms/detail/61.1069.T.20210810.1039.002.html

application capability. The cloud application simulation is performed by replaying the workload pattern generated by the actual application data set. The results show that the amount of application resources of BGRL is the most suitable for load changes, and the resource utilization rate reaches the highest, which remains stable at about 80%. Compared with the other methods, the percentage of requests violating the quality of service (QoS) is reduced by 15%－20% and 0.1%－3.26%, respectively.

Keywords: cloud resources; blended elastic scaling; reinforcement learning; logical grouping; local performance issues

近年来，云计算行业快速崛起。云计算的虚拟化概念和自动伸缩功能使计算或应用外包到云成为一种常见的范式[1]，以按需购买和按需付费模式使用资源可以节约资源和成本。但是，云环境的高度灵活性使资源伸缩问题变得更加复杂[2]。云环境中的工作负载是不确定的，这一点在涉及面向用户的应用程序中尤为突出[3]。

自动伸缩是一种调整应用程序资源量以满足不断变化的工作负载需求，同时最小化成本或资源的技术[4]。目前，已经提出了许多自动伸缩方法，这些方法可以分为响应式和预测式两种[5]。响应式方法主要包括基于阈值的策略、排队论和控制论等，预测式方法则包含时间序列分析和强化学习方法。由于提前做出了伸缩决策，预测式方法的伸缩及时性比响应式要好。时间序列分析通常包含两个阶段，第一阶段是工作量预测，第二阶段是根据预测的工作量做出扩展决策[6]。然而，许多时间序列分析方法都是基于阈值来做出伸缩决策，存在阈值设置过程复杂并且难以确定达到阈值后扩展或收缩的实例数的问题。基于强化学习(RL)的伸缩方法具有自适应性和鲁棒性[7]，不需要任何先验知识，可以确保当工作负载动态变化时应用软件的资源利用处于相对稳定的状态，因此 RL 是云环境中进行自动伸缩的一种有前景的方法。

目前的工作大部分都集中于在水平伸缩领域进行[8-10]。但是，水平伸缩通常面临动作空间难以准确设置的问题。在水平伸缩中，如果将 RL 的每个状态的动作设置为(－1，0，1)，那么面对负载急剧增加情形需要多次调整才能有效减少违反服务质量请求(QoS)，影响资源调度的及时性和动态性，还影响用户体验；如果将动作空间范围设置得过大，则会导致算法收敛慢，而且在强化学习的探索阶段随机选择的动作可能会造成意料之外的实例数大范围变化而使系统不稳定。此外，尽管水平伸缩是管理云环境中资源的常用方法，但是也存在一些水平伸缩无法处理的情况[11]。例如，某个实例上堆积了远超其处理能力的任务时，传统的水平伸缩方法虽然能检测出应用违反 QoS 数上升，并且会添加实例，但并不能有效解决问题，反而会产生额外的成本。

事实上，应用可在垂直和水平两个方向上进行伸缩。其中，垂直伸缩是指更改单个实例的计算资源量。云环境中的应用程序很容易出现许多影响 CPU 和内存等资源的本地性能问题，通过这种伸缩，可以调节单个应用实例的性能。水平伸缩指增加或减少实例。两种方法各有利弊：垂直伸缩通常较快且平滑，可以处理单个实例出现问题的情况，但能力范围有限；水平伸缩通过增减应用实例的数量来调整应用能力，通常较慢且粗粒度，对环境变化的反应不及时，而且每个实例都需要消耗一定的基础资源来维持自身运转，但可以在较大范围内伸缩，保证应用的能力满足需要。良好的伸缩策略应结合垂直伸缩和水平伸缩，然而目前大多数工作忽略了垂直伸缩。少数将 RL 应用于垂直伸缩的工作也存在状态空间和动作空间太大、算法可伸缩性差等问题。例如，Kardani-Moghaddam 等将强化学习应用于垂直伸缩，将状态空间和动作空间构建为所有实例的状态和动作的笛卡尔积，算法复杂度随着实例数增加而指数上升，导致这项工作算法训练时间长且在算法可伸缩性方面有限，原文献中也限制了最大实例数[12]。

针对这些问题，本文提出了一个分组的基于强化学习的混合式自动伸缩算法(BGRL)，强化学习算法选择资源自动伸缩问题中最常用的 Q 学习算法[13]。该算法将水平伸缩和垂直伸缩相结合，通过将应用实例进行逻辑分组的方式使强化学习状态空间和动作空间固定，不随实例数变化而变化，解决了现有的 RL 算法空间爆炸及算法可伸缩性受限问题。通过汇集所有组的意见决定水平伸缩动作，解决了现有解决方案中设置水平伸缩动作空间时所面临的性能与稳定性的取舍问题。

1 相关工作

资源伸缩决策通常是对系统性能下降的响应。但是,应用程序的特性、工作负载变化、资源共享冲突等因素都可能影响性能。因此,云环境中资源伸缩方案需要具有较强的学习能力和自适应性。

资源的自动伸缩问题通常基于监视(M)、分析(A)、计划(P)和执行(W)这种 MAPE 架构来解决。遵循这种架构,Aslanpour 等提出了一种成本感知型自动扩展框架,重点关注执行级别的可能改进[14]。该框架使用基于阈值的规则来更改系统中虚拟机的数量。尽管基于阈值的决策在解释和实现方面既简单又方便,但是缺乏适应环境变化的灵活性使得该类解决方案不是最佳选择。基于阈值的扩展策略是一种响应策略,存在资源调度不及时的问题。此外,阈值设置应基于大量数据分析,并且每种情况的阈值未必适用于其他情况。在响应式方法中,虽然基于排队论的方法应用于受工作负载变化影响较小的系统时可以看作预测算法,但在其他情况下,它仍然是响应式算法[15]。更重要的是,基于排队论的模型是系统的近似估计模型,所做出的决策可能不适合系统的实际情况。控制论方法主要用于制定响应式扩展策略,并且仅适用于缓慢变化的工作负载。当工作负载突然变化时,其模型参数无法在短时间内更新为合理值[16]。

为了使框架实现更高的适应性,强化学习是一种不错的选择。但是,基于强化学习的解决方案遭受状态和动作空间爆炸、缓慢收敛的困扰。Ghobaei-Arani 等利用 Q 学习作为 MAPE 架构中计划阶段的一部分。这些决策是马尔可夫决策表和 Q 表的组合,用于决定系统中虚拟机的添加和删除[17]。Arabnejad 等引入了 Q 学习和 SARSA 学习的模糊版本,使用监控指标上的模糊规则作为解决方案,以减少状态数,从而减小 Q 表规模[18]。Horovitz 等提出了一种在容器调度中使用 RL 动态改变实例伸缩阈值的算法,该算法根据实例数确定状态,与传统的 RL 算法相比,大大减小了状态空间的规模[19]。这些工作使用基于阈值或基于规则的技术来减少状态数量,虽然可以有效减小 RL 算法规模,但是难免会使算法准确性受损。本文采用的分组建模方式可以使算法的状态空间固定,在不受应用规模限制的同时可以收集所有实例的状态,之后再做出决策。

将 RL 应用于资源伸缩时,还会遭遇动作空间难以准确设置的问题。为此,Yang 等提出了一种基于模型的 RL 算法,以便在微服务场景中使用系统知识[20]。但是,该算法的每个状态相对应的动作数是固定的。因此,当处理流量急剧变化时,该算法不能解决重复伸缩的问题,导致违反服务级别协议或浪费资源。Horovitz 等提出的算法通过动态动作来寻找每个状态的最佳动作和最佳动作附近的可行动作,这样可以将动作空间规模固定在一个较小值[19]。但是,此算法依赖于 Q 函数的单峰特性,且动作空间改变时,新动作的 Q 设置存在问题。本文的水平伸缩动作是由多个智能体组的伸缩动作汇集而成的,这样既保证了资源伸缩的及时性和准确性,又避免了重复伸缩的问题。

最后,目前的研究工作很少考虑垂直和水平伸缩的组合,而且受限于算法规模,现有的同时考虑垂直和水平伸缩的 RL 算法通常会在算法伸缩准确性方面做出让步。例如,Rossi 等提出的利用 RL 的基于容器应用的水平和垂直伸缩算法,为了避免算法规模爆炸,规定应用的所有实例在资源量上保持一致,即垂直伸缩动作是全局性的,所有实例同步执行,这样的规定显然忽略了不同实例可能需要不同资源量的情况[21]。本文结合了垂直和水平两个方向上的伸缩,并把垂直动作细分到每个实例,可以更细粒度地调节应用性能。

2 基于强化学习的混合式伸缩算法

针对相关工作中提到的将强化学习应用在云资源混合式伸缩问题时面临的算法空间爆炸、收敛速度慢、动作空间难以准确设置等问题,本文提出了基于强化学习的分组的混合式伸缩算法。

2.1 BGRL 算法逻辑结构

图 1 描绘了本文 BGRL 算法的逻辑结构。用户将请求发送到应用的负载均衡器,负载均衡器将这些请求分配到现有的应用实例中。每个实例都包含了本地监测模块和本地伸缩模块。本地监测模块负责监视并更新该实例的状态信息,在本文中指 CPU 利用率和请求的平均响应时间。每个本地伸缩模块执行垂直伸缩动作来更改该实例资源。在应用层面由水平伸缩模块执行水平方向上增减实例的操作。

逻辑分组是指在逻辑上本文将应用程序实例分为若干个智能体组和一个阈值组,即如果目前应用包含 N 个实例,则将 N 表示为

$$N = nk + m \tag{1}$$

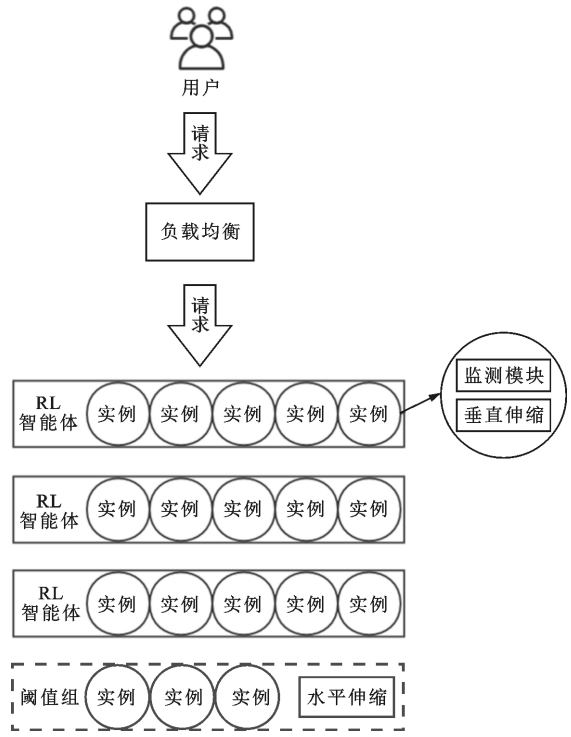

图 1　BGRL 算法逻辑结构

Fig. 1　Logical structure of BGRL model

式中：k 表示智能体组容量；n 表示智能体组数量；m 表示阈值组中的实例数。

图 2 显示了 BGRL 算法自动伸缩的主要过程。本文将应用运行过程划分为若干个时间间隔，在每个时间间隔末进行混合式伸缩。每个实例通过监测模块定期收集该实例状态，在每个伸缩过程开始时，强化学习模块收集所有实例最新状态，在逻辑上将所有实例划分为若干智能体组和一个阈值组，每个智能体组代表一个强化学习智能体，智能体组的状态由组内所有实例的状态联合组成。图 2 中，$k=5$，$n=2$，$m=3$。

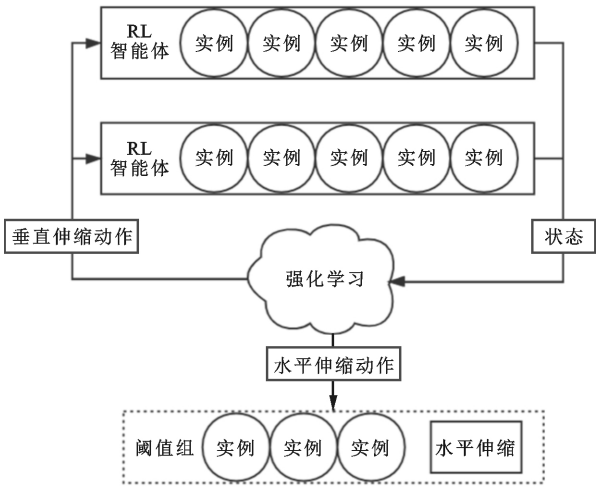

图 2　混合伸缩过程

Fig. 2　Blended scaling process

所有智能体组共同训练一个强化学习算法，通过并行学习，可以加快算法收敛速度。本文中的 RL 算法动作空间包括组内各实例垂直伸缩动作和应用级别的水平伸缩动作，垂直伸缩动作发回给智能体组，由组内各实例中垂直伸缩模块执行。水平伸缩动作由所有智能体组训练得到的水平动作汇集发送给阈值组，由水平伸缩模块在阈值组内进行增减实例的操作。

如果阈值组实例数到达 k，则该阈值组升为智能体组，并开启一个新的阈值组。如果阈值组中无实例且需要删减实例，则挑选一个智能体组降为阈值组。最后，阈值组中的实例进行基于阈值的垂直伸缩。如果系统实例数不足以构成一个智能体组，则根据系统状态进行基于阈值的水平伸缩。

BGRL 算法流程如下。

输入：Q 表，ε，σ，ε 的最小值 $\varepsilon_{\min}$，智能体组容量 k，所有实例最新状态信息 $\boldsymbol{S}_{\text{total}}$。

输出：更新后的 Q 表和 ε。

根据 $\boldsymbol{S}_{\text{total}}$ 并行更新 Q 表；
根据 k 将所有实例分为 n 个智能体组和一个阈值组；
根据阈值规则对阈值组实例进行垂直伸缩；
if $n>0$ then
　初始化水平伸缩变量 $H=0$；
　for $i=1$ to n do
　　计算第 i 个智能体组的状态 $\boldsymbol{S}_i$；
　　产生一个 0 到 1 的随机数 r；
　　if $r<\varepsilon$ then
　　　随机选择伸缩动作；
　　else
　　　选择 Q 表中状态 $\boldsymbol{S}_i$ 对应的最佳动作；
　　end if
　　将垂直动作发回第 i 个智能体组执行；
　　将水平伸缩动作计入 H；
　end for
　if $H>0$ then
　　增加 H 个应用实例；
　else if $H<0$ then
　　减少 $-H$ 个应用实例；
　end if
else
　根据阈值规则对阈值组进行水平伸缩；
end if

25 if $\varepsilon>\varepsilon_{\min}$ then

26 $\varepsilon=\varepsilon-\sigma$;

27 end if

28 输出更新后的 Q 表和 ε。

2.2 Q 学习设置

Q 学习是针对连续时间半马尔可夫决策问题的一种 RL，它是自动伸缩中最常用的算法，通常由以下要素描述。

(1)探索率 ε。强化学习类似于一个试错的学习，需要从智能体与环境的交互中发现优的策略，同时又不至于在试错的过程中丢失太多的奖励。本文使用在探索和利用之间进行权衡的标准策略——ε 贪婪策略。其中，探索(发现新的路径)和利用(选择当前最佳)就是进行决策时需要平衡的两个方面。所以，当选择动作时，会以一定的概率 ε 选择随机动作，进行那些未知领域的探索。

(2)状态 $\boldsymbol{S}$。状态 $\boldsymbol{S}$ 表示观察到的环境状态。形成状态空间的候选因素很多，例如，响应时间、CPU 利用率、内存利用率、I/O 利用率等。因为 CPU 利用率可以最有效地反映应用程序的资源利用率，并且响应时间可以最直观地反映用户体验，所以本文选择 CPU 利用率和响应时间来确定状态空间。在本文中每一个智能体组拥有 k 个实例，用 $\boldsymbol{s}_i=[u_i,r_i]$ 表示实例 i 的状态，其中，u_i 是实例 i 的 CPU 利用率，r_i 表示实例 i 的最新平均响应时间。相应地，状态 $\boldsymbol{S}$ 表示为

$$\boldsymbol{S}=[\boldsymbol{s}_1,\boldsymbol{s}_2,\cdots,\boldsymbol{s}_k] \tag{2}$$

本文将 CPU 利用率离散化为 c 个等级，平均响应时间离散化为 p 个等级，则状态空间规模为 $(cp)^k$。

(3)动作 $\boldsymbol{A}$。动作 $\boldsymbol{A}$ 指根据观察到的环境状态所采取的策略。本文的动作空间包含水平动作和垂直动作。水平动作 a_{h} 取值范围为(−1,0,1)，表示增减的实例数。每个实例的垂直动作 $a_{\mathrm{v}i}$ 取值范围也是(−1,0,1)，表示该实例增减的 CPU 资源量。本文假设每个 CPU 资源处理能力相同且每个实例具有一个共同的 CPU 上限。每次学习得到动作空间 $\boldsymbol{A}$ 表示为

$$\boldsymbol{A}=[a_{\mathrm{h}},a_{\mathrm{v}1},a_{\mathrm{v}2},\cdots,a_{\mathrm{v}k}] \tag{3}$$

不难算出，动作空间大小为 3^{k+1}。相应地，本算法 Q 表规模为 $Q_{\mathrm{size}}=(cp)^k3^{k+1}$。

通过逻辑分组，本文提出算法中的状态和动作空间规模不再随实例数量变化而变化，解决了传统基于 RL 的资源伸缩算法的动作空间和状态空间维度爆炸问题。并且，最终执行的水平动作 a_{h} 是多个智能体组学习结果的汇集，最终的水平动作取值范围为

$$-n\leqslant a_{\mathrm{h}}\leqslant n \tag{4}$$

当负载急剧增加时，最多可以增加 n 个实例，可以保证达到系统性能要求的速度，而且也避免了设置水平伸缩动作空间时面临的性能与稳定性的取舍问题。

(4)奖励 R。基于 Q 学习的资源自动伸缩算法的最终目的是保证服务 QoS 和资源利用率。所以，本文中 Q 学习的奖励 R 由 QoS 和 CPU 利用率两个部分组成。

QoS 代表了用户的要求及满意度。当系统负载上升造成服务资源不足时，会造成服务 QoS 下降，用户体验变差。通常 QoS 有可用性、响应时间、吞吐量、可靠性等度量指标。本文将智能体组内平均响应时间 r 作为 QoS 指标，响应时间定义为从任务提交到完成的等待时间。

尽管当 CPU 利用率低时可以为用户带来较高的 QoS，但是也意味着资源浪费，增加了应用运行成本。因此，在 R 中应考虑资源利用率，本文中即指智能体组内 CPU 利用率 u，该参数可以帮助决策朝着提高资源利用率方向发展。

为了在保证 QoS 的情况下尽可能提高 CPU 利用率，本文设定奖励函数 R 为

$$R(r,u)=\begin{cases}-1, & r>2r_{\mathrm{m}}\\ \dfrac{1}{2-u}, & r<r_{\mathrm{m}}\\ \dfrac{\exp(-(r-r_{\mathrm{m}})/r_{\mathrm{m}})^2}{2-u}, & \text{otherwise}\end{cases} \tag{5}$$

式中 r_{m} 代表平均响应时间的最大可接受值。

3 实验对比与分析

3.1 工作负载构建与 RL 参数设置

实验环境被仿真为一个云计算应用，一开始具有 60 个应用实例。仿真中使用的应用程序工作负载如图 3 所示，这种工作负载模式是通过重放由 Gulisano 等收集的实际应用程序数据集而产生的[22]。工作负载以请求的方式添加到应用程序的负载均衡器，负载均衡器按照各实例的 CPU 资源量以相应的概率将请求发送到实例的任务队列中，当实例空闲则会从任务队列中取出任务并完成。另外，为了模拟实际情况中可能会出现的单个实例的

局部问题，本文随机选择若干个时间间隔，使某个实例堆积过量任务。

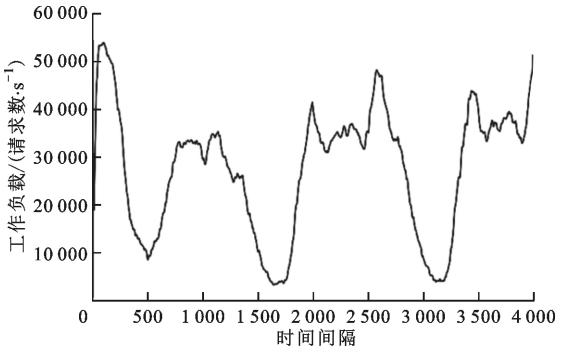

图 3　仿真中使用的应用程序工作负载

Fig. 3　Application workload used in simulation

本文采用恒定的学习率 $\alpha=0.05$ 和折现因子 $\delta=0.9$，探索率 ε 被初始化为 1，并从 1 减小到 0.1 以便在使用 ε 贪婪策略学习的初始迭代中提供更高的探索能力，算法训练次数为 $10Q_{\text{size}}$。每个实例的最大 CPU 资源量设置为 7，实例默认 CPU 资源量为 4，r_{m} 设置为 1 s。阈值组的伸缩规则为当实例 CPU 利用率超过 80%时增加一个 CPU，当利用率低于 40%时减少一个 CPU。当应用实例数小于 k 时，即无法组成一个智能体组时，则根据应用的全局 CPU 利用率以同样的阈值规则做水平伸缩。

3.2　评价指标

为全面衡量算法性能，本文除平均 CPU 利用率和平均响应时间外，增加了违反 QoS 的百分比和平均 CPU 资源量两个指标。评价指标体现了应用性能和运行成本，这是使用云或边缘云时需要衡量的重点[23-24]。本文将响应时间超过 2 s 的请求视为违反 QoS 的请求，违反 QoS 百分比即违反 QoS 的请求数与总请求数的比值。

此外，为了综合评估算法，本文从文献[21]中引入归一化成本 c

$$c(\boldsymbol{S},\boldsymbol{A})=w_{\text{adp}}C_{\text{adp}}+w_{\text{pref}}C_{\text{perf}}+w_{\text{res}}C_{\text{res}} \quad (6)$$

式中：C_{adp} 为调整成本，是垂直伸缩操作引入的恒定代价；C_{perf} 为性能损失成本，是当请求违反 QoS 时支付的性能损失；C_{res} 为运行应用程序的资源单位成本，与分配的 CPU 资源量成正比。w_{adp}、w_{pref}、w_{res} 分别是不同成本的非负权重，$w_{\text{adp}}+w_{\text{pref}}+w_{\text{res}}=1$。在式(7)的原文献中存在 3 种权重分配方式：①$w_{\text{adp}}=0.01$，$w_{\text{pref}}=0.90$，$w_{\text{res}}=0.09$；②$w_{\text{adp}}=0.01$，$w_{\text{pref}}=0.09$，$w_{\text{res}}=0.90$；③$w_{\text{adp}}=0.33$，$w_{\text{pref}}=0.33$，$w_{\text{res}}=0.33$。

比较原文献的实验结果后，本文选择第 3 种权重分配方式，因为该方式平衡了 3 种部署目标，在原文献中表现最佳。

3.3　对比算法

为了评估 BGRL 的性能，将 BGRL 与基于阈值的水平伸缩算法 THS、基于强化学习的分组的水平伸缩算法 HGRL 以及近期的文献[21]中的算法 HVSRL 做比较。HVSRL 是根据全局状态来进行水平伸缩和垂直伸缩的强化学习算法，其中垂直伸缩是所有实例同步伸缩，并未细分到单个实例。在 THS 算法中本文设置 CPU 利用率上下限分别为 80%和 40%，当应用程序的全局 CPU 利用率超过上限时则增加一个实例，当 CPU 利用率低于下限时则删除一个实例。通过与最常见的基于阈值的伸缩算法比较，可以评估 BGRL 的常规性能。HGRL 是 BGRL 删除垂直伸缩功能的退化算法，通过与该算法比较，可以评估垂直伸缩对于应用性能的提升，以及凸显出现局部性能问题时垂直伸缩的作用。

3.4　实验结果与分析

考虑到算法准确性与稳定性之间的权衡，需要选择适当的 k。由式(1)可以看出 k 决定了如何进行逻辑分组：k 较大则算法状态空间和动作空间大，学习时并行度小，算法训练时间长；k 较小则每次训练包含的应用信息少，算法准确性会下降。而且，水平动作 a_{h} 的上下限 n 由 k 和实例数 N 决定。较小的 k 可能会使系统不稳定，极端情况如当 $k=1$ 时，每个实例自成一个智能体组，可能某个时间段内工作负载少，所有组学习到的水平动作都是减少一个实例，则下一个时间段内应用将无实例，较大的 k 则会使系统在负载急剧变化时来不及做出适当调整，产生大量违反 QoS 的请求。因此，本文首先比较了不同 k 时的算法效果，结果如表 1 所示。

表 1　不同 k 时的算法对比

Table 1　Comparison of model behaviors with different k

k	违反 QoS 百分比/%	平均 CPU 利用率/%	平均 CPU 资源量	平均响应时间/s
2	3.01	71.33	390.61	0.46
3	0.51	82.14	338.23	0.50
4	0.77	84.43	325.94	0.52
5	1.36	79.18	354.47	0.57

从表 1 中可以看出，当 $k=3$，4 时算法表现较好。$k=3$ 时算法违反 QoS 百分比最低，仅为 0.51%，CPU 利用率较高，超过了 82%，使用 CPU 资源量较少；$k=4$ 时算法的 CPU 利用率最高，达到

了 84.43%，使用的 CPU 资源量最少。考虑到实际场景中减少违反 QoS 百分比更为重要，本文选择 $k=3$。

$k=3$ 时 BGRL 和其他 3 种算法的实验结果如图 4～7 所示。

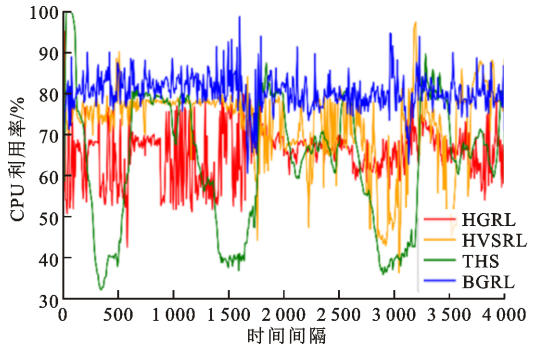

图 4 THGRL、HVSRL、THS、BGRL 的 CPU 利用率

Fig. 4 CPU utilization of THGRL, HVSRL, THS, and BGRL

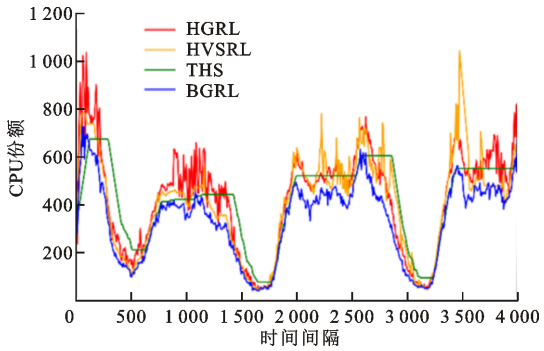

图 5 HGRL、HVSRL、THS、BGRL 的 CPU 资源量

Fig. 5 CPU resources of HGRL, HVSRL, THS, and BGRL

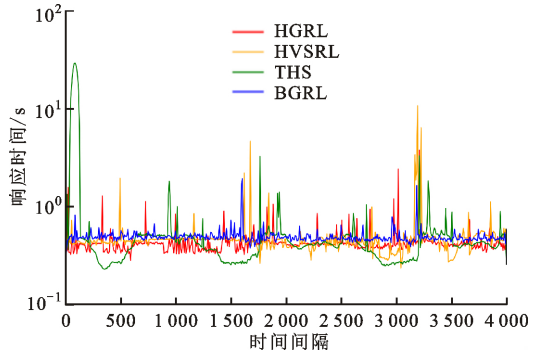

图 6 HGRL、HVSRL、THS、BGRL 的平均响应时间

Fig. 6 Average response time of HGRL, HVSRL, THS, and BGRL

从图 4 和图 5 可以看出，当面对工作负载急剧上升的情况时，THS 需要多个时间间隔才能增加足够的实例以满足应用需求，在此期间 THS 的 CPU 利用率将达到 100%，必然产生大量违反 QoS 的请求。THS 只在 CPU 利用率超过阈值范围时才进行伸缩。从图 5 可以看出，THS 存在许多资源量连续不变的时间间隔，可见 THS 对负载的变化不敏感，而且 THS 伸缩变化慢，所以 THS 的 CPU 利用率曲线和工作负载变化的曲线相似。

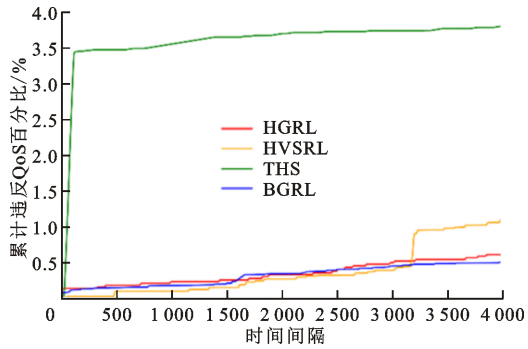

图 7 HGRL、HVSRL、THS、BGRL 的累积违反 QoS 百分比

Fig. 7 Cumulative QoS violation percentage of HGRL, HVSRL, THS, and BGRL

与此不同的是，3 种强化学习算法都通过自适应伸缩使自身的资源量与工作负载相对应，所以图 5 中 3 种强化学习算法的 CPU 资源量曲线与工作负载曲线相似，图 4 中 3 种强化学习算法的 CPU 利用率也较为稳定。但是，HSVRL 的状态空间是整个应用的全局状态，根据全局状态选择全局动作，其垂直伸缩动作是所有实例同步垂直伸缩，所以 HSVRL 的曲线会出现较大波动，如图 4 和图 5 中时间间隔在3 500左右时 HSVRL 的变化。与 BGRL 相比，HGRL 只能进行水平伸缩，无法对实例进行细粒度调节所以其资源量变化较大。BGRL 使用了虚拟分组，可以选择恰当的水平伸缩动作以及具体到单个实例的垂直伸缩动作，所以表现最优，使用的 CPU 资源量最少最贴合负载变化，CPU 利用率最高最稳定。

违反 QoS 通常由两种情况造成：①负载急剧增加时应用资源不够；②没有解决实验中模拟的个别实例出现的局部问题。从图 7 可以看出，面对一开始负载急剧增加的情况，THS 由于扩展不及时而产生了大量的违反 QoS 的请求，图 6 中相应位置也可以看到此时 THS 的平均响应时间很长，而 3 个强化学习算法都解决了该问题。另外，THS 和 HGRL 只能进行水平伸缩而无法有效处理局部问题，HVSRL 是根据全局状态对所有实例同步垂直伸缩，所以并不总是能解决局部问题，反而可能因为不恰当的动作产生更多违反 QoS 的请求。本文提

出的 BGRL 算法收集每个实例的信息并将垂直伸缩动作细分到单个实例，可以快速地解决局部问题，表现最好，平均响应时间最稳定，违反 QoS 百分比最小。

表 2 展示了 4 种算法在实验过程中的违反 QoS 百分比、平均 CPU 利用率、平均 CPU 资源量、平均响应时间以及归一化成本 c 等。

表 2　THS、HGRL、HVSRL 和 BGRL 的综合对比

Table 2 Comprehensive comparison of THS, HGRL, HVSRL, and BGRL

算法	违反 QoS 百分比/%	平均 CPU 利用率/%	平均 CPU 资源量	平均响应时间/s	c
THS	3.78	62.73	429.65	0.98	0.80
HGRL	0.62	65.05	423.41	0.43	0.61
HVSRL	1.09	70.38	397.41	0.47	0.71
BGRL	0.52	80.14	338.23	0.49	0.57

从表 2 可以看出，与 THS 相比，3 个强化学习算法在违反 QoS 百分比和平均响应时间方面都有明显改善，这主要是因为在实验一开始负载急剧增加时，强化学习算法处理效果较好。BGRL 通过细粒度的垂直伸缩使其在违反 QoS 百分比、平均 CPU 利用率、平均 CPU 资源量等方面表现最优，分别为 0.52%，80.14%以及 338.23。但是，在平均响应时间方面不如 HGRL 和 HVSRL，这是因为 HGRL 和 HVSRL 总是使用了更多的 CPU 资源，正常情况下请求能更快被处理。在归一化成本 c 方面，虽然 HVSRL 与 HGRL 在 CPU 资源量和违反 QoS 百分比上各有优势，但是前者的 c 比后者要大 0.1，这主要是因为垂直伸缩需要付出额外成本。BGRL 虽然也包含垂直伸缩，但在违反 QoS 百分比和使用的资源量上改善明显，所以 BGRL 的 c 最小。

4　结　论

本文提出了一种解决云中资源自动伸缩问题的算法 BGRL，该算法结合了水平和垂直两个方向上伸缩的优势。BGRL 使用了逻辑分组，将应用实例分为了若干个智能体组和一个阈值组，使伸缩问题的规模固定，算法可伸缩性不再受限，多个智能体组共同学习使应用总是能采取恰当的混合式伸缩。通过仿真实验，证明了本文混合式伸缩算法的明显优势。实验结果显示，与现有算法相比，BGRL 使应用资源量更贴合负载变化，系统稳定性更高，在保证 QoS 和减少使用的资源上表现优异。

在未来的工作中，将考虑如何预测工作负载变化，并将其作为强化学习的状态之一，以实现更符合应用需求的自动伸缩。

参考文献：

[1] ZHANG Jian, YANG Yang, WANG Zhibo. Outsourcing large-scale systems of linear matrix equations in cloud computing [C] // Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). Piscataway, NJ, USA: IEEE, 2016: 438-447.

[2] 龚强. 云计算关键技术之弹性伸缩控制技术认知研究 [J]. 信息技术，2014，38(1)：1-2，6.
GONG Qiang. Research on the cognition of elastic retractable control technology: one of the key technology of cloud computing [J]. Information Technology, 2014, 38(1): 1-2, 6.

[3] DEAN J, BARROSO L A. The tail at scale [J]. Communications of the ACM, 2013, 56(2): 74-80.

[4] 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, 2018, 11(2): 430-447.

[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] LORIDO-BOTRAN T, MIGUEL-ALONSO J, LOZANO J A. A review of auto-scaling techniques for elastic applications in cloud environments [J]. Journal of Grid Computing, 2014, 12(4): 559-592.

[7] 李茹杨，彭慧民，李仁刚，等. 强化学习算法与应用综述 [J]. 计算机系统应用，2020，29(12)：13-25.
LI Ruyang, PENG Huimin, LI Rengang, et al. Overview on algorithms and applications for reinforcement learning [J]. Computer Systems & Applications, 2020, 29(12): 13-25.

[8] 易鸣. 基于微服务架构应用平台的资源调度优化研究 [D]. 济南：山东大学，2020：9-11.

[9] 王天泽. 基于灰色模型的云资源动态伸缩功能研究 [J]. 软件导刊，2018，17(4)：131-134.
WANG Tianze. Research on cloud resources auto-scaling based on grey model [J]. Software Guide, 2018, 17(4): 131-134.

[10] 尚小东，张煜，郭成昊. 基于作战任务优先级的容器云弹性伸缩系统 [J]. 指挥信息系统与技术，2020，11(3)：36-43.
SHANG Xiaodong, ZHANG Yu, GUO Chenghao.

Container cloud elastic scaling system based on mission priority [J]. Command Information System and Technology, 2020, 11(3): 36-43.

[11] SHEKHAR S, ABDEL-AZIZ H, BHATTACHARJEE A, et al. Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications [C]// Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). Piscataway, NJ, USA: IEEE, 2018: 82-89.

[12] KARDANI-MOGHADDAM S, BUYYA R, RAMAMOHANARAO K. ADRL: a hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(3): 514-526.

[13] 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.

[14] ASLANPOUR M S, GHOBAEI-ARANI M, NADJARAN TOOSI A. Auto-scaling web applications in clouds: a cost-aware approach [J]. Journal of Network and Computer Applications, 2017, 95: 26-41.

[15] ALI-ELDIN A, TORDSSON J, ELMROTH E. An adaptive hybrid elasticity controller for cloud infrastructures [C]// Proceedings of the 2012 IEEE Network Operations and Management Symposium. Piscataway, NJ, USA: IEEE, 2012: 204-212.

[16] PATIKIRIKORALA T, COLMAN A. Feedback controllers in the cloud [C]// Proceedings of the 2010 APSEC. Sydney, Australia: APSEC, 2010: 39.

[17] GHOBAEI-ARANI M, JABBEHDARI S, POURMINA M A. An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach [J]. Future Generation Computer Systems, 2018, 78: 191-210.

[18] ARABNEJAD H, PAHL C, JAMSHIDI P, et al. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling [C]// Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). Piscataway, NJ, USA: IEEE, 2017: 64-73.

[19] HOROVITZ S, ARIAN Y. Efficient cloud auto-scaling with SLA objective using *Q*-learning [C]// Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud). Piscataway, NJ, USA: IEEE, 2018: 85-92.

[20] YANG Zhe, NGUYEN P, JIN Haiming, et al. MIRAS: model-based reinforcement learning for microservice resource allocation over scientific workflows [C]// Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). Piscataway, NJ, USA: IEEE, 2019: 122-132.

[21] ROSSI F, NARDELLI M, CARDELLINI V. Horizontal and vertical scaling of container-based applications using reinforcement learning [C]// 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). Piscataway, NJ, USA: IEEE, 2019: 329-338.

[22] GULISANO V, JERZAK Z, VOULGARIS S, et al. The DEBS 2016 grand challenge [C]// Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems. New York, USA: ACM, 2016: 289-292.

[23] LI Yuqing, DAI Wenkuan, GAN Xiaoying, et al. Cooperative service placement and scheduling in edge clouds: a deadline-driven approach [J/OL]. IEEE Transactions on Mobile Computing, 2021 [2021-06-01]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9361310.

[24] YOU Wencong, JIAO Lei, LI Jun, et al. Scheduling DDoS cloud scrubbing in ISP networks via randomized online auctions [C]// Proceedings of the IEEE Conference on Computer Communications. Piscataway, NJ, USA: IEEE, 2020: 1658-1667.

（编辑 陶晴）