A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning

Shubo Zhang[1], Tianyang Wu[1], Maolin Pan[1], Chaomeng Zhang[2] and Yang Yu*[1]
[1]*School of Data and Computer Science, Sun Yat-sen University. Guangzhou China*
[2]*Huawei Cloud, Huawei Technologies Co.Ltd. Shenzhen, China*
Email: {zhangshb, wuty26}@mail2.sysu.edu.cn, panml@mail.sysu.edu.cn,
zhangchaomeng@huawei.com, yuy@mail.sysu.edu.cn

***Abstract*—Due to the lightweight and flexible characteristics, containers have gradually been used for the application deployment and the basic unit for resource allocation in a cloud platform recently. Reinforcement learning (RL), as a classic algorithm, is widely used in virtual machine scheduling scenarios due to its advantages of adaptability and robustness. However, most RL methods have problems in container scheduling, such as untimely scheduling, lack of accuracy in decision-making and poor dynamics that will lead to a higher SLA violation rate. In order to solve the above problems, a predictive RL algorithm A-SARSA is proposed, which combines the ARIMA model and the neural network model. This algorithm not only ensures the predictability and accuracy of the scaling strategy, but also enables the scaling decisions to adapt to the changing workloads. Through a large number of experiments, the timeliness and effectiveness of the A-SARSA algorithm for container scheduling are verified, which can reduce the SLA violation rate dramatically while keeping the resource utilization rate at a good level.**

***Index Terms*—Auto-scaling, Container Cloud, SLA, Reinforcement Learning, ARIMA, Neural Network, Predictive**

I. Introduction

An important goal of cloud computing is that cloud service providers can save resource consumption as much as possible under the premise of ensuring SLA, that is, only allocate resources that users really need. This also makes the resource auto-scaling one of the core research contents in cloud computing. In the SLA of container cloud, the response time of the service is the most important SLO.

So far, many auto-scaling strategies have been proposed [1], such as threshold-based scaling strategies, cybernetics or queuing theory-based scaling strategies, and time series analysis or reinforcement learning (RL) based scaling strategies. These can be divided into responsive strategies and predictive strategies according to the timing of scaling. By making the decision of scaling in advance, predictive strategies have better scaling timeliness than responsive ones.

In predictive strategies, scaling strategies based on RL methods are adaptive and robust, and don't require any prior knowledge. Because the scaling strategy based on RL can ensure that the resource utilization of the application is in a relatively stable state when the workload changes dynamically, it has an excellent performance in the virtual machine scheduling scenario (e.g., [1], [2], [3]). However, due to the large scale, long startup and shutdown times, and slow migration of applications deployed on them, disadvantages of virtual machines become obvious gradully, and using lightweight containers as the basic unit of application deployment becomes more popular [4]. In this case, the object of scaling is the container. Because the start-up and shutdown time of container is shorter, container scheduling requires higher timeliness and accuracy of predictive scaling strategies than virtual machine scheduling. RL based scaling strategies have problems such as inaccurate decision making, too large state space, and fixed action space, which can lead to untimely resource scheduling and increased SLA violation rate.

First, this paper proposes two improved classical RL algorithms(Q-Learning and SARSA) which can be applied in container cloud, as the benchmarks for analysis and improvement. And then proposes a predictive auto-scaling algorithm A-SARSA used in container scheduling scenarios. Based on the SARSA, A-SARSA combines the ARIMA model and artificial neural network (ANN) model, in which the ARIMA model is used for workload prediction [5], and the ANN model is used for CPU utilization prediction [6] and response time prediction [7]. This algorithm performs state prediction based on the output of the neural network model, and then makes decisions based on the predicted state and Q table. By this way, the predictability and accuracy of the scaling is ensured, and the convergence speed of this algorithm is also improved. By discretizing the state space, the problem of state space explosion is solved. By dynamically adjusting the action number corresponding to each state during the learning process, this algorithm avoids the situation of repeated scheduling due to the fixed upper limit of single instance scaling. As a result, this algorithm reduces the SLA violation rate dramatically while keeping the resource utilization rate of an application at a good level.

The rest of this paper is organized as follows. Section II introduces the related work. Section III describes the problem and introduces the system architecture. and the Section IV introduces how the model of the A-SARSA algorithm is established and the execution process. Section V shows the experimental environment and experimental results. Section VI summarizes the paper and looks forward to the future work.

II. Related Work

Threshold-based scaling strategy [8] is a responsive strategy, which has the problem of untimely resource scheduling. In

978-1-7281-8786-0/20/$31.00 ©2020 IEEE
DOI 10.1109/ICWS49710.2020.00072

addition, the threshold setting should be based on a large amount of data analysis, and the threshold value of each case is not applicable to other cases [9]. Although the queuing method [10] can be used as a predictive algorithm when applied to a system that is less affected by workload changes, it is still a responsive algorithm in other cases. And the model based on queuing theory is an approximate estimation model of the system, the decision made may not be suitable for the actual situation of the system. The cybernetic method is mainly used in the formulation of responsive scaling strategies, and it is only suitable for slowly changing workload. When the workload changes suddenly, its model parameters cannot be updated to reasonable values in a short time [11]. Time series analysis is a predictive method [5], which includes two stages. The first stage is workload forecasting, and the second stage is to make scaling decisions based on the predicted workload. However, many time series analysis methods are based on thresholds to make scaling decisions (e.g., [12], [13]). There are problems that the threshold setting process is complicated and the number of instances to expand or contract after reaching the threshold is difficult to determine.

Most of the work related to RL is carried out with responsive Q-Learning and predictive SARSA algorithms [14]. Initially, RL algorithms were applied to server scheduling [15] or virtual machine scheduling in data centers [16]. Later, in order to solve the problem of large state space or continuity state space (which will lead to slower convergence), a deep RL method [2] using the neural networks of strong representation ability to fit the Q table or directly fit the scheduling strategy was proposed. At the same time, a fuzzy RL method that discretizes the state space and combines fuzzy reasoning to better support the dynamic allocation of resources was also proposed [3].

Due to the large difference between containers and virtual machines, RL related methods that perform well in virtual machine scheduling scenarios cannot be applied to container scheduling scenarios directly. Applying RL to container scheduling has a broad research prospect. Horovitz et al. [17] proposed a method for dynamically changing the threshold of instance scaling using RL in container scheduling. This method determines the state according to the number of instances, compared with the traditional RL method, it greatly reduces the size of the state space. However, this method is still based on the responsive threshold method. In order to reduce the learning process and improve the quality of adaptive strategies, Rossi et al. [18] proposed a RL method based on models to use system knowledge in container scheduling scenarios. But the number of actions corresponding to each state of the method is fixed. Therefore, it cannot solve the problem of repeat scheduling when dealing with drastically changing traffic, which causes SLA violation or resource waste.

Different from these methods, the A-SARSA algorithm combines with the ARIMA model and the ANN model, and supports the dynamic increase of the action number. This ensures the timeliness and accuracy of resource scheduling, and avoids the problem of repeated scheduling. In order to fit the container scheduling scenario (response time, resource utilization, etc., which varies significantly with the workload) and ensure that the Q table is reasonably updated, we chose off-policy strategy for updating the Q table [19].

III. PROBLEM DESCRIPTION AND SYSTEM DESIGN

A. Problems Description

We use software containers to deploy the application (such as docker). In order to process frequent user requests, multiple application instances should be created. These instances run in parallel, and each instance has the ability to handle user requests individually. While the application is running, its performance, such as response time, is affected by changing workloads. To meet the SLA and avoid waste of resources, we need to dynamically adjust the number of instances of the application based on the workload.

Problems to be solved. Known conditions are as follows. The workload of the application contains many characteristics (such as seasonality, suddenness, etc.), and there is no fixed change law. Users have quantitative requirements for response time. The service provider hopes to meet all the needs of users with minimal resource consumption. The goals of the auto-scaling strategy are as follows. Suitable for all types of workloads. Effectively reduce the SLA violation rate and improve the resource utilization as possible.

Existing challenges. The traditional RL method adopts the state-action-reward model for training. It scales up and down according to the state, and defaults the state of application is only affected by the actions performed by the agent. The state of an application deployed by containers is not only affected by the actions performed by the agent, but also affected by the workload and factors of the containers. This will lead to inaccurate scaling strategies formulated by traditional RL methods in container scheduling. In addition, the action space of traditional RL is fixed. In horizontal scaling, each state of the RL method usually corresponds to 5 actions (-2, -1,0, +1, +2). Therefore, when the application encounters a sharply increased workload, only adding two instances may not effectively reduce the response time. The fixed action space not only affects the timeliness and dynamics of resource scheduling, but also affects the user experience.

B. System Design

The target system architecture to solve the above problems is shown in Fig. 1. Each working component in the system is deployed on a cloud platform that supports Istio. Iisto [20] is a completely open source service mesh which comprises a data plane and a control plane.

Workload Predictor. The main component of the workload predictor is an ARIMA model that can be dynamically fitted based on real-time data. The input of the predictor is the value of the workload for the first few cycles, and the output is the predicted value of the workload for the next cycle. The predicted value output by the predictor will be uploaded in real time as a custom Exporter type metric.

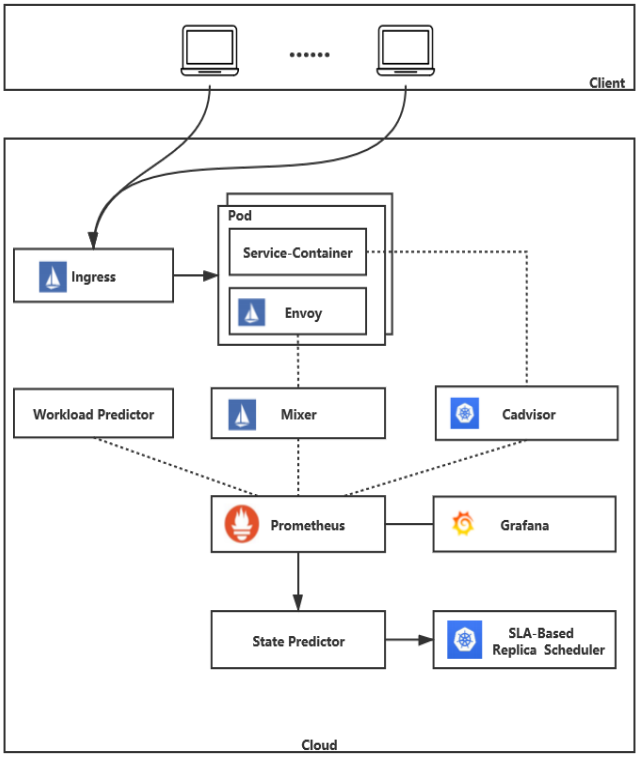


Fig. 1: System architecture

Metrics Collection. Traffic is sent to service gateway of Istio, and the service gateway will evenly distribute the traffic to each instance of the application. The envoy of each instance will report the traffic to the mixer component of Istio, thereby completing the upload of the user request rate metric. With regard to uploading container-related metrics such as the CPU utilization, memory utilization, and number of instances, CAdvisor Agent in the Kubernetes environment deployed on each node is responsible for it. Prometheus can grab metrics from Mixer, CAdvisor and Exporter and store them into a time series database. Then Grafana puts the metrics obtained from the TSDB on its own monitoring panel to achieve metrics visualization.

State Predictor. The state predictor is mainly composed of an ANN model that support dynamic training. The input of the state predictor is the values of CPU utilization, memory utilization, response time, request rate, number of instances, and the predicted request rate for the next cycle. Its output is the predicted value of the application's CPU utilization and response time when the predicted value of the request rate reaches the application. Since the state space applied in the A-SARSA algorithm is determined by the CPU utilization and response time, the state prediction value can be determined according to the output of the neural network.

SLA Based Replica Scheduler. SLA is service level agreement, which contains entries such as service type, service quality, and customer payment. It is jointly signed by the service provider and the customer, and can be as close to the user's needs as possible to ensure user satisfaction. In order to ensure that the response time of service does not violate SLA as much as possible and avoid waste of system resources, the instance scheduler will choose an action from the dynamically changing action space according to the state prediction value every cycle. Then it calls the API provided by Kubernetes API Server to change the number of application instances horizontally according to the selected action.

IV. MODEL ANALYSIS AND ALGORITHM DESCRIPTION

A. ARIMA Model

The ARIMA model is composed of an autoregressive model and a moving average model for non-stationary time series prediction. Compared with the ARIMA model, other linear models such as AR, MA, ARMA, and EWMA models are only good at predicting workloads with strong seasonal or obvious trends. If the time series is not stable, their prediction accuracy will be greatly reduced [21]. So the workload predictor in the system is designed with ARIMA model.

The ARIMA model in this paper is dynamically fitted according to a time series of fixed size. We first use historical data to fit the model. When we monitor the average user request rate for the current period, we add it to the time series and discard the oldest historical data from the time series. We then use the new time series for model fitting to predict future workloads. The forecast of the workload is usually obtained one cycle in advance.

During the ARIMA model fitting process, the time series must be stable [22], which means that the statistical characteristics do not change with different time windows. The initial time series may be unstable. In order to stabilize them, a differential operation is required. We specify that the minimum number of differences between the data changing from unstable to stable is the parameter d of the ARIMA model. And We use PACF (Partial Correlation Coefficient) to determine the order of autoregression p, and ACF (Auto Correlation Coefficient) to determine the moving average order q. Bringing the stable new data after the differential operation into the ARMA model expression and fitting, it will get (1).

$$\begin{aligned} Yt =& \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \varepsilon_t + \\ & \alpha_1 \varepsilon_{t-1} + \alpha_2 \varepsilon_{t-2} + \cdots + \alpha_q \varepsilon_t - q \end{aligned} \tag{1}$$

Finally, inversing differential the values obtained according to the above model expressions can be used to obtain the predicted workload value.

B. ANN Model

The input and output of ANN model are same as state predictor. Training the ANN model needs to capture historical metrics data with comprehensive features (the amount of data does not need to be large). The data consists of applications' metrics under the circumstances of high workload with high instance count, high workload with low instance count, low workload with low instance count, and low workload with

high instance count. The distribution of these metrics should be roughly balanced, so that the trained model can be prevented from having tendentiousness rather than generality. After training, the model can be directly applied to state prediction in actual scenes. The ANN model proposed in this paper also supports dynamic training. If the workload in actual scene has a tendency, the previously trained neural network model can continue to be dynamically trained. In this process the good neurons will be left behind while bad neurons will be eliminated. So that the ANN model can gradually fits the actual scene.

Ann model contains one input layer, one output layer and a certain number of hidden layers, and each layer has a mount of neurons. In the current scenario, the number of neurons in the input layer of the neural network is 7 (6 metrics plus a bias term), and the number of neurons in the output layer is 2. The number of hidden layers and the number of neurons in each hidden layer can be determined arbitrarily. Considering the risk of overfit and the scenario is not complicated, it is recommended to set the number of hidden layers small.

C. Classical Reinforcement Learning

There are many algorithms under the RL system, such as Q-Learning, SARSA, DQN, and DDPG. Most RL methods update the Q table through the agent's continuous learning, and make decisions based on the Q table and states. In this part, we will propose two improved classical RL algorithms related to A-SARSA. They are fitted to container cloud compared to original algorithms.

Q-Learning. The Q-Learning algorithm proposed in this paper for container scheduling is slightly different from the traditional Q-Learning algorithm. The state of the application changes quickly in the container scheduling environment, so we need to collect metrics in current cycle to obtain the state instead of directly obtaining it based on the state after actions performed in the previous cycle. After initializing the Q table, the algorithm performs the following steps in each cycle until it converges.

- Obtain the application metrics of the current cycle to determine the state S of current cycle.
- Use the ε-greedy strategy to select action A from the Q table according to the state S. The algorithm has a probability of ε to randomly select action A from the Q table, and a probability of 1-ε to select action A with the highest Q value. Its advantage is that it can guarantee the exploratory of algorithm during the learning process, that is, the comprehensiveness of the Q table's update.
- Perform action A, calculate reward R, and obtain new state S'. And find out the maximum Q value $max_aQ(S',a)$ that can be obtained by performing the action in the state S'.
- Use R and $max_aQ(S',a)$ to update the Q table according to (2), where α represents the learning rate and γ represents the discount rate.

$$Q(S,A) \leftarrow Q(S,A)+\alpha[R+\gamma max_aQ(S',a')-Q(S,A)] \quad (2)$$

In general, the Q-Leaning algorithm is a responsive algorithm, so there may be situations in which resource scheduling is not timely.

SARSA. We apply the idea of modifying the Q-Learning algorithm mentioned above to modifying the traditional SARSA algorithm, and propose the Algorithm 1. The execution process of each cycle before the algorithm converges after initialization is as follows.

- Perform action A selected in the previous cycle, obtain state S', and calculate the reward R.
- According to the state S', use the ε-greedy strategy to select the action A' to be performed in the next cycle from the Q table.
- Update the Q table by R and $Q(S',A')$ according to (3), where α represents the learning rate and γ represents the discount rate.

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)] \quad (3)$$

It can be seen that SARSA is a predictive algorithm that can predict what action to take in the next cycle in the current cycle. However, in the actual scenario, the user's request rate changes rapidly and the workload bearing capacity of each container is not as good as that of a virtual machine. Therefore, metrics such as the response time and resource utilization of container-based applications will easily change with the workload. In this case, the decision of the SARSA algorithm will lose its authenticity, that is, the predicted action may not be applicable to the next cycle.

Algorithm 1 SARSA In Container Scheduling

```
Initialize Q(S, A)
S ← observe the application state
A ← select an action by S and Q
while true do
    Take action A
    S' ← observe the application state
    Observe reward R by Eq. 4
    A' ← select an action by S' and Q
    Update Q(S, A) using Eq. 3
    A←A'
end while
```

D. A-SARSA

1) Model Establishment: There are many factors can be considered to form a state space such as response time, CPU utilization, Memory utilization, I/O utilization, etc. But there is a unfortunately fact that the more dimensions of the state space, the easier it is to cause the state space too large. Because the CPU utilization can most effectively reflect the resource utilization of the application, and the response time can reflect the user experience most intuitively, we choose the CPU utilization and response time determine the state space. We divide the response time into n levels, and divide the CPU utilization into m levels. A state space contains m * n states is constructed. This construction method makes the state space

discrete and solves the problem of state space explosion. We can determine the state of the application by detecting the CPU utilization and response time of the application.

During the initialization of the A-SARSA algorithm, the number of actions corresponding to each state is 5. If the response time of the current cycle violates the SLA, and the action determined by predicted state of previous cycle is already the action with the largest increase in the number of instances, then increase the action number of predicted state in previous cycle symmetrically (e.g., from 5 to 7). The dynamically increased action number makes up for RL methods.

2) Update Strategy And Reward Function: Although the A-SARSA algorithm is mainly developed based on the SARSA algorithm, the Q table update strategy is the off-policy strategy of Q-Learning as (2). This is because container-based applications' metrics such as resource utilization applied are relatively easy to change, and it is not possible to ensure that the state of the current cycle will not change after the workload of the next cycle arrives. That is to say the S' in (3) is not sure under the environment of container cloud. In this case, the off-policy strategy will be more in line with the actual situation than the on-policy strategy. ε and α of the algorithm will decrease as the number of iterations increases.

In order to avoid the occurrence of SLA violations and improve the CPU utilization as much as possible, the reward function needs to be formulated according to the response time, the response time threshold (RTTH) specified by the SLA and the CPU utilization. We formulated the reward function (4) in accordance with the above principles, where ρ represents CPU utilization and p is a manually set constant.

$$R = \begin{cases} \frac{1-e^{-p(1-\frac{respTime}{RTTH})}}{1-\rho} & respTime > RTTH \\ \\ \frac{1-e^{-p}}{1-\rho} & respTime < RTTH \end{cases} \tag{4}$$

3) Step of Algorithm: Algorithm 2 summarizes the A-SARSA. Initially, the value of each item in the Q table is initialized to 0, and the number of actions corresponding to each state is 5. $Q(S_t,\ A_t)$ is the expected value for performing the action A_t in the state S_t. In the first cycle of the algorithm, it needs to obtain the current state S_t and select the corresponding action with the highest expected value in the current state from Q table. If there are many actions corresponding to the highest expected value, one action is randomly selected from them for execution.

The following operations are repeated until the algorithm converges. First obtain the state S_t of the current period t, calculate the reward according to (4), and update the Q value of the specific action taken in the predicted state in the previous period according to (2), that is, $Q(S_{t-1},\ A_{t-1})$. And S_t is used to infer whether it is necessary to dynamically increase the action number of S_{t-1}. Due to the characteristics of the reward function, the newly added action will be quickly selected even the corresponding value in Q table is initialized to 0. Then Obtain the predicted value of the workload in the next cycle from the workload predictor, and input the predicted value of the workload and the metrics collected in the current cycle into the state predictor to predict the state S_{t+1}. Finally, Use the ε-greedy strategy to select the action A_{t+1} of the state S_{t+1} and execute A_{t+1}. Because it takes a period of delay as instances be fully deployed, starting to carry out an expansion or contraction command at the end of a cycle will not affect the state of the application in current cycle. The command will be executed exactly at the beginning of the next cycle, so that the number of instances can just handle the workload of the next cycle.

Algorithm 2 A-SARSA

```
Initialize Q(S, A)
S ← observe the application state
A ← select an action by S and Q
Take action A_t
while true do
    S_t ← observe the application state
    Observe reward R by Eq. 4
    if respTime > SLA and A_{t-1} = A_max then
        Increase the action num of S_{t-1}
    Update Q(S_{t-1},A_{t-1}) using Eq. 2
    Observe the predicted state S_{t+1}
    A_{t+1} ← select an action by S_{t+1} and Q
    Take action A_{t+1}
end while
```

During the operation of the algorithm, any one of the following three conditions is met, which means that the algorithm has converged.

- The number of run cycles exceeds the set number.
- Each value in the Q table is updated more than 3 times.
- Check the Q table update status every fixed number of cycles, and calculate the difference of absolute value between the items in the Q table that have been updated in the fixed cycles and the corresponding items before update. When these differences all less than one percent of the absolute values of the entries in the original Q table, and there are no items in the Q table that have not been updated from the beginning cycle to the present cycle.

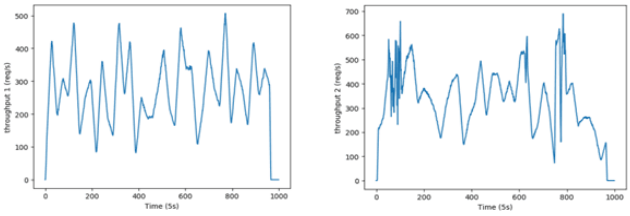
V. EXPERIMENTAL RESULTS

Experiment Settings. The experiments were performed on a cloud platform supporting Istio and Kubernetes. Applications for users and programs that runs automatic scaling algorithm are both deployed in a Kubernetes cluster. The Kubernetes cluster has a total of 6 virtual machines. The specifications of 3 virtual machines are 2-core CPU and 4G memory, and the specifications of the other 3 virtual machines are 4-core CPU and 8G memory. Their operating systems are CentOS7. Since the type of scaling during the experiment is horizontal, each

instance of application has a fixed specification of 0.5 core CPU and 1G memory.

Benchmark. The service used in the experiment is the official sample application provided by Istio: the Bookinfo application, which has a total of 4 components. A user can get an html webpage by accessing the productpage component of the application, and the application will make internal calls between the components during the process of obtaining the webpage. We check the performance of the auto-scaling algorithm by deploying, accessing, and scheduling Bookinfo application. Istio's official website has a detailed description of this application [23].

Workload Construction and Sending. We first collected real-world workloads for typical applications, namely the NASA and FIFA World Cup access datasets. Then based on their characteristics, two kinds of highly comprehensive traffic (traffic 1 and traffic 2) were constructed. Both traffic includes stable linear growth and instability (linear growth), linear decline and instability (linear decline), seasonality and instability (seasonal). The waveform diagram of them are shown as Fig. 2.



(a) Traffic-1 (b) Traffic-2

Fig. 2: Two Kinds of Traffic

We use Gatling to send traffic, which is a well-known open source testing tool that can well support simulated virtual users accessing specified applications. We write the constructed data set of traffic into a scala file and run the script responsible for sending the traffic in the Gatling tool, and the traffic will be sent according to the configuration of the scala file.

Metrics Definition. Because the image formed by metrics of user request rate, CPU utilization rate, and response time is not comprehensive enough, we define two additional metrics to specifically measure the performance of the algorithm. We are mainly concerned about two points, one is the violation rate of SLA, and the other is the average amount of resource consumption. For SLA we have used the 95th percentile response time and set it to 250 ms. For the SLA violation rate, we use the SLA violation time / total time to calculate. For the average amount of resource consumption, we calculate the average number of application's instances during running. In this experiment, we assume that users only focus on whether the response time violates the SLA, and not on how much the response time exceeds the SLA.

A. Experimental result and Comparisons

We apply the A-SARSA algorithm, the SARSA algorithm, Q-Threshold algorithm and the Model-based RL algorithm to the instance auto-scaling of the Bookinfo application. In the four instance scheduling scenarios, the traffic received by the application are the two highly comprehensive traffic mentioned above. The metrics of application under four auto-scaling strategies are shown in Table 1 (the average amount of resource consumption in the table refers to the average number of instances of the productpage component). We will conduct detailed comparisons and analysis of SARSA algorithm and A-SARSA algorithm, Q-Threshold algorithm and A-SARSA algorithm later.

The settings of the four RL related algorithms are the same during the experiment. Their learning rate α is initially set to 0.3, the discount factor γ is set to 0.8, the value of ε is set to 0.1, the value p in the reward function is set to 2, and the RTTH is set to 250ms according to the formulated SLA. Among them, the Q-Threshold algorithm is based on the expansion and contraction of the CPU utilization threshold. The initial value of the upper threshold is set to 0.7, and the initial value of the lower threshold is set to 0.2.

In order to ensure the timeliness of auto-scaling, and to ensure that the metrics collection process in each cycle is not affected by factors such as the number of instances in the previous period and the request rate. The cycle of resource scheduling should not be too long or too short. Considering it the time interval between making scaling decisions is set to 40s.

In addition, it needs to be explained that the number of instances in the figure below is the number of reallly working instance. It takes a certain amount of time for an instance to be deployed and put into work. The time for the instance scaling decision will be earlier than the number change of instances.

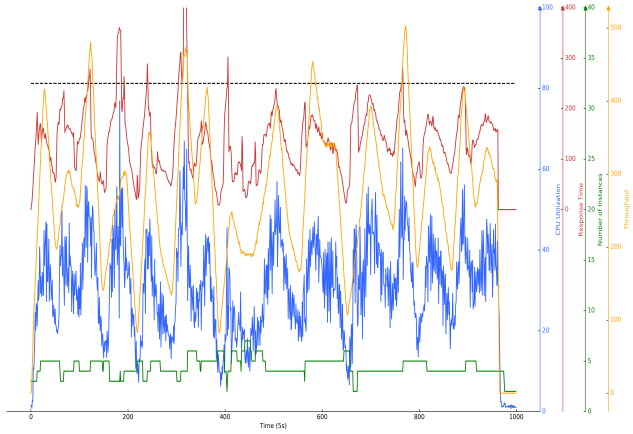
A-SARSA vs SARSA. The effect of two scaling methods respectively under two kinds of traffic is shown in Fig. 3 and Fig. 4. In traffic 1, the A-SARSA algorithm's SLA violation rate during the entire scheduling process is less than half of the SARSA algorithm's, but it needs to consume 4.79% more resources. In traffic 2, the A-SARSA algorithm not only the SLA violation rate is less than half of the SARSA algorithm's but also consumes 26.98% less resources. It can be seen that SARSA algorithm made multiple decisions to increase the number of instances in order to reduce the rapidly increasing response time when traffic 2 burst in the early days. Repeatedly increasing the number of instances in this way not only did not eliminate the violation in time, but also later caused the application's response time to be too low, which is obviously unreasonable. The A-SARSA algorithm generally does not make similar decisions.

Comprehensive analysis, the A-SARSA algorithm can more effectively and accurately perform instance scaling operations in the face of rapidly changing traffic, so that the number of instances fluctuates with the fluctuation of traffic. This can reduce the number of rapid increases in response time, avoid more negative effects, and achieve better results with less resource consumption.

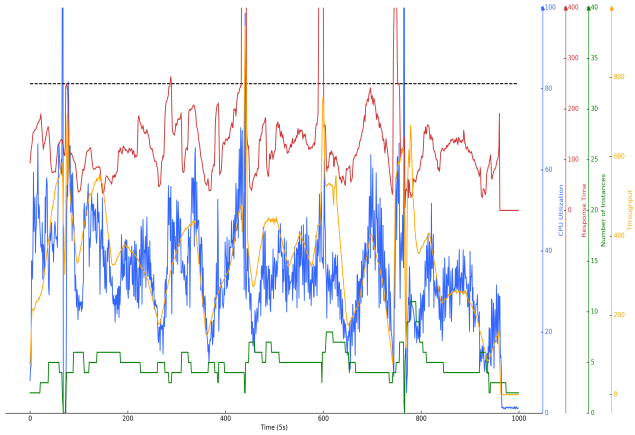
A-SARSA vs Q-Threshold. As shown in Fig. 3a and Fig. 5a, in traffic 1, the A-SARSA algorithm's SLA violation rate

TABLE I: Performance Comparison of Four Algorithms

Scaling Scenario	SLA Violation	Average Amout of Resource Comsuption	CPU Utilization
Q-Threshold and Traffic 1	16.3%	3.437	34.93%
SARSA and Traffic 1	9.4%	4.194	50.67%
Model-based RL and Traffic 1	6.7%	3.247	38.23%
A-SARSA and Traffic 1	4%	4.395	40.32%
Q-Threshold and Traffic 2	8.8%	4.532	32.60%
SARSA and Traffic 2	8.5%	6.15	46.73%
Model-based RL and Traffic 2	9.7%	3.313	44.2%
A-SARSA and Traffic 2	3.6%	4.491	33.75%

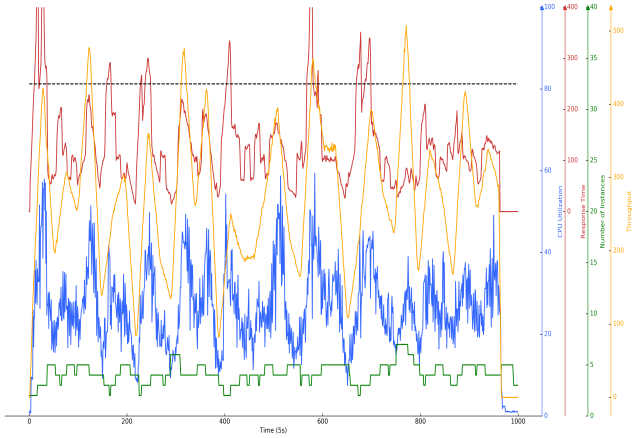


(a) A-SARSA-1

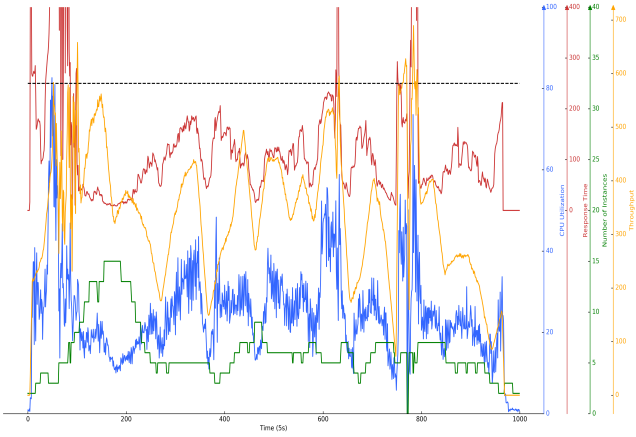


(b) A-SARSA-2

Fig. 3: A-SARSA in Workload 1 and 2



(a) SARSA-1



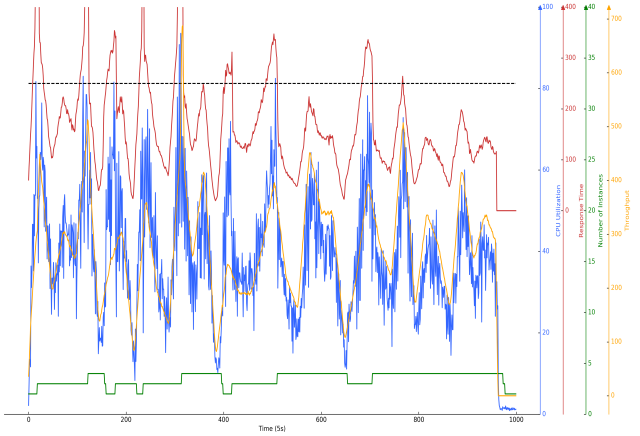
(b) SARSA-2

Fig. 4: SARSA in Workload 1 and 2

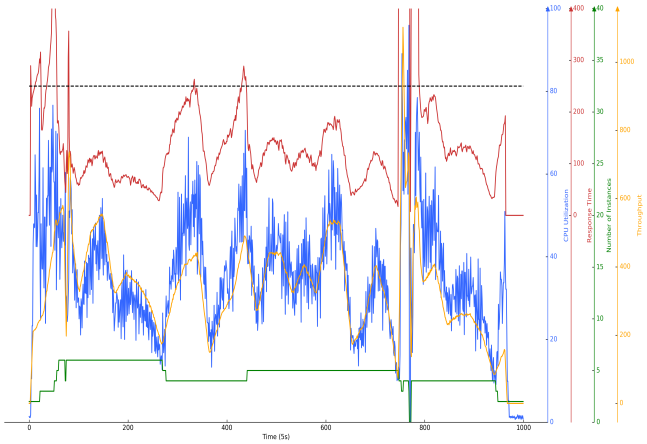
in the entire scheduling process is only a quarter of the Q-Threshold algorithm, but it requires 27.87% more resources; in traffic 2, The A-SARSA algorithm's SLA violation rate is 40.91% of the Q-Threshold algorithm, and the amount of resources consumed by the two in the entire scheduling process is almost the same, as shown in Fig. 3b and Fig. 5b.

This is because the number of instances changes slowly under the decision of the Q-Threshold algorithm. It can be seen from the above figure that the frequency of the change of instance amount during the operation of the Q-Threshold algorithm is significantly less than that of A-SARSA. Therefore, in the face of fast-growing traffic, the decision of the Q-Threshold algorithm is likely to be unable to keep up with the speed of traffic changes, resulting in SLA violation. And because the Q-Threshold algorithm is based on the threshold method, it cannot be as accurate as the A-SARSA algorithm to respond to traffic changes, and resources may be wasted.

A-SARSA vs Model-based RL. In Fig. 3 and Fig. 6 we compare two scaling methods under two kinds of traffic. In traffic 1, A-SARSA algorithm's SLA violation rate in

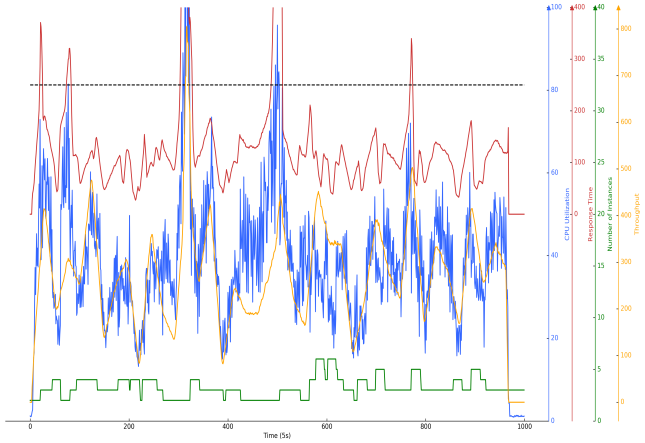


(a) Q-Threshold-1

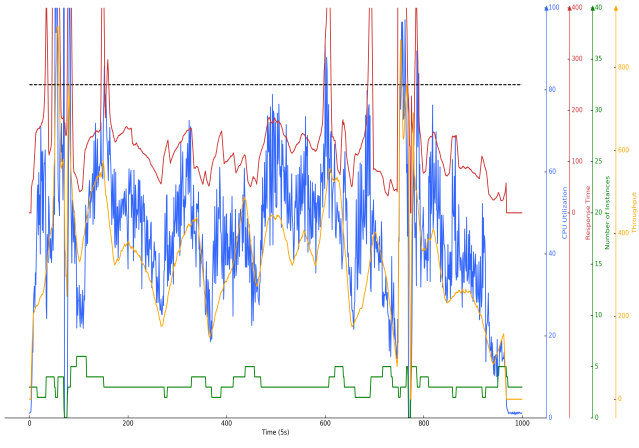


(b) Q-Threshold-2

Fig. 5: Q-Threshold in Workload 1 and 2



(a) Model-based RL-1



(b) Model-based RL-2

Fig. 6: Model-based RL in Workload 1 and 2

the entire scheduling process is 59.7% of Model-based RL algorithm, while it needs to use 35.35% more resources. In traffic 2, A-SARSA algorithm's SLA violation rate is only 37.11% of Model-based RL algorithm, using 35.56% more resources.

Although Model-based RL algorithm use much less resource than A-SARSA, it can hardly work well under rapidly changing traffic and lead to a high SLA violation rate. This is because Model-based RL algorithm is a reactive scaling algorithm. When it reacts to the change of environment, the response time of service has already exceeded SLA. And as the algorithm doesn't take traffic into account, there may be a situation that change of traffic will make previous correct action inappropriate. While its update process is based on expectation, these unproper actions will continuously influence the estimation of cost and probability of state transition in this state and may cause wrong action in the future.

B. Discussion

Overhead. From the above experimental results, it is indicated that the A-SARSA algorithm combines the advantages of the SARSA and ARIMA. During the running process of it, the number of instances can quickly change follow the workload, and at the same time, it can predict the workload. Therefore, its effect is better than those of the comparison algorithms. Although it frequently changes the number of instances, resulting in a certain amount of overhead, the results prove that it is worth it. The experimental results also show that the A-SARSA algorithm can always provide better service with relatively less resource. In contrast, other algorithms performs worse than the A-SARSA in terms of the timeliness and accuracy of scaling. During their execution process, there may be a situation that algorithm can't get up with the rapid change of traffic, which leads to SLA violation, or waste of resources.

Limitation. Because A-SARSA is based on the SARSA algorithm, it requires certain data for training to achieve better results. The effect may be bad during the start of training with cold start.

VI. CONCLUSION AND FUTURE WORK

The A-SARSA proposed in this paper focuses on solving the problems of untimely resource scheduling, inaccurate scaling decisions and repeated resource scheduling in RL applied to container scheduling scenarios. In order to ensure the accuracy of agent learning and action selection, we use a dynamic fitted ARIMA model to predict workload, and use a neural network model that supports dynamic training to predict CPU utilization and response time. This also indirectly accelerates the convergence speed of RL. After many experiments and comparisons in the same cloud platform, it can be seen that the A-SARSA is more reasonable and efficient in decision-making. Compared with other RL algorithms, it can significantly reduce the SLA violation rate, and at the same time maintain the resource utilization rate at a good level, that is, without causing waste of resources. In the future, we will explore the method of parallel learning based on the algorithm proposed in this paper, so as to further accelerate the convergence speed of it.

ACKNOWLEDGMENTS

This work is Supported by the National Key Research and Development Program of China under Grant No.2017YFB0202201; the National Natural Science Foundation of China(NSFC) under Grant No.61972427; the NSFC-Guangdong Joint Fund Project under Grant No.U1911205; the Research Foundation of Science and Technology Plan Project in Guangdong Province under Grant No.2020A0505100030. This work is also supported by Huawei Cloud's funding and experimental environment.

REFERENCES

[1] J. B. Benifa and D. Dejey, "Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, vol. 24, no. 4, pp. 1348–1363, 2019.

[2] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 129–134.

[3] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," in *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. IEEE, 2016, pp. 70–79.

[4] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker containers across multiple clouds and data centers," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 368–371.

[5] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2014.

[6] M. Borkowski, S. Schulte, and C. Hochreiner, "Predicting cloud resource utilization," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, 2016, pp. 37–42.

[7] X. Luo, Y. Lv, R. Li, and Y. Chen, "Web service qos prediction based on adaptive dynamic programming using fuzzy neural networks for cloud services," *IEEE Access*, vol. 3, pp. 2260–2269, 2015.

[8] W. Lin, J. Z. Wang, C. Liang, and D. Qi, "A threshold-based dynamic resource allocation scheme for cloud computing," *Procedia Engineering*, vol. 23, pp. 695–703, 2011.

[9] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 2. IEEE, 2013, pp. 256–261.

[10] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, pp. 1–39, 2008.

[11] Z. Chen, Y. Zhu, Y. Di, and S. Feng, "A dynamic resource scheduling method based on fuzzy control theory in cloud environment," *Journal of Control Science and Engineering*, vol. 2015, 2015.

[12] J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. S. Trivedi, "Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds," in *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*. IEEE, 2011, pp. 38–43.

[13] S. Khatua, M. M. Manna, and N. Mukherjee, "Prediction-based instant resource provisioning for cloud applications," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 597–602.

[14] J. H. Connell and S. Mahadevan, *Robot learning*. Springer Science & Business Media, 2012, vol. 233.

[15] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *2006 IEEE International Conference on Autonomic Computing*. IEEE, 2006, pp. 65–73.

[16] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *2010 IEEE 3rd international conference on cloud computing*. IEEE, 2010, pp. 410–417.

[17] S. Horovitz and Y. Arian, "Efficient cloud auto-scaling with sla objective using q-learning," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2018, pp. 85–92.

[18] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 329–338.

[19] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 64–73.

[20] Istio. [Online]. Available: https://istio.io/

[21] F. Nisar, S. Baseer *et al.*, "Survey on arima model workloads in a datacenter with respect to cloud architecture," in *2019 International Symposium on Recent Advances in Electrical Engineering (RAEE)*, vol. 4. IEEE, 2019, pp. 1–4.

[22] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[23] Bookinfo. [Online]. Available: https://istio.io/docs/examples/bookinfo/