







Enhancing Load Balancing in Computer-Aided Medical Systems Using Deep Reinforcement Learning

Yi-Xiao Wang^{1*}, Wen-Jian Tao², Shao-Peng Zhao³ and Yan-Ping Zhang⁴

^{1,2,3}Purification equipment research institute of CSIC, Handan 056027, China
shawn_wang123@163.com, 2489455090@qq.com, 3915069571@qq.com

⁴Harbin Engineering University, Harbin 150000 China
4zhangyanping19@hrbeu.edu.cn

Corresponding author: Yi-Xiao Wang, shawn_wang123@163.com

Abstract, The service mesh is the most well-known framework for developing network applications because it separates governance and business logic in microservices and achieves unified governance of heterogeneous systems. The Service Mesh has a more complicated topology structure than the typical microservice design, which makes it harder to keep the load balance. The present Service Mesh load balancing technique is rather straightforward, merely taking into account the current load status of each individual node and disregarding the mutual load impact across nodes. This paper proposes a load balancing method based on multi-agent reinforcement learning to address the aforementioned issues. This method turns the Service Mesh load balancing problem into a random game process, builds an Actor-Critic network to simulate the service mesh multi-resource scheduling strategy, and uses the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm to determine the best multi-resource scheduling strategy. The Istio framework is utilized in this study to create a service mesh environment, and the test results demonstrate that the suggested approach can produce faster response times.

Keywords: Service Mesh; Loading Balance; Multi-Agent Reinforcement Learning; MADDPG; Computer-Aided Medical Systems

DOI: <https://doi.org/10.14733/cadaps.2024.S9.265-276>

1 INTRODUCTION

With the development of Internet application technology, more and more developers abandon the single service architecture and use the micro-service architecture, but the microservice architecture combines business logic and service discovery, service registration, service meltdown and other services governance logic, resulting in the whole framework not easy to manage, increasing the burden of developers, while bringing difficulties to the platform upgrade.

Service mesh[10] is an independently run application service, providing a secure and efficient communication infrastructure layer between application services, able to use complex service topology structure, to reliably transmit various needs. The Service Mesh helps to develop and operate bugs, and developers no longer need to write and maintain policies and network codes in their applications, which are transferred to service agents and Service Meshes control planes, provided and dynamically managed by them. Service Mesh agents constitute a complex network topology structure, and in order to avoid traffic congestion and traffic abnormalities caused by complex networks, Service Mesh needs a reliable load balance strategy.

Load forecasting improves the system load balance capability by predicting the load of the application system[15], and has therefore been a research hotspot in the field of micro-services. Yoon[19] argues that the number and type of requests received by the application service system varies continuously over time, i.e. the load of the working system is about the function of time, and there is a specific connection with time, so it can be predicted by collecting information about the characteristics of the load in time. computer-aided medical systems, such as fluctuating workloads, resource heterogeneity, and the need for low-latency processing.

Current load prediction methods are mainly based on probability statistics and two main categories based on neural network models [11] [4]. In traditional statistics, there is a predictive load based on index smoothing[8], a prediction load of the self-regression linear (AR) model based on particle filtration[12], a predicted load of a self-regression moving average based on a smooth load[16], a differential-integrated self-reflexive moving average model predicting load[2], and a forecasting load of an index-based smooth and self-recursive fusion prediction model[3].

With the development of neural networks and the widespread application of big data, a large number of scholars have begun to study predictive models based on neuronal networks. For example, using the CNN network for load prediction[9] [18], using the RNN network prediction model[14], learning dynamic time characteristics using memory units, and load predictions through the LSTM/GRU network[20] [21].

This article will strengthen the introduction of learning into load balance algorithms, and proposes a load prediction algorithm based on multi-agent reinforcement learning[7], which can make decisions through independent learning network environment, comprehensive consideration of time delay and bandwidth and other factors, so as to avoid excessive traffic of a certain link leading to load imbalance, thereby improving the quality of the network service.

2 RELATED WORK

2.1. Service Mesh

Service Mesh is an infrastructure layer used to handle communications between microservices. The application has a complex micro-service topology, and the Service Mesh is responsible for the reliable delivery of requests in the topology. In actual development, a Service Mesh is a set of lightweight network agents deployed together with application services, and is transparent to application services.

The structure of the Service Mesh, as shown in Figure 1, adopts the edge car mode, providing an agent for each microservice, responsible for communication, deployment policy and configuration in the control plane.

The request only needs to be sent to the local service mesh agent by the application acting as the microservices' initiator; subsequent actions will then be taken by the grid agency, such as forwarding the request to the intended microservice after determining the load balance.

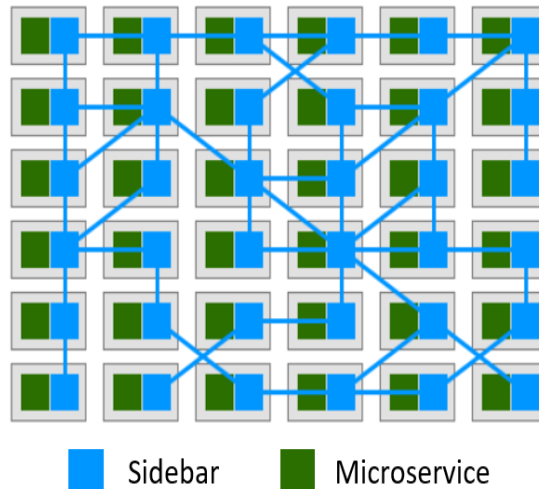


Figure 1: Microservice workflow diagram.

The network structure of the nodes in the Service Mesh forms a complex topological network, so that the difficulty of service governance is increased, so the load balancing technology becomes the key, and the load balance technology can effectively reduce the delay of requests, improve the reliability of the micro-service architecture and resource utilization rate.

2.2. Reinforcement Learning

Reinforcement learning is a class of algorithms in the field of machine learning, which mainly focuses on how intelligences take different actions in the environment to maximize the cumulative rewards in enhanced learning, intelligence by continuously trying to randomly operate, interact with the environment, and get feedback from the environment and then based on feedback decide the next operation, and ultimately through constant iteration of learning, to get the best solution.

Q learning is a model-free algorithm[17] that assumes that in the set of states S and set of actions A , Q is the expected return that action $a(a \in A)$ can obtain under the state $s(s \in S)$ at a given moment, $Q(s, a)$. The main idea of the Q-learning algorithm is to build states and actions into a table to store Q values, and then select the actions that will get the most outcomes based on Q -values.

In the service mesh, the process of calling the service is a limited Markov process. When the nodes in the service mesh are called, the status of the services grid will change, at which time the call can obtain a feedback value for the current real-time load state, which can be considered a reward function value. The higher the feedback value, the greater the likelihood that the strategy will be implemented the next time.

At time t , the service mesh chooses the node to be called, which changing the service mesh status from S_t to S_{t+1} , and the system feedback value is represented as:

$$r(t) = r(S_t, S_{t+1}) \quad (1)$$

S_t is the Service Mesh state before the call, and S_{t+1} is the services mesh status after the call. $r(\cdot)$ is used as reward function.

In the Service Mesh, the Q learning algorithm can obtain the optimal value function by numerical iteration, and the inferential formula for the Q matrix is as follows:

$$Q_{k+1}^*(S, S') \leftarrow \sum_{S'} P(S'|S)[r(S, S') + \gamma \max Q_k^*(S, S')] \quad (2)$$

S, S' represents the state of the environment after a service call, γ is the discount factor, Q^* is the maximum expected return that the Service Mesh can get after the service call; $r(S, S')$ is the immediate gain, and $\gamma \max Q_k^*(S, S')$ is the future discount gain. When the discount factor tends to 0 indicates that the reward value of the current behavior is considered more in decision-making, and the proportion of the rewards of the next behaviour is greater when approaching 1.

3 LOADING BALANCE METHOD BASED ON MULTI-AGENT REINFORCEMENT LEARNING

3.1. Multi-Agent Model on Service Mesh

The process of random play can be viewed as a Markov process, in which multiple intelligences make action decisions many times in multiple states. Each intelligence makes the best action decisions to enhance its value function based on its state, by observing the environment and forecasting the movements of other intelligence bodies.

Based on deep intensive learning techniques, the definition problem is modeled for $\mathcal{B} = \{1, 2, \dots, B\}$, using the Malkov decision-making process $G = \{\mathcal{B}, \hat{S}, \hat{A}, \hat{P}, \hat{U}\}$.

The state space \hat{S} refers to any microservice b having a state $\hat{S}_{b,t}$ at time t , including the CPU usage share, memory use share, IO read-write rate, network bandwidth, end-to-end latency of microservices applications, and request load changes of microservice b .

The action space \hat{A} represents that the independent action space for any micro-service b is \hat{A}_b , representing the resource quota used by any micro service; the joint action space of G is $\hat{A} = \hat{A}_1 \times \hat{A}_2 \times \dots \times \hat{A}_B$, where \hat{A}_i represents the activity space for microservices i , and \hat{A}_B represents activity space of microservice B ; the independent actions of any microserve b include changing the use share of the CPU, memory usage share, IO rate, and network bandwidth.

The reward function \hat{U} is the moment when t , any micro-service b is targeted by the state S , (b , t), the action A , (b , t) is taken to evaluate any microservice b is generated by action A . (b), (t) the rewards function is set to:

reward =

$$\begin{cases} \frac{\text{latency}}{SLO} + \theta_1 * \sum_i^m \sum_j^r \frac{Res_{u_{ij}}}{Res_{lim_{ij}}} & \text{if}(\text{latency} < SLO) \\ -\theta_2 * \frac{\text{latency}}{SLO} & \text{else} \end{cases} \quad (3)$$

$Res_{u_{ij}}$ represents the use share of microservices i and j resources, $Res_{lim_{ij}}$ indicates the share limit of resources in microservice i and j , latency indicates current microserve application end-to-end delay,

SLO is a service level target, θ_1 , θ_2 represents a set overweight parameter, m is the number of micro-services, and r is the amount of resources used by micro-service i .

State transfer probability \hat{P} refers to the selection of algorithms based on determination strategies, which are unique to state s , the resulting action a , which the presence of $u_{\theta}(s) = a$.

3.2. Actor-Critic Algorithm Based on Multi-Agent Deep Deterministic Policy Gradient

This article uses The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) to determine the allocation of resources for microservices. The algorithm can give the best action in a competitive environment, using only local information. In order to find the best load balancing strategy under the service mesh, this article combines the advantages of the deep Q-network(DQN)[5] and the Actor-Critic framework[6] to build an actor-critic network, as shown in Figure 2, 3 to implement service resource allocation decisions.

DQN is a value-based algorithm rather than a policy-based algorithm. It is not a strategy, but a critical one. Critical does not take actions directly, but evaluates the quality of actions

Practical methods for reinforcement learning frequently employ the Actor-Critic algorithm architecture. This framework combines the most often used framework for practical issue solving, the policy search method, and the value function estimation algorithm.

The three following traits apply to the MADDPG algorithm: (1) By learning the best approach, the best action can be applied using only local knowledge. (2) It is not necessary to be familiar with the dynamic environment model and particular communication requirements. (3) The method is applicable not only in a collaborative setting but also in a hostile one.

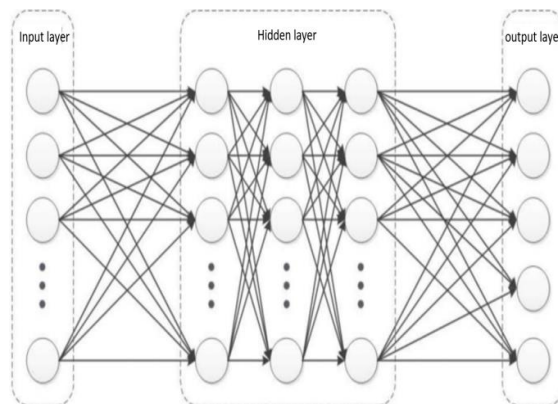


Figure 2: The actor network.

In training process, the actor network outputs actions a_t based on pre-processed information collected as state s_t . The critic network evaluates the actions of the actor network based on the overall state and action network output Q value at moment t .

At the next moment $t+1$, The actor network outputs action a_{t+1} , and critic network assesses a_{t+1} and outputs Q' . The actor network updates its parameters based on Q and the critic network updates its parameters based on Q' using the minimized loss function.

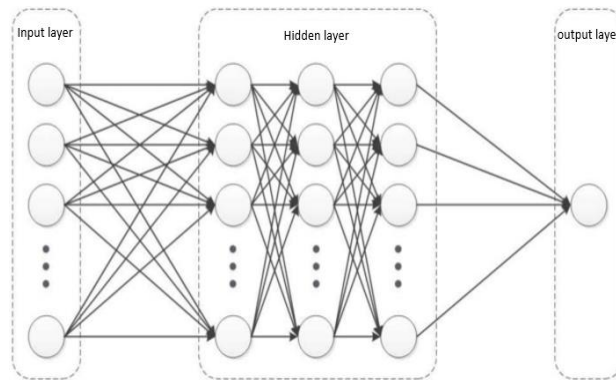


Figure 3: The critic network.

When making decisions, only using the Actor network to interact with the environment, and each microservice's own information is entered as a state after data pre-processing into the actor network, generating action. Aggregate the actions of all microservices to get the necessary joint actions executed. After the implementation of the joint action, continue to complete the resource allocation decision for the next time interval.

3.3. Model Training

The MADDPG algorithm employs a distributed execution and centralized training strategy. The Actor network simply has to be aware of its own local state during execution by observing the performance of resource utilization and runtime information to produce action. During training, the Critic network helps the Actor network learning allocation strategy through global status and action.

The actor network updates parameters by maximize the Q-value, which evaluated by the critic network; θ represents the microservice policy parameter, that include the use share of the CPU, memory usage share, IO rate, and network bandwidth; μ represents microservice strategy; s represents state, and here is $s = [x_1, x_2, \dots, x_n]$, where x_i represents the state of the first microservice.

The microservice i get the status-value function $Q_i^\mu(s, a_1 a_2, \dots a_n)$ by using the strategy μ , the state s and the joint action $[a_1 a_2, \dots a_n]$.

The actor network get the strategy grade by formula shown as:

$$\nabla_{\theta_j} J(\mu_i) = \nabla_{\theta_j} \mu_i(a_i | x_i) \nabla_{a_j} Q_j^\mu(s, a_1 a_2, \dots a_n) |_{a_i = u_i(x_i)} \quad (4)$$

θ_i represents the policy parameter for micro-service i , μ_i is the policy for micro service i , s is the model state, a_i represents the action of microservice i , x_i is the state of the first micro-serve i , J is the cost function.

For each microservice, the actor network has a critic network that obtains global information to evaluate the future total earnings of action a_t under state S_t .

In order to bring the Q value of the critical network output more in line with the long-term benefits of the action a_t , the critic network is updated using the minimize loss function as follows:

$$L(\theta_i) = (y - Q_i^\mu(s, a_1, a_2, \dots, a_n))^2 \quad (5)$$

$$y = \text{rew}_i + \gamma Q_i^{\mu'}(s', a'_1, a'_2, \dots, a'_n) |_{a'_k = \mu'_k(x_k)} \quad (6)$$

Where θ_i represents the policy parameter for microservices i , μ represents microservice strategy, s represents state, a_i indicates the action of microservice i , rew_i means action a_t receives rewards under state s_t , L represents loss function, γ represents discount quote, μ' represents target network strategy, a'_k represents action generated by target network sub-microservice k .

The training procedure is displayed in Figure 4.

First algorithm initializes the model;

At moment t , for each microservice b , algorithm input b 's own information, which is pre-processing as state data $s_{b,t}$, then b outputs action $a_{b,t}$; aggregating The action of all microservices leads to the the joint action \widehat{A}_t ; executes \widehat{A}_t then obtain rewards r_t and the next state $s_{b,t+1}$.

At next moment $t+1$, algorithm puts $(s_t, \widehat{A}_t, r_t, s_{t+1})$ into experience buffer, and set $s_{b,t+1}$ to $s_{b,t}$; iterates each micro service, and randomly extracts mini-batch samples, which are used as input of the critic network, to get the future reward y ; updates the critic network by minimizing the loss function; updates the actor network with the output of the critic network; updates every microservice's actor-critic network one by one. Algorithm repeats the above steps until the model converges.

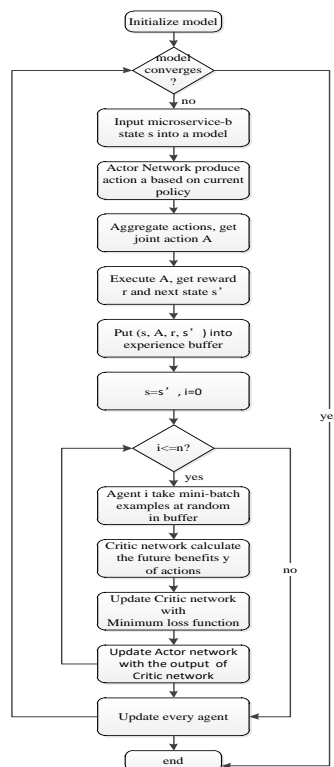


Figure 4: Model training process.

3.4. Multi-Resource Schedule System

The system based on the load balancing algorithm described in this article, as illustrated in the figure, is designed in this study. The available resources of the service mesh nodes are dynamically assigned through the resource scheduling module, therefore informing each node's load balance.

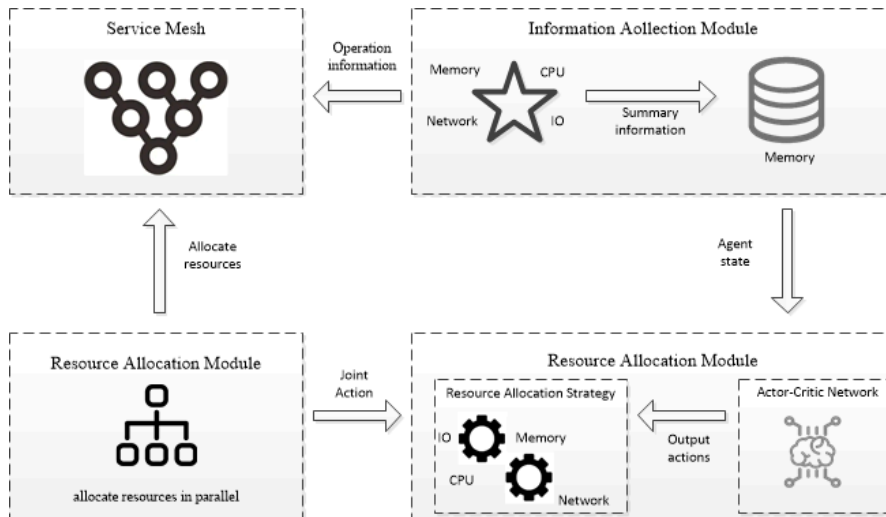


Figure 5: Multi-resource scheduling system.

After initialized the Multi-Resource Schedule System, at the moment t , each agent in the service mesh, parallel collects each microservice b 's operation information, which is the necessary data for algorithm.

The system combines the information gathered in accordance with the same time stamp. Calculate the mean and standard deviation of all the information after it has been integrated, then standardize it so that all the data are in the same order of magnitude and each agent has its own state s_{bt} , which is the input of the actor-critic network at the moment t .

The trained multi-resource decision model creates a multi-resource cooperative allocation strategy for the microservice for the present observation state s_{bt} . The objective is to ensure the tail delay SLO of the micro-service application as much as feasible while maximizing overall resource utilization. The multi-resource decision model's action strategy output results in a joint action \hat{A} , that $A_i(A_i \in \hat{A})$ is employed to allocate resources for microservice i .

Following scheduling, the algorithm gathers the microservice resource data once again and completes the aforementioned processes to guarantee the load balancing of each service node.

4 EXPERIMENT

Load forecasting can predict the system load value in the future in advance. If the load threshold is exceeded, the scheduling module will be notified to reallocate resources. Finally the resource allocation of the whole system will be more uniform, and the overall response speed of the system will be faster.

We create a Service Mesh for testing using the Isito[1] framework, use Jmeter[13] to simulate user requests in the Service Meshes, and record the response time to evaluate the performance of the system load balancing strategy.

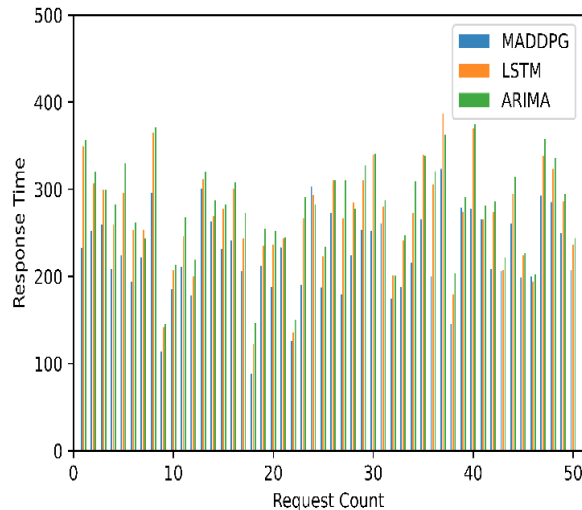


Figure 6: Comparison of system response time.

We compare the MADDPG method proposed in this paper with ARIMA[2] method and LSTM method[20]. As shown in figure 6, In the first 50 requests, our method has the shortest system response times.

This study also uses the average response time to measure the load balancing of the system after 2500 requests in order to comprehend the stability of the load balancing system. The formula for calculating the average response time is shown:

$$ART = \frac{\sum_{i=1}^n RT}{n} \quad (1.)$$

Where RT means response time of a single service call, and n is the total number of service invocations.

System	ARIMA	LSTM	MADDPG
<i>Average</i>			
<i>Response Time</i>	285ms	271ms	211ms

Table 1: Average response time.

The average response time is shown in table 1. The table shows that our method performs best and has the shortest average response time, which was decreased by around 22% and 26%, respectively, when compared to the LSTM and ARIMA methods.

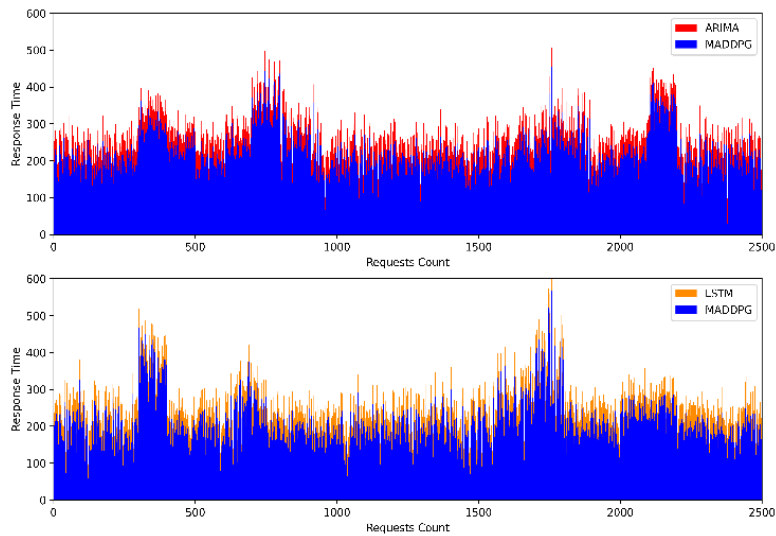


Figure 7: Overall comparison of system response time.

We provide a trend chart of the system response time to better understand the load state of the loading balance system after prolonged operation.

The overall trend of system response time is shown in figure 7, which proves that our method is better than other methods in general.

5 CONCLUSIONS

The load balancing issue in the service mesh is examined in this research, which also proposed a multi-resource optimization scheduling technique based on multi-agent reinforcement learning as a fresh approach to the problem.

By the use of the Istio framework, this paper creates a Service Mesh environment for testing. It then conducts experiments using the proposed algorithms in this environment and analyzes the outcomes. The findings indicate that when compared to previous methodologies, this article suggests faster response times in both general and partial cases. The results demonstrate that the cognitive intelligence-based approach outperforms previous methodologies in terms of response times, both in general scenarios and partial cases. This suggests that the utilization of cognitive intelligence techniques in load balancing and resource scheduling can significantly enhance the overall performance of service mesh systems.

How to further optimize the algorithm proposed in this paper to better load balancing in a Service Mesh with a more complex topological structure, so that it can be put into use more stably, will be our next research goal.

Yi-Xiao Wang, <https://orcid.org/0009-0005-7318-9363>

Wen-Jian Tao, <https://orcid.org/0009-0003-2756-6944>

Shao-Peng Zhao, <https://orcid.org/0009-0003-9018-7277>

Yan-Ping Zhang, <https://orcid.org/0009-0008-8048-4290>

ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China under Grant No. 2020YFB1710200.

REFERENCES

- [1] Calcote, L.; Butcher, Z.: Istio: Up and running: Using a Service Mesh to Connect, Secure, Control, and Observe, O'Reilly Media, 2019.
- [2] Calheiros, R.N.; Masoumi, E.; Ranjan, R.: et al. Workload Prediction using ARIMA Model and its Impact on Cloud Applications' QoS, IEEE Transactions on Cloud Computing, 3(4), 2014, 449-458. <https://doi.org/10.1109/TCC.2014.2350475>
- [3] Chen, R.: Index Smoothing and Autoregressive Fusion Forecasting Model and its Empirical Study, Liaoning Technical University, 2021.
- [4] Dama, F.; Sinoquet, C.: Time Series Analysis and Modeling to Forecast: a Survey, Arxiv preprint arXiv, 2104.00164, 2021.
- [5] Fan, J; Wang, Z.; Xie, Y. et al.: A Theoretical Analysis of Deep Q-Learning, Learning for Dynamics and Control, PMLR, 2020, 486-489.
- [6] Frémaux N, Sprekeler H, Gerstner W. Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons, PLoS Computational Biology, 2013, 9(4), e1003024. <https://doi.org/10.1371/journal.pcbi.1003024>
- [7] Gronauer, S.; Diepold, K.: Multi-Agent Deep Reinforcement Learning: a Survey, Artificial Intelligence Review, 2022, 1-49.
- [8] Khatua, S.; Manna, M.M.; Mukherjee, N.: Prediction-Based Instant Resource Provisioning for Cloud Applications, 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, IEEE, 2014, 597-602. <https://doi.org/10.1109/UCC.2014.92>
- [9] Kuo, P.H.; Huang, C.J.: A High Precision Artificial Neural Networks Model for Short-Term Energy Load Forecasting, Energies, 11(1), 2018, 213. <https://doi.org/10.3390/en11010213>
- [10] Li, W.; Lemieux, Y.; Gao, J.: et al. Service Mesh: Challenges, State of the Art, and Future Research Opportunities, IEEE International Conference on Service-Oriented System Engineering, IEEE, 2019, 122-1225. <https://doi.org/10.1109/SOSE.2019.00026>
- [11] Liu, Z.; Zhu, Z.; Gao, J.: et al. Forecast Methods for Time Series Data: a Survey, IEEE Access, 9, 2021, 91896-91912. <https://doi.org/10.1109/ACCESS.2021.3091162>
- [12] Long Feng.; Xue Dong-lin.; Chen Gui-ming.; Yang Qing.: Fault Prediction Algorithm Based on Particle Filter and Linear Autoregressive Models, Computer Technology and Development, 21(11), 2011, 133-136.
- [13] Nevedrov, D.: Using Jmeter to Performance Test Web Services, Published on dev2dev, 2006, 1-11.
- [14] Nguyen, H.M.; Woo, S.; Im, J.: et al. A workload Prediction Approach Using Models Stacking Based on Recurrent Neural Network And Autoen-Coder, 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, 2016, 929-936. <https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0133>
- [15] Peng, H.; Wen, W.S.; Tseng, M.L. et al.: A Cloud Load Forecasting Model with Nonlinear Changes Using Whale Optimization Algorithm Hybrid Strategy, Soft Computing, 25(15), 2021, 10205-10220. <https://doi.org/10.1007/s00500-021-05961-5>
- [16] Roy, N.; Dubey. A.; Gokhale, A.: Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting, 2011 IEEE 4th International Conference on Cloud Computing, IEEE, 2011, 500-507. <https://doi.org/10.1109/CLOUD.2011.42>

- [17] Watkins, C.J.C.H.; Dayan, P.: Q-learning, *Machine learning*, 8, 1992, 279-292. <https://doi.org/10.1023/A:1022676722315>
- [18] Xu, W.; LeBeau, J.M.: A Deep Convolutional Neural Network to Analyze Position Averaged Convergent Beam Electron Diffraction Patterns, *Ultramicroscopy*, 188, 2018, 59-69. <https://doi.org/10.1016/j.ultramic.2018.03.004>
- [19] Yoon, M.S.; Kamal, A.E. Zhu, Z.: Requests prediction in cloud with a cyclic window learning algorithm, 2016 IEEE Globecom Work-shops (GC Wkshps), IEEE, 2016, 1-6. <https://doi.org/10.1109/GLOCOMW.2016.7849022>
- [20] Zheng, H.; Lin, F.; Feng, X.: et al.: A Hybrid Deep Learning Model with Attention-Based Conv-LSTM Networks for Short-Term Traffic Flow Prediction, *IEEE Transactions on Intelligent Transportation Systems*, 22(11), 2020, 6910-6920. <https://doi.org/10.1109/TITS.2020.2997352>
- [21] Zheng, J.; Xu, C.; Zhang, Z. et al.: Electric Load Forecasting in Smart Grids Using Long-Short-Term-Memory Based Recurrent Neural Net-Work, 2017 51st Annual conference on Information Sciences and systems IEEE, 2017, 1-6.