Survey on Machine Learning-based Autoscaling in Cloud Computing Environments

Oliver Lemke, Sayantini Majumdar*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: oliver.lemke@tum.de, sayantini.majumdar@tum.de

***Abstract*—Modern cloud computing systems have demonstrated great aptitude for providing accessible, cheap, and scalable computing infrastructure to businesses and the public at large. However, the computing model comes with a variety of challenges for cloud service providers. Especially the task of automated distribution of computing resources, called autoscaling, has proven difficult so solve. A variety of different approaches have been proposed, chief among them machine learning-based algorithms. Thus, this paper aims to give an overview of recent developments in the field of machine learning-based autoscaling. In particular, we compare and contrast two approaches: MLScale, a supervised learning-based solution utilizing neural networks and multiple linear regression, and RLPAS, employing an algorithm based on SARSA reinforcement learning. We come to the conclusion that RLPAS' ability to predict required resource spikes and provision resources proactively, puts it at a decisive advantage compared to the reactive MLScale. However, as RLPAS is much more algorithmically complex, we propose that further research is required to show safe and effective scaling for more complex, real-world problems.**

***Index Terms*—cloud computing, autoscaling, machine learning**

1. Introduction

Cloud computing (CC) is an architecture that enables on-demand network access to a pool of computing resources, such as networks, servers, storage, applications, or other services [1]. Hardware resources are owned and managed by a cloud service provider (CSP), allowing customers remote access. Besides eliminating capital expenditure for users, the consolidation of resources also reduces operating expenses due to higher resource utilization [2, Chapter 1]. Combined with the offer of constant availability and pay-as-you-go pricing options [3], this service model is thus considered an attractive option, especially for small- and medium-sized businesses [4] [5].

In order to reliably supply a service that can handle large variations in requested operations, e.g. due to sudden user demand, CSPs have to be able to automatically scale the resources distributed to a specific application. This allocation is a highly complex and important process, as both under- and oversupplying resources will result in major costs to the provider, in the form of contract violations and excess operating expenses respectively [6, Chapter 7.4]. Yet, major CSPs like Oracle still rely on manually set autoscalers which base scaling decisions purely on simple thresholds [7] that are unable to adapt adequately to fluctuating user demand.

We reason that machine learning (ML), a field which particularly excels in complex and dynamic environments, represents the most promising approach vector towards solving this problem. Thus, this paper aims to give an overview of recent advances, especially comparing MLScale [8] and RLPAS [9], proposals utilizing supervised (SL) and reinforcement learning (RL) respectively. We show that while the papers differ considerably in their approach, both demonstrate a substantial improvement over manual threshold-based autoscalers. In addition however, we point out reactiveness as a prohibitive weakness of the MLScale algorithm and identify safe and effective scaling as an area requiring further research to achieve industry-wide adoption of an RL-based solution.

In order to do so, the paper is structured as follows: We will explain requisite theoretical knowledge in Section 2 and discuss related works in Section 3. Section 4 introduces and contrasts the two algorithms in detail, closing with a conclusion and an outlook on future development in Section 5.

2. Background

In this section, the scientific concepts underlying this paper will be discussed. Specifically, we will focus on the current applications of cloud computing, as well as the theory behind two central paradigms of ML: supervised and reinforcement learning.

2.1. Cloud computing

The main advantage of cloud computing lies in the ability for the provider to easily and swiftly split computing resources, supplying a large amount of individual customers [1]. In order to simplify this process and allow division at a more granular level, CSPs utilize virtual machines and a process known as autoscaling.

2.1.1. Virtual machines. A virtual machine (VM) is a software-based emulation of the runtime environment provided by a physical computer. In contrast to a real computer however, the resources the VM has access to, such as the CPU cores or memory, can be varied by the underlying program. As a piece of software, multiple instances of VMs can be run on a single server, each of which can be used independently and thus provided to a different customer [10, Chapter 1]. This virtualization is most common for servers, but has started to be increasingly applied to networking appliances, such as routers, switches or firewalls. Each instance of such virtualization applied to a networking service is called a virtual network function (VNF) [11].

2.1.2. Autoscaling. The virtual nature of the implementation allows for the easy up- and downscaling of the computing resources provided to a particular service as required by demand. This property is known as elasticity. Doing so automatically, or using automated policies, is called autoscaling. Applications can be scaled horizontally (in and out), representing the removing and adding of instances, as well as vertically (up and down), representing the addition or removal of an existing instance's resources [6, Chapter 8.2]. Autoscaling policies are geared towards a variety of different goals, such as improving resource utilization or decreasing operating expenses. Most notably, they aim to minimize service level agreement (SLA) violations, a type of contract stipulating the conditions of the service provided by the operator [12]. As Zhang et al. point out in [13, Section 6.1], in contrast to these complex targets, they often rely on rather simplistic metrics, such as throughput, response time, or amount of user requests, further increasing complexity.

2.2. Machine learning

Machine learning is a field of computer science focused on training an algorithm through the use of experiences and data, without programming explicit rules [14]. This approach is especially useful in environments where the ruleset is either particularly complex or not explicitly known by humans, such as natural language processing [15], computer vision [16], autonomous driving [17] or board games like Go [18], among a large variety of other use cases. There exist three basic paradigms of machine learning: supervised, unsupervised, and reinforcement learning. We will first focus on SL using neural networks and multiple linear regression, while the final section will introduce RL using SARSA and present an optimization technique called function approximation.

2.2.1. Supervised learning with neural networks. In contrast to the other two paradigms, in supervised Learning the algorithm is trained using data which includes the desired solutions [19, Chapter 1]. One such algorithm is called a neural network (NN). At a basic level, this algorithm tries to predict a set of m outputs (the solution), based upon a set of n inputs (the data). It consists of a set of simple processing units called neurons or nodes, which are organized into l layers. Each neuron ν can hold one value y. In a simple fully connected, feed-forward network each node $\nu_{i,j}$ of layer $i \in \{1, 2, ..., l-1\}$ is connected to every node $\nu_{i+1,k}$ of layer $i+1$, along with an individual adaptive weight $w_{i,j,k} \in [0,1]$ associated with the connection from neuron j in layer i to neuron k in layer $i+1$. The first layer is called the input layer, as every node is initialized with one of the n inputs. Consequently, it consists of n neurons. Similarly, the last (l^{th}) layer is called the output layer, where each of the m neurons corresponds to one output. All layers in between are the hidden layers. Given all values $y_{i,1}, ..., y_{i,p}$ of layer i, the value of the k^{th} neuron in layer $i+1$ can be calculated using

$$y_{i+1,k} = f_{i+1}(w_{i,0,k} + \sum_{j=1}^{p} w_{i,j,k} y_{i,j}), \tag{1}$$

where f_{i+1} represents a non-linear function called the activation function and $w_{i,0,k}$ represents a weighted bias for layer i and neuron $\nu_{i+1,k}$. f is often varied on a per-layer basis. To train the NN, the individual connection weights $w_{i,j,k}$ and biases $w_{i,0,k}$ can be adjusted through a process called backpropagation until the prediction achieves good accuracy when compared to a target output (vector). To make a prediction, after initializing the input layer, we can consecutively calculate the values for successive layers, until the output layer is reached [19, Chapter 10] [20, Chapter 2].

2.2.2. Supervised learning with multiple linear regression. Prediction based on multiple linear regression (MLR) uses a set of data points in order to fit a linear function

$$\hat{Y} = a + \sum_{i=1}^{n} b_i X_i. \tag{2}$$

$X_1, ..., X_n$ represent a set of n inputs called explanatory variables along with associated weights $b_1, ..., b_n$. $\hat{Y}$ represents the predicted output called the response variable, and a is the bias [21].

2.2.3. Reinforcement learning. Reinforcement learning is a subcategory of machine learning, where an agent interacts with an environment [19, Chapter 1]. The states the world can be in, the actions that can be taken in it, as well as its time, are discretized into states $s \in S$, actions $a \in A$ and time steps t. Thus, at time time t, the agent can transition from states S_t to S_{t+1} using actions A_t. Accordingly, the process can be modelled as a markov decision process (MDP) [22] [23, Chapter 3.1].

The agent is further awarded a positive or negative reward R_{t+1}, based upon which it aims to construct an optimal action-value function

$$q_\star(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]. \tag{3}$$

$q_\star : S \times A \to \mathbb{R}$ reflects the expected total reward G_t, or in other words the value of an action given a certain state. Deriving from $q_\star$, the algorithm creates the optimal policy $\pi_\star : S \to A$, mapping every state to the action that will lead to the highest expected value. $\pi_\star$ is the final decision-making function [23, Chapter 4]. This approach to RL is called model-free, as it does not require an explicit model of the environment to learn, instead simply observing the rewards in relation to the given states and taken actions.

SARSA. The state-action-reward-state-action algorithm is one such model-free approach that aims to converge to $q_\star$. This estimate is denoted by Q. In order to do so, it starts with an initial function Q and updates the function every time an action is taken:

$$\begin{aligned} Q(S_t, A_t) &:= Q(S_t, A_t) \\ &\quad + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \end{aligned} \tag{4}$$

It is proven that Q converges to $q_\star$ [23, Chapter 6.4] [24].

Function approximation. Whereas for smaller state spaces it can be enough to iteratively apply SARSA and update the policy function π, a process called policy iteration, this can become computationally infeasible even for comparatively simple tasks. The upper bound for greedy policy iteration is considered to be $O(\frac{k^n}{n})$ [25], where n is the number of states, and k the number of available actions per state. Instead of using a tabular representation of Q, function approximation utilizes a differentiable function

$$\hat{Q} : S \times A \times \mathbb{R}^d \to \mathbb{R} : (s, a, \mathbf{w}) \mapsto y, \tag{5}$$

where $\mathbf{w} := (w_1, w_2, ..., w_d)^T$ represents a vector of weights and y represents the calculated value. Applying stochastic gradient descent [26], $\mathbf{w}$ is updated after every action so that $\hat{Q}(s, a, \mathbf{w})$ approximates $Q(s, a)$ [23, Chapter 9]. However, as $d < |S|$, Q can often not be exactly approximated. This approach converges in $O(n^3)$ [27].

2.2.4. Parallel learning. In order to speed up convergence, N agents can interact with the environment at the same time and independently of each other [28]. In the implementation used by Benifa et al. [9], each agent j keeps track of its own local action-value function Q_{l_j}. Periodically, every agent shares its local estimate Q_{l_j} with every other agent, receiving the other local estimates $(Q_{g_1}, Q_{g_2}, ..., Q_{g_{N-1}})$ known as global estimates. To compute a final estimate Q_{f_j} every agent calculates a weighted average

$$Q_{f_j} = \frac{1}{2}(Q_{l_j} + \frac{\sum_{i=1}^{N-1} w_i Q_{g_i}}{N-1}). \tag{6}$$

This process is repeated until the final estimates for each agent converge to a single value [9, Section 3.1].

3. Related work

Given the wide applicability of a given solution, an extensive selection of related work is available. Singh et al. [29] and Qu et al. [30] provide a comprehensive analysis of autoscaling web applications in a cloud environment. Additionally, they introduce an expansive taxonomy further distinguishing between metrics, type, policy, and pricing, among other factors. Garì et al. [31] present an extensive survey of reinforcement learning-based autoscalers in particular, differentiating between model-free and model-based, sequential and parallel, as well as deep reinforcement and fuzzy logic learning. They conclude with a classification of the different approaches and provide a taxonomy based on their findings.

However, to the best of our knowledge, no survey exists which addresses and compares approaches in different machine learning paradigms specifically.

4. Comparison between MLScale and RLPAS

In the following chapter we will compare two contrasting approaches to autoscaling in cloud computing environments. The first paper, *"MLscale: A Machine Learning Based Application-Agnostic Autoscaler"* by Wajahat et al. [8], presents an algorithm that predicts key metrics, such as the response time, using simple application-independent inputs and makes autoscaling decisions based on the output. The prediction utilizes a simple neural network, combined with multiple linear regression for the decision making process. The second paper, *"RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment"* by Benifa et al. [9], defines a SARSA-based parallel reinforcement learning algorithm that tries to predict future workload, based upon which it scales the applications proactively.

4.1. MLScale: Autoscaling using Neural Networks and Regression

The algorithm introduced by Wajahat et al. consists of two different prediction algorithms: neural networks and multiple linear regression. In order to build the MLScale algorithm, Wajahat et al. split the program into 3 phases. Initially, the authors trained a neural network on a set of 8 application-independent input metrics $m_1, ..., m_8$, such as RR: number of requests received per second, or CPU: average CPU usage, in order to predict a single performance metric RT: the response time. The network consists of 8 input nodes, one 4-node hidden layer, and one output node. The activation function is $f(x) = \frac{1}{1+e^{-x}}$ and is only applied in the hidden layer.

To automatically scale any given application, MLScale continuously monitors all input metrics $m_1, ..., m_8$ and predicts RT. Should the response time exceed the target or even cause an SLA violation, MLScale will try to provision resources accordingly. To calculate the size of the additional resources, the program has to anticipate how this scaling will affect the new response time $\hat{RT}$. Thus, another prediction is necessary. Deriving from 2, the authors employ a simple multiple linear regression model predicting the new value $\hat{m_i}$ for every metric m_i after scaling:

$$\hat{m_i} = a + b_1 m_i \frac{w}{w+k} + b_2 m_i \frac{k}{w+k}. \tag{7}$$

$w \in \mathbb{N}$ represents the currently deployed workstations, while $k \in \mathbb{Z}$ represents the additional workstations. Positive k is to be understood as a scale-out and negative k as a scale-in. MLScale can not scale vertically.

Using the same network presented above, the predicted metrics $\hat{m_1}, ..., \hat{m_8}$ are then used to calculate the new $\hat{RT}$ after the scaling operation has concluded, based upon which the size of the scaling operation is decided.

4.2. RLPAS: Autoscaling using Parallel Reinforcement Learning

In contrast, Benifa et al. [9] attempt to construct an autoscaler using reinforcement learning. To define the MDP, the authors utilize a state set $S = \{U_{req}, U_{VM}, U_{RT}, U_{THR}\}$, representing the number of requests, percentage of allocated VMs, response time, and throughput. Additionally, the action set $A = \{A_{scale\_up}, A_{scale\_down}, A_{no\_change}\}$ is considered, each $A(VM_n, VM_{type})$ describing an amount VM_n and type $VM_{type} \in \{small, medium, large\}$ to be scaled. VM_n represents horizontal scaling, while VM_{type} represents vertical scaling. The reward

$$R_t = \frac{Perf_{VM}}{U_{VM}} \quad (8)$$

is computed utilizing

$$Perf_{VM} = \frac{RT_{SLA}}{RT_{obs}} + \frac{THR_{obs}}{THR_{SLA}} - Penalty_{RT} - Penalty_{THR}. \quad (9)$$

The agent is thus rewarded for low RT and high throughput compared to the target. Meanwhile, the *Penalty* terms are applied when the agent exceeds the SLA thresholds. They are calculated based on the amount the SLA was violated by, as well as a manual weight. This ensures SLA-compliant behaviour, but also allows the operator to manually tune how severely a violation is to be punished, and thus, how close to the target the agent operates. 8 ensures that the agent does not overprovision VMs. As it covers all functionality laid out in Section 2.1.2, we believe this to be a sensible choice of reward function. In addition, 9 allows easy extension for other metrics.

The authors use a function approximation based on the gradient descent algorithm shown in Section 2.2.3, as well as parallel learning to considerably speed up convergence [9, Figure 9]. Because $q_\star$ is estimated by taking into consideration all future rewards as shown in 3, RLPAS can account for possible future developments of the input metrics. As such, this makes it a proactive algorithm, able to scale applications in preparation for incoming changes in request rate.

4.3. Discussion

Both approaches show very promising results for solving the problem of autoscaling according to user demand. Yet, in both solutions we were able to identify weaknesses which will require further research to address.

Utilizing easily obtainable input metrics, a small feed-forward NN, and MLP, MLScale [8] provides a prediction-based reactive autoscaling algorithm. In essence, Wajahat et al. present an ML-based approach to the traditional manual threshold-based autoscaling, allowing for an automated solution utilizing more metrics than would be possible by hand. For comparison, manual thresholds often rely on as little as one or two metrics [29, Section 5.2], compared to MLScale's 8. It's main advantage lies in this relative simplicity, as the architecture of the neural network can be trained in a few seconds. As the authors mention in [8, Section 3.1], the labeled training data can be acquired by sampling only a few hours worth of normal application behaviour, ideally including a wide variety of workload and scaling actions.

However, this simplicity also leads to its largest deficiencies. For one, the paper only considers a single target metric: response time. Yet, in order to achieve the complex goals set for the algorithm, other performance indicators such as CPU utilization, throughput, or power consumption should also be taken into consideration. While the authors argue in [8, Section 3.1] that the NN could be easily extended to account for these metrics, we expect this to increase both complexity and training time. In addition, the presented method represents a reactive autoscaler, meaning the algorithm does not predict future workloads and can only react to them once they occur. As shown by Wajahat et al. in [8, Section 5], MLScale incurs a substantial (up to 5.9%) amount of SLA violations especially during large and sudden request spikes, also tending to overprovision resources in response. While this still represents an improvement in comparison to traditional scalers and achieves near optimal performance for workloads less prone to spiking [8, Table 5], the problem remains near impossible to solve using reactive methods.

In contrast, the algorithm presented by Benifa et al. [9] is of a proactive nature. As the authors are able to show in [9, Section 4.4] and especially [9, Figure 7], after an initial phase of fluctuation, RLPAS is able to achieve very steady performance metrics even for rapidly changing workloads. Unfortunately the authors do not quantify the number of SLA violations, which would allow a more direct comparision to MLScale. However, as the measurements suggest highly stable target metrics, we have no reason to believe any substantial amount of violations occurred. In addition, this approach is able to outperform competitors in a variety of different measurements, including RT, throughput, and CPU utilization. Importantly, RLPAS can scale both horizontally and vertically, while MLScale is restricted to horizontal scaling. This increases the flexibility of the algorithm and enhances its ability to find a good solution.

However, in comparison with the simple mechanisms used by MLScale, RLPAS is a more algorithmically complex solution. As Sutton points out in [23, Part II], major weaknesses of reinforcement learning tend to be two-fold: (1) poor initial performance as the agent begins to explore the environment and the Q-table values have not yet converged and (2) long convergence time. Especially for very large state spaces, the consequently larger value table can result in very long training times, thus further amplifying the first weakness (see Section 2.2.3). The authors attempt to combat this problem by combining function approximation and parallel learning in order to speed up convergence. Nonetheless, the paper only demonstrates acceptable convergence for up to 16 VMs per application [9, Figure 8] as well as a limited set of 4 state and 3 action variables [9, Section 3.1]. As this convergence issue is a well-known problem faced by the wider RL community, further research is required to show acceptable performance in such situations.

5. Conclusion

In conclusion, both machine learning-based solutions in general, and the analyzed algorithms specifically, provide a marked improvement over manual, threshold-based autoscaling methods. However, as we have shown in the preceding sections, the reactive nature of the MLScale algorithm represents a major weakness preventing its adoption in favor of more advanced, proactive algorithms, such as RLPAS. In addition, RLPAS' ability to predict multiple different important metrics, as well as scale both horizontally and vertically, gives it a further edge when optimizing for complex scenarios. As such, we conclude from our analysis that exploration in the direction of reinforcement learning appears the most promising. Although having already produced encouraging results, we believe further research into the safe and effective scaling is required to achieve industry-wide adoption.

References

[1] P. Mell and T. Grance, “The nist definition of cloud computing,” 2011-09-28 2011.

[2] M. J. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models*, 1st ed., ser. SaaS, PaaS, and IaaS. Wiley, 2014.

[3] Amazon, “Aws pricing,” accessed: 2021-12-02. [Online]. Available: https://aws.amazon.com/pricing/

[4] “Current enterprise public cloud adoption worldwide from 2017 to 2020, by service,” Flexera Software, Mar. 2021, accessed: 2021-12-02. [Online]. Available: https://www.statista.com/statistics/511508/worldwide-survey-public-coud-services-running-applications-enterprises/

[5] Statista, “Cloud computing market size in europe from 2016 to 2025, by segment,” Aug. 2021, accessed: 2021-12-02. [Online]. Available: https://www.statista.com/forecasts/1235161/europe-cloud-computing-market-size-by-segment

[6] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st ed. Addison-Wesley Professional, 2015.

[7] Oracle, “Oracle autoscaling,” 2022, accessed: 2022-02-25. [Online]. Available: https://docs.oracle.com/en-us/iaas/Content/Compute/Tasks/autoscalinginstancepools.htm

[8] M. Wajahat, A. Karve, A. Kochut, and A. Gandhi, “Mlscale: A machine learning based application-agnostic autoscaler,” *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 287–299, 2019. [Online]. Available: https://doi.org/10.1016/j.suscom.2017.10.003

[9] B. Benifa, J. V., and D. Dejey, “Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment,” *Mobile Networks and Applications*, vol. 24, pp. 1348–1363, 2019. [Online]. Available: https://doi.org/10.1007/s11036-018-0996-0

[10] R. N. Jim Smith, *Virtual Machines - Versatile Platforms for Systems and Processes*, 1st ed., ser. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2005.

[11] K. Gray and T. D. Nadeau, *Network Function Virtualization*. Elsevier Science & Technology; Morgan Kaufmann, 2017.

[12] W. T. Joe M. Butler, Ramin Yahyapour, *Service Level Agreements for Cloud Computing*, 1st ed. Springer-Verlag New York, 2011.

[13] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, Apr. 2010.

[14] T. M. Mitchell, *Machine Learning*, 1st ed., ser. McGraw-Hill series in computer science. McGraw-Hill, 1997.

[15] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021.

[16] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, “Deep learning advances in computer vision with 3d data: A survey,” *ACM Comput. Surv.*, vol. 50, no. 2, apr 2017. [Online]. Available: https://doi.org/10.1145/3042064

[17] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, Apr. 2020.

[18] D. Silver *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, pp. 354–359, 2017.

[19] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. O’Reilly Media, 2017.

[20] K. Gurney, *An Introduction to Neural Networks*. UCL Press, 1997.

[21] G. Seber and A. Lee, *Linear Regression Analysis*, 2nd ed., ser. Wiley Series in Probability and Statistics. Wiley, 2003.

[22] R. Bellman, “A markovian decision process,” *Indiana University Mathematics Journal*, vol. 6, pp. 679–684, 1957.

[23] R. S. Sutton, *Reinforcement Learning: An Introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[24] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, “Convergence results for single-step on-policy reinforcement-learning algorithms,” *Machine learning*, vol. 38, no. 3, pp. 287–308, 2000.

[25] Y. Mansour and S. Singh, “On the complexity of policy iteration,” *UAI*, 01 2013.

[26] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning*. MIT Press, 2016.

[27] A. Haider, G. Hawe, H. Wang, and B. Scotney, “Gaussian based non-linear function approximation for reinforcement learning,” *SN Computer Science*, vol. 2, no. 3, pp. 1–12, 2021.

[28] R. M. Kretchmar, “Parallel reinforcement learning,” in *The 6th World Conference on Systemics, Cybernetics, and Informatics*. Citeseer, 2002.

[29] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, “Research on auto-scaling of web applications in cloud: Survey, trends and future directions,” *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, 2019.

[30] C. Qu, R. N. Calheiros, and R. Buyya, “Auto-scaling web applications in clouds: A taxonomy and survey,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018.

[31] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. G. Garino, “Reinforcement learning-based application autoscaling in the cloud: A survey,” *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104288, 2021.