DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems

*Reproducing results from the paper

Venkata Divya Sree Pulipati
Computer and Information Technology
Purdue University
West Lafayette, USA
vpulipat@purdue.edu

Kevin Varghese Chittilapilly
Computer and Information Technology
Purdue University
West Lafayette, USA
kchittil@purdue.edu

***Abstract*—In the realm of cloud computing, service providers traditionally allocate excess resources to meet Service Level Objectives (SLOs), often opting for lower CPU utilization targets to uphold service quality amid fluctuating workloads. This practice, though, can lead to resource inefficiencies and increased power consumption in large-scale deployments. This paper presents an innovative approach, DeepScaling, designed to minimize resource costs in dynamic, large-scale microservice environments while ensuring SLO requirements are met. DeepScaling introduces three key components: a Spatio-temporal Graph Neural Network for workload forecasting, a Deep Neural Network for estimating CPU utilization considering various cloud environment factors, and an improved Deep Q Network (DQN) for generating adaptive autoscaling policies. These policies dynamically adjust the target CPU utilization to a stable level, ensuring SLO compliance with minimal resource utilization. This paper aims to reproduce this paper's [1] results with a new cloud environment to see how well do the paper solution generalize in other cloud based architectures. The current paper uses Alibaba Google Trace, 2022 for their evaluation**

***Index Terms*—Deep Scaling, Workload Prediction, CPU Utilization, STGNN (Spatio Temporal Graph Neural Network, DNN (Deep Probabilistic Neural Network)**

I. Introduction

Large-scale cloud systems face the challenge of balancing various concerns, including ensuring user quality of experience, appropriately provisioning resources without overprovisioning, and adapting to significant workload variations. Prior studies have highlighted the wastage of power resources in servers with chronically low CPU utilization, prompting the need for automatic scaling systems to maintain CPU utilization at desired levels and meet Service Level Objectives (SLOs). Existing cloud autoscaling approaches can be categorized as rule-based or learning-based schemes, with the latter gaining traction due to their adaptability to dynamic environments. However, these approaches often set lower CPU utilization targets to prevent service quality degradation, leading to potential resource wastage and excessive power consumption. To address these challenges, this paper introduces DeepScaling, a novel framework designed to minimize resource costs while ensuring SLO requirements are met in a dynamically varying, large-scale production microservice environment. DeepScaling incorporates innovative components, including a Spatio-temporal Graph Neural Network for workload forecasting, a Deep Neural Network for estimating CPU utilization, and an improved Deep Q Network for generating autoscaling policies. By adaptively refining the target CPU utilization to maintain a stable value while using minimum resources, DeepScaling outperforms existing autoscaling approaches in terms of maintaining stable service performance and resource savings. The practical applicability and scalability of DeepScaling are demonstrated through its deployment in a real production environment, making it a promising solution for large-scale cloud deployments.

II. Key Challenges

The paper also identifies some key challenges in the paper which are described in this section.

A. Workload Forecasting Challenge

Accurately forecasting the workload of different services is difficult, especially in complex cloud-native microservice architectures. Existing approaches often rely on simpler regression techniques for workload prediction, which are inadequate for capturing the intricate interactions, including those between microservices.

To address this challenge, the proposed solution suggests using Spatio-temporal graph neural networks (STGNN) for workload forecasting. STGNN models the interactions within the service call-graph and considers the complex multi-variant relationships among various workload metrics. This approach allows for a more accurate workload forecast, providing extra time to proactively initiate resources when scaling up, ultimately improving the efficiency of the autoscaling process.

B. Workload Characterization

It is challenging to accurately represent a service's workload in a way that reflects its CPU utilization properly. Attempting to predict CPU utilization solely based on historical CPU data results in significant forecasting errors, often exceeding 30%

in trials. To overcome the challenge of accurate workload characterization, the solution collects multi-dimensional workload metrics for each service. These metrics encompass various aspects of the workload, including RPC requests, file I/O operations, database access, message requests, HTTP requests, and specific auxiliary features like instance count, service ID, and timestamp. Collecting this comprehensive set of metrics aims to provide a more accurate representation of the service's workload.

C. Resource Allocation

To estimate CPU utilization accurately, especially under special conditions like periodic tasks and internal system events, the solution designs a deep probabilistic network-based model. This model is specifically developed to handle the complexities and variability in CPU consumption, ensuring more reliable CPU utilization estimations. Autoscaling aims to dynamically allocate resources (e.g., CPU, memory, instances) to services based on their current workload to ensure optimal performance and cost-effectiveness. However, determining the exact resource requirements for each service can be challenging. This is particularly true when the relationship between the number of service pods (instances) and CPU utilization is complex and non-linear.

The paper proposed to use Deep Q-Network (DQN), a type of reinforcement learning model. In the context of autoscaling, DQN can be used to make decisions on resource allocation based on historical data and reinforcement learning techniques.

III. Background and Related Work

The authors in Sun et al. [2] analyze the usage of a YARN cluster at Alibaba, and report that resource utilization is less than 55% about 80% of the time. Lu et al. [3] analyzes an Alibaba trace, showing that for most instances, the peak resource utilization is 80% or less. Existing autoscaling approaches can be broadly classified into rule-based and learning-based schemes. Rule-based autoscaling schemes use static upper and lower thresholds for certain system or application performance metrics, while learning-based schemes adapt to dynamic environments by using machine learning techniques to predict future resource requirements. However, existing autoscaling approaches often set lower CPU utilization targets to prevent service quality degradation, leading to potential resource wastage and excessive power consumption. [4]

FIRM (Anomaly Detection and Scaling) employs machine learning techniques to detect performance anomalies in services, such as when the response time of a microservice becomes unusually long. When such anomalies are detected, FIRM scales up by adding more service pods or computing resources. However, a significant limitation of FIRM is that it only triggers autoscaling after performance anomalies occur. This can result in extended periods of service anomalies because autoscaling can take time, especially at large scales. Additionally, FIRM lacks a mechanism to scale down resources when allocated more than necessary. [5]

Autopilot (CPU-Based Scaling) [6] uses the time-series of CPU utilization as input and employs a simple heuristic mechanism to determine the target CPU utilization. It then calculates the number of pods required as a linear function of this target utilization. The goal is to minimize the risk of service anomalies. However, Autopilot suffers from relatively poor system stability due to two key reasons:

1) The relationship between CPU utilization and computing resources is often non-linear, whereas Autopilot relies on a linear assumption.
2) Different microservices have varying demands for computing and I/O resources, leading to differing relationships between CPU utilization and resource requirements

To address these challenges, this paper proposes DeepScaling, a novel autoscaling framework for microservices that seeks to adapt to workload variations by estimating resource requirements at a fine granularity and maintaining resource utilizations consistently at a target level to meet SLOs while reducing over-provisioning. DeepScaling comprises three innovative components: a Spatio-temporal Graph Neural Network for workload forecasting, a Deep Neural Network for estimating CPU utilization, and an improved Deep Q Network for generating autoscaling policies. The proposed approach achieves its goal of each service running at a stable resource utilization around the refined target level, even when the workload for the service varies greatly. The paper demonstrates the practical applicability and scalability of DeepScaling through extensive experiments on a production cloud service, showing significantly improved performance compared to state-of-the-art autoscaling approach. [1]

IV. System Architecture

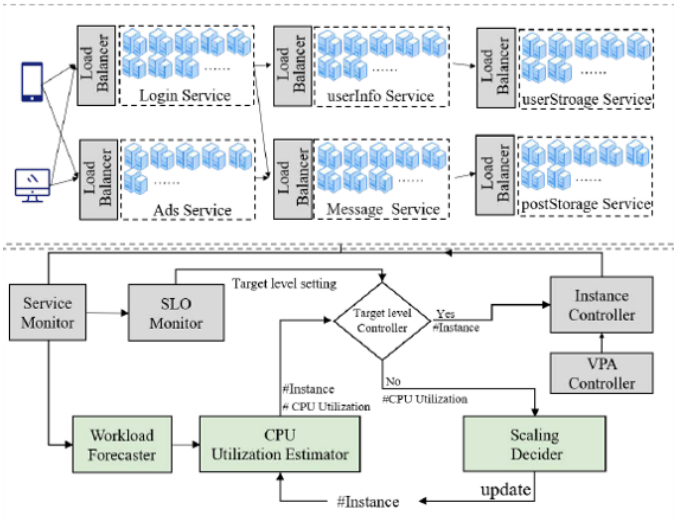

Fig. 1. System Architecture

Here is the description of how each system interacts with each other.

1) The Load Balancer1 distributes requests so that each instance of a service, and thus each pod of a service, carries approximately the same workload and has the same CPU utilization.
2) The Service Monitor focuses on collecting metrics for all services in real time
3) Workload Forecaster: We analyze the call-graph and the main workload metrics for each microservice. Then, a spatial-temporal graph neural network (STGNN) is used to forecast the workload metrics for each microservice.
4) CPU Utilization Estimator: This module estimates CPU consumption based on 7 workload metrics along with 3 specific auxiliary features. Takes inputs from the workload forecaster and Scaling Decider to generate CPU utilization estimates
5) Scaling Decider: In this module, a reinforcement learning (RL) model is adopted to generate autoscaling policies. Collaborates with the CPU Utilization Estimator to determine the optimal number of service instances.
6) Target level Controller: The CPU target level is a core parameter value for DeepScaling. The initial value of is the maximum CPU utilization value seen historically for normal execution of the service in the past. DeepScaling provides a buffer value (5%) to allow for possible model errors and other noise. It determines the desired CPU utilization level for a specific microservice within the DeepScaling system. It represents the optimal or target CPU utilization value that the system aims to maintain to meet performance goals while efficiently utilizing resources.
7) SLO Monitor (service level objectives: The SLO monitor determines whether a core microservice violates its SLO by monitoring the microservice response time (RT), GC (garbage collection) time, I/O RT, and other metrics. When the SLO monitor detects an abnormal value of a metric for the core microservice, it notifies the target level controller to lower the target level for that service, in addition to changing the controller state.
8) The Instance Controller adjusts the service resources of each microservice by increasing or decreasing the number of standard pods.
9) Vertical Pod Autoscaler (VPA) Controller: managing the configuration of Kubernetes pods; VPA Controller allows for customization of pod resource requirements

V. Dataset Exploration

We have used the Alibaba Cloud Trace 2022 microservices for our analysis. [7]. This data comprises trace data from a microservices architecture, categorizing microservices into stateless and stateful services, with communication occurring through inter-process communication, remote invocation, and indirect communication paradigms. The dataset encompasses four main components: Node, MSResource, MSRTMCR, and MSCallGraph.

The Node data includes runtime information for over 40,000 Bare Metal (BM) nodes in a production cluster, recording CPU and memory utilization at 60-second intervals. The MSResource data focuses on microservice (MS) runtime information, detailing CPU and memory utilization for over 470,000 containers across 28,000 MSs in the same cluster.

MSRTMCR provides Microservice Call Rate (MCR) and Response Time (RT) information, capturing MCR and RT for calls among 28,000 MSs with 470,000 containers, using various communication paradigms such as provider/consumer RPC, write/read to Memcached (MC) and Database (DB), and message queue (MQ) operations.

The MSCallGraph component contains information about more than twenty million call graphs among 17,000 MSs in over ten clusters. It includes details like timestamp, trace ID, service IDs, RPC IDs, UM and DM names, container IDs, communication paradigms, interfaces, and response times.

Each dataset component is timestamped, allowing users to analyze performance metrics and interactions over time. The data is organized into directories, and users can download specific intervals using a provided script. Additionally, the dataset distinguishes between stateless and stateful services, where stateful services store data in databases and Memcached. The dataset's richness allows for in-depth exploration of microservices behavior, resource utilization, and communication patterns within a large-scale production environment.

VI. Proposed Work

The authors from the DeepScaling paper [1] evaluate their pipelined frame work for auto scaling resources on a singular cloud entity adopted in ANT group and displayed results that their approach for Workload predictions, CPU utilisation estimations and then scaling the resources based on the appropriate scaling policy is efficient.

A. Paper's Implementation for Workload Metrics

The paper measures workload metrics for workload predictions by collecting 7 commonly used CPU consumption metrics to characterize each workload. These workload metrics are generated from the active execution of the complete microservice system. The seven workload metrics are:

1) RPC-in: Counts the number of incoming RPC requests for the microservice and reflects the CPU consumption for completing a specific logical function.
2) RPC-out: Counts the number of RPC requests from each microservice to other microservices in the task chain and is usually proportional to the RPC-in metric.
3) Msg-pub: Counts the number of message publications to the middle-ware message broker.
4) Msg-sub: Counts the number of message subscriptions from the middle-ware message broker.
5) DB-Access: Counts the number of database accesses (query, insert, delete, etc.).
6) Page-View (PV): Counts the number of dynamic/static web page views resulting from HTTP requests in a time interval.
7) File I/O: Counts the number of file read and write operations.

The SLO monitor captures these seven workload metrics for each service every minute, yielding seven corresponding time-series.

These metrics are then passed to an STGNN (Spatio-Temporal Graph Neural Network) for predictions of the metrics in another time instance. A Spatio-Temporal Graph Neural Network (STGNN) is a type of neural network architecture designed to model and analyze complex relationships within spatio-temporal data. This framework is particularly useful for tasks involving data that varies across both space and time, such as in the domains of video analysis, traffic prediction, or climate modeling. STGNNs leverage graph neural networks to capture spatial dependencies and temporal patterns simultaneously. Nodes in the graph represent spatial entities, such as locations or objects, while edges denote the spatial relationships between them. The temporal aspect is incorporated by considering the evolution of these entities over time. This combination of spatial and temporal information enables STGNNs to effectively learn and predict dynamic patterns in complex datasets such as microservice architecture that we are using, making them valuable tools for tasks requiring a comprehensive understanding of both spatial and temporal dynamics.

B. Our Implementation for Workload Predictions

For the use of STGNN we needed to create an Adjencency Matrix that captures the information regarding the interconnection between different nodes. However due to large size of the dataset, we were only able to focus on one node. Hence we are not able to create the complete matrix. We tried creating the matrix for one node, however the matrix were mostly zeros since only a handful of microservices within the node were talking to one another. Due to this reason we were not able to implement STGNN as proposed by the paper.

So we decided to test the paper's hypothesis that a Neural Network works well for workload prediction. By using the same proposed metrics in the paper, we built a Neural Network Architecture using LSTM (Long Short Term Memory). We used 64 hidden layers and trained data for 100 epochs. We have expanded on the results in the Results Section.

C. Paper's Implementation for CPU Utilization

The paper computes CPU utilization using a deep probabilistic regression network that models the nonlinear relationship between the workload metrics and the CPU utilization. The input to the network is the metric value from the workload forecaster and the latest number of instances given by the Scaling Decider. The output is the estimated CPU utilization .

The deep probabilistic regression network is trained using historical data of workload metrics and CPU utilization. The network is designed to accurately estimate CPU consumption even under special conditions, such as with periodic tasks and internal system events. However different microservices have different behaviors for each workload hence we also add introduce certain microservice specific auxiliary features to enhance the general applicability of the shared mode.

1) Instance-count: the number of instances (or pods) for each microservice, ranging from 1 to max-instance
2) Service-ID: the unique identifier of each microservice, ranging from 1 to max-instance
3) Time-stamp: the time-stamp during the day, in minutes, when the workload metrics are collected/forecast, ranging from 1 to 1440 (24hours).

D. Our Implementation of CPU Utilization Metrics

We were able to successfully re-implement Deep Probabilistic Neural Network for our case. Our results show that the paper's claim regarding CPU Utilization were correct. However, our implementation was limited to one Node due to the large size of the dataset. Deep Probabilistic Network works really well in this scenario since there can be sudden shifts in the CPU Utilization values and a normal Neural architecture will not be able to handle these frequent fluctuations.

E. Paper's Implementation for Autoscaling Decision Making

The function of the decision maker for autoscaling is to find the recommended number of instances for a microservice using RL (Reinforcement Learning) Model. We consider the variation of CPU utilization during the running of a microservice as the environment and take the CPU utilization estimation model as the environment observer. RL model will continually autoscale the instance count of pods based on the environment observer. The goal of the RL model is to keep the average CPU utilization stable despite variations in the workload.The input of the RL model is a tuple, consisting of the CPU utilization and the service-ID and outputs the latest instance count after performing the recommended action. Then, we use the CPU utilization estimation model to estimate the new CPU utilization and update the state . This process loops until the CPU utilization reaches the pre-set target level. The RL model outputs the instance count as the final result. Here are some important details regarding the RL Model.

1) State Space: The environment state includes both the CPU utilization of the service and the service-ID.
2) Action Space: The action space consists of three different actions: increased, decreased, and unchanged.
3) A Deep Q-Network (DQN) is used to solve the policy optimization problem (p : S -¿ A), where S is the state space, and A is the action space.
4) DQN consists of two networks, MainNet and TargetNet, with the same structure but different parameters.
5) MainNet evaluates the value function of actions a based on the latest parameters.
6) TargetNet provides target values to prevent overfitting.
7) TargetNet's parameters are a time-delayed version of MainNet, and parameters are copied from MainNet at fixed intervals.

F. Our Implementation for Autoscaling Decision Making

Due to time and compute resource limitations we were not able to re-implement RL Model for Autoscaling Decision Making.

VII. Experimental Setup

We limit the scope of our paper to reproducing the results from the first two components of the framework in DeepScaling paper [1] i.e. workload predictions and CPU estimations. The first step of our experiment is to obtain datasets which holds various work load metrics of various services running on a node and their CPU utilisation's.

According to DeepScaling paper [1], the authors looked at 7 different ways to measure CPU usage to characterize each workload. These metrics help them understand how much CPU each microservice uses when the whole system is running.Each microservice performs its own task and communicates with other microservices. measurements (metrics) are taken while the whole system of microservices is up and running.

The seven workload metrics are: RPC-in, RPC-out, Msg-pub, Msg-sub, DB-Access, Page-View(PV), File I/O. So, our paper used Alibaba Google Trace 2022 microservices as our dataset as it contains most of the metrics described in the DeepScaling paper [1]. The equivalent of the metrics RPC-in, RPC-out, Msg-pub, Msg-sub, DB-Access, Page-View(PV) in Alibaba Google Trace are consumerrpc_mcr, providerrpc_mcr, consumermq_mcr, providermq_mcr, writedb_mcr, http_mcr respectively.

The data set been collected over 13 days and total size is about 2TB. Size of data for an hour is around 52GB. For our project, we have focused on data for an hour due to it's large size. We focused on Node27596 for our analysis. We used Google Colab and VSCode for running our code.

For Workload Predictions we have used a Neural Network based on LSTM (Long Short Term Memory) with 64 hidden layers and trained the model for 100 epochs. We also used Standard Scaler for our input metrics to scale our data. Since we are dealing with a time series dataset, we have used train test split without random sampling to split our data between train and test.

For CPU Utilization Prediction we have used Deep Probabilistic Neural Network. We have used a DistributionLambda Layer in our architecture that wraps a function that takes parameters as input and returns a probability distribution. The layer then applies this function to the inputs during the forward pass. We also add MultivariateNormalDiag which is a class from TensorFlow Probability representing a multivariate normal distribution with a diagonal covariance matrix. Overall this outputs a probability distribution, specifically a multivariate normal distribution with a diagonal covariance matrix. We have also used Standard Scaler and train test split for the data and trained the model for 10 epochs

VIII. Results and Discussion

A. Workload Predictions

Here are the key results for the workload prediction metrics. In all the images the blue lines are the true values and the orange lines are the predicted values from the model.

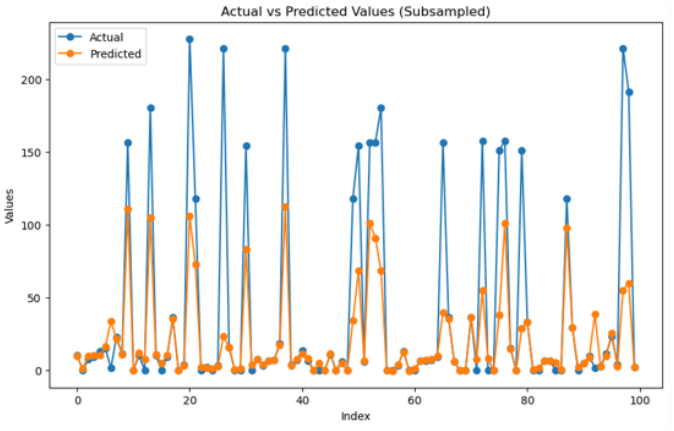

Fig. 2. WorkLoad Predictions

The given output is the prediction of the model for Provider-RPC values. We can obtain similar graph for each of the input metrics at the required time instance. We can observe that the model has a hard time predicting the sudden peaks in the value but it performs reasonably well with shorter bursts of data. We can attribute this to the lack of adjacency matrix and not using attention layers within our model. But seeing how the model is able to generalize over the overall trend of the matrix, we can conclude that neural network work well for predicting workloads.

B. CPU Utilization

Here are the key results for the CPU Utilization.

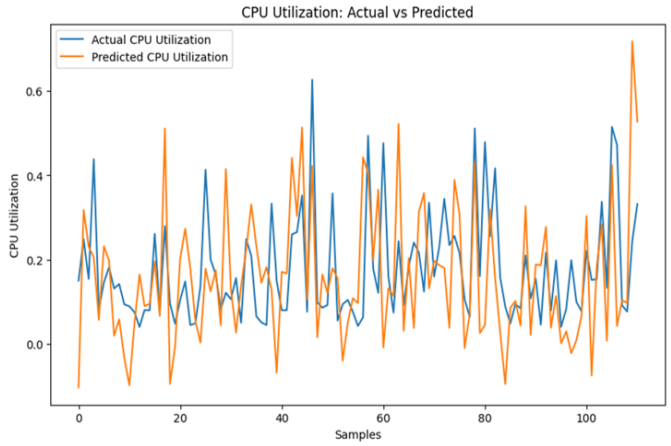

Fig. 3. CPU Utilization for 1 epoch

We can observe in this image how the model has a hard time adjusting to the various fluctuations in the dataset in the first epoch. However, it still is able to generalize in the overall

trend of the CPU Utilization. This gives the intution that the model might perform better with more epochs

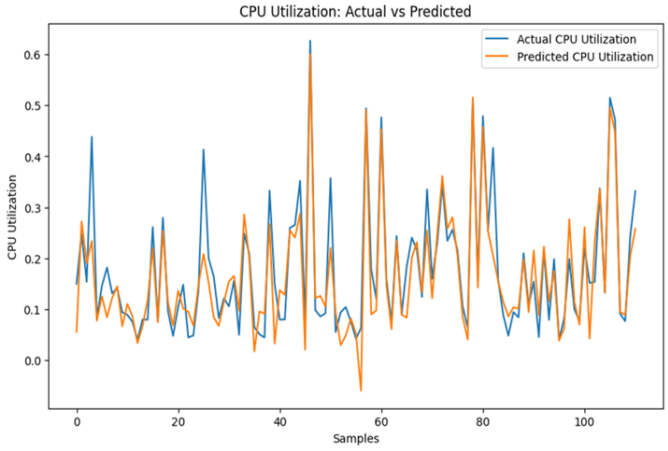

Fig. 4. CPU Utilization for 10 epochs

We can observe that after 10 epochs the model performs far better. It is able to predict the workload metric with much more certainty. It also able generalize well with sudden changes in the values. As mentioned earlier this results can be attributed to the nature of Probabilistic Neural Networks which work well for sudden changes. Another insight that we can draw is the problem of overfitting. Since we have limited data from only one Node in one hour, this result is expected. However once more data will be available, we assume that the overfitting issue will be resolved. Based on these results we can accept the authors premise of using DNN for the task of CPU Utilization.

IX. Conclusions

Based on our experiments and analysis we can conclude that both STGNN and DNN work well for workload prediction and CPU Utilization respectively. However our analysis was limited to only one Node and a limited time frame. Since the predictions are made using a neural network, we believe providing more data by considering historic data on other nodes for a long period can help us increase the accuracy of the results obtained. However, further analysis and evaluation would be needed to accurately determine if the setup would work well in an interactive microservices architecture.

References

[1] Wang, Z., Zhu, S., Li, J., Jiang, W., Ramakrishnan, K. K., Zheng, Y., ... and Liu, A. X. (2022, November). DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems. In Proceedings of the 13th Symposium on Cloud Computing (pp. 16-30).

[2] Xiaoyang Sun, Chunming Hu, Renyu Yang, et al. 2018. Rose: Cluster resource scheduling via speculative over-subscription. In IEEE International Conference on Distributed Computing Systems (ICDCS). 949–960

[3] Chengzhi Lu, Kejiang Ye, Guoyao Xu, et al. 2017. Imbalance in the cloud: An analysis on alibaba cluster trace. In IEEE International Conference on Big Data. 2884–2892.

[4] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, et al. 2012. Optimal autoscaling in a IaaS cloud. In International conference on Autonomic computing. 173–178.

[5] Haoran Qiu, Subho S Banerjee, Saurabh Jha, et al. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLOOriented Microservices. In USENIX Symposium on Operating Systems Design and Implementation (OSDI). 805–825.

[6] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, et al. 2020. Autopilot: workload autoscaling at Google. In European Conference on Computer Systems (EuroSys). 1–16.

[7] Luo, Shutian, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. "The power of prediction: Microservice auto scaling via workload learning." In Proceedings of the 13th Symposium on Cloud Computing, pp. 355-369. 2022.