Graph-PHPA: Graph-based Proactive Horizontal Pod Autoscaling for Microservices using LSTM-GNN

Hoa X. Nguyen, Shaoshu Zhu, and Mingming Liu
School of Electronic Engineering, SFI Insight Centre for Data Analytics, Dublin City University, Ireland
mingming.liu@dcu.ie

***Abstract*—Microservice-based architecture has become prevalent for cloud-native applications. With an increasing number of applications being deployed on cloud platforms every day leveraging this architecture, more research efforts are required to understand how different strategies can be applied to effectively manage various cloud resources at scale. A large body of research has deployed automatic resource allocation algorithms using reactive and proactive autoscaling policies. However, there is still a gap in the efficiency of current algorithms in capturing the important features of microservices from their architecture and deployment environment, for example, lack of consideration of graphical dependency. To address this challenge, we propose Graph-PHPA, a graph-based proactive horizontal pod autoscaling strategy for allocating cloud resources to microservices leveraging long short-term memory (LSTM) and graph neural network (GNN) based prediction methods. We evaluate the performance of Graph-PHPA using the Bookinfo microservices deployed in a dedicated testing environment with real-time workloads generated based on realistic datasets. We demonstrate the efficacy of Graph-PHPA by comparing it with the rule-based resource allocation scheme in Kubernetes as our baseline. Extensive experiments have been implemented and our results illustrate the superiority of our proposed approach in resource savings over the reactive rule-based baseline algorithm in different testing scenarios.**

***Index Terms*—microservices, autoscaling, predictive method, resource management, Graph Neural Network.**

I. Introduction

Microservices is a new architectural approach that can be applied to cloud-native applications consisting of a collection of small software services. In a microservice architecture, each service has its own functionality, and they jointly contribute to the whole operation of an application deployed on the cloud. Such an application is loosely decoupled into several small services which can be independently deployed on a potentially different platform and technological stack [1]. This architectural style allows cloud resources to be flexibly allocated to each service rather than simply allocating all precious resources to a monolithic application which is often less efficient. More specifically, cloud resources, such as vCPUs, cores and RAMs can be requested and allocated in a scalable and containerised manner both horizontally and vertically providing a flexible control strategy for effective resource management, particularly in a deployment environment subject to some specific service-level-objectives (SLOs), e.g., matching workload with the required resource,.

Proactive methods, such as machine learning methods, are being widely developed to harmonise resource provisioning for microservice applications. These methods allow different services to be dynamically scaled in a predictive manner, and recently they have demonstrated higher efficiency in scaling and faster response compared to many reactive methods for cluster operations [2]–[5]. More specifically, machine learning algorithms can be applied not only to model the tendency of workloads but also to capture patterns for resource consumption in microservice applications. Furthermore, by leveraging multiple machine learning algorithms, functionalities can complement each other and the benefits can be combined towards a more efficient autoscaling strategy for cloud resources, which has become an emerging research area in recent years. All the above studies so far, however, have not utilized graph-based approaches. Very limited work has been found using graphs to capture important features of microservices from their architecture and deployment environment [6], [7].

Motivated by the fact that Graph Convolution Networks (GCNs) have been widely used in other domains and applications, such as transportation and energy [8], [9], with promising efficacy in describing potential graphical dependency between entities in the network, we aim to leverage a graph-based approach in this paper to design a proactive horizontal pod autoscaling strategy for microservices. With this in mind, the main contributions of our work can be summarized are as follows.

1. We propose Graph-PHPA, a two-stage prediction method using both Long Short-term Memory (LSTM) and GCN, where LSTM is used for workload prediction and GCN is used to model the relation between workload and resource consumption for different microservices in the network.
2. We evaluate the performance of Graph-PHPA in a dedicated testing environment composing components from Amazon Web Services [10], Prometheus [11] , Grafana [12] and the Bookinfo application [13], using real-time workload generated based on a realistic dataset in [14].
3. We demonstrate the superiority of Graph-PHPA over the default reactive rule-based method in Kubernetes through extensive experimental studies in different setups.

The rest of this paper is organised as follows. The problem statements and the mechanisms of predictive models are introduced in Section II, where the proposed solutions are discussed in detail. Experimental setups are introduced in Section III and results are discussed in Section IV. Finally, the conclusion for the current research and potential future research directions are outlined in Section V.

II. Problem formulation and Models

A. Problem statement

We consider a scenario where n services are deployed in a Kubernetes cluster with constrained resources for vCPU share. The vector, $\mathbf{N}(t) := [N_1^{(t)}, N_2^{(t)}, ..., N_n^{(t)}]$, denotes the number of pods for each of the microservice at time t. Note that $1 \leq N_i^{(t)} \leq Q_i, \forall i = 1, ..., n$, where Q_i is the upper bound of pods that the microservice i can have. The vCPU share vector, $\mathbf{R}(t) := [R_1^{(t)}, R_2^{(t)}, ..., R_n^{(t)}]$, represents the vCPU for each microservice at time t. Note $R_i^{lb} \leq R_i^{(t)} \leq R_i^{ub}, \forall i = 1, ..., n$, where R_i^{ub} and R_i^{lb} denote the upper and lower vCPU bounds.

Our key objective is to find out the number of pods $\mathbf{N}(t)$ for microservices based on their vCPU shares $\mathbf{R}(t)$ considering the dependence among microservices and the workload input. Once the workload changes, the system can dynamically add resources for the microservices to ensure the application's performance or reduce unnecessary replicas to save operational cost. In the following sections, we shall introduce the design of predictive models in detail, which consists of the predictive model for workload, the predictive model for resource usage, and the integrated model, i.e., the proposed Graph-PHPA.

B. Predictive models

a) Predictive model for Workload: Let $a_i^{(t)}$ denote the workload of microservice i at time t. Let $\mathbf{a}_i^{(t)} := [a_i^{(t-k+1)}, ..., a_i^{(t-1)}, a_i^{(t)}]$ denote the workload feature vector of the microservice i at time t for a given time window with size k, where $t \in [k, T]$ and T denotes the length of input data. Given the vector $\mathbf{a}_i^{(t)}$, we aim to forecast the workload $\widetilde{a_i^{(t+1)}}$ of microservice i at time $(t+1)$. The LSTM model [15] is applied here to address the time series prediction problem. More specifically, we construct the LSTM model consisting of multi-layers. The cells of each layer are connected with each subsequent one. The input layer receives the sliced data for the corresponding time window. The last layer is a dense layer using the "tanh" activation function to produce the predicted workload value. Mathematically, we wish to find a learning function $\psi(.) : \mathbb{R}^k \mapsto \mathbb{R}$ which is able to address the following optimization problem through a multi-layer LSTM network:

$$\begin{aligned} &\underset{\psi}{\text{minimize}} \sum_{t=k}^{T-1} (a_i^{(t+1)} - \widetilde{a_i^{(t+1)}})^2 \\ &s.t. \quad \psi(\mathbf{a}_i^{(t)}) = \widetilde{a_i^{(t+1)}}. \end{aligned} \tag{1}$$

b) Predictive Model for Resource usage: We leverage Graph Convolution Network (GCN) to accommodate the graphical structure of the application and the dependency of microservices. The graph structure of microservices for testing application is assumed to be static and deterministic throughout our experiments. The goal of the model is to learn a function of features on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ which takes features of each node $x_i : x \in X^{N \times D}$ and a representative description of the graph structure in adjacency matrix $A \in \mathrm{R}^{N \times N}$, where X is the feature matrix, N is the number of nodes $v_i \in \mathcal{V}$, edges between nodes $(v_i, v_j) \in \mathcal{E}$, D is the number of input features. The output $Z^{N \times F}$ is a node-level feature, where F is the number of output features. In each layer of L number of layers, the output of the layer is $H^{(l+1)} = \sigma(H^l, A)$, where $H^{(0)} = X$ and $H^{(L)} = Z$. The chosen activation function $\sigma(.,.)$ and the parameters decide the differences among the GCN models. We use the layer-wise propagation rule introduced in [16], [17] as follow:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right), \tag{2}$$

where, $\tilde{A} = A + I_N$ is the adjacency matrix of the graph $\mathcal{G}$. I_N which is the identity matrix representing self connections of each node. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the diagonal node degree matrix and $W^{(l)}$ is a layer specific trainable weight matrix. $\sigma(.)$ represents an activation function, for example the $ReLU(.) = max(0, .)$. $H^{(l)} \in \mathrm{R}^{N \times D}$ is the matrix of activation in the l^{th} layer, $H^{(0)} = X$ in the first layer. Let $b_i^{(t)} \in \mathbb{R}$ be the maximum resource consumption of microservice i at time t over a time window of size k. In other words, $b_i^{(t)}$ represents the maximum amount of resource required for microservice i given the workload $\mathbf{a}_i^{(t)}$. Our objective is to find a learning model $\phi(.) : \mathbb{R}^k \mapsto \mathbb{R}$ which can address the following optimization problem:

$$\begin{aligned} &\underset{\phi}{\text{minimize}} \sum_{t=k}^{T-1} (b_i^{(t+1)} - \widetilde{b_i^{(t+1)}})^2 \\ &s.t. \quad \phi([a_i^{(t-k+2)}, ..., a_i^{(t)}, \widetilde{a_i^{(t+1)}}]) = \widetilde{b_i^{(t+1)}}. \end{aligned} \tag{3}$$

where, $\widetilde{b_i^{(t+1)}}$ is the predicted maximum amount of resource for microservice i at $t+1$ over a time window of size k. The prediction model takes the historical workload $[a_i^{(t-k+2)}, \ldots, a_i^{(t-1)}]$, current workload $a_i^{(t)}$ and the predicted workload $\widetilde{a_i^{(t+1)}}$ from (1) as input and outputs the desired amount of resource, i.e., vCPU share, for the given microservice i for the next time step.

c) Integration algorithm: Algorithm 1 presents the proposed Graph-PHPA algorithm, which assembles the two predictive models. The algorithm works as follows. At each time step t, $\mathbf{a}_i^{(t)}$ is collected from the monitoring system and is used to predict $\widetilde{a_i^{(t+1)}}$. Then, the model ϕ predicts $\widetilde{b_i^{(t+1)}}$ using $[a_i^{(t-k+2)}, ..., a_i^{(t)}, \widetilde{a_i^{(t+1)}}]$. By that, resource prediction model accounts for the future workload. For example, current resource of a microservice is $R_i^{(t)} = 2.0$ vCPU at t. However, the predicted resource $\widetilde{b_i^{(t+1)}}$ requires 2.5 vCPU, which implies that an extra 0.5 vCPU is required to be prepared for the service i at t for the upcoming workload at time $(t+1)$ as per the prediction. Thus, 0.5 vCPU, e.g., 1 pod (assuming 0.5 is the unit specification), is to be provisioned at time t to avoid delay in the initialization process of pods in practical operations. Therefore, the number of pod $N_i^{(t+1)}$ is updated accordingly based on the predicted resource $\widetilde{b_i^{(t+1)}}$ as per the resource specification for each microservice i denoted by v_i^p.

Algorithm 1 Integration Algorithm - Graph-PHPA

Input: Window size for forecasting (k), learned workload prediction model $\psi : \mathbf{a}_i^{(t)} \rightarrow \widetilde{a_i^{(t+1)}}$, learned resource prediction model $\phi : [a_i^{(t-k+2)}, ..., a_i^{(t)}, \widetilde{a_i^{(t+1)}}] \rightarrow \widetilde{b_i^{(t+1)}}$, allocated vCPU resource to microservice $R_i^{(t)}$, standard vCPU resource specification of microservice i for one pod v_i^p, allocated number of pods for the microservice i, $N_i^{(t)}$.

Output: Predicted vCPU shares $R_i^{(t+1)}$ and number of pods $N_i^{(t+1)}$ for each microservice i at time $t+1$.

for $t = 1, 2, 3, ...$ **do**
  $\widetilde{a_i^{(t+1)}} \leftarrow \psi(\mathbf{a}_i^{(t)})$
  $b_i^{(t+1)} \leftarrow \phi([\widetilde{a_i^{(t-k+2)}}, ..., a_i^{(t)}, \widetilde{a_i^{(t+1)}}])$
  $R_i^{(t+1)} \leftarrow b_i^{(t+1)}$, where $R_i^{lb} \leq R_i^{(t+1)} \leq R_i^{ub}$
  if $R_i^{(t)} < R_i^{(t+1)}$ **then**
    $N_i^{(t+1)} \leftarrow N_i^{(t)} + \left\lceil \frac{R_i^{(t+1)} - R_i^{(t)}}{v_i^p} \right\rceil, \forall i = 1, 2, \ldots, n$
    $N_i^{(t+1)} \leftarrow \min(N_i^{(t+1)}, Q_i)$
    ▷ Adding some pods
  else if $R_i^{(t)} > R_i^{(t+1)}$ **then**
    $N_i^{(t+1)} \leftarrow N_i^{(t)} - \left\lceil \frac{R_i^{(t)} - R_i^{(t+1)}}{v_i^p} \right\rceil, \forall i = 1, 2, \ldots, n$
    $N_i^{(t+1)} \leftarrow \min(N_i^{(t+1)}, Q_i)$
    ▷ Reducing some pods
  end if
end for

III. Experimental setup

A. Microservice Application Deployment

In this section, we first present our dedicated environment for deployment and testing of the microservice application, and then we discuss load generation, baseline algorithm, and evaluation methods. We evaluate our algorithm on an interactive and practical real-world microservice application, namely Bookinfo [13]. Bookinfo is a representative microservice application in the research community. The structure of the application consists of four microservices implemented using different technological stacks. The details of the microservices are outlined as follows:

- *productpage* microservice transfers requests to the *details* and *reviews*.
- *details* microservice stores the book information.
- *reviews* has three versions which contain reviews of the book. Both version v2 and version v3 invoke the *rating* and return black and red stars, respectively. Version v1, however, doesn't call *ratings*.
- *ratings* has the ranking info of books.

Bookinfo is a polyglot application, e.g., $reviews$ is written in Java, while $details$ is written in Ruby. Our experimental environment consists of a test server and the system under test, i.e., the Bookinfo application. The experimental application is deployed on Amazon Web Services (AWS) [10]. We used c5.xlarge EC2 instances for the test server with numerical resource limitation of 4 cores vCPU, 3.4 GHz, 8G memory, 100G storage. We used Kubernetes [18] as the container orchestration engine for four microservices subjected to various workload scenarios generated by the open-source load testing tool named Locust [19]. The Kubernetes manager can host 79 pods at maximum, where each node equally hosts the pods in the cluster thanks to the load balancer component.

Common Python libraries, including Pytorch [20], Tensorflow [21], and Keras [22] were used to build our prediction models. The Graph-PHPA loads the trained models and processes performance metrics to make a prediction on the desired number of pods for running in the next step, and this information is sent to the Kubernetes scheduler for scheduling. Fig. 1 presents a schematic diagram for our experimental environment. The test server was deployed on AWS EC2 and the system under test (SUT) was deployed on AWS EKS. The Locust server was hosted by Docker and all the microservices in Bookinfo were injected by Envoy Sidecar service controlled by Istio for monitoring and load balancing. We then used Prometheus [11] and Grafana [12] to monitor and collect the testing data (resource usages and workload data) for each microservice. The monitoring data was collected every minute consisting of vCPU, memory, and workload for each microservice at one minute interval.

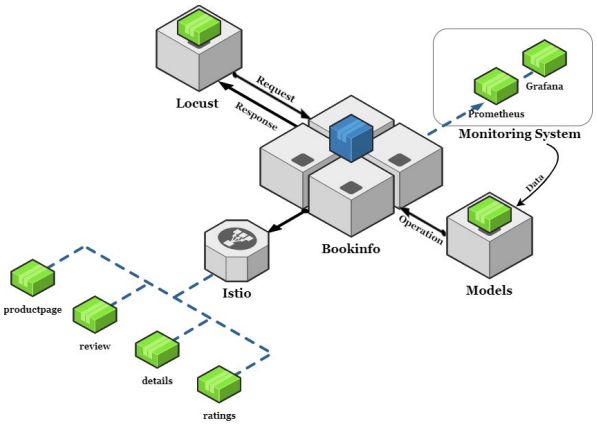

Fig. 1: Experimental environment.

B. Load generation

To evaluate the Graph-PHPA, we used a practical workload trace in our experimental environment, which is a trace of function invocations in Microsoft's Azure Functions [14] from 31/01/2021 to 13/02/2021 with time interval of 5 minutes. We extracted and scaled the invocation calls in the dataset to mock the requests to cloud cluster. The workload trace is available publicly and details can be found in [6]. Locust consequently spawned the number of requests every minute to stress microservices in Bookinfo application.

C. Evaluation of the prediction model

We evaluate Graph-PHPA in two steps. First, we train two separate models using LSTM and GCN based on the experiment data collected from Granfana. We assess the workload prediction model and the resource prediction model using MSE/MAE to identify the best hyperparameters for the models. The next phase is to predict the incoming workload and consequently the required vCPU of the next step using the above trained models. The Graph-PHPA is evaluated on real-time workload [14]. The results are then compared with the baseline algorithm using Kubernetes HPA. The evaluation results of each step are discussed in the following sections.

a) Training predictive models: The training data is collected from our testbed to train the predictive models. The models are trained with various settings and the model with the minimum MSE is used for Graph-PHPA. The whole dataset is divided into a training set (60%), validation set (20%), and test set (20%), i.e., the last 800 data points to be used in our result analysis. We used the workload prediction model to predict the number of requests for the microservices. At t, the LSTM model predicted the workload for the next time step $t+1$ based on the last k observations. We investigated various window sizes k, including 4, 6, 8, and 10 to identify the best input window size for the LSTM model. We chose the input window size of 10 for the best performance. The prediction algorithm was trained with the following hyperparameters, including optimizers (Adam at learning rate (lr) = 0.01), epochs (50), batches (64), number of units (10, 50, 75, 100, 125, 150, 175, 200), and number of hidden layers (1, 2, 3, 4, 5). The model performance was assessed and compared using MSE and MAE on the test dataset.

The settings of the resource prediction model include the input features, neural network structure, and loss function. We carefully chose these parameters to obtain an accurate prediction model. The node feature was the workload of each microservice. The workload data over a fixed-length window k was then fed into the resource prediction model as input for predicting the maximum resource consumed across the window. The adjacent matrix is computed with the assumption that the graph structure of the application is undirected with self-loops. The setup which yields the minimum MSE on the test dataset was chosen as our setup for the Graph-PHPA.

b) Graph-PHPA: We evaluate the Graph-PHPA using the real-time workload generated based on the realistic dataset on our testbed. Each pod has a limit of 1 vCPU (v_i^p = 1 vCPU) and 2GB of memory. Once the workload changes, the Graph-PHPA proactively recalculates the number of pods of each microservice using the trained predictive models, and redeploys these pods to the cluster accordingly.

IV. Experimental Results

A. Workload Forecasting Evaluation Results

Fig. 2 shows the ground truth (requests per second) and the predicted workload traces leveraging the best model we found in the work. The best model adopted an Adam optimizer (learning rate equals 0.01) with one LSTM layer of 50 hidden units and a batch size of 64. We can observe that the predict workload value can easily match the ground truth.

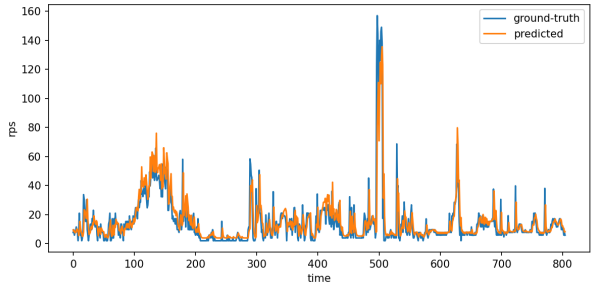

Fig. 2: Workload prediction model.

B. Resource prediction model

Fig. 3 illustrates the best resource prediction results for the "product page" microservice only as it consumes the highest amount of vCPU resource in the application. Our results have shown that the vCPU usage has been captured quite well for the microservice after fine-tuning the resource prediction model through its hyperparameters: optimizers (Adam at lr = 0.001), epochs (100), batches (256), and two GCN layers.

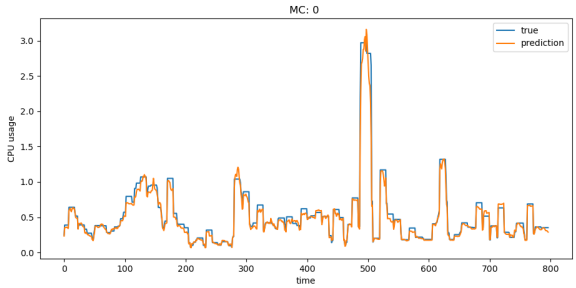

Fig. 3: vCPU prediction for $productpage$ microservice.

C. Overall System Evaluation Results

We demonstrate the comparison of our proposed Graph-PHPA algorithm with the baseline algorithm in Kubernetes for the Bookinfo application in Fig. 4, where only vCPU allocation of product-page microservice is shown. Once the vCPU utilization of any pods is greater than the user-defined value (vCPU threshold), Kubernetes dynamically adds pods to the microservice. On the other hand, Kubernetes removes a pod as the CPU utilization of all pods is less than the scale-in threshold. Our proposed algorithm effectively allocates the required pods to the microservice using the predicted workload and the trained resource allocation model. It is clearly seen in Fig. 4 that the number of pods of $productpage$ microservice is adjusted appropriately using the forecasted workload, which shows promising performance of our method compared to the baseline algorithm in Kubernetes under the same real-time test workload profile. More specifically, the lower subplot

compares our Graph-PHPA result with that of the Kubernetes autoscaler with 70% threshold. Compared to the upper subplot using 90% threshold, the lower subplot shows significant resource saving using the proposed Graph-PHPA method. Although a larger threshold can indeed reduce the number of pods, it also poses the higher risk of overloading especially in a reactively controlled manner, which may be less desirable in any practical operation.

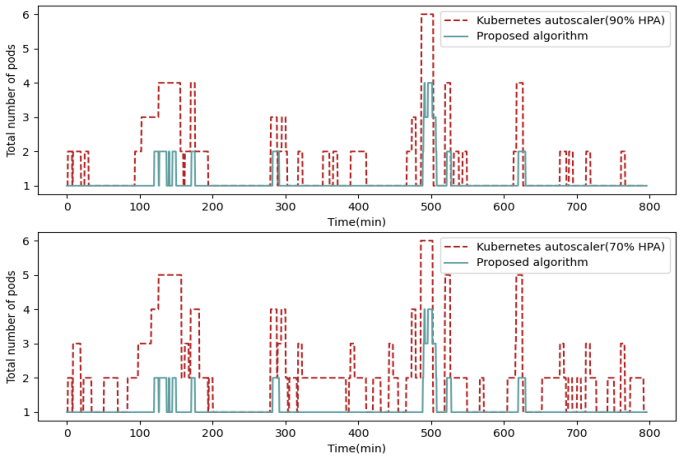

Fig. 4: Total number of pods allocated to *productpage* microservice with respect to vCPU threshold 90% (upper subplot) and 70% (lower subplot).

V. Conclusion

In this paper, we have presented a novel graph-based proactive horizontal pod autoscaling strategy for microservices using LSTM-GNN, namely Graph-PHPA. This approach is two-stage as it first predicts the upcoming workload by using LSTM and takes this output to further infer the desired amount of resources, i.e., pods. This two-stage architecture allows each individual model to be replaced in a plug-and-play manner for further development. The superiority of our proposed approach has been validated through extensive experiments implemented using a dedicated testing environment with real-time workload. Significant resource savings have been shown compared to the baseline algorithm implemented using the Kubernetes HPA. Finally, we note that one limitation of the current work is that the proposed model only focuses on the prediction of vCPU usage whilst taking account of the workload as the model attribute, which will be addressed as part of our future work. In addition to this, we will also further investigate the efficacy of our proposed algorithm by evaluating it against a group of related algorithms in our future work.

Acknowledgement

This work is supported by the Huawei Ireland Research Centre for the scalability and provisioning surveillance project and Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289_P2 (Insight SFI Research Centre for Data Analytics), co-funded by the European Regional Development Fund in collaboration with the SFI Insight Centre for Data Analytics at Dublin City University.

References

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[2] Q. Li, B. Li, P. Mercati, R. Illikkal, C. Tai, M. Kishinevsky, and C. Kozyrakis, "RAMBO: resource allocation for microservices using Bayesian optimization," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 46–49, 2021.

[3] V. Sachidananda and A. Sivaraman, "Learned autoscaling for cloud microservices with multi-armed bandits," *arXiv preprint arXiv:2112.14845*, 2021.

[4] A. A. Khaleq and I. Ra, "Intelligent autoscaling of microservices in the cloud for real-time applications," *IEEE Access*, vol. 9, pp. 35 464–35 476, 2021.

[5] A. U. Gias, G. Casale, and M. Woodside, "Atom: Model-driven autoscaling for microservices," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1994–2004.

[6] J. Park, B. Choi, C. Lee, and D. Han, "GRAF: a graph neural network based proactive resource allocation framework for SLO-oriented microservices," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 154–167.

[7] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 805–825.

[8] Z. Chen, H. Wu, N. E. O'Connor, and M. Liu, "A comparative study of using spatial-temporal graph convolutional networks for predicting availability in bike sharing schemes," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 1299–1305.

[9] W. Liao, B. Bak-Jensen, J. R. Pillai, Y. Wang, and Y. Wang, "A review of graph neural networks and their applications in power systems," *Journal of Modern Power Systems and Clean Energy*, 2021.

[10] Amazon. Amazon ec2 on-demand pricing. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-deman

[11] Prometheus. Prometheus - monitoring system time series database. [Online]. Available: https://prometheus.io

[12] Grafana. Grafana: The open observability platform. [Online]. Available: https://grafana.co

[13] Bookinfo. Bookinfo application from istio. [Online]. Available: https://istio.io/latest/docs/examples/bookinfo/

[14] Y. Zhang, Í. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 724–739.

[15] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying LSTM to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[17] S. Bloemheuvel, J. v. d. Hoogen, D. Jozinović, A. Michelini, and M. Atzmueller, "Multivariate time series regression with graph neural networks," *arXiv preprint arXiv:2201.00818*, 2022.

[18] Kubernetes. Kubernetes documentation. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[19] Locust. Locust: An open source load testing tool. [Online]. Available: https://locust.io

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[22] F. Chollet. Keras. [Online]. Available: https://keras.io