OptScaler: A Collaborative Framework for Robust Autoscaling in the Cloud

Ding Zou*
Zhejiang University
Ant Group
zoud@zju.edu.cn

Wei Lu*
Ant Group
xiaobo.lw
@antgroup.com

Zhibo Zhu
Ant Group
gavin.zzb
@antgroup.com

Xingyu Lu†
Ant Group
sing.lxy
@antgroup.com

Jun Zhou†
Ant Group
jun.zhoujun
@antgroup.com

Xiaojin Wang
Ant Group
yueying.wxj
@antgroup.com

Kangyu Liu
Ant Group
liukangyu.lky
@antgroup.com

Kefan Wang
Ant Group
kefan.wkf
@antgroup.com

Renen Sun
Ant Group
renen.sun
@antgroup.com

Haiqing Wang
Ant Group
wanghaiqing.whq
@antgroup.com

ABSTRACT

Autoscaling is a critical mechanism in cloud computing, enabling the autonomous adjustment of computing resources in response to dynamic workloads. This is particularly valuable for co-located, long-running applications with diverse workload patterns. The primary objective of autoscaling is to regulate resource utilization at a desired level, effectively balancing the need for resource optimization with the fulfillment of Service Level Objectives (SLOs). Many existing proactive autoscaling frameworks may encounter prediction deviations arising from the frequent fluctuations of cloud workloads. Reactive frameworks, on the other hand, rely on real-time system feedback, but their hysteretic nature could lead to violations of stringent SLOs. Hybrid frameworks, while prevalent, often feature independently functioning proactive and reactive modules, potentially leading to incompatibility and undermining the overall decision-making efficacy. In addressing these challenges, we propose OptScaler, a collaborative autoscaling framework that integrates proactive and reactive modules through an optimization module. The proactive module delivers reliable future workload predictions to the optimization module, while the reactive module offers a self-tuning estimator for real-time updates. By embedding a Model Predictive Control (MPC) mechanism and chance constraints into the optimization module, we further enhance its robustness. Numerical results have demonstrated the superiority of our workload prediction model and the collaborative framework, leading to over a 36% reduction in SLO violations compared to prevalent reactive, proactive, or hybrid autoscalers. Notably, OptScaler has been successfully deployed at Alipay, providing autoscaling support for the world-leading payment platform.

PVLDB Reference Format:
Ding Zou, Wei Lu, Zhibo Zhu, Xingyu Lu, Jun Zhou, Xiaojin Wang, Kangyu Liu, Kefan Wang, Renen Sun, and Haiqing Wang. OptScaler: A Collaborative Framework for Robust Autoscaling in the Cloud. PVLDB, 17(12): 4090 - 4103, 2024.
doi:10.14778/3685800.3685829

*Equal contribution
†Corresponding authors

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685829

1 INTRODUCTION

The rapid growth of cloud computing has generated a significant demand for computing resources such as CPU cores and memory [47]. As a result, efficient resource management and optimization techniques have received increasing attention to strike a balance between the cost of resources and the strict Service Level Objectives (SLOs) in cloud services. Traditionally, to address drastic workload changes and ensure SLO adherence, resources are often provisioned based on peak demand, leading to substantial resource waste [12]. In response to this challenge, autoscaling has emerged as a fundamental capability of cloud infrastructure, allowing for the automatic and dynamic scaling of resources in reaction to workload fluctuations [7]. With autoscaling, cloud providers can swiftly adapt to varying workloads without manual intervention, thereby achieving optimal performance. This capability not only provides substantial cost savings but also enhances user experience. There are two types of scaling based on how resources are adjusted: vertical scaling, which modifies the resource capacity within existing cluster nodes [14], and horizontal scaling, which involves adding or removing deployed nodes [59]. Major cloud vendors widely favor horizontal scaling due to its ease of implementation and its ability to enhance the availability and fault tolerance of applications [7, 9, 10].

In this paper, we focus on horizontal scaling in the context of co-located long-running applications (LRAs) with diverse workload patterns. LRAs, also known as online services, are integral in many cloud computing scenarios such as online marketing and content recommendation. These scenarios frequently exhibit time-varying workloads due to periodic user arrivals. LRAs differ from batch jobs, where each job is assigned to an exclusive node that is terminated after processing the job [38]. Existing autoscaling methods (also referred to as autoscalers) for LRAs can be classified as either reactive or proactive, based on the timing of scaling, and both have been widely researched [1]. Reactive autoscalers, driven by real-time system feedback [39, 43], adaptively adjust resources at run-time after unexpected outcomes occur and are favored for their simplicity of implementation. However, real-world workloads often exhibit

periodic fluctuations [16], and the reactive autoscaling's hysteretic nature could lead to violations of SLOs.

In this regard, [36] experimented with the autoscaling methods from Amazon, Microsoft, and Google, leading to the conclusion that their reactive methods had performance pitfalls. In contrast, proactive autoscalers, which anticipate future workload and scale resources beforehand, might avoid the pitfalls of reactive scaling [11, 57]. However, proactive autoscalers with sophisticated prediction techniques may struggle to capture the full picture of uncertain LRAs' workloads, resulting in prediction errors and resource wastage. To enhance the performance of proactive autoscaling under uncertainty (e.g., unpredictable bursts or anomalies), integrating proactive and reactive methods to build a hybrid autoscaler is essential, as highlighted in [47]. An ideal paradigm involves the proactive method handling foreseeable workload patterns while the reactive method corrects unexpected deviations.

In practical applications, hybrid autoscalers face their own set of challenges. Firstly, the use of proactive and reactive modules in existing hybrid autoscalers often leads to independent operation, potentially resulting in conflicting scaling decisions when the outputs of these modules differ. While an enhanced prediction model in the proactive module could mitigate these conflicts, [47] stresses that reconciling conflicting outputs to reach a final implementation decision remains a significant challenge in production. Secondly, the intricate nature of the cloud environment poses challenges in developing a robust hybrid autoscaler. For example: 1) Hardware speed limitations can constrain scaling capacity, possibly causing disparities between intended and actual scaling decisions, ultimately undermining autoscaling performance; 2) Systematic noise in scaling metrics, such as CPU utilization, has the potential to mislead both proactive and reactive scaling decisions; 3) The co-location of LRAs frequently leads to resource contention, further exacerbating the uncertainty in scaling metrics.

The primary goal of this study is to develop an autoscaling framework for LRAs that effectively tackles the challenges previously mentioned. To confront the first challenge, we introduce a novel **collaborative** autoscaling framework, an advancement of existing hybrid frameworks. This includes an innovative proactive module with enhanced prediction capabilities, a real-time reactive module to address scaling errors stemming from prediction inaccuracies, and an optimization module to manage the trade-off between resource costs and SLO satisfaction. The term **collaborative** indicates that our framework orchestrates proactive and reactive modules to make holistic final decisions, thus avoiding potential conflicts between the two modules. To achieve this, our collaborative framework constructs the optimization module with dynamic inputs from both proactive and reactive modules. The optimization objective is to assist the cloud system in attaining the desired scaling metrics under dynamic workloads. The interpretable optimization objective and constraints (i.e., practical restrictions in the cloud) also improve the interpretability of our framework, an aspect often lacking in many black-box or machine-learning-based autoscalers. Enhanced transparency within the framework can further establish user trust and facilitate potential upgrades of autoscalers [47].

To tackle the second challenge, we explicitly model the essential components of the complex cloud system to enable robust scaling decisions. Specifically: 1) We integrate Model Predictive Control (MPC) into our optimization module to address the speed limitation of scaling. In contrast to traditional control methods such as Proportional-Integral-Derivative (PID) [5], MPC is recognized for its superior potency. In our context, MPC enables optimization of the scaling decision at the current time while accounting for potential workload bursts in future time intervals; 2) We mitigate the impact of noises in scaling metrics by employing the chance-constraint method [45], further enhancing the robustness of MPC; 3) We develop an adaptive estimator within the reactive module, continually adjusted based on real-time system feedback, to monitor the collaborative impact of co-located LRAs on scaling metrics.

The enhancement and collaboration of each module in OptScaler has proven highly effective for managing co-located LRAs with diverse workload patterns. To support this, the following questions are thoroughly investigated:

(1) **Question 1**: Can OptScaler offer superior workload prediction ability compared to prevalent methods?
(2) **Question 2**: Can OptScaler make more robust scaling decisions than mainstream autoscaling frameworks (e.g., reactive, proactive, and hybrid frameworks) across various real-world workload patterns?
(3) **Question 3**: Taking the autoscaling framework as a whole, what is the overall advantage of OptScaler compared to state-of-the-art autoscalers?

Contributions. Addressing the above questions, OptScaler contributes to cloud resource management in three key aspects:

(1) OptScaler develops an innovative proactive module that surpasses state-of-the-art prediction methods when tested on challenging public and internal workload traces;
(2) OptScaler creatively leverages an optimization module to enable the proactive module to collaborate with the reactive module in making final scaling decisions. The incorporation of MPC mechanism and chance constraints into the optimization module enhances its robustness;
(3) OptScaler demonstrates its superiority as a comprehensive autoscaling framework, reducing over 36% more SLO violations compared to a state-of-the-art hybrid framework. OptScaler has successfully supported the online autoscaling of LRAs at Alipay, the world-leading payment platform.

The remainder of the paper is organized as follows. Section 2 reviews related works. Section 3 provides background information about our proposed autoscaling framework. Section 4 elaborates on the proposed framework, encompassing the proactive, reactive, and optimization modules. Section 5 compares experimental results from the proposed framework and prevalent frameworks. Section 6 describes the deployment of OptScaler along with the online results at Alipay. Section 7 concludes this paper.

2 RELATED WORKS

Autoscaling methods play a critical role in managing cloud resources, with active research in both theoretical and practical implementation across various cloud systems. Table 1 compares our work with representative literature. It is observed that the majority of autoscalers rely on standalone proactive or reactive modules.

In the context of workload prediction, statistical models such as Auto regression or Exponential smoothing are commonly applied

Table 1: Comparison of the proposed OptScaler with the existing representative works on autoscaling

Type	Literature	Optimization[c]	Uncertainty[d]
Proactive	Zhou2023AHPA[59], Wang2023[51], Das2016[8]	✗	✗
	Dezhabad2018[11], Zhang2013[58], Roy2011[42]	✓	✗
	Poppe2023[37], Qiu2023AWARE[40], Xue2022[57], Jamshidi2016[19]	✗	✓
	Pan2023MagicScaler[33], Qian2022RobustScaler[38], Luo2022[26]	✓	✓
Reactive	Cahoon2022Doppler[6], Rzadca2020Autopilot[43], Farokhi2016[14]	✗	✗
	Qiu2020FIRM[39]	✗	✓
	Gaggero2018[15]	✓	✗
	Persico2017[35]	✓	✓
Hybrid[a]	Singh2021RHAS[46], Jv2018HAS[22], Ali2012[2], Urgaonkar2008[48]	✗	✗
Collaborative[b]	**OptScaler**	✓	✓

[a] The work employs both proactive and reactive modules but they operate independently and may produce conflicting scaling decisions.
[b] The work orchestrates proactive and reactive modules to produce a holistic scaling decision.
[c] The work adopts any optimization technique that can handle practical scaling restrictions and increase interpretability of scaling decisions.
[d] The work considers uncertainty in the cloud system.

[19, 21, 22, 40, 58], with recent trends also embracing deep learning models like RNN and Transformer [25, 26, 33, 53]. When it comes to scaling decisions, existing works predominantly use queuing models to make stable decisions. Some also employ analytical methods like PID [3, 5] or pretrained estimators [59], while others rely on customized rules based on prior knowledge [22, 28, 43, 52, 54]. Additionally, optimization techniques and Reinforcement Learning (RL) have also been utilized, and a minority of literature addresses noise through fuzzy decisions [19, 35, 50].

However, as pointed out by [47], a standalone proactive or reactive autoscaler may fall short of production-ready standards. Only a small number of studies (15 out of 104) have attempted to harness the power of hybrid autoscaling, with the dominant approach being to **choose between** scaling decisions made by proactive and reactive modules. For instance, [1, 2, 41] adopted the reactive decision when the two conflicted, and [22, 46, 48] switched between proactive and reactive decisions based on predefined rules. Notably, our proposed framework revolutionizes this approach by fostering full collaboration between the proactive and reactive modules, with both contributing to the final scaling decision. The former provides future workload inputs, while the latter traces system dynamics between workloads and scaling metrics through real-time feedback.

Previous researchers have also explored leveraging optimization techniques such as Model Predictive Control (MPC) in autoscaling. For example, MPC has been used to adjust the number of virtual machines to meet response time requirements [42], study the impact of control frequency on resource efficiency and proactive scaling overhead [31], solve server placement problems [15, 58], and facilitate combined horizontal and vertical scaling with load distribution among machines [18]. However, to our knowledge, MPC (or any other optimization technique) has never been applied in a hybrid/collaborative autoscaler. Our paper advances this area of research by presenting a new approach to exploring the potential of optimization in facilitating collaborative autoscaling.

3 PRELIMINARIES

In this section, we first present background about cloud resource deployment. Then, we briefly introduce workload patterns and scaling metrics, both are exploited by OptScaler for robust scaling.

3.1 Basics of Resource Deployment in the Cloud

In cloud computing, computing resources are organized within a multilevel hierarchical structure. For instance, within Alipay's Recommendation Platform, the cloud computing resource is logically divided into several **clusters** [24]. Each cluster encompasses a variable number of **nodes** (or machines) and supports a group of LRAs. Each node possesses its own dedicated (and usually equalized) computing resources and can be virtually segmented into multiple **containers** [23]. Each container has the capacity to accommodate a single LRA, and for fault tolerance purposes, an LRA's workload is evenly spread across all host containers. To achieve a well-balanced loading scheme, each node in the cluster deploys one container for each LRA, resulting in all nodes sharing an identical container configuration.

It is important to note that two practical constraints impact the scaling within the LRA scenario. In order to ensure safety, upper and lower bounds for the number of nodes in each cluster must be maintained. Furthermore, the addition or removal of multiple nodes can be conducted in parallel, but there are limitations on the speed and concurrency of node addition/deletion.

3.2 Patterns of Workload and CPU Utilization

Variations in workload patterns impact the predictive complexity, significantly affecting the performance of the proactive module and consequently, the entire autoscaling framework. For accurate workload prediction, it is essential that workloads exhibit periodic behaviors. Using randomly sampled workloads of two LRAs from Alipay (from Jan/22/2024 to Feb/04/2024), we have depicted the normalized workload patterns in the left subplot of Figure 1. These patterns reveal periodic behaviors in both LRAs, indicating the potential benefits of autoscaling. Specifically, LRA 1 demonstrates daily fluctuations, while LRA 2 exhibits a weekly pattern with lower workloads during weekends.

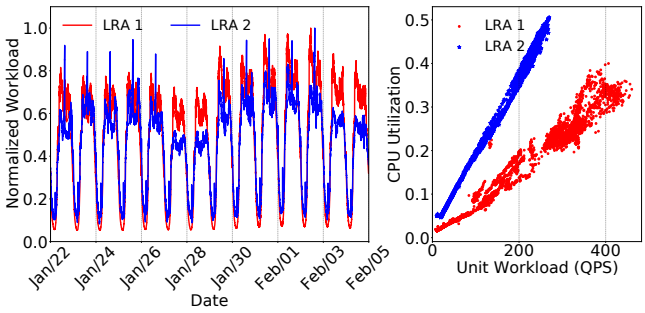

Figure 1: Periodical workloads (left) and linear correlation between unit workload and CPU utilization (right) for two randomly chosen LRAs from Alipay.

Moreover, as highlighted in [4], there exists a close relationship between SLOs and system metrics such as CPU and memory utilization within a cloud environment, directly influencing the efficacy of cloud resource management. With the primary resource stress stemming from the limited CPU capacity, CPU utilization has been selected as the primary scaling metric. To strike a balance between SLOs and resource conservation, CPU utilization should closely approach (but not exceed) a designated threshold. Consequently, understanding the dynamics between the unit workload (i.e., workload of LRAs in each node) and the resultant CPU utilization is crucial for our framework design.

Initially, we focus on a single node hosting a solitary LRA, assuming that this rule applies similarly when hosting multiple LRAs (i.e., co-location). As shown in the right subplot of Figure 1, we observe an explicit linear correlation between the unit workload and the mean CPU utilization. This correlation underscores two key insights: 1) LRAs exhibit varying impacts on CPU utilization. LRA 2 proves to be more resource-intensive than LRA 1, evident from the significantly higher CPU utilization for LRA 2 compared to LRA 1 under the same unit workload (e.g., 200 QPS or queries per second); 2) Under a linear estimator, the variance of residuals sharply increases as the unit workload rises, exacerbating estimator uncertainty. To address these insights, we propose two enhancements to the standard linear estimator. Firstly, we incorporate an uncertainty term, with the variance being a function of the unit workload. Secondly, we implement an online linear regression (OLR) [29] as a feedback mechanism to dynamically adapt the linear estimator at runtime with the latest feedback, without requiring complete retraining of the estimator with the entire dataset. More details on these enhancements will be provided in Section 4.2.

4 PROPOSED FRAMEWORK

When implementing horizontal scaling, OptScaler could focus on the scaling of the number of nodes in each cluster for making holistic scaling decisions. The OptScaler framework, illustrated in Figure 2, comprises the following components:

- **Proactive module** consists of a workload prediction model trained on historical workloads, providing multi-timestep predictions of future workloads for all LRAs of interest;

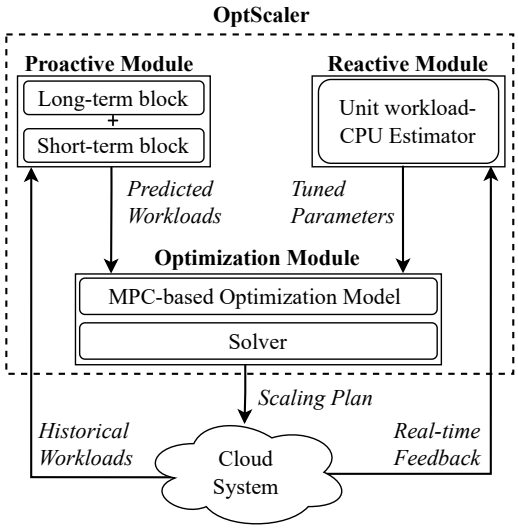

Figure 2: Flowchart of the proposed OptScaler framework.

- **Reactive module** offers a self-tuning estimator of CPU utilization. It is initialized using historical data of the unit workloads and the corresponding CPU utilization, and continuously fine-tuned by real-time system feedback;
- **Optimization module** takes input from both proactive and reactive modules, producing an optimal scaling plan under practical restrictions. The plan is then deployed to the cloud system.

4.1 Proactive Module

In Proactive Module, we employ a workload prediction model to anticipate the future workloads of different LRAs. The casting results are further forwarded to the optimization module for CPU utilization management. The following challenges must be addressed for effective autoscaling: 1) Long-term forecasting: In order to complete node addition/removal within each time interval amidst rapid workload fluctuations, the optimization module must plan several steps in advance. This necessitates long-term forecasting, such as predicting workload a day in advance. 2) Model efficiency: Due to the presence of multiple LRAs with diverse weekly and daily workload patterns, constructing a separate prediction model for each LRA would be inefficient. It is advantageous to have a single model that can accommodate the various workload patterns, providing convenience for the online service management.

To address the above challenges, a workload prediction model is introduced in Figure 3. Formally, let $y_n^t \in \mathbb{R}$ denotes the workload at time step t of the n-th LRA, the task is to predict the future values $\boldsymbol{y}_n^{t+1:t+H} = [y_n^{t+1}, \ldots, y_n^{t+H}]$ based on the historical values $\boldsymbol{y}_n^{t-C:t} = [y_n^{t-C}, \ldots, y_n^t]$ and other covariates. We herein use bold symbols to denote **vectors**. Typically, it can be formalized as:

$$\boldsymbol{y}_n^{t+1:t+H} = \mathcal{F}_\Theta(\boldsymbol{y}_n^{t-C:t}, \boldsymbol{z}_n^{t-C:t+H}, n), \tag{1}$$

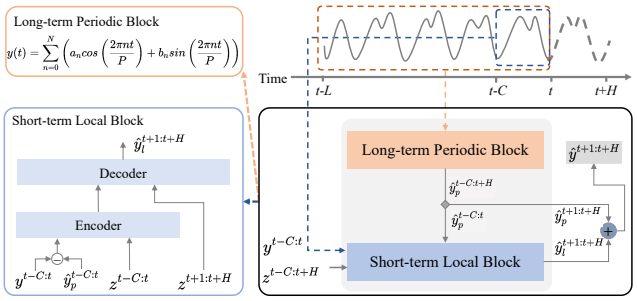

Figure 3: Framework of the workload prediction model in the Proactive Module.

where $\mathcal{F}_\Theta(\cdot)$ denotes the workload prediction model, Θ is the learning parameters, $\boldsymbol{z}_n^{t-C:t+H}$ is the known covariates (e.g., date) of the n-th LRA, and $n \in \{1, 2, \ldots, N\}$ is the index of LRA[1].

It is evident that $\mathcal{F}_\Theta(\cdot)$ handles the workload prediction task for various LRAs with the parameter Θ and distinguishes LRAs with index n. Specifically, $\mathcal{F}_\Theta(\cdot)$ comprises a Long-term Periodic Block and a Short-term Local Block to accomplish this task effectively.

- **Long-term Periodic Block**. Capture too long series input for forecasting workloads of LRAs, each with varying weekly or daily patterns, can be challenging. To address this, we employ the Fourier series to represent the diverse periodicities of different LRAs. With a periodicity P and truncation order $\hat{N}$ of the Fourier series expansion, the long-term periodicity can be formalized as $y^t = \sum_{n=0}^{\hat{N}}(a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P}))$. The Fourier coefficients $[a_0, b_0, \ldots, a_{\hat{N}}, b_{\hat{N}}]$, learned from historical data, facilitate the inference of future periodicity without the need for extensive historical inputs.
- **Short-term Local Block**. This block learns the local pattern of workloads, such as local trend and influence. For this purpose, we apply a Seq2seq structure consisting of three steps: 1) According to historical values $\boldsymbol{y}^{t-C:t}$ and their periodic estimation $\hat{\boldsymbol{y}}_p^{t-C:t}$, we calculate the residual of historical series $\boldsymbol{y}_l^{t-C:t} = \boldsymbol{y}^{t-C:t} - \hat{\boldsymbol{y}}_p^{t-C:t}$; 2) We extract local pattern of historical residual with series $\boldsymbol{y}_l^{t-C:t}$ and corresponding covariates $\boldsymbol{z}^{t-C:t}$. A linear-complexity Flow-Attention [55] $f(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}; \boldsymbol{\theta}_{attn})$ is applied for long sequences with high efficiency. Specifically, let $\boldsymbol{h}_{in} = [\boldsymbol{y}_l^{t-C:t}, \boldsymbol{z}^{t-C:t}]$ denote the whole historical representation, the encoder is $\boldsymbol{h}_{enc} = f(\boldsymbol{h}_{in}, \boldsymbol{h}_{in}, \boldsymbol{h}_{in}; \boldsymbol{\theta}_{enc})$, where $\boldsymbol{\theta}_{enc}$ is the parameters of encoder; 3) Based on the encoder output $\boldsymbol{h}_{enc}$ and covariates $\boldsymbol{z}^{t-C:t+H}$, the decoder estimates the local residual in future. With the same Flow-Attention structure, the decoder is $\hat{\boldsymbol{y}}_l^{t+1:t+H} = f(\boldsymbol{z}^{t-C:t}, \boldsymbol{z}^{t+1:t+H}, \boldsymbol{h}_{enc}; \boldsymbol{\theta}_{dec})$, where $\boldsymbol{\theta}_{dec}$ is the learning parameter.

In the Short-term Local Block, two practical techniques are employed to enhance long-term forecasting. The first technique, Flow-Attention, demonstrates computational efficiency with linear complexity relative to the length of the series. The second technique involves the use of a non-autoregressive decoder, where the forecasting of time step t is independent of the results from the preceding time step. This non-autoregressive decoder helps avoid accumulative errors and offers swift inference speed.

[1]For simplicity, we ignore the index n of LRA in the following.

Algorithm 1 Widrow-Hoff algorithm in the Reactive Module

Input: Choose parameter $\eta > 0$
Initialize: ${\boldsymbol{w}_{\boldsymbol{k}}}^0$ according to historical data training
for $t = 1$ to T **do**
  Get unit workload $\frac{\boldsymbol{y}^t}{x^t} \in \mathbb{R}^N$
  Estimate $\hat{c}^t = f(\frac{\boldsymbol{y}^t}{x^t}) \in \mathbb{R}$
  Observe $c^t \in \mathbb{R}$
  Get feedback error $e = \hat{c}^t - c^t$
  Update ${\boldsymbol{w}_{\boldsymbol{k}}}^t = {\boldsymbol{w}_{\boldsymbol{k}}}^{t-1} - \eta e \frac{\boldsymbol{y}^t}{x^t}$
end for
return ${\boldsymbol{w}_{\boldsymbol{k}}}^T$

As illustrated in the flowchart Figure 2, the predicted workload of each LRA is the sum of the outputs from Long-term Periodic Block and Short-term Local Block:

$$\hat{\boldsymbol{y}}^{t+1:t+H} = \hat{\boldsymbol{y}}_p^{t+1:t+H} + \hat{\boldsymbol{y}}_l^{t+1:t+H} \tag{2}$$

During model training, quantile loss [34, 60] helps minimize the ratio of the forecasting results that are lower than the actual values. This approach enhances the reliability of subsequent scaling decisions from the perspective of proactive module.

4.2 Reactive Module

In Reactive Module, we build a linear estimator with an uncertainty term to map the unit workloads of LRAs to the average CPU utilization of all nodes in the cluster, which could be formulated as:

$$c = f\left(\frac{\boldsymbol{y}}{x}\right) = w_b + \frac{\boldsymbol{y}^\top \boldsymbol{w}_{\boldsymbol{k}}}{x} + \epsilon \tag{3}$$

where $f(\cdot)$ denotes the estimator; $\boldsymbol{y} \in \mathbb{R}^N$ is the workload vector for all N LRAs; x is the number of nodes in the cluster to be decided; c denotes the estimated CPU utilization, which is a function of unit workload $\frac{\boldsymbol{y}}{x}$; $\boldsymbol{w}_{\boldsymbol{k}}$ and w_b are the slope and intercept for the linear model, respectively. $\boldsymbol{w}_{\boldsymbol{k}}$ can be seen as the weights of the LRAs, and w_b is naturally the overhead load of CPU; ϵ is the uncertainty term compensating for the residual of a linear model. ϵ obeys the following normal distribution:

$$\epsilon \sim N(0, \sigma^2(\frac{\boldsymbol{y}}{x})) \sim N(0, (\sigma_b + \frac{\boldsymbol{y}^\top \boldsymbol{\sigma}_k}{x})^2) \tag{4}$$

where the standard deviation of ϵ is modelled as a linear function (with coefficient $\boldsymbol{\sigma}_k$ and σ_b) of unit workload. In this way, we ensure that the uncertainty term obeys the rules shown in Figure 1 that the variance of CPU utilization increases with larger unit workloads.

Based on the above formulations, we now introduce the reactive mechanism. Initially, all the parameters (including $\boldsymbol{w}_{\boldsymbol{k}}$, w_b, $\boldsymbol{\sigma}_k$ and σ_b) are initialized using maximum likelihood estimation [30] on the historical data with non-negative constraints. Subsequently, we employ OLR [29] as a means to adjust $\boldsymbol{w}_{\boldsymbol{k}}$ using real-time feedback. The feedback mechanism of OLR resembles that of a supervised learning algorithm, aiming to minimize the cumulative square loss of a linear function in an online setting. The pseudocode is shown

in Algorithm 1, where η is a parameter to control the strength of feedback tuning, in analogy with the learning rate in a machine learning context, and the real-time feedback error $e = \hat{c}^t - c^t$ is derived from the cloud. In practice, each time when new feedback is available, we could choose the most recent (i.e., $T = 1$) or a series of (i.e., $T > 1$) feedback, and apply OLR to update $\boldsymbol{w_k}$ quickly. Instead of retraining the linear model using the whole dataset each time, OLR focuses on the newest T feedback, and its simplicity and efficiency become key advantages in an online system.

4.3 Optimization Module

Our optimization module is designed to enable collaboration between the proactive module and the reactive module, fostering holistic decision-making. It leverages MPC to dynamically take inputs from both modules and consider all the practical constraints in the cloud. As stated in Section 1, MPC shows promise for addressing future workload bursts when the scaling speed is restricted.

In general, for each control action, MPC takes the latest system state and solves a constrained optimization model over a sliding window of future time intervals. It only applies the first solution over the horizon and repeats the procedure the next time [17]. In our context, we repeatedly optimize scaling decisions of future D time intervals at the beginning of every time interval. Each time interval $d \in \{1, 2, \ldots, D\}$ spans h minutes (e.g., 30 minutes), which can be much longer than the resolution of workload predictions (e.g., 1 minute). Given that each cluster is highly autonomous with its exclusive LRAs and resources, we will focus on a single cluster to build the optimization model. At time step t (i.e., the beginning of a time interval), the following model of D time intervals is solved:

$$\max_{\boldsymbol{u}} \quad \sum_{d=1}^{D} c^d \tag{5}$$

$$s.t. \quad |u^d| \le \frac{h}{\tau} \cdot s, \qquad \forall d \tag{6}$$

$$x^d = x^0 + \sum_{j=1}^{d} u^j, \qquad \forall d \tag{7}$$

$$c^d = \max_{j \in \{d, d+1\}} f\left(\frac{\boldsymbol{y}^j}{x^d}\right), \qquad \forall d \tag{8}$$

$$c^d \le c^*, \qquad \forall d \tag{9}$$

$$X^{min} \le x^d \le X^{max}. \qquad \forall d \tag{10}$$

Inputs: x^0 represents the initial number of nodes, capturing the number of nodes deployed at time t. $\boldsymbol{y}^d$ denotes the peak value of the predicted workload in time interval d (i.e., $\boldsymbol{y}^{t+(d-1)h:t+dh}$) taken from the proactive module. $f(\cdot)$ denotes the CPU utilization estimator taken from the reactive module.
Variables: u^d denotes the maximum change of the number of nodes in time interval d, and $\boldsymbol{u} = [u^1, \ldots, u^D] \in \mathbb{R}^D$ is the vector. Only u^1 is returned for deployment. x^d denotes the number of nodes provisioned during time interval d.
Objective: (5) is the objective that maximizes the CPU utilization c^d (without exceeding a desired level c^*) over all D time intervals.
Constraints: (6)-(10) are explained as follows:

- Constraint (6): The speed limitation of scaling, where h is the time interval between two successive scaling actions, while τ and s represent the time cost and the concurrency of single node addition/removal action, respectively;
- Constraint (7): The state transition equation that defines the relationship between x^d and x^0;
- Constraint (8): The calculation of CPU utilization c^d, estimated as the maximum value in two adjacent time intervals $\{d, d+1\}$. This is due to the fact that scaling decisions are made in advance, where the resource for time interval $d+1$ is provisioned ahead during d, leading to $f(\boldsymbol{y}^{d+1}/x^d)$);
- Constraint (9): The upper bound c^* for c^d ;
- Constraint (10): The upper and lower bounds for the number of nodes x^d.

Reformulation: Due to uncertainty term ϵ in $f(\cdot)$ as shown in (3), constraint (9) is easily violated. We reformulate constraints (8) and (9) by chance constraint (11) that guarantees satisfaction of constraint (9) with probability α:

$$\mathbb{P}\{c^* \ge c^d = \max_{j \in \{d, d+1\}} f\left(\frac{\boldsymbol{y}^j}{x^d}\right)\} \ge \alpha, \quad \forall d \tag{11}$$

Equivalent transformation: Consider the formulation of $f(\cdot)$ in (3), constraint (11) can be converted to a deterministic equivalent form under the theory of chance-constraint method [45]:

$$c^* \ge c^d = \frac{1}{x^d} \max_{j \in \{d, d+1\}} (\psi^{-1}(\alpha) \cdot \boldsymbol{\sigma}_k + \boldsymbol{w}_k)^\top \boldsymbol{y}^j + w_b + \psi^{-1}(\alpha) \cdot \sigma_b, \qquad \forall d \tag{12}$$

where $\psi^{-1}(\cdot)$ is the inverse of the cumulative distribution function of a standard normal distribution. For example, $\psi^{-1}(\alpha) \approx 1.28$ when $\alpha = 0.9$. Define $m^d = \max_{j \in \{d, d+1\}} (\psi^{-1}(\alpha) \cdot \boldsymbol{\sigma}_k + \boldsymbol{w}_k)^\top \boldsymbol{y}^j$, which can be calculated before solving the optimization model, (12) can be easily rearranged to an equivalent linear form:

$$x^d \ge \frac{m^d}{c^* - w_b - \psi^{-1}(\alpha) \cdot \sigma_b}. \qquad \forall d \tag{13}$$

Constraint (12) also restricts c^d from exceeding c^*, hence, by omitting constant terms, objective (5) is equivalent to

$$\max_{\boldsymbol{u}} \quad \sum_{d=1}^{D} \frac{m^d}{x^d}. \tag{14}$$

Solver: Now the optimization problem (5)-(10) under uncertainty can be reformulated as a robust counterpart with a non-linear objective (14) and linear constraints (6), (7), (10), (13). This reformulated problem can be solved using off-the-shelf solvers (e.g. IPOPT [49]) and the solution u^1 can be rounded for implementation.
Robustness: In summary, we utilize MPC and chance constraints to ensure the satisfaction of practical constraints and bolster the robustness of the solution.

5 NUMERICAL RESULTS

In this section, we provide a detailed exploration of the three research questions in Section 1. We specifically investigate the performance of the proposed OptScaler in the context of co-located LRAs with diverse workload patterns. Question 1 is addressed in Section 5.3, while Questions 2 and 3 are discussed in Section 5.4.

5.1 Workload Dataset

To facilitate a comparison of different autoscalers, we conduct experiments using datasets from the public Azure Functions Trace [44] and the internal Alipay Applets Trace. We select five workloads from LRAs in the Azure Trace and three from the Alipay Trace. Each dataset spans a duration of two weeks (Jul/15 to Jul/28) and maintains a minute-level resolution. Each LRA is identified by the last four bits of its hash ID. We categorize the workloads into three sets, denoted as S_A, S_B, and S_C, based on the complexity of their workload patterns. Sets S_A and S_C are derived from the Azure Trace, while set S_B is derived from the Alipay Trace. LRAs within each set will be co-located in a cluster, resulting in three clusters of interest. Subsequently, experiments on workload prediction and autoscaling will be carried out for these three clusters.

Figure 4 provides details about workloads in the three sets. The left three subplots show the time series of workloads (measured in QPS). Along with the legend, σ denotes the standard deviation of daily peaks of the normalized series. A higher σ value signifies a more pronounced variation in the daily peaks, posing a greater challenge for prediction models. The right three subplots show the regression statistics: daily coefficient of auto-correlation (i.e., **Daily AR**), weekly coefficient of auto-correlation (i.e., **Weekly AR**), and spectral entropy (i.e., **Entropy**); the first two statistics detect the strength of daily/weekly periodicity, and the spectral entropy normalizes the overall complexity of the workloads [20]. We take the bar value of Entropy as 1.0 minus the original spectral entropy, such that for all the three regression statistics, a lower bar indicates a higher level of prediction difficulty. Clearly, the complexity for S_A is much lower than that of S_B and S_C.

In set S_A, both workloads (4b3e and 8df4) exhibit steady fluctuations of daily peaks with a low standard deviation ($\sigma \leq 0.1$) and demonstrate clear daily periodicity, with both daily and weekly auto-correlation values exceeding 0.9. For set S_B, the workloads display weak periodicity (e.g., d82f), elusive spikes (e.g., 86f3), and volatile daily peaks, with the standard deviation reaching up to 0.25. In set S_C, three workloads (28ac, 7e75, and 98b1) show similar trends. Specifically, they all experience dramatic workload increases and unstable daily peaks, with a standard deviation of ≥ 0.1. Additionally, two spikes of different magnitudes between 6 PM and 9 PM each day further complicates accurate prediction and robust scaling, especially for workload 28ac. In summary, the workload patterns in the three sets effectively represent various scenarios of co-located LRAs, each with distinct levels of complexity. These diverse patterns present a significant challenge and effectively distinguish our proposed autoscaling framework from others.

5.2 Experiment Settings

During offline simulations, we emulate the actual implementation of OptScaler in a production cloud. The data from the initial 12 days will be used as training data for the workload prediction model. In the last 2 days (Jul/27 to Jul/28), OptScaler will be triggered every half hour (i.e., $h = 30$) to control the CPU utilization to approach the target $c^* = 0.5$.

Additionally, we take into account the scaling limitations, determined by the time interval h, the time cost τ of a single node addition/removal, and the concurrency of scaling s in (6). We set $\tau = 5$ minutes and $s = 4$ as in a real system. Consequently, with $h = 30$, a maximum of 24 nodes could be added or removed in each time interval. Furthermore, we establish the upper bound $X^{max} = 400$, and the lower bound $X^{min} = 80$ for S_A and S_B, and $X^{min} = 20$ for S_C, considering that the total workload often approaches zero in the latter case. We set the parameter $\alpha = 0.95$ for the chance constraint (11) for all optimization-based frameworks. As for OLR (see Algorithm 1), we select the feedback strength $\eta = 2e-4$ by minimizing the cumulative loss on the training data.

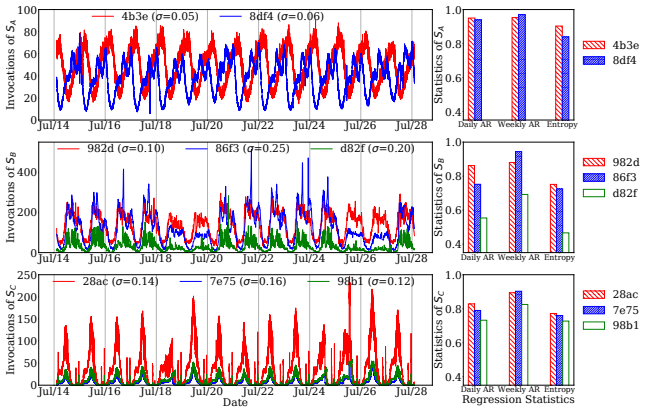

Figure 4: Time series (left) and regression statistics (right) for Azure Trace (S_A, S_C) and Alipay Trace (S_B). The complexity of prediction and scaling increases from S_A to S_B and S_C.

5.3 Comparison of Workload Prediction Models

5.3.1 Prediction Models. To address Question 1, we compare our workload prediction model with several traditional and state-of-the-art time series prediction models, namely ARIMA [27], Autoformer [56], DEPTS [13], and NBEATS [32]. ARIMA represents a traditional time series prediction model in statistics. Due to the workload pattern in Section 3.2, we adopt ARIMA with the exogenous regressors of Fourier series. Autoformer is a neural network model designed for long-term prediction based on auto-correlation mechanism and a seq2seq structure. DEPTS and NBEATS are two types of neural networks tailored for uni-variate time series prediction, incorporating the neural expansion analysis method. In the rolling window experiment, we train the ARIMA model every half hour using the most recent 24-hour data, owing to its fast training speed. As for the neural network-based models, training occurs at midnight each day. Throughout the training process, data preceding the training time is included in the training dataset, while the data from the last two days is utilized for validation. All these neural network-based models are trained with the 0.5-quantile loss.

5.3.2 Evaluation Metrics. We focus on the long-term prediction performance, that is, predicting the workload in the following 360 minutes every $h = 30$ minutes. The metric, weighted absolute percentage error (**WAPE**), is adopted to evaluate different models:

$$WAPE = \frac{1}{N}\sum_{n=1}^{N}\frac{\sum_{(t,\delta)\in\Omega}|y_n^{t+\delta} - \hat{y}_n^{t+\delta}|}{\sum_{(t,\delta)\in\Omega}|y_n^{t+\delta}|}, \tag{15}$$

Table 2: Comparison of five prediction models, testing on WAPE for the Minute-level and Interval-peak accuracy. Lower values are better, and the best are bolded.

	Minute-level			Interval-peak		
	S_A	S_B	S_C	S_A	S_B	S_C
ARIMA	0.087	0.280	0.432	0.107	0.286	0.420
Autoformer	0.079	0.258	0.467	0.085	0.274	0.454
DEPTS	0.066	0.243	0.356	0.082	0.257	0.342
NBEATS	0.067	**0.218**	0.352	0.085	0.178	0.377
Our model	**0.063**	0.224	**0.309**	**0.080**	**0.171**	**0.341**

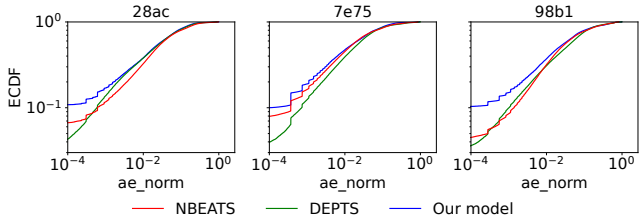

Figure 5: The empirical cumulative distribution function (ECDF) of normalized absolute error (*ae_norm*) with respect to Minute-level accuracy for three LRAs in set S_C. Both x-axis and y-axis are in log scale.

where n is the index of time series (LRAs in our experiments). t is the starting time of each rolling window, and δ is the increment in time, that is, (t, δ) means the δ-th step after the starting time t of the rolling window. Ω denotes the whole evaluation space.

We evaluate the prediction accuracy in two aspects: 1) The **Minute-level** accuracy, that is, whether the model can accurately predict the workload of each LRA in each minute, since all the models are built on the data in minute-level resolution; 2) The **Interval-peak** accuracy, that is, whether the model can correctly capture the peak workload in the given interval used in MPC-based optimization, i.e., $\boldsymbol{y}^j \in \mathbb{R}^N$ in constraint (8). This accuracy is critical to the subsequent optimization.

5.3.3 Experimental Results for Question 1. Table 2 illustrates the superior performance of OptScaler compared to four other prediction models across LRAs in S_A, S_B, and S_C, as measured by the Weighted Absolute Percentage Error (WAPE) from two different perspectives. Notably, all methods exhibit considerably better performance on set S_A in comparison to S_B and S_C. This observation aligns with the discussion in subsection 5.1, where it is evident that workloads in S_A display relatively strong periodicity and maintain a steady peak trend, whereas those in S_B and S_C demonstrate diverse fluctuations. Furthermore, our model consistently outperforms ARIMA by a significant margin. This further attests to the effectiveness of our Short-term Local Block, as both models extract the periodic property using Fourier series. In terms of Minute-level accuracy, our model outperforms the others in S_A and S_C and demonstrates the second best performance in S_B. Additionally, in the context of Interval-peak accuracy, a critical aspect in our proposed autoscaling framework, our model outshines all compared models.

To further highlight this superiority, we present Figure 5, which illustrates the performance of our model in comparison to the two most competitive neural network-based models (NBEATS and DEPTS) presented in Table 2. Specifically, the Minute-level accuracy of three LRAs in set S_C is compared using their empirical cumulative distribution functions (ECDF) of normalized absolute error (*ae_norm*, i.e., the absolute error divided by the maximal absolute error for each workload). As depicted in Figure 5, our method achieves notably higher empirical cumulative distribution on smaller *ae_norm* values. The ECDF curves of our method consistently surpass the others across all LRAs, indicating the superior performance of our method on complex workloads in set S_C.

In general, in our model, the Fourier series in long-term periodic block could capture various workload patterns without the necessity of long series input to the model, and the linear-complexity attention mechanism in short-term local block learns the local trend and fluctuation; the combination of these two blocks gives our model an edge over neural network based models like NBEATS and DEPTS in terms of efficiency and accuracy. Our model can also handle multiple time series by one model with fewer parameters, which is much more suitable for the scenario of co-located LRAs.

5.4 Comparison of Autoscaling Frameworks

To answer Questions 2 and 3 stated at the beginning of Section 5, we conduct an ablation experiment to compare the performance of OptScaler with prevalent autoscaling frameworks. Specifically, we respond to Question 2 in Section 5.4.3 by comparing all four frameworks, given the same input of the prediction results; then, we address Question 3 in Section 5.4.4 by comparing OptScaler with the best combination from the existing prediction models and autoscaling frameworks.

5.4.1 Autoscaling Frameworks. We investigate four autoscaling frameworks, i.e., reactive, proactive, hybrid, and collaborative:

- **Autopilot** [43] serves as the representative of **reactive** autoscalers. Introduced by Google in 2020, it is considered as an industrial benchmark in this context. Autopilot determines the optimal resource configuration by identifying the best-matched historical time intervals to the current window. We implement Autopilot's horizontal scaling based on its publicly available paper [43];
- **Madu** [26] exemplifies a typical **proactive** autoscaler, leveraging the capacity of both prediction and optimization. We adapt Madu to our context, formulating the optimization model as (5)-(10). As a pure proactive autoscaler, the CPU utilization estimator will not be updated after initialization;
- **HAS** [22] represents a **hybrid** autoscaler that adopts the mainstream concept of independent proactive and reactive modules. Without compromising on quality, we integrate HAS into our context, where the reactive decision replaces the proactive one when the observed CPU utilization exceeds a predefined bound. The current bound is set to be $0.9c^*$. The reactive decision at current time t is calculated by $u^t = \max\{X^{min} - x^{t-1}, \min\{(c^{t-1}/c^* - 1)x^{t-1}, h \cdot$

$s/\tau,\ X^{max} - x^{t-1}\}\}$, adjusting the resource to eliminate the scaling error at $t-1$. The min operation indicates $c^t \leq c^*$ if workload at t does not increase, while the max operation ensures the lower bound in (10) holds.

- **OptScaler** is our proposed **collaborative** framework.

Accordingly, we customize the following two parameters for the above autoscaling frameworks:

- S is a parameter for Autopilot that denotes the percentile of the most recent samples of CPU utilization. A higher value of S indicates a more conservative scaling preference. Here we consider two different values for S, namely $S \in \{90\%, 95\%\}$ for Autopilot;
- D represents time horizon in the optimization model for Madu, HAS, and OptScaler. In our context, we use a 30−minute interval (i.e., $h = 30$) for each d. A value of $D = 1$ signifies a standard optimization model focusing on immediate scaling needs, while $D = 11$ represents an MPC-based optimization model that can anticipate future events within the following 6-hour (as per constraint (8), we need to consider one more time interval than D). Therefore, in the case of $D = 11$, we will end our experiment at 6 PM on Jul/28.

5.4.2 Evaluation Metrics. As an end-to-end autoscaling framework, our primary objective is to achieve a balance between the cost of computing resources and the stringent SLOs in cloud service. With this goal in mind, we have developed three evaluation metrics, where lower metric values indicate better performance:

- S_{vr} denotes the SLO violation rate, representing the percentage of minutes with SLO violation (i.e., $c^d > c^*$) during the experiment period. Therefore, a high value of S_{vr} indicates a prolonged duration of SLO failure;
- V_{sum} denotes the accumulated magnitude of SLO violation, calculated as the sum of $max(c^d - c^*, 0)$ for all the minutes during the experiment period. It is important to note that a high S_{vr} may be accompanied by a low V_{sum}; therefore, lower metric values signify desirable SLO satisfaction;
- R_{avg} denotes the cost of resources, calculated as the average number of installed nodes x^d during the experiment period;

5.4.3 Experimental Results for Question 2. The statistical results are summarized in Table 3, and the metrics are explained in Section 5.4.2. From the table, OptScaler excels in SLO satisfaction while maintaining relatively low resource cost, particularly for complex workloads (e.g., S_B and S_C). Further details are provided below:

- Under the quantile of $S = 90\%$, Autopilot demonstrates an unacceptable level of safety across all sets; with $S = 95\%$, it achieves an averaged performance in preventing SLO violations across all experiment groups, but the resource cost R_{avg} is notably higher than in other experiment groups, particularly in S_A and S_C, potentially limiting its applicability under tight resource budgets. The high magnitude of V_{sum} presents another issue for Autopilot to address;
- Madu demonstrates high risks in terms of both SLO violation rate S_{vr} and magnitude V_{sum}. This can be attributed to the disparity between estimated CPU utilization and the real system feedback. Without adjustment by a reactive module, this gap will consistently undermine the reliability of final scaling decisions. This underscores the assertion in [47] that a reactive scaling component should be a part of every production-ready autoscaler, complementing an unsatisfactory proactive counterpart;
- OptScaler outperforms Autopilot in terms of both safety and resource costs. When compared to the hybrid autoscaler HAS with $D = 11$, OptScaler incurs a slightly higher resource cost R_{avg} (1% ∼ 12% more) to achieve a significantly lower SLO violation rate S_{vr} (53% ∼ 75% less) and magnitude V_{sum} (56% ∼ 82% less). This advantage is even more pronounced when compared to Autopilot and Madu. It demonstrates OptScaler's superiority in SLO satisfaction, particularly for challenging workload patterns such as in sets S_B and S_C;
- The comparison between $D = 1$ and $D = 11$ reveals that the multi-timestep optimization structure in MPC brings a consistent improvement of the rate S_{vr} and magnitude V_{sum} of SLO violation in both OptScaler and HAS.

We have further generated Figure 6 to present the experimental details of CPU utilization for clusters linked with sets S_A, S_B, and S_C. In this figure, we display the **Ideal** line for c^*, along with two **Mean** lines representing actual CPU utilization under different parameter settings. The light **Range** area illustrates the $\alpha = 0.95$ confidence interval of the actual CPU utilization. For improved clarity, we only depict the Range area for the line demonstrating better performance, as indicated in Table 3; specifically, we display $S = 95\%$ for Autopilot and $D = 11$ for the three optimization-based autoscalers. Any part of the line above the Ideal line signifies the risk of SLO violations. From Figure 6, we observe four key insights:

- Autopilot struggles with significant fluctuations in CPU utilization due to the absence of workload prediction. As showcased for S_C in Figure 4 and Figure 6, from 6 AM to 9 AM on Jul 27 (indicated as 27-06 to 27-09 on the x-axis), Autopilot exhibits delayed responsiveness to the substantial surge in workloads, potentially leading to severe SLO violations. Using $S = 95\%$ only serves as a partial solution. The underlying problem is that Autopilot only considers a limited historical window in preparing resources for the future. Consequently, when a more significant workload spike occurs, the risk of violations could reoccur;
- The pure proactive scaler Madu also experiences SLO violations due to the absence of a reactive module to rectify scaling errors, leading to a wider range of SLO violations than HAS and OptScaler, as evidenced in the period from 28-06 to 28-12 for S_C in Figure 6. This presents an undesirable outcome for cloud service providers. In contrast, the hybrid scaler HAS demonstrates less frequent SLO violations, underscoring the effectiveness of including both proactive and reactive modules in ensuring SLO compliance;
- OptScaler outperforms HAS by maintaining a safer fluctuation that significantly reduces both the duration and magnitude of SLO violations, as also reflected in S_{vr} and V_{sum} in Table 3. For example, for set S_C in Figure 4 and Figure 6, during two workload spikes around 6 PM and 9 PM (27-18 and 27-21 on the x-axis), HAS experiences two instances of SLO violations with CPU utilization reaching

Table 3: Comparison on four types of autoscalers. Each autoscaler is tested with sets S_A, S_B, and S_C. Three metrics are evaluated under different parameter settings. Lower metric values are desired, and the best are bolded.

Type	Name	Param	S_A[a]			S_B[a]			S_C[a]		
			$S_{vr}(\%)$	V_{sum}	R_{avg}	$S_{vr}(\%)$	V_{sum}	R_{avg}	$S_{vr}(\%)$	V_{sum}	R_{avg}
Reactive	Autopilot	$S = 90\%$	6.1	1.87	115.9	13.8	45.85	**193.0**	9.6	34.52	132.6
		$S = 95\%$	2.8	0.87	119.0	1.8	4.98	266.7	4.3	9.64	158.8
Proactive	Madu	$D = 1$	13.7	8.87	**99.0**	5.6	13.20	214.9	7.0	8.74	**69.5**
		$D = 11$	13.6	8.65	99.1	2.7	3.22	252.5	6.5	8.34	70.0
Hybrid	HAS	$D = 1$	1.6	0.73	105.1	5.3	13.96	223.0	3.6	3.36	81.3
		$D = 11$	1.5	0.58	104.6	2.2	2.21	263.2	2.8	3.13	80.4
Collaborative	**OptScaler (proposed)**	$D = 1$	1.2	0.26	113.0	2.1	3.56	226.3	1.1	0.63	84.8
		$D = 11$	**0.7**	**0.25**	112.3	**0.6**	**0.61**	265.7	**0.7**	**0.55**	90.5

[a] Due to the randomness in sampling the CPU utilization feedback, all evaluation metrics are averaged over five repetitive runs and calculated in minute-level resolution. S_{vr} denotes the SLO violation rate, V_{sum} denotes the accumulated magnitude of SLO violation, and R_{avg} denotes the resource cost.

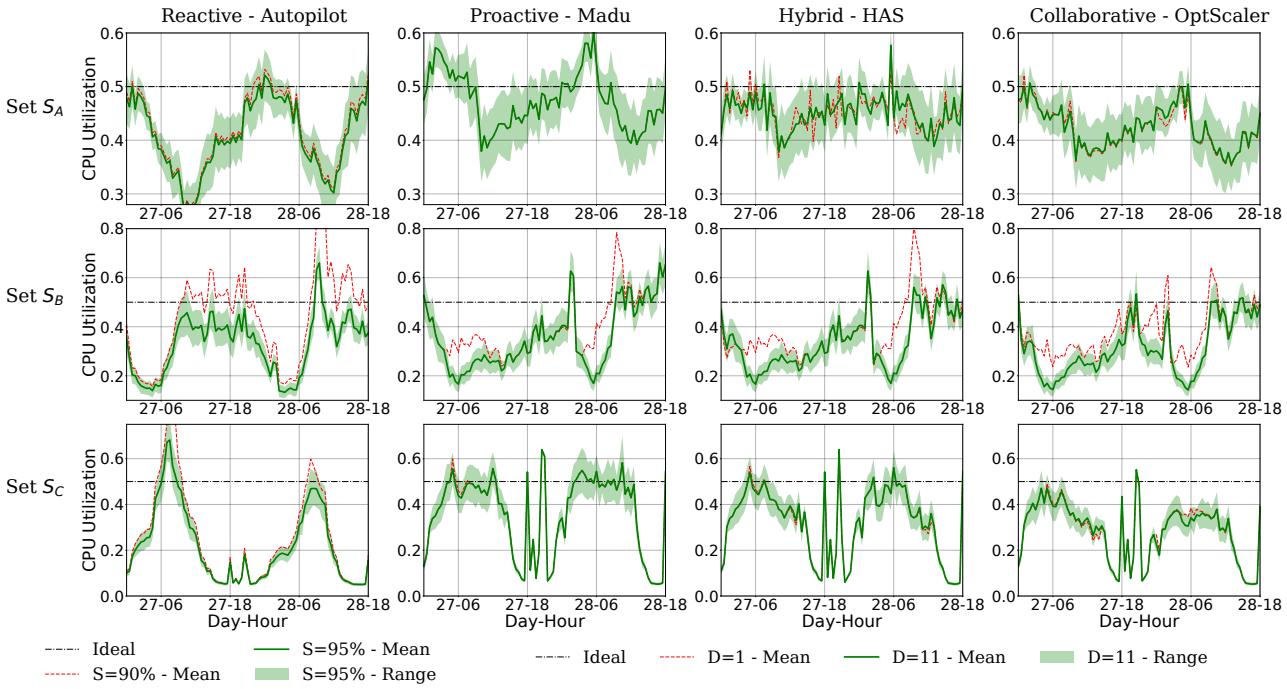

Figure 6: Experimental results of CPU utilization of four types of autoscalers. Each autoscaler (column) is tested with sets S_A (top row), S_B (middle row), and S_C (bottom row) under two different parameter settings (as indicated by the bottom legends).

0.63, while OptScaler encounters a single violation with less severe impact at 0.51;

- OptScaler, HAS, and Madu exhibit similar overall trends, but OptScaler consistently maintains CPU utilization at a much safer level, particularly under $D = 11$. Interestingly, the increase in the time horizon D from 1 to 11 in MPC-based optimization results in less satisfactory CPU utilization for HAS and Madu compared to OptScaler. This is ascribed to their CPU estimator's inability to adapt to the evolving cloud environment characterized by less regular workload patterns. This illustrates the significance of incorporating a reactive module to align with MPC-based optimization, as implemented in OptScaler.

In summary, Autopilot exhibits limitations in effectively managing substantial workload fluctuations, owing to its hysteretic nature. However, when considering the three autoscalers equipped with a proactive module, OptScaler, as a collaborative autoscaler, demonstrates the most resilient performance at a slightly higher

resource cost, followed by HAS, the hybrid autoscaler, with Madu, a pure proactive autoscaler, ranking at the bottom. This observation suggests that OptScaler effectively strikes a balance between resource cost and the risk of SLO violations, particularly for the co-located LRAs characterized by complex workload patterns.

5.4.4 Experimental Results for Question 3. We compare OptScaler with the existing HAS framework (the second-best performer in Table 3) using NBEATS (the second-best prediction model in Table 2). We design the experiments in an ablative manner, i.e., we adopt NBEATS to replace the workload prediction model in OptScaler to illustrate the impact of different prediction models on the final decision. The results are presented in Table 4.

Upon examining Table 4, a comparison between Exp.2 and Exp.1 reveals a relatively 27% $\sim$ 60% improvement of S_{vr} by upgrading the autoscaling framework alone. Comparing Exp.3 to Exp.2 demonstrates a further improvement of 25% $\sim$ 53% in S_{vr}, along with significant reductions in both V_{sum} and R_{avg}, achieved by advancing the prediction model from NBEATS to our model. Consequently, when comparing OptScaler as a whole (Exp.3) to a best hybrid framework (HAS with NBEATS in Exp.1), OptScaler incurs 4% $\sim$ 15% more resource costs in order to achieve 36% $\sim$ 81% fewer SLO violations. This demonstrates the clear benefits of OptScaler for LRAs with demanding SLOs. In essence, Table 4 serves to illustrate that the robustness of OptScaler is derived from enhancements in both the workload prediction model and the autoscaling framework.

5.4.5 Further discussions. The experimental results above provide a broad response to Question 2 and 3. However, in practice, the scenario may vary based on different scaling frequencies and unforeseen workloads. Therefore, in this subsection, we further enrich our investigation by delving into the following specific situations.

Sensitivity analysis. In order to explore the impact of hyperparameters in OptScaler, we conducted sensitivity experiments on a key parameter, η, which represents the strength of feedback in the reactive module (refer to Algorithm 1). We examined various values of η, ranging from $5e-3$ to 0 in increments of order of magnitude. When $\eta = 0$, it effectively simulates the closed state of the reactive module (comparable to Madu). The results are presented in Figure 7, where the left and right y-axes depict the crucial metrics of resource cost R_{avg} (shown as bars) and SLO violation rate S_{vr} (depicted as lines), respectively. Lower values are favourable for both metrics. We normalized each metric as a percentage of the corresponding result with $\eta = 2e-4$ (the default value of OptScaler), allowing for a comparison of performance across different η values.

The findings portrayed in Figure 7 reveal a distinct trend wherein R_{avg} and S_{vr} move inversely as η decreases from $5e-3$ to 0. Thus, it becomes evident that η significantly influences the trade-off between the two evaluation metrics. Opting for a higher feedback strength with η proves effective in suppressing SLO violations. However, an overly aggressive reactive strategy with a large η also leads to an over-provisioning of resources. Figure 7 suggests that a relatively optimal range for η falls between 5e-4 to 1e-4 across all sets. It is important to recognize that in real-world implementation, the selection of η should be fine-tuned according to the local policies governing SLOs and resource budgets.

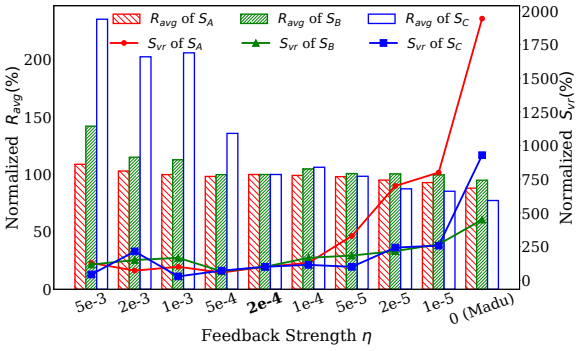

Figure 7: Sensitivity experiments of feedback strength η in OptScaler show a clear balance between resource cost R_{avg} (plotted as bars) and SLO violation rate S_{vr} (plotted as lines). Metrics are evaluated under $D = 11$ and normalized with respect to the results of $\eta = 2e-4$, the lower values the better.

Unpredictable workload. To further evaluate the robustness of OptScaler, we mock some totally unpredictable spikes by directly increasing the workloads to 10 times of the original value based on S_A, and test the response of OptScaler and its reactive competitor Autopilot to that workload. Note that Autopilot's ability to recognize the unprecedented spike depends on a percentile parameter S as mentioned in Section 5.4.1. A higher S makes it more sensitive to unpredictable workload. Our results in Table 5 show that when $S = 95\%$, Autopilot tends to ignore the unprecedented spike, especially when the duration of the spike is short (e.g. 30-minute). In contrast, OptScaler recognizes the event and scales up the resources accordingly, resulting in a lower SLO violation S_{vr}; when S continues to increase (e.g. 99%), both OptScaler and Autopilot can quickly respond to the event; however, a higher S will lead to an over-conservative Autopilot, and OptScaler leads to significantly less resource cost R_{avg} than Autopilot does when they achieve a similar level of S_{vr}.

Scaling frequency. In practice, clusters may vary in scaling frequency, hence we enrich our experiments by investigating autoscaling frameworks under different intervals h. Here we test $h = 10$ for 10-minute interval, and $h = 60$ for 1-hour interval. We omit tables for lack of space, and the results are summarized as follows: 1) OptScaler achieves the best S_{vr} and V_{sum} under both h, followed by HAS as the second-best, which aligns with that of $h = 30$; 2) Under the same horizon D, OptScaler generally costs more resources in the scenario of 1-hour interval than that of 10-minute, as it considers a longer period of future workloads; 3) The evaluation metrics of Autopilot shows no significant differences under different h, as it has no proactive or optimization module, and h has very limited impact on the evolving process of scaling metrics.

6 DEPLOYMENT

Before deployment, we tested OptScaler in a production cluster with co-located LRAs to verify its effectiveness in the online environment. We refrained from comparing OptScaler with other frameworks due to concerns about their SLO violations exceeding 2%, as indicated by S_{vr} in Table 3. We conducted our experiment from 8 AM to 12

Table 4: Comparison between OptScaler and the existing hybrid framework of combining NBEATS and HAS. Ablative studies are conducted on all sets under $D = 11$. Metrics with lower values are desired, and the best are bolded.

ID	Autoscaling Framework	Prediction Model	S_A			S_B			S_C		
			$S_{vr}(\%)$	V_{sum}	R_{avg}	$S_{vr}(\%)$	V_{sum}	R_{avg}	$S_{vr}(\%)$	V_{sum}	R_{avg}
Exp.1	HAS	NBEATS	1.1	0.60	**105.9**	2.0	3.04	**254.3**	3.7	8.75	**78.8**
Exp.2	OptScaler	NBEATS	0.8	0.40	114.5	0.8	0.76	277.6	1.5	5.32	133.8
Exp.3	**OptScaler**	**Our model**	**0.7**	**0.25**	112.3	**0.6**	**0.61**	265.7	**0.7**	**0.55**	90.5

Table 5: Comparison on the response of OptScaler and Autopilot to unpredictable workloads on set S_A. Lower metric values are desired, and the best are bolded.

Name	Param	30-minute spike		3-hour spike	
		$S_{vr}(\%)$	R_{avg}	$S_{vr}(\%)$	R_{avg}
Autopilot	$S = 95\%$	3.7	**120.9**	12.7	567.0
	$S = 99\%$	2.3	569.6	12.6	609.7
OptScaler	$D = 1$	2.4	306.4	12.6	**316.7**
	$D = 11$	**2.2**	308.3	**12.5**	317.6

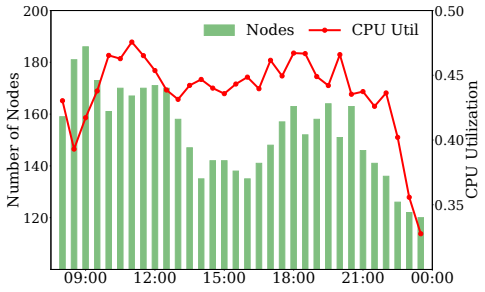

Figure 8: Online experimental results of OptScaler on the number of installed nodes (bars with left y-axis) and CPU utilization (red line with right y-axis). OptScaler demonstrates its ability to regulate CPU utilization.

PM. From the result shown in Figure 8, OptScaler achieved a steady CPU utilization (between 0.4 and 0.48) during the whole daytime. It started to decrease after 10 PM as the total workload dropped to half of that in the daytime. Besides, the previous resource cost was fixed at 190 to ensure safety, and the cost R_{avg} by OptScaler was 152.9, saving 19.5% of cloud resources.

In addition, OptScaler benefits users for its interpretability. It allows users to trace back to specific conditions and parameters in the model to understand why a scaling decision is made. When an unexpected scaling occurs, this model-based system can facilitate the debugging and make further improvement. Hence, users are likely to trust OptScaler more than other black-box scaling methods.

OptScaler has been successfully deployed as an integrated platform of prediction and decision-making at Alipay, supporting the autoscaling of more than 100 online LRAs. To access the platform of OptScaler, a cluster typically undergoes the following steps: 1) The user configures necessary parameters for the cluster and the algorithm; 2) The cloud system monitors and collects the workloads and metrics of the cluster, and the data will be saved to a database; 3) OptScaler loads historical workload data for the proactive module to train model offline at a regular pace, and outputs the prediction results at real-time; 4) OptScaler loads the latest system feedback from the database to fine-tune the reactive module; 5) The optimization module makes a final scaling decision, which is returned to the cloud cluster for implementation.

7 CONCLUSION

We present OptScaler as a pioneering collaborative autoscaling framework, designed as an upgrade to the widely used hybrid framework. OptScaler stands out as the first framework to address the collaboration of proactive and reactive methods through sophisticated optimization techniques, such as Model Predictive Control (MPC). Unlike approaches that employ proactive and reactive methods independently, which can lead to incompatibilities, OptScaler orchestrates the strengths of both proactive and reactive modules to effectively manage workload fluctuations and system uncertainty, significantly enhancing the robustness of cloud clusters. OptScaler surpasses other prevalent autoscaling frameworks thanks to its superior prediction model and collaborative mechanism. Offline experiments demonstrate that OptScaler effectively mitigates the risk of SLO violations by a minimum of 36% in comparison to other frameworks. In online experiments, OptScaler maintains desirable CPU utilization while achieving substantial savings of up to 19.5% in cloud resources. It is noteworthy that OptScaler has already been deployed online to support the autoscaling of LRAs at Alipay, a leading global payment platform.

REFERENCES

[1] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2017. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing* 11, 2 (2017), 430–447.

[2] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. 2012. An adaptive hybrid elasticity controller for cloud infrastructures. In *2012 IEEE Network Operations and Management Symposium*. IEEE, 204–212.

[3] Amazon. 2023. *Dynamic scaling for Amazon EC2 Auto Scaling*. Retrieved June 28, 2024 from https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html

[4] Sumit Bose, Anjaneyulu Pasala, Dheepak A, Sridhar Murthy, and Ganesan Malaiyandisamy. 2011. *SLA Management in Cloud Computing: A Service Provider's Perspective*. 413 – 436. https://doi.org/10.1002/9780470940105.ch16

[5] Stephen Burroughs, Helge Dickel, Martin van Zijl, Vladimir Podolskiy, Michael Gerndt, Robi Malik, and Panos Patros. 2021. Towards Autoscaling with Guarantees on Kubernetes Clusters. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. 295–296. https://doi.org/10.1109/ACSOS-C52956.2021.00073

[6] Joyce Cahoon, Wenjing Wang, Yiwen Zhu, Katherine Lin, Sean Liu, Raymond Truong, Neetu Singh, Chengcheng Wan, Alexandra M Ciortea, Sreraman Narasimhan, et al. 2022. Doppler: automated SKU recommendation in migrating SQL workloads to the cloud. *PVLDB* 15, 12 (2022), 3509–3521.

[7] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–40.

[8] Sudipto Das, Feng Li, Vivek R Narasayya, and Arnd Christian König. 2016. Automated demand-driven resource scaling in relational database-as-a-service. In *Proceedings of the 2016 International Conference on Management of Data*. 1923–1934.

[9] Fan Deng, Jie Lu, Shi-Yu Wang, Jie Pan, and Li-Yong Zhang. 2019. A distributed PDP model based on spectral clustering for improving evaluation performance. *World Wide Web* 22 (2019), 1555–1576.

[10] Fan Deng, Shiyu Wang, Liyong Zhang, Xiaoqian Wei, and Jingping Yu. 2018. Establishment of attribute bitmaps for efficient XACML policy evaluation. *Knowledge-Based Systems* 143 (2018), 93–101.

[11] Naghmeh Dezhabad and Saeed Sharifian. 2018. Learning-based dynamic scalable load-balanced firewall as a service in network function-virtualized cloud computing environments. *The Journal of Supercomputing* 74, 7 (4 2018), 3329–3358. https://doi.org/10.1007/s11227-018-2387-5

[12] Jianru Ding, Ruiqi Cao, Indrajeet Saravanan, Nathaniel Morris, and Christopher Stewart. 2019. Characterizing service level objectives for cloud services: Realities and myths. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 200–206.

[13] Wei Fan, Shun Zheng, Xiaohan Yi, Wei Cao, Yanjie Fu, Jiang Bian, and Tie-Yan Liu. 2022. DEPTS: Deep Expansion Learning for Periodic Time Series Forecasting. In *International Conference on Learning Representations*. https://openreview.net/forum?id=AJAR-JgNw__

[14] Soodeh Farokhi, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. 2016. A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Computer Systems* 65 (2016), 57–72.

[15] Mauro Gaggero and Luca Caviglione. 2018. Model predictive control for energy-efficient, quality-aware, and secure virtual machine placement. *IEEE Transactions on Automation Science and Engineering* 16, 1 (2018), 420–432.

[16] Alessio Gambi, Waldemar Hummer, Hong-Linh Truong, and Schahram Dustdar. 2013. Testing elastic computing systems. *IEEE Internet Computing* 17, 6 (2013), 76–82.

[17] KS Holkar and Laxman M Waghmare. 2010. An overview of model predictive control. *International Journal of control and automation* 3, 4 (2010), 47–63.

[18] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. 2018. Combined vertical and horizontal autoscaling through model predictive control. In *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings 24*. Springer, 147–159.

[19] Pooyan Jamshidi, Claus Pahl, and Nabor C Mendonça. 2016. Managing uncertainty in autonomic cloud elasticity controllers. *IEEE Cloud Computing* 3, 3 (2016), 50–60.

[20] Xiaodong Jiang, Sudeep Srivastava, Sourav Chatterjee, Yang Yu, Jeffrey Handler, Peiyi Zhang, Rohan Bopardikar, Dawei Li, Yanjun Lin, Uttam Thakore, Michael Brundage, Ginger Holt, Caner Komurlu, Rakshita Nagalla, Zhichao Wang, Hechao Sun, Peng Gao, Wei Cheung, Jun Gao, Qi Wang, Marius Guerard, Morteza Kazemi, Yulin Chen, Chong Zhou, Sean Lee, Nikolay Laptev, Tihamér Levendovszky, Jake Taylor, Huijun Qian, Jian Zhang, Aida Shoydokova, Trisha Singh, Chengjun Zhu, Zeynep Baz, Christoph Bergmeir, Di Yu, Ahmet Koylan, Kun Jiang, Ploy Temiyasathit, and Emre Yurtbay. 2022. *Kats*. Retrieved June 28, 2024 from https://github.com/facebookresearch/Kats

[21] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. 2024. Time-LLM: Time series forecasting by reprogramming large language models. In *International Conference on Learning Representations (ICLR)*.

[22] Bibal Benifa JV and Dejey Dharma. 2018. HAS: hybrid auto-scaler for resource scaling in cloud environment. *J. Parallel and Distrib. Comput.* 120 (2018), 1–15.

[23] Kubernetes. 2023. *Kubernetes Documentation / Concepts / Containers*. Retrieved June 28, 2024 from https://kubernetes.io/docs/concepts/containers/

[24] Kubernetes. 2023. *Kubernetes Documentation / Reference / Glossary*. Retrieved June 28, 2024 from https://kubernetes.io/docs/reference/glossary/?all=true#term-cluster

[25] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2023. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625* (2023).

[26] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2022. The Power of Prediction: Microservice Auto Scaling via Workload Learning. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 355–369. https://doi.org/10.1145/3542929.3563477

[27] Spyros Makridakis and Michele Hibon. 1997. ARMA models and the Box–Jenkins methodology. *Journal of forecasting* 16, 3 (1997), 147–163.

[28] Microsoft. 2023. *Overview of autoscale in Azure*. Retrieved June 28, 2024 from https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview

[29] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of machine learning*. MIT press.

[30] In Jae Myung. 2003. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology* 47, 1 (2003), 90–100.

[31] Yukio Ogawa, Go Hasegawa, and Masayuki Murata. 2018. Hierarchical and frequency-aware model predictive control for bare-metal cloud applications. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 11–20.

[32] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2019. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437* (2019).

[33] Zhicheng Pan, Yihang Wang, Yingying Zhang, Sean Bin Yang, Yunyao Cheng, Peng Chen, Chenjuan Guo, Qingsong Wen, Xiduo Tian, Yunliang Dou, et al. 2023. MagicScaler: Uncertainty-Aware, Predictive Autoscaling. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3808–3821.

[34] Youngsuk Park, Danielle Maddix, François-Xavier Aubet, Kelvin Kan, Jan Gasthaus, and Yuyang Wang. 2022. Learning quantile functions without quantile crossing for distribution-free time series forecasting. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 8127–8150.

[35] Valerio Persico, Domenico Grimaldi, Antonio Pescape, Alessandro Salvi, and Stefania Santini. 2017. A fuzzy approach based on heterogeneous metrics for scaling out public clouds. *IEEE Transactions on Parallel and Distributed Systems* 28, 8 (2017), 2117–2130.

[36] Vladimir Podolskiy, Anshul Jindal, and Michael Gerndt. 2018. Iaas reactive autoscaling performance challenges. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 954–957.

[37] Olga Poppe, Pablo Castro, Willis Lang, and Jyoti Leeka. 2023. Proactive Resource Allocation Policy for Microsoft Azure Cognitive Search. *ACM SIGMOD Record* 52, 3 (2023), 41–48.

[38] Huajie Qian, Qingsong Wen, Liang Sun, Jing Gu, Qiulin Niu, and Zhimin Tang. 2022. Robustscaler: Qos-aware autoscaling for complex workloads. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2762–2775.

[39] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*. 805–825.

[40] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2023. {AWARE}: Automate workload autoscaling with reinforcement learning in production cloud systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 387–402.

[41] Víctor Rampérez, Javier Soriano, David Lizcano, and Juan A Lara. 2021. FLAS: A combination of proactive and reactive auto-scaling architecture for distributed services. *Future Generation Computer Systems* 118 (2021), 56–72.

[42] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 500–507.

[43] Krzysztof Rzadca, Paweł Findeisen, Jacek Świderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Paweł Krzysztof Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: Workload Autoscaling at Google Scale. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16. https://dl.acm.org/doi/10.1145/3342195.3387524

[44] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[45] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. 2021. *Lectures on stochastic programming: modeling and theory*. SIAM.

[46] Parminder Singh, Avinash Kaur, Pooja Gupta, Sukhpal Singh Gill, and Kiran Jyoti. 2021. RHAS: robust hybrid auto-scaling for web applications in cloud computing. *Cluster Computing* 24, 2 (2021), 717–737.

[47] Martin Straesser, Johannes Grohmann, Jóakim von Kistowski, Simon Eismann, André Bauer, and Samuel Kounev. 2022. Why is it not solved yet? challenges for production-ready autoscaling. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*. 105–115.

[48] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. 2008. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3, 1 (2008), 1–39.

[49] Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106 (2006), 25–57.

[50] Shiyu Wang. 2024. NeuralReconciler for Hierarchical Time Series Forecasting. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 731–739.

[51] Shiyu Wang, Yinbo Sun, Xiaoming Shi, Zhu Shiyi, Lin-Tao Ma, James Zhang, YangFei Zheng, and Liu Jian. 2023. Full scaling automation for sustainable development of green data centers. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 6264–6271.

[52] Shiyu Wang, Yinbo Sun, Yan Wang, Fan Zhou, Lin-Tao Ma, James Zhang, and YangFei Zheng. 2023. Flow-Based End-to-End Model for Hierarchical Time Series Forecasting via Trainable Attentive-Reconciliation. In *International Conference on Database Systems for Advanced Applications*. Springer, 167–176.

[53] Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and JUN ZHOU. 2024. TimeMixer: Decomposable Multiscale Mixing for Time Series Forecasting. In *International Conference on Learning Representations (ICLR)*.

[54] Shiyu Wang, Fan Zhou, Yinbo Sun, Lintao Ma, James Zhang, and Yangfei Zheng. 2022. End-to-end modeling of hierarchical time series using autoregressive transformer and conditional normalizing flow-based reconciliation. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1087–1094.

[55] Haixu Wu, Jialong Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2022. Flowformer: Linearizing transformers with conservation flows. *Proceedings of the 39 th International Conference on Machine Learning* 162 (2022).

[56] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems* 34 (2021), 22419–22430.

[57] Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, et al. 2022. A Meta Reinforcement Learning Approach for Predictive Autoscaling in the Cloud. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4290–4299.

[58] Qi Zhang, Quanyan Zhu, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. 2013. Dynamic service placement in geographically distributed clouds. *IEEE Journal on Selected Areas in Communications* 31, 12 (2013), 762–772.

[59] Zhiqiang Zhou, Chaoli Zhang, Lingna Ma, Jing Gu, Huajie Qian, Qingsong Wen, Liang Sun, Peng Li, and Zhimin Tang. 2023. AHPA: adaptive horizontal pod autoscaling systems on alibaba cloud container service for kubernetes. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 15621–15629.

[60] Zhibo Zhu, Ziqi Liu, Ge Jin, Zhiqiang Zhang, Lei Chen, Jun Zhou, and Jianyong Zhou. 2021. MixSeq: connecting macroscopic time series forecasting with microscopic time series data. *Advances in Neural Information Processing Systems* 34 (2021), 12904–12916.